AN INTRODUCTION TO MESHFREE METHODS AND THEIR PROGRAMMING

An Introduction to Meshfree Methods and Their Programming

by

G.R. LIU

National University of Singapore, Singapore

and

Y.T. GU

National University of Singapore, Singapore

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN-10 1-4020-3228-5 (HB) Springer Dordrecht, Berlin, Heidelberg, New York
ISBN-10 1-4020-3468-7 (e-book) Springer Dordrecht, Berlin, Heidelberg, New York
ISBN-13 978-1-4020-3228-8 (HB) Springer Dordrecht, Berlin, Heidelberg, New York
ISBN-13 978-1-4020-3468-8 (e-book) Springer Dordrecht, Berlin, Heidelberg, New York

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

Printed on acid-free paper

Printed in the Netherlands.

Dedication

To Zuona

Yun, Kun, Run,

and my family

for the support and encouragement

G. R. Liu

To Qingxia

and Zhepu

for the love, support and encouragement

To my mentor, Professor Liu for his guidance

Y. T. Gu

Table of Contents

Preface xiii

Authors xix

1 Fundamentals 1

1.1 Numerical simulation 1
1.2 Basics of mechanics for solids 3
1.2.1 Equations for three-dimensional solids 4
1.2.1.1 Stress components 4
1.2.1.2 Strain-displacement equations 5
1.2.1.3 Constitutive equations 6
1.2.1.4 Equilibrium equations 7
1.2.1.5 Boundary conditions and initial conditions 8
1.2.2 Equations for two-dimensional solids 9
1.2.2.1 Stress components 9
1.2.2.2 Strain-displacement equation 10
1.2.2.3 Constitutive equations 11
1.2.2.4 Equilibrium equations 12
1.2.2.5 Boundary conditions and initial conditions 12
1.3 Strong-forms and weak-forms 13
1.4 Weighted residual method 14
1.4.1 Collocation method 17
1.4.2 Subdomain method 18
1.4.3 Least squares method 19
1.4.4 Moment method 20
1.4.5 Galerkin method 20
1.4.6 Examples 21
1.4.6.1 Use of the collocation method 23
1.4.6.2 Use of the subdomain method 23
1.4.6.3 Use of the least squares method 24
1.4.6.4 Use of the moment method 24
1.4.6.5 Use of the Galerkin method 25
1.4.6.6 Use of more terms in the approximate solution 26
1.5 Global weak-form for solids 27
1.6 Local weak-form for solids 34
1.7 Discussions and remarks 36

2 Overview of meshfree methods 37

2.1 Why Meshfree methods 37
2.2 Definition of Meshfree methods 39
2.3 Solution procedure of MFree methods 40
2.4 Categories of Meshfree methods 44
2.4.1 Classification according to the formulation procedures 45
2.4.1.1 Meshfree methods based on weak-forms 45
2.4.1.2 Meshfree methods based on collocation techniques 46
2.4.1.3 Meshfree methods based on the combination of weak-form and collocation techniques 47
2.4.2 Classification according to the function approximation schemes 47
2.4.2.1 Meshfree methods based on the moving least squares approximation 48
2.4.2.2 Meshfree methods based on the integral representation method for the function approximation 48
2.4.2.3 Meshfree methods based on the point interpolation method 49
2.4.2.4 Meshfree methods based on the other meshfree interpolation schemes 49
2.4.3 Classification according to the domain representation 49
2.4.3.1 Domain-type meshfree methods. 50
2.4.3.2 Boundary-type meshfree methods. 50
2.5 Future development 51

3 Meshfree shape function construction 54

3.1 Introduction 54
3.1.1 Meshfree interpolation/approximation techniques 55
3.1.2 Support domain 58
3.1.3 Determination of the average nodal spacing 58
3.2 Point interpolation methods 60
3.2.1 Polynomial PIM shape functions 61
3.2.1.1 Conventional polynomial PIM 61
3.2.1.2 Weighted least square (WLS) approximation 67
3.2.1.3 Weighted least square approximation of Hermite-type 69
3.2.2 Radial point interpolation shape functions 74
3.2.2.1 Conventional RPIM 74
3.2.2.2 Hermite-type RPIM 81
3.2.3 Source code for the conventional RPIM shape functions 86
3.2.3.1 Implementation issues 86
3.2.3.2 Program and data structure 88

3.2.3.3 Examples of RPIM shape functions 90
3.3 Moving least squares shape functions 97
3.3.1 Formulation of MLS shape functions 97
3.3.2 Choice of the weight function 102
3.3.3 Properties of MLS shape functions 106
3.3.4 Source code for the MLS shape function 108
3.3.4.1 Implementation issues 108
3.3.4.2 Program and data structure 111
3.3.4.3 Examples of MLS shape functions 111
3.4 Interpolation error using Meshfree shape functions 114
3.4.1 Fitting of a planar surface 118
3.4.2 Fitting of a complicated surface 118
3.5 Remarks 122
Appendix 124
Computer programs 131

4 Meshfree methods based on global weak-forms 145

4.1 Introduction 145
4.2 Meshfree radial point interpolation method 148
4.2.1 RPIM formulation 148
4.2.2 Numerical implementation 155
4.2.2.1 Numerical integration 155
4.2.2.2 Properties of the stiffness matrix 157
4.2.2.3 Enforcement of essential boundary conditions 158
4.2.2.4 Conformability of RPIM 160
4.3 Element Free Galerkin method 161
4.3.1 EFG formulation 161
4.3.2 Lagrange multiplier method for essential boundary conditions 163
4.4 Source code 167
4.4.1 Implementation issues 167
4.4.1.1 Support domain and the influence domain 167
4.4.1.2 Background cells 169
4.4.1.3 Method to enforce essential boundary conditions 169
4.4.1.4 Shape parameters used in RBFs 169
4.4.2 Program description and data structures 171
4.5 Example for two-dimensional solids – a cantilever beam 177
4.5.1 Using MFree_Global.f90 179
4.5.2 Effects of parameters 186
4.5.2.1 Parameter effects on RPIM method 187
4.5.2.2 Parameter effects on EFG method 191
4.5.3 Comparison of convergence 193
4.5.4 Comparison of efficiency 194

4.6 Example for 3D solids 196
4.7 Examples for geometrically nonlinear problems 198
4.7.1 Simulation of upsetting of a billet 199
4.7.2 Simulation of large deflection of a cantilever beam 200
4.7.3 Simulation of large deflection of a fixed-fixed beam 201
4.8 MFree2D© 201
4.9 Remarks 204
Appendix 205
Computer programs 219

5 Meshfree methods based on local weak-forms 237

5.1 Introduction 237
5.2 Local radial point interpolation method 239
5.2.1 LRPIM formulation 239
5.2.2 Numerical implementation 246
5.2.2.1 Type of local domains 246
5.2.2.2 Property of the stiffness matrix 247
5.2.2.3 Test (weight) function 248
5.2.2.4 Numerical integration 248
5.3 Meshless Local Petrov-Galerkin method 250
5.3.1 MLPG formulation 250
5.3.2 Enforcement of essential boundary conditions 252
5.3.3 Commons on the efficiency of MLPG and LRPIM 253
5.3.3.1 Comparison with FEM 254
5.3.3.2 Comparison with MFree global weak-form methods 254
5.4 Source code 254
5.4.1 Implementation issues 254
5.4.2 Program description and data structures 256
5.5 Examples for two dimensional solids – a cantilever beam 262
5.5.1 The use of the MFree_local.f90 262
5.5.2 Studies on the effects of parameters 267
5.5.2.1 Parameters effects on LRPIM 268
5.5.2.2 Parameter effects on MLPG 274
5.5.3 Comparison of convergence 276
5.5.4 Comparison of efficiency 278
5.6 Remarks 279
Appendix 281
Computer programs 292

6 Meshfree collocation methods 310

6.1 Introduction 310
6.2 Techniques for handling derivative boundary conditions 311

6.3 Polynomial point collocation method for 1D problems 313
6.3.1 Collocation equations for 1D system equations..... 313
6.3.1.1 Problem description 313
6.3.1.2 Function approximation using MFree shape functions 314
6.3.1.3 System equation discretization 315
6.3.1.4 Discretization of Dirichlet boundary condition 316
6.3.1.5 Discretized system equation with only Dirichlet boundary conditions 316
6.3.1.6 Discretized system equations with DBCs 317
6.3.2 Numerical examples for 1D problems 323
6.4 Stabilization in convection-diffusion problems using MFree methods 335
6.4.1 Nodal refinement 338
6.4.2 Enlargement of the local support domain 338
6.4.3 Total upwind support domain 339
6.4.4 Adaptive upwind support domain 341
6.4.5 Biased support domain..... 342
6.5 Polynomial point collocation method for 2D problems 343
6.5.1 PPCM formulation for 2D problems..... 344
6.5.2 Numerical examples 346
6.6 Radial point collocation method for 2D problems 352
6.6.1 RPCM formulation 352
6.6.2 RPCM for 2D Poisson equations 352
6.6.3 RPCM for 2D convection-diffusion problems..... 354
6.6.3.1 Steady state convection-diffusion problem..... 354
6.6.3.2 Linear dynamic convection-diffusion equations..... 359
6.7 RPCM for 2D solids..... 364
6.7.1 Hermite-type RPCM 364
6.7.2 Use of regular grid (RG)..... 371
6.8 Remarks 378

7 Meshfree methods based on local weak form and collocation.....380

7.1 Introduction..... 380
7.2 Meshfree collocation and local weak-form methods 381
7.2.1 Meshfree collocation method..... 381
7.2.2 Meshfree weak-form method..... 382
7.2.3 Comparisons of meshfree collocation and weak-form methods..... 383
7.3 Formulation for 2-D statics 384
7.3.1 The idea 384
7.3.2 Local weak-form..... 386
7.3.3 Discretized system equations..... 387

7.3.4 Numerical implementation 390
7.3.4.1 Property of stiffness matrix 390
7.3.4.2 Type of local domains 391
7.3.4.3 Numerical integration 391
7.4 Source code 391
7.4.1 Implementation issues 392
7.4.2 Program description 392
7.5 Examples for testing the code 393
7.6 Numerical examples for 2D elastostatics 400
7.6.1 1D truss member with derivative boundary conditions 400
7.6.2 Standard patch test 401
7.6.3 Higher-order patch test 403
7.6.4 Cantilever beam 407
7.6.5 Hole in an infinite plate 410
7.7 Dynamic analysis for 2-D solids 410
7.7.1 Strong-form of dynamic analysis 412
7.7.2 Local weak-form for the dynamic analysis 412
7.7.3 Discretized formulations for dynamic analysis 413
7.7.3.1 Free vibration analysis 414
7.7.3.2 Direct analysis of forced vibration 415
7.7.4 Numerical examples 416
7.7.4.1 Free vibration analysis 417
7.7.4.2 Forced vibration analysis 417
7.8 Analysis for incompressible flow problems 423
7.8.1 Simulation of natural convection in an enclosed domain 423
7.8.1.1 Governing equations and boundary conditions 423
7.8.1.2 Discretized system equations 424
7.8.1.3 Numerical results for the problem of natural convection 427
7.8.2 Simulation of the flow around a cylinder 434
7.8.2.1 Governing equation and boundary condition 434
7.8.2.2 Computation procedure 437
7.8.2.3 Results and discussion 437
7.9 Remarks 443
Appendix 445
Computer programs 450

Reference 454

Index 473

Preface

The finite difference method (FDM) has been used to solve differential equation systems for centuries. The FDM works well for problems of simple geometry and was widely used before the invention of the much more efficient, robust *finite element method (FEM)*. FEM is now widely used in handling problems with complex geometry. Currently, we are using and developing even more powerful numerical techniques aiming to obtain more accurate approximate solutions in a more convenient manner for even more complex systems. The meshfree or meshless method is one such phenomenal development in the past decade, and is the subject of this book.

There are many MFree methods proposed so far for different applications. Currently, three monographs on MFree methods have been published.

- *Mesh Free Methods, Moving Beyond the Finite Element Method* by GR Liu (2002) provides a systematic discussion on basic theories, fundamentals for MFree methods, especially on MFree weak-form methods. It provides a comprehensive record of well-known MFree methods and the wide coverage of applications of MFree methods to problems of solids mechanics (solids, beams, plates, shells, etc.) as well as fluid mechanics.
- *The Meshless Local Petrov-Galerkin (MLPG) Method* by Atluri and Shen (2002) provides detailed discussions of the meshfree local Petrov-Galerkin (MLPG) method and its variations. Formulations and applications of MLPG are well addressed in their book.
- *Smooth Particle Hydrodynamics; A Meshfree Particle Method* by GR Liu and Liu (2003) provides detailed discussions of MFree particle methods, specifically smoothed particle hydrodynamics (SPH) and some of its variations. Applications of the SPH method in fluid mechanics, penetration, and explosion have also been addressed in this book, and a general computer source code of SPH for fluid mechanics is provided.

Readers may naturally question the purpose of this book and the difference between this book and others, especially that by GR Liu (2002).

The second and the third books are related to specific MFree methods, which have clearly different scopes from this book. The book by GR Liu (2002) is the first book published with a comprehensive coverage on many major MFree methods. It covers all the relatively more mature meshfree methods based on weak-form formulations with systematic description and broad applications to solids, beams, plates, shell, fluids, etc. However, the starting point in that book is relatively high. It requires a relatively strong background on mechanics as well as numerical simulations. In addition, some expressions in this book were not given in detail, and no computer source code was provided, because of space limitation.

After the publication of the first book, the first author received many constructive comments, including requests for source codes and for more detailed descriptions on fundamental issues. This book is therefore intended to complement the first book and provide the reader with more details of the fundamentals of meshfree methods accompanied with detailed explanation on the implementation and coding issues together with the source codes. This book covers only the very basics of meshfree weak-form methods, but provides intensive details on meshfree methods based on the strong-form and weak-strong-form formulations. The relationship of this book and the book by GR Liu (2002) is detailed in Table 0.1. This shows that there is very little duplication of materials between the two; they are complementary. The authors hope that this monograph will help beginning researchers, engineers and students have a smooth start in their study and further exploration of meshfree techniques.

The purpose of this book is, hence, to provide the fundamentals of MFree methods in as much detail as possible. Some typical MFree methods, such as EFG, MLPG, RPIM, and LRPIM, are discussed in great detail. The detailed numerical implementations and programming for these methods are also provided. In addition, the MFree collocation (strong-form) methods are also detailed. Many well-tested computer source codes for MFree methods are provided. The application and the performance of the codes provided can be checked using the examples attached. Input and output files are provided in table form for easy verification of the codes. All computer codes are developed by the authors based on existing numerical techniques for FEM and the standard numerical analysis. These codes consist of most of the basic MFree techniques, and can be easily extended to other variations of more complex procedures of MFree methods.

Releasing this set of source codes is to suit the needs of readers for an easy comprehension, understanding, quick implementation, practical applications of the existing MFree methods, and further improvement and

Table 0.1. The relationship between this book and the meshfree method book by GR Liu (2002)

Topics	Book by GR Liu (2002)		This book	
	Content	Source code	Content	Source code
Weighted residual methods	Briefed	NA	Detailed explicitly with 1D examples	NA
Weak-forms	Detailed	NA	Briefed	NA
MFree shape functions	Detailed with emphasizes on MLS, PIM and RPIM	No	Detailed for MLS, PIM WLS, RPIM, and Hermite-type	Provided
MFree global weak-form methods	Detailed for EFG, PIM and RPIM	No	Detailed for EFG and RPIM	Provided
MFree local Petrov-Galerkin weak-form methods	Detailed for MLPG, LPIM and LRPIM	No	Detailed for MLPG and LRPIM	Provided
MFree collocation methods	No	No	Detailed for various techniques	No
MFree weak-strong form methods	No	No	Detailed for MWS-LS and MWS-RPIM	Provided
Boundary-type MFree methods	Detailed for BPIM and BRPIM	No	No	NA
Coupled methods	Detailed for EFG/BEM, MLPG/FEM/BEM	No	No	NA
SPH	Detailed for fluid mechanics problems	No	No	NA
Applications to solids	1D and 2D solids	No	1D, 2D and 3D solids	Partially provided
Applications to beam, plate and shell structures	Yes	No	No	NA
Applications to fluid mechanics problems	Detailed for SPH, MLPG and LRPIM	No	Detailed using MWS	No
Material non-linear problems	Yes	No	No	NA
Geometric non-linear problems	No	NA	Provided examples of RPIM	No
Convection-dominated problems	No	No	Detailed for 1D and 2D problems using MFree strong-form methods	No
MFree2D©	Detailed for usage and techniques used	No	No	NA

NA: not applicable.

development of their own MFree methods. All source codes provided in this book are developed and tested based on the MS Windows and MS Developer Studio 97 (Visual FORTRAN Professional Edition 5.0.A) on a personal computer. After slight revisions, these programs can also be executed in other platforms and systems, such as the UNIX system on workstations. In our research group these codes are frequently ported between the Windows and UNIX systems, and there has been no technical problem.

Outline of this book

Chapter 1: The weighted residual methods are introduced and discussed. Various numerical approaches derived from the weighted residual method are introduced and examined using 1D examples. The fundamental and theories of solid mechanics and weak-forms are also provided.

Chapter 2: An overview of MFree methods is provided, including the background, classifications, and basic procedures in MFree methods.

Chapter 3: Fundamental and theories of MFree interpolation /approximation schemes for shape function construction, especially, MLS, PIM, WLS, and RPIM, and Hermite-type shape functions, are systemically introduced. Source codes of two standard subroutines of computing MLS and RPIM shape functions are provided.

Chapter 4: Formulations of the MFree global weak-form methods, EFG and RPIM, are presented in detail. A standard source code of RPIM and EFG is provided.

Chapter 5: Formulations of the MFree local weak-form methods, MLPG and LRPIM, are presented in great detail. A standard source code of LRPIM is provided.

Chapter 6: Fundamentals and procedures of the MFree collocation methods are systemically discussed. The issues related to the stability and accuracy in the strong-form methods are discussed in detail. In particular, the effects of the presence of the derivative boundary conditions are examined in great detail.

Chapter 7: The MFree methods based on combination of local weak form and collocation are derived and discussed in detail. A standard source code is provided.

The book is written for senior university students, graduate students, researchers, professionals in engineering and science. Readers of this book can be any one from a beginner student to a professional researcher as well as engineers who are interested in learning and applying MFree methods to solve their problems. Knowledge of the finite element method is not required but it would help in the understanding and comprehension of many concepts and procedures of MFree methods. Basic knowledge of solids mechanics would also be helpful. The codes provided for practise might be the most effective way to learn the basics of MFree methods.

Acknowledgement

The authors' work in the area of meshfree methods discussed in this book has been profoundly influenced by the works by Prof. T. Belytschko, Prof. S. N. Atluri, and others. Without their significant contributions in this area, this book would not exist.

Many of our colleagues and students have supported and contributed to the writing of this book. The authors would like to express their sincere thanks to all of them. Special thanks to X. Liu, Y.L. Wu, K.Y. Dai, L. Yan, G.Y. Zhang, etc. Many of them have contributed examples to this book in addition to their hard work in carrying out a number of projects related to meshfree methods at the Centre for Advanced Computations in Engineering Science (ACES). Special thanks also go to Y. Liu, Bernard Kee, Jerry Quek, etc. for reading the drafts of this thick volume and providing very useful editorial comments.

The authors are grateful to Professor Gladwell for editing the manuscript; his constructive comments and suggestions improved readability of the book.

Finally, the authors would also like to thank A*STAR, Singapore, and the National University of Singapore for their partial financial sponsorship in some of the research projects undertaken by the authors and their teams related to the topic of this book.

G.R. Liu

Y.T. Gu

Authors

Dr. G.R. Liu received his PhD from Tohoku University, Japan in 1991. He was a Postdoctoral Fellow at Northwestern University, U. S. A. He is currently the Director of the Centre for Advanced Computations in Engineering Science (ACES), National University of Singapore. He serves as the President of the Association for Computational Mechanics (Singapore). He is also an Associate Professor at the Department of Mechanical Engineering, National University of Singapore. He has provided consultation services to many national and international organizations. He authored more than **300** technical publications including more than 200 international journal papers and six authored books, including the popular book "Mesh Free Method: moving beyond the finite element method", and a bestseller "Smooth Particle Hydrodynamics-a meshfree particle method". He is the Editor-in-Chief of the International Journal of Computational Methods and an editorial member of a number of other journals. He is the recipient of the **Outstanding University Researchers Awards** (1998), the **Defence Technology Prize** (National award, 1999), the **Silver Award** at **CrayQuest 2000** Nationwide competition, the **Excellent Teachers** (2002/2003) title, the **Engineering Educator Award** (2003), and the **APCOM Award for Computational Mechanics** (2004). His research interests include Computational Mechanics, Mesh Free Methods, Nano-scale Computation, Micro bio-system computation, Vibration and Wave Propagation in Composites, Mechanics of Composites and Smart Materials, Inverse Problems and Numerical Analysis.

Dr. Y.T. Gu received his B.E. and M. E. degrees from Dalian University of Technology (DUT), China in 1991 and 1994, respectively, and received his PhD from the National University of Singapore (NUS) in 2003. He is currently a research fellow at the Department of Mechanical Engineering in NUS. He has conducted a number of research projects related to meshfree methods, and he has authored more than 40

technical publications including more than 20 international journal papers. His research interests include Computational Mechanics, Finite Element Analysis and Modeling, Meshfree (meshless) Methods, Boundary Element Method, Mechanical Engineering, Ship and Ocean Engineering, Computational Microelectromechanical Systems (MEMS), High Performance Computing Techniques, Dynamic and Static Analyses of Structures, etc.

Chapter 1

FUNDAMENTALS

This chapter provides the fundamentals of mechanics for solids, as this type of problems will be frequently dealt with in this book. Several widely used numerical approximation methods are outlined in a concise manner using one dimensional (1D) problems to address fundamental issues in numerical methods. Readers with experience in mechanics and numerical methods may skip this chapter, but this chapter introduces the terms used in the book.

1.1 NUMERICAL SIMULATION

Phenomena in nature, whether mechanical, geological, electrical, chemical, electronic, or biological, can often be described by means of algebraic, differential, or integral equations. One would like to obtain exact solutions analytically for these equations. Unfortunately, we can only obtain exact solutions for small parts of practical problems because most of these problems are complex; we must use numerical procedures to obtain approximate solutions. Nowadays, engineers and scientists have to be conversant with numerical techniques for different types of problems. Because of the rapid development of computer technology, numerical simulation techniques using computers (or computational simulation) have increasingly become an important approach for solving complex and practical problems in engineering and science.

The main idea of numerical simulation is to transform a complex practical problem into a simple discrete form of mathematical description, recreate and solve the problem on a computer, and finally reveal the phenomena virtually according to the requirements of the analysts. It is often possible to find a numerical or approximate solution for a complex problem efficiently, as long as a proper numerical method is used.

Numerical simulations follow a similar procedure to serve a practical purpose. There are necessary steps in the procedure, as shown in Figure 1.1.

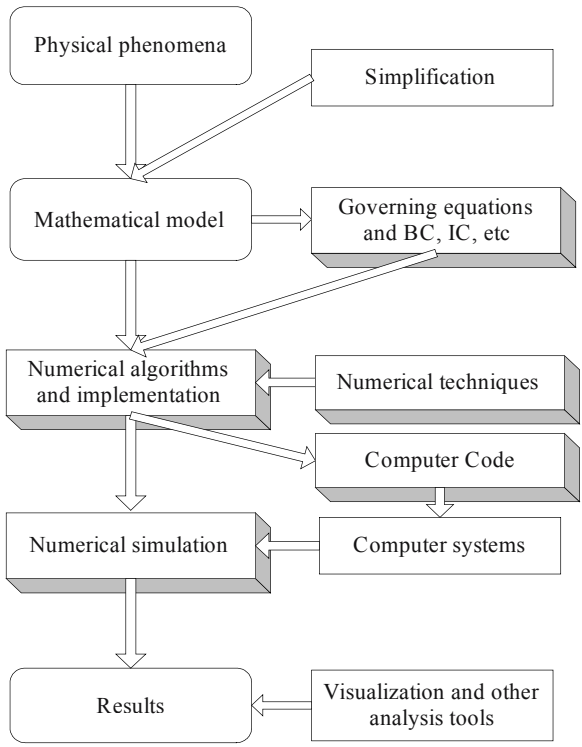


Figure 1.1. Procedure of conducting a numerical simulation. This book deals with topics related to the items in the shaded frames.

Step 1: Identity and isolate the physical phenomenon;

Step 2: Establish mathematical models for this phenomenon with some possible simplifications and acceptable assumptions. These mathematical models are generally expressed in terms of *field variables* in governing

equations with proper boundary conditions (BCs) and/or initial conditions (ICs). The governing equations are usually a set of ordinary differential equations (ODEs), partial differential equations (PDEs), or integral equations. Boundary and/or initial conditions are needed to complement the governing equations for determining the field variables in space and/or time. This step is the base for a numerical simulation.

Step 3: Describe the mathematical model in a proper numerical procedure and algorithm. The major aim of this step is to produce computer code performing the numerical simulation. For different numerical techniques, the numerical algorithm and implementation are different, and hence the computer codes are also different.

Step 4: Numerically simulate the problem. Te computer systems and the computer codes obtained in Step 3 are used to simulate the practical problem.

Step 5: Observe and analyze the simulation results that are obtained in Step 4. Visualization software packages are often very useful tools for presenting the data produced by computers as they are usually complex in nature and large in volume.

In this procedure, we find that a numerical technique determines the algorithm and codes used in the numerical simulation. In order to obtain a successful simulation result representing the true physics, we need a reliable and efficient numerical technique. Many researchers have been developing the numerical techniques or numerical approximation methods. Several efficient approximation methods have been proposed and developed so far, such as the finite difference method (FDM), the finite element method (FEM), the boundary element method (BEM), and the *meshless* or *meshfree* methods (shortened as *MFree methods* in this book)[†] to be discussed in this book.

1.2 BASICS OF MECHANICS FOR SOLIDS

In this book, MFree formulations are presented mainly for mechanics problems of solids and fluid flows. In this section, the basic equations of solids are briefly introduced for future reference.

[†] A detailed definition of MFree methods will be presented in Chapter 2.

1.2.1 Equations for three-dimensional solids

1.2.1.1 Stress components

Consider a continuum of three-dimensional (3D) elastic solids with a volume Ω and a surface boundary Γ, as shown in Figure 1.2. The solid is supported at various locations and is subjected to external forces that may be distributed over the volume or/and on the boundary. When the solid is *stressed*, it will deform resulting in a *displacement* field. The field variables of interest are the displacements. The displacements and the stress level can be different from point to point in the solid depending on the configuration of solid, loading, and boundary conditions.

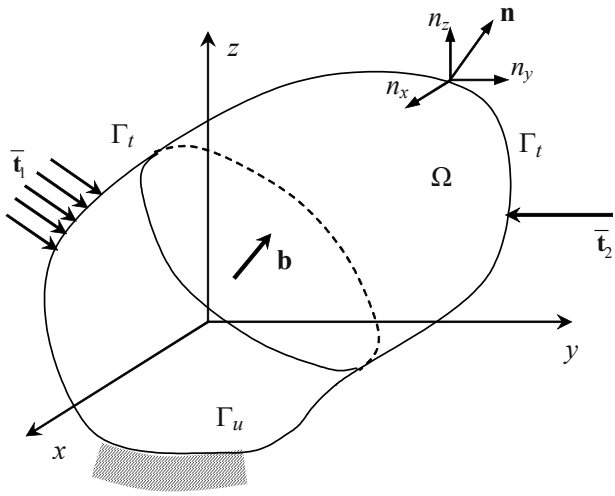


Figure 1.2. A continuum of solids.
Ω: the problem domain considered; Γ: the global boundary of the problem domain; Γ_t: the traction boundary (or force, derivative, natural boundary); Γ_u: the displacement boundary (or Dirichlet, essential boundary); $\mathbf{n}=\{n_x, n_y, n_z\}^T$: the outward normal vector on the boundary.

At any point in the solid, there are, in general, six components of stress to describe the state stressed, as indicated on the surface of a small cubic "cell" shown in Figure 1.3. On each surface, there will be one component of normal stress, and two components of shear stress. The sign convention for the subscript is that the first letter represents the surface on which the stress is acting, and the second letter represents the direction of the stress. Note that there are also stresses acting on the other three hidden surfaces. As the normal to these surfaces are in the directions opposite to the corresponding coordinates, positive directions of the stresses should also be in the directions

opposite to the coordinates. There are a total of nine stress components shown on the cubic cell. These nine components are the components of the stress tensor. By taking moments of forces about the central axes of the cubic cell at the state of equilibrium, it is easy to confirm that

$$\sigma_{xy} = \sigma_{yx}\ ;\ \sigma_{xz} = \sigma_{zx}\ ;\ \sigma_{zy} = \sigma_{yz} \tag{1.1}$$

Therefore, there are six independent stress components in total at a particular point in a solid. The stresses are often written in the vector form

$$\boldsymbol{\sigma}^{\mathrm{T}} = \left\{ \sigma_{xx} \quad \sigma_{yy} \quad \sigma_{zz} \quad \sigma_{yz} \quad \sigma_{xz} \quad \sigma_{xy} \right\} \tag{1.2}$$

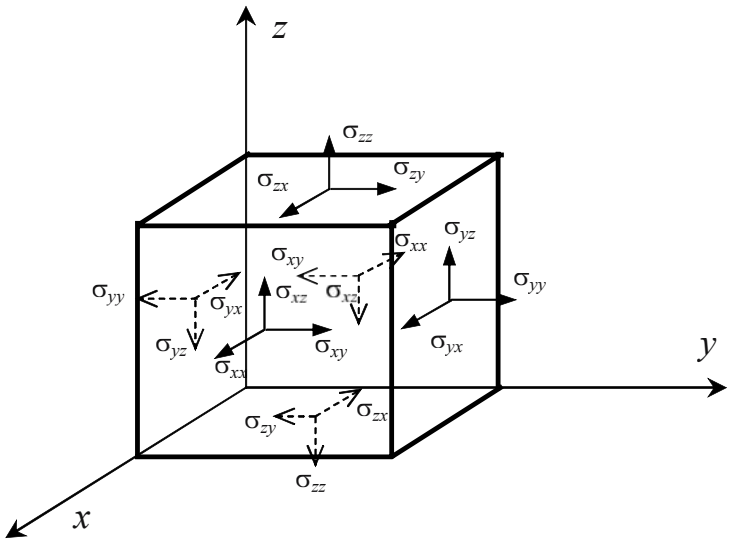


Figure 1.3. Stress components on a small cubic cell in a stressed three-dimensional solid.

1.2.1.2 Strain-displacement equations

The strain-displacement equation gives the relationship between displacements and strains. There are six strain components at a point in solids corresponding to the six stress components, which can also be written in a similar vector form of

$$\boldsymbol{\varepsilon}^{\mathrm{T}} = \left\{ \varepsilon_{xx} \quad \varepsilon_{yy} \quad \varepsilon_{zz} \quad \varepsilon_{yz} \quad \varepsilon_{xz} \quad \varepsilon_{xy} \right\} \tag{1.3}$$

A strain is a rate of displacement per unit length. The components of strain can be obtained by derivatives of the displacements for small

deformation in solids. The strain-displacement relation can be written in the following matrix form.

$$\boldsymbol{\varepsilon} = \mathbf{L}\mathbf{u} \tag{1.4}$$

where **u** is the displacement vector having the form of

$$\mathbf{u} = \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{1.5}$$

where u, v and w are displacement components in x, y and z directions, respectively.

In Equation (1.4), **L** is a matrix differential operator given by

$$\mathbf{L} = \begin{bmatrix} \partial/\partial x & 0 & 0 \\ 0 & \partial/\partial y & 0 \\ 0 & 0 & \partial/\partial z \\ 0 & \partial/\partial z & \partial/\partial y \\ \partial/\partial z & 0 & \partial/\partial x \\ \partial/\partial y & \partial/\partial x & 0 \end{bmatrix} \tag{1.6}$$

1.2.1.3 Constitutive equations

The constitutive equation gives the relationship between the stress and the strain for a given material. It is often called a generalized Hooke's law. The generalized Hooke's law for general anisotropic elastic materials can be given in the following matrix form.

$$\boldsymbol{\sigma} = \mathbf{D}\,\boldsymbol{\varepsilon} \tag{1.7}$$

where **D** is a matrix of material constants, which have to be obtained through experiments. The constitutive equation can be written explicitly as

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{xz} \\ \sigma_{xy} \end{Bmatrix} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & D_{15} & D_{16} \\ & D_{22} & D_{23} & D_{24} & D_{25} & D_{26} \\ & & D_{33} & D_{34} & D_{35} & D_{36} \\ & & & D_{44} & D_{45} & D_{46} \\ & sy. & & & D_{55} & D_{56} \\ & & & & & D_{66} \end{bmatrix} \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{yz} \\ \varepsilon_{xz} \\ \varepsilon_{xy} \end{Bmatrix} = \mathbf{D}\,\boldsymbol{\varepsilon} \tag{1.8}$$

Note that D_{ij}=D_{ji}. There are a total of 21 possible independent material constants D_{ij}. For different types of anisotropic materials, there will be fewer independent material constants (see, e.g., GR Liu and Xi, 2001). For isotropic material, which is the simplest type of material, **D** can be gradually reduced to

$$\mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{12} & 0 & 0 & 0 \\ & D_{11} & D_{12} & 0 & 0 & 0 \\ & & D_{11} & 0 & 0 & 0 \\ & & & (D_{11}-D_{12})/2 & 0 & 0 \\ & sy. & & & (D_{11}-D_{12})/2 & 0 \\ & & & & & (D_{11}-D_{12})/2 \end{bmatrix} \tag{1.9}$$

where

$$D_{11} = \frac{E(1-\nu)}{(1-2\nu)(1+\nu)}; D_{12} = \frac{E\nu}{(1-2\nu)(1+\nu)}; \frac{D_{11}-D_{12}}{2} = G \tag{1.10}$$

in which E, ν and G are Young's modulus, Poisson's ratio, and shear modulus of the material, respectively. There are only two independent constants among these three constants:

$$G = \frac{E}{2(1+\nu)} \tag{1.11}$$

1.2.1.4 Equilibrium equations

The equilibrium equation gives the relationship between the stress and the external force. Using equilibrium conditions of forces in a small block in a solid, we can obtain the following equilibrium equations in a concise matrix form for three-dimensional elastodynamics.

$$\mathbf{L}^{\mathrm{T}}\boldsymbol{\sigma} + \mathbf{b} = \rho\ddot{\mathbf{u}} + c\dot{\mathbf{u}} \tag{1.12}$$

where ρ is the mass density, c is the damping coefficient, $\ddot{\mathbf{u}} = \dfrac{\partial^2 \mathbf{u}}{\partial t^2}$ is the acceleration vector**,** $\dot{\mathbf{u}} = \dfrac{\partial \mathbf{u}}{\partial t}$ is the velocity vector, and **b** is the vector of external body forces in x, y, and z directions:

$$\mathbf{b} = \begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} \tag{1.13}$$

Using Equations (1.4) and (1.7), we can write the dynamic equilibrium Equation (1.12) in terms of displacements:

$$\mathbf{L}^{\mathrm{T}}\mathbf{DLu} + \mathbf{b} = \rho\ddot{\mathbf{u}} + c\dot{\mathbf{u}} \tag{1.14}$$

This is the general form of the dynamic equilibrium equation for three-dimensional elasticity. If the loads applied on the solid are static, then the concern is only on the static status of the solid, and the static equilibrium equation can be obtained simply by dropping the dynamic terms in Equation (1.14), which yields

$$\mathbf{L}^{\mathrm{T}}\mathbf{DLu} + \mathbf{b} = \mathbf{0} \tag{1.15}$$

Equation (1.12) can also be written in the following form using the tensor notation.

$$\sigma_{ij,j} + b_i = \rho\ddot{u}_i + c\dot{u}_i \tag{1.16}$$

where i, j=(1, 2, 3) representing, respectively, x, y and z directions.

Equation (1.12) or Equation (1.16) is the equilibrium equation of three-dimensional elastodynamics. The equilibrium equation is often called the *governing equation* for solids; it is a partial differential equation (PDE) with the displacement vector as the unknown function of field variables.

1.2.1.5 Boundary conditions and initial conditions

The governing Equation (1.12) or Equation (1.16) must be complemented with boundary conditions and initial conditions.

Traction boundary condition: $\sigma_{ij}n_j = \overline{t}_i$ on Γ_t (1.17)

Displacement boundary condition: $u_i = \overline{u}_i$ on Γ_u (1.18)

Displacement initial condition: $\mathbf{u}(\mathbf{x}, t_0) = \mathbf{u}_0(\mathbf{x})$ $\mathbf{x} \in \Omega$ (1.19)

Velocity initial condition: $\dot{\mathbf{u}}(\mathbf{x}, t_0) = \mathbf{v}_0(\mathbf{x})$ $\mathbf{x} \in \Omega$ (1.20)

where $\overline{u}_i$, $\overline{t}_i$, $\mathbf{u}_0$ and $\mathbf{v}_0$ denote the prescribed displacements, tractions, initial displacements and velocities, respectively, and n_j is a component of the vector of the unit outward normal on the boundary of the domain Ω (see

Figure 1.2). The traction boundary condition is, in general, a type of *derivative boundary condition* or *natural boundary condition* (in the weak-form context). The displacement boundary conditions are often called the *Dirichlet* or *essential boundary conditions* in the weak-form context.

In summary, the governing equation (Equation (1.12) or Equation (1.16)), the constitutive equation (Equation (1.7)) and the strain-displacement equation (1.4) together with boundary conditions and initial conditions (Equations (1.17)~(1.20)) form a *boundary value problem* (BVP) and the *initial value problem* (IVP) for three-dimensional solids. The entire set of equations is called system equations.

Note that equations obtained in this section are applicable to 3D elastic solids. Theoretically, these equations for 3D solids can be applied to all other types of structures such as trusses, beams, plates and shells, because they are all made of 3D solids. However, treating all the structural components as 3D solid makes computation very expensive, and practically impossible. Therefore, theories for making good use of the geometrical advantage of different types of solids and structural components have been developed. Application of these theories in a proper manner can reduce analytical and computational effort drastically.

1.2.2 Equations for two-dimensional solids

1.2.2.1 Stress components

For two-dimensional (2D) solids as shown in Figure 1.4, it is assumed that the geometry of the domain is independent of z-axis, and all the external loads and supports are independent of the z coordinate, and applied only in the x-y plane. This assumption reduces the 3D equations to 2D equations. There are two types of typical states of 2D solids. One is *plane stress*, and another is *plane strain*. Plane stress solids are solids whose thickness in the z direction is very small compared with dimensions in the x and y directions. As external forces are applied only in the x-y plane, and stresses in z direction (σ_{zz}, σ_{xz}, σ_{yz}) are all zero. There are only three in-plane stresses, (σ_{xx}, σ_{yy}, σ_{xy}).

Plane strain solids are solids whose thickness in the z direction is very large compared with dimensions in the x and y directions. External forces are applied uniformly along the z-axis, and the movement in the z direction at any point is constrained. The strain components in z direction (ε_{zz}, ε_{xz}, ε_{yz}) are all zero, there are only three in-plane strains, (ε_{xx}, ε_{yy}, ε_{xy}) to deal with.

The system equations for 2D solids can be obtained by simply omitting the terms related to the z direction in the system equations for 3D solids. Equations for isotropic materials are given as follows.

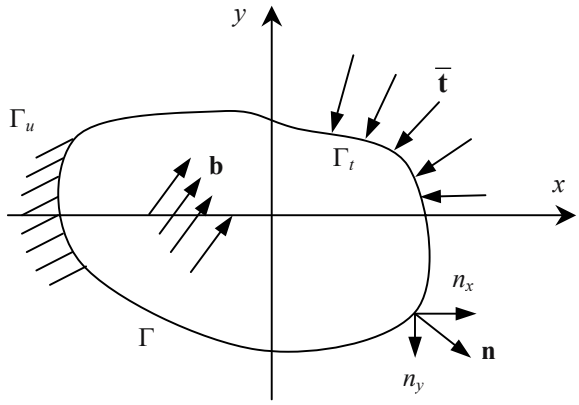


Figure 1.4. A two-dimensional continuum of solids.
Ω: the problem domain considered; Γ: the global boundary of the problem domain; Γ_t: the traction boundary (or force boundary); Γ_u: the displacement boundary; $\mathbf{n}=\{n_x, n_y\}^T$: the outward normal vector on the boundary.

The stress components are

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{Bmatrix} \tag{1.21}$$

where the shear stress component, σ_{xy}, is often denoted τ_{xy}.

There are three corresponding strain components at any point in 2D solids, which can also be written in a similar vector form

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{Bmatrix} \tag{1.22}$$

1.2.2.2 Strain-displacement equation

The strain-displacement relation can also be written in the following matrix form.

$$\boldsymbol{\varepsilon} = \mathbf{L}\mathbf{u} \tag{1.23}$$

where the displacement vector is

$$\mathbf{u} = \begin{Bmatrix} u \\ v \end{Bmatrix} \tag{1.24}$$

and the differential operator matrix, **L**, is given by

$$\mathbf{L} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix} \tag{1.25}$$

1.2.2.3 Constitutive equations

Hooke's law for 2D elastic solids has the following matrix form:

$$\boldsymbol{\sigma} = \mathbf{D}\,\boldsymbol{\varepsilon} \tag{1.26}$$

where **D** is a matrix of material constants, which have to be obtained through experiments. For isotropic materials in the plane stress state, we have

$$\mathbf{D} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \qquad \text{(Plane stress)} \tag{1.27}$$

For solids in the plane strain state, the matrix of material constants **D** can be obtained by simply replacing E and ν, respectively, with $E/(1-\nu^2)$ and $\nu/(1-\nu)$, which leads to

$$\mathbf{D} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1 & \dfrac{\nu}{1-\nu} & 0 \\ \dfrac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \dfrac{1-2\nu}{2(1-\nu)} \end{bmatrix} \qquad \text{(Plane strain)} \tag{1.28}$$

1.2.2.4 Equilibrium equations

The equilibrium equations for 2D elastic solids can be easily obtained by removing the terms and omitting the differential operations related to the z coordinate from Equation (1.12), i.e.,

$$\mathbf{L}^{\mathrm{T}}\boldsymbol{\sigma} + \mathbf{b} = \rho\ddot{\mathbf{u}} + c\dot{\mathbf{u}} \tag{1.29}$$

where $\mathbf{b}$ is the external force vector given by

$$\mathbf{b} = \begin{Bmatrix} b_x \\ b_y \end{Bmatrix} \tag{1.30}$$

Equation (1.29) has exactly the same form as Equation (1.12). For static problems, the equilibrium equations can be written as

$$\mathbf{L}^{\mathrm{T}}\boldsymbol{\sigma} + \mathbf{b} = 0 \tag{1.31}$$

Equation (1.29) or (1.31) is much easier to solve then their counterpart equations for 3D solids. Equation (1.29) can be also written in the following form using tensor notations:

$$\sigma_{ij,j} + b_i = \rho\ddot{u}_i + c\dot{u}_i \tag{1.32}$$

where i, j=(1, 2) represent, respectively, x and y directions, ρ is the mass density, c is the damping coefficient, u_i is the displacement, $\ddot{u}_i = \dfrac{\partial^2 u_i}{\partial t^2}$ is the acceleration, $\dot{u}_i = \dfrac{\partial u_i}{\partial t}$ is the velocity, σ_{ij} is the stress, b_i is the body force, and $(\,)_{,j}$ denotes $\dfrac{\partial}{\partial x_j}$.

1.2.2.5 Boundary conditions and initial conditions

The boundary conditions and initial conditions can be written as

Traction boundary condition: $\sigma_{ij} n_j = \overline{t}_i$ on Γ_t (1.33)

Displacement boundary condition: $u_i = \overline{u}_i$ on Γ_u (1.34)

Displacement initial condition: $\mathbf{u}(\mathbf{x}, t_0) = \mathbf{u}_0(\mathbf{x})$ $\mathbf{x} \in \Omega$ (1.35)

Velocity initial condition: $$\dot{\mathbf{u}}(\mathbf{x},t_0)=\mathbf{v}_0(\mathbf{x}) \quad \mathbf{x}\in\Omega \tag{1.36}$$

in which $\overline{u}_i$, $\overline{t}_i$, $\mathbf{u}_0$ and $\mathbf{v}_0$ denote the prescribed displacements, tractions, initial displacements and velocities, respectively, and n_j is the component of the unit outward normal vector on the boundary (see Figure 1.4).

In summary, the governing equation, the constitutive equation, and the strain-displacement equation together with the boundary conditions and initial conditions form a set of system equations defining the boundary value problem (BVP) and the initial value problem (IVP) for two-dimensional solids.

1.3 STRONG-FORMS AND WEAK-FORMS

Partial differential equations (PDEs) developed in Section 1.2 are *strong-forms* of system equations. Obtaining the *exact* solution for a strong-form of system equation is ideal, but unfortunately it is very difficult for practical engineering problems that are usually complex in nature. One example of a strong-form numerical method is the widely used finite difference method (FDM). FDM uses the *finite differential representation* (Taylor series) of a function in a local domain and solves system equations of strong-form to obtain an *approximate* solution. However, FDM requires a regular mesh of grids, and can usually work only for problems with simple and regular geometry and boundary conditions. In a strong-form formulation, it is assumed that the approximate unknown function (u, v, w in this case) should have sufficient degree of consistency, so that it is differentiable up to the order of the PDEs.

The *weak-form*, in contrast to the strong-form, requires a weaker consistency on the approximate function. This is achieved by introducing an integral operation to the system equation based on a mathematical or physical principle. The weak-form provides a variety of ways to formulate methods for approximate solutions for complex systems. Formulation based on weak-forms can usually produce a very stable set of discretized system equations that produces much more accurate results.

This book will use weak-form formulations to form discretized system equations of MFree weak-form methods† for mechanics problems of solids

† A detailed discussion of the categories for mesh-free methods will be discussed in Chapter 2.

and fluids (see Chapters 4 and 5). The strong-form formulation based on the collocation approach will also be used to formulate the so-called MFree strong-form methods (or MFree collocation method, see Chapter 6). In addition, both of them will be combined to formulate the MFree weak-strong (MWS) form method (see Chapter 7), where the local weak-form is utilized on and near the natural boundary to obtain stabilized solution.

The consistency requirement on the approximate functions for field variables in the weak-form formulation is quite different from that for the strong form. For a 2*k*th order differential governing system equation, the strong-form formulation assumes the field variable possesses a continuity of 2*k*th order. The weak-form formulation, however, requires usually a continuity of only *k*th order.

There are two major categories of principles used for constructing weak-forms: variational and weighted residual methods. The Galerkin weak-form and the Petrov-Galerkin weak-form may be the most widely used approaches for establishing system equations; they are applicable for deriving MFree formulations. Hamilton's principle is often employed to produce approximated system equations for dynamic problems, and is also applicable to MFree methods. The minimum total potential energy principle has been a convenient tool for deriving discrete system equations for FEM and many other types of approximation methods. The weighted residual method is a more general and powerful mathematical tool that can be used for creating discretized system equations for many types of engineering problems. It has been and will still be used for developing new MFree methods. All these approaches will be adapted in this book for creating discretized system equations for various types of MFree methods.

1.4 WEIGHTED RESIDUAL METHOD

The weighted residual method is a general and extremely powerful method for obtaining approximate solutions for ordinary differential equations (ODEs) or partial differential equations (PDEs). Many numerical methods can be based on the general weighted residual method. Hence, this section discusses some of those numerical methods using a simple example problem. This section is written in reference to the text books by Finlayson (1972), Brebbia (1978), Wang and Shao (1996), and Zienkiewicz and Taylor (2000). The materials are chosen, organized and presented for easy reference in describing MFree methods in later chapters.

As discussed in Section 1.2, many problems in engineering and physics are governed by ODEs or PDEs with a set of boundary conditions. Consider the following (partial) differential equation.

$$F(u) + b = 0 \qquad \text{in problem domain } \Omega \tag{1.37}$$

where F is a differential (partial) operator that is defined as a process when applied to the scalar function u produces a function $-b$. The boundary condition is given as

$$G(u) = g \qquad \text{on the boundary } \Gamma \tag{1.38}$$

where G is a differential (partial) operator for the boundary condition.

Most engineering problems which are expressed in ODEs or PDEs can only be solved in an approximate manner, by which the function u is first approximated by

$$u^h(x) = \sum_i^n \alpha_i B_i(x) = \mathbf{B}\boldsymbol{\alpha} \tag{1.39}$$

where $B_i(x)$ is the ith term *basis function* or *trial function*, α_i is the unknown coefficient for the ith term basis function, and n is the number of basis functions used. These basis functions are usually chosen so as to satisfy certain given conditions, called *admissibility conditions*, relating to the essential boundary conditions and the requirement of continuity.

In practice, the number of basis functions used in Equation (1.39), n, is small, hence the governing Equation (1.37) and the boundary conditions, Equation (1.38), cannot usually be satisfied exactly. Substituting Equation (1.39) into Equations (1.37) and (1.38), we generally should have

$$F(u^h) + b \neq 0 \tag{1.40}$$

$$G(u^h) - g \neq 0 \tag{1.41}$$

Hence, we can obtain the following *residual* functions R_s and R_b, respectively, for the system equations defined in the problem domain and the boundary conditions defined on the boundaries.

$$R_s = F(u^h) + b \tag{1.42}$$

$$R_b = G(u^h) - g \tag{1.43}$$

If Equation (1.39) is the exact solution of the governing Equation (1.37) and the boundary conditions Equation (1.38), residuals R_s and R_b will be zero. However, the exact solution is usually unavailable for many practical problems, and R_s and R_b are, in general, not zero. Note that R_s and R_b

change with the approximate functions chosen. We can use some techniques to properly obtain an approximate function so as to make the residual as "small" as possible; we force the residual to zero in an average sense by setting weighted integrals of residuals to zero. For example, we impose

$$\int_{\Omega} \widehat{W}_i R_s \mathrm{d}\Omega + \int_{\Gamma} \widehat{V}_i R_b \mathrm{d}\Gamma = 0 \tag{1.44}$$

where i=1, 2, …, n, $\widehat{W}$ and $\widehat{V}$ are a set of given weight functions for the residuals R_s and R_b, respectively.

Note that the approximate solution, Equation (1.39), can be chosen to satisfy the boundary conditions. In such cases, R_b is zero, and Equation (1.44) becomes

$$\int_{\Omega} \widehat{W}_i R_s \mathrm{d}\Omega = 0 \tag{1.45}$$

This is the formulation of the weighted residual method that is often used in establishing numerical procedures (e.g., the FEM etc.).

Note also that in Equation (1.44), it is possible to use the same weight functions for both $\widehat{W}$ and $\widehat{V}$.

Substituting Equations (1.42) and (1.43) into Equation (1.44), we can obtain

$$\int_{\Omega} \widehat{W}_i \left[F(u^h) + b \right] \mathrm{d}\Omega + \int_{\Gamma} \widehat{V}_i \left[G(u^h) - g \right] \mathrm{d}\Gamma = 0 \tag{1.46}$$

Using Equation (1.39), we have

$$\int_{\Omega} \widehat{W}_i \left[F(\mathbf{B}\boldsymbol{\alpha}) + b \right] \mathrm{d}\Omega + \int_{\Gamma} \widehat{V}_i \left[G(\mathbf{B}\boldsymbol{\alpha}) - g \right] \mathrm{d}\Gamma = 0 \tag{1.47}$$

Equation (1.47) can be re-written more explicitly for i=1, 2, …, n as follows.

$$\left.\begin{aligned}
&\int_{\Omega} \widehat{W}_1 \left[F(\mathbf{B}\boldsymbol{\alpha}) + b \right] \mathrm{d}\Omega + \int_{\Gamma} \widehat{V}_1 \left[G(\mathbf{B}\boldsymbol{\alpha}) - g \right] \mathrm{d}\Gamma = 0 \\
&\int_{\Omega} \widehat{W}_2 \left[F(\mathbf{B}\boldsymbol{\alpha}) + b \right] \mathrm{d}\Omega + \int_{\Gamma} \widehat{V}_2 \left[G(\mathbf{B}\boldsymbol{\alpha}) - g \right] \mathrm{d}\Gamma = 0 \\
&\qquad\qquad \cdots \\
&\int_{\Omega} \widehat{W}_n \left[F(\mathbf{B}\boldsymbol{\alpha}) + b \right] \mathrm{d}\Omega + \int_{\Gamma} \widehat{V}_n \left[G(\mathbf{B}\boldsymbol{\alpha}) - g \right] \mathrm{d}\Gamma = 0
\end{aligned}\right\} \tag{1.48}$$

From Equation (1.48), we can obtain n equations for n unknowns α_i (i=1,2, …, n). Solving these equations, we can obtain α_i, and then obtain the

approximate solution, which makes residuals, R_s and R_b, vanish in an average sense. When 1) the weight functions $\widehat{W}_i$, $\widehat{V}_i$ and the basis functions $B_i(x)$ are linearly independent; 2) the basis functions $B_i(x)$ are continuous of a certain order; 3) the weight functions and the basis function have certain degree of overlapping; 4) and when $n \to \infty$, the approximate solution Equation (1.39) will converge to the exact solution of the problem, if the solution of the problem is unique and continuous.

This is the general form of the *weighted residual method*. It should be noted that Equation (1.48) is a set of integral equations that is obtained from the original ODEs or PDEs. Therefore, the weighted residual method provides a way to transform an ODE or PDE to an integral form.

This integral equation helps to "smear" out the possible error induced by the function approximations, so as to stabilize the solution and improve the accuracy. The integral operation can also reduce the requirement for the order of continuity on the approximate function via integrals by parts to reduce the order of the differential operators. It is termed a *weak-form*, meaning that it weakens the requirement for continuity on the approximate function.

In the weighted residual method, the selection of weight functions will affect its performance. Different numerical approximation methods can be obtained by selecting different weight functions. In the following sub-sections, several such methods are discussed.

1.4.1 Collocation method

Instead of trying to satisfy the ODE or PDE in an average form, we can try to satisfy them at only a set of chosen points that are distributed in the domain. This is the so-called *collocation method* that seems to be first used by Slater (1934) for problems of electronic energy bounds in metals. Early development and applications of the collocation method include the works by Barta (1937), Frazer et al. (1937), Lanczos (1938), etc. The Lanczos' method, known as the *orthogonal collocation method*, uses Chebyshev polynomials and their roots as collocation points.

The standard formulation of the collocation method can be easily obtained by using *Dirac delta functions* $\delta(x - x_i)$ as the weight functions in Equation (1.44), i.e.,

$$\begin{cases} \widehat{W}_i = \delta(x - x_i) \\ \widehat{V}_i = \delta(x - x_i) \end{cases} \tag{1.49}$$

where i=1,2, …, n, and the Dirac delta function, $\delta(x - x_i)$, has the following property:

$$\begin{cases} \delta(x-x_i)=0, & x \neq x_i \\ \int_{x_i-c}^{x_i+c} \delta(x-x_i)dx=1, & c \to 0 \end{cases} \tag{1.50}$$

Thus we derive the collocation method from the weighted residual formulation by substituting Equation (1.49) into Equation (1.44):

$$\begin{aligned} &\int_{\Omega}\delta(x-x_i)R_s\mathrm{d}\Omega+\int_{\Gamma}\delta(x-x_i)R_b\mathrm{d}\Gamma \\ &=\int_{\Omega}\delta(x-x_i)\left[F(\mathbf{B}\boldsymbol{\alpha})+b\right]\mathrm{d}\Omega+\int_{\Gamma}\delta(x-x_i)\left[G(\mathbf{B}\boldsymbol{\alpha})-g\right]\mathrm{d}\Gamma=0 \end{aligned} \tag{1.51}$$

which becomes:

$$\int_{\Omega}\delta(x-x_i)R_s\mathrm{d}\Omega+\int_{\Gamma}\delta(x-x_i)R_b\mathrm{d}\Gamma=R_s(x_i)+R_b(x_i)=0 \tag{1.52}$$

or

$$\left[F(\mathbf{B}(x_i)\boldsymbol{\alpha})+b\right]+\left[G(\mathbf{B}(x_i)\boldsymbol{\alpha})-g\right]=0 \tag{1.53}$$

Equation (1.52) is applicable to n points chosen in the problem domain, which means that the collocation method forces the residuals to zero at the points x_i (i=1,2, …, n) chosen in the domain.

1.4.2 Subdomain method

The *subdomain method* is similar to the collocation method. The difference is that instead of requiring the residual function to be zero at certain points, we make the integral of the residual function over n regions (or subdomains), Ω_i (i=1,2, …, n), to be zero. This method was first developed by Biezeno and Koch (1923), Biezeno (1923), Biezeno and Grammel (1955). In the subdomain method, we use the weight function that has the following form

$$\widehat{W}_i=\begin{cases} 1, & \text{within } \Omega_i \\ 0, & \text{outside } \Omega_i \end{cases} \tag{1.54}$$

where i=1,2, …, n. Hence, Equation (1.44) becomes

$$\begin{aligned} &\int_{\Omega}\widehat{W}_iR_s\mathrm{d}\Omega+\int_{\Gamma}\widehat{V}_iR_b\mathrm{d}\Gamma=\int_{\Omega_i}\widehat{W}_iR_s\mathrm{d}\Omega+\int_{\Gamma_i}\widehat{V}_iR_b\mathrm{d}\Gamma= \\ &\qquad\int_{\Omega_i}\left[F(\mathbf{B}\boldsymbol{\alpha})+b\right]\mathrm{d}\Omega+\int_{\Gamma_i}\left[G(\mathbf{B}\boldsymbol{\alpha})-g\right]\mathrm{d}\Gamma=0 \end{aligned} \tag{1.55}$$

where Γ_i is the boundary of the intersection between the subdomain Ω_i and the global problem boundary Γ.

Equation (1.55) means that the subdomain method enforces the residuals to zero in a weighted average sense in *n* subdomains chosen in the problem domain.

1.4.3 Least squares method

The *least squares method* (LSM) was originated by Gauss in 1795 and Legendre in 1806 (see, e.g., Hall, 1970; Finlayson 1972). Picone (1928) applied the LSM to solve differential equations. In the LSM, we first define the following functional

$$J(\alpha_i) = \int_{\Omega} R_s \cdot R_s \, \mathrm{d}\Omega \tag{1.56}$$

and then seek for the minimum value of the functional *J*, which requires that

$$\frac{\partial J}{\partial \alpha_i} = \int_{\Omega} \frac{\partial (R_s \cdot R_s)}{\partial \alpha_i} \mathrm{d}\Omega = 2 \int_{\Omega} \frac{\partial (R_s)}{\partial \alpha_i} R_s \, \mathrm{d}\Omega = 0 \tag{1.57}$$

or

$$\int_{\Omega} \frac{\partial (R_s)}{\partial \alpha_i} R_s \, \mathrm{d}\Omega = 0 \tag{1.58}$$

This means, in the context of the weighted residual method, that the weight function is chosen as the following form.

$$\widehat{W}_i = \frac{\partial R_s}{\partial \alpha_i} = \frac{\partial F(u^h)}{\partial \alpha_i} \tag{1.59}$$

We can similarly obtain

$$\widehat{V}_i = \frac{\partial R_b}{\partial \alpha_i} = \frac{\partial G(u^h)}{\partial \alpha_i} \tag{1.60}$$

Hence, Equation (1.44) becomes

$$\int_{\Omega} \widehat{W}_i R_s \, \mathrm{d}\Omega + \int_{\Gamma} \widehat{V}_i R_b \, \mathrm{d}\Gamma = \int_{\Omega} \frac{\partial R_s}{\partial \alpha_i} R_s \, \mathrm{d}\Omega + \int_{\Gamma} \frac{\partial R_b}{\partial \alpha_i} R_b \, \mathrm{d}\Gamma = 0 \tag{1.61}$$

where i=1,2, …, n, which gives n equations for n coefficients α_i. Solving these n equations for α_i leads to an approximate solution.

1.4.4 Moment method

The weight functions can be chosen to be monomials of, 1, x, x^2,…, x^n. In this way, successive higher "moments" of the residuals are forced to be zero. This technique, called the *moment method*, was invented by Yamada (1947) and Fujita (1951).

The Moment method can be simply formulated as follows. Let

$$\widehat{W}_i = \widehat{V}_i = x_i^{i-1}, \quad i=1,2, \ldots, n \tag{1.62}$$

Equation (1.44) becomes

$$\int_\Omega \widehat{W}_i R_s \mathrm{d}\Omega + \int_\Gamma \widehat{V}_i R_b \mathrm{d}\Gamma = \int_\Omega x_i^{i-1} R_s \mathrm{d}\Omega + \int_\Gamma x_i^{i-1} R_b \mathrm{d}\Gamma = 0 \tag{1.63}$$

which gives n equations for n coefficients α_i. Solving these n equations for α_i yields an approximate solution. Note that the results set of equations is often ill-conditioned. An alterative is to use Chebyshev polynomials in stead of monomials.

1.4.5 Galerkin method

The *Galerkin method* (Galerkin, 1915) can be viewed as a particular weighted residual method, in which the trial functions used for the approximation of the field function are also used as the weight functions.

$$\begin{cases} \widehat{W}_i = B_i \\ \widehat{V}_i = -B_i \end{cases} \tag{1.64}$$

Equation (1.44) now becomes

$$\begin{aligned} &\int_\Omega B_i R_s \mathrm{d}\Omega - \int_\Gamma B_i R_b \mathrm{d}\Gamma = \\ &\qquad \int_\Omega B_i \left[F(\mathbf{B}\boldsymbol{\alpha}) + b \right] \mathrm{d}\Omega - \int_\Gamma B_i \left[G(\mathbf{B}\boldsymbol{\alpha}) - g \right] \mathrm{d}\Gamma = 0 \end{aligned} \tag{1.65}$$

which gives n equations for n coefficients α_i. Solving these n equations for α_i yields an approximate solution.

The Galerkin method has some advantages. First the system matrix obtained by the Galerkin method is usually symmetric. In addition, in many cases, the Galerkin method leads to the same formulations obtained by the energy principles, and hence has certain physical foundations. Therefore, the Galerkin method is regarded so far as the most effective version of the weighted residual method, and is widely used in numerical methods, in particular the finite element method (FEM). Note that to obtain the

formulations of the FEM using the weighted residual method, Equation (1.45) is often used, in which only the residual for the governing equation is considered. The boundary conditions (BCs) are treated separately for the essential BCs and the natural BCs. The former is handled after obtaining the discretized system equations, and the latter is implemented after performing integration by parts. This procedure will also be followed in forming the MFree weak-form methods (Chapters 4 and 5).

1.4.6 Examples

In order to illustrate these approximation methods, consider a simple example problem of a truss member. A truss member is a solid whose dimension in one direction is much larger than those in the other two directions, as shown in Figure 1.5. The force is applied only in the x direction, and the axial displacement u is only a function of x. Therefore, the axial displacement u in a truss member is governed by the following equilibrium equations.

$$EA\frac{\mathrm{d}^2u}{\mathrm{d}x^2}+b(x)=0 \tag{1.66}$$

where E is the Young's modulus, A is the cross-section area, and $b(x)$ is a distributed external axial force applied along the truss member.

We assume that the solution is constrained by the essential (displacement) boundary conditions.

$$u\big|_{\substack{x=0\\x=L}}=0 \tag{1.67}$$

where L is the length of the truss member.

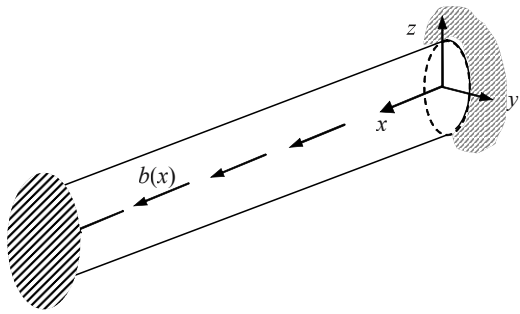


Figure 1.5. A uniform truss member subjected to an axial loading distributed in the x direction.

For simplicity, $E=1.0$, $A=1.0$, $b(x)=12x^2$, and L=1.0 are used in this example. The following exact solution of the problem for the axial displacement can be easily obtained by solving the differential Equation (1.66) together with the boundary conditions Equation (1.67).

$$u^{\text{exact}}(x) = -x^4 + x \tag{1.68}$$

In seeking an approximate solution for the axial displacement, we assume that the solution has the following form.

$$u^h(x) = x(x-L)(\sum_{i=1}^{n} \alpha_i x^{i-1}) \tag{1.69}$$

where α_i is the unknown coefficient to be determined, and $B_i = x(x-L)x^{i-1}$ is *i*th trial function. Note that the basis function is deliberately chosen to satisfy the displacement boundary conditions Equation (1.67).

As the assumed displacement satisfies the boundary conditions, there is no residual on the boundary (i.e., R_b=0). The approximate solution has continuity of all orders throughout the problem domain. However, Equation (1.69) may not exactly satisfy the equilibrium Equation (1.66), and the following residual exists in the problem domain:

$$R(x) = \frac{\mathrm{d}^2 u^h(x)}{\mathrm{d}x^2} + b(x) \tag{1.70}$$

In the approximate solution Equation (1.69), n can be taken as 1, 2, …,. Because the B_i are linearly independent and *complete*[‡], Equation (1.69) will converge to the exact solution when $n \to \infty$. For simplicity, we choose only one and two terms (n=1 and 2) so that the solution is an approximation.

- When n=1, the approximate solution can be written as

$$u^h{}_1(x) = \alpha_1 x(x-L) = \alpha_1 x(x-1.0) \tag{1.71}$$

and the corresponding residual is

$$R_1(x) = 2\alpha_1 + 12x^2 \tag{1.72}$$

- When n=2, the approximate solution can be written as

$$\begin{aligned} u^h{}_2(x) &= \alpha_1 x(x-L) + \alpha_2 x^2(x-L) \\ &= \alpha_1 x(x-1) + \alpha_2 x^2(x-1) \end{aligned} \tag{1.73}$$

and the corresponding residual is

[‡] Meaning that there is no skip of orders: $B_i = x(x-L)x^{i-1}$ for all i=1, 2, …n.

$$R_2(x) = 2\alpha_1 + \alpha_2(6x-2) + 12x^2 \tag{1.74}$$

1.4.6.1 Use of the collocation method

When one term is used in the approximate solution (n=1), the middle point on the truss (or $x = \frac{L}{2} = 0.5$) is chosen as the point for the collocation method. Using the collocation form given in Equation (1.52) and the residual formulation given in Equation (1.72), we can obtain α_1 and then the following approximate solution using one term.

$$u^h(x) = -1.5x(x-1.0) \tag{1.75}$$

For two terms in approximate solutions (n=2), we choose two points on the truss (or $x = \frac{L}{3}$ and $x = \frac{2L}{3}$) as the collocation points. With Equations (1.52) and (1.74), we can obtain α_1, α_2, and the following approximate solution.

$$u^h(x) = -\frac{2}{3}x(x-1) - 2x^2(x-1) \tag{1.76}$$

1.4.6.2 Use of the subdomain method

If the whole domain is used as the integration domain of the subdomain method, using Equation (1.55) and the unit weight functions, the formulation of the subdomain method using one term in the approximate solution (n=1) can be written as

$$\int_0^1 \widehat{W}_1(x) R_1(x)\mathrm{d}x = \int_0^1 (2\alpha_1 + 12x^2)\mathrm{d}x = 4+2\alpha_1 = 0 \tag{1.77}$$

which gives $\alpha_1 = -2.0$. Hence, the approximate solution using one term is obtained as

$$u^h(x) = -2.0x(x-1.0) \tag{1.78}$$

For two terms in the approximate solution (n=2), we use two subdomains and two unit weight functions, i.e.,

$$\begin{cases} \Omega_1: \ \widehat{W}_1 = 1.0, & 0 \le x \le 0.5 \\ \Omega_2: \ \widehat{W}_2 = 1.0, & 0.5 \le x \le 1.0 \end{cases} \tag{1.79}$$

Equation (1.55) and Equation (1.74) give two coefficients of α_1 and α_2. The approximate solution using two terms is

$$u^h(x) = -1.0x(x-1) - 2.0x^2(x-1) \tag{1.80}$$

1.4.6.3 Use of the least squares method

In the least squares method, the weight function is chosen as

$$\widehat{W}_i = \frac{\partial R(x)}{\partial \alpha_i} \tag{1.81}$$

For one term in the approximate solution (n=1), we use the following weight function

$$\widehat{W}_1 = \frac{\partial R_1(x)}{\partial \alpha_1} = 2.0 \tag{1.82}$$

With Equation (1.61) and Equation (1.82), we can obtain $\alpha_1 = -2.0$. The approximate solution becomes

$$u^h(x) = -2.0x(x-1.0) \tag{1.83}$$

With two terms in the approximate solution (n=2), we use the following two weight functions, i.e.

$$\begin{cases} \widehat{W}_1 = \dfrac{\partial R_2(x)}{\partial \alpha_1} = 2.0 \\ \widehat{W}_2 = \dfrac{\partial R_2(x)}{\partial \alpha_2} = 6x - 2 \end{cases} \tag{1.84}$$

From Equation (1.61) and Equation (1.84), we can obtain $\alpha_1 = -1.0$ and $\alpha_2 = -2.0$. The approximate solution using two terms is found as

$$u^h(x) = -1.0x(x-1) - 2.0x^2(x-1) \tag{1.85}$$

It should be noted that the coefficient matrix for solving the unknown coefficient α_i is symmetric in the least squares method.

1.4.6.4 Use of the moment method

In the moment method, the weight function is chosen as

$$\widehat{W}_i = x^{i-1} \tag{1.86}$$

For one term in the approximate solution (n=1), we use the following weight function:

$$\widehat{W}_1 = x^0 = 1 \tag{1.87}$$

Using Equation (1.63) and Equation (1.87), we obtain $\alpha_1 = -2.0$, and, hence, the approximate solution

$$u^h(x) = -2.0x(x-1.0) \tag{1.88}$$

With two terms in the approximate solution (n=2), we use two weight functions, i.e.

$$\begin{cases} \widehat{W}_1 = x^{(1-1)} = 1.0 \\ \widehat{W}_2 = x^{(2-1)} = x \end{cases} \tag{1.89}$$

From Equation (1.63) and Equation (1.89), we can obtain $\alpha_1 = -1.0$ and $\alpha_2 = -2.0$. Finally, the approximate solution using two terms is

$$u^h(x) = -1.0x(x-1) - 2.0x^2(x-1) \tag{1.90}$$

1.4.6.5 Use of the Galerkin method

For one term in the approximate solution (n=1), we use the following weight function:

$$\widehat{W}_1 = B_1 = x(x-1) \tag{1.91}$$

Using Equation (1.65) and Equation (1.91), we can obtain $\alpha_1 = -1.8$ and, therefore, the approximate solution using one term is

$$u^h(x) = -1.8x(x-1.0) \tag{1.92}$$

With taking two terms in the approximate solution (n=2), we use the following two weight functions:

$$\begin{cases} \widehat{W}_1 = B_1 = x(x-1) \\ \widehat{W}_2 = B_2 = x^2(x-1) \end{cases} \tag{1.93}$$

Using Equation (1.65) and Equation (1.93), we can obtain the following set of equations in the matrix form of

$$\begin{bmatrix} -\frac{1}{3} & -\frac{1}{6} \\ -\frac{1}{6} & -\frac{2}{15} \end{bmatrix} \begin{Bmatrix} \alpha_1 \\ \alpha_2 \end{Bmatrix} = \begin{Bmatrix} \frac{3}{5} \\ \frac{2}{5} \end{Bmatrix} \tag{1.94}$$

It can be seen that the coefficient matrix obtained using the Galerkin method is symmetric.

Solving these equations to yield $\alpha_1 = -0.8$ and $\alpha_2 = -2.0$, we find the approximate solution as

$$u^h(x) = -0.8x(x-1) - 2.0x^2(x-1) \tag{1.95}$$

The approximate solutions obtained by the five approximation methods are listed in Table 1.1, and plotted in Figure 1.6~Figure 1.9 for easy comparison. Figure 1.6 and Figure 1.7 plot the weight functions and the results of displacements obtained using the analytical solution and the one-term approximate solution, respectively. Figure 1.8 and Figure 1.9 plot the weight functions and the curves obtained using the analytical solution and the two-term approximate solutions, respectively. These table and figures show that the accuracy of the approximated results is different for different approximation methods and for different terms used in the approximate solutions. Usually, more terms used in the approximate solution lead to higher accuracy. This can be easily observed from Figure 1.9. Note that the Galerkin method leads to the best results for this example problem. It provides the solution with best balanced over- and under-estimation of the exact solution over the entire problem domain, as clearly shown in Figure 1.8 and Figure 1.9. The solutions of other methods are one-side biased.

1.4.6.6 Use of more terms in the approximate solution

To study the convergence of the weighted residual methods, we discuss results of the three-term approximate solution in this section. We omit details and present only the results. Readers are also encouraged to obtain the solution using more than 3 terms.

When three terms (n=3) are used, the approximate solution, Equation (1.69), can be written as

$$u_3^h(x) = \alpha_1 x(x-1) + \alpha_2 x^2(x-1) + \alpha_3 x^3(x-1) \tag{1.96}$$

and the corresponding residual is

$$R_3(x) = 2\alpha_1 + \alpha_2(6x-2) + \alpha_3(12x^2 - 6x) + 12x^2 \tag{1.97}$$

The five versions of weighted residual methods all give the same coefficients, α_1, α_3 and α_3:

$$\begin{cases} \alpha_1 = -1 \\ \alpha_2 = -1 \\ \alpha_3 = -1 \end{cases} \tag{1.98}$$

So that, the approximate solution using three terms is

$$\begin{aligned} u^h(x) &= -1.0x(x-1) - 1.0x^2(x-1) - 1.0x^3(x-1) \\ &= -x^4 + x \end{aligned} \tag{1.99}$$

The approximate solution is the same as the exact solution that is given in Equation (1.68). This means that all these five weighted residual methods give the exact solution when three terms are used in the approximate solution given in Equation (1.69). The same conclusion can be drawn when more than 3 terms are used. For quantitative analysis, the following norm is defined as the error indicator.

$$e = \frac{1}{N}\sum_{j}^{N} \frac{\left| u(x_j)^{\text{num}} - u(x_j)^{\text{exact}} \right|}{\left| u(x_j)^{\text{exact}} \right|} \tag{1.100}$$

where $u(x_j)^{\text{num}}$ and $u(x_j)^{\text{exact}}$ are, respectively, displacements at point x_j (j=1,2, …, N) obtained using the numerical methods and the analytical method, N is the number of uniform points used to study the error, and N=21 is used here.

Figure 1.10 plots the convergence curves of different weighted residual methods using different terms in the approximate solution. When 3 or more terms are used, all these five weighted residual methods converge to the exact solution.

This example shows that if the exact solution is included in the basis (or trial) functions, the different versions of weighted residual methods will reproduce the exact solution. This *reproducibility property* makes the method fundamentally credible, and is essential to any numerical method.

1.5 GLOBAL WEAK-FORM FOR SOLIDS

The Galerkin weak-form can be derived directly from the energy principles for problems of solid mechanics. One of these is the *minimum total potential energy principle*. This principle states that for a structural system that is at an equilibrium state, the total potential energy in the system is stationary for a given set of admissible displacements. This principle can be used in a straightforward manner in the following three simple steps:

Table 1.1. Comparisons between the analytical and approximate results*

Solutions		x=0.25		x=0.5		x=0.75	
		u	Error(%)	u	Error(%)	u	Error(%)
	Exact: $u^{\text{exact}}(x) = -x^4 + x$	0.2461	/	0.4375	/	0.4336	/
One term	a) $u^h(x) = -1.5x(x-1.0)$	0.2812	14.2	0.375	-14.2	0.2812	-35.13
	b) $u^h(x) = -2.0x(x-1.0)$	0.375	52.38	0.5	14.28	0.375	-13.51
	c) $u^h(x) = -2.0x(x-1.0)$	0.375	52.38	0.5	14.28	0.375	-13.51
	d) $u^h(x) = -2.0x(x-1.0)$	0.375	52.38	0.5	14.28	0.375	-13.51
	e) $u^h(x) = -1.8x(x-1.0)$	0.3375	37.14	0.45	2.85	0.3375	-22.16
Two terms	a) $u^h(x) = -\frac{2}{3}x(x-1) - 2x^2(x-1)$	0.2188	-11.11	0.4167	-4.76	0.4063	-6.31
	b) $u^h(x) = -1.0x(x-1) - 2.0x^2(x-1)$	0.2813	14.28	0.5	14.28	0.4687	8.11
	c) $u^h(x) = -1.0x(x-1) - 2.0x^2(x-1)$	0.2813	14.28	0.5	14.28	0.4687	8.11
	d) $u^h(x) = -1.0x(x-1) - 2.0x^2(x-1)$	0.2813	14.28	0.5	14.28	0.4687	8.11
	e) $u^h(x) = -0.8x(x-1) - 2.0x^2(x-1)$	0.2438	-0.95	0.45	2.86	0.4312	-0.54

* error $= \dfrac{(u^{\text{num}} - u^{\text{exact}})}{u^{\text{exact}}}$, where u^{num} and u^{exact} are the results obtained by numerical methods and the exact result, respectively. (a) the collocation method; (b) the subdomain method; (c) the least squares method; (d) the moment method; (e) the Galerkin method

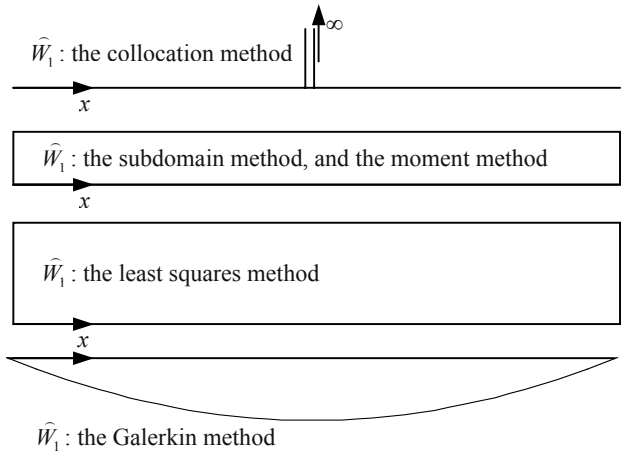


Figure 1.6. Weight functions used in different weight residual methods when the approximate solution is $u^h{}_1(x) = \alpha_1 x(x-L)$ with α_1 being a coefficient.

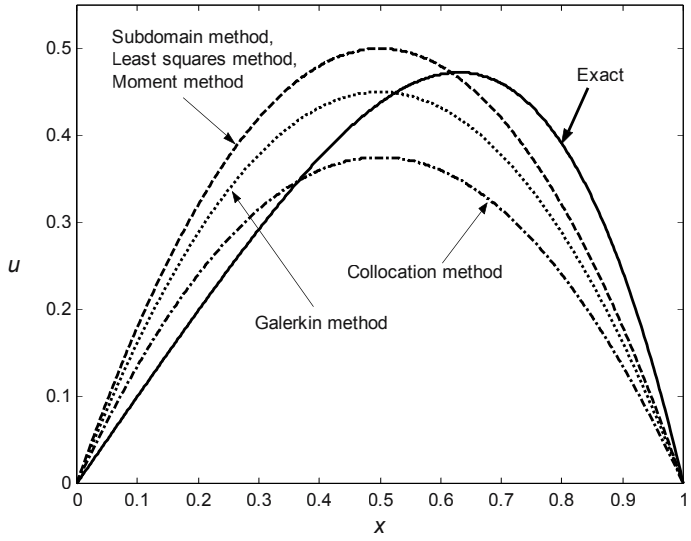


Figure 1.7. Displacement results for the truss member obtained using the analytical method and five different weighted residual methods; the approximate solution is $u^h{}_1(x) = \alpha_1 x(x-L)$.

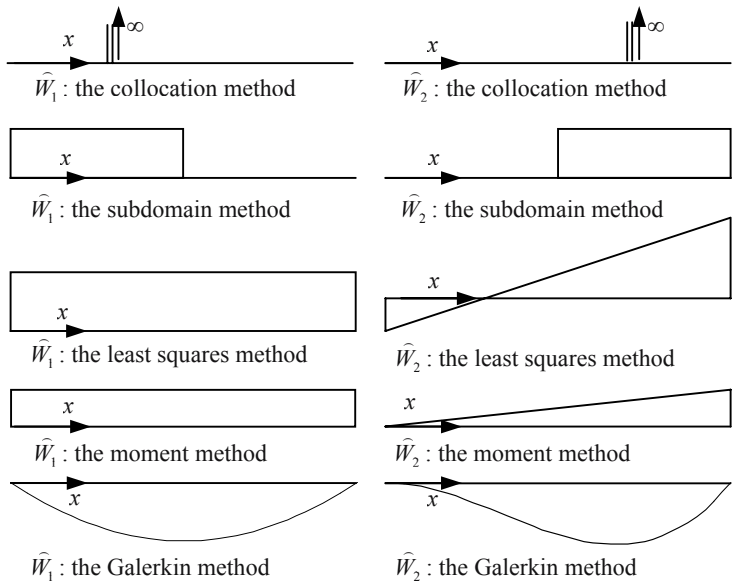


Figure 1.8. Weight functions used in different weight residual methods when the approximate solution is $u^h{}_2(x) = \alpha_1 x(x-L) + \alpha_2 x^2(x-L)$ with α_1 and α_2 being coefficients.

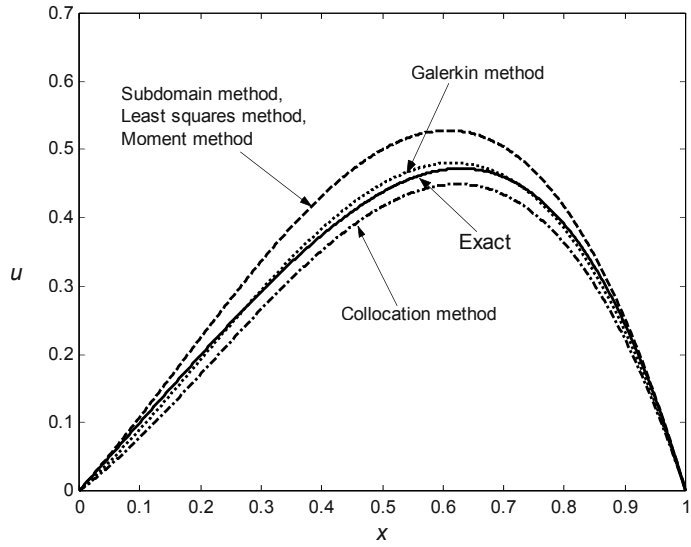


Figure 1.9. Displacement results for the truss member obtained using the analytical method and five different versions of weighted residual methods; the approximate solution is $u^h{}_2(x) = \alpha_1 x(x-L) + \alpha_2 x^2(x-L)$.

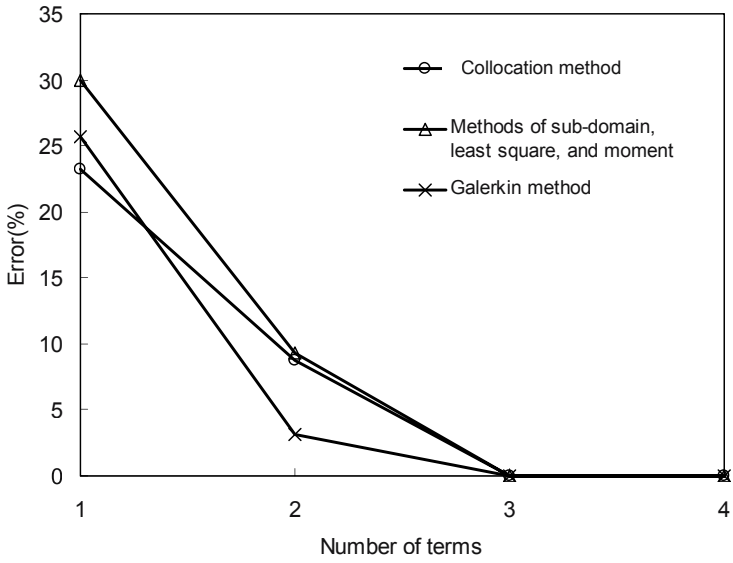


Figure 1.10. Convergence of the results of the axial displacements obtained using different weighted residual methods with different terms in the approximate solution.

1) Approximate the field function (displacement) in terms of the nodal variables using the trial or shape functions; let **d** be the vector consisting of all the nodal displacements in the problem domain.

2) Express the total potential energy, Π, in terms of the nodal variables **d**. For solids and structures of elastic materials, the total potential energy can be expressed as

$$\Pi=\Pi_s-W_f \tag{1.101}$$

where Π_s is the strain energy, and the W_f is the work done by the external forces.

3) Use the stationary conditions to create a set of discretized system equations.

$$\frac{\partial \Pi}{\partial \mathbf{d}}=\begin{Bmatrix} \dfrac{\partial \Pi}{\partial d_1} \\ \dfrac{\partial \Pi}{\partial d_2} \\ \vdots \\ \end{Bmatrix}=0 \tag{1.102}$$

The number of equations created is equal to the number of the total numbers of the nodal variables. The solution of this problem can be obtained by solving Equation (1.102).

For solids and structures of elastic materials, the strain energy of the system can be expressed as

$$\Pi_s = \frac{1}{2}\int_{\Omega} \boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma}\mathrm{d}\Omega \tag{1.103}$$

The work done by the external forces is

$$W_f = \int_{\Omega} \mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega + \int_{\Gamma_t} \mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma \tag{1.104}$$

where Ω is the problem domain, Γ_t stands for the boundary of the solids on which traction forces are prescribed.

Hence, the total potential energy can be expressed as

$$\Pi = \frac{1}{2}\int_{\Omega} \boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma}\mathrm{d}\Omega - \int_{\Omega} \mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega - \int_{\Gamma_t} \mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma \tag{1.105}$$

The variation of the potential energy can be written as

$$\delta\Pi = \delta\left(\frac{1}{2}\int_{\Omega} \boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma}\mathrm{d}\Omega - \int_{\Omega} \mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega - \int_{\Gamma_t} \mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma\right) \tag{1.106}$$

Moving the variation operation into the integral operations, we obtain

$$\delta\Pi = \frac{1}{2}\int_{\Omega}\delta(\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma})\mathrm{d}\Omega - \int_{\Omega}\delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega - \int_{\Gamma_t}\delta\mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma \tag{1.107}$$

because the changing of the order does not affect the results, as they operates on different arguments (variation is on the coefficients of the functions and the integration is on the coordinates). The integrand in the first integral term can be written as follows using the chain rule of variation.

$$\delta(\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma}) = \delta\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma} + \boldsymbol{\varepsilon}^{\mathrm{T}}\delta\boldsymbol{\sigma} \tag{1.108}$$

We note that

$$\boldsymbol{\varepsilon}^{\mathrm{T}}\delta\boldsymbol{\sigma} = (\boldsymbol{\varepsilon}^{\mathrm{T}}\delta\boldsymbol{\sigma})^{\mathrm{T}} = \delta\boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{\varepsilon} \tag{1.109}$$

Using the constitutive equation of solids and the symmetry of the matrix of material constants **D**, we have

$$\delta\boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{\varepsilon} = \delta(\mathbf{D}\boldsymbol{\varepsilon})^{\mathrm{T}}\boldsymbol{\varepsilon} = \delta\boldsymbol{\varepsilon}^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\boldsymbol{\varepsilon} = \delta\boldsymbol{\varepsilon}^{\mathrm{T}}\mathbf{D}\boldsymbol{\varepsilon} = \delta\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma} \tag{1.110}$$

Therefore, Equation (1.108) becomes

$$\delta(\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma}) = 2\delta\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma} \tag{1.111}$$

and Equation (1.107) now becomes

$$\delta\Pi = \int_{\Omega}\delta\,\boldsymbol{\varepsilon}^{\mathrm{T}}\,\boldsymbol{\sigma}\,\mathrm{d}\Omega - \int_{\Omega}\delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega - \int_{\Gamma_t}\delta\mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma \tag{1.112}$$

The minimum total potential energy principle requires $\delta\Pi = 0$. Hence, the following Galerkin weak-form can be obtained:

$$\int_{\Omega}\delta\,\boldsymbol{\varepsilon}^{\mathrm{T}}\,\boldsymbol{\sigma}\,\mathrm{d}\Omega - \int_{\Omega}\delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega - \int_{\Gamma_t}\delta\mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma = 0 \tag{1.113}$$

Equation (1.113) can also be viewed as the principle of *virtual work*, which states that if a solid body is in its equilibrium states, the total virtual work performed by all the stresses in the body and all the external forces applied on the body vanishes, when the body is subjected to a *virtual displacement*. The virtual work can be viewed as an alternative statement of equilibrium equation. In our situation given in Equation (1.113), we can suppose that the solid is subjected to a virtual displacement of $\delta\mathbf{u}$. The first term in Equation (1.113) is the virtual work done by the internal stress in the problem domain, Ω; the 2nd term is the virtual work done by the external body force; the 3rd term is the virtual work done by the external tractions on the boundaries, Γ_t. Therefore, using the principle of virtual work, we can actually write out Equation (1.113) directly without going through the above procedure.

For static linear elastic problems, using the stress-strain relation, and then the strain-displacement relation, we can express Equation (1.113) as follows in terms of the displacement vector $\mathbf{u}$.

$$\int_{\Omega}\delta\big(\mathbf{L}\mathbf{u}\big)^{\mathrm{T}}\,\mathbf{D}\big(\mathbf{L}\mathbf{u}\big)\,\mathrm{d}\Omega - \int_{\Omega}\delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega - \int_{\Gamma_t}\delta\mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma = 0 \tag{1.114}$$

This is the Galerkin weak-form written in terms of displacements, and it is convenient because the displacement is to be approximated in FEM or MFree methods. Equation (1.114) can also be derived from Equation (1.45) by performing integration by parts.

It should be noted that in the weak-form of Equation (1.114) the traction boundary conditions (see Equations (1.17) and (1.33)) have been imposed naturally in the same system equation. Hence, the traction (derivative) boundary conditions in solids, Equations (1.17) and (1.33), are often called *natural boundary conditions* in numerical methods based on the weak-forms. However, in the weak-form of Equation (1.114), the displacement boundary

conditions, Equations (1.18) and (1.34), are not considered. To obtain solutions for weak-forms, it is essential to satisfy the displacement boundary conditions through other proper means. Therefore, the displacement boundary conditions are often called *essential boundary conditions* in numerical methods based on the weak-forms. One simple technique to satisfy the essential boundary conditions is to have the approximate solution satisfy these boundary conditions, as presented in Sub-section 1.4.6. Techniques used to satisfy the essential boundary conditions will be discussed in the following chapters for MFree methods.

The above equation of Galerkin weak-form is very handy in application to problems of solid mechanics, because one does not need to perform integration by parts any more. The discretized system equation can be derived very easily using approximated displacements that satisfy the *admissible* conditions. This Galerkin procedure will be applied repeatedly in the following chapters for many MFree methods.

Note that in using the above-mentioned Galerkin procedure one does not have to know the strong-form of the governing equation.

1.6 LOCAL WEAK-FORM FOR SOLIDS

In deriving local weak-forms, the Petrov-Galerkin procedure has to be used. The Petrov-Galerkin procedure is often used in the FEM formulation for convection dominated systems to obtain a stabilized solution (Zienkiewicz and Taylor, 2000).

The local Petrov-Galerkin weak-forms have been used to formulate the meshless Petrov-Galerkin (MLPG) method (Atluri et al., 1999b). The local weak-form can be obtained from the subdomain weighted residual method discussed in Section 1.4. In this section, the local weak-forms for solids are presented.

In a problem domain Ω, the governing Equation (1.31) of two-dimensional solids at a point $\mathbf{x}_I$ is approximately satisfied by a subdomain weighted residual method. A local weak-form of the partial differential Equation (1.31), over a subdomain (a local quadrature domain) Ω_q bounded by Γ_q can be obtained using the weighted residual method locally

$$\int_{\Omega_q} \widehat{W}_I (\sigma_{ij,j} + b_i) \mathrm{d}\Omega = 0 \tag{1.115}$$

where $\widehat{W}_I$ is the weight function or the test function centered at the point $\mathbf{x}_I$.

The first term on the left hand side of Equation (1.115) can be integrated by parts to obtain

$$\int_{\Omega_q} \widehat{W}_I \sigma_{ij,j} \mathrm{d}\Omega = \int_{\Gamma_q} \widehat{W}_I n_j \sigma_{ij} \mathrm{d}\Gamma - \int_{\Omega_q} \widehat{W}_{I,j} \sigma_{ij} \mathrm{d}\Omega \tag{1.116}$$

where n_j is the jth component of the unit outward normal vector (see Figure 1.4) on the boundary. Substituting Equation (1.116) into Equation (1.115), we can obtain the following local weak-form:

$$\int_{\Gamma_q} \widehat{W}_I \sigma_{ij} n_j \mathrm{d}\Gamma - \int_{\Omega_q} \left[\widehat{W}_{I,j} \sigma_{ij} - \widehat{W}_I b_i \right] \mathrm{d}\Omega = 0 \tag{1.117}$$

Equation (1.117) is the local Petrov-Galerkin weak-form for two-dimensional solids.

Equation (1.117) suggests that instead of solving the strong-form of the system equation given in Equation (1.31), we employ a relaxed weak-form with integration over a small local quadrature domain. This integration operation can "smear" out the numerical error, and therefore make the discrete equation system much more accurate compared to the numerical procedures that operate directly on the strong-forms of system equations. In other words, using Equation (1.117) for any node at $\mathbf{x}_I$, we transform a global boundary value problem into a localized boundary value problem over a local quadrature domain. In the present formulation, the equilibrium equation and boundary conditions are satisfied in all local quadrature domains Ω_q and on their boundary Γ_q. Although the quadrature domains affect the solution, the equilibrium equation and the boundary conditions will be approximately satisfied in the global domain Ω and on its boundary Γ as long as the union of all the local quadrature domains covers the global domain, Ω, and the global boundary, Γ, well.

Because the local weak-form is obtained by the weighted residual method, the test (weight) function plays an important role. Theoretically, any test function is acceptable as long as the condition of continuity is satisfied, and all the weight functions defined for all the nodes in the problem domain are linearly independent. For example, in the MLPG method (Atluri et al., 1999b), the locally supported bell-shaped weight functions can be used so that the integrations are performed locally and no global integration is required. Detailed discussions of the weight functions will be presented in Chapter 3 and Chapter 5.

The main disadvantage of the local Petrov-Galerkin method is that the system matrix is usually not symmetric. The detailed properties of the local weak-form will be discussed in Chapter 5 and Chapter 7.

1.7 DISCUSSIONS AND REMARKS

The basic equations of the solid mechanics were presented. Different versions of the weighted residual methods were introduced and demonstrated using a simple example.

The weighted residual methods will possess the *convergence property*, meaning that the approximate solution of the weighted residual methods will approach the exact solution when the number of the basis functions used increases, as long as

1) The weight functions $\widehat{W}_i$, $\widehat{V}_i$ and the basis functions $B_i(x)$ are linearly independent.

2) The basis functions $B_i(x)$ have a certain order of continuity.

3) The weight functions and the basis functions have a certain degree of overlapping.

The simple example solved using these five different methods (collocation, subdomain, moment, least squares and Galerkin) confirmed the convergence property. This example showed that the weighted residual methods possess the *reproducibility property*, meaning that they are capable of producing the exact solution as long as the independent basis functions contain the components of the exact solution. The convergence and reproducibility properties make the weighted residual methods as reliable ways of obtaining approximate solutions. However, the stability and accuracy of the solution depend on the quality of basis functions; the choices of weight functions; and "matchablility" of the weight and trial (basis) functions. The Galerkin method that uses the same functions for the weight and trial functions often performs the best.

For problems with complicated domains, choosing an independent set of trial (basis) functions for the entire problem domain is often very difficult. Therefore, we usually use *local* shape functions in a *piecewise* manner as the trial functions. The details of creating local MFree shape functions will be given in Chapter 3. The choice of different weight functions leads to different formulation procedures for meshfree methods, such as the collocation scheme, Galerkin weak-form formulation, Petrov-Galerkin weak form formulation, etc. Various MFree methods will be formulated in Chapters 4~7 for problems in mechanics of solids and fluids.

Chapter 2

OVERVIEW OF MESHFREE METHODS

2.1 WHY MESHFREE METHODS

One of the most important advances in the field of numerical methods was the development of the finite element method (FEM) in the 1950s. In the FEM, a continuum with a complicated shape is divided into elements, *finite elements*. The individual elements are connected together by a topological map called a *mesh*. The FEM is a robust and thoroughly developed method, and hence it is widely used in engineering fields due to its versatility for complex geometry and flexibility for many types of linear and non-linear problems. Most practical engineering problems related to solids and structures are currently solved using well developed FEM packages that are commercially available.

However, the FEM has the inherent shortcomings of numerical methods that rely on meshes or elements that are connected together by nodes in a properly *predefined* manner. The following limitations of FEM are becoming increasingly evident:

1) High cost in creating an FEM mesh

The creation of a mesh for a problem domain is a prerequisite in using any FEM code and package. Usually the analyst has to spend most of the time in such a mesh creation, and it becomes the major component of the cost of a computer aided design (CAD) project. Since operator costs now outweigh the cost of CPU (central processing unit) time of the computer, it is

desirable that the meshing process can be fully performed by the computer without human intervention. This is not always possible without compromising the quality of the mesh for the FEM analysis, especially for problems of complex three-dimensional domains.

2) Low accuracy of stress

Many FEM packages do not accurately predict stress. The stresses obtained in FEM are often discontinuous at the interfaces of the elements due to the *piecewise* (or element-wise) *continuous* nature of the displacement field assumed in the FEM formulation. Special techniques (such as the use of the so-called *super-convergence* points or patches) are required in the post-processing stage to recover accurate stresses.

3) Difficulty in adaptive analysis

One of the current new demands on FEM analysis is to ensure the accuracy of the solution; we require a solution with a *desired accuracy*. To achieve this purpose, a so-called *adaptive analysis* must be performed.

In an adaptive analysis using FEM, *re-meshing* (*re-zoning*) is required to ensure proper *connectivity*. For this re-meshing purpose, complex, robust and adaptive mesh generation processors have to be developed. These processors are limited to two-dimensional problems. Technical difficulties have precluded the automatic creation of hexahedron meshes for arbitrary three-dimensional domains. In addition, for three-dimensional problems, the computational cost of re-meshing at each step is very expensive, even if an adaptive scheme were available. Moreover, an adaptive analysis requires "mappings" of field variables between meshes in successive stages of the analysis. This mapping process can often lead to additional computation as well as a degradation of accuracy in the solution.

4) Limitation in the analyses of some problems

- Under large deformations, considerable loss in accuracy in FEM results can arise from the element distortions.
- It is difficult to simulate crack growth with arbitrary and complex paths which do not coincide with the original element interfaces.
- It is very difficult to simulate the breakage of material with large number of fragments; the FEM is based on continuum mechanics, in which the elements cannot be broken; an element must either stay as a whole, or disappear completely. This usually leads to a misrepresentation of the breakage path. Serious error can occur because the problem is non-linear and the results path-dependent.

The root of these problems is the use of *elements* or *mesh* in the formulation stage. The idea of getting rid of the elements and meshes in the

process of numerical treatments has naturally evolved, and the concepts of *meshfree* or *meshless* methods have been shaped up. For convenience, these methods are shortened as *MFree* methods in this book.

2.2 DEFINITION OF MESHFREE METHODS

The definition of an MFree method (GR Liu, 2002) is:

An MFree method is a method used to establish system algebraic equations for the whole problem domain without the use of a predefined mesh for the domain discretization.

MFree methods use a set of *nodes* scattered within the problem domain as well as sets of nodes scattered on the boundaries of the domain to *represent* (not discretize) the problem domain and its boundaries. These sets of scattered nodes are called *field nodes*, and they do not form a mesh, meaning it does not require any *a priori* information on the relationship between the nodes for the *interpolation* or *approximation*[†] of the unknown functions of field variables.

What is the requirement for an MFree method?

The *minimum* requirement for an MFree method is

- A predefined mesh is not required in the field variable interpolation or approximation.

The *ideal* requirement for an MFree method is

- No mesh is required at throughout the process of formulating and solving the problem of a given arbitrary geometry governed by partial differential system equations subject to boundary conditions.

Many MFree methods have found good applications, and shown very good potential to become powerful numerical tools. However, the MFree methods are still in their developing stage, and there are technical problems that need to be resolved before the methods can become efficient and useful tools for complex engineering problems.

[†] We distinguish *interpolation* and *approximation*. Interpolation refers to an approximation procedure that reproduces the exact values of the approximated function at the nodes. All the other approximation procedures that do not return nodal function values are called approximation. Both interpolation and approximation are used in MFree methods; the standard FEM uses interpolation based on elements.

2.3 SOLUTION PROCEDURE OF MFREE METHODS

In this section, the solution procedure of MFree methods will be outlined. It will be introduced based on the comparisons with the familiar finite element method (FEM).

Figure 2.1 shows two procedures of FEM and the MFree method. This tells us:

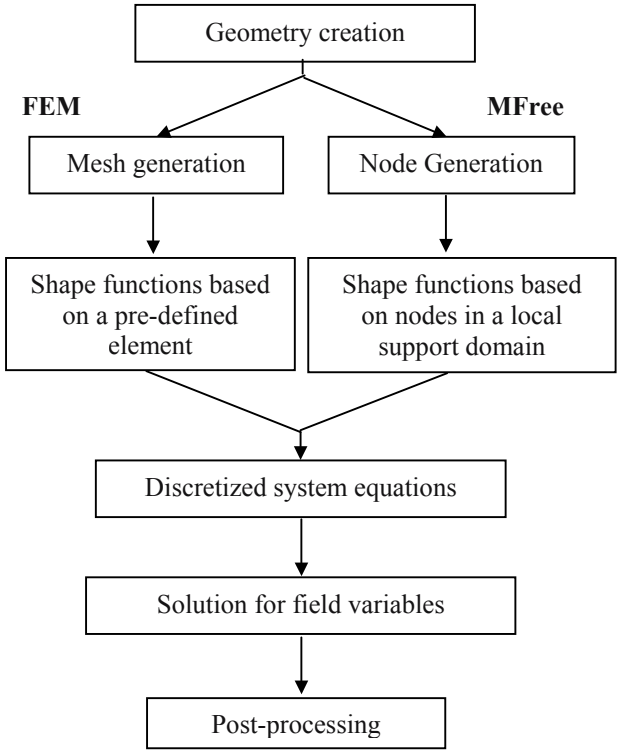


Figure 2.1. Flowcharts for FEM and MFree method.

1) The methods depart at the stage of mesh creation.
2) The constructions of the shape functions in these two methods are different. In the finite element method, the shape functions are constructed using predefined elements, and the shape functions are

the same for the entire element. In MFree methods, however, the shape functions constructed are usually only for a particular point of interest based on selected local nodes; the shape functions can change when the point of interest changes.

3) The methods follow the similar procedure once the global discretized system equation is established. Therefore, many techniques developed for the FEM can be used in MFree methods.

Comparisons between the finite element method and the MFree method are listed in Table 2.1.

Table 2.1. Differences between FEM and MFree method

Items	FEM	MFree method
Mesh	Yes	No
Shape function creation	Based on pre-defined elements	Based on local support domains
Discretized system stiffness matrix	Banded, symmetric	Banded, may or may not be symmetric depending on the method used.
Imposition of essential boundary condition	Easy and standard	Special treatments may be required, depending on the method used
Computation speed	Fast	Slower compared to the FEM depending on the method used.
Accuracy	Accurate compared to FDM	More accurate than FEM
adaptive analysis	Difficult for 3D cases	Easier
Stage of development	Well developed	Infant, with many challenging problems
Commercial software packages availability	Many	Few

We now list the steps in an MFree method with discussions on major differences with the finite element method.

Step 1: Domain representation

In the MFree method, the problem domain and its boundary are first modelled and represented by using sets of nodes scattered in the problem domain and on its boundary. Since these nodes carry the values of the field variables in an MFree formulation, they are often called *field nodes*. The density of the nodes depends on the accuracy required and resources

available. The nodal distribution is usually not uniform. Since adaptive algorithms can be used in MFree methods, the density is eventually controlled automatically and adaptively in the code; the initial nodal distribution becomes not important. An MFree method should be able to work for an arbitrary nodal distribution.

In the finite element method, this step is different: meshing needs to be performed to discretize the geometry and create the *elements*. The domain has to be meshed properly into elements of specific shapes such as triangles and quadrilaterals. No overlapping or gaps are allowed. Information, such as the *element connectivity*, has also to be created during the meshing for later creation of system equations. Mesh generation is a very important part of the *pre-process* of the finite element method. It is ideal to have an entirely automated mesh generator; unfortunately, it is not practically available for general situations.

Figure 2.2 shows the differences of the domain representation in the MFree method and the FEM.

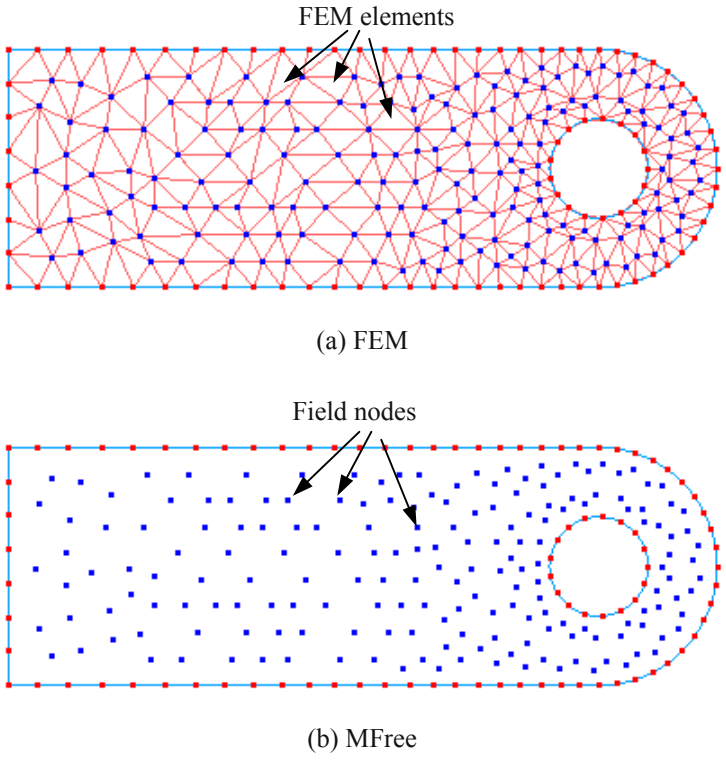


Figure 2.2. Domain representation in FEM and MFree method.

Step 2: Function interpolation/approximation

Since there is no mesh of elements in an MFree method, the field variable (e.g., a component of the displacement) u at any point at $\mathbf{x}=(x, y, z)$ within the problem domain is interpolated using function values at field nodes within a small local *support domain* of the point at $\mathbf{x}$, i.e.,

$$u(\mathbf{x}) = \sum_{i=1}^{n} \phi_i(\mathbf{x}) u_i \quad = \mathbf{\Phi}^{\mathrm{T}}(\mathbf{x})\mathbf{U}_s \tag{2.1}$$

where n is the number of the nodes that are included in the local support domain of the point at $\mathbf{x}$, u_i is the nodal field variable at the ith node, $\mathbf{U}_s$ is the vector that collects all the field variables at these n nodes, and $\phi_i(\mathbf{x})$ is the shape function of the ith node determined using these nodes included in the support domain of $\mathbf{x}$. As the shape functions will not be used regarded as zero outside the local support domain in an MFree method, we often say that the shape functions is *locally support*.

A local support domain of a point $\mathbf{x}$ determines the number of nodes to be used to support or approximate the function value at $\mathbf{x}$. The support domain can have different shapes and its dimension and shape can be different for different points of interest $\mathbf{x}$, as shown in Figure 2.3; they are usually circular or rectangular.

In the finite element method, the shape functions are constructed using pre-defined elements. In fact, if the so-called natural coordinate systems are used, the shape functions in the natural coordinates are the same for all the elements of the same type. These shape functions are usually pre-determined for different types of elements before the finite element analysis starts.

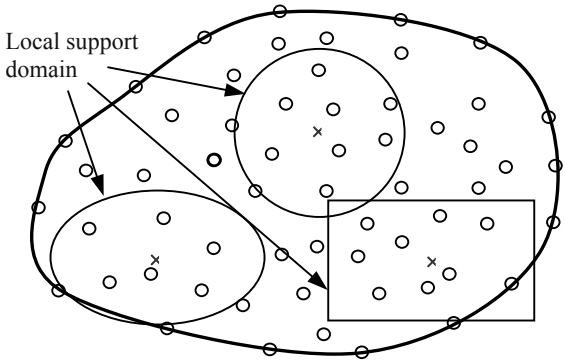


Figure 2.3. Local support domains used in the MFree method to construct shape functions.

Step 3: Formation of system equations

The discrete equations of an MFree method can be formulated using the shape functions and strong or weak form system equation given in Chapter 1. These equations are often written in nodal matrix form and are assembled into the global system matrices for the entire problem domain. The discretized system equations of MFree methods are similar to those of FEM in terms of bandness and sparseness, but they can be asymmetric depending on the method used.

Step 4: Solve the global MFree equations

This is similar to that for FEM, except solvers for asymmetric matrix systems may be needed.

2.4 CATEGORIES OF MESHFREE METHODS†

The development of some of the MFree methods can be traced back more than seventy years to the collocation methods (Slater, 1934; Barta, 1937; Frazer et al., 1937; Lanczos, 1938, etc). Some of the early MFree methods were the vortex method (Chorin, 1973; Bernard, 1995), finite difference method (FDM) with arbitrary grids, or the general FDM (GFDM) (Girault, 1974; Pavlin and Perrone, 1975; Snell et al, 1981; Liszka and Orkisz, 1977; 1980; Krok and Orkisz, 1989). Another well-known MFree method is the Smoothed Particle Hydrodynamics (SPH) that was initially used for modelling astrophysical phenomena such as exploding stars and dust clouds that had no boundaries. Most of the earlier research work on SPH is reflected in the publications of Lucy (1977), and Monaghan and his co-workers (Gingold and Monaghan, 1977; Monaghan and Lattanzio, 1985; Monaghan, 1992). Detailed discussions on some of the recent developments for SPH can be found in the book by GR Liu and Liu (2003). Overall, there has been less research devoted to MFree strong-form methods. This may be partly because the MFree strong-form method was less robust than the method based on the weak-form, and partly because research was concentrated on the finite element method (FEM) which used weak-forms; it was then a natural step to MFree weak-form methods.

† MFree methods and techniques presented in this section will be discussed in detail in the following chapters.

From the early 1990s, there has been an increase in research devoted to MFree weak-form methods, and a group of MFree methods has been greatly proposed. Examples of these methods are the diffuse element method (DEM) (Nayroles et al., 1992), the element free Galerkin (EFG) method (Belytschko et al, 1994a), the reproducing kernel particle method (RKPM) (Liu et al, 1995), the point interpolation methods (GR Liu and Gu, 2001c; Wang and GR Liu, 2000), the meshless local Petrov-Galerkin method (MLPG) (Atluri 1998a), the boundary node method (BNM) (Mukherjee and Mukherjee, 1997), the boundary point interpolation method (BPIM) (Gu and GR Liu, 2001e,2002a,2003b), the meshfree weak-strong (MWS) form method (GR Liu and Gu, 2002d; 2004), etc. These methods do not require a mesh at least for the field variable interpolations. The approximation functions are constructed by using a set of arbitrary nodes, and no element or connectivity of the nodes is needed for the function approximation. Adaptive analyses and simulations using MFree methods become very efficient and much easier to implement, even for problems which pose difficulties for the traditional FEM.

Many MFree methods have been proposed and achieved remarkable progress over the past years: we now classify them in different ways for easy understanding and later referencing.

2.4.1 Classification according to the formulation procedures

According to the formulation procedures, MFree methods fall into three categories:

2.4.1.1 Meshfree methods based on weak-forms

These are called *MFree weak-form methods* in this book. In MFree weak-form methods, the governing partial differential equations (PDEs) with derivative boundary conditions are first transformed to a set of so-called weak-form integral equations using different techniques discussed in Chapter 1. The weak-forms are then used to derive a set of algebraic system equations through a numerical integration process using sets of background cells that may be constructed globally or locally in the problem domain.

MFree weak-form methods were relatively under developed before 1990, but there has been a substantial increase in research effort since then. Several important papers have been published. The first was by Nayroles et al. (1992); they applied the moving least squares (MLS) approximation proposed by Lancaster and Salkauskas (1981) to the Galerkin weak-form to formulate the diffuse element method (DEM). Belytschko et al. (1994a) published another important paper on the element free Galerkin (EFG)

method based on the DEM. He and co-workers have also made significant contribution in further developing, improving and popularizing EFG for many mechanics problems. The MFree weak-form methods have been developed at a very fast pace since 1994; there are now many different versions of MFree weak-form methods.

MFree weak-form methods based on the global weak-forms are called *MFree global weak-form methods*, and those based on local weak-forms are called *MFree local weak-form methods*.

MFree global weak-form methods are based on the global Galerkin weak-form for equations of problems and the MFree shape functions. Two typical MFree global weak-form methods: the element-free Galerkin (EFG) method (Belytschko et al., 1994a) and the radial point interpolation method (RPIM) (GR Liu and Gu, 2001c; Wang and GR Liu, 2000; 2002a), will be discussed in Chapter 4. Another typical MFree global weak-form method is the reproducing kernel particle method (RKPM) proposed by Liu and co-workers in 1995 (Liu et al., 1995). The main idea of RKPM is to improve the SPH approximation to satisfy consistency requirements using a correction function. RKPM has been used in nonlinear and large deformation problems (Chen et al., 1996; Chen et al., 1998; Liu and Jun, 1998), inelastic structures (Chen et al., 1997), structural acoustics (Uras et al., 1997), fluid dynamics (Liu and Jun et al., 1997), and so on.

MFree local weak-form methods were developed by Atluri and coworkers based on the local Petrov-Galerkin weak-form, and the MFree shape functions. The detailed discussions of the meshless local Petrov-Galerkin (MLPG) method (Atluri and Zhu, 1998a, 1998b, 2000a, 2000b; Atluri and Shen, 2002) and the local radial point interpolation method (LRPIM) (GR Liu and Gu, 2001c; GR Liu and Yan et al., 2002) will be presented in Chapter 5.

Some other MFree weak-form methods have also been developed, such as the *hp*-cloud method (Armando and Oden, 1995), the partition of unity finite element method (PUFEM) (Melenk and Babuska, 1996; Babuska and Melenk, 1997), the finite spheres method (De and Bathe, 2000), the free mesh method (Yagawa and Yamada, 1996), and so on.

2.4.1.2 Meshfree methods based on collocation techniques

These MFree methods are called MFree collocation methods or *MFree strong-form methods* in this book. In these methods, the strong-forms of governing equations and equations for boundary conditions are *directly* discretized at the field nodes using simple *collocation* techniques to obtain a set of discretized system equations. MFree strong-form methods have a long

history. The finite difference method with arbitrary grids or the general finite difference method (GFDM) (Girault, 1974; Pavlin and Perrone, 1975; Snell et al, 1981; Liszka and Orkisz, 1977; 1980; Krok and Orkisz, 1989), MFree collocation methods (see, e.g. Kansa, 1990; Wu, 1992; Zhang and Song et al., 2000; Liu X et al., 2002; 2003a-e; etc.), and the finite point method (FPM) (Oñate et al., 1996; 1998; 2001; etc.) are all typical MFree strong-form methods.

MFree strong-form methods have some attractive advantages: a simple algorithm, computational efficiency, and *truly* meshfree. However, MFree strong-form methods are often unstable, not robust, and inaccurate, especially for problems with derivative boundary conditions. Several strategies may be used to impose the derivative boundary conditions in the strong-form methods, such as the use of fictitious nodes, the use of the Hermite-type MFree shape functions, the use of a regular grid on the derivative boundary, etc. Detailed discussions appear in Chapter 6.

2.4.1.3 Meshfree methods based on the combination of weak-form and collocation techniques

These MFree methods are called *MFree weak-strong (MWS) form* methods in this book. The MWS method was developed by GR Liu and Gu (2002d, 2003b). The key idea of the MWS method is that in establishing the discretized system equations, both the strong-form and the local weak-form are used for the same problem, but for different groups of nodes that carries different types of equations/conditions. The local weak-form is used for all the nodes that are on or near boundaries with derivative boundary conditions. The strong-form is used for all the other nodes (called collocatable nodes to be defined in Chapter 7). The MWS method uses least background cells for the integration, and it is currently the *almost* ideal MFree method that can provide stable and accurate solutions for mechanics problems.

There are also MFree methods based on the integral representation method for function approximations, such as the Smooth Particle Hydrodynamics (SPH) methods (Lucy, 1977; Gingold and Monaghan, 1977; GR Liu and Liu, 2003, etc.). In the standard SPH method, the function approximation is performed in a *weak* (integral) form, but strong-form equations are directly discretized at the particles.

2.4.2 Classification according to the function approximation schemes

The method of function interpolation/approximation based on arbitrary nodes is one of the most important issues in an MFree method. Without

robust interpolation/approximation tools being developed, MFree methods would not exist. Hence, MFree methods may be classified according to the MFree interpolation/approximation methods used.

2.4.2.1 Meshfree methods based on the moving least squares approximation

The moving least squares (MLS) approximation was originated by mathematicians working on data fitting and surface construction (Lancaster and Salkauskas, 1981). The detailed discussions of MLS will be presented in Chapter 3. The invention of the MLS approximation was the key to the development of many MFree weak-form methods, because the MLS can provide a continuous approximation for a field function over the entire problem domain. It is now widely used in many types of MFree methods for constructing MFree shape functions. Nayroles et al. (1992) used the MLS approximation for the first time to develop the so-called diffuse element method (DEM). Many MFree methods have been since developed based on the MLS approximation, such as the element-free Galerkin (EFG) method (Belytschko et al., 1994a) and the meshless local Petrov-Galerkin (MLPG) method (Atluri and Zhu, 1998a). EFG and MLPG will be described in Chapters 4 and 5, respectively.

2.4.2.2 Meshfree methods based on the integral representation method for the function approximation

These MFree methods use integral forms of function approximations. The widely used smooth particle hydrodynamic (SPH) method (Lucy, 1977; Gingold and Monaghan, 1977; GR Liu and Liu, 2003) and the reproducing kernel particle method (RKPM) (Liu et al., 1995) can belong to this category.

Smooth Particle Hydrodynamic (SPH) was first invented to solve astrophysical problems in three-dimensional open space, in particular polytropes (Lucky, 1977; Gingold and Monaghan, 1977). The basic idea of SPH is that the state of a system can be represented by arbitrarily distributed particles, and then the SPH approximation is used to discretize the strong-form of the PDEs of the problem. The applications of SPH include astrophysical problems and related fluid dynamics procedure, such as the simulation of binary stars and stellar collisions (Benz, 1988; Monaghan, 1992), incompressible flows (e.g., Liu MB and GR Liu et al., 2001), elastic flow (Swegle et al., 1992), gravity currents (Monaghan, 1995), heat transfer (Cleary, 1998), and so on. Recently, the SPH method has been applied for the simulations of high (or hyper) velocity impact (HVI) problems. Libersky and his co-workers have made substantial contributions in the application of SPH to impact problems (Libersky and Petscheck, 1991; Libersky et al.,

1995; Randles and Libersky, 1996). GR Liu and his co-workers have used SPH to simulate explosion and penetration (GR Liu and Liu et al., 2001a,b; Liu MB and GR Liu et al., 2003a, 2003c-f). A so-called discontinuous SPH has also been formulated for simulating the discontinuity at the front of shock waves (Lam et al., 2003e).

The major shortcomings of the SPH method include tensile instability, lack of consistency in field variable approximation, and difficulty in enforcing boundary conditions. Some improvements and modifications of the SPH method have been achieved (Monaghan and Lattanzio, 1985; Swegle et al., 1995; Morris, 1996; GR Liu and Liu et al., 2002; Liu MB and GR Liu, 2003b).

2.4.2.3 Meshfree methods based on the point interpolation method

The point interpolation method (PIM) is an MFree interpolation technique that was used by GR Liu and his colleagues (GR Liu and Gu, 2001a) to construct shape functions using nodes distributed locally to formulate MFree weak-form methods. Different from the MLS approximation, PIM uses interpolations to construct shape functions that possess *Kronecker delta function property*. Two different types of PIM formulations using the polynomial basis (GR Liu and Gu, 2001c) and the radial function basis (RBF) (Wang and GR Liu, 2000) have been developed. MFree methods using PIM shape functions will be discussed in Chapters 4, 5, 6, and 7.

2.4.2.4 Meshfree methods based on the other meshfree interpolation schemes

These methods include MFree methods based on the *hp*-cloud method (Durarte and Odenm 1995), the partition of unity (PU) (Melenk and Babuska, 1996; Babuska and Melenk, 1997) method, etc. This book will not cover these methods.

Note that all these interpolation/approximation methods can be applied in strong-form methods. More details on this can be found in Chapter 6.

2.4.3 Classification according to the domain representation

Similar to the classification of finite element method (FEM) and boundary element method (BEM), MFree methods may also be largely categorized into the following two categories:

2.4.3.1 Domain-type meshfree methods.

In these methods, both the problem domain and the boundaries are represented by field nodes. The discretized system equations are obtained using the weak-form or strong-form or both for the whole domain.

2.4.3.2 Boundary-type meshfree methods.

MFree ideas have also been extended to the Boundary Integral Equation (BIE) to formulate boundary-type MFree methods. In these MFree methods, only the boundaries of the problem domain are represented by a set of nodes. No node is needed within the problem domain. The boundary integral equation (BIE) is first established using the Green's functions. The discretized system equations are then obtained from boundary nodes using MFree shape functions.

Mukherjee and co-workers proposed the boundary node method (BNM) (Mukherjee and Mukherjee, 1997; Kothnur et al., 1999). In BNM, the boundary of the problem domain is represented by a set of properly scattered nodes. BIEs of problems considered are discretized using the MLS approximation based only on a group of arbitrarily distributed boundary nodes. BNM has been applied to three-dimensional problems of potential theory and elasto-statics (Chati and Mukherjee, 2000; Chati et al., 1999, 2001). Very good results were reported. However, because the MLS shape functions lack the delta function property, it is difficult to satisfy the boundary conditions accurately in BNM. This problem becomes even more serious in BNM because many boundary conditions need to be satisfied. The method used in BNM to impose boundary conditions doubles the number of system equations compared with the conventional BEM. This makes BNM computationally much more expensive than the BEM.

Another boundary-type MFree method is the local boundary integral equation (LBIE) method (Zhu et al., 1998a, 1998b; Sladek et al., 2002). In LBIE, the domain and the boundary of the problem are represented by distributed nodes. For each field node, BIE is used in a regular local domain to construct system equations. The LBIE has been successfully used to solve linear and non-linear boundary value problems (Zhu et al., 1998a, 1998b; Zhu et al., 1999; Atluri et al., 2000).

Gu and GR Liu used the PIM and RPIM shape functions in BIEs of PDEs to formulate two boundary-type MFree methods (GR Liu and Gu, 2004a): the boundary point interpolation method (BPIM) (Gu and GR Liu, 2002a) and the boundary radial point interpolation method (BRPIM) (Gu and GR Liu, 2001a,e, 2003b). In BPIM and BRPIM, since the shape functions have the Kronecker delta function property, the boundary conditions can be

enforced as easily as in the conventional BEM. Hence, the BPIM and BRPIM are much more efficient than the methods using MLS shape functions.

In the late eighties, alternative boundary element formulations were developed based on generalized variational principles. DeFigueiredo and Brebbia (1991) proposed a hybrid boundary integral equation (HBIE). The HBIE leads to a symmetric stiffness matrix, which makes HBIE easy and accurate to combine with other numerical methods that produce symmetric system matrices. A hybrid boundary point interpolation method (HBPIM) and a hybrid boundary radial point interpolation method (HBRPIM) (Gu and GR Liu, 2002b, 2003a) were also formulated for solving boundary value problems. HBPIM and HBRPIM are formulated using the PIM and RPIM shape functions in HBIE. In HBPIM and HBRPIM, the *stiffness* matrices obtained are symmetric. This property of symmetry can be an added advantage in coupling HBPIM and HBRPIM with other established MFree methods that produce symmetric system matrices.

Some of the MFree methods are summarized in Table 2.2 based on the above-classifications.

2.5 FUTURE DEVELOPMENT

Table 2.3 lists a matrix of different possible ways to formulate an MFree method. It is clearly shown again from this matrix that MFree methods are proposed based on different combinations of interpolation /approximation techniques and formulation procedures. It should be noted that there are still some empty entries in the matrix. These empty entries may not be possible to be filled, or may not result in a good method a class of problems, but provide a window of possibilities for future development of ideal MFree methods.

The authors believe that the development of MFree methods has not only led to a group of useful numerical methods that are useful for a different classes of engineering problems, but also frees the minds of researchers from conventional ideas of numerical methods for further exploration of new numerical methods. The following four areas could be the future possible direction to develop ideal MFree methods.

- Development of new method for MFree function approximation;
- Development of new formulation procedures;

- Development of MFree methods based on different combinations of function approximations and formulation procedures;
- Development of MFree methods based on combinations of methods of function interpolation/approximations or formulation procedures for different parts of the problem domain or for different types of equations.

Table 2.2. Three categories of MFree methods

Classification	Categories	Example MFree methods[†]
Based on formulation procedure	MFree methods based on strong-forms of governing equations	MFree collocation methods, FPM etc.
	MFree methods based on weak-forms of governing equations	EFG, RPIM, MLPG, LRPIM , etc.
	MFree methods based on the combination of weak-form and strong-form	MWS, etc.
Based on interpolation /approximation method	MFree methods using MLS	EFG, MLPG, etc.
	MFree methods using integral representation method for function approximations	SPH, etc.
	MFree methods using PIM	RPIM, LRPIM, etc.
	MFree methods using other meshfree interpolation schemes.	PUFEM, *hp*-cloud, etc.
Based on domain representation	Domain-type MFree methods	SPH, EFG, RPIM, MLPG, LRPIM, etc.
	Boundary-type MFree methods	BNM, LBIE, BPIM, BRPIM, HBRPIM, etc.

[†] See Chapters 4-7 for more details on these methods.

Table 2.3. Matrix of different possible ways to formulate MFree methods

MFree methods / Formulation procedures / Interpolation /approximation schemes	Strong-form (collocation)	Galerkin weak-form	Petrov-Galerkin weak-form	Weak-strong-form	Least squares	Boundary Integral Equation	…
MLS	FPM	EFG	MLPG	MWS-MLS	?	BNM	?
WLS	Collocation GFDM	?	?	?	?	?	
RBF(global)	MFS collocation	Galerkin method	?	?	?	?	?
PIM	PPCM	PIM	LPIM	?	?	BPIM	?
RPIM	RPCM	RPIM	LRPIM	MWS-RPIM	?	BRPIM	?
Hermite-type interpolation /approximation	PPCM RPCM	?	LPIM, LRPIM	?	?	?	?
SPH(RKPM)	SPH, RKPM	RKPM	?	?	?	?	?
PU	*hp*-cloud	PUFEM *hp*-cloud	?	?	?	?	?
…	?	?	?	?	?	?	?

?: yet to be developed or unknown to the authors or it may not be possible.

Chapter 3

MESHFREE SHAPE FUNCTION CONSTRUCTION

3.1 INTRODUCTION

As we have seen in Chapter 2, in seeking for an approximate solution to a problem governed by PDEs and boundary conditions, one first needs to approximate the unknown field function using trial (shape) functions, before any formulation procedure can be applied to establish the discretized system equations. This chapter discusses various techniques for MFree shape function constructions. These shape functions are *locally supported*, because only a set of field nodes in a small local domain are used in the construction and the shape function is not used or regarded as zero outside the local domain. Such a local domain is termed the *support domain* or *influence domain* or *smoothing domain*[†].

In the finite element method (FEM), the shape functions are created using interpolation techniques based on elements formed by a set of fixed nodes. This type of interpolation is termed *stationary element* based interpolation. In MFree methods, the problem domain is usually represented by field nodes that are, in general, arbitrarily distributed. The field variables at an arbitrary point in the problem domain are approximated using a group of field nodes in a local support domain. Hence, a *moving domain* based interpolation/

[†] The difference between the support domain and the influence domain will be presented in Chapter 4.

approximation technique is necessary to construct the MFree shape function for the approximation of the field variables using a set of *arbitrarily* distributed nodes. In the development of an MFree method, the construction of efficient MFree shape functions is the foremost issue needed to be settled.

3.1.1 Meshfree interpolation/approximation techniques

A good method for creating MFree shape functions should satisfy some basic requirements.

1) It should be sufficiently robust for reasonably arbitrarily distributed nodes (*arbitrary nodal distribution*).
2) It should be numerically stable (*stability*).
3) It should satisfy up to a certain order of consistency (*consistency*).
4) It should be compactly supported (*compact*), i.e., it should be regarded as zero outside a bounded region, the *support domain*.
5) The approximated unknown function using the shape function should be compatible[†] (*compatibility*) throughout the problem domain when a global weak-form is used, or should be compatible within the local quadrature domain when a local weak-form is used.
6) It is ideal if the shape function possesses the Kronecker delta function property (*Delta function property*), i.e. the shape function is unit at the node and zero at other nodes in the support domain.
7) It should be computationally efficient (*efficiency*).

The requirement of *arbitrary nodal distribution* is essential for developing a robust MFree method for practical engineering problems.

The *stability* condition concerns two issues. The first is the *interpolation stability*, meaning that the shape functions constructed should be stable with respect to small perturbations of node locations in the support domain. This requires the moment matrix created using the arbitrarily distributed nodes to be well-conditioned. The interpolation stability will be briefly addressed in Sections 3.2 and 3.3. The second issue is the *solution stability*, meaning that the numerical solution using the shape functions together with a formulation procedure should not have the so-called *numerical* or *unphysical oscillations* that have been observed from, for example, convection dominated problems (see Chapter 6). For the second instability, even if the local interpolation is

[†] In an MFree method (or even FEM), the shape functions are constructed in a piecewise manner based on local support domains. Therefore, the field function approximated using these shape functions may not be continous when the support domain moves in the global problem domain. If the approximation is continous, we say it is compatiable, otherwise incompotable.

stable, the solution could be unstable due to the mismatch of the interpolation scheme (or formulation procedure) with the physical nature of the problem. Changing the interpolation schemes is a possible way to solve this problem. The formulation procedure can also play a very important role in producing a set of discretized system equation that produces a stable solution. This requires a properly designed formulation procedure based on the nature of the problem to have the dominant terms properly reflected in the formulation. This aspect of numerical treatment is addressed to certain degree of satisfaction in the finite difference method (FDM) by changing the interpolation scheme using so-called *upwind* grids (Courant et al., 1953; Runchall and Wolfstein, 1969; Spalding 1972; Barrett, 1974; etc.). It has also been well studied by Guymon et al., (1970), Adey and Brebbia (1974), Zienkiewicz et al. (1975), Christie et al. (1976), Morton (1985), Donea et al, (1985), Hughes et al. (1988), Oñate (1998), and many others for the FEM. More detailed discussions on this issue can be found in the book by Zienkiewicz and Taylor (2000) and the references provided there.

Unfortunately, the instability in MFree methods for convection dominated problems has not been properly addressed, and the issue is far from conclusive. Hence, this book will not provide concrete discussions on this topic, but will discuss some of the techniques in Chapter 6 for convection dominated problems.

Another type of solution instability often encountered is the well-known *tensile instability* that arises in applying the SPH approximation to hydrodynamics with material strength. Some discussions and measures have been developed (Swegle et al., 1995; Balsara; 1995; Dyka and Ingel, 1995; Morris 1996; Dyka et al., 1997; Monaghan, 2000; Randles and Libersky, 2000; Gray et al. 2001; GR Liu and Liu, 2003).

The consistence is important for an accurate function approximation and convergence of the MFree method.

The compact support is required to produce a set of sparse discretized system equations that can be solved effectively. This is extremely important for large systems.

When a global weak form is used, the global compatibility of the shape function, meaning that it has to be compatable in the entire (global) problem domain, is required. When local weighted residual methods of collocation methods are used for establishing the discretized system equations, only local compatibility in the local weighted domain is required.

The Kronecker delta function property is not rigid because one can use special measures to impose essential boundary conditions if the MFree shape function does not have this property (see Chapters 4~5).

The development of effective construction methods for MFree shape functions has been one of the hottest areas in the research of MFree methods. Several MFree approximation formulations have been proposed. GR Liu (2002) classified these formulations into three large categories based on the types of theories of function approximation/representation, i.e., the integral representation, the series representation, and the differential representation. Table 3.1 lists some of the techniques under these categories.

Table 3.1. Categories of MFree interpolation techniques

Categories	**MFree approximation techniques**
Integral representation	Smoothed Particle Hydrodynamics (SPH)
	Reproducing Kernel Particle Method (RKPM)
Series representation	Moving Least Squares (MLS)
	Point Interpolation Methods (PIM, RPIM)
	Partition of Unity (PU) methods
Differential representation	General Finite difference method (GFDM)

- In the integral representation method, the function is represented using its information in a local domain (smoothing domain or influence domain) via a weighted integral operation. The consistency is achieved by properly choosing the weight function. It is often used in the so-called smoothed particle hydrodynamics (SPH).
- The series representation methods have a long history. They are well developed in FEM and are now used in MFree methods based on arbitrary distributed nodes. The consistency is ensured by the completeness of the basis functions. The moving least square (MLS) approximation is the most widely used method. The point interpolation method (PIM) using radial basis function (or RPIM) is also often used. Both MLS and RPIM will be discussed in this chapter in detail.
- The differential representation method has also been developed and used for a long time in the finite difference method (FDM). The finite difference approximation is not globally compatible (see, e.g., Zienkiewicz and Tayler, 2000), and the consistency is ensured by the theory of Taylor series. Differential representation methods are usually used for establishing system equations based on strong-form formulations, such as FDM and the general finite difference method (GFDM).

In the following sections, we will discuss the point interpolation method (PIM) and the moving least squares (MLS) approximation in detail. There are other methods of constructing MFree shape functions, such as the SPH approximation, the *hp*-clouds method, and partitions of unity finite element method, and so on. Readers can refer to the related references for more details and more precise descriptions.

Before introducing the MFree interpolants, the concept of support domain that is often used in the MFree interpolation operations is introduced.

3.1.2 Support domain

The accuracy of interpolation for the point of interest depends on the nodes in the support domain as shown in Figure 3.1. Therefore, a suitable support domain should be chosen to ensure an efficient and accurate approximation. For a point of interest at $\mathbf{x}_Q$, the dimension of the support domain d_s is determined by

$$d_s=\alpha_s d_c \tag{3.1}$$

where α_s is the dimensionless size of the support domain, and d_c is the nodal spacing near the point at $\mathbf{x}_Q$. If the nodes are uniformly distributed, d_c is simply the distance between two neighboring nodes. When nodes are non-uniformly distributed, d_c can be defined as an average nodal spacing in the support domain of $\mathbf{x}_Q$.

The dimensionless size of the support domain α_s controls the actual dimension of the support domain. For example, α_s=2.1 means a support domain whose radius is 2.1 times the average nodal spacing. The actual number of nodes, n, can be determined by counting all the nodes included in the support domain. Note that α_s should be pre-determined by the analyst before analysis, and it is usually determined by carrying out numerical experiments for a class of benchmark problems for which we already have solutions. Generally, an α_s=2.0~3.0 leads to good results for many problems that we have studied.

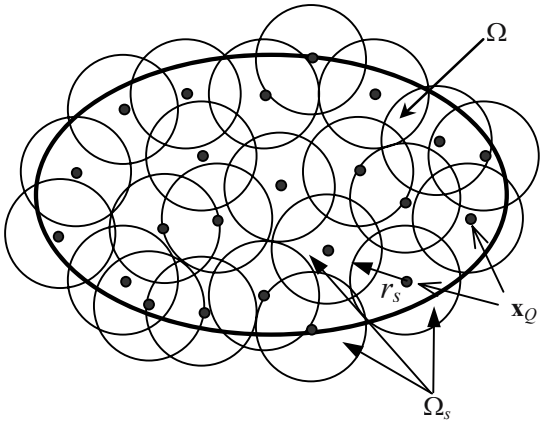
Note that the support domain is usually centered by a point of interest at $\mathbf{x}_Q$. Biased support domains can also be used for special problems such as convection dominated problems (see, Section 6.4).

3.1.3 Determination of the average nodal spacing

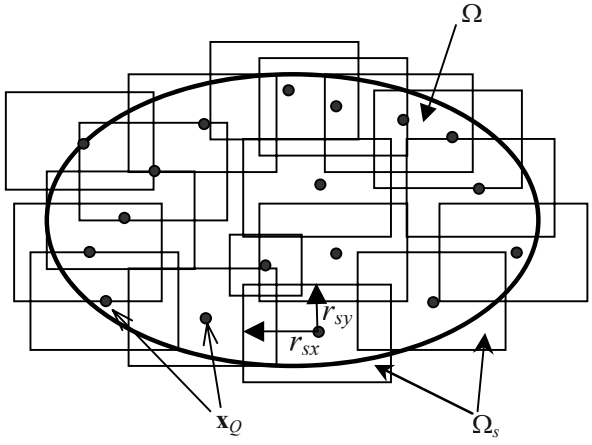
For one-dimensional cases, the simplest method of defining an average nodal spacing could be

$$d_c = \frac{D_s}{(n_{Ds}-1)} \tag{3.2}$$

where D_s is an estimated d_s (in Equation (3.1)) that does not have to be very accurate but should be known and is a reasonably good estimate of d_s, and n_{Ds} is the number of nodes covered by the domain with the dimension of D_s.



(a)



(b)

Figure 3.1. Support domains of points of interest at $\mathbf{x}_Q$ in MFree models. (a) circular support domains (r_s: the dimension of the support domain); (b) rectangular support domains (r_{sx} and r_{sy}: dimensions of the support domain in x and y directions). The support domain is centred by $\mathbf{x}_Q$.

For two-dimensional cases, the simplest method of defining an average nodal spacing could be

$$d_c = \frac{\sqrt{A_s}}{\sqrt{n_{A_s}} - 1} \tag{3.3}$$

where A_s is the area of the estimated support domain. The estimate does not have to be accurate, but should be known and should be a reasonably good estimate; n_{As} is the number of nodes covered by the estimated domain with the area of A_s.

Similarly, for three-dimensional cases, the simplest method of defining an average nodal spacing could be

$$d_c = \frac{\sqrt[3]{V_s}}{\sqrt[3]{n_{Vs}} - 1} \tag{3.4}$$

where V_s is the volume of the estimated support domain, and n_{Vs} is the number of nodes covered by the estimated domain with the volume of V_s.

After determining d_c, using Equation (3.1), we can easily determine the dimension of the support domain d_s for a point at $\mathbf{x}_Q$ in a domain with non-uniformly distributed nodes. The procedure is

1. Estimate d_s for the point at $\mathbf{x}_Q$, which gives D_s or A_s or V_s;
2. Count nodes that are covered by D_s or A_s or V_s, which yields n_{D_s}, n_{A_s}, and n_{V_s};
3. Use Equation (3.2) or (3.3) or(3.4) to calculate d_c;
4. Calculate d_s using Equation (3.1), for a given (desired) dimensionless size of support domain.

3.2 POINT INTERPOLATION METHODS

The point interpolation method (PIM) is one of the series representation methods for the function approximation, and is useful for creating MFree shape functions. Consider a scalar function $u(\mathbf{x})$ defined in the problem domain Ω that is represented by a set of scattered nodes. The PIM approximates $u(\mathbf{x})$ at a point of interest $\mathbf{x}$ in the form of

$$u(\mathbf{x}) = \sum_{i=1}^{m} B_i(\mathbf{x}) a_i \tag{3.5}$$

where the $B_i(\mathbf{x})$ are the basis function defined in the space Cartesian coordinates $\mathbf{x}^T=[x, y]$, m is the number of basis functions, and the a_i are the coefficient.

For function approximation, a local support domain is first formed for the point of interest at $\mathbf{x}$ which includes a total of n field nodes. For the conventional point interpolation method (PIM), $n=m$ is used that results in the conventional PIM shape functions that pass through the function values at each scattered node within the defined support domain. For the weighted least square (WLS) approximation or the moving least squares (MLS) approximation, n is always larger than m.

There are two types of PIM shape functions have been developed so far using different forms of basis functions. Polynomial basis functions (GR Liu and Gu, 1999; 2001a) and radial basis functions (RBF) (Wang and GR Liu, 2000; GR Liu, 2002) have often been used in MFree methods. These two-types of PIMs will be discussed in the following sections.

3.2.1 Polynomial PIM shape functions

3.2.1.1 Conventional polynomial PIM

Using polynomials as basis functions in the interpolation is one of the earliest interpolation schemes. It has been widely used in establishing numerical methods, such as the FEM. Consider a continuous function $u(\mathbf{x})$ defined in a domain Ω, which is represented by a set of field nodes. The $u(\mathbf{x})$ at a point of interest $\mathbf{x}$ is approximated in the form of

$$u(\mathbf{x})=\sum_{i=1}^{m} p_i(\mathbf{x})a_i=\underbrace{\{p_1(\mathbf{x}) \quad p_2(\mathbf{x}) \quad \cdots \quad p_m(\mathbf{x})\}}_{\mathbf{p}^T}\underbrace{\begin{Bmatrix} a_1 \\ \vdots \\ a_m \end{Bmatrix}}_{\mathbf{a}}=\mathbf{p}^T\mathbf{a} \tag{3.6}$$

where $p_i(\mathbf{x})$ is a given monomial in the polynomial basis function in the space coordinates $\mathbf{x}^T=[x, y]$, m is the number of monomials, and a_i is the coefficient for $p_i(\mathbf{x})$ which is yet to be determined. The $p_i(\mathbf{x})$ in Equation (3.6) is built using Pascal's triangles (see Figure 3.2), and a complete basis is usually (but not always) preferred. For one-dimensional (1-D) and two-dimensional (2-D) space, the linear basis functions are given by

$$\mathbf{p}^T(\mathbf{x})=\{1 \quad x\} \qquad m=2,\ p=1 \qquad \text{(1-D)} \tag{3.7}$$

$$\mathbf{p}^T(\mathbf{x})=\{1 \quad x \quad y\} \qquad m=3,\ p=1 \qquad \text{(2-D)} \tag{3.8}$$

and the quadratic basis functions are

$$\mathbf{p}^T(\mathbf{x})=\{1 \quad x \quad x^2\} \qquad m=3,\ p=2 \qquad \text{(1-D)} \tag{3.9}$$

$$\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = \{1 \quad x \quad y \quad x^2 \quad xy \quad y^2\} \qquad m=6,\ p=2 \quad \text{(2-D)} \tag{3.10}$$

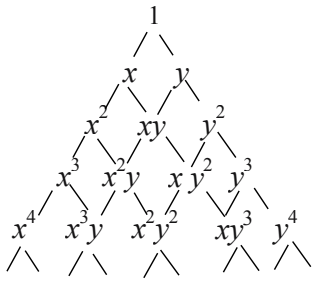


Figure 3.2. Pascal triangle of monomials for two-dimensional domains.

The complete polynomial basis of order p can be written in the following general form.

$$\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = \{1 \quad x \quad x^2 \quad \cdots \quad x^{p-1} \quad x^p\} \qquad \text{(1-D)} \tag{3.11}$$

$$\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = \{1 \quad x \quad y \quad x^2 \quad xy \quad y^2 \quad \cdots \quad x^p \quad y^p\} \qquad \text{(2-D)} \tag{3.12}$$

In order to determine the coefficients a_i, a support domain is formed for the point of interest at $\mathbf{x}$, with a total of n field nodes included in the support domain. Note that in the conventional PIM, the number of nodes in the local support domain, n, always equals the number of basis functions of, m, i.e., $n=m$. The coefficients a_i in Equation (3.6) can then be determined by enforcing $u(\mathbf{x})$ in Equation (3.6) to pass through the nodal values at these n nodes. This yields n equations with each for one node, i.e.,

$$\left.\begin{aligned} u_1 &= \sum_{i=1}^{m} a_i p(\mathbf{x}_1) = a_1 + a_2 x_1 + a_3 y_1 + \cdots + a_m p_m(\mathbf{x}_1) \\ u_2 &= \sum_{i=1}^{m} a_i p(\mathbf{x}_2) = a_1 + a_2 x_2 + a_3 y_2 + \cdots + a_m p_m(\mathbf{x}_2) \\ &\vdots \\ u_n &= \sum_{i=1}^{m} a_i p(\mathbf{x}_n) = a_1 + a_2 x_n + a_3 y_n + \cdots + a_m p_m(\mathbf{x}_n) \end{aligned}\right\} \tag{3.13}$$

which can be written in the following matrix form.

$$\mathbf{U}_s = \mathbf{P}_m \mathbf{a} \tag{3.14}$$

where

$$\mathbf{U}_s = \{u_1 \quad u_2 \quad u_3 \quad ... \quad u_n\}^{\mathrm{T}} \tag{3.15}$$

is the vector of nodal function values, and

$$\mathbf{a} = \{a_1 \quad a_2 \quad a_3 \quad ... \quad a_n\}^{\mathrm{T}} \tag{3.16}$$

is the vector of unknown coefficients, and

$$\mathbf{P}_m = \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 & ..... & p_m(\mathbf{x}_1) \\ 1 & x_2 & y_2 & x_2y_2 & ..... & p_m(\mathbf{x}_2) \\ 1 & x_3 & y_3 & x_3y_3 & ..... & p_m(\mathbf{x}_3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & y_n & x_ny_n & ..... & p_m(\mathbf{x}_n) \end{bmatrix} \tag{3.17}$$

is the so-called *moment matrix*. Because of n=m in PIM, $\mathbf{P}_m$ is hence a square matrix with the dimension of (n×n or m×m).

Solving Equation (3.14) for **a**, we obtain

$$\mathbf{a} = \mathbf{P}_m^{-1}\mathbf{U}_s \tag{3.18}$$

In obtaining the foregoing equations, we have assumed $\mathbf{P}_m^{-1}$ exists, and left the issue regarding non-existence of $\mathbf{P}_m^{-1}$ to be addressed later.

It is noted that coefficients **a** are constants even if the point of interest at **x** changes, as long as the same set of n nodes are used in the interpolation, because $\mathbf{P}_m$ is a matrix of constants for this given set of nodes.

Substituting Equation (3.18) back into Equation (3.6) and considering n=m yield

$$u(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{P}_m^{-1}\mathbf{U}_s = \sum_{i=1}^{n} \phi_i u_i = \mathbf{\Phi}^{\mathrm{T}}(\mathbf{x})\mathbf{U}_s \tag{3.19}$$

where $\mathbf{\Phi}(\mathbf{x})$ is a vector of shape functions defined by

$$\mathbf{\Phi}^{\mathrm{T}}(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{P}_m^{-1} = \{\phi_1(\mathbf{x}) \quad \phi_2(\mathbf{x}) \quad \cdots \quad \phi_n(\mathbf{x})\} \tag{3.20}$$

The derivatives of the shape functions can be easily obtained because the PIM shape function is of polynomial form. The lth derivatives of PIM shape functions can be written as

$$\mathbf{\Phi}^{(l)}(\mathbf{x}) = \begin{Bmatrix} \phi_1^{(l)}(\mathbf{x}) \\ \phi_2^{(l)}(\mathbf{x}) \\ \vdots \\ \phi_n^{(l)}(\mathbf{x}) \end{Bmatrix} = \frac{\partial^l \mathbf{p}^{\mathrm{T}}(\mathbf{x})}{\partial \mathbf{x}^l}\mathbf{P}_m^{-1} \tag{3.21}$$

The properties of PIM shape functions (GR Liu, 2002) can be summarized as follows.

1) Consistency

The consistency of the polynomial PIM shape function depends on the highest complete order of the monomial $p_i(\mathbf{x})$ used in Equation (3.6). If the complete order of monomial is p, the shape function will possess C^p consistency. This is because the PIM shape functions can reproduce the monomials that are included in the basis used to construct the shape functions. To demonstrate, we consider a field given by

$$f(\mathbf{x}) = \sum_{j=1}^{k} p_j(\mathbf{x}) b_j, \qquad k \le m \tag{3.22}$$

where $p_j(\mathbf{x})$ are monomials that are included in Equation (3.6). Such a given field can always be written in the form of Equation (3.6).

$$f(\mathbf{x}) = \sum_{j=1}^{m} p_j(\mathbf{x}) a_j = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a} = \mathbf{p}^{\mathrm{T}}(\mathbf{x}) \begin{Bmatrix} b_1 \\ \vdots \\ b_k \\ 0 \\ \vdots \\ 0 \end{Bmatrix} \tag{3.23}$$

Using n (here n=m) nodes in the support domain of $\mathbf{x}$, we can obtain the vector of nodal function values $\mathbf{U}_s$ as

$$\begin{aligned} \mathbf{U}_s &= \begin{Bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_k) \\ f(x_{k+1}) \\ \vdots \\ f(x_n) \end{Bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1 y_1 & ..... & p_m(\mathbf{x}_1) \\ 1 & x_2 & y_2 & x_2 y_2 & ..... & p_m(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & y_k & x_k y_k & ..... & p_m(\mathbf{x}_k) \\ 1 & x_{k+1} & y_{k+1} & x_{k+1} y_{k+1} & ..... & p_m(\mathbf{x}_{k+1}) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & y_n & x_n y_n & ..... & p_m(\mathbf{x}_n) \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ ... \\ b_k \\ 0 \\ ... \\ 0 \end{Bmatrix} \\ &= \mathbf{P}_m \mathbf{a} \end{aligned} \tag{3.24}$$

Substitute Equation (3.24) into Equation (3.19), we have

$$u(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x})\mathbf{U}_s = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{P}_m^{-1}\mathbf{P}_m\mathbf{a} = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a} = \sum_{j=1}^{k} p_j(\mathbf{x}) b_j \tag{3.25}$$

This proves that any polynomial field function given by Equation (3.6) will be exactly reproduced by PIM interpolation, as long as the basis functions of the field function are included in the basis functions of the PIM shape functions. This can always be done as long as the moment matrix $\mathbf{P}_m$ is invertible so as to ensure the uniqueness of the solution for the coefficient **a**.

2) Reproducibility

On the extension of proving the consistency of the polynomial PIM shape functions, we can conclude that PIM shape functions can reproduce any functions (not necessarily a polynomial) that are included in the basis functions.

3) Linear independence

PIM shape functions are linearly independent in the support domain. This is because basis functions are linear independent, and $\mathbf{P}_m$ is assumed to be invertible.

4) Delta function property

Shape functions have the Kronecker delta function property, that is

$$\phi_i\left(\mathbf{x}=\mathbf{x}_j\right)=\begin{cases}1 & i=j,\quad j=1,2,\cdots,n\\ 0 & i\neq j,\quad i,j=1,2,\cdots,n\end{cases} \tag{3.26}$$

This is because the PIM shape functions are created to pass thorough nodal values.

5) Partitions of unity

If the constant is included in the basis, the $\phi_i\left(\mathbf{x}\right)$ is form a partition of unity, i.e.,

$$\sum_{i=1}^{n}\phi_i(\mathbf{x})=1 \tag{3.27}$$

This can be proven easily from the reproducibility feature of the polynomial PIM shape functions. For a given constant field $u(\mathbf{x})=c$, we have

$$u_1=u_2=\cdots=u_n=c \tag{3.28}$$

Because the constant field can be reproduced using PIM shape functions, we obtain

$$u(\mathbf{x})=c=\sum_{i=1}^{n}\phi_i u_i=c\sum_{i=1}^{n}\phi_i \tag{3.29}$$

which leads to

$$\sum_{i=1}^{n} \phi_i = 1 \tag{3.30}$$

This shows that the polynomial PIM shape functions possess the partitions of unity.

6) Linear reproducibility

PIM shape functions have the linear reproducibility, i.e.,

$$\sum_{i=1}^{n} \phi_i(\mathbf{x}) x_i = \mathbf{x} \tag{3.31}$$

if the complete 1st order monomials are included in the basis. This can also be proven easily from the reproducibility of the PIM shape functions.

7) Polynomial form

PIM shape functions and their derivatives have polynomial forms.

8) Compact support

The PIM shape function is constructed using nodes in a compact support domain, and its' value at any point outside the support domain is regarded as zero when it is used in MFree method.

9) Compatibility

In using PIM shape functions, the compatibility in the global domain is not ensured when the local support domain is used, and the field function approximated could be discontinuous when nodes enter or leave the moving support domain. Because no bell shape weight function is used in PIM, the nodes in the support domain are updated suddenly, meaning that when the nodes are entering or leaving the support domain, they are actually "jumping" into or out of the support domain (GR Liu and Gu, 2004c). Care must be taken when a global weak-form is used together with PIM shape functions with compact supports. The global compatibility is not an issue when the strong-forms or the local weak-forms are used.

Note that our discussion is based on the assumption that $\mathbf{P}_m^{-1}$ exists. This condition cannot always be satisfied depending on the locations of the nodes in the support domain and the terms of monomials used in the basis. If an inappropriate polynomial basis is chosen for a given set of nodes, it may yield in a badly conditioned or even singular moment matrix. There are a number of ways to solve the singularity problem. The most practical method is the use of the matrix triangularization algorithm (MTA) (GR Liu and Gu, 2001d, 2003a) and the use of the radial basis functions (RBFs) in place of the polynomial basis (Sub-section 3.2.2). In addition, the weighted least

square (WLS) method can also be used to overcome the singularity problem in the polynomial PIM. The WLS approximation will be discussed in the following section.

3.2.1.2 Weighted least square (WLS) approximation

The weighted least square (WLS) approximation is a widely used technique for data fitting. In the WLS, the number of basis, m, is usually pre-determined according to the requirements on the consistency for shape functions. Using Equation (3.5), we can write a two-dimensional field function $u(\mathbf{x})$ approximated using the polynomial basis as follows.

$$\begin{aligned} u^h(\mathbf{x}) &= \sum_{i=1}^{m} p_i(\mathbf{x}) a_i = a_1 + a_2 x + a_3 y + \cdots + a_m p_m(\mathbf{x}) \\ &= \underbrace{\{1 \quad x \quad y \quad \cdots \quad p_m(\mathbf{x})\}}_{\mathbf{p}^{\mathrm{T}}} \underbrace{\begin{Bmatrix} a_1 \\ \vdots \\ a_m \end{Bmatrix}}_{\mathbf{a}} = \mathbf{p}^{\mathrm{T}}\mathbf{a} \end{aligned} \tag{3.32}$$

where a_i (i=1, 2, …, m) are the coefficients to be determined, and $\mathbf{p}$ is the vector of basis functions.

To determine coefficients $\mathbf{a}$ in Equation (3.32), n nodes are selected in the local support domain for the approximation. Note that in the WLS, $n>m$ is used. Using Equation (3.32) for all these n nodes, we can obtain the similar equations of Equations (3.14)~(3.17), i.e.,

$$\mathbf{U}_s = (\mathbf{P}_m)_{(n\times m)} \mathbf{a}_{(m\times 1)} \tag{3.33}$$

The moment matrix, $\mathbf{P}_m$, is

$$\mathbf{P}_m = \begin{bmatrix} 1 & x_1 & y_1 & x_1 y_1 & ..... & p_m(\mathbf{x}_1) \\ 1 & x_2 & y_2 & x_2 y_2 & ..... & p_m(\mathbf{x}_2) \\ 1 & x_3 & y_3 & x_3 y_3 & ..... & p_m(\mathbf{x}_3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & y_n & x_n y_n & ..... & p_m(\mathbf{x}_n) \end{bmatrix}_{(n\times m)} \tag{3.34}$$

Note that $\mathbf{P}_m$ is not a square matrix because $n>m$.

Equation (3.33) is a set of overdetermined system of equations due to $n>m$ meaning that the number of equations is more than the number of unknowns. We can solve Equation (3.33) for $\mathbf{a}$ using the standard weighted least squares (WLS) method by minimizing the following weighted discrete $\mathbf{L}_2$ norm:

$$J = \sum_{i=1}^{n} \widehat{W}_i [u^h(\mathbf{x}_i) - u(\mathbf{x}_i)]^2 \tag{3.35}$$

where $\widehat{W}_i$ (i=1, 2, …, n) is the weight coefficient associated with the function value at the ith node in the support domain, and u_i becomes the "nodal parameter" of u at $\mathbf{x}=\mathbf{x}_i$. The stationary condition for J is

$$\frac{\partial J}{\partial \mathbf{a}} = 0 \tag{3.36}$$

which leads to the following linear relation between $\mathbf{a}$ and $\mathbf{U}_s$

$$\mathbf{P}_m^{\mathrm{T}} \widehat{\mathbf{W}} \mathbf{P}_m \mathbf{a} = \mathbf{P}_m^{\mathrm{T}} \widehat{\mathbf{W}} \mathbf{U}_s \tag{3.37}$$

where $\widehat{\mathbf{W}}$ is the diagonal matrix constructed from the weight constants, i.e.,

$$\widehat{\mathbf{W}}_{(n\times n)} = \begin{bmatrix} \widehat{W}_1 & \widehat{W}_2 & \cdots & \widehat{W}_n \end{bmatrix} \tag{3.38}$$

Note that the weights used here are considered as constants (not functions of $\mathbf{x}$) that define the different influences of the nodes in the approximation. The further nodes should have smaller influences while closer nodes have bigger influences, $\widehat{W}_i$ can be computed from any weight function with the bell shape that will be provided in Section 3.3.2. For example, the following formulations of $\widehat{W}_i$ can be used.

$$\widehat{W}_i = \widehat{W}(\mathbf{x}_i) = \frac{e^{-(\frac{r}{c})^2} - e^{-(\frac{r_s}{c})^2}}{1 - e^{-(\frac{r_s}{c})^2}} \tag{3.39}$$

$$r = \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{3.40}$$

where (x, y) is coordinate of the point of interest, r_s is the size of local supported domain, and c is a constant to be determined by the analyst before calculation.

We now let

$$\mathbf{A} = \mathbf{P}_m^{\mathrm{T}} \widehat{\mathbf{W}} \mathbf{P}_m \tag{3.41}$$

$$\mathbf{B} = \mathbf{P}_m^{\mathrm{T}} \widehat{\mathbf{W}} \tag{3.42}$$

Solving Equation (3.37) for $\mathbf{a}$ yields

$$\mathbf{a} = (\mathbf{P}_m^{\mathrm{T}}\widehat{\mathbf{W}}\mathbf{P}_m)^{-1}(\mathbf{P}_m^{\mathrm{T}}\widehat{\mathbf{W}})\mathbf{U}_s \tag{3.43}$$

or

$$\mathbf{a} = \mathbf{A}^{-1}\mathbf{B}\mathbf{U}_s \tag{3.44}$$

Substituting Equation (3.44) back into Equation (3.32), we have

$$u^h(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}\mathbf{a} = \mathbf{p}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{B}\mathbf{U}_s = \mathbf{\Phi}^{\mathrm{T}}\mathbf{U}_s \tag{3.45}$$

where the vector of shape functions $\mathbf{\Phi}$ is

$$\mathbf{\Phi}^{\mathrm{T}} = \mathbf{p}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{B} = \{\phi_1 \quad \phi_2 \quad \cdots \quad \phi_n\} \tag{3.46}$$

where ϕ_i (i=1, 2, …, n) is the shape function corresponding to the ith node in the support domain.

Equation (3.45) is the approximation equation for the WLS. Because the weighted least squares method is used, the shape functions so constructed do not have the Kronecker delta function property, which can cause difficulties in imposing essential boundary conditions, if it is used in MFree methods based on global weak-from such as the Galerkin weak-form (Chapter 4). However, it is not a big issue in the MFree methods based on local weak-forms (Chapter 5) or the MFree collocation methods (Chapter 6), because the *direct interpolation method* can be used to enforce the essential boundary conditions. Note also that the WLS shape functions are compatible only in the local support domain rather than in the global domain. This is not a problem when the WLS shape functions are used in the local weak-form methods or collocation methods, but care needs to be taken when it is used for global weak-form formulation, for which the moving least squares (MLS) to be discussed in Section 3.3 is a better choice.

3.2.1.3 Weighted least square approximation of Hermite-type

In some problems, the normal derivatives of field functions at some nodes are important and need to be considered as independent variables. For example, in order to impose the stress (derivative) boundary conditions in the analysis of solid mechanics problems using the MFree strong-form methods (see, Chapter 6), the normal derivatives of displacements at the nodes on the derivative boundaries (called DB-nodes) are often considered as independent variables in the function approximation. This is the so-called Hermite-type approximation. In this section, the Hermit-type WLS is discussed, which is an extension of the WLS and will be used in Chapter 6.

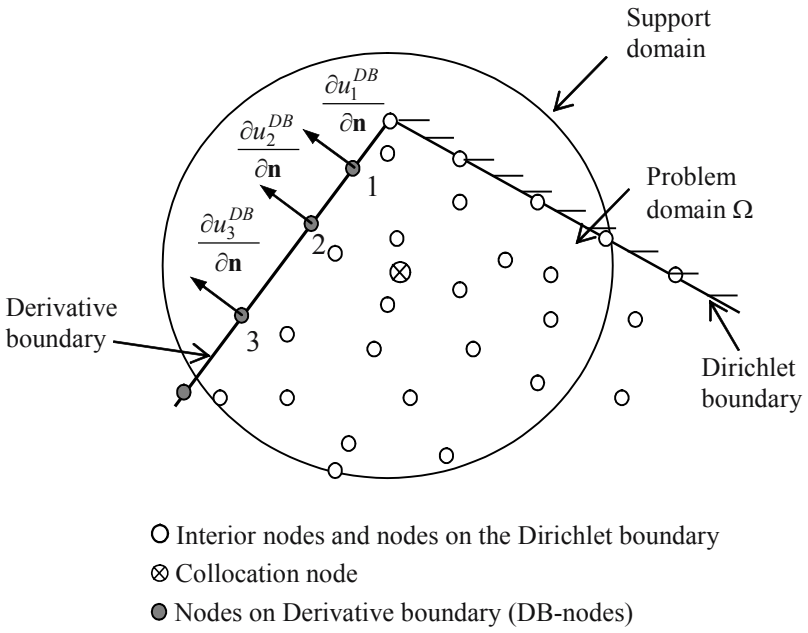


Figure 3.3 Hermite-type interpolation with normal derivatives as additional degrees of freedom.

To determine coefficients **a** in Equation (3.32), *n* nodes are selected in the local support domain for the approximation. The normal derivatives of the function at the DB-nodes shown in Figure 3.3 are considered as variables in addition to the variables of the nodal function values. Applying Equation (3.32) to all these *n* nodes, we have

$$\left.\begin{aligned} u_1 &= a_1 + a_2 x_1 + a_3 y_1 + \cdots + a_m p_m(\mathbf{x}_1) \\ u_2 &= a_1 + a_2 x_2 + a_3 y_2 + \cdots + a_m p_m(\mathbf{x}_2) \\ &\vdots \\ u_n &= a_1 + a_2 x_n + a_3 y_n + \cdots + a_m p_m(\mathbf{x}_n) \end{aligned}\right\} \tag{3.47}$$

or

$$\mathbf{U}_s = \mathbf{P}_m \mathbf{a} \tag{3.48}$$

where the moment matrix $\mathbf{P}_m$ is given in Equation (3.34), and

$$\mathbf{U}_s = \{u_1 \quad u_2 \quad u_3 \quad \ldots \quad u_n\}^{\mathrm{T}} \tag{3.49}$$

Now applying Equation (3.32) to these n_{DB} DB-nodes, we have

$$\frac{\partial u(\mathbf{x}_i^{DB})}{\partial \mathbf{n}} = l_{xi}\frac{\partial u(\mathbf{x}_i^{DB})}{\partial x} + l_{yi}\frac{\partial u(\mathbf{x}_i^{DB})}{\partial y} \tag{3.50}$$

and $\mathbf{n}$ is the vector of unit outwards normal, and l_{xi} and l_{yi} are the direction cosines for the outward normal at the DB-node at (x_i^{DB}, y_i^{DB}), which are defined by

$$\begin{cases} l_{xi} = \cos(\mathbf{n}, x_i^{DB}) \\ l_{yi} = \cos(\mathbf{n}, y_i^{DB}) \end{cases} \tag{3.51}$$

Using Equation (3.50) for all DB-nodes, we can obtain

$$\left.\begin{aligned} &\frac{\partial u(\mathbf{x}_1^{DB})}{\partial \mathbf{n}} = 0a_1 + a_2 l_{x1} + a_3 l_{y1} + a_4(y_1^{DB} l_{x1} + x_1^{DB} l_{y1}) + \cdots \\ &\frac{\partial u(\mathbf{x}_2^{DB})}{\partial \mathbf{n}} = 0a_1 + a_2 l_{x2} + a_3 l_{y2} + a_4(y_2^{DB} l_{x2} + x_2^{DB} l_{y2}) + \cdots \\ &\vdots \\ &\frac{\partial u(\mathbf{x}_{n_{DB}}^{DB})}{\partial \mathbf{n}} = 0a_1 + a_2 l_{xn_{DB}} + a_3 l_{yn_{DB}} + a_4(y_{n_{DB}}^{DB} l_{xn_{DB}} + x_{n_{DB}}^{DB} l_{yn_{DB}}) + \cdots \end{aligned}\right\} \tag{3.52}$$

Equation (3.52) can be written as the matrix form of

$$\mathbf{U}_s' = \mathbf{P}_D\mathbf{a} \tag{3.53}$$

where $\mathbf{U}_s'$ is the vector that collects all the normal derivatives of function values at the DB-nodes

$$\mathbf{U}_s'^{\mathrm{T}} = \left\{ \frac{\partial u(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}} \quad \frac{\partial u(\mathbf{x}_2^{\mathrm{DB}})}{\partial \mathbf{n}} \quad \cdots \quad \frac{\partial u(\mathbf{x}_{n_{\mathrm{DB}}}^{\mathrm{DB}})}{\partial \mathbf{n}} \right\} \tag{3.54}$$

and the moment matrix $\mathbf{P}_D$ is

$$\mathbf{P}_D = \begin{bmatrix} 0 & l_{x1} & l_{y1} & \cdots & l_{x1}\dfrac{\partial p_m(\mathbf{x}_1^{DB})}{\partial x} + l_{y1}\dfrac{\partial p_m(\mathbf{x}_1^{DB})}{\partial y} \\ 0 & l_{x2} & l_{y2} & \cdots & l_{x2}\dfrac{\partial p_m(\mathbf{x}_2^{DB})}{\partial x} + l_{y2}\dfrac{\partial p_m(\mathbf{x}_2^{DB})}{\partial y} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & l_{xn_{DB}} & l_{yn_{DB}} & \cdots & l_{xn_{DB}}\dfrac{\partial p_m(\mathbf{x}_{n_{DB}}^{DB})}{\partial x} + l_{yn_{DB}}\dfrac{\partial p_m(\mathbf{x}_{n_{DB}}^{DB})}{\partial y} \end{bmatrix} \tag{3.55}$$

The dimension of $\mathbf{P}_D$ is ($n_{DB}\times m$).

Combining Equations (3.48) and (3.53) gives

$$\mathbf{U}_s^D = \mathbf{P}\mathbf{a} = \begin{Bmatrix} \mathbf{P}_m \\ \mathbf{P}_D \end{Bmatrix} \mathbf{a} \tag{3.56}$$

where $\mathbf{U}_s^D$ is the vector that collects all the nodal values of the function at n nodes and all nodal normal derivatives of the function at the n_{DB} DB-nodes, i.e.,

$$\mathbf{U}_s^D = \left\{ u(\mathbf{x}_1) \quad \cdots \quad u(\mathbf{x}_n) \quad \frac{\partial u(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}} \quad \cdots \quad \frac{\partial u(\mathbf{x}_{n_{\mathrm{DB}}}^{\mathrm{DB}})}{\partial \mathbf{n}} \right\}^{\mathrm{T}} \tag{3.57}$$

in which $\mathbf{x}_1$, $\mathbf{x}_2$~$\mathbf{x}_n$ are coordinates for the n nodes in the support domain, $\mathbf{x}_1^{\mathrm{DB}}, \mathbf{x}_2^{\mathrm{DB}} \sim \mathbf{x}_{n_{\mathrm{DB}}}^{\mathrm{DB}}$ are coordinates of the DB-nodes whose normal derivatives are considered as independent variables.

In Equation (3.56), $\mathbf{P}$ is the moment matrix that can be written as

$$\mathbf{P} = \begin{bmatrix} (\mathbf{P}_m)_{n\times m} \\ (\mathbf{P}_D)_{n_{\mathrm{DB}}\times m} \end{bmatrix}_{((n+n_{\mathrm{DB}})\times m)} \tag{3.58}$$

Equation (3.56) is a set of overdetermined system of equations due to $n+n_{\mathrm{DB}}>m$ meaning that the number of equations is larger than the number of unknowns. We can obtain the solution for Equation (3.56) using the standard weighted least squares method by minimizing the following weighted discrete $\mathbf{L}_2$ norm of

$$J = \sum_{i=1}^{n} \widehat{W}_i [u^h(\mathbf{x}_i) - u(\mathbf{x}_i)]^2 + \sum_{j=1}^{n_{DB}} \widehat{W}_j^{\,DB} [\frac{\partial u^h(\mathbf{x}_j^{\mathrm{DB}})}{\partial \mathbf{n}} - \frac{\partial u(\mathbf{x}_j^{\mathrm{DB}})}{\partial \mathbf{n}}]^2 \tag{3.59}$$

where $\widehat{W}_i$ and $\widehat{W}_j^{\,DB}$ are weight coefficients, and u_i and $\frac{\partial u(\mathbf{x}_j^{\mathrm{DB}})}{\partial \mathbf{n}}$ are the *nodal parameters* of u at $\mathbf{x}=\mathbf{x}_i$ and the normal derivatives of u at $\mathbf{x}=\mathbf{x}_j^{\mathrm{DB}}$. The stationary condition of J requires

$$\frac{\partial J}{\partial \mathbf{a}} = 0 \tag{3.60}$$

which leads to the following linear relation between $\mathbf{a}$ and $\mathbf{U}_s^D$

$$\mathbf{P}^{\mathrm{T}}\widehat{\mathbf{W}}\mathbf{P}\mathbf{a} = \mathbf{P}^{\mathrm{T}}\widehat{\mathbf{W}}\mathbf{U}_s^D \tag{3.61}$$

where $\widehat{\mathbf{W}}$ is the diagonal matrix constructed using a weight function, i.e.,

$$\widehat{\mathbf{W}} = \begin{bmatrix} \widehat{W}_1 & & & & & \\ & \ddots & & & 0 & \\ & & \widehat{W}_n & & & \\ & & & \widehat{W}_1^{DB} & & \\ & 0 & & & \ddots & \\ & & & & & \widehat{W}_{n_{DB}}^{DB} \end{bmatrix}_{(n+n_{\mathrm{DB}})\times(n+n_{\mathrm{DB}})} = \begin{bmatrix} \mathbf{W}_0 & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_D \end{bmatrix} \tag{3.62}$$

In which $\widehat{W}_i$ (i=1,2, …, n) is the weight coefficient associated with the function value for the ith node in the support domain, and $\widehat{W}_j^{DB}$ (j=1, 2, .., n_{DB}) is the weight coefficient associated with the jth DB-node. The weight function $\widehat{W}_i$ can be obtained using any function such as the one given in Equation (3.39), and $\widehat{W}_j^{DB}$ can be given independently in the similar manner. The weight function $\widehat{W}_j^{DB}$ can also be obtained using $\widehat{W}_j$. For example, we may define

$$\widehat{W}_j^{DB} = \frac{1}{\beta}\widehat{W}(\mathbf{x} - \mathbf{x}_j^{DB}) \tag{3.63}$$

where β is the constant to be determined before analysis.

Considering the fact that $\widehat{\mathbf{W}}$ is a diagonal matrix, we now let

$$\mathbf{A} = (\mathbf{P}^{\mathrm{T}}\widehat{\mathbf{W}}\mathbf{P}) = (\mathbf{P}_m^{\mathrm{T}}\widehat{\mathbf{W}}_0\mathbf{P}_m) + (\mathbf{P}_D^{\mathrm{T}}\widehat{\mathbf{W}}_D\mathbf{P}_D) \tag{3.64}$$

$$\mathbf{B} = (\mathbf{P}^{\mathrm{T}}\widehat{\mathbf{W}}) = (\mathbf{P}_m^{\mathrm{T}}\widehat{\mathbf{W}}_0) + (\mathbf{P}_D^{\mathrm{T}}\widehat{\mathbf{W}}_D) \tag{3.65}$$

Solving Equation (3.61) for **a** yields

$$\mathbf{a} = (\mathbf{P}^{\mathrm{T}}\mathbf{W}\mathbf{P})^{-1}(\mathbf{P}^{\mathrm{T}}\mathbf{W})\mathbf{U}_s^D \tag{3.66}$$

or

$$\mathbf{a} = \mathbf{A}^{-1}\mathbf{B}\mathbf{U}_s^D \tag{3.67}$$

Substituting Equation (3.67) back into Equation (3.32), we have

$$u^h(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}\mathbf{a} = \mathbf{p}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{B}\mathbf{U}_s^D = \mathbf{\Phi}^{\mathrm{T}}\mathbf{U}_s^D \tag{3.68}$$

The vector of shape functions can be expressed as follows

$$\mathbf{\Phi}^{\mathrm{T}} = \mathbf{p}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{B} = \begin{Bmatrix} \phi_1 & \cdots & \phi_i & \cdots & \phi_n & \phi_1^H & \cdots & \phi_j^H & \cdots & \phi_{n_{DB}}^H \end{Bmatrix} \tag{3.69}$$

where ϕ_i (i=1,2, …, n) is the shape function corresponding to the ith node in the support domain, and ϕ_j^H (j=1,2, …, n_{DB}) is the shape function corresponding to the jth DB-node.

Similar to the WLS shape functions, these Hermite-type WLS shape functions do not have the Kronecker delta function property. Special treatments are needed to enforce the essential boundary conditions.

3.2.2 Radial point interpolation shape functions

3.2.2.1 Conventional RPIM

In order to avoid the singularity problem in the polynomial PIM, the radial basis function (RBF) is used to develop the radial point interpolation method (RPIM) shape functions for MFree weak-form methods (GR Liu and Gu, 2001c; Wang and Liu, 2000; 2002a,c). The RPIM shape functions will be used frequently in this book for both MFree weak-form and strong-form methods. The RPIM interpolation augmented with polynomials can be written as

$$u(\mathbf{x}) = \sum_{i=1}^{n} R_i(\mathbf{x}) a_i + \sum_{j=1}^{m} p_j(\mathbf{x}) b_j = \mathbf{R}^{\mathrm{T}}(\mathbf{x})\mathbf{a} + \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{b} \tag{3.70}$$

where $R_i(\mathbf{x})$ is a radial basis function (RBF), n is the number of RBFs, $p_j(\mathbf{x})$ is monomial in the space coordinates $\mathbf{x}^{\mathrm{T}}=[x, y]$, and m is the number of polynomial basis functions. When m=0, pure RBFs are used. Otherwise, the RBF is augmented with m polynomial basis functions. Coefficients a_i and b_j are constants yet to be determined.

In the radial basis function $R_i(\mathbf{x})$, the variable is only the distance between the point of interest $\mathbf{x}$ and a node at $\mathbf{x}_i$,

$$r = \sqrt{(x-x_i)^2 + (y-y_i)^2} \quad \text{for 2-D problems} \tag{3.71}$$

There are a number of types of radial basis functions (RBF), and the characteristics of RBFs have been widely investigated (Kansa,1990; Sharan et al.,1997; Franke and Schaback, 1997; etc.). Four often used RBFs, the multi-quadrics (MQ) function, the Gaussian (Exp) function, the thin plate spline (TPS) function, and the Logarithmic radial basis function, are listed in Table 3.2. In utilizing RBFs, several *shape parameters* need be determined for good performance. In general, these parameters can be determined by numerical examinations for given types of problems (see, e.g., Wang and GR Liu, 2000; 2002c). For example, in the MQ-RBF, there are two shape parameters: α_c and q, to be determined by the analyst. When $q = \pm 0.5$, it is the standard MQ-RBF. Wang and GR Liu (2001a, 2002c) left the parameter

q open to any real variable, and found that q=0.98 or 1.03 led to good results in the analysis of two-dimensional solid and fluid mechanics problems. This will be investigated further in Chapter 4 and Chapter 5.

Table 3.2 Typical radial basis functions with dimensionless shape parameters

	Name	†Expression	Shape Parameters
1	Multi-quadrics (MQ)	$R_i(x,y)=(r_i^2+(\alpha_c d_c)^2)^q$	$\alpha_c \geq 0$, q
2	Gaussian (EXP)	$R_i(x,y)=\exp[-\alpha_c(\frac{r_i}{d_c})^2]$	α_c
3	Thin Plate Spline (TPS)	$R_i(x,y)=r_i^\eta$	η
4	Logarithmic	$R_i(x,y)=r_i^\eta \log r_i$	η

† Note: d_c is a characteristic length that relates to the nodal spacing in the local support domain of the point of interest **x**, and it is usually the average nodal spacing for all the nodes in the local support domain as discussed in Section 3.1.

Table 3.3. Formulations of the compactly supported radial basis function (CSRBF)

CSRBF	Formulation	Ref.
Wu-C2	$R(x,y)=(1-\frac{r}{\delta})^5(8+40\frac{r}{\delta}+48\frac{r^2}{\delta^2}+25\frac{r^3}{\delta^3}+5\frac{r^4}{\delta^4})$	Wu(1995)
Wu-C4	$R(x,y)=(1-\frac{r}{\delta})^6(6+36\frac{r}{\delta}+82\frac{r^2}{\delta^2}+72\frac{r^3}{\delta^3}+30\frac{r^4}{\delta^4}+5\frac{r^5}{\delta^5})$	Wu(1995)
Wendland-C2	$R(x,y)=(1-\frac{r}{\delta})^4(1+4\frac{r}{\delta})$	Wendland (1995)
Wendland-C4	$R(x,y)=(1-\frac{r}{\delta})^6(3+18\frac{r}{\delta}+35\frac{r^2}{\delta^2})$	Wendland (1995)
Wendland-C6	$R(x,y)=(1-\frac{r}{\delta})^8(1+8\frac{r}{\delta}+25\frac{r^2}{\delta^2}+32\frac{r^3}{\delta^3})$	Wendland (1995)

δ: the size of the local support

In addition, the so-called compactly supported radial basis functions (CSRBFs) have also been developed (Wu, 1995; Wendland, 1995). Several CSRBFs are listed in Table 3.3. In contrast to the CSRBF, RBFs listed in Table 3.2 can be called the classical RBFs. These CSRBFs are strictly positive definite for all r less than or equal to some fixed value, and can be constructed to have desired amount of smoothness of $2k$. In a CSRBF, there is a shape parameter, δ, that determines the dimension of the local support for the CSRBF. When $r \geq \delta$, their values is regarded as zero. Studies by authors' group (GR Liu and Gu, et al., 2004) failed to find clear advantages of CSRBFs over the classic RBFs for their surface fitting and mechanics problems.

The polynomial term in Equation (3.70) is not always necessary. Studies have found the following conclusions.

1) The RPIM shape functions with pure RBFs usually cannot pass the standard patch tests, meaning that they fail to reconstruct exactly a linear polynomial field. Adding polynomial terms up to the linear order can ensure the C^1 consistency that is needed to pass the standard patch test.

2) In general, adding polynomials can always improve the accuracy of the results, at least no bad effect has been observed for MFree weak-form methods.

3) Adding polynomial reduces the sensitivity of the shape parameters, and will provide us much more freedom and a wider range in choosing shape parameters. This is true at least for MFree weak-form methods.

4) Adding polynomial can improve the interpolation stability for some RBFs. To ensure an invertible moment matrix of RBF, the polynomial augmented into RBF cannot be arbitrary (Schaback and Wendland, 2000). A low degree polynomial is often used to augment RBF to guarantee the non-singularity of the matrix (Cheng et al., 2003). For example, for an MQ-RBF, the linear polynomial can ensure an invertible moment matrix of RBF (Schaback and Wendland, 2000).

In order to determine a_i and b_j in Equation (3.70), a support domain is formed for the point of interest at $\mathbf{x}$, and n field nodes are included in the support domain. Coefficients a_i and b_j in Equation (3.70) can be determined by enforcing Equation (3.70) to be satisfied at these n nodes surrounding the point of interest $\mathbf{x}$. This leads to n linear equations, one for each node. The matrix form of these equations can be expressed as

$$\mathbf{U}_s = \mathbf{R}_0\mathbf{a} + \mathbf{P}_m\mathbf{b} \tag{3.72}$$

where the vector of function values $\mathbf{U}_s$ is

$$\mathbf{U}_s = \{u_1 \quad u_2 \quad \cdots \quad u_n\}^{\mathrm{T}} \tag{3.73}$$

the moment matrix of RBFs is

$$\mathbf{R}_0 = \begin{bmatrix} R_1(r_1) & R_2(r_1) & \cdots & R_n(r_1) \\ R_1(r_2) & R_2(r_2) & \cdots & R_n(r_2) \\ \cdots & \cdots & \cdots & \cdots \\ R_1(r_n) & R_2(r_n) & \cdots & R_n(r_n) \end{bmatrix}_{(n\times n)} \tag{3.74}$$

the polynomial moment matrix is

$$\mathbf{P}_m^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \\ y_1 & y_2 & \ldots & y_n \\ \vdots & \vdots & \ddots & \vdots \\ p_m(\mathbf{x}_1) & p_m(\mathbf{x}_2) & \ldots & p_m(\mathbf{x}_n) \end{bmatrix}_{(m\times n)} \tag{3.75}$$

the vector of coefficients for RBFs is

$$\mathbf{a}^{\mathrm{T}} = \{a_1 \quad a_2 \quad \ldots \quad a_n\} \tag{3.76}$$

the vector of coefficients for polynomial is

$$\mathbf{b}^{\mathrm{T}} = \{b_1 \quad b_2 \quad \ldots \quad b_m\} \tag{3.77}$$

In Equation (3.74), r_k in $R_i(r_k)$ is defined as

$$r_k = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{3.78}$$

However, there are $n+m$ variables in Equation (3.72). The additional m equations can be added using the following m constraint conditions.

$$\sum_{i=1}^{n} p_j(\mathbf{x}_i) a_i = \mathbf{P}_m^{\mathrm{T}} \mathbf{a} = 0, \qquad j=1, 2, \ldots, m \tag{3.79}$$

Combing Equations (3.72) and (3.79) yields the following set of equations in the matrix form

$$\tilde{\mathbf{U}}_s = \begin{bmatrix} \mathbf{U}_s \\ \mathbf{0} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{R}_0 & \mathbf{P}_m \\ \mathbf{P}_m^{\mathrm{T}} & \mathbf{0} \end{bmatrix}}_{\mathbf{G}} \begin{Bmatrix} \mathbf{a} \\ \mathbf{b} \end{Bmatrix} = \mathbf{G}\mathbf{a}_0 \tag{3.80}$$

where

$$\mathbf{a}_0^{\mathrm{T}} = \{a_1 \quad a_2 \quad \cdots \quad a_n \quad b_1 \quad b_2 \quad \cdots \quad b_m\} \tag{3.81}$$

$$\tilde{\mathbf{U}}_s = \{u_1 \quad u_2 \quad \cdots \quad u_n \quad 0 \quad 0 \quad \cdots \quad 0\} \tag{3.82}$$

Because the matrix $\mathbf{R}_0$ is symmetric, the matrix $\mathbf{G}$ will also be symmetric. Solving Equation (3.80), we obtain

$$\mathbf{a}_0 = \begin{Bmatrix} \mathbf{a} \\ \mathbf{b} \end{Bmatrix} = \mathbf{G}^{-1}\tilde{\mathbf{U}}_s \tag{3.83}$$

Equation (3.70) can be re-written as

$$u(\mathbf{x}) = \mathbf{R}^{\mathrm{T}}(\mathbf{x})\mathbf{a} + \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{b} = \{\mathbf{R}^{\mathrm{T}}(\mathbf{x}) \;\; \mathbf{p}^{\mathrm{T}}(\mathbf{x})\}\begin{Bmatrix} \mathbf{a} \\ \mathbf{b} \end{Bmatrix} \tag{3.84}$$

Using Equation (3.83) we can obtain

$$u(\mathbf{x}) = \{\mathbf{R}^{\mathrm{T}}(\mathbf{x}) \;\; \mathbf{p}^{\mathrm{T}}(\mathbf{x})\}\mathbf{G}^{-1}\tilde{\mathbf{U}}_s = \tilde{\mathbf{\Phi}}^{\mathrm{T}}(\mathbf{x})\tilde{\mathbf{U}}_s \tag{3.85}$$

where the RPIM shape functions can be expressed as

$$\begin{aligned} \tilde{\mathbf{\Phi}}^{\mathrm{T}}(\mathbf{x}) &= \{\mathbf{R}^{\mathrm{T}}(\mathbf{x}) \;\; \mathbf{p}^{\mathrm{T}}(\mathbf{x})\}\mathbf{G}^{-1} \\ &= \{\phi_1(\mathbf{x}) \quad \phi_2(\mathbf{x}) \quad \cdots \quad \phi_n(\mathbf{x}) \quad \phi_{n+1}(\mathbf{x}) \quad \cdots \quad \phi_{n+m}(\mathbf{x})\} \end{aligned} \tag{3.86}$$

Finally, the RPIM shape functions corresponding to the nodal displacements vector $\mathbf{\Phi}(\mathbf{x})$ are obtained as

$$\mathbf{\Phi}^{\mathrm{T}}(\mathbf{x}) = \{\phi_1(\mathbf{x}) \quad \phi_2(\mathbf{x}) \quad \cdots \quad \phi_n(\mathbf{x})\} \tag{3.87}$$

Equation (3.85) can be re-written as

$$u(\mathbf{x}) = \mathbf{\Phi}^{\mathrm{T}}(\mathbf{x})\mathbf{U}_s = \sum_{i=1}^{n} \phi_i u_i \tag{3.88}$$

The derivatives of $u(\mathbf{x})$ are easily obtained as

$$u_{,l}(\mathbf{x}) = \mathbf{\Phi}_{,l}^{\mathrm{T}}(\mathbf{x})\mathbf{U}_s \tag{3.89}$$

where l denotes either the coordinates x or y. A comma designates a partial differentiation with respect to the indicated spatial coordinate that follows.

Note that $\mathbf{R}_0^{-1}$ usually exists for arbitrarily scattered nodes (Powell, 1992; Schaback, 1994; Wendland, 1998). In addition, the order of polynomial used in Equation (3.70) is relatively low. Therefore, in general, there is no singularity problem in the RPIM as a small number of nodes (typically 10~40 for 2D problems) are used in the local support domain.

Note that the moment matrix $\mathbf{R}_0$ can be badly conditioned when the number of nodes increases. This is observed in MFree global collocation methods that use all the nodes in the entire problem domain in the formulation. One of the features of the global collocation methods is that a symmetric formulation is possible (Wu, 1992). This book, however, will not discuss these methods.

There are several advantages of using RBFs as a basis in constructing PIM shape functions that use local compact support domains.

- Using RBFs can effectively solve the singularity problem of the polynomial PIM.
- RPIM shape functions are stable[†] and hence flexible for arbitrary and irregular nodal distributions.
- RPIM shape functions can be easily created for three-dimensional domains, because the only variable is the distance r in a RBF. For three-dimensional interpolation, we simply change the distance expression to

$$r = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \tag{3.90}$$

- RPIM shape functions are better suited than MLS functions for fluid dynamics problems (see, Chapter 7).

However, RPIM also has some shortcomings, such as

- RPIM shape functions usually give less accurate solutions for solid problems compared to MLS and the polynomial PIM shape functions.
- Some shape parameters must be determined carefully, because they can affect the accuracy and the performance of the RPIM shape functions used in MFree methods.
- RPIM shape functions are usually computationally more expensive than the polynomial PIM because more nodes are required in the local support domain.

The properties of RPIM shape functions (GR Liu, 2002) are listed in this section.

1) Delta function property

RPIM shape functions have the Kronecker delta function property.

[†] Small changes in nodes locations or number of nodes in the support domain will not give rise to a big change in the RPIM shape functions created.

2) Partitions of unity

RPIM shape functions have the property of partitions of unity, i.e.,

$$\sum_{i=1}^{n} \phi_i = 1 \tag{3.91}$$

if the linear polynomial terms are added in the basis (m=3 in Equation (3.70)), and hence there is a constant term in the basis functions. The property of partitions of unity can be easily proven using the properties of reproduction of the RPIM shape functions.

If the pure RBF is used (m=0 in Equation, (3.70)), the property of partitions of unity can be easily proven for CSRBFs, as there are clearly constant terms in CSRBFs (see, Table 3.3). However, there is no constant term explicitly shown in some RBFs, such as the MQ-RBF. Additional treatment is needed for these RBFs to explicitly reveal the constant term.

An arbitrary function that has continous derivatives of all orders can be expressed by an infinite Taylor series expansion. For example, for the MQ-RBF, the Taylor series expansion in the vicinity of r=0 is

$$\begin{aligned} b(r) &= b(0) + b'(0)r + \frac{b''(0)}{2!}r^2 + \cdots \\ &= C^{2q} + b'(0)r + \frac{b''(0)}{2!}r^2 + \cdots \end{aligned} \tag{3.92}$$

We clearly see that there is a constant term in Equation (3.92) because $C \neq 0$ in MQ-RBF. The presence of this constant basis facilitates the reproduction of a constant field following the reproducibility property of the RPIM shape functions. Note that the condition for the reproduction of a constant field in the local domain is that all RBFs used in the RPIM have to be evaluated exactly, meaning that the expansion in Equation (3.92) needs to have infinite terms. This will be confirmed in the example presented in Sub-section 3.2.3.3. Therefore, Equation (3.91) may not be satisfied "exactly" but "approximately" in the numerical tests, because that there are always numerical truncation errors in the computation of a RBF caused by the use of finite terms in the Taylor series expansion.

Note here that TPS-RBF and Logarithmic-RBF do not satisfy the condition of $b(0) \neq 0$. Hence, the polynomial terms are needed in TPS-RBF and Logarithmic-RBF to ensure the property of partitions of unity for the RPIM shape functions.

3) Compact support

RPIM shape functions are compactly supported, as they are constructed using nodes in a compact support domain, and they are not used or are regarded as zero outside the support domain.

4) Continuity

The RPIM shape functions usually possess higher continuity because of the high continuity of the radial basis functions.

5) Reproducibility

The RPIM with at least polynomial terms can ensure an exact reproduction of linear polynomials.

Note that some RBFs will not have linear reproducibility meaning the RPIM cannot reproduce a linear field function without being augmented with linear polynomial terms. For example, in the case of the MQ-RBF, there exist no linear terms in its Taylor expansion form of Equation (3.92) due to $b'(0) = 0$. This could be one of the major reasons for the poor *h*-convergence in using MQ-RBFs. Hence, the linear polynomial terms are sometimes added in the RPIM to improve the performance in this regard.

6) Compatibility

In using RPIM shape functions, the compatibility in the global domain is not ensured when the local support domain is used, and the field function approximated could be discontinuous when nodes enter or leave the moving support domain.

3.2.2.2 Hermite-type RPIM

As shown in Figure 3.3, if there are DB-nodes within the support domain of a point of interest, their normal derivatives are considered as the additional unknowns. This implies that the DB-nodes not only have function values but also normal derivatives as variables. This is achieved by adopting the following Hermite-type interpolation using RBFs. The formulation procedure is similar to those given in Sub-section 3.2.2.1, except that it takes into consideration the additional normal derivative degrees of freedom (DOFs) for DB-nodes, which is again similar to the Hermite–type WLS discussed in section 3.2.1.3.

The approximation of a field function $u(\mathbf{x})$ can be written in a linear combination of RBFs at all the n nodes within the local support domain and the normal derivatives at the DB-nodes, i.e.,

$$u^h(\mathbf{x}) = \sum_{i=1}^{n} R_i(\mathbf{x}) a_i + \sum_{j=1}^{n_{\mathrm{DB}}} \frac{\partial R_j^{\mathrm{DB}}(\mathbf{x})}{\partial \mathbf{n}} b_j + \sum_{k=1}^{m} p_k(\mathbf{x}) c_k \tag{3.93}$$

where a_i, b_j and c_k are coefficients to be determined, n is the number of nodes in the local support domain (including the DB-nodes), n_{DB} is the number of the DB-nodes in the local support domain, m is the number of

polynomial terms for augmentation, p_k is a monomial, and $\mathbf{n}$ is the vector of the unit outward normal on the boundary at the DB-node.

In Equation (3.93), $R_i(\mathbf{x})$ and $R_j^{\mathrm{DB}}(\mathbf{x})$ are RBFs that have been discussed in Sub-section 3.2.2.1. We have

$$\begin{aligned} R_i(\mathbf{x}) &= R(\|\mathbf{x}-\mathbf{x}_i\|) \\ R_j^{\mathrm{DB}}(\mathbf{x}) &= R(\|\mathbf{x}-\mathbf{x}_j^{\mathrm{DB}}\|) \\ \frac{\partial R_j^{\mathrm{DB}}(\mathbf{x})}{\partial \mathbf{n}} &= l_{xj}\frac{\partial R_j^{\mathrm{DB}}(\mathbf{x})}{\partial x} + l_{yj}\frac{\partial R_j^{\mathrm{DB}}(\mathbf{x})}{\partial y} \end{aligned} \tag{3.94}$$

where $\mathbf{x}_i$ is the coordinate for the ith node in the local support domain, $\mathbf{x}_j^{\mathrm{DB}}$ is the coordinate for the jth DB-node, and $l_{xj}=\cos(\mathbf{n},x_j)$ and $l_{yj}=\cos(\mathbf{n},y_j)$ are direction cosines.

Note that because the derivatives of the field function at the DB-nodes are treated as unknowns, we use the derivatives of the same radial basis functions as the basis in Equation (3.93) for the DB-nodes. One may choose to use any other type of basis functions for these DB-nodes, as long as they are independent of the other basis used in Equation (3.93).

Equation (3.93) can be re-written in the following matrix form

$$u^h(\mathbf{x}) = \mathbf{B}^{\mathrm{T}}\mathbf{a}_0 \tag{3.95}$$

where the vector of basis function $\mathbf{B}$ has the form of

$$\mathbf{B}^{\mathrm{T}} = \left[R_1 \quad \cdots \quad R_n \quad \frac{\partial R_1^{\mathrm{DB}}}{\partial n} \quad \cdots \quad \frac{\partial R_{n_{\mathrm{DB}}}^{\mathrm{DB}}}{\partial n} \quad 1 \quad x \quad y \quad \cdots \quad p_m(\mathbf{x}) \right] \tag{3.96}$$

and the vector of coefficients $\mathbf{a}_0$ is given by

$$\mathbf{a}_0^{\mathrm{T}} = \left\{ a_1 \quad a_2 \quad \ldots \quad a_n \quad b_1 \quad \ldots \quad b_{n_{\mathrm{DB}}} \quad c_1 \quad \ldots \quad c_m \right\} \tag{3.97}$$

The coefficients a_i, b_j and c_k in Equation (3.93) are determined by making the interpolations pass through all n nodal function values within the support domain and equal the derivatives values of the function at the DB-nodes.

- For all the n nodes in the local support domain (including the DB-nodes), we have

$$u_l = u^h(\mathbf{x}_l) = \sum_{i=1}^{n} R_i(\mathbf{x}_l)a_i + \sum_{j=1}^{n_{\mathrm{DB}}} \frac{\partial R_j^{\mathrm{DB}}(\mathbf{x}_l)}{\partial \mathbf{n}} b_j + \sum_{k=1}^{m} p_k(\mathbf{x}_l)c_k \tag{3.98}$$

  where l=1, 2, ..., n.

- For all the DB-nodes, we have

$$\frac{\partial u_l^{\mathrm{DB}}}{\partial \mathbf{n}} = \sum_{i=1}^{n} \frac{\partial R_i(\mathbf{x}_l^{\mathrm{DB}})}{\partial \mathbf{n}} a_i + \sum_{j=1}^{n_{\mathrm{DB}}} \frac{\partial^2 R_j^{\mathrm{DB}}(\mathbf{x}_l^{\mathrm{DB}})}{\partial \mathbf{n}^2} b_j + \sum_{k=1}^{m} \frac{\partial p_k(\mathbf{x}_l^{\mathrm{DB}})}{\partial \mathbf{n}} c_k \tag{3.99}$$

  where l=1, 2, ..., n_{DB}.

- To obtain a unique solution, we impose the following constraints.

$$\sum_{i=1}^{n} p_k(\mathbf{x}_i) a_i + \sum_{j=1}^{n_{\mathrm{DB}}} p_k(\mathbf{x}_j) b_j = 0\,, \quad k=1, 2, ..., m \tag{3.100}$$

Arranging Equations (3.98)~(3.100) together leads to the following set of equations in matrix form.

$$\tilde{\mathbf{U}}_s = \begin{Bmatrix} \mathbf{U}_s \\ \dfrac{\partial \mathbf{U}_{\mathrm{DB}}}{\partial \mathbf{n}} \\ 0 \end{Bmatrix} = \underbrace{\begin{bmatrix} \mathbf{R}_0 & \mathbf{R}_{\mathrm{DB}} & \mathbf{P}_m \\ \mathbf{R}_{\mathrm{DB}}^{\mathrm{T}} & \mathbf{R}_c & \mathbf{P}_{\mathrm{DB}} \\ \mathbf{P}_m^{\mathrm{T}} & \mathbf{P}_{\mathrm{DB}}^{\mathrm{T}} & 0 \end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{Bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{Bmatrix}}_{\mathbf{a}_0} = \mathbf{G}\mathbf{a}_0 \tag{3.101}$$

where **G** is the generalized moment matrix that consists of:

the polynomial moment matrix evaluated at n nodes,

$$\mathbf{P}_m^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & ... & 1 \\ x_1 & x_2 & ... & x_n \\ y_1 & y_2 & ... & y_n \\ \vdots & \vdots & \ddots & \vdots \\ p_m(\mathbf{x}_1) & p_m(\mathbf{x}_2) & ... & p_m(\mathbf{x}_n) \end{bmatrix}_{(m\times n)} \tag{3.102}$$

and the moment matrix of the 1st derivatives of polynomials evaluated at n_{DB} DB-nodes,

$$\mathbf{P}_{\mathrm{DB}} = \begin{bmatrix} 0 & l_{\mathrm{DB}x1} & l_{\mathrm{DB}y1} & \cdots & \dfrac{\partial p_m(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}} \\ 0 & l_{\mathrm{DB}x2} & l_{\mathrm{DB}y2} & \cdots & \dfrac{\partial p_m(\mathbf{x}_2^{\mathrm{DB}})}{\partial \mathbf{n}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & l_{\mathrm{DB}xn} & l_{\mathrm{DB}yn} & \cdots & \dfrac{\partial p_m(\mathbf{x}_{n_{\mathrm{DB}}}^{\mathrm{DB}})}{\partial \mathbf{n}} \end{bmatrix}_{(n_{\mathrm{DB}}\times m)} \tag{3.103}$$

the moment matrix of RBFs evaluated at n nodes,

$$\mathbf{R}_0 = \begin{bmatrix} R_1(\mathbf{x}_1) & R_2(\mathbf{x}_1) & \cdots & R_n(\mathbf{x}_1) \\ R_1(\mathbf{x}_2) & R_2(\mathbf{x}_2) & \cdots & R_n(\mathbf{x}_2) \\ \cdots & \cdots & \cdots & \cdots \\ R_1(\mathbf{x}_n) & R_2(\mathbf{x}_n) & \cdots & R_n(\mathbf{x}_n) \end{bmatrix}_{(n\times n)} \tag{3.104}$$

the moment matrix of 1st normal derivatives of RBFs evaluated at the DB-nodes,

$$\mathbf{R}_{\mathrm{DB}}^{\mathrm{T}} = \begin{bmatrix} \dfrac{\partial R_1(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}} & \dfrac{\partial R_2(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}} & \cdots & \dfrac{\partial R_n(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}} \\ \dfrac{\partial R_1(\mathbf{x}_2^{\mathrm{DB}})}{\partial \mathbf{n}} & \dfrac{\partial R_2(\mathbf{x}_2^{\mathrm{DB}})}{\partial \mathbf{n}} & \cdots & \dfrac{\partial R_n(\mathbf{x}_2^{\mathrm{DB}})}{\partial \mathbf{n}} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial R_1(\mathbf{x}_{n_{DB}}^{\mathrm{DB}})}{\partial \mathbf{n}} & \dfrac{\partial R_2(\mathbf{x}_{n_{DB}}^{\mathrm{DB}})}{\partial \mathbf{n}} & \cdots & \dfrac{\partial R_n(\mathbf{x}_{n_{DB}}^{\mathrm{DB}})}{\partial \mathbf{n}} \end{bmatrix}_{(n_{\mathrm{DB}}\times n)} \tag{3.105}$$

and the moment matrix of 2nd normal derivatives of RBFs evaluated at DB-nodes,

$$\mathbf{R}_c = \begin{bmatrix} \dfrac{\partial}{\partial \mathbf{n}}\left(\dfrac{\partial R_1(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}}\right) & \dfrac{\partial}{\partial \mathbf{n}}\dfrac{\partial R_2(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}} & \cdots & \dfrac{\partial}{\partial \mathbf{n}}\left(\dfrac{\partial R_{n_{\mathrm{DB}}}(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}}\right) \\ \dfrac{\partial}{\partial \mathbf{n}}\left(\dfrac{\partial R_1(\mathbf{x}_2^{\mathrm{DB}})}{\partial \mathbf{n}}\right) & \dfrac{\partial}{\partial \mathbf{n}}\left(\dfrac{\partial R_2(\mathbf{x}_2^{\mathrm{DB}})}{\partial \mathbf{n}}\right) & \cdots & \dfrac{\partial}{\partial \mathbf{n}}\left(\dfrac{\partial R_{n_{\mathrm{DB}}}(\mathbf{x}_2^{\mathrm{DB}})}{\partial \mathbf{n}}\right) \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial}{\partial \mathbf{n}}\left(\dfrac{\partial R_1(\mathbf{x}_{n_{\mathrm{DB}}}^{\mathrm{DB}})}{\partial \mathbf{n}}\right) & \dfrac{\partial}{\partial \mathbf{n}}\left(\dfrac{\partial R_2(\mathbf{x}_{n_{\mathrm{DB}}}^{\mathrm{DB}})}{\partial \mathbf{n}}\right) & \cdots & \dfrac{\partial}{\partial \mathbf{n}}\left(\dfrac{\partial R_{n_{\mathrm{DB}}}(\mathbf{x}_{n_{\mathrm{DB}}}^{\mathrm{DB}})}{\partial \mathbf{n}}\right) \end{bmatrix}_{(n_{\mathrm{DB}}\times n_{\mathrm{DB}})} \tag{3.106}$$

In Equation (3.101), the extended vector of nodal variables is

$$\tilde{\mathbf{U}}_s^{\mathrm{T}} = \begin{bmatrix} u(\mathbf{x}_1) & \cdots & u(\mathbf{x}_n) & \dfrac{\partial u(\mathbf{x}_1^{\mathrm{DB}})}{\partial \mathbf{n}} & \cdots & \dfrac{\partial u(\mathbf{x}_{n_{\mathrm{DB}}}^{\mathrm{DB}})}{\partial \mathbf{n}} & 0 & \cdots & 0 \end{bmatrix} \tag{3.107}$$

the vector of nodal function variables is

$$\mathbf{U}_s = \{u_1 \quad u_2 \quad u_3 \quad \ldots \quad u_n\}^{\mathrm{T}} \tag{3.108}$$

and the vector of nodal normal derivative variables is

$$\frac{\partial \mathbf{U}_{\mathrm{DB}}}{\partial \mathbf{n}} = \left\{ \frac{\partial u_1^{\mathrm{DB}}}{\partial \mathbf{n}} \quad \frac{\partial u_2^{\mathrm{DB}}}{\partial \mathbf{n}} \quad \cdots \quad \frac{\partial u_{n_{DB}}^{\mathrm{DB}}}{\partial \mathbf{n}} \right\}^{\mathrm{T}} \tag{3.109}$$

It is clear that $\mathbf{G}$ in Equation (3.101) is symmetrical. For the same reasons mentioned in Sub-section 3.2.2.1, $\mathbf{G}$ is also, in general, invertible. Hence, we can solve Equation (3.101) for $\mathbf{a}_0$ to obtain

$$\mathbf{a}_0 = \mathbf{G}^{-1}\tilde{\mathbf{U}}_s \tag{3.110}$$

Substituting Equation (3.110) back to Equation (3.93) yields

$$\mathbf{u}^h(\mathbf{x}) = \mathbf{B}^{\mathrm{T}}\mathbf{a}_0 = \mathbf{B}^{\mathrm{T}}\mathbf{G}^{-1}\tilde{\mathbf{U}}_s = \mathbf{\Phi}^{\mathrm{T}}\tilde{\mathbf{U}}_s \tag{3.111}$$

where $\mathbf{\Phi}$ is a vector of the shape functions given by

$$\begin{aligned} \mathbf{\Phi}^{\mathrm{T}} &= \mathbf{B}^{\mathrm{T}}\mathbf{G}^{-1} \\ &= \left[\phi_1 \quad \cdots \quad \phi_i \quad \cdots \quad \phi_n \quad \phi_1^H \quad \cdots \quad \phi_{n_{\mathrm{DB}}}^H \quad \phi_1^p \quad \cdots \quad \phi_m^p \right]_{(n+n_{DB}+m)\times 1} \end{aligned} \tag{3.112}$$

Finally, the approximated function and its derivatives can be obtained using Equation (3.111).

$$\begin{aligned} u^h(\mathbf{x}) &= \sum_{i=1}^{n} \phi_i u_i + \sum_{j=1}^{n_{\mathrm{DB}}} \phi_j^H \frac{\partial u_j^{\mathrm{DB}}}{\partial \mathbf{n}} \\ \frac{\partial u^h(\mathbf{x})}{\partial x} &= \sum_{i=1}^{n} \frac{\partial \phi_i}{\partial x} u_i + \sum_{j=1}^{n_{\mathrm{DB}}} \frac{\partial \phi_j^H}{\partial x} \frac{\partial u_j^{\mathrm{DB}}}{\partial \mathbf{n}} \\ \frac{\partial u^h(\mathbf{x})}{\partial y} &= \sum_{i=1}^{n} \frac{\partial \phi_i}{\partial y} u_i + \sum_{j=1}^{n_{\mathrm{DB}}} \frac{\partial \phi_j^H}{\partial y} \frac{\partial u_j^{\mathrm{DB}}}{\partial \mathbf{n}} \\ &\cdots\cdots \end{aligned} \tag{3.113}$$

Because of the existence of $\mathbf{G}^{-1}$ for arbitrarily scattered nodes, there is no singularity problem in the process of computing the Hermite-type RPIM shape functions. In addition, the Hermite-type RPIM shape functions are stable and very flexible for arbitrary nodal distributions. They will be used in the MFree collocation methods discussed in Chapter 6.

3.2.3 Source code for the conventional RPIM shape functions

In this section, a standard subroutine, RPIM_ShapeFunc_2D.f90, for computing the conventional RPIM shape functions for two-dimensional problems is provided. This subroutine is written in FORTRAN 90.

Note that all programs provided in this book are developed and tested based on the MS Windows and MS Developer Studio 97 (Visual FORTRAN Professional Edition 5.0.A) in a personal computer. After slight revisions, these programs can also be executed in other platforms and systems, such as the UNIX system on workstations. In our research group these codes are frequently ported between Windows and UNIX systems, and there has been no technical problem.

3.2.3.1 Implementation issues

1) Determination of the support domain

For a two-dimensional domain, Ω, the support domain for a point of interest can be of various shapes. Circular and rectangular support domains are often used, and shown in Figure 3.1(a) and Figure 3.1(b), respectively. The rectangular support domain is simple to construct and easy to implement. Hence, in this section and following sections, the rectangular support domains are used.

Using the rectangular support domain, the dimension of the support domain is determined by d_{sx} and d_{sy} in x and y directions, respectively, i.e.,

$$\begin{cases} d_{sx} = \alpha_{sx} d_{cx} \\ d_{sy} = \alpha_{sy} d_{cy} \end{cases} \tag{3.114}$$

where α_{sx} and α_{sy} are the dimensionless sizes of the support domain in x and y directions. For simplicity, one often uses $\alpha_{sx}=\alpha_{sy}$, and d_{cx} and d_{cy} are the nodal spacings in x and y directions in the vicinity of the interpolation point at $\mathbf{x}_Q$. (see, Figure 3.1). If the nodes are uniformly distributed, d_{cx} is simply the distance in x direction between two neighboring nodes, and d_{cy} is simply the distance in y direction between two neighboring nodes. When the nodes are non-uniformly distributed, d_{cx} and d_{cy} can be defined as an average nodal spacing in the support domain of $\mathbf{x}_Q$ using the simple procedure discussed in Sub-section 3.1.3.

2) Shape parameters in radial basis functions

In the present subroutine of computing RPIM shape functions, the Multi-quadrics (MQ)-RBF, Gaussian (EXP)-RBF, and Thin Plate Spline (TPS)-

RBF, are coded. As shown in Table 3.2, the shape parameters in RBFs have to be pre-determined.

- For the MQ-RBF, there are two shape parameters: α_c and q. In the standard RBF, $q = \pm 0.5$ is often used. Wang and GR Liu (2001a, 2002c) have left q open to any real number and found q=0.98 or 1.03 good for solid and fluid mechanics. Both q and α_c are now dimensionless constants, and will be investigated later in this chapter for surface fitting and in Chapters 4, 5, 6 for mechanics problems. The nodal spacing d_c is calculated using

$$d_c = \sqrt{d_{cx}^2 + d_{cy}^2} \tag{3.115}$$

  where d_{cx} and d_{cy} are nodal spacings in the x and y directions efined in Equation (3.114).

- For the EXP-RBF, there is only one shape parameter, α_c; it is usually a positive number smaller than 1.0.

- For the TPS-RBF, the only shape parameter is η.

Shape parameters affect the performance of RBFs. Generally, there are no theoretically best values. They have been determined by intensive numerical investigations for classes of problems for weak-from formulations (GR Liu, 2002; Wang and GR Liu, 2002c). This issue will be further studied in Chapters 4 and 5 for MFree weak-form methods, in Chapter 6 for MFree strong-form methods, and in Chapter 7 for MFree weak-strong form methods.

3) Calculation of RPIM shape function

Equation (3.86) is used to compute RPIM shape functions. Direct inversion of $\mathbf{G}$ is avoided using a linear equation solver. Equation (3.86) can be re-written as

$$\begin{aligned} \tilde{\mathbf{\Phi}}^{\mathrm{T}}(\mathbf{x})\mathbf{G} &= \{\mathbf{R}^{\mathrm{T}}(\mathbf{x}), \mathbf{p}^{\mathrm{T}}(\mathbf{x})\}\mathbf{G}^{-1}\mathbf{G} \\ &= \{\mathbf{R}^{\mathrm{T}}(\mathbf{x}), \mathbf{p}^{\mathrm{T}}(\mathbf{x})\} \end{aligned} \tag{3.116}$$

Hence, we have

$$(\tilde{\mathbf{\Phi}}^{\mathrm{T}}(\mathbf{x})\mathbf{G})^{\mathrm{T}} = \{\mathbf{R}^{\mathrm{T}}(\mathbf{x}), \mathbf{p}^{\mathrm{T}}(\mathbf{x})\}^{\mathrm{T}} \tag{3.117}$$

or

$$\mathbf{G}^{\mathrm{T}}\tilde{\mathbf{\Phi}}(\mathbf{x}) = \begin{Bmatrix} \mathbf{R}(\mathbf{x}) \\ \mathbf{p}(\mathbf{x}) \end{Bmatrix} \tag{3.118}$$

Solving Equation (3.118) using a standard linear equation solver, we can obtain RPIM shape functions directly without computing $\mathbf{G}^{-1}$.

Derivatives of the RPIM shape functions can be obtained using Equation (3.118).

$$\frac{\partial}{\partial \mathbf{x}}\left(\mathbf{G}^{\mathrm{T}}\tilde{\mathbf{\Phi}}(\mathbf{x})\right)=\frac{\partial}{\partial \mathbf{x}}\begin{Bmatrix}\mathbf{R}(\mathbf{x})\\ \mathbf{p}(\mathbf{x})\end{Bmatrix}=\begin{Bmatrix}\dfrac{\partial \mathbf{R}(\mathbf{x})}{\partial \mathbf{x}}\\ \dfrac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}}\end{Bmatrix} \tag{3.119}$$

or

$$\mathbf{G}^{\mathrm{T}}\frac{\partial \tilde{\mathbf{\Phi}}(\mathbf{x})}{\partial \mathbf{x}}=\begin{Bmatrix}\dfrac{\partial \mathbf{R}(\mathbf{x})}{\partial \mathbf{x}}\\ \dfrac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}}\end{Bmatrix} \tag{3.120}$$

The 2nd derivative is

$$\mathbf{G}^{\mathrm{T}}\frac{\partial^2 \tilde{\mathbf{\Phi}}(\mathbf{x})}{\partial \mathbf{x}^2}=\begin{Bmatrix}\dfrac{\partial^2 \mathbf{R}(\mathbf{x})}{\partial \mathbf{x}^2}\\ \dfrac{\partial^2 \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}^2}\end{Bmatrix} \tag{3.121}$$

Therefore, the derivatives of the RPIM shape functions can also be obtained by solving Equations (3.120) and (3.121) using the standard linear equation solver.

4) Flowchart of the subroutine

The flowchart of the subroutine RPIM_ShapeFunc_2D.f90 is shown in Figure 3.4.

3.2.3.2 Program and data structure

The main subroutine RPIM_ShapeFunc_2D calls two subroutines that are shown in Figure 3.4 and Appendix 3.1.

1) Main Subroutine RPIM_ShapeFunc_2D

This subroutine is used to compute RPIM shape functions and their derivatives for a two-dimensional domain. In the current program, up to second order derivatives of shape functions are given. The user can modify this subroutine to compute higher-order derivatives of shape functions without too much difficulty. In addition, the polynomial terms added in the

radial basis are up to linear (*mbasis*=3). If *mbasis*=0 is used, the program produces pure RPIM shape functions without polynomial augmentation. The user can add more polynomial terms by changing the subroutine slightly.

The dummy arguments used in the subroutine RPIM_ShapeFunc_2D are listed in Appendix 3.2. The source code of the subroutine RPIM_ShapeFunc_2D is listed in Program 3.1.

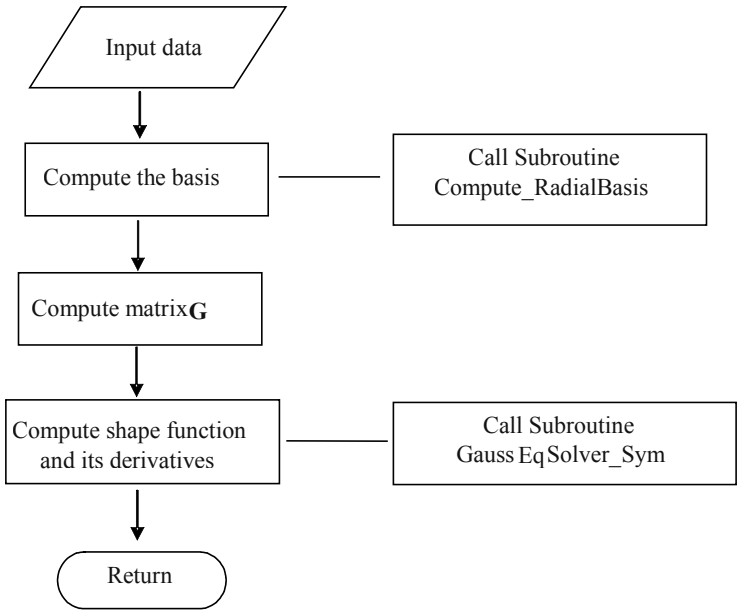


Figure 3.4. Flowchart of the program of RPIM_ShapeFunc_2D.f90 for computing RPIM shape functions.

2) Subroutine Compute_RadialBasis

Source code location: Program 3.2.
Dummy arguments: Appendix 3.3.
Function: to compute RBF $R(r)$ and its derivatives. In the current program, MQ-RBF, EXP-RBF and TPS-RBF are included. The user can change this subroutine slightly to include any other RBF (e.g., CSRBF).

3) Subroutine GaussEqSolver_Sym

Source code location: Program 3.3.
Dummy arguments: Appendix 3.4.
Function: it is a standard equation solver using the Gauss elimination method. To use this solver, the coefficient matrix must be symmetric.

3.2.3.3 Examples of RPIM shape functions

An example is presented to illustrate the properties of the conventional RPIM shape function and its derivative created using 25 nodes in the support domain. These 25 (5×5) nodes, as show in Figure 3.5, are regularly and evenly distributed in a rectangular domain: $x_i \in [-1,\ 1]$ and $y_i \in [-1,\ 1]$. The coordinates of these 25 nodes are listed in Table 3.4. The RPIM shape functions created can be evaluated at any point in the domain, and plotted in *x*-*y* space. In this study, a total of 61×61 points is used to evaluate and plot the shape functions.

A simple main program is listed in Program 3.4. In this program, the influence domain is used as an alternative to the support domain. The detailed discussions of comparisons between the support domain and the influence domain are presented in Chapter 4. The size of the influence domain for different field nodes is adjusted to ensure all 25 field nodes are included in the interpolation for each evaluation point.

1) The RPIM shape functions and their derivatives

Three typical radial basis functions, MQ-RBF, Exp-RBF and TPS-RBF, are used, and RPIM-MQ, RPIM-EXP and RPIM-TPS will be used in the following to denote RPIM shape functions using MQ, EXP and TPS radial basis functions, respectively.

Figure 3.6~Figure 3.8 show the RPIM-MQ shape functions and their derivatives. The shape parameters, a_c=2.0, d_c=0.5, and q=0.5 are used with *mbasis*=0 (no polynomial augmentation). Figure 3.9 shows the RPIM-EXP shape functions. The shape parameters are a_c=0.03, d_c=0.5, and *mbasis*=0. Figure 3.10 shows the RPIM-TPS shape functions. The shape parameters are η=4.001 and *mbasis*=0.

Appendix 3.5 lists a sample output of RPIM-MQ result from this program for the sampling point at $\mathbf{x}^T$=[0.2, 0.4]. From this appendix, we can observe that RPIM shape function satisfy the partitions of unity. As discussed in Sub-section 3.2.2.1, however, Equation (3.91) may not be satisfied exactly because there are always numerical truncation errors in the computation of complex RBFs. The summation of shape functions is not exact but approximate 1.0, as shown in Appendix 3.5.

Appendix 3.6 lists a sample output of RPIM-MQ for this same program for the sampling point at $\mathbf{x}^T$=[0, 0]. This appendix shows that the RPIM shape functions have the Kronecker delta function property. For example, in Appendix 3.6, at point $\mathbf{x}^T$=[0, 0] where node 13 is located, we have

$$\phi_i(x) = \begin{cases} 1 & i = 13 \quad (\mathbf{x} = \mathbf{x}_{13}) \\ 0 & i \neq 13 \quad (\mathbf{x} \neq \mathbf{x}_{13}) \end{cases} \tag{3.122}$$

This confirms numerically that that RPIM shape function possesses the Kronecker delta function property.

The distribution of ϕ and $\frac{\partial \phi}{\partial x}$ for the central node 13 along the line y=0 is plotted in Figure 3.11 and Figure 3.12.

Table 3.4. Coordinates of 25 field nodes shown in Figure 3.5

Node	x_I	y_I	Node	x_I	y_I
1	-1	1	**14**	0	-0.5
2	-1	0.5	**15**	0	-1
3	-1	0	**16**	0.5	1
4	-1	-0.5	**17**	0.5	0.5
5	-1	-1	**18**	0.5	0
6	-0.5	1	**19**	0.5	-0.5
7	-0.5	0.5	**20**	0.5	-1
8	-0.5	0	**21**	1	1
9	-0.5	-0.5	**22**	1	0.5
10	-0.5	-1	**23**	1	0
11	0	1	**24**	1	-0.5
12	0	0.5	**25**	1	-1
13	0	0			

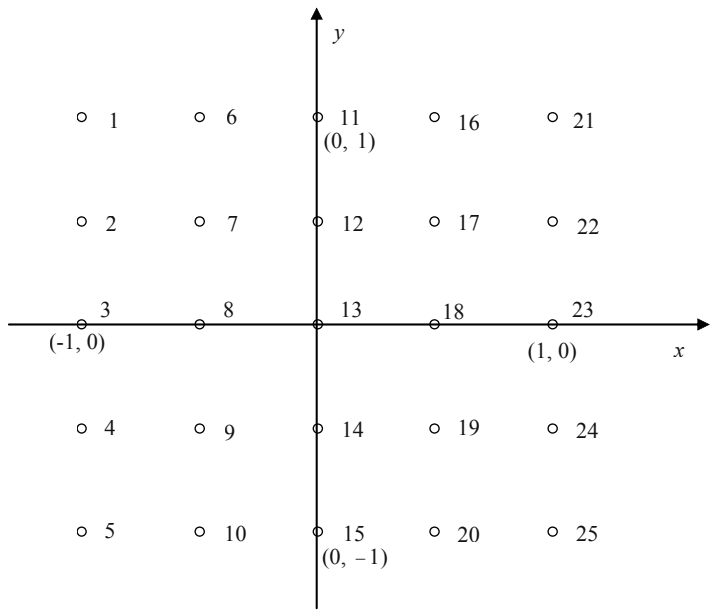


Figure 3.5. A total of 25 regularly distributed field nodes used to compute MFree shape functions.

2) The effect of shape parameters

The effects of shape parameters of RBFs are examined by plotting the shape function ϕ for the central node 13 along the line of y=0.

Figure 3.13 shows the RPIM-MQ shape functions for different parameters of $q=-0.5$, q=0.5 and q=1.03. It is found that there is a little difference in the shapes of the shape functions for different q values.

Figure 3.14 shows the RPIM-EXP shape functions for different parameters of α_c =0.03, α_c =0.1 and α_c =0.3. It is found that a small α_c leads to a large negative value for the shape functions.

Figure 3.15 shows the RPIM-TPS shape functions for different parameters of η =4.001, η =5.001 and η =6.001. It is found that there is a little difference in shape functions for different η values.

The RPIM shape functions with different terms polynomial augmentation of *mbasis*=0 and 3 are also obtained. It is found that the effect of *mbasis* on the shape of shape functions is insignificant. Therefore, figures of different *mbasis* are not plotted here.

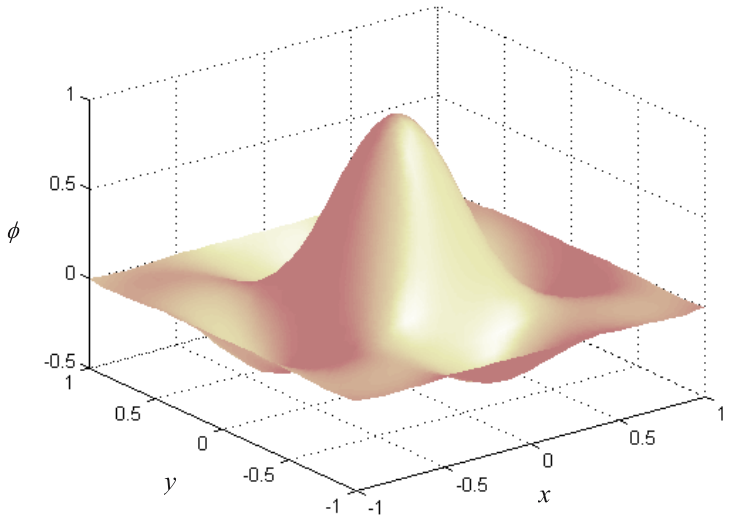


Figure 3.6. RPIM shape function for node 13 at point $\mathbf{x}^T$=[0, 0] obtained using 25 nodes shown in Figure 3.5 (MQ-RBF is used with shape parameters of $q = 0.5$, $\alpha_c = 2.0$, $d_c = 0.5$, and $mbasis = 0$.).

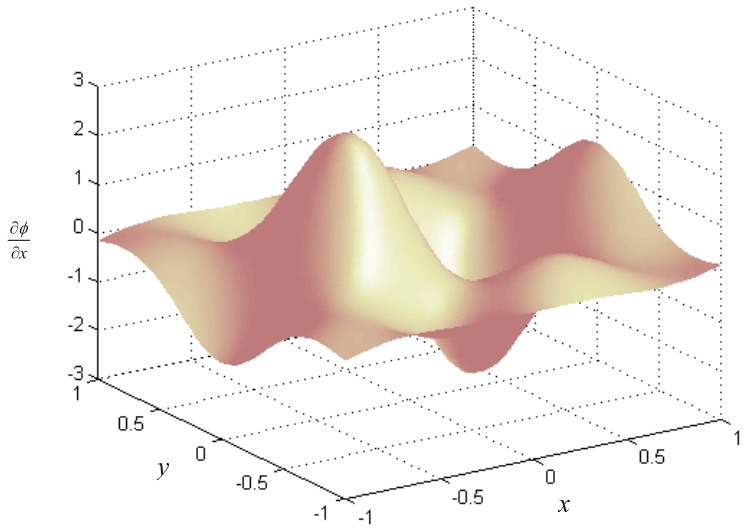


Figure 3.7. The first-order derivative of the RPIM shape function for node 13 at x^T=[0, 0] obtained using 25 nodes shown in Figure 3.5 in the support domain (MQ-RBF is used with shape parameters of $q = 0.5$, $\alpha_c = 2.0$, $d_c = 0.5$, and $mbasis = 0$.).

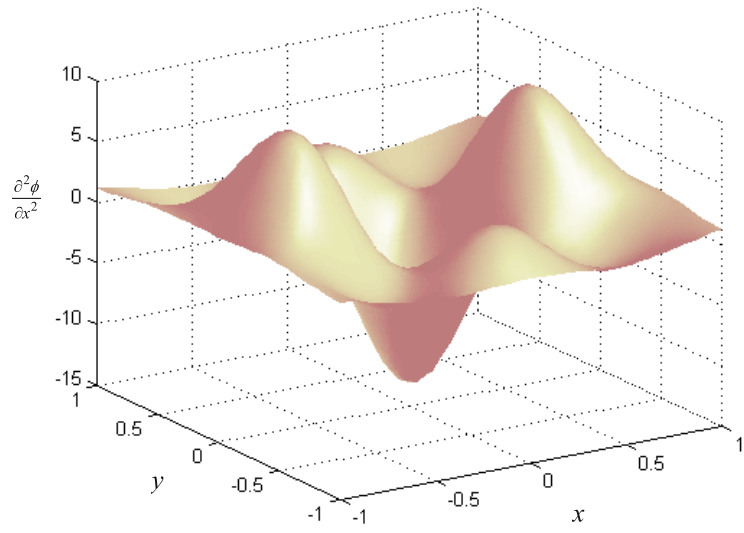


Figure 3.8. The second-order derivative of the RPIM shape function for node 13 at $\mathbf{x}^T$=[0, 0] obtained using 25 nodes shown in Figure 3.5 in the support domain (MQ-RBF is used with shape parameters of $q = 0.5$, $\alpha_c = 2.0$, $d_c = 0.5$, and $mbasis = 0$.).

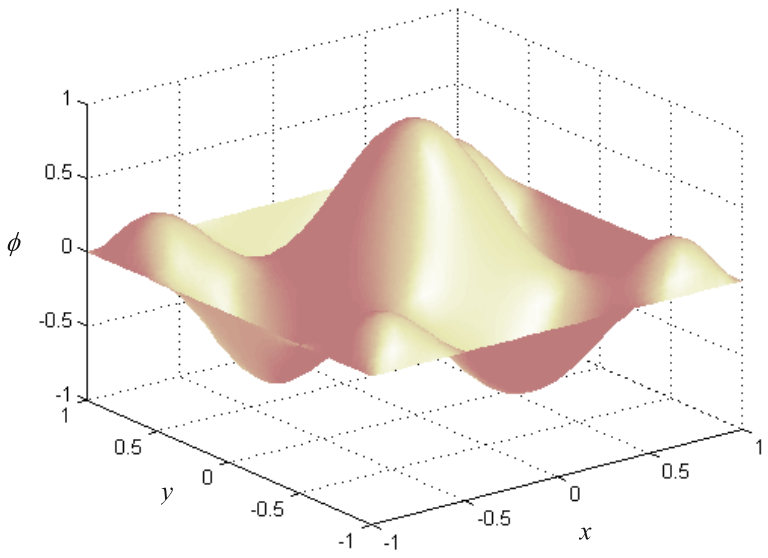


Figure 3.9. RPIM shape function for node 13 at $\mathbf{x}^T$=[0, 0] obtained using 25 nodes shown in Figure 3.5 (EXP-RBF is used with shape parameters of $\alpha_c = 0.03$, $d_c = 0.5$ and $mbasis = 0$.).

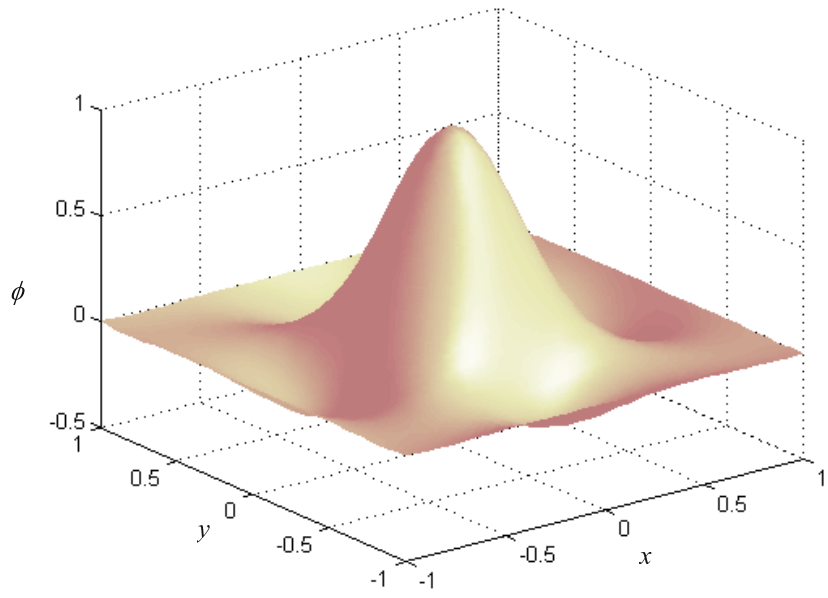


Figure 3.10. RPIM shape functions for $\mathbf{x}^T$=[0, 0] obtained using 25 nodes shown in Figure 3.5 (TPS-RBF is used with shape parameters of $\eta_t = 4.001$, and $mbasis = 0$).

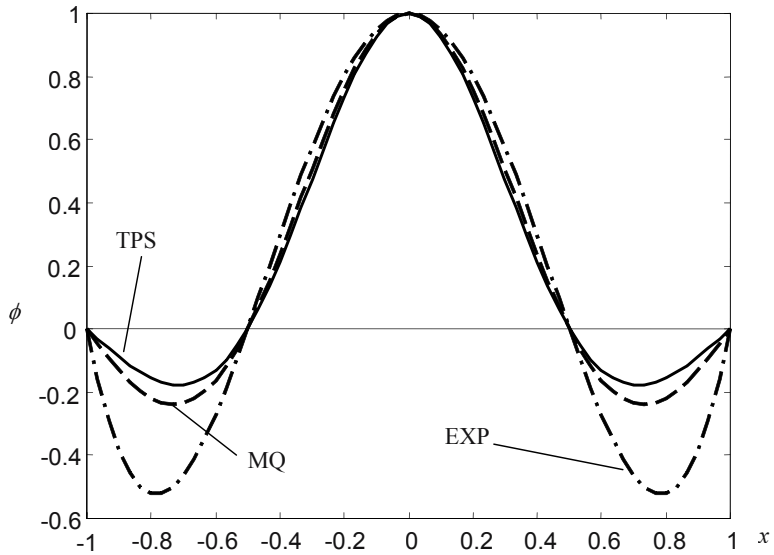
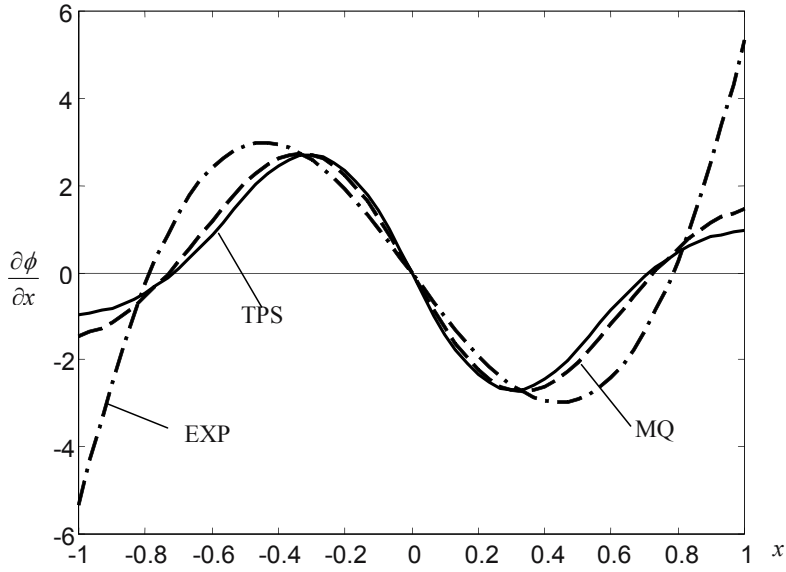


Figure 3.11. RPIM shape functions for the node 13 at $\mathbf{x}^T$=[0,0] along the line of y=0 obtained using different RBFs.



(b)

Figure 3.12. The 1st derivatives of RPIM shape functions for the node 13 at $\mathbf{x}^T$=[0,0] along the line of y=0 obtained using different RBFs.

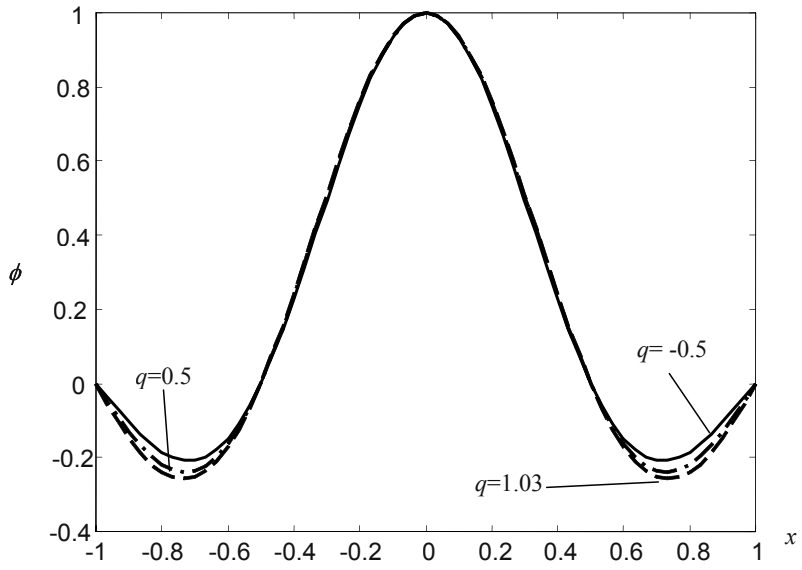


Figure 3.13. RPIM-MQ shape function for node 13 at $\mathbf{x}^{\mathrm{T}}$=[0,0] along the line of y=0 obtained using different q
(in MQ-RBF, shape parameters. $\alpha_c = 2.0$, $d_c = 0.5$, and $mbasis = 0$).

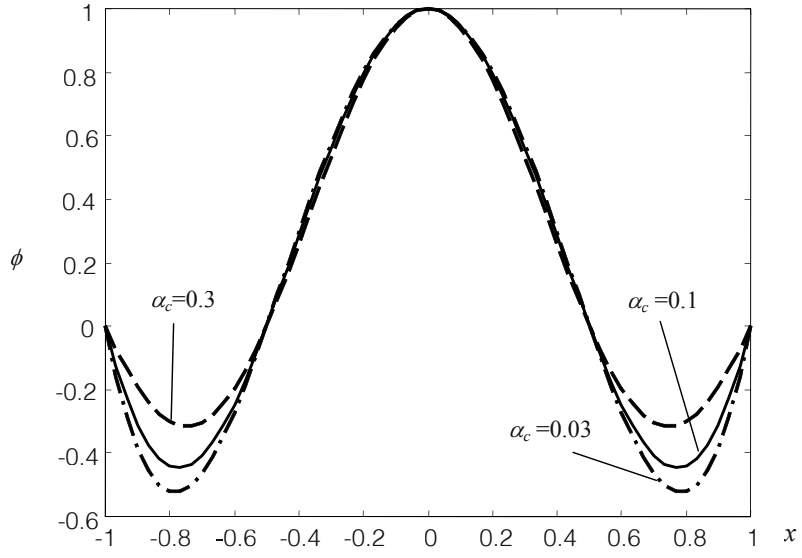


Figure 3.14. RPIM-EXP shape function for node 13 at $\mathbf{x}^{\mathrm{T}}$=[0,0] along the line of y=0 obtained using different α
(EXP-RBF with $d_c = 0.5$, $mbasis = 0$).

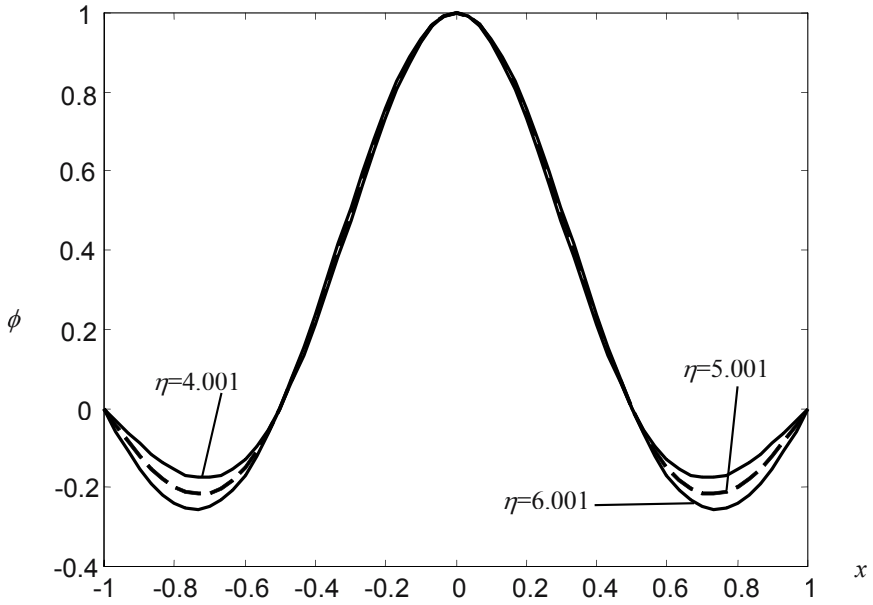


Figure 3.15. RPIM-TPS shape function for node 13 at $\mathbf{x}^T$=[0,0] along the line of y=0 obtained using different η (TPS-RBF with $mbasis = 0$).

3.3 MOVING LEAST SQUARES SHAPE FUNCTIONS

The moving least squares (MLS) approximation was devised by mathematicians in data fitting and surface construction (Lancaster and Salkausdas 1981; Cleveland 1993). It can be categorized as a method of series representation of functions. An excellent description of the MLS method can be found in a paper by Lancaster and Salkausdas (1981). The MLS approximation is now widely used in MFree methods for constructing MFree shape functions.

3.3.1 Formulation of MLS shape functions

Consider an unknown scalar function of a field variable $u(\mathbf{x})$ in the domain, Ω. The MLS approximation of $u(\mathbf{x})$ is defined at $\mathbf{x}$ as

$$u^h(\mathbf{x}) = \sum_{j=1}^{m} p_j(\mathbf{x}) a_j(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a}(\mathbf{x}) \tag{3.123}$$

where $\mathbf{p}(\mathbf{x})$ is the basis function of the spatial coordinates, $\mathbf{x}^{\mathrm{T}}=[x, y]$ for two-dimensional problem, and m is the number of the basis functions. The basis function $\mathbf{p}(\mathbf{x})$ is often built using monomials from the Pascal triangle to ensure minimum completeness. In some special problems, enhancement functions can, however, be added to the basis to improve the performance of the MLS approximation. We use only pure polynomial bases in this book.

In Equation (3.123), $\mathbf{a}(\mathbf{x})$ is a vector of coefficients given by

$$\mathbf{a}^{\mathrm{T}}(\mathbf{x}) = \{a_1(\mathbf{x}) \quad a_2(\mathbf{x}) \quad \cdots \quad a_m(\mathbf{x})\} \tag{3.124}$$

Note that the coefficient vector $\mathbf{a}(\mathbf{x})$ in Equation (3.123) is a function of $\mathbf{x}$. The coefficients $\mathbf{a}$ can be obtained by minimizing the following weighted discrete $\mathbf{L}_2$ norm.

$$J = \sum_{i=1}^{n} \widehat{W}(\mathbf{x} - \mathbf{x}_i)[\mathbf{p}^{\mathrm{T}}(\mathbf{x}_i)\mathbf{a}(\mathbf{x}) - u_i]^2 \tag{3.125}$$

where n is the number of nodes in the support domain of $\mathbf{x}$ for which the weight function $\widehat{W}(\mathbf{x} - \mathbf{x}_i) \neq 0$, and u_i is the *nodal parameter* of u at $\mathbf{x}=\mathbf{x}_i$. Equation (3.125) is a functional, a weighted residual, that is constructed using the approximated values and the nodal parameters of the unknown field function. Because the number of nodes, n, used in the MLS approximation is usually much larger than the number of unknown coefficients, m, the approximated function, u^h, does not pass through the nodal values, as shown in Figure 3.16.

The stationarity of J with respect to $\mathbf{a}(\mathbf{x})$ gives

$$\partial J / \partial \mathbf{a} = 0 \tag{3.126}$$

which leads to the following set of linear relations.

$$\mathbf{A}(\mathbf{x})\mathbf{a}(\mathbf{x})=\mathbf{B}(\mathbf{x})\mathbf{U}_s \tag{3.127}$$

where $\mathbf{U}_s$ is the vector that collects the nodal parameters of field function for all the nodes in the support domain.

$$\mathbf{U}_s = \{u_1 \quad u_2 \quad \ldots \quad u_n\}^{\mathrm{T}} \tag{3.128}$$

and $\mathbf{A}(\mathbf{x})$ is called the weighted *moment matrix* defined by

$$\mathbf{A}(\mathbf{x}) = \sum_{i=1}^{n} \widehat{W}_i(\mathbf{x})\mathbf{p}(\mathbf{x}_i)\mathbf{p}^{\mathrm{T}}(\mathbf{x}_i) \tag{3.129}$$

where

$$\widehat{W}_i(\mathbf{x}) = \widehat{W}(\mathbf{x}-\mathbf{x}_i) \tag{3.130}$$

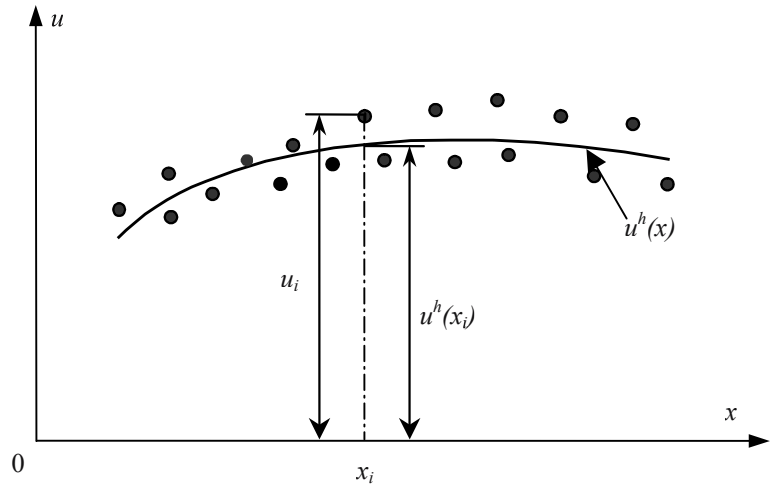


Figure 3.16. The approximate function $u^h(x)$ and the nodal parameters u_i in the MLS approximation.

For a two-dimensional problem and using the linear basis (m=3) defined in Equation (3.8), $\mathbf{A}$ is a symmetric 3×3 matrix that can be explicitly written as

$$\begin{aligned}
\mathbf{A}_{3\times3}(\mathbf{x}) &= \sum_{i=1}^{n} \widehat{W}_i(\mathbf{x})\mathbf{p}(\mathbf{x}_i)\mathbf{p}^{\mathrm{T}}(\mathbf{x}_i) \\
&= \widehat{W}(\mathbf{x}-\mathbf{x}_1)\begin{bmatrix} 1 & x_1 & y_1 \\ x_1 & x_1^2 & x_1y_1 \\ y_1 & x_1y_1 & y_1^2 \end{bmatrix} + \widehat{W}(\mathbf{x}-\mathbf{x}_2)\begin{bmatrix} 1 & x_2 & y_2 \\ x_2 & x_2^2 & x_2y_2 \\ y_2 & x_2y_2 & y_2^2 \end{bmatrix} + \\
&\quad \cdots + \widehat{W}(\mathbf{x}-\mathbf{x}_n)\begin{bmatrix} 1 & x_n & y_n \\ x_n & x_n^2 & x_ny_n \\ y_n & x_ny_n & y_n^2 \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^{n}\widehat{W}_i & \sum_{i=1}^{n}x_i\widehat{W}_i & \sum_{i=1}^{n}y_i\widehat{W}_i \\ \sum_{i=1}^{n}x_i\widehat{W}_i & \sum_{i=1}^{n}x_i^2\widehat{W}_i & \sum_{i=1}^{n}x_i\,y_i\widehat{W}_i \\ \sum_{i=1}^{n}y_i\widehat{W}_i & \sum_{i=1}^{n}x_i\,y_i\widehat{W}_i & \sum_{i=1}^{n}y_i^2\widehat{W}_i \end{bmatrix}_{(3\times3)}
\end{aligned} \tag{3.131}$$

The matrix $\mathbf{B}$ in Equation (3.127) is defined as

$$\mathbf{B}(\mathbf{x})=[\widehat{W}_1(\mathbf{x})\mathbf{p}(\mathbf{x}_1)\ \ \widehat{W}_2(\mathbf{x})\mathbf{p}(\mathbf{x}_2)\ \ldots\ \ \widehat{W}_n(\mathbf{x})\mathbf{p}(\mathbf{x}_n)] \tag{3.132}$$

which is a $3\times n$ matrix, and can be expressed explicitly as

$$\begin{aligned}\mathbf{B}_{3\times n}(\mathbf{x}) &= \left[\widehat{W}(\mathbf{x}-\mathbf{x}_1)\begin{bmatrix}1\\x_1\\y_1\end{bmatrix}\ \ \widehat{W}(\mathbf{x}-\mathbf{x}_2)\begin{bmatrix}1\\x_2\\y_2\end{bmatrix}\ \ \cdots\ \ \widehat{W}(\mathbf{x}-\mathbf{x}_n)\begin{bmatrix}1\\x_n\\y_n\end{bmatrix}\right]\\ &= \begin{bmatrix}\widehat{W}_1 & \widehat{W}_2 & \cdots & \widehat{W}_n\\ x_1\widehat{W}_1 & x_2\widehat{W}_2 & \cdots & x_n\widehat{W}_n\\ y_1\widehat{W}_1 & y_2\widehat{W}_2 & \cdots & y_n\widehat{W}_n\end{bmatrix}_{(3\times n)}\end{aligned} \tag{3.133}$$

Solving Equation (3.127) for $\mathbf{a}(\mathbf{x})$, we have

$$\mathbf{a}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x})\mathbf{U}_s \tag{3.134}$$

Substituting the above equation back into Equation (3.123), we obtain

$$u^h(\mathbf{x}) = \sum_{i=1}^{n}\phi_i(\mathbf{x})u_i = \mathbf{\Phi}^{\mathrm{T}}(\mathbf{x})\mathbf{U}_s \tag{3.135}$$

where $\mathbf{\Phi}(\mathbf{x})$ is the vector of MLS shape functions corresponding n nodes in the support domain of the point $\mathbf{x}$, and can be written as,

$$\mathbf{\Phi}^{\mathrm{T}}(\mathbf{x}) = \begin{Bmatrix}\phi_1(\mathbf{x}) & \phi_2(\mathbf{x}) & \cdots & \phi_n(\mathbf{x})\end{Bmatrix}_{(1\times n)} = \underbrace{\mathbf{p}^{\mathrm{T}}(\mathbf{x})}_{1\times 3}\underbrace{\mathbf{A}^{-1}(\mathbf{x})}_{3\times 3}\underbrace{\mathbf{B}(\mathbf{x})}_{3\times n} \tag{3.136}$$

The shape function $\phi_i(\mathbf{x})$ for the ith node is defined by

$$\phi_i(\mathbf{x}) = \sum_{j=1}^{m} p_j(\mathbf{x})(\mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x}))_{ji} = \mathbf{p}^{\mathrm{T}}(\mathbf{x})(\mathbf{A}^{-1}\mathbf{B})_i \tag{3.137}$$

Note that the WLS formulation mentioned in Sub-section 3.2.1.2 is very similar to the MLS formulation. In the MLS, the coefficient $\mathbf{a}$ is the function of $\mathbf{x}$ which makes the approximation of weighted least squares move continuously. Therefore, the MLS shape function will be continuous in the entire global domain, as long as the weight functions are chosen properly. This global continuity feature is preferred in the MFree global weak-form methods (Chapter 4). In WLS, however, because the coefficient $\mathbf{a}$ in Equation (3.32) is the constant, the WLS shape functions are piecewise continuous, as discussed in Section 3.2.1.2. The WLS approximation can be viewed as a special form of the MLS approximation.

For the convenience in obtaining the partial derivatives of the shape functions, Equation (3.136) is re-written as (Belytschko et al. 1996b)

$$\boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{x}) = \boldsymbol{\gamma}^{\mathrm{T}}(\mathbf{x})\mathbf{B}(\mathbf{x}) \tag{3.138}$$

where

$$\boldsymbol{\gamma}^{\mathrm{T}} = \mathbf{p}^{\mathrm{T}}\mathbf{A}^{-1} \tag{3.139}$$

Since $\mathbf{A}$ is symmetric, $\boldsymbol{\gamma}(\mathbf{x})$ can be obtained from Equation (3.139)

$$\mathbf{A}\boldsymbol{\gamma} = \mathbf{p} \tag{3.140}$$

The partial derivatives of $\boldsymbol{\gamma}$ can then be obtained by solving the following equations.

$$\mathbf{A}\boldsymbol{\gamma}_{,i} = \mathbf{p}_{,i} - \mathbf{A}_{,i}\boldsymbol{\gamma} \tag{3.141}$$

$$\mathbf{A}\boldsymbol{\gamma}_{,ij} = \mathbf{p}_{,ij} - (\mathbf{A}_{,i}\boldsymbol{\gamma}_{,j} + \mathbf{A}_{,j}\boldsymbol{\gamma}_{,i} + \mathbf{A}_{,ij}\boldsymbol{\gamma}) \tag{3.142}$$

$$\begin{aligned}\mathbf{A}\boldsymbol{\gamma}_{,ijk} = \mathbf{p}_{,ijk} - (&\mathbf{A}_{,i}\boldsymbol{\gamma}_{,jk} + \mathbf{A}_{,j}\boldsymbol{\gamma}_{,ik} + \mathbf{A}_{,k}\boldsymbol{\gamma}_{,ij} + \mathbf{A}_{,ij}\boldsymbol{\gamma}_{,k} \\ &+ \mathbf{A}_{,ik}\boldsymbol{\gamma}_{,j} + \mathbf{A}_{,jk}\boldsymbol{\gamma}_{,i} + \mathbf{A}_{,ijk}\boldsymbol{\gamma})\end{aligned} \tag{3.143}$$

where i, j and k denote coordinates x and y, and a comma designates a partial derivative with respect to the indicated spatial coordinate that follows. The partial derivatives of the shape function $\boldsymbol{\Phi}$ can be obtained using the following expressions.

$$\boldsymbol{\Phi}^{\mathrm{T}}_{,i} = \boldsymbol{\gamma}^{\mathrm{T}}_{,i}\mathbf{B} + \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{B}_{,i} \tag{3.144}$$

$$\boldsymbol{\Phi}^{\mathrm{T}}_{,ij} = \boldsymbol{\gamma}^{\mathrm{T}}_{,ij}\mathbf{B} + \boldsymbol{\gamma}^{\mathrm{T}}_{,i}\mathbf{B}_{,j} + \boldsymbol{\gamma}^{\mathrm{T}}_{,j}\mathbf{B}_{,i} + \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{B}_{,ij} \tag{3.145}$$

$$\begin{aligned}\boldsymbol{\Phi}^{\mathrm{T}}_{,ijk} = &\boldsymbol{\gamma}^{\mathbf{T}}_{,ijk}\mathbf{B} + \boldsymbol{\gamma}^{\mathrm{T}}_{,ij}\mathbf{B}_{,k} + \boldsymbol{\gamma}^{\mathrm{T}}_{,ik}\mathbf{B}_{,j} + \boldsymbol{\gamma}^{\mathrm{T}}_{,jk}\mathbf{B}_{,i} \\ &+\boldsymbol{\gamma}^{\mathrm{T}}_{,i}\mathbf{B}_{,jk} + \boldsymbol{\gamma}^{\mathrm{T}}_{,j}\mathbf{B}_{,ik} + \boldsymbol{\gamma}^{\mathrm{T}}_{,k}\mathbf{B}_{,ij} + \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{B}_{,ijk}\end{aligned} \tag{3.146}$$

In the MLS approximation, a support domain, defined in Equation (3.1), can be formed for any point of interest. Field nodes included in this support domain are used to perform the MLS approximation for the unknown function at this point. The number of nodes, n, chosen in the support domain, should be sufficient to ensure that the matrix $\mathbf{A}$ in Equation (3.134) is invertible, so as to provide the interpolation stability (Condition 2 in Sub-section 3.1.1). The choice of n depends on the nodal distribution and the number of basis functions, m. In order to ensure the existence of $\mathbf{A}^{-1}$ and a well-conditioned $\mathbf{A}$, we usually let $n >> m$. Unfortunately, there is no theoretical best value of n, and it has to be determined by numerical experiments.

3.3.2 Choice of the weight function

Equation (3.137) shows that the continuity of the MLS shape function Φ is governed by the continuity of the basis function $\mathbf{p}$ as well as the smoothness of the matrices $\mathbf{A}$ and $\mathbf{B}$. The latter is governed by the smoothness of the weight function. Therefore, the weight function plays an important role in the performance of the MLS approximation. In the reported studies so far, $\widehat{W}(\mathbf{x}-\mathbf{x}_i)$ is always chosen to have the following properties.

- $\widehat{W}(\mathbf{x}-\mathbf{x}_i)>0$ within the support domain
- $\widehat{W}(\mathbf{x}-\mathbf{x}_i)=0$ outside the support domain
- $\widehat{W}(\mathbf{x}-\mathbf{x}_i)$ monotonically decreases from the point of interest at $\mathbf{x}$
- $\widehat{W}(\mathbf{x}-\mathbf{x}_i)$ is sufficient smooth, especially on the boundary of Ω_s

(3.147)

The last condition in Equation (3.147) is to ensure a smooth inclusion and exclusion of nodes when the support domain moves, so as to guarantee the compatibility of the MLS shape function in the entire problem domain.

The choice of the weight function is more or less arbitrary as long as the requirements in Equation (3.147) are met. The exponential function and spline functions are often used in practice. Among them, the most commonly used weight functions are listed below.

- The cubic spline function (W1) has the following form of

$$\widehat{W}_i(\mathbf{x})=\begin{cases}2/3-4\overline{r}_i^2+4\overline{r}_i^3 & \overline{r}_i\le 0.5\\ 4/3-4\overline{r}_i+4\overline{r}_i^2-4/3\overline{r}_i^3 & 0.5<\overline{r}_i\le 1\\ 0 & \overline{r}_i>1\end{cases} \tag{3.148}$$

  which has 2nd order continuity (see, e.g., GR Liu and Liu, 2003).

- The quartic spline function (W2) is given by

$$\widehat{W}_i(\mathbf{x})=\begin{cases}1-6\overline{r}_i^2+8\overline{r}_i^3-3\overline{r}_i^4 & \overline{r}_i\le 1\\ 0 & \overline{r}_i>1\end{cases} \tag{3.149}$$

  which has 3rd order continuity (see, e.g., GR Liu and Liu, 2003).

- The exponential function (W3) is expressed as

$$\widehat{W}_i(\mathbf{x}) = \begin{cases} e^{-(\overline{r}_i/\alpha)^2} & \overline{r}_i \le 1 \\ 0 & \overline{r}_i > 1 \end{cases} \tag{3.150}$$

Where α is a constant of shape parameter, and

$$\overline{r}_i = \frac{d_i}{r_w} = \frac{|\mathbf{x} - \mathbf{x}_i|}{r_w} \tag{3.151}$$

in which $d_i = |\mathbf{x} - \mathbf{x}_i|$ is the distance from node $\mathbf{x}_i$ to the sampling point $\mathbf{x}$, and r_w is the size of the support domain for the weight function.

As the derivatives of all orders of W3 are continuous within the support domain, it is continuous at all orders within the interior of the support domain. However, all the derivatives even the functions itself are not exactly zero on the boundary of the support domain. Therefore, theoretically, W3 cannot provide compatibility of any order. Fortunately, these non-zero values of the function and its derivatives are very small on the boundary of the support domain. In practical numerical analyses, W3 provides very high order compatibility with a very small numerical error, provided the support domain is sufficiently large.

Figure 3.17 plots all these three weight functions and their first derivatives.

Note that it is easy to construct a weight function with a desired order of continuity using the following common formulation (see, e.g., GR Liu and Liu, 2003) of spline weight function.

$$\widehat{W}_i(\mathbf{x}) = \begin{cases} \sum_{j=0}^{l} b_j \overline{r}_i^{\,j} & 0 \le \overline{r}_i \le 1 \\ 0 & \overline{r}_i \ge 1 \end{cases} \tag{3.152}$$

where l is the order of the spline function, and b_j are the coefficients that can be determined by the required conditions.

For example, a 4th order spline function can be written in the general form of

$$\widehat{W}_i(\mathbf{x}) = \begin{cases} b_0 + b_1\overline{r}_i + b_2\overline{r}_i^{\,2} + b_3\overline{r}_i^{\,3} + b_4\overline{r}_i^{\,4} & 0 \le \overline{r}_i \le 1 \\ 0 & \overline{r}_i \ge 1 \end{cases} \tag{3.153}$$

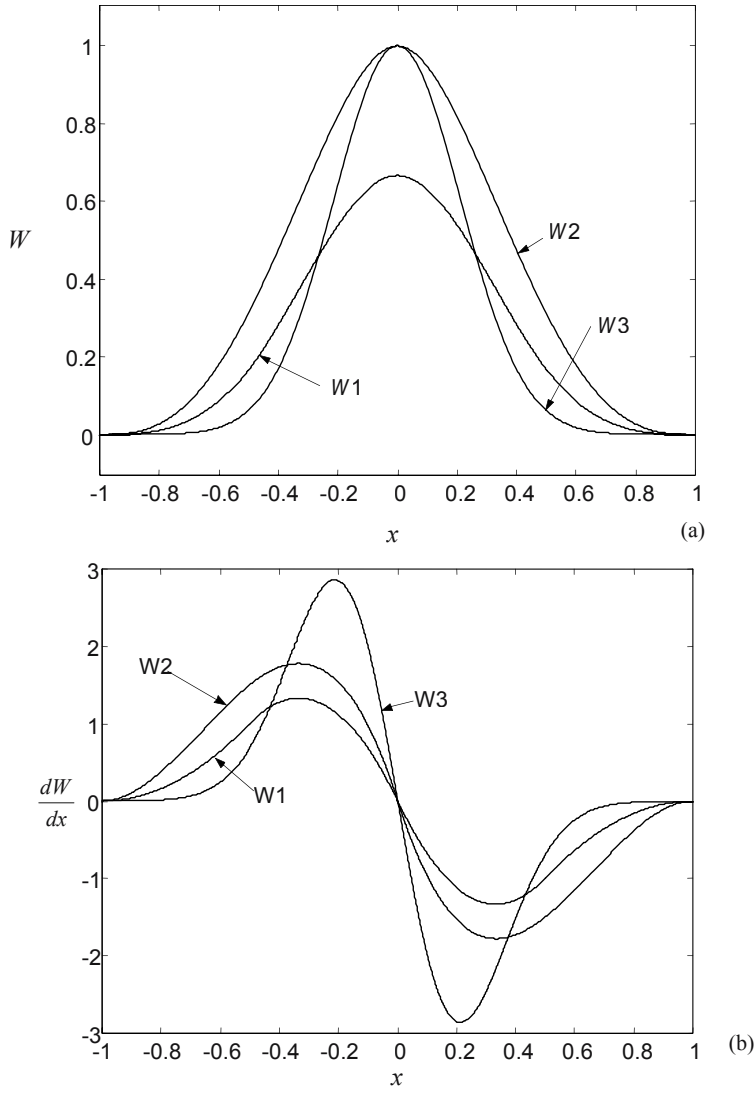


Figure 3.17. Weight functions and their first order derivatives. W1: cubic spline; W2: quartic spline; W3: exponential function (α=0.3).
(a) weight functions; (b) the first order derivatives.

We now require the weight function to satisfy the following conditions.

- Unity condition states that the weight function is one at the centre of the support domain where $\overline{r}_i = 0$:

$$\widehat{W}_i\Big|_{\overline{r}_i=0}=1 \tag{3.154}$$

- Compact support condition states that 1*st* and 2*nd* derivatives of the weight function are all zero at the boundary of the support domain where $\overline{r}_i=1$. This compact support condition leads to the following set of equations.

$$\begin{cases} \widehat{W}_i\Big|_{\overline{r}_i=1}=0 \\ \dfrac{\partial\widehat{W}_i}{\partial\overline{r}}\Big|_{\overline{r}_i=1}=0 \\ \dfrac{\partial^2\widehat{W}_i}{\partial\overline{r}^2}\Big|_{\overline{r}_i=1}=0 \end{cases} \tag{3.155}$$

- The condition of symmetry states that the 1*st* derivative of the weight function is zero at the centre of the support domain where $\overline{r}_i=0$. The condition of symmetry gives the following equation.

$$\frac{\partial\widehat{W}_i}{\partial\overline{r}}\Big|_{\overline{r}_i=0}=0 \tag{3.156}$$

Using Equations (3.154), (3.155) and (3.156), we can obtain the following set of equations.

$$\begin{cases} \widehat{W}_i\Big|_{\overline{r}_i=0}=1 \\ \dfrac{\partial\widehat{W}_i}{\partial\overline{r}}\Big|_{\overline{r}_i=0}=0 \\ \widehat{W}_i\Big|_{\overline{r}_i=1}=0 \\ \dfrac{\partial\widehat{W}_i}{\partial\overline{r}}\Big|_{\overline{r}_i=1}=0 \\ \dfrac{\partial^2\widehat{W}_i}{\partial\overline{r}^2}\Big|_{\overline{r}_i=1}=0 \end{cases} \quad \text{or} \quad \begin{cases} b_0=1 \\ b_1=0 \\ b_0+b_1+b_2+b_3+b_4=0 \\ b_1+2b_2+3b_3+4b_4=0 \\ 2b_2+6b_3+12b_4=0 \end{cases} \tag{3.157}$$

Solving the above set of equations for b_i yields

$$\begin{cases} b_0 = 1 \\ b_1 = 0 \\ b_2 = -6 \\ b_3 = 8 \\ b_4 = -3 \end{cases} \tag{3.158}$$

Substituting these coefficients back into Equation (3.153), we obtain the following weight function.

$$\widehat{W}_i(\mathbf{x}) = \begin{cases} 1 - 6\overline{r}_i^2 + 8\overline{r}_i^3 - 3\overline{r}_i^4 & 0 \le \overline{r}_i \le 1 \\ 0 & \overline{r}_i \ge 1 \end{cases} \tag{3.159}$$

This is the quartic spline weight function (W2) given by Equation (3.149).

Similarly, any other spline weight function (with required order of continuity and shape profile) can be constructed. Atluri et al. (1999b) also mentioned a similar method for constructing the weight function. More details on systematic ways to construct weight (smoothed) functions can be found in GR Liu and Liu (2003) including the construction of piecewise weight functions.

3.3.3 Properties of MLS shape functions

1) Consistency

By the definition, the consistency of the MFree shape functions is the ability of the shape functions to reproduce the complete order of polynomial. The consistency of the MLS approximation depends on the complete order of the monomial employed in the polynomial basis. If the complete order of monomial is k, the shape function will possess C^k consistency. This can be easily demonstrated (Krongauz and Belytschko, 1996; GR Liu, 2002) as follows.

Consider a field given by

$$u(\mathbf{x}) = \sum_{j=1}^{k} p_j(\mathbf{x})\,\alpha_j(\mathbf{x}), \qquad k \le m \tag{3.160}$$

Such a given field can always be written in the form of

$$u(\mathbf{x}) = \sum_{j=1}^{k} p_j(\mathbf{x})\alpha_j(\mathbf{x}) + \sum_{l=k+1}^{m} p_l(\mathbf{x}) \cdot 0 \tag{3.161}$$

If we let $a_i(\mathbf{x}) = \alpha_j(\mathbf{x})$, j=1,2, …, k, J in Equation (3.125) will vanish and it will necessarily be a minimum, which leads to

$$u^h(\mathbf{x}) = \sum_j^k p_j(\mathbf{x})\alpha_j(\mathbf{x}) = u(\mathbf{x}) \tag{3.162}$$

This proves that any monomial included in the basis of MLS will be exactly reproduced by the MLS approximation.

2) Reproduction

In the MFree method, the concept of reproduction is separated from that of the consistency (GR Liu, 2002). This is because different types of basis functions can be used in constructing MFree shape functions. Reproduction is the ability of the shape function to reproduce functions that are in the basis function used to construct the shape functions. The function may not be a polynomial, such as the radial basis function (RBF) in the radial point interpolation method (RPIM). However, consistency emphasizes the reproducibility of complete order of polynomials. This is the main difference between consistency and reproduction.

Similar to the demonstration of consistency, it can be proven that the MLS approximation can reproduce exactly any function that appears in the basis. This property will be very useful in the practical application. For example, we know that there is a singular stress field near the tip of a crack. If only the normal polynomial basis is used, the computational error will be certainly very large. If we can enrich the basis by including a singular functions into the basis, the reproduction property of MLS will ensure the reproduction of the singular field. As results, the solution accuracy can be significantly improved without too much additional cost (see, e.g., Belytschko et al., 1995a,b). Of course, one has to ensure that the weighted moment matrix computed using Equation (3.129) is still invertible and well-conditioned when these enriched basis functions are included, which can otherwise be a problem sometimes.

3) Partitions of unity

If the constant is included in the basis, the MLS shape function $\phi_i(\mathbf{x})$ is of the partition of unity, i.e.,

$$\sum_{i=1}^{n} \phi_i(\mathbf{x}) = 1 \tag{3.163}$$

This can be proven easily from the reproducibility feature of the MLS approximation. Detailed discussions can be found in Sub-section 3.2.1.1.

4) Lack of Kronecker delta function property

The MLS approximation is obtained by a special least squares method. As shown in Figure 3.16, the function obtained by the MLS approximation is

a smooth curve (or surface) and it does not pass through the nodal values. Therefore, the MLS shape functions given in Equation (3.137) do not, in general, satisfy the Kronecker delta condition. Thus,

$$\phi_i(\mathbf{x}_j) \neq \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{3.164}$$

This property will be demonstrated later in the numerical examples.

3.3.4 Source code for the MLS shape function

3.3.4.1 Implementation issues

1) Determination of the support domain

As for the RPIM subroutine discussed in Sub-section 3.2.3, the rectangular support domains are used. The dimension of the support domain is determined by d_{sx} and d_{sy} in the x and y directions, respectively, which are given by Equation (3.114).

2) Determination of weight functions

As discussed above, the weight function plays an important role in the performance of the MLS approximation. All weight functions discussed in Section 3.3.2 can be used. In the program given here, the weight functions of the cubic spline function (W1, Equation (3.148)) and the quartic spline function (W2, Equation (3.149)), are included. Because the rectangular support domains are used, the weight functions need to be slightly modified. We define now

$$\widehat{W}_i(\mathbf{x}) = \widehat{W}_{ix}(\mathbf{x}) \cdot \widehat{W}_{iy}(\mathbf{x}) = \widehat{W}(r_{ix}) \cdot \widehat{W}(r_{iy}) \tag{3.165}$$

where $\widehat{W}_{ix}$ and $\widehat{W}_{iy}$ are any of the standard 1D weight functions in x and y directions, respectively, given in Sub-section 3.3.2 with

$$r_{ix} = \frac{\|x - x_i\|}{d_{sx}} \tag{3.166}$$

$$r_{iy} = \frac{\|y - y_i\|}{d_{sy}} \tag{3.167}$$

When cubic spline weight function (W1) is used, we have

$$\widehat{W}(r_{ix}) = \begin{cases} 2/3 - 4r_{ix}^2 + 4r_{ix}^3 & r_{ix} \le 0.5 \\ 4/3 - 4r_{ix} + 4r_{ix}^2 - 4/3r_{ix}^3 & 0.5 < r_{ix} \le 1 \\ 0 & r_{ix} > 1 \end{cases} \tag{3.168}$$

$$\widehat{W}(r_{iy}) = \begin{cases} 2/3 - 4r_{iy}^2 + 4r_{iy}^3 & r_{iy} \le 0.5 \\ 4/3 - 4r_{iy} + 4r_{iy}^2 - 4/3r_{iy}^3 & 0.5 < r_{iy} \le 1 \\ 0 & r_{iy} > 1 \end{cases} \tag{3.169}$$

When the quartic spline weight function (W2) is used, we have

$$\widehat{W}(r_{ix}) = \begin{cases} 1 - 6r_{ix}^2 + 8r_{ix}^3 - 3r_{ix}^4 & 0 \le \overline{r}_{ix} \le 1 \\ 0 & \overline{r}_{ix} > 1 \end{cases} \tag{3.170}$$

$$\widehat{W}(r_{iy}) = \begin{cases} 1 - 6r_{iy}^2 + 8r_{iy}^3 - 3r_{iy}^4 & 0 \le \overline{r}_{iy} \le 1 \\ 0 & \overline{r}_{iy} > 1 \end{cases} \tag{3.171}$$

3) Calculation of MLS shape function

The MLS shape function is given by Equation (3.137). If we use this equation to calculate the MLS shape function, $\mathbf{A}^{-1}$ has to be computed, which is not efficient. In addition, the computation of derivatives of the MLS shape function is quite complex. Hence, the recurrence formulation presented in Equations (3.140)~(3.146) is often preferred and used in the program.

First, Equation (3.140) is solved by a standard linear equation solver to obtain $\boldsymbol{\gamma}$.

Second, Equations (3.141)~(3.143) are solved to obtain derivatives of $\boldsymbol{\gamma}$.

Third, Equations (3.138) and (3.144) ~ (3.146) are used to calculate the MLS shape function and its derivatives.

This procedure avoids direct inversion $\mathbf{A}^{-1}$; it is efficient to obtain arbitrary order derivatives of the MLS shape function.

4) Flowchart of the subroutine

The flowchart of the subroutine of the MLS approximation is shown in Figure 3.18.

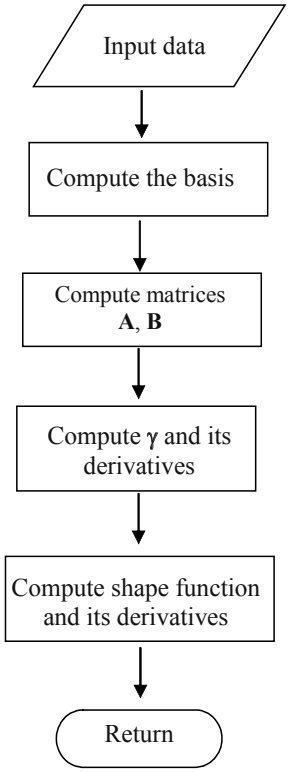


Figure 3.18. Flowchart of the program of MLS_ShapeFunc_2D.f90 for computing the MLS shape functions.

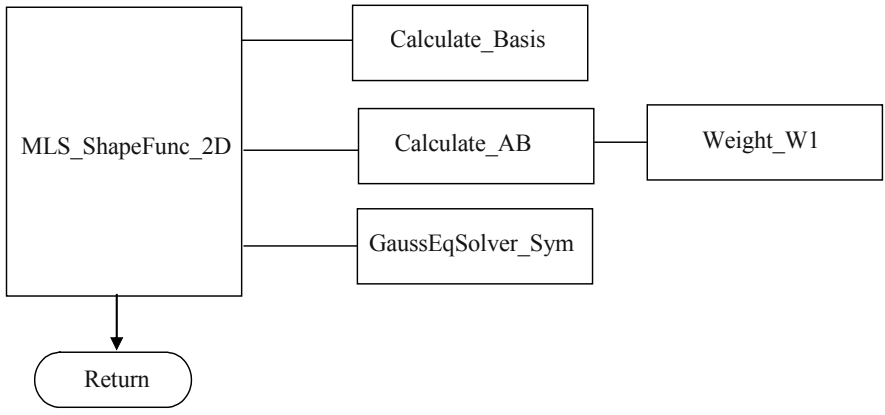


Figure 3.19. Macro flowchart for subroutine MLS_ShapeFunc_2D.

3.3.4.2 Program and data structure

The main subroutine MLS_ShapeFunc_2D calls several sub-subroutines. The macro chart for the MLS_ShapeFunc_2D can be seen in Figure 3.19. The functions of these sub-subroutines are listed in Appendix 3.7. The subroutine GaussEqSolver_Sym of a standard equation solver has been given in Sub-section 3.2.3.2.

1) Subroutines Weight_W1 and Weight_W2

Source code location: Program 3.5 and Program 3.6.
Dummy arguments: Appendix 3.8.
Function: to compute the cubic spline function (W1) and the quartic spline function (W2) given in Equations (3.165)~(3.171).

2) Subroutine Compute_Basis

Source code location: Program 3.7.
Dummy arguments: Appendix 3.9.
Function: to compute the basis function and its derivatives. In the current program, the basis of Equation (3.12) is used. In fact, the user can easily change the number of basis functions through the control constant, *mm*, that is the number of monomials used in the basis functions (i.e., *mm* is the *m* used in Equation (3.123)).

3) Subroutine Compute_AB

Source code location: Program 3.8.
Dummy arguments: Appendix 3.10.
Function: to compute matrices **A** and **B** those are given in Equations (3.129) and (3.132).

4) Main Subroutine MLS_ShapeFunc_2D

Source code location: Program 3.9.
Dummy arguments: Appendix 3.11.
Function: to compute MLS shape functions and their derivatives for a two-dimensional domain.

3.3.4.3 Examples of MLS shape functions

An example is presented to illustrate the properties of the MLS shape function, and its derivative are computed using 25(5×5) nodes. These 25 nodes are regularly distributed in a rectangular domain: $x_i \in [-1,\ 1]$ and $y_i \in [-1,\ 1]$, as shown in Figure 3.5. Coordinates of these 25 nodes are listed in Table 3.4. To evaluate and plot the shape function and its derivatives, a resolution of 61×61 points is used.

The main program listed in Program 3.10, is used to evaluate MLS shape functions. In the listed source code, the number of monomials used in the basis function, m, is 3. The user can easily change m to 6.

1) MLS shape functions and their derivatives

The program obtains the MLS shape functions and their derivatives at these 61×61 points first using m=3 and the weight function W1. The MLS shape functions ϕ and their derivatives $\partial\phi/\partial x$ and $\partial\phi/\partial y$ for the central node 13 (see Figure 3.5) are plotted in Figure 3.20~Figure 3.22. The ϕ and $\partial\phi/\partial x$ for the central node 13 along the line y=0 are plotted in Figure 3.23. Appendix 3.12 lists a sample output for shape functions at the evaluation point $\mathbf{x}^T$=[0, 0]. Appendix 3.12 confirms that MLS shape functions have the following properties.

First, by adding up the values of ϕ_i at all the 25 nodes, we can confirm the fact that the MLS shape function is of a partition of unity, i.e.

$$\sum_{i=1}^{n} \phi_i(\mathbf{x}) = 1 \tag{3.172}$$

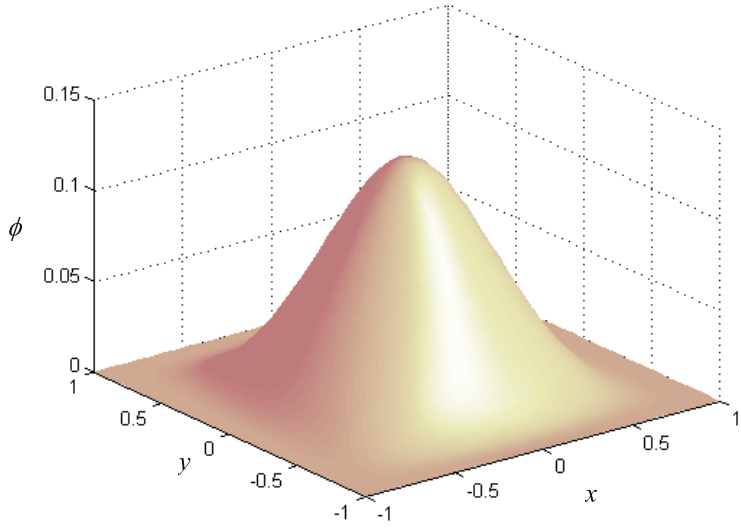


Figure 3.20. MLS shape function for node 13 at $\mathbf{x}^T$=[0, 0] obtained using 25 nodes shown in Figure 3.5.

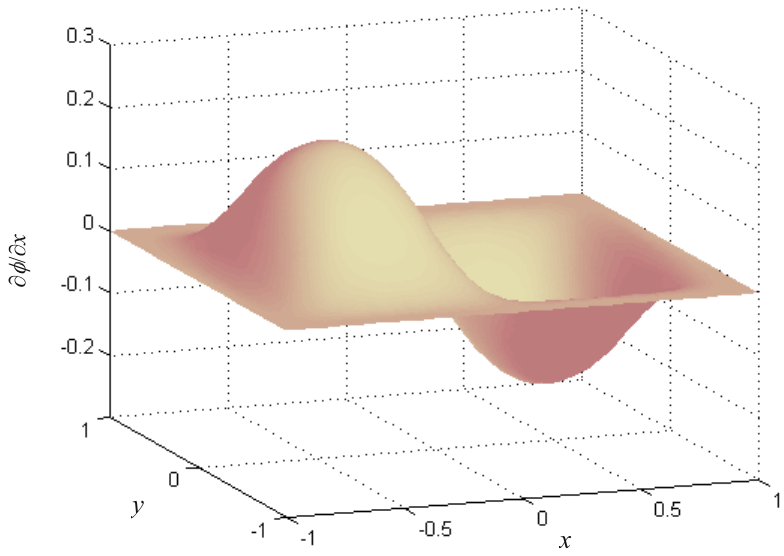


Figure 3.21. The first-order derivative of the MLS shape function for node 13 at $\mathbf{x}^T$=[0, 0] obtained using 25 nodes shown in Figure 3.5.

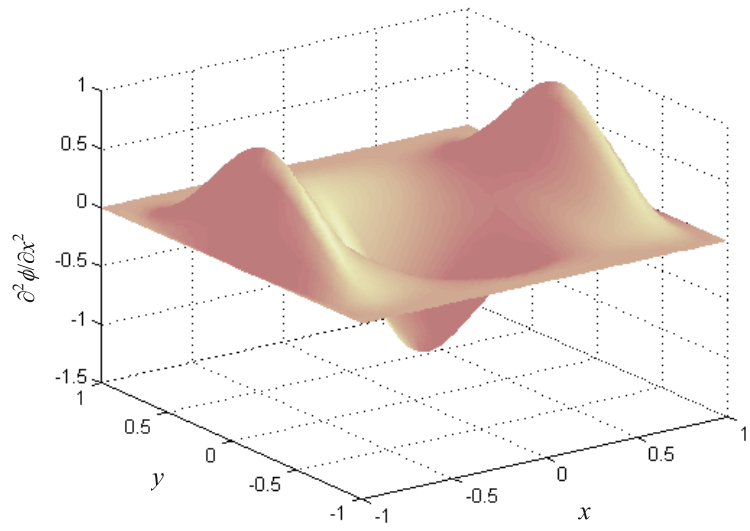


Figure 3.22. The second-order derivative of the MLS shape function for node 13 at $\mathbf{x}^T$=[0, 0] obtained using 25 nodes shown in Figure 3.5.

Second, in Appendix 3.12, the point $\mathbf{x}^T$=[0, 0] is located at the field node 13. However, $\phi_{13}(\mathbf{x})$=0.1467≠1.0. The MLS shape functions do not satisfy the Kronecker delta condition.

Third, although only the low order basis is used, the MLS shape functions have high order continuity due to the use of the weight function. In this example, although only the linear basis function (m=3) is used, the shape function has higher order continuity. This fact is evident from the values of $\partial\phi_i/\partial x$ that are not constants but very smoothly, as also shown in Figure 3.21. Note that even the 2nd derivatives $\partial^2\phi_i/\partial x^2$ of the shape functions are also smooth as shown in Figure 3.22.

2) Effect of weight functions

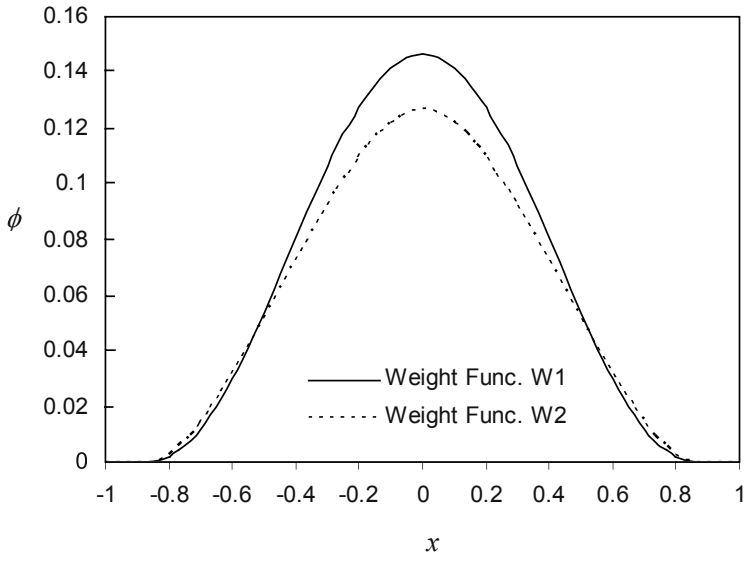
Weight functions W1 and W2 are used to construct the shape functions. Results of ϕ and $\partial\phi/\partial x$ for the central field node 13 along a line y=0 are plotted in Figure 3.23. This figure shows that the weight function will affect the MLS shape function. When the order of basis is the same, the shape function will inherit the continuity of the weight functions. Because these two weight functions (W1 and W2) have different shapes and order of continuities, the MLS shape functions of different weight functions shown in Figure 3.23 are clearly different.

3) Effect of the order of basis functions

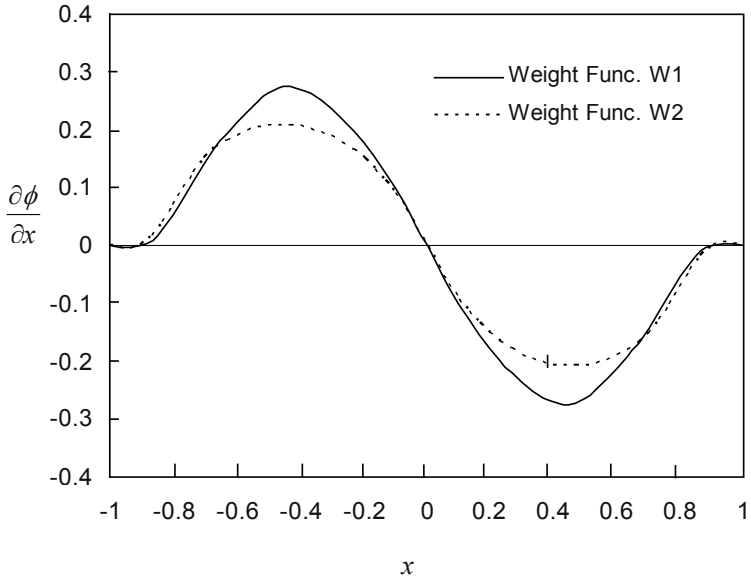
MLS shape functions and their derivatives using linear basis function (m=3) and the quadric basis function (m=6) are computed. Results for the central node 13 along the line y=0 and different m are plotted in Figure 3.24. The basis function affects the MLS shape function. When m becomes larger, the $\phi_{13}(\mathbf{x}=\mathbf{0})$ increases. If m=n=25, the $\phi_{13}(\mathbf{x}=\mathbf{0})$=1. In this case, the MLS approximation will become an interpolation of passing nodal values, and the MLS shape function will become the PIM shape function that possesses the Kronecker delta property if the moment matrix $\mathbf{A}$ is invertible. Of course, when m>n, the MLS approximation will fail because $\mathbf{A}^{-1}$ will not exist.

3.4 INTERPOLATION ERROR USING MESHFREE SHAPE FUNCTIONS

The MFree methods firstly depend upon the quality of MFree shape functions. Hence, the interpolation errors using MFree shape functions are examined through surface fitting operations for given functions.

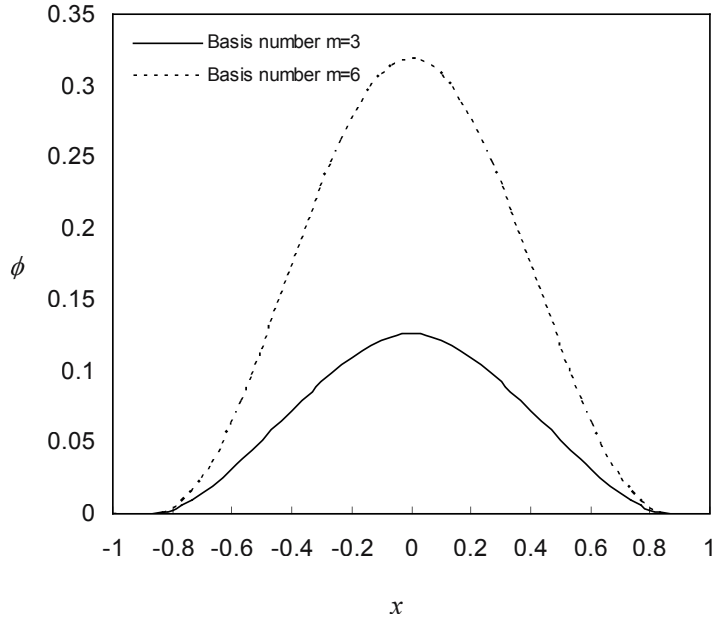


(a)

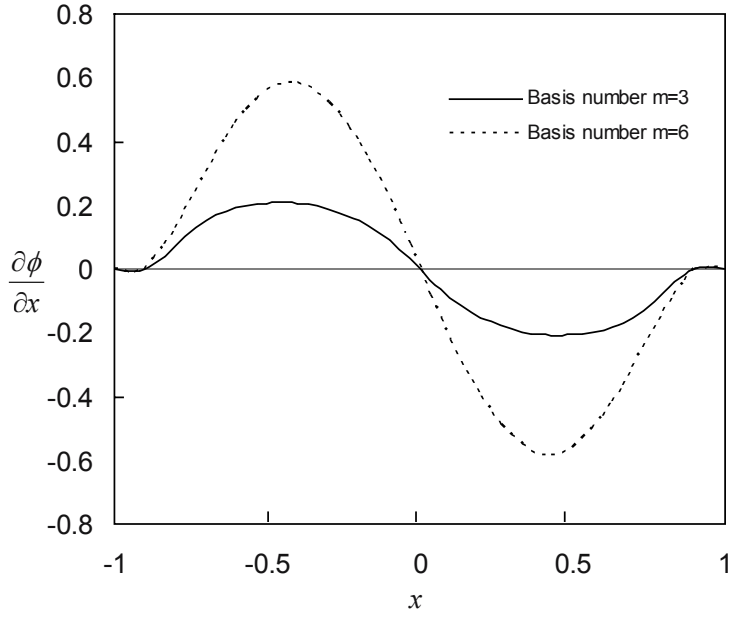


(b)

Figure 3.23. Effects of weight functions on the MLS shape function of node 13 and its derivative. The results are plotted along the line of y=0 using different weight functions. (a) the shape function; (b) the first-order derivative of the shape function.



(a)



(b)

Figure 3.24. MLS shape functions of node 13 obtained using different numbers of basis functions m=3 and m=6. The results are plotted along the line of y=0. (a) the shape function; (b) the first-order derivative of the shape function.

The MLS and RPIM shape functions are studied here as they will be used to develop MFree methods in the following chapters. A domain of (x, y)∈[0,10]×[0,10] is considered for the surface fitting. The domain is represented by 11×11 uniformly distributed field nodes with a constant nodal distance d_c=1.0. A total of 100 points of (x, y)∈[0.4, 9.4]×[0.4, 9.4] with distance h=1.0 are considered as interpolation points; they are intentionally chosen not to coincide with the field nodes to obtain a fair assessment of the fitting accuracy. In order to perform the interpolation for an interpolation point, a rectangular local support domain is used. The dimension of the local support domain is defined in Equation (3.114), in which the nodal spacing d_{cx} and d_{cy} in the x and y directions, respectively, are all set to 1.0.

The RPIM-MQ augmented with linear polynomials and the MLS approximation using the linear basis and the cubic spline weight (W1) function are investigated. The approximated value of the field function $f(\mathbf{x})$ for each interpolation point $\mathbf{x}$ can be interpolated using the nodes in the support domain and the shape functions. Let the approximated function be denoted by $\tilde{f}(\mathbf{x})$, we then have

$$\tilde{f}(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x})\mathbf{F}_s = \sum_{i=1}^{n} \phi_i f_i \tag{3.173}$$

where ϕ_i is the MLS or RPIM shape function, and n is the number of field nodes used in the support domain. The $\mathbf{F}_s$ is the vector that collects the true nodal function values (calculated analytically using the given function) for these n field nodes, and f_i is the function value for the ith field node.

The derivatives of $f(\mathbf{x})$ at an interpolation point $\mathbf{x}$ can also be approximated using shape functions, i.e.,

$$\frac{\partial \tilde{f}(\mathbf{x})}{\partial x_i} = \frac{\partial \mathbf{\Phi}(\mathbf{x})}{\partial x_i}\mathbf{F}_s = \sum_{i=1}^{n} \frac{\partial \phi_i}{\partial x_i} f_i \tag{3.174}$$

The following norms are used as error indicators. The average fitting errors of function values over the entire domain are defined as

$$e_t = \frac{1}{N}\sum_{i=1}^{N} \left| \frac{\tilde{f}_i - f_i}{f_i} \right| \tag{3.175}$$

where N is the total number of interpolation points in the entire domain, f_i is the exact values of function, and $\tilde{f}_i$ is the approximated values of function. In this example, N=100 is used.

The average fitting error of the 1st derivative of the approximated (fitted) function at the interpolation point i is defined as

$$e_t' = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\tilde{f}_i' - f_i'}{f_i'}\right| \tag{3.176}$$

where f_i' is the exact value of the 1st derivative of the function, and $\tilde{f}_i'$ is values of the 1st derivative of the approximated function.

3.4.1 Fitting of a planar surface

A two-dimensional plane is considered first. i.e.,

$$f_1(x, y) = x + y + 1.0 \tag{3.177}$$

It can be observed that both RPIM-MQ and MLS can exactly fit the plane to the machine accuracy (10^{-16}). This confirms the linear reproduction property of these shape functions. It should be mentioned here that the surface fitting for the plane will have errors ($10^{-5} \sim 10^{-7}$) if the linear polynomial term is not included in the RPIM-MQ. This is because the RPIM-MQ without the augment of a linear polynomial term cannot exactly reproduce linear polynomials, only a good approximation.

3.4.2 Fitting of a complicated surface

The following non-polynomial surface is fitted using the RPIM-MQ and MLS shape functions.

$$f_2(x, y) = \sin(x \cdot \frac{2\pi}{10})\cos(y \cdot \frac{2\pi}{10}) + 1.5 \tag{3.178}$$

1) Shape parameters of the RPIM-MQ

The effects of two shape parameters, q and α_c, in the MQ-RBF are first studied. In the studies of shape parameters, α_s=3.5 is used for the support domain.

In the study of q, α_c=1.0 is fixed. The average fitting errors e_t are obtained for different q and plotted in Figure 3.25. It can be found that the interpolation quality changes with q. The fitting error decreases when q is in the vicinity of 1.0, 2.0 and 3.0. However, if q=1.0, 2.0, and 3.0, the RPIM-MQ will fail due to the singularity of the moment matrix. When q>3.0, the error is also very large due to the badly conditioned moment matrix.

In addition, the condition number of interpolation matrix of RPIM becomes larger as q approaches 1.0 or 2.0 or 3.0. The preferred value of parameter q is $q \approx 1.0$ or 2.0 (but not equal 1.0 or 2.0, say, 0.98 or 1.03 or 1.99). This confirms the feature of the RPIM shape functions that more

accurate fitting results are obtained when the moment matrix approaches (but is not yet) singular. However, when q is too close to 1.0 or 2.0 where the moment matrix is nearly singular, the results are not stable due to the badly conditioned moment matrix. Therefore, in using RPIM shape functions, one needs to keep a balance between accuracy and stability. Hence, q=0.98 or 1.03 is recommended for many problems (GR Liu, 2002).

The fitting errors using different α_c are plotted in Figure 3.26. In the studies of α_c, q=0.5 is fixed. It can be found that a smaller α_c leads to a larger interpolation error. The effect of a_c is less than that of q.

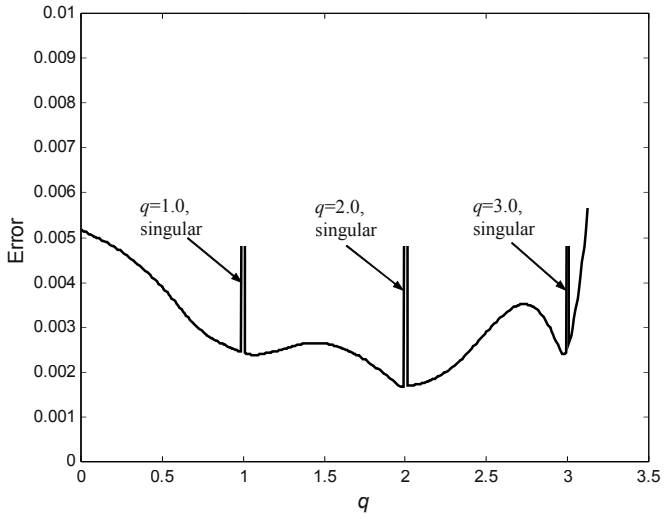


Figure 3.25. Error e_t of surface fitting using RPIM-MQ shape functions with different q. (MQ-RBF is used with shape parameters: $\alpha_c = 1.0$, $d_{cx} = d_{cy} = 1.0$; the size of support domain is $\alpha_s = 3.5$, and $m = 0$.).

2) Comparation studies of accuracy

The interpolation errors of MLS and RPIM-MQ are compared in Figure 3.27. From these two figures, the following conclusions can be drawn.

a) The accuracy in the fitted function itself is higher than that in the derivatives. The higher the derivatives, the lower the accuracy.

b) Using the RPIM-MQ shape functions gives satisfactory accuracy in the surface fitting. The fitting accuracy improves with the increase of the size of the support domain. In addition, the shape parameters of RBF chosen affect the fitting results. The fitting accuracy is unstable sometimes when the moment matrix is too close to singular.

c) MLS with linear basis is less accurate in this example. The increase in the size of the support domain cannot improve the fitting results. Lower accuracy in surface fitting is because the MLS shape functions do no pass through the nodal values.

3) Convergence studies

In the convergence study, regularly and evenly distributed 36 (6×6), 121 (11×11), 441 (21×21), 961 (31×31), 1296 (36×36), 1681 (41×41), and 6561 (81×81), nodes are used. The convergence curves are numerically obtained are plotted in Figure 3.28. Note that in Figure 3.28 h is in fact the nodal spacing d_c defined in Sub-section 3.1.2. To coincide with the common definition of h-convergence, h is used here and in the following chapters in the studies of h-convergence. The following remarks can be made from Figure 3.28.

- The accuracy of RPIM-MQ is higher than that of the MLS. However, the convergent process of RPIM-MQ is not very stable when finer nodes are used, although the accuracy is still much better than that of MLS. Further tuning of the shape parameters are necessary.
- The MLS has very steady convergence for the surface fitting.

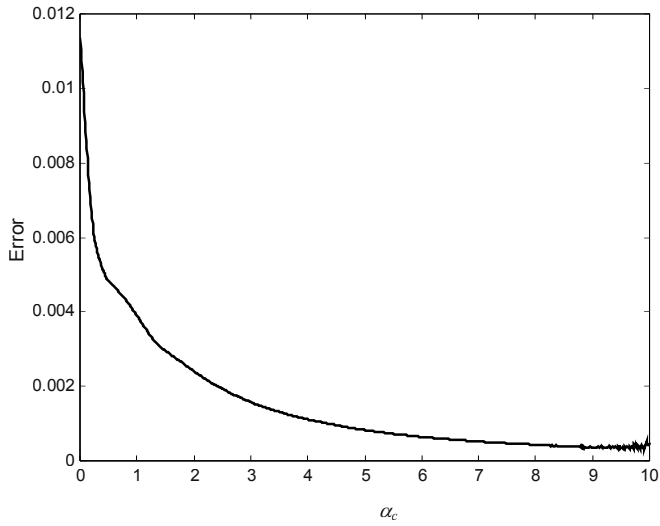
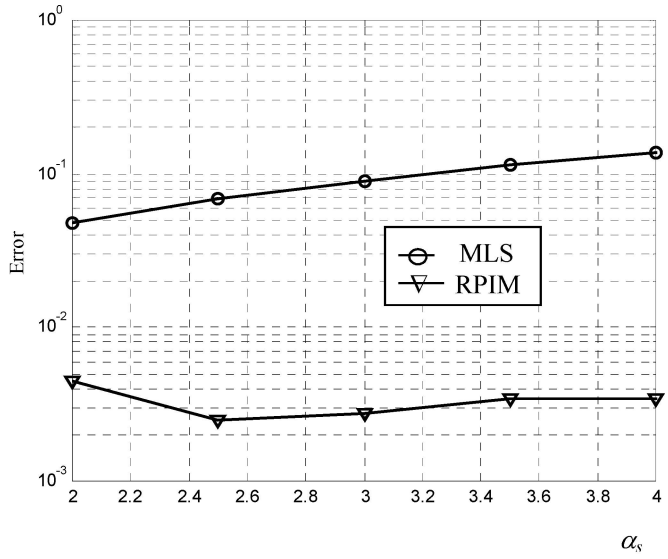
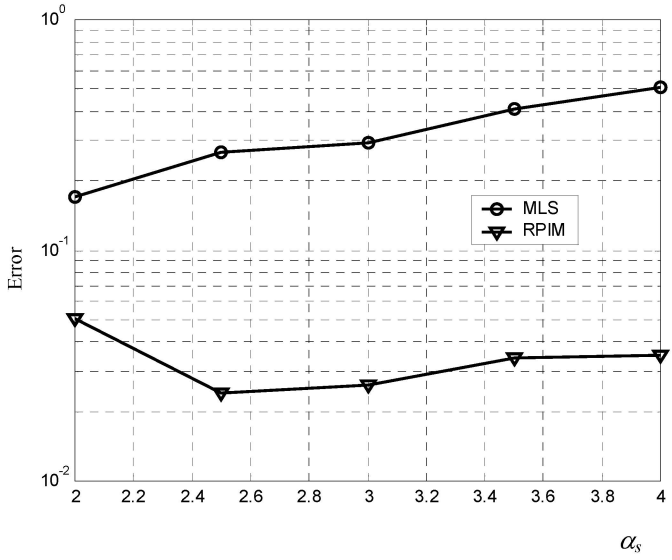


Figure 3.26. Error e_t of surface fitting using RPIM-MQ for different α_c. (MQ-RBF is used with shape parameters: $q = 0.5$, $d_{cx} = d_{cy} = 1.0$; the size of support domain of $\alpha_s = 3.5$, and $m = 0$.).



(a) function



(b) 1st derivative

Figure 3.27. Error e_t in surface fitting using MLS and RPIM-MQ shape functions created using different size of support domains.
In MQ-RBF, $q = 1.03$, $\alpha_c = 4$, $d_{cx} = d_{cy} = 1.0$, and $m = 0$ are used; In MLS, the linear basis is used.

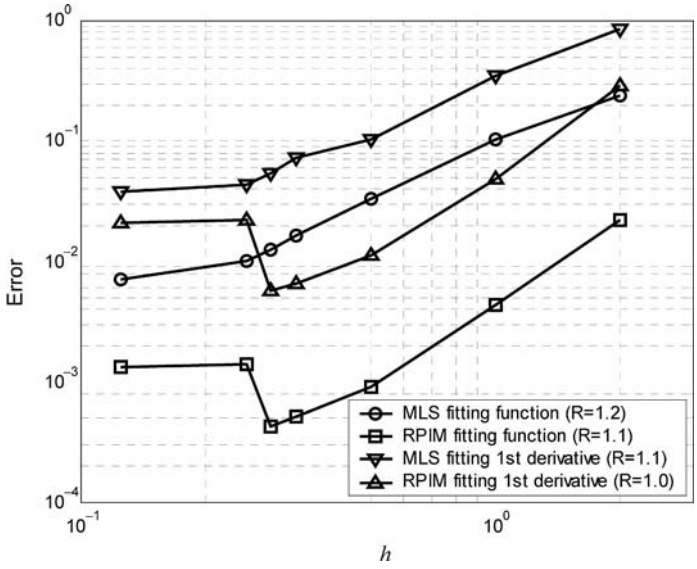


Figure 3.28. Error e_t in surface fitting using MLS and RPIM-MQ shape functions. (In MQ-RBF, q=1.03, $\alpha_c = 4$, $d_{cx} = d_{cy} = 1.0$, $\alpha_s = 3.5$, and $mbasis = 0$ are used; In MLS, the linear basis is used)

It should be mentioned here that the interpolation error is only one part of total error in an MFree method in solving a problem of computational mechanics. The studies of shape parameters presented in this section are only for checking the interpolation quality and the producibility of MLS and RPIM shape functions. The accuracy will be also studied in the following chapters in the analysing actual problems of computational mechanics.

3.5 REMARKS

In MFree methods, the first and one of the most important problems that we face is the MFree function approximation. In contrast to the FEM, there is no pre-defined element in MFree models that can be used in the function approximation. One of the challenges in MFree methods is how to construct shape functions efficiently without using any pre-defined relations between nodes.

This chapter presents a number of ways to meet this challenge. These MFree shape functions possess the following important features.

- Reproducibility: they are capable of reproducing what is contained in the basis; this is essential and crucial for any numerical method to produce accurate solutions.
- Convergence: this allows the error of the approximation of a function that is sufficiently smooth to approach zero when the nodal spacing is sufficiently reduced.

None of these shape functions depends upon any fixed relation between nodes. This brings freedom in the formulation of an MFree method. It is also easy to construct an MFree shape function with high orders; this is needed for the solution of high order PDEs. The MFree shape functions reduce the effort spent in post-processing. Unfortunately, these freedoms also lead to some challenging problems, for example, compatibility, efficiency, and accuracy.

APPENDIX

Appendix 3.1. Subroutines used in the program of RPIM_ShapeFunc_2D.f90

Subroutines	Functions
RPIM_ShapeFunc_2D	Compute the RPIM shape function and their derivatives.
Compute_RadialBasis	Compute the basis function vector and its derivatives for a point.
GaussEqSolver_Sym	Solve the linear symmetric equation using Gauss elimination.

Appendix 3.2. Dummy arguments used in the subroutine RPIM_ShapeFunc_2D

Variable	Type	Usage	Function
nx	Integer	Input	Dimension of this problem; *nx*=2 for 2D problem
numnode	Integer	Input	Number of field nodes
ndex	Integer	Input	Number of field nodes in the support domain
mbasis	Integer	Input	Number of monomials used in the augmented RBF
nRBF	Integer	Input	Types of RBF. *nRBF*=1: MQ; *nRBF*=2: Exp; *nRBF*=3: TSP
alfc	Long real	Input	Shape parameter of RBF
q	Long real	Input	Shape parameter of RBF
dc	Long real	Input	Nodal spacing
X(*nx*, *numnode*)	Long real	Input	Coordinates x and y for all field nodes. $x(1,i)=x_i$; $x(2,i)=y_i$
Gpos(*nx*)	Long real	Input	Coordinates of the point of interest. g*pos*(1)=*x*, *gpos*(2)=*y*

Nv(*ndex*)	Integer	Input	Field nodes in the local support domain
Phi(10, *ndex*)	Long real	Output	RPIM shape functions and their derivatives. *phi*(1,*i*)~ *phi*(10, *i*): $\phi_i, \frac{\partial \phi_i}{\partial x}, \frac{\partial \phi_i}{\partial y}, \frac{\partial^2 \phi_i}{\partial x^2}, \frac{\partial^2 \phi_i}{\partial x \partial y}, \frac{\partial^2 \phi_i}{\partial y^2}$ $\frac{\partial^3 \phi_i}{\partial x^3}, \frac{\partial^3 \phi_i}{\partial x^2 \partial y}, \frac{\partial^3 \phi_i}{\partial x \partial y^2}, \frac{\partial^3 \phi_i}{\partial y^3}$

Appendix 3.3. Dummy arguments used in the subroutine Compute_RadialBasis

Variable	Type	Usage	Function
ndex	Integer	Input	Number of field nodes used in the support domain
mbasis	Integer	Input	Number of monomials used in the augmented RBF
nRBF	Integer	Input	Types of RBF. *nRBF*=1: MQ; *nRBF*=2: Exp; *nRBF*=3: TSP
alfc	Long real	Input	Shape parameter of RBF
q	Long real	Input	Shape parameter of RBF
dc	Long real	Input	Nodal spacing
x, y	Long real	Input	Coordinates of the point considered.
Xv(*ndex*)	Long real	Input	Coordinates *x* and *y* for field nodes in the support domain.
Rk(10, *ndex*)	Long real	Output	RBF and its derivatives. *rk*(1,*i*)~ *rk*(10, *i*): $R_i, \frac{\partial R_i}{\partial x}, \frac{\partial R_i}{\partial y}, \frac{\partial^2 R_i}{\partial x^2}, \frac{\partial^2 R_i}{\partial x \partial y}, \frac{\partial^2 R_i}{\partial y^2}, \ldots$

Appendix 3.4. Dummy arguments used in the subroutine GaussEqSolver_Sym

Variable	Type	Usage	Function
n	Integer	Input	Number of linear equations.
ma	Integer	Input	Number of rows of matrix **A**.

A(*ma*, *n*)	Long real	Input	Coefficient matrix of **Ax**=**B**.
B(*n*)	Long real	Input Output	Right-hand side vector of **Ax**=**B** when input. The solution when output.
ep	Long real	Input	Required tolerance
kwji	Integer	Output	Control constant. When there is a unique solution, *kwij*=0; Else *kwij*=1

Appendix 3.5. An output sample of the shape function for node 13 evaluated at point $\mathbf{x}^{\mathrm{T}} = \{0.2, 0.4\}$ using the subroutine RPIM_ShapeFunc_2D and 25 field nodes shown in Figure 3.5 * +

Node	x	y	Phi	dPhidx	dPhidy	dPhidxx	dPhidyy
1	-1.000	-1.000	-0.00151	0.00227	0.00120	0.02160	-0.01028
2	-1.000	-0.500	-0.00398	-0.00624	0.03283	0.19871	0.05061
3	-1.000	0.000	0.00737	0.01075	-0.13463	-0.22282	0.10253
4	-1.000	0.500	0.04017	0.03350	0.10152	-1.85485	-0.29777
5	-1.000	1.000	-0.00765	-0.00509	-0.00138	0.48947	0.12937
6	-0.500	-1.000	-0.00322	-0.00307	0.04331	0.10175	0.03833
7	-0.500	-0.500	0.01287	0.01082	-0.12416	-0.60414	0.03134
8	-0.500	0.000	-0.03362	-0.04869	0.41194	1.48867	-1.05243
9	-0.500	0.500	-0.16227	-0.03285	-0.21122	6.42777	2.27300
10	-0.500	1.000	0.02342	-0.03011	-0.13106	-0.86947	-1.43074
11	0.000	-1.000	0.01691	-0.04493	-0.15219	-0.18499	-0.30862
12	0.000	-0.500	-0.05798	0.17452	0.56904	0.54431	0.90340
13	0.000	0.000	0.18692	-0.46696	-1.98023	-1.79731	2.40048
14	0.000	0.500	0.67546	-2.02700	1.20211	-6.76875	-6.92754
15	0.000	1.000	-0.05197	0.25104	0.37398	0.59049	4.11842
16	0.500	-1.000	0.00714	0.04825	-0.09769	0.08985	-0.24436
17	0.500	-0.500	-0.04112	-0.20805	0.38631	0.12677	0.67096
18	0.500	0.000	0.10019	0.53426	-1.23158	0.24573	1.28296
19	0.500	0.500	0.39863	2.33843	0.76557	0.12128	-3.96495
20	0.500	1.000	-0.04728	-0.30139	0.18662	0.32000	2.38442
21	1.000	-1.000	-0.00108	0.00051	0.04183	-0.03091	0.05135
22	1.000	-0.500	0.00946	0.02808	-0.10457	-0.26924	0.04927
23	1.000	0.000	-0.00557	-0.01987	0.23208	0.26835	-0.74743
24	1.000	0.500	-0.08927	-0.33834	-0.10411	2.12881	1.51954
25	1.000	1.000	0.02827	0.10102	-0.07641	-0.56253	-1.03623

$$\sum_i \phi_i = 1.00029$$

* *Phi*: ϕ_i; *dPhidx*: $\frac{\partial \phi_i}{\partial x}$; *dPhidy*: $\frac{\partial \phi_i}{\partial y}$; *dPhidxx*: $\frac{\partial^2 \phi_i}{\partial x^2}$; *dPhidyy*: $\frac{\partial^2 \phi_i}{\partial y^2}$

+ MQ-RBF is used with $q = 0.5$, $\alpha_c = 2.0$, $d_c = 0.5$ and *mbasis*=0

Appendix 3.6. An output sample of the shape function for node 13 evaluated at point $\mathbf{x}^{\mathbf{T}} = \{0.,0.\}$ using the subroutine RPIM_ShapeFunc_2D and 25 field nodes shown in Figure 3.5* +

Node	x	y	Phi	dPhidx	dPhidy	dPhidxx	dPhidyy
1	-1.000	-1.000	0.00000	-0.02604	-0.02604	0.05957	0.05957
2	-1.000	-0.500	0.00000	-0.03106	0.02755	0.17930	-0.07751
3	-1.000	0.000	0.00000	0.39911	0.00000	-1.27964	0.06856
4	-1.000	0.500	0.00000	-0.03106	-0.02755	0.17930	-0.07751
5	-1.000	1.000	0.00000	-0.02604	0.02604	0.05957	0.05957
6	-0.500	-1.000	0.00000	0.02755	-0.03106	-0.07751	0.17930
7	-0.500	-0.500	0.00000	0.02556	0.02556	-0.33220	-0.33220
8	-0.500	0.000	0.00000	-1.66139	0.00000	7.73756	0.37809
9	-0.500	0.500	0.00000	0.02556	-0.02556	-0.33220	-0.33220
10	-0.500	1.000	0.00000	0.02755	0.03106	-0.07751	0.17930
11	0.000	-1.000	0.00000	0.00000	0.39911	0.06856	-1.27964
12	0.000	-0.500	0.00000	0.00000	-1.66139	0.37809	7.73756
13	0.000	0.000	1.00000	0.00000	0.00000	-13.11775	-13.11775
14	0.000	0.500	0.00000	0.00000	1.66139	0.37809	7.73756
15	0.000	1.000	0.00000	0.00000	-0.39911	0.06856	-1.27964
16	0.500	-1.000	0.00000	-0.02755	-0.03106	-0.07751	0.17930
17	0.500	-0.500	0.00000	-0.02556	0.02556	-0.33220	-0.33220
18	0.500	0.000	0.00000	1.66139	0.00000	7.73756	0.37809
19	0.500	0.500	0.00000	-0.02556	-0.02556	-0.33220	-0.33220
20	0.500	1.000	0.00000	-0.02755	0.03106	-0.07751	0.17930
21	1.000	-1.000	0.00000	0.02604	-0.02604	0.05957	0.05957
22	1.000	-0.500	0.00000	0.03106	0.02755	0.17930	-0.07751
23	1.000	0.000	0.00000	-0.39911	0.00000	-1.27964	0.06856
24	1.000	0.500	0.00000	0.03106	-0.02755	0.17930	-0.07751
25	1.000	1.000	0.00000	0.02604	0.02604	0.05957	0.05957

$$\sum_i \phi_i = 1.00000$$

* *Phi*: ϕ_i; *dPhidx*: $\frac{\partial \phi_i}{\partial x}$; *dPhidy*: $\frac{\partial \phi_i}{\partial y}$; *dPhidxx*: $\frac{\partial^2 \phi_i}{\partial x^2}$; *dPhidyy*: $\frac{\partial^2 \phi_i}{\partial y^2}$

+ MQ-RBF is used with $q = 0.5$, $\alpha_c = 2.0$, $d_c = 0.5$ and *mbasis*=0

Appendix 3.7. Subroutines used in the program of MLS_ShapeFunc_2D.f90

Subroutines	Functions
MLS_ShapeFunc_2D	Compute the MLS shape function and their derivatives
Compute_Basis	Compute the polynomial basis vector and its derivatives at a given point
Compute_AB	Compute **A** and **B** matrices in the MLS given in Equations (3.131) and (3.133)
Weight_W1(or Weight_W2)	Compute the cubic spline function (W1) (or the quartic spline function, W2) defined in Equations (3.148) and (3.149), respectively
GaussEqSolver_Sym	Solve the linear symmetric equation using the Gauss elimination method

Appendix 3.8. Dummy arguments used in the subroutines Weight_W1 and Weight_W2

Variable	**Type**	**Usage**	**Function**
nx	Integer	Input	Dimension of this problem; *nx*=2 for 2D problem
numnode	Integer	Input	Number of field nodes
ndex	Integer	Input	Number of field nodes used in the support domain
Nv(*ndex*)	Integer	Input	Field nodes in the support domain
Dif(*nx*,*ndex*)	Long real	Input	Distances: $dif(1,i) = \|x - x_I\|$, $dif(2,i) = \|y - y_I\|$
Ds*(nx, numnode)*	Long real	Input	The size of the support domain: $ds(1,i)=d_{sxi}$, $ds(2,i)=d_{syi}$
W(*ndex,10*)	Long real	Output	Weight function and its derivatives: $\widehat{W}(i,1) = \widehat{W}_i$; $\widehat{W}(i,2) = \partial\widehat{W}_i / \partial x$; $\widehat{W}(i,3) = \partial\widehat{W}_i / \partial y$; $\widehat{W}(i,4) = \partial^2\widehat{W}_i / \partial x^2$ $\widehat{W}(i,5) = \partial^2\widehat{W}_i / \partial x\partial y$; $\widehat{W}(i,6) = \partial^2\widehat{W}_i / \partial y^2$ …

Appendix 3.9. Dummy arguments used in the subroutine Compute_Basis

Variable	**Type**	**Usage**	**Function**
nx	Integer	Input	Dimension of this problem; *nx*=2 for 2D problem
mm	Integer	Input	Number of monomials used in the basis
Gpos(*nx*)	Long real	Input	Coordinates of the point of interest: $gpos(1) = x$, $gpos(2) = y$
Gp*(mm,* 10*)*	Long real	Output	Basis function and its derivatives: $gp(1,1)\sim(6,1)=\mathbf{p}^{\mathrm{T}}=\{1,x,y,xy,x^2,y^2\}$; $gp(1,2)\sim(6,2)=(\frac{\partial\mathbf{p}}{\partial x})^{\mathrm{T}}=\{0,1,0,y,2x,0\}$; $gp(1,3)\sim(6,3)=(\frac{\partial\mathbf{p}}{\partial y})^{\mathrm{T}}=\{0,0,1,x,0,2y\}$; ……

Appendix 3.10. Dummy arguments used in the subroutine Compute_AB

Variable	Type	Usage	Function
nx	Integer	Input	Dimension of this problem; *nx*=2 for 2D problem
numnode	Integer	Input	Number of field nodes
ndex	Integer	Input	Number of field nodes in the support domain
mm	Integer	Input	Number of monomials used in the basis
X(*nx*, *numnode*)	Long real	Input	x and y coordinates for all field nodes: $x(1, i)=x_i$; $x(2, i)=y_i$
Gpos(*nx*)	Long real	Input	Coordinates of the point of interest: $gpos(1) = x, gpos(2) = y$
Nv(*ndex*)	Integer	Input	Field nodes used in the support domain
Ds*(nx, numnode)*	Long real	Input	The dimension of the support domain: $ds(1,i)=d_{sxi}$, $ds(2,i)=d_{syi}$
A(*mm*,*mm*,10)	Long real	Output	**A** matrix and its derivatives: $A(i,j,1)\sim A(i,j,10)$ are A_{ij}, $(\frac{\partial \mathbf{A}}{\partial x})_{ij}$, $(\frac{\partial \mathbf{A}}{\partial y})_{ij}$, $(\frac{\partial^2 \mathbf{A}}{\partial x^2})_{ij}$, ……
B(*mm*,*mm*,10)	Long real	Output	**B** matrix and its derivatives: $B(i,j,1)\sim B(i,j,10)$ are B_{ij}, $(\frac{\partial \mathbf{B}}{\partial x})_{ij}$, $(\frac{\partial \mathbf{B}}{\partial y})_{ij}$, $(\frac{\partial^2 \mathbf{B}}{\partial x^2})_{ij}$, ……

Appendix 3.11. Dummy arguments used in the subroutine MLS_ShapeFunc_2D

Variable	Type	Usage	Function
nx	Integer	Input	Dimension of this problem; *nx*=2 for 2D problem
numnode	Integer	Input	Number of field nodes
ndex	Integer	Input	Number of field nodes in the support domain

mm	Integer	Input	Number of monomials used in the basis
x(*nx*, *numnode*)	Long real	Input	*x* and *y* coordinates for all field nodes: $x(1, i)=x_i$; $x(2, i)=y_i$
Gpos(*nx*)	Long real	Input	Coordinates of the point of interest: $gpos(1) = x$, $gpos(2) = y$
Nv(*ndex*)	Integer	Input	Field nodes used in the support domain
Ds*(nx, numnode)*	Long real	Input	The dimension of the support domain: $ds(1,i)=d_{sxi}$, $ds(2,i)=d_{syi}$
Phi(10, *ndex*)	Long real	Output	MLS shape functions and their derivatives.

Appendix 3.12. An output sample of the shape function for node 13 evaluated at point $\mathbf{x}^{\mathrm{T}} = \{0, 0\}$ using the subroutine MLS_ShapeFunc_2D, 25 field nodes shown in Figure 3.5, and weight function W1 and *mm*=3*

Node	x	y	Phi	dPhidx	dPhidy	dPhidxx	dPhidyy
1	-1.000	-1.000	0.00917	-0.02488	-0.02488	0.03444	0.03444
2	-1.000	-0.500	0.02037	-0.05529	-0.04598	0.07653	-0.02474
3	-1.000	0.000	0.03667	-0.09953	0.00000	0.13776	-0.01939
4	-1.000	0.500	0.02037	-0.05529	0.04598	0.07653	-0.02474
5	-1.000	1.000	0.00917	-0.02488	0.02488	0.03444	0.03444
6	-0.500	-1.000	0.02037	-0.04598	-0.05529	-0.02474	0.07653
7	-0.500	-0.500	0.04527	-0.10218	-0.10218	-0.05498	-0.05498
8	-0.500	0.000	0.08148	-0.18392	0.00000	-0.09897	-0.04310
9	-0.500	0.500	0.04527	-0.10218	0.10218	-0.05498	-0.05498
10	-0.500	1.000	0.02037	-0.04598	0.05529	-0.02474	0.07653
11	0.000	-1.000	0.03667	0.00000	-0.09953	-0.01939	0.13776
12	0.000	-0.500	0.08148	0.00000	-0.18392	-0.04310	-0.09897
13	0.000	0.000	0.14667	0.00000	0.00000	-0.07757	-0.07757
14	0.000	0.500	0.08148	0.00000	0.18392	-0.04310	-0.09897
15	0.000	1.000	0.03667	0.00000	0.09953	-0.01939	0.13776
16	0.500	-1.000	0.02037	0.04598	-0.05529	-0.02474	0.07653
17	0.500	-0.500	0.04527	0.10218	-0.10218	-0.05498	-0.05498
18	0.500	0.000	0.08148	0.18392	0.00000	-0.09897	-0.04310
19	0.500	0.500	0.04527	0.10218	0.10218	-0.05498	-0.05498
20	0.500	1.000	0.02037	0.04598	0.05529	-0.02474	0.07653
21	1.000	-1.000	0.00917	0.02488	-0.02488	0.03444	0.03444
22	1.000	-0.500	0.02037	0.05529	-0.04598	0.07653	-0.02474
23	1.000	0.000	0.03667	0.09953	0.00000	0.13776	-0.01939
24	1.000	0.500	0.02037	0.05529	0.04598	0.07653	-0.02474
25	1.000	1.000	0.00917	0.02488	0.02488	0.03444	0.03444

$$\sum_i \phi_i = 1.000000$$

* *Phi*: ϕ_i; *dPhidx*: $\frac{\partial \phi_i}{\partial x}$; *dPhidy*: $\frac{\partial \phi_i}{\partial y}$; *dPhidxx*: $\frac{\partial^2 \phi_i}{\partial x^2}$; *dPhidyy*: $\frac{\partial^2 \phi_i}{\partial y^2}$

COMPUTER PROGRAMS

Program 3.1. Source code of Subroutine RPIM_ShapeFunc_2D

```
    SUBROUTINE RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,&
                                 alfc,dc,q,nRBF, mbasis)
!---------------------------------------------------------------------
! Compute RPIM shape functions and their derivatives
! Input--gpos,x,nv,ds,alfc,dc,q,nx,numnode,ndex,mm,nRBF,nbasis
!        nRBF=1: MQ;  2:  EXP;    3: TSP
! Output--phi
!    From 1 to 10 of the two dimension of phi denotes
!         phi,dphix,dphiy,dphixx,dphixy,dphiyy
!         dphidxxx,dphidxxy, dphidxyy, dphidyyy
!---------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       dimension gpos(nx),x(nx,numnode),nv(ndex),rk(ndex+mbasis)
       dimension phi(10,ndex),xv(nx,ndex),rr(10,ndex+mbasis)
       dimension a(ndex+mbasis,ndex+mbasis),g0(ndex+mbasis,ndex+mbasis)

       if(nrbf.eq.1) then
          rc=alfc*dc           ! For MQ;
       endif
       if(nrbf.eq.2) then
          q=alfc/dc/dc         ! For EXP;
       endif
       ep=1.d-20
       mg=ndex+mbasis
       do i=1,mg
          do j=1,mg
             g0(i,j)=0.
          enddo
       enddo
       do i=1,ndex
          nn=nv(i)
          xv(1,i)=x(1,nn)
          xv(2,i)=x(2,nn)
       enddo
! ****************** Assemble the matrix of G0
       do i=1,ndex
          nn=nv(i)
          call Compute_RadialBasis(x(1,nn),x(2,nn),xv,rr,ndex,rc,q,nRBF,mbasis)
          do j=1,ndex
             g0(i,j)=rr(1,j)
          enddo
          if(mbasis.gt.0) then
             g0(i,ndex+1)=1.
             g0(i,ndex+2)=x(1,nn)
             g0(i,ndex+3)=x(2,nn)
             g0(ndex+1,i)=1.
             g0(ndex+2,i)=x(1,nn)
             g0(ndex+3,i)=x(2,nn)
          endif
       enddo

! ****************** Solve linear equation to obtain shape function
       do i=1,mg
          do j=1,mg
             a(i,j)=g0(i,j)
          enddo
```

```
       enddo
       call Compute_RadialBasis(gpos(1),gpos(2),xv,rr,ndex,rc,q,nRBF,mbasis)

       do i=1,mg
          rk(i)=rr(1,i)
       enddo
       call GaussEqSolver_Sym(mg,mg,a,rk,ep,kwji)
       if(kwji.eq.1) then
        write(*,*)'Fail...'
        pause
        endif
       do i=1,ndex
          phi(1,i)=rk(i)
       enddo

! ****************** Solve linear equation to obtain dphidx
       do i=1,mg
          do j=1,mg
             a(i,j)=g0(i,j)
          enddo
       enddo

       do i=1,mg
          rk(i)=rr(2,i)
       enddo
       call GaussEqSolver_Sym(mg,mg,a,rk,ep,kwji)
       do i=1,ndex
          phi(2,i)=rk(i)
       enddo

! ****************** Solve linear equation to obtain dphidy
       do i=1,mg
          do j=1,mg
             a(i,j)=g0(i,j)
          enddo
       enddo
       do i=1,mg
          rk(i)=rr(3,i)
       enddo
       call GaussEqSolver_Sym(mg,mg,a,rk,ep,kwji)
       do i=1,ndex
          phi(3,i)=rk(i)
       enddo

! ****************** Solve linear equation to obtain dphidxx
       do i=1,mg
          do j=1,mg
             a(i,j)=g0(i,j)
          enddo
       enddo
       do i=1,mg
          rk(i)=rr(4,i)
       enddo
       call GaussEqSolver_Sym(mg,mg,a,rk,ep,kwji)
       do i=1,ndex
          phi(4,i)=rk(i)
       enddo

! ****************** Solve linear equation to obtain dphidxy
       do i=1,mg
          do j=1,mg
             a(i,j)=g0(i,j)
          enddo
       enddo
       do i=1,mg
           rk(i)=rr(5,i)
       enddo
       call GaussEqSolver_Sym(mg,mg,a,rk,ep,kwji)
       do i=1,ndex
          phi(5,i)=rk(i)
```

```
       enddo

! ****************** Solve linear equation to obtain dphidyy
       do i=1,mg
          do j=1,mg
             a(i,j)=g0(i,j)
          enddo
       enddo
       do i=1,mg
          rk(i)=rr(6,i)
       enddo
       call GaussEqSolver_Sym(mg,mg,a,rk,ep,kwji)
       do i=1,ndex
          phi(6,i)=rk(i)
       enddo

       return
      END
```

Program 3.2. Source code of Subroutine Compute_RadialBasis

```
    SUBROUTINE Compute_RadialBasis(x,y,xv,rk,ndex,R,q,nRBF,mbasis)
!---------------------------------------------------------------------
!  Compute radial basis after added linear polynomial.
!  Input: x,y,xv[],ndex, r,q,nRBF,mbasis
!         nRBF: 1: MQ;  2: Exp;  3: TPS
!  Output--rk[10,ndex+mbasis]
!         From 1 to 10 denotes
!         r,drx,dry,drdxx,drdxy,drdyy
!         drdxxx,drdxxy, drdxyy, drdyyy
!---------------------------------------------------------------------

       implicit real*8 (a-h,o-z)
       dimension xv(2,ndex),rk(10,ndex+mbasis)
       do i=1,ndex+mbasis
          do j=1,10
             rk(j,i)=0
          enddo
       enddo

       do 10 i=1,ndex
          rr2=(x-xv(1,i))**2+(y-xv(2,i))**2
          if(nRBF.eq.1) then      ! MQ
             rk(1,i)=(rr2+R**2)**q
             rk(2,i)=2.*q*(rr2+R**2)**(q-1.)*(x-xv(1,i))
             rk(3,i)=2.*q*(rr2+R**2)**(q-1.)*(y-xv(2,i))
             rk(4,i)=2.*q*(rr2+R**2)**(q-1.)+4.*(q-1)*q* &
                     (x-xv(1,i))**2*(rr2+R**2)**(q-2)
             rk(5,i)=4.*(q-1)*q*(x-xv(1,i))*(y-xv(2,i))* &
                     (rr2+R**2)**(q-2)
             rk(6,i)=2.*q*(rr2+R**2)**(q-1.)+4.*q*(q-1)* &
                     (y-xv(2,i))**2*(rr2+R**2)**(q-2)
          endif

          if(nRBF.eq.2) then  ! EXP
             rk(1,i)=exp(-q*rr2)
             rk(2,i)=-2.*q*exp(-q*rr2)*(x-xv(1,i))
             rk(3,i)=-2.*q*exp(-q*rr2)*(y-xv(2,i))
             rk(4,i)=-2*q*exp(-q*(rr2))+4*q*q*(x-xv(1,i))**2*exp(-q*rr2)
             rk(5,i)=4.*q*q*exp(-q*(rr2))*(y-xv(2,i))*(x-xv(1,i))
             rk(6,i)=-2*q*exp(-q*(rr2))+4*q*q*(y-xv(2,i))**2*exp(-q*rr2)
          endif

          if(nRBF.eq.3) then   ! TSP
             rk(1,i)=(rr2)**(0.5*q)
```

```
            rk(2,i)=q*(x-xv(1,i))*(rr2)**(0.5*q-1)
            rk(3,i)=q*(y-xv(2,i))*(rr2)**(0.5*q-1)
            rk(4,i)=q*(rr2)**(0.5*q-1)+2.*q*(0.5*q-1)*(x-xv(1,i))**2*(rr2) &
                    **(0.5*q-2)
            rk(5,i)=q*(0.5*q-1)*(x-xv(1,i))*(y-xv(2,i))*(rr2)**(0.5*q-2)
            rk(6,i)=q*(rr2)**(0.5*q-1)+2.*q*(0.5*q-1)*(y-xv(2,i))**2*(rr2) &
                    **(0.5*q-2)
         endif
10      continue

      if(mbasis.gt.0) then
         rk(1,ndex+1)=1.
         rk(1,ndex+2)=x
         rk(1,ndex+3)=y
         rk(2,ndex+2)=1.
         rk(3,ndex+3)=1.
      endif
      return
      END
```

Program 3.3. Source code of Subroutine GaussEqSolver_sym

```
    Subroutine GaussEqSolver_Sym(n,ma,a,b,ep,kwji)
!---------------------------------------------------------------------
!  Solve sysmmetric linear equation ax=b by using Gauss elimination.
!  If kwji=1, no solution;if kwji=0,has solution
!  Input--n,ma,a(ma,n),b(n),ep,
!  Output--b,kwji
!---------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      dimension a(ma,n),b(n),m(n+1)
      do 10 i=1,n
10    m(i)=i
      do 120 k=1,n
         p=0.0
         do 20 i=k,n
            do 20 j=k,n
               if(dabs(a(i,j)).gt.dabs(p)) then
                  p=a(i,j)
                  io=i
                  jo=j
               endif
20       continue
         if(dabs(p)-ep) 30,30,35
30       kwji=1
         return
35       continue
         if(jo.eq.k) go to 45
         do 40 i=1,n
            t=a(i,jo)
            a(i,jo)=a(i,k)
            a(i,k)=t
40       continue
         j=m(k)
         m(k)=m(jo)
         m(jo)=j
45       if(io.eq.k) go to 55
         do 50 j=k,n
            t=a(io,j)
            a(io,j)=a(k,j)
            a(k,j)=t
50       continue
         t=b(io)
         b(io)=b(k)
         b(k)=t
55       p=1./p
         in=n-1
         if(k.eq.n) go to 65
```

```
          do 60 j=k,in
60        a(k,j+1)=a(k,j+1)*p
65        b(k)=b(k)*p
          if(k.eq.n) go to 120
          do 80 i=k,in
             do 70 j=k,in
70              a(i+1,j+1)=a(i+1,j+1)-a(i+1,k)*a(k,j+1)
80              b(i+1)=b(i+1)-a(i+1,k)*b(k)
120       continue
          do 130 i1=2,n
             i=n+1-i1
             do 130 j=i,in
130       b(i)=b(i)-a(i,j+1)*b(j+1)
          do 140 k=1,n
             i=m(k)
140       a(1,i)=b(k)
          do 150 k=1,n
150       b(k)=a(1,k)
          kwji=0
    return
    END
```

Program 3.4. Source code of main program of using RPIM subroutine

```
!---------------------------------------------------------------------
! Main program for testing the RPIM shape function.
! Call Subroutine RPIM_ShapeFunc_2D( ).
! 25 field nodes (5X5) in domain [x,y]--[-1,1;-1,1].
! 61X61 sampling points are used to plot 2-D RPIM shape Func.
!---------------------------------------------------------------------
    implicit real*8 (a-h,o-z)
    parameter(nx=2,numnode=25)
    dimension x(nx,numnode),nv(numnode), gpos(nx),phi(10,numnode)
    open(2,file='phi.dat') ! Output file
    write(2,50)
    nRBF=1                      ! Using MQ-RBF
    q=0.5
    alfc=2.0
    dc=0.5
    mbasis=0                    ! Number of basis
    xlength=2.
    ylength=2.
    ndivx=4
    ndivy=4
    xstep=xlength/ndivx
    ystep=ylength/ndivy
    nn=0
    do i=1,ndivx+1
       do j=1,ndivy+1
          nn=nn+1
          x(1,nn)=-1.+(i-1)*xstep   !x coordinates of field nodes
          x(2,nn)=-1.+(j-1)*ystep   !y coordinates of field nodes
       enddo
    enddo
    do i=1,numnode
       nv(i)=i                      ! Field nodes in support domain
    enddo
    ndex=25
    nce=numnode/2+1     ! the node in the centre of 25 field nodes
    nm=61
    ste=2./(nm-1)
    do ix=1,nm
       do il1=1,numnode
          do il2=1,10
             phi(il1,il2)=0
          enddo
       enddo
       gpos(1)=-1.+ste*(ix-1)
```

```
       do j=1,nm
          gpos(2)=-1.+ste*(j-1)
          if((abs(gpos(1)).le.1).and.(abs(gpos(2)).le.1)) then
             call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,&
                                    alfc,dc,q,nRBF, mbasis)
          else
          endif
! ***********Output RPIM shape function
          if((abs(gpos(1)).le.1.d-8).and.(abs(gpos(2)).le.1.d-8)) then
             do kk=1,ndex
                nd=nv(kk)
                write(2,100)nv(kk),x(1,nd),x(2,nd),phi(1,kk), &
                phi(2,kk),phi(3,kk),phi(4,kk),phi(6,kk)
             enddo
          endif
       enddo
    enddo
    write(2,150)
50  format(1x,'Node', 5x,'x', 7x,'y', 8x,'Phi', 6x,'dPhidx', &
         5x,'dPhidy', 4x, 'dPhidxx', 4x,'dPhidyy',/,80('-'))
100 format(1x,i4, 2f8.3,5f11.5)
150 format(80('-'))
    END
```

Program 3.5. Source code of Subroutine Weight_W1

```
   SUBROUTINE Weight_W1(dif,nv,ds,w,nx,ndex,numnode)
!----------------------------------------------------------------------
! Cubic spline weight function
! input--dif,nv,ds,nx,ndex,numnode
! output--w
! from 1 to 10 column of w denotes w,dwdx,dwdy,dwdxx,dwdxy,dwdyy
!----------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       dimension dif(nx,ndex),nv(numnode),ds(nx,numnode),w(ndex,10)
       ep=1.0e-20
       do 10 i=1,ndex
          nn=nv(i)
          difx=dif(1,i)
          dify=dif(2,i)
          if(dabs(difx).le.ep) then
             drdx=0.
          else
             drdx=(difx/dabs(difx))/ds(1,nn)
          end if
          if (dabs(dify).le.ep) then
             drdy=0.
          else
             drdy=(dify/dabs(dify))/ds(2,nn)
          end if
          rx=dabs(dif(1,i))/ds(1,nn)
          ry=dabs(dif(2,i))/ds(2,nn)
        if(rx.gt.0.5) then
          wx=(4./3.)-4.*rx+4.*rx*rx-(4./3.)*rx*rx*rx
          dwxdx=(-4.+8.*rx-4.*rx*rx)*drdx
          dwxdxx=(8.-8.*rx)*drdx*drdx
          dwxdxxx=(-8.)*drdx*drdx*drdx
        else if(rx.le.0.5) then
          wx=(2./3.)-4.*rx*rx+4.*rx*rx*rx
          dwxdx=(-8.*rx+12.*rx*rx)*drdx
          dwxdxx=(-8.+24.*rx)*drdx*drdx
          dwxdxxx=(24.)*drdx*drdx*drdx
        end if
```

```
        if(ry.gt.0.5) then
          wy=(4./3.)-4.*ry+4.*ry*ry-(4./3.)*ry*ry*ry
          dwydy=(-4.+8.*ry-4.*ry*ry)*drdy
          dwydyy=(8.-8.*ry)*drdy*drdy
          dwydyyy=(-8.)*drdy*drdy*drdy
        else if(ry.le.0.5) then
          wy=(2./3.)-4.*ry*ry+4.*ry*ry*ry
          dwydy=(-8.*ry+12.*ry*ry)*drdy
          dwydyy=(-8.+24.*ry)*drdy*drdy
          dwydyyy=(24.)*drdy*drdy*drdy
        end if
          w(i,1)=wx*wy
          w(i,2)=wy*dwxdx
          w(i,3)=wx*dwydy
          w(i,4)=wy*dwxdxx
          w(i,5)=dwxdx*dwydy
          w(i,6)=wx*dwydyy
          w(i,7)=wy*dwxdxxx
          w(i,8)=dwxdxx*dwydy
          w(i,9)=dwxdx*dwydyy
          w(i,10)=wx*dwydyyy
10     continue
    return
    end
```

Program 3.6. Source code of Subroutine Weight_W2

```
    SUBROUTINE Weight_W2(dif,nv,ds,w,nx,ndex,numnode)
!---------------------------------------------------------------------
! Quartic spline weight function
! input--dif,nv,ds,nx,ndex,numnode
! output--w
! from 1 to 10 column of w denotes w,dwdx,dwdy,dwdxx,dwdxy,dwdyy
!---------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       dimension dif(nx,ndex),nv(numnode),ds(nx,numnode),w(ndex,10)
       ep=1.0e-20
       do 10 i=1,ndex
          nn=nv(i)
          difx=dif(1,i)
          dify=dif(2,i)
          if(dabs(difx).le.ep) then
             drdx=0.
          else
             drdx=(difx/dabs(difx))/ds(1,nn)
          end if
          if (dabs(dify).le.ep) then
             drdy=0.
          else
             drdy=(dify/dabs(dify))/ds(2,nn)
          end if
          rx=dabs(dif(1,i))/ds(1,nn)
          ry=dabs(dif(2,i))/ds(2,nn)
          wx=1.-6.*rx*rx+8.*rx*rx*rx-3.*rx*rx*rx*rx
          dwxdx=(-12.*rx+24.*rx*rx-12.*rx*rx*rx)*drdx
          dwxdxx=(-12.+48.*rx-36.*rx*rx)/(ds(1,nn)*ds(1,nn))
          dwxdxxx=(48.-72*rx)*drdx**3
          wy=1.-6.*ry*ry+8.*ry*ry*ry-3.*ry*ry*ry*ry
          dwydy=(-12.*ry+24.*ry*ry-12.*ry*ry*ry)*drdy
          dwydyy=(-12.+48.*ry-36.*ry*ry)/(ds(2,nn)*ds(2,nn))
          dwydyyy=(48.-72*ry)*drdy**3
          w(i,1)=wx*wy
          w(i,2)=wy*dwxdx
          w(i,3)=wx*dwydy
```

```
          w(i,4)=wy*dwxdxx
          w(i,5)=dwxdx*dwydy
          w(i,6)=wx*dwydyy
          w(i,7)=wy*dwxdxxx
          w(i,8)=dwxdxx*dwydy
          w(i,9)=dwxdx*dwydyy
          w(i,10)=wx*dwydyyy
10     continue
    return
    end
```

Program 3.7. Source code of Subroutine Compute_Basis

```
    SUBROUTINE Compute_Basis(gpos,gp,nx,mm)
!-------------------------------------------------------------------
! Compute basis functions and their derivatives
! Input-gpos,nx,mm
! Output-gp
! From 1 to 10 columns of gp: p,dpdx,dpdy,dpdxx,dpdxy,dpdyy,
!                             dpdxxx,dpdxxy,dpdxyy,dpdyyy
!-------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       dimension gpos(nx),gp(10,mm)
       do i=1,mm
          do j=1,10
             gp(i,j)=0.0
          enddo
       enddo
       gp(1,1)=1.0
       gp(1,2)=gpos(1)
       gp(1,3)=gpos(2)
       gp(1,4)=gpos(1)*gpos(1)
       gp(1,5)=gpos(1)*gpos(2)
       gp(1,6)=gpos(2)*gpos(2)
       gp(2,2)=1.0
       gp(2,4)=2.0*gpos(1)
       gp(2,5)=gpos(2)
       gp(3,3)=1.0
       gp(3,5)=gpos(1)
       gp(3,6)=2.0*gpos(2)
       gp(4,4)=2.0
       gp(5,5)=1.0
       gp(6,6)=2.0
    return
    end
```

Program 3.8. Source code of Subroutine Compute_AB

```
    SUBROUTINE Compute_AB(gpos,x,nv,ds,a,b,nx,numnode,ndex,mm)
!-------------------------------------------------------------------
! Compute A matrix and B matrix and their derivatives
! input--gpos,x,nv,dm,nx,numnode,ndex,mm
! output--a,b
! From 1 to 10 of the third dimension of a denotes
!    a,dax,day,daxx,daxy,dayy, dadxxx,dadxxy,dadxyy,dadyyy
! From 1 to 10 of the third dimension of b denotes
!    b,dbx,dby,dbxx,dbxy,dbyy,dbdxxx,dbdxxy,dbdxyy,dbdyyy
!-------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
```

```
       dimension gpos(nx),x(nx,numnode),nv(numnode),ds(nx,numnode)
       dimension a(mm,mm,10),b(mm,ndex,10)
       dimension xv(nx,ndex),dif(nx,ndex),w(ndex,10),p(6,ndex),pp(mm,mm)

       do i=1,ndex
          nn=nv(i)
          xv(1,i)=x(1,nn)
          xv(2,i)=x(2,nn)
          p(1,i)=1.0
          p(2,i)=xv(1,i)
          p(3,i)=xv(2,i)
          p(4,i)=xv(1,i)*xv(1,i)
          p(5,i)=xv(1,i)*xv(2,i)
          p(6,i)=xv(2,i)*xv(2,i)
          dif(1,i)=gpos(1)-xv(1,i)
          dif(2,i)=gpos(2)-xv(2,i)
       enddo
       call Weight_W1(dif,nv,ds,w,nx,ndex,numnode)
! ************* Compute b and its derivatives
       do 20 ii=1,mm
          do 20 jj=1,ndex
             do 20 kk=1,10
                b(ii,jj,kk)=p(ii,jj)*w(jj,kk)
20     continue
! ************* Compute a and its derivatives
       do 25 ie=1,mm
          do 25 je=1,mm
             do 25 ke=1,10
                a(ie,je,ke)=0.
25     continue
       do 30 iii=1,ndex
          do 40 ik=1,mm
             do 40 jk=1,mm
                pp(ik,jk)=p(ik,iii)*p(jk,iii)
40        continue
          do 50 ikk=1,mm
             do 50 jkk=1,mm
                do 50 kkk=1,10
                   a(ikk,jkk,kkk)=a(ikk,jkk,kkk)+w(iii,kkk)*pp(ikk,jkk)
50        continue
30     continue
    return
    end
```

Program 3.9. Source code of Subroutine MLS_ShapeFunc_2D( )

```
    SUBROUTINE MLS_ShapeFunc_2D(gpos,x,nv,ds,phi,nx,numnode,ndex,mm)
!-------------------------------------------------------------------
! Compute MLS shape functions and their derivatives
! Input--gpos,x,nv,ds,nx,numnode,ndex,mm
! Output--phi
! From 1 to 10 of the two dimension of phi denotes
!     phi,dphix,dphiy,dphixx,dphixy,dphiyy
!     dphidxxx,dphidxxy, dphidxyy, dphidyyy
!-------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       dimension gpos(nx),x(nx,numnode),nv(numnode)
       dimension  ds(nx,numnode),xv(nx,ndex)
       dimension gp(10,mm),gam(mm,10),a(mm,mm,10)
       dimension b(mm,ndex,10),c(mm),aa(mm,mm),phi(10,ndex)
       do i1=1,mm
          do j1=1,10
             gp(j1,i1)=0.0
```

```
         enddo
      enddo

      call Compute_Basis(gpos,gp,nx,mm)
      call Compute_AB(gpos,x,nv,ds,a,b,nx,numnode,ndex,mm)
      ep=1.0e-20
      do 10 in=1,mm
         c(in)=gp(1,in)
10    continue
      do 20 i1=1,mm
         do 20 j1=1,mm
            aa(i1,j1)=a(i1,j1,1)
20    continue
      do i1=1,mm
         do j1=1,10
            gam(i1,j1)=0.0
         enddo
      enddo

! ************* Compute gam
      call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
21    format(1x,' gam kwji=',i2)
      do 25 k1=1,mm
         gam(k1,1)=c(k1)
25    continue
! ************* Compute dgamdx
      do 30 in=1,mm
         c(in)=0.
         do 30 jn=1,mm
            c(in)=c(in)+a(in,jn,2)*gam(jn,1)
30    continue
      do 35 kn=1,mm
         c(kn)=gp(2,kn)-c(kn)
35    continue
      do 40 i1=1,mm
         do 40 j1=1,mm
            aa(i1,j1)=a(i1,j1,1)
40    continue
      call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
      do 45 k1=1,mm
         gam(k1,2)=c(k1)
45    continue
! ************* Compute dgamdy
      do 50 in=1,mm
         c(in)=0.
            do 50 jn=1,mm
               c(in)=c(in)+a(in,jn,3)*gam(jn,1)
50    continue
      do 55 kn=1,mm
         c(kn)=gp(3,kn)-c(kn)
55    continue
      do 60 i1=1,mm
         do 60 j1=1,mm
            aa(i1,j1)=a(i1,j1,1)
60    continue
      call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
      do 65 k1=1,mm
         gam(k1,3)=c(k1)
65    continue
! ************* Compute dgamdxx
      do 70 in=1,mm
         c(in)=0.
         do 70 jn=1,mm
            c(in)=c(in)+a(in,jn,4)*gam(jn,1)+2.0*a(in,jn,2)*gam(jn,2)
70    continue
      do 75 kn=1,mm
         c(kn)=gp(4,kn)-c(kn)
75    continue
      do 80 i1=1,mm
         do 80 j1=1,mm
```

```
             aa(i1,j1)=a(i1,j1,1)
80      continue
        call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
        do 85 k1=1,mm
           gam(k1,4)=c(k1)
85      continue
! ************* Compute dgamdxy
        do 90 in=1,mm
           c(in)=0.
           do 90 jn=1,mm
              c(in)=c(in)+a(in,jn,5)*gam(jn,1)+a(in,jn,2)*gam(jn,3)+ &
                    a(in,jn,3)*gam(jn,2)
90      continue
        do 95 kn=1,mm
           c(kn)=gp(5,kn)-c(kn)
95      continue
        do 100 i1=1,mm
           do 100 j1=1,mm

              aa(i1,j1)=a(i1,j1,1)
100     continue
        call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
        do 105 k1=1,mm
           gam(k1,5)=c(k1)
105     continue

! ************* Compute dgamdyy

        do 110 in=1,mm
           c(in)=0.
           do 110 jn=1,mm
              c(in)=c(in)+a(in,jn,6)*gam(jn,1)+2.0*a(in,jn,3)*gam(jn,3)
110      continue
        do 115 kn=1,mm
           c(kn)=gp(6,kn)-c(kn)
115     continue
        do 120 i1=1,mm
           do 120 j1=1,mm
              aa(i1,j1)=a(i1,j1,1)
120     continue
        call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
        do 125 k1=1,mm
           gam(k1,6)=c(k1)
125     continue

! ************* Compute dgamdxxx
        do in=1,mm
           c(in)=0.
           do jn=1,mm
              c(in)=c(in)+a(in,jn,7)*gam(jn,1)+3*a(in,jn,4)*gam(jn,2)+ &
                    3*a(in,jn,2)*gam(jn,4)
          enddo
        enddo

        do kn=1,mm
          c(kn)=gp(7,kn)-c(kn)
        enddo

        do  i1=1,mm
          do j1=1,mm
            aa(i1,j1)=a(i1,j1,1)
          enddo
        enddo

        call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
        do  k1=1,mm
           gam(k1,7)=c(k1)
        enddo

! ************* Compute dgamdxxy
```

```
      do in=1,mm
         c(in)=0.
         do jn=1,mm
            c(in)=c(in)+a(in,jn,8)*gam(jn,1)+ &
                  a(in,jn,4)*gam(jn,3)+2*a(in,jn,5)*gam(jn,2)+ &
                  2*a(in,jn,2)*gam(jn,5)+a(in,jn,3)*gam(jn,4)
         enddo
      enddo

      do kn=1,mm
         c(kn)=gp(8,kn)-c(kn)
      enddo

      do  i1=1,mm
         do  j1=1,mm
            aa(i1,j1)=a(i1,j1,1)
         enddo
      enddo

      call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
      do k1=1,mm
         gam(k1,8)=c(k1)
      enddo
! ************* Compute dgamdxyy
      do in=1,mm
         c(in)=0.
         do jn=1,mm
            c(in)=c(in)+a(in,jn,9)*gam(jn,1)+ &
                  a(in,jn,6)*gam(jn,2)+2*a(in,jn,5)*gam(jn,3)+ &
                  2*a(in,jn,3)*gam(jn,5)+a(in,jn,2)*gam(jn,6)
         enddo
      enddo

      do kn=1,mm
         c(kn)=gp(9,kn)-c(kn)
      enddo

      do  i1=1,mm
         do  j1=1,mm
            aa(i1,j1)=a(i1,j1,1)
         enddo
      enddo

      call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
      do k1=1,mm
         gam(k1,9)=c(k1)
      enddo
! ************* Compute dgamdyyy
      do in=1,mm
         c(in)=0.
         do jn=1,mm
            c(in)=c(in)+a(in,jn,10)*gam(jn,1)+ &
                  3*a(in,jn,6)*gam(jn,3)+3*a(in,jn,3)*gam(jn,6)
         enddo
      enddo

      do kn=1,mm
         c(kn)=gp(10,kn)-c(kn)
      enddo

      do  i1=1,mm
         do  j1=1,mm
            aa(i1,j1)=a(i1,j1,1)
         enddo
      enddo

      call GaussEqSolver_Sym(mm,mm,aa,c,ep,kwji)
      do k1=1,mm
         gam(k1,10)=c(k1)
      enddo
```

```
!! ************* Compute Phi and their derivatives

       do 130 iph=1,ndex
          do iiii=1,10
             phi(iiii,iph)=0.0
          enddo
          do 130 jph=1,mm
             phi(1,iph)=phi(1,iph)+gam(jph,1)*b(jph,iph,1)
             phi(2,iph)=phi(2,iph)+gam(jph,2)*b(jph,iph,1)+ &
                        gam(jph,1)*b(jph,iph,2)
             phi(3,iph)=phi(3,iph)+gam(jph,3)*b(jph,iph,1)+ &
                        gam(jph,1)*b(jph,iph,3)
             phi(4,iph)=phi(4,iph)+gam(jph,4)*b(jph,iph,1)+ &
                        2.0*gam(jph,2)*b(jph,iph,2)+gam(jph,1)*b(jph,iph,4)
             phi(5,iph)=phi(5,iph)+gam(jph,5)*b(jph,iph,1)+ &
                        gam(jph,2)*b(jph,iph,3)+gam(jph,3)*b(jph,iph,2)+ &
                        gam(jph,1)*b(jph,iph,5)
             phi(6,iph)=phi(6,iph)+gam(jph,6)*b(jph,iph,1)+ &
                        2.0*gam(jph,3)*b(jph,iph,3)+gam(jph,1)*b(jph,iph,6)
             phi(7,iph)=phi(7,iph)+gam(jph,7)*b(jph,iph,1)+ &
                        3.0*gam(jph,4)*b(jph,iph,2)+3*gam(jph,2)*b(jph,iph,4)+ &
                        gam(jph,1)*b(jph,iph,7)
             phi(8,iph)=phi(8,iph)+gam(jph,8)*b(jph,iph,1)+ &
                        2.0*gam(jph,5)*b(jph,iph,2)+2*gam(jph,2)*b(jph,iph,5)+ &
                        gam(jph,1)*b(jph,iph,8)+gam(jph,4)*b(jph,iph,3)+ &
                        gam(jph,3)*b(jph,iph,4)
             phi(9,iph)=phi(9,iph)+gam(jph,9)*b(jph,iph,1)+ &
                        2.0*gam(jph,5)*b(jph,iph,3)+2*gam(jph,3)*b(jph,iph,5)+ &
                        gam(jph,1)*b(jph,iph,9)+gam(jph,6)*b(jph,iph,2)+ &
                        gam(jph,2)*b(jph,iph,6)
             phi(10,iph)=phi(10,iph)+gam(jph,10)*b(jph,iph,1)+ &
                        3.0*gam(jph,6)*b(jph,iph,3)+3*gam(jph,3)*b(jph,iph,6)+ &
                        gam(jph,1)*b(jph,iph,10)
130    continue
    return
    end
```

Program 3.10. Source code of main program of using MLS approximation

```
!---------------------------------------------------------------------
! Main program for testing the MLS shape function.
! Call Subroutine MLS_ShapeFunc_2D( ).
! 25 field nodes (5X5) in domain [x,y]--[-1,1;-1,1].
! 61X61 interpolation points are used to plot 2-D MLS shape Func.
!---------------------------------------------------------------------
    implicit real*8 (a-h,o-z)
    parameter(nx=2,numnode=25)
    dimension x(nx,numnode),nv(numnode), gpos(nx)
    dimension phi(10,numnode),ds(nx,numnode)

    open(2,file='phi.dat') ! Output file
    write(2,50)
    mm=3                   ! Number of basis
    xlength=2.
    ylength=2.
    ndivx=4
    ndivy=4
    xstep=xlength/ndivx
    ystep=ylength/ndivy
    nn=0
    do i=1,ndivx+1
       do j=1,ndivy+1
          nn=nn+1
          x(1,nn)=-1.+(i-1)*xstep   !x coordinates of field nodes
```

```
          x(2,nn)=-1.+(j-1)*ystep   !y coordinates of field nodes
       enddo
    enddo

    do i=1,numnode
       nv(i)=i
       ds(1,i)=0.
       ds(2,i)=0.
    enddo
    ndex=25

    do  j=1,numnode
       xn=x(1,j)
       yn=x(2,j)
       rx0=abs(xn-1)
       if(rx0.lt.abs(xn+1)) rx0=abs(xn+1)
       ry0=abs(yn-1)
       if(ry0.lt.abs(yn+1)) ry0=abs(yn+1)
          ds(1,j)=rx0    ! rw for weight function (support domain)
          ds(2,j)=ry0
    enddo

    nce=numnode/2+1     ! the node in the centre of 25 field nodes
    nm=61
    ste=2./(nm-1)
    do ix=1,nm
       do il1=1,numnode
          do il2=1,10
             phi(il1,il2)=0
          enddo
       enddo
       gpos(1)=-1.+ste*(ix-1)
       do j=1,nm
          gpos(2)=-1.+ste*(j-1)
          if((abs(gpos(1)).le.1).and.(abs(gpos(2)).le.1)) then
             call MLS_ShapeFunc_2D(gpos,x,nv,ds,phi,nx,numnode,ndex,mm)
          else
          endif
! ***********Output MLS shape function
          if((abs(gpos(1)).le.1.d-8).and.(abs(gpos(2)).le.1.d-8)) then
             do kk=1,ndex
                nd=nv(kk)
                write(2,100)nv(kk),x(1,nd),x(2,nd),phi(1,kk), &
                phi(2,kk),phi(3,kk),phi(4,kk),phi(6,kk)
             enddo
          endif
       enddo
    enddo
    write(2,150)
50  format(1x,'Node', 5x,'x', 7x,'y', 8x,'Phi', 6x,'dPhidx', &
           5x,'dPhidy', 4x, 'dPhidxx', 4x,'dPhidyy',/,80('-'))
100 format(1x,i4, 2f8.3,5f11.5)
150 format(80('-'))
    end
```

Chapter 4

MESHFREE METHODS BASED ON GLOBAL WEAK-FORMS

4.1 INTRODUCTION

MFree methods based on the global weak-form (or *MFree global weak-form methods*) are usually based on the Galerkin weak-form defined over the *global* problem domain, using *locally supported* MFree shape functions discussed in Chapter 3.

The first MFree global weak-form method was the diffuse element method (DEM) proposed by Nayroles et al.(1992). In DEM, the MLS approximation proposed by Lancaster and Salkauskas (1981) for surface fitting was used to create the shape functions. The Galerkin weak-form over the global problem domain is employed to construct the discretized system equations.

In 1994, Belytschko et al. (1994a) proposed the element free Galerkin (EFG) method in their important paper, in which the MLS approximation was used in the Galerkin weak-form to establish a set of algebraic equations. In the EFG method, the problem domain is represented by a set of properly distributed nodes. The MLS approximation is used to construct shape functions based only on a group of arbitrarily distributed nodes in a local domain. A set of background cells are required to evaluate the integrals resulted from the use of the Galerkin weak-form.

Belytschko and his colleagues have reported that the EFG method is very accurate (Belytschko, et al, 1994a; 1996a), and the rate of convergence of

the EFG method obtained from numerical tests is higher than that of FEM (Belytschko, et al, 1994a). In addition, the irregularity of nodes does not affect the performance of the EFG method (Belytschko, et al, 1994a). The EFG method has been successfully applied to a large variety of problems including two-dimensional (2-D) and three-dimensional (3-D) linear and nonlinear elastic problems (Belytschko, et al, 1994a; Lu et al., 1994; Belytschko et al., 1997; Jun, 1996; Chen and Guo, 2001), fracture and crack growth problems (Belytschko, et al, 1994b; Belytschko, et al, 1995a, 1995b, 1995c; Krysl and Belytschko 1999; Lu et al., 1995), plate and shell structures (Krysl and Belytschko, 1995; 1996; GR Liu and Chen, 2000, 2001; Liu L and GR Liu et al., 2001, 2002a,b; Chen and GR Liu et al., 2001,2003;), electromagnetic field problems (Cingoski et al., 1998), piezoelectric structures (GR Liu and Dai et al., 2004, 2003), and so on. In addition, techniques of coupling EFG method with FEM (Belytschko and Organ, 1995; Hegen, 1996) and with BEM (GR Liu and Gu, 2000c, 2000d; Gu and GR Liu, 2001b; 2003a) have also been proposed. All these applications and extensions indicate that the EFG method is gradually becoming a mature and practical computational approach in the area of computational mechanics, thanks to the use of the MLS approximation to achieve stability in function approximation, and use of Galerkin procedure to provide stable and well-behaved discretized global system equations.

In developing the EFG method, the following issues have been or still are under intensive study.

1) EFG shape functions constructed using the MLS approximation lack the Kronecker delta function property. Special techniques are, therefore, needed in the implementation of essential boundary conditions. Several techniques have been developed to enforce essential boundary conditions in EFG and will be discussed in Section 4.3.

2) Global numerical integrations are needed for calculating the system matrices. Hence, a global background cell structure has to be used for these integrations, so that the method is not truly meshless. The issues in the global numerical integration of the EFG method have been studied by some researchers. Beissel and Belytschko (1996) have proposed a stabilized nodal integration procedure to avoid the use of background cells

3) The EFG method is computationally more expensive than FEM. This is because a) a set of algebraic equations has to be solved for each sampling point to construct the MLS shape functions; b) the node searching has to be performed during the construction of the MLS shape functions; c) the resultant system matrix has, in general, a

larger bandwidth due to the fact that more nodes are used in the construction of the MLS shape functions.

GR Liu and Gu (1999, 2001a) proposed the MFree point interpolation methods (PIM) based on the Galerkin weak-form. In PIM, the problem domain is represented by properly distributed nodes. The polynomial point interpolation method (PIM) is used to construct shape functions based only on a group of nodes arbitrarily distributed in a local domain. A global background cell structure is required to evaluate the integrals in the Galerkin weak-forms.

The main feature of PIM is that their shape functions possess Kronecker delta function property. Essential boundary conditions can be easily enforced as in FEM. However, in the polynomial PIM, the *moment matrix* can be singular. Hence, a two-stage matrix triangularization algorithm (MTA) is proposed to overcome this problem automatically excluding the nodes and the terms of the polynomial basis used in the formation of the moment matrix (Liu and Gu, 2003a). The MTA is a novel approach to solve the problem of the singular moment matrix in the construction of PIM shape functions. However, due to the incompatibile nature of the polynomial PIM shape functions, the PIM based on the Galerkin weak-form is not robust for irregular nodal distributions, especially when too many nodes are used in the local support domain resulting in too high order of polynomials, which leads to a too drastic variation in the PIM shape functions.

The radial point interpolation method (RPIM) (GR Liu and Gu, 2001c; Wang and GR Liu, 2000; 2002a) that uses radial basis functions (RBF) is also proposed to overcome the singularity issue. RPIM is stable and robust for arbitrary nodal distributions. Therefore, RPIM is currently used more widely than the polynomial PIM. RPIM has been successfully applied to 2D and 3D solid mechanics (GR Liu and Gu, 2001c; GR Liu and Yan et al., 2002; GR Liu and Zhang et al, 2003a), geometrically nonlinear problems (GR Liu and Dai and Lim, 2003), problems of smart materials (GR Liu and Dai et al., 2002, 2003), plate and shell structures (Liu L and GR Liu et al., 2002a; Chen, 2003), material non-linear problems in civil engineering (Wang et al., 2001b; 2002b), and so on.

Note that the shape parameters of the RBFs have to be properly selected in RPIM. In addition, the RPIM shape functions do not possess global compatibility (GR Liu, 2002; GR Liu and Gu, 2004b), which can have some effects when it is used in a global energy principle such as the Galerkin weak-form. Note that the global compatibility is not an issue when a local weak-form or a collocation procedure is used.

In this chapter, two MFree global weak-form methods, the RPIM and the EFG methods, will be presented and examined in detail. We choose to discuss RPIM first because its formulation procedure is closer to the

standard FEM procedure, and easier to comprehend. It should be noted that historically the RPIM is based on the EFG method by replacing MLS shape functions with the RPIM shape functions, and using the Galerkin weak-form.

4.2 MESHFREE RADIAL POINT INTERPOLATION METHOD

4.2.1 RPIM formulation

Consider the following standard two-dimensional problem of linear elasticity defined in the domain Ω bounded by Γ. The partial differential equation and boundary conditions for a two-dimensional solid mechanics problem have been given in Sub-section 1.2.2 and can be written in the form of

Equilibrium equation:	$\mathbf{L}^{\mathrm{T}}\boldsymbol{\sigma}+\mathbf{b}=0$	in Ω	(4.1)
Natural boundary condition:	$\boldsymbol{\sigma}\,\mathbf{n}=\overline{\mathbf{t}}$	on Γ_t	(4.2)
Essential boundary condition:	$\mathbf{u}=\overline{\mathbf{u}}$	on Γ_u	(4.3)

where

L: differential operator defined by Equation (1.25);

$\boldsymbol{\sigma}^{\mathrm{T}}=\left\{\sigma_{xx} \quad \sigma_{yy} \quad \tau_{xy}\right\}$: the stress vector;

$\mathbf{u}^{\mathrm{T}}=\{u \quad v\}$: the displacement vector;

$\mathbf{b}^{\mathrm{T}}=\{b_x \quad b_y\}$: the body force vector;

$\overline{\mathbf{t}}$: the prescribed traction on the traction (natural) boundaries;

$\overline{\mathbf{u}}$: the prescribed displacement on the displacement (essential) boundaries;

n: the vector of unit outward normal at a point on the natural boundary (see Figure 1.4).

The standard variational (weak) form of Equation (4.1) is posed as follows (see Section 1.4).

$$\int_{\Omega}(\mathbf{L}\delta\mathbf{u})^{\mathrm{T}}(\mathbf{D}\mathbf{L}\mathbf{u})\mathrm{d}\Omega-\int_{\Omega}\delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega-\int_{\Gamma_t}\delta\mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma=0 \tag{4.4}$$

where **D** is the matrix of elastic constants given in Equation (1.27) for the plane stress and Equation (1.28) for the plane strain.

Note that Equation (4.4) is a weak-form defined over the global problem domain, Ω. In order to evaluate the integrals in Equation (4.4), the global problem domain is discretized into a set of the so-called *background cells* that are not overlapping, as shown in Figure 4.1. To evaluate the integrals along the natural boundary, a set of curve (for 2D problem) background cells (no overlapping) is used.

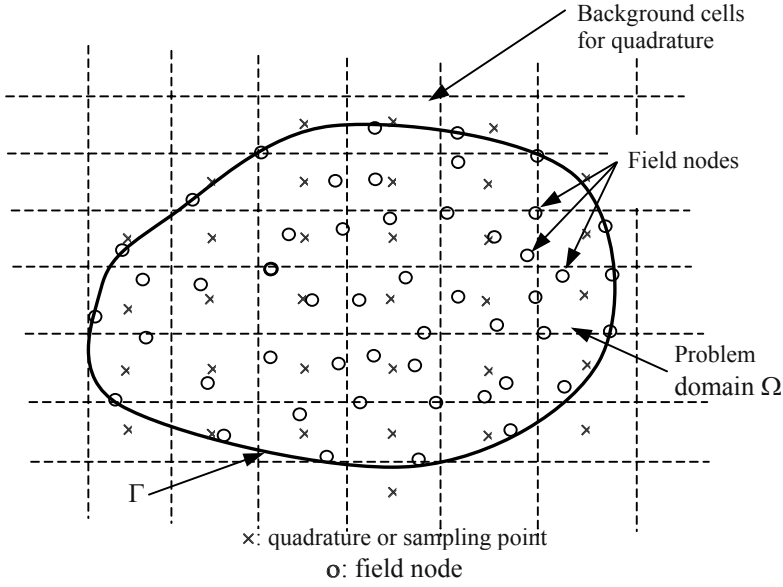


Figure 4.1. Background cells used in MFree global weak-form methods. The problem domain Ω is represented by field nodes. The background cell structure is used to evaluate the integrations in the weak-form.

The problem domain is now represented by a set of field nodes for the purpose of field variable (displacement) approximation. These nodes are numbered sequentially from 1 to *N* for the entire problem domain. The RPIM shape functions presented in Sub-section 3.2.2 are used to approximate the displacements at any point of interest using a set of nodes in a local support domain of the point.

$$\mathbf{u}^h_{(2\times 1)} = \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} = \mathbf{\Phi}_{(2\times 2n)} \mathbf{u}_{(2n\times 1)} \tag{4.5}$$

where $\mathbf{\Phi}$ is the matrix of shape functions, n is the number of nodes in the local support domain, and $\mathbf{u}$ is the vector of the displacements at the n field nodes in the support domain. In Equation (4.5), the numbers in parentheses of the subscript denote the dimensions of matrices or vectors. The same convention is used throughout this book. Equation (4.5) can also be written in the following form of *nodal* summation.

$$\mathbf{u}^h_{(2\times 1)} = \sum_I^n \begin{bmatrix} \phi_I & 0 \\ 0 & \phi_I \end{bmatrix} \begin{Bmatrix} u_I \\ v_I \end{Bmatrix} = \sum_I^n \mathbf{\Phi}_I \mathbf{u}_I \tag{4.6}$$

where $\mathbf{\Phi}_I$ is the matrix of shape functions of node I, and $\mathbf{u}_I$ is the nodal displacements.

In Equation (4.6), $\mathbf{u}^h$ is the approximated displacements of a point of interest that can be a sampling point or a quadrature point.

From Equation (4.6), we can obtain

$$\delta\mathbf{u}^h_{(2\times 1)} = \mathbf{\Phi}_{(2\times 2n)} \delta\mathbf{u}_{(2n\times 1)} = \sum_I^n \mathbf{\Phi}_I \delta\mathbf{u}_I \tag{4.7}$$

Using Equations (1.23) and (4.6), the strains can be obtained using the approximated displacements.

$$\begin{aligned}
\boldsymbol{\varepsilon}_{(3\times 1)} &= \mathbf{L}\mathbf{u}^h = \mathbf{L}_{(3\times 2)} \mathbf{\Phi}_{(2\times 2n)} \mathbf{u}_{(2n\times 1)} \\
&= \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} \\
&= \underbrace{\begin{bmatrix} \dfrac{\partial \phi_1}{\partial x} & 0 & \cdots & \dfrac{\partial \phi_n}{\partial x} & 0 \\ 0 & \dfrac{\partial \phi_1}{\partial y} & \cdots & 0 & \dfrac{\partial \phi_n}{\partial y} \\ \dfrac{\partial \phi_1}{\partial y} & \dfrac{\partial \phi_1}{\partial x} & \cdots & \dfrac{\partial \phi_n}{\partial y} & \dfrac{\partial \phi_n}{\partial x} \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix}}_{\mathbf{u}} = \mathbf{B}_{(3\times 2n)} \mathbf{u}_{(2n\times 1)} \\
&= \sum_I^n \mathbf{B}_I \mathbf{u}_I
\end{aligned} \tag{4.8}$$

where $\mathbf{B}$ is the *strain matrix* and $\mathbf{B}_I$ is the strain matrix for node I. Similarly,

$$\mathbf{L}\delta\mathbf{u}^h = \mathbf{L}_{(3\times 2)}\boldsymbol{\Phi}_{(2\times 2n)}\delta\mathbf{u}_{(2n\times 1)} = \mathbf{B}_{(3\times 2n)}\delta\mathbf{u}_{(2n\times 1)} = \sum_I^n (\mathbf{B}_I)_{(3\times 2)}(\delta\mathbf{u}_I)_{(2\times 1)} \tag{4.9}$$

We can now obtain the stress vector using the constitutive equations for the material at the point in the problem domain.

$$\boldsymbol{\sigma} = \mathbf{D}\,\boldsymbol{\varepsilon} = \mathbf{D}_{(3\times 3)}\mathbf{B}_{(3\times 2n)}\mathbf{u}_{(2n\times 1)} = \sum_I^n \mathbf{D}_{(3\times 3)}(\mathbf{B}_I)_{(3\times 2)}(\mathbf{u}_I)_{(2\times 1)} \tag{4.10}$$

Substituting Equations (4.8) and (4.9) into the first term of Equation (4.4), we have

$$\begin{aligned}\int_\Omega (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}}(\mathbf{D}\mathbf{L}\mathbf{u})\mathrm{d}\Omega &= \int_\Omega (\sum_I^n \mathbf{B}_I\delta\mathbf{u}_I)^{\mathrm{T}}(\sum_J^n \mathbf{D}\mathbf{B}_J\mathbf{u}_J)\mathrm{d}\Omega \\ &= \int_\Omega \sum_I^n\sum_J^n \delta\mathbf{u}_I^{\mathrm{T}}[\mathbf{B}_I^{\mathrm{T}}\mathbf{D}\mathbf{B}_J]\mathbf{u}_J\mathrm{d}\Omega\end{aligned} \tag{4.11}$$

Note that until this stage, I and J are based on the local numbering system for the nodes in the local support domain. We can now change the numbering system from the local one to the global one that records all the field nodes in the entire domain in a unique manner from 1 to N, the total number of nodes in the problem domain[†]. Therefore, both I and J in Equation (4.11) can now vary from 1 to N. When node I and node J are not in the same local support domain, the integrand vanishes and hence the integral. With this operation, Equation (4.11) can be expressed as

$$\int_\Omega (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}}(\mathbf{D}\mathbf{L}\mathbf{u})\mathrm{d}\Omega = \int_\Omega \sum_I^N\sum_J^N \delta\mathbf{u}_I^{\mathrm{T}}[\mathbf{B}_I^{\mathrm{T}}\mathbf{D}\mathbf{B}_J]\mathbf{u}_J\mathrm{d}\Omega \tag{4.12}$$

We now move the integration inside the summations to arrive at

$$\int_\Omega (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}}(\mathbf{D}\mathbf{L}\mathbf{u})\mathrm{d}\Omega = \sum_I^N\sum_J^N \delta\mathbf{u}_I^{\mathrm{T}}(\underbrace{\int_\Omega \mathbf{B}_I^{\mathrm{T}}\mathbf{D}\mathbf{B}_J\mathrm{d}\Omega}_{\mathbf{K}_{IJ}})\mathbf{u}_J \tag{4.13}$$

where $\mathbf{K}_{IJ}$, which is a 2×2 matrix, is called the *nodal stiffness matrix* and is defined as

[†] This can be done using an index matrix that gives the relationship between the local node number and the global node number similar to that is done in the conventional finite element method.

$$\mathbf{K}_{IJ} = \int_{\Omega} (\mathbf{B}_I^{\mathrm{T}})_{2\times 3} \mathbf{D}_{3\times 3} (\mathbf{B}_J)_{3\times 2} \mathrm{d}\Omega \tag{4.14}$$

Note that when node I and node J are not in the same support domain of the same quadrature point of integration, $\mathbf{K}_{IJ}$ vanishes.

Equation (4.13) can be now expressed as

$$\int_{\Omega} (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}} (\mathbf{D}\mathbf{L}\mathbf{u}) \mathrm{d}\Omega = \sum_{I}^{N} \sum_{J}^{N} \delta\mathbf{u}_I^{\mathrm{T}} \mathbf{K}_{IJ} \mathbf{u}_J \tag{4.15}$$

Note that the summation in the right-hand-side of this equation is in fact an assembly process. To view this, we perform the following operation.

$$\begin{aligned} \sum_{I}^{N} \sum_{J}^{N} \delta\mathbf{u}_I^T \mathbf{K}_{IJ} \mathbf{u}_J &= \delta\mathbf{u}_1^T \mathbf{K}_{11} \mathbf{u}_1 + \delta\mathbf{u}_1^T \mathbf{K}_{12} \mathbf{u}_2 + \cdots + \delta\mathbf{u}_1^T \mathbf{K}_{1N} \mathbf{u}_N \\ &+ \delta\mathbf{u}_2^T \mathbf{K}_{21} \mathbf{u}_1 + \delta\mathbf{u}_2^T \mathbf{K}_{22} \mathbf{u}_2 + \cdots + \delta\mathbf{u}_2^T \mathbf{K}_{2N} \mathbf{u}_N \\ &\vdots \\ &+ \delta\mathbf{u}_N^T \mathbf{K}_{N1} \mathbf{u}_1 + \delta\mathbf{u}_N^T \mathbf{K}_{N2} \mathbf{u}_2 + \cdots + \delta\mathbf{u}_N^T \mathbf{K}_{NN} \mathbf{u}_N \\ &= \delta\mathbf{U}^T \mathbf{K}\mathbf{U} \end{aligned} \tag{4.16}$$

Finally, Equation (4.13) becomes

$$\int_{\Omega} (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}} (\mathbf{D}\mathbf{L}\mathbf{u}) \mathrm{d}\Omega = \delta\mathbf{U}^{\mathrm{T}} \mathbf{K}\mathbf{U} \tag{4.17}$$

where $\mathbf{K}$ is the *global stiffness matrix* in the form of

$$\mathbf{K}_{(2N\times 2N)} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} & \cdots & \mathbf{K}_{1N} \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \cdots & \mathbf{K}_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}_{N1} & \mathbf{K}_{N2} & \cdots & \mathbf{K}_{NN} \end{bmatrix} \tag{4.18}$$

The dimension of the matrix $\mathbf{K}$ should be $(2N)\times(2N)$, because nodal stiffness matrix $\mathbf{K}_{IJ}$ is of 2×2, and the total number of nodes in the problem domain is N.

In Equation (4.17), the vector $\mathbf{U}$ is the *global displacement* vector that collects the nodal displacements of all the nodes in the entire problem domain, which has the form of

$$\mathbf{U}_{(2N\times 1)} = \begin{Bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \end{Bmatrix} = \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_N \\ v_N \end{Bmatrix} \tag{4.19}$$

The length of vector **U** should be (2*N*).

Substituting Equation (4.6) into the second term of Equation (4.4), and using the same arguments in deriving the stiffness matrix, we have

$$\int_{\Omega}\delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega=\int_{\Omega}\left(\delta\sum_{I}^{n}\mathbf{\Phi}_{I}\mathbf{u}_{I}\right)^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega \tag{4.20}$$

Using the same arguments given below Equation (4.11), Equation (4.20) can be expressed as

$$\int_{\Omega}\delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega=\int_{\Omega}\left(\delta\sum_{I}^{N}\mathbf{\Phi}_{I}\mathbf{u}_{I}\right)^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega \tag{4.21}$$

We now move the integration inside the summations to arrive at

$$\int_{\Omega}\delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega=\sum_{I}^{N}\delta\mathbf{u}_{I}^{\mathrm{T}}\underbrace{\int_{\Omega}\mathbf{\Phi}_{I}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega}_{\mathbf{F}_{I}^{b}} \tag{4.22}$$

where $\mathbf{F}_{I}^{(b)}$ is the *nodal body force* vector that is defined as

$$\mathbf{F}_{I}^{b}=\int_{\Omega}\mathbf{\Phi}_{I}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega \tag{4.23}$$

where **b** is the body force vector.

The last summation in Equation (4.22) can be expanded and then grouped to produce of matrices as follows.

$$\begin{aligned}\sum_{I}^{N}\delta\mathbf{u}_{I}^{\mathrm{T}}\underbrace{\int_{\Omega}\mathbf{\Phi}_{I}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega}_{\mathbf{F}_{I}^{b}}&=\sum_{I}^{N}\delta\mathbf{u}_{I}^{\mathrm{T}}\mathbf{F}_{I}^{b}\\&=\delta\mathbf{u}_{1}^{\mathrm{T}}\mathbf{F}_{1}^{b}+\delta\mathbf{u}_{2}^{\mathrm{T}}\mathbf{F}_{2}^{b}+\cdots+\delta\mathbf{u}_{N}^{\mathrm{T}}\mathbf{F}_{N}^{b}\\&=\left\{\delta\mathbf{u}_{1}^{\mathrm{T}}\quad\cdots\quad\delta\mathbf{u}_{N}^{\mathrm{T}}\right\}_{(1\times2N)}\begin{Bmatrix}\mathbf{F}_{1}^{b}\\\vdots\\\mathbf{F}_{N}^{b}\end{Bmatrix}_{(2N\times1)}\\&=\delta\mathbf{U}^{\mathrm{T}}\mathbf{F}^{b}\end{aligned} \tag{4.24}$$

where $\mathbf{F}^{(b)}$ is the *global body force* vector assembled using the nodal body force vectors for all nodes in the entire problem domain, and $\mathbf{F}^{(b)}$ is defined as

$$\mathbf{F}^b = \begin{Bmatrix} \mathbf{F}_1^b \\ \vdots \\ \mathbf{F}_N^b \end{Bmatrix}_{(2N\times 1)} \tag{4.25}$$

The length of vector $\mathbf{F}^{(b)}$ should be 2*N*.

The treatment for the last term in Equation (4.4) is exactly the same as that for the second term of Equations (4.20)~(4.25), except that the body force vector is replaced by the traction vector and the integrations are replaced by the boundary integrations. Hence, we can obtained

$$\begin{aligned} \int_{\Gamma_t} \delta\mathbf{u}^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma &= \sum_I^n \delta\mathbf{u}_I^{\mathrm{T}} \underbrace{\int_{\Gamma_t} \boldsymbol{\Phi}_I^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma}_{\mathbf{F}_I^{(t)}} \\ &= \delta\mathbf{U}^{\mathrm{T}} \sum_I^N \underbrace{\int_{\Gamma_t} \boldsymbol{\Phi}_I^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma}_{\mathbf{F}_I^{(t)}} = \delta\mathbf{U}^{\mathrm{T}} \mathbf{F}^{(t)} \end{aligned} \tag{4.26}$$

where $\mathbf{F}_I^{(t)}$ is the *nodal traction force* vector

$$(\mathbf{F}_I^{(t)})_{(2\times 1)} = \int_{\Gamma_t} \boldsymbol{\Phi}_I^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma \tag{4.27}$$

In Equation (4.26), $\mathbf{F}^{(t)}$ is the *global traction force* vector assembled using the nodal traction force vectors. The length of vector $\mathbf{F}^{(t)}$ should be 2*N*.

Substituting Equations (4.17), (4.24) and (4.26) into Equation (4.4), we have

$$\delta\mathbf{U}^{\mathrm{T}}\mathbf{K}\mathbf{U} - \delta\mathbf{U}^{\mathrm{T}}\mathbf{F}^{(b)} - \delta\mathbf{U}^{\mathrm{T}}\mathbf{F}^{(t)} = \mathbf{0} \tag{4.28}$$

or

$$\delta\mathbf{U}^{\mathrm{T}}[\mathbf{K}\mathbf{U} - \mathbf{F}^{(b)} - \mathbf{F}^{(t)}] = \mathbf{0} \tag{4.29}$$

Because $\delta\mathbf{U}$ is arbitrary, the above equation can be satisfied only if

$$\mathbf{K}\mathbf{U} - \mathbf{F}^{(b)} - \mathbf{F}^{(t)} = \mathbf{0} \tag{4.30}$$

or

$$\mathbf{K}\mathbf{U} = \mathbf{F}^{(b)} + \mathbf{F}^{(t)} \tag{4.31}$$

It can be re-written as

$$\mathbf{KU} = \mathbf{F} \tag{4.32}$$

where $\mathbf{F}$ is the *global force* vector given by

$$\mathbf{F} = \mathbf{F}^{(b)} + \mathbf{F}^{(t)} \tag{4.33}$$

Equation (4.32) is the final discretized system equations for the MFree RPIM. The nodal displacements can be obtained by solving Equation (4.32) after enforcing the displacement boundary conditions that will be introduced in the following section.

After obtaining nodal displacements, the strain and stress components can be retrieved using Equations (4.8) and (4.10), respectively.

4.2.2 Numerical implementation

4.2.2.1 Numerical integration

In the above discussion, all integrations are over the global problem domain Ω and the global traction boundary Γ_t. In order to evaluate these global integrals, the problem domain is discretized into a set of background cells (see Figure 4.1). Hence, a global integration can be expressed as a summation of integrals over these cells:

$$\int_{\Omega} \mathbf{G} d\Omega = \sum_{k}^{n_c} \int_{\Omega_k} \mathbf{G} d\Omega \tag{4.34}$$

where n_c is the number of background cells, $\mathbf{G}$ represents the integrand, and Ω_k is the domain of the kth background cell.

The Gauss quadrature scheme that is commonly used in the FEM is employed to perform the integrations numerically over these cells. When n_g Gauss points are used in each background cell, Equation (4.34) becomes

$$\int_{\Omega} \mathbf{G} d\Omega = \sum_{k}^{n_c} \int_{\Omega_k} \mathbf{G} d\Omega = \sum_{k}^{n_c} \sum_{i=1}^{n_g} \widehat{w}_i \mathbf{G}(\mathbf{x}_{Qi}) \left| \mathbf{J}_{ik}^{D} \right| \tag{4.35}$$

where $\widehat{w}_i$ is the Gauss weighting factor for the ith Gauss point at $\mathbf{x}_{Qi}$, and $\mathbf{J}_{ik}^{D}$ is the Jacobian matrix for the area integration of the background cell k, at which the Gauss point $\mathbf{x}_{Qi}$ located.

Similarly, we can obtain the formulation of the curve Gauss quadrature as

$$\int_{\Gamma_t} \mathbf{G} d\Gamma = \sum_{l}^{n_{ct}} \int_{\Gamma_{tl}} \mathbf{G} d\Omega = \sum_{l}^{n_{ct}} \sum_{i=1}^{n_{gt}} \widehat{w}_i \mathbf{G}(\mathbf{x}_{Qi}) \left| \mathbf{J}_{il}^{B} \right| \tag{4.36}$$

where $\widehat{w}_i$ is the Gauss weighting factor for the ith Gauss point $\mathbf{x}_{Qi}$, $\mathbf{J}_{il}^{B}$ is the Jacobian matrix for the curve integration of the sub-boundary (a 1D curve for a 2D problem domain) l for the Gauss point at $\mathbf{x}_{Qi}$, n_{ct} is the number of the curve cells that are used to discretize boundary Γ_t, and n_{gt} is number of Gauss points used in a sub-curve.

In order to obtain numerically the nodal stiffness matrix $\mathbf{K}_{IJ}$, the formulation of the numerical quadrature for Equation (4.14) can be written as

$$\mathbf{K}_{IJ} = \sum_{k}^{n_c} \sum_{i=1}^{n_g} \underbrace{\widehat{w}_i \mathbf{B}_I^{\mathrm{T}}(\mathbf{x}_{Qi}) \mathbf{D} \mathbf{B}_J(\mathbf{x}_{Qi}) \left| \mathbf{J}_{ik}^{D} \right|}_{\mathbf{K}_{IJ}^{ik}} = \sum_{k}^{n_c} \sum_{i=1}^{n_g} (\mathbf{K}_{IJ}^{ik})_{(2\times 2)} \tag{4.37}$$

where $\mathbf{K}_{IJ}^{ik}$ is defined as

$$\mathbf{K}_{IJ}^{ik} = \widehat{w}_i \mathbf{B}_I^{\mathrm{T}}(\mathbf{x}_{Qi}) \mathbf{D} \mathbf{B}_J(\mathbf{x}_{Qi}) \left| \mathbf{J}_{ik}^{D} \right| \tag{4.38}$$

and the dimension of $\mathbf{K}_{IJ}^{ik}$ is 2×2 .

Note that Equation (4.37) means that the nodal matrix $\mathbf{K}_{IJ}$ is obtained numerically by the summation of contributions from all the quadrature points whose local support domains include both the Ith and the Jth nodes. If node I and node J are not in the local support domain for the quadrature point at $\mathbf{x}_{Qi}$, $\mathbf{K}_{IJ}^{ik}$ vanishes.

Similarly, we can obtained the nodal body force vector $\mathbf{F}_I^{(b)}$ given in Equation (4.23)

$$\mathbf{F}_I^{(b)} = \sum_{k}^{n_c} \sum_{i=1}^{n_g} \underbrace{\widehat{w}_i \mathbf{\Phi}_I^{\mathrm{T}}(\mathbf{x}_{Qi}) \mathbf{b}(\mathbf{x}_{Qi}) \left| \mathbf{J}_{ik}^{D} \right|}_{\mathbf{F}_I^{ik(b)}} = \sum_{k}^{n_c} \sum_{i=1}^{n_g} \mathbf{F}_I^{ik(b)} \tag{4.39}$$

where $\mathbf{F}_I^{ik(b)}$ is defined as

$$\mathbf{F}_I^{ik(b)} = \widehat{w}_i \mathbf{\Phi}_I^{\mathrm{T}}(\mathbf{x}_{Qi}) \mathbf{b}_I(\mathbf{x}_{Qi}) \left| \mathbf{J}_{ik}^{D} \right| \tag{4.40}$$

and the lengeth of $\mathbf{F}_I^{ik(b)}$ is 2 .

The nodal traction force vector $\mathbf{F}_I^{(t)}$ given in Equation (4.27)

$$\mathbf{F}_I^{(t)} = \sum_{l}^{n_{ct}} \sum_{i=1}^{n_{gt}} \underbrace{\widehat{w}_i \boldsymbol{\Phi}_I^{\mathrm{T}}(\mathbf{x}_{Qi}) \overline{\mathbf{t}}(\mathbf{x}_{Qi}) \left|\mathbf{J}_{il}^B\right|}_{\mathbf{F}_I^{il(t)}} = \sum_{l}^{n_{ct}} \sum_{i=1}^{n_{gt}} \mathbf{F}_I^{il(t)} \tag{4.41}$$

where $\mathbf{F}_I^{il(t)}$ is defined as

$$\mathbf{F}_I^{il(t)} = \widehat{w}_i \boldsymbol{\Phi}_I^{\mathrm{T}}(\mathbf{x}_{Qi}) \cdot \overline{\mathbf{t}}_I(\mathbf{x}_{Qi}) \left|\mathbf{J}_{il}^B\right| \tag{4.42}$$

and the lengeth of $\mathbf{F}_I^{il(t)}$ is 2 .

In the RPIM method, the matrices are assembled based on the quadrature points. Note that different quadrature points use different support domains. This means that the shape function matrix $\boldsymbol{\Phi}$ and the strain matrix $\mathbf{B}$ may be different for different quadrature points. This is different from FEM where all Gauss points in one element use the same nodes (of the same element) to perform the interpolation.

The numerical integration in an MFree global weak-form method is one of the most important numerical issues, and has been studied by many researchers (Dolbow and Belytschko, 1999; GR Liu and Yan, 1999; GR Liu, 2002). Two conclusions may be drawn from their studies.

1) The total number of quadrature points n_Q should be at least 2/3 of the total number of the unfixed field nodes, $\tilde{N}$, in the problems domain, i.e.,

$$3n_Q > N_u \approx 2\tilde{N} \quad \text{or} \quad n_Q > \frac{2}{3}\tilde{N} \text{ for 2D problems} \tag{4.43}$$

Note that this rule is a necessary, not a sufficient requirement.

2) Other aspects (e.g., accuracy and convergence) should also be considered to select a proper number of quadrature points. We have studied this issue using benchmark problems. It has been found that the sufficient requirement on the total number of quadrature points is (GR Liu, 2002)

$$n_Q = (3 \sim 9)\tilde{N} \text{ for 2D problems} \tag{4.44}$$

Note that these studies were performed for the EFG method, but the conclusions are largely applicable to RPIM.

4.2.2.2 Properties of the stiffness matrix

Since **D** is sysmmetric, we can get

$$[\mathbf{B}_I^{\mathrm{T}}\mathbf{D}\mathbf{B}_J]^{\mathrm{T}} = [\mathbf{B}_J^{\mathrm{T}}\mathbf{D}^{\mathrm{T}}\mathbf{B}_I] \tag{4.45}$$

Hence, we have

$$[\mathbf{K}_{IJ}]^{\mathrm{T}} = \mathbf{K}_{JI} \tag{4.46}$$

which means that the global stiffness matrix **K** is *symmetric*.

The global stiffness matrix **K** is assembled using the corresponding nodal matrices, and $K_{IJ} \neq 0$ only when the nodes I and J are covered by the support domain of at least one quadrature point. If nodes I and J are far apart and they do not share the same support domain of any quadrature point, $\mathbf{K}_{IJ}$ vanishes. Therefore, as long the support domain is compact and does not cover too widely the problem domain, many $\mathbf{K}_{IJ}$ will be zero, and the global stiffness matrix **K** will be *sparse*. If the nodes are properly numbered, **K** will be also *banded*.

In summary, the global stiffness matrix **K** in the MFree RPIM method is banded, symmetric and sparse.

4.2.2.3 Enforcement of essential boundary conditions

This RPIM formulation, the traction boundary conditions (see Equation (4.2)) has been naturally formulated into the discretized system equation using the Galerkin weak-form. Therefore, the traction boundary condition is often called the *natural* boundary condition. However, the displacement boundary conditions (see Equation (4.3)) are not treated in the formulation process. It is, therefore, *essential* to impose them separately before or after Equation (4.32) is established. Hence, the displacement boundary condition is termed as the *essential boundary condition*. Because RPIM shape functions possess the Kronecker delta function property, the essential boundary conditions can be easily enforced as in the FEM (see, e.g., GR Liu and Quek, 2003). The following two methods that are widely used in FEM to enforce essential boundary conditions can be used in RPIM.

a) Direct method

The ith displacement component is prescribed by setting

$$u_i = \overline{u}_i \tag{4.47}$$

Such an essential boundary condition can then be enforced directly into the system Equation (4.32) through the following modifications to the stiffness matrix and the global force vector.

The global stiffness matrix, **K**, is changed to

$$\mathbf{K}=\begin{bmatrix} K_{11} & \cdots & K_{1(i-1)} & 0 & K_{1(i+1)} & \cdots & K_{1(2N)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ K_{(i-1)1} & \cdots & K_{(i-1)(i-1)} & 0 & K_{(i-1)(i+1)} & \cdots & K_{(i-1)(2N)} \\ 0 & \vdots & 0 & 1 & 0 & \cdots & 0 \\ K_{(i+1)1} & \cdots & K_{(i+1)(i-1)} & 0 & K_{(i+1)(i+1)} & \cdots & K_{(i+1)(2N)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ K_{(2N)1} & \cdots & K_{(2N)(i-1)} & 0 & K_{(2N)(i+1)} & \cdots & K_{(2N)(2N)} \end{bmatrix} \tag{4.48}$$

The components in the global force vector are changed to

$$F_j \Rightarrow \begin{cases} \overline{u}_i & i=j \\ F_j - K_{ji}\overline{u}_i & i\neq j \end{cases} \tag{4.49}$$

Solving Equation (4.32) using the modified stiffness matrix and the force vector, we can obtain all the displacement components, and Equation (4.47) is satisfied exactly.

The direct method can exactly enforce essential boundary conditions, but changing matrices and vectors needs additional computational operations. In addition, the algorithm of the direct method is also complicated.

b) Penalty method

The penalty method is a convenient alternative for enforcing the essential boundary conditions, in which the diagonal entry, K_{ii}, in the stiffness matrix, is changed to

$$K_{ii} \Rightarrow \alpha \cdot K_{ii} \tag{4.50}$$

where α is the penalty coefficient that is the much larger number than the components of the stiffness matrix **K**. The stiffness matrix, **K**, is then changed to

$$\mathbf{K}=\begin{bmatrix} K_{11} & \cdots & K_{1(i-1)} & K_{1i} & K_{1(i+1)} & \cdots & K_{1(2N)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ K_{(i-1)1} & \cdots & K_{(i-1)(i-1)} & K_{(i-1)i} & K_{(i-1)(i+1)} & \cdots & K_{(i-1)(2N)} \\ K_{(i)1} & \vdots & K_{i(i-1)} & \alpha K_{ii} & K_{i(i+1)} & \cdots & K_{i(2N)} \\ K_{(i+1)1} & \cdots & K_{(i+1)(i-1)} & K_{(i+1)i} & K_{(i+1)(i+1)} & \cdots & K_{(i+1)(2N)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ K_{(2N)1} & \cdots & K_{(2N)(i-1)} & K_{(2N)i} & K_{(2N)(i+1)} & \cdots & K_{(2N)(2N)} \end{bmatrix} \tag{4.51}$$

In the global force vector **F**, only the component F_i is changed as follows

$$F_j \Rightarrow \begin{cases} \alpha K_{ii}\overline{u}_i & i = j \\ F_j & i \neq j \end{cases} \tag{4.52}$$

We now solve Equation (4.32) using the modified stiffness matrix and the force vector, all the displacement components can be obtained, and Equation (4.47) is satisfied approximately.

The penalty method has some advantages: there are only two changes of matrices, and the algorithm is very simple. However, the penalty method can only approximately satisfy the essential boundary conditions. In addition, the accuracy is affected by selection of the penalty coefficient; it can be difficult to select a proper penalty coefficient. Ways of choosing the penalty coefficient will be presented in Section 4.4.

4.2.2.4 Conformability of RPIM

The compatibility requirement is common to all the methods based on the global energy principles, because a possible gap or overlap (incompatibility) may affect the energy in the system and destroy the balance of the equation of the energy principle. The remedy is to use the constrained form of energy principles that takes into account the energy caused by incompatibility. Because the RPIM interpolation is not always compatible in the global domain (GR Liu and Gu, 2004a), the enforcement of the compatibility is needed on the incompatible curve Γ_c in the problem domain Ω to produce the conforming RPIM (CRPIM). The constrained variational (weak) form of CRPIM for two-dimension elasto-static problems is posed as follows using the penalty method to ensure the compatibility.

$$\begin{aligned} \int_{\Omega} (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}} (\mathbf{D}\mathbf{L}\mathbf{u}) \mathrm{d}\Omega - \int_{\Omega} \delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega - \int_{\Gamma_t} \delta\mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma \\ + \int_{\Gamma_c} \delta(\mathbf{u}^{+} - \mathbf{u}^{-})\boldsymbol{\alpha}(\mathbf{u}^{+} - \mathbf{u}^{-})\mathrm{d}\Gamma = 0 \end{aligned} \tag{4.53}$$

where $\boldsymbol{\alpha}$ is the matrix of the penalty constants, and $\mathbf{u}^{+}$ and $\mathbf{u}^{-}$ are the displacements on the two sides of the incompatible interface, Γ_c. Hence, the compatibility on the interfaces Γ_c of the neighboring integration cells is enforced by the penalty term. If the last term in the left-hand-side of Equation (4.53) is excluded, the formulation leads to the conventional non-conforming RPIM (NRPIM).

The so-called CRPIM was proposed by GR Liu and Gu (2004b), and further studies of CRPIM and NRPIM have concluded that CRPIM leads to slightly more accurate results than the NRPIM. However, the NRPIM has also been found to be convergent and lead to satisfactory results. The

NRPIM is simpler than the CRPIM. Hence, only the conventional RPIM or NRPIM has been discussed in detail in this book.

4.3 ELEMENT FREE GALERKIN METHOD

4.3.1 EFG formulation

Consider a two-dimensional problem of solid mechanics in a domain Ω bounded by Γ. The strong-form of system equation is given by Equations (4.1)~(4.3). The element-free Galerkin (EFG) method uses the moving least squares (MLS) shape functions (see Section 3.3). Because the MLS approximation lacks the Kronecker delta function property, the constrained Galerkin weak-form should be posed as follows.

$$\int_{\Omega}\delta(\mathbf{Lu})^{\mathrm{T}}\,\mathbf{D}(\mathbf{Lu})\,\mathrm{d}\Omega \;-\int_{\Omega}\delta\,\mathbf{u}^{\mathrm{T}}\mathbf{b}\,\mathrm{d}\Omega \;-\int_{\Gamma_t}\delta\,\mathbf{u}^{\mathrm{T}}\overline{\mathbf{t}}\,\mathrm{d}\Gamma - \delta\int_{\Gamma_u}\frac{1}{2}(\mathbf{u}-\overline{\mathbf{u}})^{\mathrm{T}}\,\boldsymbol{\alpha}(\mathbf{u}-\overline{\mathbf{u}})\,\mathrm{d}\Gamma = 0 \tag{4.54}$$

where $\boldsymbol{\alpha} = \lceil\, \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_k \,\rfloor$ is a diagonal matrix of penalty factors, where k=2 for 2D, and k=3 for 3D. The penalty factors α_i (i=1, 2,…,k) can be a function of coordinates and can be different from each other, but must be given. In practice, we often assign them the same constant of large positive number.

Note that in using EFG, the global compatibility of the shape function is ensured by the weight functions appropriately chosen in the MLS approximation. Hence, the constrained term to ensure compatibility is not required in the weak-form of Equation (4.54).

Using the MLS shape functions constructed using n nodes in the local support domain (see Section 3.3), we have

$$\mathbf{u}^{h}_{(2\times1)} = \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} = \boldsymbol{\Phi}_{(2\times2n)}\mathbf{u}_{(2n\times1)} \tag{4.55}$$

where $\boldsymbol{\Phi}$ is a matrix of the MLS shape functions arranged in the form of

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \tag{4.56}$$

In Equation (4.55), u_I and v_I are the parameters of displacements (not the nodal displacement, see Figure 3.16) for the Ith node, because the MLS shape functions do not have the Kronecker delta function property. It is different from RPIM, in which u_I and v_I are the nodal displacements because RPIM shape functions have the Kronecker delta function property.

Substituting the foregoing expression for all the displacement components of **u** into the weak-form Equation (4.54), and following the exact procedure detailed in Subsection 4.2 yield the following global discretized system equations of the EFG method.

$$\left[\mathbf{K} + \mathbf{K}^{\alpha} \right] \mathbf{U} = \mathbf{F} + \mathbf{F}^{\alpha} \tag{4.57}$$

where **U** is the vector of nodal parameters of displacements for all nodes in the entire problem domain, **K** is the global stiffness matrix assembled using the nodal stiffness matrices, and **F** is the global external force vector assembled using the nodal force vectors, Equations (4.23) and (4.27). The additional matrix $\mathbf{K}^{\alpha}$ is the *global penalty stiffness matrix* assembled in the same manner as for assembling **K** using the *nodal penalty stiffness matrix* defined by

$$\mathbf{K}_{IJ}^{\alpha} = \int_{\Gamma_u} \mathbf{\Phi}_I^{\mathrm{T}} \boldsymbol{\alpha} \mathbf{\Phi}_J \mathrm{d}\Gamma \tag{4.58}$$

Note that $\mathbf{K}_{IJ}^{\alpha}$ is a 2×2 matrix.

In Equation (4.57), the additional force vector $\mathbf{F}^{\alpha}$ is caused by the essential boundary conditions; it is formed in the same way as **F**, but using the nodal penalty force vector $\mathbf{F}_I^{\alpha}$ defined by

$$\mathbf{F}_I^{\alpha} = \int_{\Gamma_u} \mathbf{\Phi}_I^{\mathrm{T}} \boldsymbol{\alpha} \bar{\mathbf{u}} \mathrm{d}\Gamma \tag{4.59}$$

The length of $\mathbf{F}_I^{\alpha}$ is 2.

Similar to Equations (4.37) and (4.41), the integrations in the penalty stiffness matrix and the penalty force vector can also be obtained using the standard Gauss quadrature. Note that, in Equations (4.58) and (4.59), integrations are curve integrations for 2D problems. The integration is performed along the essential boundary, and hence matrix $\mathbf{K}^{\alpha}$ will have entries only for the nodes near the essential boundaries Γ_u, which are covered by the support domains of the Gauss quadrature points on Γ_u.

Equation (4.57) is the final discretized system equation for the EFG method with the penalty method to enforce essential boundary conditions. The Galerkin procedure makes the stiffness matrices $\mathbf{K}$ and $\mathbf{K}^{\alpha}$ symmetric. If the problem domain is sufficiently supported without rigid body movement, $[\mathbf{K}+\mathbf{K}^{\alpha}]$ will be positive definite; a standard linear algebra equation solver can be used to solve Equation (4.57) for the nodal displacement parameters.

In order to obtain the integrals in the EFG method, a global background mesh of cells is required, as in RPIM. The background mesh of cells can be independent of the field nodes that are used for the field variable approximation. In each cell, Gauss quadrature can be employed, and the number of quadrature points depends largely on the nodal density, as discussed in Sub-section 4.2.2.1.

In the present EFG formulation, the penalty method is used to enforce essential boundary conditions. The advantage of using the penalty method is that the dimension, symmetry and positive definite properties of the stiffness matrix are achieved, as long as the penalty factors chosen are positive. In addition, the symmetry and the bandness of the system matrix are preserved.

However, the penalty method has the following shortcomings.

- Essential boundary conditions are imposed only approximately, depending on the magnitude of the penalty coefficients. Theoretically, the larger the penalty coefficients, the more accurate the enforcement of the essential boundary conditions.
- It is difficult to choose a set of penalty factors that are universally applicable for all kinds of problems. One hopes to use large possible penalty factors, but too large penalty factors often give numerical problems, as we experienced in the imposition of multi-point boundary condition in the finite element methods. Trials may be needed to choose a proper penalty factor.
- The results obtained are generally less accurate than those obtained from the method of Lagrange multipliers (to be discussed in the following sub-section).

Despite these disadvantages, the penalty method is widely used.

4.3.2 Lagrange multiplier method for essential boundary conditions

The penalty method provides an efficient way to implement essential boundary conditions, and is used by many researchers e.g., Zhu and Atluri

(1998). Several other strategies have also been developed for alleviating its defects, such as, the Lagrange multiplier method (Belytschko et al., 1994a), the method using the modified variational principle (Lu et al., 1994), the method coupling with the finite elements (Krongauz and Belytschko,1996), the orthogonal transform technique (Atluri et al., 1999b), the constrained MLS method (Yang, 1999), and so on. The Lagrange multiplier method is introduced in this section.

The Lagrange multiplier method was used to enforce the essential boundary condition in the EFG method by Belytschko et al. (1994a). The functional related to the essential boundary condition, Equation (4.3), is written in an integral form using the Lagrange multiplier $\boldsymbol{\lambda}$:

$$\int_{\Gamma_u} \boldsymbol{\lambda}^{\mathrm{T}} (\mathbf{u} - \overline{\mathbf{u}}) \mathrm{d}\Gamma \tag{4.60}$$

The weak-form Equation (4.54) can then be re-written as

$$\begin{aligned} \int_{\Omega} (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}} (\mathbf{D}\mathbf{L}\mathbf{u}) \mathrm{d}\Omega - \int_{\Omega} \delta\mathbf{u}^{\mathrm{T}} \mathbf{b} \mathrm{d}\Omega - \int_{\Gamma_t} \delta\mathbf{u}^{\mathrm{T}} \overline{\mathbf{t}} \mathrm{d}\Gamma \\ - \int_{\Gamma_u} \delta\boldsymbol{\lambda}^{\mathrm{T}} \left(\mathbf{u} - \overline{\mathbf{u}}\right) \mathrm{d}\Gamma - \int_{\Gamma_u} \delta\mathbf{u}^{\mathrm{T}} \boldsymbol{\lambda} \mathrm{d}\Gamma = 0 \end{aligned} \tag{4.61}$$

The last two terms in Equation (4.61) are produced by the method of Lagrange multipliers for handling essential boundary conditions for cases when $\mathbf{u} - \overline{\mathbf{u}} \neq 0$ that violates the condition of Equation (4.3). The Lagrange multipliers $\boldsymbol{\lambda}$ can be viewed as *smart forces* that force $\mathbf{u} - \overline{\mathbf{u}} = 0$.

In order to obtain the discretized formulation, the Lagrange multipliers $\boldsymbol{\lambda}$ in Equation (4.61), which are unknown functions of the coordinates, need to be interpolated using their nodal values and shape functions for nodes on the essential boundaries.

$$\boldsymbol{\lambda}^h = \begin{Bmatrix} \lambda_u \\ \lambda_v \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & \cdots & N_{n_\lambda} & 0 \\ 0 & N_1 & \cdots & 0 & N_{n_\lambda} \end{bmatrix} \begin{Bmatrix} \lambda_{u1} \\ \lambda_{v1} \\ \vdots \\ \lambda_{un_\lambda} \\ \lambda_{vn_\lambda} \end{Bmatrix} = \mathbf{N}(s)_{(2\times 2n_\lambda)} \boldsymbol{\lambda}_{(2n_\lambda \times 1)} \tag{4.62}$$

where n_λ is the number of nodes used for this interpolation, N_I is the shape function for the Ith node on the essential boundary, s is the arc-length along the essential boundary, $\boldsymbol{\lambda}$ is the vector of the nodal Lagrange multipliers of field nodes on the essential boundary. Equation (4.62) can also be written in the following *nodal* matrix form.

$$\boldsymbol{\lambda}_{(2\times1)} = \sum_{I}^{n_\lambda} \begin{bmatrix} N_I & 0 \\ 0 & N_I \end{bmatrix} \begin{Bmatrix} \lambda_{u_I} \\ \lambda_{v_I} \end{Bmatrix} = \sum_{I}^{n} \mathbf{N}_I \boldsymbol{\lambda}_I \tag{4.63}$$

where $\mathbf{N}_I$ is the matrix of shape functions for node I on the essential boundary.

In Equations (4.62) and (4.63), the shape function $N_I(s)$ can be the Lagrange interpolants used in the conventional FEM. The Lagrange interpolant of order n can be given in the general form of

$$N_k^n(s) = \frac{(s-s_0)(s-s_1)\cdots(s-s_{k-1})(s-s_{k+1})\cdots(s-s_n)}{(s_k-s_0)(s_k-s_1)\cdots(s_k-s_{k-1})(s_k-s_{k+1})\cdots(s_k-s_n)} \tag{4.64}$$

If we choose to use the first order Lagrange interpolant (the linear interpolation), we have n=1 and the Lagrange interpolants at point s=s_0 and s=s_1 becomes

$$N_0(s) = \frac{(s-s_1)}{(s_0-s_1)}, \quad N_1(s) = \frac{(s-s_0)}{(s_1-s_0)} \tag{4.65}$$

In a simple case, the essential boundaries are discretized using line segments. The Lagrange multiplier at s is interpolated using two nodes at the two ends of this line segments.

Equation (4.62) gives the variation of the Lagrange multiplier as

$$\delta\boldsymbol{\lambda}^h = \mathbf{N}\delta\boldsymbol{\lambda} \tag{4.66}$$

Hence, Equations (4.55) and (4.66) give the fourth term in Equation (4.61):

$$\begin{aligned}
&\int_{\Gamma_u} \delta\boldsymbol{\lambda}^{\mathrm{T}} \left(\mathbf{u} - \overline{\mathbf{u}}\right) \mathrm{d}\Gamma \\
&= \int_{\Gamma_u} \delta \left(\sum_{I=1}^{n_\lambda} \mathbf{N}_I \boldsymbol{\lambda}_I \right)^{\mathrm{T}} \sum_{J=1}^{n} \boldsymbol{\Phi}_J \mathbf{u}_J \mathrm{d}\Gamma - \int_{\Gamma_u} \delta \left(\sum_{I=1}^{n_\lambda} \mathbf{N}_I \boldsymbol{\lambda}_I \right)^{\mathrm{T}} \overline{\mathbf{u}} \\
&= \sum_{I=1}^{n_\lambda} \sum_{J=1}^{n} \delta\boldsymbol{\lambda}_I^{\mathrm{T}} \underbrace{\int_{\Gamma_u} \mathbf{N}_I^{\mathrm{T}} \boldsymbol{\Phi}_J \mathrm{d}\Gamma}_{-\mathbf{G}_{IJ}^{\mathrm{T}}} \mathbf{u}_J - \sum_{I=1}^{n_\lambda} \delta\boldsymbol{\lambda}_I^{\mathrm{T}} \underbrace{\int_{\Gamma_u} \mathbf{N}_I^{\mathrm{T}} \overline{\mathbf{u}} \mathrm{d}\Gamma}_{-\mathbf{q}_I} \\
&= -\sum_{I=1}^{n_{\lambda t}} \sum_{J=1}^{N} \delta\boldsymbol{\lambda}_I^{\mathrm{T}} \mathbf{G}_{IJ}^{\mathrm{T}} \mathbf{u}_J + \sum_{I=1}^{n_{\lambda t}} \delta\boldsymbol{\lambda}_I^{\mathrm{T}} \mathbf{q}_I \\
&= \delta\boldsymbol{\Lambda}^{\mathrm{T}} \left(-\mathbf{G}^{\mathrm{T}}{}_{(2n_{\lambda t}\times 2N)} \mathbf{U}_{s(2n_{\lambda t}\times 1)} + \mathbf{Q}_{(2n_{\lambda t}\times 1)} \right)
\end{aligned} \tag{4.67}$$

where $\boldsymbol{\Lambda}$ is a vector that collects the nodal Lagrange multipliers for all field nodes on essential boundaries, $n_{\lambda t}$ is the total number of nodes on the essential boundary, and the nodal matrix $\mathbf{G}_{IJ}$ is defined as

$$\mathbf{G}_{IJ}^{\mathrm{T}} = -\int_{\Gamma_u} \mathbf{N}_I^{\mathrm{T}} \mathbf{\Phi}_J \mathrm{d}\Gamma \tag{4.68}$$

which has the dimension 2×2. In Equation (4.67), $\mathbf{q}_I$ is a vector defined as,

$$\mathbf{q}_I = -\int_{\Gamma_u} \mathbf{N}_I^{\mathrm{T}} \bar{\mathbf{u}} \mathrm{d}\Gamma \tag{4.69}$$

In Equation (4.67), $\mathbf{G}$ is the global matrix formed by assembling $\mathbf{G}_{IJ}$ defined in Equation (4.68), and $\mathbf{Q}$ is the global vector formed by assembling $\mathbf{q}_I$ defined in Equation (4.69).

Similarly, the last term in Equation (4.61) becomes

$$\begin{aligned}
\int_{\Gamma_u} \left(\mathbf{\Phi}\delta\mathbf{u}\right)^{\mathrm{T}} \boldsymbol{\lambda} \mathrm{d}\Gamma &= \int_{\Gamma_u} \left(\sum_{I}^{n} \mathbf{\Phi}_I \delta\mathbf{u}_I \right)^{\mathrm{T}} \left(\sum_{J}^{n_\lambda} \mathbf{N}_J \boldsymbol{\lambda}_J \right) \mathrm{d}\Gamma \\
&= \sum_{I=1}^{n_\lambda} \sum_{J=1}^{n} \delta\mathbf{u}_I^{\mathrm{T}} \underbrace{\int_{\Gamma_u} \mathbf{\Phi}_I^{\mathrm{T}} \mathbf{N}_J \mathrm{d}\Gamma}_{-\mathbf{G}_{IJ}} \boldsymbol{\lambda}_J \\
&= -\sum_{I=1}^{n_{\lambda I}} \sum_{J=1}^{N} \delta\mathbf{u}_I^{\mathrm{T}} \mathbf{G}_{IJ} \boldsymbol{\lambda}_J \\
&= -\delta\mathbf{U}_S^{\mathrm{T}} \mathbf{G} \boldsymbol{\Lambda}
\end{aligned} \tag{4.70}$$

As in Equations (4.37) and (4.41), the integrations in the nodal matrix $\mathbf{G}_{IJ}$ and the nodal vector $\mathbf{q}_I$ can also be obtained using the standard Gauss quadrature scheme.

Substituting Equations (4.67) and (4.70) into Equation (4.61), we obtain

$$\delta\mathbf{U}^T [\mathbf{KU} - \mathbf{F}] + \delta\boldsymbol{\Lambda}^{\mathrm{T}} (\mathbf{G}^{\mathrm{T}}\mathbf{U} - \mathbf{Q}) + \delta\mathbf{U}^{\mathrm{T}} \mathbf{G}\boldsymbol{\Lambda} = 0 \tag{4.71}$$

or

$$\delta\mathbf{U}^T [\mathbf{KU} + \mathbf{G}\boldsymbol{\Lambda} - \mathbf{F}] + \delta\boldsymbol{\Lambda}^{\mathrm{T}} (\mathbf{G}^{\mathrm{T}}\mathbf{U} - \mathbf{Q}) = 0 \tag{4.72}$$

where $\mathbf{K}$ is the global stiffness matrix and $\mathbf{F}$ is the global force vector, both of which have been discussed in Sub-section 4.3.1.

Because both $\delta\mathbf{U}$ and $\delta\boldsymbol{\Lambda}$ are arbitrary, this equation can be satisfied only if

$$\begin{cases} \mathbf{KU} + \mathbf{G}\boldsymbol{\Lambda} - \mathbf{F} = 0 \\ \mathbf{G}^{\mathrm{T}}\mathbf{U} - \mathbf{Q} = 0 \end{cases} \tag{4.73}$$

The above two equations can be written in the following matrix form of

$$\begin{bmatrix} \mathbf{K} & \mathbf{G} \\ \mathbf{G}^{\mathrm{T}} & \mathbf{0} \end{bmatrix}_{(2N+2n_{\lambda t})\times(2N+2n_{\lambda t})} \begin{Bmatrix} \mathbf{U} \\ \mathbf{\Lambda} \end{Bmatrix}_{(2N+2n_{\lambda t})\times 1} = \begin{Bmatrix} \mathbf{F} \\ \mathbf{Q} \end{Bmatrix}_{(2N+2n_{\lambda t})\times 1} \tag{4.74}$$

Equation (4.74) is the final discretized system equations for the EFG method using the Lagrange multiplier method. Solving Equation (4.74) gives the results of nodal parameters of the displacements for this problem, and the displacements at any point including at the field nodes in the problem domain can be obtained from Equation (4.55).

The Lagrange multiplier method is accurate in imposing the essential boundary conditions. However, it will increase the number of variables by $\mathbf{\Lambda}$ and the dimension of the system matrix. Depending on the number of the nodes on the essential boundaries, the solution efficiency can be drastically reduced. It also leads to an un-banded and non-positive definite stiffness matrix, which reduces the efficiency significantly in solving the discretized equations. Note that the enlarged system matrix is still symmetric.

4.4 SOURCE CODE

In this section, a computer source code, MFree_Global.f90, of these two MFree global weak-form methods, RPIM and EFG, is provided. This code is developed in FORTRAN 90 for easy comprehension. Combined with subroutines RPIM_ShapeFunc_2D and MLS_ ShapeFunc_2D given in Chapter 3, this source code performs computations with either the RPIM or the EFG method.

4.4.1 Implementation issues

4.4.1.1 Support domain and the influence domain

In the construction of meshfree shape functions, one of the most important issues is to determine the local support domain mentioned in Sub-section 3.1.2. The concept of the influence domain is also used in the MFree methods to construct the shape functions.

The *influence domain* is defined as a domain for a field node that it has an influence upon. The centre of the influence domain is the field node. In contrary, the support domain is the area chosen for the meshfree interpolation for a point of interest at $\mathbf{x}$ (which is often a quadrature point $\mathbf{x}_Q$). The centre of the support domain is usually a quadrature point that can also

be a field node. Figure 4.2(a) clearly shows the difference between an influence domain and a support domain.

The influence domain, as shown in Figure 4.2(b), is used in the following manner for selecting nodes for interpolation. To construct the MFree shape function for a point of interest, a field node will be involved in the shape function construction for this point when this point is in the influence domain of this field node. In other words, if the influence domain of a field node covers the point of interest, this field node will take part in the construction of shape functions for this point. Using the influence domain to replace the support domain has several advantages.

- The influence domain works well for domains with irregularly distributed nodes.
- The influence domain is defined for every field node in the problem domain, and it can be different from node to node to represent the area of influence of the node. Since the dimension of the influence domain can be different from node to node, some nodes can have more influence than others, and to prevent unbalanced nodal distribution for constructing shape functions.
- Because the number of field nodes is usually much less than the number of quadrature points, there are fewer influence domains than support domains. This makes the procedure computationally more efficient.

For these reasons, the influence domain is used in this book in the development of computer code.

The influence domain for a field node can be arbitrary in shape, and its dimensions of the influence domain can be determined using a similar procedure described in Chapter 3. For a two-dimensional domain and when a rectangular influence domain is used, the size of the influence domain is determined by d_{ix} and d_{iy} in the x and y directions, respectively, i.e.

$$\begin{cases} d_{ix} = \alpha_{ix} d_{cx} \\ d_{iy} = \alpha_{iy} d_{cy} \end{cases} \tag{4.75}$$

where d_{cx} and d_{cy} are, respectively, the nodal spacing in the x and y directions, have been defined in Sub-section 3.1.2, and α_{ix} and α_{iy} are the dimensionless sizes of the influence domain in x and y directions, respectively. They control the actual sizes of the influence domain in relation to the nodal spacing. If α_{ix}=2.5, for example, the size of the influence domain in the x-direction is 2.5 times the nodal spacing.

Note that selecting nodes for the interpolation/approximation can be time consuming for large scale problems, and hence special algorithm, such as the

bucket algorithm (GR Liu, 2002) and the tree algorithm (see, e.g., GR Liu and Liu, 2003) should be used.

4.4.1.2 Background cells

To perform the numerical integrations in the MFree global weak-form method, the global background cells, as shown in Figure 4.1 and Figure 4.2, are needed.

The background cells can be rectangular or triangular for a two-dimensional domain. Triangular background cells are well suited to problems with complex geometry. For simplicity, the rectangular background cells are, however, used in the book.

4.4.1.3 Method to enforce essential boundary conditions

The methods to enforce essential boundary conditions in the EFG method have been discussed in Sub-sections 4.3.1 and 4.3.2. The penalty method is used for the EFG method in the attached code.

Because the RPIM shape functions possess the Kronecker delta function property, the essential boundary conditions can be enforced directly and accurately without any additional treatment. For uniformity, the penalty method that has been presented in Sub-section 4.2.2.3 is used in the RPIM to enforce the special nodal displacements.

One major issue in using the penalty method is how to properly choose the penalty coefficient. Based on the practice in FEM, the penalty coefficient α can be determined by

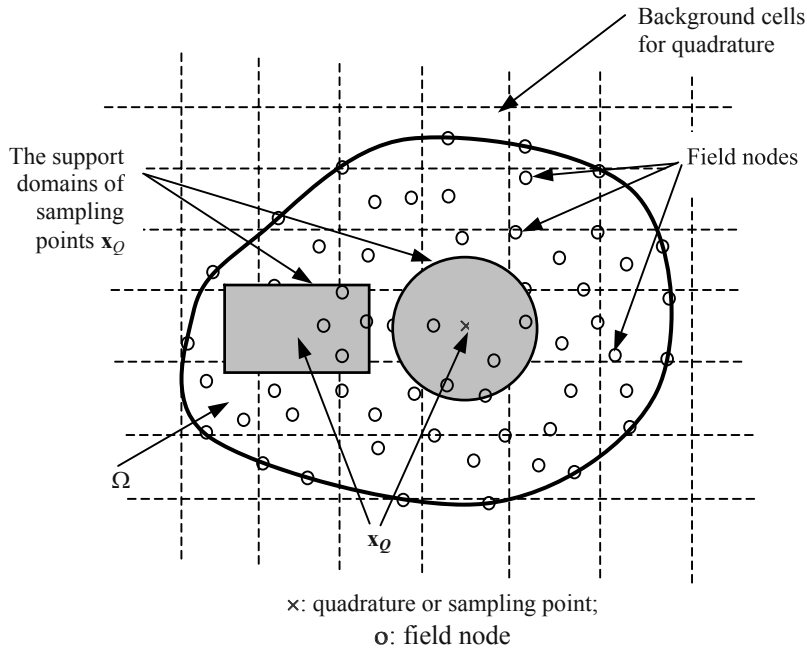
$$\alpha = 10^4 \sim 10^8 \times (K_{II})_{\max} \tag{4.76}$$

where $(K_{II})_{\max}$ is the maximum diagonal element of the global stiffness matrix.

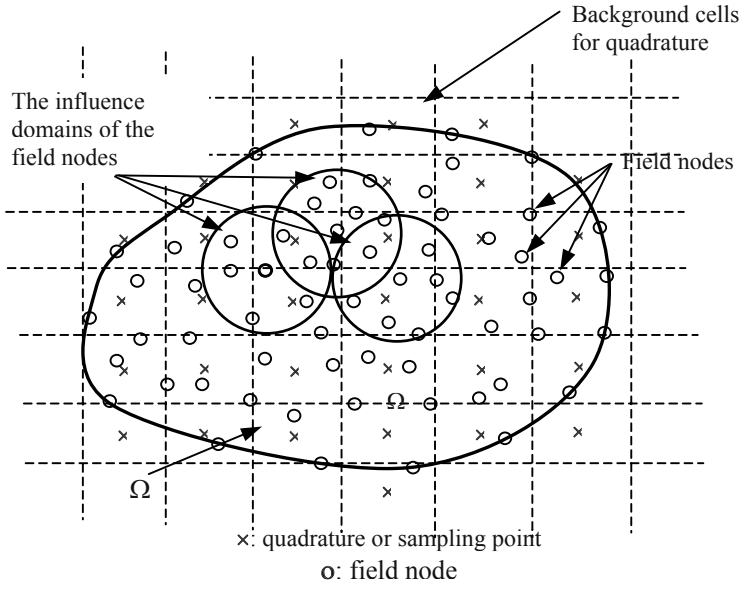
4.4.1.4 Shape parameters used in RBFs

In the RPIM method, the radial basis functions are used to construct MFree shape functions. In the subroutine of RPIM_ShapeFunc_2D, the Multi-quadrics (MQ) RBF, Gaussian (EXP) RBF, and Thin Plate Spline (TPS) RBF are used. For simplicity, only results of MQ-RBF are discussed here. Results for other RBFs can be obtained similarly.

In the MQ-RBF, there are two shape parameters: α_c and q (see Sub-section 3.2.2). Choices of these two shape parameters will affect the performance of the RPIM. The parameters are studied by numerical examinations because there are still no successful rigorous methods to determine theoretically their best values.



(a) The centre of the support domain is a quadrature or sampling point



(b) The centre of the influence domain is the field node

Figure 4.2. The background cells, the support domain, and influence domains used in the MFree global weak-form methods.

4.4.2 Program description and data structures

The flowchart of the source code, MFree_Global.f90 is shown in Figure 4.3. The procedure of an analysis using MFree methods is as follows.

- The geometry of the problem domain is created and a set of field nodes is generated to represent the problem domain.
- The global background cells are used for numerical integrations.
- The system matrices are assembled through two loops. The outer loop is for all the cells of the background mesh, and the inner loop is for all the Gauss quadrature points in a cell.
- The boundary conditions are enforced.
- The system equation is solved using the standard Gaussian elimination equation solvers.
- The post-processing is performed to analyze the final results (displacements and stresses) of the problem considered.

The procedure is similar to that in the conventional FEM. The head files and main program of MFree_Global.f90 are listed in Program 4.1~Program 4.3, respectively.

The main program of the MFree_Global.f90 calls several subroutines. The macro flowchart for the program is presented in Figure 4.4. The functions performed by these subroutines are listed in Appendix 4.1.

1) Programs for the RPIM and EFG

The attached programs call the subroutine RPIM_ShapeFunc_2D for the construction of RPIM shape functions. It can be easily changed to the program of the EFG method by calling the subroutine MLS_ShapeFunc_2D instead. Both subroutines, RPIM_ShapeFunc_2D and MLS_ShapeFunc_2D, have been given in Chapter 3. It should be noted that RPIM_ShapeFunc_2D is not only called in the main program of MFree_Global.f90 but also in some other subroutines. Hence, to perform the computation using the EFG, all the calls for RPIM_ShapeFunc_2D should be replaced.

2) Major variables

There are some major variables used in the main program and subroutines. These variables are listed in Appendix 4.2; they can be largely classified as follows:

- Variables for describing the problem, for example, the material constants, coordinates of field nodes, boundary conditions, background cells, and so on;
- Variables for computing system matrices, for example, the Gauss points, influence domains, shape parameters, penalty coefficients, shape functions and its derivatives, and so on;

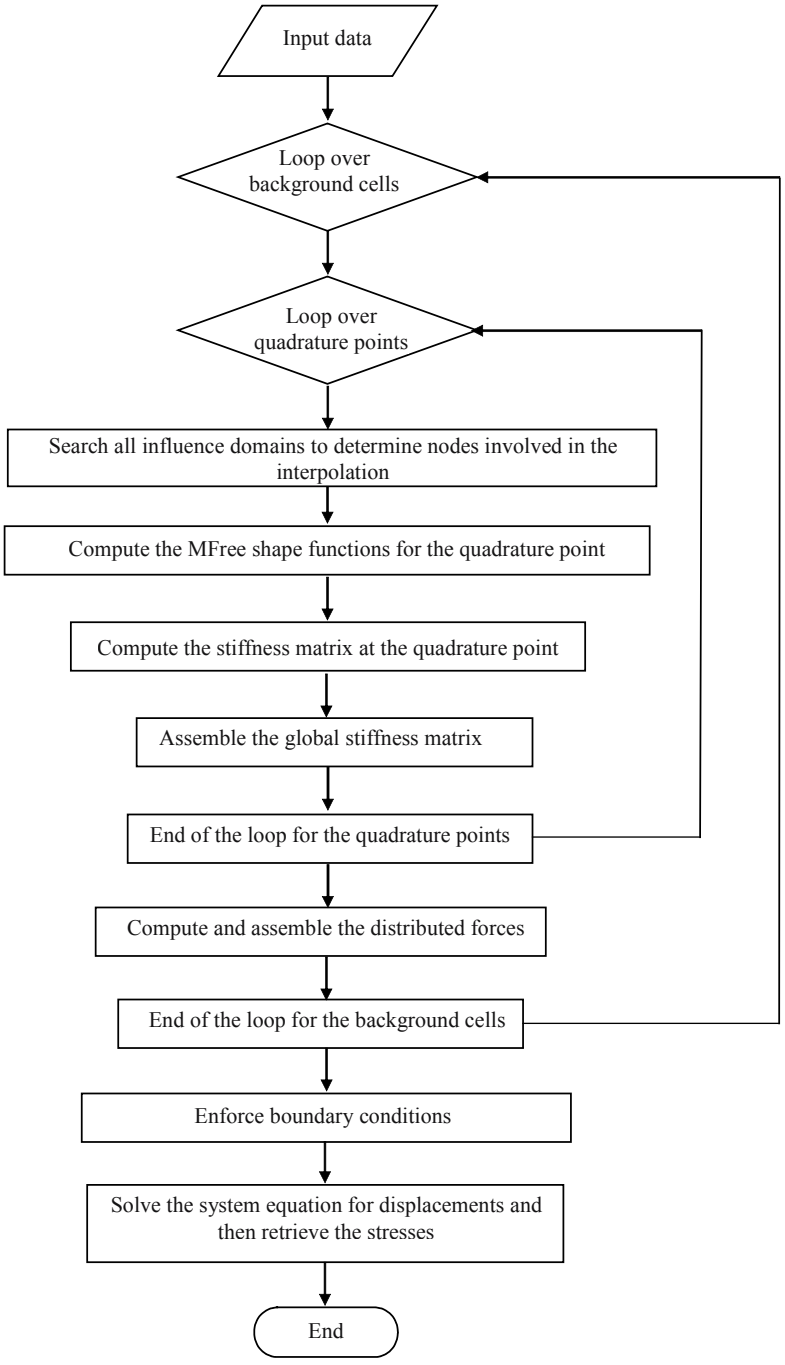


Figure 4.3. Flowchart of the program of MFree_Global.f90

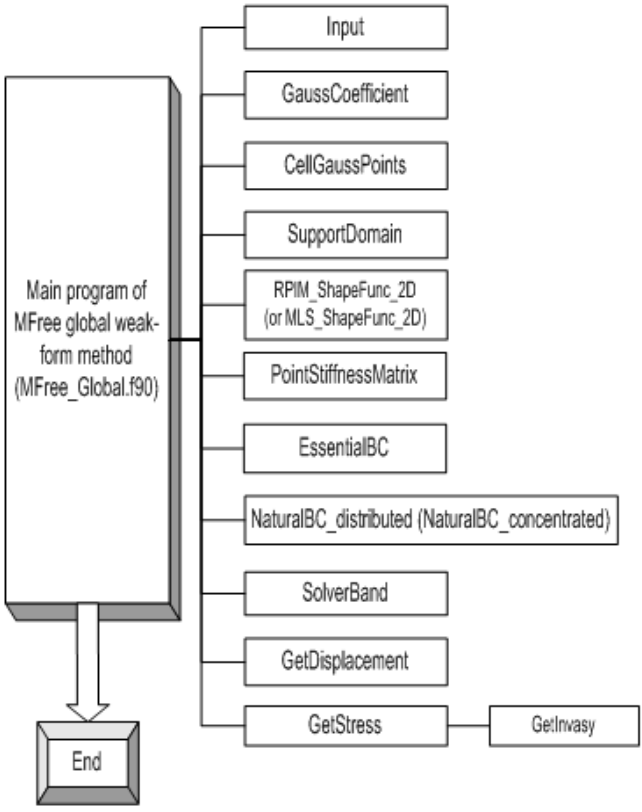


Figure 4.4. Macro flowchart of MFree_Global.f90

- Variables for system matrices and vectors, for example, the global stiffness matrix, the global force vector, and so on;
- Variables related to the solutions, for example, nodal displacements, nodal stresses, error in the energy norm, and son on.

As these global variables will be used in main program and subroutines, they will not be explained again in the descriptions for the following subroutines.

3) Subroutine *Input*

Source code location: Program 4.4.

Function: This subroutine is to input data from external file. In this subroutine, the stress-strain matrix, **D**, is also computed.

4) Subroutine *GaussCoefficient*

Source code location: Program 4.5.

Dummy arguments: Appendix 4.3.
Function: This subroutine is to set all coefficients of standard Gauss quadrature.

5) Subroutines *CellGaussPoints*

Source code location: Program 4.6.
Dummy arguments: Appendix 4.4.
Function: This subroutine is to set the Gauss points in a background cell and to calculate the Jacobian values at the Gauss points. In the present program, quadrilateral background cells are used. The background cells for other shapes (e.g. triangular and circular) can also be used. Readers can modify this subroutine slightly for other shapes of background cells.

6) Subroutine *SupportDomain*

Source code location: Program 4.7.
Dummy arguments: Appendix 4.5.
Function: This subroutine is to determine the support domain for an interpolation point for the construction of MFree shape functions. The influence domains are used in this book(Sub-section 4.4.1.1). In the beginning of the computation (in the main program), an influence domain is assigned to each field node. The nodes involved in the interpolation are then found through checking all influence domains for all field nodes. If the interpolation point is located in the influence domain of a field node, the field node will be recorded and used in the interpolation for the construction of shape functions. Note that rectangular influence domains are used in this code.

7) Subroutine *PointStiffnessMatrix*

Source code location: Program 4.8.
Dummy arguments: Appendix 4.6.
Function: This subroutine is to compute the stiffness matrix of a quadrature point using Equation (4.37).

8) Subroutine *EssentialBC*

Source code location: Program 4.9.
Dummy arguments: Appendix 4.7.
Function: This subroutine is to enforce essential boundary conditions. In the present program, the penalty method, which has been discussed in Sub-sections 4.2.2.3 and 4.3.1, is used.

9) Subroutines *NaturalBC_concentrated* and *NaturalBC_distributed*

Source code location: Program 4.10 and Program 4.11.

Dummy arguments: Appendix 4.8 and Appendix 4.9.
Function: These two subroutines are used, respectively, to implement concentrated and distributed natural boundary conditions. Readers can easily modify it for other types of natural boundary conditions. In the subroutine NaturalBC_distributed, the distributed natural boundary conditions used in Section 4.5 (Equation (4.84)) are used to compute the nodal force vector using Equations (4.23) and (4.27). The global force vector is obtained by assembling all nodal vectors.

10) Subroutine *SolverBand*

Source code location: Program 4.12.
Dummy arguments: Appendix 4.10.
Function: This subroutine is to solve the linear algebraic system equation with an asymmetric banded matrix (e.g., Xu, 1995). In fact, the stiffness matrix in an MFree global weak-form method is symmetric (see Sub-section 4.2.2). However, an asymmetric banded stiffness matrix has to be used in the Chapter 5. To avoid listing too many standard routines that are available in standard libraries, only the equation solver for an asymmetric banded matrix is presented in this book. Readers can replace this solver by simply calling other more effective solvers in the computer system for symmetric matrices.

Note that for easy comprehension of the program, the one-dimensional storage technique that is also commonly available is not used in the present program. The global stiffness matrix stored in a 2D array is formed in exactly the same way as shown in the formulation, and the 2D stiffness matrix is fed into the subroutine of the equation solver. In this subroutine, the 1D stored banded matrix is first obtained from the original matrix. The standard equation solver using the Gaussian elimination is used to obtain the results. Readers can replace this solver with other more powerful solvers, once the procedure is understood.

11) Subroutine *GetDisplacement*

Source code location: Program 4.13.
Dummy arguments: Appendix 4.11.
Function: This subroutine is to compute the actual displacements for any point (including field nodes) of interest.

If only the field nodes are considered, this subroutine is useful only in the EFG method. As discussed in Chapter 3, the MLS approximation does not pass through the nodal function values. Hence, **U** that solved from Equation

(4.57) are only the nodal parameters for displacements. In order to get the actual displacements at any point (including the field nodes) in the problem domain, we need to use the MLS approximation again, i.e.

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}^h_{(2\times 1)}(\mathbf{x}) = \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} \tag{4.77}$$

where $\mathbf{u}(\mathbf{x})$ is the displacement vector of a point $\mathbf{x}$, ϕ_i is the MLS shape functions, u_i is the nodal parameters obtained from Equation (4.57). The presented subroutine computes the final nodal displacements for all the field nodes.

This subroutine is unnecessary for the RPIM method to compute the displacements for field nodes. Because the RPIM shape functions have the Kronecker delta function property, **U** obtained from Equation (4.32) gives already the actual nodal displacements. However, this subroutine is necessary to obtain the displacements at a point that is not a field node.

12) Subroutine *GetStress*

Source code location: Program 4.14.
Dummy arguments: Appendix 4.12.
Function: This subroutine is to compute stress components for the point of interest using Equation (4.10).

For the error analysis, we define the following energy norm as an error indicator, as the accuracy in strains or stresses is much more critical than that in the displacements.

$$e_e = \sqrt{\frac{1}{2}\int_{\Omega} (\boldsymbol{\varepsilon}^{\text{Num}} - \boldsymbol{\varepsilon}^{\text{Exact}})^{\text{T}} \mathbf{D} (\boldsymbol{\varepsilon}^{\text{Num}} - \boldsymbol{\varepsilon}^{\text{Exact}}) \text{d}\Omega} \tag{4.78}$$

where $\boldsymbol{\varepsilon}^{\text{Num}}$ and $\boldsymbol{\varepsilon}^{\text{Exact}}$ are strain vectors obtained by the numerical method and the analytical method, respectively. In the presented subroutine, stress components at all Gauss points and field nodes are computed.

In the subroutine GetStress, a subroutine to perform the inversion of a matrix is used. The subroutine GetInvasy is presented in Program 4.15. In this subroutine, the Gauss-Jordan method is adopted.

4.5 EXAMPLE FOR TWO-DIMENSIONAL SOLIDS – A CANTILEVER BEAM

Numerical studies are conducted for a cantilever beam that is often used for benchmarking numerical methods because the analytic solution for this problem is known. The studies for this example have following purposes:

a) To demonstrate the standard analysis procedure using MFree global weak-form methods;
b) To show the usage of the present programs of RPIM and EFG;
c) To study the effects of shape parameters of RPIM;
d) To investigate the effects of the size of support (influence) domain;
e) To examine the numerically the convergence of RPIM and EFG;
f) To study the efficiency of RPIM and EFG;

To provide a quantitative analysis, a cantilever beam subjected to a parabolic traction at the free end as shown in Figure 4.5 is considered. The beam has a unit thickness (t=1.0) and a plane stress problem is considered. The exact solution of this problem is available and listed as follows (Timoshenko and Goodier, 1970).

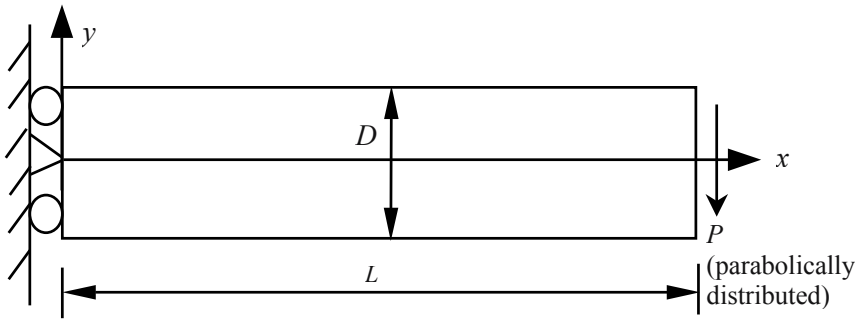


Figure 4.5. Cantilever beam subjected to a parabolic traction at the free end.

- The displacement in the x direction is given by:

$$u(x,y) = -\frac{Py}{6EI}\left[(6L-3x)x + (2+\nu)\left(y^2 - \frac{D^2}{4}\right)\right] \tag{4.79}$$

where the moment of inertia I , for a beam with rectangular cross-section and unit thickness is given by

$$I = \frac{D^3}{12} \tag{4.80}$$

- The displacement in the y direction:

$$v(x,y) = \frac{P}{6EI}\left[3\nu y^2(L-x) + (4+5\nu)\frac{D^2 x}{4} + (3L-x)x^2\right] \tag{4.81}$$

- The normal stress on the cross-section of the beam

$$\sigma_{xx}(x,y) = -\frac{P(L-x)y}{I} \tag{4.82}$$

- The normal stress in the y direction

$$\sigma_{yy} = 0 \tag{4.83}$$

- The shear stress on the cross-section of the beam

$$\tau_{xy}(x,y) = \frac{P}{2I}\left[\frac{D^2}{4} - y^2\right] \tag{4.84}$$

In this book, the units used are the standard international (SI) units unless specially mentioned. In this example, the parameters for this cantilever beam are

Loading (integration of the distributed traction): $P = -1000$

Young's modulus: $E = 3\times10^7$

Poisson's ratio: $\nu = 0.3$

The height of the beam: $D = 12$

The length of the beam: $L = 48$

The thickness of the beam: unit.

On the right boundary (x=L), the applied external traction force is computed from the analytical formula Equation (4.84). The force is distributed in the form of a parabola on the cross-section at the right end of the beam

$$t_{xy}\Big|_{x=L} = \frac{P}{2I}\left[\frac{D^2}{4} - y^2\right] \tag{4.85}$$

At the left boundary (x=0), the essential boundary conditions are given using the analytic formulae Equations (4.79) and (4.81). i.e.,

$$u\Big|_{x=0} = -\frac{P(2+\nu)}{6EI}\left[y^2 - \frac{D^2}{4}\right] \tag{4.86}$$

$$v\big|_{x=0} = \frac{P\nu L}{2EI} y^2 \tag{4.87}$$

4.5.1 Using MFree_Global.f90

In order to illustrate the present code, MFree_Global.f90, the above mentioned two-dimensional beam is analyzed following the steps given below:

Step 1: Preparation of the input data

The problem considered should be modelled in this step, which includes:
(1) Defining the geometry of the problem domain;
(2) Creating field nodes to represent the problem domain;
(3) Creating background cells for the numerical integration;
(4) Setting essential boundary conditions;
(5) Determining parameters, such as the number of Gauss points, the size of influence domains, shape parameters of RPIM, penalty coefficients, and so on.

This step prepares the input data file. For the cantilever beam problem, the problem domain is simple. Hence, the geometry data file can be easily obtained. For a complex practical problem, a pre-processor may be needed to generate the input data file (e.g. field nodes, background cells, and so on). MFree2D© (introduced in Section 4.8) has a convenient pre-processor: MFreePro that can be used for a generating the geometry data for complex 2D domain.

An example of the input data file is shown in Appendix 4.13. The domain of the beam is represented by regularly distributed 175 (25×7) field nodes as plotted in Figure 4.6. A total of 40 (10×4) regularly rectangular background cells are used for the numerical integrations. Note that the background cells are independent of the field nodes.

This data file contains largely three parts.

- The parameters of problem description.
- Data related field nodes and background cells.
- Definition of the boundary conditions.

For this beam problem, the exact boundary conditions are the essential boundary conditions on the left end obtained using Equations (4.86) and (4.87), and the natural boundary conditions on the right end of this beam obtained using Equation (4.85). There is no concentrated nodal force in this example.

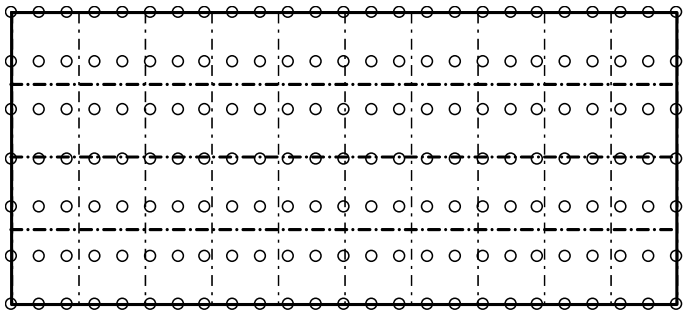


Figure 4.6. Nodal arrangement and the background cells for the cantilever beam. A total of 175 (25×7) regular field nodes and 40 (10×4) background cells are used.

Step 2: Execution of the program.

The output results of RPIM and EFG are listed in Appendix 4.14~ Appendix 4.17. The error in the energy norm given in Equation (4.78) is also presented.

Step 3: Analysis of the output data.

This step can be performed using a post-processor like MFree Post (GR Liu, 2002). Since this example problem is simple, and the output date file is small, any other commercial program, such as Matlab, MS-Excel, etc., can be used to produce the drawing of the results.

Results obtained using the RPIM method are plotted in Figure 4.7~Figure 4.9. The deflection of the beam is plotted in Figure 4.7 and Figure 4.8. For comparison, the analytical results of displacements computed using Equations (4.79) and (4.81) are also plotted in the same figure. There is good agreement between the RPIM method results and the analytical results. The results of stress, σ_{xx} , and shear stress, τ_{xy} , are plotted in Figure 4.9. Compared with the analytical results, the RPIM method produces very good results even for stresses.

Results of the EFG method are plotted in Figure 4.10 and Figure 4.11. The deflection of the beam is plotted in Figure 4.10. For comparison, the analytical results of displacements given by Equation (4.79) and (4.81) are also plotted in the same figure. The results of stress, σ_{xx} , and shear stress, τ_{xy} , are plotted in Figure 4.11. Compared with the analytical results, the EFG method has also produced very accurate stresses.

Two models with nodal distributions of 189 regular nodes and 189 irregular nodes shown in Figure 4.12 are used to test the present code. Stresses σ_{xx} and τ_{xy} are first obtained using the RPIM method and plotted in Figure 4.13. Stresses σ_{xx} and τ_{xy} are also obtained using the EFG method and plotted in Figure 4.14; the nodal irregularity has little effect on the results, and this is true for both the RPIM method and the EFG method.

For comparison, the conventional FEM results using bi-linear elements are computed and results are plotted in Figure 4.15 and Figure 4.16. For the regular nodal distribution of 189 nodes (160 bi-linear FEM elements), FEM obtains less accurate but still acceptable results. However, for the irregular nodal distribution of 189 nodes, the FEM results are very bad. This example clearly demonstrates the advantage of MFree methods over the conventional FEM on the robustness of using irregular field nodes in computing the stresses.

Note that, in the conventional FEM, stresses at the field nodes are obtained by simply averaging the nodal stresses of the surrounding elements. Better stress results can, of course, be obtained by interpolation of the stresses at the Gauss points or the so-called super-convergent points.

Note also that the performance of an MFree method is usually affected by the parameters. In the following sections, the effects of some important parameters used in both RPIM and EFG methods are studied using the present code.

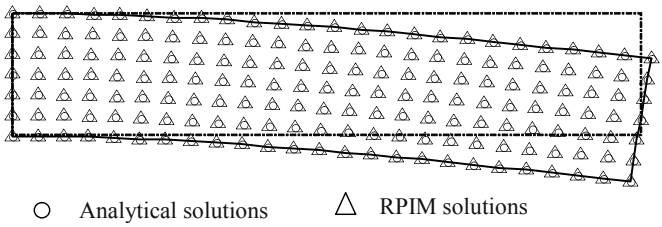


Figure 4.7. Deflection of the beam obtained using RPIM and 175 field nodes. The MQ-RBF is used in RPIM and the parameters used are α_c=1.0, q=1.03 and $\alpha_i = 3.0$. The linear polynomial terms are added in the RPIM-MQ. Note that the displacements plotted are magnified by 500 times.

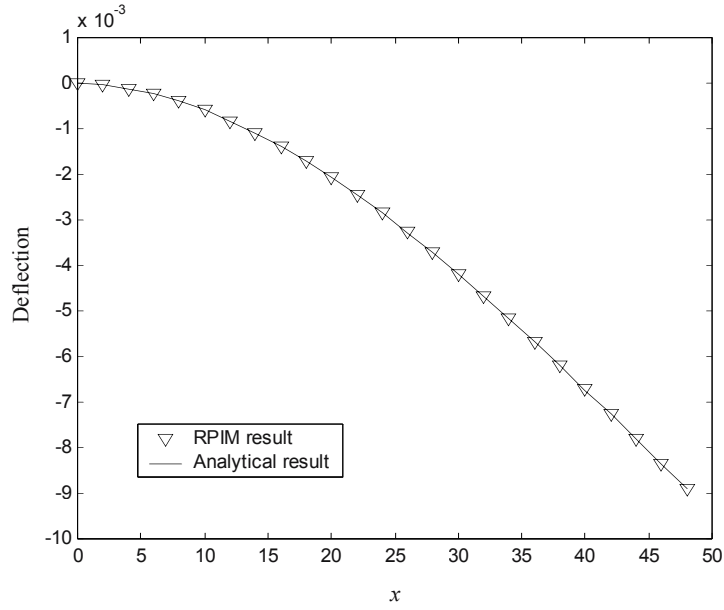


Figure 4.8. Deflections *v* along the central axis at $y = 0$ of the beam obtained using RPIM-MQ and 175 field nodes.

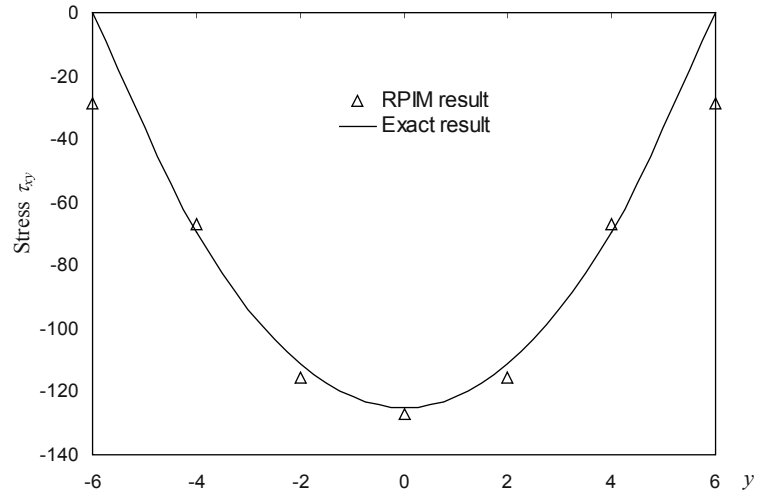


Figure 4.9. Shear stress distributions on the cross-section of the beam at $x = L/2$ obtained using RPIM and 175 field nodes. The MQ-RBF is used in the RPIM and the parameters used are $\alpha_c = 1.0,\ q = 1.03,$ and $\alpha_i = 3.0$. The linear polynomial terms are added in RPIM-MQ.

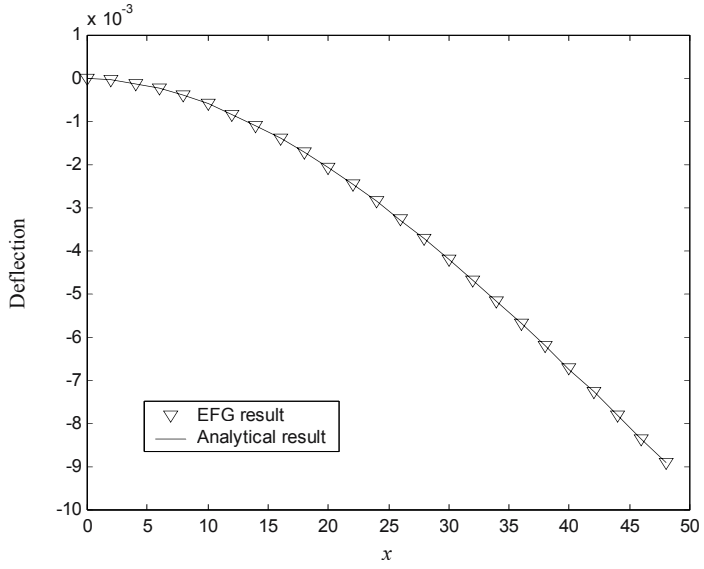


Figure 4.10. Deflections *v* along the central axis at $y = 0$ of the beam obtained using EFG and 175 field nodes. The parameter used is $\alpha_i = 3.0$. The linear polynomial basis is used in MLS.

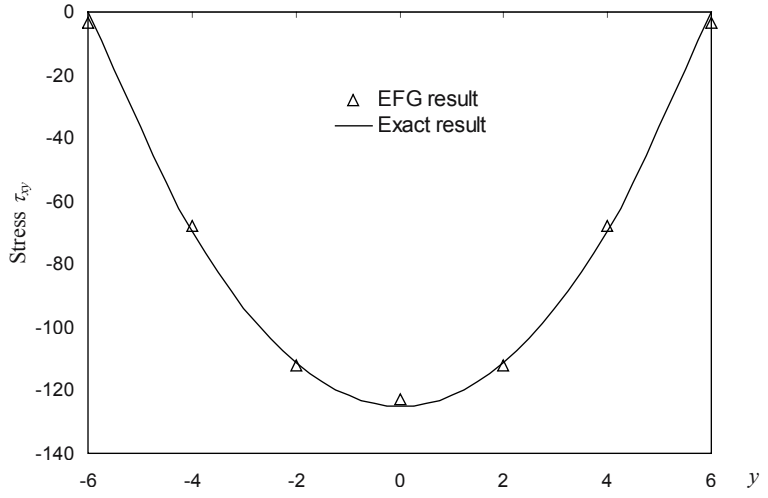
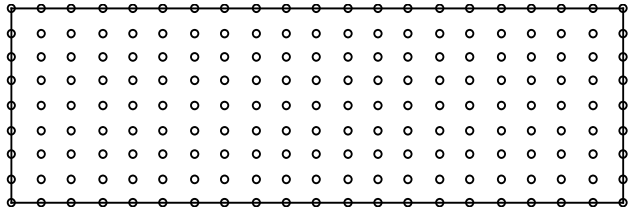
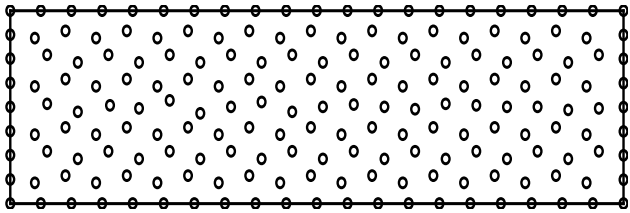


Figure 4.11. Shear stress distributions on the cross-section of the beam at $x = L/2$ obtained using EFG and 175 field nodes. The parameter is $\alpha_i = 3.0$. The linear polynomial basis is used in MLS.



(a)



(b)

Figure 4.12. Nodal arrangements used to model the cantilever beam. (a) 189 regular nodes; (b) 189 irregular nodes.

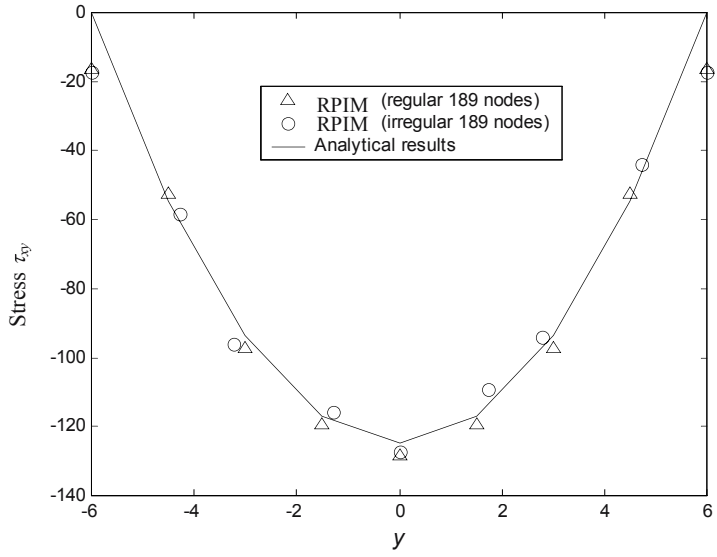


Figure 4.13. Shear stress distributions on the cross-section of the beam at $x = L/2$ obtained using RPIM and 189 field nodes. The MQ RBF is used in RPIM and the parameters used are $\alpha_c = 1.0$, $q = 1.03$, and $\alpha_i = 3.0$. The linear polynomial terms are added in RPIM-MQ.

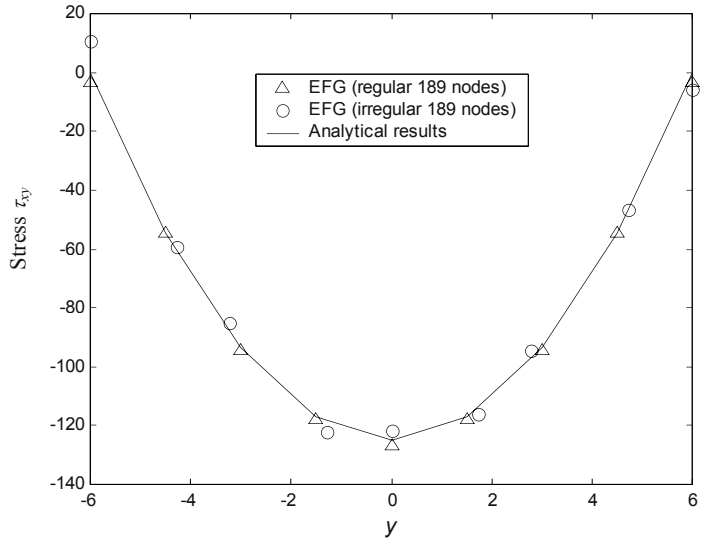


Figure 4.14. Shear stress distributions on the cross-section of the beam at $x = L/2$ obtained using EFG and 189 field nodes. The parameter used is $\alpha_i = 3.0$. The linear polynomial basis is used in MLS.

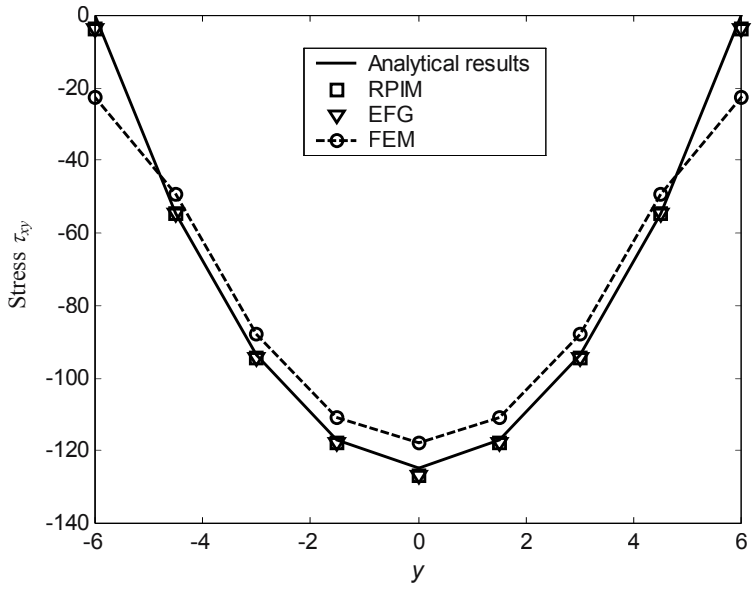


Figure 4.15. Shear stress distributions on the cross-section of the beam at $x = L/2$ obtained using different methods and 189 regular field nodes (bi-linear elements for FEM, $\alpha_{\mathrm{i}} = 3.5$ for RPIM and EFG).

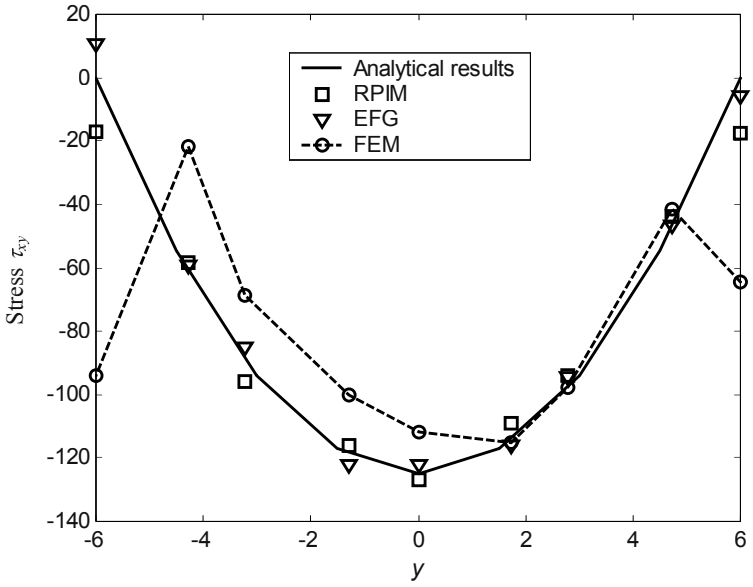


Figure 4.16. Shear stress distributions on the cross-section of the beam at $x = L/2$ obtained using different methods and 189 irregular field nodes. The mesh distortion effects on the FEM solution (using bi-linear elements) are obvious.

4.5.2 Effects of parameters

In the following studies, we consider the same cantilever beam problem because we know the analytical solution. The problem domain is represented by 189 (21×9) regularly distributed nodes, and 160 (20×8) rectangular background cells are used for numerical integrations. In each background cell, 4×4 Gauss points are employed. As the number of Gauss points used satisfies the sufficient requirement given in Equation (4.44), we considered the numerical integration to be sufficiently accurate. For quantitative and accurate analysis, the exact essential boundary conditions and exact natural boundary conditions are also used. In the exact natural boundary conditions, the distributed traction is employed at the right end of the beam. Hence, the curve integration is required on the boundary of the right end of the beam. The error in the energy norm defined by Equation (4.78) is used as an error indicator. In the RPIM method, the linear polynomial terms are added in the RPIM-MQ. In the EFG method, the linear basis and the cubic spline weight function (W1) are employed in the MLS approximation.

4.5.2.1 Parameter effects on RPIM method

a) Shape parameters used in RPIM

Only MQ-RBF is studied in this sub-section. More detailed discussions on the parameters of other RBF are presented in the paper by Wang and GR Liu (2002c). In the MQ-RBF, there are two shape parameters (see Table 3.2) to be investigated. The nodal spacing is a constant of $d_c = L/20 = 2.4$.

GR Liu (2002) and co-workers have found that parameter q has great influence on the performance of RPIM than that of parameter α_c. Therefore, q is investigated first with α_c fixed at 1.0, 2.0 and 4.0. Errors in the energy norm defined by Equation (4.78) for five different values of q ($q = -0.5$, 0.5, 0.98, 1.03 and 1.2) are computed and plotted in Figure 4.17. When $q = -0.5$ and 0.5, they are the classical MQ-RBFs. Wang and Liu (2002c) have discovered that when q=0.98 and 1.03 the RPIM-MQ performs the best. From Figure 4.17, it can be confirmed that q=0.98 and 1.03 give good results. GR Liu (2002) also found that the RPIM results become better when q is near the integers (e.g. 1.0) and the condition number of the RPIM moment matrix is large. However, when q equals an integer (e.g., q=1), the moment matrix is singular and the computation fails. We state without showing the data that when q is too large the error will significantly increase because of the too large condition number of the moment matrix.

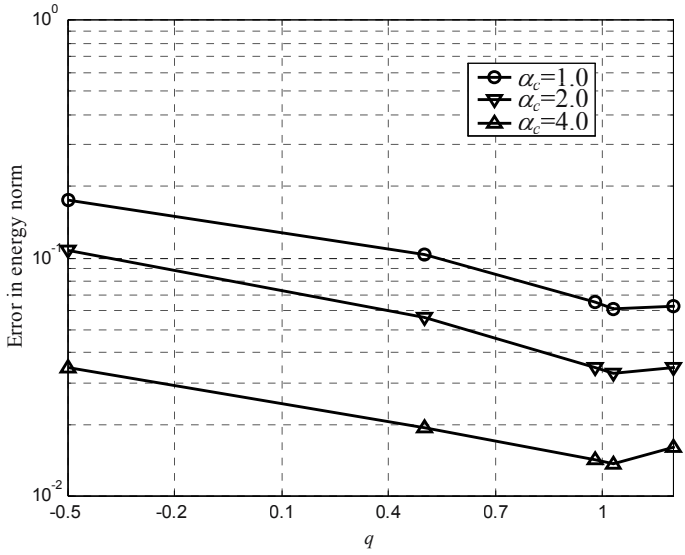


Figure 4.17. Influence of q on the RPIM-MQ, in which $\alpha_c = 1.0$, 2.0 and 4.0 are used. It can be found that q=0.98 and 1.03 give accurate results.

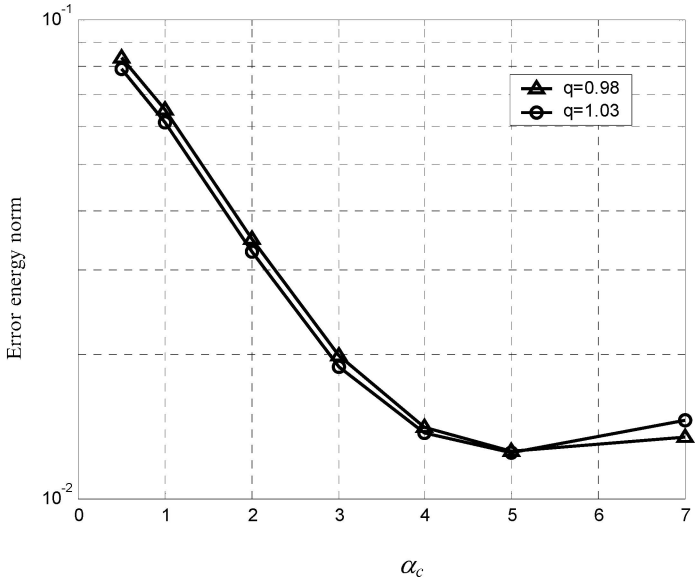


Figure 4.18. Influence of α_c on the RPIM-MQ in which q=0.98 and 1.03 are used. It can be found that the results for $\alpha_c = 3.0 \sim 7.0$ are more accurate.

GR Liu (2002) has found that α_c should be in the range of 1.0~6.0. In this study, α_c is further studied for a wider range of 0.5~7.0. Errors in the energy norm for different values of α_c are plotted in Figure 4.18. It is found that all α_c in the range studied can lead to satisfactory results, $\alpha_c = 3.0 \sim 7.0$ are preferred.

Hence, q=1.03 and $\alpha_c = 4.0$ are used in the following studies.

b) Dimension of the influence domain

The dimension of influence domains is defined in Equation (4.75), where d_{cx} and d_{cy} are nodal spacing in x and y directions near the field node i. In this study, $d_{cx} = L/20 = 2.4$ and $d_{cx} = D/8 = 1.5$ are used. The actual dimension of influence domains will be determined by changing α_{ix} and α_{iy}, which are dimensionless sizes in x and y directions. For simplicity, α_{ix}=α_{iy}=α_i is used in this study.

Errors in the energy norm computed using different α_i are plotted in Figure 4.19. The shape parameters of MQ-RBF are $q = 1.03$ and $\alpha_c = 4.0$. It can be found that the error changes with α_i, and the results of $\alpha_i = 2.0$, 3.0 and 4.0 are all very accurate. The error for $\alpha_i \le 1.5$ and $\alpha_i = 2.5$, 3.5 or 4.5 are relative large. The reason of bad results of $\alpha_i \le 1.5$ is that the influence

domain is too small, and there are not enough field nodes included for interpolation. Although the influence domains of $\alpha_i = 2.5$, 3.5 or 4.5 are big enough, the accuracy is also not very good. We suspect the reason is that $\alpha_i = 2.5$, 3.5 or 4.5 cannot match well with shape parameters. A more detailed study is needed.

Errors in the energy norm obtained using the RPIM with the parameters of $q = 1.03$ and $\alpha_c = 1.0$ are plotted in Figure 4.20. It can be found that the error e_e is more stable for this set of shape parameters and results of $\alpha_i \geq 3.0$ are very good. The aim of these studies is to show that some parameters must be carefully selected in RPIM-MQ to obtain good results. It is fortunate that the range of parameters is usually quite wide.

From the results of Figure 4.19 and Figure 4.20, $\alpha_i = 3.0$ are used in the following studies. In addition, considering the results presented in Figure 4.17~Figure 4.20 and the conclusions obtained by GR Liu (2002), q=1.03 together with $\alpha_c = 4.0$ is generally stable and accurate for many problems considered. Hence, q=1.03 and $\alpha_c = 4.0$ are used in the following studies.

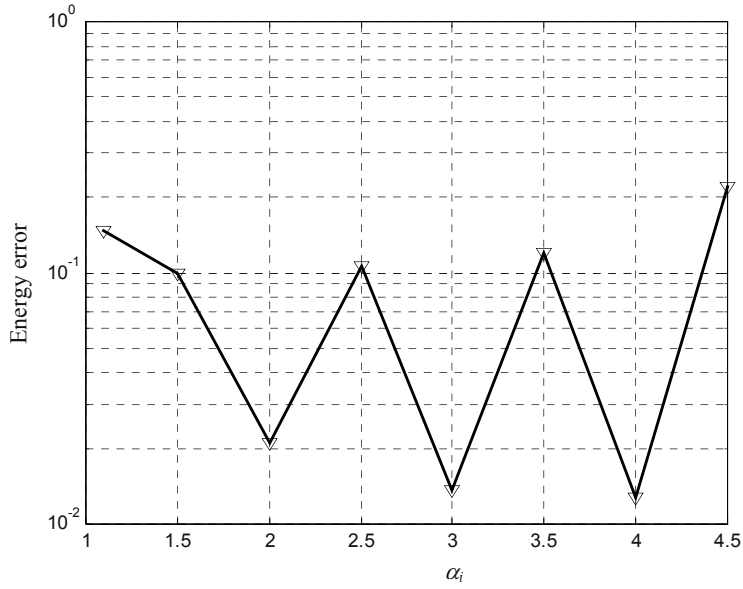


Figure 4.19. The effects of the dimension of influence (support) domain α_i on the RPIM-MQ (q=1.03, $\alpha_c = 4.0$).

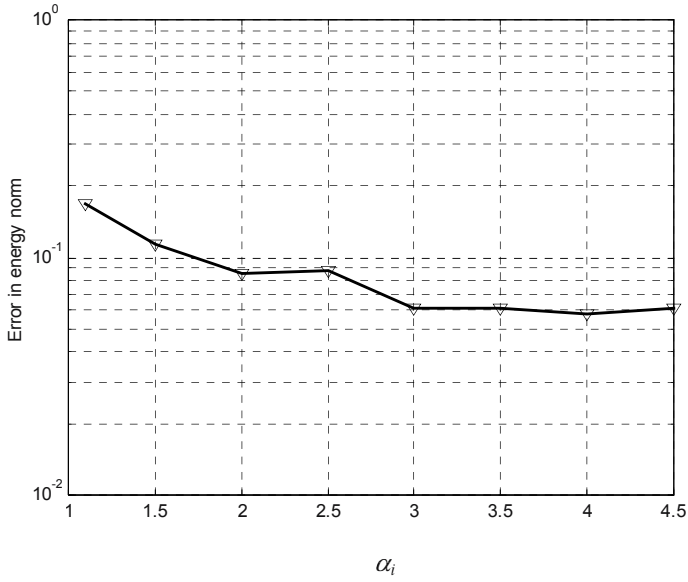


Figure 4.20. The effects of the dimension of influence (support) domain α_i on the RPIM-MQ ($\alpha_c = 1.0$ and q=1.03).

c) Convergence

The convergence of RPIM is numerically studied using regularly distributed 18 (3×6), 55(5×11), 112(7×16), 189(9×21), 403(13×31) and 697 (17×41) field nodes. The convergence curves are shown in Figure 4.21. For comparison, the convergence curve for FEM that uses the bi-linear elements is also plotted in the same figure. In this figure, h is in fact the nodal spacing, d_c, and it is equivalent to the element size (in x direction) in the FEM analysis in this case. The convergence rates, R, computed by linear regression are given in Figure 4.21. Note that the method of calculating the convergence rate can affect very much the values of the convergence rate due to the nature of the convergence process. In the early stage, the error reduces much faster than in a later stage, where the results are very close to the exact solution that is in polynomial form. For example, if only the right-most two points are used to calculate the convergence rate, the R value can be much higher. This is probably one of the reasons why different convergence rates are reported in different references.

Figure 4.21 shows the following conclusions:

- The accuracy of the RPIM method is much higher than that of FEM.

- The convergence rates of the RPIM are much higher than that of the Galerkin FEM, which is 1.0 for bi-linear elements.

Note again that the shape parameters chosen in the MQ-RBF will affect the convergence rate and the accuracy of the RPIM method.

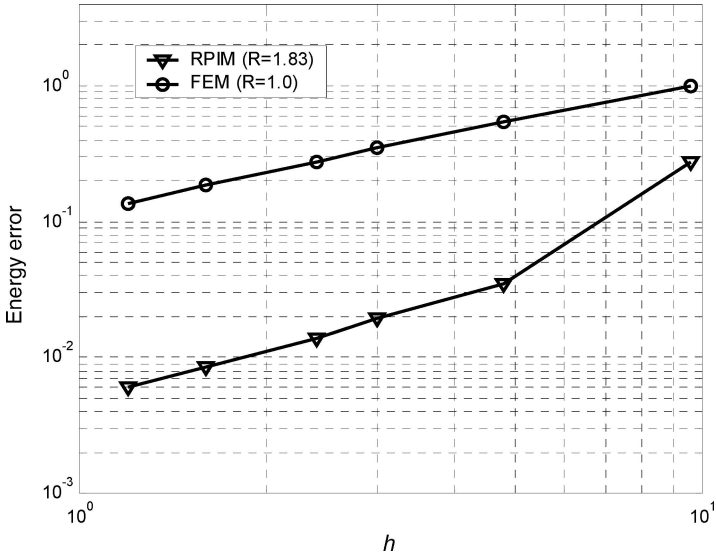


Figure 4.21. Numerical convergence of RPIM-MQ in error e_e in energy norm. The parameters used are $\alpha_c = 4.0$, and $\alpha_i = 3.0$. Linear polynomial terms are added in RPIM-MQ. R is the convergence rate computed by linear regression using all points in the figure.

4.5.2.2 Parameter effects on EFG method

1) Dimension of the influence domain

The size of influence domains is defined in Equation (4.75) where $d_{cx} = L/20 = 2.4$ and $d_{cx} = D/8 = 1.5$ for this problem. Errors of the energy norm for different α_i are plotted in Figure 4.22. It can be found that the error changes with α_i and the results for $2.0 \le \alpha_i \le 4.0$ are very good. When the influence domain is too small ($\alpha_i < 2.0$) or too big ($\alpha_i > 4.0$), the error of EFG results increases.

When the influence domain is too small ($\alpha_i < 2.0$), there are not enough nodes used to perform the function approximation for the field variables. The smoothness of MLS shape functions reduces. When the influence

domain is too large ($\alpha_i > 4.0$), the MLS shape functions become too smooth to represent the local properties of the field variables. In addition, large influence domains will also increase the computational cost. Hence, a proper influence domain should be used in the EFG method, and $\alpha_i = 2.5$ is found by this and other studies to be very good for many problems, and will be used in the following studies of the EFG method.

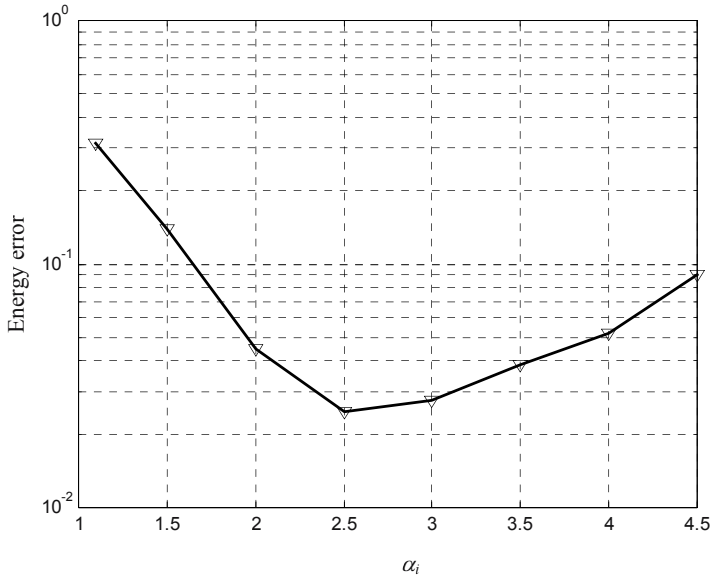


Figure 4.22. Effects of the dimension of influence (support) domain α_i on EFG.

2) Convergence

The convergence of EFG is numerically studied using regularly distributed 18 (3×6), 55(5×11), 112(7×16), 189(9×21), 403(13×31) and 697 (17×41) nodes, and the convergence curves are plotted in Figure 4.23. The convergence rate, *R*, is computed via linear regression. From Figure 4.23, it is observed that convergence rates of the EFG method is about 1.45 and is higher than that of the Galerkin bi-linear FEM. It should be mentioned here that only the linear basis is used in MLS to obtain the EFG results of Figure 4.23. The higher convergence rate of EFG is due to the fact that the MLS shape functions possess higher order smoothness inherited from the weight function used. Note also that the accuracy of the EFG method is much higher than that of the FEM.

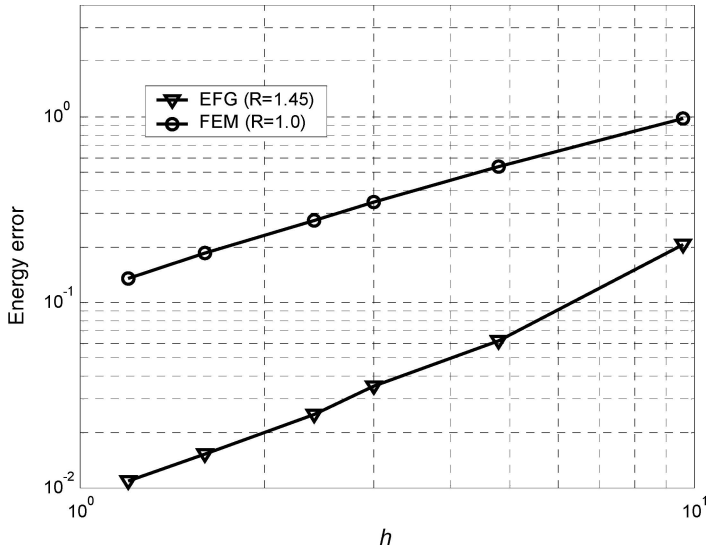


Figure 4.23. Numerical convergence of EFG results. The parameter used is $\alpha_i = 2.5$. The linear polynomial basis is used in the MLS approximation. R is the convergence rate computed by linear regression using all the points in the figure.

4.5.3 Comparison of convergence

For comparison, the numerically obtained convergence curves of RPIM, EFG and FEM are computed and plotted in Figure 4.24. From this figure, the following remarks can be made:

a) Both the convergence rates and the accuracies of RPIM and EFG are better than those of the bi-linear FEM. This is because the MFree shape functions have higher interpolation accuracy than the bi-linear FEM shape functiond, due to the use of more nodes in the construction of MFree shape functions.

b) The convergence rate and accuracy of the RPIM method are slightly better than those of the EFG method.

It should be mentioned here that the convergence is studied numerically based on regularly distributed nodes. If the irregularly distributed nodes are used, the convergence and accuracy of RPIM method and the EFG method will be much better than those of FEM, as shown, for example, in Figure 4.24.

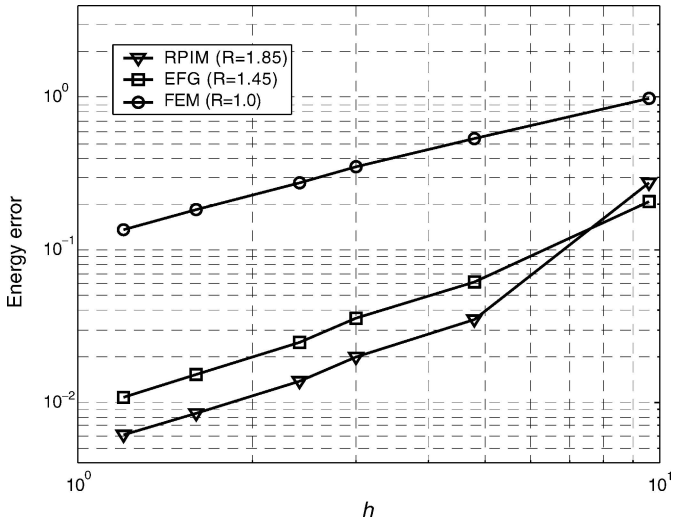


Figure 4.24. Comparison of numerical convergences of RPIM, EFG and linear FEM in error e_e of energy norm. R is the convergence rate.

4.5.4 Comparison of efficiency

The computational cost vs. the accuracy is a fair indicator to evaluate numerical methods. A successful numerical method should obtain high accuracy at a low computational cost. Regularly distributed 18, 55, 189 and 403 nodes are used to calculate the curves of error against the CPU time of RPIM, EFG and FEM. These curves obtained and plotted in Figure 4.25, where $\alpha_i = 3.0$ and $\alpha_i = 2.5$ are used in RPIM and EFG, respectively.

It should be noted that the computational cost of an MFree method mainly comes from two parts:

1) The first part is the cost of the interpolation, which mainly comes from computing the inverse of the moment matrix. Therefore, the cost of the interpolation is mainly determined by the dimension of the moment matrix. The dimensions of the moment matrices of RPIM are $n \times n$ (n is the number of the field nodes in the support domain), and the dimension of the moment matrix of EFG is $m \times m$ (m is the number of basis, $m = 3$ for the linear basis). Because of $n >> m$, the interpolation cost of RPIM is usually much higher than that of EFG.

2) The second part is the cost to solve the final discretized system equation, which depends on the maximum bandwidth of the global stiffness matrix. The maximum bandwidth of the final stiffness

matrix increases with the number of nodes chosen in the support domains, for a given numbering system. The support domains used in RPIM is usually bigger than those used in EFG. The computational cost of RPIM in solving the final system equation is therefore higher.

The RPIM is first compared with the EFG method, in which the penalty method is used to enforce the essential boundary conditions. From Figure 4.25, the following remarks can be observed:

a) For a desired accuracy (such as 10^{-1} error in the energy norm), the cost of EFG (with penalty method) is the lower than that of RPIM.

b) For a given cost (say 20 s), the accuracy of EFG is better than that of RPIM.

For this discussion, one can conclude that the efficiency of the EFG method (using penalty method to enforce essential boundary conditions) is better than that of the RPIM method.

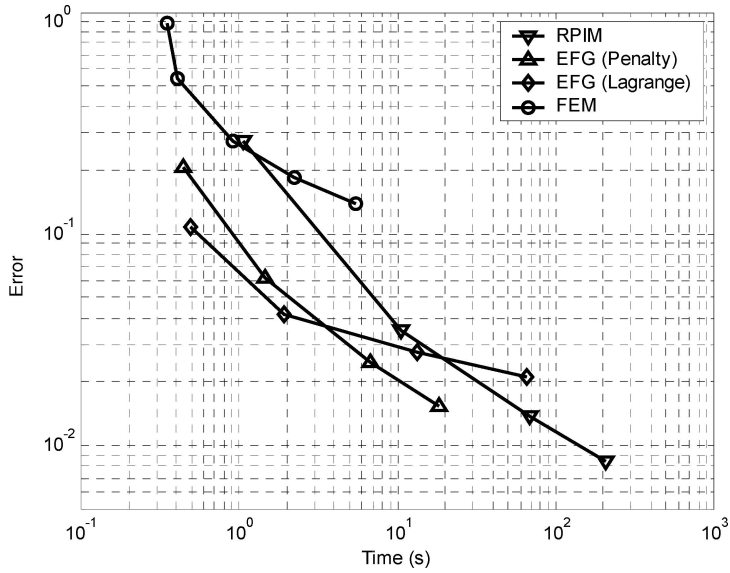


Figure 4.25. Comparison of the computational efficiencies of RPIM, EFG and FEM in error e_e in energy norm. In RPIM-MQ, the parameters are $\alpha_c = 4.0$, $q = 1.03$, $\alpha_i = 3.0$, and *m=3*. In EFG, the parameter is $\alpha_i = 2.5$, the weight function W1 and the linear polynomial basis are used in the MLS approximation. In FEM, bi-linear elements are used.

If the Lagrange multiplier method is used, the dimension of the global stiffness matrix will increase, and the stiffness matrix will become an

unbanded matrix as shown in Equation (4.74). This will significantly increase the computational cost of the EFG method especially for large systems. To prove this point, the curves of error against the CPU time of the EFG method using the Lagrange multiplier method is also plotted in Figure 4.25. It is found that the EFG method using the Lagrange multiplier method is less efficient than RPIM.

For comparison, the curves of error against the CPU time of the conventional FEM using bi-linear elements are also plotted in Figure 4.25. It is found that FEM needs more CPU time to obtain the desire computational accuracy than both RPIM and EFG; the conventional FEM is less efficient than RPIM or EFG.

4.6 EXAMPLE FOR 3D SOLIDS

Because of the robustness and effectiveness of the MFree RPIM method, the RPIM has successfully been applied to many types of problems (see, e.g., Chapter 2). In this section a simple example problem of a three-dimensional (3D) solid is solved using the RPIM. The materials used in this section are largely from the work by GR Liu and Zhang et al. (2003), where more examples can be found.

The standard basic equations of 3D elastic solids were given in Sub-section 1.2.1. The procedures used in Section 4.2 gives the discretized system equations of the RPIM for 3D elastic solids. Detailed discussions are omitted because it is largely similar to the 2D case. Readers may derive these formulations following the procedures given in Section 4.2. Note that the construction of RPIM shape functions for 3D domain is very similar to the 2D RPIM shape functions, and the RBFs are distance functions; need only change the formula for calculating the distance.

Consider a 3D cantilever beam (shown in Figure 4.26) with a circular hole subjected to a uniformly distributed load of f=125. The left end of the beam is fixed, and the right end of the beam is a half circle. The geometric and material constants for the beam are: length (to the centre of the internal circle): L=48; height: D=12; width: T=8; radius of the outer half-circle: R=6; radius of the internal circle: r =2; Young's modulus: $E = 3\times10^7$, and Poisson's ratio: $\nu = 0.3$.

The results of displacements and stresses are computed for all field nodes using both RPIM and the FEM. For simplicity, only the results of the vertical displacement at point A at (48, 4, 8) (see, Figure 4.26) are presented

here. The FEM results, obtained using the commercial software package ANSYS with a very fine mesh of 11109 elements (Solid92-type 10-node tetrahedral element) shown in Figure 4.27, are taken as the reference solution for the comparison study. The FEM reference solution is found as $v_{\mathrm{A}}^{\mathrm{Ref}} = -0.11211 \times 10^{-2}$.

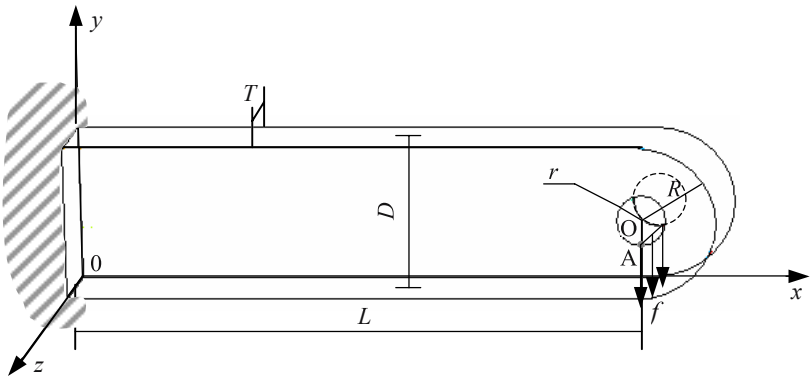


Figure 4.26. A 3D cantilever beam subjected to a uniformly distributed load.

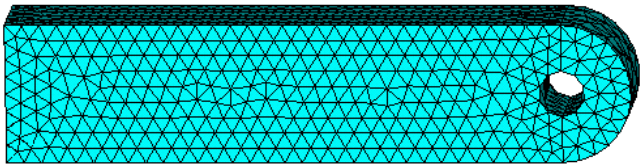


Figure 4.27. FEM mesh for the 3D cantilever beam.

The RPIM-MQ is used to solve this problem. In the MQ-RBF, the shape parameters used are q=1.03 and α_c=4.0. Irregularly distributed nodes shown in Figure 4.28 are used. The tetrahedral background cells are used for the numerical integrations. In each tetrahedral background cell, 4 Gauss points are employed.

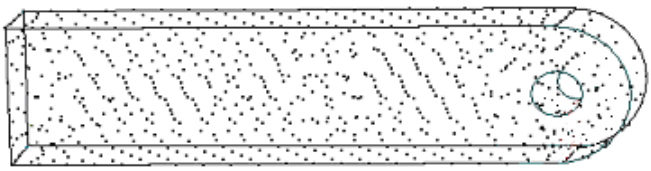


Figure 4.28. Irregular nodal arrangements for the 3D cantilever beam.

Results of displacements obtained using RPIM are listed in Table 4.1. The FEM results obtained using ANSYS (using 4-node elements) with the same nodes as those used in RPIM are also listed. From this table, it can be found that the RPIM gives much better results than that of the FEM.

Table 4.1. Vertical displacement at point A, v_A, obtained using the RPIM and FEM using exactly the same sets of nodes

Number of nodes (number of cells)	RPIM(1)		RPIM(2)		ANSYS	
	v_A	Error (%)	v_A	Error (%)	v_A	Error (%)
196 (538)	-0.1109E-2	1.07	-0.1125E-2	0.40	-0.8307E-3	25.89
1146 (4685)	-0.1133E-2	1.06	-0.1137E-2	1.46	-0.1046E-2	6.63
1596 (6815)	-0.1125E-2	0.41	-0.1134E-2	1.16	-0.1060E-2	5.41
1999 (8771)	-0.1125E-2	0.38	-0.1137E-2	1.43	-0.1067E-2	4.78

(1): 70 nearest nodes are used to construct RPIM shape functions;
(2): 50 nearest nodes are used to construct RPIM shape functions.
(3) Error $= \left| v_{\mathrm{A}} - v_{\mathrm{A}}^{\mathrm{Ref}} \right| \Big/ \left| v_{\mathrm{A}}^{\mathrm{Ref}} \right|$.
(4) Reference solution: $v_{\mathrm{A}}^{\mathrm{Ref}} = -0.11211 \times 10^{-2}$ obtained using ANSYS and very fine mesh (11109 elemets).

4.7 EXAMPLES FOR GEOMETRICALLY NONLINEAR PROBLEMS

The purpose of this section is to show some simple examples of the applications of the RPIM to geometrically nonlinear solid mechanics problems. The detailed description of this work can be found in a paper by Dai et al. (2003). For applications to material non-linear problems, readers may refer to the recent work by Dai et al. (2004).

The standard Newton-Raphson iteration procedure and the formulation in material description are used in the study. The standard basic equations and formulation procedures are largely the same as those used in the FEM (e.g., Zienkiewicz and Taylor, 2000). The difference is mainly in the creation of the shape functions. Hence, detailed discussions are omitted here. Readers are recommended to refer to the books on nonlinear FEM (see, e.g., Zienkiewicz and Taylor, 2000).

In these examples, compressible hyperelastic neo-Hookean materials are used with Lame constants of $\mu = 0.5\times10^4\, N/cm^2$ and $\lambda = (1/3)\times10^4\, N/cm^2$. The plane strain state is considered in this section.

The RPIM-MQ shape functions are computed with q = 1.03 and α_c=1.0 augmented with six (2nd order) monomials. In the following studies, $\alpha_s = 1.5$ is used for the local support domains. Gauss quadrature using 4×4 Gauss points is employed in each background cell.

4.7.1 Simulation of upsetting of a billet

A two-dimensional billet subjected to deep compression is studied using RPIM. The initial dimensions of the billet are 4*cm* wide and 6*cm* high shown in Figure 4.29. The domain is initially represented by 6×6 uniform nodes, and 5×5 rectangular background cells are used for the integration. The billet is loaded via displacement control on the upper surface with the bottom surface fully fixed. A Newton-Raphson iteration procedure is used with increments of vertical displacement equal to 0.2*cm*. Figure 4.30 shows the progression of deformation at different steps. It is seen that the billet is compressed as much as 56% compared to its original height. The same problem is also analyzed using the conventional non-linear FEM. It is found that when the FEM is used (Zienkiewicz and Taylor, 2000), the convergence stops at the amount of 50% of compression. An irregular node distribution is also used in the RPIM for the simulation, and results are plotted in Figure 4.31.

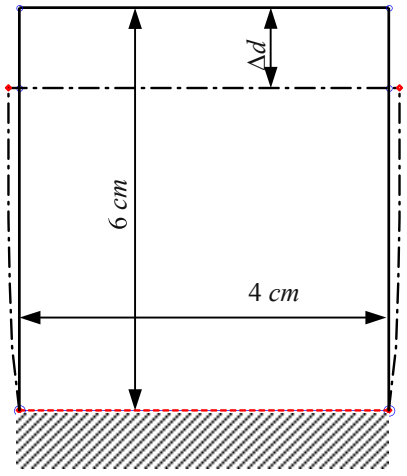
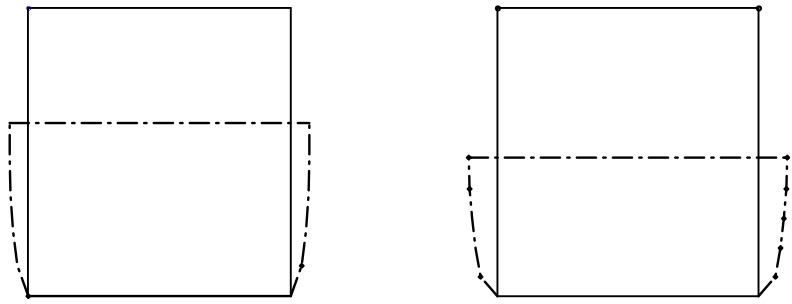


Figure 4.29. Schematical drawing of the initial and deformed billet subjected to deep compression.



(a) $\Delta d = 2.0$ cm (b) $\Delta d = 2.8$ cm

Figure 4.30. Deformed profile of a compressed billet simulated using RPIM and 6×6 regular nodes.

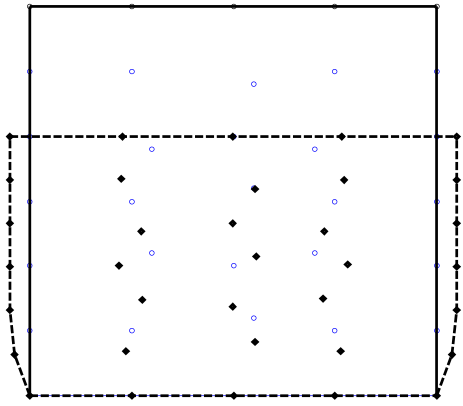


Figure 4.31. The deformed profile of a compressed billet simulated using RPIM and irregular nodes. Circles: initial positions of the nodes; Diamonds: positions of nodes in the deformed billet.

4.7.2 Simulation of large deflection of a cantilever beam

In this example, a large deformation analysis is performed for a cantilever beam subjected to a progressively increasing load at the middle point on the cross-section at the free end with each load step of $\Delta F = 16.0\ N$. The dimensions of the beam are $(10cm \times 2cm)$ and it is initially represented using (11×3) regularly distributed nodes (see Figure 4.32). The analysis is carried out using twenty load incremental steps (n = 20). The simulation converges very fast, and less than 4 iterations are needed in each load increment.

Figure 4.33 illustrates different stages of deformation for the beam obtained using RPIM. The tip deflections at different load steps are plotted in Figure 4.33. It can be seen that, the nonlinear analysis reveals the stiffer effect of the beam compared to the linear behavior.

4.7.3 Simulation of large deflection of a fixed-fixed beam

This example analyzes the large deformation of a beam with both left and right sides fully fixed. The beam is subjected to a uniformly distributed and progressively increasing load with each load step of $\Delta f = 80.0$ *N/cm.* After twenty steps of loading, the final profile of the beam is shown in Figure 4.34. The deflections at the mid-node at different load steps are plotted in Figure 4.35. Geometrically non-linear effects similar to the case of the cantilever beam are observed.

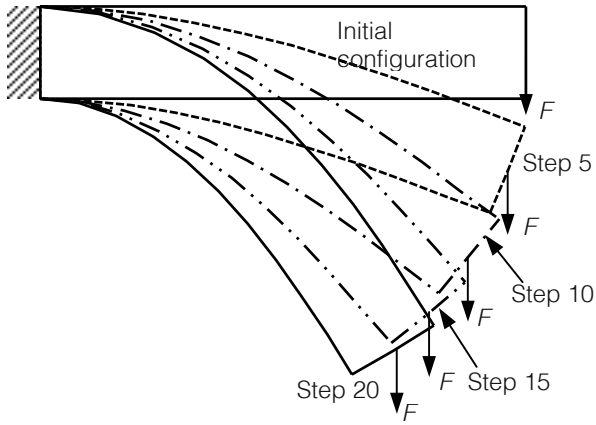


Figure 4.32. Large deformations of a cantilever beam at different steps simulated using RPIM.

4.8 MFREE2D©

MFree2D© is an adaptive stress analysis software package developed by GR Liu and co-workers (GR Liu and Tu, et al., 2000) based on EFG and RPIM. It was showcased in 1999 in the APCOM'99 conference. MFree2D©is designed for 2D stress and strain analysis in solid mechanics and structural mechanics subjected to static loadings. The software consists

of three major processors: *MFreePre*, *MFreeApp* and *MFreePost*. *MFreePre* is a preprocessor to formulate the input required by *MFreeApp*; the latter performs computations and yields results which are then fed to *MFreePost* for post processing.

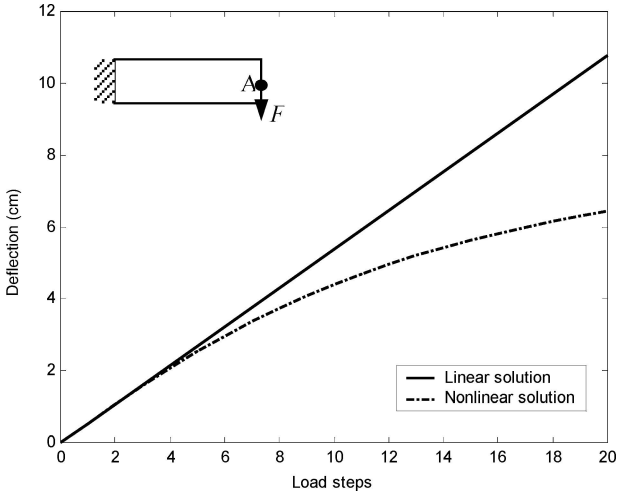


Figure 4.33. Deflections at point A at the middle of the cross-section at the free end of a cantilever beam simulated using RPIM.

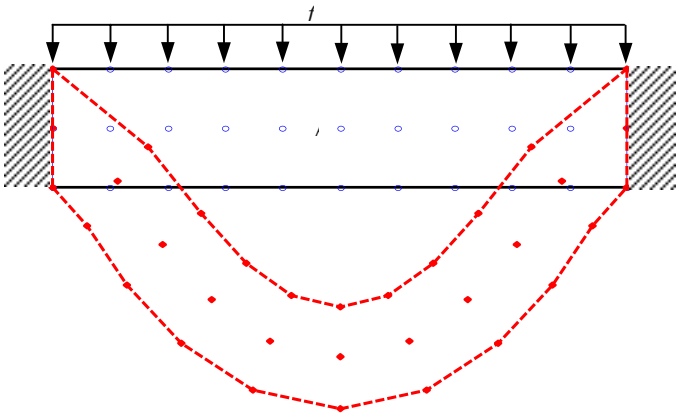


Figure 4.34. Initial and final profiles of a fixed-fixed beam subjected to a uniformly distributed load. RPIM is used and the loading keeps vertical in the loading process.

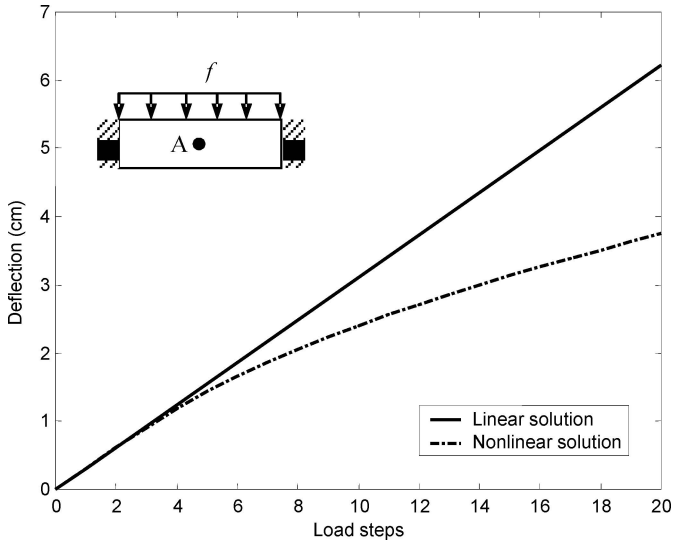


Figure 4.35. Deflections at point A at the middle of the central axis of the fixed-fixed beam simulated using RPIM.

These three processors can work either in an integrated manner or independently. One salient feature of MFree2D© is that it is designed to be user-friendly and thus, has few input requirements from users. The main features of MFree2D© include:

- The problem domain is discretized using scattered nodes and the discretization is fully automatic.
- Adaptive refinement techniques are implemented to ensure the results have a desired accuracy.
- User-friendly graphical-user-interface (GUI).

In the current version of MFree2D©, the RPIM method and the EFG method are available, and Visual C++ is used as the programming language. MFree2D© can be downloaded from the website (http://www.nus.edu.sg/ACES).

The source codes provided in this book are largely consistent with the MFree2D©. However, for easy understanding and comprehension, only FORTRAN source codes are provided in this book for simple problems.

4.9 REMARKS

MFree global Galerkin weak-form methods are discussed in this chapter. The MFree RPIM method based on the radial point interpolation and the EFG method based on the MLS approximation are detailed. A computer code of RPIM and EFG for linear elasticity is provided. Numerical studies are presented to show the implementation of the present code. The performance and convergence of RPIM and EFG are studied numerically and compared. It may be concluded that the accuracy, convergence, and efficiency of RPIM and EFG are better than the conventional FEM.

From the studies in this chapter, we can make the following important remarks:

a) The compatibility of the trial (shape) functions in the whole domain is required in MFree global weak-form methods.

b) In RPIM, the recommended shape parameters for the MQ-RBF are q=1.03 and α_c=4.0.

c) The accuracy of solutions changes with the sizes of the influence domains α_i. In RPIM, α_i =3.0 is recommended. In EFG, we recommend α_i =2.5.

d) The convergence rates of both the RPIM and EFG methods are good. The convergence rate of the RPIM is better than that of EFG.

e) The efficiency of the EFG method (using penalty method to enforce essential boundary conditions) is better than that of RPIM.

f) The EFG method with the Lagrange multiplier method for enforcing the essential boundary conditions is much less efficient than RPIM.

g) The bi-linear FEM is less efficient than RPIM or EFG.

Note that as the solution for the cantilever beam has polynomial form, methods using MLS shape functions (with polynomial basis) perform better than methods using RPIM shape functions (with RBF basis). For more complex problems whose solutions are not in the polynomial form, the situation can change, as observed in the surface fitting tests presented in Chapter 3.

APPENDIX

Appendix 4.1. Major subroutines used in MFree_Global.f90 and their functions

Subroutines	Functions
Input	Input data from an external input file
GaussCoefficient	Obtain coefficients of Gauss points
CellGaussPoints	Set Gauss points for a cell
SupportDomain	Determine the support domain for an interpolation point
MLS_ShapeFunc_2D (or RPIM_ShapeFunc_2D)	Compute shape functions and their derivatives at an interpolation point
PointStiffnessMatrix	Compute the stiffness matrix for a quadrature point
EssentialBC	Enforce essential boundary conditions
NaturalBC	Implement natural boundary conditions
SolveBand	Solve system equation
GetDisplacement	Obtain the final displacements using the RPIM or the MLS shape functions
GetStress	Retrieve the strain, stress, and compute error in the energy norm

Appendix 4.2. The major variables used in MFree_Global.f90

Variable	Type	Usage	Function
Young, anu	Long real	Input	Young's modulus and Poisson ratio
Dmat (3,3)	Long real	Compute	The matrix of elastic constants
nx	Integer	Parameter	Dimension of this problem; *nx*=2 for 2D problem
ng	Integer	Parameter	Shape of the background cells, and *ng*=4 is used for a rectangular cell
numnode	Integer	Input	Number of field nodes

x(*nx*, *numnode*)	Long real	Input	Coordinates x and y for all field nodes: $x(1,i)=x_i$; $x(2,i)=y_i$
numq	Integer	Input	Number of background points to form background cells
xc(*nx*, *numnode*)	Long real	Input	Coordinates x and y for background points: $xc(1,i)=x_i$; $xc(2,i)=y_i$
numcell	Integer	Input	Number of background cells
noCell(*ng*, *numcell*)	Integer	Input	Node ID for background cells
nquado	Integer	Input	Number of Gauss points used in one dimension in a background cell. For rectangular background cell, the total Gauss points used for a 2D cell is *nquado*×*nquado*.
npEBCnum	Integer	Input	Number of nodes with essential boundary conditions
npEBC, **pEBC**	Integer long real	Input	**npEBC**(1,*i*): ID of field nodes with the essential boundary condition; if **npEBC**(2,*i*)=1 then u_x is prescribed in **pEBC**(1,*i*); if **pEBC**(3,*i*)=1 then u_y is prescribed in **pEBC**(2,*i*)
npNBCnum	Integer	Input	Number of nodes with natural boundary conditions
npNBC, pNBC	Integer long real	Input	**npNBC**(1,*i*): ID of field nodes with the natural boundary condition: if **npNBC**(2,i)=1 then f_x is prescribed in **pNBC**(1,i); if **pNBC**(3,i)=1 then f_y is prescribed in **pEBC**(2,i)
alfs	Long real	Input	Dimensionless size of support (influence) domain
pAlf	Long real	Input	Penalty coefficient
Ds*(nx,* numnode*)*	Long real	Compute	The size of the influence domain: $ds(1,i)=d_{sxi}$, $ds(2,i)=d_{syi}$
ndex	Integer	Compute	Number of field nodes in the support domain for an interpolation point
Ph(10, *ndex*)	Long real	Compute	Meshfree shape functions and their derivatives.
Ak	Long real	Compute	Global stiffness matrix

Force	Long real	Compute	Global force vector
disp	Long real	Compute	Displacement vector, $disp(2*i\text{-}1)=u_i$; $disp(2*i\text{-}1)=v_i$
Stress	Long real	Compute	The array to store the stress components for all field nodes

Appendix 4.3. The dummy arguments used in the subroutine GaussCoefficient

Variable	Type	Usage	Function
k	Integer	Input	Number of Gauss points in 1D
v(2,*k*)	Long real	Output	The array for the coefficient of Gauss points, *v*(1, *i*): coefficient for the coordinate of a Gauss point; *v*(2, *i*): Gauss weight for this Gauss point

Appendix 4.4. The dummy arguments used in the subroutine CellGaussPoints

Variable	Type	Usage	Function
ibk	Integer	Input	ID of the background cell considered
numgauss	Integer	Input	Number of Gauss points in a cell
Gauss (*nx,nquado*)	Long real	Input	The array for the coefficients of Gauss points; *Gauss*(1, *i*): coefficient for the coordinate of a Gauss point; *Gauss* (2, *i*): Gauss weight for this Gauss point
gs(4,*numg*)	Integer	Output	Array storing information of Gauss points for a cell: *gs*(1, *i*): coordinate *x* for Gauss point *i*; *gs*(2, *i*): coordinate *y* for Gauss point *i*; *gs*(3, *i*): Gauss weight for Gauss point *i*; *gs*(4, *i*): Jacobian value for this cell

Appendix 4.5. The dummy arguments used in the subroutine SupportDomain

Variable	Type	Usage	Function
Gpos(*nx*)	Long real	Input	Coordinates of a point of interest
ndex	Integer	Output	Number of field nodes used in the support domain
nv(*ndex*)	Integer	Output	Node ID of field nodes selected for the construction of shape functions

Appendix 4.6. The dummy arguments used in the subroutine PointStiffnessMatrix

Variable	Type	Usage	Function
ndex	Integer	Input	Number of field nodes used in the local domain for the construction of shape functions
Weight	Long real	Input	Gauss weight for a Gauss point
ajac	Long real	Input	Jacobian value for the cell
Ph(10, *ndex*)	Long real	Input	Shape functions and their derivatives.
GSPk(2***ndex*,2***ndex*)	Long real	Output	Stiffness matrix for the Gauss point

Appendix 4.7. The dummy arguments used in the subroutine EssentialBC

Variable	Type	Usage	Function
numnode	Integer	Input	Total number of field nodes
pAlf	Long real	Input	Penalty coefficient
alfs	Long real	Input	Dimensionless size for support (influence) domain
Ds*(nx, numnode)*	Long real	Input	The size of the influence domain
npEBCnum	Integer	Input	Number of nodes with essential boundary conditions
npEBC(3,100), **pEBC**(*nx*,100)	Integer, long real	Input	Essential boundary condition

Ak(2**numnode*, 2**numnode*)	Long real	Input and output	Global stiffness matrix
Force(2**numnode*)	Long real	Input and output	Global force vector

Appendix 4.8. The dummy arguments used in subroutine NaturalBC_concentrated

Variable	Type	Usage	Function
numnode	Integer	Input	Total number of field nodes
alfs	Long real	Input	Dimensionless size for support (influence) domain
Ds*(nx, numnode)*	Long real	Input	The size of the influence domain
npNBCnum	Integer	Input	Number of nodes with natural boundary conditions.
npNBC, pNBC	Integer long real	Input	Natural boundary condition
ep	Long real	Input	Tolerance
Force(2**numnode*)	Long real	Input and output	Global force vector

Appendix 4.9. The dummy arguments used in subroutine NaturalBC_distributed

Variable	Type	Usage	Function
numnode	Integer	Input	Total number of field nodes
alfs	Long real	Input	Dimensionless size for support (influence) domain
Ds*(nx, numnode)*	Long real	Input	The size of the influence domain: $ds(1,i)=d_{sxi}$, $ds(2,i)=d_{syi}$
x(*nx*, *numnode*)	Long real	Input	Coordinates x and y for all field nodes
numq	Integer	Input	Number of background points to form background cells
xc(*nx*, *numnode*)	Long real	Input	Coordinates x and y for

			background points
nquado	Integer	Input	Number of Gauss points used in one dimension in a background cell.
Gauss (*nx,nquado*)	Long real	Input	The array for the coefficients of Gauss points:
in, jn	Integer	Input	Two ends of the sub-boundary Γ_t
Force(2**numnode*)	Long real	Input and output	Global force vector

Appendix 4.10. The dummy arguments used in the subroutine SolverBand

Variable	Type	Usage	Function
neq	Integer	Input	Number of equations
nmat	Integer	Input	Number of rows of the array **Ak**
Ak(*neq*, *neq*)	Long real	Input	Coefficient matrix of the equation
fp	Long real	Input, output	Input: the right hand side of the equations;Output: the solution of the equations

Appendix *4.11.* The dummy arguments used in the subroutine GetDisplacement

Variable	Type	Usage	Function
nx	Integer	Input	*nx*=2 for 2D problem
numnode	Integer	Input	Total number of field nodes
x(*nx*, *numnode*)	Long real	Input	Coordinates *x* and *y* for all field nodes
alfs	Long real	Input	Dimensionless size for support (influence) domain
Ds*(nx, numnode)*	Long real	Input	The size of the influence domain
u2(*nx*, *numnode*)	Long real	Input	Nodal parameters of displacements for field nodes
Disp(*nx,numnode)*	Long real	Output	Actual nodal displacements for field nodes

Appendix 4.12. The dummy arguments used in the subroutine GetStress

Variable	Type	Usage	Function
nx	Integer	Input	*nx*=2 for 2D problem
numnode	Integer	Input	Total number of field nodes
x(*nx*, *numnode*)	Long real	Input	Coordinates *x* and *y* for all field nodes. $x(1,i)=x_i$; $x(2,i)=y_i$
alfs	Long real	Input	Dimensionless size of support (influence) domain
Ds*(nx, numnode)*	Long real	Input	The size of the influence domain. $ds(1,i)=d_{sxi}$, $ds(2,i)=d_{syi}$
Stress (3,*numnode*)	Long real	Output	Array storing stress components of field nodes

Appendix 4.13. The iput data file: Input175_55.dat used in MFree_Global.f90. As shown in Figure 4.6, A total of 175 regular field nodes and 40 background cells are used

```
*L,H,E,v,P,
     48.00000    12.00000  0.3000E+08     0.30000  1000.00000
 *numnode, unuse
         175           0
 *ndivx,ndivy
          24           6
 *numq,numcell
          55          40
 *ndivxq,ndivyq
          10           4
 *nquado,alf
           4   100000000.000000
 *Influ.  domain: ALfs
   3.0
 *Field nodes: x[]
    1   0.00000   6.00000         42  10.00000  -6.00000
    2   0.00000   4.00000         43  12.00000   6.00000
    3   0.00000   2.00000         44  12.00000   4.00000
    4   0.00000   0.00000         45  12.00000   2.00000
    5   0.00000  -2.00000         46  12.00000   0.00000
    6   0.00000  -4.00000         47  12.00000  -2.00000
    7   0.00000  -6.00000         48  12.00000  -4.00000
    8   2.00000   6.00000         49  12.00000  -6.00000
    9   2.00000   4.00000         50  14.00000   6.00000
   10   2.00000   2.00000         51  14.00000   4.00000
   11   2.00000   0.00000         52  14.00000   2.00000
   12   2.00000  -2.00000         53  14.00000   0.00000
   13   2.00000  -4.00000         54  14.00000  -2.00000
   14   2.00000  -6.00000         55  14.00000  -4.00000
   15   4.00000   6.00000         56  14.00000  -6.00000
   16   4.00000   4.00000         57  16.00000   6.00000
   17   4.00000   2.00000         58  16.00000   4.00000
   18   4.00000   0.00000         59  16.00000   2.00000
   19   4.00000  -2.00000         60  16.00000   0.00000
   20   4.00000  -4.00000         61  16.00000  -2.00000
   21   4.00000  -6.00000         62  16.00000  -4.00000
   22   6.00000   6.00000         63  16.00000  -6.00000
   23   6.00000   4.00000         64  18.00000   6.00000
   24   6.00000   2.00000         65  18.00000   4.00000
   25   6.00000   0.00000         66  18.00000   2.00000
   26   6.00000  -2.00000         67  18.00000   0.00000
   27   6.00000  -4.00000         68  18.00000  -2.00000
   28   6.00000  -6.00000         69  18.00000  -4.00000
   29   8.00000   6.00000         70  18.00000  -6.00000
   30   8.00000   4.00000         71  20.00000   6.00000
   31   8.00000   2.00000         72  20.00000   4.00000
   32   8.00000   0.00000         73  20.00000   2.00000
   33   8.00000  -2.00000         74  20.00000   0.00000
   34   8.00000  -4.00000         75  20.00000  -2.00000
   35   8.00000  -6.00000         76  20.00000  -4.00000
   36  10.00000   6.00000         77  20.00000  -6.00000
   37  10.00000   4.00000         78  22.00000   6.00000
   38  10.00000   2.00000         79  22.00000   4.00000
   39  10.00000   0.00000         80  22.00000   2.00000
   40  10.00000  -2.00000         81  22.00000   0.00000
   41  10.00000  -4.00000         82  22.00000  -2.00000
```

```
  83  22.00000  -4.00000        130  36.00000   0.00000
  84  22.00000  -6.00000        131  36.00000  -2.00000
  85  24.00000   6.00000        132  36.00000  -4.00000
  86  24.00000   4.00000        133  36.00000  -6.00000
  87  24.00000   2.00000        134  38.00000   6.00000
  88  24.00000   0.00000        135  38.00000   4.00000
  89  24.00000  -2.00000        136  38.00000   2.00000
  90  24.00000  -4.00000        137  38.00000   0.00000
  91  24.00000  -6.00000        138  38.00000  -2.00000
  92  26.00000   6.00000        139  38.00000  -4.00000
  93  26.00000   4.00000        140  38.00000  -6.00000
  94  26.00000   2.00000        141  40.00000   6.00000
  95  26.00000   0.00000        142  40.00000   4.00000
  96  26.00000  -2.00000        143  40.00000   2.00000
  97  26.00000  -4.00000        144  40.00000   0.00000
  98  26.00000  -6.00000        145  40.00000  -2.00000
  99  28.00000   6.00000        146  40.00000  -4.00000
 100  28.00000   4.00000        147  40.00000  -6.00000
 101  28.00000   2.00000        148  42.00000   6.00000
 102  28.00000   0.00000        149  42.00000   4.00000
 103  28.00000  -2.00000        150  42.00000   2.00000
 104  28.00000  -4.00000        151  42.00000   0.00000
 105  28.00000  -6.00000        152  42.00000  -2.00000
 106  30.00000   6.00000        153  42.00000  -4.00000
 107  30.00000   4.00000        154  42.00000  -6.00000
 108  30.00000   2.00000        155  44.00000   6.00000
 109  30.00000   0.00000        156  44.00000   4.00000
 110  30.00000  -2.00000        157  44.00000   2.00000
 111  30.00000  -4.00000        158  44.00000   0.00000
 112  30.00000  -6.00000        159  44.00000  -2.00000
 113  32.00000   6.00000        160  44.00000  -4.00000
 114  32.00000   4.00000        161  44.00000  -6.00000
 115  32.00000   2.00000        162  46.00000   6.00000
 116  32.00000   0.00000        163  46.00000   4.00000
 117  32.00000  -2.00000        164  46.00000   2.00000
 118  32.00000  -4.00000        165  46.00000   0.00000
 119  32.00000  -6.00000        166  46.00000  -2.00000
 120  34.00000   6.00000        167  46.00000  -4.00000
 121  34.00000   4.00000        168  46.00000  -6.00000
 122  34.00000   2.00000        169  48.00000   6.00000
 123  34.00000   0.00000        170  48.00000   4.00000
 124  34.00000  -2.00000        171  48.00000   2.00000
 125  34.00000  -4.00000        172  48.00000   0.00000
 126  34.00000  -6.00000        173  48.00000  -2.00000
 127  36.00000   6.00000        174  48.00000  -4.00000
 128  36.00000   4.00000        175  48.00000  -6.00000
 129  36.00000   2.00000
*Points for BK cells:xc[]
   1   0.00000   6.00000         29  24.00000  -3.00000
   2   0.00000   3.00000         30  24.00000  -6.00000
   3   0.00000   0.00000         31  28.80000   6.00000
   4   0.00000  -3.00000         32  28.80000   3.00000
   5   0.00000  -6.00000         33  28.80000   0.00000
   6   4.80000   6.00000         34  28.80000  -3.00000
   7   4.80000   3.00000         35  28.80000  -6.00000
   8   4.80000   0.00000         36  33.60000   6.00000
   9   4.80000  -3.00000         37  33.60000   3.00000
  10   4.80000  -6.00000         38  33.60000   0.00000
  11   9.60000   6.00000         39  33.60000  -3.00000
  12   9.60000   3.00000         40  33.60000  -6.00000
  13   9.60000   0.00000         41  38.40000   6.00000
```

```
  14   9.60000  -3.00000          42  38.40000   3.00000
  15   9.60000  -6.00000          43  38.40000   0.00000
  16  14.40000   6.00000          44  38.40000  -3.00000
  17  14.40000   3.00000          45  38.40000  -6.00000
  18  14.40000   0.00000          46  43.20000   6.00000
  19  14.40000  -3.00000          47  43.20000   3.00000
  20  14.40000  -6.00000          48  43.20000   0.00000
  21  19.20000   6.00000          49  43.20000  -3.00000
  22  19.20000   3.00000          50  43.20000  -6.00000
  23  19.20000   0.00000          51  48.00000   6.00000
  24  19.20000  -3.00000          52  48.00000   3.00000
  25  19.20000  -6.00000          53  48.00000   0.00000
  26  24.00000   6.00000          54  48.00000  -3.00000
  27  24.00000   3.00000          55  48.00000  -6.00000
  28  24.00000   0.00000
*Background cells: noCell[ ]
 1     1     2     7     6        21    26    27    32    31
 2     2     3     8     7        22    27    28    33    32
 3     3     4     9     8        23    28    29    34    33
 4     4     5    10     9        24    29    30    35    34
 5     6     7    12    11        25    31    32    37    36
 6     7     8    13    12        26    32    33    38    37
 7     8     9    14    13        27    33    34    39    38
 8     9    10    15    14        28    34    35    40    39
 9    11    12    17    16        29    36    37    42    41
10    12    13    18    17        30    37    38    43    42
11    13    14    19    18        31    38    39    44    43
12    14    15    20    19        32    39    40    45    44
13    16    17    22    21        33    41    42    47    46
14    17    18    23    22        34    42    43    48    47
15    18    19    24    23        35    43    44    49    48
16    19    20    25    24        36    44    45    50    49
17    21    22    27    26        37    46    47    52    51
18    22    23    28    27        38    47    48    53    52
19    23    24    29    28        39    48    49    54    53
20    24    25    30    29        40    49    50    55    54
*Essential B.C.: numEBC
           7
 *Node,iUx,iUy,Ux,Uy
     1    1    1    -0.00000E-25    -0.60000E-04
     2    1    1    -0.70988E-05    -0.26667E-04
     3    1    1    -0.56790E-05    -0.66667E-05
     4    1    1     0.00000E-25     0.00000E-25
     5    1    1     0.56790E-05    -0.66667E-05
     6    1    1     0.70988E-05    -0.26667E-04
     7    1    1     0.00000E-25    -0.60000E-04
 *Concentrated Natural B.C.: numFBC
           7
 *Node,iTx,iTy,Tx,Ty
   169    1    1   0.00000   0.0
   170    1    1   0.00000   0.0
   171    1    1   0.00000   0.0
   172    1    1   0.00000   0.0
   173    1    1   0.00000   0.0
   174    1    1   0.00000   0.0
   175    1    1   0.00000   0.0
*RBF shape parameters: nRBF ALFc Dc and q
1 1.0 2.0 1.03
Number of basis
3
*End of input
```

Appendix 4.14. A output sample for displacements obtained using RPIM-MQ

No. of field nodes	u	v
1	-0.14420E-12	-0.60000E-04
2	-0.70988E-05	-0.26667E-04
3	-0.56790E-05	-0.66667E-05
4	0.27967E-25	0.23162E-13
5	0.56790E-05	-0.66667E-05
6	0.70988E-05	-0.26667E-04
7	0.14420E-12	-0.60000E-04
8	0.13062E-03	-0.94703E-04
9	0.80083E-04	-0.61811E-04
10	0.37954E-04	-0.42925E-04
11	-0.40548E-19	-0.36163E-04
12	-0.37954E-04	-0.42925E-04
13	-0.80083E-04	-0.61811E-04
14	-0.13062E-03	-0.94703E-04
15	0.25631E-03	-0.17293E-03
⋮	⋮	⋮
162	0.15929E-02	-0.83322E-02
163	0.10553E-02	-0.83308E-02
164	0.52603E-03	-0.83301E-02
165	0.15070E-16	-0.83298E-02
166	-0.52603E-03	-0.83301E-02
167	-0.10553E-02	-0.83308E-02
168	-0.15929E-02	-0.83322E-02
169	0.15958E-02	-0.88763E-02
170	0.10573E-02	-0.88767E-02
171	0.52704E-03	-0.88767E-02
172	0.14420E-16	-0.88772E-02
173	-0.52704E-03	-0.88767E-02
174	-0.10573E-02	-0.88767E-02
175	-0.15958E-02	-0.88763E-02

* The parameters used are $q = 1.03$, $\alpha_c = 1.0$, $d_c = 2.0$, and $\alpha_s = 3.0$. The linear polynomial terms are added in the RPIM-MQ.

Appendix 4.15. A output sample for stress obtained using RPIM-MQ

No. of field nodes	σ_{xx}	σ_{yy}	τ_{xy}
⋮	⋮	⋮	⋮
71	0.10568E+04	-0.39596E+02	-0.29675E+02
72	0.75714E+03	-0.10557E+01	-0.63827E+02
73	0.37425E+03	-0.85657E+01	-0.10754E+03
74	-0.11539E-10	-0.25580E-10	-0.12147E+03
75	-0.37425E+03	0.85657E+01	-0.10754E+03
76	-0.75714E+03	0.10557E+01	-0.63827E+02
77	-0.10568E+04	0.39596E+02	-0.29675E+02
78	0.10975E+04	-0.12785E+02	-0.30017E+02
79	0.73249E+03	0.12724E+02	-0.62639E+02
80	0.37442E+03	0.34166E+01	-0.10795E+03
81	0.21032E-11	0.36380E-11	-0.12223E+03
82	-0.37442E+03	-0.34166E+01	-0.10795E+03
83	-0.73249E+03	-0.12724E+02	-0.62639E+02
84	-0.10975E+04	0.12785E+02	-0.30017E+02
85	0.10131E+04	-0.12238E+02	-0.31899E+02
86	0.66202E+03	0.13350E+02	-0.66944E+02
87	0.33873E+03	0.35527E+01	-0.11747E+03
88	-0.47578E-10	-0.72987E-10	-0.12724E+03
89	-0.33873E+03	-0.35527E+01	-0.11747E+03
90	-0.66202E+03	-0.13350E+02	-0.66944E+02
91	-0.10131E+04	0.12238E+02	-0.31899E+02
92	0.90478E+03	-0.17776E+02	-0.30125E+02
93	0.60570E+03	0.64940E+01	-0.69757E+02
94	0.30801E+03	0.16526E+01	-0.12693E+03
95	0.14495E-10	-0.11369E-11	-0.13219E+03
96	-0.30801E+03	-0.16526E+01	-0.12693E+03
97	-0.60570E+03	-0.64940E+01	-0.69757E+02
98	-0.90478E+03	0.17776E+02	-0.30125E+02
⋮	⋮	⋮	⋮

Energy error:= 0.9082E-01

* The parameters used are $q = 1.03$, $\alpha_c = 1.0$, $d_c = 2.0$ and $\alpha_s = 3.0$. The linear polynomial terms are added in the RPIM-MQ.

Appendix 4.16. A output sample for displacements obtained using EFG

No. of field nodes	*u*	*v*
1	-0.18141E-11	-0.60000E-04
2	-0.70988E-05	-0.26667E-04
3	-0.56790E-05	-0.66667E-05
4	-0.34940E-20	0.41393E-12
5	0.56790E-05	-0.66667E-05
6	0.70988E-05	-0.26667E-04
7	0.18141E-11	-0.60000E-04
8	0.12862E-03	-0.93474E-04
9	0.81176E-04	-0.61905E-04
10	0.36948E-04	-0.43004E-04
11	-0.71820E-14	-0.36360E-04
12	-0.36948E-04	-0.43004E-04
13	-0.81176E-04	-0.61905E-04
14	-0.12862E-03	-0.93474E-04
15	0.25717E-03	-0.17076E-03
⋮	⋮	⋮
162	0.15972E-02	-0.83525E-02
163	0.10576E-02	-0.83511E-02
164	0.52674E-03	-0.83503E-02
165	-0.13310E-13	-0.83500E-02
166	-0.52674E-03	-0.83503E-02
167	-0.10576E-02	-0.83511E-02
168	-0.15972E-02	-0.83525E-02
169	0.15999E-02	-0.88983E-02
170	0.10594E-02	-0.88983E-02
171	0.52766E-03	-0.88983E-02
172	-0.13175E-13	-0.88984E-02
173	-0.52766E-03	-0.88983E-02
174	-0.10594E-02	-0.88983E-02
175	-0.15999E-02	-0.88983E-02

* The parameter used is $\alpha_s = 3.0$. The linear polynomial basis ($mbasis = 3$) and the cubic spline weight function are used in the MLS approximation.

Appendix 4.17. A output sample for stress obtained using EFG

No. of field nodes	σ_{xx}	σ_{yy}	τ_{xy}
⋮	⋮	⋮	⋮
71	0.11587E+04	0.61486E+00	-0.46400E+01
72	0.78147E+03	-0.50179E+00	-0.69152E+02
73	0.39129E+03	-0.23088E+01	-0.11362E+03
74	-0.21668E-06	-0.14721E-06	-0.12400E+03
75	-0.39129E+03	0.23088E+01	-0.11362E+03
76	-0.78147E+03	0.50179E+00	-0.69152E+02
77	-0.11587E+04	-0.61486E+00	-0.46400E+01
78	0.10826E+04	0.32433E+01	-0.11241E+02
79	0.72576E+03	-0.70275E+00	-0.69979E+02
80	0.36205E+03	-0.10124E+00	-0.11343E+03
81	0.19471E-06	0.12793E-06	-0.12696E+03
82	-0.36205E+03	0.10124E+00	-0.11343E+03
83	-0.72576E+03	0.70275E+00	-0.69979E+02
84	-0.10826E+04	-0.32433E+01	-0.11241E+02
85	0.10055E+04	0.31019E+01	-0.40273E+01
86	0.65869E+03	0.27783E+01	-0.67951E+02
87	0.33011E+03	-0.18849E+01	-0.11189E+03
88	-0.15056E-06	-0.11161E-06	-0.12269E+03
89	-0.33011E+03	0.18849E+01	-0.11189E+03
90	-0.65869E+03	-0.27783E+01	-0.67951E+02
91	-0.10055E+04	-0.31019E+01	-0.40273E+01
92	0.92005E+03	0.33212E+01	-0.18044E+01
93	0.61740E+03	-0.44910E+00	-0.68726E+02
94	0.30798E+03	0.24173E+00	-0.11219E+03
95	0.11759E-06	0.90103E-07	-0.12345E+03
96	-0.30798E+03	-0.24173E+00	-0.11219E+03
97	-0.61740E+03	0.44910E+00	-0.68726E+02
98	-0.92005E+03	-0.33212E+01	-0.18044E+01
⋮	⋮	⋮	⋮

Energy error:= 0.3280E-01

* The parameter used is $\alpha_s = 3.0$. The linear polynomial basis ($mbasis = 3$) and the cubic spline weight function are used in the MLS approximation.

COMPUTER PROGRAMS

Program 4.1. Source code of Parameter.h

```
      parameter(ninput=4,noutput=2, &
                      nx=2,ng=4, &
                      numd=1000,ncn=1000, numdq=1000,numc=1000, &
                      nqc=4,numg=nqc*nqc, &
                      ep=1.d-15)
```

Program 4.2. Source code of Variables.h

```
      dimension pEBC(2,100),npEBC(3,100),npNBC(3,100),pNBC(2,100)
      dimension Dmat(3,3)
      dimension x(nx,numd),noCell1(ng,ncn),ds(nx,numd)
      dimension xc(nx,numdq),noCell(ng,numc)
      dimension gauss(nx,nqc),gs(ng,numg)
      dimension gpos(nx),nv(numd),ph(10,numd)
      dimension ak(2*numd,2*numd),GSPk(4*numd*numd)
      dimension ne(2*numd),force(2*numd)
      dimension u2(nx,numd),disp(2*numd)
      dimension Stressnode(3,numd)
      common/para/xlength,ylength,p,young,anu,aimo
      common/rpim/ALFC,DC,Q,nRBF
      common /basis/mbasis
```

Program 4.3. The source code of the main program of MFree_Global.f90

```
!-----------------------------------------------------------------------
! main program--2D FORTRAN 90 CODE-MFree global weak-form methods
! Using square support domain and square background cells
! input file   -- input.dat
! output file  -- result.dat
! include file -- parameter.h, variable.h
!-----------------------------------------------------------------------

      implicit real*8 (a-h,o-z)
      include 'parameter.h'
      include 'variables.h'

      open(ninput,file='Input175_55.dat')
      open(noutput,file='result.dat',status='unknown')
! ************* Input data
      call input(x,numd,nx,numnode,ndivx,ndivy,ndivxq,ndivyq,&
            nconn2,nquado,pAlf,Dmat,ALFs,numcell,numq,noCell,ncn,xc,&
            npEBCnum,npEBC,pEBC,npNBCnum,npNBC,pNBC)
      numgauss=nquado*nquado  !total number of Gauss points in a cell

! ************* Determine sizes of influence domains -- uniform nodal spacing
      xspace=xlength/ndivx
      yspace=ylength/ndivy
      do i=1,numnode
         ds(1,i)=alfs*xspace
```

```
         ds(2,i)=alfs*yspace
      enddo
! ************ Coefficients of Gauss points,Weights and Jacobian for a cell
      call GaussCoefficient(nquado,gauss)
      do ik=1,ng
         do jk=1,numgauss
            gs(ik,jk)=0
         enddo
      enddo
      do ik=1,2*numd
         force(ik)=0.
         do jk=1,2*numd
            ak(ik,jk)=0.
         enddo
      enddo

! ************ Loop for background cells
      do 10 ibk=1,numcell
         write(*,*)'Cell No.=',ibk
! ************ Set Gauss points for this cell
         call CellGaussPoints(ibk,numcell,nquado,numq,numgauss, &
                              xc,noCell,gauss,gs)
! ************ Loop over Gauss points to assemble discrete equations
         do 20 ie=1,numgauss
            gpos(1)=gs(1,ie)  ! Gauss point x
            gpos(2)=gs(2,ie)  ! Gauss point y
            weight=gs(3,ie)   ! weight coefficent
            ajac=gs(4,ie)     ! Jacobian
! ************ Determine the support domain of Gauss point
            ndex=0
            call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
            do ik=1,3*ndex
               do jk=1,10
                  ph(jk,ik)=0.
               enddo
            enddo
! ************ Construct RPIM shape functions for a Gauss point
            call RPIM_ShapeFunc_2D(gpos,x,nv,ph,nx,numnode,ndex,&
                                   alfc,dc,q,nRBF, mbasis)

            do ik=1,2*ndex
              ne(ik)=0
            enddo
            do ine=1,ndex
               n1=2*ine-1
               n2=2*ine
               ne(n1)=2*nv(ine)-1
               ne(n2)=2*nv(ine)
            enddo
            mbdb=4*ndex*ndex
            do kbdb=1,mbdb
               GSPk(kbdb)=0.
            enddo
! ************ Compute the stiffness matrix for a Gauss point
            call PointStiffnessMatrix(ndex,weight,ajac,ph,Dmat,GSPk)
            nb=2*ndex
            do ikk=1,nb
               do jkk=1,nb
                  m1=ne(ikk)
                  m2=ne(jkk)
                  nbdb=(jkk-1)*nb+ikk
                  ak(m1,m2)=ak(m1,m2)+GSPk(nbdb)
               enddo
            enddo
20       continue      ! end of loop for Gauss points
! ************ Implement natural BC
         in=0
         jn=0
         nn=noCell(3,ibk)
         if(xc(1,nn).eq.xlength) in=nn
```

```
          nn= noCell(4,ibk)
          if(xc(1,nn).eq.xlength) jn=nn
          if((in.ne.0).and.(jn.ne.0)) then
             call naturalBC_distributed(numnode,numq,in,jn, &
                                alfs,x,xc,ds,gauss,nquado,force)
          endif
10     continue      ! end of loop for cells

! ************* Boundary conditions: essential BC
       write(*,*)' Boundary conditions....'
       nak=2*numd
       call EssentialBC(numnode,pAlf,alfs,x,ds,ak,force,npEBCnum,npEBC,pEBC)

! ************* Boundary conditions: concentrated natural BC
       call NaturalBC_concentrated(x,nx,numnode,force,ds,alfs,npNBCnum,npNBC,pNBC)
       nak=2*numd
       b=1.d-10
! ************* Solve equation to get the solutions
       write(*,*)' Solving....'
       call SolverBand(ak,force,2*numnode,2*numd)
       nnn=2*numd
       do  ik=1,nx
          do  jk=1,numnode
             u2(ik,jk)=0.
          enddo
       enddo
       do ik=1,numnode
          jk=2*ik-1
          u2(1,ik)=force(jk)
          u2(2,ik)=force(jk+1)
       enddo
! ************* Get the final displacement
       call GetDisplacement(x,ds,u2,disp,alfs,nx,numnode)

! ************* Get stress
       call GetStress(x,noCell,ds,Dmat,u2,alfs,nx,numnode,numgauss,&
                      xc,gauss,nquado,ng,numq,numcell, ENORM,Stressnode)
   STOP
   END
```

Program 4.4. Source code of Subroutine Input( )

```
   SUBROUTINE Input(x,numd,nx,numnode,ndivx,ndivy,ndivxq,ndivyq,&
                    nconn2,nquado,pAlf,Dmat,ALFs,numcell,numq,noCell,ncn,xc,&
                    npEBCnum,npEBC,pEBC,npNBCnum,npNBC,pNBC)
!-----------------------------------------------------------------------
! Input data from outside
! Output—all variables are output
!-----------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       common/para/xlength,ylength,p,young,anu,aimo
       COMMON/rpim/ALFC,DC,Q,nRBF
       common /basis/mbasis
       CHARACTER*40 NAM
       dimension npEBC(3,100),pEBC(2,100),npNBC(3,100),pNBC(2,100)
       dimension x(nx,numd),Dmat(3,3),noCell(4,ncn),xc(nx,numd)

       read(4,10)nam
       read(4,*) xlength,ylength,young,anu,p
       read(4,10)nam
       read(4,*)numnode,nconn2
       read(4,10)nam
       read(4,*) ndivx,ndivy
       read(4,10)nam
```

```
       read(4,*)numq,numcell
       read(4,10)nam
       read(4,*)ndivxq,ndivyq
       read(4,10)nam
       read(4,*)nquado,pAlf
       read(4,10)nam
       read(4,*)ALFs
       numgauss=nquado*nquado
       read(4,10)nam
       do i=1,numnode
          read(4,*)j,x(1,i),x(2,i)
       enddo
       read(4,10)nam
       do i=1,numq
          read(4,*)j,xc(1,i),xc(2,i)
       enddo
       read(4,10)nam
       do j=1,numcell
          read(4,*)i,noCell(1,j),noCell(2,j),noCell(3,j),noCell(4,j)
       enddo
       read(4,10)nam
       read(4,*)npEBCnum
       read(4,10)nam
       do i=1,npEBCnum
          read(4,*)npEBC(1,i),npEBC(2,i),npEBC(3,i),pEBC(1,i),pEBC(2,i)
       enddo
       read(4,10)nam
       read(4,*)npNBCnum
       read(4,10)nam
       do i=1,npNBCnum
          read(4,*)npNBC(1,i),npNBC(2,i),npNBC(3,i),pNBC(1,i),pNBC(2,i)
       enddo
       read(4,10)nam
       READ(4,*)nRBF, alfc,dc, q
       read(4,10)nam
       READ(4,*)mbasis

! ************* Compute material matrix D[] for the plane stress
       you=young/(1.-anu*anu)
       aimo=(1./12.)*ylength**3
       Dmat(1,1)=you
       Dmat(1,2)=anu*you
       Dmat(1,3)=0.
       Dmat(2,1)=anu*you
       Dmat(2,2)=you
       Dmat(2,3)=0.
       Dmat(3,1)=0.
       Dmat(3,2)=0.
       Dmat(3,3)=0.5*(1.-anu)*you
 10    format(a40)
   RETURN
   END
```

Program 4.5. Source code of Subroutine GaussCoefficient( )

```
   SUBROUTINE GaussCoefficient(k,v)
!-------------------------------------------------------------------------
! This subroutine returns a matrix with Gauss points and their weights
! input--k: k -- number of Gauss points;
! output--v(2,k): weight matrix of k Gauss points
!-------------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      dimension v(2,k)
      SELECT CASE (k)
```

```
        Case (2)
           v(1,1)=-.57735
           v(1,2)=-v(1,1)
           v(2,1)=1.00000
           v(2,2)=v(2,1)
        Case (3)
           v(1,1)=-.77459
           v(1,2)=-.00000
           v(1,3)=-v(1,1)
           v(2,1)=.55555
           v(2,2)=.88888
           v(2,3)=v(2,1)
        Case (4)
           v(1,1)=-.86113
           v(1,2)=-.33998
           v(1,3)=-v(1,2)
           v(1,4)=-v(1,1)
           v(2,1)=.34785
           v(2,2)=.65214
           v(2,3)=v(2,2)
           v(2,4)=v(2,1)
        Case (6)
           v(1,1)=-.93246
           v(1,2)=-.66120
           v(1,3)=-.23861
           v(1,4)=-v(1,3)
           v(1,5)=-v(1,2)
           v(1,6)=-v(1,1)
           v(2,1)=.17132
           v(2,2)=.36076
           v(2,3)=.46791
           v(2,4)=v(2,3)
           v(2,5)=v(2,2)
           v(2,6)=v(2,1)
        Case (8)
           v(1,1)=-.96028
           v(1,2)=-.79666
           v(1,3)=-.52553
           v(1,4)=-.18343
           v(1,5)=-v(1,4)
           v(1,6)=-v(1,3)
           v(1,7)=-v(1,2)
           v(1,8)=-v(1,1)
           v(2,1)=.10122
           v(2,2)=.22238
           v(2,3)=.31370
           v(2,4)=.36268
           v(2,5)=v(2,4)
           v(2,6)=v(2,3)
           v(2,7)=v(2,2)
           v(2,8)=v(2,1)
     end select
   RETURN
   END
```

Program 4.6. Source code of Subroutine CellGaussPoints

```
   SUBROUTINE CellGaussPoints(ibk,numcell,k,numq,numgauss,xc,noCell,gauss,gs)
!--------------------------------------------------------------------------
! This subroutine to set up Gauss points,Jacobian and weights for a cell
! input--ibk: the No.  of the consider cell;
!        numq: number of points for background cells;
!        numcell: number of background cells;
!        numgauss: number of Gauss points in a cell;
!        k: number of Gauss points used, numgauss=k*k for 2D cell;
!        xc(nx,numq): coordinates of points for background cells;
```

```
!        noCell(ng,numcell): No.  of points to form this cell;
!        gauss(2,k): coefficients of Gauss points;
!        nx,ng: parameters are defined in file parameter.h.
! output--gs(ng,numgauss): coordinate of the Gauss points, weight and Jacobian
!-------------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      include 'parameter.h'
      dimension xc(nx,numq),noCell(ng,numcell),gauss(nx,k),gs(ng,numgauss)
      dimension psiJ(ng),etaJ(ng),xe(ng),ye(ng),aN(ng),aNJpsi(ng),aNJeta(ng)

      index=0
      psiJ(1)=-1.
      psiJ(2)=1.
      psiJ(3)=1.
      psiJ(4)=-1.
      etaJ(1)=-1.
      etaJ(2)=-1.
      etaJ(3)=1.
      etaJ(4)=1.
      l=k
      ie=ibk
      do j=1,ng
         je=noCell(j,ie)
         xe(j)=xc(1,je)
         ye(j)=xc(2,je)
      enddo

      do 10 i=1,l
         do 10 j=1,l
            index=index+1
            eta=gauss(1,i)
            psi=gauss(1,j)
            do ik=1,ng
               aN(ik)=.25*(1.+psi*psiJ(ik))*(1.+eta*etaJ(ik))
               aNJpsi(ik)=.25*psiJ(ik)*(1.+eta*etaJ(ik))
               aNJeta(ik)=.25*etaJ(ik)*(1.+psi*psiJ(ik))
            enddo
            xpsi=0.
            ypsi=0.
            xeta=0.
            yeta=0.
            do jk=1,ng
               xpsi=xpsi+aNJpsi(jk)*xe(jk)
               ypsi=ypsi+aNJpsi(jk)*ye(jk)
               xeta=xeta+aNJeta(jk)*xe(jk)
               yeta=yeta+aNJeta(jk)*ye(jk)
            enddo
            ajcob=xpsi*yeta-xeta*ypsi
            xq=0.
            yq=0.
            do kk=1,ng
               xq=xq+aN(kk)*xe(kk)
               yq=yq+aN(kk)*ye(kk)
            enddo
            gs(1,index)=xq
            gs(2,index)=yq
            gs(3,index)=gauss(2,i)*gauss(2,j)
            gs(4,index)=ajcob
10    continue
   RETURN
   END
```

Program 4.7. Source code of Subroutine SupportDomain

```
   SUBROUTINE SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
!-------------------------------------------------------------------------------
! This subroutine to determines nodes in the support domain of a Gauss point
! input--numnode: total number of field nodes;
!        nx=2: for 2D problem;
!        x(nx,numnode): coordinates of all field nodes;
!        numgauss: number of Gauss points in a cell;
!        gpos(2): x and y coordinate of a Gauss point;
!        ds(nx,numnode): sizes of support domain;
! input and output-- ndex: when input ndex=0;
!            when return ndex is the number of nodes in the support domain
! output--nv(ndex): No.  of field nodes in the support domain
!-------------------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      dimension gpos(nx),x(nx,numnode),ds(nx,numnode),nv(numnode)
      eps=1.e-16
      ndex=0
      do ik=1,numnode
         nv(ik)=0
      enddo
      do ik=1,numnode
         dx=ds(1,ik)-dabs(gpos(1)-x(1,ik))
         dy=ds(2,ik)-dabs(gpos(2)-x(2,ik))
         if((dx.ge.eps).and.(dy.ge.eps)) then
            ndex=ndex+1
            nv(ndex)=ik
         end if
      enddo
   RETURN
   END
```

Program 4.8. Source code of Subroutine PointStiffnessMatrix

```
   SUBROUTINE PointStiffnessMatrix(ndex,weight,ajac,ph,Dmat,GSPk)
!-------------------------------------------------------------------------------
! This subroutine to calculate sparse stiff matrix
! input--ndex: the number of nodes in the support domain;
!        weight: weight of Gauss quadrature;
!        ajac: Jacobian;
!        dphix: first dirivetive of x of shape function;
!        dphiy: first dirivetive of y of shape function;
!        Dmat(3,3): the matrix of strain-stress;
! output--GSPk(2ndex,2ndex): sub-stiffness matrix of the Gauss point
!-------------------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      dimension ph(10,ndex),Dmat(3,3),GSPk(2*ndex,2*ndex)
      dimension bmat(3,2*ndex),dphix(ndex),dphiy(ndex)
      nb=2*ndex
        do i=1,ndex
           dphix(i)=ph(2,i)
           dphiy(i)=ph(3,i)
        enddo
      do ib=1,3
         do jb=1,nb
            Bmat(ib,jb)=0.
         enddo
      enddo
      do in=1,ndex
         j=2*in-1
         k=2*in
         Bmat(1,j)=dphix(in)
         Bmat(1,k)=0.
         Bmat(2,j)=0.
         Bmat(2,k)=dphiy(in)
         Bmat(3,j)=dphiy(in)
```

```
         Bmat(3,k)=dphix(in)
      enddo
      do ii=1,nb
         do jj=1,nb
            GSPk(ii,jj)=0.
         enddo
      enddo
      do ii=1,nb
         do jj=1,nb
            do kk=1,3
               do mm=1,3
                  GSPk(ii,jj)=GSPk(ii,jj)+weight*ajac*Bmat(kk,ii)* &
                              Dmat(kk,mm)*Bmat(mm,jj)
               enddo
            enddo
         enddo
      enddo
   RETURN
   END
```

Program 4.9. Source code of Subroutine EssentialBC

```
   SUBROUTINE EssentialBC(numnode,pAlf,alfs,x,ds,ak,af,npEBCnum,npEBC,pEBC)
!-----------------------------------------------------------------------------
! This subroutine to enforce point essential bc's using penalty method;
! input--numnode: total number of field nodes;
!        pAlf: penalty Fac; npEBCnum: number of e.  b.c points
!        alfs: coefficient of support domain
!        x(nx,numnode): coordinates of all field nodes;
! input and output-- ak[]: stiffness matrix;
!                    af{}:force vector.
!-----------------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      include 'parameter.h'
      COMMON/rpim/ALFC,DC,Q,nRBF
      common/basis/mbasis
      dimension npEBC(3,100),pEBC(2,100)
      dimension x(nx,numnode),ds(2,numnode),ak(2*numd,2*(numnode)),af(2*numnode)
      dimension nv(numnode),ph(10,numnode), x2(2)

      maxak=0.
      do iebc=1,2*numnode
         if(abs(ak(iebc,iebc)).gt.maxak) maxak=abs(ak(iebc,iebc))
      enddo

      do 10 iebc=1,npEBCnum
         ie=npEBC(1,iebc)
         x2(1)=x(1,ie)
         x2(2)=x(2,ie)
         ndex=0
!         call support(x2,x,ds,nv(1),numnode,nx,ndex)
         call SupportDomain(numnode,nx,x2,x,ds,ndex,nv)
         do ik=1,ndex
            do jk=1,10
               ph(jk,ik)=0.
            enddo
         enddo
         call RPIM_ShapeFunc_2D(x2,x,nv,ph,nx,numnode,ndex,&
                                alfc,dc,q,nRBF, mbasis)

         do iee=1,ndex
            ine=nv(iee)
            do ii=1,ndex
               jne=nv(ii)
               if(npEBC(2,iebc).eq.1) then
```

```
                ak((ine*2-1),(jne*2-1))=ak((ine*2-1),(jne*2-1))-pAlf*maxak* &
                      ph(1,iee)*ph(1,ii)
             endif
             if(npEBC(3,iebc).eq.1) then
                ak((ine*2),(jne*2))=ak((ine*2),(jne*2))-pAlf*maxak* &
                      ph(1,iee)*ph(1,ii)
             endif
          enddo
          if(npEBC(2,iebc).eq.1) then
             uu=pEBC(1,iebc)
             af(ine*2-1)=af(ine*2-1)-pAlf*uu*maxak*ph(1,iee)
          endif
          if(npEBC(3,iebc).eq.1) then
             uu=pEBC(2,iebc)
             af(ine*2)=af(ine*2)-pAlf*uu*maxak*ph(1,iee)
          endif
       enddo
10   continue
  RETURN
  END
```

Program 4.10. Source code of Subroutine NaturalBC_concentrated

```
  SUBROUTINE NaturalBC_concentrated(x,nx,numnode,af,ds,alfs,npNBCnum,npNBC,pNBC)
      implicit real*8 (a-h,o-z)
      dimension npNBC(3,100),pNBC(2,100)
      COMMON/rpim/ALFC,DC,Q,nRBF
      common/basis/mbasis
      dimension af(2*numnode),x(nx,numnode),ds(nx,numnode)
      dimension ph(10,numnode),gpos(2),nv(numnode)
      do 10 iebc=1,npNBCnum
         ie=npNBC(1,iebc)
         gpos(1)=x(1,ie)
         gpos(2)=x(2,ie)
         ndex=0
         call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
         do kph=1,3*ndex
            do ik=1,10
               ph(ik,kph)=0.
            enddo
         enddo
         call RPIM_ShapeFunc_2D(gpos,x,nv,ph,nx,numnode,ndex,&
                                alfc,dc,q,nRBF, mbasis)
         do iee=1,ndex
            ie=nv(iee)
            uu=pNBC(1,iebc)
            af(ie*2-1)=af(ie*2-1)+ph(1,iee)*uu
            uu=pNBC(2,iebc)
            af(ie*2)=af(ie*2)+ph(1,iee)*uu
         enddo
10    continue
  RETURN
  END
```

Program 4.11. Source code of Subroutine NaturalBC_distributed

```
  SUBROUTINE naturalBC_distributed(numnode,numq,in,jn,alfs,x,xc,ds, &
                                   gauss,nquado,force)
!----------------------------------------------------------------------------
```

```
! This subroutine to enforce point natural bc's;
! input—numnode, numq, in,jn,alfs,x,xc,ds,gauss, nquado.
! input and output--  force{}:force vector.
!-----------------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      include 'parameter.h'
      common/para/xlength,ylength,p,young,anu,aimo
      COMMON/rpim/ALFC,DC,Q,nRBF
      COMMON /basis/mbasis
      dimension xc(nx,numq),gauss(2,nquado),force(2*numnode),x(nx,numnode)
      dimension ph(10,numnode),gpos(2),nv(numnode),ds(nx,numnode)
      ax=0.5*(xc(1,in)-xc(1,jn))
      ay=0.5*(xc(2,in)-xc(2,jn))
      bx=0.5*(xc(1,in)+xc(1,jn))
      by=0.5*(xc(2,in)+xc(2,jn))
      do il=1,nquado
         gpos(1)=ax*gauss(1,il)+bx
         gpos(2)=ay*gauss(1,il)+by
         weight=gauss(2,il)
         ajac=0.5*sqrt((xc(1,in)-xc(1,jn))**2+(xc(2,in)-xc(2,jn))**2)
         aimo=(1./12.)*ylength**3
         ty=(-1000./(2.*aimo))*(ylength*ylength/4.-gpos(2)*gpos(2))
        call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
        do kph=1,ndex
           do ik=1,10
              ph(ik,kph)=0.
           enddo
        enddo
        call RPIM_ShapeFunc_2D(gpos,x,nv,ph,nx,numnode,ndex,&
                               alfc,dc,q,nRBF, mbasis)
        do ie=1,ndex
           nn=nv(ie)
           force(2*nn)=force(2*nn)+weight*ajac*ph(1,ie)*ty
        enddo
      enddo
   END
```

Program 4.12. Source code of Subroutine SolverBand

```
   SUBROUTINE SolverBand(ak,fp,neq,nmat)
!-----------------------------------------------------------------
! Sloving linear equations; it calls BandSolver & GaussSolver
! input—ak,fp,neq,nmat
! output--fp
!-----------------------------------------------------------------

       implicit real*8 (a-h,o-z)
       dimension ak(nmat,nEq),fp(nmat)
       real(8), allocatable :: tp(:,:)
       real(8), allocatable :: stfp(:,:)
       allocate (tp(1:neq,1:nmat))
       allocate (stfp(1:neq,1:neq))

       ep=1.d-10
       do i=1,nEq
          do j=1,nEq
             stfp(i,j)=0.
             tp(i,j)=0.
          enddo
       enddo
       do i=1,nEq
          do j=1,nEq
             stfp(i,j)=ak(i,j)
          enddo
       enddo
       ni=nEq
```

```
      Lp=0 ! half band width
      do 20 i=1,ni
         do j=ni,i,-1
         if(stfp(i,j).ne.0.) then ! stfp[,] stifness matrix
            if(abs(j-i).gt.Lp) Lp=abs(j-i)
               go to 21
            endif
         enddo
 21    continue
      do j=1,i
         if(stfp(i,j).ne.0.) then
            if(abs(j-i).gt.Lp) Lp=abs(j-i)
               go to 20
            endif
         enddo
 20    continue

      ilp=2*lp+1  ! band width
      nm=nEq
      if(ilp.lt.nEq) then
         call BandSolver(stfp,fp,tp,nm,lp,ilp,nmat) ! solver for band matrix
      else
         call GaussSolver(nEq,nmat,ak,fp,ep,kkkk) ! standard solver
      endif
      deallocate (tp)
      deallocate (stfp)
  END


  SUBROUTINE BANDSOLVER(A,F,B,N,L,IL,nmat)
!---------------------------------------------------------------------
! Slover for banded linear equations
!---------------------------------------------------------------------

      implicit real*8 (a-h,o-z)
      DIMENSION A(N,N),F(N)
      DIMENSION B(N,nmat),d(n,1)
      M=1
      LP1=L+1
      DO I=1,N
         DO K=1,IL
            B(I,K)=0.
            IF(I.LE.LP1) B(I,K)=A(I,K)
            IF(I.GT.LP1.AND.I.LE.(N-L)) B(I,K)=A(I,I+K-LP1)
            IF(I.GT.(N-L).AND.(I+K-LP1).LE.N) B(I,K)=A(I,I+K-LP1)
         ENDDO
      ENDDO
      DO I=1,N
         D(I,1)=F(I)
      ENDDO
      IT=1
      IF (IL.NE.2*L+1) THEN
         IT=-1
         WRITE(*,*)'***FAIL***'
         RETURN
      END IF
      LS=L+1
      DO 100 K=1,N-1
         P=0.0
            DO I=K,LS
               IF (ABS(B(I,1)).GT.P) THEN
                  P=ABS(B(I,1))
                  IS=I
               END IF
            ENDDO
         IF (P+1.0.EQ.1.0) THEN
            IT=0
            WRITE(*,*)'***FAIL***'
            RETURN
         END IF
```

```
         DO J=1,M
            T=D(K,J)
            D(K,J)=D(IS,J)
            D(IS,J)=T
         ENDDO
         DO J=1,IL
            T=B(K,J)
            B(K,J)=B(IS,J)
            B(IS,J)=T
         ENDDO
         DO J=1,M
            D(K,J)=D(K,J)/B(K,1)
         ENDDO
         DO J=2,IL
            B(K,J)=B(K,J)/B(K,1)
         ENDDO
         DO I=K+1,LS
            T=B(I,1)
            DO J=1,M
               D(I,J)=D(I,J)-T*D(K,J)
            ENDDO
            DO J=2,IL
               B(I,J-1)=B(I,J)-T*B(K,J)
            ENDDO
            B(I,IL)=0.0
         ENDDO
         IF (LS.NE.N) LS=LS+1
 100   CONTINUE
       IF (ABS(B(N,1))+1.0.EQ.1.0) THEN
          IT=0
          WRITE(*,*)'***FAIL***'
          RETURN
       END IF
       DO J=1,M
          D(N,J)=D(N,J)/B(N,1)
       ENDDO
          JS=2
       DO 150 I=N-1,1,-1
          DO K=1,M
             DO J=2,JS
                D(I,K)=D(I,K)-B(I,J)*D(I+J-1,K)
             ENDDO
          ENDDO
          IF (JS.NE.IL) JS=JS+1
 150   CONTINUE

       if(it.le.0) write(*,*) "BandSolver failed"
       DO I=1,N
          F(I)=D(I,1)
       ENDDO
    RETURN
    END



   SUBROUTINE GaussSolver(n,mk,a,b,ep,kwji)
!-----------------------------------------------------------------------
! Stnadard Gauss elimination slover for linear equations that are
!    not suitably solved by BandSolver.
!-----------------------------------------------------------------------

       implicit real*8 (a-h,o-z)
       dimension a(mk,mk),b(mk)
       integer, allocatable :: m(:)
       allocate (m(2*mk))
       ep=1.0e-10
       kwji=0
       do i=1,n
          m(i)=i
       enddo
```

```
      do 20 k=1,n
         p=0.0
         do 30 i=k,n
            do 30 j=k,n
            if(abs(a(i,j)).le.abs(p)) goto 30
               p=a(i,j)
               io=i
               jo=j
 30      continue
         if(abs(p)-ep) 200,200,300
200      kwji=1
         return
300      if(jo.eq.k) goto 400
         do i=1,n
            t=a(i,jo)
            a(i,jo)=a(i,k)
            a(i,k)=t
         enddo
         j=m(k)
         m(k)=m(jo)
         m(jo)=j
400      if(io.eq.k) goto 500
         do j=k,n
            t=a(io,j)
            a(io,j)=a(k,j)
            a(k,j)=t
         enddo
         t=b(io)
         b(io)=b(k)
         b(k)=t
 500     p=1.0/p
         in=n-1
         if(k.eq.n) goto 600
         do j=k,in
            a(k,j+1)=a(k,j+1)*p
         enddo
600      b(k)=b(k)*p
         if(k.eq.n) goto 20
         do i=k,in
            do j=k,in
               a(i+1,j+1)=a(i+1,j+1)-a(i+1,k)*a(k,j+1)
            enddo
            b(i+1)=b(I+1)-a(i+1,k)*b(k)
         enddo
20    continue
      do i1=2,n
         i=n+1-i1
         do j=i,in
            b(i)=b(i)-a(i,j+1)*b(j+1)
         enddo
      enddo
      do k=1,n
         i=m(k)
         a(1,i)=b(k)
      enddo
      do k=1,n
         b(k)=a(1,k)
      enddo
      kwji=0
      deallocate (m)
      return
  END
```

Program 4.13. Source code of Subroutine GetDisplacement

```
   SUBROUTINE GetDisplacement(x,ds,u2,disp,alfs,nx,numnode)
!------------------------------------------------------------------------------
```

```
! This subroutine to get the final displacements from
!         displacement parameters using the MFree interpolation;
! input--numnode: total number of field nodes;
!        alfs: coefficent of support support
!        x(nx,numnode): coordinates of all field nodes;
!        u2(2*numnode): displacement parameters;
! input and output-- disp: final displacements.
!-----------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       COMMON/rpim/ALFC,DC,Q,nRBF
       common/basis/mbasis
       dimension x(nx,numnode),ds(nx,numnode),gpos(nx),u2(nx,numnode)
       dimension disp(2*numnode)
       dimension ph(10,numnode), nv(numnode)

       write(2,*)'Displacements of field nodes'
       nn=2*numnode
       do i=1,nn
          disp(i)=0.
       enddo
       ind=0
       do 50 id=1,numnode
          ind=ind+1
          gpos(1)= x(1,id)
          gpos(2)=x(2,id)
          ndex=0
          call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
          do kph=1,ndex
             do ik=1,10
                ph(ik,kph)=0.
             enddo
          enddo
          call RPIM_ShapeFunc_2D(gpos,x,nv,ph,nx,numnode,ndex,&
                                 alfc,dc,q,nRBF, mbasis)
          nc1=2*ind-1
          nc2=2*ind
          do kk=1,ndex
             m=nv(kk)
             disp(nc1)=disp(nc1)+ph(1,kk)*u2(1,m)
             disp(nc2)=disp(nc2)+ph(1,kk)*u2(2,m)
          enddo
 50    continue
       do ii=1,numnode
          write(2,52) ii,disp(2*ii-1),disp(2*ii)
       enddo
 52    format(1x,i5,2e20.5)
   RETURN
   END
```

Program 4.14. Source code of Subroutine GetStress

```
   SUBROUTINE GetStress(x,noCell,ds,Dmat,u2,alfs,nx,numnode,numgauss,&
                    xc,gauss,nquado,ng,numq,numcell, ENORM,Stressnode)
!-----------------------------------------------------------------------
! This subroutine to get the stress and energy error;
! input--numnode: total number of field nodes;
!        numcell: number of cells;
!        numq: total number of points for cells;
!        alfs: coefficent of support support;
!        x(nx,numnode): coordinates of all field nodes;
!        xc(nx,numcell): coordinates of all points for cells;
!        u2(2*numnode): displacement parameters;
!        ds(nx,numnode): sizes of influence domain;
!        Dmat(3,3): material matrix;
```

```
!        nquado: number of Gauss points in a cell;
!        gauss(nx,nquado): coefficients of Gauss points;
!        numgauss: total number of Gauss points in all cells;
! output-- Enorm: energy error;
!          Stressnode:stress for field nodes;
! compute out--Stress: stress for Gauss points;
!             stressex, strne: exact stresse for beam problem.
!-------------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       common/para/xlength,ylength,p,young,anu,aimo
       COMMON/rpim/ALFC,DC,Q,nRBF
       common/basis/mbasis
       dimension noCell(4,numcell),ds(nx,numnode),x(nx,numnode),u(2*numnode)
       dimension  xc(nx,numnode),gauss(nx,nquado)
       dimension Dmat(3,3),u2(nx,numnode)
       dimension Stressnode(3,numnode),strne(3,numnode)
       dimension stress(3),stressex(3),err(3),Dinv(3,3),der(3)

       integer, allocatable :: nv(:)
       integer, allocatable :: ne(:)
       real(8), allocatable :: ph(:,:)
       real(8), allocatable :: gpos(:)
       real(8), allocatable :: gs(:,:)
       real(8), allocatable :: bmat(:,:)
       allocate ( nv(1:numnode) )
       allocate ( ne(1:2*numnode) )
       allocate ( ph(1:10,1:3*numnode) )
       allocate ( gpos(1:nx) )
       allocate ( gs(1:ng,1:numgauss) )
       allocate ( bmat(1:3,1:2*numnode) )

       close(37)
       open(37, file='midstr.dat')
       do id=1,3
          do jd=1,3
             Dinv(id,jd)=Dmat(id,jd)
          enddo
       enddo
       invd=3
       ep=1.d-10
       call GetINVASY(invd,invd,Dinv,EP)
       do iu=1,numnode
          ju=2*iu-1
          ku=2*iu
          u(ju)=u2(1,iu)
          u(ku)=u2(2,iu)
       enddo
       enorm=0.
!****************Compute energy error
       do 100 ibk=1,numcell
          ind=0
          call CellGaussPoints(ibk,numcell,nquado,numq,numgauss,&
                               xc,noCell,gauss,gs)
          do 200 is=1,numgauss
               do i=1,3
                stress(i)=0.
                stressex(i)=0.
             enddo
             ind=ind+1
             gpos(1)= gs(1,is)
             gpos(2)=gs(2,is)
             weight=gs(3,is)
             ajac=gs(4,is)
               call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
             do kph=1,3*ndex
                do ik=1,10
                   ph(ik,kph)=0.
                enddo
             enddo
```

```
            call RPIM_ShapeFunc_2D(gpos,x,nv,ph,nx,numnode,ndex,&
                                   alfc,dc,q,nRBF, mbasis)
            nb=2*ndex
            do in=1,nb
               ne(in)=0
            enddo
            do ine=1,ndex
               n1=2*ine-1
               n2=2*ine
               ne(n1)=2*nv(ine)-1
               ne(n2)=2*nv(ine)
            enddo
            do ib=1,3
               do jb=1,nb
                  Bmat(ib,jb)=0.
               enddo
            enddo
            do inn=1,ndex
               j=2*inn-1
               k=2*inn
               Bmat(1,j)=ph(2,inn)
               Bmat(1,k)=0.
               Bmat(2,j)=0.
               Bmat(2,k)=ph(3,inn)
               Bmat(3,j)=ph(3,inn)
               Bmat(3,k)=ph(2,inn)
            enddo
            do ii=1,3
               do kk=1,3
                  do mm=1,nb
                     mn=ne(mm)
                     stress(ii)=stress(ii)+&
                                Dmat(ii,kk)*Bmat(kk,mm)*u(mn)
                  enddo
              enddo
            enddo
!****************Exact stress for beam problem
            stressex(1)=(1./aimo)*p*(xlength-gpos(1))*gpos(2)
            stressex(2)=0.
            stressex(3)=-0.5*(p/aimo)*(0.25*ylength*ylength-gpos(2)*gpos(2))
            do ier=1,3
               err(ier)=stress(ier)-stressex(ier)
            enddo
             do jer=1,3
               der(jer)=0.
               do ker=1,3
                  der(jer)=der(jer)+Dinv(jer,ker)*err(ker)
               enddo
            enddo
            err2=0.
            do mer=1,3
               err2=err2+weight*ajac*(0.5*der(mer)*err(mer))
            enddo
            enorm=enorm+err2
 200      continue
 100   continue

!****************Compute nodal stresses
       write(2,*)'Stress of field nodes'
       do is=1,numnode
          gpos(1)= x(1,is)
          gpos(2)=x(2,is)
          do ii=1,3
             Stressnode(ii,is)=0.
          enddo
          ndex=0
          call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
          do kph=1,3*ndex
             do ik=1,10
                ph(ik,kph)=0.
```

```
            enddo
         enddo
         call RPIM_ShapeFunc_2D(gpos,x,nv,ph,nx,numnode,ndex,&
                                alfc,dc,q,nRBF, mbasis)

         nb=2*ndex
         do  in=1,nb
            ne(in)=0
         enddo
         do ine=1,ndex
            n1=2*ine-1
            n2=2*ine
            ne(n1)=2*nv(ine)-1
            ne(n2)=2*nv(ine)
         enddo
         do ib=1,3
            do jb=1,nb
               Bmat(ib,jb)=0.
            enddo
         enddo
         do inn=1,ndex
            j=2*inn-1
            k=2*inn
            Bmat(1,j)=ph(2,inn)
            Bmat(1,k)=0.
            Bmat(2,j)=0.
            Bmat(2,k)=ph(3,inn)
            Bmat(3,j)=ph(3,inn)
            Bmat(3,k)=ph(2,inn)
         enddo
         do ii=1,3
            do kk=1,3
               do mm=1,nb
                  mn=ne(mm)
                  Stressnode(ii,is)=Stressnode(ii,is)+&
                                    Dmat(ii,kk)*Bmat(kk,mm)*u(mn)
               enddo
            enddo
         enddo
         strne(1,is)=(1./aimo)*p*(xlength-gpos(1))*gpos(2)
         strne(2,is)=0.
         strne(3,is)=-0.5*(p/aimo)*(0.25*ylength*ylength-gpos(2)*gpos(2))

         write(2,220) is,Stressnode(1,is),Stressnode(2,is),Stressnode(3,is)
         if(abs(gpos(1)-24).le.1.d-5) then
            write(37,240) is,gpos(2),Stressnode(1,is),Stressnode(2,is), &
                          Stressnode(3,is),strne(1,is),strne(2,is),strne(3,is)
         endif
      enddo
      enorm=dsqrt(enorm)
      write(2,230) enorm
220   format(1x,i5,3e20.5)
230   format(1x,'Energy Error=',e20.10)
240   format(1x,i5,f8.3,6e15.5)
      deallocate ( nv)
      deallocate ( ne)
      deallocate ( ph)
      deallocate ( gpos)
      deallocate ( gs )
      deallocate ( bmat)
 RETURN
 END
```

Program 4.15. Source code of Subroutine GetInvasy

```
 SUBROUTINE GetInvasy(N,MA,A,EPS)
```

```
!-------------------------------------------------------------------------------
! This subroutine to get INVARSION OF A(N,N) USING THE GAUSS-JODON METHOD.
!            MATRIX A MUST BE DEFINITE BUT MAY BE ASYMMETRIC.
! input--N: dimension of A;
!        MA: max number of rows of A;
!        EPS: tolerance;
! Input and output--A[N,N]: the matrix in the input and invarsion in output;
!-------------------------------------------------------------------------------
       IMPLICIT REAL*8 (A-H,O-Z)
       DIMENSION A(MA,N)
       DO 10 K=1,N
          C=A(K,K)
          IF(DABS(C).LE.EPS)pause
          C=1.0/C
          A(K,K)=1.0
          DO J=1,N
             A(K,J)=A(K,J)*C
          ENDDO
          DO 10 I=1,N
             IF(I.EQ.K)GOTO 10
             C=A(I,K)
             A(I,K)=0.0
             DO J=1,N
                A(I,J)=A(I,J)-A(K,J)*C
             ENDDO
 10    CONTINUE
   RETURN
   END
```

Chapter 5

MESHFREE METHODS BASED ON LOCAL WEAK-FORMS

5.1 INTRODUCTION

In Chapter 4, the MFree methods (EFG and RPIM) based on global Galerkin weak-forms were introduced. In these MFree methods, global background cells are needed for numerical integrations in computing the system matrices. These MFree methods are, therefore, said not "truly" MFree methods. The use of the global weak-form requires the system equation in the global integral form to be satisfied over the entire problem domain, and hence, a set of background cells has to be used for the numerical integration. To avoid the use of global background cells, a so-called local weak-form is used to develop the meshless local Petrov-Galerkin (MLPG) method (Atluri and Zhu, 1998a, b; 2000a, b). Some other variations of MLPG are also proposed. MFree methods based on local weak-forms are called MFree local weak-form methods in this book.

When a local weak-form can be used for a field node, the numerical integrations are carried out over a local quadrature domain defined for the node, which can also be the local domain where the test (weight) function is defined. The local domain can have a regular and simple shape (such as spheres, rectangulars, ellipsoids, etc.) for an internal node, and the integration can be done numerically within the local domain. For a node on or near the boundary of complicated geometry, only a local mesh is required.

Therefore, no global background mesh is required. As in the EFG method, the MLS approximation is used to construct the shape functions in MLPG.

Atluri and Zhu (1998a) solved the Laplace equation, Poisson equation and potential flow problems using MLPG. The MLPG method has been improved and extended by Atluri's group (Atluri et al., 1999b) and other researchers over the years. MLPG has been applied to solve elastostatics and elastodynamics problems of solids (Atluri and Zhu, 2000a,b; Gu and GR Liu, 2001c), 4th order ODEs (or PDEs) for thin beams (Atluri et al, 1999a) and thick beams (shear deformable beams)(Cho and Atluri, 2001), plate structures (Gu and GR Liu, 2001f; Long and Atluri, 2002), linear fracture problems (Ching and Batra, 2001), fluid mechanics problems (Lin and Atluri, 2000; 2001; GR Liu and Wu et al., 2001, 2002), and so on. An error analysis of MLPG has been carried out by Kim and Atluri (2000).

MLPG does not need a global mesh for either function approximation or integration. The procedure is quite similar to numerical methods based on the strong-form formulation, such as the finite difference method (FDM). However, because the MLS approximation is used in MLPG, special treatments are needed to enforce the essential boundary conditions.

GR Liu and his co-workers applied the concept of MLPG and developed two variations of MFree local weak-form methods, the local point interpolation method (LPIM) (GR Liu and Gu, 2001b) and the local radial point interpolation method (LRPIM) (GR Liu and Yan et al., 2002; GR Liu and Gu, 2001c). In the LPIM, polynomial PIM shape functions (see, Sub-section 3.2.1) that have the delta function property are used. However, as polynomial basis functions are used, the interpolation moment matrix can be singular and hence the matrix triangularization algorithm (MTA) (GR Liu and Gu, 2001d, 2003a) has to be used. The radial PIM (RPIM) shape function (see, Sub-section 3.2.2) that also has the delta function property is another effective alternative, and has been used to formulate the local radial point interpolation method (LRPIM) method (GR Liu and Gu, 2001c; GR Liu and Yan et al., 2002) that is very robust for domains with randomly distributed nodes because of the excellent interpolation stability of RBFs.

Note that in a local weak-form method, global compatibility is not required.

LRPIM has been successfully applied to solid mechanics (e.g., GR Liu and Gu, 2000b, 2001b,c,e, 2002a; Xiao et al., 2003a,b,c), soil mechanics problems (Yan, 2001), fluid mechanics (GR Liu and Wu 2002), 4th order ODEs (or PDEs) for beam structures (Gu and GR Liu, 2001d), microelectronic mechanical system (MEMS) (Li and Wang et al., 2004), and so on.

In this Chapter, MLPG and LRPIM are discussed in detail. Formulations are obtained for two-dimensional elastostatics. A source code for these two MFree local weak-form methods is provided with detailed descriptions.

Numerical examples are presented to examine the present code. The formulations of MLPG and LRPIM are quite similar, and the difference is mainly in the type of MFree shape functions used, and the resultant differences in the formulation procedure (can consider LRPIM as a special MLPG). LRPIM is first discussed because it is simpler in the formulation than MLPG and hence easier to understand. Note that LRPIM was developed after the MLPG by replacing MLS shape function with the RPIM shape function.

5.2 LOCAL RADIAL POINT INTERPOLATION METHOD

5.2.1 LRPIM formulation

Consider a solid mechanics problem defined in the domain Ω shown in Figure 5.1. For a field node *I*, the governing equation (Equation (4.1)) is satisfied using a locally weighted residual method, leading to a weak form equation for this node. The local weighted residual form defined over a local quadrature domain Ω_q bounded by Γ_q, (shown in Figure 5.1) has the following form.

$$\int_{\Omega_q} \widehat{W}_I (\sigma_{ij,j} + b_i) \mathrm{d}\Omega = 0 \tag{5.1}$$

where $\widehat{W}_I$ is the weight or test function centred usually at node *I*. Equation (5.1) is applied to all the nodes in the problem domain.

When the local weighted residual formulation rather than the global energy principle is used to create the discretized system equations node by node, the compatibility of the shape functions in the whole domain is not required. As long as the field approximation is continuous at any point in the local quadrature domain, the shape function is differentiable (for an integration by parts) and the resultant integrand is integrable, the solution will exist[†]. In other words, the MFree local weak-form method only requires the local compatibility in the local quadrature domain. The RPIM shape function satisfies all these requirements, in addition to its delta function property. This feature of the local weighted residual formulation was

[†] Stabilization measures may be required depending on the type of problem. See, for example, Section 6.4.

follows the fact that the MFree local weak-form method using shape functions of at least linear consistency can easily pass the standard patch tests.

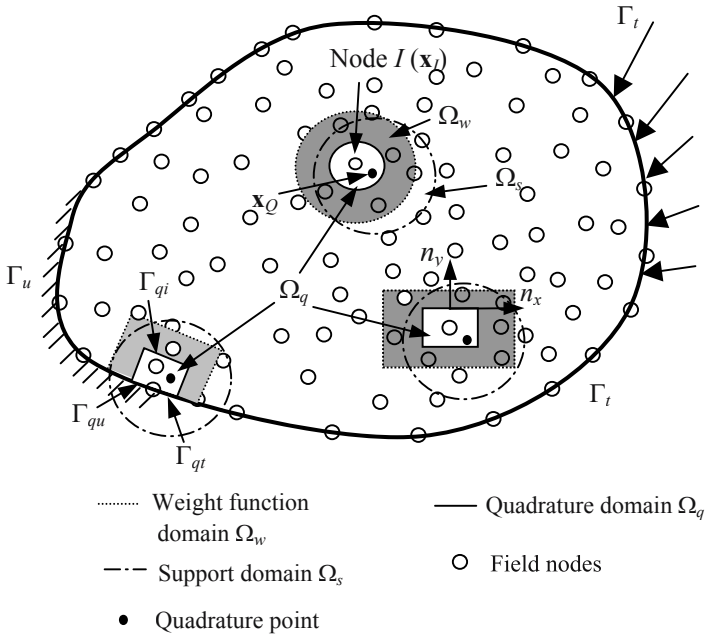


Figure 5.1. A problem domain and boundaries modeled using the MFree local weak-form methods. Weight function domain Ω_w and quadrature domain Ω_q for field nodes, and the support domain Ω_s for a Gauss quadrature point x_Q.

The first term on the left hand side of Equation (5.1) can be integrated by parts to arrive at

$$\int_{\Omega_q} \widehat{W}_I \sigma_{ij,j} \mathrm{d}\Omega = \int_{\Gamma_q} \widehat{W}_I n_j \sigma_{ij} \mathrm{d}\Gamma - \int_{\Omega_q} \widehat{W}_{I,j} \sigma_{ij} \mathrm{d}\Omega \tag{5.2}$$

where n_j is the jth component of the vector of the unit outward normal on the boundary (see Figure 1.4 and Figure 5.1). Substituting Equation (5.2) into Equation (5.1), we can obtain the following local weak-form.

$$\int_{\Gamma_q} \widehat{W}_I \sigma_{ij} n_j \mathrm{d}\Gamma - \int_{\Omega_q} \left[\widehat{W}_{I,j} \sigma_{ij} - \widehat{W}_I b_i \right] \mathrm{d}\Omega = 0 \tag{5.3}$$

Figure 5.1 shows that the boundary Γ_q for the local quadrature domain, Ω_q has composed by three parts, i.e., $\Gamma_q = \Gamma_{qi} \cup \Gamma_{qu} \cup \Gamma_{qt}$, where

Γ_{qi} is the internal boundary of the quadrature domain, which does not intersect with the global boundary Γ;

Γ_{qt} is the part of the natural boundary that intersects with the quadrature domain;

Γ_{qu} is the part of the essential boundary that intersects with the quadrature domain.

Therefore, Equation (5.3) can be re-written as

$$\int_{\Gamma_{qi}} \widehat{W}_I \sigma_{ij} n_j \mathrm{d}\Gamma + \int_{\Gamma_{qu}} \widehat{W}_I \sigma_{ij} n_j \mathrm{d}\Gamma + \int_{\Gamma_{qt}} \widehat{W}_I \sigma_{ij} n_j \mathrm{d}\Gamma - \int_{\Omega_q} \left[\widehat{W}_{I,j} \sigma_{ij} - \widehat{W}_I b_i \right] \mathrm{d}\Omega = 0 \tag{5.4}$$

For a local quadrature domain located entirely within the global domain, there is no intersection between Γ_q and the global boundary Γ. We then have $\Gamma_{qi}=\Gamma_q$ and there is no integral over Γ_{qu} and Γ_{qt}. In such a case, Equation (5.4) becomes

$$\int_{\Gamma_{qi}} \widehat{W}_I \sigma_{ij} n_j \mathrm{d}\Gamma - \int_{\Omega_q} \left[\widehat{W}_{I,j} \sigma_{ij} - \widehat{W}_I b_i \right] \mathrm{d}\Omega = 0 \tag{5.5}$$

In this local weak-form, Equations (5.4) and (5.5), the Petrov-Galerkin method can be used, in which the trial and test functions are selected from different spaces. The weight function $\widehat{W}_I$ is purposely selected so that it vanishes on Γ_{qi} to simplify the local weak-form. Note that the weight functions mentioned in Chapter 3, e.g. the cubic or quartic spline (W1 and W2), can be chosen to be zero along the boundary of the internal quadrature domains, hence they can be used as the weight functions in LRPIM. If the weight function satisfies this property, the local weak-forms of Equations (5.4) for a node whose local quadrature domain intersects with the global boundaries can be re-written as

$$\int_{\Gamma_{qu}} \widehat{W}_I \sigma_{ij} n_j \mathrm{d}\Gamma + \int_{\Gamma_{qt}} \widehat{W}_I \sigma_{ij} n_j \mathrm{d}\Gamma - \int_{\Omega_q} \left[\widehat{W}_{I,j} \sigma_{ij} - \widehat{W}_I b_i \right] \mathrm{d}\Omega = 0 \tag{5.6}$$

Equation (5.5) that is for a node whose local quadrature domain does not intersect with the global boundaries can be re-written as

$$- \int_{\Omega_q} \left[\widehat{W}_{I,j} \sigma_{ij} - \widehat{W}_I b_i \right] \mathrm{d}\Omega = 0 \tag{5.7}$$

We note the relation between the stress and the traction on the boundary

$$\sigma_{ij} n_j = t_i \tag{5.8}$$

Imposing Equation (5.8) and the natural boundary condition Equation (4.2) into Equation (5.4), we obtain:

$$\int_{\Omega_q} \widehat{W}_{I,j}\sigma_{ij}\mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{W}_I t_i \mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{W}_I t_i \mathrm{d}\Gamma = \int_{\Gamma_{qt}} \widehat{W}_I \overline{t}_i \mathrm{d}\Gamma + \int_{\Omega_q} \widehat{W}_I b_i \mathrm{d}\Omega \tag{5.9}$$

Equation (5.9) suggests that the strong-form of the system equation given in Equations (4.1) is changed to a relaxed weak-form with integrations over a small local quadrature domain. This integral operation can smear out the numerical error, and therefore make the discretized system more accurate than the MFree procedures that operate directly on the strong-forms of system equations. The LRPIM ensures the satisfaction of the equilibrium equation at a node in an integral sense over a local quadrature domain, but it does not ensure the satisfaction of the strong system equation exactly at the node. The size of the local quadrature domain determines the extent of the 'relaxation'.

In order to obtain the discretized system equations, the global problem domain Ω is represented by properly distributed field nodes. Using the RPIM shape functions (see sub-section 3.2.2), we can approximate the trial function for the displacement at a point $\mathbf{x}$ as

$$\mathbf{u}^h_{(2\times 1)}(\mathbf{x}) = \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} = \mathbf{\Phi}_{(2\times 2n)}\mathbf{u}_{(2n\times 1)} \tag{5.10}$$

where n is the number of nodes in the support domain of a sampling point at $\mathbf{x}$, and Φ is the matrix of RPIM shape functions constructed using these nodes. Note that these n field nodes are numbered from 1 to n, and it is a local numbering system for the support domain. The field node also has a global number that is uniquely given to all field nodes from 1 to N. This global numbering system is used to assemble all the local nodal matrices to form the global matrix. Hence, an index is needed to record the global number for a field node used in the support domain for the purpose of assembling the global matrixes.

As in Equations (4.8)~(4.10), we can obtain the strain and stress as

$$\boldsymbol{\varepsilon}_{(3\times 1)} = \mathbf{B}_{(3\times 2n)}\mathbf{u}_{(2n\times 1)} \tag{5.11}$$

$$\boldsymbol{\sigma}_{(3\times 1)} = \mathbf{D}_{(3\times 3)}\,\boldsymbol{\varepsilon}_{(3\times 1)} = \mathbf{D}_{(3\times 3)}\mathbf{B}_{(3\times 2n)}\mathbf{u}_{(2n\times 1)} \tag{5.12}$$

where **D** is the matrix of elastic constants of defined in Equation (2.27) or (2.28), and **B** is the *strain matrix* given in Equation (4.8), i.e.

$$\mathbf{B}_{(3\times 2n)}=\begin{bmatrix} \dfrac{\partial \phi_1}{\partial x} & 0 & \cdots & \dfrac{\partial \phi_n}{\partial x} & 0 \\ 0 & \dfrac{\partial \phi_1}{\partial y} & \cdots & 0 & \dfrac{\partial \phi_n}{\partial y} \\ \dfrac{\partial \phi_1}{\partial y} & \dfrac{\partial \phi_1}{\partial x} & \cdots & \dfrac{\partial \phi_n}{\partial y} & \dfrac{\partial \phi_n}{\partial x} \end{bmatrix} \tag{5.13}$$

We now change Equation (5.4) to the following matrix form to derive the discretized system equations in a matrix form.

$$\int_{\Omega_q} \widehat{\mathbf{V}}_I^{\mathrm{T}} \boldsymbol{\sigma} \mathrm{d}\Omega - \int_{\Gamma_{qt}} \widehat{\mathbf{W}}_I \mathbf{t} \mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I \mathbf{t} \mathrm{d}\Gamma = \int_{\Gamma_{qt}} \widehat{\mathbf{W}}_I \overline{\mathbf{t}} \mathrm{d}\Gamma + \int_{\Omega_q} \widehat{\mathbf{W}}_I \mathbf{b} \mathrm{d}\Omega \tag{5.14}$$

where $\widehat{\mathbf{W}}$ is a matrix of weight functions given by

$$\widehat{\mathbf{W}}_I = \widehat{\mathbf{W}}(\mathbf{x}, \mathbf{x}_I)_{(2\times 2)} = \begin{bmatrix} \widehat{W}(\mathbf{x}, \mathbf{x}_I) & 0 \\ 0 & \widehat{W}(\mathbf{x}, \mathbf{x}_I) \end{bmatrix} \tag{5.15}$$

In Equation (5.14), $\widehat{\mathbf{V}}_I$ is a matrix that collects the derivatives of the weight functions:

$$\widehat{\mathbf{V}}_I = \widehat{\mathbf{V}}(\mathbf{x}, \mathbf{x}_I)_{(3\times 2)} = \begin{bmatrix} \widehat{W}_{,x}(\mathbf{x}, \mathbf{x}_I) & 0 \\ 0 & \widehat{W}_{,y}(\mathbf{x}, \mathbf{x}_I) \\ \widehat{W}_{,y}(\mathbf{x}, \mathbf{x}_I) & \widehat{W}_{,x}(\mathbf{x}, \mathbf{x}_I) \end{bmatrix} \tag{5.16}$$

It is in fact the strain matrix caused by the variation of the weight (test) functions.

The tractions **t** at a point **x** can be written as

$$\mathbf{t} = \begin{Bmatrix} t_x \\ t_y \end{Bmatrix} = \underbrace{\begin{bmatrix} n_x & 0 & n_y \\ 0 & n_y & n_x \end{bmatrix}_{(2\times 3)}}_{\mathbf{n}} \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{Bmatrix} = \mathbf{n}_{(2\times 3)} \mathbf{D}_{(3\times 3)} \mathbf{B}_{(3\times 2n)} \mathbf{u}_{(2n\times 1)} \tag{5.17}$$

in which (n_x, n_y) is the vector of the unit outward normal on the boundary (see Figure 1.4).

$$\mathbf{n} = \begin{bmatrix} n_x & 0 & n_y \\ 0 & n_y & n_x \end{bmatrix} \tag{5.18}$$

Substitution of Equations (5.16)~(5.18) into Equation (5.14) leads to the following discretized systems of equations for the *I*th field node.

$$\int_{\Omega_q} \widehat{\mathbf{V}}_I^{\mathrm{T}} \mathbf{DBu} \mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{nDBu} \mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{nDBu} \mathrm{d}\Gamma = \int_{\Gamma_{qt}} \widehat{\mathbf{W}}_I^{\mathrm{T}} \overline{\mathbf{t}} \mathrm{d}\Gamma + \int_{\Omega_q} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{b} \mathrm{d}\Omega \tag{5.19}$$

The matrix form of Equation (5.19) can be written as

$$(\mathbf{K}_I)_{2\times 2n} (\mathbf{u})_{2n\times 1} = (\mathbf{f}_I)_{2\times 1} \tag{5.20}$$

where $\mathbf{K}_I$ is a matrix called *nodal stiffness matrix* for the *I*th field node, which is computed using

$$\mathbf{K}_I = \int_{\Omega_q} \widehat{\mathbf{V}}_I^{\mathrm{T}} \mathbf{DB} \mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{nDB} \mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{nDB} \mathrm{d}\Gamma \tag{5.21}$$

In Equation (5.20), $\mathbf{f}_I$ is a *nodal force vector* with contributions from body forces applied in the problem domain, and tractions applied on the natural boundary.

$$\mathbf{f}_I = \int_{\Gamma_{qt}} \widehat{\mathbf{W}}_I^{\mathrm{T}} \overline{\mathbf{t}} \mathrm{d}\Gamma + \int_{\Omega_q} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{b} \mathrm{d}\Omega \tag{5.22}$$

In Equation (5.20), $\mathbf{u}$ is the vector collecting displacements for the field nodes that are included in any of the support domains of the quadrature points in the quadrature domain of the *I*th field node.

Equation (5.20) gives the general form of system equations for a field node. For the local quadrature domain of a field node located entirely within the global domain, there is no intersection between Γ_q and the global boundary, Γ, and the weak-form is given in Equation (5.5). In this case, $\mathbf{K}_I$ and $\mathbf{f}_I$ can be obtained using, respectively,

$$\mathbf{K}_I = \int_{\Omega_q} \widehat{\mathbf{V}}_I^{\mathrm{T}} \mathbf{DB} \mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{nDB} \mathrm{d}\Gamma \tag{5.23}$$

and

$$\mathbf{f}_I = \int_{\Omega_q} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{b} \mathrm{d}\Omega \tag{5.24}$$

We use Gauss quadrature to obtain the integrals in Equations (5.21) and (5.22). Note that in the formulation for $\mathbf{K}_I$ and $\mathbf{f}_I$, there are area integrals, and curve integrals. Consider a rectangular local quadrature domain, in the

standard Gauss quadrature, Equations (5.21) and (5.22) can be expressed as follows.

$$\begin{aligned}\mathbf{K}_I = &\sum_{k=1}^{n_g} \widehat{w}_k \widehat{\mathbf{V}}_I^{\mathrm{T}}(\mathbf{x}_{Qk})\mathbf{DB}(\mathbf{x}_{Qk})\left|\mathbf{J}_q^D\right| \\ &- \sum_{k=1}^{n_{gt}} \widehat{w}_k \widehat{\mathbf{W}}_I^{\mathrm{T}}(\mathbf{x}_{Qk})\mathbf{nDB}(\mathbf{x}_{Qk})\left|\mathbf{J}_{qi}^B\right| \\ &- \sum_{k=1}^{n_{gt}} \widehat{w}_k \widehat{\mathbf{W}}_I^{\mathrm{T}}(\mathbf{x}_{Qk})\mathbf{nDB}(\mathbf{x}_{Qk})\left|\mathbf{J}_{qu}^B\right|\end{aligned} \tag{5.25}$$

$$\mathbf{f}_I = \sum_{k=1}^{n_{gt}} \widehat{w}_k \widehat{\mathbf{W}}_I^{\mathrm{T}}(\mathbf{x}_{Qk})\overline{\mathbf{t}}\left|\mathbf{J}_{qt}^B\right| + \sum_{k=1}^{n_g} \widehat{w}_k \widehat{\mathbf{W}}_I^{\mathrm{T}}(\mathbf{x}_{Qk})\mathbf{b}\left|\mathbf{J}_q^D\right| \tag{5.26}$$

where n_g is the total number of Gauss points in the quadrature domain, n_{gt} is number of Gauss points used in a sub-curve, $\widehat{w}_k$ is the Gauss weighting factor for Gauss point $\mathbf{x}_{Qk}$, $\mathbf{J}_q^D$ is the Jacobian matrix for the area integration of the local quadrature domain, and $\mathbf{J}_{qi}^B$, $\mathbf{J}_{qu}^B$ and $\mathbf{J}_{qt}^B$ are, respectively, the Jacobian matrices for the curve integration of the sub-boundaries Γ_{qi}, Γ_{qu} and Γ_{qt}.

Note that different Gauss points in the same local quadrature domain may use different support domains. This means that the matrices in Equations (5.25) and (5.26) could be different for different Gauss points.

Equation (5.20) presents two linear equations for the *I*th field node. Using Equation (5.20) for all the *N* field nodes in the entire problem domain, we obtain a total of 2*N* independent linear equations. Assemble all these 2*N* equations based on the global numbering system to obtain the final global system equations in the form of

$$\underbrace{\begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1(2N-1)} & K_{1(2N)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ K_{(2I-1)1} & K_{(2I-1)2} & \cdots & K_{(2I-1)(2N-1)} & K_{(2I-1)(2N)} \\ K_{(2I)1} & K_{(2I)2} & \cdots & K_{(2I)(2N-1)} & K_{(2I)(2N)} \\ \vdots & & \ddots & \vdots & \vdots \\ K_{(2N)1} & K_{(2N)2} & \cdots & K_{(2N)(2N-1)} & K_{(2N)(2N)} \end{bmatrix}}_{\mathbf{K}_{2N\times 2N}} \underbrace{\begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_I \\ v_I \\ \vdots \\ u_N \\ v_N \end{Bmatrix}}_{\mathbf{U}_{2N\times 1}} = \underbrace{\begin{Bmatrix} f_{1x} \\ f_{1y} \\ \vdots \\ f_{Ix} \\ f_{Iy} \\ \vdots \\ f_{Nx} \\ f_{Ny} \end{Bmatrix}}_{\mathbf{F}_{2N\times 1}} \tag{5.27}$$

*I*th node

or

$$\mathbf{K}_{2N\times 2N}\mathbf{U}_{2N\times 1}=\mathbf{F}_{2N\times 1} \tag{5.28}$$

Equation (5.27) shows that the two nodal equations for the Ith node have been assembled into the (2I-1)th and 2Ith rows in the global equations.

Note that the assembling to form Equation (5.28) is different from that in the conventional FEM and the MFree global weak-form methods, such as EFG. In the FEM and EFG, the element or nodal matrices are stamped symmetrically into the global matrix. In the MFree local weak-form methods, however, the nodal matrix is stacked together row-by-row to form the global matrix. This stacking procedure is similar to that in the finite difference method (FDM).

Equation (5.28) is the final discretized system equation of LRPIM. Note that the essential boundary conditions, Equation (4.3), are not considered in the LRPIM formulations. Because the RPIM shape functions have the Kronecker delta function property, the essential boundary conditions can be enforced in LRPIM as easily as in the RPIM or the conventional FEM. The procedure has also been detailed in Sub-section 4.2.2. After enforcing essential boundary conditions, we can solve the modified system equation for displacements for all field nodes and then to compute the stresses using Equations (4.10) and (5.12).

5.2.2 Numerical implementation

5.2.2.1 Type of local domains

Gauss quadrature is needed to evaluate the integrations in Equations (5.21) and (5.22). As shown in Figure 5.1, for a field node $\mathbf{x}_I$, a local quadrature cell Ω_q is needed for the Gauss quadrature. For each Gauss quadrature point $\mathbf{x}_Q$, the RPIM shape functions are constructed to obtain the integrand. Therefore, for a field node $\mathbf{x}_i$, there exist three local domains:

a) the local quadrature domain Ω_q (size r_q);

b) the local weight (test) function domain Ω_w where $w_i\neq 0$ (size r_w);

c) the local support domain Ω_s for $\mathbf{x}_Q$ (size r_s).

These three local domains are arbitrary as long as the condition $r_q\leq r_w$ is satisfied. It has been noted that when an appropriate weight function is used, the local weak-form, Equation (5.9), can be simplified because the integration along the internal boundary Γ_{qi} vanishes. Hence, for simplicity in this book, we always use $r_q=r_w$.

The size of the local quadrature domain (r_q) for node I and the size of the support domain (r_s) are defined as

$$r_q=\alpha_q d_{cI} \tag{5.29}$$

$$r_s=\alpha_s d_{cI} \tag{5.30}$$

where d_{cI} is the nodal spacing near node I, which is defined in Sub-section 3.1.3, α_q and α_s are dimensionless sizes chosen to control the actual domain sizes. The effects of α_q and α_s will be investigated later.

5.2.2.2 Property of the stiffness matrix

The system stiffness matrix **K** in the present LRPIM is sparse as long as the support domain of RPIM is compact. If the field nodes are properly numbered, **K** is banded. Note also that **K** is usually asymmetric (Atluri and Shen, 2002). The asymmetry has two causes:

1) The Petrov-Galerkin formulation uses different functions for the trial and test functions. Furthermore, the sizes and/or the shapes of the local support domains for constructing the trial and test functions can also be different. In addition, the sizes and/or the shapes of the local quadrature domain may differ for different field nodes. Hence, the domain integration in Equation (5.21) is, generally, asymmetric, i.e.

$$\int_{\Omega_q^{(k)}} \widehat{\mathbf{V}}_k^{\mathrm{T}} \mathbf{D}\mathbf{B}_l \mathrm{d}\Omega \neq \int_{\Omega_q^{(l)}} \widehat{\mathbf{V}}_l^{\mathrm{T}} \mathbf{D}\mathbf{B}_k \mathrm{d}\Omega \tag{5.31}$$

where $\Omega_q^{(k)}$ and $\Omega_q^{(l)}$ are local quadrature domains for the kth and the lth field nodes, respectively, and $\widehat{\mathbf{V}}_k$ and $\widehat{\mathbf{V}}_l$ are matrices of derivatives of the weight functions used for the kth and the lth field nodes, respectively. $\mathbf{B}_l$ and $\mathbf{B}_k$ are the strain matrices of the lth and the kth field nodes.

2) The part of **K** from the boundary integrations in Equation (5.21) is asymmetric. The sizes and/or shapes of the local quadrature domains may be different for different field nodes; this means that boundary integrations in Equation (5.21) are, in general, asymmetric, i.e.

$$\begin{aligned} &\int_{\Gamma_{qi}^{(k)}} \widehat{\mathbf{W}}_k^{\mathrm{T}} \mathbf{n}\mathbf{D}\mathbf{B}_l \mathrm{d}\Gamma - \int_{\Gamma_{qu}^{(k)}} \widehat{\mathbf{W}}_k^{\mathrm{T}} \mathbf{n}\mathbf{D}\mathbf{B}_l \mathrm{d}\Gamma \\ &\quad \neq \int_{\Gamma_{qi}^{(l)}} \widehat{\mathbf{W}}_l^{\mathrm{T}} \mathbf{n}\mathbf{D}\mathbf{B}_k \mathrm{d}\Gamma - \int_{\Gamma_{qu}^{(l)}} \widehat{\mathbf{W}}_l^{\mathrm{T}} \mathbf{n}\mathbf{D}\mathbf{B}_k \mathrm{d}\Gamma \end{aligned} \tag{5.32}$$

Therefore, **K** is asymmetric, i.e.,

$$K_{kl} \neq K_{lk} \tag{5.33}$$

In conclusion, the stiffness matrix **K** in the LRPIM is generally sparse, banded and asymmetric.

Note that in LRPIM, the essential boundary conditions can be directly implemented as in the RPIM and FEM due to the fact that the RPIM shape functions possess the Kronecker delta function property. Because the system equation of LRPIM is assembled based on nodes as in the finite difference method (FDM), the rows in the matrix **K** for the nodes on the essential boundary need not be computed. This can save some computational costs. This simple treatment is possible because 1) the RPIM shape functions have the delta function property and 2) the rows of the **K** are based on nodes.

5.2.2.3 Test (weight) function

As LRPIM can be regarded as a local weighted residual method, the test (weight) function plays an important role in the performance of this method. Theoretically, any test function is acceptable as long as the condition of continuity is satisfied. However, the local weak-form is based on a local quadrature domain of a field node with the node at the centre. It can be shown that test functions which decrease in magnitude with increasing distance from the centre yield better results. We use the test functions that depend only on the distance between the two points: the cubic spline function (W1), the 4th-order spline weight function (W2) and other weight function given in Sub-section 3.3.2.

To simplify Equation (5.9), we can deliberately select the test functions so that they vanish over Γ_{qi}. This can be achieved using, for example, the 4th-order spline weight function (W2) with $r_q=r_w$ (see Sub-section 3.3.2) because $\widehat{W}(r)$ is zero when $r= r_q$.

There is a wide range of weight functions that can be used in LRPIM, including all weight functions that are used to form different weighted residual methods (see, Section 1.4). Atluri and Shen (2002) used six weight functions in MLPG. These weight functions can also be used in LRPIM.

Although there are many types of weight functions, the spline weight functions (e.g., W1 or W2) are the most popular; it is the most convenient to use and accurate. Hence, in this book, we focus on the use of these spline weight functions.

5.2.2.4 Numerical integration

The integrations in LRPIM can be performed over regularly-shaped local quadrature domains for internal nodes; circles, ellipses, rectangles, or triangles in two-dimensional problems; spheres, rectangular parallelepipeds,

or ellipsoids in three-dimensional problems. These local domains can be automatically generated during computation.

Issues of numerical integrations in MFree methods have been discussed in detail in the existing publications (Atluri et al.,1999b; Dolbow and Belytschko,1999; GR Liu, 2002). Insufficiently accurate numerical integration may cause deterioration in the numerical solution and rank-deficiency in the stiffness matrix. The difficulty of the numerical integration for LRPIM comes mainly from the complexity of integrands. First, the shape functions constructed are complicated, and have different forms in each integration region. The derivatives of shape functions can even have oscillations. Second, the overlapping of local support domains complicates the integrands further. In order to ensure the accurate numerical integration, Ω_q should be further divided into small regular partitions (see Figure 5.8). In each small region, the number of Gauss quadrature points should be chosen to ensure sufficient accuracy (Atluri et al. 1999b).

If the rectangular quadrature domain is used, the standard Gauss quadrature can easily be performed. Circles centred on the field node are often used; they have no directional bias, and have simple weight functions.

To obtain the numerical integration for a circular quadrature domain, a mapping technique has been used, as shown in Figure 5.2.

The circular quadrature domain is divided into four quarters.

The quarter is mapped into a rectangle region.

The rectangle region is mapped to a standard square for Gauss quadrature.

The standard Gauss quadrature is used.

For simplicity, the rectangular quadrature is used in the following study.

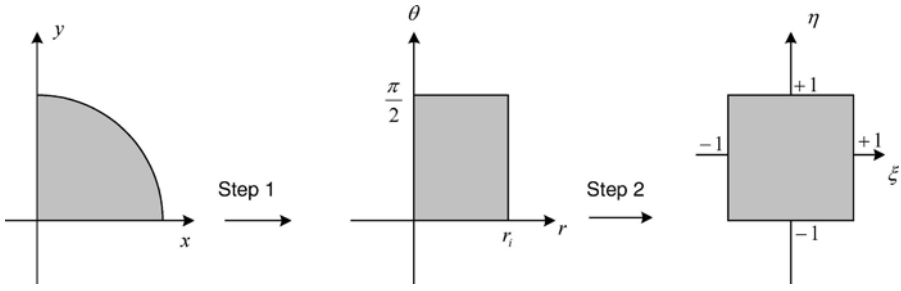


Figure 5.2. Transformation of a quarter circular domain into a standard square.

5.3 MESHLESS LOCAL PETROV-GALERKIN METHOD

The MLPG is developed by Atluri et al. (Atluri and Zhu, 1998a,b; Alturi and Shen, 2002); this section provides a concise introduction; the complete record of MLPG is given in Atluri and Shen (2002).

5.3.1 MLPG formulation

Consider a two-dimensional problem of solid mechanics in domain Ω bounded by Γ whose strong-form of governing equation and the essential boundary conditions are given in Equations (4.1) to (4.3). In the MLPG, the local weak-form can be obtained from the following weighted residual method.

$$\int_{\Omega_q} \widehat{W}_I(\sigma_{ij,j} + b_i)\mathrm{d}\Omega - \alpha \int_{\Gamma_{qu}} \widehat{W}_I\left(u_i - \overline{u}_i\right)\mathrm{d}\Gamma = 0 \tag{5.34}$$

where $\widehat{W}$ is the weight or test function. Note that the second integral in Equation (5.34) is the curve integral to enforce the essential boundary conditions, because the MLS shape functions used in MLPG lack the Kronecker delta function property. In Equation (5.34), the penalty method is used to enforce the essential boundary conditions. Ω_q is the local domain of quadrature for node I, Γ_{qu} is the part of the essential boundary that intersect with the quadrature domain Ω_q, and α is the penalty factor used in Chapter 4. Here we use the same penalty factor for all the displacement (essential) boundary conditions.

The displacements at a sampling point **x** are approximated using the MLS shape functions (see, Section 3.3) in the following form

$$\mathbf{u}^h_{(2\times 1)}(\mathbf{x}) = \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} = \mathbf{\Phi}_{(2\times 2n)}\mathbf{u}_{(2n\times 1)} \tag{5.35}$$

where ϕ_I is the MLS shape function, and $\mathbf{\Phi}$ is the matrix formed with MLS shape functions.

Substituting the foregoing expression for all the displacement components of **u** into the local weak-form Equation (5.34), and following the exact procedure detailed in Sub-section 5.2.1 yield the following nodal discretized system equations of MLPG for the *I*th field node.

$$\mathbf{K}_I\mathbf{u}=\mathbf{f}_I \tag{5.36}$$

where $\mathbf{K}_I$ is a matrix called the *nodal stiffness matrix* for the *I*th field node,

$$\begin{aligned}\mathbf{K}_I = &\int_{\Omega_q} \widehat{\mathbf{V}}_I^{\mathrm{T}}\mathbf{D}\mathbf{B}\mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{n}\mathbf{D}\mathbf{B}\mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{n}\mathbf{D}\mathbf{B}\mathrm{d}\Gamma \\ &+\boldsymbol{\alpha}\int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\Phi\mathrm{d}\Gamma\end{aligned} \tag{5.37}$$

and $\mathbf{f}_I$ is a *nodal force vector* with contributions from body forces applied in the problem domain, tractions applied on the natural boundary, as well as the penalty force term.

$$\mathbf{f}_I = \int_{\Gamma_{qt}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma + \int_{\Omega_q} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega + \boldsymbol{\alpha}\int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\overline{\mathbf{u}}\mathrm{d}\Gamma \tag{5.38}$$

Compared with Equations (5.21) and (5.22), the last terms in Equations (5.37) and (5.38) are new. They are required for imposing the essential boundary conditions. For a field node whose local quadrature domain lies entirely within the global domain, there is no intersection between Γ_q and the global boundary Γ, and the local weak-form is given in Equation (5.5). In this case, $\mathbf{K}_I$ and $\mathbf{f}_I$ have the same formulations as Equations (5.22) and (5.23).

We use Gauss quadrature to obtain the integrals in Equations (5.37) and (5.38); the algorithm is the same as that used in Equations (5.25) and (5.26) for LRPIM.

Equation (5.38) presents two linear equations for the *I*th field node. Using Equation (5.38) for all *N* field nodes in the entire problem domain and assembling all these 2*N* equations, we can obtain the final global system equations in the discretized linear algebraic form for MLPG, i.e.

$$\mathbf{K}_{2N\times 2N}\mathbf{U}_{2N\times 1} = \mathbf{F}_{2N\times 1} \tag{5.39}$$

Solving the above equation, we can obtain the nodal parameters of displacements and then compute the actual displacements at any point (including field nodes) in the problem domain using Equation (5.35). Finally the strains and stresses can be obtained using Equations (5.11) and (5.12).

5.3.2 Enforcement of essential boundary conditions

In Sub-section 5.3.1, the penalty method has been used to enforce essential boundary conditions in MLPG. In fact, other methods for enforcing essential boundary conditions in EFG, which have been discussed in Section 4.3, can also be used in MLPG.

Note that, in MLPG, the system equation is constructed node by node. There are only two rows in the global stiffness matrix and the global force vector that are related to each field node. With this structural feature of the system equation of MLPG, the following *direct interpolation method* can be used to enforce essential boundary conditions.

Assume the displacements at the *I*th field node on the essential boundary are prescribed as

$$\begin{cases} u_I^h = \overline{u}_I \\ v_I^h = \overline{v}_I \end{cases} \tag{5.40}$$

Using the MLS approximation, one has

$$\mathbf{u}_I^h = \begin{Bmatrix} u_I^h \\ v_I^h \end{Bmatrix} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} = \mathbf{\Phi u} = \begin{Bmatrix} \overline{u}_I \\ \overline{v}_I \end{Bmatrix} \tag{5.41}$$

Equation (5.41) produces two linear equations for the *I*th field node, and can be re-written explicitly as

$$\begin{cases} \phi_1 u_1 + \phi_2 u_2 + \cdots + \phi_n u_n = \overline{u}_I \\ \phi_1 v_1 + \phi_2 v_2 + \cdots + \phi_n v_n = \overline{v}_I \end{cases} \tag{5.42}$$

In Equation (5.40), both u and v of the *I*th node are prescribed. For some field nodes, it could be that only one of the two displacement components (u or v) is prescribed. Therefore, only one of the linear equations in Equation (5.42) can be obtained from the essential boundary condition for the prescribed displacement component at this field node. The other equation for the unprescribed displacement should still be obtained as in Equation (5.4).

Equation (5.42) is assembled (stacked) directly into the system equations for field nodes to obtain the modified global system equations of

$$\mathbf{K}_{2N\times 2N}\mathbf{U}_{2N\times 1} = \mathbf{F}_{2N\times 1} \tag{5.43}$$

where the modified global stiffness matrix $\mathbf{K}$ is

$$\mathbf{K}=\begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1(2N-1)} & K_{1(2N)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \phi_1 & 0 & \cdots & \phi_N & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_N \\ \vdots & & \ddots & \vdots & \vdots \\ K_{(2N)1} & K_{(2N)2} & \cdots & K_{(2N)(2N-1)} & K_{(2N)(2N)} \end{bmatrix} \begin{matrix} \\ \\ \leftarrow (2I\text{-}1)\text{th row} \\ \\ \leftarrow 2I\text{th row} \\ \\ \end{matrix} \tag{5.44}$$

The modified global force vector **F** is

$$\mathbf{F}=\begin{Bmatrix} f_{1x} \\ \vdots \\ \overline{u}_I \\ \overline{v}_I \\ \vdots \\ f_{Ny} \end{Bmatrix}_{(2N\times 1)} \begin{matrix} \\ \\ \leftarrow (2I\text{-}1)\text{th row} \\ \leftarrow 2I\text{th row} \\ \\ \\ \end{matrix} \tag{5.45}$$

For simplicity and without losing generality, $\phi_1 \sim \phi_N$ are used in Equation (5.44). Note that because the MLS shape functions are constructed in a compact support domain, the number of field nodes, n, selected in the support domain for the Ith node will usually be much smaller than the total number of field nodes, N (i.e., $n << N$). Therefore, many of $\phi_1 \sim \phi_N$ will be zero.

This direct interpolation method for the treatment of essential boundary condition is straightforward and very effective. It was used in the boundary node method (BNM) by Mukherjee and Mukherjee (1997), suggested for MLPG by Atluri et al. (1999b), and implemented in the MLPG for 2D solids by GR Liu and Yan (2000).

5.3.3 Commons on the efficiency of MLPG and LRPIM

There are many advantages in using MFree local weak-form methods, e.g. LRPIM and MLPG.

1) No global background cell is needed for the integrations.
2) The implementation procedures are as simple as numerical methods based on the strong-form formulation, such as the FDM.
3) No global compatibility of the shape functions is required, because no global energy principle is used in the formulation.

However, these advantages of MFree local weak-form methods do not come without some cost. The following study shows the fact that the MFree local weak-form method is generally less efficient than the MFree global weak-form method, and of course the FEM.

5.3.3.1 Comparison with FEM

Compared with FEM, the LRPIM and MLPG are computationally more expensive if the same field nodes are used. The additional computational computation cost mainly comes from: 1) the MFree interpolation, 2) the numerical integrations, and 3) solving the asymmetric stiffness matrix. A detailed study on the efficiency is conducted by comparing with FEM, and the results are presented using the numerical examples given in Sub-section 5.5.4.

5.3.3.2 Comparison with MFree global weak-form methods

Compared with MFree global weak-form methods, such as EFG and RPIM, discussed in Chapter 4, the major disadvantages of LRPIM and MLPG are the additional parameters introduced and the asymmetric system matrix. The additional parameters in LRPIM and MLPG include the sizes of local quadrature domains and the choice of the test function, etc. The asymmetric matrix will increase the computational cost in LRPIM and MLPG, as will be shown in the example problems given in Sub-section 5.5.4.

5.4 SOURCE CODE

In this section, a standard source code, MFree_Local.f90, of the MFree local weak-form method is provided and described in detail. This code is developed using FORTRAN 90. Combined with some of the subroutines given in Chapter 3 and Chapter 4, the code functions as either LRPIM or MLPG, respectively.

5.4.1 Implementation issues

1) Local quadrature domains

To perform the integrations for the local weak-form, local quadrature domains are needed. The local quadrature domain can be as simple as possible for the internal nodes. Rectangular domains are simple and easy to use, and they are used in this book.

For a rectangular quadrature domain, the dimension of the quadrature domain can be determined by r_{qx} and r_{qy} in x and y directions, respectively.

$$\begin{cases} r_{qx} = \alpha_{qx} d_{cx} \\ r_{qy} = \alpha_{qy} d_{cy} \end{cases} \tag{5.46}$$

where α_{qx} and α_{qy} are dimensionless sizes of the quadrature domain in x and y direction, respectively, and d_{cx} and d_{cy} are the local nodal spacings in x and y directions, which have been defined in Sub-section 3.1.3.

2) Method to enforce essential boundary conditions

The methods of enforcing essential boundary conditions in the LRPIM and MLPG have been discussed in Sections 5.2 and 5.3. The direct interpolation method is one of the most efficient methods for MLPG; it is used in this code.

3) Global error in energy norm

For the error analysis, the energy norm defined in Equation (4.78) is used as an error indicator, as the accuracy in strains or stresses is much more critical than the displacements. Note that the integration in Equation (4.78) is over the global domain. Hence, in order to get the global error in energy norm, global background cells, that can be the same as those used in the RPIM (or EFG), have to be used purely for the error assessment.

4) Flowchart of the subroutine

The flowchart of the computer code, MFree_Local.f90, is plotted in Figure 5.3. The procedure of LRPIM is very different from that of FEM and RPIM (EFG).

The major steps in a LRPIM analysis are as follows

- The geometry of the problem domain is modelled and a set of field nodes is generated to represent the problem domain;
- The influence domains are set for all field nodes;
- The system matrices are assembled through two loops;
    - The outer loop is for all the field nodes. At the beginning of this loop, a local quadrature domain is first constructed.
    - The inner loop is for all the Gauss quadrature points in the quadrature domain.
- The boundary conditions are enforced;
- The system equation is solved using the standard equation solver;
- The post-processing is performed to plot the final results including displacements, stresses, etc.

5.4.2 Program description and data structures

The main program of MFree_Local.f90 calls several subroutines. The macro chart for the program is given in Figure 5.4. The functions of these subroutines used in the main program are listed in Appendix 5.1. The main program is listed in Programs 5.1 and 5.2.

1) Programs for LRPIM and MLPG

The present program listed in the following Program calls the subroutine RPIM_ShapeFunc_2D that is for the construction of RPIM shape functions. Hence, the present program is for LRPIM. This program can be easily changed to the program for MLPG by replacing all the subroutine RPIM_ShapeFunc_2D with the subroutine MLS_ShapeFunc_2D. These two subroutines, RPIM_ShapeFunc_2D and MLS_ShapeFunc_2D, have been described in Chapter 3.

The source code of the main program is listed in Program 5.2.

2) Major variables

The major variables used in the program are listed in Appendix 5.2. The include file of variables, variableslocal.h, is listed in Program 5.1.

a. Most global variables are similar to the global variables that are presented in the program MFree_Global.f90 in Chapter 4.

Note that some of subroutines used in MFree_Local.f90 are the same as those used in the program MFree_Global.f90 (see Appendix 5.1). Therefore, the descriptions for these subroutines are not repeated here.

3) Subroutine *Input*

Source code location: Program 5.3.

Function: This subroutine performs simple operations of inputting data from a given external data file, and hence is self-explanatory, and easy to understand.

4) Subroutine *Qdomain*

Source code location: Program 5.4.

Dummy arguments: Appendix 5.3.

Function: This subroutine is to construct the local quadrature domain for a field node, and it is designed to construct a rectangular quadrature domain. Coordinates of four vertexes of the local quadrature domain are stored in the array *xc*. Readers can modify this subroutine slightly for creating other shapes of quadrature domains.

Note here that one of major challenges in MFree local weak-form methods is to develop an efficient algorithm for automatically forming the

local quadrature domains, especially for nodes on or near boundaries of a problem domain of a complex shape.

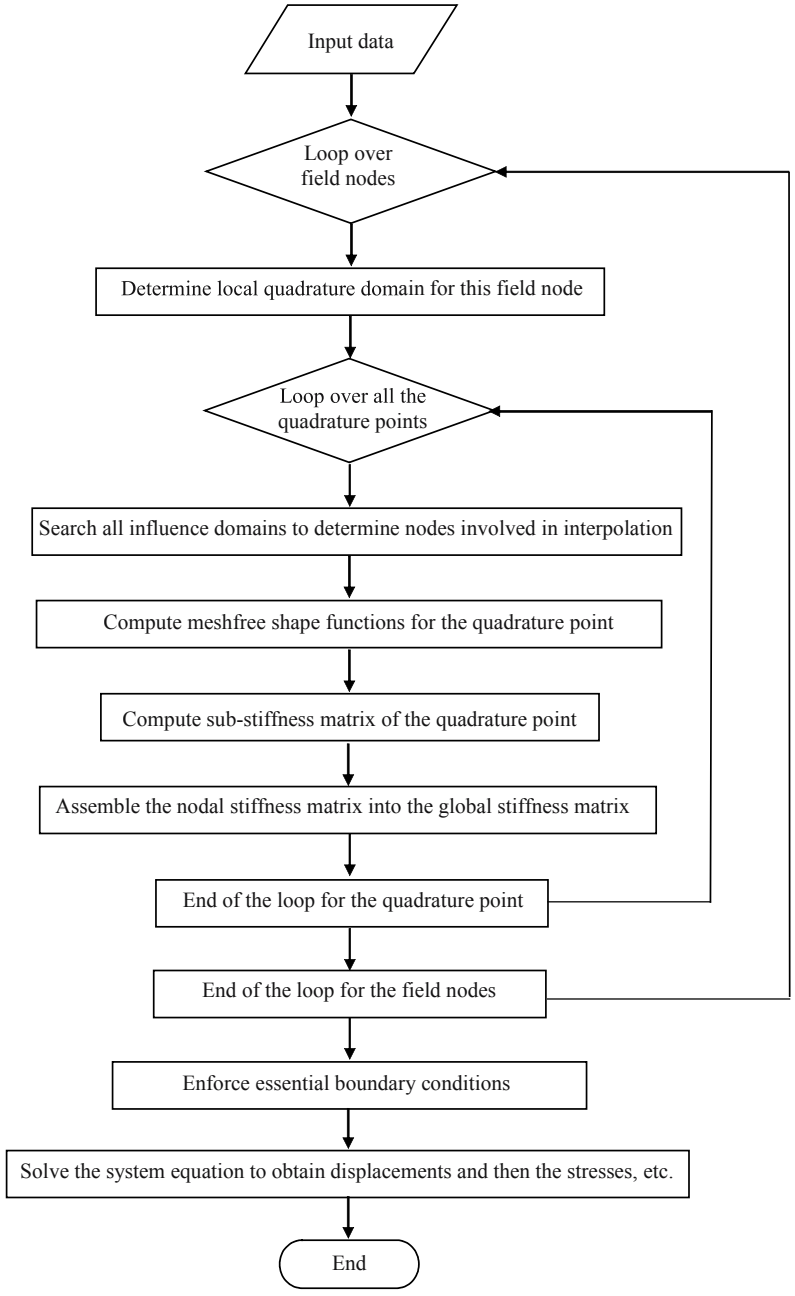


Figure 5.3. Flowchart of the program of MFree_Local.f90.

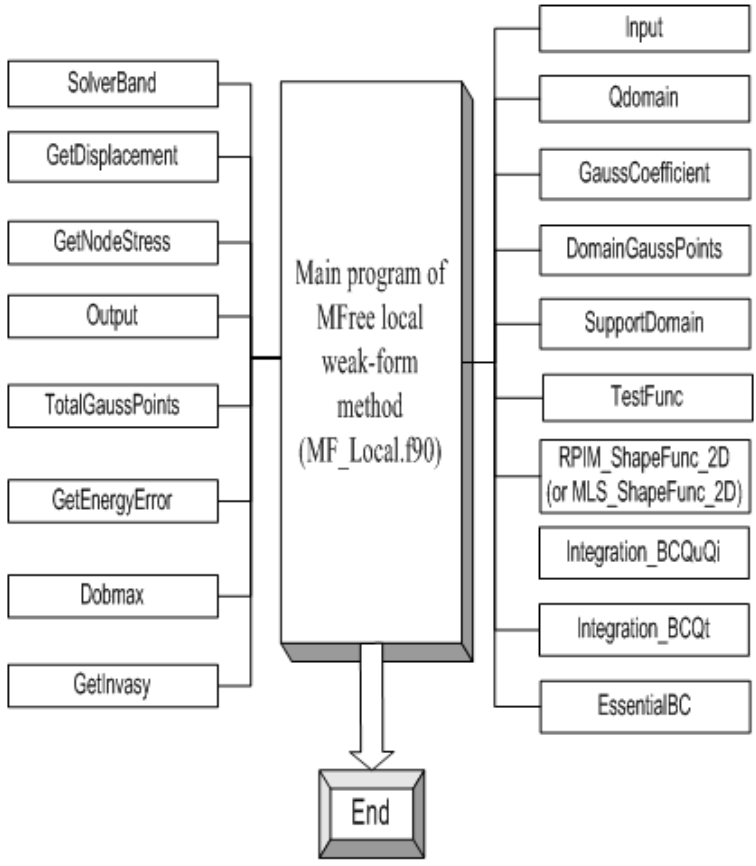


Figure 5.4. Macro flowchart of the program of MFree_Local.f90.

5) Subroutine *DomainGaussPoints*

Source code location: Program 5.5.
Dummy arguments: Appendix 5.4.
Function: This subroutine is to set the Gauss points and calculate the Jacobian for a local quadrature domain. In the present program, rectangular local quadrature domains are used. Hence, the subroutine is designed to set Gauss points for a quadrilateral quadrature domain.

6) Subroutine *TestFunc*

Source code location: Program 5.6

Function: This subroutine is to compute test or weight functions (the quartic spline function) and their derivatives. The field node is at the center of the weight functions. Note that the weight function domain is the same as the quadrature domain (r_q=r_w) defined in Equation (5.46).

7) Subroutine *Integration_BCQuQi*

Source code location: Program 5.7.
Dummy arguments: Appendix 5.5.
Function: This subroutine is to compute the integrations on the boundaries, Γ_{qu} and Γ_{qi}, of the quadrature domain that intersect with the global boundary. The integration is defined in the last two terms in Equation (5.21). Because the rectangular quadrature domains are used and the problem domain considered is also a rectangle, the integrations on these sub-boundaries are curve integrations along a line. These integrations can be obtained using the standard curve Gauss quadrature scheme. The main flowchart of this subroutine is shown in Figure 5.5.

8) Subroutine *Integration_BCQt*

Source code location: Program 5.8.
Dummy arguments: Appendix 5.6.
Function: This subroutine is to compute the integrations on the boundary, Γ_{qt}, of the quadrature domain that intersects with the global force boundary. The integration is defined in the first term in Equation (5.22). Because the rectangular quadrature domains are used and the problem domain considered is also rectangular, the integration on the sub-boundary Γ_{qt} is a curve integration scheme. The flowchart of this subroutine is shown in Figure 5.6.

Note here that subroutines of Integration_BCQu and Integration_BCQt are two important subroutines used in MFree local weak-form methods. How to efficiently achieve these boundary integrations is another major challenge in MFree local weak-form methods, especially for a problem domain with a complex geometry.

9) Subroutine *EssentialBC*

Source code location: Program 5.9.
Dummy arguments: Appendix 5.7.
Function: This subroutine is to implement the essential boundary conditions.

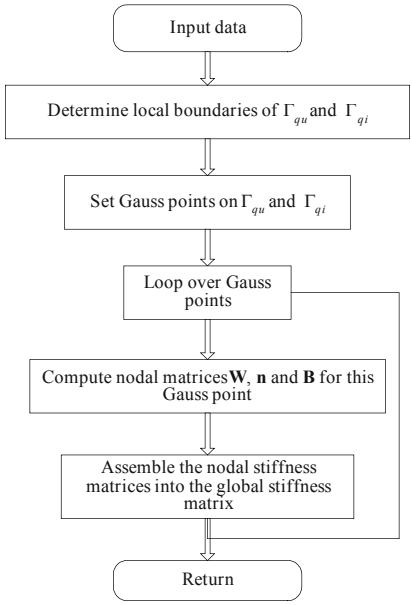


Figure 5.5. Flowchart of the subroutine Integration_BCQuQi.

10) Subroutine *GetDisplacement*

Source code location: Program 5.10.
Dummy arguments: Appendix 4.10.
Function: This subroutine is used only in MLPG to obtain the final displacements. This subroutine is unnecessary for LRPIM if only nodal displacements are interested, as the RPIM shape functions possess the delta function property (Sub-section 3.2.2).

11) Subroutine *GetNodeStress*

Source code location: Program 5.11.
Dummy arguments: Appendix 5.8.
Function: This subroutine is to compute stress components at all field nodes using Equations (4.10) and (5.12).

12) Subroutine *Output*

Source code location: Program 5.12.

Function: This subroutine performs the simple task of outputting the results. The source code of this subroutine is listed.

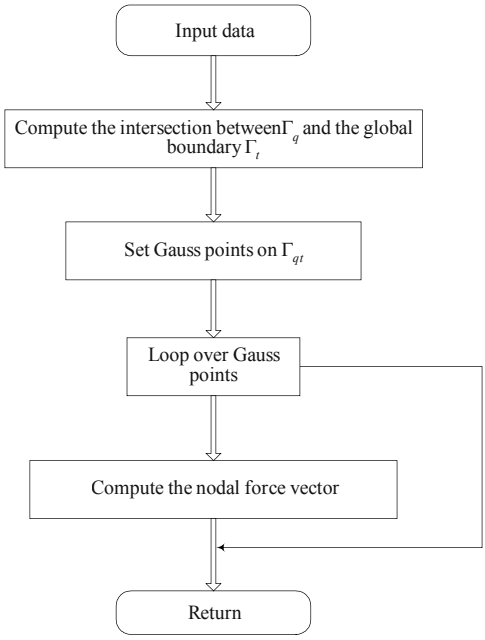


Figure 5.6. Flowchart of the subroutine Integration_BCQt.

13) Subroutine *TotalGaussPoints*

Source code location: Program 5.13.

Function: This subroutine is to obtain Gauss points for a global background mesh.

14) Subroutine *GetEnergyError*

Source code location: Program 5.14.

Function: This subroutine is to compute the global error in energy norm of the solution using Equation (4.78).

15) Subroutine *Dobmax* and *GetInvasy*

The source code of the subroutine Dobmax for computing multiplication of two matrices is listed in Program 5.15. The subroutine GetInvasy that is listed in Program 4.14 is used to compute the inversion of a matrix.

5.5 EXAMPLES FOR TWO DIMENSIONAL SOLIDS – A CANTILEVER BEAM

To provide a quantitative analysis, a cantilever beam subjected to a parabolic traction at the free end as shown in Figure 4.5 is considered. The beam has a unit thickness and is in a plane stress. The exact solution of this problem is listed in Equations (4.79)~(4.84). The study of this simple example has the following purposes.

a) To demonstrate the standard procedure of an MFree local weak-form method;
b) To show the usage of the present programs, MFree_Local.f90, of LRPIM and MLPG;
c) To investigate the effects of the shape parameters of MQ-RBF in LRPIM;
d) To investigate the effects of the size of local domains;
e) To study numerically the convergence of LRPIM and MLPG;
f) To study the efficiency of LRPIM and MLPG.

5.5.1 The use of the MFree_local.f90

To use this program of MFree_local.f90, three steps, which are similar to that discussed in Chapter 4, may be followed:

Step 1: Preparation of input file

The problem should be modeled in this step. The aim of this step is to prepare the input data file for the program.

An example of input data file for the beam problem is listed in Appendix 5.9. The field nodes used in this file is plotted in Figure 5.7. The domain of the beam is represented by regularly distributed 55 (11×5) field nodes. This data file can be largely divided into five parts.

Part 1: this part includes the parameters of description of the problem including: Length and Width of the problem domain; Young's modules; Poisson's ratio; The distributed traction; Total number of field nodes; Global boundary information ($x_{\max}$, $x_{\min}$, $y_{\min}$, $y_{\max}$).

Part 2: this part provides the parameters for determination of sizes of the local domains, including: Sizes of the local quadrature domain (in x, y directions); Number of sub-partitions used to divide the quadrature domain (in x, y directions); Number of Gauss points used in each partition; Size of the local influence domain.

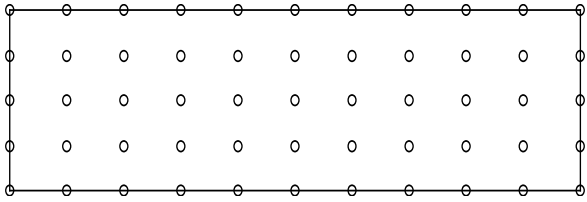


Figure 5.7. An MFree model with 55 regular field nodes used to represent the problem domain and boundaries.

Part 3: this part contains the detailed coordinates of field nodes: Number of node, x_i and y_i.

Part 4: this part defines the essential boundary conditions and the natural boundary conditions. The exact essential boundary conditions (see Equations (4.79) and (4.81)) and natural boundary conditions (see Equation (4.84)) are used to compute these values.

Part 5: this part includes the global background cells and the coordinates of the vertexes of the background cells that are used only to compute the global error in energy norm in the solution.

Step 2: Execution of the program

After the preparation of the input data file, the program can be executed to obtain the results. LRPIM is first used, and results are listed in Appendix 5.10 and Appendix 5.11. In Appendix 5.10, the displacements at field nodes are listed, and the stresses at the field nodes are listed in Appendix 5.11. In the output, the error in energy norm is also presented in Appendix 5.11.

The MLPG method is also used, and results are listed in Appendix 5.12 and Appendix 5.13. The displacements at field nodes are listed in Appendix 5.12, and the stresses at field nodes are listed in Appendix 5.13 together with the error in energy norm.

Step 3: Analysis of the output data

The task of this step can be performed using any post-processor like MFree Post in the sofyware of MFree 2D© (GR Liu, 2002).

Results of LRPIM are plotted in Figure 5.9~Figure 5.10. The MQ-RBF with linear polynomial terms is used in LRPIM, and the parameters used are $\alpha_c = 1.0$, $q = 1.03$, and $d_c = 3.0$. For the local influence domains, $d_{cx} = 4.8$, $d_{cy} = 3.0$, and $\alpha_i = 3.0$ are used. For local quadrature domains, $\alpha_q = 2.0$ is used. To ensure the accuracy of numerical integration, the local quadrature domain is further divided into $n_{dx} \times n_{dy}$ small sub-partitions, as shown in Figure 5.8. In this study, we let $n_{dx} = n_{dy} = n_d$ and $n_d = 2$. In each sub-

partition, a total of 16 (4×4) Gauss points are used. The cubic spline function is used as the test function for the local Petrov-Galerkin weak-form.

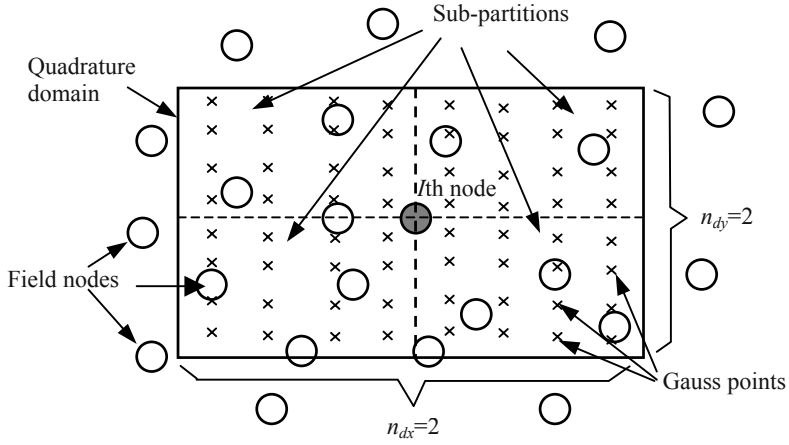


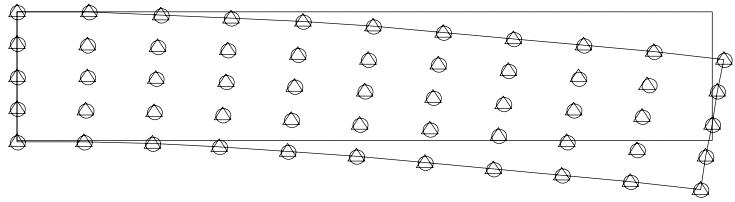
Figure 5.8. A local quadrature domain is divided to $n_{dx} \times n_{dy}$ sub-partitions. A total 4×4 Gauss points are used in each partition.

The deflection results are plotted in Figure 5.9. For comparison, the analytical results of displacements are also plotted in the same figure. A very good agreement can be found between LRPIM results and the analytical results. The results of shear stress, τ_{xy}, are plotted in Figure 5.10. Compared with the analytical results, LRPIM gives a reasonably good result even for stresses.

Results of MLPG are analyzed in Figure 5.11~Figure 5.12. In computing the results shown in these figures, the linear polynomial basis and the cubic spline weight function are used in the MLS approximation. For the local influence domains, $d_{cx} = 4.8$, $d_{cy} = 3.0$, and $\alpha_i = 3.0$ are used. For local quadrature domains, $\alpha_q = 1.5$, $4(2 \times 2)$ sub-partitions, and 16 (4×4) Gauss points in each partition are used. The cubic spline function is used as the test function for the local weak-form. The deflection results are plotted in Figure 5.11. For comparison, the analytical results of displacements are also plotted in the same figure. A very good agreement between MLPG result and the analytical result is found. The results of the shear stress, τ_{xy}, are plotted in Figure 5.12. Compared with the analytical results, the results given by MLPG are very good.

Two nodal distributions of 189 regular nodes and 189 irregular nodes shown in Figure 5.13 are used to test the present code further. Shear stresses τ_{xy} are computed using LRPIM and plotted in Figure 5.14. The same stresses

τ_{xy} are also obtained using MLPG and plotted in Figure 5.15. Compared with analytical results, results of both LRPIM and MLPG are very good. It is seen that the nodal irregularity has little effects on the results, and this is true for both LRPIM and MLPG.



○: Analytical solution; Δ: Numerical solution.

Figure 5.9. Deflections of the beam obtained using LRPIM and 55 regularly distributed field nodes. Note that the displacements plotted are magnified by 500 times.

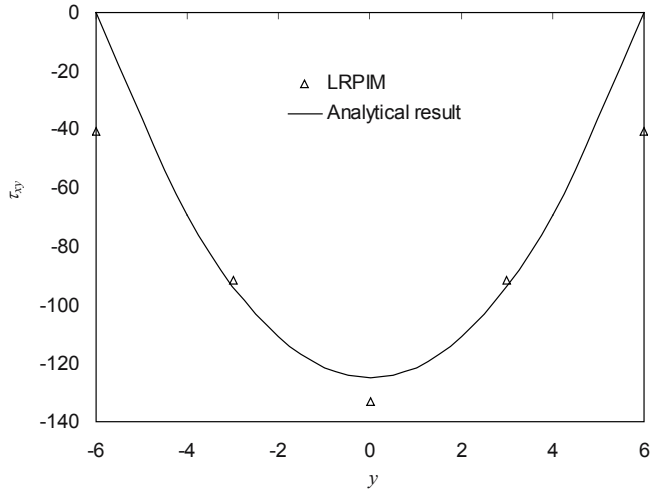


Figure 5.10. Shear stress τ_{xy} distribution on the cross-section at $x=L/2$ of the beam obtained using the LRPIM and 55 regular field nodes.

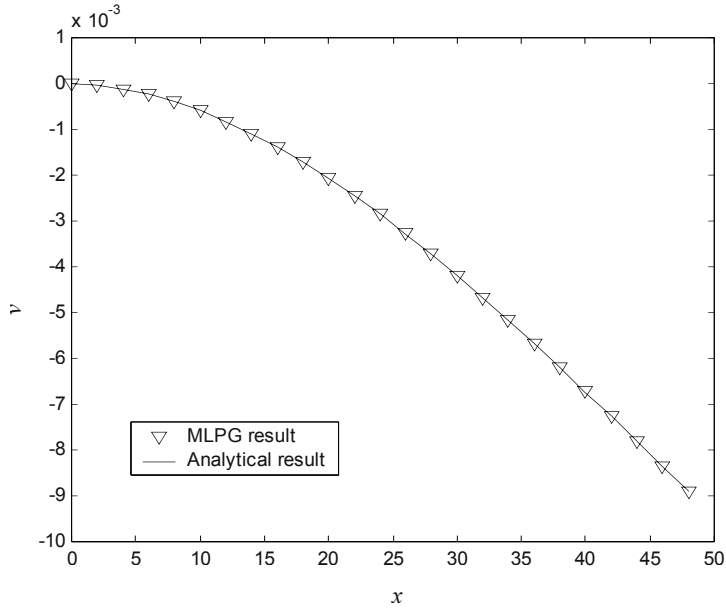


Figure 5.11. Deflections at the central axis at $y = 0$ of the beam obtained using the MLPG and 55 regular field nodes.

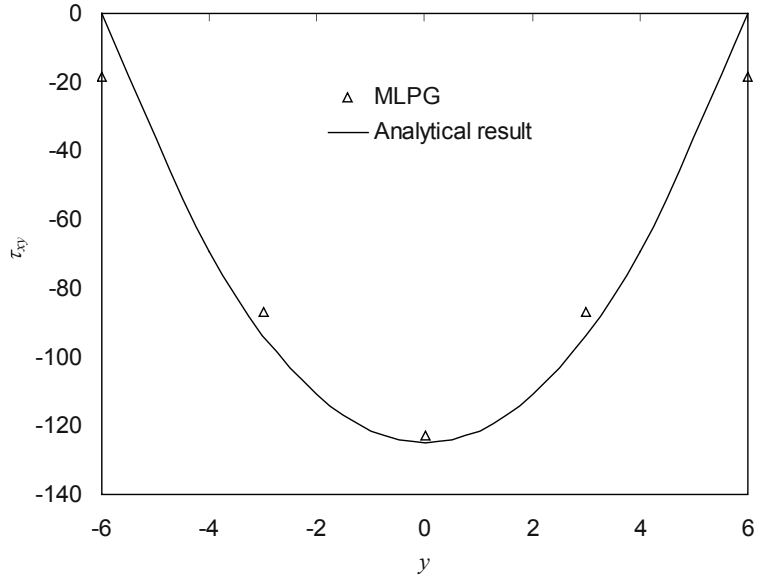


Figure 5.12. Shear stress distribution on the cross-section at $x=L/2$ of the beam obtained using the MLPG and 55 regular field nodes.

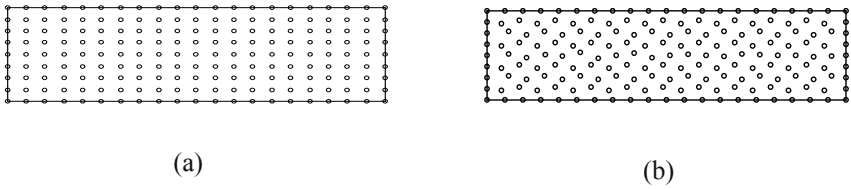


Figure 5.13. Nodal arrangements for the cantilever beam. (a) 189 regular nodes; (b) 189 irregular nodes.

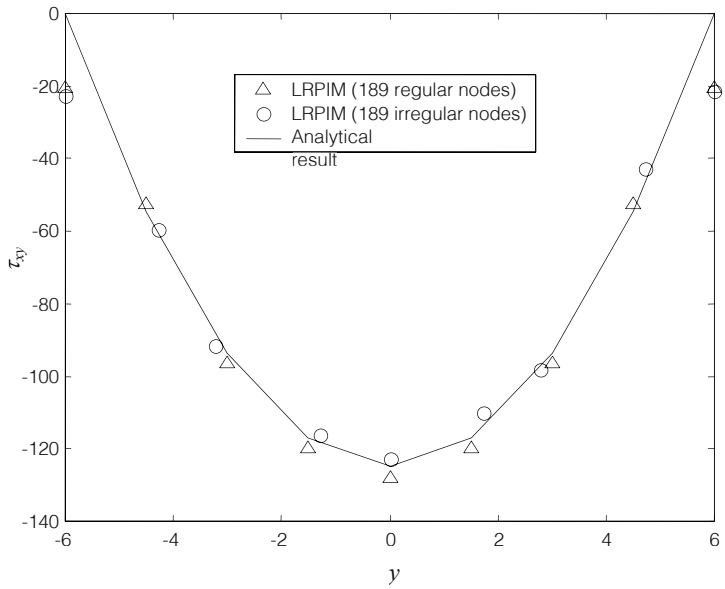


Figure 5.14. Shear stress distributions on the cross-section at $x = L/2$ of the beam obtained using the LRPIM and 189 field nodes.

5.5.2 Studies on the effects of parameters

In the following studies, the problem domain is represented using 189 (21×9) regular nodes. For quantitative and accurate analyses, the exact essential boundary conditions and natural boundary conditions are used. The error in energy norm, Equation (4.78), is used as an accuracy indicator. In

LRPIM, the linear polynomial terms are added in the RPIM-MQ. In MLPG, the linear basis is used in the MLS approximation.

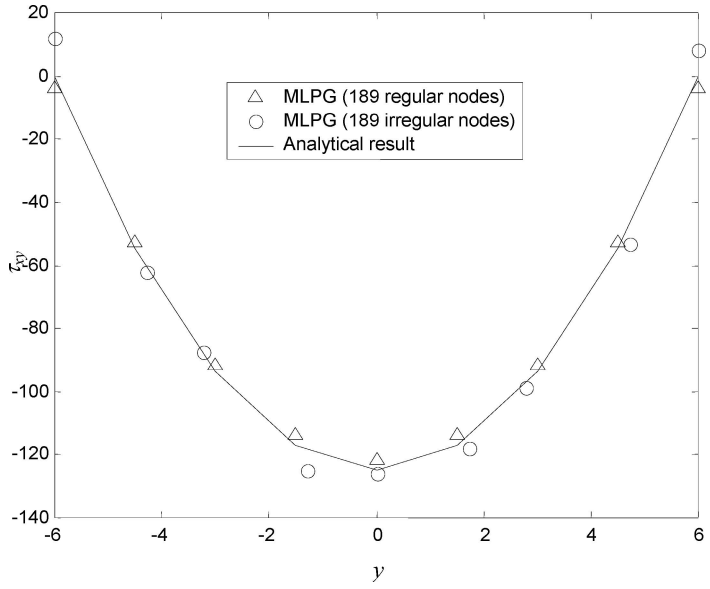


Figure 5.15. Shear stress distributions on the cross-section at $x = L/2$ of the beam obtained using the MLPG and 189 field nodes.

5.5.2.1 Parameters effects on LRPIM

a) Shape parameters of RBF

The shape parameters of the MQ-RBF are studied. More detailed discussion on the effects of RBF parameters for other RBFs are presented in the paper by Wang and GR Liu et al. (2002c) and a book by GR Liu (2002). Readers can also slightly modify the present codes and input data file to conduct their own study on other RBFs.

In MQ-RBF, there are two shape parameters, α_c and q, that have been discussed in Section 3.2. Because the regular nodes are used, d_c that is a parameter of the nodal spacing is a constant of $d_c = L/20 = 2.4$. In this study, $\alpha_i = 3.0$ is used for the construction of support (influence) domains.

First, q is investigated, while α_c is fixed at 1.0, 2.0 and 4.0. Errors in energy norm for five different values of q ($q = -0.5$, 0.5, 0.98, 1.03 and 1.2) are plotted in Figure 5.16. From Figure 5.16, it can be confirmed that q=0.98 and q=1.03 with α_c=4.0 give better results for this problem. According to the conclusions of the study by GR Liu (2002), q=1.03 is generally stable and accurate for many problems. Hence, q=1.03 is used in the following studies.

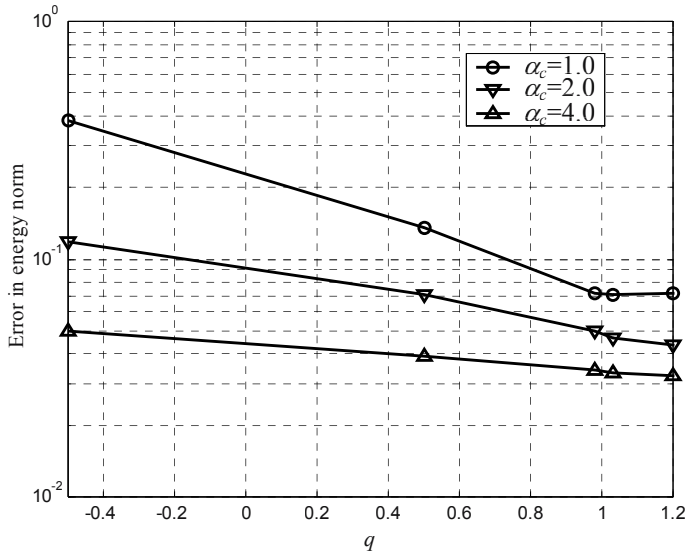


Figure 5.16. Influence of q on the accuracy of the results obtained using the LRPIM-MQ. It can be found that $\alpha_c = 4.0$, q=0.98 and 1.03 give accurate results.

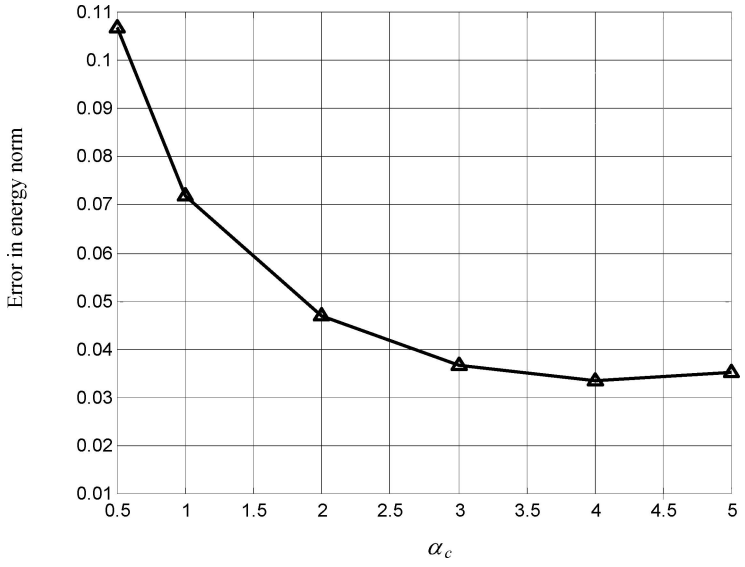


Figure 5.17. Influence of α_c on the accuracy of the results obtained using LRPIM-MQ (q=1.03). It can be found that the results of $\alpha_c = 3.0 \sim 5.0$ are more accurate.

In the study on α_c, the range of 0.5~7.0 with q fixed at 1.03 is now considered. Errors in energy norm for different values of α_c are plotted in Figure 5.17. From this figure, we can find that all α_c in the considered range can lead to satisfactory results. The results of $\alpha_c = 3.0 \sim 7.0$ are slightly better. For convenience, $\alpha_c = 4.0$ will be used in the following studies.

Comparing with those in Section 4.5, the findings from this study are very much the same, and hence the same shape parameters are used for both RPIM and LRPIM.

b) Effects of the size of local quadrature domain

The size of the local quadrature domain affects the accuracy of the LRPIM solutions. The sizes of quadrature domains are defined in Equation (5.29), in which $d_{cx} = L/20 = 2.4$ and $d_{cy} = D/8 = 1.5$ are used in this study. The sizes of quadrature domains will be, therefore, determined by α_{qx} and α_{qy}, which are dimensionless coefficients in x and y directions, respectively. For simplicity, $\alpha_{qx}=\alpha_{qy}=\alpha_q$ is used. The errors in energy norms for different α_q are obtained and plotted in Figure 5.18; the accuracy for solutions generally is improved by increasing the size of the quadrature domain.

When the quadrature domain is too small ($\alpha_q \leq 1.0$), the error in results will become unacceptably large. This is because a local residual formulation with a very small quadrature domain for the weight function behaves more like a purely strong-form formulation (a collocation method). Strong-form formulation is usually less accurate than a weak integral form formulation, in which the integration smears the error over the integral domain (Liu and Han, 2003). More detail on this topic will be given in Chapter 6.

When the quadrature domain is large enough ($\alpha_q \geq 1.5$), results obtained are very good. However, it is difficult to obtain accurate numerical integrations for a large local quadrature domain (see Sub-section 5.2.2.4). Because more regular small partitions and Gauss quadrature points are needed, the numerical integration in a large quadrature domain becomes computationally expensive and is not really necessary. Figure 5.18 shows that a too large local quadrature domain is not necessary to give a significant improvement in the accuracy. Hence, α_q=1.5-2.5 is an economical choice that gives good results. In the following studies of LRPIM, α_q=2.0 is used.

c) Effects of numerical integration

As discussed above, there are difficulties in obtaining accurate numerical integration because of the complexities of integrands (see Sub-section 5.2.2.4). To study effects of numerical integrations in more detail, a local quadrature domain Ω_q with α_q=2.0 is used. The local quadrature domain is further divided into $n_{dx} \times n_{dy}$ small partitions, as shown in Figure 5.8. In this study, we let $n_{dx} = n_{dy} = n_d$. In each partition, 4×4 Gauss points are used.

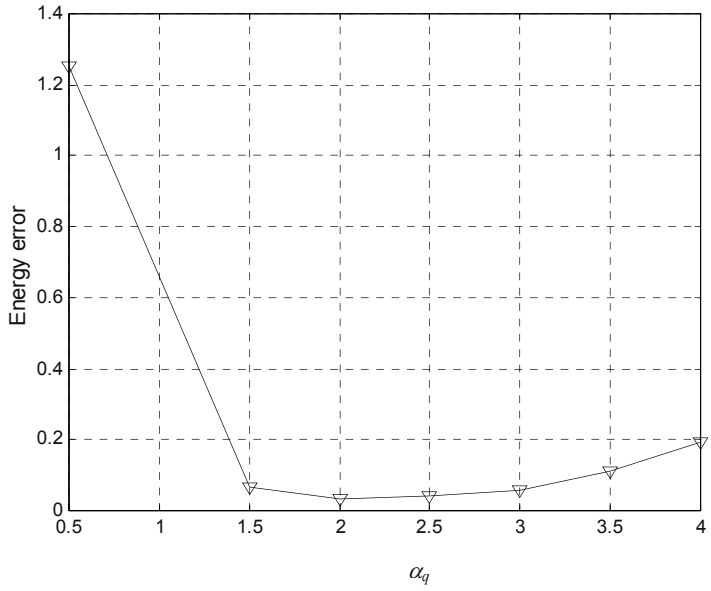


Figure 5.18. Influence of the sizes of local quadrature domain on the accuracy of the results obtained using LRPIM.

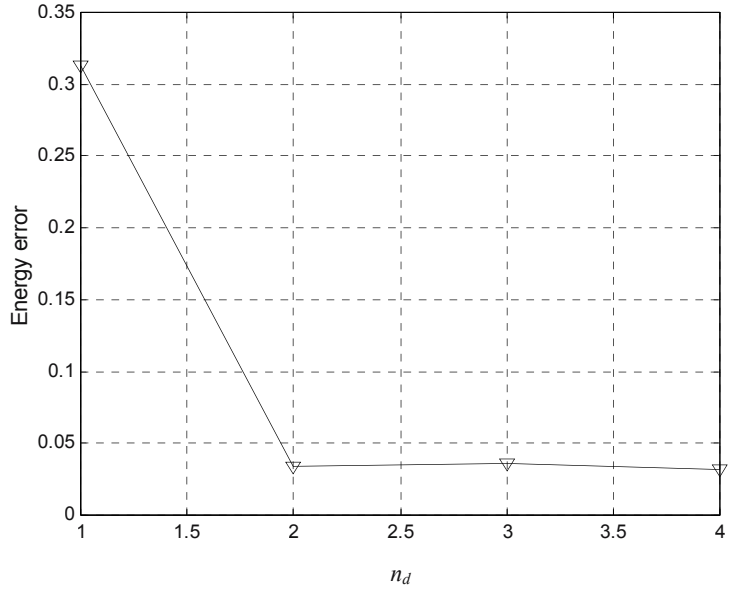


Figure 5.19. Influence of the number of sub-partitions on the accuracy of the results obtained using LRPIM-MQ (q=1.03, αc=4.0).

Results of errors in energy norms for different n_d are obtained and plotted in Figure 5.19. It can be observed that the accuracy of solutions improves with the increase of n_d due to the improvement of the accuracy of numerical integrations. Hence, in order to ensure an accurate numerical integration, Ω_q should be divided into some regular sub-partitions. In each sub-partition, sufficient Gauss quadrature points should be used.

However, the increase of the number of sub-partitions and Gauss points will increase the computational cost. A good and economical choice is n_d=2.

d) Effects of the size of the influence domain

The size of influence domains is defined in Equation (4.75), d_{cx} and d_{cy} are the nodal spacings in x and y directions near the field node I. In this study, $d_{cx} = L/20 = 2.4$ and $d_{cy} = D/8 = 1.5$ are used. The size of influence domains is determined by α_{ix} and α_{iy}, which are dimensionless coefficients in x and y directions. For simplicity, we use $\alpha_{ix}=\alpha_{iy}=\alpha_i$.

Errors in energy norms for different α_i are plotted in Figure 5.20 for two cases. The shape parameters of MQ-RBF are $q = 1.03$ and $\alpha_c = 4.0$ for case 1; $q = 1.03$ and $\alpha_c = 1.0$ for case 2. It can be found that the accuracy changes with α_i, and the results of $\alpha_i \geq 2.0$ are very good. The reason of the bad results obtained using $\alpha_i \leq 1.5$ is that the influence domain is too small. There are not enough field nodes included for interpolation. For a too large influence domain, e.g. $\alpha_i \geq 4.0$, the accuracy is good, but the computational cost will also increase accordingly for the inclusion of large number of nodes in the interpolation. An economical choice is α_s=2.0~3.0 for reasonably good results. In the following studies on LRPIM, $\alpha_i = 2.5$ will be used.

e) Convergence

In the numerical convergence study, regularly and evenly distributed 18 (3×6), 28 (4×7), 55(5×11), 112(7×16), 189(9×21), and 403(13×31) field nodes are used. The convergence curves obtained numerically are shown in Figure 5.21, where h is equivalent to the element size (in x direction) in the FEM analysis in this case. The convergence rate, R, that is computed via linear regression is also given in Figure 5.21. From Figure 5.21, it is observed that convergence rate of LRPIM is about 1.5. However, the convergence is not a straight line.

It should be mentioned again that the shape parameters chosen in the MQ-RBF will affect the convergence rate and the accuracy of the LRPIM.

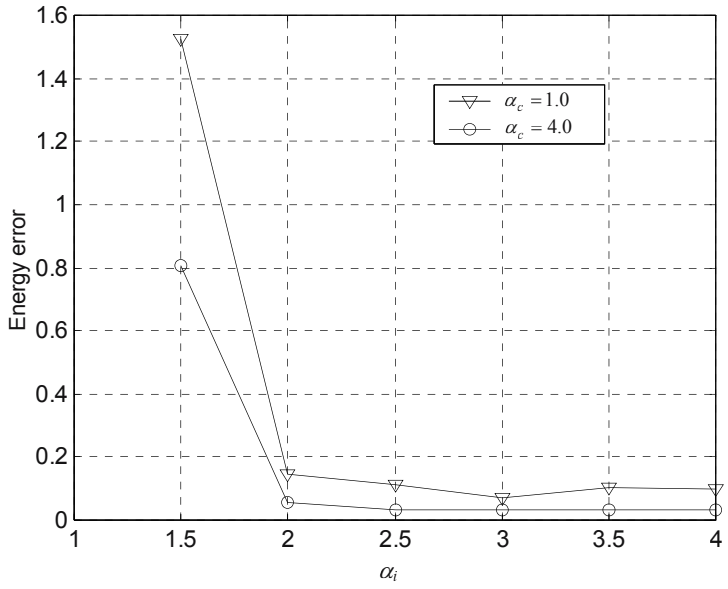


Figure 5.20. Influence of the sizes of local influence domain on the accuracy of the results obtained using the LRPIM (q=1.03). The size of local influence domain is defined as: $r_{ix} = \alpha_i d_{cx}$ and $r_{iy} = \alpha_i d_{cy}$.

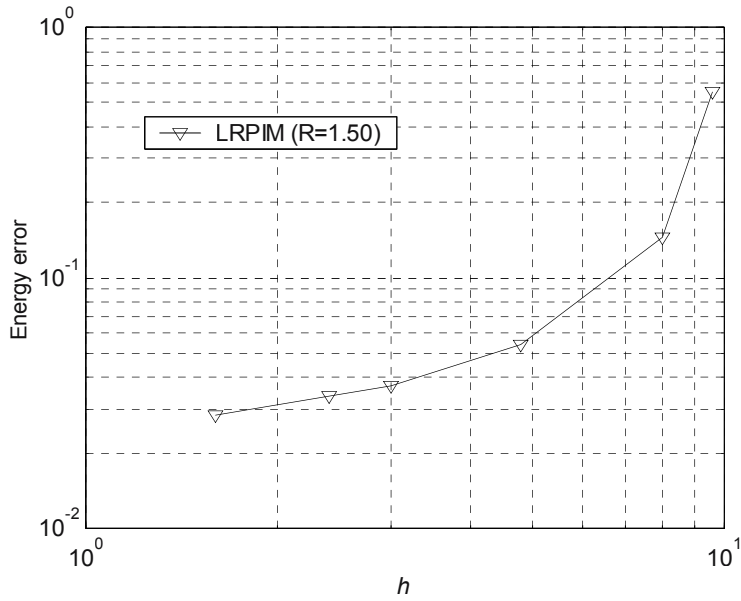


Figure 5.21. Numerical convergence of LRPIM-MQ in error e_e of energy norm. R is the convergence rate computed by linear regression.

5.5.2.2 Parameter effects on MLPG

1) Effects of the size of local quadrature domain

In this study, $d_{cx} = L/20 = 2.4$, $d_{cy} = D/8 = 1.5$, and $\alpha_{qx}=\alpha_{qy}=\alpha_q$ are used. Several quadrature domains with different α_q are investigated, and the errors in the energy norm in the solution of the cantilever beam problem have been plotted in Figure 5.22. From this figure, it can be found that the accuracy for solutions generally improves with the increase of the size of the quadrature domain. When the quadrature domain is too small (α_q<1.0), the error of the results will become unacceptably large. When the quadrature domain is large enough (α_q≥1.5), results obtained are very good. The reasons are similar to the discussions in LRPIM. However, a too large local quadrature domain (α_q≥3.0) does not necessarily lead to a significant improvement in the accuracy. Hence, α_q=1.5-2.5 is an economical choice in MLPG for a reasonably accurate solution. In the following studies on MLPG, α_q=1.5 is used.

2) Effects of numerical integration

As discussed above in LRPIM, to obtain accurate numerical integrations, the local quadrature domain is divided into $n_{dx} \times n_{dy}$ small sub-partitions, as shown in Figure 5.8. In this case, $n_{dx} = n_{dy} = n_d$ is used, and there are 4×4 Gauss points in each partition. Results of errors in energy norms for different n_d are obtained and plotted in Figure 5.23. This figure shows that the accuracy of solutions improves with the increase of n_d due to the improvement of the accuracy in the numerical integrations. However, the increase of the number of sub-partitions and Gauss points will increase the computational cost. In the following studies on MLPG, n_d=2 is used.

3) Size of the influence domain

In the study of the effects of the influence domains, $d_{cx} = L/20 = 2.4$, $d_{cy} = D/8 = 1.5$, and $\alpha_{ix} = \alpha_{iy} = \alpha_i$ are used. Errors in energy norm for different α_i are plotted in Figure 5.24. It can be found that the accuracy changes with α_i and the results for $2.0 \le \alpha_i \le 4.0$ are very good.

It is found that a too small influence domain ($\alpha_i < 2.0$) leads to large errors. The inaccuracy of a too small influence domain is caused by the fact that there are not enough nodes to perform accurate approximation for the field variables.

A too large influence domain ($\alpha_i > 4.0$) will considerably increase the computational cost. Hence, a proper influence domain should be used in

MLPG. Our studies have found that $\alpha_i = 2.5$ is a good choice and will be used in the following studies on MLPG.

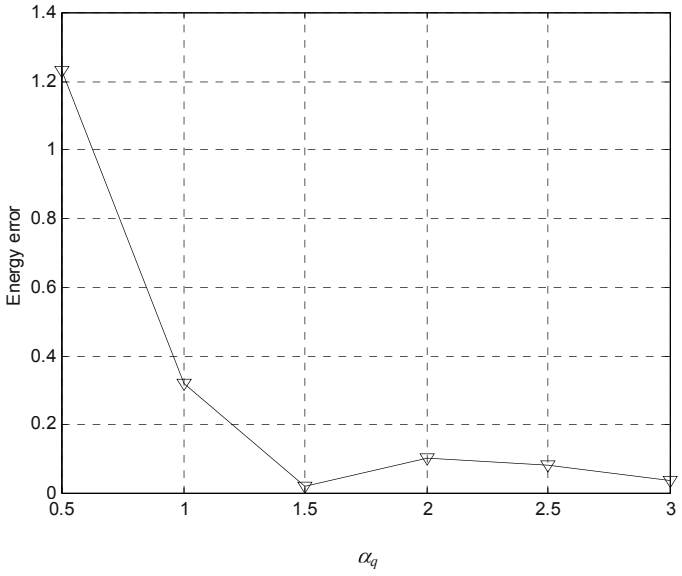


Figure 5.22. Influence of the sizes of local quadrature domain on the accuracy of the results obtained using MLPG.

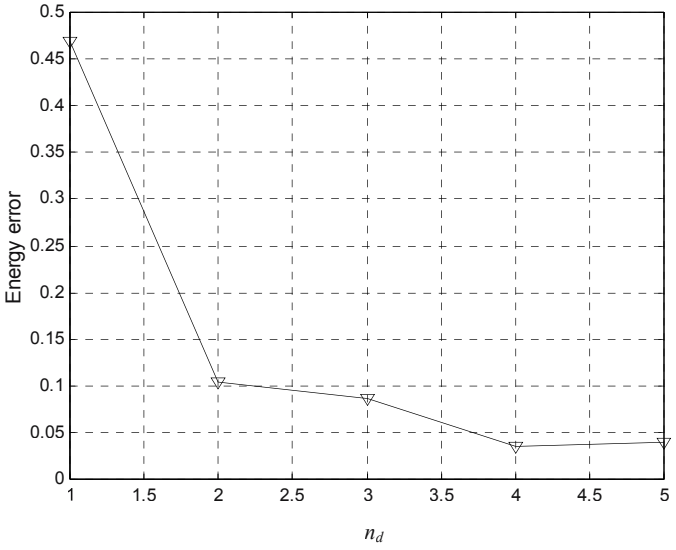


Figure 5.23. Influence of the number of sub-partitions for numerical integrations on the accuracy of the results obtained using the MLPG.

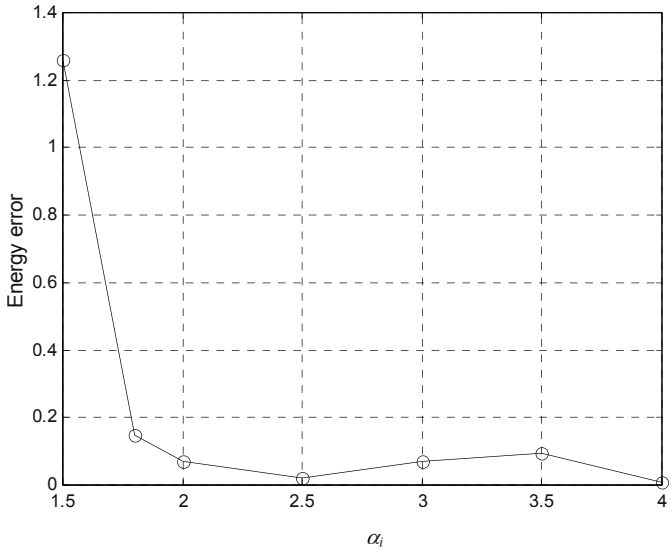


Figure 5.24. Influence of the sizes of local influence domain on the accuracy of the results obtained using the MLPG.

4) Convergence

The convergence of MLPG is studied numerically using regularly and evenly distributed 18 (3×6), 28 (4×7), 55(5×11), 112(7×16), 189(9×21), and 403(13×31) field nodes. The convergence curve of MLPG results obtained numerically is shown in Figure 5.25. The convergence rates, R, computed via linear regression are also given in Figure 5.25. It is observed that the convergence rate of MLPG is about 1.67. Note that only the linear basis is used in the MLS approximation to obtain the MLPG results shown in Figure 5.25.

5.5.3 Comparison of convergence

For comparison between methods, an intensive numerical study has been carried. The convergence curves of LRPIM, MLPG, RPIM, EFG and FEM computed for the same cantilever beam under exactly the same conditions, and are plotted together in Figure 5.26. The same results for RPIM, EFG and FEM have already been presented in Sub-section 4.5.3. From Figure 5.26, the following remarks can be made:

a) Both the convergence rates and the accuracies of LRPIM and MLPG are much better than those of FEM using bi-linear elements.

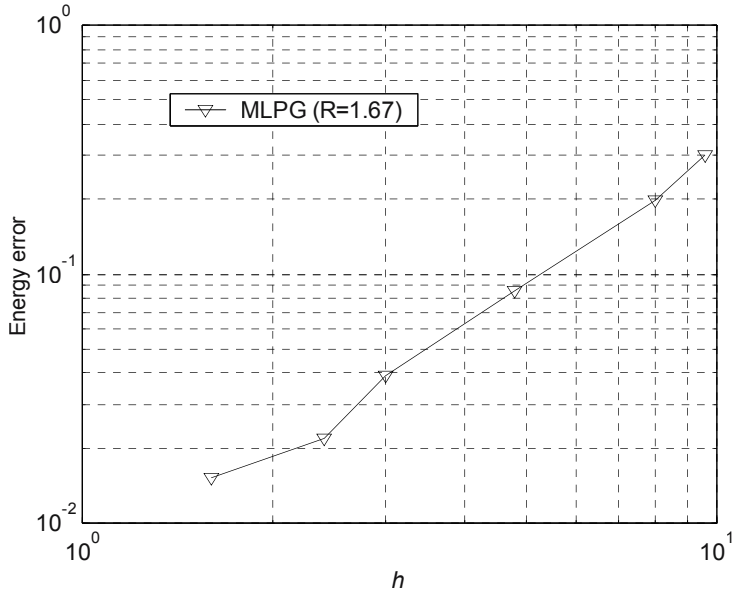


Figure 5.25. Numerical convergence of MLPG in error e_e of energy norm. R is the convergence rate computed by linear regression.

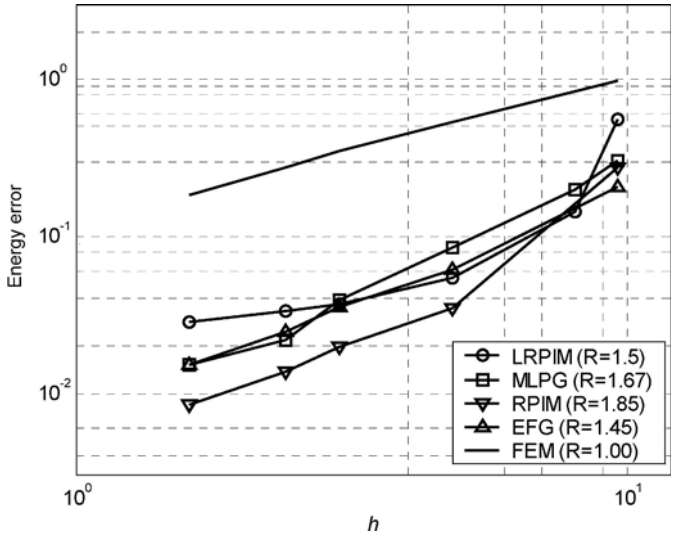


Figure 5.26. Comparison of convergence curves of LRPIM, MLPG, RPIM, EFG and bi-linear FEM in error e_e of energy norm. R is the convergence rate.

b) The convergence rate of MLPG is slightly better than that of LRPIM. In addition, the convergence rate of MLPG is better than that of EFG and their accuracies are very close.
c) Both accuracy and convergence rate of LRPIM are slightly worse than those of RPIM. In addition, although the convergence rate and the accuracy of LRPIM are very good, the convergence process of the LRPIM slows down at finer nodal distributions.

5.5.4 Comparison of efficiency

A successful numerical method should obtain high accuracy at a lower computational cost. For a fair comparison, both the accuracy in results and the cost to get the results are investigated. Regularly distributed 18, 55, 189 and 403 nodes are used to calculate the error against the computation time curves for LRPIM, MLPG, RPIM, EFG and bi-linear FEM. These curves are plotted in Figure 5.27 for easy comparison. In this efficiency study, $\alpha_i = 2.5$ is used in LRPIM, RPIM, EFG, and MLPG.

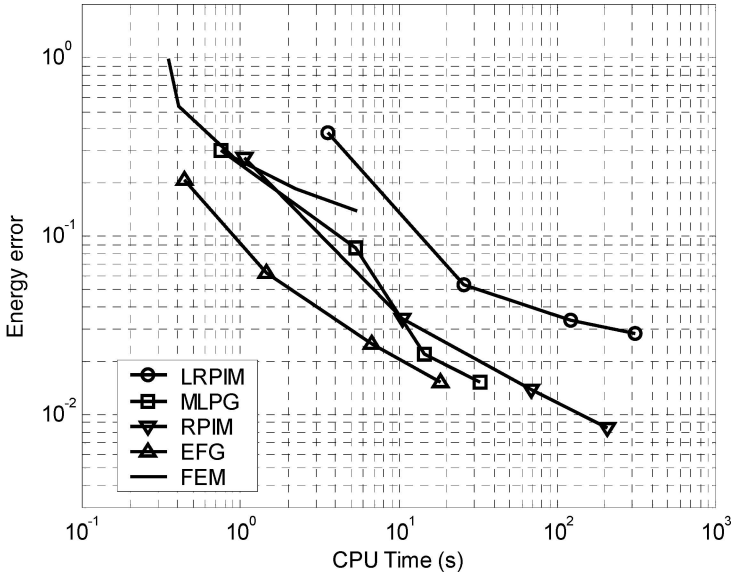


Figure 5.27. Comparison of the computational efficiencies of LRPIM, MLPG, RPIM, EFG and bi-linear FEM in error e_e of energy norm.

It can be found form Figure 5.27 that

1) The efficiencies of MFree methods are better than that of FEM.
2) The EFG method shows the best performance.

3) LRPIM needs more computational time than MLPG. In other words, the efficiency of MLPG is better than that of LRPIM. This is because of their difference in the interpolations. RPIM shape functions need more computation than the MLS shape functions.
4) The efficiencies of the MFree local Petrov-Galerkin weak-form methods (LRPIM and MLPG) are lower than the corresponding counter-part of the MFree global Galerkin weak-from methods (RPIM and EFG). It is because the system matrices in the LRPIM and MLPG are asymmetric. There seems to be a trade off between the efficiency and the use of background mesh.

Note that when the Lagrange multiplier method is used in EFG or MLPG, their efficiency will drop, as discussed in Section 4.5 and shown in Figure 4.24.

5.6 REMARKS

MFree local weak-form methods, LRPIM and MLPG, are presented in this chapter. The numerical implementations of both LRPIM and MLPG discussed. A computer code is provided. The present code is examined using numerical examples. LRPIM and MLPG are studied to reveal the effects of different parameters, convergence, performances, etc. From these studies in this chapter, we can make the following important remarks:

a) The compatibility of the trial (shape) functions in the whole domain is not required in MFree local weak-form methods.
b) For local weak-forms, the global background cells are successfully avoided. The integration in the MFree local weak-form methods is performed in a local quadrature domain with simple shapes for internal nodes.
c) In LRPIM, the shape parameters of MQ-RBF are recommended with the shape parameters fixed at q=1.03 and α_c=4.0.
d) When the local quadrature domains used in LRPIM and MLPG are large enough ($\alpha_q \geq 1.5$), results obtained are very good, and α_q=2.0 is recommended. In order to ensure accurate numerical integration, Ω_q should be divided into some regular sub-partitions, and 2×2 is recommended. In each sub-partition, sufficient Gauss quadrature points should be used, and 16 (4×4) Gauss points are recommended.

e) The accuracy of solutions change with the sizes of the influence domains α_i and the results obtained using $2.0 \le \alpha_i \le 4.0$ are very good. We recommend α_i=2.5.

f) The convergence rates of both the LRPIM and MLPG are very good. They are all about 1.5. The convergence rate and the efficiency of MLPG are slightly better than these of LRPIM.

Note that these remarks are based on the simple cantilever beam problem, whose solution is of simple polynomial forms.

The present MFree local Petrov-Galerkin weak-form methods (e.g. LRPIM and MLPG) possess the following advantages over their counterpart of the MFree global Galerkin weak-form methods (e.g. RPIM and EFG).

1) No global background integration cells is needed, which is one step closer to truly meshfree.

2) The implementation procedure is node based. It is similar to the methods based on strong-forms, yet possesses high accuracy as long as the local quadrature domains are sufficiently large.

However, MFree local weak-form methods possess some disadvantages.

1) Some parameters need to be determined via numerical tests, as these parameters usually do not have theoretical optimum values.

2) The system matrix is usually asymmetric, which affects the efficiency of the method.

Much more research work is needed to improve MFree local weak-form methods, especially in dealing with the integrations for nodes near and on the boundaries, and the asymmetry of the discretized system equations.

APPENDIX

Appendix 5.1. Major subroutines used in MFree_Local.f90 and their functions

Subroutines	Functions	Location
Input	Input data from the external data file	Program 5.3
Qdomain	Set quadrature domain for a field node	Program 5.4
GaussCoefficient	Obtain coefficients of Gauss points	Program 4.5
DomainGaussPoints	Set Gauss points for a quadrature domain	Program 5.5
SupportDomain	Determine the support domain for a quadrature point	Program 4.7
RPIM_ShapeFunc_2D (or MLS_ShapeFunc_2D)	Compute shape functions and their derivatives of an interpolation point	Program 3.1 (Program 3.9)
TestFunc	Compute the cubic spline weight function	Program 5.6
Integration_BCQuQi	Compute boundary integrations on Γ_{qu} and Γ_{qi}	Program 5.7
Integration_BCQt	Compute boundary integration on Γ_{qt}	Program 5.8
EssentialBC	Enforce essential boundary conditions	Program 5.9
SolverBand	Solve system equations	Program 4.12
GetDisplacement	Obtain the actual displacements using the RPIM or the MLS shape functions	Program 5.10
GetNodeStress	Retrieve the strain and stress for field nodes	Program 5.11
Output	Output results	Program 5.12
TotalGaussPoints	Set Gauss points for global cells	Program 5.13
GetEnergyError	Compute error in energy norm	Program 5.14
GetInvasy	Obtain the inversion of a matrix	Program 4.14
Dobmax	Compute multiplication of two matrices	Program 5.15

Appendix 5.2. The global variables used in MFree_Global.f90

Variable	**Type**	**Usage**	**Function**
numnode	Integer	Input	Number of field nodes
x(*nx*, *numnode*)	Long real	Input	Coordinates x and y for all field nodes: $x(1, i)=x_i$; $x(2, i)=y_i$
xc(*nx*, 4)	Long real	Work array	Coordinates x and y for a rectangular quadrature domain: $xc(1,i)=x_i$; $xc(2,i)=y_i$
ngx,ngy	Integer	Input	Number of sub-partitions for a quadrature domain in x and y directions
nquado	Integer	Input	Number of Gauss points used in one dimension in a partition.
npEBCnum,	Integer	Input	Number of field nodes with essential boundary conditions
npEBC(3,100), **pEBC**(*nx*,100)	Integer long real	Input	Essential boundary condition.
npNBCnum,	Integer	Input	Number of field nodes with natural boundary conditions
npNBC(3,100), **pNBC(*nx*,100)**	Integer long real	Input	Natural boundary condition
alfs	Long real	Input	Dimensionless sizes of support (influence) domains
Ds*(nx, numnode)*	Long real	Work array	The size of the influence domain: $ds(1,i)=d_{sxi}$, $ds(2,i)=d_{syi}$
ndex	Integer	Input	Number of field nodes in the local domain
Ph(10, *ndex*)	Long real	Output	Shape functions and their derivatives:
Ak(2**numnode*, 2**numnode*)	Long real	Work array	Global stiffness matrix
Force(2**numnode*)	Long real	Work array	Global force vector
disp(2**numnode*)	Long real	Work array	Displacement vector: $disp(2*i-1)=u_i$; $disp(2*i-1)=v_i$
Stress(3, *numnode*)	Long real	Work array	The array to store the stress components for all field nodes

Appendix 5.3. Dummy arguments used in the subroutine Qdomain

Variable	Type	Usage	Function
rqx, rqy	Long real	Input	Sizes of the quadrature domain
xn,yn	Long real	Input	Coordinates of the field node considered
xm(4)	Long real	Input	Geometrical description of the global boundary (designed for a rectangular domain): $xm(1)=x_{min}$; $xm(2)=x_{max}$, $xm(3)=y_{max}$; $xm(4)=y_{min}$
xc(*nx*, 4)	Long real	Output	Coordinates x and y for a rectangular quadrature domain: $xc(1,i)=x_i$; $xc(2,i)=y_i$

Appendix 5.4 Dummy arguments used in the subroutine DomainGaussPoints

Variable	Type	Usage	Function
xc(*nx*, 4)	Long real	Input	Coordinates x and y for a rectangular quadrature domain: $xc(1,i)=x_i$; $xc(2,i)=y_i$
Gauss(*nx,nquado*)	Long real	Input	Coefficients of Gauss point
nquado	Integer	Input	Number of Gauss points used in 1D in the domain considered. For a rectangular partition, total Gauss points is *nquado*×*nquado*.
numgauss	Integer	Input	Total number of Gauss points for a domain. It is *nquado* × *nquado*.
nxc	Integer	Input	*nxc*=4 for a rectangular quadrature domain
gs(4, *numgauss*)	Integer	Output	Gauss points for a cell: $gs(1,i)$: x for Gauss point i; $gs(2,i)$: y for Gauss point i; $gs(3,i)$: Gauss weighting factor; $gs(4,i)$: Jacobian value for this cell

Appendix 5.5. Dummy arguments used in the subroutine Integration_BCQuQi

Variable	Type	Usage	Function
nod	Integer	Input	ID of the field node considered
numnode	Integer	Input	Total number of field nodes
x(*nx*, *numnode*)	Long real	Input	Coordinates x and y for all field nodes. $x(1,i)=x_i$; $x(2,i)=y_i$
xc(*nx*, 4)	Long real	Output	Coordinates x and y for a rectangular quadrature domain: $xc(1,i)=x_i$; $xc(2,i)=y_i$
nquado	Integer	Input	Number of Gauss points used in the domain considered.
xspace,yspace	Long real	Input	Sizes of the quadrature domain (e.g. *rqx, rqy*)
xm(4)	Long real	Input	Geometrical description of the global boundary (designed for a rectangular domain): $xm(1)=x_{min}$; $xm(2)=x_{max}$, $xm(3)=y_{max}$; $xm(4)=y_{min}$
Ds*(nx, numnode)*	Long real	Input	The size of the influence domain. $ds(1,i)=d_{sxi}$, $ds(2,i)=d_{syi}$
alfs	Long real	Input	Dimensionless coefficient for support (influence) domain
mk	Integer	Input	Maxium number of rows of **Ak**
Ak	Long real	Input output	Global stiffness matrix

Appendix 5.6. Dummy arguments used in the subroutine Integration_BCQt

Variable	Type	Usage	Function
nod	Integer	Input	ID of the field node considered
numnode	Integer	Input	Total number of field nodes
x(*nx*, *numnode*)	Long real	Input	Coordinates x and y
xc(*nx*, 4)	Long real	Output	Coordinates x and y for a rectangular quadrature domain
nquado	Integer	Input	Number of Gauss points used in the domain considered.
xspace,yspace	Long real	Input	Sizes of the quadrature domain (e.g. *rqx, rqy*)

xm(4)	Long real	Input	Geometrical description of the global boundary
Xcent(2)	Long real	Input	x and y coordinates for the field node considered
f(2)	Long real	output	Nodal force vector

Appendix 5.7. Dummy arguments used in the subroutine EssentialBC

Variable	Type	Usage	Function
numnode	Integer	Input	Total number of field nodes
alfs	Long real	Input	Dimensionless size for a support (influence) domain
Ds*(nx, numnode)*	Long real	Input	The size of the influence domain
npEBCnum	Integer	Input	Number of field nodes with essential boundary conditions
npEBC(3,100), **pEBC**(*nx*,100)	Integer, long real	Input	Essential boundary condition
mk	Integer	Input	Maxium number of rows of **Ak**
Ak(2**numnode*, 2**numnode*)	Long real	Input and output	Global stiffness matrix
Fk(2**numnode*)	Long real	Input and output	Global force vector

Appendix 5.8. Dummy arguments used in the subroutine GetNodeStress

Variable	Type	Usage	Function
nx	Integer	Input	*nx*=2 for 2-D problem
numnode	Integer	Input	Total number of field nodes
x(*nx*, *numnode*)	Long real	Input	Coordinates x and y for all field nodes
alfs	Long real	Input	Dimensionless size for the support (influence) domain
Ds*(nx, numnode)*	Long real	Input	The size of the influence domain
U2(2, *numnode*)	Long real	Input	Displacement vector
Stress	Long real	Output	Stress matrix

Appendix 5.9. The data file:Local_Input55.dat used in MFree_Local.f90. A total of 55 regular field nodes is used

```
*L,H,E,v,P,
48. 12. 3.e7 .3 1000.
*numnode
55
* Global BC: Xmin,Xmax,Ymax, Ymin
0. 48. 6. -6.
* Nodal spacing: Dcx,Dcy
4.8 3.0
* Local quadrature domain: Aqx,Aqy
2. 2.
* Num. of sub-partitions: Nsx,Nsy
2 2
*Influence domain
3.
*Num. of Gauss Points
4
*RBF shape parameters: nRBF ALFc, dc and q
1 1.0 3.0 1.03
*Num. of Basis
3
*Field nodes: x[xi,yi]
 1      0.0000      6.0000         29     24.0000     -3.0000
 2      0.0000      3.0000         30     24.0000     -6.0000
 3      0.0000      0.0000         31     28.8000      6.0000
 4      0.0000     -3.0000         32     28.8000      3.0000
 5      0.0000     -6.0000         33     28.8000      0.0000
 6      4.8000      6.0000         34     28.8000     -3.0000
 7      4.8000      3.0000         35     28.8000     -6.0000
 8      4.8000      0.0000         36     33.6000      6.0000
 9      4.8000     -3.0000         37     33.6000      3.0000
10      4.8000     -6.0000         38     33.6000      0.0000
11      9.6000      6.0000         39     33.6000     -3.0000
12      9.6000      3.0000         40     33.6000     -6.0000
13      9.6000      0.0000         41     38.4000      6.0000
14      9.6000     -3.0000         42     38.4000      3.0000
15      9.6000     -6.0000         43     38.4000      0.0000
16     14.4000      6.0000         44     38.4000     -3.0000
17     14.4000      3.0000         45     38.4000     -6.0000
18     14.4000      0.0000         46     43.2000      6.0000
19     14.4000     -3.0000         47     43.2000      3.0000
20     14.4000     -6.0000         48     43.2000      0.0000
21     19.2000      6.0000         49     43.2000     -3.0000
22     19.2000      3.0000         50     43.2000     -6.0000
23     19.2000      0.0000         51     48.0000      6.0000
24     19.2000     -3.0000         52     48.0000      3.0000
25     19.2000     -6.0000         53     48.0000      0.0000
26     24.0000      6.0000         54     48.0000     -3.0000
27     24.0000      3.0000         55     48.0000     -6.0000
28     24.0000      0.0000
*Num. of Essential BC: numFBC
5
*Node,iUx,iUy,Ux,Uy
   1    1   1   0.000000000000E+00 -0.599999982119E-04
   2    1   1  -0.718749978580E-05 -0.149999995530E-04
   3    1   1   0.000000000000E+00  0.000000000000E+00
   4    1   1   0.718749978580E-05 -0.149999995530E-04
```

```
    5    1   1   0.000000000000E+00 -0.599999982119E-04
 *Num. of concentated loading: numFBC
 5
 *Node,iTx,iTy,Tx,Ty
 51 1 1  0  0.0   52 1 1  0. 0.0
 53 1 1  0. 0.0   54 1 1  0. 0.0
 55 1 1  0. 0.0
 * Num. of nodes and cells(for en. error)
 55 40
 *Nodes for cells: xc[  ]
 1      0.0000      6.0000          29     24.0000     -3.0000
 2      0.0000      3.0000          30     24.0000     -6.0000
 3      0.0000      0.0000          31     28.8000      6.0000
 4      0.0000     -3.0000          32     28.8000      3.0000
 5      0.0000     -6.0000          33     28.8000      0.0000
 6      4.8000      6.0000          34     28.8000     -3.0000
 7      4.8000      3.0000          35     28.8000     -6.0000
 8      4.8000      0.0000          36     33.6000      6.0000
 9      4.8000     -3.0000          37     33.6000      3.0000
10      4.8000     -6.0000          38     33.6000      0.0000
11      9.6000      6.0000          39     33.6000     -3.0000
12      9.6000      3.0000          40     33.6000     -6.0000
13      9.6000      0.0000          41     38.4000      6.0000
14      9.6000     -3.0000          42     38.4000      3.0000
15      9.6000     -6.0000          43     38.4000      0.0000
16     14.4000      6.0000          44     38.4000     -3.0000
17     14.4000      3.0000          45     38.4000     -6.0000
18     14.4000      0.0000          46     43.2000      6.0000
19     14.4000     -3.0000          47     43.2000      3.0000
20     14.4000     -6.0000          48     43.2000      0.0000
21     19.2000      6.0000          49     43.2000     -3.0000
22     19.2000      3.0000          50     43.2000     -6.0000
23     19.2000      0.0000          51     48.0000      6.0000
24     19.2000     -3.0000          52     48.0000      3.0000
25     19.2000     -6.0000          53     48.0000      0.0000
26     24.0000      6.0000          54     48.0000     -3.0000
27     24.0000      3.0000          55     48.0000     -6.0000
28     24.0000      0.0000
 *No. of nodes in cells[1,2,3,4]
      1   1   2   7   6                  21  26  27  32  31
      2   2   3   8   7                  22  27  28  33  32
      3   3   4   9   8                  23  28  29  34  33
      4   4   5  10   9                  24  29  30  35  34
      5   6   7  12  11                  25  31  32  37  36
      6   7   8  13  12                  26  32  33  38  37
      7   8   9  14  13                  27  33  34  39  38
      8   9  10  15  14                  28  34  35  40  39
      9  11  12  17  16                  29  36  37  42  41
     10  12  13  18  17                  30  37  38  43  42
     11  13  14  19  18                  31  38  39  44  43
     12  14  15  20  19                  32  39  40  45  44
     13  16  17  22  21                  33  41  42  47  46
     14  17  18  23  22                  34  42  43  48  47
     15  18  19  24  23                  35  43  44  49  48
     16  19  20  25  24                  36  44  45  50  49
     17  21  22  27  26                  37  46  47  52  51
     18  22  23  28  27                  38  47  48  53  52
     19  23  24  29  28                  39  48  49  54  53
20  24  25  30  29                     40  49  50  55  54
 *END of data file
```

Appendix 5.10. A output sample for displacements obtained using MQ LRPIM

No. of field nodes	*u*	*v*
1	0.56898E-13	-0.60000E-04
2	-0.71875E-05	-0.15000E-04
3	-0.19977E-13	0.11007E-13
4	0.71875E-05	-0.15000E-04
5	0.23840E-13	-0.60000E-04
6	0.31081E-03	-0.20687E-03
7	0.15043E-03	-0.16341E-03
8	-0.11083E-13	-0.15038E-03
9	-0.15043E-03	-0.16341E-03
10	-0.31081E-03	-0.20687E-03
.....	.....	.....
31	0.13105E-02	-0.38899E-02
32	0.64903E-03	-0.38727E-02
33	0.29249E-14	-0.38668E-02
34	-0.64903E-03	-0.38727E-02
35	-0.13105E-02	-0.38899E-02
36	0.14157E-02	-0.50129E-02
37	0.70169E-03	-0.50000E-02
38	0.27869E-14	-0.49955E-02
39	-0.70169E-03	-0.50000E-02
40	-0.14157E-02	-0.50129E-02
41	0.14905E-02	-0.62077E-02
42	0.73916E-03	-0.61990E-02
43	0.28311E-14	-0.61960E-02
44	-0.73916E-03	-0.61990E-02
45	-0.14905E-02	-0.62077E-02
46	0.15364E-02	-0.74499E-02
47	0.76229E-03	-0.74455E-02
48	0.28175E-14	-0.74440E-02
49	-0.76229E-03	-0.74455E-02
50	-0.15364E-02	-0.74499E-02
51	0.15513E-02	-0.87164E-02
52	0.76992E-03	-0.87171E-02
53	0.28155E-14	-0.87169E-02
54	-0.76992E-03	-0.87171E-02
55	-0.15513E-02	-0.87164E-02

*The parameters used are

$\alpha_c = 1.0,\ q = 1.03$ and $d_i = 3.0$ for MQ RBF;

$d_{cx} = 4.8,\ d_{cy} = 3.0,$ and $\alpha_s = 3.0$ for the local influence domains;

$\alpha_q = 2.0$ and $n_g \times n_g = 2 \times 2$ for local quadrature domains;.

The linear polynomial terms are added in MQ RPIM;

The cubic spline function is used as the test function for the local Petrov_galerkin weak form.

Appendix 5.11. A output sample for stress obtained using MQ LRPIM

No. of field nodes	σ_{xx}	σ_{yy}	τ_{xy}
.....	.....	.....	.....
21	0.10836E+04	-0.52377E+02	-0.69790E+02
22	0.54814E+03	0.10130E+02	-0.95905E+02
23	0.63871E-08	0.79569E-08	-0.14641E+03
24	-0.54814E+03	-0.10130E+02	-0.95905E+02
25	-0.10836E+04	0.52377E+02	-0.69790E+02
26	0.89328E+03	-0.48896E+02	-0.68019E+02
27	0.45566E+03	0.24844E+01	-0.90400E+02
28	-0.34138E-08	-0.16003E-08	-0.13671E+03
29	-0.45566E+03	-0.24844E+01	-0.90400E+02
30	-0.89328E+03	0.48896E+02	-0.68019E+02
31	0.71423E+03	-0.36210E+02	-0.66052E+02
32	0.36336E+03	0.50542E+01	-0.89572E+02
33	0.16079E-08	0.45941E-09	-0.13599E+03
34	-0.36336E+03	-0.50542E+01	-0.89572E+02
35	-0.71423E+03	0.36210E+02	-0.66052E+02
36	0.53176E+03	-0.28593E+02	-0.65551E+02
37	0.27039E+03	0.25663E+01	-0.87817E+02
38	-0.54533E-09	0.11596E-10	-0.13328E+03
39	-0.27039E+03	-0.25663E+01	-0.87817E+02
40	-0.53176E+03	0.28593E+02	-0.65551E+02
41	0.35854E+03	-0.17493E+02	-0.64554E+02
42	0.18269E+03	0.15669E+01	-0.86972E+02
43	0.17923E-09	-0.52410E-10	-0.13191E+03
44	-0.18269E+03	-0.15669E+01	-0.86972E+02
45	-0.35854E+03	0.17493E+02	-0.64554E+02
46	0.15814E+03	-0.17753E+02	-0.66139E+02
47	0.75874E+02	-0.36322E+01	-0.88991E+02
48	-0.68326E-10	0.85947E-10	-0.13401E+03
49	-0.75874E+02	0.36322E+01	-0.88991E+02
50	-0.15814E+03	0.17753E+02	-0.66139E+02
.....	.....	.....	.....
			Energy error:= 0.2419E+00

*The parameters used are

$\alpha_c = 1.0,\ \ q = 1.03$ and $d_i = 3.0$ for MQ RBF;

$d_{cx} = 4.8,\ \ d_{cy} = 3.0,$ and $\alpha_s = 3.0$ for the local influence domains;

$\alpha_q = 2.0$ and $n_g \times n_g = 2 \times 2$ for local quadrature domains;.

The linear polynomial terms are added in MQ RPIM;

The cubic spline function is used as the test function for the local Petrov_galerkin weak form.

Appendix 5.12. A output sample for displacements obtained using MLPG

No. of field nodes	*u*	*v*
1	0.34272E-15	-0.60000E-04
2	-0.71875E-05	-0.15000E-04
3	-0.39980E-16	0.68236E-15
4	0.71875E-05	-0.15000E-04
5	-0.46273E-15	-0.60000E-04
6	0.29397E-03	-0.20657E-03
7	0.14080E-03	-0.16710E-03
8	0.22333E-16	-0.15389E-03
9	-0.14080E-03	-0.16710E-03
10	-0.29397E-03	-0.20657E-03
.....	.....	.....
31	0.13229E-02	-0.38436E-02
32	0.65460E-03	-0.38257E-02
33	-0.90234E-17	-0.38197E-02
34	-0.65460E-03	-0.38257E-02
35	-0.13229E-02	-0.38436E-02
36	0.14346E-02	-0.49791E-02
37	0.71048E-03	-0.49657E-02
38	-0.10666E-16	-0.49612E-02
39	-0.71048E-03	-0.49657E-02
40	-0.14346E-02	-0.49791E-02
41	0.15146E-02	-0.61915E-02
42	0.75033E-03	-0.61826E-02
43	-0.12135E-16	-0.61796E-02
44	-0.75033E-03	-0.61826E-02
45	-0.15146E-02	-0.61915E-02
46	0.15619E-02	-0.74550E-02
47	0.77406E-03	-0.74506E-02
48	-0.12619E-16	-0.74491E-02
49	-0.77406E-03	-0.74506E-02
50	-0.15619E-02	-0.74550E-02
51	0.15784E-02	-0.87437E-02
52	0.78212E-03	-0.87439E-02
53	-0.18770E-16	-0.87438E-02
54	-0.78212E-03	-0.87439E-02
55	-0.15784E-02	-0.87437E-02

*The parameters used are

$d_{cx} = 4.8,\ d_{cy} = 3.0,$ and $\alpha_s = 3.0$ for the local influence domains;

$\alpha_q = 1.5$ and $n_g \times n_g = 2 \times 2$ for local quadrature domains;.

The linear polynomial basis and the cubic spline weight function are used in the MLS approximation;

The cubic spline function is as the test function for the local weak form.

Appendix 5.13. A output sample for stress obtained using MLPG

No. of field nodes	σ_{xx}	σ_{yy}	τ_{xy}
.....	.....	.....	.....
21	0.11874E+04	0.68411E+01	-0.18401E+02
22	0.59427E+03	-0.78572E+00	-0.88914E+02
23	-0.25722E-10	-0.10118E-10	-0.12424E+03
24	-0.59427E+03	0.78572E+00	-0.88914E+02
25	-0.11874E+04	-0.68411E+01	-0.18401E+02
26	0.99634E+03	0.62591E+01	-0.16038E+02
27	0.49745E+03	-0.23578E+00	-0.86858E+02
28	0.31903E-11	-0.27569E-11	-0.12228E+03
29	-0.49745E+03	0.23578E+00	-0.86858E+02
30	-0.99634E+03	-0.62591E+01	-0.16038E+02
31	0.79699E+03	0.48532E+01	-0.18323E+02
32	0.39835E+03	-0.34690E+00	-0.90428E+02
33	-0.20520E-10	0.63380E-11	-0.12649E+03
34	-0.39835E+03	0.34690E+00	-0.90428E+02
35	-0.79699E+03	-0.48532E+01	-0.18323E+02
36	0.59791E+03	0.35348E+01	-0.16586E+02
37	0.29898E+03	-0.27567E+00	-0.88766E+02
38	-0.56843E-11	-0.97771E-11	-0.12486E+03
39	-0.29898E+03	0.27567E+00	-0.88766E+02
40	-0.59791E+03	-0.35348E+01	-0.16586E+02
41	0.39737E+03	0.24781E+01	-0.19015E+02
42	0.19772E+03	-0.17138E+00	-0.92160E+02
43	-0.10289E-10	0.14779E-11	-0.12879E+03
44	-0.19772E+03	0.17138E+00	-0.92160E+02
45	-0.39737E+03	-0.24781E+01	-0.19015E+02
46	0.19294E+03	-0.39955E-01	-0.16144E+02
47	0.97666E+02	-0.45676E+00	-0.88943E+02
48	0.79581E-12	0.86402E-11	-0.12531E+03
49	-0.97666E+02	0.45676E+00	-0.88943E+02
50	-0.19294E+03	0.39955E-01	-0.16144E+02
.....	.....	.....	.....
			Energy error:= 0.5573E-01

*The parameters are

$d_{cx} = 4.8,\ d_{cy} = 3.0,$ and $\alpha_s = 3.0$ for the local influence domains;

$\alpha_q = 1.5$ and $n_g \times n_g = 2 \times 2$ for local quadrature domains;.

The linear polynomial basis and the cubic spline weight function are used in the MLS approximation;

The cubic spline function is as the test function for the local weak form.

COMPUTER PROGRAMS

Program 5.1. The include file VariablesLocal.h

```
      parameter(nx=2,ng=4,ndim=600)
      common/para/xlength,ylength,p,young,anu,aimo
      common/rpim/ALFC,DC,Q,nRBF
      common/basis/mbasis
      common/localdomains/dcx,dcy,dex,dey,ngx,ngy
      dimension Dmat(3,3),x(nx,ndim),conn(ng,ndim),xBK(nx,ndim)
      dimension npEBC(3,100),pEBC(2,100)
      dimension nbc(100),ibcn(2,100),bcn(2,100),xnbcl(2,100)
      dimension nv(ndim),gpos(nx),gauss(nx,20),xm(4)
      dimension phi(10,ndim),ds(2,ndim)
      dimension gss(4,ndim), gst(4,10*ndim)
      dimension ak(2*ndim,2*ndim),fk(2*ndim)
      dimension xc(2,4),xcc(2,4),dsi(2),xcent(2),f2(2)
      dimension fbcl(2,4)
      dimension u2(2,ndim),u22(2,ndim),displ(2*ndim),stress(3,ndim)
      dimension bb(3,2),bbt(2,3),ww(3,2),ek(2,2),bd(2,3)
```

Program 5.2. The main program of MFree_local.f90

```
!-------------------------------------------------------------------------
! main program--2D FORTRAN 90 CODE-MFree local weak-form methods
! Using rectangular support domain and rectangular background cells
! input file   -- input.dat
! output file  -- result.dat
! include file -- variablelocal.h
!-------------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      include 'variableslocal.h'
      ir=4
      open(ir,file=' Local_Input55.dat ',status='old')
      open(2,file='result.dat',status='unknown')
      maxmatrix=2*ndim

! ************** Input boundaries / parameters
      call Input(ir,x,ndim,nx,numnode,xm, &
                 nquado,Dmat,ALFs,numcell,numq,xBK,conn,&
                 npEBCnum,npEBC,pEBC,npNBCnum,nbc,ibcn,bcn)

! ************** Determine domains of influence--uniform nodal spacing
      xspace=dcx*dex
      yspace=dcy*dey
      xstep=xspace/dex
      ystep=yspace/dey
      do j=1,numnode
         ds(1,j)=alfs*xstep
         ds(2,j)=alfs*ystep
      enddo
! ************** Coefficients of Gauss points,Weights and Jacobian for each cell
      call GaussCoefficient(nquado,gauss)
      eps=1.e-16
      b=-100*eps
      do iak=1,2*numnode
         fk(iak)=0.0
         do jak=1,2*numnode
            ak(iak,jak)=0.
```

```
         enddo
      enddo

! ************* Loop for field nodes
      do 100 nod=1,numnode
         write(*,*)'Field Node=',nod
         xn=x(1,nod)
         yn=x(2,nod)
         xss=xspace
         yss=yspace
         numgauss=nquado*nquado
         call QDomain(xss,yss,xn,yn,xm,xc) ! Local quadrature domain
         nxc=ng         ! for the rectangular domain
! ************* Local quadrature domain is divided to sub-paritions
         xgs=(xc(1,4)-xc(1,1))/ngx
         ygs=(xc(2,1)-xc(2,2))/ngy
         x0=xc(1,1)
         do 60 iix=1,ngx
            xx=x0+(iix-1)*xgs
            y0=xc(2,1)
            do 60 jjy=1,ngy
               yy=y0-(jjy-1)*ygs
               xcc(1,1)=xx
               xcc(2,1)=yy
               xcc(1,2)=xx
               xcc(2,2)=yy-ygs
               xcc(1,3)=xx+xgs
               xcc(2,3)=yy-ygs
               xcc(1,4)=xx+xgs
               xcc(2,4)=yy
               call DomainGaussPoints(xcc,gauss,gss,nx,ng,nxc,nquado,numgauss)

! ************* Loop quadrature points
               numgauss=nquado*nquado
               do 30  ie=1,numgauss
                  gpos(1)=gss(1,ie)
                  gpos(2)=gss(2,ie)
                  weight=gss(3,ie)
                  ajac=gss(4,ie)
                   ndex=0
                  call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
                  do kph=1,ndex
                        do ii=1,10
                        phi(ii,kph)=0.
                     enddo
                  enddo
                  dsi(1)=xspace
                  dsi(2)=yspace
                  xcent(1)=xn
                  xcent(2)=yn
                  call TestFunc(dsi,xcent,gpos,w,wx,wy)
                  Call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,alfc,dc,&
                                         q,nRBF, mbasis)
                  ik1=nod*2-1
                  ik2=nod*2
! ************* Get Stiffness Matrix
                  do ine=1,ndex
                     n1=2*nv(ine)-1
                     n2=2*nv(ine)
                     do ii=1,3
                        do jj=1,2
                           bbt(jj,ii)=0.
                           bb(ii,jj)=0.
                           ww(ii,jj)=0.
                        enddo
                     enddo
                     bb(1,1)=phi(2,ine)
                     bb(2,2)=phi(3,ine)
                     bb(3,1)=phi(3,ine)
                     bb(3,2)=phi(2,ine)
```

```
                        ww(1,1)=wx
                        ww(2,2)=wy
                        ww(3,1)=wy
                        ww(3,2)=wx
                        do ii=1,3
                           do jj=1,2
                              bbt(jj,ii)=ww(ii,jj)
                           enddo
                        enddo
                        call DOBMAX(bbt,2,3,2,dmat,3,3,bd,2)
                        call dobmax(bd,2,3,2,bb,2,3,ek,2)
                        ak(ik1,n1)=ak(ik1,n1)+weight*ajac*ek(1,1)
                        ak(ik1,n2)=ak(ik1,n2)+weight*ajac*ek(1,2)
                        ak(ik2,n1)=ak(ik2,n1)+weight*ajac*ek(2,1)
                        ak(ik2,n2)=ak(ik2,n2)+weight*ajac*ek(2,2)
                     enddo
 30                continue  !End of integration for local quadrature domain

! *************** B.C. Integrations
                   NNQ=nquado
                   call Integration_BCQt(nx,ng,xcc,f2,x,numnode,NNQ,&
                                         xm,xss,yss,xcent)
                   fk(2*nod-1)=fk(2*nod-1)+f2(1)
                   fk(2*nod)=fk(2*nod)+f2(2)
                   call Integration_BCQuQi(nx,ng,nod,xcc,x,numnode,nNQ,dmat,xm,xss&
                        ,YSS,ak,maxmatrix,alfs,ds)
 60          continue
 100      continue !   End of loop for field nodes

! *************** Boundary conditions: essential
          call EssentialBC(x,numnode,ak,fk,maxmatrix,ds,alfs,npEBCnum,npEBC,pEBC)

! *************** Solve equation to get the solutions
          ep=1.0e-20
          neq=2*numnode
          write(*,*)'Solve equation...'
          call SolverBand(ak,fk,neq,maxmatrix)
          do kk=1,numnode
             u2(1,kk)=fk(2*kk-1)
             u2(2,kk)=fk(2*kk)
          enddo
! *************** get the final displacement
          call GetDisplacement(x,ds,u2,displ,alfs,nx,numnode)
          do kk=1,numnode
             u22(1,kk)=displ(2*kk-1)
             u22(2,kk)=displ(2*kk)
          enddo

! *************** Get stress
          call GetNodeStress(x,ds,Dmat,u2,Stress,alfs,nx,numnode)
          call Output(x,numnode,u2,u22,Stress)   ! ouput results

! *************** Get energy error using global BK cells
          write(*,*)'Computing global energy error...'
          ngst=numcell*nquado**2
          call TotalGaussPoints(xBK,conn,gauss,gst,nx,ng,&
                                numq,numcell,nquado,ngst)
          call GetEnergyError(nx,ng,xBK,numq,u2,dmat,ds,&
                              ngst,gst,alfs)

          write(*,*)'THE END'
     STOP
     END
```

Program 5.3. Source code of Subroutine Input

```
   SUBROUTINE Input(ir,x,numd,nx,numnode,xm,nquado,Dmat,ALFs,numcell,numq,&
```

```
                    xc,conn,npEBCnum,npEBC,pEBC,npNBCnum,nNBC,npNBC,pNBC)
!------------------------------------------------------------------------------
! This subroutine is to input data from data file
! input--ir
! output—all other variables
!------------------------------------------------------------------------------

       implicit real*8 (a-h,o-z)
       common/para/xlength,ylength,p,young,anu,aimo
       COMMON/rpim/ALFC,DC,Q,nRBF
       common/basis/mbasis
       common/localdomains/dcx,dcy,dex,dey,ngx,ngy
       CHARACTER*50 NAM
       dimension npEBC(3,100),pEBC(2,100)
       dimension nNBC(100),npNBC(2,100),pNBC(2,100)
       dimension x(nx,numd),Dmat(3,3),xm(4)
       dimension conn(4,numd),xc(nx,numd)
       read(4,10)nam
       read(ir,*) xlength,ylength,young,anu,p
       read(ir,10)nam
       read(ir,*)numnode
          read(ir,10)nam
       read(ir,*)xm(1),xm(2),xm(3),xm(4)
          read(ir,10)nam
       read(ir,*)dcx,dcy
        read(ir,10)nam
       read(ir,*)dex,dey
        read(ir,10)nam
       read(ir,*)ngx,ngy
       read(ir,10)nam
       read(ir,*)ALFs
       read(ir,10)nam
       read(ir,*)nquado
       read(ir,10)nam
       READ(ir,*)nRBF, alfc,dc,q
       read(ir,10)nam
       READ(ir,*)mbasis
       read(ir,10)nam
       do i=1,numnode
          read(ir,*)j,x(1,i),x(2,i)
       enddo
       read(ir,10)nam
       read(ir,*)npEBCnum
       read(ir,10)nam
       do i=1,npEBCnum
          read(ir,*)npEBC(1,i),npEBC(2,i),npEBC(3,i),pEBC(1,i),pEBC(2,i)
       enddo
       read(ir,10)nam
       read(ir,*)npNBCnum
       read(ir,10)nam
       do i=1,npNBCnum
          read(ir,*)nNBC(i),npNBC(1,i),npNBC(2,i),pNBC(1,i),pNBC(2,i)
       enddo
       read(ir,10)nam
       read(ir,*)numq,numcell
       read(ir,10)nam
       do i=1,numq
          read(ir,*)j,xc(1,i),xc(2,i)
       enddo
       read(ir,10)nam
       do j=1,numcell
          read(ir,*)i,conn(1,j),conn(2,j),conn(3,j),conn(4,j)
       enddo

! ************* Compute material matrix D[] for the plane stress
       you=young/(1.-anu*anu)
       aimo=(1./12.)*ylength**3
       Dmat(1,1)=you
       Dmat(1,2)=anu*you
       Dmat(1,3)=0.
```

```
      Dmat(2,1)=anu*you
      Dmat(2,2)=you
      Dmat(2,3)=0.
      Dmat(3,1)=0.
      Dmat(3,2)=0.
      Dmat(3,3)=0.5*(1.-anu)*you
 10   format(a50)
  RETURN
  END
```

Program 5.4. Source code of Subroutine Qdomain

```
  SUBROUTINE QDomain(xs,ys,x,y,xm,xc)
!-------------------------------------------------------------------------
! This subroutine is to construct local quadrature domain for a field node
! input--xs, ys: sizes of quadrature domain;
!       x,y: coordinates of the field node;
!       xm(4): (xmin, xmax,ymax,ymin) for the global boundary;
! output-- xc(2,4): coordinates of points for the quadrature domain;!
!-------------------------------------------------------------------------

      implicit real*8 (a-h,o-z)
      common/para/xlength,ylength,p,young,anu,aimo
      common/node/numnode,numcell,dex,dey,nquado
      dimension xm(4),xc(2,4)
      xl=x-xs
      xr=x+xs
      yu=y+ys
      yd=y-ys
      if(xl.le.xm(1)) xl=xm(1)
      if(xr.ge.xm(2)) xr=xm(2)
      if(yu.ge.xm(3)) yu=xm(3)
      if(yd.le.xm(4)) yd=xm(4)
      xc(1,1)=xl
      xc(2,1)=yu
      xc(1,2)=xl
      xc(2,2)=yd
      xc(1,3)=xr
      xc(2,3)=yd
      xc(1,4)=xr
      xc(2,4)=yu
  RETURN
  END
```

Program 5.5. Source code of Subroutine DomainGaussPoints

```
  SUBROUTINE DomainGaussPoints(xc,gauss,gs,nx,ng,nxc,k,numgauss)
!-------------------------------------------------------------------------
! This subroutine is to set up Gauss points,Jacobian and weights
!        for a the local quadrature domaincell
! input--nxc: number of vertexes of the local quadrature domain, nxc=4;
!        numgauss: number of Gauss points in the domain;
!        k: number of Gauss points used, numgauss=k*k for 2-D domain;
!        xc(nx,nxc): coordinates of points for background cells;
!        gauss(2,k): coefficients of Gauss points;
!        nx,ng: parameters are defined in file parameter.h.
! output--gs(ng,numgauss): coordinate of the Gauss points, weight and Jacobian
!-------------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      dimension xc(nx,nxc),gauss(nx,k)
      dimension gs(ng,numgauss),psiJ(ng),etaJ(ng),xe(ng),ye(ng),aN(ng)
      dimension aNJpsi(ng),aNJeta(ng)
      index=0
      psiJ(1)=-1.
```

```
      psiJ(2)=1.
      psiJ(3)=1.
      psiJ(4)=-1.
      etaJ(1)=-1.
      etaJ(2)=-1.
      etaJ(3)=1.
      etaJ(4)=1.
      l=k
      do j=1,ng
         xe(j)=xc(1,j)
         ye(j)=xc(2,j)
      enddo
      do 80 i=1,l
         do 80 j=1,l
            index=index+1
            eta=gauss(1,i)
            psi=gauss(1,j)
            do ik=1,ng
               aN(ik)=.25*(1.+psi*psiJ(ik))*(1.+eta*etaJ(ik))
               aNJpsi(ik)=.25*psiJ(ik)*(1.+eta*etaJ(ik))
               aNJeta(ik)=.25*etaJ(ik)*(1.+psi*psiJ(ik))
            enddo
            xpsi=0.
            ypsi=0.
            xeta=0.
            yeta=0.
            do jk=1,ng
               xpsi=xpsi+aNJpsi(jk)*xe(jk)
               ypsi=ypsi+aNJpsi(jk)*ye(jk)
               xeta=xeta+aNJeta(jk)*xe(jk)
               yeta=yeta+aNJeta(jk)*ye(jk)
            enddo
            ajcob=xpsi*yeta-xeta*ypsi
            xq=0.
            yq=0.
            do kk=1,ng
               xq=xq+aN(kk)*xe(kk)
               yq=yq+aN(kk)*ye(kk)
            enddo
            gs(1,index)=xq
            gs(2,index)=yq
            gs(3,index)=gauss(2,i)*gauss(2,j)
            gs(4,index)=ajcob
80    continue
 RETURN
 END
```

Program 5.6. Source code of Subroutine TestFunc

```
   SUBROUTINE TestFunc (dsi,xcent,xg,w,wxx,wyy)
!---------------------------------------------------------------------
! Cubic spline test (weight) function
! input—dsi: size of weight domain;
!       xcent: center of the weight domain;
!       xg: coordinate of point considered;
! output—w, wxx,wyy
!---------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)
      dimension dsi(2),xcent(2)
      dimension xg(2)
      ep=1.e-15
      difx=xg(1)-xcent(1)
      dify=xg(2)-xcent(2)
      if(dabs(difx).le.ep) then
         drdx=0.
      else
```

```
         drdx=(difx/dabs(difx))/dsi(1)
      end if
      if (dabs(dify).le.ep) then
         drdy=0.
      else
         drdy=(dify/dabs(dify))/dsi(2)
      end if
      rx=abs(xg(1)-xcent(1))
      ry=abs(xg(2)-xcent(2))
      rx=rx/dsi(1)
      ry=ry/dsi(2)
      if (rx.gt.0.5) then
         wx=(4./3.)-4.*rx+4*rx*rx-(4./3.)*rx**3
         dwx=(-4.+8.*rx-4.*rx*rx)*drdx
      else
         wx=(2./3.)-4.*rx*rx+4.*rx**3
         dwx=(-8.*rx+12.*rx*rx)*drdx
      endif
      if (ry.gt.0.5) then
         wy=(4./3.)-4.*ry+4*ry*ry-(4./3.)*ry**3
         dwy=(-4.+8.*ry-4.*ry*ry)*drdy
      else
         wy=(2./3.)-4.*ry*ry+4.*ry**3
         dwy=(-8.*ry+12.*ry*ry)*drdy
      endif
      if(rx.gt.1.) wx=0.
      if(ry.gt.1.) wy=0.
      w=wx*wy
      wxx=wy*dwx
      wyy=wx*dwy
  RETURN
  END
```

Program 5.7. Source code of Subroutine Integration_BCQuQi

```
  SUBROUTINE Integration_BCQuQi(nx,ng,nod,xc,x,numnode,nquado,dmat,&
                    xm,xspace,yspace,ak,mk,alfs,ds)
!-------------------------------------------------------------------------
! The subroutine is to compute the integrations on the internal
!        and the essential sub-boundaries;
! input—nx,ng,nod,xc,x,numnode,nquado,dmat,xm,xspace,mk,alfs,ds
! Input & output—ak
!-------------------------------------------------------------------------

      implicit real*8 (a-h,o-z)
      common/para/xlength,ylength,p,young,anu,aimo
      common/rpim/ALFC,DC,Q,nRBF
      common/basis/mbasis
      dimension x(2,numnode),nv(50),xc(2,4),gauss(2,20)
      dimension xcent(2),dsi(2),ak(mk,mk),ek(2,2)
      dimension gs(4,100),gpos(2),xm(4),dmat(3,3)
      dimension phi(10, numnode),ds(2,numnode)
      dimension bb(3,2),bn(2,3),bnd(2,3),ebb(2,2),ss(2,2)
       call GaussCoefficient(nquado,gauss)
      ik1=2*nod-1
      ik2=2*nod
      xcent(1)=x(1,nod)
      xcent(2)=x(2,nod)
      do i=1,2
         do j=1,2
            eK(i,j)=0.
         enddo
      enddo
      dsi(1)=xspace
      dsi(2)=yspace
```

```
! ****************INTEGRATION FOR LEFT B.C. Qu

      do i=1,2
         do j=1,3
            bn(i,j)=0.
            bb(j,i)=0.
            bnd(i,j)=0.
         enddo
      enddo
      ax=0.5*(xc(1,4)-xc(1,1))
      ay=0.5*(xc(2,4)-xc(2,1))
      bx=0.5*(xc(1,4)+xc(1,1))
      by=0.5*(xc(2,4)+xc(2,1))
      do il=1,nquado
         gpos(1)=ax*gauss(1,il)+bx
         gpos(2)=ay*gauss(1,il)+by
         weight=gauss(2,il)
         ajac=0.5*sqrt((xc(1,4)-xc(1,1))**2+(xc(2,4)-xc(2,1))**2)
         call TestFunc(dsi,xcent,gpos,w,wx,wy)
         ndex=0
         call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
         call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,alfc,dc,&
                                q,nRBF, mbasis)
         do ine=1,ndex
            n1=2*nv(ine)-1
            n2=2*nv(ine)
            do i=1,2
               do j=1,3
                  bn(i,j)=0.
                  bb(j,i)=0.
                  bnd(i,j)=0.
               enddo
            enddo
            do i=1,2
               do j=1,2
                  eK(i,j)=0.
                  ss(i,j)=0.
               enddo
            enddo
            bb(1,1)=phi(2,ine)
            bb(2,2)=phi(3,ine)
            bb(3,1)=phi(3,ine)
            bb(3,2)=phi(2,ine)
            bn(1,3)=1.
            bn(2,2)=1.

            IF(XC(2,1).lt.xm(3)) then
               ss(1,1)=1.
               ss(2,2)=1.
            endif
            call DOBMAX(bn,2,3,2,dmat,3,3,bnd,2)
            call dobmax(bnd,2,3,2,bb,2,3,ebb,2)
            call dobmax(ebb,2,2,2,ss,2,2,ek,2)
            ak(ik1,n1)=ak(ik1,n1)-w*weight*ajac*ek(1,1)
            ak(ik1,n2)=ak(ik1,n2)-W*weight*ajac*ek(1,2)
            ak(ik2,n1)=ak(ik2,n1)-W*weight*ajac*ek(2,1)
            ak(ik2,n2)=ak(ik2,n2)-W*weight*ajac*ek(2,2)
         enddo
      enddo
! ****************INTEGRATION FOR DOWN B.C. Qu
      do i=1,2
         do j=1,3
            bn(i,j)=0.
            bb(j,i)=0.
            bnd(i,j)=0.
         enddo
      enddo
      ax=0.5*(xc(1,2)-xc(1,3))
      ay=0.5*(xc(2,2)-xc(2,3))
      bx=0.5*(xc(1,2)+xc(1,3))
```

```
      by=0.5*(xc(2,2)+xc(2,3))
      do il=1,nquado
         gpos(1)=ax*gauss(1,il)+bx
         gpos(2)=ay*gauss(1,il)+by
         weight=gauss(2,il)
         ajac=0.5*sqrt((xc(1,2)-xc(1,3))**2+(xc(2,2)-xc(2,3))**2)
         call TestFunc(dsi,xcent,gpos,w,wx,wy)
         ndex=0
         call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
         call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,alfc,dc,&
                                q,nRBF, mbasis)
         do ine=1,ndex
            n1=2*nv(ine)-1
            n2=2*nv(ine)
            do i=1,2
            do j=1,3
              bn(i,j)=0.
              bb(j,i)=0.
              bnd(i,j)=0.
            enddo
            enddo
            do i=1,2
            do j=1,2
              eK(i,j)=0.
              ss(i,j)=0.
            enddo
            enddo
            bb(1,1)=phi(2,ine)
            bb(2,2)=phi(3,ine)
            bb(3,1)=phi(3,ine)
            bb(3,2)=phi(2,ine)
            bn(1,3)=-1.
            bn(2,2)=-1.
            IF(XC(2,2).gt.xm(4)) then
              ss(1,1)=1.
              ss(2,2)=1.
            endif
            call DOBMAX(bn,2,3,2,dmat,3,3,bnd,2)
            call dobmax(bnd,2,3,2,bb,2,3,ebb,2)
            call dobmax(ebb,2,2,2,ss,2,2,ek,2)
            ak(ik1,n1)=ak(ik1,n1)-w*weight*ajac*ek(1,1)
            ak(ik1,n2)=ak(ik1,n2)-W*weight*ajac*ek(1,2)
            ak(ik2,n1)=ak(ik2,n1)-W*weight*ajac*ek(2,1)
            ak(ik2,n2)=ak(ik2,n2)-W*weight*ajac*ek(2,2)
         enddo
      enddo

! ****************INTEGRATION FOR RIGHT B.C. Qu
      do i=1,2
      do j=1,3
         bn(i,j)=0.
         bb(j,i)=0.
         bnd(i,j)=0.
      enddo
      enddo
      ax=0.5*(xc(1,4)-xc(1,3))
      ay=0.5*(xc(2,4)-xc(2,3))
      bx=0.5*(xc(1,4)+xc(1,3))
      by=0.5*(xc(2,4)+xc(2,3))
      do il=1,nquado
         gpos(1)=ax*gauss(1,il)+bx
         gpos(2)=ay*gauss(1,il)+by
         weight=gauss(2,il)
         ajac=0.5*sqrt((xc(1,4)-xc(1,3))**2+(xc(2,4)-xc(2,3))**2)
         call TestFunc(dsi,xcent,gpos,w,wx,wy)
         ndex=0
         call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
         call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,alfc,dc,&
                                q,nRBF, mbasis)
```

```
         do ine=1,ndex
            n1=2*nv(ine)-1
            n2=2*nv(ine)
            do i=1,2
            do j=1,3
            bn(i,j)=0.
            bb(j,i)=0.
            bnd(i,j)=0.
            enddo
            enddo
            do i=1,2
            do j=1,2
            eK(i,j)=0.
            ss(i,j)=0
            enddo
            enddo
            bb(1,1)=phi(2,ine)
            bb(2,2)=phi(3,ine)
            bb(3,1)=phi(3,ine)
            bb(3,2)=phi(2,ine)
            bn(1,1)=1.
            bn(2,3)=1.
            IF(XC(1,4).lt.xm(2)) then
              ss(1,1)=1.
              ss(2,2)=1.
            endif
            call DOBMAX(bn,2,3,2,dmat,3,3,bnd,2)
            call dobmax(bnd,2,3,2,bb,2,3,ebb,2)
            call dobmax(ebb,2,2,2,ss,2,2,ek,2)
            ak(ik1,n1)=ak(ik1,n1)-w*weight*ajac*ek(1,1)
            ak(ik1,n2)=ak(ik1,n2)-W*weight*ajac*ek(1,2)
            ak(ik2,n1)=ak(ik2,n1)-W*weight*ajac*ek(2,1)
            ak(ik2,n2)=ak(ik2,n2)-W*weight*ajac*ek(2,2)
         enddo
      enddo

! ****************INTEGRATION FOR LEFT B.C. Qu
      do i=1,2
      do j=1,3
          bn(i,j)=0.
          bb(j,i)=0.
          bnd(i,j)=0.
      enddo
      enddo
      ax=0.5*(xc(1,2)-xc(1,1))
      ay=0.5*(xc(2,2)-xc(2,1))
      bx=0.5*(xc(1,2)+xc(1,1))
      by=0.5*(xc(2,2)+xc(2,1))
      do il=1,nquado
         gpos(1)=ax*gauss(1,il)+bx
         gpos(2)=ay*gauss(1,il)+by
         weight=gauss(2,il)
         ajac=0.5*sqrt((xc(1,2)-xc(1,1))**2+(xc(2,2)-xc(2,1))**2)
         call TestFunc(dsi,xcent,gpos,w,wx,wy)
         ndex=0
         call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
         call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,alfc,dc,&
                                q,nRBF, mbasis)
         do ine=1,ndex
         n1=2*nv(ine)-1
         n2=2*nv(ine)
         do i=1,2
         do j=1,3
         bn(i,j)=0.
         bb(j,i)=0.
         bnd(i,j)=0.
         enddo
         enddo
         do i=1,2
         do j=1,2
```

```
         eK(i,j)=0.
         ss(i,j)=0.
         enddo
         enddo
         bb(1,1)=phi(2,ine)
         bb(2,2)=phi(3,ine)
         bb(3,1)=phi(3,ine)
         bb(3,2)=phi(2,ine)
         bn(1,1)=-1.
         bn(2,3)=-1.
         ss(1,1)=1.
         ss(2,2)=1.
         call DOBMAX(bn,2,3,2,dmat,3,3,bnd,2)
         call dobmax(bnd,2,3,2,bb,2,3,ebb,2)
         call dobmax(ebb,2,2,2,ss,2,2,ek,2)
         ak(ik1,n1)=ak(ik1,n1)-w*weight*ajac*ek(1,1)
         ak(ik1,n2)=ak(ik1,n2)-W*weight*ajac*ek(1,2)
         ak(ik2,n1)=ak(ik2,n1)-W*weight*ajac*ek(2,1)
         ak(ik2,n2)=ak(ik2,n2)-W*weight*ajac*ek(2,2)
        enddo
      enddo
  RETURN
  END
```

Program 5.8. Source code of Subroutine Integration_BCQt

```
   SUBROUTINE Integration_BCQt(nx,ng,xc,f,x,numnode,nquado, &
             xm,xspace,yspace,xcent)
!-----------------------------------------------------------------------
! The subroutine is to compute the integrations on the natural sub-boundary;
! input— nx,ng,xc, x,numnode,nquado,xm,xspace,yspace,xcent
! Input & output— f;
!-----------------------------------------------------------------------

       implicit real*8 (a-h,o-z)
       common/para/xlength,ylength,p,young,anu,aimo
       common/rpim/ALFC,DC,Q,nRBF
       common/basis/mbasis
       dimension x(2,numnode),nv(50),f(2),xc(2,4),gauss(2,20)
       dimension xcent(2),dsi(2),fbcl(2,4)
       dimension gs(4,100),gpos(2),xm(4)
       dimension phi(10, numnode)
       call GaussCoefficient(nquado,gauss)
       do j=1,2
          f(j)=0.
       enddo
       dsi(1)=xspace
       dsi(2)=yspace
! ***************** Set global force BC for a rectangular domain
       do j=1,4
          fbcl(1,j)=0.
          fbcl(2,j)=0.
       enddo
       fbcl(2,2)=1.0  ! force in y direction at right end is not zero

! ***************** INTEGRATION FOR UP B.C.
       IF(XC(2,1).GE.xm(3)) then
          txx=fbcl(1,3)
          tyy=fbcl(2,3)
          ax=0.5*(xc(1,4)-xc(1,1))
          ay=0.5*(xc(2,4)-xc(2,1))
          bx=0.5*(xc(1,4)+xc(1,1))
          by=0.5*(xc(2,4)+xc(2,1))
          do il=1,nquado
             gpos(1)=ax*gauss(1,il)+bx
             gpos(2)=ay*gauss(1,il)+by
             weight=gauss(2,il)
```

```
            ajac=0.5*sqrt((xc(1,4)-xc(1,1))**2+(xc(2,4)-xc(2,1))**2)
            call TestFunc(dsi,xcent,gpos,w,wx,wy)
            f(1)=f(1)+w*weight*ajac*txx
            f(2)=f(2)-w*weight*ajac*tyy
         enddo
      endif

! ***************** INTEGRATION FOR DOWN B.C.
      IF(XC(2,2).lE.xm(4)) then
         txx=fbcl(1,4)
         tyy=fbcl(2,4)
         ax=0.5*(xc(1,2)-xc(1,3))
         ay=0.5*(xc(2,2)-xc(2,3))
         bx=0.5*(xc(1,2)+xc(1,3))
         by=0.5*(xc(2,2)+xc(2,3))
         do il=1,nquado
            gpos(1)=ax*gauss(1,il)+bx
            gpos(2)=ay*gauss(1,il)+by
            weight=gauss(2,il)
            ajac=0.5*sqrt((xc(1,2)-xc(1,3))**2+(xc(2,2)-xc(2,3))**2)
            call TestFunc(dsi,xcent,gpos,w,wx,wy)
            f(1)=f(1)+w*weight*ajac*txx
            f(2)=f(2)-w*weight*ajac*tyy
         enddo
      endif

!  ***************** INTEGRATION FOR RIGHT B.C.
      IF(XC(1,4).GE.xm(2)) then
         txx=fbcl(1,2)
         tyy=fbcl(2,2)
         ax=0.5*(xc(1,4)-xc(1,3))
         ay=0.5*(xc(2,4)-xc(2,3))
         bx=0.5*(xc(1,4)+xc(1,3))
         by=0.5*(xc(2,4)+xc(2,3))
         do il=1,nquado
            gpos(1)=ax*gauss(1,il)+bx
            gpos(2)=ay*gauss(1,il)+by
            weight=gauss(2,il)
            ajac=0.5*sqrt((xc(1,4)-xc(1,3))**2+(xc(2,4)-xc(2,3))**2)
            call TestFunc(dsi,xcent,gpos,w,wx,wy)
            aimo=(1./12.)*ylength**3
            ty=-(-1000./(2.*aimo))*(ylength*ylength/4.-gpos(2)*gpos(2))
            f(1)=f(1)+w*weight*ajac*0.*txx
            f(2)=f(2)-w*weight*ajac*ty*tyy ! Exact force B.C.
         enddo
      endif
  RETURN
  END
```

Program 5.9. Source code of Subroutine EssentialBC

```
  SUBROUTINE EssentialBC(x,numnode,ak,fk,mk,ds,alfs,npEBCnum,npEBC,pEBC)
!----------------------------------------------------------------------------
! This subroutine to cenforce point essential bc's using the direct method;
! input--numnode: total number of field nodes;
!        npEBCnum: number of e. b.c points
!        alfs: coefficent of support support
!        x(nx,numnode): coordinates of all field nodes;
! input and output-- ak[]: stifness matrix;
!                    fk{}:force vector.
!----------------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)
      common/para/xlength,ylength,p,young,anu,aimo
      common/rpim/ALFC,DC,Q,nRBF
      common /basis/mbasis
      dimension x(2,numnode),ds(2,numnode)
```

```
      dimension npEBC(3,100),pEBC(2,100)
      dimension ak(mk,mk),fk(2*numnode)
      dimension f(2*numnode),phi(10,numnode),nv(numnode),gpos(2)
      nx=2
      eps=2.2204e-16
      do 135 ib=1,npEBCnum
         in=npEBC(1,ib)
         ll=in*2-1
         lr=in*2
         if(npEBC(2,ib).eq.1) f(ll)=pEBC(1,ib)
         if(npEBC(3,ib).eq.1) f(lr)=pEBC(2,ib)
 135   continue

      do 231 ib=1,npEBCnum
         in=npEBC(1,ib)
         gpos(1)=x(1,in)
         gpos(2)=x(2,in)
         ll=in*2-1
         lr=in*2
         ndex=0
         call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
         Call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,alfc,dc,&
                                q,nRBF,mbasis)

         if(npEBC(2,ib).eq.1) then
            do ii=1,2*numnode
               ak(ll,ii)=0.
            enddo
            do ii=1,ndex
               mm=nv(ii)
               ak(ll,mm*2-1)=phi(1,ii)
               ak(ll,mm*2)=0.
            enddo
         endif

         if(npEBC(3,ib).eq.1) then
            do ii=1,2*numnode
               ak(lr,ii)=0.
            enddo
            do ii=1,ndex
               mm=nv(ii)
               ak(lr,mm*2)=phi(1,ii)
               ak(lr,mm*2-1)=0.
            enddo
         endif
 231    continue

        do 165 ib=1,npEBCnum
           in=npEBC(1,ib)
           ll=in*2-1
           lr=in*2
           if(npEBC(2,ib).eq.1) fk(ll)=f(ll)
           if(npEBC(3,ib).eq.1) fk(lr)=f(lr)
  165   continue
  RETURN
  END
```

Program 5.10. Source code of Subroutine GetDisplacement

```
    SUBROUTINE GetDisplacement(x,ds,u2,disp,alfs,nx,numnode)
!----------------------------------------------------------------------
! The subroutine is to compute the final nodal displacements
! input— x,ds,u2, alfs,nx,numnode
! Output— disp;
!----------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
```

```
       common/rpim/ALFC,DC,Q,nRBF
       common /basis/mbasis
       dimension x(nx,numnode),ds(nx,numnode),gpos(nx),u2(nx,numnode)
       dimension nv(numnode),phi(10,numnode),aa(nx,numnode),disp(2*numnode)
       do i=1,2*numnoden
          disp(i)=0.
       enddo
       ind=0
       do 50 id=1,numnode
          ind=ind+1
          gpos(1)= x(1,id)
          gpos(2)=x(2,id)
          ndex=0
          call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
          do kph=1,ndex
             do ii=1,10
                phi(ii,kph)=0.
             enddo
          enddo
          call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,alfc,dc,&
                                 q,nRBF, mbasis)
          nc1=2*ind-1
          nc2=2*ind
          do kk=1,ndex
             m=nv(kk)
             disp(nc1)=disp(nc1)+phi(1,kk)*u2(1,m)
             disp(nc2)=disp(nc2)+phi(1,kk)*u2(2,m)
          enddo
 50    continue
  RETURN
  END
```

Program 5.11. Source code of Subroutine GetNodeStress

```
  SUBROUTINE GetNodeStress(x,ds,Dmat,u2,stress,alfs,nx,numnode)
!-----------------------------------------------------------------------
! The subroutine is to compute the nodal stress components.
! input— x,ds,Dmat,u2,alfs,nx,numnode;
! Output— stress;
!-----------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       common/para/xlength,ylength,p,young,anu,aimo
       common/rpim/ALFC,DC,Q,nRBF
       common/basis/mbasis
       dimension ds(nx,numnode),gpos(nx),x(nx,numnode)
       dimension nv(numnode),phi(10,numnode),aa(nx,numnode),ne(2*numnode)
       dimension stress(3,numnode),Bmat(3,2*numnode)
       dimension Dmat(3,3),u2(nx,numnode),u(2*numnode)

       do iu=1,numnode
          u(2*iu-1)=u2(1,iu)
          u(2*iu)=u2(2,iu)
       enddo
       do i=1,3
          do j=1,numnode
             stress(i,j)=0.
          enddo
       enddo
       ind=0
       do 200 is=1,numnode
          ind=ind+1
          gpos(1)=x(1,is)
          gpos(2)=x(2,is)
          ndex=0
          call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
          do kph=1,ndex
```

```
            do ii=1,10
               phi(ii,kph)=0.
            enddo
         enddo
         Call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,alfc,dc,&
                                q,nRBF, mbasis)
         nb=2*ndex
         do in=1,nb
            ne(in)=0
         enddo
         do ine=1,ndex
            n1=2*ine-1
            n2=2*ine
            ne(n1)=2*nv(ine)-1
            ne(n2)=2*nv(ine)
         enddo
         do ib=1,3
            do jb=1,nb
               Bmat(ib,jb)=0.
            enddo
         enddo
         do inn=1,ndex
            j=2*inn-1
            k=2*inn
            m1=ndex+inn
            m2=2*ndex+inn
            Bmat(1,j)=phi(2,inn)
            Bmat(1,k)=0.
            Bmat(2,j)=0.
            Bmat(2,k)=phi(3,inn)
            Bmat(3,j)=phi(3,inn)
            Bmat(3,k)=phi(2,inn)
         enddo
         do ii=1,3
            do kk=1,3
               do mm=1,nb
                  mn=ne(mm)
                  stress(ii,ind)=stress(ii,ind)+Dmat(ii,kk)*Bmat(kk,mm)*u(mn)
               enddo
            enddo
         enddo
200   continue
 RETURN
 END
```

Program 5.12. Source code of Subroutine Output

```
 SUBROUTINE Output(x,numnode,u2,u22,str)
!-----------------------------------------------------------------------
! The subroutine is to output resultscompute the final nodal displacements
! Output— all;
!-----------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)
      common/para/xlength,ylength,p,young,anu,aimo
      dimension x(2,numnode),u2(2,numnode),str(3,numnode),u22(2,numnode)
      write(2,*)'**************<DISPLACEMENT OF NODES>**************'
      do i=1,numnode
         nn=2*i-1
         kk=2*i
         write(2,10)i,x(1,i),x(2,i),u22(1,i),u22(2,i)
      enddo
      write(2,*)'**************<STRESSES OF NODES>**************8'
      do i=1,numnode
         nn=2*i-1
         kk=2*i
```

```
          write(2,20)i,x(1,i),x(2,i),str(1,i),str(2,i),str(3,i)
       enddo
 10   format(1x,i3,1x,2f10.5,3E15.5)
 20   format(1x,i3,1x,2f10.5,1x,3E15.5)
  RETURN
  END
```

Program 5.13. Source code of Subroutine TotalGaussPoints

```
  SUBROUTINE TotalGaussPoints(xc,conn,gauss,gs,nx,ng,numq,&
       numcell,k,numgauss)
!-----------------------------------------------------------------------
! The subroutine is to set up Gauss points,Jacobian and weights
!        for the global background cells;
! input— xc,conn,gauss,nx,ng,numq,numcell,k,numgauss
! Output— gs;
!-----------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       dimension xc(nx,numq),conn(ng,numcell),gauss(nx,k)
       dimension gs(ng,numgauss),psiJ(4),etaJ(4),xe(4),ye(4),aN(4)
       dimension aNJpsi(4),aNJeta(4)
       index=0
       psiJ(1)=-1.
       psiJ(2)=1.
       psiJ(3)=1.
       psiJ(4)=-1.
       etaJ(1)=-1.
       etaJ(2)=-1.
       etaJ(3)=1.
       etaJ(4)=1.
       l=k
       do 10 ie=1,numcell
!  determine nodes in each cell
          do j=1,4
             je=conn(j,ie)
             xe(j)=xc(1,je)
             ye(j)=xc(2,je)
          enddo
          do 30 i=1,l
             do 30 j=1,l
             index=index+1
             eta=gauss(1,i)
             psi=gauss(1,j)
!            write(2,*)'psi,eta',psi,eta
             do ik=1,ng
                aN(ik)=.25*(1.+psi*psiJ(ik))*(1.+eta*etaJ(ik))
                aNJpsi(ik)=.25*psiJ(ik)*(1.+eta*etaJ(ik))
                aNJeta(ik)=.25*etaJ(ik)*(1.+psi*psiJ(ik))
             enddo
             xpsi=0.
             ypsi=0.
             xeta=0.
             yeta=0.
             do jk=1,ng
                xpsi=xpsi+aNJpsi(jk)*xe(jk)
                ypsi=ypsi+aNJpsi(jk)*ye(jk)
                xeta=xeta+aNJeta(jk)*xe(jk)
                yeta=yeta+aNJeta(jk)*ye(jk)
             enddo
             ajcob=xpsi*yeta-xeta*ypsi
             xq=0.
             yq=0.
             do kk=1,ng
                xq=xq+aN(kk)*xe(kk)
                yq=yq+aN(kk)*ye(kk)
             enddo
             gs(1,index)=xq
```

```
            gs(2,index)=yq
            gs(3,index)=gauss(2,i)*gauss(2,j)
            gs(4,index)=ajcob
 30       continue
 10    continue
  RETURN
  END
```

Program 5.14. Source code of Subroutine GetEnergyError

```
  SUBROUTINE GetEnergyError(nx,ng,x,numnode,u2,dmat,ds,numgauss,gs,alfs)
!-----------------------------------------------------------------------
! The subroutine is to compute the global energy;
! input- all;
!-----------------------------------------------------------------------

       IMPLICIT REAL*8(A-H,O-Z)
       common/para/xlength,ylength,p,young,anu,aimo
       common/rpim/ALFC,DC,Q,nRBF
       common/basis/mbasis
       dimension x(2,numnode),u2(2,numnode),dmat(3,3),str(3,numgauss)
       dimension ph(10,numnode),gs(4,numgauss)
       dimension bx(3,2*numnode),dipl(2*numnode),db(3,2*numnode)
       dimension dbu(3),gpos(2),nv(numnode)
       dimension err(3),Dinv(3,3),der(3),stressex(3,numgauss)
       dimension ds(2,numnode),ddd(3)

       enorm=0.
       errext=0.
       do id=1,3
          do jd=1,3
             Dinv(id,jd)=Dmat(id,jd)
          enddo
       enddo
       invd=3
       call getinvasy(INVD,INVD,Dinv,EP)
       do 10 nod=1,numgauss
          xn=gs(1,nod)
          yn=gs(2,nod)
          weight=gs(3,nod)
          ajac=gs(4,nod)
          gpos(1)=xn
          gpos(2)=yn
          ndex=0
          call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
          do i=1,ndex
             nn=nv(i)
             n1=2*i-1
             n2=i*2
             dipl(n1)=u2(1,nn)
             dipl(n2)=u2(2,nn)
          enddo
          do ii=1,10
             do jj=1,ndex
                ph(ii,jj)=0.
             enddo
          enddo
          call RPIM_ShapeFunc_2D(gpos,x,nv,ph,nx,numnode,ndex,alfc,&
                                 dc,q,nRBF, mbasis)   ! RPIM Shape function
!         call MLS_ShapeFunc_2D(gpos,x,nv,ds,ph,nx,numnode,ndex,mbasis) ! MLPG
          do i=1,2*ndex
             bx(1,i)=0.
             bx(2,i)=0.
             bx(3,i)=0.
          enddo
          do i=1,ndex
             n1=i*2-1
```

```
            n2=2*i
            bx(1,n1)=ph(2,i)
            bx(2,n2)=ph(3,i)
            bx(3,n1)=ph(3,i)
            bx(3,n2)=ph(2,i)
         enddo
         m=2*ndex
         nn=2*numnode
         call DOBMAX(dmat,3,3,3,bx,m,3,db,3)
         call DOBMAX(db,3,m,3,dipl,1,nn,dbu,3)
         str(1,nod)=dbu(1)
         str(2,nod)=dbu(2)
         str(3,nod)=dbu(3)
!*******  Exact stress for beam problem
         stressex(1,nod)=(1./aimo)*p*(xlength-gpos(1))*gpos(2)
         stressex(2,nod)=0.
         stressex(3,nod)=-0.5*(p/aimo)*(0.25*ylength*ylength-gpos(2)*gpos(2))

         do ier=1,3
            err(ier)=str(ier,nod)-stressex(ier,nod)
         enddo
         do jer=1,3
            der(jer)=0.
            ddd(jer)=0.
            do ker=1,3
               der(jer)=der(jer)+Dinv(jer,ker)*err(ker)
               ddd(jer)=ddd(jer)+Dinv(jer,ker)*stressex(ker,nod)
            enddo
         enddo
         err2=0.
         eex=0.
         do mer=1,3
            err2=err2+weight*ajac*(0.5*der(mer)*err(mer))
            eex=eex+weight*ajac*(0.5*ddd(mer)*stressex(mer,nod))
         enddo
         enorm=enorm+err2
         errext=errext+eex
 10    continue
       enorm=dsqrt(enorm)
       errext=sqrt(errext)
       write(2,*)'**************<Global energy error>**************'
       write(2,180)enorm
180    format(1x,'The global energu error:',e20.8)
  RETURN
  END
```

Program 5.15. Source code of Subroutine Dobmax

```
  SUBROUTINE DOBMAX(A,N,M1,M3,B,M2,M4,C,M5)
!    This subroutine is used to calculate A[N][M1]*B[M1][M2]=C[N][M2].
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION A(M3,M1),B(M4,M2),C(M5,M2)
       DO I=1,N
          DO J=1,M2
             C(I,J)=0.0
          ENDDO
       ENDDO
       DO I=1,N
          DO J=1,M2
             DO K=1,M1
                C(I,J)=C(I,J)+A(I,K)*B(K,J)
             ENDDO
          ENDDO
       ENDDO
  RETURN
  END
```

Chapter 6

MESHFREE COLLOCATION METHODS

6.1 INTRODUCTION

MFree collocation methods (or MFree strong-form methods) have a long history. To approximate strong-form of PDEs using MFree methods, the PDE is usually discretized at nodes by some forms of collocation. There are various MFree strong-form methods, e.g., the vortex method (Chorin, 1973; Bernard, 1995), the finite difference method(FDM) with irregular grids or the so-called general FDM (GFDM) (Girault,1974; Pavlin and Perrone,1975; Snell et al,1981; Liszka and Orkisz,1977; 1980; Krok and Orkisz), the finite point method (FPM) (Oñate et al., 1996,1998, 2001), the *hp*-meshless cloud method (Liszka et al., 1996), the meshfree collocation method (Kansa, 1990; Wu, 1992; Xu et al, 1999; Zhang et al.,2000; Liu X et al.,2002, 2003a-d), etc.

MFree strong-form methods have following advantages:

- The procedure for discretizing the governing equations is straightforward, and the algorithms for implementing the discretized equation are simple. The discretized equations can be obtained directly from the strong-forms of PDEs governing the problem.
- They are, in general, computationally efficient. The PDEs are discretized directly without using weak-forms, and hence no numerical integration is required.
- They are truly meshless: no mesh is used for both field variable approximations and numerical integrations.

Owing to these advantages, MFree strong-form methods have been studied and used in computational mechanics with some success, especially in fluid mechanics. There are, however, the following two major issues that have prevented the use of collocation methods with irregular grids or nodes.

One such an issue is the singularity of the moment matrix arising in the process of function approximation. The use of weighted least square method (Krok and Orkisz, 1989) has provided an effective way to solve this problem. The matrix triangularization algorithm (MTA) proposed by GR Liu and Gu (2001d, 2003a) is a novel procedure to overcome the singularity problem in the point interpolation method (PIM) that uses polynomial basis. The PIM shape functions so created possess the delta function property (see, e.g., Section 3.2). Kansa (1990) has also solved this kind of singularity problem using radial basis functions (RBFs). The Kansa method is a global collocation method that uses all the grids in the problem domain, which leads to a fully populated system matrix. Since the RBFs are used, the moment matrix is, in general, not singular. A more stable symmetric formulation has also been proposed by Wu (1992). In addition, RBF is also used for creating RPIM shape functions using *local* nodes for MFree methods based on the global weak-form (GR Liu and Gu, 2001c; Wang et al., 2000; 2002a, Section 4.2), local weak-form (GR Liu and Gu, 2000b, 2001b, c,e, 2002a; GR Liu and Yan et al., 2000, 2002; Xiao and McCharthy, 2003a,b,c; Section 5.2) and strong-form (Liu X et al., 2002, 2003a~e, Section 6.3).

Another key issue that has been preventing the idea of collocation methods with irregular grids or nodes from practical applications is the presence of *derivative boundary conditions* (DBCs). It is well-known that the boundary conditions (BCs) are crucial in a collocation method. We emphases specifically that it is the DBCs (not Dirichlet BCs) that are the true culprit responsible for the poor accuracy and instability problems in the MFree strong-form methods using arbitrary nodes. Therefore, we will discuss this issue at great length with many examples of 1D and 2D problems in the next section.

6.2 TECHNIQUES FOR HANDLING DERIVATIVE BOUNDARY CONDITIONS

In using an MFree strong-form method to solve a problem governed by a set of partial differential equations (PDEs), the problem is represented by a set of nodes that are *arbitrarily* distributed in the problem domain and the

boundaries. Strong-form methods can produce accurate results for PDEs, when the boundary conditions are all of Dirichlet type[†]. If there is any derivative boundary condition, the accuracy of the solution deteriorates drastically, and the solution can be unstable: small changes in the setup of the problem can lead to a large change in the solution. The discretized system equation behaves, like an ill-posed problem in which errors introduced into the system are magnified in the output.

For convenience, we denote the boundary with derivative boundary conditions (BDCs) as the *derivative boundary*, and a node on the derivation boundary as a "*DB-node*".

A number of strategies can be used to impose the DBCs in the strong-form methods. Six of them are listed below.

1) The direct collocation (DC) method: The DBCs are discretized by simple collocation to obtain a set of separate equations that are different from the governing system equations. In other words, there is no special treatment for DBCs.

2) The method using fictitious points (FP): along the derivative boundaries, a set of fictitious points is added outside the problem domain along the derivative boundary. In this case, two sets of equations are established at each DB-node: one for the DBC, and the other for the governing equation.

3) The Hermite-type collocation (HC) method: this uses additional derivative variables for the DB-nodes to enforce the DBCs. This treatment has been used by many researchers, such as Zhang et al. (2000), etc.

4) The method using regular grids (RG): in this method, one or several layers of regularly distributed nodes are used in the problem domain along the derivative boundary. The standard differential scheme used in FDM is adopted for these regular nodes. The DBCs can then be implemented using the same procedure as that in the standard FDM.

5) The use of dense nodes (DN) in the derivative boundaries (see, e.g., Liszka et al., 1996).

6) The MFree weak-strong (MWS) form method: being a combination of the local weak-form and the strong-form, the DBCs can be naturally satisfied through the local weak-form. The MWS method is proposed by GR Liu and Gu (2002d, 2003b). It can efficiently and

[†] We assume of course that the problem is well-posed, the moment matrix is not singular or badly conditioned, and a reasonable collocation scheme is used.

completely solve the problem of the enforcement of DBCs in the strong-form methods, and it will be detailed in Chapter 7.

There are also other means to stabilize the solution of meshfree collocation methods, such as adding in higher order differential terms in strong form equations for stabilization (Oñate et al., 1998, 2001). In the following sections, MFree strong-form methods with the first five types of treatments for DBCs will be used to examine in detail for one-dimensional (1D) and two-dimensional (2D) problems.

Note that the source code used in this chapter is not provided because 1) it is very simple and straightforward; 2) Chapter 7 contains the same routines for strong-form methods.

6.3 POLYNOMIAL POINT COLLOCATION METHOD FOR 1D PROBLEMS

In this section, we use simple 1D problems to illustrate the collocation procedure for establishing the discretized system equations together with five different ways to deal with the DBCs.

For 1D problems, the polynomial PIM shape functions work best; we will use these, and call the procedure as polynomial point collocation method (PPCM). Other types of shape functions discussed in Chapter 3 are of course applicable to 1D problems, and some of them will be used later for 2D problems.

6.3.1 Collocation equations for 1D system equations

6.3.1.1 Problem description

Consider problems governed by the following general second-order ordinary differential equation (ODE) in 1D domain, Ω.

$$A_2(x)\frac{d^2u}{dx^2} + A_1(x)\frac{du}{dx} + A_0(x)u + q_A(x) = 0 \tag{6.1}$$

where u is the unknown scalar field function, the coefficients A_0, A_1 and A_2 are given and may depend upon x, and q_A is a given source term that can be also a function of x. There are two-types of boundary conditions:

- DBC:

$$B_1(x)\frac{du(x)}{dx} + B_0(x)u(x) + q_B(x) = 0 \tag{6.2}$$

where x is a point on the derivative boundary Γ_{DB}, B_0 and B_1 are given functions of x, and q_B is a given source term on Γ_{DB}.

- Dirichlet boundary condition:

$$u(x) - \overline{u} = 0 \tag{6.3}$$

where x is a point on the Dirichlet boundary denoted by Γ_u, and $\overline{u}$ is the specified value for the field function.

6.3.1.2 Function approximation using MFree shape functions

Assume that there are N_d internal (domain) nodes and $N_b = N_{\mathrm{DB}} + N_u$ boundary nodes, where N_{DB} is the number of DB-nodes and N_u is the number of nodes on the Dirichlet boundary. The collocation points could be different (in term of both locations and numbers) from the field nodes, but we always take them to be the same in this book.

For convenience, consider a 1D domain shown in Figure 6.1, and x_1 is on the Dirichlet boundary and x_L is on the derivative boundary. Therefore, N_{DB}=1 and N_u=1. The problem domain is represented by N field nodes numbered sequentially with $x_N = x_L$. Hence, there are $N-2$ internal nodes.

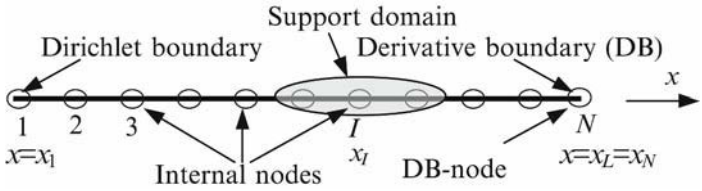


Figure 6.1. Nodal distribution used in a 1D problem domain.

Using the MFree shape functions introduced in Chapter 3, we have the following formulae for approximating the unknown function and their derivatives at the collocation node at x_I.

$$u_I^h = u^h(x_I) = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{u}_s \tag{6.4}$$

$$\frac{\partial u_I^h}{\partial x} = \frac{\partial \boldsymbol{\Phi}^{\mathrm{T}}}{\partial x}\mathbf{u}_s \tag{6.5}$$

$$\frac{\partial^2 u_I^h}{\partial x^2} = \frac{\partial^2 \mathbf{\Phi}^{\mathrm{T}}}{\partial x^2}\mathbf{u}_s \tag{6.6}$$

where $\mathbf{\Phi}$ is the vector of shape functions, and $\mathbf{u}_s$ is the vector that collects nodal values of the unknown function, i.e.,

$$\mathbf{\Phi}^{\mathrm{T}} = \{\phi_1 \quad \phi_2 \quad \cdots \quad \phi_n\} \tag{6.7}$$

$$\mathbf{u}_s^{\mathrm{T}} = \{u_1 \quad u_2 \quad \cdots \quad u_n\} \tag{6.8}$$

in which n is the number of nodes used in the local support domain of x_I where the shape functions are created.

6.3.1.3 System equation discretization

For an internal node at x_I, Equation (6.4) gives the discretized governing Equation (6.1) can be obtained by simple collocations at x_I:

$$\underbrace{\left(A_2(x_I)\frac{d^2\mathbf{\Phi}^{\mathrm{T}}}{dx^2} + A_1(x_I)\frac{d\mathbf{\Phi}^{\mathrm{T}}}{dx} + A_0(x_I)\right)}_{\mathbf{K}_I}\mathbf{u}_s = \underbrace{-q_A(x_I)}_{f_I} \tag{6.9}$$

or in the matrix form

$$\mathbf{K}_I\mathbf{u}_s = f_I \tag{6.10}$$

where $\mathbf{K}_I$ is the nodal matrix for the collocation node at x_I, which can be written in detail as

$$\begin{aligned}\mathbf{K}_I &= A_2(x_I)\frac{d^2\mathbf{\Phi}^{\mathrm{T}}}{dx^2} + A_1(x_I)\frac{d\mathbf{\Phi}^{\mathrm{T}}}{dx} + A_0(x_I)\\ &= \left\{A_2(x_I)\frac{d^2\phi_1}{dx^2} + A_1(x_I)\frac{d\phi_1}{dx} + A_0(x_I) \quad \cdots \right.\\ &\qquad \left. A_2(x_I)\frac{d^2\phi_n}{dx^2} + A_1(x_I)\frac{d\phi_n}{dx} + A_0(x_I)\right\}\end{aligned} \tag{6.11}$$

The dimension of $\mathbf{K}_I$ is $(1\times n)$.

In Equation (6.9), f_I is given by

$$f_I = -q_A(x_I) \tag{6.12}$$

Note that Equation (6.10) is established for all the internal nodes, and for the DB-nodes if so required.

6.3.1.4 Discretization of Dirichlet boundary condition

For a node at x_1 that is on the Dirichlet boundary, the Dirichlet boundary condition Equation (6.3) can be re-written as

$$\underbrace{\mathbf{\Phi}^{\mathrm{T}}}_{\mathbf{K}_1}\mathbf{u}_s = \underbrace{\overline{u}_1}_{f_1} \tag{6.13}$$

where $\mathbf{K}_1$ is the nodal matrix for the collocation node at x_1 given by

$$\mathbf{K}_1 = \mathbf{\Phi}^{\mathrm{T}} = \begin{Bmatrix}\phi_1 & \phi_2 & \cdots & \phi_n\end{Bmatrix} \tag{6.14}$$

where ϕ_i is created using n nodes in the support domain of node 1. In Equation (6.13), f_1 is given by

$$f_1 = \overline{u}_1 \tag{6.15}$$

Note that if shape functions with the delta function property, such as PIM and RPIM shape functions are used, we should have

$$\mathbf{K}_1 = \mathbf{\Phi}^{\mathrm{T}} = \begin{Bmatrix}1 & 0 & \cdots & 0\end{Bmatrix}^{\mathrm{T}} \tag{6.16}$$

Without losing generalization, we use Equation (6.14).

6.3.1.5 Discretized system equation with only Dirichlet boundary conditions

When the problem has only Dirichlet boundaries at both ends of the 1D problem domain, we should also have the Dirichlet boundary condition equation for node N:

$$\underbrace{\mathbf{\Phi}^{\mathrm{T}}}_{\mathbf{K}_N}\mathbf{u}_s = \underbrace{\overline{u}_N}_{f_N} \tag{6.17}$$

where the nodal matrix for the collocation nodes at x_N is

$$\mathbf{K}_N = \mathbf{\Phi}^{\mathrm{T}} = \begin{Bmatrix}\phi_1 & \phi_2 & \cdots & \phi_n\end{Bmatrix} \tag{6.18}$$

where ϕ_i is created using n nodes in the support domain of node N. In Equation (6.17), f_N is given by

$$f_N = \overline{u}_N \tag{6.19}$$

Assembling Equations (6.9), (6.13) and (6.17) for the corresponding nodes, we can obtain the system equations as

$$\mathbf{K}_{(N\times N)}\mathbf{U}_{(N\times 1)} = \mathbf{F}_{(N\times 1)} \tag{6.20}$$

where the global system matrix $\mathbf{K}$ has the form of

$$\mathbf{K}_{(N\times N)}=\begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1(N-1)} & K_{1N} \\ K_{21} & K_{22} & \cdots & K_{2(N-1)} & K_{2N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ K_{(N-1)1} & K_{(N-1)2} & \cdots & K_{(N-1)(N-1)} & K_{(N-1)N} \\ K_{N1} & K_{N2} & \cdots & K_{N(N-1)} & K_{NN} \end{bmatrix}$$

Related to Dirichlet BC from Equation (6.14)

Related to the system PDE from Equation (6.11)

Related to Dirichlet BC from Equation (6.18)

(6.21)

and the global vector **F** consist of

$$\mathbf{F}^{\mathrm{DC}}_{(N\times 1)}=\begin{Bmatrix} \overline{u}_1 \\ -q_A(x_2) \\ \vdots \\ -q_A(x_{N-1}) \\ \overline{u}_N \end{Bmatrix}$$

Related to Dirichlet BC from Equation (6.15)

Related to the system PDE from Equation (6.12)

Related to Dirichlet BC from Equation (6.19)

(6.22)

In Equation (6.20), **U** is the vector that collects all nodal values, i.e.

$$\mathbf{U}_{(N\times 1)}=\begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \end{Bmatrix} \tag{6.23}$$

6.3.1.6 Discretized system equations with DBCs

In the following, we discuss how to construct collocation system equations for problems with both a Dirichlet BC at x_1 and a derivative boundary condition at the DB-node at x_N. The treatment for the governing equations and the Dirichlet boundary condition is the same as for those discussed in Sub-section 6.3.1.3 and 6.3.1.4. As discussed in Section 6.2, some special treatments are needed to impose the DBC. Treatments (1)~(4) listed in Section 6.2 are discussed here. Note that because the formulations of the treatment (5) listed in Section 6.2 are exactly the same as those the treatment (1) that is the direct collocation (DC) method, they are not presented here.

1) The direct collocation (DC) method

Substituting Equation (6.4) into the DBCs, Equations (6.2), we have

$$\underbrace{(B_1(x_N)\frac{d\boldsymbol{\Phi}^{\mathrm{T}}}{dx}+B_0(x_N)\boldsymbol{\Phi}^{\mathrm{T}})}_{\mathbf{K}_N}\mathbf{u}_s=\underbrace{-q_B(x_N)}_{f_N} \tag{6.24}$$

where $\mathbf{K}_N$ is the nodal matrix for the collocation node at x_N, which can be written as

$$\begin{aligned}\mathbf{K}_N &= B_1(x_N)\frac{d\boldsymbol{\Phi}^{\mathrm{T}}}{dx}+B_0(x_N)\boldsymbol{\Phi}^{\mathrm{T}}\\ &=\left\{B_1(x_N)\frac{d\phi_1}{dx}+B_0(x_N)\phi_1 \quad \cdots \quad B_1(x_N)\frac{d\phi_n}{dx}+B_0(x_N)\phi_n\right\}\end{aligned} \tag{6.25}$$

and f_N is given by

$$f_N=-q_B(x_N) \tag{6.26}$$

Assembling Equations (6.9), (6.13) and (6.24) for the corresponding nodes, we can obtain the discretized system equations as

$$\mathbf{K}^{\mathrm{DC}}_{(N\times N)}\mathbf{U}^{\mathrm{DC}}_{(N\times 1)}=\mathbf{F}^{\mathrm{DC}}_{(N\times 1)} \tag{6.27}$$

where the global system matrix $\mathbf{K}^{\mathrm{DC}}$ has the form of

$$\mathbf{K}^{\mathrm{DC}}_{(N\times N)}=\begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1(N-1)} & K_{1N}\\ K_{21} & K_{22} & \cdots & K_{2(N-1)} & K_{2N}\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ K_{(N-1)1} & K_{(N-1)2} & \cdots & K_{(N-1)(N-1)} & K_{(N-1)N}\\ K_{N1} & K_{N2} & \cdots & K_{N(N-1)} & K_{NN}\end{bmatrix}$$

Related to Dirichlet BC from Equation (6.14)

Related to the governing PDE from Equation (6.11)

Related to DBC from Equation (6.25)

(6.28)

and the global source vector $\mathbf{F}^{\mathrm{DC}}$ consists of

$$\mathbf{F}^{\mathrm{DC}}_{(N\times 1)}=\begin{Bmatrix}\overline{u}\\ -q_A(x_2)\\ \vdots\\ -q_A(x_{N-1})\\ -q_B(x_N)\end{Bmatrix}$$

Related to Dirichlet BC from Equation (6.15)

Related to the system PDE from Equation (6.12)

Related to DBC from Equation (6.26)

(6.29)

In Equation (6.27), $\mathbf{U}^{\mathrm{DC}}$ is the vector that collects all nodal values, i.e.

$$\mathbf{U}^{\mathrm{DC}}_{(N\times 1)} = \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \end{Bmatrix} \tag{6.30}$$

Solving Equation (6.27) gives the nodal values of u for all field nodes, provided $\mathbf{K}^{\mathrm{DC}}$ is not singular.

Note that the assembling is different from that in the conventional FEM and the MFree global weak-form methods, such as EFG and RPIM. In the FEM, EFG and RPIM, the element or nodal matrices are stamped symmetrically into the global matrix. In the collocation method, however, the nodal matrix is stacked together row-by-row to form the global matrix, which is very much similar to the procedure used in the MFree local weak-form methods discussed in Chapter 5.

Note also that the global system matrix $\mathbf{K}^{\mathrm{DC}}$ given in Equation (6.28) is, usually, sparse because of the use of the local support domain that contains usually a very small portion of the field nodes, and many of entries in $\mathbf{K}^{\mathrm{DC}}$ are zero. It is, however, asymmetric for the reasons given in Sub-section 5.2.2.

2) The method using the fictitious point (FP)

In order to impose the DBC, a fictitious point beyond the DB-node is added outside the problem domain. The coordinate of this fictitious point is

$$x_{N+1} = x_N + d_c \tag{6.31}$$

where d_c is the nodal spacing given by

$$d_c = x_N - x_{N-1} \tag{6.32}$$

Hence, an additional degree of freedom (DOF), u_{N+1} is added into the system, and the discretized global system equation becomes

$$\mathbf{K}^{\mathrm{FP}}_{(N+1)\times(N+1)}\mathbf{U}^{\mathrm{FP}}_{(N+1)\times 1} = \mathbf{F}^{\mathrm{FP}}_{(N+1)\times 1} \tag{6.33}$$

where the global stiffness matrix $\mathbf{K}^{\mathrm{FP}}$ becomes

$$\mathbf{K}^{\mathrm{FP}}_{(N+1)\times(N+1)} = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1N} & K_{1(N+1)} \\ K_{21} & K_{22} & \cdots & K_{2N} & K_{2(N+1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ K_{N1} & K_{N2} & \cdots & K_{NN} & K_{N(N+1)} \\ K_{(N+1)1} & K_{(N+1)2} & \cdots & K_{(N+1)N} & K_{(N+1)(N+1)} \end{bmatrix}$$

Related to Dirichlet BC from Equation (6.14)

Related to the system PDE from Equation (6.11)

Related to DBC from Equation (6.25)

(6.34)

Note that there are two equations to be satisfied at the DB-node at x_N: Equations (6.9) and (6.24). The global source vector **F** becomes

$$\mathbf{F}^{\mathrm{FP}}_{(N+1)\times 1} = \begin{Bmatrix} \overline{u} \\ -q_A(x_2) \\ \vdots \\ -q_A(x_N) \\ -q_B(x_N) \end{Bmatrix}$$

Related to Dirichlet BC from Equation (6.15)

Related to the system PDE from Equation (6.12)

Related to DBC from Equation (6.26)

(6.35)

where the global vector of nodal function values $\mathbf{U}^{\mathrm{FP}}$ is

$$\mathbf{U}^{\mathrm{FP}}_{(N+1)\times 1} = \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \\ u_{N+1} \end{Bmatrix} \tag{6.36}$$

Solving these N+1 equations given in Equation (6.33) for the N+1 unknowns, we obtain the nodal values for all field nodes including the fictitious point.

3) The Hermite-type collocation (HC) method

In the Hermite-type approximation, the derivative variable for the DB-node is added as an additional DOF. For an internal collocation node at x_I, if its local support domain does not include the DB-node, the conventional MFree shape functions are used, and the Equations (6.9)~(6.12) are used to derive the collocation equations. If its support domain includes the DB-node, the following formulation is used based on the Hermite-type shape functions (see, Chapter 3):

$$u^h(x_I) = \mathbf{\Phi}^{\mathrm{T}}\mathbf{u}_s = \left\{\phi_1 \quad \cdots \quad \phi_n \quad \phi^H\right\}\begin{Bmatrix} u_1 \\ \vdots \\ u_n \\ u'_N \end{Bmatrix} \tag{6.37}$$

where $\mathbf{\Phi}$ is the vector of shape functions obtained using the Hermite-type approximation, ϕ^H is the shape function related to the derivative DOF u'_N, $\mathbf{u}_s$ is the vector that collects nodal function values, and $u'_N = \dfrac{du(x_N)}{dx}$ which is the additional derivative DOF. Hence, the derivatives of u at the node I can be approximated using

$$\begin{aligned} \frac{\partial u_I^h}{\partial x} &= \frac{\partial \mathbf{\Phi}^{\mathrm{T}}}{\partial x}\mathbf{u}_s = \left\{\frac{\partial \phi_1}{\partial x} \quad \cdots \quad \frac{\partial \phi_n}{\partial x} \quad \frac{\partial \phi^H}{\partial x}\right\}\mathbf{u}_s, \\ \frac{\partial^2 u_I^h}{\partial x^2} &= \frac{\partial^2 \mathbf{\Phi}^{\mathrm{T}}}{\partial x^2}\mathbf{u}_s = \left\{\frac{\partial^2 \phi_1}{\partial x^2} \quad \cdots \quad \frac{\partial^2 \phi_n}{\partial x^2} \quad \frac{\partial^2 \phi^H}{\partial x^2}\right\}\mathbf{u}_s \end{aligned} \tag{6.38}$$

Therefore, for an internal node whose support domain includes the DB-node, the nodal matrix $\mathbf{K}_I$ derived using Equation (6.9) is re-written as

$$\begin{aligned} \mathbf{K}_I &= A_2(x_I)\frac{d^2\mathbf{\Phi}^{\mathrm{T}}}{dx^2} + A_1(x_I)\frac{d\mathbf{\Phi}^{\mathrm{T}}}{dx} + A_0(x_I) \\ &= \left\{ A_2(x_I)\frac{d^2\phi_1}{dx^2} + A_1(x_I)\frac{d\phi_1}{dx} + A_0(x_I) \quad \cdots \right. \\ &\qquad A_2(x_I)\frac{d^2\phi_n}{dx^2} + A_1(x_I)\frac{d\phi_n}{dx} + A_0(x_I) \\ &\qquad \left. A_2(x_I)\frac{d^2\phi^H}{dx^2} + A_1(x_I)\frac{d\phi^H}{dx} + A_0(x_I) \right\}_{1\times(n+1)} \end{aligned} \tag{6.39}$$

For the DB-node, the Hermite-type approximation, Equation (6.37), is used. There are now two equations should be satisfied at the DB-node at x_N. One is Equation (6.9) that results in the similar nodal matrix $\mathbf{K}_I$ presented in Equation (6.39), and the other is Equation (6.24) where the nodal matrix $\mathbf{K}_{N+1}$ for the collocation node at x_N can be re-written as

$$\begin{aligned}\mathbf{K}_{N+1} &= B_1(x_N)\frac{d\mathbf{\Phi}^{\mathrm{T}}}{dx} + B_0(x_N)\mathbf{\Phi}^{\mathrm{T}} \\ &= \left\{ B_1(x_N)\frac{d\phi_1}{dx} + B_0(x_N)\phi_1 \quad \cdots \right. \\ &\qquad B_1(x_N)\frac{d\phi_n}{dx} + B_0(x_N)\phi_n \\ &\qquad \left. B_1(x_N)\frac{d\phi^H}{dx} + B_0(x_N)\phi^H \right\}_{1\times(n+1)}\end{aligned} \tag{6.40}$$

The discretized global system equation becomes

$$\mathbf{K}^{\mathrm{HC}}_{(N+1)\times(N+1)}\mathbf{U}^{\mathrm{HC}}_{(N+1)\times 1} = \mathbf{F}^{\mathrm{HC}}_{(N+1)\times 1} \tag{6.41}$$

where the global matrix $\mathbf{K}^{\mathrm{HC}}$ is given by

$$\mathbf{K}^{\mathrm{HC}}_{(N+1)\times(N+1)} = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1N} & K_{1(N+1)} \\ K_{21} & K_{22} & \cdots & K_{2N} & K_{2(N+1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ K_{N1} & K_{N2} & \cdots & K_{NN} & K_{N(N+1)} \\ K_{(N+1)1} & K_{(N+1)2} & \cdots & K_{(N+1)N} & K_{(N+1)(N+1)} \end{bmatrix}$$

Related to Dirichlet BC from Equation (6.14)

Related to the system PDE from Equation (6.11) or (6.39)

Related to DBC from Equation (6.40)

(6.42)

the global vector $\mathbf{U}^{\mathrm{HC}}$ is given by

$$\mathbf{U}^{\mathrm{HC}}_{(N+1)\times 1} = \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \\ u'_N \end{Bmatrix} \tag{6.43}$$

and the global vector $\mathbf{F}^{\mathrm{HC}}$ has the same form as $\mathbf{F}^{\mathrm{FP}}$ presented in Equation (6.35).

Solving Equation (6.41) for N+1 unknowns, we can obtain the nodal function values for all field nodes.

4) The method using regular grids (RG)

Three regularly distributed nodes, $x_{N\text{-}2}$, $x_{N\text{-}1}$, x_N are used inside the problem domain near the derivative boundary. The following standard finite difference scheme is used to construct the 1st derivative at the DB-node at x_N.

$$\frac{\partial u(x_N)}{\partial x} = \frac{3u_N - 4u_{N-1} + u_{N-2}}{x_N - x_{N-2}} \tag{6.44}$$

Replacing Equation (6.5) with this equation, we can obtain the discretized system equation for the DB-node from the DBC Equation (6.2). Together with Equations (6.9), (6.13), and (6.24), we can obtain N equations for N unknowns of nodal function values. Solving these N equations gives the nodal values for all field nodes. Note that the procedure for the RG method is exactly the same as the DC method, except that the derivative for the DB-node is approximated using Equation (6.44) instead of Equation (6.5).

6.3.2 Numerical examples for 1D problems

In this section, several 1D examples are numerically analyzed reveal the features of the collocation method with different treatments for the DBCs. Because the analytical (exact) solutions are available, it is easy to conduct a detailed analysis of errors in the numerical solutions. The following norms are defined as error indicators in this chapter.

The error in the solution of function value is defined as e_0:

$$e_0 = \sqrt{\frac{\sum_{i=1}^{N}\left(u_i^{\text{exact}} - u_i^{\text{num}}\right)^2}{\sum_{i=1}^{N}(u_i^{\text{exact}})^2}} \tag{6.45}$$

where u_i^{exact} is exact values of the function, and u_i^{num} is numerical values of function obtained using the numerical methods.

The errors in the 1st derivatives of the function is defined as e_x

$$e_x = \sqrt{\frac{\sum_{i=1}^{N}\left(u_{i,x}^{\text{exact}} - u_{i,x}^{\text{num}}\right)^2}{\sum_{i=1}^{N}(u_{i,x}^{\text{exact}})^2}} \tag{6.46}$$

where $u_{i,x}^{\text{exact}}$ is the exact values of the 1st derivative, and $u_{i,x}^{\text{num}}$ is the numerical value of the 1st derivative obtained using the numerical methods.

The rates of h-convergence of the relative error norms in numerical results, $R(e)$, are defined as

$$R(e)=\frac{\mathrm{Log}_{10}(e_{i+1}/e_i)}{\mathrm{Log}_{10}(d_c^{i+1}/d_c^i)} \tag{6.47}$$

where e should be e_0 or e_x, and d_c^{i+1} and d_c^i are the uniform nodal spacing of two consecutive nodes.

Example 6.1: Wave propagation problem with Dirichlet boundary conditions

One-dimensional problem governed by the following second-order linear ordinary differential equation (ODE) is solved by the polynomial point collocation method (PPCM), where the polynomial PIM shape functions are used in Equations (6.4)~(6.6) for the field function approximation.

$$\frac{d^2u}{dx^2}+\lambda u=0, \qquad x\in(0,1) \tag{6.48}$$

which is subjected to the following Dirichlet boundary conditions

$$u(0)=0, \qquad u(1)=1.0 \tag{6.49}$$

Equation (6.48) governs different types of physical problems depends on the value of λ. When λ>0, Equation (6.48) is the well-known 1D wave propagation problem, and the exact solution can be easily found

$$u^{\text{exact}}(x)=\frac{\sin\sqrt{\lambda}x}{\sin\sqrt{\lambda}} \tag{6.50}$$

Three models with 21, 41 and 81 regularly distributed nodes are used to discretize the 1D problem domain. Three different kinds of interpolation schemes using 3 nodes, 5 nodes and 7 nodes, as shown in Figure 6.2, are adopted in the interpolations. There is no DBC in this example. The conventional polynomial PIM is used to construct shape functions.

The errors in the numerical results of function u and its derivative $u_{,x}$ are listed in Table 6.1~Table 6.2. For further illustration, some representative results have also been plotted in Figure 6.3~Figure 6.4.

It can be found that very good (numerical) convergence rates have been obtained using the PPCM.

1) For the 3-node interpolation scheme, the convergence rate of e_0 is about 2.0 and the convergence rate of e_x is close to 2.0.

2) For the 5-node interpolation scheme, the convergence rate of e_0 is about 4.0-5.0 and the convergence rate of e_x is the nearly same as e_0.

3) For the 7-node interpolation scheme, the convergence rate of e_0 is about 6.0-7.0 and the convergence rate of e_x is the also same as its e_0.

4) The error for the derivative solution is slightly bigger than that for the corresponding function solution.

It can be found from these tables that the errors in the numerical results obtained using the PPCM seems to be of the order $O(d_c{}^{p+1})$ for both the field function and its derivatives, where d_c is the nodal spacing. For example, for the case of 5-node scheme (p=4), when the nodal density is doubled, the error decrease to $\left(\frac{1}{2}\right)^{4+1} = \frac{1}{32}$ times as shown in Table 6.1. For the case of 7-node scheme (p=6), when the nodal density is doubled, the error decrease to $\left(\frac{1}{2}\right)^{6+1} = \frac{1}{128}$ times as shown in Table 6.1. Note also that the errors for the first derivatives of the field functions are also about the same order of $O(d_c{}^{p+1})$, as seen in Table 6.2. This only exception is for the case of 3-node scheme for which the error is of the order of $O(d_c{}^{p})$. These facts demonstrate that the collocation method is stable and convergent for problems without DBCs.

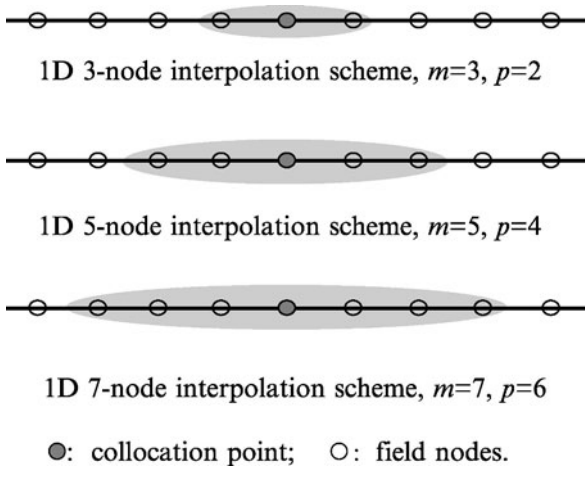


Figure 6.2. Interpolation schemes with different sizes of support domains for 1D problems (m: number of polynomial basis; p: complete order of the polynomial basis).

Table 6.1. *h*- and *p*- convergence of u obtained numerically using different interpolation schemes

Model \ λ		1.0		10.0		100.0	
		e_0(%)	R	e_0(%)	R	e_0(%)	R
3-node scheme (p=2)	21	1.97×10^{-3}	/	13.82	/	15.32	/
	41	4.85×10^{-4}	2.02	3.84	1.85	4.24	1.85
	81	1.20×10^{-4}	2.02	0.99	1.96	1.09	1.96
5-node scheme (p=4)	21	2.72×10^{-6}	/	0.043	/	0.36	/
	41	9.23×10^{-8}	4.88	2.40×10^{-4}	7.49	7.88×10^{-3}	5.51
	81	3.22×10^{-9}	4.84	1.11×10^{-4}	1.11	1.25×10^{-3}	2.65
7-node scheme (p=6)	21	5.14×10^{-9}	/	1.74×10^{-3}	/	0.11	/
	41	4.24×10^{-11}	6.92	1.62×10^{-5}	6.75	1.70×10^{-3}	6.06
	81	1.36×10^{-12}	4.97	1.53×10^{-7}	6.72	1.69×10^{-5}	6.65

R: the convergence rate

Table 6.2. *h*- and *p*- convergence of $u_{,x}$ obtained numerically using different interpolation schemes

Model \ λ		1.0		10.0		100.0	
		e_x(%)	R	e_x (%)	R	e_x (%)	R
3-node scheme (p=2)	21	4.12×10^{-2}	/	13.99	/	16.97	/
	41	1.03×10^{-2}	2.00	3.89	1.85	4.74	1.84
	81	2.58×10^{-3}	2.00	1.00	1.96	1.22	1.96
5-node scheme (p=4)	21	1.46×10^{-5}	/	0.04	/	0.31	/
	41	1.09×10^{-6}	3.75	3.08×10^{-4}	7.11	1.18×10^{-2}	4.74
	81	7.43×10^{-8}	3.87	1.16×10^{-4}	1.41	1.73×10^{-3}	2.77
7-node scheme (p=6)	21	2.28×10^{-8}		1.75×10^{-3}		0.12	
	41	2.64×10^{-10}	6.43	1.63×10^{-5}	6.75	1.83×10^{-3}	6.04
	81	2.15×10^{-11}	3.62	1.56×10^{-7}	6.71	1.91×10^{-5}	6.59
quadratic FEM	21	4.17×10^{-2}		0.31		3.25	
	41	1.04×10^{-2}	2.00	9.76×10^{-2}	1.68	0.98	1.73
	81	2.60×10^{-3}	2.00	2.56×10^{-2}	1.93	0.257	1.93

R: the convergence rate

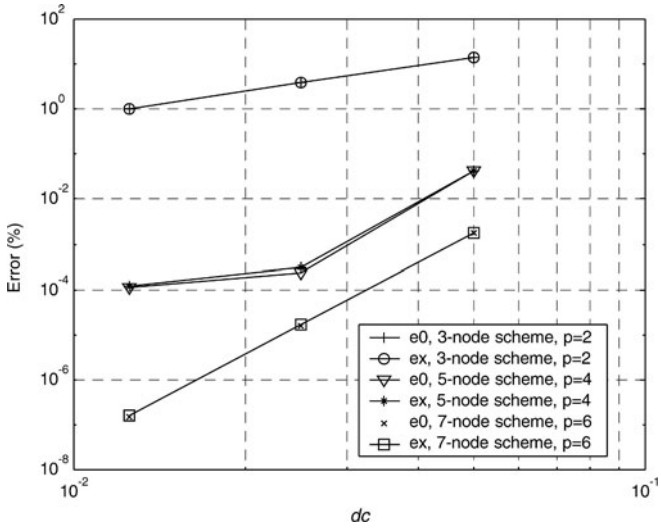


Figure 6.3. *h*-convergence of the PPCM with different interpolation schemes ($\lambda = 10$), where d_c is the nodal spacing.

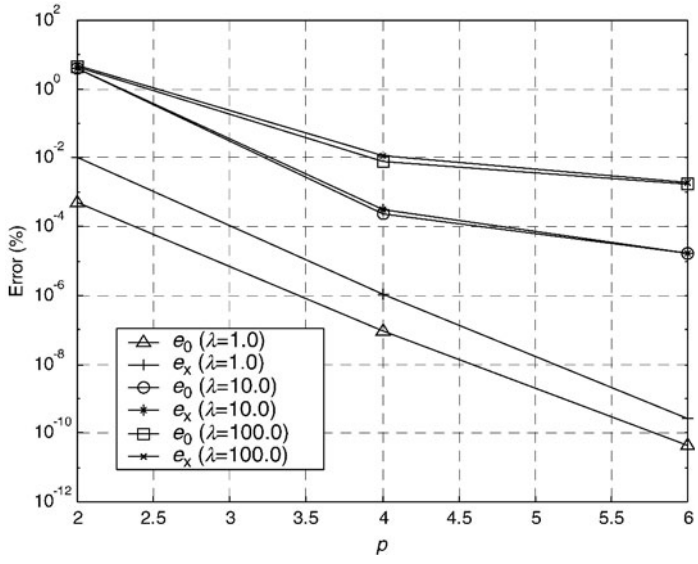


Figure 6.4. *p*-convergence of the PPCM using uniform 41 nodes for the wave propagation problems.

For comparisons, the results are also obtained using the quadratic FEM. It is well-known that the convergence rate for the function value obtained

using the FEM is of the order of $O(h^{p+1})$ (Zienkiewicz and Taylor, 2000) that is the same as that of the collocation method. Note also that in the conventional FEM, the error in derivative results is of the order of $O(h^p)$ [†] which is surely higher than that of the collocation method that is of the order of $O(d_c{}^{p+1})$.

Note from this example that in the absent of the presence derivative boundary conditions, the present PPCM can obtain stable and very accurate solutions for the 1D problems. We have also studied the boundary layer problems (when λ<0), and similar results were found.

Example 6.2: 1D truss member with derivative boundary conditions

Consider a truss member or bar with force (derivative) boundary conditions, as shown in Figure 6.5. The mechanics of the bar were discussed in Sub-section 1.4.6. The bar is governed by the following equations:

- Governing equation in the form of ODE:

$$EA\frac{\mathrm{d}^2u}{\mathrm{d}x^2}+b(x)=0 \tag{6.51}$$

where E is the Young's modulus, A is the cross-section area, u is the axial displacement in the x direction, b is the body force in x direction, and L is the length of the truss element. For simplicity, $E=1.0$, $A=1.0$, L=1.0.

Two cases of the $b(x)$ are considered. The source force term with the polynomial form that was used in Section 1.4 is first considered. Due to the reproducibility of the polynomial PIM, very accurate results were obtained using the collocation method with different treatments for DBCs.

To study numerically the convergence and accuracy of the collocation methods, a more complex source term of non-polynomial form $b(x)=-(2.3\pi)^2\sin(2.3\pi x)$ is used in this study.

- Displacement (Dirichlet) boundary condition is given by:

$$u\big|_{x=0}=0 \tag{6.52}$$

which means that the bar is fixed at x=0 as shown in Figure 6.5.

- Force (derivative) boundary condition is given by:

[†] The rate can change in the FEM, if the so-called super-convergence points can be found. These kinds of special points may also exist in the collocation methods. Here, we discuss only the results sampled at arbitrary points.

$$f = A\sigma_x\big|_{x=L} = EA\left.\frac{\mathrm{d}u}{\mathrm{d}x}\right|_{x=L} = -2.3\pi\cos(2.3\pi x) \tag{6.53}$$

or

$$\left.\frac{\mathrm{d}u}{\mathrm{d}x}\right|_{x=1} = -2.3\pi\cos(2.3\pi) \tag{6.54}$$

which means that a concentrated force is applied at $x=L$.

The exact solution of the problem can be easily obtained by solving analytically ODE with these boundary conditions, which yields

$$u^{\text{exact}}(x) = -\sin(2.3\pi x) \tag{6.55}$$

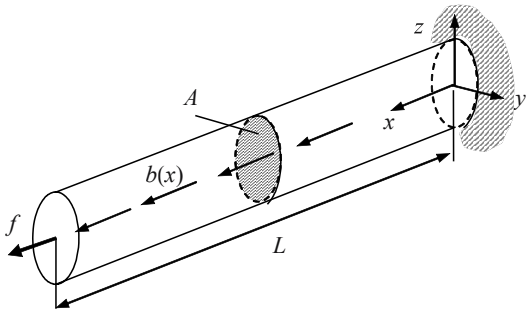


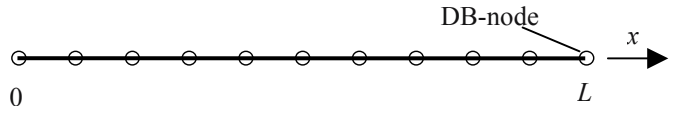
Figure 6.5. A uniform truss member fixed at $x=0$ and subjected to an axial loading distributed in x direction and a concentrated force at $x=L$.

The same problem can be solved by imposing the following displacement (Dirichlet) boundary conditions at $x=L$.

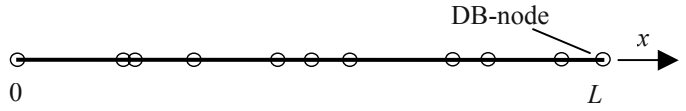
$$u\big|_{x=L=1} = -\sin(2.3\pi) \tag{6.56}$$

which obtained simply using Equation (6.55). In this case, the problem can be solved without using any derivative boundary conditions.

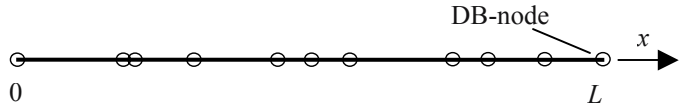
In seeking for an approximate numerical solution, we represent the 1D truss member with regularly and irregularly distributed nodes shown in Figure 6.6. The polynomial point collocation method (PPCM) is again used to discretize Equations (6.51)~(6.54). The five different techniques presented in Section 6.2 and Sub-section 6.3.1 are used to treat the force (derivative) boundary condition in the following manner:



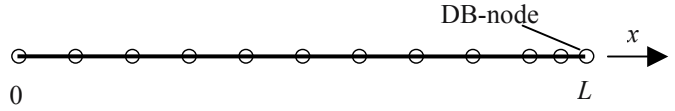
(a) 10 regular nodes



(b) 10 irregular nodes



(c) 10 irregular nodes used in RG method



(d) 11 field nodes used in the method of use of dense nodes

Figure 6.6. Nodal distributions on the 1D truss member

1) In the direct collocation (DC) method, the conventional polynomial PIM shape functions are used and the force boundary condition, Equation (6.54), is directly discretized by collocation.
2) In the method using fictitious points (FP), a fictitious point is added at x=1.1. Two equations, Equations (6.51) and (6.54), are imposed at the DB-node at x=L using the conventional polynomial PIM shape functions.
3) In the Hermite-type collocation (HC) method, the Hermite-type polynomial PIM (see Sub-section 3.2.2) shape functions are used. The additional derivative variable, $\mathrm{d}u/\mathrm{d}x$, at the DB-node is added as an additional unknown or DOF.

4) In the method using regular grid (RG): the conventional polynomial PIM shape functions are used, and the standard finite difference scheme given in Equation (6.44) is used to approximate the first order derivative of the displacement at the DB-node.

5) The method of using dense nodes (DN) near the derivative boundaries, one more node is used in the problem domain near the DB-node.

Three interpolation schemes shown in Figure 6.2 are used. To reveal the effect of the DBC on the accuracy of the solution, the average relative error is used as the error indicator, which is defined as

$$e = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|u_i^{\text{num}} - u_i^{\text{exact}}\right|}{\left|u_i^{\text{exact}}\right|} \tag{6.57}$$

where u_i^{num} and u_i^{exact} are the displacement at the ith node obtained using the numerical methods and the exact solution given in Equation (6.55), respectively, and N is the number of field nodes. Note that the case 0 is for the problem with the Dirichlet boundary conditions, Equations (6.52) and (6.56).

Table 6.3 lists the error in numerical results obtained using the collocation methods and the 3-node interpolation scheme shown in Figure 6.2. From Table 6.3, we can make the following remarks.

1) If the problem is subjected only to Dirichlet BCs without any DBC, the collocation method yields very good results. The error is small, only e=0.51% for the regular model. The error for the irregular model is 1.36%, which is about 2.7 times larger than that for regular nodes. This is because the largest nodal spacing for the irregular model is about 2.0 times that of the regular node model. This example indicates the effects of the nodal irregularity on the accuracy of the solution of the PPCM.

2) The presence of the DBC leads to large errors in the solution. If no special treatment for the force boundary condition (the direct collocation method) is used, the error of the direct collocation method becomes 11.3%. The error magnification is more than 22 times.

3) Special treatments for handling the force (derivative) boundary conditions can improve the accuracy of the solution.

4) The Hermite–type collocation method (HC) produces the accurate and stable results for both regular and irregular nodal distributions. The error magnification is about 5 times for the regular nodal distribution.

The error for the irregular nodal distribution is only slightly larger than that without the DBC (case 0).

5) The FP method works reasonably well for the model of regular nodes, but not very well for the irregular nodes.

6) The RG method has the same accuracy as the DC method for the model of regular nodes because the three nodes closest to the Derivative boundary are used in Equation (6.44) that results in the same formulation as the DC in this case. To use the RG method, the 10 irregular nodes shown in Figure 6.6(c) (not Figure 6.6(b)) are used. Table 6.3 shows that the RG method leads to large error for the irregular model.

7) The DN method that uses one more node in the problem domain near the DB-node shown in Figure 6.6(d) leads to good result. This confirms that the use of dense nodes near the derivative boundaries can improve the accuracy of solution. This may be because the use of dense nodes can better approximate the derivative of the function.

8) For the DC and the HC methods, the results of the model of irregular nodes are better than that for regular nodes. This maybe because, in the irregular model shown in Figure 6.6 (b), the nodal spacing near the DB-node is smaller than that in the regular model.

Table 6.3. Relative errors e (%) in the displacement results obtained using the PPCM with different schemes handling the DBCs

Cases	Schemes	Regular nodes (R_m)	Irregular nodes(R_m)
0	Dirichlet BC	0.51 (1.0)	1.36 (2.67)
1	DC	11.3 (22.2)	1.21 (2.37)
2	FP	1.63 (3.2)	7.96 (15.61)
3	HC	2.68 (5.3)	1.42 (2.78)
4	RG	11.3 (22.2)	6.12 (12.0)
5	DN	1.68 (3.3)	/

- 3 nearest nodes are used in the local support domain
- $R_m = \dfrac{e_{\text{case\_}i}}{e_{\text{case\_0\_regular}}}$ is the error magnification rate.

To study the *h*-convergence of these methods for this 1D truss problem, regularly distributed 6, 11, 21, 41 and 81 nodes are used. To study the *p*-convergence, the models of 41 regular nodes with 3-node, 5-node and 7-node interpolation schemes are used. The relative errors, e, in the

displacement results obtained by four different methods (excluding the DN method) are listed in Table 6.4 and plotted in Figure 6.7 and Figure 6.8. All these results re-confirm the fact that these special treatments for enforcement of the force boundary condition are necessary to improve the accuracy of the solution. Again, the Hermite–type collocation method produces better results for all these cases.

Table 6.4. *h*- and *p*- convergence of *u* using different methods with different interpolation schemes and different nodal distributions

		Dirichlet BC	DC	FP	HC	RG
	Number of Nodes	*e*(%)	*e*(%)	*e*(%)	*e*(%)	*e*(%)
3-node (*p*=2)	11	5.10E-1	1.13E+1	1.63E+0	2.68	1.13E+1
	21	6.59E-2	1.21E+0	1.99E-1	2.72E-1	1.21E+0
	41	8.37E-3	1.35E-1	2.46E-2	2.91E-2	1.35E-1
	81	1.05E-3	1.58E-2	3.05E-3	3.31E-3	1.58E-2
5-node (*p*=4)	11	6.30E-2	5.46E+0	5.70E-1	2.92E-1	7.56E+0
	21	1.35E-3	1.73E-1	1.62E-2	7.58E-3	9.55E-1
	41	3.21E-5	4.68E-3	4.06E-4	1.74E-4	1.10E-1
	81	9.12E-07	1.29E-4	1.07E-5	4.28E-6	1.29E-2
7-node (*p*=6)	11	4.31E-2	1.57E+0	8.46E-2	1.55E-2	9.92E+0
	21	1.95E-4	2.24E-2	1.26E-3	1.01E-4	1.02E+0
	41	9.33E-7	1.59E-4	8.42E-6	6.30E-6	1.12E-1

Table 6.4, Figure 6.7 and Figure 6.8 draw the following conclusions.

1) This example illustrate the dominant effects of the DBC
2) Although the accuracies of the different methods are different, their convergence rates are nearly the same.
3) The errors in the numerical results obtained using the PPCM seems again to be of the order of $O(d_c{}^{p+1})$, where d_c is the nodal spacing, regardless the presence of the DBC.

4) The RG method does not achieve the *p*-convergence (see, Figure 6.8) because the three nodes closest to the derivative boundary are always used in Equation (6.44) for all the different interpolation schemes used for the internal nodes. Since the error is largely controlled by the large error induced by the boundary conditions, the accuracy will not be improved, despite the increase of the order of the interpolation for all the internal nodes, and the *p*-convergence of the RG method is poor as shown in Figure 6.8.

The simple 1D example demonstrates that the enforcement of derivative boundary conditions (DBCs) is the major technical issue in the use and the development of MFree strong-form methods. A special treatment is required to enforce the DBCs, and for this 1D problem, the Hermite method seems to work well for both regular and irregular nodes.

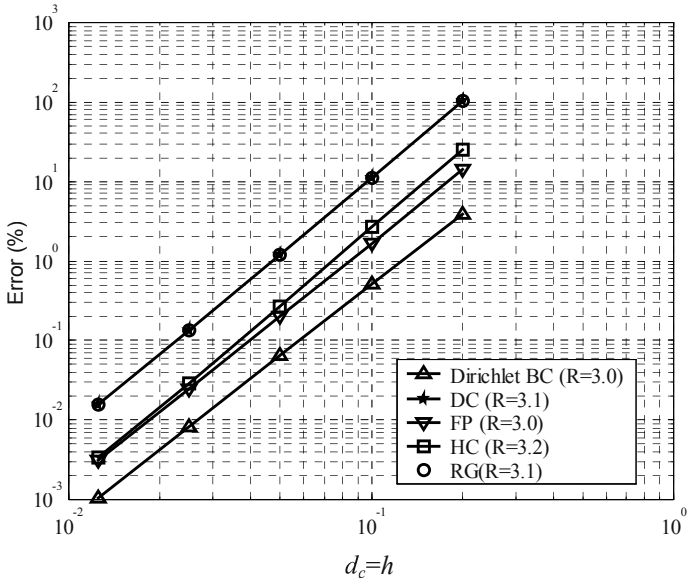


Figure 6.7. *h*-convergence in relative errors in the numerical results obtained using the PPCM with different schemes handling the DBCs (3-node scheme). The R is the convergence rate, and d_c is the nodal spacing

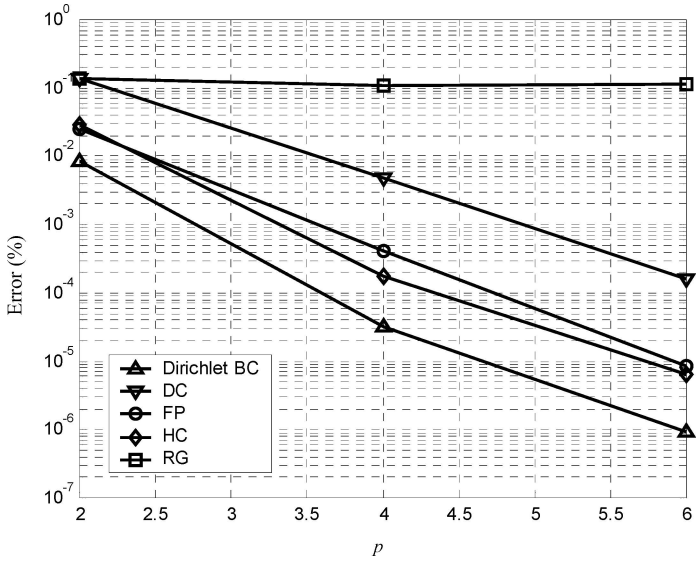


Figure 6.8. *p*-convergence of the PPCM using different schemes handling the DBCs (41 regular nodes)

6.4 STABILIZATION IN CONVECTION-DIFFUSION PROBLEMS USING MFREE METHODS

Many practical problems in engineering are governed by the so-called *convection-diffusion equations*, and hence the convection-diffusion problem is important in computational mechanics. In a convection-diffusion equation, there are *convective* and *diffusive* terms, and there is a well-known technical issue in the analysis for convection-diffusion problem using the numerical methods: the *instability* in the solution when the problem becomes *convection dominated*. Much research has been performed to solve the instability problem, and an overview on this topic can be found in the book by Zienkiewicz and Taylor (2000). Many useful techniques have been developed for stabilizing the numerical solution for FDM (Courant, et al. 1952; Runchall et al. 1969; Spalding, 1972; etc.), FEM (e.g., Zienkiewicz and Taylor, 2000), FPM (Oñate et al., 1996), and the GFDM (e.g., Cheng et al., 1999, 2002). In GFDM used for CFD problems by Cheng et al. (1999, 2002), a simple idea similar to the upwind stabilization scheme is used by choosing more nodes on the side of the support domain facing the flow. In this section, the stability problem in the analysis of the convection-diffusion

problem using MFree methods is discussed through a 1D example problem of steady state convection-diffusion. The techniques studied are very simple and easy to implement in MFree methods, and are in principle applicable also to 2D or 3D problems. To simplify the issue, our discussion in this section is confined for problems with only Dirichlet boundary conditions.

Consider a 1D steady-state convection-diffusion problem governed by the following equations (Zienkiewicz and Taylor, 2000).

- Governing equation:

$$V\frac{du}{dx}-\frac{d}{dx}(k\frac{du}{dx})+q=0\,,\ x\in(0,\,1) \tag{6.58}$$

  where u is a scalar field variable, V, k and q are all given constants, and u, V, k and q carry different physical meanings for different engineering problems.

- Dirichlet boundary condition:

$$\begin{cases} u\big|_{x=0}=0 \\ u\big|_{x=1}=1 \end{cases} \tag{6.59}$$

Equation (6.58) is an ordinary differential equation (ODE) of second order with constant coefficients, and it is a special case of Equations (6.1). The exact solution is

$$u^{\text{exact}}(x)=-\frac{q}{V}+\frac{k}{V}c_1e^{\frac{V}{k}x}+c_2 \tag{6.60}$$

where

$$c_1=(1+V/q)\frac{1}{k/h(e^{V/k}-1)} \tag{6.61}$$

$$c_2=-kc_1/V \tag{6.62}$$

The stability of the numerical solution of this problem depends on the so-called the *Peclet number*

$$Pe=\frac{Vd_c}{2k} \tag{6.63}$$

where d_c is the nodal spacing.

In this example, the problem domain is represented using 21 regularly distributed nodes, and hence $d_c=0.05$, and PPCM is used. To simplify the problem, the source term is omitted: q=0 is used. The support domain is

defined to select 3 nearest nodes for computing the PIM shape functions. Figure 6.9 shows the results of u obtained using the PPCM for different Peclet numbers. The accuracy of solutions deteriorates as Pe increases, if no special treatment is performed. When Pe is large, Equation (6.58) becomes convection dominated, and the accuracy of the standard numerical result becomes oscillatory. If only the conduction term is omitted (k=0), which leads to $Pe \to \infty$, the standard numerical procedure fails.

When the equation is convection dominated, the second term in Equation (6.58) is negligible, and the down stream boundary condition $u\big|_{x=1} = 1$, to affects only a narrow region to form a thin *boundary layer*. The thin boundary layer is difficult to reproduce by a standard numerical method, and results become oscillatory.

This type of instability can occur in many numerical methods including FEM, FDM and the MFree method if no special treatment is implemented. The key to overcoming this problem is to effectively capture the upstream information in the approximation of the field variables. The so-called *upwind scheme* widely used in FDM was developed precisely for this purpose (Courant, et al. 1952; Runchall et al. 1969; Spalding, 1972; etc.). In the following, we discuss some simple strategies in MFree methods to deal with this instability problem.

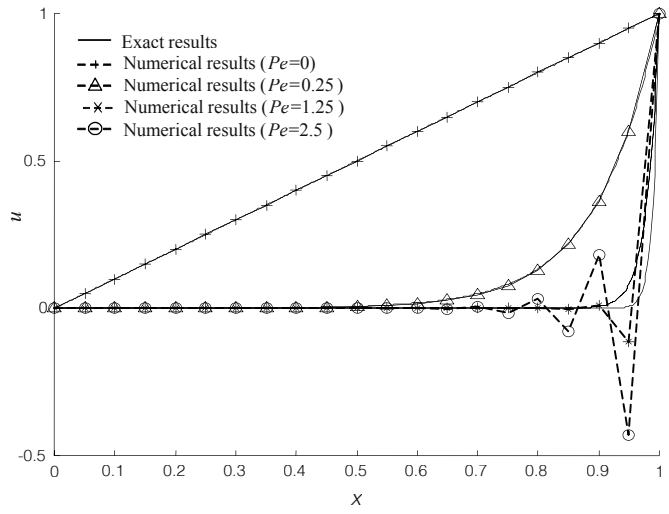


Figure 6.9. Results for the convection-diffusion problem with different Peclet numbers. A total of 21 regularly distributed nodes are used in the PPCM and the support domain is defined to select 3 nearest nodes.

6.4.1 Nodal refinement

It is known that the instability is directly related to the Peclet number. Therefore, a natural way to stabilize the solution is to reduce the Peclet number by reducing the nodal spacing d_c for given V and k.

To confirm this argument, two models with 21 and 41 regularly distributed nodes are used to solve the same problem. The local support domain is defined to select the 3 nearest nodes, and results are plotted in Figure 6.10. It can be found that using finer field nodes is one of the simple ways to alleviate the instability problem. Note however that this is not an effective way to solve the instability problem. Increasing the nodal density only in the boundary layer can certainly be more efficient.

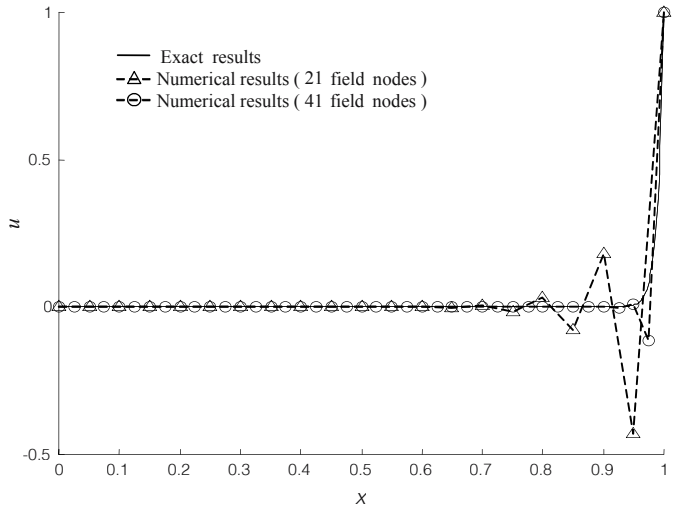


Figure 6.10. Results of the 1D convection-diffusion problem with different nodal distributions. V=100 and k=1 are considered and the support domain is defined to select 3 nearest nodes.

6.4.2 Enlargement of the local support domain

The instability is caused by the failure to capture the upstream information. The simplest way to capture the upstream information is naturally to use more nodes in the interpolations. This may not be done easily in FDM or FEM, but can be done without any difficulty in MFree methods by simply enlarging the support domain of the collocation node near the boundary layer.

Three types of local support domains, selecting 3, 5 and 7 nearest field nodes, are used to solve the same problem, and results obtained using the

PPCM are plotted in Figure 6.11. The accuracy and stability of solutions are significantly improved by the enlargement of the local support domain. Note that the overlap feature in the MFree interpolations may help also to stabilize the solution.

Note that the enlargement of the local support domain needs to be done only for the interpolation points (collocation nodes) that are in and near the boundary layer.

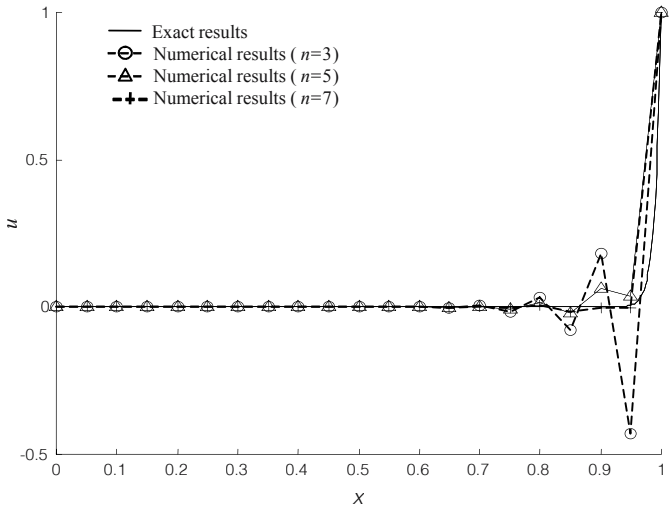
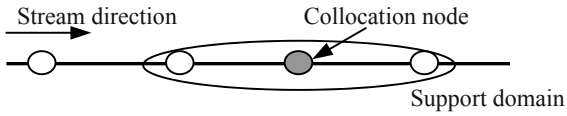


Figure 6.11. Results of the convection-diffusion problem solved using the PPCM with different support domains. *Pe*=2.5 is considered and a total of 21 regularly distributed nodes are used.

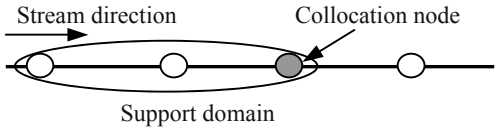
6.4.3 Total upwind support domain

Similar to the upwind difference scheme used in the FDM, the local *upwind support domain*, as shown in Figure 6.12(b), is proposed here and implemented in the PPCM to stabilize the solution. Results for $Pe = 0.25$ and $Pe = 2.5$ are obtained and plotted in Figure 6.13. The upwind support domain improves the accuracy and stability for large Peclet numbers because it can fully capture the information from upstream. However, it gives poor results for cases of smaller Peclet numbers because of the fully asymmetric interpolation using the upwind support domain, which mis-represents the conduction term that is a 2nd derivative operator and should be symmetric. In contrast, when using the normal symmetric local support domain, it gives good results for small Peclet numbers but unstable results for large Peclet numbers. Hence, the ideal support domain should change

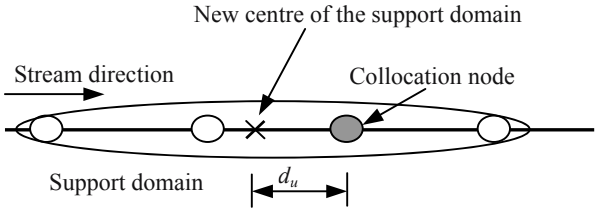
with Peclet number, i.e., when *Pe* increases, the support domain should be biased towards the upwind side. We term such a support domain an *adaptive upwind support domain*.



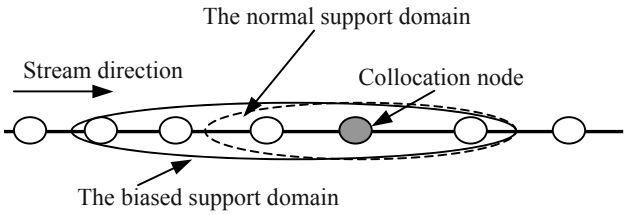
(a) the normal support domain that is symmetric with respect to the collocation point



(b) the upwind support domain that is fully biased on the upwind side

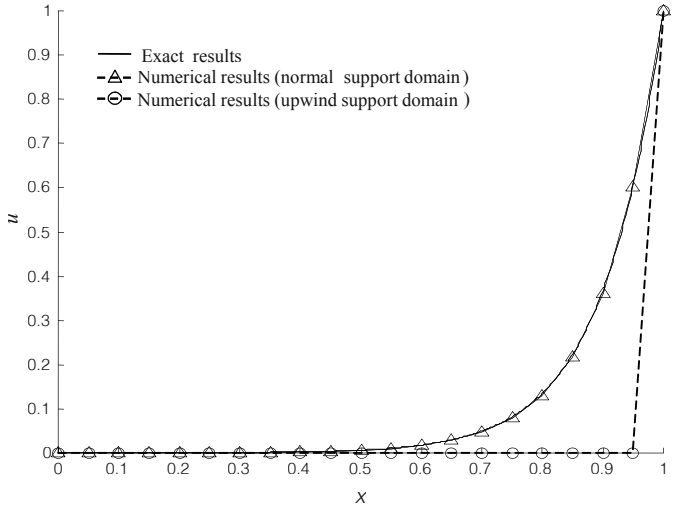


(c) Construction of an adaptive upwind local support domain with an offset distance d_u

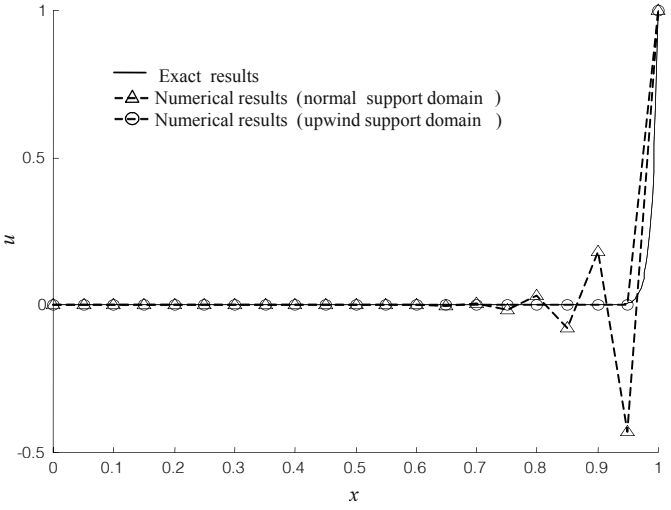


(d) Construction of a biased support domain by deliberately adding two more nodes in the support domain in the upstream direction

Figure 6.12. Different types of local support domains.



(a) *Pe*=0.25



(b) *Pe*=2.5

Figure 6.13. Results of the convection-diffusion problem with normal and upwind support domain. The support domain is defined to select 3 nearest nodes.

6.4.4 Adaptive upwind support domain

The adaptive upwind support domain can be defined using the following formula (Zienkiewicz et al., 1975; Christie et al., 1976; Zienkiewicz and Taylor, 2000; Atluri and Shen, 2002)

$$d_u = (\coth|Pe| - 1/|Pe|) \cdot r_s \tag{6.64}$$

where d_u is the central offset distance against the stream direction from the collocation node as shown in Figure 6.12(c), and r_s is the dimension of the support domain. Clearly Equation (6.64) satisfies the following two conditions.

- When *Pe*=0, the central support domain should be used and $d_u = 0$.
- When *Pe*=∞, fully upwind support domain should be used and $d_u = r_s$.

Figure 6.14 shows that Equation (6.64) works well for both large and small Peclet numbers (the results are not presented here). It is one of the most effective methods to ensure the stability of convection dominated problems.

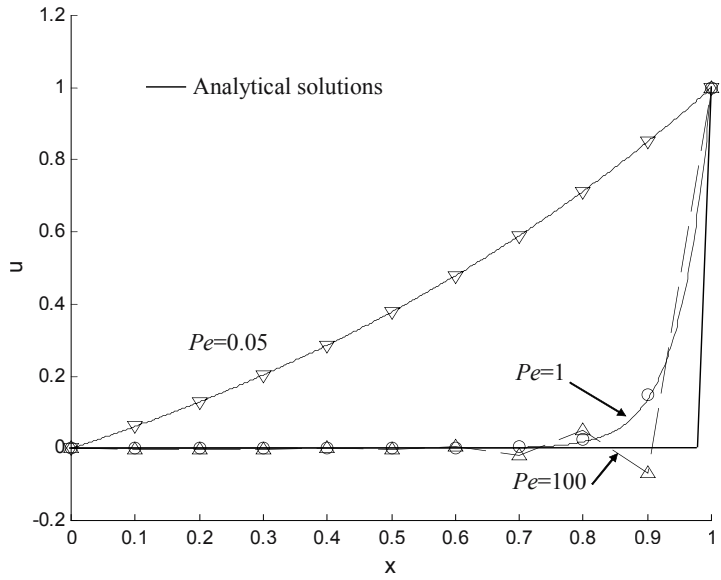


Figure 6.14. Results of the convection-diffusion problem with adaptive upwind support domain s.

6.4.5 Biased support domain

Another effective and simple way to establish a *biased support domain* is to deliberately select more nodes in the upstream direction when constructing the local support domain for a collocation node (Cheng and GR Liu, 1999, 2002). Figure 6.12(d) shows a biased support domain based on a normal support domain by adding two more nodes that are in the upstream

direction. The PPCM with the biased support domain gives accurate result both large and small Peclet numbers. Due to the freedom in selecting the support domain in MFree methods, the method of using the biased support domain is very effective and easy to use in practical applications (Cheng and GR Liu, 1999, 2002).

In summary, using MFree methods to analyze the convection dominated problem, the above mentioned simple methods can be used overcome the instability issues in convection dominated problems. In these methods, the adaptive upwind support domain and the enlargement of the local support domain are the most effective methods and they are very easy to use because of the freedom of selecting the support domain in an MFree method. Comparing with the conventional FDM and FEM, the MFree method has a very attractive advantage in solving the convection dominated problems because it can easily overcome the instability problem even without the need of any special treatment.

We have discussed MFree strong-form methods. MFree weak-form methods can be modified in a similar way. Therefore, in solving a convection-diffusion problem, the similar conclusions can be drawn for the weak-form methods. In addition, the use of different weak-forms can be other effective alternatives, as those used in the FEM (Zienkiewicz and Taylor, 2000): the Petrov-Galerkin weak-form (Zienkiewicz et al., 1975; Hughes and Brooks, 1982; Kelly et al., 1980), the finite increment calculus (FIC) (Oñate , 1998), etc.

6.5 POLYNOMIAL POINT COLLOCATION METHOD FOR 2D PROBLEMS

This section introduces PPCM for solving 2D problems. When there is only Dirichlet BC, the conventional polynomial PIM shape functions are used. For problems with DBCs, the conventional PIM shape functions are still used for all the nodes whose support domains do not contain any DB-nodes, but for nodes whose support domains contain at least one DB-node, shape functions created using the Hermite-type weighted least square (WLS) polynomial approximation (see, Sub-section 3.2.1) are used.

6.5.1 PPCM formulation for 2D problems

Let us consider problems governed by the following general second-order partial differential equation (PDE) defined in a 2D domain, Ω:

$$\Psi(u) = A_{11}(x,y)\frac{\partial^2 u}{\partial x^2} + 2A_{12}(x,y)\frac{\partial^2 u}{\partial x \partial y} + A_{22}(x,y)\frac{\partial^2 u}{\partial y^2} + A_{10}(x,y)\frac{\partial u}{\partial x} + A_{20}(x,y)\frac{\partial u}{\partial y} + A_{00}(x,y)u + q_A(x,y) = 0 \tag{6.65}$$

where u is an unknown field function whose physical significance depends on the physical problems, q_A is a given source term, and the coefficients A_{11}~A_{00} could depend upon x and y but are all given. There can be two-types of boundary conditions:

- Derivative boundary condition (DBC):

$$L_{DB}(u) = \mathbf{n}^{\mathrm{T}} \cdot \nabla u + q_B = 0 \qquad \text{on } \Gamma_{DB} \tag{6.66}$$

  where q_B is the specified source term on the DBC Γ_{DB}, $\mathbf{n}$ is the vector of the unit outward normal, and ∇ is the vector differential (gradient) operator that is defined by

$$\nabla u = \begin{Bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{Bmatrix} u = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \end{Bmatrix} \tag{6.67}$$

- Dirichlet boundary condition:

$$u - \overline{u} = 0 \qquad \text{on } \Gamma_u \tag{6.68}$$

  where $\overline{u}$ is the specified value of u on the Dirichlet boundary Γ_u.

Assume that there are $N = N_d + N_b$ field nodes in total with N_d internal (domain) nodes and $N_b = N_{\mathrm{DB}} + N_u$ boundary nodes, where N_{DB} is the number of DB-nodes and N_u is the number of nodes on the Dirichlet boundary. Hermite-type collocation (HC) is used to impose the DBC; the derivatives at the DB-nodes shown in Figure 3.3 are considered as additional unknowns or DOFs. For simplicity, we collocate at the field nodes.

For the internal nodes and DB-nodes, the following N_d+N_{DB} equations can be obtained using the following collocation approach.

$$\begin{aligned}\Psi(u_i^h) = {} & A_{11}(x_i,y_i)\frac{\partial^2 u_i^h}{\partial x^2} + 2A_{12}(x_i,y_i)\frac{\partial^2 u_i^h}{\partial x\partial y} + A_{22}(x_i,y_i)\frac{\partial^2 u_i^h}{\partial y^2} + \\ & A_{10}(x_i,y_i)\frac{\partial u_i^h}{\partial x} + A_{20}(x_i,y_i)\frac{\partial u_i^h}{\partial y} + A_{00}(x_i,y_i)u_i^h + q_A(x_i,y_i) \\ & = 0\end{aligned} \tag{6.69}$$

where i=1, 2, …, (N_d+N_{DB}), u_i^h is obtained using the Hermite-type WLS shape functions, and its derivatives are obtained using the following equations.

$$\begin{aligned} & u_i^h = u^h(\mathbf{x}_i) = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{u}_s\,, \quad \frac{\partial u_i^h}{\partial x} = \frac{\partial \boldsymbol{\Phi}^{\mathrm{T}}}{\partial x}\mathbf{u}_s\,, \quad \frac{\partial^2 u_i^h}{\partial x^2} = \frac{\partial^2 \boldsymbol{\Phi}^{\mathrm{T}}}{\partial x^2}\mathbf{u}_s \\ & \frac{\partial u_i^h}{\partial y} = \frac{\partial \boldsymbol{\Phi}^{\mathrm{T}}}{\partial y}\mathbf{u}_s\,, \qquad \frac{\partial^2 u_i^h}{\partial x\partial y} = \frac{\partial^2 \boldsymbol{\Phi}^{\mathrm{T}}}{\partial x\partial y}\mathbf{u}_s\,, \quad \frac{\partial^2 u_i^h}{\partial y^2} = \frac{\partial^2 \boldsymbol{\Phi}^{\mathrm{T}}}{\partial y^2}\mathbf{u}_s \end{aligned} \tag{6.70}$$

where $\boldsymbol{\Phi}$ is the vector of shape functions, and $\mathbf{u}_s$ is the vector that collects the values of the unknown function at all nodes and the 1st derivatives of the function at DB-nodes in the support domain if the Hermite-type shape functions are used.

The following N_{DB} equations can be obtained from the DBCs at the DB-nodes.

$$\mathbf{n}^{\mathrm{T}}\cdot\nabla u_i^h + q_B = \mathbf{n}^{\mathrm{T}}\cdot\nabla\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{u}_s + q_B = 0\,, \quad i = 1,2,\cdots,N_{\mathrm{DB}} \tag{6.71}$$

The following N_u equations can be obtained from Dirichlet boundary condition for nodes on the Dirichlet boundary:

$$u_i^h - \overline{u} = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{u}_s - \overline{u} = 0\,, \quad i = 1,2,\cdots,N_u \tag{6.72}$$

Following the procedure in Section 6.3 for 1D problems, we obtain a set of discretized system equations. In the Hermite-type collocation, two equations are imposed at each DB-node: one equation resulting from the DBC, and the other from the governing PDE.

When PDE is nonlinear in u, an iterative scheme, such as the well-known Newton-Raphson iteration scheme, can be adopted to solve the nonlinear discretized equations.

6.5.2 Numerical examples

In the study of 2D problems, e_0 given in Equation (6.45), e_x given in Equation (6.46) and e_y is defined in the following are used as error indicators.

$$e_y = \sqrt{\frac{\sum_{i=1}^{N}\left(u_{i,y}^{\text{exact}} - u_{i,y}^{\text{num}}\right)^2}{\sum_{i=1}^{N}(u_{i,y}^{\text{exact}})^2}} \tag{6.73}$$

where e_y is the relative error in the 1st derivative of the function with respect to the y coordinate, $u_{i,y}^{\text{exact}}$ is the exact value of the 1st derivatives with respect to the y coordinate, and $u_{i,y}^{\text{num}}$ is the numerical value obtained using the collocation methods.

Example 6.3: PPCM for 2D nonlinear PDEs with Dirichlet BCs

We consider the following PDE that is often seen in chemical engineering

$$\nabla^2 u = k u^{\overline{n}} \tag{6.74}$$

where the parameter k is called the *Thiele modulus* in chemical engineering and represents the ratio of kinetic to transport resistances in the domain, and $\overline{n}$ is the order of reaction. When the Thiele modulus is large, a boundary layer with a thickness of the order of (1/k) is presented. The PPCM using the conventional polynomial PIM shape functions is again used to solve this problem.

The problem domain is a cylindrical container that is idealized as unit circle, and the following Dirichlet conditions on the entire circumferential boundary are considered:

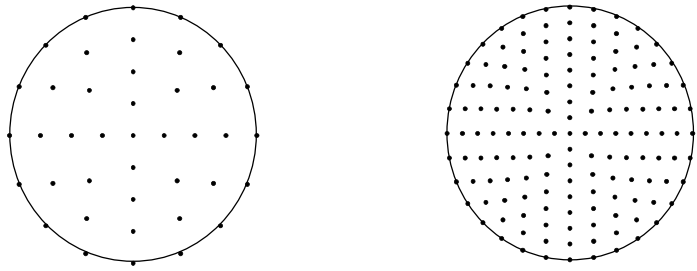
$$u\big|_{r=1} = 1.0 \tag{6.75}$$

This example can be found in the paper by Balakrishnan and Ramachandran (2001). The first-order reaction ($\overline{n} = 1$) and the second order reaction ($\overline{n} = 2$) are studied here. When $\overline{n} = 1$, the analytical solution is available (Balakrishnan and Ramachandran, 2001).

In order to investigate the effect of the parameter k on the solution, k=9 and 100 are chosen. When k=9, a coarse nodal distribution of 41 nodes and a finer nodal distribution of 145 nodes shown in Figure 6.15 (a) and (b) are adopted. In the computation, the sizes of support domain are adjusted to select the 6 nearest neighboring nodes in the interpolation domain. The polynomial basis used to construct PIM shape functions is

$$\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = \left\{1 \quad x \quad y \quad xy \quad x^2 \quad y^2\right\} \tag{6.76}$$

which is of complete 2nd order (p=2). The results obtained are listed in Table 6.5. Comparing with the results provided by Balakrishnan and Ramachandran (2001), the PPCM gives reasonably good results.



(a) 41 nodes (b) 145 nodes

Figure 6.15. Two models of different nodal distributions for the circular domain for the problem defined in Example 6.3 with k=9.

Since the PIM shape functions are used, we can now perform a rough error analysis based on the numerical results listed in Table 6.5. When 41 nodes are used, one can roughly estimate the nodal spacing using Equation (3.3):

$$d_{c(41)} = \frac{\sqrt{A}}{\sqrt{41}-1} = \frac{\sqrt{\pi \times 1^2}}{\sqrt{41}-1} \approx 0.328 \tag{6.77}$$

When 145 nodes are used the estimated nodal spacing is

$$d_{c(145)} = \frac{\sqrt{A}}{\sqrt{145}-1} = \frac{\sqrt{\pi \times 1^2}}{\sqrt{145}-1} \approx 0.1605 \tag{6.78}$$

Since the same number of nodes is used in the local support domains, we can estimate numerically the convergence rate

$$R_e^p = \frac{e_{(41)}}{e_{(145)}} = \frac{(d_{c(41)})^p}{(d_{c(145)})^p} = \frac{(0.328)^2}{(0.1605)^2} \approx 4.176 \tag{6.79}$$

Compared with numerical results R_e shown in Table 6.5, it is found this rough estimation is good.

Table 6.5. Results of u for Example 6.3 obtained using the PPCM and different nodal distributions (k=9)

$\overline{n}$	r	Reference solution	$u(r)$ (41 nodes)	$e_{(41)}$ (%)	$u(r)$ (145 nodes)	$e_{(145)}$ (%)	R_e
2	0.0	0.3955	0.4056	2.55	0.3983	0.71	3.59
	0.25	0.4181	0.4287	2.54	0.4210	0.69	3.68
	0.50	0.4943	0.5049	2.14	0.4976	0.67	3.19
	0.75	0.6568	0.6678	1.67	0.6600	0.49	3.41
1	0.0	0.2048	0.2153	5.13	0.2076	1.37	3.74
	0.25	0.2347	0.2456	4.64	0.2375	1.19	3.90
	0.50	0.3373	0.3471	2.91	0.3404	0.92	3.16
	0.75	0.5581	0.5678	1.74	0.5613	0.57	2.05

- The support domain is so chosen to select 6 nearest nodes.
- $e_{(41)} = (u_{41}^{\text{num}} - u^{\text{ref}}) \Big/ u^{\text{ref}}$, $e_{(145)} = (u_{145}^{\text{num}} - u^{\text{ref}}) \Big/ u^{\text{ref}}$, and $R_e = e_{(41)} \Big/ e_{(145)}$.

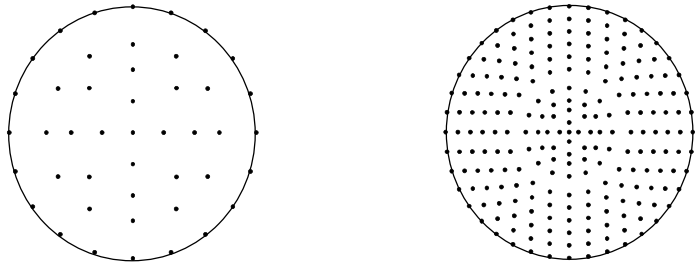
Table 6.6. Results of u for Example 6.3 obtained using the PPCM and different nodal distributions (k=100)

n	r	Reference solution	$u(r)$ (45 nodes)	Error(%)	$u(r)$ (221 nodes)	Error(%)
2	0.0	0.0689	0.0947	37.446	0.0813	18.00
	0.25	0.0783	0.1087	38.825	0.0923	17.88
	0.50	0.1258	0.1617	28.537	0.1376	9.38
	0.70	0.2480	0.2916	17.581	0.2363	-4.72
	0.90	0.5289	/	/	0.5423	2.53
1	0.0	3.55×10^{-4}	0.0015		4.21×10^{-4}	18.59
	0.25	0.0017	0.0040	135.294	0.0014	-17.65
	0.50	0.0097	0.0210	116.495	0.0111	14.43
	0.70	0.0599	0.1158	93.322	0.0673	12.35
	0.90	0.3884	/	/	0.4041	4.04

- The support domain is so chosen to select 6 nearest nodes.
- Error $= (u^{\text{num}} - u^{\text{ref}}) \Big/ u^{\text{ref}}$

When k=100, two nodal distributions of 45 nodes and 221 nodes shown in Figure 6.16 (a) and (b) are used. The results are listed in Table 6.6. Table 6.6 shows that the accuracy is significantly reduced due to the presence of the boundary layer. An improvement can be made by using finer nodes near the circular boundary to capture the sharp variation of the thin boundary layer.

This example clearly shows that the PPCM using the conventional PIM shape functions works well even for non-linear problems, as long as there is no DBC.



(a) 45 nodes (b) 221 nodes

Figure 6.16. Two models of different nodal distributions for the circle domain for the problem defined in Example 6.3 with k=100.

Example 6.4: Poisson's equation with derivative boundary conditions

Consider the following PDE defined in a square domain.

$$\nabla^2 u + u = (2 + 3x)e^{x-y}, \quad (x, y) \in [0,1] \times [0,1] \tag{6.80}$$

with the following fixed boundary conditions:

- Dirichlet boundary conditions

$$u(x, y)\big|_{x=0} = 0; \quad u(x, y)\big|_{y=0} = xe^x \tag{6.81}$$

- Derivative boundary conditions

$$\left.\frac{\partial u}{\partial x}\right|_{x=1} = 2e^{1-y}; \quad \left.\frac{\partial u}{\partial y}\right|_{y=1} = -xe^{x-1} \tag{6.82}$$

The exact solution for this problem is

$$u^{\text{exact}}(x, y) = xe^{x-y} \tag{6.83}$$

To show the effects of the presence of the DBCs, we conduct a study for this problem with and without the DBC. To study the problem without the DBC, the DBC in Equation (6.82) is replaced using the following Dirichlet boundary conditions to achieve the identical analytical solution.

$$u\big|_{x=1} = e^{1-y}; \quad u\big|_{y=1} = xe^{x-1} \tag{6.84}$$

Two models of regularly distributed 11×11 and 21×21 field nodes are used to represent the square domain. The results obtained using the PPCM for this problem with and without the DBC are listed in Table 6.7 for easy comparison. This table shows that the presence of the DBC increases the errors in the numerical results. This reconfirms that the DBC induce error in the PPCM methods.

Table 6.7. Errors in the numerical results obtained using PPCM with and without DBC (using 9 nodes in interpolation, p=2)

	11×11			21×21		
	e_0(%)	e_x(%)	e_y(%)	e_0(%)	e_x(%)	e_y(%)
Without DBC	0.67	3.00	6.84	5.4e-3	8.9e-2	4.9e-2
With DBC	1.82	3.24	7.60	0.20	0.16	0.39

We then study this problem with the DBC given in Equation (6.82); special treatment is needed to enforce the DBCs.

If there is no DB-node included in the support domain of a collocation node, the conventional polynomial PIM shape function discussed in Sub-section 3.2.1 is used. The matrix triangularization algorithm (MTA) (GR Liu, 2002) is used to avoid the singularity of the moment matrix. If there are DB-nodes included in the support domain, the Hermite-type WLS approximation discussed in Sub-section 3.2.1.3 is used to construct the shape functions. Because the derivatives of DB-nodes are already the DOFs in the Hermite-type approximation, the derivative boundary conditions are expected to be better enforced, as we discussed in the 1D problems.

For 11×11 regular nodes, there are 19 DB-nodes, and 19 additional DOFs of normal derivative. The errors of numerical solutions obtained for different weight coefficients β_j (see, e.g., Equation (3.63)) are listed in Table 6.8. In obtaining the results in Table 6.8, the dimensionless size of local support domain is chosen as α_s=1.5, and d_c=0.5 in Equation (3.39). For comparison, the direct collocation (DC) method based on the conventional PIM shape function (without Hermite-type approximation) is used to directly enforce the derivative boundary conditions, and results are also listed in Table 6.8. This table shows that the use of Hermite-type approximation can improve the accuracy of the solution especially for reducing error in the function value, e_0, if a large β (>10^3) is employed.

For 21×21 regular nodes, there are 39 DB-nodes and 39 additional DOFs for normal derivatives. The error results are listed in Table 6.9. The results show that when weight coefficients β_j in Equation (3.63) are chosen as 10^3 or 10^4, the accuracy is improved by the use of Hermite-type approximation.

Table 6.8. Errors in the numerical results obtained using the PPCM with different weight coefficients β (11×11 regular nodes, α_s=1.5, c=0.5, p=2)

β	e_0 (%)	e_x (%)	e_y (%)
1	4.90	4.47	10.72
10^1	3.36	3.78	8.76
10^2	2.08	3.39	7.54
10^3	1.15	3.23	7.02
10^4	0.90	3.22	6.94
10^5	0.86	3.22	6.93
without Hermite approximation	1.82	3.24	7.61

Table 6.9. Errors in the numerical results obtained using PPCM with different weight coefficients β (21×21 regular nodes, α_s=1.5, c=0.5, p=2)

β	e_0 (%)	e_x (%)	e_y (%)
1	1.36	1.06	2.37
10^1	1.04	0.81	1.79
10^2	0.47	0.39	0.77
10^3	0.12	0.13	0.18
10^4	0.08	0.09	0.28
10^5	0.11	0.11	0.33
without Hermite approximation	0.20	0.16	0.39

Table 6.8 and Table 6.9 show that the results obtained using DC method are acceptable because the regularly distributed nodes are used in the computation. If the irregular nodes are used, the solution of the DC method is inaccurate and often unstable[†]. In such cases, the collocation method with Hermite-type approximation gives better results.

Table 6.8 and Table 6.9 show that the finer nodal distribution can lead to a significant improvement on the accuracy of the numerical solutions both for the function values and the derivatives. The rate of improvement for the derivatives is better than that for the function values. This finding is true for both with or without Hermite-type approximation. This could be because of

[†] Results can change significantly, even with small changes in nodal locations.

the double positive effects of the refinement on the PDE approximation and the treatment of the DBCs.

6.6 RADIAL POINT COLLOCATION METHOD FOR 2D PROBLEMS

6.6.1 RPCM formulation

The radial point interpolation method (RPIM) was discussed in Subsection 3.2.2; the use of radial basis functions (RBFs) can overcome the singularity of the moment matrix in the PIM. In this section, a radial point collocation method (RPCM) is introduced. The procedures are largely the same as those introduced in Section 6.5, except that the PIM shape function is replaced by the RPIM shape function. Therefore, the detailed formulation is omitted to allow more rooms for the discussions of example problems solved using the RPCM with different ways to deal with the DBCs.

The material used in this section is largely based on the work by Liu X and GR Liu et al. (2002, 2003c,d).

6.6.2 RPCM for 2D Poisson equations

In this section, several examples are used to study numerically the performance of RPCM.

Example 6.5: Poisson's equation with derivative boundary conditions

The 2D Poisson's equation given by Equation (6.80) is considered again with the following mixed boundary conditions.

- Boundary condition I

  Dirichlet boundary conditions

$$u(x,y)\big|_{y=0} = xe^{x}; \quad u(x,y)\big|_{y=1} = xe^{x-1} \tag{6.85}$$

  DBCs

$$\left.\frac{\partial u}{\partial x}\right|_{x=0} = e^{-y}; \quad \left.\frac{\partial u}{\partial x}\right|_{x=1} = 2e^{1-y} \tag{6.86}$$

- Boundary condition II

Dirichlet boundary conditions:

$$u(x,y)\big|_{x=0} = 0\,; \quad u(x,y)\big|_{y=0} = xe^{x} \tag{6.87}$$

DBCs

$$\left.\frac{\partial u}{\partial x}\right|_{x=1} = 2e^{1-y}\,; \quad \left.\frac{\partial u}{\partial y}\right|_{y=1} = -xe^{x-1} \tag{6.88}$$

The exact solution for this problem is given in Equation (6.83).

The RPCM-Exp method is used to analyze this problem, and the shape parameter in Exp-RBF is chosen as α_c =1.0.

Regularly distributed 11×11 nodes are first used to represent the problem domain, and 9 nodes are used in the interpolation scheme is employed. The results obtained with two different shape functions: the conventional RPIM and the Hermite-type RPIM shape functions, are listed in Table 6.10. From this table, the relative errors of function using the conventional RPIM and Hermite-type RPIM are 20.08% and 0.30%, respectively. This demonstrates the fact that the Hermite-type interpolation significantly improves the accuracy of the solution, because it can enforce DBCs more accurately.

Table 6.10 Errors in the numerical results obtained using RPCM-Exp without polynomial and 121 regular nodes (9-node scheme and α_c=1.0)

	Boundary conditions I			Boundary conditions II		
	$e_0(\%)$	$e_x(\%)$	$e_y(\%)$	$e_0(\%)$	$e_x(\%)$	$e_y(\%)$
DC	8.47	8.53	36.92	20.08	16.10	40.32
HC	3.30	2.77	9.65	0.30	1.63	6.40

*DC: using the conventional RPIM; HC: using Hermite-type RPIM

A total of 121 irregularly scattered nodes is used to investigate the stability of the results obtained using the RPCM for an highly irregular nodal distribution. The numerical results are listed in Table 6.11 for boundary II; RPCM using the Hermite-type RPIM shape function is stable even for highly irregularly scattered nodes. Hermite-type RPIM shape functions can significantly improve the accuracy.

The influence of the size of the local support domain is studied and listed in Table 6.11. The solution obtained using RPCM-Exp is closer to the exact solution as the size of the support domain increases. This is true for both the conventional RPIM and the Hermite-type RPIM shape functions. For the former, the relative errors in the solution of function values are 14.98%,

3.15% and 0.25%, respectively, for the support domain of α_s=1.0, 1.5 and 2.0; for the latter, the relative errors of function values are 2.34%, 0.10% and 0.03%, respectively, for the support domain of α_s=1.0, 1.5 and 2.0.

Table 6.11. Errors in the numerical results obtained using RPCM-Exp without polynomial (α_c=1.0) and different sizes of the local support domain (using 121 highly irregular nodes; boundary condition II)

	DC			HC		
α_s	e_0(%)	e_x(%)	e_y(%)	e_0(%)	e_x(%)	e_y(%)
1.0	14.98	11.77	28.80	2.34	3.31	13.57
1.5	3.15	3.03	10.30	0.10	0.50	1.14
2.0	0.25	0.23	0.63	0.03	0.14	0.20

6.6.3 RPCM for 2D convection-diffusion problems

6.6.3.1 Steady state convection-diffusion problem

The 2D steady state convection-diffusion problems are governed by PDEs that are independent of time. They can be solved using the RPCM based on both the conventional RPIM shape functions and the Hermite-type RPIM shape functions.

Example 6.6: 2D steady state convection-diffusion problem

Consider a 2D problem governed by the following convection-diffusion PDE.

$$-\nabla\cdot(\mathbf{D}\nabla u)+\mathbf{v}\cdot\nabla u+\beta u=q(x,y),(x,y)\in\Omega=[0,1]\times[0,1] \tag{6.89}$$

where

$$\mathbf{D}=\begin{bmatrix}\epsilon & 0\\ 0 & \epsilon\end{bmatrix},\qquad \mathbf{v}=\{(3-x)\quad(4-y)\},\qquad \beta=1 \tag{6.90}$$

in which ϵ is a given diffusion coefficient. Two sets of boundary conditions are considered.

Boundary condition I

- Dirichlet boundary conditions:

$$u\Big|_{\substack{x=0\\x=1\\y=1}}=0 \tag{6.91}$$

- DBCs:

$$\left.\frac{\partial u}{\partial n}\right|_{y=0} = 0 \tag{6.92}$$

Boundary condition II:

- Dirichlet boundary conditions:

$$u\Big|_{\substack{x=0\\x=1\\y=0}} = 0 \tag{6.93}$$

- DBCs:

$$\left.\frac{\partial u}{\partial n}\right|_{y=1} = 0 \tag{6.94}$$

The exact solutions for these problem with these two types of boundary conditions are the same and given by

$$u^{\text{exact}} = \sin(x)\left(1-e^{-\frac{2(1-x)}{\epsilon}}\right)y^2\left(1-e^{-\frac{3(1-y)}{\epsilon}}\right) \tag{6.95}$$

A total of 41×41 regularly distributed nodes is first used to represent the problem domain. It is then solved using 1681 *randomly* distributed nodes shown in Figure 6.17 that are created using the Halton random model. The RPIM-Exp with the shape parameter α_c=10.0 is used in RPCM-Exp for both the regular nodal model and the random nodal model. The 9 nodes in the local support domains are applied to the regular model. The dimensionless size of support domain is chosen as α_s=2.0 for the random nodal model. Note that the total numbers of nodes used in both regular and irregular models are roughly the same.

The results obtained using RPCM-Exp with two different shape functions, the conventional RPIM (see Sub-section 3.2.2.1) and the Hermite-type RPIM, are listed in Table 6.12 and Table 6.13. It is seen that the Hermite-type RPIM shape functions give better results for large diffusion coefficients of 10.0, 1.0 and 0.10 for both models of regular and random nodal distributions.

Table 6.13 shows that the use of Hermite-type RPIM shape functions improves accuracy slightly, especially for large diffusion coefficients. This holds for both cases of boundary conditions when regular nodes are used.

Table 6.13 shows that the accuracy is not improved by the use of the Hermite-type RPIM shape functions for the case of boundary condition I, while the random nodal model is employed. For the random model with

boundary condition II, the use of the Hermite-type RPIM shape function gives better accuracy.

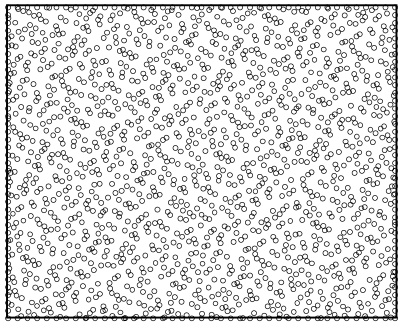


Figure 6.17. A total of 1681 randomly distributed nodes created using the Halton random model (interior nodes: 1521; boundary nodes: 160).

Table 6.12. Errors in the numerical results obtained using the RPCM-Exp without polynomial augmented (α_c=10.0) and 41×41 regular nodes (9-node interpolation scheme)

	Conventional RPIM shape function			Hermite-type RPIM shape function		
			Boundary condition I			
ϵ	e_0(%)	e_x(%)	e_y(%)	e(%)	e_x(%)	e_y(%)
10.0	5.04	5.41	3.89	0.97	1.80	2.56
1.0	4.38	4.97	4.85	0.46	1.86	4.67
0.1	2.55	17.13	28.63	0.95	17.52	28.71
0.01	28.12	72.58	79.86	27.87	72.50	79.85
			Boundary condition II			
ϵ	e_0(%)	e_x(%)	e_y(%)	e_0(%)	e_x(%)	e_y(%)
10.0	8.25	7.89	7.57	1.64	2.42	1.56
1.0	9.62	9.62	10.73	1.16	2.25	1.26
0.1	18.24	24.84	29.02	2.43	17.90	13.46
0.01	fail			fail		

ϵ: the diffusion coefficient.

Note that when ϵ is very small, the problem becomes convection dominated, for which the instability in the solution has been very well known for many numerical methods including the FDM and FEM, as discussed in Section 6.4. It can be concluded that the instability problem can be alleviated using enlarged local support domains or adaptive upwind support domains in the MFree method. Table 6.14 lists the results of

different sizes of support domains for the case of ϵ=0.01 that is a highly convection dominated case. To obtain the results in Table 6.14, RPCM-Exp with α_c=10.0 and 1681 regularly distributed nodes are used. This table shows that the accuracy of solution significantly improves with the enlargement of the local support domain. It confirms again that the enlargement of the support domain can help to stabilize the solution of a 2D convection dominated problem.

Table 6.13. Errors in the numerical results obtained using RPCM-Exp (α_c=10.0) and 1681 irregular nodes (the size of support domain: α_s=2.0)

	Conventional RPIM shape function			Hermite-type RPIM shape function		
			Boundary condition I			
ϵ	e_0(%)	e_x(%)	e_y(%)	e_0(%)	e_x(%)	e_y(%)
10.0	0.08	0.19	0.15	0.08	0.19	0.14
1.0	0.06	0.17	0.12	0.06	0.164	0.12
0.1	0.15	2.67	2.27	0.15	2.67	2.27
0.01	fail			fail		
			Boundary condition II			
ϵ	e_0(%)	e_x(%)	e_y(%)	e_0(%)	e_x(%)	e_y(%)
10.0	0.26	0.82	0.76	0.13	0.24	0.14
1.0	0.27	1.21	0.98	0.10	0.29	0.14
0.1	3.85	12.08	5.96	1.63	3.55	2.29
0.01	fail			fail		

ϵ: the diffusion coefficient.

Table 6.14. Errors in the numerical results obtained using RPCM-Exp (α_c=10.0) and 1681 regular nodes for ϵ=0.01

	Boundary condition I			Boundary condition II		
α_s	e_0(%)	e_x(%)	e_y(%)	e_0(%)	e_x(%)	e_y(%)
1.01	341.9	100.4	100.1	fail	fail	fail
1.45	27.8	72.5	79.8	fail	fail	fail
2.05	6.7	28.7	32.3	101.1	133.6	134.2
3.05	4.2	13.5	17.1	61.7	94.5	90.5

To study the efficiency of using the adaptive support domain discussed in Sub-section 6.4.4, this problem with small values of ϵ is analyzed. For simplicity, only the following Dirichlet boundary conditions are considered.

$$u\Big|_{\substack{y=0\\y=1}}^{x=0\\x=1} = 0 \tag{6.96}$$

The adaptive upwind support domain is defined by assuming the following formula

$$d_u = (\coth|\mathbf{Pe}| - 1/|\mathbf{Pe}|) \cdot d_s \tag{6.97}$$

where d_u is the central offset distance against the stream direction from the collocation node as shown in Figure 6.18, **Pe** is the vector of the local Peclet numbers, and d_s is the size of the local support domain.

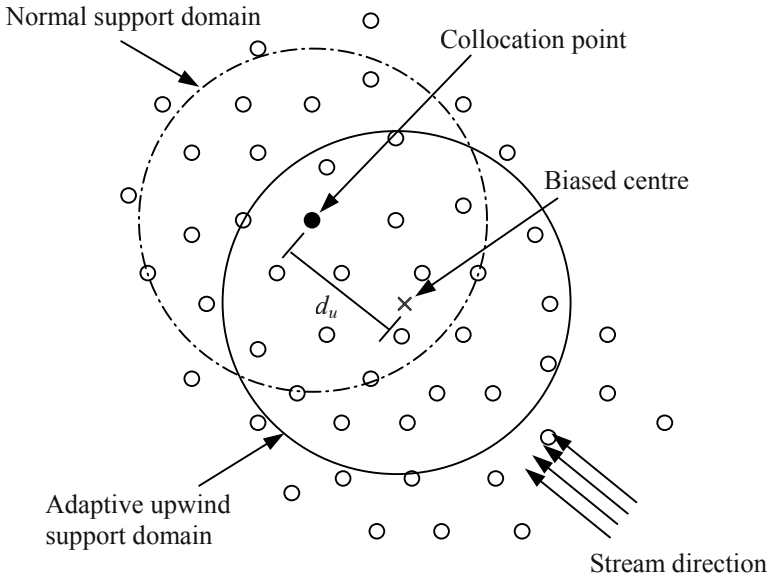


Figure 6.18. Construction of an adaptive upwind local support domain in a 2D problem domain using offset distance d_u.

Errors in the numerical results obtained using the RPCM-Exp with different ϵ are listed in Table 6.15. Table 6.15 shows that the adaptive upwind support domains can stabilize the solution, and gives the good results for this convection-dominated problem.

In summary, for a PDE with DBCs, the RPCM based on the Hermite-type RPIM shape functions produces better results than that based on the conventional RPIM shape functions, because the DBCs can be more accurately enforced using the Hermite-type interpolation. However, the use of the Hermite-type RPIM shape functions for 2D cases is not as effective as that observed for 1D cases. This may imply that the effects of the DBCs are more severe in multi-dimensional problems.

In addition, the instability problem in the 2D convection-dominated problems can be alleviated using enlarged local support domains and adaptive upwind support domains.

Table 6.15. Errors $e_0(\%)$ in the numerical results obtained using the RPCM-Exp (α_c=10.0) and 121 regular nodes for different c

	$c=10^{-2}$	$c=10^{-3}$	$c=10^{-4}$	$c=10^{-6}$
	$e_0(\%)$	$e_0(\%)$	$e_0(\%)$	$e_0(\%)$
Normal support domains	55.5	342.9	496.6	518.1
Adaptive support domains	2.0	2.1	2.1	2.1

* The size of local support domain: d_s=2.0d_c.

6.6.3.2 Linear dynamic convection-diffusion equations

Consider a 2D problem governed by the following time-dependent convection-diffusion equation.

$$L(u)=\rho\frac{\partial u}{\partial t}-\nabla^T(\mathbf{D}\nabla u)+\mathbf{v}^T\cdot\nabla u-q_A=0,\quad \text{in } \Omega \tag{6.98}$$

where **D** is defined in Equation (6.90), ρ and q_A are all given constants, **v** is the vector of velocities

$$\mathbf{v}^{\mathrm{T}}=\begin{Bmatrix} v_x & v_y \end{Bmatrix} \tag{6.99}$$

The following boundary and initial conditions are considered.

- DBC:

$$L_{\mathrm{DB}}(u)=\mathbf{n}^{\mathrm{T}}\mathbf{D}\nabla u+q_B=0,\quad \text{on } \Gamma_{\mathrm{DB}} \tag{6.100}$$

  where **n** is the unit outward normal vector on the boundary, and q_B is the specified source term on the derivative boundary.

- Dirichlet boundary condition:

$$u_i-\overline{u}_i=0,\quad \text{on } \Gamma_u \tag{6.101}$$

  where $\overline{u}$ is the specified u.

- Initial condition:

$$u(\mathbf{x},t)\big|_{t=0}=u^0(\mathbf{x}) \tag{6.102}$$

In Equation (6.98) the field variable u is a function of the spatial coordinate and time.

The problem domain is now represented using N_d internal (domain) nodes and $N_b = N_{\text{DB}} + N_u$ boundary nodes, which N_{DB} is the number of DB-nodes and N_u is the number of the nodes on the Dirichlet boundaries.

For the approximation of the field function u in the space coordinates, the conventional RPIM shape functions are used for all the internal nodes whose support domains do not include DB-nodes, and the Hermite-type RPIM shape functions are used for DB-nodes and internal nodes whose support domains include at least one DB-nodes, as we have done for the static problems. Using the RPCM, we can obtain a set of system equations at time t.

For the internal nodes and DB-nodes, we have

$$\rho \frac{\partial u_i^h}{\partial t} - R_i = 0, \quad i = 1, 2, \cdots, N_d \tag{6.103}$$

where u_{im}^h is the approximate value of u at node i and time t, and

$$R_i = \nabla^T (\mathbf{D} \nabla u_i^h) - \mathbf{v}^T \cdot \nabla u_i^h + q_A \tag{6.104}$$

For DB-nodes, we have in addition to Equation (6.103)

$$\mathbf{n}^T \mathbf{D} \nabla u_i^h + q_B = 0, \quad i = 1, 2, \cdots, N_{\text{DB}} \tag{6.105}$$

For nodes on the Dirichlet boundary Γ_u, we obtain

$$u_i^h - \overline{u}_i = 0, \quad i = 1, 2, \cdots, N_u \tag{6.106}$$

With the conventional and the Hermite-type RPIM shape functions, u_i^h can be approximated as

$$u_i^h(t) = \sum_{j=1}^{n} \phi_j(\mathbf{x}_i) u_j(t) \tag{6.107}$$

and its derivatives can be obtained by following equations:

$$\nabla u_i^h(t) = \sum_{j=1}^{n} \nabla \phi_j(\mathbf{x}_i) u_j(t) \tag{6.108}$$

where n is the number of nodes used in the local support domain. Note that in these equations, additional derivative DOFs are included.

The standard Crank-Nicolson scheme is used to perform the time integration, which leads to

$$\rho \frac{u_{m+1}^h - u_m^h}{\Delta t} - \frac{1}{2}(R_{m+1} + R_m) = 0 \tag{6.109}$$

where the subscript represents the time sequence, and Δt is the time step.

In the following, a numerical example will be used to study the performance of the RPCM for linearly dynamic convection-diffusion problems.

Example 6.7: Rotating Gaussian wave problem:

This problem is a special case of Equation (6.98) with coefficients given as $\mathbf{D} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $\rho = 1$, $q_A = 0$, $v_x = y$, and $v_y = -x$.

The problem domain is defined in a square domain of $\Omega = [-1, \ 1] \times [-1, \ 1]$, and the Dirichlet boundary conditions is given by on the entire problem boundary $\partial\Omega$:

$$u(\mathbf{x}, t) = 0, \qquad \mathbf{x} \in \partial\Omega \tag{6.110}$$

The initial condition is

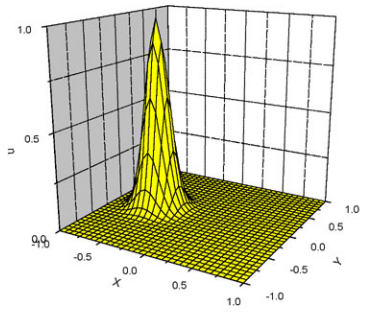
$$u(\mathbf{x}, t = 0) = e^{-\frac{r^2}{2\sigma_0^2}} \tag{6.111}$$

where $r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$ is the distance between a point at (x, y) and the initial position of the center at $(x_0, y_0) = (-0.5, 0.0)$, and $\sigma_0 = 1/8$ is the constant controlling the size of the Gaussian wave.

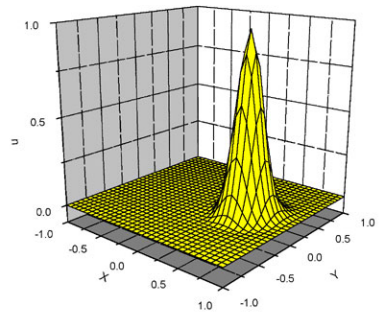
Two regular nodal distributions of 33×33 nodes and 65×65 nodes are used to represent the problem domain, and the conventional RPIM-MQ shape functions are used for the spatial discretization of the field variables. The dimension of the support domain of a collocation node to be processed is so chosen in order to select 9 or 25 nearest nodes in the support domain. Note that the results of RPIM-Exp can also be similarly obtained, although only the results of RPIM-MQ are presented here.

The elevations of the rotating Gaussian wave are plotted in Figure 6.19 and Figure 6.20. Figure 6.19 is obtained using 33×33 nodes, in the procedure of time integration, 100 time steps with time interval of $\Delta t = 2\pi/100$, and shape parameters of $d_c = 2/32, \alpha_c = 3.0$, and q=0.5 are employed. Figure 6.20 is obtained using 65×65 nodes model, 1000 time steps with time interval of $\Delta t = 2\pi/1000$, and shape parameters of $d_c = 2/64, \alpha_c = 3.0$, and q=0.5. To compare the accuracy of the various

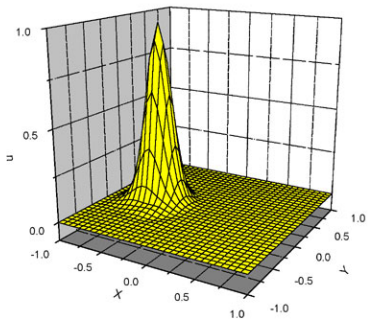
schemes, the maximum and minimum values of the computed solutions and the error by the present method are listed in Table 6.16~Table 6.18.



(a) Initial value, $t = 0$,
$e_0 = 0.00\%, u_{\max} = 1.0, u_{\min} = 0.0$



(b) Half revolution,
$t = \pi, e_0 = 4.6\%, u_{\max} = 0.98, u_{\min} = -0.0045$



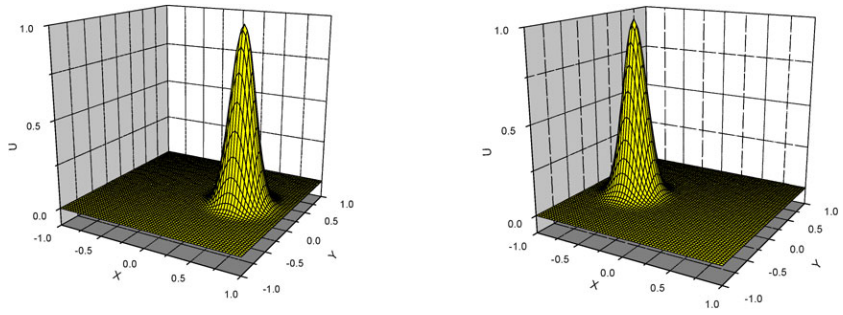
(c) Full revolution, $t = 2\pi, e_0 = 4.7\%, u_{\max} = 0.9876, u_{\min} = -0.0045$

Figure 6.19. The solution for the rotating Gaussian wave solved using RPIM-MQ shape functions and 33×33 regular nodes.

Table 6.16. Errors in the numerical results obtained using the RPCM with RPIM-MQ shape functions and 33×33 regular nodes with different shape parameters α_c (9 nearest nodes selected in the support domain)

Half revolution					
α_c	e_0(%)	e_x(%)	e_y(%)	u_{max}	u_{min}
3.0	31.42	48.77	67.54	0.8433	-0.1623
6.0	41.14	59.30	82.87	0.8279	-0.1650
Full revolution					
3.0	31.44	48.80	67.60	0.8435	-0.1624
6.0	41.18	59.34	82.94	0.8283	-0.1641

* The shape parameter used in MQ: q=0.5



(a) Half revolution, $t = \pi$, $e_0 = 3.40\%, u_{\max} = 0.99, u_{\min} = -0.0002$

(b) Full revolution, $t = 2\pi$, $e_0 = 3.47\%, u_{\max} = 0.99, u_{\min} = -0.0002$

Figure 6.20. The solution for the rotating Gaussian wave solved using RPIM-MQ shape functions and 65×65 regular nodes.

Table 6.17. Errors in the numerical results obtained using RPCM with RPIM-MQ shape functions and 33×33 regular nodes for different shape parameters α_c (25 nearest nodes selected in the support domain)

α_c	e_0(%)	e_x(%)	e_y(%)	u_{max}	u_{min}
			Half revolution		
3.0	4.68	7.80	10.45	0.9873	-0.0045
6.0	9.20	12.98	25.02	0.9569	-0.0172
			Full revolution		
3.0	4.68	7.80	10.45	0.9876	-0.0045
6.0	9.22	13.00	25.07	0.9571	-0.0173

* The shape parameter used in MQ: q=0.5

Table 6.18. Errors in the numerical results obtained using RPCM-MQ with RPIM-MQ shape functions and 65×65 regular nodes for different shape parameters α_c (25 nearest nodes selected in the support domain)

α_c	e_0(%)	e_x(%)	e_y(%)	u_{max}	u_{min}
			Half revolution		
3.0	3.41	2.99	5.15	0.9987	-0.00026
6.0	0.68	0.93	1.51	1.00039	-0.00001
			Full revolution		
3.0	3.47	3.01	5.18	0.9988	-0.00026
6.0	0.68	0.88	1.42	1.00050	-0.00001

* The shape parameter used in MQ: q=0.5

Note that the problem designed in Example 6.7 is a purely convection problem, which can clearly show the instability issue discussed in Section 6.4. We have reconfirmed from this example that the solution can be stabilized using denser nodes in the model and more nodes in creating the RPIM shape functions. The use of a large α_c in the RPIM-MQ can also improve the accuracy in the stabilized solution as shown in Table 6.18. This is because

1) The use of denser nodes produces better resolution for the Gaussian wave.
2) The use of more nodes in the local support domain allows the local convection effects farther.
3) A large α_c makes the RPIM shape functions have large values in the distant area, which also allows the convection effects farther.

Note that the better way to avoid the instability in the convection dominated problems is to use the adaptive upwind support domains discussed in Section 6.4 and Example 6.8. Another effective method to overcome the instability problem in the dynamic convection-dominated problem is the characteristic-based method that is widely used in the FEM (see, e.g., Zienkiewicz and Taylor, 2000), and the FPM (Oñate , 1996).

6.7 RPCM FOR 2D SOLIDS

Applying MFree strong-form methods (collocation methods) to solids mechanics problems is usually much more challenging, because of the presence of both normal and shear force boundary conditions. These force boundary conditions are all of derivatives types, and related to the derivatives of the displacements in both normal and tangential directions at a point on the derivative (or stress or traction) boundaries. This section will discuss a number of attempts performed by Liu X and GR Liu et al. (2002, 2003c,d,e) in dealing with those problems, when a collocation method is used.

6.7.1 Hermite-type RPCM

Consider a problem of two-dimensional elastostatics of isotropic materials. The standard governing equations (strong-form) for the plane stress case (Timoshenko and Goodier, 1970) defined in the *x*-*y* plane can be expressed explicitly as

$$\frac{E}{1-\mu^2}\left(\frac{\partial^2 u}{\partial x^2}+\frac{1-\mu}{2}\frac{\partial^2 u}{\partial y^2}+\frac{1+\mu}{2}\frac{\partial^2 v}{\partial x\partial y}\right)+b_x=0 \tag{6.112}$$

$$\frac{E}{1-\mu^2}\left(\frac{\partial^2 v}{\partial y^2}+\frac{1-\mu}{2}\frac{\partial^2 v}{\partial x^2}+\frac{1+\mu}{2}\frac{\partial^2 u}{\partial x\partial y}\right)+b_y=0 \tag{6.113}$$

where E and μ ν are Young's modulus and Poisson's ratio, u and v are the two components of the displacement in x and y directions, respectively, and b_x and b_y are the two components of the external body forces applied at x and y directions, respectively.

On the derivative (stress) boundaries, Γ_t, where the traction forces in two directions are specified, the stress boundary conditions are

$$\begin{cases} n_x\sigma_{xx}+n_y\tau_{xy}=\overline{t}_x \\ n_y\sigma_{yy}+n_x\tau_{xy}=\overline{t}_y \end{cases} \tag{6.114}$$

where σ_x, σ_y, and τ_{xy} are stress components, n_x and n_y are two components of the unit outward normal vector in x and y directions on the boundary Γ_t (see Figure 1.4), and $\overline{t}_x$ and $\overline{t}_y$ denote the prescribed tractions in x and y directions, respectively.

Substituting the expressions of stresses in terms of displacement components (see, e.g, Sub-section 1.2.2) into Equation (6.114), we can obtain the strong-forms of the stress boundary conditions in term of displacements:

$$\frac{E}{1-\mu^2}\left[n_x\left(\frac{\partial u}{\partial x}+\mu\frac{\partial v}{\partial y}\right)+n_y\frac{1-\mu}{2}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]=\overline{t}_x \tag{6.115}$$

$$\frac{E}{1-\mu^2}\left[n_y\left(\mu\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+n_x\frac{1-\mu}{2}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right]=\overline{t}_y \tag{6.116}$$

Equations (6.115) and (6.116) can be written in the following matrix form

$$\begin{Bmatrix}\overline{t}_x\\ \overline{t}_y\end{Bmatrix}=\frac{E}{1-\mu^2}\begin{bmatrix} n_x\frac{\partial}{\partial x}+n_y\frac{1-\mu}{2}\frac{\partial}{\partial y} & n_x\mu\frac{\partial}{\partial y}+n_y\frac{1-\mu}{2}\frac{\partial}{\partial x} \\ n_y\mu\frac{\partial}{\partial x}+n_x\frac{1-\mu}{2}\frac{\partial}{\partial y} & n_y\frac{\partial}{\partial y}+n_x\frac{1-\mu}{2}\frac{\partial}{\partial x}\end{bmatrix}\begin{Bmatrix}u\\ v\end{Bmatrix} \tag{6.117}$$

These equations show that the boundary conditions are given in terms of derivatives of the field variable (displacements) with respect to both coordinates. It is therefore a typical type of DBC for solids mechanics problems; is also termed as *natural* boundary conditions in the context of weak-form formulation (see also Section 1.5).

Equations (6.112), (6.113), (6.115) and (6.116) are for plane stress problems. For plane strain problems, the strong-forms of governing equations and stress or natural boundary conditions can be easily obtained from Equations (6.112), (6.113), (6.115) and (6.116) by replacing the Young's modules E with

$$\overline{E} = \frac{E}{1-\mu^2} \tag{6.118}$$

and the Poisson ratio ν with

$$\overline{\mu} = \frac{\mu}{1-\mu} \tag{6.119}$$

The displacement boundary conditions can be expressed as:

$$u = \overline{u}, \qquad \text{on } \Gamma_u \tag{6.120}$$

$$v = \overline{v}, \qquad \text{on } \Gamma_u \tag{6.121}$$

where $\overline{u}$ and $\overline{v}$ are the displacements in the x and y directions specified on the displacement boundaries. Equations (6.120) and (6.121) are the *essential* boundary conditions in the context of weak-form formulation (see also Section 1.5).

These equations are directly approximated using MFree shape functions, such as the MLS approximation, the RPIM, etc., and discretized by collocating at the field nodes. Note that special treatments are needed for the discretization of the equations of the stress (derivative) boundary conditions Equations (6.115) and (6.116). In this section, the Hermite-type RPIM discussed in Sub-section 3.2.2.2 is used for the implementation of the stress boundary conditions.

For the internal nodes or nodes on the displacement boundaries whose support domains do not include DB-nodes, the conventional RPIM shape functions are used. The displacement at a point **x** can be approximated as

$$\mathbf{u}(\mathbf{x}) = \mathbf{\Phi}^{\mathrm{T}}\mathbf{u}_s = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} \tag{6.122}$$

where $\mathbf{\Phi}$ is the matrix of the conventional RPIM shape functions, and $\mathbf{u}_s$ is the vector that collects all nodal displacements at the n nodes in the local support domain.

Substituting Equation (6.122) into Equations (6.112), (6.113), (6.115) and (6.116), we can obtain a set of discretized system equations for these nodes. Detailed procedures are similar to those presented in Sub-section 6.3.1 for 1D problems.

For DB-nodes and internal nodes whose support domains include DB-nodes, the displacement at a point **x** can be approximated using the Hermite-type RPIM shape functions (see Sub-section 3.2.2.2) to arrive at

$$\mathbf{u}(\mathbf{x}) = \mathbf{\Phi}^{\mathrm{T}}\mathbf{u}_s \tag{6.123}$$

where the matrix $\mathbf{\Phi}$ of shape functions is obtained using equations given in Sub-section 3.2.2.2, i.e.,

$$\mathbf{\Phi}^{\mathrm{T}} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 & \phi_1^H & 0 & \cdots & \phi_{n_{\mathrm{DB}}}^H & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n & 0 & \phi_1^H & \cdots & 0 & \phi_{n_{\mathrm{DB}}}^H \end{bmatrix} \tag{6.124}$$

In Equation (6.123), $\mathbf{u}_s$ is the vector that collects all nodal displacements at the n nodes in the local support domain and nodal normal derivatives of the DB-nodes on the stress boundaries in the support domain.

$$\mathbf{u}_s = \left\{ u_1 \quad v_1 \quad \cdots \quad u_n \quad v_n \quad \frac{\partial u_1^{DB}}{\partial n} \quad \frac{\partial v_1^{DB}}{\partial n} \quad \cdots \quad \frac{\partial u_{n_{DB}}^{DB}}{\partial n} \quad \frac{\partial v_{n_{DB}}^{DB}}{\partial n} \right\}^{\mathrm{T}} \tag{6.125}$$

The displacement at a collocation node **x** can be re-written as

$$\begin{Bmatrix} u(\mathbf{x}) \\ v(\mathbf{x}) \end{Bmatrix} = \sum_{i=1}^{n} \begin{bmatrix} \phi_i & 0 \\ 0 & \phi_i \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \end{Bmatrix} + \sum_{j=1}^{n_{DB}} \begin{bmatrix} \phi_j^H & 0 \\ 0 & \phi_j^H \end{bmatrix} \begin{Bmatrix} \dfrac{\partial u_j^{DB}}{\partial n} \\ \dfrac{\partial v_j^{DB}}{\partial n} \end{Bmatrix} \tag{6.126}$$

The governing Equations (6.112) and (6.113) are discretized at all DB-nodes and the internal nodes whose support domain contain DB-nodes using Equation (6.126) and the similar procedures presented in Sub-section 6.3.1.

For the DB-nodes, the stress boundary conditions, Equations (6.115) and (6.116), are also imposed using the Hermite-type RPIM shape functions. Substituting Equation (6.126) into Equation (6.117), we have

$$\begin{Bmatrix} \bar{t}_x \\ \bar{t}_y \end{Bmatrix} = \sum_{i=1}^{n} \frac{E}{1-\mu^2} \begin{bmatrix} n_{xi}\dfrac{\partial \phi_i}{\partial x} + n_{yi}\dfrac{1-\mu}{2}\dfrac{\partial \phi_i}{\partial y} & n_{xi}\mu\dfrac{\partial \phi_i}{\partial y} + n_{yi}\dfrac{1-\mu}{2}\dfrac{\partial \phi_i}{\partial x} \\ n_{yi}\mu\dfrac{\partial \phi_i}{\partial x} + n_{xi}\dfrac{1-\mu}{2}\dfrac{\partial \phi_i}{\partial y} & n_{yi}\dfrac{\partial \phi_i}{\partial y} + n_{xi}\dfrac{1-\mu}{2}\dfrac{\partial \phi_i}{\partial x} \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \end{Bmatrix} + \sum_{j=1}^{n_{DB}} \frac{E}{1-\mu^2} \begin{bmatrix} n_{xj}\dfrac{\partial \phi_j^H}{\partial x} + n_{yj}\dfrac{1-\mu}{2}\dfrac{\partial \phi_j^H}{\partial y} & n_{xj}\mu\dfrac{\partial \phi_j^H}{\partial y} + n_{yj}\dfrac{1-\mu}{2}\dfrac{\partial \phi_j^H}{\partial x} \\ n_{yj}\mu\dfrac{\partial \phi_j^H}{\partial x} + n_{xj}\dfrac{1-\mu}{2}\dfrac{\partial \phi_j^H}{\partial y} & n_{yj}\dfrac{\partial \phi_j^H}{\partial y} + n_{xj}\dfrac{1-\mu}{2}\dfrac{\partial \phi_j^H}{\partial x} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial u_j^b}{\partial n} \\ \dfrac{\partial v_j^b}{\partial n} \end{Bmatrix} \tag{6.127}$$

Using Equation (6.127), the force (derivative) boundary conditions can be implemented. Some numerical examples will be presented in the following.

Example 6.8: 2D Cook membrane's problem

The Cook membrane's problem shown in Figure 6.21 is solved using the radial point collocation method (RPCM). The membrane is subjected to a uniformly distributed shear load at the free end. The parameters are E=1, μ=1/3, and F=1. A reference solution for this problem was given by Hueck and Wriggers (1995) using FEM with a fine mesh of 128×128 elements. The vertical displacement at point C is found in the FEM analysis as v_c =23.96; the maximum principal stresses at the point A is σ_{Amax}=0.2369, and the minimum principal stresses at the point B σ_{Bmin}= –0.2035 .

The thin plate spline (TPS) RBF augmented with the constant term (m=1) is used in the construction of the Hermite-type RPIM shape functions. Two nodal distributions of 9×9 nodes and 17×17 nodes shown in Figure 6.22 are used. The numerical results obtained are plotted in Figure 6.23~Figure 6.25 for different sizes of local support domains. The solutions obtained using the finer nodal distribution are generally much better than those from the coarser nodal distributions. The results obtained using the RPCM converge with the increase of the size of the support domain. The accuracy of the solution is poor if α_s<4.5. Note that for most problems studied in this book or the book by GR Liu (2002), $\alpha_s \leq 3.0$ is usually used. It shows that solving solid mechanics problems using collocation methods is generally much more

demanding on the number of nodes used in the support domain, compared with the weak-form methods.

The convergence is not monotonic. This is a typical feature of the collocation methods or MFree strong-form method in general, in contrast to the methods based on the energy principles where the convergence is usually monotonic, and hence it is easier for error bound estimation. Finally, we mention, without showing detailed results, that if the direct RPCM is used for this problem, the results are poor. The use of the Hermite-type interpolation is essential for this problem due to the presence of the complex DBCs.

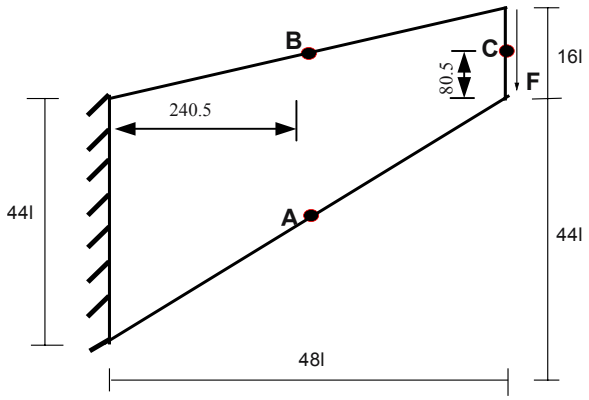
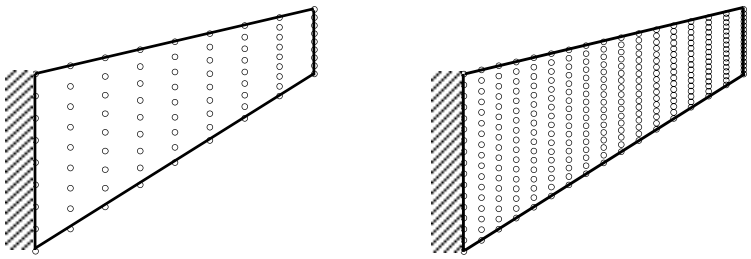


Figure 6.21. Cook's membrane problem.



(a) 9×9 nodes (b) 17×17 nodes

Figure 6.22. Two nodal distributions used in the RPCM.

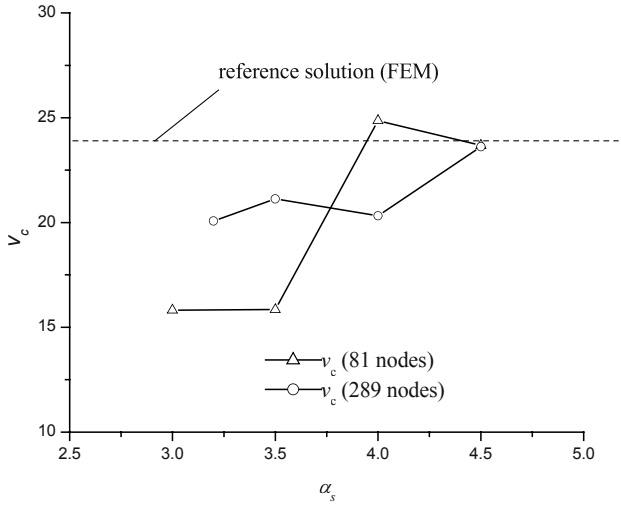


Figure 6.23. The vertical displacement at point C for the Cook's membrane problem solved using Hermite-type RPCM-TPS with shape parameter of η=4.

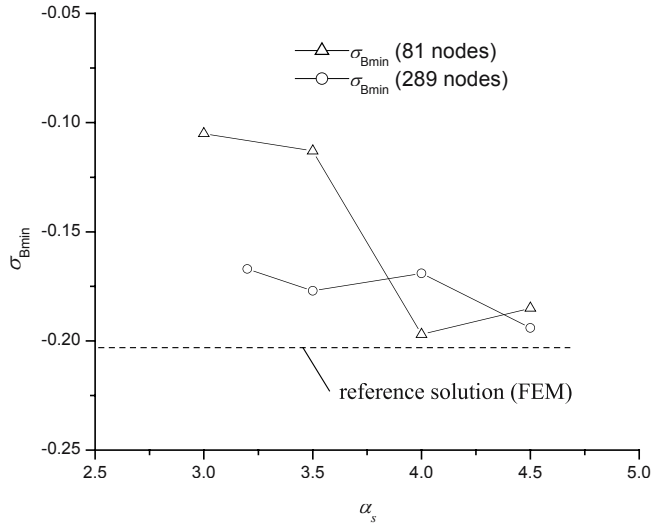


Figure 6.24. The minimum principal stress at point B for the Cook's membrane problem solved using Hermite-type RPCM-TPS with shape parameter of η=4.

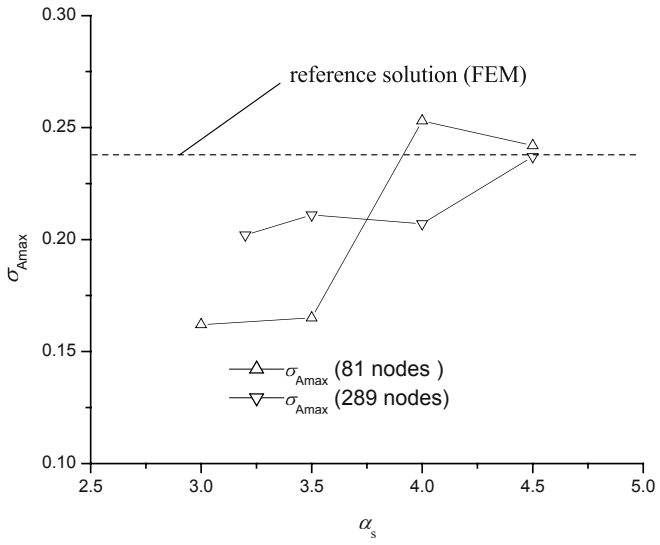


Figure 6.25. The maximum principal stress of at point A for the Cook's membrane problem solved using the Hermite-type RPCM-TPS with shape parameter of η=4.

6.7.2 Use of regular grid (RG)

FDM is successful for both fluid and solids mechanics problems with DBCs (e.g. Klerber, 1998). This is because regular grids are used, and a proper procedure for implementing the DBC can be formulated. This fact has motivated many to use FDM grids in the MFree methods on the derivative boundary to handle the DBCs in CFD problems (Liszka and Orkisz, 1977; Cheng and GR Liu, 1999, 2002). This section reports the attempts made by Liu X et al. (2003e) for solids mechanics problems using regular nodes on and near derivative boundaries. The conventional RPCM procedure is used for all the other nodes that may not be regularly distributed.

The force boundary conditions are given in Equations (6.115) and (6.116), and involve the 1st derivatives of both displacements. In order to approximate these derivatives, several layers nodes regularly distributed are placed along the force boundaries. The standard difference scheme used in FDM is then employed for these nodes to construct the discretized equations for these derivatives. The force boundary conditions are then enforced in a similar manner used in the conventional FDM, while other nodes in the problem domain can still be irregularly distributed.

Two types of boundary shapes are considered in this study: the straight boundary as shown in Figure 6.26 and the curve (circle) boundary as shown in Figure 6.27.

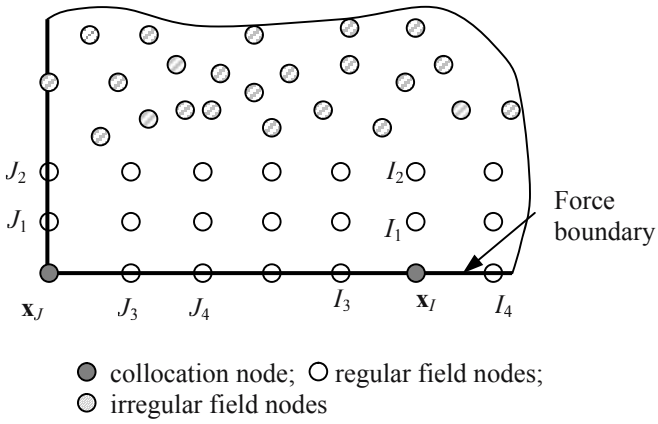


Figure 6.26. Regular grids used to approximate the DBCs on a straight boundary.

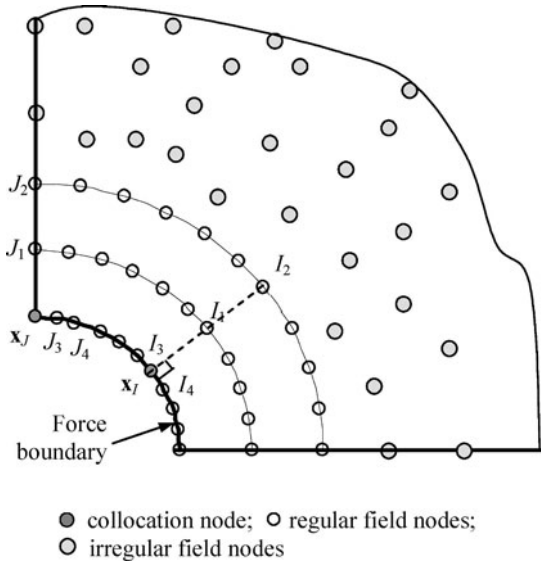


Figure 6.27. Regular grids used to approximate the DBCs on a curves boundary.

When the force boundary is straight, three layers of regular nodes are used, as shown in Figure 6.26.

- For a collocation node at $\mathbf{x}_I$ that is not at the corner, four nodes, I_1, I_2, I_3 and I_4, around $\mathbf{x}_I$ are used to approximate the 1st derivatives of displacements, u and v, using the standard difference scheme:

$$\frac{\partial u}{\partial x}=\frac{u_{I_4}-u_{I_3}}{x_{I_4}-x_{I_3}},\qquad \frac{\partial u}{\partial y}=\frac{-3u_I+4u_{I_1}-u_{I_2}}{y_{I_2}-y_I}$$
$$\frac{\partial v}{\partial x}=\frac{v_{I_4}-v_{I_3}}{x_{I_4}-x_{I_3}},\qquad \frac{\partial v}{\partial y}=\frac{-3v_I+4v_{I_1}-v_{I_2}}{y_I-y_{I_2}} \tag{6.128}$$

- For a collocation node at $\mathbf{x}_J$ that is at the corner, four nodes, J_1, J_2, J_3 and J_4, around $\mathbf{x}_J$ are used to approximate the 1st derivatives of displacements, u and v, using the standard difference scheme:

$$\frac{\partial u}{\partial x}=\frac{-3u_J+4u_{J_3}-u_{J_4}}{x_{J_4}-x_J},\qquad \frac{\partial u}{\partial y}=\frac{-3u_J+4u_{J_1}-u_{J_2}}{y_{J_2}-y_J}$$
$$\frac{\partial v}{\partial x}=\frac{-3v_J+4v_{J_3}-v_{J_4}}{x_{J_4}-x_J},\qquad \frac{\partial v}{\partial y}=\frac{-3v_J+4v_{J_1}-v_{J_2}}{y_{J_2}-y_J} \tag{6.129}$$

Substituting Equation (6.128) or (6.129) into Equations (6.115) and (6.116), we obtain a set of discretized equations for the force boundary conditions.

When the force boundary is a curve (e.g. circle), three layers of regular nodes are used, as shown in Figure 6.27.

1) For a collocation node at $\mathbf{x}_I$ that is not at the corner, four nodes, I_1, I_2, I_3 and I_4, around $\mathbf{x}_I$ are used to express the 1st derivatives of displacements, u and v, using the standard difference scheme in the polar coordinates:

$$\frac{\partial u}{\partial \theta}=\frac{u_{I_4}-u_{I_3}}{\theta_{I_4}-\theta_{I_3}},\qquad \frac{\partial u}{\partial r}=\frac{-3u_I+4u_{I_1}-u_{I_2}}{r_{I_2}-r_I}$$
$$\frac{\partial v}{\partial \theta}=\frac{v_{I_4}-v_{I_3}}{\theta_{I_4}-\theta_{I_3}},\qquad \frac{\partial v}{\partial r}=\frac{-3v_I+4v_{I_1}-v_{I_2}}{r_{I_2}-r_I} \tag{6.130}$$

2) For a collocation node $\mathbf{x}_J$ that is at the corner, four nodes, J_1, J_2, J_3 and J_4, around $\mathbf{x}_J$ are used to get the 1st derivatives of displacements, u and v, using the standard difference scheme in the polar coordinates:

$$\frac{\partial u}{\partial \theta}=\frac{-3u_J+4u_{J_3}-u_{J_4}}{\theta_{J_4}-\theta_J},\qquad \frac{\partial u}{\partial r}=\frac{-3u_J+4u_{J_1}-u_{J_2}}{r_{J_2}-r_J}$$
$$\frac{\partial v}{\partial \theta}=\frac{-3v_J+4v_{J_3}-v_{J_4}}{\theta_{J_4}-\theta_J},\qquad \frac{\partial v}{\partial r}=\frac{-3v_J+4v_{J_1}-v_{J_2}}{r_{J_2}-r_J} \tag{6.131}$$

For a circular boundary, the derivative for x and y can be obtained using the following equations of coordinate transformation (Reddy, 1993).

$$\begin{cases}\dfrac{\partial u}{\partial x}=\cos\theta\dfrac{\partial u}{\partial r}-\dfrac{\sin\theta}{r}\dfrac{\partial u}{\partial \theta}\\ \dfrac{\partial u}{\partial y}=\sin\theta\dfrac{\partial u}{\partial r}+\dfrac{\cos\theta}{r}\dfrac{\partial u}{\partial \theta}\end{cases} \tag{6.132}$$

$$\begin{cases}\dfrac{\partial v}{\partial x}=\cos\theta\dfrac{\partial v}{\partial r}-\dfrac{\sin\theta}{r}\dfrac{\partial v}{\partial \theta}\\ \dfrac{\partial v}{\partial y}=\sin\theta\dfrac{\partial v}{\partial r}+\dfrac{\cos\theta}{r}\dfrac{\partial v}{\partial \theta}\end{cases} \tag{6.133}$$

Substituting Equations (6.130)~(6.132) and (6.133) into Equations (6.115) and (6.116), we find a set of discretized equations for the force boundary conditions.

For all other nodes, the standard collocation scheme similar to that presented in Sub-section 6.3.1 is employed using the conventional RPIM shape functions.

For easy description, the above present method is terms as RPCM-RG in the following numerical examples.

Example 6.9: An infinite plate with a circular hole

Consider a plate with a central circular hole of $x^2+y^2\le a^2$ at the centre subjected to a unidirectional tensile load of 1.0 in the x-direction as shown in Figure 6.28. Due to symmetry, only the upper right quadrant of the plate is modelled (see, Figure 6.29). Symmetry conditions are imposed on the left and bottom edges. The inner boundary is traction free. Plane strain conditions are assumed, and the material constants are $E=1.0\times10^3$, and $\mu=0.3$.

When $b/a\ge5$, the solution of the finite plate is very close to that of the infinite plate (Roark and Young, 1975), for which there is an analytical solution:

$$u_r = \frac{\sigma}{4\eta}\left\{ r\left[\frac{\kappa-1}{2}+\cos 2\theta\right] + \frac{a^2}{r}\left[1+(1+\kappa)\cos 2\theta\right] - \frac{a^4}{r^3}\cos 2\theta \right\} \tag{6.134}$$

$$u_\theta = \frac{\sigma}{4\eta}\left[(1-\kappa)\frac{a^2}{r} - r - \frac{a^4}{r^3}\right]\sin 2\theta \tag{6.135}$$

where

$$\eta = \frac{E}{2(1+\mu)} \qquad \kappa = \begin{cases} 3-4\mu & \text{plane strain} \\ \dfrac{3-\mu}{1+\mu} & \text{plane stress} \end{cases} \tag{6.136}$$

The analytical solution for the stresses of an infinite plate is

$$\sigma_x(x,y) = 1 - \frac{a^2}{r^2}\left\{\frac{3}{2}\cos 2\theta + \cos 4\theta\right\} + \frac{3a^4}{2r^4}\cos 4\theta \tag{6.137}$$

$$\sigma_y(x,y) = -\frac{a^2}{r^2}\left\{\frac{1}{2}\cos 2\theta - \cos 4\theta\right\} - \frac{3a^4}{2r^4}\cos 4\theta \tag{6.138}$$

$$\sigma_{xy}(x,y) = -\frac{a^2}{r^2}\left\{\frac{1}{2}\sin 2\theta + \sin 4\theta\right\} + \frac{3a^4}{2r^4}\sin 4\theta \tag{6.139}$$

where (r,θ) are the polar coordinates with θ measured counter-clockwise from the positive x axis.

The displacement boundary conditions are given by

$$u = 0\,, \text{ on the edge of } x=0 \tag{6.140}$$

$$v = 0\,, \text{ on the edge of } y=0 \tag{6.141}$$

which ensure the symmetry of the problem.

Traction (derivative) boundary conditions given by the exact solution Equations (6.137)~(6.139) are imposed on the right (x=5) and top (y=5) edges. Clearly, this problem has more complex traction (natural) boundary conditions.

A total of 165 nodes are used to represent the problem domain, and the nodal arrangement is shown in Figure 6.29. The displacements obtained by RPCM-RG based on the RPIM-MQ shape functions and the analytical methods are almost identical. As the stresses are more critical for accuracy

assessment, detailed results of stresses distribution for stress σ_{xx} along x=0 computed using the RPCM-RG with different parameters are shown in Figure 6.30. Figure 6.30 shows that the RPCM-RG method yields satisfactory results even for stresses for these sets of shape parameters used. Note that these sets of good shape parameters are not the same as those found in other examples. In addition, we have found again that when the RG method is not used, the RPCM failed to give a reasonable result.

For comparison, results obtained using EFG and RPIM methods (see, Chapter 4) are also plotted in Figure 6.30. It is shown that EFG and RPIM produce more accurate stresses than the RPCM-RG method. This confirms that a weak-form method is usually more accurate than a strong-form method. It should be noted that in the weak-form methods, EFG and RPIM, no special treatment is needed to enforce the force (derivative) boundary conditions. The solutions obtained using these weak-form methods are much less sensitive to the shape parameters, and wide ranges of good shape parameters have been found and shared with many types of problems (GR Liu, 2002). However, a background mesh is needed for the numerical integrations.

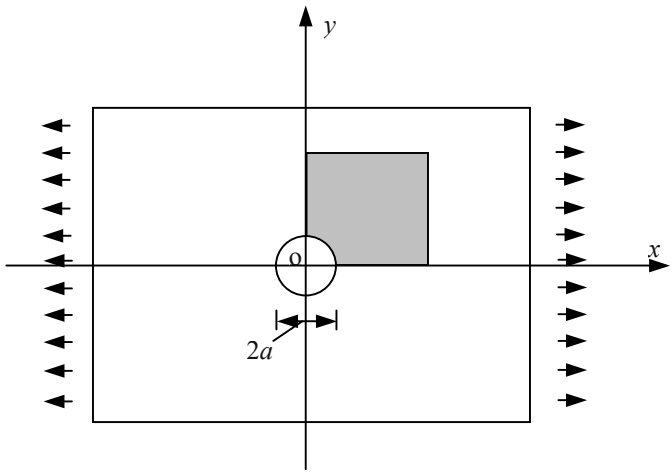


Figure 6.28. A 2D solid with a central hole subjected to a unidirectional tensile load. Only the upper right quadrant with dimension of 5×5 of the plate is modeled.

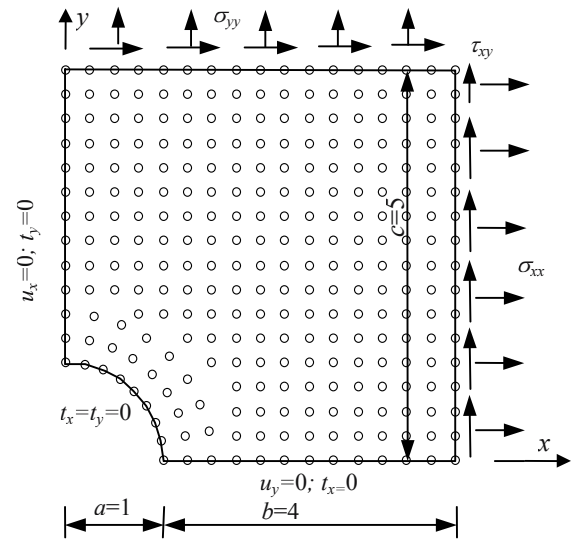


Figure 6.29. One quarter model of nodes and boundary conditions in a finite 2D solid with a central hole subjected to a unidirectional tensile load in the x direction.

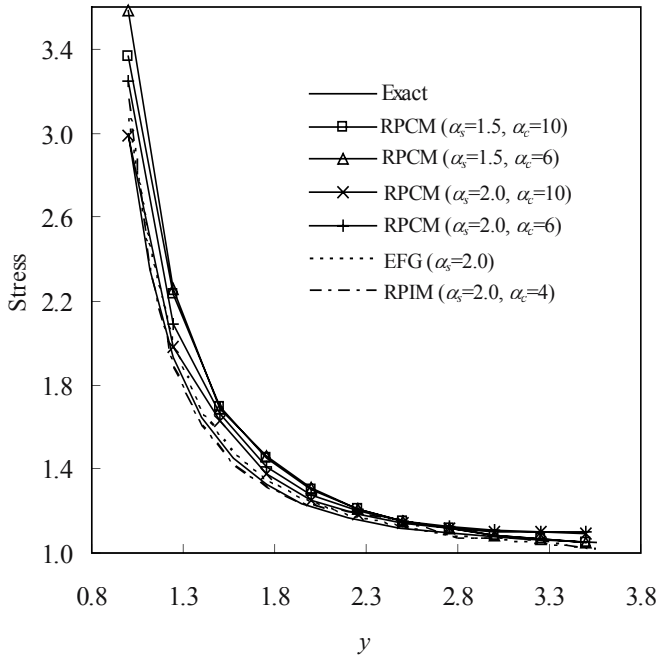


Figure 6.30. Normal stress (σ_{xx}) of the plate along the section of x=0 obtained using different methods (q=0.5, m=0).

6.8 REMARKS

In this Chapter, the MFree strong-form methods are discussed. Several numerical examples are presented to study the performance and efficiency. The strong-form method is a truly meshless method; there is no mesh for function approximation or the numerical integration. It is usually efficient, especially there are no DBCs.

However, the major technical issue for the strong-form methods is the enforcement of the DBCs. The error induced from the DBCs cannot be effectively controlled. This often leads to instability and. Several strategies were introduced to treat the DBCs, and these methods were examined in numerical examples. We draw the following conclusions:

1) The collocation method is able to reproduce the exact solution if it is included in MFree shape functions (see, Section 1.4).

2) The direct collocation (DC) method is simple and straightforward, and it is efficient for problems with only Dirichlet boundary conditions. The DC method is usually unstable or inaccurate for problems with DBCs.

3) The method using fictitious points (FP) can usually obtain satisfactory results. However, it increases the number of equations and needs additional meshing work.

4) The Hermite-type collocation (HC) method based on the Hermite-type shape functions is effective to enforce DBCs when a set of properly tuned parameters are used for a given problem. However, it increases the number of degree of freedom and thus increases the computational cost. The HC method works well for 1D problems, but not so consistently well for 2D problems.

5) Although the method of using regular grid (RG) nodes can more or less effectively handle the DBCs, additional meshing work is needed and it is difficult to use for some problems. In addition, the coding for the equations of DBCs is troublesome.

6) A good method for 1D problems may not be good for 2D problems. In general, 2D problem with derivative boundary conditions are more difficult to handle. The situation may be even worse for 3D problems.

7) The use of denser nodes on the derivative boundaries often improves the solution.

8) One of the most critical problems in use of the collocation methods is the poor robustness. A method tuned for one problem, may not work for others, and one set of parameters tuned for one problem may not work well for others. This unfortunate feature of collocation methods or MFree strong-form methods in general has not been observed in MFree weak-form methods.

In summary, there is still no way to totally solve the DBC issue in strong-form methods.

MFree method has clear advantages in handling instability issues in the convection-diffusion problems. Our study has found that

- Due to the overlap feature in the MFree interpolations /approximations, the solution for convection dominated problems is general stable and convergent as long as the shape function is of 2nd order or higher, meaning that the solution approaches the exact solution when the nodes are refined.
- The solution can be further improved by using more nodes in the interpolation/approximations; this can be done without any technical difficulty in an MFree method, in which the nodes used in the support domain are not prefixed and can be selected as desired manner.
- More accurate solutions can be obtained using adaptive upwind support domains.

Chapter 7

MESHFREE METHODS BASED ON COMBINATION OF LOCAL WEAK-FORM AND COLLOCATION

7.1 INTRODUCTION

MFree methods fall into three categories (Chapter 2): MFree collocation methods (or MFree strong-form methods), discussed in Chapter 6; MFree weak-form methods, such as the RPIM method, the EFG method, LRPIM method and the MLPG method, discussed in Chapters 4 and 5; MFree methods based on the combinations of both the strong-form and the weak-form or short for MFree weak-strong form method.

An MFree weak-strong (MWS) form method was proposed recently by GR Liu and Gu (2002d); it aimed to remove the background mesh for integration as much as possible, and yet to obtain stable and accurate solutions even for PDEs with derivative boundary conditions. The MWS method has been successfully developed and used in solid mechanics (Gu and GR Liu, 2005; GR Liu and Gu, 2003b) and fluid mechanics (GR Liu and Wu et al., 2004; GR Liu and Gu et al., 2003c).

This chapter is devoted entirely to MWS. Justification and motivation precede the formulation, implementation and coding issues. The convergence of the MWS method is studied numerically by comparison with other methods. Finally, examples from elastostatics, elastodynamics and fluid mechanics are presented to illustrate its efficiency, accuracy, and robustness.

7.2 MESHFREE COLLOCATION AND LOCAL WEAK-FORM METHODS

The MWS method is designed to combine the advantages of strong-form and weak-form methods and to avoid their shortcomings. This can be performed only after a thorough examination of the features of both types of methods, presented in the following two sub-sections.

7.2.1 Meshfree collocation method

The MFree strong-form methods were discussed in detail in Chapter 6, where the strong-forms of the governing equations and boundary conditions are discretized simply by collocation techniques. The MFree strong-form methods possess the following attractive advantages:

- They are truly meshless.
- The procedure is straightforward, and the algorithms and coding are simple, when there are only Dirichlet boundary conditions.
- They are computationally efficient, and the solution is accurate when there are only Dirichlet boundary conditions.

However, MFree strong-form methods have disadvantages:

- They are often unstable and less accurate, especially for problems governed by PDEs with derivative boundary conditions.
- Derivative boundary conditions (DBCs) involve a set of separate differential equations defined on the boundary; these are different from the governing equations defined in the problem domain. These DBCs require special treatments.
- Unlike integration, which is a smoothing operator, differentiation is a roughening operator; it magnifies errors in an approximation. This magnified error is partially responsible for the instability of the solution of PDEs (see discussions in Section 6.1). Hence, MFree strong-form methods are often unstable. Special treatments such as those discussed in Chapter 6 are employed to implement the derivative boundary conditions in MFree strong-form methods. However, such treatments cannot always control the error. As demonstrated in Chapter 6, a technique suitable for one problem may not work for another, even one of the same types. A set of parameters tuned for one problem may not work for another.

7.2.2 Meshfree weak-form method

MFree weak-form methods, such as the element-free Galerkin (EFG) method, the radial point interpolation method (RPIM), the meshless local Petrov-Galerkin method (MLPG), and the local radial point interpolation method (LRPIM), were discussed in detail in Chapters 4 and 5. The common feature of MFree weak-form methods is that the PDE (strong-form) of a problem is first replaced by or converted into an integral equation (global or local) based on a principle (weighted residual methods, energy principle etc.). Weak-form system equations can then be derived by integration by parts (see, Chapters 4 and 5).

A set of system equations of MFree weak-form methods can be obtained from the discretization of the weak-form using meshfree interpolation techniques.

There are four features of the local weak-form (see, Chapter 5).

1) The integral operation can smear the error over the integral domain and, therefore improve the accuracy in the solution. It acts like some kind of regularization to stabilize the solution.
2) The requirement of the continuity for the trial function is reduced or weakened, due to the order reduction of the differential operation resulting from the integration by parts.
3) The force (derivative) boundary conditions can be naturally implemented using the boundary integral term resulting from the integration by parts.
4) The system equations in the domain and the derivative boundary conditions are conveniently combined into one single equation.

These features give MFree weak-form methods the following advantages.

- They exhibit good stability and excellent accuracy for many problems.
- The traction (derivative) boundary conditions can be naturally and conveniently incorporated into the same weak-form equation. No additional equations or treatments are needed and no errors are introduced in the enforcement of traction boundary conditions.
- A method developed properly using a weak-form formulation is applicable to many other problems. A set of parameters tuned for one method for a problem can be used for a wide range of problems. This robustness of the weak-form methods have been demonstrated through many practical problems. It is this robustness that makes the weak-form methods applicable to many practical engineering problems.

However, MFree global weak-form methods are meshfree only in terms of the interpolation of the field variables. Background cells have to be used to integrate a weak-form over the global domain. The numerical integration makes them computationally expensive, and the background mesh for the integration means that the method is not truly meshless. To remove the global integration background mesh, methods based on the local Petrov-Galerkin weak-forms have been proposed, such as the meshless local Petrov-Galerkin (MLPG) method discussed in Chapter 5, the local boundary integral equation (LBIE) method (Zhu et al., 1998,1999), the method of finite spheres (De and Bathe, 2000), the local point interpolation method (LPIM) (Liu and Gu, 2001b), the local radial PIM (LRPIM) that developed based on the idea of MLPG, etc.

In the MFree local weak-form methods, the local integral domain in the interior of the problem domain is usually of a regular shape. It can be as simple as possible and can be automatically constructed in the process of computation. The MFree local weak-form methods have obtained satisfactory results in solid mechanics and fluid mechanics (Atluri and Shen, 2002; GR Liu, 2002).

Although the MFree local weak-form methods made a significant step in developing ideal meshfree methods, the numerical integration is still burdensome, especially for nodes on or near boundaries with complex shape. The local integration can still be computationally expensive for some practical problems. It is therefore desirable to minimize the need for numerical integrations.

7.2.3 Comparisons of Meshfree collocation and weak-form methods

Both MFree strong-form methods and MFree local weak-form methods have their own advantages and shortcomings, as discussed in Sub-sections 7.2.1 and 7.2.2, and they are largely *complementary*. Therefore, their proper combination could be beneficial.

Close comparison of the MFree strong-form methods and the MFree local weak-form methods reveals the following facts.

1) The implementation schemes of these two types of MFree methods are similar. They all construct the discretized equations one-by-one based on the field nodes, and the system equation is assembled (*stacked*) in a node-by-node manner. This is different from the MFree global weak-form methods, in which the discretized equations are constructed and assembled based on the integration cells and the quadrature points.

2) If the delta function is used as the weight function in MFree local weak-form methods, the MFree local weak-form method becomes an MFree strong-form method. This can be easily demonstrated as follows.

   Let the weight function be

$$\widehat{W}_I = \delta(\mathbf{x} - \mathbf{x}_I) \tag{7.1}$$

   The local weak-form becomes

$$\int_{\Omega_q} \delta(\mathbf{x} - \mathbf{x}_I)(\sigma_{ij,j} + b_i)\mathrm{d}\Omega = 0 \tag{7.2}$$

   The property of the delta function leads to

$$\sigma_{ij,j}(\mathbf{x}_I) + b_i(\mathbf{x}_I) = 0 \tag{7.3}$$

   This is exactly the discretized strong-form equations or the collocation formulation for node *I*.

3) In the MFree strong-form method, instability and computational error are mainly produced by the presence of DBCs. In the weak form methods, by contrary, DBCs can be easily and accurately enforced by using a sufficiently large local integral domain.

4) In the MFree strong-form method, the essential boundary conditions can be imposed conveniently and accurately. In the MFree local weak-form method, however, the essential boundary conditions require special treatments (such as the penalty method or Lagrange multiplier method) when MLS shape functions are used.

5) The number of field nodes on or near the derivative boundary is much less than that of the internal nodes plus the nodes on the essential boundaries. In the MFree local weak-form method, most computational cost for numerical integrations comes from the integration for internal nodes and the nodes on the essential boundary.

7.3 FORMULATION FOR 2-D STATICS

7.3.1 The idea

Consider the two-dimensional solid mechanics problem with a problem domain Ω shown in Figure 7.1. The problem domain and boundaries are represented by sets of irregular field nodes. The key idea of MWS is that in

establishing the discretized system equations, both the strong-form and the local weak-form are used for the same problem but for different sets of nodes.

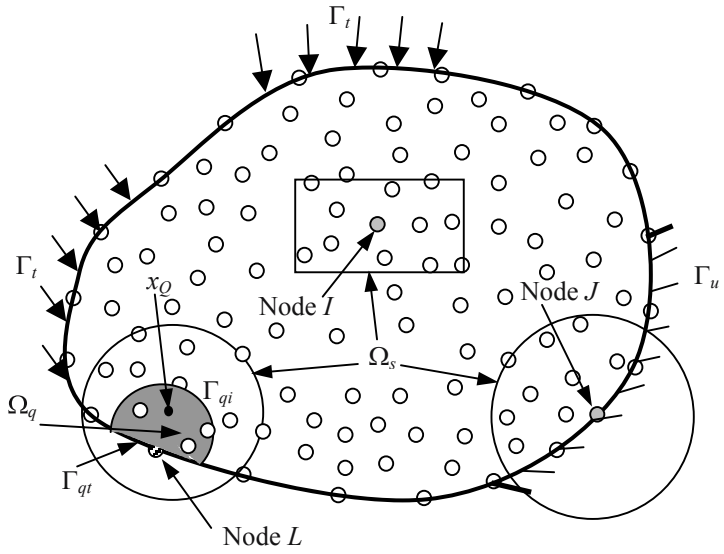


Figure 7.1. Concept of the MWS method: the local Petrov-Galerkin weak-form is used for the field nodes (e.g. the Lth node) that are on or near the derivative boundaries. Strong-form is used for all the rest nodes (e.g. the Ith and Jth nodes). Ω_s is the local support domain. Ω_q is the local quadrature domain.

For a field node, a simple quadrature domain (see Section 5.2) is defined in Figure 7.1, where Ω_q denotes the local quadrature domain for the field node. For a node whose quadrature domain Ω_q does not intersect with the global derivative boundaries Γ_t, the strong-form (collocation) is used. Otherwise, the local Petrov-Galerkin weak-form is used.

In MWS, for all the nodes whose local quadrature domains do not intersect with derivative boundaries, no numerical integrations are needed. The local integrations are needed only for the few nodes on or near the derivative boundaries. The derivative boundary conditions can then be easily imposed together with the system equation to produce stable and accurate solutions. MFree interpolation techniques that have been discussed in Chapter 3 can be used in the weak-strong-form. The detailed formulation will be presented in the following section.

For convenience of description, we define *DBR-nodes* and *collocatable nodes*. A DBR-node is a node on a problem boundary, on which the derivative (natural) boundary conditions are specified, or an internal node whose local quadrature domain intersects with the derivative boundaries.

DBR stands for Derivative Boundary Related. A collocatable node is an internal node that is not a DBR-node or a node on the essential boundaries.

In summary, the strategy of the MWS method is that the local weak-forms are used to establish discretized system equations for all the DBR-nodes and for collocatable nodes, the strong-form of PDEs will be directly discretized by collocation using MFree shape functions.

7.3.2 Local weak-form

The local weak-form is used for all the DBR-nodes. We use the local Petrov-Galerkin weak-form of the governing equations for 2D solids presented in Sections 5.2 and 5.3. The local weak-form for *I*th node can be written as

$$\int_{\Omega_q} \widehat{W}_I(\sigma_{ij,j} + b_i)\mathrm{d}\Omega = 0 \tag{7.4}$$

where $\widehat{W}$ is the weight function. The Petrov-Galerkin weak-form was used by Atluri et al. (1999b) to formulate the MLPG method, as detailed in Section 5.3. Equation (7.4) is different from Equation (5.34) where there is an additional term for imposing essential boundary conditions. In MWS, however, the Petrov-Galerkin weak-form is used only on DBR-nodes where there is no essential boundary condition.

The first term on the left hand side of Equation (7.4) can be integrated by parts. The boundary Γ_q for the local quadrature domain usually comprises three parts: the internal boundary Γ_{qi}, the essential boundary Γ_{qu} and the derivative boundary Γ_{qt}; Equation (7.4) becomes

$$\int_{\Omega_q} \widehat{W}_{I,j}\sigma_{ij}\mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{W}_I t_i \mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{W}_I t_i \mathrm{d}\Gamma = \int_{\Gamma_{qt}} \widehat{W}_I \overline{t}_i \mathrm{d}\Gamma + \int_{\Omega_q} \widehat{W}_I b_i \mathrm{d}\Omega \tag{7.5}$$

Equation (7.5) is the local Petrov-Galerkin weak-form to be used in MWS. Equation (7.5) shows that the derivative (or traction) boundary conditions have been incorporated naturally into the local weak-form of the system equation. No additional equation for derivative boundary conditions is needed.

The test (weight) function plays an important role in the performance of the local weak-form. For simplicity, the test functions are selected such that they vanish over Γ_{qi}. This can be easily done using the weight functions given in Chapter 3, such as the 4th-order spline weight function (W2 given in Equation (3.149)). Hence, Equation (7.5) can be simplified because the integration along the internal boundary Γ_{qi} vanishes. We therefore have the following local weak-form for all the DBR-nodes.

$$\int_{\Omega_q} \widehat{W}_{I,j}\sigma_{ij}\mathrm{d}\Omega - \int_{\Gamma_{qu}} \widehat{W}_I t_i \mathrm{d}\Gamma = \int_{\Gamma_{qt}} \widehat{W}_I \overline{t}_i \mathrm{d}\Gamma + \int_{\Omega_q} \widehat{W}_I b_i \mathrm{d}\Omega \tag{7.6}$$

7.3.3 Discretized system equations

As shown in Figure 7.1, the global problem domain Ω is represented by a set of irregularly distributed field nodes. Using the MLS or RPIM shape functions, we can have

$$\mathbf{u}^h_{(2\times 1)}(\mathbf{x}) = \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} = \mathbf{\Phi}_{(2\times 2n)}\mathbf{u}_{(2n\times 1)} \tag{7.7}$$

where n is the number of nodes in the support domain of a sampling point $\mathbf{x}$. $\mathbf{\Phi}$ is the matrix of shape functions. The sample point is a field node when the strong-form is used, and it is a Gauss point when the local weak-form is used. Note that these n field nodes are numbered from 1 to n, and it is a local numbering system for these field nodes used in the support domain. The field node has also a global number that is uniquely given to all field nodes from 1 to N. This global numbering system is used to assemble all the local nodal matrices together to form the global matrix. Hence, an index is needed to record the global number for a field node used in the support domain.

With Equation (7.7) and the equations given in Sub-section 1.2.2, the product of $\mathbf{L}\mathbf{u}^h$, which gives the strains, becomes

$$\boldsymbol{\varepsilon}_{(3\times 1)} = \mathbf{B}_{(3\times 2n)}\mathbf{u}_{(2n\times 1)} \tag{7.8}$$

where $\mathbf{B}$ is the *strain matrix* given in Sub-section 4.2.1. The stress vector can be written as

$$\boldsymbol{\sigma}_{(3\times 1)} = \mathbf{D}_{(3\times 3)}\boldsymbol{\varepsilon}_{(3\times 1)} = \mathbf{D}_{(3\times 3)}\mathbf{B}_{(3\times 2n)}\mathbf{u}_{(2n\times 1)} \tag{7.9}$$

where $\mathbf{D}$ is the matrix of elastic constants that is defined in Sub-section 1.2.2 for the plane stress problem and the plane strain problem. Substituting Equations (7.7)~(7.9) into strong-form of the Equation (1.31), we can obtain

$$\underbrace{\begin{bmatrix} \dfrac{\partial}{\partial x} & 0 & \dfrac{\partial}{\partial y} \\ 0 & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix}}_{\mathbf{L}^T} \mathbf{D} \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} \phi_1 & 0 & \cdots & \phi_n & 0 \\ 0 & \phi_1 & \cdots & 0 & \phi_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} + \underbrace{\begin{Bmatrix} b_x(\mathbf{x}_I) \\ b_y(\mathbf{x}_I) \end{Bmatrix}}_{\mathbf{b}_I^s} = \mathbf{0} \tag{7.10}$$

This can be written in the matrix form

$$\underbrace{\mathbf{L}^{\mathrm{T}}{}_{(2\times3)}\mathbf{D}_{(3\times3)}\mathbf{L}_{(3\times2)}\boldsymbol{\Phi}(\mathbf{x}_I)_{(2\times2n)}}_{\mathbf{K}^s_{I(2\times2n)}}\mathbf{u}_{(2n\times1)} + \mathbf{b}_I^s = \mathbf{0} \tag{7.11}$$

where the superscript *s* stands for strong-form, and $\mathbf{K}_I^s$ and $\mathbf{b}_I^s$ are, respectively, the nodal stiffness matrix and the nodal body force vector for the *I*th node obtained by the strong-form (collocation) method. Equation (7.11) can be re-written as

$$\mathbf{K}_I^{(s)}\mathbf{u} + \mathbf{b}_I^{(s)} = \mathbf{0} \tag{7.12}$$

where

$$\mathbf{K}^s_{I(2\times2n)} = \mathbf{L}^{\mathrm{T}}{}_{(2\times3)}\mathbf{D}_{(3\times3)}\mathbf{L}_{(3\times2)}\boldsymbol{\Phi}(\mathbf{x}_I)_{(2\times2n)} \tag{7.13}$$

$$\mathbf{b}^s_{I(2\times1)} = \begin{Bmatrix} b_x(\mathbf{x}_I) \\ b_y(\mathbf{x}_I) \end{Bmatrix} \tag{7.14}$$

Equation (7.12) is the discretized system equations for the *I*th field node created using the strong-form and simple collocation procedure. It consists of two linear equations that are for the *I*th field node. No numerical integration is needed to obtain Equation (7.12), and only simple collocations are performed.

Using the similar algorithm as that in Chapter 5, we can obtain the following formulation for the local Petrov-Galerkin weak-form for Equation (7.5) for the *I*th DBR-node

$$\begin{aligned} &\int_{\Omega_q} \widehat{\mathbf{V}}_I^{\mathrm{T}}\mathbf{DBu}\mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{nDBu}\mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{nDBu}\mathrm{d}\Gamma \\ &\qquad = \int_{\Gamma_{qt}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma + \int_{\Omega_q} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega \end{aligned} \tag{7.15}$$

where $\widehat{\mathbf{W}}$ (defined in Sub-section 5.2.1) is a matrix of weight functions, $\widehat{\mathbf{V}}$ (defined in Sub-section 5.2.1) is a matrix that collects the derivatives of the weight functions, the vector of tractions $\mathbf{t}$ at point $\mathbf{x}$ is defined in Equation (5.17), and $\mathbf{n}$ is a matrix defined in Sub-section 5.2.1 collecting the components of the unit outwards normal vector on the boundary. Substituting Equation (7.7) into Equation (7.15) leads to the following discretized systems of linear equations for the *I*th node.

$$\mathbf{K}_I^{(w)}\mathbf{u} = \mathbf{f}_I^{(w)} \tag{7.16}$$

where the superscript *w* stands for weak-form , and $\mathbf{K}_I^{(w)}$ is the nodal stiffness' matrix for the *I*th field node created using the local weak-form. It can be expressed as

$$\mathbf{K}_I^{(w)} = \int_{\Omega_q} \widehat{\mathbf{V}}_I^{\mathrm{T}}\mathbf{D}\mathbf{B}\mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{n}\mathbf{D}\mathbf{B}\mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{n}\mathbf{D}\mathbf{B}\mathrm{d}\Gamma \tag{7.17}$$

and $\mathbf{f}_I^{(w)}$ is a nodal force vector for the *I*th field node created using the local weak-form:

$$\mathbf{f}_I^{(w)} = \int_{\Gamma_{qt}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\overline{\mathbf{t}}\mathrm{d}\Gamma + \int_{\Omega_q} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega\alpha \tag{7.18}$$

This consists of contributions from body forces applied in the problem domain and tractions applied on the derivative boundary.

Equation (7.16) is a set of the discretized system equations for the *I*th field node using the local Petrov-Galerkin weak-form. It consists of two linear equations for the *I*th DBR-node.

Using Equations (7.12) and (7.16), we can express the discretized system equations for the *I*th field node in the following general form

$$\mathbf{K}_I\mathbf{u} = \mathbf{f}_I \tag{7.19}$$

where

$$\mathbf{K}_I = \begin{cases} \mathbf{K}_I^{(w)}, & \Omega_q(\mathbf{x}_I) \cap \Gamma_t \neq \varnothing \\ \mathbf{K}_I^{(s)}, & \Omega_q(\mathbf{x}_I) \cap \Gamma_t = \varnothing \end{cases} \tag{7.20}$$

or in detail

$$\mathbf{K}_I = \begin{cases} \int_{\Omega_q} \widehat{\mathbf{V}}_I{}^{\mathrm{T}}\mathbf{D}\mathbf{B}\mathrm{d}\Omega - \int_{\Gamma_{qi}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{n}\mathbf{D}\mathbf{B}\mathrm{d}\Gamma - \int_{\Gamma_{qu}} \widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{n}\mathbf{D}\mathbf{B}\mathrm{d}\Gamma & \Omega_q(\mathbf{x}_I) \cap \Gamma_t \neq \varnothing \\ \mathbf{L}^{\mathrm{T}}\mathbf{D}\mathbf{L}\ \boldsymbol{\Phi} & \Omega_q(\mathbf{x}_I) \cap \Gamma_t = \varnothing \end{cases} \tag{7.21}$$

and

$$\mathbf{F}_I = \begin{cases} \mathbf{F}_I^{(w)}, & \Omega_q(\mathbf{x}_I) \cap \Gamma_t \neq \varnothing \\ \mathbf{F}_I^{(s)}, & \Omega_q(\mathbf{x}_I) \cap \Gamma_t = \varnothing \end{cases} \tag{7.22}$$

or in detail

$$\mathbf{F}_I = \begin{cases} \int_{\Gamma_{qt}} \widehat{\mathbf{W}}_I^{\mathrm{T}} \overline{\mathbf{t}} \mathrm{d}\Gamma + \int_{\Omega_q} \widehat{\mathbf{W}}_I^{\mathrm{T}} \mathbf{b} \mathrm{d}\Omega & \Omega_q(\mathbf{x}_I) \cap \Gamma_t \neq \varnothing \\ \mathbf{b}_I^{(s)} & \Omega_q(\mathbf{x}_I) \cap \Gamma_t = \varnothing \end{cases} \tag{7.23}$$

With Equation (7.19) for all N field nodes in the entire problem domain, and assembling all these $2N$ equations together, we obtain the final global system equations and expressed in the following form.

$$\mathbf{K}_{(2N\times 2N)} \mathbf{U}_{(2N\times 1)} = \mathbf{F}_{(2N\times 1)} \tag{7.24}$$

where $\mathbf{K}$ is the global stiffness matrix and $\mathbf{F}$ is the global force vector. Solving this equation for $\mathbf{U}$ after imposing essential boundary conditions, we can obtain the displacements for all the field nodes and then can retrieve the stresses using Equations (4.10) and (5.12).

Note that it is easy to enforce the essential boundary conditions, because the strong-form method is used. If the MFree shape functions possess the delta function property, the equations for the nodes on the essential boundary need not even be created. If the shape functions do not possess the Delta function property, the direct interpolation method can be used. Detailed discussions can be found in Chapter 6.

7.3.4 Numerical implementation

7.3.4.1 Property of stiffness matrix

From Equations (7.21) and (7.24), it can be easily seen that the system stiffness matrix, $\mathbf{K}$, in MWS is sparse and banded as long as the support domain of meshfree interpolation is compactly supported. However, $\mathbf{K}$ is usually unsymmetric.

The global stiffness matrix in MWS comprises two parts: the nodal stiffness matrices obtained from the strong-form and the local weak-form. The asymmetry of the stiffness matrix is inherited from the nature of the local Petrov-Galerkin weak-form, which has been discussed in Sub-section 5.2.2. The portion of the stiffness matrices coming from the use of strong-form may be symmetric if the same support domains are used for all the field nodes. However, this requirement usually cannot be met unless one uses a set of regular grids as in the conventional FDM model. Therefore, the global

stiffness matrix in MWS is usually unsymmetric. Making the global stiffness matrix symmetric would improve the efficiency and the stability† of the MWS method.

7.3.4.2 Type of local domains

In MWS, the local weak-form is used for the DBR-nodes. Similar to the MFree local weak-form methods, for any DBR-node at $\mathbf{x}_I$, there exist three local domains as discussed in Sub-section 5.2.2.

For all collocatable nodes, the strong-forms are used via the collocation procedure. As shown in Figure 7.1, there is only one local domain, the support domain Ω_s used for field variable approximation, for a collocatable node. The size of the local support domain has been defined in Equation (5.30), and the suggested size is $\alpha_s = 1.5 \sim 3.0$.

7.3.4.3 Numerical integration

Integrations in MWS are performed only for the few DBR-nodes. However, care should still be taken to obtain accurate numerical integrations. As discussed in Sub-section 5.2.2 (see Figure 7.1), the local quadrature domain Ω_q should be sufficiently large ($\alpha_q = 1.5 \sim 2.0$ is recommended), and it should be divided into small partitions, and sufficient Gauss quadrature points should be used in each of the small partitions. A more detailed discussion of local numerical integrations can be found in the book by GR Liu (2002). For complex quadrature domains, triangular background cells may be used.

7.4 SOURCE CODE

In this section, a standard computer code, MFree_MWS.f90, of the MWS method is given. This code is developed using FORTRAN 90. Combined with Subroutines RPIM_ShapeFunc_2D and MLS_ShapeFunc_2D given in Chapter 3, the code can perform the task of the MWS method using both RPIM and MLS shape functions. For the convenience of description in later comparison studies, we use MWS-RPIM to denote the MWS method using RPIM shape functions, and MWS-MLS to denote the MWS method using the MLS shape functions.

† It is generally true that a symmetric system seems to be more stable than an unsymmetric one.

7.4.1 Implementation issues

Numerical implementations used in the code MFree_MWS.f90 are similar to those used in the code MFree_Local.f90 presented in Sub-section 5.4.1. Hence, numerical implementations of MFree_MWS.f90 are only briefly described here.

As in the discussions in Sub-section 4.4.1, the influence domains are used for construction of the meshfree shape functions. The dimensions of the influence domain can be determined as in Sub-section 4.4.1. In the code MFree_MWS.f90, rectangular influence domains are used. The dimension of the influence domain is defined in Equations (4.75).

Because the requirement for the consistency of trial functions in the strong-form is higher (e.g., 2nd order for 2D solids) than that in the weak-form (e.g. 1st order for 2D solids)[†], a basis with higher order should be used in the MLS approximation. The parabolic polynomial basis (*mbasis*=6) is therefore used in the MWS-MLS. In addition, the 4th-order spline weight function is used as the weight function in computing the MLS shape functions.

In the present MFree_MWS.f90 code, rectangular quadrature domains are used. For problems with derivative boundaries of complex shapes, quadrature domains consists of triangular cells should be used. The sizes of the rectangular quadrature domain have been defined in Sub-section 5.2.2. The direct interpolation method is used to enforce the essential boundary conditions.

For error analysis, the energy norm defined in Equation (4.78) is used as an error indicator. Note that the integration in Equation (4.78) is over the global domain. Hence, in order to assess the global error in the energy norm, global background cells that are the same as these used in the RPIM (or EFG) have to be used.

7.4.2 Program description

The flowchart of MFree_MWS.f90 is shown in Figure 7.2. The procedure of the MWS method is similar to that in the MFree local weak-form method. The main difference comes in the construction of the nodal stiffness matrix. In the flowchart of the MWS method, the geometry of the problem domain is modelled and a set of nodes is generated to represent the problem domain. The system matrices are assembled through loops for all the field nodes. The local quadrature domain is constructed for each node,

---

[†] See, for example, the discussions given in Section 5.2.2 in the book by GR Liu (2002) for the detailed argument on consistence.

and then a checking is performed to determine whether the local quadrature domain intersects with the derivative boundaries.

- If the local quadrature domain does not intersect with the derivative boundaries, it is then noted as a collocatable node, and the nodal stiffness is obtained directly through collocation using the strong-form.
- If the local quadrature domain intersects with the derivative boundaries, it goes into the inner loop. In the inner loop, the nodal stiffness matrix is obtained through another loop for all Gauss quadrature points in the quadrature domain of this DBR-node.

After the construction of the global discretized system equations, the essential boundary conditions are enforced by direct interpolation. The algebraic system equations are solved using a standard linear equation solver (for banded unsymmetric matrix) to obtain the nodal displacements or the parameters of the nodal displacements. Finally, the nodal stress and the global error in the energy norm are computed.

The source code of the main program of MFree_MWS.f90 is listed in Program 7.1. The main program of the MWS method calls several subroutines. The macro chart for the program is the same as Figure 5.4. The functions of these subroutines are listed in Appendix 7.1. Because all the subroutines used in MFree_MWS.f90 are the same as those used in the program MFree_Local.f90, the source codes of these subroutines are not repeated in this chapter. The same global variables as given in Appendix 5.2 are used in MFree_MWS.f90. In this chapter, the quartic spline function is used as the test function in the local weak-form. The source code of this test function is provided in Program 6.2. The including file, variableslocal.h, is given in Program 5.1.

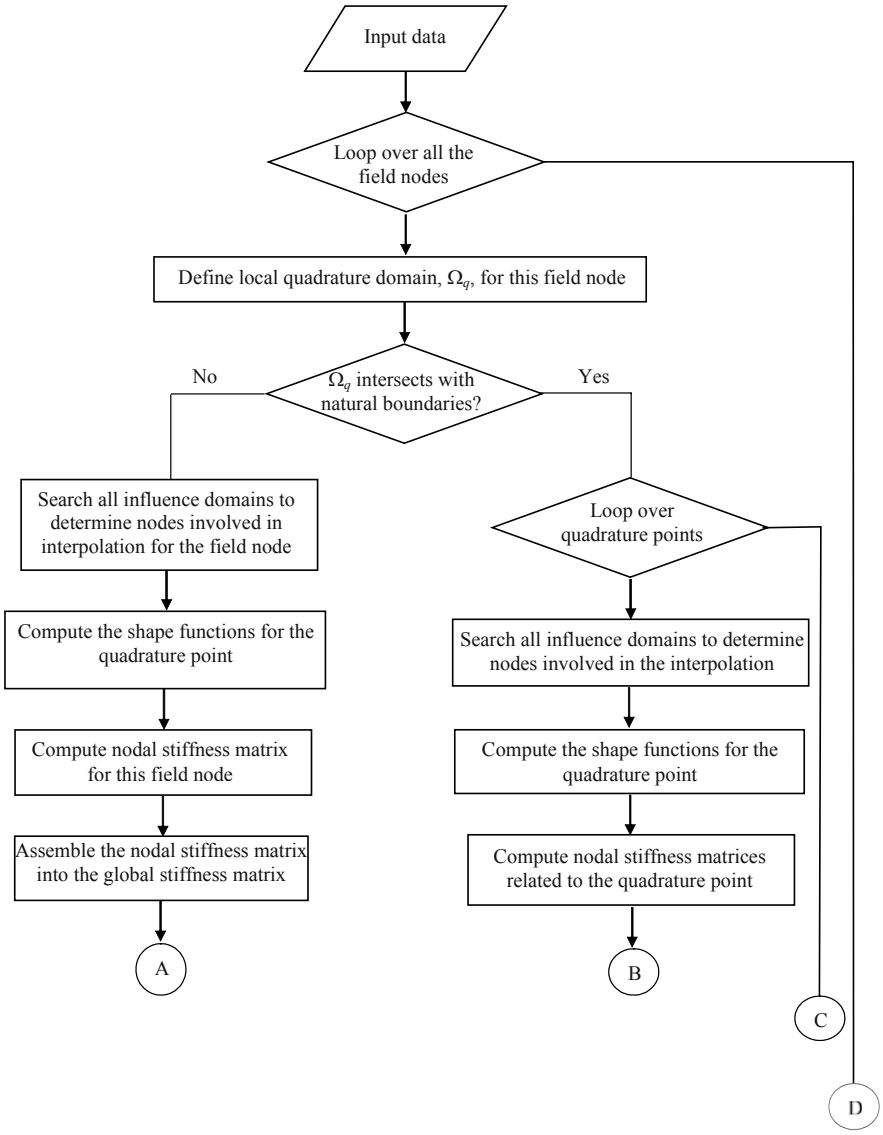
7.5 EXAMPLES FOR TESTING THE CODE

The code is tested on a cantilever beam subjected to a parabolic traction at the free end as shown in Figure 4.5. The beam has a unit thickness and is in plane stress. The exact solution of this problem is given in Equations (4.79)~(4.84). As in discussions in Chapter 5, the following three steps should be followed:

Step 1: Preparation of input file of this program

The data file is similar to that used in Appendix 5.9. A sample input data file used in MWS.f90 is given in Appendix 7.2. This input data file has the

same structure as that used in MFree_local.f90 (Sub-section 5.5.1), in which the beam problem is represented by regularly and evenly distributed 189 (21×9) field nodes, as shown in Figure 4.12(a).



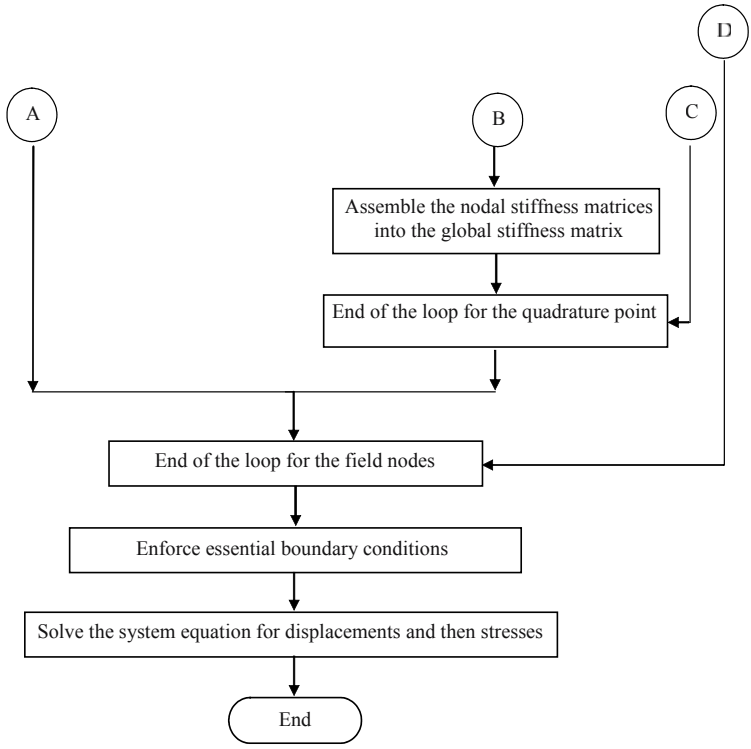


Figure 7.2. Flowchart for the program of the MFree Weak-Strong (MWS) form method, MFree_MWS.f90.

Step 2: Execution of the program

The MWS-RPIM results are first obtained and are listed in the output files given in Appendix 7.3. In the end of the output, the error in the energy norm is also presented.

The MWS-MLS results are listed in Appendix 7.4. At the end of the output, the error in the energy norm is also presented.

Step 3: Analysis of the output results

Results obtained using MWS-RPIM are presented in Figure 7.3 and Figure 7.4. In this study, the MQ-RBF is used together with the linear polynomial terms and the parameters used are: $\alpha_c = 4.0$, q=1.03, d_c=2.4, and $\alpha_i = 3.0$. For local quadrature domains, $\alpha_q = 1.5$ and n_g=2 are used. The quartic spline function (W2) is employed as the test function for the local weak-form. The results of deflections are plotted in Figure 7.3. For comparison, the analytical results from Equations (4.79) and (4.81) are plotted in the same figure; there is

good agreement. The results of shear stress, τ_{xy} , are plotted in Figure 7.4. The MWS-RPIM method gives accurate results, even for stresses.

The cantilever beam is also modelled using 189 irregularly distributed nodes, as shown in Figure 7.5. Results are obtained using the MWS-RPIM method and plotted in Figure 7.6 and Figure 7.7. Again there is good agreement with the analytical results.

Results of the MWS-MLS are presented in Figure 7.8 and Figure 7.9. In this study, the parameters used are α_t=3.0, α_q=1.5, and n_g=2. The parabolic polynomial basis (*mbasis*=6) is used in computing the MLS shape functions, and the quartic spline weight function (W2) is used as the weight function in both MLS shape functions and the local weak-form. The deflections are plotted in Figure 7.8, and the shear stress, τ_{xy}, are plotted in Figure 7.9. Again there is good agreement with the analytical results.

The 189 irregular nodes in Figure 7.5 are also used. Results are obtained using the MWS-MLS method and plotted in Figure 7.10 and Figure 7.11. Again there is good agreement with the analytical results.

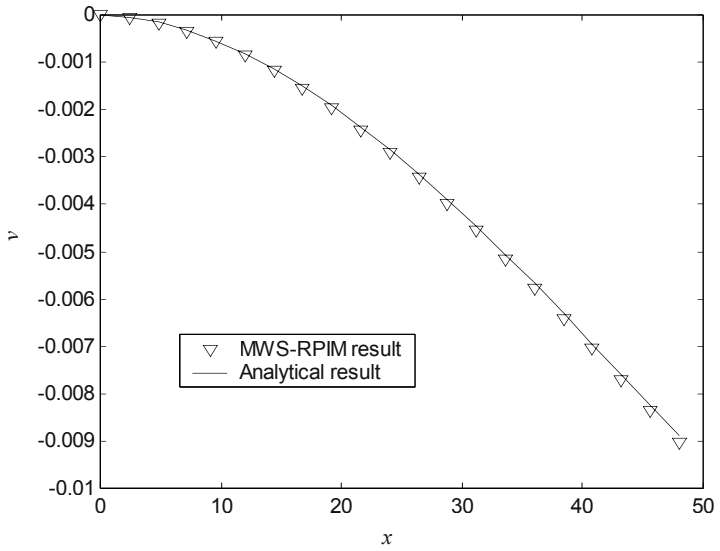


Figure 7.3. Deflections v on the central axis at y=0 of the beam obtained using the MWS-RPIM method and 189 regularly distributed field nodes.

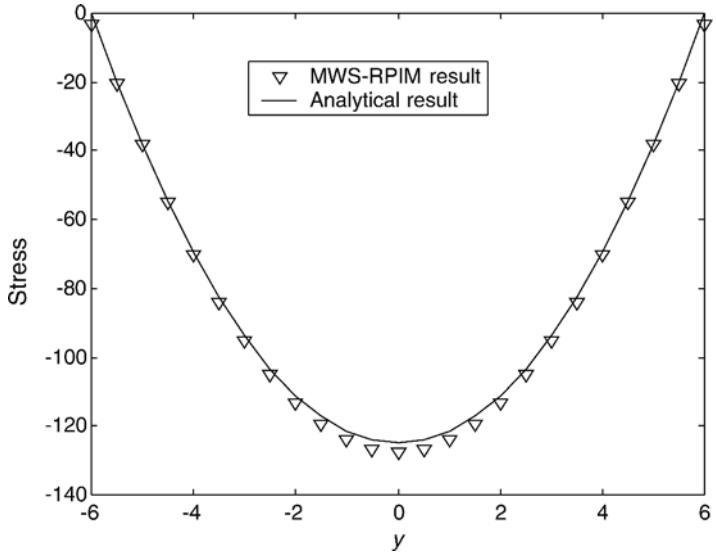


Figure 7.4. Shear stresses on the cross-section at $x=L/2$ of the beam obtained using the MWS-RPIM method and 189 regularly distributed field nodes.

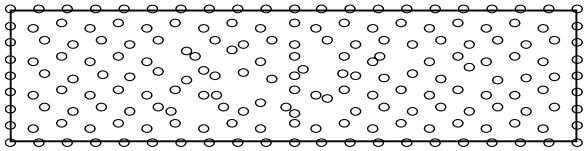


Figure 7.5. A total of 189 irregularly distributed nodes.

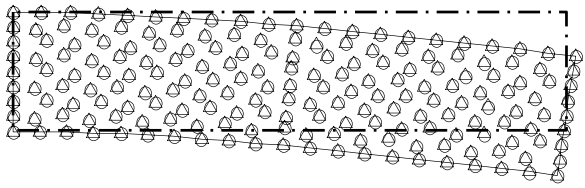


Figure 7.6. Deflection of the beam obtained using the MWS-RPIM method and 189 irregularly distributed field nodes. Note that the displacements plotted are magnified by 500 times.

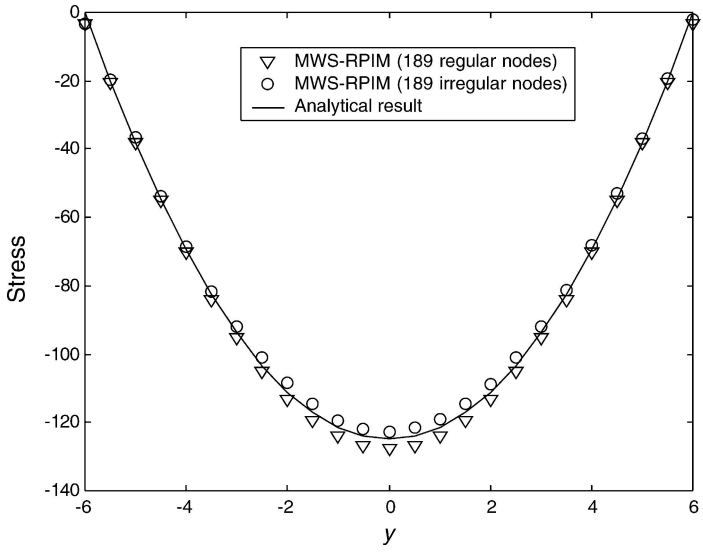


Figure 7.7. Shear stresses on the cross-section at $x=L/2$ of the beam obtained using the MWS-RPIM method and 189 field nodes.

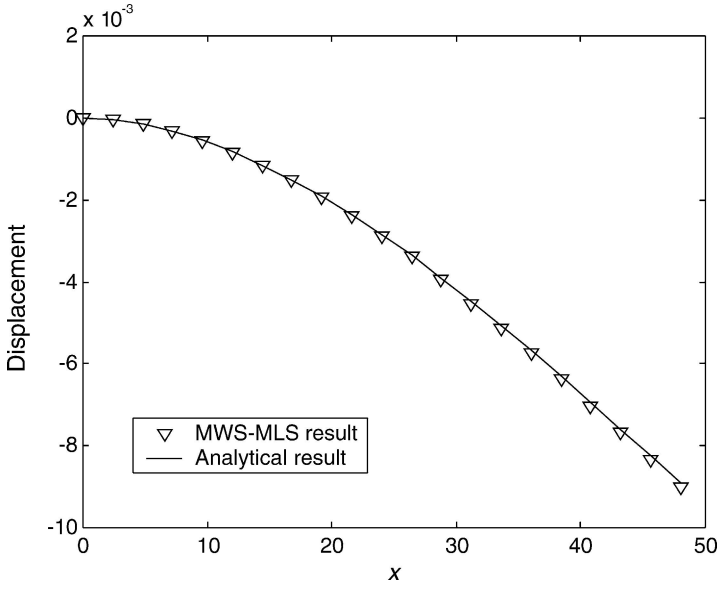


Figure 7.8. Deflections v on the central axis at $y=0$ of the beam obtained using the MWS-MLS method and 189 regularly distributed field nodes.

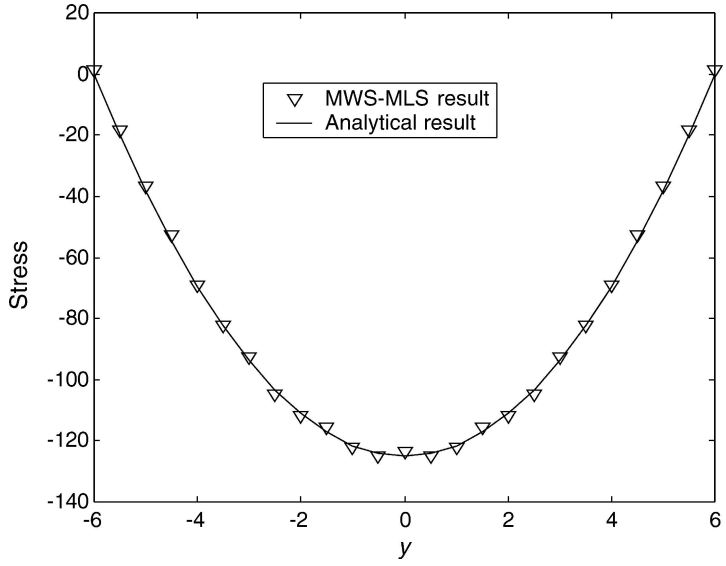
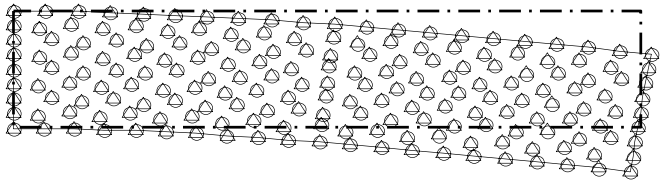


Figure 7.9. Shear stresses on the cross-section at $x=L/2$ of the beam obtained using the MWS-MLS method and 189 regularly distributed field nodes.



O Analytical solution Δ MWS-MLS

Figure 7.10. Deflections of the beam obtained using the MWS-MLS method and 189 irregularly distributed field nodes. Note that the displacements plotted are magnified by 500 times.

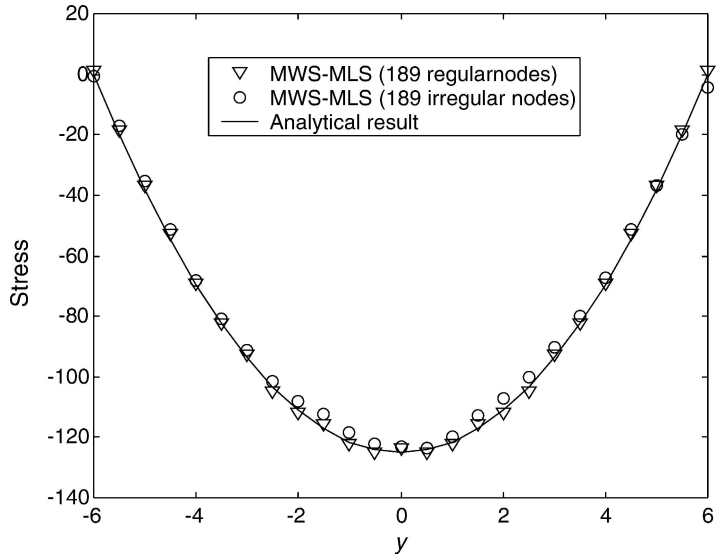


Figure 7.11. Shear stresses on the cross-section at $x=L/2$ of the beam obtained using the MWS-MLS method and 189 field nodes is used.

7.6 NUMERICAL EXAMPLES FOR 2D ELASTOSTATICS

7.6.1 1D truss member with derivative boundary conditions

The problem of the truss member discussed in Example 6.1 of Chapter 6 is analyzed using the MWS method. All conditions and parameters are exactly as in Example 6.1. As discussed in Chapter 6, special treatments are required to impose the derivative boundary conditions. Table 7.1 lists results of different methods to solve this truss problem using the polynomial PIM shape functions and 11 field nodes (both regular and irregular nodes, shown in Figure 6.7). The table shows that the MWS method produces the best result for both regular and irregular nodal distributions.

Table 7.1 Relative errors *e* (%) in results obtained different methods*

Case	Schemes	Regular nodes	Irregular nodes
0	Dirichlet BC	0.51	1.36
1	DC	11.3	6.12
2	FP	1.63	7.56
3	HC	2.68	3.05
4	RG	11.3	6.12
5	MWS	1.24	2.98

* 3 nearest nodes are used in the local support domain;
In MWS, $\alpha_q = 1.5 d_c$ is used for the local quadrature domain and 8 Gauss points are used in the quadrature domain. To ensure local compatibility, the same support domain is used for all Gauss points in a quadrature domain in the construction of PIM shape functions.

7.6.2 Standard patch test

This numerical example is to perform the standard patch test that is often used in the FEM. Three patches shown in Figure 7.12 are tested. Figure 7.12 (a) shows a patch with 15 irregular distributed nodes. Figure 7.12 (b) shows one with 25 nodes including 9 irregularly-placed interior nodes. Figure 7.12 (c) shows one with 55 nodes including 39 irregularly distributed internal nodes.

The dimensions of these patch tests are presented in Figure 7.12. The material parameters are E=1.0 and ν=0.3. In these patch tests, the displacements are prescribed along all boundaries by a linear function of x and y:

$$u_i = x_i + y_i \tag{7.25}$$

$$v_i = x_i - y_i \tag{7.26}$$

Satisfaction of the patch test requires that the displacement of any interior node be given by the same linear functions, Equation (7.25) and (7.26), and the strains and stresses should be constant in the patch. Because there is no traction (derivative) boundary condition in these patch tests, all nodes are collocatable nodes, and the strong-forms are used to construct the discretized system equation. For the influence domain, $\alpha_{ix} = \alpha_{iy} = \alpha_i = 1.6$ is used. Both RPIM-MQ (with linear polynomial terms) and MLS shape functions are used. In the MLS approximation, the parabolic basis and the weight function W2 are used.

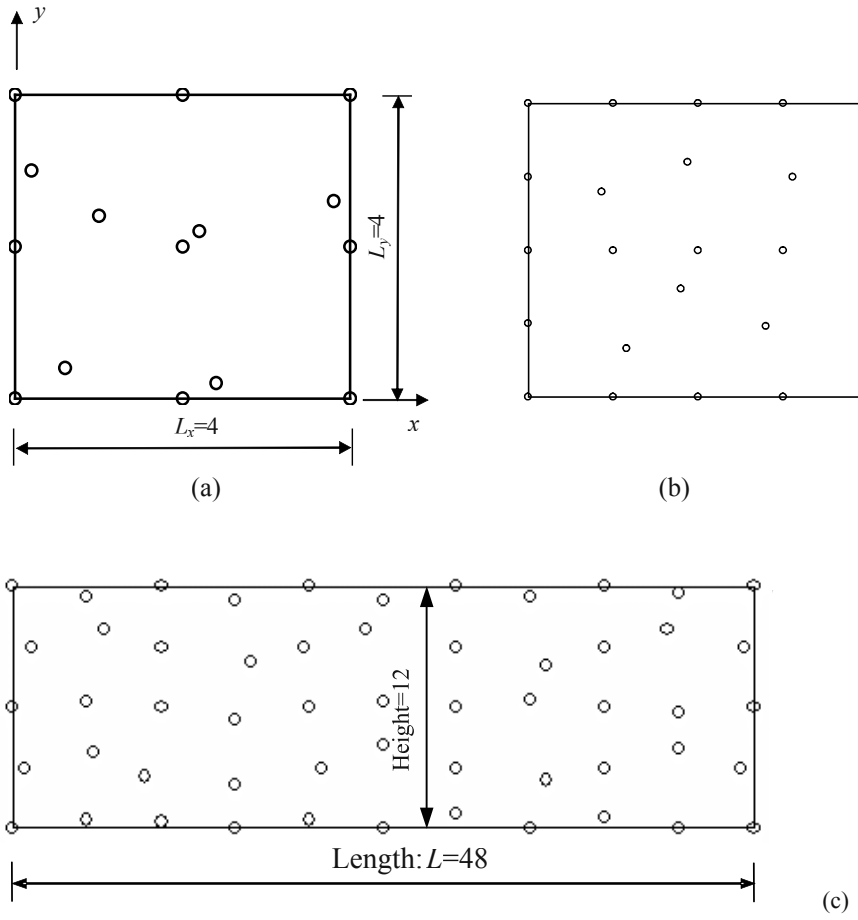


Figure 7.12. Standard patch tests. (a) patch with 15 irregular nodes; (b) patch with 25 irregular nodes; (c) patch with 55 irregular nodes.

The MWS method can pass all the patch tests. If RPIM shape functions (with m=3) are used, the relative displacement error is less than 10^{-15}. It is also confirmed that if the polynomial terms are not included in the RPIM-MQ shape functions, these patch tests cannot be passed exactly, as discussed by GR Liu (2002). If MLS shape functions are used, the relative displacement error is 10^{-11}.

The requirements for the MWS method to pass the patch test are listed as follows:

1) The shape functions have at least linear consistence. This means that the MFree shape functions used should at least be able to reproduce a linear function.

2) The essential boundary conditions must be accurately imposed.

The RPIM-MQ with linear polynomial terms and MLS shape functions can satisfy the first requirement easily because linear polynomials are included in the basis. Without additional linear terms, RPIM-MQ shape functions do not satisfy the first requirement; there will be errors in the results for these patch tests.

RPIM-MQ shape functions can also satisfy the second requirement, as they have the Kronecker delta function property. However, the MLS shape function has no delta function property. The second requirement cannot be exactly satisfied when the MLS shape function is used without additional treatments. Although the MWS-MLS with the direct interpolation method can pass the standard patch test, the enforcement of essential boundary conditions will introduce some numerical error. Hence, for the standard patch test problem, the error of MWS-MLS with the direct interpolation method is larger than that of the MWS-RPIM. For MWS-MLS to accurately pass the patch test, the Lagrange multiplier method should be used.

7.6.3 Higher-order patch test

In these examples of the standard patch tests, there is no the derivative boundary. Hence, no local weak-form is used. In order to fully examine the efficiency of the MWS formulation, the following high-order patch tests are studied. As shown in Figure 7.13, a patch is subjected to two types of loading at the right end.

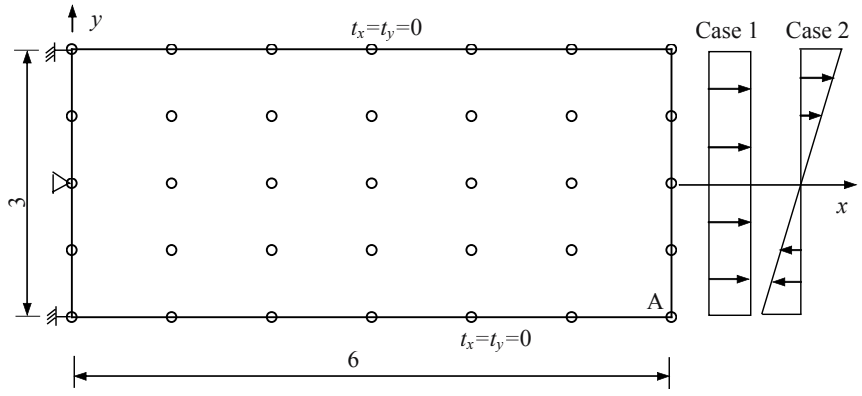


Figure 7.13. A higher-order patch and regular nodal distribution.

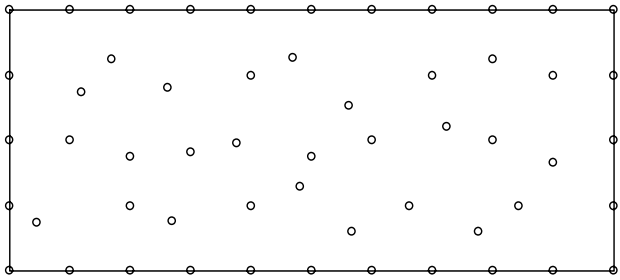


Figure 7.14. The irregular nodal distribution for the high order patch test.

1) Case 1: a uniform axial stress with unit intensity is applied on the right end. The exact solution of displacements for this problem with E=1 and v=0.25 is:

$$u_i = x_i \tag{7.27}$$

$$v_i = -\frac{y_i}{4} \tag{7.28}$$

2) Case 2, a linearly varying normal stress is applied on the right end. The exact solution of displacements for this problem with E=1 and v=0.25 is:

$$u_i = \frac{2x_i y_i}{3} \tag{7.29}$$

$$v_i = -\frac{x_i^2 + y_i^2/4}{3} \tag{7.30}$$

For the construction of the RPIM and MLS shape functions, the influence domains with $\alpha_i = 2.5$ are used in this study.

Case 1 is passed exactly (to very high accuracy) by the presented MWS method using both RPIM with the linear polynomial terms and MLS shape functions. In the MLS approximation, the parabolic basis and the weight function W2 is used. This case demonstrates that the MWS method exactly implements the traction (derivative) boundary condition and leads to an exact solution for this problem in which the analytical displacement solution is linear.

The computational results of displacements at point A (at the lower-right corner, see Figure 7.13) for case 2 are shown in Table 7.2. There is an error in solving case 2 of the high order patch test using the MWS methods.

Table 7.2 Relative errors (%) of u_x at point A for case 2 of the higher-order patch test (using regular nodes)

	α_q=1.0		α_q=1.5	
	u(error)	v(error)	u(error)	v(error)
MWS-RPIM	-6.682 (11.362%)	-13.793 (13.175%)	-6.099 (1.644%)	-12.572 (3.157%)
LRPIM (full local weak-form)	-6.403 (6.712%)	-13.100 (7.489%)	-6.073 (1.214%)	-12.544 (2.923%)
MSW-MLS	-5.955 (-0.758%)	-12.113 (-0.609%)	-5.973 (-0.449%)	-12.141 (-0.386%)
MLPG (full local weak-form)	-5.956 (-0.728%)	-12.118 (-0.572%)	-5.985 (-0.245%)	-12.163 (-0.199%)
Exact	-6.00	-12.1875	-6.00	-12.1875

The reason for the error mainly comes from the errors of the numerical integration for the complex DBCs. In order to study the effect of the numerical integration, results of two different sizes of quadrature domains are obtained and listed in Table 7.2. The error decreases when a lager quadrature domain is used. When $\alpha_{qx} = \alpha_{qy} = \alpha_q = 1.0$, the local quadrature domain is too small to effectively smear the error along the derivative boundary. It is be found that the accuracy of the solution improves with the improvement of the numerical integration by use of more Gauss quadrature points and more sub-partitions for the numerical integrations.

The irregularly distributed nodes for this high patch test, as shown in Figure 7.14, are also used in this study, and results are listed in Table 7.3. The MWS method can also give acceptable results for this irregular nodal distribution.

For comparison, results of MFree local radial point interpolation method (LRPIM) and MLPG methods, which use local weak-forms entirely for all the field nodes, are listed in Table 7.2 and Table 7.3. LRPIM leads to more accurate results than MWS-RPIM, and MLPG has nearly the same accuracy as MWS-MLS.

The MFree strong-form method (the collocation method) that uses strong-forms entirely for all field nodes is also used in the high order patch test. It has been found that the MFree collocation method can also produce satisfactory results for case 1, whose force boundary condition is simple. However, it leads to large errors (>15%) for case 2 with regular nodal

distribution. Displacement results of irregular nodes using MFree collocation method based on RPIM are listed in Table 7.3. The error is even more than 40%. The solution of the MFree collocation method is also unstable. It is sensitive to the nodal distribution and parameters used in the model. The error and instability mainly come from the error in the implementation of the complex force (derivative) boundary conditions in case 2. Compared with the pure collocation method, the present MWS method has better accuracy and stability for this high order patch test due to the use of the local weak-form for the DBR-nodes.

Results of several MFree methods used for patch tests are summarized in Table 7.4.

Table 7.3. Relative errors (%) of u_x at point A for case 2 of the higher-order patch test (using irregular nodes, α_q=1.5)

	Exact	Collocation (RPIM)	MWS-RPIM	LRPIM	MWS-MLS	MLPG
u	-6.00	-8.786	-6.389	-5.951	-5.976	-5.982
Error	/	46.6%	6.491%	-0.808%	-0.396%	-0.291%
v	-12.1875	-16.202	-13.234	-12.020	-12.168	-12.172
Error	/	49.3%	8.586%	-1.408%	-0.160%	-0.159%

Table 7.4. Summarization of patch tests

	Standard patch test	Higher-order patch test (case 1)	Higher-order patch test (case 2)
MWS-RPIM	Pass	Pass	Pass with small error
MWS-MLS	Pass	Pass	Pass with small error
LRPIM	Pass	Pass	Pass with small error
MLPG	Pass	Pass	Pass with small error
Collocation method	Pass	Pass	Cannot pass

7.6.4 Cantilever beam

The cantilever beam shown in Figure 4.5 is reconsidered for further study numerically on convergence and stability of the MWS method. The results of displacements and stresses were discussed and presented in Section 7.5.

The collocation method that uses pure strong-forms is also used to solve the same problem under the same conditions. The error obtained using the collocation method is large even for regular nodes. It fails for the irregularly distributed nodes. The solution of the MFree collocation method is also unstable. Compared with the pure collocation method, the MWS method has better accuracy and stability for this problem. In the following studies, the MWS results are compared with those for stable methods such as the LRPIM, MLPG, and FEM.

a) Convergence study

The convergences of the MWS methods are first numerically studied for this cantilever beam problem. Regularly distributed 18 (3×6), 55(5×11), 112(7×16), 189(9×21) and 403(13×31) nodes are used. The convergence curves of error in energy norm obtained numerically are shown in Figure 7.15. For comparison, the convergence curves for LRPIM, MLPG, and FEM using bi-linear elements are plotted in the same figure. The h is the nodal spacing d_c in the MFree methods, and is equivalent to the maximum element size (in x direction) in the FEM analysis in this case. The convergence rates, R, computed via linear regression are also given in Figure 7.15. From Figure 7.15, we can find the following:

1) MFree methods have better accuracy and convergence than the conventional FEM using bi-linear elements.
2) Using local weak-forms for all field nodes, the LRPIM and MLPG have slightly better accuracy than the MWS method. This is because the use of strong-forms for the collocatable nodes in MWS reduces slightly the accuracy.
3) The MWS-MLS method has good convergence rate and high accuracy. Compared with MLPG, the MWS-MLS has nearly same convergence and accuracy.
4) The convergence process of the MWS-RPIM using MQ-RBF is not good when finer nodes are used although the accuracy is acceptable. Further tuning of the shape parameter may be necessary.

The poor convergence of the MWS-RPIM (MQ) may be attributed to the property of the MQ-RBF that is often found poor performance in h-convergence. The properties of RPIM-MQ have been studied by Gu and GR

Liu (2003b) in detail for mechanics problems. It was found that pure MQ-RBF cannot always ensure to exactly reproduce a linear field function. This could be one of the major reasons for the poor *h*-convergence in using MQ-RBF. Another cause for the poor convergence is the shape parameters chosen in the RBFs. When a proper shape parameter of MQ-RBF is used, its convergence improves. Unfortunately, there is no theoretical optimal value for these shape parameters. Other RBFs (e.g. Gaussian RBF, the compactly supported RBFs, etc.) could be used to improve the convergence of the MWS-RPIM. To find an efficient method to improve the *h*-convergence of the MWS-RPIM is still an open issue.

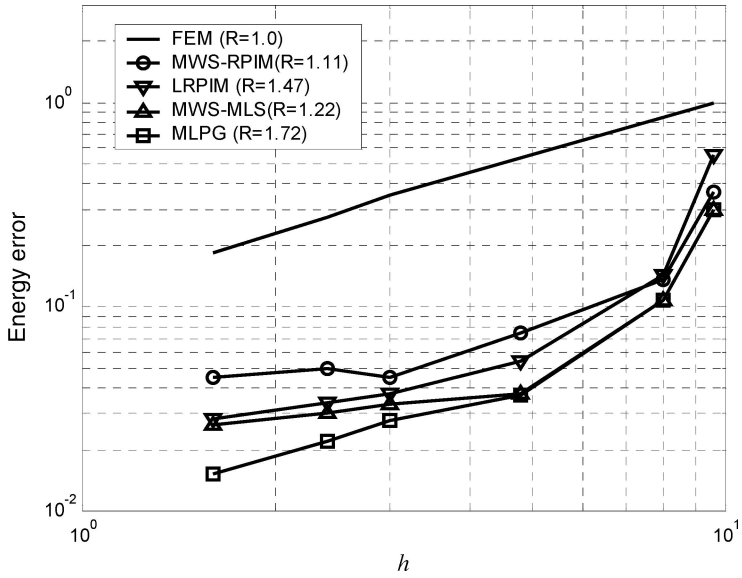


Figure 7.15. Comparisons of convergences of MWS, LRPIM, MLPG, and FEM in error e_e of energy norm. R is the convergence rate. The same parameters are used in MWS-RPIM and LRPIM; The same parameters are used in MWS-MLS and MLPG.

b) Efficiency of the MWS method

In the efficiency study, regularly distributed 55, 189 and 403 nodes are used. The influence domain with α_i=3.0 is used to construct shape functions. The CPU times of MWS, LRPIM and MLPG are listed in Table 7.5. From this table, it can be found that MWS-RPIM and MWS-MLS use much less CPU time than their counterparts of pure local weak-form methods, LRPIM and MLPG.

Table 7.5. CPU time (s) required by different methods for the cantilever beam problem

	MWS-RPIM	LRPIM	MWS-MLS	MLPG
55 nodes	43.710	50.060	2.060	5.360
189 nodes	66.730	310.630	7.270	14.541
403 nodes	123.160	822.710	13.840	32.245

* Computer system used: DataMini PC, Intel Pentium 4 CPU 1.80 GHz.

Note that the computational cost must be considered together with the accuracy for a fair comparison. A successful numerical method should obtain high accuracy at a low computational cost. The curves of error in energy norm against the computation time for the MWS methods are computed and plotted in Figure 7.16. For comparison, the same curves for LRPIM and MLPG are computed and plotted in the same figure. From Figure 7.16, the following points can be observed:

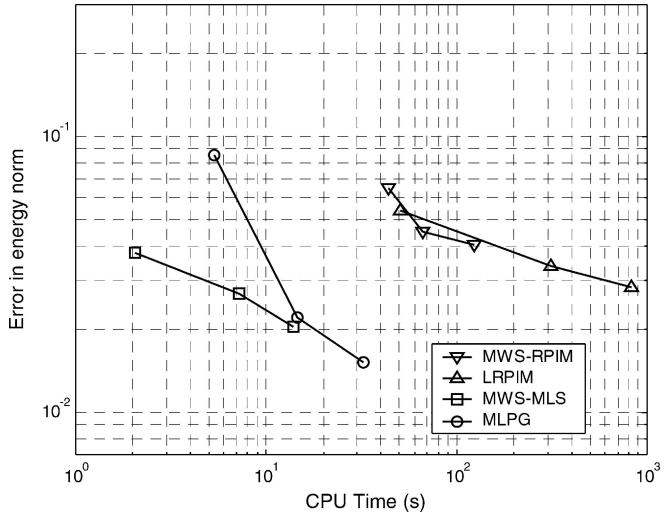


Figure 7.16. Comparison of efficiencies of MWS, LRPIM, and MLPG. The data of LRPIM and MLPG are obtained from Chapter 5.

- The MWS methods are more efficient than their corresponding MFree local weak-form methods.

- For the same nodal distribution, the MWS methods need much less CPU time. This is because in MWS numerical integrations for all the collocatable nodes are avoided by the use of the strong-form and the simple collocation procedure.
- The MWS-MLS and MLPG have better efficiency than the MWS-RPIM and LRPIM, respectively. This is because the MLS approximation has better efficiency than the RPIM-MQ.

7.6.5 Hole in an infinite plate

Consider the plate with a central circular hole discussed in Example 6.12. The same conditions are used as those employed in Example 6.12. The analytical solutions for an infinite plate (Roark and Young, 1975) are given in Equations (6.142)-(6.147). Due to symmetry, only the upper right quadrant of the plate is modelled. Symmetry conditions are imposed on the left and bottom edges. On the inner boundary of the hole, the boundary conditions are traction free. Traction boundary conditions given by the exact solution Equations (6.145)-(6.147) are imposed on the right (x=5) and top (y=5) edges. Clearly, this problem has more complex traction (derivative) boundary conditions than the beam problem.

A total of 165 nodes is used to represent the plate, and the nodal arrangement is shown in Figure 7.17. The results for the displacements obtained using the MWS and the analytical methods are identical. As the stresses are more critical for accuracy assessment, detailed results of stresses distribution for stress σ_{xx} along x=0 computed using the MWS are shown in Figure 7.18. Figure 7.18 shows that the MWS method yields satisfactory results even for stresses for this problem; they are less accurate near the boundaries.

7.7 DYNAMIC ANALYSIS FOR 2-D SOLIDS

The MWS method is also used to analyze the linear elastodynamics of two-dimensional solids. The standard strong-form of the initial/boundary value problem for 2D linear elastodynamics is given in Equation (1.32). The boundary conditions and the initial conditions are given in Equations (1.33)~(1.36).

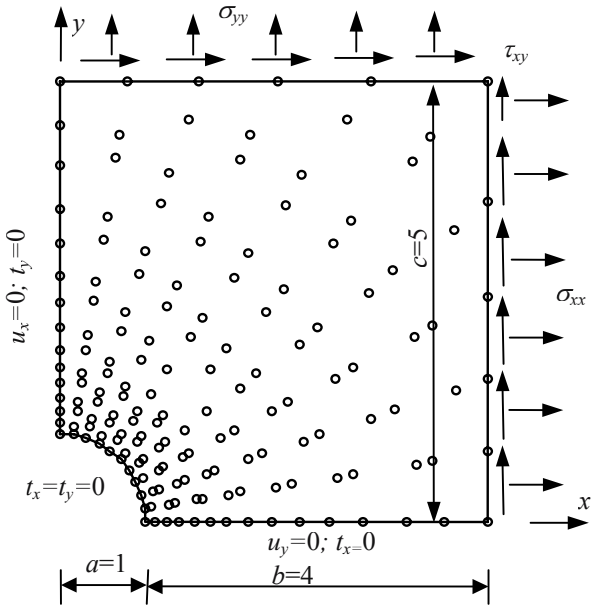


Figure 7.17. Nodes and boundary conditions in the quarter model of the plate with a central hole subjected to a unit unidirectional tensile load in the *x* direction.

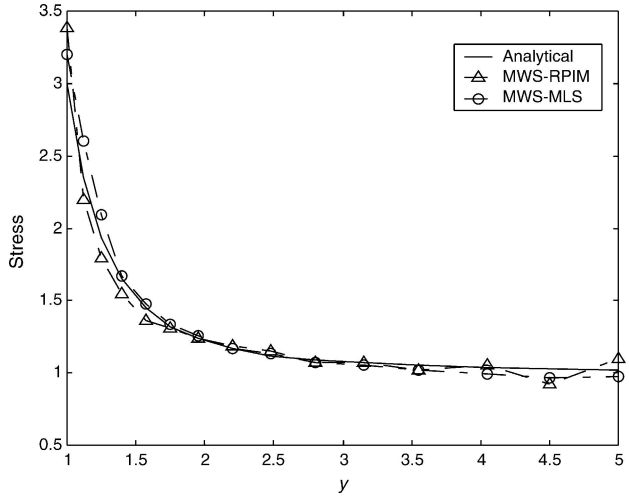


Figure 7.18. Stress (σ_{xx}) distributions along the section of x=0 in the plate obtained using the MWS method and 165 regularly distributed nodes.

As shown in Figure 7.1, the problem domain and boundaries are represented by properly scattered nodes. MWS is used to establish the discretized system equations, the strong-forms are used for collocatable nodes, and the local weak-form is used for DBR-nodes.

7.7.1 Strong-form of dynamic analysis

Equation (1.32) for isotropic materials can be written in terms of displacements in the following standard strong-form.

$$\begin{cases} \dfrac{E}{1-\nu^2}\left(\dfrac{\partial^2 u}{\partial x^2}+\dfrac{1-\nu}{2}\dfrac{\partial^2 u}{\partial y^2}+\dfrac{1+\nu}{2}\dfrac{\partial^2 v}{\partial x\partial y}\right)+b_x-\rho\dfrac{\partial^2 u}{\partial t^2}-c\dfrac{\partial u}{\partial t}=0 \\ \dfrac{E}{1-\nu^2}\left(\dfrac{\partial^2 v}{\partial y^2}+\dfrac{1-\nu}{2}\dfrac{\partial^2 v}{\partial x^2}+\dfrac{1+\nu}{2}\dfrac{\partial^2 u}{\partial x\partial y}\right)+b_y-\rho\dfrac{\partial^2 v}{\partial t^2}-c\dfrac{\partial v}{\partial t}=0 \end{cases} \tag{7.31}$$

where E and ν are Young's modulus and Poisson's ratio, ρ is the mass density, u and v are displacements in x and y directions, respectively, and b_x and b_y are the body forces applied in x and y directions. The collocation method is used directly to discretize Equation (7.31) for all the collocatable nodes.

7.7.2 Local weak-form for the dynamic analysis

For a DBR-node, a local weak-form is used. A local Petrov-Galerkin weak-form for the Ith node of the partial differential Equation (7.31) over a local quadrature domain Ω_q bounded by Γ_q, can be obtained using the weighted residual method or the local Petrov-Galerkin method (Gu and GR Liu, 2001c):

$$\int_{\Omega_q} \widehat{W}_I(\sigma_{ij,j}+b_i-\rho\ddot{u}_i-c\dot{u}_i)\mathrm{d}\Omega=0 \tag{7.32}$$

where $\widehat{W}$ is the weight function.

The first term on the left hand side of Equation (7.32) can be integrated by parts to arrive at

$$\int_{\Gamma_q}\widehat{W}_I\sigma_{ij}n_j\mathrm{d}\Gamma+\int_{\Omega_q}[-\widehat{W}_{I,j}\sigma_{ij}+\widehat{W}_I(b_i-\rho\ddot{u}_i-c\dot{u}_i)]\mathrm{d}\Omega=0 \tag{7.33}$$

The local quadrature domain Ω_q of a node $\mathbf{x}_I$ can be a domain of an arbitrary shape in which $\widehat{W}_I \neq 0$. The boundary Γ_q for the local quadrature domain usually comprises three parts: the internal boundary Γ_{qi}, the boundaries Γ_{qu} and Γ_{qt}, over which the essential and derivative boundary conditions are

specified. Imposing the derivative boundary conditions and considering $\sigma_{ij}n_j = t_i$, we find that Equation (7.33) becomes

$$\begin{aligned}\int_{\Omega_q}(\widehat{W}_I\rho\ddot{u}_i + \widehat{W}_I c\dot{u}_i + \widehat{W}_{I,j}\sigma_{ij})\mathrm{d}x - \int_{\Gamma_{qi}}\widehat{W}_I t_i\mathrm{d}\Gamma - \int_{\Gamma_{qu}}\widehat{W}_I t_i\mathrm{d}\Gamma \\ = \int_{\Gamma_{qt}}\widehat{W}_I\overline{t}_i\mathrm{d}\Gamma + \int_{\Omega_q}\widehat{W}_I b_i\mathrm{d}\Omega\end{aligned} \tag{7.34}$$

Equation (7.34) shows that the traction (derivative) boundary conditions have been incorporated naturally into the local weak-form of the system equation. There is no need for another set of equations to enforce the derivative boundary conditions.

7.7.3 Discretized formulations for dynamic analysis

The global problem domain Ω is represented by a set of distributed nodes. In the dynamic analysis, **u** is a function of both space co-ordinate and time. Only the equations for the space coordinates are discretized. Using the RPIM and MLS shape functions, we have

$$\begin{aligned}\mathbf{u}(\mathbf{x},t) &= \begin{Bmatrix}u(\mathbf{x},t)\\ v(\mathbf{x},t)\end{Bmatrix} = \sum_{j=1}^{n}\begin{bmatrix}\phi_j(\mathbf{x}) & 0\\ 0 & \phi_j(\mathbf{x})\end{bmatrix}\begin{Bmatrix}u_j(t)\\ v_j(t)\end{Bmatrix} \\ &= \mathbf{\Phi}(\mathbf{x})_{(2\times 2n)}\mathbf{u}(t)_{(2n\times 1)}\end{aligned} \tag{7.35}$$

where $\mathbf{u}(t)$ is the vector of nodal displacements at time t, $\mathbf{\Phi}$ is the matrix of shape functions. Substituting Equation (7.35) into the strong-form Equation (7.31) and local weak-form Equation (7.34) , using the same procedure as in Section 7.3, we can obtain the following discretized system equations for the Ith field node.

$$\mathbf{M}_I\ddot{\mathbf{u}}(t) + \mathbf{C}_I\dot{\mathbf{u}}(t) + \mathbf{K}_I\mathbf{u}(t) = \mathbf{F}_I(t) \tag{7.36}$$

where **u** is the vector of nodal displacements (or nodal displacement parameters) for nodes in the support domain of the Ith field node. Detailed formulations of $\mathbf{K}_I$ and $\mathbf{F}_I$ have been presented in Section 7.3. The nodal mass matrix $\mathbf{M}_I$ is defined as

$$\mathbf{M}_I = \begin{cases}\mathbf{M}_I^{(w)} = \int_{\Omega_q}\rho\widehat{\mathbf{W}}_I^{\mathrm{T}}\mathbf{\Phi}\mathrm{d}\Omega, & \Omega_q(\mathbf{x}_I)\cap\Gamma_t \neq \varnothing \\ \mathbf{M}_I^{(s)} = -\rho\mathbf{\Phi}, & \Omega_q(\mathbf{x}_I)\cap\Gamma_t = \varnothing\end{cases} \tag{7.37}$$

and the nodal damping matrix $\mathbf{C}_I$ is defined as

$$\mathbf{C}_I = \begin{cases} \mathbf{C}_I^{(w)} = \int\limits_{\Omega_q} c\widetilde{\mathbf{W}}_I^{\mathrm{T}}\mathbf{\Phi}\mathrm{d}\Omega, & \Omega_q(\mathbf{x}_I)\cap\Gamma_t \neq \varnothing \\ \mathbf{C}_I^{(s)} = -c\mathbf{\Phi}, & \Omega_q(\mathbf{x}_I)\cap\Gamma_t = \varnothing \end{cases} \tag{7.38}$$

Equation (7.36) presents 2 linear equations for the *I*th field node. Using Equations (7.36) for all *N* field nodes in the entire problem domain, and assembling all these 2*N* equations, we can obtain the final global system equations in the following matrix form.

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{C}\dot{\mathbf{U}} + \mathbf{K}\mathbf{U} = \mathbf{F} \tag{7.39}$$

Equation (7.39) is the system equation of the MWS method for dynamic analyses of two-dimensional solids. Solving this equation, we can obtain displacements for all field nodes and then retrieve all the stresses at any point in the problem domain using again the RPIM or MLS shape functions.

7.7.3.1 Free vibration analysis

For free vibration analysis, the aims are to obtain the natural frequencies and the corresponding vibration modes. Therefore, no damping and body force need be considered. The displacement $\mathbf{u}(\mathbf{x}, t)$ can be written as a harmonic function of time as follows

$$\begin{aligned} \mathbf{u}(\mathbf{x},t) &= \begin{Bmatrix} \hat{u}(\mathbf{x})\sin(\omega t+\varphi) \\ \hat{v}(\mathbf{x})\sin(\omega t+\varphi) \end{Bmatrix} \\ &= \sum_{j=1}^{n} \begin{bmatrix} \phi_j(\mathbf{x}) & 0 \\ 0 & \phi_j(\mathbf{x}) \end{bmatrix} \begin{Bmatrix} \hat{u}_j \\ \hat{v}_j \end{Bmatrix} \sin(\omega t+\varphi) \\ &= \mathbf{\Phi}(\mathbf{x})\hat{\mathbf{u}}\sin(\omega t+\varphi) \end{aligned} \tag{7.40}$$

where ω is the natural frequency and φ is the phase of the harmonic motion, $\hat{u}$ and $\hat{v}$ are the amplitudes for displacement components in x and y directions, respectively.

Substituting Equation (7.40) into the strong-form and the local weak-form, we can obtain the final system equation in terms of the amplitudes of the modal displacements for free vibration analysis.

$$(\mathbf{K} - \omega^2\mathbf{M})\hat{\mathbf{U}} = \mathbf{0} \tag{7.41}$$

where $\hat{\mathbf{U}}$ is the vector of amplitudes of all nodal displacements or displacement parameters when the MLS shape functions are used. Equation (7.41) can also be written in the following typical eigenvalue equation

$$(\mathbf{K} - \lambda\mathbf{M})\mathbf{q} = 0 \tag{7.42}$$

where $\lambda = \omega^2$ is so-called eigenvalue, and $\mathbf{q}$ is the eigenvector. This equation can be solved using a standard eigenvalue solver to obtain eigenvalues λ_i (i=1, 2,…, N) and the corresponding $\mathbf{q}_i$. The natural frequencies of the structures are then given by $\omega_i = \sqrt{\lambda_i}$. The vibration modes (or shapes of the vibration modes) correspond to the eigenvectors.

Note that in MWS-MLS, because the nodal displacement parameters are first obtained, the eigenvector $\mathbf{q}$ obtained is also for the nodal parameters. The MLS shape functions should be used again to obtain the true eigenvector, e.g. using the subroutine, GetDisplacement, given in Chapter 4.

7.7.3.2 Direct analysis of forced vibration

The system equation of forced vibration analysis is given in Equation (7.39). The methods of solving Equation (7.39) are similar to those in FEM, and fall into two categories: modal analysis and direct analysis (see, e.g., GR Liu and Quek, 2002). The direct analysis methods are utilized in this chapter. Several direct analysis methods have been used to solve the dynamic Equation (7.39), such as the well-known central difference method (CDM) and the Newmark method (see, e.g., Petyt, 1990; GR Liu and Quek, 2002). The standard Newmark method is used in the following numerical examples.

The Newmark method is a generalization of the linear acceleration method. This method assumes that the acceleration varies linearly within the time interval (t, t+Δt), which gives

$$\ddot{\mathbf{u}}_{t+\Delta t} = \ddot{\mathbf{u}}_t + \frac{1}{\Delta t}(\ddot{\mathbf{u}}_{t+\Delta t} - \ddot{\mathbf{u}}_t)\tau \qquad \text{for } 0 \le \tau \le \Delta t \tag{7.43}$$

where $0 \le t \le \Delta t$, and

$$\dot{\mathbf{u}}_{t+\Delta t} = \dot{\mathbf{u}}_t + [(1-\delta)\ddot{\mathbf{u}}_t + \delta\ddot{\mathbf{u}}_{t+\Delta t}]\Delta t \tag{7.44}$$

$$\mathbf{u}_{t+\Delta t} = \mathbf{u}_t + \dot{\mathbf{u}}_t\Delta t + [(\frac{1}{2}-\beta)\ddot{\mathbf{u}}_t + \beta\ddot{\mathbf{u}}_{t+\Delta t}]\Delta t^2 \tag{7.45}$$

The response at time t+Δt is obtained by evaluating the equation of motion at time t+Δt. The Newmark method is, therefore, an implicit method. For coding purpose, the flowchart of the Newmark method is given in Figure 7.19.

The Newmark method is unconditionally stable provided that

$$\begin{cases} \delta \ge 0.5 \\ \beta \ge \dfrac{1}{4}(\delta + 0.5)^2 \end{cases} \tag{7.46}$$

One can find that $\delta = 0.5$ and $\beta = 0.25$ lead to acceptable results for most problems considered. Therefore, $\delta = 0.5$ and $\beta = 0.25$ are used in this chapter.

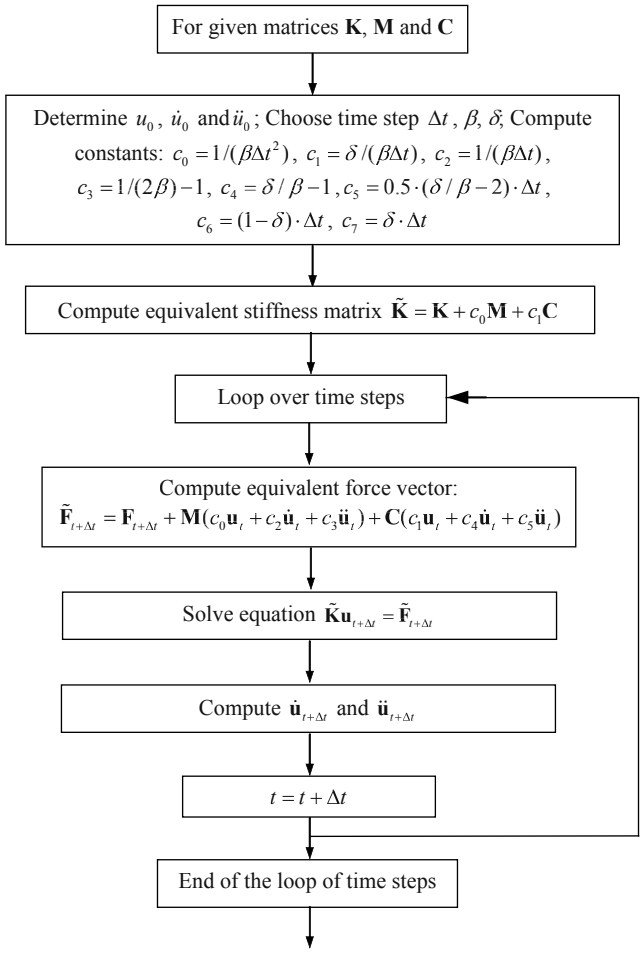


Figure 7.19. Flowchart of the Newmark algorithm for solving a set of forced vibration equations.

7.7.4 Numerical examples

Several numerical examples of two-dimensional elastodynamics are studied to examine the efficiency and performance of the MWS method for dynamic analyses. The standard international (SI) units are used in

following examples unless specially mentioned. For simplicity, the MWS method based on the MLS approximation (MWS-MLS) is used in the following numerical examples. Results of dynamic analysis by the MWS method based on RPIM (MWS-RPIM) can be obtained by replacing the MLS shape functions with the RPIM shape functions.

7.7.4.1 Free vibration analysis

The present MWS method is used for the free vibration analysis of the cantilever beam shown in Figure 4.5. The parameters are the same as those in the example in Sub-section 7.6.4. The mass density of the beam is ρ=1.0. Three kinds of nodal arrangements (55 regular nodes, 189 regular nodes and 189 irregular nodes) are used. In the free vibration analyses, α_i=3.5 is used for the influence domain to construct MLS shape functions.

Frequencies of three nodal distributions obtained by the MWS method are listed in Table 7.6. The results obtained by the FEM commercial software package, ANSYS, using bi-linear rectangular elements with the same number of nodes are listed in the same table. This table shows that the results of the present MWS method are in good agreement with those obtained using FEM. The convergence of the MWS method is also demonstrated in Table 7.6. As the number of nodes increases, results obtained by the present MWS method approach to the exact reference results obtained using the FEM with an extremely fine mesh.

The first six eigenmodes obtained by the MWS-MLS method are plotted in Figure 7.20. Comparing with FEM (ANSYS) results, they are almost identical.

Frequencies results of the beam modeled with 189 irregular nodes are listed in Table 7.6. This table shows that good results are obtained using the irregular distribution nodal arrangement. The stability and high accuracy in the results for irregular nodal distributions are significant features of the present MWS method.

7.7.4.2 Forced vibration analysis

The forced vibration of the same cantilever beam shown in Figure 4.5 is analyzed. The parameters are the same as in the example in Sub-section 7.6.4. For simplicity, the mass density of the beam is ρ=1.0.

In this numerical example for the forced vibration analysis, the beam in subjected to a parabolic traction at the free end, P=1000$g(t)$, where $g(t)$ is the time function. Two functions of $g(t)$ shown in Figure 7.21 are considered. A total 189 uniformed nodes, as shown in Figure 4.12(a), are used to discretize the problem domain. Displacements and stresses for all nodes are obtained using the MWS-MLS method. Detailed results of vertical

displacement, v_A, at the middle point A at the free end of the beam are presented.

Table 7.6. Natural frequencies of the cantilever beam obtained using MWS-MLS and FEM with different nodal distributions

Mode	55 nodes		189 nodes			Reference (FEM 4850 DOFs*)
	MWS	FEM	MWS (regular nodes)	MWS (irregular nodes)	FEM	
1	26.7693	28.60	27.8370	27.7909	27.76	27.72
2	141.3830	144.12	141.1300	141.3111	141.79	140.86
3	179.7013	179.77	179.9077	179.9932	179.82	179.71
4	327.0243	328.47	323.8497	323.0334	328.01	323.89
5	527.3999	523.36	522.3307	522.7783	534.23	523.43
6	539.0598	532.41	537.1464	537.4757	538.08	536.57
7	730.1131	716.35	727.2628	727.5968	751.15	730.04
8	886.5635	859.23	881.5703	881.7091	887.69	881.28
9	896.9009	875.84	896.1059	897.2380	920.36	899.69
10	1004.7952	956.34	997.7824	998.1700	1022.78	1000.22

* DOF—degree of freedom

- **Dynamic relaxation**

If $g(t)$ is a step-function , as shown in Figure 7.21, the long time response should approach the static results for the beam subjected to a static force. This approach of the dynamic analysis is the so-called dynamic relaxation, which can be used as one of the means of examining the stability and accuracy of a numerical procedure.

In our problem, a constant loading is suddenly loaded to this structure, and then kept unchanged. If the damping is neglected, a steady harmonic vibration should be observed with the static deformation (given by the static analysis) as the equilibrium position. If damping is considered, the response should converge to the static deformation.

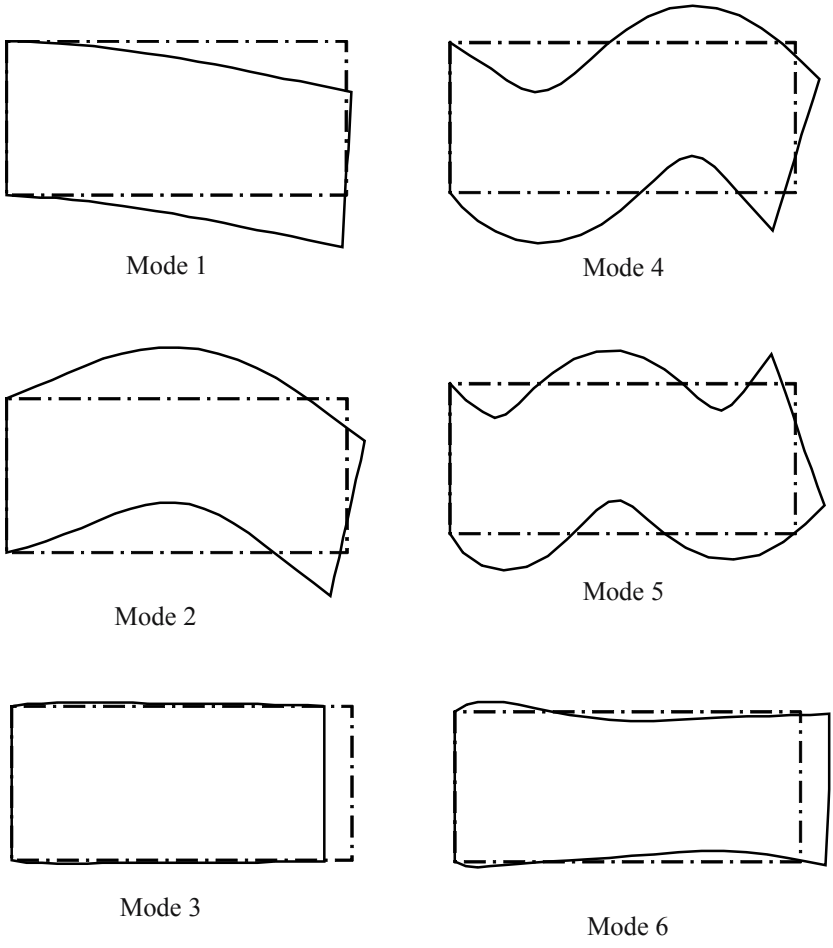


Figure 7.20. Vibration modes for the cantilever beam using the MWS-MLS method and 189 irregular nodes.

The present MWS-MLS method is used to perform the dynamic relaxation analysis; the time step is $\Delta t = 4 \times 10^{-3}$ is used. The response of the vertical displacement, v_A, at the middle point at the free end of the beam is first computed with no damping; the response is a steady harmonic vibration with respect to the static deformation, whose analytical value (see, Section 4.5) is $v_A = 0.0089$.

The same results for c=0.4 are then computed. Table 7.7 lists results of several time steps near $t = 50\text{s}$. MWS gives stable and convergent results, as shown in Figure 7.22. The response converges to $v_A = 0.00885$. Compared with the exact static solution of $v_A = 0.0089$, the error is about 0.5%.

- **Transient response**

The transient response of the beam subjected to a triangular loading P=1000$g(t)$ is now considered. The function $g(t)$ is shown in Figure 7.21(b). The present MWS-MLS method is used to obtain the transient response with and without damping (c=0). The Newmark method is used in this analysis. The result for c=0 is plotted in Figure 7.23 and Figure 7.24. Many time steps are calculated to examine the stability and accuracy of the MWS-MLS method and code. Figure 7.24 shows that the response becomes a stabilized harmonic vibration at about 1.0s. A stable result is obtained using the MWS-MLS method.

The result for c=0.4 is plotted in Figure 7.25. The amplitude of the vibration decreases with time because of the effects of damping; a stable and accurate result is obtained.

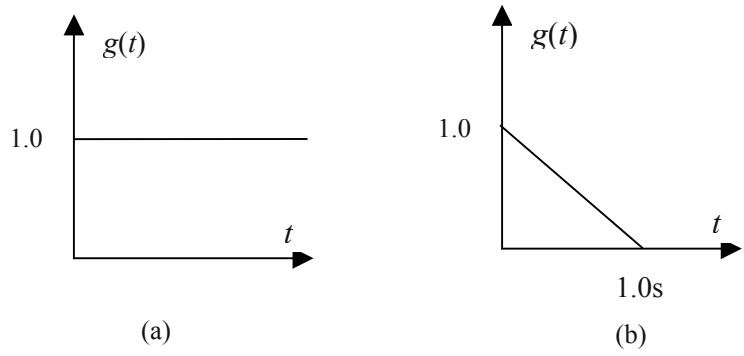


Figure 7.21. Time function $g(t)$:
(a) time-step function; (b) triangular-pulse function.

Table 7.7. Results of displacements v_A excited by the time-step load (damping coefficient c=0.4, several time steps near t=50s)

No. of time step	Time (s)	Displacement v_A
11875	0.475000E+02	-0.00883255
12000	0.480000E+02	-0.00883264
12125	0.485000E+02	-0.00882592
12250	0.490000E+02	-0.00883220
12375	0.495000E+02	-0.00884123
12500	0.500000E+02	-0.00884174

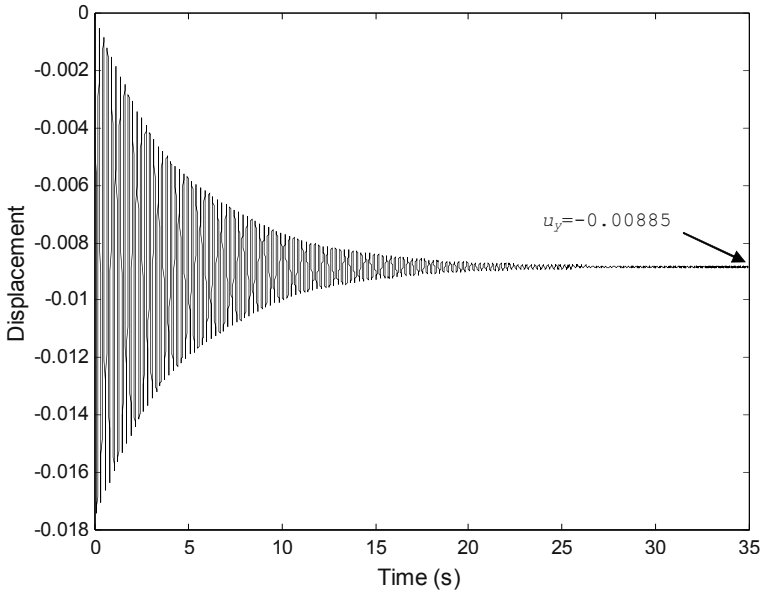


Figure 7.22. Displacements v_A at the middle point at the free end of the beam excited by the time-step load (damping coefficient c=0.4).

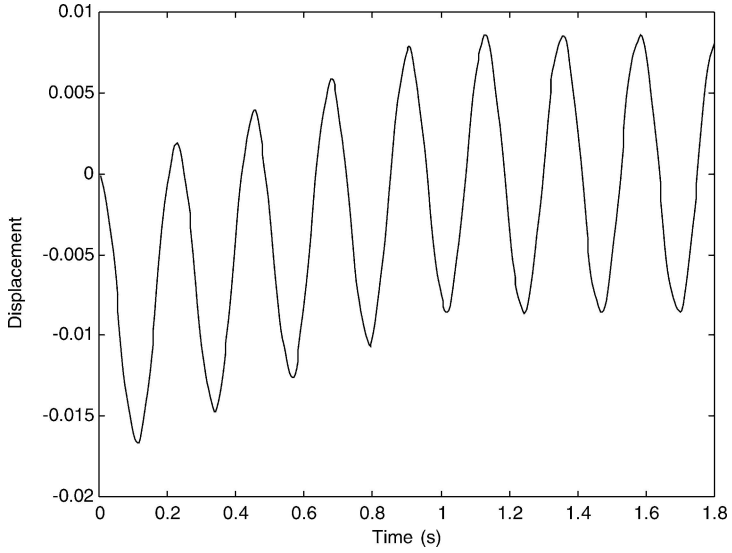


Figure 7.23. Early transient response of the displacement v_A at the middle point at the free end of the beam excited by the triangular-pulse load (damping coefficient c=0).

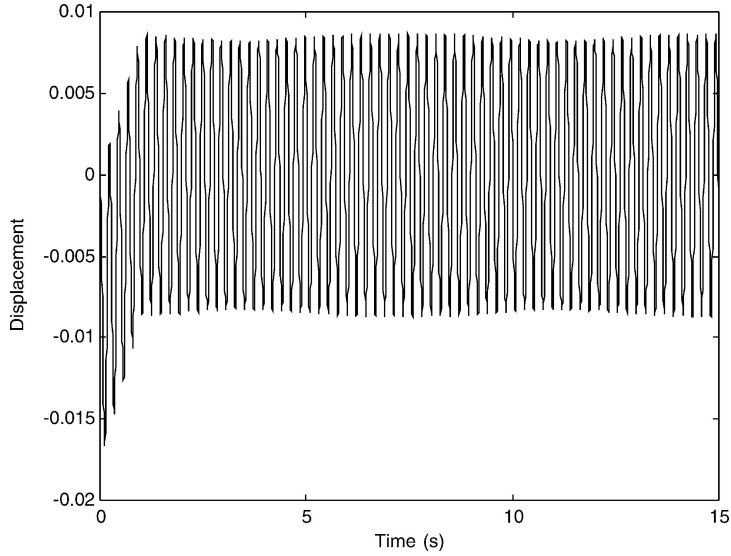


Figure 7.24. Long time response of the displacement v_A at the middle point at the free end of the beam excited by the triangular-pulse load (damping coefficient c=0).

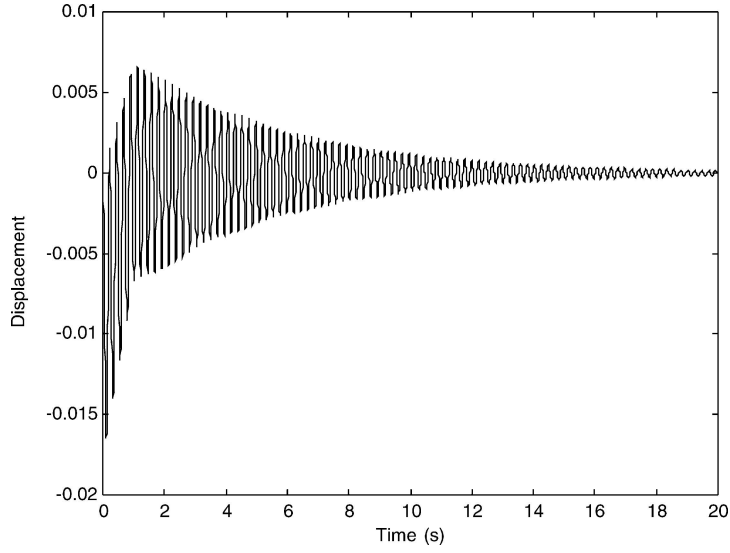


Figure 7.25. Transcient displacement v_A at the middle point at the free end of the beam excited by the triangular-pulse load (damping coefficient c=0.4).

7.8 ANALYSIS FOR INCOMPRESSIBLE FLOW PROBLEMS

The MFree Weak-Strong (MWS) form method has been applied to fluid dynamics problems by GR Liu and Wu et al.(2004). Based on their work, this section introduces the detailed formulations of MWS for incompressible fluids and some examples. No source code will be provided for the fluid problems, as we are still in the process of improving the code. The purpose of this section is to demonstrate that the MWS method can be easily formulated and works well for simulating fluid flows.

7.8.1 Simulation of natural convection in an enclosed domain

7.8.1.1 Governing equations and boundary conditions

The problem domain is given in Figure 7.26. The standard set of governing equations of natural convection in an enclosed domain in terms of vorticity and stream function can be written in the Cartesian coordinate system as follow (Hughes and Brighton, 1991).

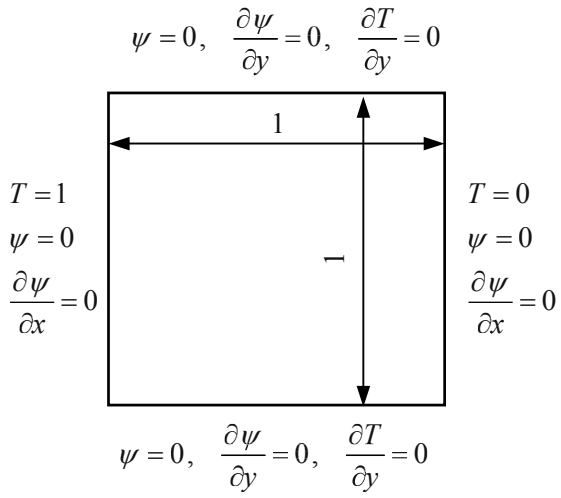


Figure 7.26. Schematic drawing of the problem domain for the natural convection problem.

- The stream function equation is:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \omega \tag{7.47}$$

- The vorticity equation is:

$$u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \Pr(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}) - \Pr \cdot \mathrm{Ra} \cdot \frac{\partial T}{\partial x} \tag{7.48}$$

- The heat transfer equation is

$$u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \tag{7.49}$$

where ω, ψ, T, Pr, and Ra are, respectively, the vorticity, stream function, temperature, Prandtl number and Rayleigh number, and u, v are the components of velocity in the x and y directions, which can be calculated using the stream function.

$$\begin{cases} u = \dfrac{\partial \psi}{\partial y} \\ v = -\dfrac{\partial \psi}{\partial x} \end{cases} \tag{7.50}$$

The boundary conditions are listed as follows:

$$1)\ x = 0,\ 0 \le y \le 1 \colon T = 1, \psi = 0,\ \frac{\partial \psi}{\partial x} = 0, \tag{7.51}$$

$$2)\ x = 1,\ 0 \le y \le 1 \colon T = 0, \psi = 0,\ \frac{\partial \psi}{\partial x} = 0, \tag{7.52}$$

$$3)\ y = 0,\ 0 \le x \le 1 \colon \frac{\partial T}{\partial y} = 0, \psi = 0,\ \frac{\partial \psi}{\partial y} = 0, \tag{7.53}$$

$$4)\ y = 1,\ 0 \le x \le 1 \colon \frac{\partial T}{\partial y} = 0, \psi = 0,\ \frac{\partial \psi}{\partial y} = 0\ . \tag{7.54}$$

There are two types of boundary conditions: Dirichlet and Neumann.

7.8.1.2 Discretized system equations

For RPIM or the MLS shape functions, the discretized equation of the MWS method for natural convection can be written as:

1) For the collocatable nodes, the following strong-form of discretized equations (for the Ith node) is used.

For the stream function equation,

$$\sum_{k=1}^{n}(\phi_k)_{,xx}\psi_k+\sum_{k=1}^{n}(\phi_k)_{,yy}\psi_k=\omega_I \tag{7.55}$$

For the vorticity equations,

$$\begin{aligned}u_I\sum_{k=1}^{n}(\phi_k)_{,x}\omega_k+v_I\sum_{k=1}^{n}(\phi_k)_{,y}\omega_k=&-Ra\,\mathrm{Pr}\sum_{k=1}^{n}(\phi_k)_{,x}T_k\\&+\mathrm{Pr}\left(\sum_{k=1}^{n}(\phi_k)_{,xx}\omega_k+\sum_{k=1}^{n}(\phi_k)_{,yy}\omega_k\right)\end{aligned} \tag{7.56}$$

For the heat transfer equations,

$$u_I\sum_{k=1}^{n}(\phi_k)_{,x}T_k+v_I\sum_{k=1}^{n}(\phi_k)_{,y}T_k=\sum_{k=1}^{n}(\phi_k)_{,xx}T_k+\sum_{k=1}^{n}(\phi_k)_{,yy}T_k \tag{7.57}$$

The velocities are computed using stream function values:

$$u_I=\sum_{k=1}^{n}(\phi_k)_{,x}\psi_k\ ,\ v_I=\sum_{k=1}^{n}(\phi_k)_{,y}\psi_k \tag{7.58}$$

where n is the number of nodes used for constructing the MFree shape functions, u_I, and v_I are the components of velocity for the Ith collocatable node in the x and y directions, respectively.

2) For DBR-nodes, the following local weak-form (for the Ith node) is used:

For the stream function equation,

$$C_{Ik}\psi_k-E_{Ik}\psi_k=-A_{Ik}\omega_k \tag{7.59}$$

For the vorticity equations,

$$B_{Ik}\omega_k+\mathrm{Pr}\cdot C_{Ik}\omega_k-\mathrm{Pr}\cdot E_{Ik}\omega_k=-\mathrm{Pr}\cdot\mathrm{Ra}\cdot D_{Ik}T_k \tag{7.60}$$

For the heat transfer equations,

$$B_{Ik}T_k+C_{Ik}T_k-E_{Ik}T_k=0 \tag{7.61}$$

In Equations (7.59)-(7.61),

$$A_{Ik}=\iint_{\Omega_q}\phi_k\widehat{W}_I\,d\Omega \tag{7.62}$$

$$B_{Ik} = \iint_{\Omega_q} [u \cdot \frac{\partial \phi_k}{\partial x} + v \cdot \frac{\partial \phi_k}{\partial y}] \cdot \widehat{W}_I d\Omega \tag{7.63}$$

$$C_{Ik} = \iint_{\Omega_q} (\frac{\partial \phi_k}{\partial x} \frac{\partial \widehat{W}_I}{\partial x} + \frac{\partial \phi_k}{\partial y} \frac{\partial \widehat{W}_I}{\partial y}) d\Omega \tag{7.64}$$

$$D_{Ik} = \iint_{\Omega_q} \frac{\partial \phi_k}{\partial x} \widehat{W}_I d\Omega \tag{7.65}$$

$$E_{Ik} = \int_{\Gamma_{qu}} \frac{\partial \phi_k}{\partial \mathbf{n}} \widehat{W}_I d\Gamma \tag{7.66}$$

where $\widehat{W}_I(\mathbf{x})$ is the test function centered by the Ith node, and Ω_q is the local quadrature domain of the Ith node. The single integration E_{ik} along Γ_{qu} is implemented appropriately according to different essential boundary conditions for ω, ψ and T. The double integration for A_{Ik}, B_{Ik}, C_{Ik}, and D_{Ik} can be evaluated by Gauss quadrature using the transformation strategy (GR Liu, 2002). Note that all these integrations can be carried out over the local domain with a regular shape centered at the Ith node.

Equations (7.55)-(7.61) are used for all the field nodes, which gives a set of discretized system equations for the entire domain.

3) For a field node on the essential boundary, the essential boundary conditions for ψ and T can be simply given as follows:

$$\begin{cases} \psi_I = 0, \text{ when node } I \text{ is on the whole wall boundary} \\ T_I = 1, \text{ when node } I \text{ is on the hot wall} \\ T_I = 0, \text{ when node } I \text{ is on the cool wall} \end{cases} \tag{7.67}$$

The essential boundary condition can be directly imposed using the direct interpolation method discussed in Sub-section 5.3.2.

4) The boundary condition for vorticity ω:

There is no explicit boundary condition for the vorticity. However, it is found that the implementation of the vorticity condition is equivalent to the approximation of the second order derivatives of the stream function at the boundary. Therefore, the Dirichlet boundary condition for vorticity ω can be interpreted as a Neumann boundary condition for the stream function ψ. Thus, the boundary condition for vorticity can be derived by taking the local weak-form of Equations (7.47) on the wall boundary, as shown in Equation (7.59), i.e.

$$-A_{Ik}\omega_k = (C_{Ik}\psi_k - E_{Ik}\psi_k) \tag{7.68}$$

7.8.1.3 Numerical results for the problem of natural convection

The resultant algebraic Equations (7.55)~(7.61) are a set of non-linear equations. Therefore, an iterative loop is needed. The iteration is stopped, when the L_∞ norm of the residuals for ψ , ω and T in Equations (7.55)~(7.57) and Equations (7.59)~(7.61) are less than 10^{-3}.

Four different nodal distributions shown in Figure 7.27 are used for the square cavity problem to examine the efficiency of the present MWS method. To compare quantitatively the computational accuracy of the present MWS method with that of other methods, such as MLPG, LRPIM, and FDM, the following quantities are calculated.

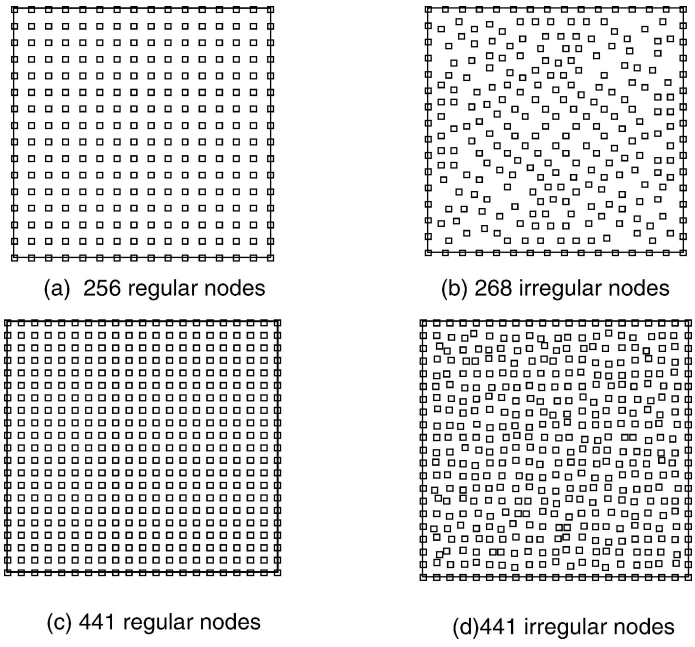


Figure 7.27. Different nodal distributions used for the square cavity problem of natural convection.

1) $|\psi_{\max}|$: maximum absolute value of the stream function

2) $u_{\max}$: maximum horizontal velocity on the vertical mid-plane of the cavity

3) $v_{\max}$: maximum vertical velocity on the horizontal mid-plane of the cavity
4) $Nu_{\max}$: maximum value of the local Nusselt number on the boundary at x=0
5) $Nu_{\min}$: minimum value of the local Nusselt number on the boundary at x=0

where Nu is the local *Nusselt* number

$$Nu\big|_{x=0} = -\frac{\partial T}{\partial x}\bigg|_{x=0} \tag{7.69}$$

The energy norm , E_r, is defined as an error indicator:

$$E_r = \frac{1}{5}\sum_{j}^{5}\left(\frac{\xi_j^{\text{num}} - \xi_j^{\text{exact}}}{\xi_j^{\text{exact}}}\right)^2 \tag{7.70}$$

where ξ_j^{num} and ξ_j^{exact} (j=1~5) represent, respectively, the five quantities computed using the numerical methods and using the exact solutions. Since there is no analytical solution for the problem, the benchmark numerical solution of Davis (1983) is adopted as the exact solution.

The main feature of the MFree methods is that the numerical solution can be obtained using irregularly distributed nodes. To determine the maximum and minimum variable values in the whole problem domain as well as post-processing the results (after the converged solution on field nodes have been obtained), the function values on a fine uniform mesh of 101×101 are calculated. This can be done using the corresponding interpolation procedure which was used in the discretization process for the methods. It is noted that the uniform mesh of 101×101 is independent of the implementation for different methods, as it is only used for the post-visualization. In the following, all the results shown in the tables and figures are based on the function values on this post-visualization mesh of 101×101 resolution.

First, we compare the rates of convergence and corresponding CPU time required for the present MWS, MLPG, LRPIM and FDM for Ra=10^3, using the same uniform nodal distribution. For comparison, all the parameters in the MFree interpolation schemes are kept the same for the MWS methods and other MFree methods. For example, the dimensionless size of influence domain α_i for the MLS scheme is taken as 3.0 for both MWS-MLS and MLPG. The dimensionless shape parameter α_c, shape parameter q, and the number of nodes in the support domain n in RPIM-MQ scheme are taken as $\alpha_c = 8.0$, q=1.03, n=30 respectively for both MWS-RPIM and the LRPIM.

Figure 7.28 shows the convergence results obtained numerically, where h is the nodal spacing. We find the following conclusions.

1) The MFree methods are more accurate than FDM, and their convergence rates are also better than that of FDM.
2) The MWS methods are less accurate than the corresponding MFree local weak-form methods (i.e. LRPIM and MLPG) when the same number of nodes is used. In other words, the MWS-MLS is less accurate than MLPG, and MWS-RPIM is less accurate than the LRPIM method. This finding is the same as that obtained for solid mechanics presented in the previous sections.
3) The MLS-based MFree method is less accurate than the RPIM-based MFree method for this problem. This finding is opposite to that for solid mechanics problems.
4) The MWS methods (i.e. MWS-RPIM and MWS-MLS) have slightly slower convergence rates than the corresponding MFree local weak-form methods (i.e. LRPIM and MLPG).
5) MWS-RPIM has better convergence rate than the MWS-MLS. This finding is also opposite to that for solid mechanics problems.

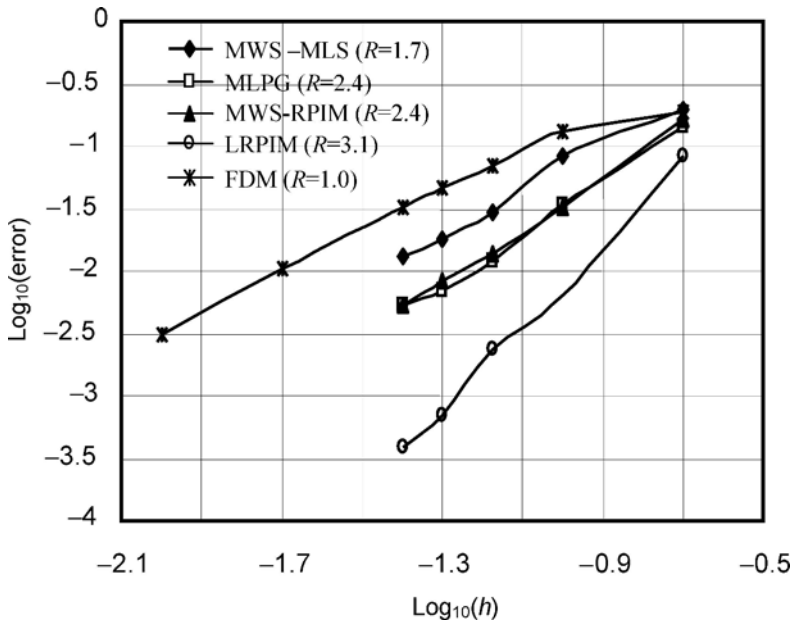


Figure 7.28. Comparison of the convergence rates, R, for different methods for the natural convection problem.

It should be noted that the accuracy of MWS-RPIM and LRPIM depend on the proper choice of the shape parameters of RBF. For present MWS-RPIM (MQ) to analyze the problem of natural convection with Ra=10^3 and 10^4, α_c can be chosen 6~9 for n=20~30 (n is the number of field nodes selected in the support domain). For Ra=10^5, α_c should be around 1.0, and n should not be larger than 12 to achieve good accuracy. Therefore, the choice of these parameters depends also on the Rayleigh number of the fluid problems. Because the same model of nodes is used for problems with different Rayleigh number, an adaptive scheme is required.. The MWS-RPIM achieves this adaptivity by changing the shape parameters and the number of local nodes.

Figure 7.29 and Figure 7.30 show the running time against the number of field nodes, N, in the problem domain used in MWS, MLPG and LRPIM. The running time is obtained by running the codes on a Compaq Alpha-server supercomputer. The number of field nodes N corresponds to the different nodal spacing d_c (or h).

In the simulation, it is found that neither the MWS-MLS nor MLPG achieve convergent results using the iterative scheme to solve the algebraic equations. Therefore, the algebraic equations have to be solved using a modified Gaussian elimination procedure at each iteration step. Figure 7.30 shows that the running time of the MWS-MLS is much less than that for MLPG. This is because, in MLPG, CPU time is consumed in constructing the shape function for the Gauss points inside the quadrature domain for each field node. In MWS-MLS, however, the strong-form equation is used for all the collocatable nodes that are the majority of all the nodes. Therefore, only the shape functions need to be computed for these field nodes. These shape functions can be determined first and stored for use in the entire iteration process, which reduces computational cost greatly. If the number of nodes is large, the direct solver adopted by both MWS-MLS and MLPG becomes computationally expensive.

Similarly, the MWS-RPIM spends much less running time on calculating the shape function for Gauss points and numerical integration than LRPIM. More importantly, it is found that a stationary iterative scheme, such as SOR scheme, can be used in MWS-RPIM to solve the algebraic equations systems. Therefore, the computational complexity for MWS-RPIM is only about $O(N)$. On the other hand, although LRPIM can achieve high accuracy using less nodes, the weak-form equation over every field node does not make the traditional stationary iterative scheme (such as Gauss-Seidel, SOR scheme) converge. In conclusion, a more expensive direct solver has to be used to solve the algebraic equations; the computational complexity is $O(N^3)$ because the matrices are unsymmetric, as shown in Figure 7.30. Therefore, MWS-RPIM (MQ) is more efficient than LRPIM, especially for

solving large scale problem. This is a unique feature of the MWS-RPIM method for fluid problem, which we did not find for solid problems.

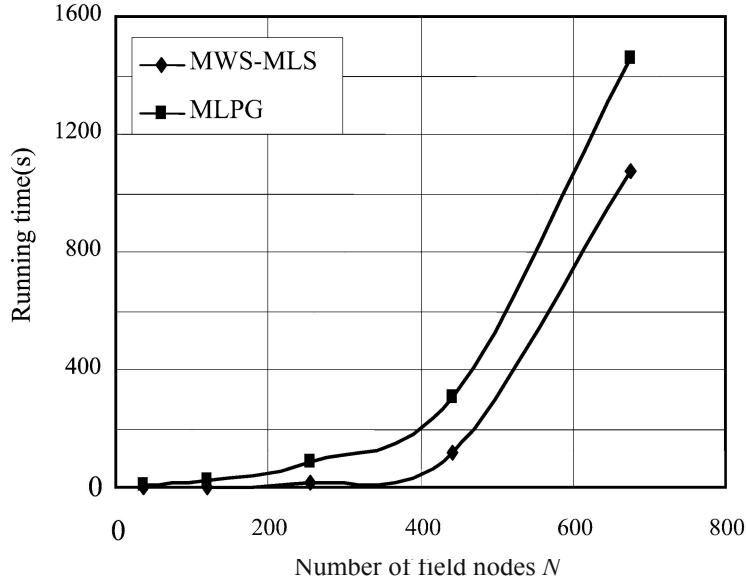


Figure 7.29. Comparison of running time required by the MWS-MLS and MLPG for the natural convection problem.

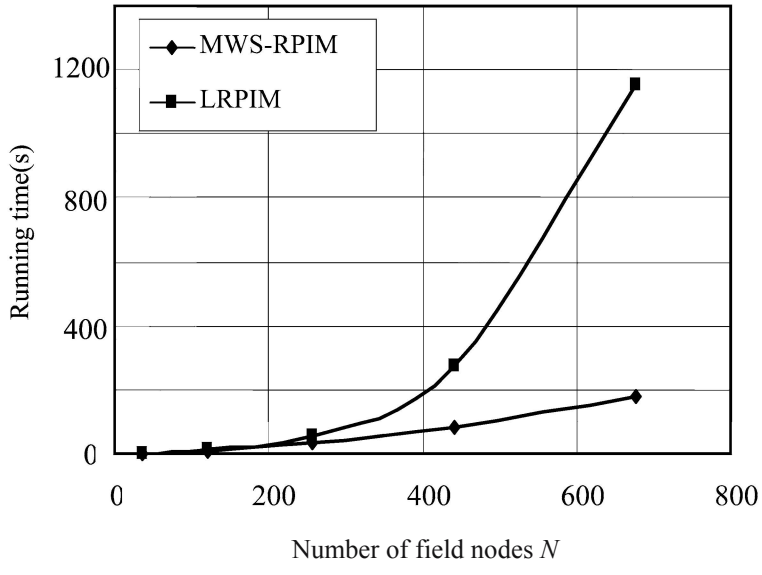


Figure 7.30. Comparison of running time required by the MWS-RPIM and LRPIM.

Table 7.8~Table 7.10 list the numerical results for different sets of nodes for Rayleigh numbers 10^3,10^4,10^5 respectively. For all the sets of nodes, the results of MWS agree well with the benchmark solution given by Davis (1983). The streamlines and isotherms for Ra=10^3, 10^5 are shown in Figure 7.31~Figure 7.32.

Table 7.8. Comparison of numerical results for the problem of natural convection in the square cavity (Ra=10^3)

Method	Nodal distribution	Results (difference % with Davis's solution)				
		$\lvert\psi_{\max}\rvert$	$u_{\max}$	$v_{\max}$	$Nu_{\max}$	$Nu_{\min}$
MWS-MLS	256 regular nodes	1.117 (-4.86)	3.546 (-2.82)	3.609 (-2.38)	1.477 (-1.86)	0.706 (2.02)
	268 irregular nodes	1.140 (-2.90)	3.696 (1.29)	3.594 (-2.79)	1.498 (-0.47)	0.718 (3.76)
MWS-RPIM	256 regular nodes	1.196 (1.87)	3.681 (0.88)	3.734 (1.00)	1. 528 (1.53)	0.684 (-1.16)
	268 irregular nodes	1.192 (1.53)	3.688 (1.07)	3.731 (0.92)	1.525 (1.33)	0.686 (-0.87)
Davis (1983)		1.174	3.649	3.697	1.505	0.692

Table 7.9 Comparison of numerical results for the problem of natural convection in the square cavity (Ra=104)

Method	Nodal distribution	Results (difference % with Davis's solution)				
		$\lvert\psi_{\max}\rvert$	$u_{\max}$	$v_{\max}$	$Nu_{\max}$	$Nu_{\min}$
MWS-MLS	256 regular nodes	4.809 (-5.17)	15.752 (-2.63)	18.698 (-4.68)	3.609 (2.30)	0.581 (-0.85)
	268 irregular nodes	4.963 (-2.13)	16.689 (3.16)	19.427 (-0.97)	3.746 (6.18)	0.543 (-7.34)
MWS-RPIM	256 regular nodes	5.169 (1.93)	16.373 (1.21)	20.017 (2.04)	3.756 (6.46)	0.577 (-1.54)
	268 irregular nodes	5.174 (2.03)	16.447 (1.66)	20.071 (2.31)	3.740 (6.01)	0.580 (-1.02)
Davis (1983)		5.071	16.178	19.617	3.528	0.586

Table 7.10. Comparison of numerical results for the problem of natural convection in the square cavity ($Ra=10^5$)

Method	Nodal distribution	Results (difference % with Davis's solution)				
		$\lvert\psi_{max}\rvert$	u_{max}	v_{max}	Nu_{max}	Nu_{min}
MWS-MLS	441 regular nodes	9.463 (-1.55)	36.787 (5.92)	61.431 (-10.4)	8.772 (13.67)	0.713 (-2.19)
	441 irregular nodes	10.098 (5.06)	36.689 (5.64)	70.093 (2.19)	10.597 (37.32)	0.743 (1.92)
MWS-RPIM	441 regular nodes	9.772 (1.66)	35.209 (1.38)	66.044 (-3.71)	10.070 (30.49)	0.699 (-4.12)
	441 irregular nodes	9.918 (3.18)	37.863 (9.02)	64.964 (-5.29)	8.507 (10.24)	0.579 (-20.5)
Davis (1983)		9.612	34.730	68.590	7.717	0.729

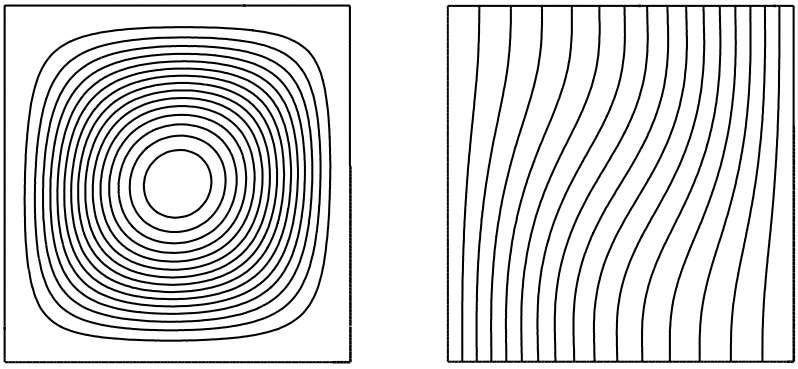


Figure 7.31. Streamlines and isotherms for the cavity flow ($Ra=10^3$) obtained using the MWS-MLS and 268 irregularly distributed nodes.

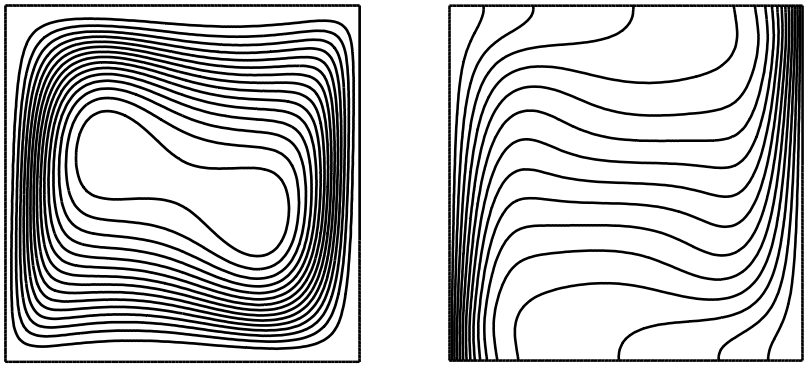


Figure 7.32. Streamlines and isotherms for cavity flow ($Ra=10^5$) obtained using the MWS-RPIM and 441 irregularly distributed nodes.

7.8.2 Simulation of the flow around a cylinder

The incompressible, viscous fluid flow around a circular cylinder is a classical problem in fluid mechanics. Despite the simplicity of the cylinder geometry, the flow field is in fact very complex in nature. Because of its relevance to engineering problems and importance in the fundamental understanding of fluid flows, numerous theoretical, numerical and experimental investigations on a fluid flow passing a circular cylinder have been reported in the past century. It serves as a good sample problem for validating a new numerical method for unsteady two-dimensional Navier-Stokes equations. In the sub-section, the MWS method is used to solve this sample problem.

7.8.2.1 Governing equation and boundary condition

Consider an incompressible, viscous fluid flow at a constant velocity U_∞ in the x direction passing a stationary cylinder of radius a, as shown Figure 7.33.

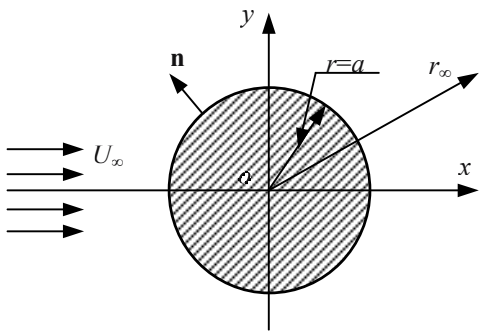


Figure 7.33. Configuration of a fluid flow around a circular cylinder.

The standard dimensionless two-dimensional Navier-Stokes equations for dynamic fluid flows in the vorticity-stream function form are as follows

The equation for the stream function is

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \omega \tag{7.71}$$

The equation for the vorticity is

$$\frac{\partial \omega}{\partial t}+u\frac{\partial \omega}{\partial x}+v\frac{\partial \omega}{\partial y}=\frac{1}{\mathrm{Re}}(\frac{\partial^2 \omega}{\partial x^2}+\frac{\partial^2 \omega}{\partial y^2}) \tag{7.72}$$

where Re is Reynolds number defined as

$$\mathrm{Re}=\frac{U_\infty D}{\nu} \tag{7.73}$$

where D is the cylinder diameter, and ν is the kinematic viscosity.

The boundary conditions of the problem are:

i) Free stream velocity U at the in-flow boundary:

$$\begin{cases}\psi = U_\infty \cdot y \\ \omega = 0\end{cases} \tag{7.74}$$

ii) Non-slip condition slip on the surface of the cylinder;

$$\begin{cases}\omega = \dfrac{\partial^2 \psi}{\partial n^2} \\ \psi = 0\end{cases} \tag{7.75}$$

where **n** is the unit outward normal on the surface of the cylinder (See, Figure 7.33)

iii) Uniform flow at $x=-\infty$ and $y=\pm\infty$.

$$\begin{cases}\omega = 0 \\ \psi = \psi\big|_{\text{uniform flow}}\end{cases} \tag{7.76}$$

iv) Zero-gradient condition at $x=+\infty$

$$\begin{cases}\dfrac{\partial \omega}{\partial x}=0 \\ \dfrac{\partial \psi}{\partial x}=0\end{cases} \tag{7.77}$$

The initial condition for the flow field is assumed and computed using the following formulae, i.e.

$$\psi\big|_{t=0}=\sqrt{x^2+y^2} \tag{7.78}$$

which serves as an artificial initiator for the numerical iteration to solve the non-linear problem.

With the same notation as in Sub-section 7.8.1, the discretized strong-forms for the equations of the stream function and vorticity, respectively, at a collocatable node can be written as follwos:

$$\sum_{k=1}^{n}(\phi_k)_{,xx}\psi_k+\sum_{k=1}^{n}(\phi_k)_{,yy}\psi_k=\omega_I \tag{7.79}$$

$$\begin{aligned}\frac{d\omega_I}{dt}+u_I\sum_{k=1}^{n}(\phi_k)_{,x}\omega_k+v_I\sum_{k=1}^{n}(\phi_k)_{,y}\omega_k\\ =\frac{1}{\mathrm{Re}}\left(\sum_{k=1}^{n}(\phi_k)_{,xx}\omega_k+\sum_{k=1}^{n}(\phi_k)_{,yy}\omega_k\right)\end{aligned} \tag{7.80}$$

where n is the number of nodes used for constructing the MFree shape functions.

The discretized equations in local weak-form for a DBR-node can be written as follows.

For the equation of the stream function,

$$C_{Ik}\psi_k-E_{Ik}\psi_k=-A_{Ik}\omega_k \tag{7.81}$$

For the equation of the vorticity,

$$\frac{d\omega_I}{dt}+B_{Ik}\omega_k+\frac{1}{\mathrm{Re}}\cdot C_{Ik}\omega_k-\frac{1}{\mathrm{Re}}\cdot E_{Ik}\omega_k=0 \tag{7.82}$$

where $A_{Ik},B_{Ik},C_{Ik},E_{Ik}$ are defined in Equations (7.62)~(7.66). As discussed in Sub-section 7.8.1, the boundary condition for vorticity can be discretized as in Equation (7.68).

For this unsteady fluid flow problem, there is a time derivative in Equations (7.79)~(7.82). In the present model, the time derivative is approximated using an explicit three-step formulation based on a Taylor series expansion in time; this is a kind of difference method. From Taylor's series, a function f in time can be written as

$$f(t+\Delta t)=f(t)+\Delta t\frac{\partial f(t)}{\partial t}+\frac{\Delta t^2}{2}\frac{\partial^2 f(t)}{\partial t^2}+\frac{\Delta t^3}{6}\frac{\partial^3 f(t)}{\partial t^3}+O(\Delta t^4) \tag{7.83}$$

where Δt is the time interval. Approximating Equation (7.83) up to third-order accuracy, we can write the three-step formulation as:

$$f(t+\frac{\Delta t}{3})=f(t)+\frac{\Delta t}{3}\frac{\partial f(t)}{\partial t} \tag{7.84}$$

$$f(t+\frac{\Delta t}{2}) = f(t) + \frac{\Delta t}{2}\frac{\partial f(t+\Delta t/3)}{\partial t} \tag{7.85}$$

$$f(t+\Delta t) = f(t) + \Delta t\frac{\partial f(t+\Delta t/2)}{\partial t} \tag{7.86}$$

7.8.2.2 Computation procedure

To solve the resultant set of non-linear algebraic equations for the unsteady fluid flow problem, a time-matching iterative procedure is used. The procedure adopted here includes the following steps:

1) assume that at time t= 0 the unsymmetrical initial flow field is given as

$$\begin{cases} \psi\big|_{t=0} = \sqrt{x^2+y^2} \\ \omega\big|_{t=0} = 0 \end{cases} \tag{7.87}$$

2) calculate the unknown field values of velocities u and v using Equation (7.58);

3) solve the vorticity equations that are built using Equation (7.80) or (7.82) using three-step time marching scheme given in Equations (7.84)-(7.86);

4) solve the stream-function equations that are built using Equation (7.79) or (7.81) by SOR iteration scheme until the L_∞ norm of residuals for ψ is less than 10^{-2}, because the accuracy of the stream-function is very important for a stable simulation.

5) the procedure is repeated until the prescribed time-step or the final time is reached.

7.8.2.3 Results and discussion

Simulations of small and moderate Reynolds number flow (Re=20 and Re=100, respectively) are carried out using the present MWS method. The computational domain is shown in Figure 7.34, where a is the radius of the cylinder.

Two different types of nodal distributions are adopted, as shown in Figure 7.35. In these two nodal distributions, the nodes within the area $r = \sqrt{x^2+y^2} \le 3.5$ are generated by MFree2D©. The region is distributed by regular nodes in model I (Figure 7.35(a)) and by irregularly scattered nodes in model II (Figure 7.35(b)). Both model I and model II contain many field

nodes. For simplicity, only MWS-RPIM (MQ) is used to simulate this problem. The dimensionless shape parameter α_c, shape parameter q, and the number of nodes in the support domain n in present RPIM-MQ scheme are $\alpha_c = 4.0$, q=1.03, n=20 respectively.

For Re=20, the unsymmetrical initial flow field becomes symmetrical and the flow appears to be laminar steady flow as shown in Figure 7.36; for Re=100, the flow field eventually settles into a periodic oscillatory pattern. The fine sequences for the vorticity are shown in Figure 7.37 and the streamlines of the fluid flow are plotted in Figure 7.38. The pattern of the fluid flow has been confirmed by other experimental and numerical results. It is generally agreed that in two dimensions the vortex shedding begins at a critical Reynolds number around 49. For Reynolds numbers less than the critical value ($\mathrm{Re}_{\mathrm{critical}}$=49), the introduced perturbation is gradually dissipated by viscosity. Above this critical Reynolds number, the introduced perturbation will trigger the vortex shedding process to form a Von Karman vortex street, as given in Figure 7.37.

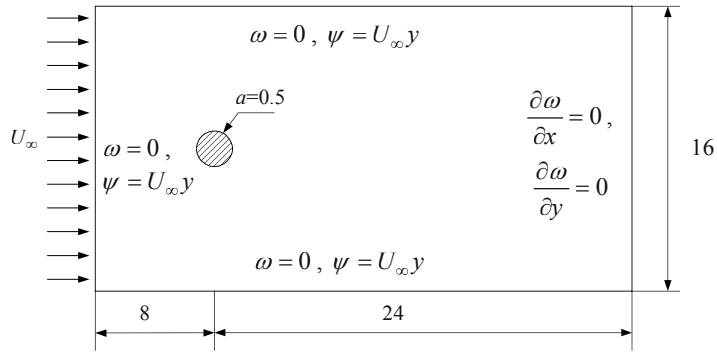


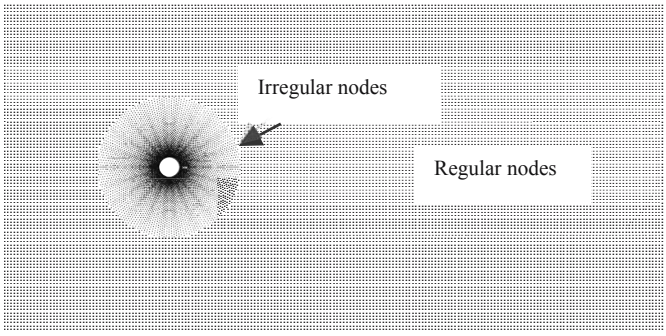
Figure 7.34. Problem domain for the simulation of the fluid flow around a circular.

Figure 7.36 shows the streamlines for Re=20 when the flow reaches its final steady state. In Figure 7.36, a pair of stationary recirculating eddies develops behind the cylinder. The length of the recirculating region, L, from the rearmost point of the cylinder to the end of the wake, the separation angle agree θ_s , and the drag coefficient C_D are compared with previous computational and experimental data as listed in Table 7.11. The geometrical and dynamical parameters agree well with those in the literature.

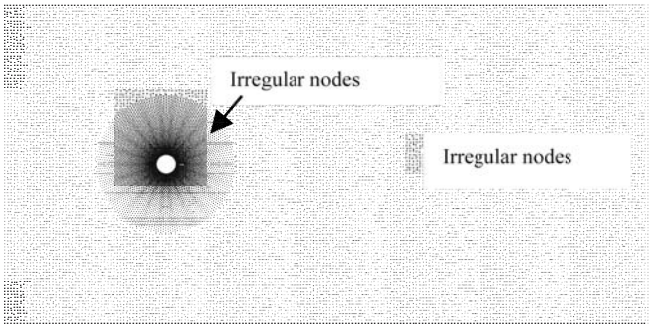
Figure 7.38 shows time-dependent behavior of streamline contours for Re=100. Figure 7.37 and Figure 7.38 show that the most attractive feature of the vortex shedding behind a circular cylinder, the periodic variation of the flow field, has been successfully reproduced. The two characteristic parameters, the drag and lift coefficients, are

$$\begin{cases} C_D = \dfrac{\mathbf{F} \cdot x}{\rho U^2 a} \\ C_L = \dfrac{\mathbf{F} \cdot y}{\rho U^2 a} \end{cases} \tag{7.88}$$

where **F** is the total force acting on the circular cylinder, which arises from the surface pressure and shear stress. Figure 7.39 shows these two parameters at a late stage. The flow is periodically oscillatory; the lift coefficient oscillates more strongly than the drag coefficient. The drag coefficient varies nearly twice as fast as the lift coefficient. This is because of the drag coefficient is affected by vortex shedding processes from both sides of the cylinder.



(a) model I



(b) model II

Figure 7.35. Two types of nodal distributions used in the numerical simulation using the MWS-RPIM.

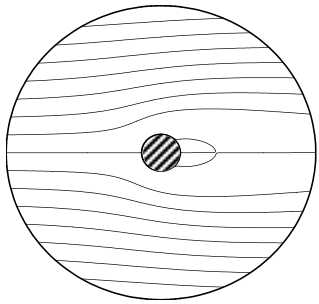


Figure 7.36. Streamlines of the fluid flow near the cylinder at the final steady state for Re=20.

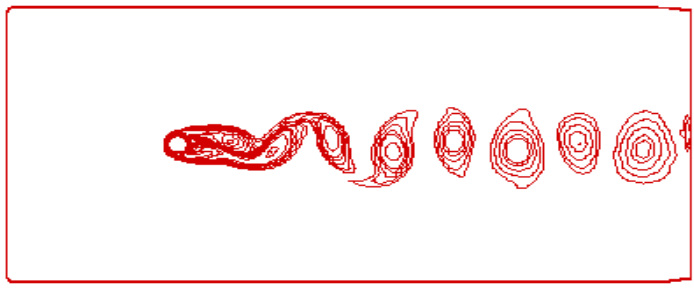
Table 7.11. Comparison of geometrical and dynamical parameters with those in the literature

Sources	Results		
	L/a	θ_s	C_D
MWS-RPIM (Model Ⅰ)	1.86	43.21	2.076
MWS-RPIM (Model Ⅱ)	1.84	44.74	2.103
Dennis and Chang (1980)	1.88	43.7	2.045
Nieuwstadt and Keller (1973)	1.786	43.37	2.053

Table 7.12. Comparison of the average $\bar{C}_D$, and S_t

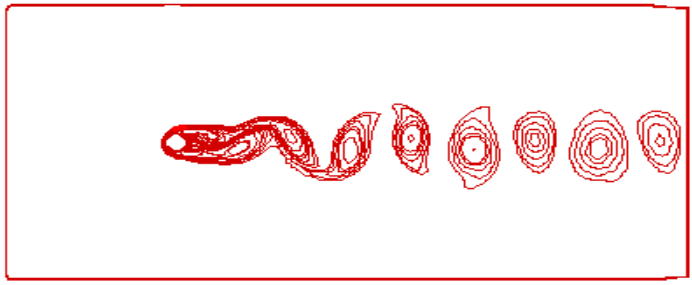
	Results	
	$\bar{C}_D$	S_t
MWS-RPIM (Model Ⅰ)	1.257	0.167
MWS-RPIM (Model Ⅱ)	1.273	0.167
Jordan and Fromm(1972)	1.28	-
Braza et al. (1986)	1.28	0.16
He and Doolen (1997)	1.287	0.161

The average drag coefficient and Strouhal number ($S_t = fD/U$, where f is the shedding frequency) are listed in Table 7.12. The vortex shedding frequency is obtained by measuring the final period of the lift coefficient. All the results agree well.

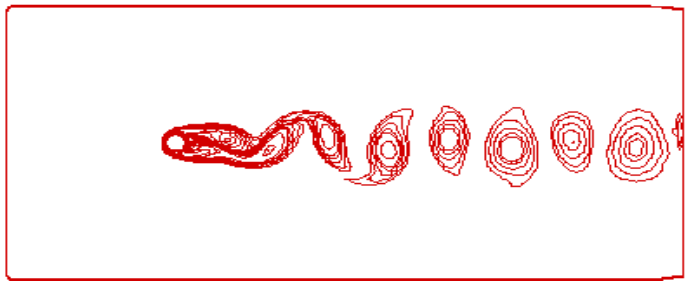


(d) t_0

(d) t_0+2s



(d) t_0+4s



(d) t_0+6s

Figure 7.37. Vorticity distribution for the fluid flow around a cylinder (Re=100) after the steady state at t_0.

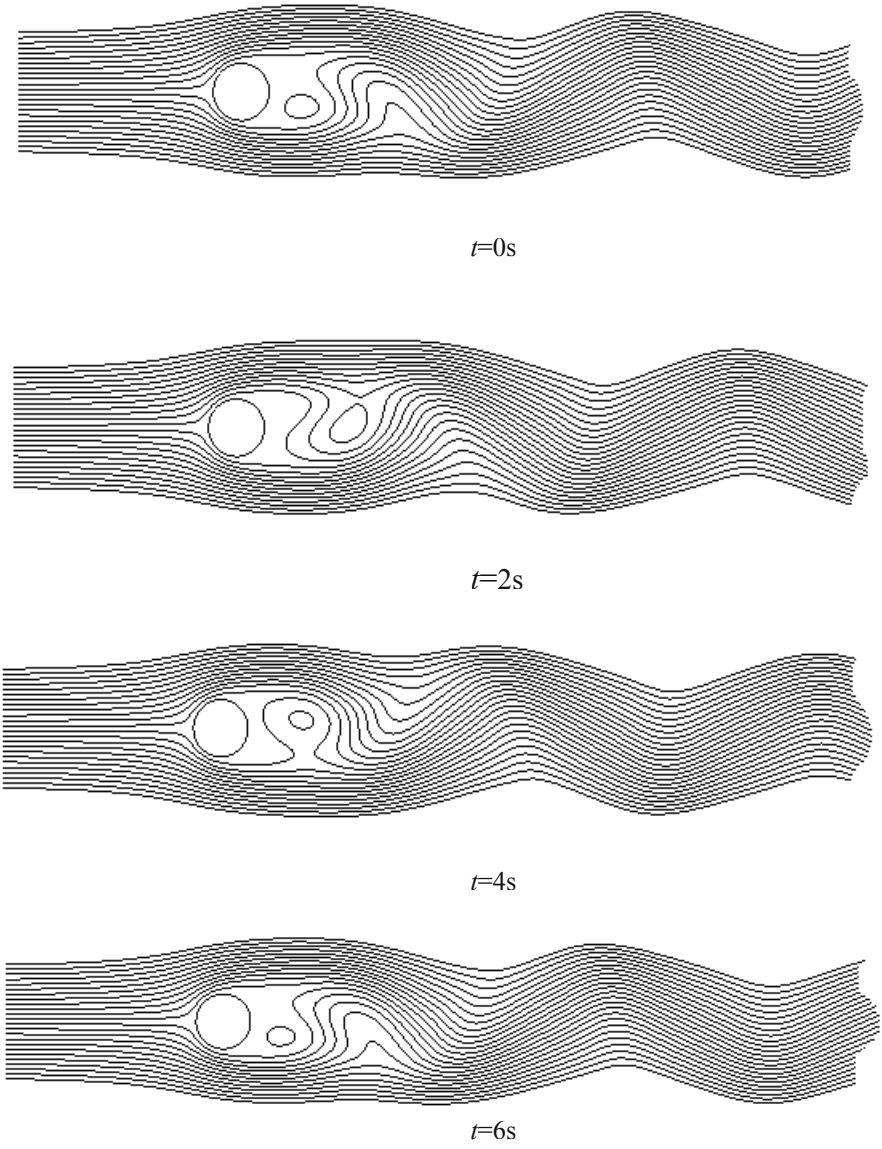


Figure 7.38. Time-evolution of streamlines of the fluid flow around a cylinder for Re=100 (Model I).

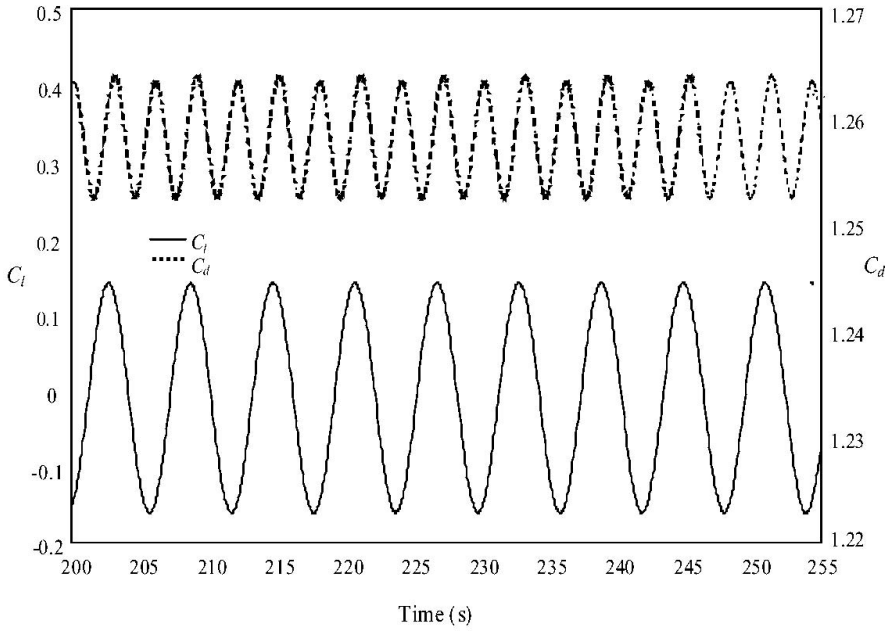


Figure 7.39. Time-evolution of Lift and Drag coefficients for Re=100 (Model I).

7.9 REMARKS

In this Chapter, the MFree weak-strong (MWS) form method was presented for problems of solid and fluid mechanics. In MWS, both the strong-form and the Petrov-Galerkin local weak-form are used. The strong-form with collocation method is used for the collocatable nodes, whose local quadrature domains do not intersect with derivative boundaries. No numerical integration is needed for these nodes. The local weak-form is used only for the DBR-nodes that are on or near the derivative boundaries, and the derivative boundary conditions can then be easily imposed together with the system equations to produce stable and accurate solutions. The MWS method was illustrated for problems of statics, free and forced vibration of structures, and incompressible flow. It performed well. The following remarks may be made.

1) MWS-MLS is more efficient than MLPG for both the solid and fluid mechanics problems tested.
2) MWS-RPIM is far more efficient than LRPIM, especially for the fluid mechanics problems tested.
3) MLS shape functions perform better than RPIM shape functions in solid mechanics. However, RPIM shape functions are better in fluid mechanics.

MWS provides an alternative avenue to develop new MFree methods and adaptive analysis for the numerical analysis of problems in solid and fluid mechanics.

APPENDIX

Appendix 7.1. Major subroutines used in MFree_MWSl.f90 (for solid mechanics problem only) and their functions

Subroutines	Functions	Location
Input	Input data from the external data file	Program 5.3
Qdomain	Construct the quadrature domain for a field node	Program 5.4
GaussCoefficient	Obtain coefficients of Gauss points	Program 4.5
DomainGaussPoints	Compute the array of the information of Gauss points for a quadrature domain	Program 5.5
SupportDomain	Determine the support domain for a quadrature point	Program 4.7
RPIM_ShapeFunc_2D (MLS_ShapeFunc_2D)	Construct shape functions and their derivatives.	Program 3.1 (Program 3.9)
TestFunc	Compute the quartic spline weight function	Program 7.2
Integration_BCQuQi	Perform boundary the integration on Γ_{qu} and Γ_{qi}	Program 5.7
Integration_BCQt	Perform boundary the integration on Γ_{qt}	Program 5.8
EssentialBC	Enforce essential boundary conditions	Program 5.9
SolverBand	Solve system equations	Program 4.12
GetDisplacement	Compute the finial displacements	Program 5.10
GetNodeStress	Compute the stress components for field nodes	Program 5.11
Output	Output results	Program 5.12
TotalGaussPoints	Compute the matrix of information of Gauss points for the global cells	Program 5.13
GetEnergyError	Compute global error in the energy norm	Program 5.14
GetInvasy	Compute the inversion for a matrix	Program 4.15
Dobmax	Compute multiplication of two matrices	Program 5.15

Appendix 7.2. The data file, Input189.dat, used in MFree_MWS.f90

```
*L,H,E,v,P,
48.  12.  3.e7 .3 1000.
*numnode
189
* Global BC: Xmin,Xmax,Ymax, Ymin
0.  48.  6.  -6.
* Nodal spacing: Dcx,Dcy
2.4 1.5
* Local quadrature domain: Aqx,Aqy
1.5 1.5
* Num.  of sub-partitions: Nsx,Nsy
2   2
*Influence domain
3.
*Num.  of Gauss Points
4
*RBF shape parameters: nRBF ALFc, dc and q
1  4.0  2.4  1.03
*Num.  of Basis
3
*Field nodes: x[xi,yi]

      1    .00000   6.00000
      2    .00000   4.50000
      3    .00000   3.00000
      4    .00000   1.50000
      5    .00000    .00000
      6    .00000  -1.50000
      7    .00000  -3.00000
      8    .00000  -4.50000
      9    .00000  -6.00000
     10   2.40000   6.00000
 .
 .
 .
    180  45.60000  -6.00000
    181  48.00000   6.00000
    182  48.00000   4.50000
    183  48.00000   3.00000
    184  48.00000   1.50000
    185  48.00000    .00000
    186  48.00000  -1.50000
    187  48.00000  -3.00000
    188  48.00000  -4.50000
    189  48.00000  -6.00000

*Num.  of Essential BC: numFBC
9
*Node,iUx,iUy,Ux,Uy
 1  1 1   0.000000E+00 -0.599999E-04
 2  1 1  -0.628906E-05 -0.337499E-04
 3  1 1  -0.718749E-05 -0.149999E-04
 4  1 1  -0.449218E-05 -0.374999E-05
 5  1 1   0.000000E+00  0.000000E+00
 6  1 1   0.449218E-05 -0.374999E-05
 7  1 1   0.718749E-05 -0.149999E-04
 8  1 1   0.628906E-05 -0.337499E-04
 9  1 1  0.000000E+00 -0.599999E-04
*Num. Concentrated loading: numFBC
```

```
9
*Node,iTx,iTy,Tx,Ty
   189    1    1   0.00000   0.0
   188    1    1   0.00000   0.0
   187    1    1   0.00000   0.0
   186    1    1   0.00000   0.0
   185    1    1   0.00000   0.0
   184    1    1   0.00000   0.0
   183    1    1   0.00000   0.0
   182    1    1   0.00000   0.0
   181    1    1   0.00000   0.0
* Num.  of nodes and cells(for en.  error)
189  160
*Nodes for cells: xc[  ]
      1    .00000   6.00000
      2    .00000   4.50000
      3    .00000   3.00000
      4    .00000   1.50000
      5    .00000    .00000
      6    .00000  -1.50000
      7    .00000  -3.00000
      8    .00000  -4.50000
      9    .00000  -6.00000
     10   2.40000   6.00000
      .
      .
      .
    180  45.60000  -6.00000
    181  48.00000   6.00000
    182  48.00000   4.50000
    183  48.00000   3.00000
    184  48.00000   1.50000
    185  48.00000    .00000
    186  48.00000  -1.50000
    187  48.00000  -3.00000
    188  48.00000  -4.50000
    189  48.00000  -6.00000

*No.  of nodes in cells[1,2,3,4]

    1      1      2     11     10
    2      2      3     12     11
    3      3      4     13     12
    4      4      5     14     13
    5      5      6     15     14
      .
      .
      .
  156    175    176    185    184
  157    176    177    186    185
  158    177    178    187    186
  159    178    179    188    187
  160    179    180    189    188


*END of data file
```

Appendix 7.3. A output sample for stress obtained using MWS-RPIM

No. of field nodes	σ_{xx}	σ_{yy}	τ_{xy}
82	0.11007E+04	-0.21716E+01	-0.93984E+01
83	0.82845E+03	-0.10702E+00	-0.57459E+02
84	0.55283E+03	-0.66389E+00	-0.99072E+02
85	0.27649E+03	-0.36205E+00	-0.12285E+03
86	0.21699E-09	-0.14311E-06	-0.13079E+03
87	-0.27649E+03	0.36205E+00	-0.12285E+03
88	-0.55283E+03	0.66389E+00	-0.99072E+02
89	-0.82845E+03	0.10702E+00	-0.57459E+02
90	-0.11007E+04	0.21717E+01	-0.93984E+01
91	0.99841E+03	-0.19636E+01	-0.93281E+01
92	0.75146E+03	-0.95292E-01	-0.57053E+02
93	0.50145E+03	-0.60194E+00	-0.98374E+02
94	0.25079E+03	-0.32842E+00	-0.12198E+03
95	-0.17631E-07	-0.44176E-07	-0.12987E+03
96	-0.25079E+03	0.32842E+00	-0.12198E+03
97	-0.50145E+03	0.60194E+00	-0.98374E+02
98	-0.75146E+03	0.95292E-01	-0.57053E+02
99	-0.99841E+03	0.19636E+01	-0.93281E+01
100	0.89681E+03	-0.17655E+01	-0.92725E+01
101	0.67499E+03	-0.86294E-01	-0.56689E+02
102	0.45042E+03	-0.54133E+00	-0.97744E+02
103	0.22528E+03	-0.29571E+00	-0.12120E+03
104	-0.48267E-08	-0.88478E-08	-0.12904E+03
105	-0.22528E+03	0.29571E+00	-0.12120E+03
106	-0.45042E+03	0.54133E+00	-0.97744E+02
107	-0.67499E+03	0.86294E-01	-0.56689E+02
108	-0.89681E+03	0.17655E+01	-0.92725E+01
109	0.79582E+03	-0.15750E+01	-0.92171E+01
110	0.59899E+03	-0.77155E-01	-0.56361E+02
111	0.39970E+03	-0.47954E+00	-0.97182E+02
112	0.19991E+03	-0.25727E+00	-0.12051E+03
113	0.47603E-08	-0.37796E-08	-0.12830E+03
114	-0.19991E+03	0.25727E+00	-0.12051E+03
115	-0.39970E+03	0.47954E+00	-0.97182E+02
116	-0.59899E+03	0.77155E-01	-0.56361E+02
117	-0.79582E+03	0.15750E+01	-0.92171E+01
		Error in the energy norm:=	0.538919E-01

*The parameters used are:

$\alpha_c = 4.0,\ \ q = 1.03$ and $d_c = 2.4$ for MQ-RBF;

$d_{cx} = 2.4,\ \ d_{cy} = 1.5,$ and $\alpha_s = 3.0$ for the local influence domains;

$\alpha_q = 1.5$ and $n_g \times n_g = 2 \times 2$ for local quadrature domains;.

The linear polynomial terms are added in the MQ-RPIM;

The quartic spline function is used as the test function for the local weak form.

Appendix 7.4. A output sample for stress obtained using MLS MWS

No. of field nodes	σ_{xx}	σ_{yy}	τ_{xy}
82	0.11080E+04	0.34628E+00	0.64555E+00
83	0.83117E+03	-0.37000E+00	-0.53871E+02
84	0.55385E+03	0.46561E+00	-0.94146E+02
85	0.27732E+03	-0.42962E+00	-0.11676E+03
86	0.19094E-04	-0.23530E-04	-0.12552E+03
87	-0.27732E+03	0.42965E+00	-0.11676E+03
88	-0.55385E+03	-0.46559E+00	-0.94146E+02
89	-0.83117E+03	0.36996E+00	-0.53871E+02
90	-0.11080E+04	-0.34621E+00	0.64554E+00
91	0.10072E+04	0.23823E+00	-0.15794E+01
92	0.75554E+03	-0.31540E+00	-0.53106E+02
93	0.50342E+03	0.45185E+00	-0.94331E+02
94	0.25217E+03	-0.46547E+00	-0.11705E+03
95	-0.70059E-05	0.85493E-05	-0.12478E+03
96	-0.25217E+03	0.46545E+00	-0.11705E+03
97	-0.50342E+03	-0.45186E+00	-0.94331E+02
98	-0.75554E+03	0.31542E+00	-0.53106E+02
99	-0.10072E+04	-0.23827E+00	-0.15794E+01
100	0.90723E+03	-0.62574E+00	-0.10710E+00
101	0.68009E+03	0.74438E+00	-0.53588E+02
102	0.45397E+03	-0.10080E+01	-0.94175E+02
103	0.22604E+03	0.98990E+00	-0.11682E+03
104	-0.11025E-04	0.12912E-04	-0.12528E+03
105	-0.22604E+03	-0.98992E+00	-0.11682E+03
106	-0.45397E+03	0.10080E+01	-0.94175E+02
107	-0.68009E+03	-0.74437E+00	-0.53588E+02
108	-0.90723E+03	0.62572E+00	-0.10710E+00
109	0.80534E+03	0.57034E+00	0.73683E-01
110	0.60423E+03	-0.59938E+00	-0.53722E+02
111	0.40246E+03	0.75836E+00	-0.94254E+02
112	0.20185E+03	-0.69266E+00	-0.11691E+03
113	0.22259E-04	-0.26170E-04	-0.12530E+03
114	-0.20185E+03	0.69269E+00	-0.11691E+03
115	-0.40246E+03	-0.75833E+00	-0.94254E+02
116	-0.60423E+03	0.59935E+00	-0.53722E+02
117	-0.80534E+03	-0.57027E+00	0.73682E-01
		Error in the energy norm:= **0.1737E-01**	

*The parameters used are

$d_{cx} = 2.4,\ d_{cy} = 1.5,$ and $\alpha_s = 3.0$ for the local influence domains;

$\alpha_q = 1.5$ and $n_g \times n_g = 2 \times 2$ for local quadrature domains;.

The second order polynomial basis (*mbasis*=6) and the quartic spline weight function are used for MLS approximation;

The quartic spline function is used as the test function for the local weak form.

COMPUTER PROGRAMS

Program 7.1. The source code of main program of MFree_MWS.f90

```
!---------------------------------------------------------------------
! The main program--2D FORTRAN 90 CODE-MWS method
! Using rectangular quadrature and influence domains
! input file   -- input189.dat
! output file  -- result.dat
! include file -- variableslocal.h
!---------------------------------------------------------------------
       implicit real*8 (a-h,o-z)
       include 'variableslocal.h'
       ir=4  ! for input data
       open(ir,file='Input189.dat',status='old')
       open(2,file='result.dat',status='unknown')
       maxmatrix=2*ndim
! ************** Input data
       call Input(ir,x,ndim,nx,numnode,xm, nquado,Dmat,&
                  ALFs,numcell,numq,xBK,conn,&
                  nbnum,npEBC,pEBC,nbcnum,nbc,ibcn,bcn)
! ************** Determine influence domains --uniform nodal spacing
       xspace=dcx*dex        ! Size of quadrature domain
       yspace=dcy*dey
       xstep=xspace/dex
       ystep=yspace/dey
       do j=1,numnode
          ds(1,j)=alfs*xstep ! Size of influence domain
          ds(2,j)=alfs*ystep
       enddo
! ************** Coef.  of Gauss points and Weights
       call GaussCoefficient(nquado,gauss)
       eps=1.e-16
       do iak=1,2*numnode
          fk(iak)=0.0
          do jak=1,2*numnode
             ak(iak,jak)=0.
          enddo
       enddo

! ************** Loop for field nodes
       do 100 nod=1,numnode
          write(*,*)'Field Node=',nod
          xn=x(1,nod)
          yn=x(2,nod)
          xss=xspace
          yss=yspace
          numgauss=nquado*nquado
          call QDomain(xss,yss,xn,yn,xm,xc) ! Local quadrature domain
          if((xc(2,1).lt.xm(3)).and.(xc(2,2).gt.xm(4))&
             .and.(xc(1,3).lt.xm(2))) then
! ************** using strong form
             gpos(1)=xn
             gpos(2)=yn
             ndex=0
             call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv) ! support domain
             do kph=1,ndex
                do ii=1,10
                   phi(ii,kph)=0.
                enddo
```

```
            enddo
            call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,ndex,&
                                   alfc,dc,q,nRBF, mbasis)
            ie1=2*nod-1
            ie2=ie1+1
            do ill=1,ndex
               mm=nv(ill)
               m1=2*mm-1
               m2=2*mm
               ak(ie1,m1)=young*(phi(4,ill)+&
                          0.5*(1.-anu)*phi(6,ill))/(1-anu**2)
               ak(ie1,m2)=young*(phi(5,ill)*0.5*(1.+anu))/(1-anu**2)
               ak(ie2,m1)=young*(phi(5,ill)*0.5*(1.+anu))/(1-anu**2)
               ak(ie2,m2)=young*(phi(6,ill)+&
                          0.5*(1.-anu)*phi(4,ill))/(1-anu**2)
            enddo
         else
! ************* using local weak form
            nxc=ng         ! for the rectangular domain
            xgs=(xc(1,4)-xc(1,1))/ngx
            ygs=(xc(2,1)-xc(2,2))/ngy
            x0=xc(1,1)
! ************* Local quadrature domain is divided to sub-partitions
            do 60 iix=1,ngx
               xx=x0+(iix-1)*xgs
               y0=xc(2,1)
               do 60 jjy=1,ngy
                  yy=y0-(jjy-1)*ygs
                  xcc(1,1)=xx
                  xcc(2,1)=yy
                  xcc(1,2)=xx
                  xcc(2,2)=yy-ygs
                  xcc(1,3)=xx+xgs
                  xcc(2,3)=yy-ygs
                  xcc(1,4)=xx+xgs
                  xcc(2,4)=yy
! ************* Gauss points for a sub-partition
                 call DomainGaussPoints(xcc,gauss,gss,nx,ng,nxc,&
                                        nquado,numgauss)
! ************* Loop quadrature points
                  numgauss=nquado*nquado
                  do 30  ie=1,numgauss
                     gpos(1)=gss(1,ie)
                     gpos(2)=gss(2,ie)
                     weight=gss(3,ie)
                     ajac=gss(4,ie)
                     ndex=0
                     call SupportDomain(numnode,nx,gpos,x,ds,ndex,nv)
                     do kph=1,ndex
                        do ii=1,10
                          phi(ii,kph)=0.
                        enddo
                     enddo
                     dsi(1)=xspace
                     dsi(2)=yspace
                     xcent(1)=xn
                     xcent(2)=yn
                     call TestFunc(dsi,xcent,gpos,w,wx,wy) ! test function
                     Call RPIM_ShapeFunc_2D(gpos,x,nv,phi,nx,numnode,&
                                            ndex,alfc,dc,q,nRBF, mbasis)
                     ik1=nod*2-1
                     ik2=nod*2
! ************* Get nodal stiffness matrix and assembling
                     do ine=1,ndex
                       n1=2*nv(ine)-1
                       n2=2*nv(ine)
                       do ii=1,3
                          do jj=1,2
                             bbt(jj,ii)=0.
                             bb(ii,jj)=0.
```

```
                              ww(ii,jj)=0.
                           enddo
                        enddo
                        bb(1,1)=phi(2,ine)
                        bb(2,2)=phi(3,ine)
                        bb(3,1)=phi(3,ine)
                        bb(3,2)=phi(2,ine)
                        ww(1,1)=wx
                        ww(2,2)=wy
                        ww(3,1)=wy
                        ww(3,2)=wx
                        do ii=1,3
                           do jj=1,2
                              bbt(jj,ii)=ww(ii,jj)
                           enddo
                        enddo
                        call dobmax(bbt,2,3,2,dmat,3,3,bd,2)
                        call dobmax(bd,2,3,2,bb,2,3,ek,2)
                        ak(ik1,n1)=ak(ik1,n1)+weight*ajac*ek(1,1)
                        ak(ik1,n2)=ak(ik1,n2)+weight*ajac*ek(1,2)
                        ak(ik2,n1)=ak(ik2,n1)+weight*ajac*ek(2,1)
                        ak(ik2,n2)=ak(ik2,n2)+weight*ajac*ek(2,2)
                     enddo
 30                continue  !End of integ.  for local quadrature domain

! ************* B.C.  Integrations
                   call Integration_BCQt(nx,ng,xcc,f2,x,numnode,nquado,&
                                         xm,xss,yss,xcent)
                   fk(2*nod-1)=fk(2*nod-1)+f2(1)
                   fk(2*nod)=fk(2*nod)+f2(2)
                   call Integration_BCQuQi(nx,ng,nod,xcc,x,numnode,&
                         nquado,dmat,xm,xss,YSS,ak,maxmatrix,alfs,ds)
 60           continue
           endif
 100    continue !   End of loop for field nodes

! ************* Boundary conditions: essential
       call EssentialBC(x,numnode,ak,fk,maxmatrix,ds,alfs,&
                        nbnum,npEBC,pEBC)

! ************* Solve equation to get the solutions
       neq=2*numnode  ! number of equations
       write(*,*)'Solve equation...'
       call SolverBand(ak,fk,neq,maxmatrix)
       do kk=1,numnode
          u2(1,kk)=fk(2*kk-1)
          u2(2,kk)=fk(2*kk)
       enddo
! ************* Get the final displacement
       call GetDisplacement(x,ds,u2,displ,alfs,nx,numnode)
       do kk=1,numnode
          u22(1,kk)=displ(2*kk-1)
          u22(2,kk)=displ(2*kk)
       enddo

! ************* Get stress for field nodes
       call GetNodeStress(x,ds,Dmat,u2,Stress,alfs,nx,numnode)
       call Output(x,numnode,u2,u22,Stress)   ! ouput results

! ************* Get error in the energy norm using global BK cells
       write(*,*)'Computing global error in the energy norm...'
       ngst=numcell*nquado**2
       call TotalGaussPoints(xBK,conn,gauss,gst,nx,ng,&
                             numq,numcell,nquado,ngst)
       call GetEnergyError(nx,ng,xBK,numq,u2,dmat,ds,&
                           ngst,gst,alfs)
       write(*,*)'THE END'
  STOP
  END
```

Program 7.2. The source code of subroutine TestFunc

```
  SUBROUTINE TestFunc(dsi,xcent,xg,w,wxx,wyy)
!--------------------------------------------------------------------
! The quartic spline test (weight) function
! input—dsi: size of weight domain;
!       xcent: center of the weight domain;
!       xg: coordinate of point considered;
! output—w, wxx,wyy
!--------------------------------------------------------------------
       IMPLICIT REAL*8(A-H,O-Z)
       dimension dsi(2),xcent(2)
       dimension xg(2)
       ep=1.e-15
       difx=xg(1)-xcent(1)
       dify=xg(2)-xcent(2)
       if(dabs(difx).le.ep) then
          drdx=0.
       else
          drdx=(difx/dabs(difx))/dsi(1)
       end if
       if (dabs(dify).le.ep) then
          drdy=0.
       else
          drdy=(dify/dabs(dify))/dsi(2)
       end if
       rx=abs(xg(1)-xcent(1))
       ry=abs(xg(2)-xcent(2))
       rx=rx/dsi(1)
       ry=ry/dsi(2)
       wx=1.-6*rx*rx+8.*rx**3-3.*rx**4
       dwx=(-12.*rx+24.*rx**2-12.*rx**3)*drdx
       wy=1.-6*ry*ry+8.*ry**3-3.*ry**4
       dwy=(-12.*ry+24.*ry**2-12.*ry**3)*drdy
       if(rx.gt.1.) wx=0.
       if(ry.gt.1.) wy=0.
       w=wx*wy
       wxx=wy*dwx
       wyy=wx*dwy
  RETURN
  END
```

REFERENCES

Abbassian F, Dawswell DJ and Knowles NC (1987), Free vibration benchmarks. Glasgow: National Engineering Laboratory.

Adey RA and Brebbia CA (1974), FEM solution of effluent dispersion, in Num. Meth. In Fl. Mech. (eds Brebbia and Connor), 325-354, Pentech Press, UK,

Armando DC and Oden JT (1995), Hp clouds-a meshless method to solve boundary value problems. TICAM Report 95-05, University of Texas at Austin.

Atluri SN and Shen SP (2002), The Meshless Local Petrov-Galerkin (MLPG) method. Tech Sciemce Press. Encino USA.

Atluri SN and Zhu T (1998a), A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. Computational Mechanics, 22, 117-127.

Atluri SN and Zhu T (1998b), A new meshless local Petrov-Galerkin (MLPG) approach to nonlinear problems in computer modeling and simulation. Computer Modeling and Simulation in Engineering, 3(3), 187-196.

Atluri SN and Zhu T (2000a), New concepts in meshless methods. Computational Mechanics, 22, 117-127.

Atluri SN and Zhu T (2000b), The meshless local Petrov-Galerkin (MLPG) approach for solving problems in elsto-statics. Computational Mechanics, 25,169-179.

Atluri SN, Cho JY and Kim HG(1999a), Analysis of thin beams, using the meshless local Petrov-Galerkin (MLPG) method, with generalized moving least squares interpolation. Computational Mechanics, 24, 334-347.

Atluri SN, Kim HG and Cho JY(1999b), A critical assessment of the truly meshless local Petrov-Galerkin (MLPG), and Local Boundary Integral Equation (LBIE) methods. Computational Mechanics, 24,348-372.

Atluri SN, Sladek J, Sladek V and Zhu T(2000), Local boundary integral equation (LBIE) and it's meshless implementation for linear elasticity. Computational Mechanics, 25(2), 180-198.

Babuska I and Melenk JM(1997), The partition of unity method. Int J. Numer. Meth Eng., 40(4) 727-758.

Balakrishnan K and Ramachandran PA(2001), Osculatory Interpolation in the Method of Fundamental Solution for Nonlinear Poisson Problems. J. of Comput. Phys., 172, 1-18.

Barrett KE (1974), The numerical solution of singular perturbation boundary value problem. Q. J. Mech. Appl. Math., 27, 57-68.

Batra J (1964), Uber die Naherungsweise Losung einiger Zweidiensionaler Elastizitats aufgben. Z. Angew. Math. Mech. 17, 184-185.

Beissel S and Belytschko T (1996), Nodal integration of the element-free Galerkin method. Computer Methods in Applied Mechanics and Engineering, 139, 49-74.

Belytschko T and Organ D (1995) Coupled finite element-element–free Galerkin method. Computational Mechanics, 17, 186-195.

Belytschko T, Guo Y, Liu WK and Xiao SP(2000), A unified stability of meshless particle methods. Int. J. Numer. Methods Engng. 48,1359-1400.

Belytschko T, Krongauz Y, Organ D and Liu WK (1996b), Smoothing and accelerated computations in the element free Galerkin method. J. of Comp. And Appl. Math., 74, 111-126.

Belytschko T, Krongauz Y, Organ D, Fleming M and Krysl P(1996a), Meshless Method: an overview and recent development. Comput. Meth. Appl. Mech. Eng., 139,3-47., 74, 111-126.

Belytschko T, Krysl P, and Krongauz Y(1997), A three-dimensional explicit element-free Galerkin method. INT J NUMER METH FL, 24(12), 1253-1270.

Belytschko T, Lu YY and Gu L (1994a) Element-free Galerkin methods. Int. J. Numer. Methods Engrg., 37, 229-256.

Belytschko T, Lu YY and Gu L (1994b), Fracture and crack growth by element-free Galerkin methods. Model. Simul. Sci. Compt. Engrg., 2, 519-534.

Belytschko T, Lu YY and Gu L (1995a), Crack propagation by element free Galerkin methods. Engineering Fracture Mechanics, 51, 295-315.

Belytschko T, Lu YY and Gu L (1995b), Element Free Galerkin methods for static and dynamic fracture. Int. J. of Solids and Structures, 32, 2547-2570.

Belytschko T, Lu YY and Gu L (1995c), Element Free Galerkin methods for static and dynamic fracture. Int. J. of Solids and Structures, 32, 2547-2570.

Braza M, Chassaing P and H.Ha Minh (1986), Numerical study and physical Analysis of the pressure and velocity fields in the near wake of a circular cylinder. J. Fluid Mech., 165, 79.

Benz W(1988), Applications of smoothed particle hydrodynamics (SPH) to astrophysical problems. Comput. Phys. Comm., 48, 97-105.

Bernard PS(1995), A deterministic vortex sheet method for boundary layer flow. J. of Computational Physics, 117,132-145.

Biezeno CB, and Koch, JJ (1923), Over een Nieuwe Methode ter Berekening van Vlokke Platen met Toepassing op Enkele voor de Techniek Belangrijke Belastingsgevallen, Ing. Grav. 38, 25-36.

Biezeno CB (1923-1924), Over een Vereenvoudiging en over een Uitbreiding van de Methode van Ritz. Christiaan Huygens 3.69.

Biezeno CB and Grammel R (1955), Engineering Dynamics. Vol. I Theory of Elasticity. Blackie, Glasgow and London.

Biezeno CB and Grammel R (1955), Over een Nieuwe Methode ter Berekening van Vlokke Platen met Toepassing on Enkele voor de Techniek Belangrijke Belastingsgevallen, Ing. Graw. 38, 25036.

Bogomolny A (1985), Fundamental Solution Method for Elliptic Boundary Value Problems. SIAM J. Numer. Anal., 22, 644-669.

Brebbia CA (1978), The Boundary Element Method for Engineers. Pentech Press, London, Halstead Press, New York.

Brebbia CA and Georgiou P(1979), Combination of boundary and finite elements in elastostatics. Appl. Math. Modeling, 3, 212-219.

Brebbia CA, Telles JC and Wrobel LC (1984), Boundary Element Techniques. Springer Verlag, Berlin.

Breitkopf P, Rassineux A and Villon P(2000), A spatial tessellation algorithm for solving of mechanical problems with meshfree methods. Advances in Computational engineering & Sciences(eds Atluri SN and Brust FW, Proceeding of International Conference on Computational Engineering & Science. 1386-1391.Los Angeles 2000.

Chati MK and Mukherjee S (2000), The boundary node method for three-dimensional problems in potential theory. Int. J. Numer. Methods Engrg., 47, 1523-1547.

Chati MK, Mukherjee S and Mukherjee YX (1999), The boundary node method for three-dimensional linear elasticity. Int. J. Numer. Methods Engrg.,46, 1163-1184.

Chati MK, Mukherjee S and Paulino GH(2001), The meshless hypersingular boundary node method for three-dimensional potential theory and linear elasticity problems. Engineering Analysis with Boundary Elements, 25(8), 639-653.

Chen JS, Pan C and Wu CT(1997), Large deformation analysis of rubber based on a reproducing kernel particle method. Comp. Mech., 19, 153-168.

Chen JS, Pan C, Wu C T and Liu WK (1996), Reproducing kernal perticle methods for large deformation analysis of nonlinear structures. Comput. Meth. Appl. Mech. Eng., 139, 195-227.

Chen JS, Roque C, Pan C and Button ST(1998), Analysis of metal forming process based on meshless method. J. of Material Processing Tech., 80-81, 642-646.

Chen JS, Han WM, You Y, et al. (2003), A reproducing kernel method with nodal interpolation property. Int J. Numer. Meth. Eng. 56 (7): 935-960.

Chen WH and Guo XM(2001), Element Free Galerkin Method for three-dimensional structural analysis. Computer Modeling in Engineering & Science, 4(2), 497-508.

Chen XL (2003), Meshfree techniques for plate structures. PhD thesis. National University of Singapore.

Chen XL, Liu GR, and Lim SP(2001), Deflection analyses of laminates using EFG method, International conference on scientific and engineering computing, March 19-23, Beijing, P. R. China, 67.

Chen XL, Liu GR and Lim SP (2003), An element free Galerkin method for the free vibration analysis of composite laminates of complicated shape, Composite Structure, 59 (2): 279-289.

Cheng AHD, Golberg MA, Kansa EJ and Zammito G (2003), Exponential convergence and H-c Multiquadric collocation method for PDE. Num. Methods for PDE, 19(5): 571-594.

Cheng AHD, Lafe O and Grilli S (1994), Dual-reciprocity BEM based on global interpolation functions. Engineering Analysis with Boundary Elements, 13, 303-311.

Cheng M and Liu GR (1999), finite point method for analysis of fluid flow . Proceedings 4th Asia-Pacific Conference on Computational Mechanics (ed. By Wang, Lee and Ang), Dec. 15-17 1999, Singapore, 1015-1020.

Cheng M and Liu GR (2002), A novel finite point method for flow simulation. Int. J Numer. Meth. Fl.. 39 (12): 1161-1178.

Ching HK and Batra RC(2001), Determination of Crack Tip Fields in Linear Elastostatics by the Meshless Local Petrov-Galerkin (MLPG) Method. CMES, 2(2), 273-290.

Christie I, Griffiths DF, Mitchell AR and Zienkiewicz OC (1976), Finite element methods for second order differential equations with significant first derivatives. Int. J. Num. Meth. Eng., 10, 1389-1396.

Cho JY, Kim HG and Atluri SN(2001), Analysis of shear flexible beams, using the meshless local Petrov-Galerkin method based on locking-free formulation. Engineering Computations, 18(1/2), 215-240.

Chorin AJ(1973), Numerical study of slightly viscous flow. J. of Fluid Mechanics, 57,785-796.

Cingoski V, Miyamoto N and Yamashita H(1998), Element-free Galerkin method for electromagnetic field computations. IEEE Transactions on Magnetics, 34(5), Part 1, 3236-3239.

Cleary, PW(1998), Modeling Confined Multi-material Heat and Mass Flows Using SPH. Applied Mathematical Modeling, 22, 981-993.

Cleveland WS (1993), Visualizing Data, AT&T Bell Laboratories, Murray Hill, NJ.

Courant R, Isaacson E and Rees M (1952), On the solution of non-linear hyperbolic differential equations by finite differences. Comm. Pure Appl. MathV, 243-255.

Dai KY, Liu GR, Lim KM and Chen XL(2004), An element-free Galerkin method for static and free vibration analysis of shear-deformable laminated composite plates, Journal of Sound and Vibration, 269 (3-5): 633-652.

Dai KY, Liu GR and Lim KM (2003), A meshfree method for analysis of geometrically nonlinear problems. submitted to Comput. Mech.

Dai KY, Liu GR and Han X(2004), Inelastic analysis for 2D solids based on deformation theory using a meshfree method. submitted.

Davis GV (1983), Natural convection of air in a square cavity: a benchmark numerical solution, Int.J.Numer.Meth.Fluids, 3, 249-264.

De S and Bathe KJ (2000), The method of finite spheres. Computational Mechanics, 25, 329-345.

DeFigueiredo TGB (1991), A New Boundary Element Formulation in Engineering. Springer-Verlag, Berlin.

Dennis SCR and Chang GZ(1980), Numerical solutions for steady flow past a circular cylinder at Reynolds number up to 100, J. Fluid Mech. 42, 471.

Dolbow J and Belytschko T (1998), An introduction to programming the meshless Element Free Galerkin method. ARCH COMPUT METHOD E 5 (3), 207-241.

Dolbow J. and Belytschko T(1999), Numerical Integration of the Galerkin Weak Form in Meshfree Methods. Computational Mechanics, 23, 219-230.

Donea J. et al. (2000), High-order accurate time-stepping schemes for convection-diffusion problems, Comput. Methods Appl. Mech. Engrg. 182, 249-275.

Donea J, Belytschko T and Smolinski P (1985), A generalized Galerkin method for steady state convection-diffusion problems with application ti quadratic shape function. Comp. Mech. Appl. Mech. Eng., 48, 25-43.

Duarte CA and Oden JT(1996), An hp adaptive method using clouds. Comput. Methods Appl. Mech. Engrg., 139, 237-262.

Dumont NA (1988), The hybrid Boundary Element Method. In Brebbia CA (Ed) Boundary Elements IX vol. 1. Springer-Verlag, Berlin, 117-130.]

Fasshauer GE(1997), Solving partial differential equations by collocation with radial abasis functions. In Alain LeMehaute, Christophe RAbut, and LL Schumaker. Eds., Surface fitting and Multiresolution Methods, 131-138. Vanderbilt University Press, Nashville, Tennessee.

Fornberg B(1980), A numerical study of steady viscous flow past a circular cylinder, J. Fluid Mech. , 98,819.

Franke C and Schaback R(1997), Solving Partial Differential Equations by Collocation Using Radial Basis Functions. Applied Mathematics and Computation, 93, 73-82.

Frazer RA, Jones WP and Skan SW (1937), Approximations to functions and to the solutions of Differential Equations. Great Britain Aero Counc. London. Rep. and Memo. No. 1799.

Fujita H(1951), Mem. Coll. Agr. Kyoto Imp. Univ. 59, 31.

Galerkin BG (1915), Rods and plates. Series in Some Problems of Elastic Equilibrium of Rods and plates. Vestn. Inzh. Tch. (USSR) 19, 897-908. Translation 63-18924, Clearinghouse, Fed. Sci. Tech. Info., Springfield., Virginia.

Gingold RA and Moraghan JJ (1977), Smooth particle hydrodynamics: theory and applications to non spherical stars. Man. Not. Roy. Astron. Soc. 181, 375-389.

Girault V (1974), Theory of a GDM on irregular networks. SIAM J. Num. Anal. 11, 260-282.

Golberg and Chen CS(1999), The method of fundamental solutions for potential, Hermholtz and diffusion problems. Chapter 4, Boundary Integral Methods: Numerical and Mathematical Aspects, ed. MA Golberg, WIT Press & Computational Mechanics Publications, Boston, Southampton, 105-176.

Golberg MA, Muleshkov AS, Chen CS and Cheng AHD (2003), Polynomial particular solutions for certain partial differential operators. Num. Methods for PDE (proof).

Gordon WJ and Wixom JA(1978), Shepard's Method of 'Metric Interpolation' to Bivariate and Multivariate data. Math. Comput., 32:253-264.

Gu YT (2003), Meshfree methods and their newest development. Advances in Mechanics(revised).

Gu YT (2002), Development of Meshfree techniques for computational mechanics. PhD thesis, National University of Singapore.

Gu YT and Liu GR (2001a), Using radial function basis in a boundary-type meshless method, boundary point interpolation method (BPIM). Inter. Conf. On Sci. & Engr. Comput., March 19-23,2001, Beijing. 68.

Gu YT and Liu GR(2001b), A coupled element free Galerkin/Boundary element method for stress analysis of two-dimensional solids, Computer Methods in Applied Mechanics and Engineering, 190/34, 4405-4419.

Gu YT and Liu GR(2001c), A meshless local Petrov-Galerkin (MLPG) method for free and forced vibration analyses for solids, Computational Mechanics 27 (3), 188-198.

Gu YT and Liu GR(2001d), A local point interpolation method for static and dynamic analysis of thin beams, Comput. Methods Appl. Mech. Engrg., 190, 5515-5528.

Gu YT and Liu GR(2001e), A boundary point interpolation method (BPIM) using radial function basis, First MIT Conference on Computational Fluid and Solid Mechanics, June 2001, MIT, 1590-1592.

Gu YT and Liu GR(2001f), A meshless Local Petrov-Galerkin (MLPG) formulation for static and free vibration analyses of thin plates, Computer Modeling in Engineering & Sciences, 2(4), 463-476.

Gu YT and Liu GR (2002a), A boundary point interpolation method for stress analysis of solids. Computational Mechanics, 28, 47-54.

Gu YT and Liu GR (2002b), A Hybrid Boundary Point Interpolation Method (HBPIM) and its Coupling with EFG Method. Advances in Meshfree and X-FEM Methods, Proceeding of the 1st Asian Workshop in Meshfree Methods (Ed. Liu GR), Singapore. 167-175.

Gu YT and Liu GR(2003a), Hybrid boundary point interpolation methods and their coupling with the element free Galerkin method. Engineering Analysis with Boundary Elements, 27(9), 905-917.

Gu YT and Liu GR(2003b), A boundary radial point interpolation method (BRPIM) for 2-D structural analyses. Structural Engineering and Mechanics, An International Journal. 15(5), 535-550.

Gu YT and Liu GR (2005), A Meshfree Weak-Strong (MWS) form method for time dependent problems. Computational Mechanics, 35, 134-145.

Guymon GL, Scott VH and Herrmann LR (1970), A general numerical solution of the two dimensional diffusion-convection equation by the FEM. Water Resources Res., 6, 1611-1617.

Hall T (1970), CarL Friedrich Gauss. MIT Press, Cambridge, Massachusetts.

He XY and Doolen GD(1997), Lattice Boltzmann method on a curvilinear coordinate system: Flow around a Circular cylinder, Journal Compu.Physics, 134, 306-315.

Hegen D (1996), Element-free Galerkin methods in combination with finite element approaches. Comput. Methods Appl. Mech. Engrg., 135,143-166.

Hueck U, Wriggers P(1995), A formulation for the 4-node quadrilateral element. Int. J. Numer. Methods Engrg., 38, 3007-3037.

Hughes WF and Brighton JA (1991), Schaum's outline of Theory and Problems of Fluid Dynamics(second edition), McGRAW-Hill, Inc., New York.

Hughes TJR and Brooks AN (1982), A theoretical framework for Petrov-Galerkin methods with discontinuous weighting function. Finite Elements in Fluids (eds Gallagher RH et al.), Vol. 4, 47-65. Wiley, Chichester.

Hughes TJR and Brooks A (1979), A multi-dimensional upwind scheme with no cross wind diffusion, in Finite elements for convection Dominated Flows (ed. Hughes TJR), AMD 34, ASME.

Hughes TJR, Franca LP, Hulbert GM, Johan Z and Sakhib F (1988), The Galerkin least square method for advective diffusion equations, in Recent Developments in Comp. Fl. Mech. (eds Tezduyar and Hughes), AMD 95, ASME.

Jordan SK.and Fromm (1972), Oscillatory drag, lift, and torque on a circular cylinder in a uniform flow. Physics of Fluids, 15,371-376.

Jun, SA(1996), Meshless method for nonlinear solid mechanics. RIKEN Review, 14, 33-34.

Kansa EJ(1990), Multiquadrics-A Scattered Data Approximation Scheme with Applications to Computational Fluid dynamics. Computers Math. Applic., 19(8/9),127-145.

Kelly DW, Nakazawa S and Zienkiewicz OC (1980), A note on anisotropic balancing dispation in the finite element method approximation to convective diffusion problems Int. J. Num. Meth. Eng., 15, 1705-1711.

Kim HG and Atluri SN (2000), Arbitrary placement of secondary nodes, and error control, in the meshless local Petrov-Galerkin (MLPG) method. Computer Modeling in Engineering & Sciences, 1(3), 11-32.

Kleiber M. (Ed.) (1998), Handbook of Computational Solid Mechanics. Springer, Berlin.

Kothnur VS, Mukherjee S and Mukherjee YX(1999), Two-dimensional linear elasticity by the boundary node method. Int. J. of Solids and Structures, 36,1129-1147.

Krok J and Orkisz J (1989), A unified approach to the FE generalized variational FD method for nonlinear mechanics. Concept and numerical approach. 353-362. Springer-Verlag.

Krongauz Y and Belytschko T (1996), Enforcement of essential boundary conditions in meshless approximations using finite element. Comput. Methods in Appl. Mech. and Engrg., 131, 133-145.

Krysl P and Belytschko T (1995), Analysis of thin plates by the element-free Galerkin method. Computational Mechanics, 17(1-2), 26-35.

Krysl P and Belytschko T(1996), Analysis of thin shells by the element-free Galerkin method. International Journal of Solids and Structures, 33(20-22), 3057-3080.

Krysl P and Belytschko T(1999), Element free Galerkin method for dynamic propagation of arbitrary 3-D cracks. International Journal for Numerical Methods in Engineering, 44(6), 767-800.

Lam KY, Liu GR, Liu MB and Zong Z (2000), Numerical simulation of underwater shock using SPH methodology. Computational Mechanics and Simulation of Underwater Explosion Effects- US/Singapore workshop, pp. 1-6, November 2000, Washington D.C.

Lanczos C (1938), Trigonometric Interpolation of Empirical and Anlytical Functions, J. Math. Phys. 17, 123-199.

Lancaster P and Salkauskas K (1981), Surfaces generated by moving least squares methods. Math. Comput., 37, 141-158.

Lancaster P and Salkauskas K (1986), Curve and surface fitting, an introduction. Academic Press.

Li SF and Liu WK (2002), Meshfree and particle methods and their applications. Applied Mechanics Review, 55, 1-34.

Li Hua, Wang QX and Lam KY (2004), A variation of local point interpolation method (vLPIM) for analysis of microelectromechanical systems (MEMS) device. Engineering Analysis with Boundary Elements. 28 (10): 1261-1270.

Libersky LD and Petscheck AG(1991), Smoothed Particle Hydrodynamics with Strength of Materials. in H. Trease, J. Fritts and W. Crowley eds., Proceedings of The Next Free Lagrange Conference, Springer-Verlag, NY, 395,248-257.

Libersky LD, Randles PW and Carney TC(1995), SPH calculations of fragmentation. Proceedings of 3rd US congress on computational Mechanics, Dallas, TX.

Lin H and Atluri SN (2001), Analysis of incompressible Navier-Stokes flows by the meshless MLPG method. Computer Modeling in Engineering & Sciences, 2(2), 117-142.

Lin H. and Atluri SN (2000), Meshless Local Petrov-Galerkin (MLPG) method for conection-difusion problems. Computer Modeling in Engineering & Sciences, 1(2), 45-60.

Liszka T and Orkisz J (1977), Finite difference methods of arbitrary irregular meshes in non-linear problems of applied mechanics. In Proc. 4th Int. Conf. on Structural Mech. In Reactor Tech, San Francisco, USA.

Liszka T and Orkisz J (1980), The finite difference methods at arbitrary irregular grids and its applications in applied mechanics. Comp. Struct., 11, 83-95.

Liszka TJ, Duarte CA and Tworzydlo WW(1996), Hp-Meshless cloud method. Comput. Methods Appl. Mech. Engrg., 139, 263-288.

Liu GR(1999), A Point Assembly Method for Stress Analysis for Solid, in Impact Response of Materials & Structures, V. P. W. Shim et al. eds, Oxford, 475-480.

Liu GR (2002), Mesh Free Methods: Moving Beyond the Finite Element Method. CRC press, Boca Raton, USA.

Liu GR and Chen XL(2000), Static buckling of composite laminates using EFG method, Proc. of the 1st Int. Conf. On Structural Stability and Dynamics, December 7-9, Taipei, Taiwan. 321-326.

Liu GR and Chen XL(2001), A mesh-free method for static and free vibration analyses of thin plates of complicated shape, Journal of Sound and Vibration. 241 (5), 839-855.

Liu GR, Dai KY, Lim KM and Gu YT(2002), A point interpolation mesh free method for static and frequency analysis of two-dimensional piezoelectric structures. Computational Mechanics, 29 (6), 510-519.

Liu GR, Dai KY, Lim KM and Gu YT (2003), A radial point interpolation method for simulation of two-dimensional piezoelectric structures, Smart Materials and Structures. 12: 171-180.

Liu GR, Dai KY and Lim KM (2003), A meshfree method for analysis of geometrically nonlinear problems (submitted).

Liu GR and Gu YT(1999), A point interpolation method. Proceedings of 4th Asia-Pacific Conference on Computational Mechanics, Dec. 1999, Singapore, 1009-1014.

Liu GR and Gu YT(2000a), Meshless local Petrov-Galerkin (MLPG) method in combination with finite element and boundary element approaches, Comput. Mech., 26, 536-546.

Liu GR and Gu YT(2000b), Vibration analyses of 2-D solids by the local point interpolation method (LPIM), Proceedings of 1st international conference on structural stability and dynamics, Taiwan, Dec. 7-9, 411-416.

Liu GR and Gu YT(2000c), Coupling of element free Galerkin and hybrid boundary element methods using modified variational formulation, Computational Mechanics, 26, 2, 166-173.

Liu GR and Gu YT(2001a), A point interpolation method for two-dimensional solids. Int. J. Muner. Meth. Engng, 50, 937-951.

Liu GR and Gu YT(2001b), A local point interpolation method for stress analysis of two-dimensional solids, Structural Engineering and Mechanics, 11(2), 221-236.

Liu GR and Gu YT(2001c), A local radial point interpolation method (LR-PIM) for free vibration analyses of 2-D solids. J. of Sound and Vibration, 246(1), 29-46.

Liu GR and Gu YT(2001d), A matrix triangularization algorithm for point interpolation method, Proceedings of the Asia-Pacific Vibration Conference, Nov. 2001, ed. Bangchun Wen, Hangzhou, China, 1151-1154.

Liu GR and Gu YT(2001e), Truly meshless methods based on the local concept and its applications, International Congress on Computational Engineering Science (ICES'01), 19-25, August 2001, Puerto Vallarta Mexico (Keynote Lecture).

Liu GR and Gu YT(2002a), Comparisons of two meshfree local point interpolation methods for structural analyses. Computational Mechanics 2002, 29, 107-121.

Liu GR and Gu YT(2002b), Boundary meshfree methods based on the boundary point interpolation methods, Boundary Elements XXIV (eds. CA Brebbia et al.), 57-66.

Liu GR and Gu YT(2002d), A truly meshless method based on the strong-weak form. Advances in Meshfree and X-FEM Methods, Proceeding of the 1st Asian Workshop in Meshfree Methods (Ed. Liu GR), Singapore, 259-261.

Liu GR and Gu YT(2003a), A matrix triangularization algorithm for point interpolation method. Computer Methods in Applied Mechanics and Engineering, 192/19 pp. 2269-2295.

Liu GR and Gu YT(2003b), A meshfree method: Meshfree Weak-Strong (MWS) form method, for 2-D solids. Computational Mechanics 33(1), 2-14.

Liu GR, Gu YT and Wu YL (2003), A Meshfree Weak-Strong form (MWS) method for fluid mechanics. Proceeding of International Workshop on MeshFree Methods, July 21-23, 2003, Lisbon, Portugl, pp. 73-78.

Liu GR and Gu YT(2004a), Boundary meshfree methods based on the boundary point interpolation methods, Engineering Analysis with Boundary Elements, 28/5, 475-487.

Liu GR, Gu YT and Hoo YK(2004b), Parameter studies for radial basis functions. ICCM 2004,15-17, Dec.2004, Singapore.

Liu GR and Gu YT(2004c), Assessment and applications of point interpolation methods for computational mechanics, International Journal for Numerical Methods in Engineering, 59:1373–1397.

Liu GR and Han X(2003), Computational Inverse Techniques in Nodestructive Evaluation. CRC press, Boca Raton, USA.

Liu GR, Liu MB, and Lam KY(2002), A General Approach for Constructing Smoothing Functions for Meshfree Methods, The Ninth International Conference on Computing in Civil and Building Engineering, April 3-5, 2002, Taipei, Taiwan.

Liu GR, Liu MB and Lam KY(2001a)., Smoothed particle hydrodynamics for numerical simulation of high explosive explosions, The first international symposium on advanced fluid information (AFI-2001), Institute of fluid Science, Tohoku University, Sendai, Japan.

Liu GR, Liu MB, Zong, Z., and Lam KY(2001b), Simulation of the high explosive detonation process using SPH methodology, First MIT conference on Computational Solid and Fluid Mechanics, June 2001, MIT, USA.

Liu GR, Lam KY and Lu C (1998), Computational simulation of sympathetic explosion in a high performance magazine. Proc. of 3rd Weapon Effects Seminar, Singapore.

Liu GR and Liu MB (2003), Smoothed particle hydrodynamics – a meshfree particle method. World Scientific, Singapore.

Liu GR and Quek SS (2002), Finite Element Method :A Practical Course. World Seientific, Singapore.

Liu GR and Yan L(1999), A study on numerical integrations in element free methods. Proceeding of APCOM '99, Singapore, 979-984, 1999.

Liu GR and Yan L (2000), A Modified Meshless Local Petrov-Galerkin Method for Solid Mechanics, Advances in Computational Engineering and Sciences, eds: Atluri and Brust, 2000, Tech Science Press, Palmdale, 1374-1379,2000.

Liu GR, Yan L, Wang JG and Gu YT (2002), Point interpolation method based on local residual formulation using radial basis functions. Structural Engineering and Mechanics, Vol. 14, No. 6, 713-732.

Liu GR and Yang KY(1999), A New Meshless Method for Stress Analysis for Solids and Structure, Proceedings of Fourth Asia-Pacific Conference on Computational Mechanics, 15-17 December, Singapore.

Liu GR and Yang KY(1998), A Penalty Method for Enforce Essential Boundary Conditions in Element Free Galerkin Method, In Proc. of the 3rd HPC Asia'98, 1998, Singapore: 715-721.

Liu GR, Yang KY and Cheng M (1999), Some Recent Development in Element Free Methods in Computational Mechanics, Proceedings of Fourth Asia-Pacific Conference on Computational Mechanics, 15-17 December, Singapore.

Liu GR and Xi ZC(2001), Elastic Waves in Anisotropic Laminates, CRC Press, USA.

Liu GR and Tu, ZH(2002), An Adaptive Procedure Based on Background Cells for Meshless Methods, Comput. Methods in Appl. Mech. and Engrg., 191, 1923-1943.

Liu GR, Tu ZH and Wu YG (2000), MFree2D?: an adaptive stress analysis package based on mesh free technology, The 3rd South Africa Conference on Applied Mechanics. 11-13 Jan, 2000,Durban, South Africa.

Liu GR and Tu, ZH (2001), MFree2D: an adaptive stress analysis package based on mesh-free technology, First MIT Conference on Computational Fluid and Solid Mechanics, June 2001, MIT, 327-329, 2001.

Liu GR, Gu YT, Tu ZH, Huang XM, Wang JG and Wu YG (2002). MFree2D. http://www.nus.edu.sg/ACES.

Liu GR and Wu YL(2002), Application of Meshless Point Interpolation Method with Matrix Triangularization Algorithm to Natural Convection in Advances in Meshless and X-FEM Methods, Proceedings of the 1st Asian Workshop on Meshfree Methods (Edited by Liu GR), World Scientific, Singapore, 135-139.

Liu GR, Wu YL, and Gu YT(2001), Application of Meshless local Petrov-Galerkin (MLPG) approach to Fluid Flow Problem, Proceedings of the First Asian-Pacific Congress on Computational Mechanics, 20-23 November,2001, Sydney, Australia.

Liu GR, Wu YL and Gu YT (2002), A Meshless Petrov-Galerkin Method for fluid mechanics. (submitted).

Liu GR, Wu YL, Ding H (2004), Meshfree weak-strong (MWS) form method and its application to incompressible flow problems. International Journal for Numerical methods fluids, 46 (10): 1025-1047.

Liu GR, Zhang GY and Gu YT (2003), The radial interpolation method for 3D solid problems (submitted).

Liu L, Liu GR and VBC Tan (2001), Element free analyses for static and free vibration of thin shells, Proceedings of the Asia-Pacific Vibration Conference, Nov. 2001, ed. Bangchun Wen, Hangzhou, China.

Liu L and Tan VBC(2002a), A meshfree method for dynamics analysis of thin shells, In Liu G. R. (Ed.): Advances in Meshfree and X-FEM methods, pp. 90-95.

Liu L, Liu GR, Tan VBC (2002b),Element free method for static and free vibration analysis of spatial thin shell structures. Comput. Method Appl. M. 191 (51-52): 5923-5942.

Liu MB, Liu GR, Zong Z and Lam KY(2001), Numerical simulation of incompressible flows by SPH, International Conference on Scientific & Engineering Computing, 72, March 2001, Beijing, China.

Liu MB, Liu GR and Lam KY(2002a), Investigations into water mitigations using a meshless particle method, Shock Waves 12(3):181-195.

Liu MB, Liu GR and Lam KY(2003a), Comparative study of the real and artificial detonation models in underwater explosions, Engineering Simulation, 25(1).

Liu MB, Liu GR and Lam KY(2003b), Constructing smoothing functions in smoothed particle hydrodynamics with applications, Journal of Computational and Applied Mathematics. 155 (2): 263-284.

Liu MB, Liu GR and Lam KY(2003c), Meshfree particle simulation of the explosion process for high explosive in shaped charge, Shock Waves. 12 (6): 509-520.

Liu MB, Liu GR, Zong Z and Lam KY (2003d), Computer simulation of the high explosive explosion using smoothed particle hydrodynamics methodology, Computers & Fluids, 32(3): 305-322.

Liu MB, Liu GR and Lam KY(2003e), A one dimensional meshfree particle formulation for simulating shock waves, Shock Waves. 13 (3): 201-211.

Liu MB, Liu GR, Zong Z and Lam KY(2003f), Smoothed particle hydrodynamics for numerical simulation of underwater explosions, Computational Mechanics, 30(2):106-118.

Liu WK and Jun S (1998), Multiple scale reproducing kernel particle method for large deformation problems. Int. J. for Num. Methods in Engre, 141, 1339-1679.

Liu WK, Jun S and Zhang YF (1995), Reproducing kernel particle methods. Int. J. Numer. Methods Engrg., 20, 1081-1106.

Liu WK, Jun S, Sihling DT, Chen Y and Hao W(1997b), Multiresolution reproducing kernel particle method for computational fluid dynamics. International Journal for Numerical Methods in Fluids, 24(12), 1391-1415.

Liu Xin, Liu GR, Tai Kang and Lam KY(2002), Radial Basis Point Interpolation Collocation Method For 2-d Solid Problem, Advances in Meshfree and X-FEM Methods, proceedings of the 1st Asian Workshop on Meshfree methods (Eds. G. R. Liu), World Scientific, 35-40.

Liu Xin, Liu GR, Tai Kang and Lam KY(2003a), Collocation-based Meshless Method for the Solution of Transient Convection-Diffusion Equation. Presented in "First EMS -SMAI -SMF Joint Conference Applied Mathematics and Applications of Mathematics (AMAM 2003), Nice, France, 10-13 February 2003, List of Posters, Posters Session 2 (48).

Liu Xin, Liu GR, Tai Kang, Gu YT and Lam KY(2003b), Polynomial Point Interpolation Collocation Method for the Solution of Partial Differential Equations. Advances in Computational Mathematics (AiCM)(Accepted) .

Liu Xin, Liu GR, Tai Kang, Gu YT and Lam KY(2003b), Polynomial Point Interpolation Collocation Method for the Solution of Partial Differential Equations. Advances in Computational Mathematics (AiCM)(Accepted) .

Liu Xin, Liu GR, Tai Kang and Lam KY(2003d), Radial Point Interpolation Collocation Method for the Solution of Transient Convection-Diffusion Problems. (submitted).

Long SY and Atluri SN (2002), A meshless local Petrov-Galerkin (MLPG) method for solving the bending problems of a thin plate. CMES. Vol. 3, No. 1,53-64.

Lu YY, Belytschko T and Gu L(1994), A New Implementation of the Element Free Galerkin Method. Comput. Meth. Appl. Mech. Eng., 113,397-414.

Lu YY, Belytschko T and Tabbara M(1995), Element-free Galerkin method for wave propagation and dynamic fracture, Computer Methods in Applied Mechanics and Engineering, 126, 131-153.

Lucy L(1977), A Numerical Approach to Testing the Fission Hypothesis, Astron. J., 82, 1013-1024.

McLain DH(1974), Drawing Contours from Arbitrary Data Points. Comput. J., 17:318-324.

Meirovitch L(1980), Computational Methods in Structural Dynamics. Sijthoff & Noordhoff: Grningen.

Melenk, JM and Babuska I(1996), The Partition of Unity Finite Element Method: Basic Theory and Applications. Computer Methods in Applied Mechanics and Engineering, 139, 289-314.

Monaghan JJ and Lattanzio JC (1985), A refined particle method for astrophysical problems. Astronomy and Srtophsics, 149, 135-143.

Monaghan JJ(1982), Why Particle Methods Work. SIAM J. SCI. SAT. COMPUT., 3(4), 423-433.

Monaghan JJ(1992), Smoothed particle hydrodynamics, Annu. Rev. Astron. Astrophys, 30, 543-574.

Moraghan JJ(1995), Simulating gravity currents with SPH lock gates, Applied Mathematics Reports and Preprints, Monash University.

Morton KW (1985), Generlaed Galerkin methods for hyperbolic problems. Comp. Meth. Appl. Mech. Eng., 52, 847-871.

Morris JP(1996), Stablity properties of SPH. Publ. Astron. Soc. Aust., 13, 97.

Mukherjee YX and Mukherjee S (1997), Boundary node method for potential problems. Int. J. Num. Methods in Engrg. 40, 797-815.

Nagashima T (1999), Node-by node meshless approach and its application to structural analyses. Int. J. Numer. Methods Engng.,46,341-385.

Nay RA and Utku S (1972). An alternative for the finite elemet method. Variat. Mech.

Nayroles B, Touzot G and Villon P (1992), Generalizing the finite element method: diffuse approximation and diffuse elements. Computational Mechanics, 10, 307-318.

Nieuwstadt F and Keller HB(1973), Viscous flow past circular cylinders, Comput.& Fluids,1,59.

Oñate E (1998), Derivation of stabilized equations for numerical solution of sdvictive-diffusive transport and fluid flow problems. Comp. Meth. Appl. Mech. Eng., 151, 233-265.

Oñate E Perazzo F, and Miquel J, (2001), A finite point method for elasticity problems, Computers and Structures, 79, 2151-2163.

Oñate E, Idelsohn S, Zienkiewicz OZ and Taylor RL(1996), A finite point method in computational mechanics. Applications to convective transport and fluid flow. Int. J. Numer. Methods Engrg., 39, 3839-3867.

Ouatouati AE and Johnson DA(1999), A New Approach for Numerical Modal Analysis Using the Element Free Method. Int. J. Num. Meth. Eng., 46,1-27.

Partridge PW (1004), Dual reciprocity BEM-local versus global approximation functions for diffusion, convection and other problems. Engineering Analysis with Boundary Elements, 14, 349-356.

Pavlin V and Perrone N (1975), Finite difference energy rechniques for arbitrary meshes. Comp. Struct. 5, 45-58.

Petyt M (1990), Introduction to finite element vibration analysis. Cambridge University Press, Cambridge.

Picone M (1928), Sul Metodo delle Minme Potenze Ponderate e sul Metodo di Ritz per il Calcolo Approssimato nei Problemi della Fisca-Matematica, Rend. Circ. Mat. Palermo 52, 225-253.

Poulos HG, Davis EH (1974), Elastic solution for soil and rock mechanics. Wiley, London.

Powell MJD (1992), The Theory of Radial Basis Function Approximation in 1990. Advanced in Numerical Analysis, Eds. FW. Light: 303-322.

Randles PW and Libersky LD(1996), Smoothed Particle Hydrodynamics Some Recent Improvements and Applications, Comput. Methods Appl. Mech. Engrg., 138, 375-408.

Reddy JN (1993), An introduction to the Finite Element Method. New York: McGraw Hill, 2nd edition.

Roark RJ andYoung WC (1975), Formulas for stress and strain. McGrawhill.

Robert DB (1979), Formulas for natural frequency and mode shape. New York: Van Nostrand Reinhold Company.

Rossow MP, and Katz IN (1978), Hierarchal Finite Elements and Precomputed Arrays, Int. J. Numer. Methods Engrg., 10, 976-999.

Schaback R (1994), Approximation of Polynomials by Radial basis Functions. Wavelets, Images and Surface Fitting, (Eds. Laurent P.J., Mehaute Le and Schumaker L.L, Wellesley Mass.), 459-466.

Runchall AK and Wolfstein M (1969), Numerical integration procedure for the steady state N-S equations. J Mech. Eng. Sci., 11, 445-453.

Schaback R and Wendland H (2000), Characterzation and construction of radial basis functions. In: Multivariate Approximation and Applications, (eds.) N. Dyn, D. Leviatan, D. Levin & A. Pinkus, Cambridge University Press.

Sharan M, Kansa EJ and Gupta S(1997), Application of the Multiquadric Method for Numerical Solution of Elliptic Partial Differential Equations. Applied Mathematics and Computation, 84,275-302.

Shepard D (1968), A two-dimensional function for irregularly spaced points. Proc. Of ACM National Conference, 517-524.

Sladek J, Sladek V and Atluri SN (2002),Application of the local boundary integral equation method to boundary-value problems. Int. Appl. Mech., 38 (9): 1025-1047.

Slater JC(1934), Electronic Energy Bands in Metals. Phys. Rev. 45, 794-801.

Snell C, Vesey DG and Mullord P (1981), The application of a general fFDM to some boundary value problems. Comp. Struct., 13, 547-552.

Spalding DB (1972), A novel finite difference formulation for differential equations involving both first and second derivatives. Int. J. Num. Meth. Eng., 4, 551-559.

Strang G (1976), Linear Algebra and its application. Academic Press: New York.

Stroud AH and Secrest D (1966), Gaussian Quadrature Formulas. Prentice-Hall, New York.

Sun X (1994), Scattered Hermite interpolation using radial basis function. Linear Algebra Appl. 207, 135-146.

Swegle J. W. at al. (1992), Report at Sandia National laboratories, August, 1992.

Swegle JW, Hicks DL and Attaway SW(1995), Smoothed particle hydrodynamics stability analysis. Journal of Computational Physics, 116, 123-134.

Thomson WT(1993), Theory of vibration with applications (4th ed). Englewood Cliffs, N.J. Prentice Hall.

Timoshenko S, Woinowsky-krieger S (1995), Theory of plates and shells. New York: McGraw Hill, 2nd edition.

Timoshenko SP and Goodier JN (1970), Theory of Elasticity, 3rd Edition. McGraw-hill, New York.

Tritton DJ (1959), Experiments on the flow past a circular cylinder at low Reynolds numbers. Journal of Fluid Mechanics, 6, 547-567.

Tu, Z.H. and Liu GR, An error estimate based on background cells for meshless methods. Advances in Computational Engineering and Sciences, eds: Atluri and Brust, Tech Science Press, Palmdale, 1487-1492, 2000.

Uras RA, Chang CT, Chen Y and Liu WK(1997), Multiresolution reproducing kernel particle method in acoustics. J. of Computational Acoustics, 5, 71-94.

Wang JG and Liu GR(2000), Radial point interpolation method for elastoplastic problems. Proc. of the 1st Int. Conf. On Structural Stability and Dynamics, Dec. 7-9, 2000, Taipei, Taiwan, 703-708,2000.

Wang JG and Liu GR(2001a), Radial point interpolation method for no-yielding surface models. Proc. Of the first M.I.T. Conference on Computational Fluid and Solid Mechanics, 12-14 June 2001, 538-540.

Wang JG, Liu GR and Wu YG(2001b), A point interpolation method for simulating dissipation process of consolidation. Computer Method Appl. M. 190 (45): 5907-5922.

Wang JG and Liu GR (2002a) A point interpolation meshless method based on radial basis functions. Int. J. Numer. Meth. Eng. 54 (11): 1623-1648.

Wang JG, Liu GR and Lin P (2002b), Numerical analysis of Biot's consolidation process by radial point interpolation method. Int. J. Solids Struct. 39 (6): 1557-1573.

Wang JG and Liu GR(2002c), On the optimal shape parameters of radial basis functions used for 2-D meshless methods . Computer Method Appl. M. 191 (23-24): 2611-2630.

Wang MC and Shao M(1996), FEM fundamentals and numerical methods (in Chinese). Tsing Hua University Publisher, Beijing, China.

Wendland H(1995), Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree. Adv. Comput. Math. 4, 389-396.

Wendland H(1998), Error estimates for interpolation by compactly supported radial basis functions of minimal degree. J. Approximation Theory, 93, 258-396.

Wu Z (1992), Hermite-Birkhoff interpolation of scattered data by radial basis functions. Approx. Theory Appl. 8, 1-10.

Wu Z (1995), Compactly supported positive definite radial functions. Adv. Comput. Math. 4, 283-292.

Wu Z(1998), Solving PDE with radial basis functions and the error estimation. Adv. in Comput. Math., Lectures in Pure and Applied Mathematics, 202, eds. Z. Chen, Y.Li,. CA Micchelli, Y Xu, and M Dekkon.

Wu YL and Liu GR (2003), A meshfree formulation of local radial point interpolation method (LRPIM) for incompressible flow simulation. Computational Mechanics,30 (5-6) 355-365

Xiao JR and McCarthy MA(2003a), A local Heaviside weighted meshless method for two-dimensional solids using radial basis functions. COMPUT MECH 31 (3-4): 301-315.

Xiao JR and McCarthy MA (2003b), Meshless analysis of the obstacle problem for beams by the MLPG method and subdomain variational formulations. EUR J MECH A-SOLID. 22 (3), 385-399.

Xiao JR and McCarthy MA (2003c), On the use of radial basis functions in a local weighted meshless method. Second M.I.T. Conference on Computational Fluid and Solid Mechanics. June 17-20.

Xu SL (1995), FORTRAN algorithms and programs (2nd edition) (in Chinese). Tsinghua University Publisher, Beijing China.

Xu XG and Liu GR(1999), A local-function approximation method for simulating two-dimensional incompressible flow. Proceedings 4th Asia-Pacific Conference on Computational Mechanics(ed. By Wang CM, Lee KH and Ang KK), Dec. 15-17 1999, Singapore, 1021-1026.

Yagawa G and Yamada T(1996), Free mesh method, a kind of meshless finite element method. Comput. Mech., 18, 383-386.

Yamada H(1947), Rep. Res. Inst. Fluid Eng. Kyushu Univ., 3, 29.

Yan L (2000), Development of meshless method in computational mechanics. National University of Singapore, thesis of M. Eng.

Yang KY(1999), Development of meshfree techniques for stress analysis. National University of Singapore, thesis of M. Eng.

Zhang X, Song KZ, Lu MW and Liu X (2000), Meshless methods based on collocation with radial basis functions. Comput. Mechanics, 26(4), 333-343.

Zhu T and Atluri SN(1998), A modified collocation & a penalty formulation for enforcing the essential boundary conditions in the element free Galerkin method. Comput. Mech., 21, 211-222.

Zhu T, Zhang JD and Atluri SN (1998), Meshless local boundary integral equation (LBIE) method for solving nonlinear problems. Computational Mechanics, 22(2), 174-186.

Zhu T, Zhang JD and Atluri SN (1999), A meshless numerical method based on the local boundary integral equation (LBIE) to solve linear and non-linear boundary value problems. Engineering Analysis with Boundary Elements, 23, (5-6), 375-389.

Zhu T, Zhang JD and Atluri SN(1998), Local boundary integral equation (LBIE) method in computational mechanics, and a meshless discretization approach. Computational Mechanics, 21(3), 223-235.

Zienkiewicz OC and Taylor RL (2000), The Finite Element Method. 5th edition, Butterworth Heinemann, Oxford, UK.

Zienkiewicz OC, et al.(1975), Newtonian and non-Newtonian viscous incompressible flow. Temperature induced flows and finite element solutions, in math. Of FE and appl. (ed. Whiteman), Vol. II, Academic Press, London.

Index

A

Adaptive analysis, 38,41,444

Amplitude, 414, 420

Analytical solution, 26, 181, 186,265,342,346,349,375,410, 420

Approximation function, 45

Approximate solution, 1,2 314-19, 20-27, 30, 33, 34, 36, 54

Admissible, 27, 34

B

Background mesh, 163, 171, 238, 261, 279, 336, 381, 383

Background cell, 45, 47, 146, 147, 149, 155, 169, 170, 171, 174, 179, 186, 197, 199

Bandwidth, 147, 194

Basis function, 15, 17, 22, 36, 57, 61, 62, 65, 66, 67, 80, 86, 98, 101, 107, 111, 163, 171, 112, 114

BIE18, 50

BNM, 45,52,253

Body force, 7,12,33,148,153,154,156,244, 250, 328

Boundary conditions, 3,5,8,9,12,15,21,22,23,34,35,39,4 9,50,54,56,69,74, 148, 158, 171,179,255,311,312,313,354, 371,372,378,381,410,435

Boundary integral equation, 50,51,383

Boundary element method, 3,49

Boundary type method, 50,52

Boundary point interpolation method, 45,50,51

Boundary node method, 45,253

BPIM,45,46

Boundary radial point interpolation method, 50,51

BRPIM,50,51,52

C

Cartesian coordinates, 61

CFD, 335,371

Collocation method, 14,18,23, 44,46,47,52,56,69,79,85,270,311-313,319,323-329, 331,333,343, 346,351,352, 364,368,369,378, 381,388, 405,406,407,412,443

Compatibility, 55,56,66,81,102, 103,123,147,160,161,204,238,

239,253,279,401

Compact support, 56,66,79,80, 105, 253

Computational fluid dynamics, *see* CFD

Condition number, 118,187

Conforming, 160

Conforming radial point interpolation method, *see* CRPIM

Consistency, 13,14,46,49,55,57, 64, 65,76,106,107,240,397

Constitutive equation, 6,9,11,13, 32,151

Continuity, 14,15,17,22,35,49,81, 100, 102,103, 106, 114,248,382

Convergence, 26,27,31,36,38,56, 81,120,123,145,157,177,190,191, 192,193,204,262, 272, 276, 278, 279,280

Convergence rate, 190,192,193, 272, 276,324,407, 429

CRPIM, 160,161

Crank-Nicholson method, 360

Cubic spline function, 102, 108, 111,248, 264

Curve integration, 156,162,186, 259

D

Damping, 7,12,413,414,418,419, 420

DBC, 311,313,317,323,331,334, 344,345,349,352,371,378,352,371, 378,381,384,405

DBR,385,386,388,391,393,406, 412,425,436,443

Derivative boundary condition, *see* DBC

Derivative boundary related, *see* DBR

Deflection, 180,181,182,183, 200, 201,264, 395, 396

Defuse element method, see DEM

Defuse element method (DEM), 45, 48, 145

Degree of freedom (DOF), 319, 378, 418

Delta function property, 49, 50, 55, 56, 65,69, 74, 79, 90, 91,107, 146,147, 158, 161, 162, 169, 176, 238, 246, 248, 250, 390, 403

Deformation, 6,38,46, 199, 201, 418, 419

Differential function representation, 57

Dirichlet boundary condition, 314,317,324,328,329,336,344,345 ,349,352,354,355,357,359,361,37 8,381,426

Dirac delta function, 17

Discrete equation, 35, 44, 220

Discretized equation, 167, 310, 345, 371, 373, 374, 383, 424, 436

Displacement boundary condition, 8,12,21,34,155,158, 366,375

Displacement vector, 6,8,10, 33, 148, 152, 176, 207

Domain of influence, *See* influence domain,

Domain of integration, *See* quadrature domain

Domain of weight function, *See* weight function domain

Domain of support, *See* support domain

Domain type method, 50, 52

Dynamics, 7,8, 79, 238, 423

Dynamic equilibrium equations, 8

E

Efficiency, 47, 55, 123, 167, 177, 194,195,204, 253, 278, 279, 280, 357,378, 391, 408, 410, 416, 427

EFG, 45,46,48,145,146,148, 161,164,167,171,186,191,194,201, 204,237,254,276,279,319,376,380, 382,392

Eigenvalues, 414,415

Eigenvectors,415

Element free Galerkin method, *See* EFG
Elasticity, 8,148,204

Elastic constants, 148,243,387

Energy norm, 173,176,186, 187,191,195,255,261,270,392,395, 428

Error in the energy norm, 173, 183, 195,392,393,395

Equilibrium equation, 7,8,12, 21,22,33,35,148,242

Error indicator, 27,117,176, 186, 255, 32, 331,346, 392, 428

Essential boundary condition, 9,15,34,41,56,69,74,146,147,148, 158,163,169,178,195,204,238,250, 252,255,259,366,392,403,426

Exact solution, 1,13,15,17,22, 26,27,36,177,190,262,324,329,331, 349,355,393,404,428

EXP, 74,75,86,87,90,169,353

Exponential function, 102,104

F

FDM, 3,44,56,246,312, 337, 371,390,429

FE, FEM, 3,16,33,37,41,55,61,155,158,181, 196,254,327,417

Field variable, 2,4,8,39,43,97, 49, 274,336,360,391

Field approximation, 239

Field variable approximation, 49,163, 310,391

Finite element method, *see* FEM

Finite difference method, *see* FDM

Finite differential representation,13

Finite point method (FPM), 47, 310
Finite series representation, 13

Fluid dynamics, 46,48,79,423

Forced vibration analysis, 415, 417

Free vibration, 414,417

Frequency, 414

Functional, 19,98,164

Function approximation,

45,48,51,60,69,191,238,314

G

Galerkin method, 20,25

Gauss point, 155,179,245, 264,387

Gauss quadrature, 155,166, 274, 391

Gaussian RBF, *see* EXP

Generalized Hooke's law, 6

Governing equations, 8,34,46, 29, 315,328, 336,368,423,434

Green's function, 50

Global stiffness matrix, 152, 162, 166, 252, 319, 390
Global weak-form, 27, 46, 145, 254,383

H

Hamilton's principle, 14

High-order patch test, 403

Hooke's law, 6,11

hp-clouds method, 58

I

Incompatibility, 160

Incompressible flow, 48,423, 443

Influence domain, 54,167, 170, 174, 255,392

Initial conditions, 8,12,13, 359, 361,410

Initial value problem, 9,13

Integral function representation, 57

Integration by parts, 21,33, 239,382

Instability, 49,55,56,335,364,384

Iteration, 198,199,345,427

Interpolation domain, see *quadrature domain*

J

Jacobian matrix, 155,156, 245, 258

K

Kronecker delta function, 65, 79, 90,107,146,250

L

Lagrange multiplier, 163, 164, 165, 167,279,384,403

Local PIM, LPIM, *See* Local point interpolation method,

Local point interpolation method, 238, 383

Local quadrature domain, 34,55, 239, 240,246,248,254,256,385

Local radial PIM, LRPIM, *See* Local radial point interpolation method,

Local radial point interpolation method, 46,52,238,239, 253, 254, 382,407

Local support, 43,54,55,66,76,

149,246

Local weak form, 34,46,238,239, 250,312,381,386,314,425

M

Mass density, 7,12,412,417

Matrix triangulization algorithm, 66,147,238,311

Mesh, 37,42,197

Mesh generation, 38,40

Meshless local Petrov-Galerkin, 45,46,48, 237,250,382,383

MFree method, 39

MFree2D, 201

MFree weak-strong method, *see* MWS

MLPG, *See* Meshless local Petrov-Galerkin method

MLS, *See* Moving least square,

Moment matrix, 63m 67, 71, 98, 147

Moving domain interpolation, 54

Moving least square, MLS, 48,57,97,108,161,250,415

MWS, 47, 381

MTA, *See* Matrix triangulization algorithm

N

Natural boundary conditions, 9, 33,148,158,174,242,366,385

Natural convection, 423

Natural coordinates, 43

Navier-Stokes Equation, 434

Newmark method, 415

Nodal displacement, 31,78,150, 260,367,393,413

Nodal displacement parameter, 163,413,415

Nodal distribution, 42,55,181, 264, 346,349,400,427

Nodal force vector, 162,175, 244, 261, 389

Nodal matrix, 44, 156, 164, 166, 246, 315

Nodal spacing, 58,60,86, 87, 168, 247

Nodal stiffness matrix, 151,152, 156, 244, 251, 388,389

Nonlinear analysis, 201

Normal stress, 4,178,404

Nonconforming RPIM (NRPIM), 160
Numerical integration, 45, 155, 169, 248, 311, 391

O

ODE, 3,14,313,324,336

ordinary differential equation, *see* ODE
Order of polynomial, 78,106,107

Order of continuity, 17,36, 103,106

Order of consistency, 55

P

Partial differential equation, *see* PDE

Partitions of unity, 65,80,107, 112

PUFEM, 46,52

Pascal triangle, 62,98

PDE, 3,8,14,311,346,349,354

Penalty factors, 161,163,250

Penalty stiffness matrix, 162

Penalty method, 159,160,169, 174, 250, 252,384

Polynomial point collocation method, *see* PPCM

Point interpolation method, 45, 49, 60

PIM, *See* Point interpolation method,

PIM shape function, 60

Plane strain, 9,11,148,366,375

Plane stress, 9,11,148,366,393

PPCM, 313, 324, 344

Principles, 20,27,160,369

Poisson's ratio, 7

Polynomial basis, 7,49,61,62,74, 106, 325,346

Post-processor, 180,263

Potential energy, 14,27,31,32,33

Pre-processor, 179

Q

Quadrature domain, 35,55,239, 240, 246, 248,249,254,256, 264, 270,274,385,412

Quartic spline functions, 102, 108, 109,111

R

Radial basis function, *see* (RBF,

RPCM, 352-377

Radial Point Collocation Method, *see* RPCM

Radial point interpolation method, *see* RPIM

Rayleigh numbers, 424,430,432

RBF, 74,75,86,169,352

Relative error, 323,331,346

Reproducing Kernel Particle Method, RKPM, 45,46,48,57

Reproducibility, 27,36,65,66,81

RPIM, 50,52,57,74,79,80,86,148

S

Shape function, 48,54,61, 73, 74, 97

Shape parameters, 74,75,86,92, 118,169

Shear modulus, 7

Shear stress, 4,10,178

Smoothed particle hydrodynamics, *see*, SPH

Smoothing kernel,

Smoothing length,

SPH, 44,47,48,57

Stability, 55,335,336

Standard patch test, 76,240, 401, 406

Stiffness matrix, 157,159,247, 390

Strain-displacement relations, 6,10,33

Strain, 5,6,10,150,242, 387

Strain energy, 31,32

Strong forms, 13,46,47,161,311, 380, 381,387,412

Support domain, 43,54,55,59, 86, 108, 167,170

Surface fitting, 114,115,117,118

System equation, 9,13,44,155, 161, 167,251,424

T

Taylor series, 13,57,80,436

Time step, 361,419,437

Traction boundary condition, 8,12,158,386

Transient response, 420

V

Variational principle, 51,164

Vibration, *see* free vibration, forced vibration

W

Weak forms, 13,27,33,34,45, 47, 146,237,382,383

Weight function domain, 240, 259

Weight (test) functions,16,17, 57,98,102,108,240,241,248,258, , 384,386

weight coefficient, 68,72,73,350, 351,

Weighted residual method, 14, 34,239,250,412

Y

Young's modulus, 7,21,178, 365,412