UNIVERSITY OF LJUBLJANA
FACULTY OF MATHEMATICS AND PHYSICS

Mathematics – 3rd cycle

Jure Slak

ADAPTIVE RBF-FD METHOD

PhD Thesis

Advisor: dr. Gregor Kosec

Ljubljana, 2020

UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 3. stopnja

Jure Slak

ADAPTIVNA RBF-FD METODA

Doktorska disertacija

Mentor: dr. Gregor Kosec

Ljubljana, 2020

Acknowledgments

I would like to acknowledge the financial support given by the ARRS research core funding No. P2-0095 and the Young Researcher program PR-08346.

I would like to thank my mentor Gregor Kosec for his support and guidance during the past years, for many fruitful discussions about the research topic and otherwise, and for giving me the freedom to work independently, yet being there for me when needed. I have learned a lot of things from him, and will continue to do so.

Thanks are extended to Roman, the former head of the Parallel and Distributed Systems Laboratory, to all other members of the laboratory, and the E6 department of the Jožef Stefan Institute, for creating a supporting and empowering environment.

I wish to also thank the PhD committee members, Stéphane Bordas, Bor Plestenjak and Emil Žagar, for carefully reading the thesis, correcting many mistakes and suggesting a lot of improvements, raising the quality of this work.

For enabling me to focus on the research and not worry about much else, I have to thank my parents and my sister, who supported me throughout my studies, and have started and reinforced my interest in math and science.

I would also like to thank my schoolmates and friends, Anja, Vesna and Žiga, for many lunches full of debates, rants and exchanges of various experiences.

Finally, I would like to extend my sincerest thanks to my girlfriend Ines, who stood by my side and shared my troubles, always being supporting and caring and helping me in any way she could. This includes reading the first final drafts of the thesis and without her, it would not have been in the state it is now.

Zahvala

Za začetek bi rad omenil finančno podporo ARRS temeljnega programa P2-0095 in programa mladih raziskovalcev PR-08346.

Svojemu mentorju, Gregorju Koscu, bi se rad zahvalil za podporo in nasvete med preteklimi leti, za veliko plodnih pogovorov o raziskovalnih problemih in drugih stvareh, ter za dano svobodo, potrebno za samostojno delo. Od njega sem se veliko naučil in upam, da se bom še marsičesa.

Zahvaljujem se tudi Romanu Trobcu, kot takratnemu vodji Laboratorija za vzporedno in porazdeljeno računanje, vsem drugim članom laboratorija in odseka E6 Instituta Jožefa Stefana, za prijetno in vzpodbudno delovno okolje.

Rad bi se zahvalil tudi članom komisije, Stéphanu Bordasu, Boru Plestenjaku in Emilu Žagarju za natančno branje disertacije, za mnogo popravkov in pripomb, ki so nedvomno izboljšale kvaliteto dela.

Da sem lahko praktično nemoteno študiral in mi ni bilo treba skrbeti za mnogo drugega, se zahvaljujem svojim staršem in sestri, ki so me podpirali tekom študija ter že od malega vzpodbujali moje zanimanje za znanost in matematiko.

Rad bi se zahvalil svojim sošolcem in prijateljem, posebej Anji, Vesni in Žigi, s katerimi smo imeli mnogo kosil, napolnjenih z debatami, skupinskim pritoževanjem in izmenjavo različnih izkušenj.

Nazadnje bi se rad zahvalil še mojemu dekletu, Ines, ki mi je stala ob strani, me podpirala, si z mano delila vse težave in mi pomagala po svojih najboljših močeh. To vključuje tudi branje prvih končnih osnutkov disertacije in brez nje delo gotovo ne bi bilo v stanju, kot je sedaj.

Abstract

Radial-basis-function-generated finite differences (RBF-FD) is a method for solving partial differential equations (PDEs), which is developed into a fully automatic adaptive method during the course of this work. RBF-FD is a strong form meshless method, which means that it does not require a mesh of the problem domain, but uses only a set of nodes as the basis for the discretization. A large part of this PhD is dedicated to algorithms for meshless node generation. A new algorithm for construction of variable density meshless discretizations in arbitrary spatial dimensions is developed. It can generate points in the interior and on the boundary, has provable minimal spacing requirements, can generate N points in $O(N \log N)$ time and the resulting node sets are compatible with RBF-FD. This algorithm is used as the basis of a newly proposed h-adaptive procedure for elliptic problems. The behavior of the procedure is analyzed on classical 2D and 3D adaptive Poisson problems. Furthermore, several contact problems from linear elasticity are solved, demonstrating successful adaptive derefinement and refinement, with the densest parts of the discretization being more than a million times denser than the coarsest. Finally, the software developed for this work and broader research is presented and published online as an open source library for solving PDEs with strong form methods.

Math. Subj. Class. (2010): 65N50, 65N99, 65D99, 65Y20, 68Q25, 65-04, 74B05, 65D25, 65D05, 65Y05

Keywords: meshfree methods, meshless methods, radial basis functions, partial differential equations, adaptivity, refinement, node generation, finite differences, scattered data

Povzetek

Končne diference, generirane z radialnimi baznimi funkcijami (RBF-FD), so metoda za numerično reševanje parcialnih diferencialnih enačb (PDE), ki jo v delu razvijemo v avtomatsko adaptivno metodo. RBF-FD spada med brezmrežne metode, ki enačbe rešujejo v močni obliki. Brezmrežnost pomeni, da metoda ne potrebuje diskretizacije domene problema v obliki mreže, temveč lahko za diskreten izračun uporabi le množico ustrezno razporejenih točk. Velik del doktorata je posvečen algoritmom za generiranje diskretizacijskih točk. Razvit je tudi nov algoritem za konstrukcijo diskretizacij v poljubnih dimenzijah s prostorsko spremenljivo diskretizacijsko razdaljo. Algoritem je primeren za generiranje točk v notranjosti in na robu domene, dokazano ohranja predpisano minimalno razdaljo med točkami in potrebuje $O(N \log N)$ časa za generiranje N točk. Konstruirane množice točk so kompatibilne z RBF-FD metodo in posledično algoritem uporabimo kot osnovo novega postopka za h-adaptivno reševanje eliptičnih problemov. Obnašanje postopka je analizirano na klasičnih dvo- in tro-dimenzionalnih Poissonovih problemih. Poleg tega je rešenih tudi več kontaktnih problemov iz linearne elastostatike, s katerimi pokažemo uspešno goščenje in redčenje diskretizacije, ki se avtomatsko prilagaja problemu, pri čemer razmerje med najgostejšimi in najredkejšimi deli diskretizacije naraste tudi do več milijonov. Na koncu je predstavljena programska oprema, razvita za to delo in širše raziskave, objavljena pa je tudi na spletu kot odprtokodna knjižnica, namenjena reševanju PDE v močni obliki z brezmrežnimi metodami.

Math. Subj. Class. (2010): 65N50, 65N99, 65D99, 65Y20, 68Q25, 65-04, 74B05, 65D25, 65D05, 65Y05

Ključne besede: brezmrežne metode, radialne bazne funkcije, parcialne diferencialne enačbe, adaptivnost, zgoščevanje, generiranje točk, končne diference, razpršeni podatki

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

Numerical solving of Partial Differential Equations (PDEs) began in 1911, when Richardson used the now classical finite difference approximation to compute stresses in a dam [Ric11]. From the beginning the practical applications drove the development of numerical methods for solving PDEs. Geometrical constraints of regular grids were too severe, and the Finite Element Method (FEM) was developed in the 1940s [Cou43; Hre41], which uses a division of the domain into simpler parts as the basis of the discretization and solves the problem in its weak form. It rose in popularity in the following decades along with the openly available software implementations. Another benefit of the method are its solid mathematical foundations, provided by Strang and Fix [SF73] or Ciarlet and Raviart [CR73] in 1973, along with later developed error and stability analyses. Other popular methods, such as the Finite Volume Method, also use similar mesh-based discretizations and the weak form of the problem. However, mesh generation has always been a difficult problem. It is often the most difficult part of the computational procedure, that often requires manual intervention and cannot be fully automated, especially in 3D. Problems with concentrated solution gradients, such as local stress concentrations in elastostatics, require finer meshes in critical areas. Furthermore, the areas where a fine mesh is required are not necessarily known beforehand and dynamic adaptation of the mesh might be desired, which can further complicate the mesh generation.

As a response to this difficulties, numerical methods have been developed that do not require a mesh for discretization of the domain. Some only use a mesh on the boundary, such as the Boundary Element Method (BEM), and others only use a set of nodes in the domain interior and on the boundary. Such methods were named *meshless* or *mesh-free* methods. The connectivity relation between nodes that defines a mesh is substituted with a list of local neighboring nodes. Despite the simplicity of the meshless discretization when compared to mesh-based discretization, the points still cannot be positioned arbitrarily and must not be too close nor leave too large parts of the domain uncovered, since this might cause instabilities or lack of convergence. One aim of this thesis is to provide an efficient node generator that produces variable density node sets for irregular domains in arbitrary dimensions.

A recently popularized meshless method is the radial-basis-function-generated finite differences (RBF-FD) method, a generalization of the finite differences method to scattered nodes that shows good error and stability behavior. A downside of many meshless methods, including RBF-FD, is a lack of solid mathematical foundations, such as global error estimates and stability criteria. Nonetheless, RBF-FD has been increasingly used in practical applications. The other goal of this thesis is to explore its refinement potential, and develop a fully automatic adaptive solution procedure, which adapts the nodal distribution to the problem requirements using the developed node generation algorithm.

Thesis outline

The thesis presents the necessary tools for fully automatic adaptivity in order: it first discusses function approximation on scattered data, then local strong form approximations of differential operators, then the construction of domain discretizations and finally, adaptivity. Specifically:

- Chapter 1 discusses function approximation on scattered data, which is a prerequisite for all meshless methods considered in this work, and for adaptivity itself. It provides a summary of the main methods used for scattered data approximation with the focus on RBF interpolation. The non-singularity theorems for RBF interpolation with Gaussians, Multiquadrics, Inverse multiquadrics and polyharmonics are proven. We also review error and stability results to help with the understanding of the behavior of local operator approximations.
- Chapter 2 introduces the RBF-FD method for solving PDEs. It is put in comparison with least-squares based methods and some illustrative examples are discussed, to address the problems commonly faced when using RBFs with shape parameters. This helps us guide the decision towards using monomials augmented polyharmonic splines.
- Chapter 3 defines the domain discretization and the procedure to construct it. The algorithms for point placing in the domain interior and on the boundary are presented and compared with two other algorithms. We prove several properties of the algorithms and the produced discretizations and analyze other important properties empirically. This chapter lays the groundwork for adaptivity.
- Chapter 4 presents the fully automatic adaptive procedure for elliptic problems. The behavior of the procedure is analyzed on classical problems and then the procedure is applied to contact problems.
- Chapter 5 gives a short overview of the implementation and the Medusa library, including the reasoning for some of the design choices. It is not meant to serve as technical documentation (that is the role of `http://e6.ijs.si/medusa/docs/`) but as a description of ideas that made the creation of the library possible, and the importance of having it available.

Literature

The main sources for the main topics in this work are as follows:

- Scattered data approximation topics were mainly sourced from [Wen04].
- The content regarding RBF-FD and similar methods was sourced mainly from the book [FF15a] and review papers [Ngu+08] and [FF15c].
- Domain discretization algorithms that we compare our algorithms with are published in papers [FF15b] and [SKF18].

- Linear elasticity topics are mainly sourced from [Sla12] with additional material from [Sad14]
- Contact problems were mainly sourced from the book [WD00].

Contributions

Chapters 3 and 4 present the original work of the author, Jure Slak. The Medusa library, presented in Chapter 5 is the work of Gregor Kosec and Jure Slak along with other contributors to the code and related materials as listed in [SK19e].

Parts of this work have been published in the following papers, or are included in the following pre-prints.

[SK19b] J. Slak and G. Kosec, *Refined meshless local strong form solution of Cauchy–Navier equation on an irregular domain*, Engineering Analysis with Boundary Elements **100**:3–13, Mar. 2019, DOI: 10.1016/j.enganabound.2018.01.001 (cited on pp. 93, 118, 121).

[SK19c] J. Slak and G. Kosec, *Adaptive radial basis function-generated finite differences method for contact problems*, International Journal for Numerical Methods in Engineering **119**(7):661–686, Aug. 2019, DOI: 10.1002/nme.6067 (cited on pp. 93, 94, 107).

[SK19d] J. Slak and G. Kosec, *On generation of node distributions for meshless PDE discretizations*, SIAM Journal on Scientific Computing **41**(5):A3202–A3229, Oct. 2019, DOI: 10.1137/18M1231456 (cited on pp. 59, 61, 62, 85, 88).

[SK19e] J. Slak and G. Kosec, *Medusa: A C++ library for solving PDEs using strong form mesh-free methods*, arXiv:1912.13282, Dec. 2019, URL: https://arxiv.org/abs/1912.13282 (cited on pp. 3, 158).

[DKS20] U. Duh, G. Kosec, and J. Slak, *Fast variable density node generation on parametric surfaces with application to mesh-free methods*, arXiv:2005.08767, May 2020, URL: https://arxiv.org/abs/2005.08767 (cited on p. 70).

The present author also co-authored the following papers and pre-prints.

[JSK19] M. Jančič, J. Slak, and G. Kosec, *Analysis of high order dimension independent RBF-FD solution of Poisson's equation*, arXiv:1909.01126, Sept. 2019, URL: https://arxiv.org/abs/1909.01126 (cited on p. 92).

[Kos+19] G. Kosec, J. Slak, M. Depolli, R. Trobec, K. Pereira, S. Tomar, T. Jacquemin, S. P. A. Bordas, and M. A. Wahab, *Weak and strong from meshless methods for linear elastic problem under fretting contact conditions*, Tribology International **138**:392–402, Oct. 2019, DOI: 10.1016/j.triboint.2019.05.041 (cited on pp. 121, 123).

[Mak+19] M. Maksić, V. Djurica, A. Souvent, J. Slak, M. Depolli, and G. Kosec, *Cooling of overhead power lines due to the natural convection*, International Journal of Electrical Power & Energy Systems **113**:333–343, Dec. 2019, DOI: 10.1016/j.ijepes.2019.05.005.

[Moč+20] J. Močnik Berljavac, P. K. Mishra, J. Slak, and G. Kosec, *RBF-FD analysis of 2D time-domain acoustic wave propagation in heterogeneous Earth's subsurface*, arXiv:2001.01597, Jan. 2020, URL: https://arxiv.org/abs/2001.01597.

Additionally, the author presented the following conference papers at international conferences [SK16], [SK18b], [SK18a], [SK18d], [KS18c], [KS18b], [SK19b], [SSK19a], [KS19b], [KS19a], [SK20] and coauthored the following conference papers: [KS18a], [MSK19], [SK18c], [SSK19b], [DKS19], [Duh+20], [JSK20].

Hardware and software

All numerical simulations were performed on a laptop computer with Intel® Core™ i7-7700HQ CPU @ 2.80GHz processor and 16 GB of DDR4 SODIMM memory with 2400 MT/s transfer rate.

All described procedures were implemented in C++17 [ISO17] and compiled with g++ (Arch Linux 9.3.0-1) 9.3.0 on Linux kernel 4.19.121-1-MANJARO. By default, flags

```
-std=c++17 -fopenmp -O3 -Wall -Wextra -Wfloat-conversion
-Wno-deprecated-copy -Wno-maybe-uninitialized -pedantic -DNDEBUG
```

were used. Additionally, Eigen was used with the Intel®Math Kernel library (MKL) [Int18] with the Pardiso sparse linear solver. The cmake build system (v. 3.17.2) was used to build the executables. The software for all analyses in this work, for producing figures and .tex sources as well are available at: https://gitlab.com/jureslak/phd.

The software makes use of the Medusa library (v. 3e7409f64), which is available at https://gitlab.com/e62Lab/medusa. The Medusa library in turn includes the Eigen library for matrix manipulation (version 3.3.7 with manually added multi-indexing support) [GJ+10], nanoflann library for k-d trees [BR14], RapidXml [Kal11] for handling XML files, HDF5 software suite for handling HDF5 files [Fol+11] and [Fos+11] for output formatting.

Most post-processing was done with Matlab [Mat17], which includes the preparation of most figures, except for the ones explicitly mentioned below. The external Matlab package export_fig [Alt20] was extremely helpful with exporting high-quality images. Drawings in figures 1.3, 3.1, 3.4, 4.11, 4.16, 4.19, 4.23 and 4.24 were created with Inkscape graphics editor [Ink20].

Chapter 1

Function approximation on scattered data

The problem of function interpolation on scattered data is formulated as follows: given *data sites* or *nodes* $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$ and function values $u_i = u(\boldsymbol{x}_i)$, construct a function $\hat{u}$, such that $\hat{u}(\boldsymbol{x}_i) \approx u_i$. An example of such setup is shown in Figure 1.1. The data sites X are assumed to be *scattered*, i.e., having no regular structure. If that were not the case, and the data sites were distributed, for example, in a regular grid, approximation schemes from one-dimensional interpolation can be generalized and applied.

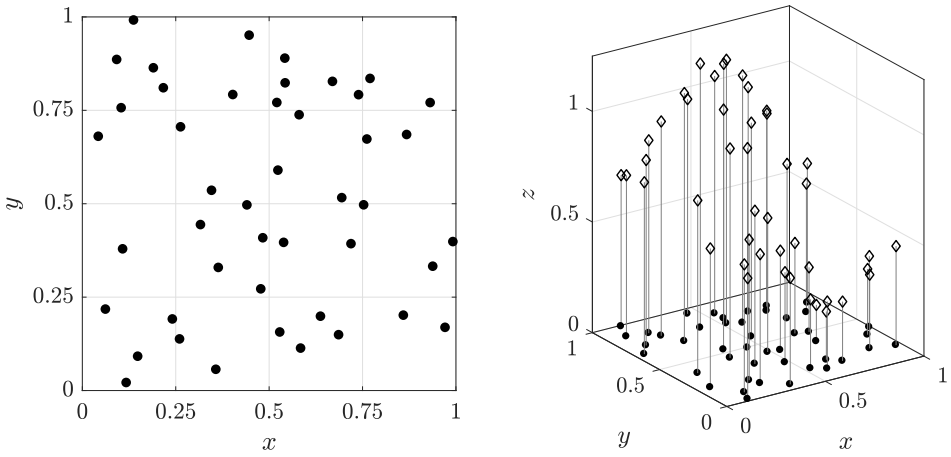


Figure 1.1: An example set of scattered nodes $X = \{\boldsymbol{x}_i = (x_i, y_i)\}_{i=1}^{45}$ (left) and their corresponding function values u_i (right). Function values are sampled from a function $u_{\text{ex}}(x, y) = \cos(5x(1-y)) \exp(-6\left((x-0.5)^2 + (y-0.6)^2\right)) + 0.5$.

One option is to add some structure to the data sites by e.g. triangulation and then use triangulation-based interpolation schemes. This option can be inefficient in practice, and can cause artifacts in the interpolant that originate from the triangulation itself and not from the data. An example of this is shown in Figure 1.2.

Another option is to extend polynomial interpolation to a multivariate case, but the Mairhuber-Curtis theorem tells us that there will inevitably be some point configurations, where the interpolation problem is not well-posed. Radial basis function interpolation, first used by Hardy in 1970s [Har71] does not require any additional structure on

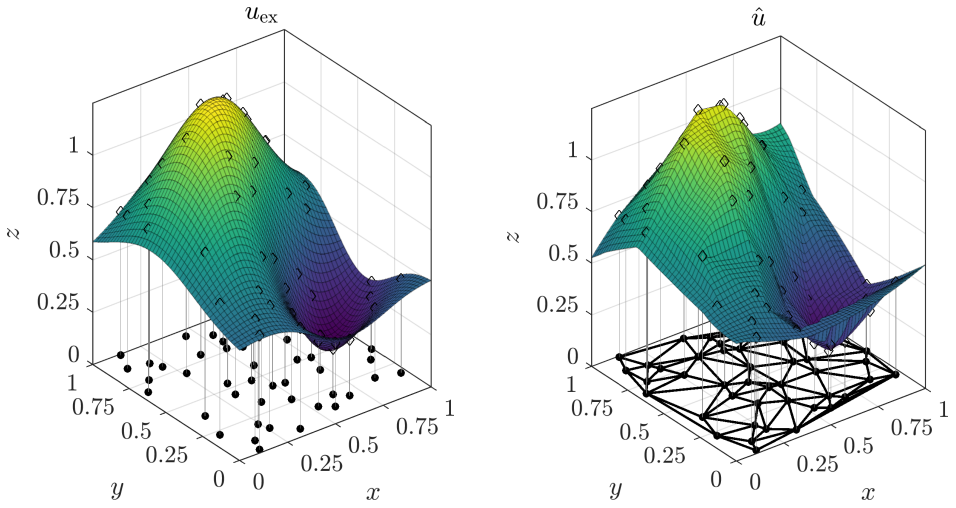


Figure 1.2: The original function u_{ex} (left) and the reconstruction $\hat{u}$ from pairs $(\boldsymbol{x}_i, u_i)$, using linear interpolation over the Delaunay triangulation of X (right). The artifacts from triangulation edges are clearly visible in $\hat{u}$.

the nodes. Later, the connection between positive definite kernels and RBF-interpolation was discovered and many well-posedness theorems for RBF interpolation were proven. In this chapter, we review the methods for scattered data interpolation with emphasis on RBF interpolation. First, we review the results of the Mairhuber-Curtis theorem in Section 1.1, followed by an introduction to positive definite kernels in Section 1.2. In Section 1.3 we review known methods for scattered approximation that are most commonly used with meshless methods, i.e. Sheppard's interpolation, moving least squares and RBF interpolation, where we also prove the RBF nonsingularity theorems. Finally, some error bounds and stability observations for RBF interpolation are given in Section 1.4. Further information on RBFs and their usage can be found in the monograph by Buhmann [Buh03].

1.1 Mairhuber-Curtis theorem

Mairhuber-Curtis theorem deals with generalization of polynomial interpolation from 1-dimensional case to higher-dimensional spaces. Univariate polynomial interpolation has the property that given data sites $x_1 < x_2 < \ldots < x_n$ and their corresponding function values f_i, there exists a unique interpolating polynomial $p \in \mathbb{P}_{n-1}(\mathbb{R})$, such that $p(x_i) = f_i$.

This is a desirable property, that motivates the definition of Haar spaces. The following definitions and ideas for the proofs follow [Wen04, p. 19].

Definition 1.1.1 (Haar space)**.** Suppose that $\Omega \subset \mathbb{R}^d$ contains at least n points. A linear space $V \subset C(\Omega)$ of continuous functions is called a Haar space iff for any set of distinct data sites $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \Omega$ and corresponding function values $\{f_1, \ldots f_n\} \subseteq \mathbb{R}$ there exists exactly one function $p \in V$, such that $p(\boldsymbol{x}_i) = f_i$, for $i = 1, \ldots, n$.

With above definition, we can say that $\mathbb{P}_{n-1}(\mathbb{R})$ is a n-dimensional Haar space for any $\Omega \subseteq \mathbb{R}$ with at least n distinct points.

An alternative characterization of Haar spaces is given in the following statement.

Proposition 1.1.2 (Characterization of Haar spaces)**.** *Suppose that $\Omega \subset \mathbb{R}^d$ contains at least n points. Then, V is a Haar space iff for any distinct points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \Omega$ and any basis $\{u_1, \ldots, u_n\}$ of V, the determinant of the collocation matrix is nonzero:*

$$\det(u_j(\boldsymbol{x}_i)) \neq 0. \tag{1.1.1}$$

Proof. We prove the stated equivalence by proving that $\det(u_j(\boldsymbol{x}_i)) \neq 0$ implies that V is a Haar space and that $\det(u_j(\boldsymbol{x}_i)) = 0$ implies that V is not a Haar space.

Let the determinant of the collocation matrix be nonzero and let $\{(\boldsymbol{x}_i, f_i)\}$ be the data sites and their corresponding values. Since $u_1, \ldots, u_n$ form a basis of V, the sought function p must be of the form $p = \sum_{j=1}^{n} \alpha_j u_j$. The interpolation conditions

$$\sum_{j=1}^{n} \alpha_j u_j(\boldsymbol{x}_i) = f_i \tag{1.1.2}$$

form a system of linear equations, whose determinant is nonzero by assumption, and thus its solution, the coefficients α_j, are unique.

Conversely, let the collocation matrix $A = [u_j(\boldsymbol{x}_i)]_{i,j=1}^{n}$ be singular, which means that there exists a nonzero vector $\boldsymbol{\alpha}$ in its kernel. Writing $A\boldsymbol{\alpha} = \mathbf{0}$ explicitly gives

$$\forall i = 1, \ldots, n: \quad \sum_{j=1}^{n} \alpha_j u_j(\boldsymbol{x}_i) = 0, \tag{1.1.3}$$

which means that the function $p = \sum_{j=1}^{n} \alpha_j u_j$ interpolates values $\{(\boldsymbol{x}_i, 0)\}_{i=1}^{n}$. But, so does the zero function, and since $\boldsymbol{\alpha} \neq 0$, the interpolant is non-unique and V is not a Haar space. □

The Mairhuber-Curtis theorem deals with existence of Haar spaces in higher spatial dimensions. The result is famously negative, except in simple situations, and it gives a hint towards alternative interpolation schemes.

Theorem 1.1.3 (Mairhuber-Curtis)**.** *Let $d \geq 2$ and suppose that $\Omega \subseteq \mathbb{R}^d$ contains at least n points and an interior point. Then there exists no Haar space on Ω of dimension $n \geq 2$.*

Proof. Let U be any linear subspace of $C(\Omega)$ with basis $\{u_1, \ldots, u_n\}$ and $\boldsymbol{x}_0$ be an interior point of Ω. Since $\boldsymbol{x}_0$ is an interior point, there exists a radius r, such that $B(\boldsymbol{x}_0, r) \subseteq \Omega$. Because $d \geq 2$ and $n \geq 2$, we can have the following configuration: let $\boldsymbol{x}_1(t)$ and $\boldsymbol{x}_2(t)$ be continuous paths with $t \in [0, 1]$, such that $\boldsymbol{x}_1(0) = \boldsymbol{x}_2(1)$ and $\boldsymbol{x}_2(0) = \boldsymbol{x}_1(1)$, but are otherwise disjoint. Additionally, choose distinct points $\boldsymbol{x}_3, \ldots \boldsymbol{x}_n$ that do not lie on any of the curves $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$. An example of such configuration is shown in Figure 1.3.

Let us consider the determinant of the collocation matrices

$$D(t) := \det([u_j(\boldsymbol{x}_i(t))]_{i,j=1}^{n}). \tag{1.1.4}$$

The collocation matrices at $t = 0$ and $t = 1$ only have the first two rows swapped, and so $D(0) = -D(1)$ must hold. But, since $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are continuous, so is $D(t)$, and by intermediate value theorem there exists a t^*, such that $D(t^*) = 0$.

We now have the situation that for distinct points $\{\boldsymbol{x}_1(t^*), \boldsymbol{x}_2(t^*), \boldsymbol{x}_3, \ldots, \boldsymbol{x}_n\}$ the determinant of the collocation matrix is zero, and by the characterization of Haar spaces (Proposition 1.1.2), U is not a Haar space. □

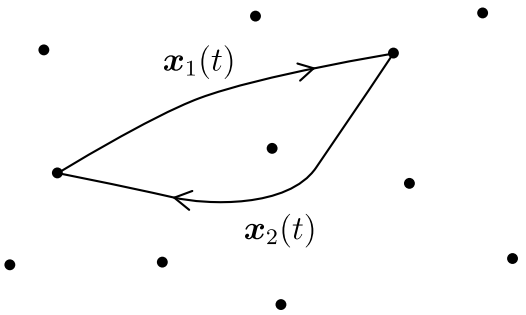


Figure 1.3: An example configuration used in the proof of the Mairhuber-Curtis theorem.

The consequence of the Mairhuber-Curtis theorem is that any functional space will always have singular point configurations, when considering multivariate interpolation (of at least 2 points). The take-away is, as succinctly put by Fasshauer [Fas11, p. 14], that "*The linear function space used for multivariate interpolation should be data-dependent*". One promising option on how to achieve that is presented in the next section.

1.2 Positive definite kernels and reproducing kernel Hilbert spaces

A simple approach to data-dependent scattered data interpolation is to use kernel-based function space, with basis of the form $\{\mathcal{K}(\cdot, \boldsymbol{x}_1), \ldots, \mathcal{K}(\cdot, \boldsymbol{x}_n)\}$, where $\mathcal{K}\colon \Omega \times \Omega \to \mathbb{R}$ is a function, called *kernel*.

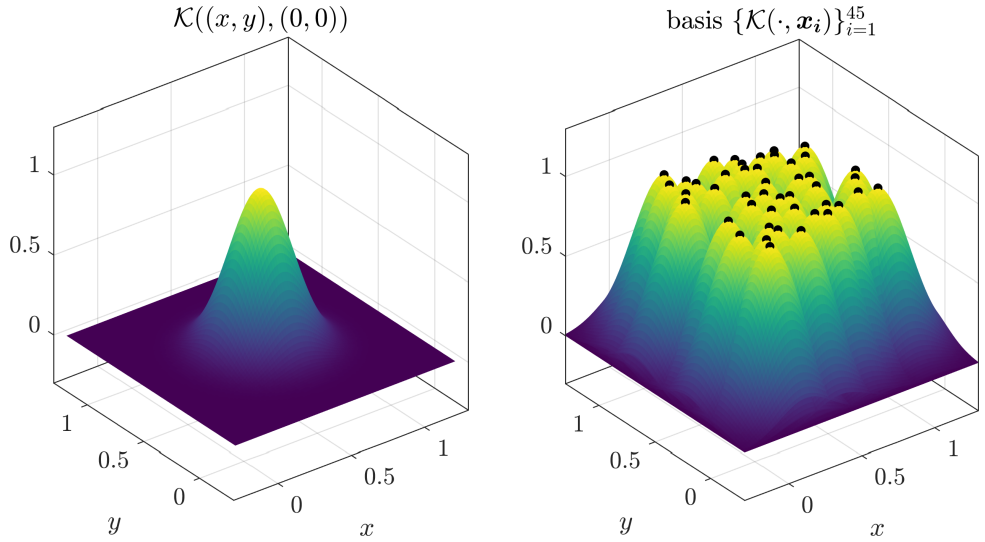


Figure 1.4: Plot of a radial kernel $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \exp((\varepsilon\|\boldsymbol{x} - \boldsymbol{y}\|)^2), \varepsilon = \frac{1}{4}$ (left) and a node-dependent basis generated by $\mathcal{K}$ (right).

An interpolant constructed on data sites $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ with function values f_i from this space would be of the form

$$p(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_j) \tag{1.2.1}$$

and the interpolation conditions

$$\forall i = 1, \dots, n: \quad p(\boldsymbol{x}_i) = f_i \tag{1.2.2}$$

can be assembled in a system of linear equations

$$\begin{bmatrix} \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_1) & \cdots & \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \tag{1.2.3}$$

compactly written as $K\boldsymbol{\alpha} = \boldsymbol{f}$.

An important family of kernels are symmetric positive definite kernels.

Definition 1.2.1 (Symmetric kernel)**.** A kernel $\mathcal{K}\colon \Omega \times \Omega \to \mathbb{R}$ is called symmetric, iff $K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{y}, \boldsymbol{x})$ for all $\boldsymbol{x}, \boldsymbol{y} \in \Omega$.

Definition 1.2.2 (Positive definite kernel)**.** A symmetric kernel $\mathcal{K}\colon \Omega \times \Omega \to \mathbb{R}$ is called positive definite on Ω iff its associated matrix $K = [\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$ is positive definite for any $n \in \mathbb{N}$ and for any set of distinct points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subseteq \Omega$.

It follows directly from the definition of positive definite kernels that linear systems of the form (1.2.3) arising from scattered data interpolation are nonsingular, since the matrix K is guaranteed to be symmetric and positive definite (thus invertible).

It is important to note that the existence of a basis $\{\mathcal{K}(\cdot, \boldsymbol{x}_1), \dots, \mathcal{K}(\cdot, \boldsymbol{x}_n)\}$ is not in contradiction to Mairhuber-Curtis theorem due to its data-dependence. The current proof of the theorem fails for data dependent bases, because when the points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are continuously swapped, two columns of the interpolation matrix swap along with the rows, not causing a sign change in the determinant, making us unable to deduce anything about the zeros of the determinant of the collocation matrix.

One of the simplest kernels are the *radial* kernels. These are kernels of the form

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \phi(\|\boldsymbol{x} - \boldsymbol{y}\|), \tag{1.2.4}$$

where $\phi\colon [0, \infty) \to \mathbb{R}$ is a univariate function, and $\|\cdot\|$ is usually the Euclidean norm. Interpolation with these types of kernels is called interpolation with radial basis functions, and will be considered in more detail later, in Section 1.3.3, along with the characterization of functions ϕ which generate positive definite kernels.

Positive definite kernels are strongly tied with reproducing kernel Hilbert spaces, i.e. Hilbert spaces which posses a *reproducing kernel*.

Definition 1.2.3 (Reproducing kernel)**.** Let $\mathcal{H}$ be a Hilbert space of functions $f\colon \Omega \to \mathbb{R}$. A kernel $\mathcal{K}\colon \Omega \times \Omega \to \mathbb{R}$ is called a reproducing kernel for $\mathcal{H}$ if functions $\mathcal{K}(\cdot, \boldsymbol{y}) \in \mathcal{H}$ for all $\boldsymbol{y} \in \Omega$ and $f(\boldsymbol{y}) = \langle f, \mathcal{K}(\cdot, \boldsymbol{y}) \rangle$ for all $f \in \mathcal{H}$ and $\boldsymbol{y} \in \Omega$.

Proposition 1.2.4 (Uniqueness of reproducing kernels)**.** *The reproducing kernel of a Hilbert space is unique.*

Proof. Suppose we have two reproducing kernels, $\mathcal{K}_1$ and $\mathcal{K}_2$. Since $f(\boldsymbol{y}) = \langle f, \mathcal{K}_1(\cdot, \boldsymbol{y}) \rangle$ and $f(\boldsymbol{y}) = \langle f, \mathcal{K}_2(\cdot, \boldsymbol{y}) \rangle$, we have

$$\langle f, \mathcal{K}_1(\cdot, \boldsymbol{y}) - \mathcal{K}_2(\cdot, \boldsymbol{y}) \rangle = 0 \tag{1.2.5}$$

for all $f \in \mathcal{H}$ and $y \in \Omega$. Since $\mathcal{K}_1(\cdot, \boldsymbol{y}) \in \mathcal{H}$ and $\mathcal{K}_2(\cdot, \boldsymbol{y}) \in \mathcal{H}$, so is $\mathcal{K}_1(\cdot, \boldsymbol{y}) - \mathcal{K}_2(\cdot, \boldsymbol{y})$ and substituting this nor f in (1.2.5), we obtain $\|\mathcal{K}_1(\cdot, \boldsymbol{y}) - \mathcal{K}_2(\cdot, \boldsymbol{y})\|^2 = 0$ for all $\boldsymbol{y} \in \Omega$. □

To understand how reproducing kernels are connected to positive definite kernels, we will need a connection with the dual space, given by the following characterization.

Proposition 1.2.5 (Characterization of reproducing kernel Hilbert spaces)**.** *$\mathcal{H}$ has a reproducing kernel if and only if all the point evaluation functionals are continuous.*

Proof. We denote the point evaluation functions at point $\boldsymbol{y} \in \Omega$ with $\delta_{\boldsymbol{y}}$. Using the reproducing kernel $\mathcal{K}$, we can express $\delta_{\boldsymbol{y}}(f) = \langle f, \mathcal{K}(\cdot, \boldsymbol{y})\rangle$. Its continuity follows from the continuity of the inner product.

Conversely, assume that all point evaluation functions $\delta_{\boldsymbol{y}}$ are continuous. By Riesz's representation theorem, there exists a function $k_{\boldsymbol{y}} \in \mathcal{H}$, such that $\delta_{\boldsymbol{y}}(f) = \langle f, k_{\boldsymbol{y}}\rangle$ for all $f \in \Omega$. Defining $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = k_{\boldsymbol{y}}(\boldsymbol{x})$ gives the reproducing kernel. □

With above characterization, we are equipped to show some of the special features that reproducing kernels have.

Proposition 1.2.6 (Properties of reproducing kernels)**.** *Let $\mathcal{H}$ be a real Hilbert space of functions $f\colon \Omega \to \mathbb{R}$ with reproducing kernel $\mathcal{K}$. Then*

- $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \langle \mathcal{K}(\cdot, \boldsymbol{x}), \mathcal{K}(\cdot, \boldsymbol{y})\rangle_{\mathcal{H}} = \langle \delta_{\boldsymbol{x}}, \delta_{\boldsymbol{y}}\rangle_{\mathcal{H}^*}$
- *$\mathcal{K}$ is symmetric, i.e.* $\forall \boldsymbol{x}, \boldsymbol{y} \in \Omega\colon \mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{K}(\boldsymbol{y}, \boldsymbol{x})$
- *Convergence in norm implies pointwise convergence, i.e. if $f_n \to f$ in $\|\cdot\|_{\mathcal{H}}$, then $f_n(\boldsymbol{x}) \to f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \Omega$.*

Proof. The inner product of $\delta_{\boldsymbol{x}}$ and $\delta_{\boldsymbol{y}}$ can be computed using Riesz' representation mapping $\mathcal{R}\colon \mathcal{H}^* \to \mathcal{H}$ and using the fact that $\mathcal{R}(\delta_{\boldsymbol{y}}) = \mathcal{K}(\cdot, y)$ which was shown in proof of Proposition 1.2.5:

$$\langle \delta_{\boldsymbol{x}}, \delta_{\boldsymbol{y}}\rangle_{\mathcal{H}^*} = \langle \mathcal{R}\delta_{\boldsymbol{x}}, \mathcal{R}\delta_{\boldsymbol{y}}\rangle_{\mathcal{H}} = \langle \mathcal{K}(\cdot, \boldsymbol{x}), \mathcal{K}(\cdot, \boldsymbol{y})\rangle. \tag{1.2.6}$$

Using the reproducing property of $\mathcal{K}$, we get that

$$\langle \mathcal{K}(\cdot, \boldsymbol{x}), \mathcal{K}(\cdot, \boldsymbol{y})\rangle = \mathcal{K}(\boldsymbol{y}, \boldsymbol{x}), \tag{1.2.7}$$

but, using the symmetry of the inner product we have

$$\langle \mathcal{K}(\cdot, \boldsymbol{x}), \mathcal{K}(\cdot, \boldsymbol{y})\rangle = \langle \mathcal{K}(\cdot, \boldsymbol{y}), \mathcal{K}(\cdot, \boldsymbol{x})\rangle = \mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) \tag{1.2.8}$$

which shows the first and second properties. The third property follows by estimating

$$|f_n(\boldsymbol{x}) - f(\boldsymbol{x})| = |\langle f_n - f, \mathcal{K}(\cdot, \boldsymbol{x})\rangle| \leq \|f_n - f\|\|\mathcal{K}(\cdot, \boldsymbol{x})\| \tag{1.2.9}$$

using Cauchy-Schwarz inequality. □

Finally, we are able to show that all reproducing kernels are positive semi-definite.

Theorem 1.2.7 (Positive semi-definiteness of reproducing kernels)**.** *A reproducing kernel $\mathcal{K}$ is positive semi-definite. Additionally, $\mathcal{K}$ is positive definite if and only if the point evaluation functionals are linearly independent.*

Proof. We consider the quadratic form $\boldsymbol{\alpha}^\mathsf{T} K \boldsymbol{\alpha}$ with $K = [\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$ for any $n \in \mathbb{N}$, for any set of distinct points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subseteq \Omega$ and for any $\boldsymbol{\alpha} \in \mathbb{R}^n$, $\boldsymbol{\alpha} \neq \mathbf{0}$. We know that $\mathcal{K}$ is symmetric from Proposition 1.2.6. To see that it is positive-semi definite, we observe that

$$\sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle \delta_{\boldsymbol{x}_i}, \delta_{\boldsymbol{x}_j} \rangle_{\mathcal{H}^*} \tag{1.2.10}$$

$$= \left\langle \sum_{i=1}^n \alpha_i \delta_{\boldsymbol{x}_i}, \sum_{j=1}^n \alpha_j \delta_{\boldsymbol{x}_j} \right\rangle_{\mathcal{H}^*} \tag{1.2.11}$$

$$= \left\| \sum_{i=1}^n \alpha_i \delta_{\boldsymbol{x}_i} \right\|_{\mathcal{H}^*}^2 \geq 0. \tag{1.2.12}$$

The final sum can be zero with nonzero $\boldsymbol{\alpha}$ only if $\delta_{\boldsymbol{x}_i}$ are linearly dependent. □

Reproducing kernel Hilbert spaces are very helpful in establishing error bounds or optimality properties for interpolants. However, in practice, we often start with a positive definite kernel, without the Hilbert space it belongs to. The construction of the reproducing kernel Hilbert space is based on the idea that it must include all functions $f = \sum_{j=1}^n \alpha_j \mathcal{K}(\cdot, \boldsymbol{x}_j)$ and that for such f, if reproducing kernel Hilbert space $\mathcal{H}$ would already be known, it would hold that

$$\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle \mathcal{K}(\cdot, \boldsymbol{x}_i), \mathcal{K}(\cdot, \boldsymbol{x}_j) \rangle_{\mathcal{H}} = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{1.2.13}$$

This fact can be used to define $\mathcal{H}_\mathcal{K}(\Omega) = \operatorname{span}\{\mathcal{K}(\cdot, \boldsymbol{y});\ \boldsymbol{y} \in \Omega\}$ equipped with a bilinear form

$$\left(\sum_{i=1}^n \alpha_i \mathcal{K}(\cdot, \boldsymbol{x}_i), \sum_{j=1}^n \beta_j \mathcal{K}(\cdot, \boldsymbol{y}_j) \right)_\mathcal{K} = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \beta_j \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{y}_j). \tag{1.2.14}$$

It can be shown [Wen04, p. 137] that $(\cdot, \cdot)_\mathcal{K}$ as defined above defines an inner product on $\mathcal{H}_\mathcal{K}$ with $\mathcal{K}$ as the reproducing kernel. Space $\mathcal{H}_\mathcal{K}$ defined this way is not yet a Hilbert space (it is in fact a pre-Hilbert space), but a suitable completion can be found. This Hilbert space is then called *native space* for a given kernel and denoted by $\mathcal{N}_{\mathcal{K},X}$. It will be useful for proving error bounds for RBF interpolation in Section 1.4.

1.3 Methods for scattered data approximation

We will now review three methods for meshless scattered data interpolation, that are also commonly used as a foundation for strong form meshless methods: Sheppard's interpolation, Moving Least Squares (MLS) and RBF interpolation.

1.3.1 Sheppard's interpolation

Sheppard's interpolation, sometimes called "inverse distance weighting" is a simple approach to scattered data interpolation developed by Sheppard in 1960s [She68]. The function value at a point $\boldsymbol{x}$ is computed as a weighted sum of all the data sites

$$\hat{u}(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}, & \text{if } \boldsymbol{x} \in X, \\ \dfrac{\sum_{i=1}^{n} w_i(\boldsymbol{x}) u_i}{\sum_{i=1}^{n} w_i(\boldsymbol{x})}, & \text{otherwise,} \end{cases} \tag{1.3.1}$$

where the weight $w_i(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}_i\|^{-p}$ is given by p-th power of the inverse distance. An example of this is shown in Figure 1.5. Continuity of $\hat{u}$ depends on the continuity of the weights w_i. The most common choice for exponents is $p = 2$, with large values of p causing the points closer to $\boldsymbol{x}$ to have larger influence than the ones further away and as $p \to \infty$ the scheme degrades into nearest-neighbor interpolation, where every point is assigned the value of its nearest data site, and the resulting interpolant is a piecewise constant function on Voronoi cells of X.

Modified Sheppard's interpolation

When number of data sites n is large, it can become computationally inefficient to compute Sheppard's interpolant (1.3.1). A modification was soon proposed [GW78] to restrict the weighted average only to closest nodes within a given radius R, which is equivalent to using the weights

$$w_i(\boldsymbol{x}) = \left(\frac{\max(0, R - \|\boldsymbol{x} - \boldsymbol{x}_i\|)}{R\|\boldsymbol{x} - \boldsymbol{x}_i\|} \right)^2. \tag{1.3.2}$$

This approximation makes the computation more efficient and the approximant $\hat{u}$ remains continuous and well defined, provided that the sphere around $\boldsymbol{x}$ of radius R includes at least one point. Because the sum over the data sites can omit points outside of radius R, the computation of the interpolant can be sped up using data structures that support spatial range queries, such as k-d trees. Thus, $O(n \log n)$ preprocessing time is needed, but each evaluation costs $O(k)$, where k is the largest number of data sites in a sphere of radius R.

The weighted sum in (1.3.1) can also be restricted to k nearest nodes, for $k \ll n$. We will call this k-Sheppard's approximation. This also allows for a significant improvement in running time over the classical version, from $O(n)$ to $O(k)$ with $O(n \log n)$ preprocessing, where k is now a fixed integer and not a variable number of nodes, making this version more robust in terms computational requirements. However, the approximant $\hat{u}$ is not necessarily continuous along the edges of the Voronoi cells induced by nodes X. An extreme case of this is 1-Sheppard's interpolation, which is just nearest neighbor interpolation. Sheppard's interpolation and its variants are shown in Figure 1.5.

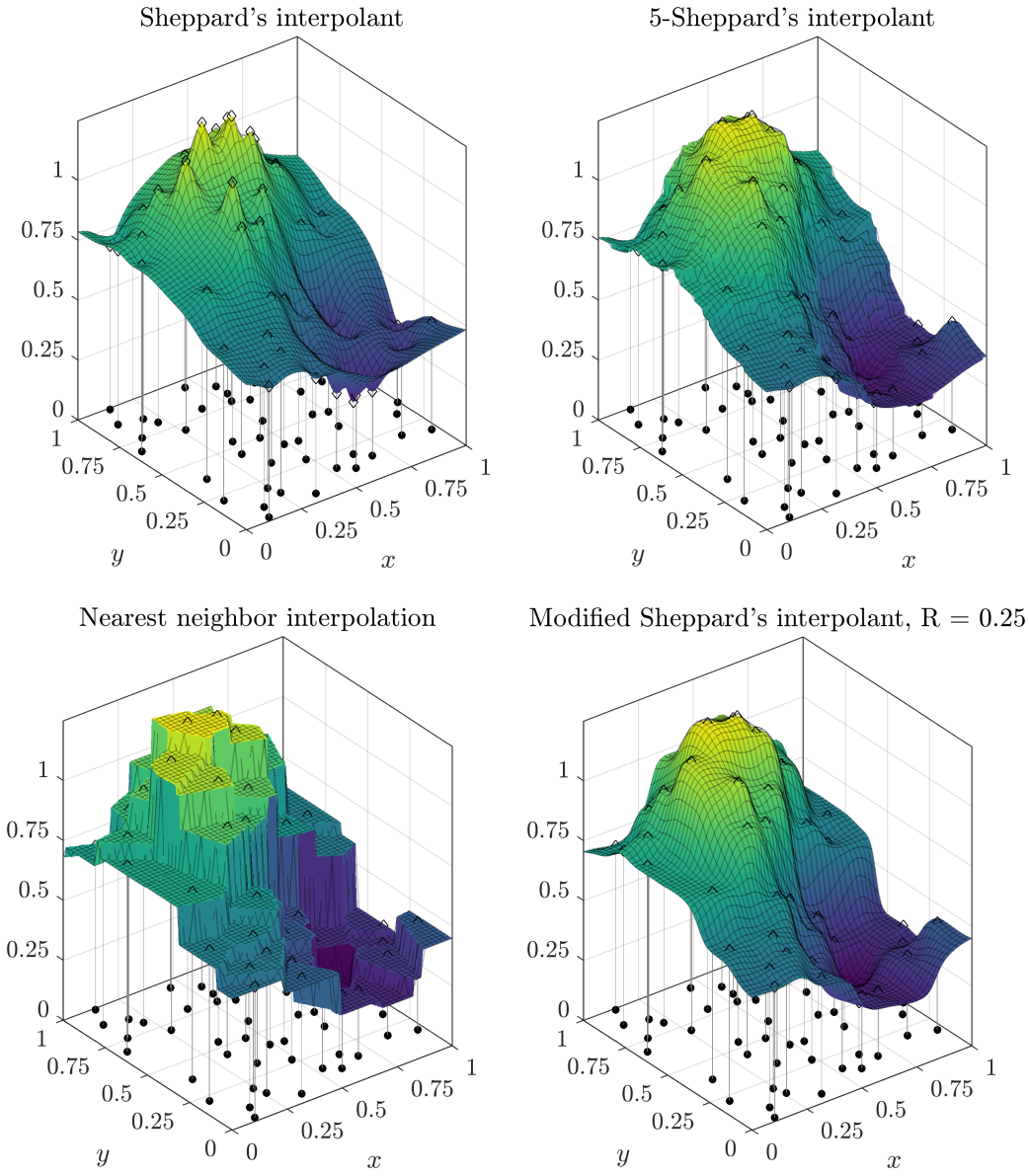


Figure 1.5: Variants of Sheppard's interpolation for $p = 2$ shown on data $(\boldsymbol{x}_i, u_i)$ taken from u_{ex}. The original function u_{ex} is shown in Figure 1.2.

1.3.2 Moving least squares

Moving least squares approximation was developed in 1970s [Lan79] with the main idea to solve a local weighted least squares problem for every evaluation point $\boldsymbol{x}$. Using the same notation as above, the moving least squares approximant $\hat{u}$ is defined at a point $\boldsymbol{x}$ as $\hat{u}(\boldsymbol{x}) = p^*(\boldsymbol{x})$, where p^* is a multivariate polynomial which minimizes the weighted least squares error:

$$p^* = \underset{p \in \mathbb{P}_m(\mathbb{R}^d)}{\arg\min} \sum_{i=1}^{n} (u_i - p(\boldsymbol{x}_i))^2 \omega(\boldsymbol{x}, \boldsymbol{x}_i), \tag{1.3.3}$$

where p is sought in the space of d-variate polynomials of total degree not greater than m. The weight function ω is most often of the form $\omega(\boldsymbol{x}, \boldsymbol{y}) = \omega_\delta(\boldsymbol{x} - \boldsymbol{y})$, where $\omega_\delta(\boldsymbol{x}) =$

$\omega(\boldsymbol{x}/\delta)$ and ω is a compactly supported function $\omega\colon \mathbb{R}^2 \to [0,\infty)$. This ensures that the weight is translation-invariant and only data sites in the ball around $\boldsymbol{x}$ with radius δ influence the approximant value $\hat{u}(\boldsymbol{x})$.

To obtain p^* in practice, we first have to fix a basis $(p_1,\dots,p_\ell)$ of $\mathbb{P}_m(\mathbb{R}^d)$, with $\ell = \binom{m+d}{d}$. Additionally, when constructing the approximant around a point $\boldsymbol{x}$, any data sites $\boldsymbol{x}_i$ for which $\omega(\boldsymbol{x},\boldsymbol{x}_i)=0$ are discarded. Next, we assume that the remaining data sites (still called $X=\{\boldsymbol{x}_1,\dots,\boldsymbol{x}_n\}$) form a $\mathbb{P}_m(\mathbb{R}^d)$-unisolvent set. Writing the sought polynomial p^* as

$$p^* = \sum_{j=1}^{\ell} \alpha_j^* p_j \tag{1.3.4}$$

and substituting it into (1.3.3), we can write the minimization problem in terms of $\boldsymbol{\alpha}^*$ as

$$\boldsymbol{\alpha}^* = \operatorname*{arg\,min}_{\boldsymbol{\alpha}\in\mathbb{R}^\ell} \sum_{i=1}^{n} e_i^2 \omega(\boldsymbol{x},\boldsymbol{x}_i), \quad e_i = u_i - \sum_{j=1}^{\ell} \alpha_j p_j(\boldsymbol{x}_i) \tag{1.3.5}$$

which can be further rewritten into matrix form as

$$\boldsymbol{\alpha}^* = \operatorname*{arg\,min}_{\boldsymbol{\alpha}\in\mathbb{R}^\ell} \|\boldsymbol{e}\|_{\boldsymbol{\omega}}^2, \quad \boldsymbol{e} = P\boldsymbol{\alpha} - \boldsymbol{u}, \tag{1.3.6}$$

where P is a $n\times\ell$ matrix

$$P = \begin{bmatrix} p_1(\boldsymbol{x}_1) & \cdots & p_\ell(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ p_1(\boldsymbol{x}_n) & \cdots & p_\ell(\boldsymbol{x}_n) \end{bmatrix}, \tag{1.3.7}$$

$\boldsymbol{u}$ is the vector of function values and $\boldsymbol{\omega}$ is the vector of weights $\boldsymbol{\omega} = (\omega(\boldsymbol{x},\boldsymbol{x}_i))_{i=1}^n$.

The weighted least squares problem can be rewritten into standard form by introducing a diagonal matrix W with entries $W_{ii} = \sqrt{\omega(\boldsymbol{x},\boldsymbol{x}_i)}$ and finding $\boldsymbol{\alpha}^*$ is reduced to

$$\boldsymbol{\alpha}^* = \operatorname*{arg\,min}_{\boldsymbol{\alpha}\in\mathbb{R}^\ell} \|WP\boldsymbol{\alpha} - W\boldsymbol{u}\|^2. \tag{1.3.8}$$

This system is of full rank, because elements of W are positive and X was assumed to be unisolvent. The overdetermined system can be solved either via normal equations

$$P^\mathsf{T} W^2 P\boldsymbol{\alpha}^* = P^\mathsf{T} W^2 \boldsymbol{u}, \tag{1.3.9}$$

or using QR, SVD or other decompositions to obtain $\boldsymbol{\alpha}^*$. After obtaining $\boldsymbol{\alpha}^*$, the value of $\hat{u}$ can be easily obtained via (1.3.4).

If the coefficients $\boldsymbol{\alpha}$ are computed at a certain point $\boldsymbol{p}$, the function

$$p^*(\boldsymbol{x}) = \sum_{i=1}^{\ell} \alpha_i^*(\boldsymbol{p}) p_i(\boldsymbol{x}) \tag{1.3.10}$$

is the weighted least squares (WLS) approximant around $\boldsymbol{p}$ and only one overdetermined system needs to be solved to compute it.

However, if coefficients $\boldsymbol{\alpha}$ are computed for each evaluation point $\boldsymbol{x}$, the resulting approximant is called a moving least squares (MLS) approximant, and can be thought of as an envelope of all WLS approximants. The resulting function is

$$\hat{u}(\boldsymbol{x}) = \sum_{i=1}^{\ell} \alpha_i(\boldsymbol{x}) p_i(\boldsymbol{x}). \tag{1.3.11}$$

The top row of Figure 1.6 shows WLS and MLS approximants of the function u_{ex}. Monomial order $m = 2$ was used with basis $\{1, x, y, x^2, xy, y^2\}$ and a weight $\omega(\boldsymbol{x}, \boldsymbol{x}_i) = \exp(-\|\boldsymbol{x}-\boldsymbol{x}_i\|^2/\sigma^2)$ with $\sigma = 0.25$. The bottom left plot shows MLS with the compactly supported weight $\omega(\boldsymbol{x}, \boldsymbol{x}_i) = \phi_2(\|\boldsymbol{x} - \boldsymbol{x}_i\|/\delta), \phi_2(s) = (1 - 3s^2 + 2s^3)_+$. The spikes in the plot are from the points $\boldsymbol{x}$ where there were less than 6 neighboring data sites and the least squares system was under-determined. With $\delta = 0.25$ the number of neighbor included in a circle with radius δ varied from 1 to 14. If a larger δ were chosen, the function would be smooth.

The complexity of WLS is $O(\ell^2 n + \ell^3)$ for computation of $\boldsymbol{\alpha}$ and $O(\ell)$ per evaluation. The complexity of the MLS approximation is $O(\ell^2 n + \ell^3)$ *per evaluation*, since we need to solve an over-determined system at each point $\boldsymbol{x}$. The value ℓ denotes the number of basis functions and the value n the number of data sites $\boldsymbol{x}_i$, for which $\omega(\boldsymbol{x}, \boldsymbol{x}_i) > 0$. This means that the cost can be significantly reduced by using such ω that $\omega(\boldsymbol{x}, \cdot)$ is compactly supported. Another option is to always use only k closest nodes, $\ell \leq k \leq n$, and we call this approximation k-MLS. This guarantees a good time complexity of $O(\ell^2 k + \ell^3)$, but, similarly to k-Sheppard's interpolation, there can be discontinuities along the boundaries of Voronoi cells where the set of closest nodes changes. An example of k-MLS approximation with basis $\{1, x, y, x^2, xy, y^2\}$ and a weight $\omega(\boldsymbol{x}, \boldsymbol{x}_i) = \exp(-\|\boldsymbol{x} - \boldsymbol{x}_i\|^2/\sigma^2)$ with $\sigma = 0.25$ is shown in the bottom right corner of Figure 1.6.

Special case when $\ell = 1$

When least squares approximation includes only a constant polynomial, the solution $\boldsymbol{\alpha}^*$ is a scalar and can be found in a closed form. We choose the basis $(p_1 = 1)$, which gives $P = [1, \ldots, 1]$ and using the normal equations (1.3.9) we obtain

$$\hat{u}(\boldsymbol{x}) = \boldsymbol{\alpha}^* p_1 = \frac{\sum_{i=1}^n u_i w(\boldsymbol{x}, \boldsymbol{x}_i)}{\sum_{i=1}^n w(\boldsymbol{x}, \boldsymbol{x}_i)}.$$

This can be recognized as a form similar to Sheppard's interpolation (1.3.1), but with different weights. Such $\hat{u}$ do not in general interpolate values u_i, but can be though of as a moving average of u_i.

Special case when $\ell = n$

If the size of the basis ℓ is the same as the number of data sites n, the least squares problem becomes an interpolation problem with conditions $\hat{u}(\boldsymbol{x}_i) = u_i$. The minimal error norm of 0 is achieved at p^* that is the solution of $P\boldsymbol{\alpha}^* = \boldsymbol{u}$ (note that P is square $n \times n$). In particular, this means that the weights $w(\boldsymbol{x}, \boldsymbol{x}_i)$ are irrelevant and that all dependency of $\boldsymbol{\alpha}^*$ on $\boldsymbol{x}$ is gone.

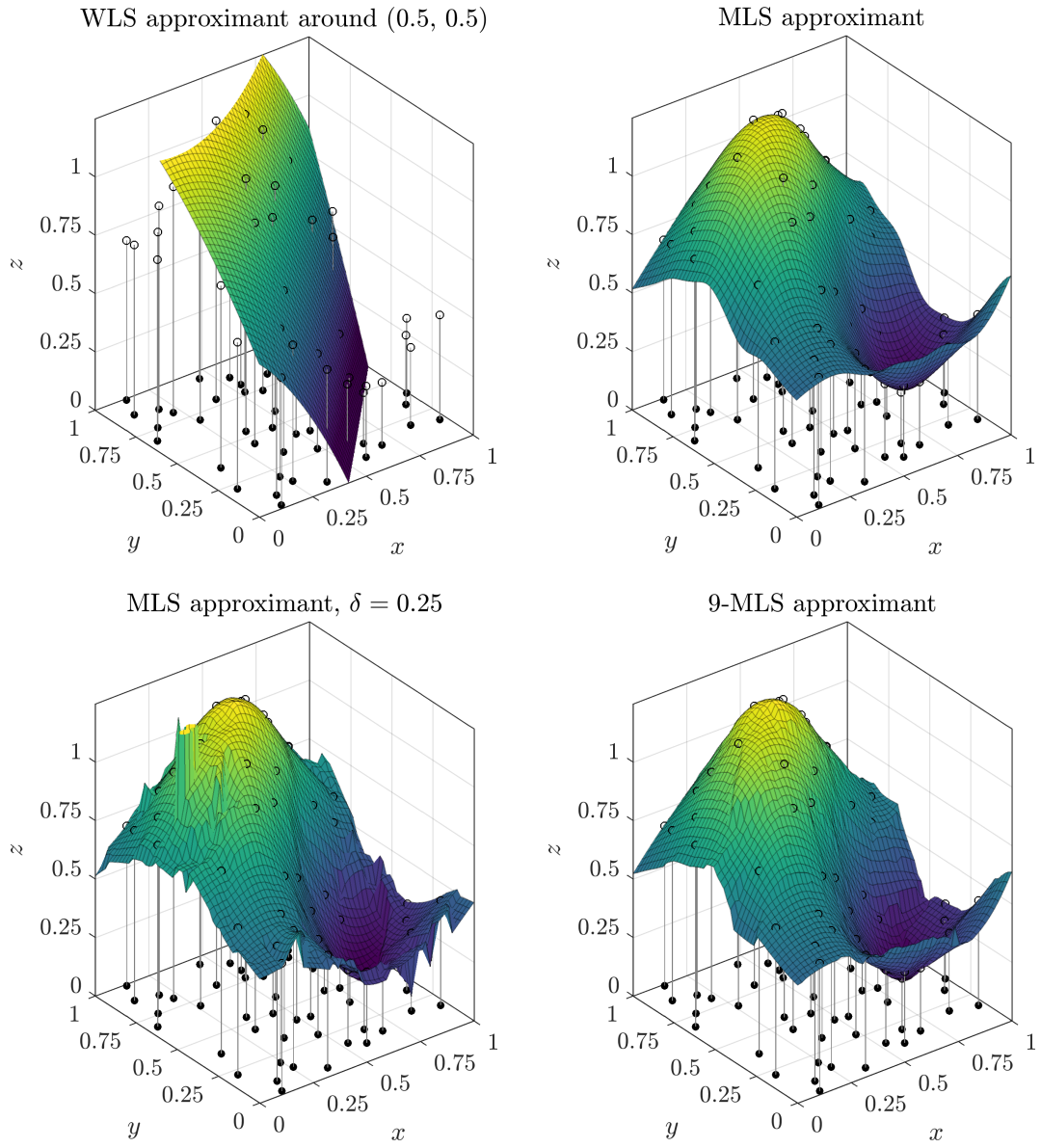


Figure 1.6: Variants of least squares approximation shown on data $(\boldsymbol{x}_i, u_i)$ taken from u_{ex}. The original function u_{ex} is shown in Figure 1.2.

1.3.3 Radial basis functions

Radial basis functions offer a simple way to perform data-dependent interpolation on a set of data sites $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ with corresponding function values f_i. Given a function $\phi\colon [0, \infty) \to \mathbb{R}$, a radial basis function (RBF) centered at $\boldsymbol{c}$ is defined as

$$\phi_{\boldsymbol{c}}\colon \mathbb{R}^d \to \mathbb{R}, \quad \phi_{\boldsymbol{c}}(\boldsymbol{x}) = \phi(\|\boldsymbol{x} - \boldsymbol{c}\|). \tag{1.3.12}$$

The value $\boldsymbol{c}$ is called the *center* of the RBF.

The name *radial* function is justified, since $\phi_{\boldsymbol{c}}$ is radial in the sense of the following definition.

Definition 1.3.1 (Radial function)**.** A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called *radial* if it is of the form $f(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|)$ for some $\phi\colon [0, \infty) \to \mathbb{R}$.

Example 1.3.2. Commonly used radial functions include

- *Gaussian:* $\phi(r) = \exp(-(\varepsilon r)^2)$,
- *Multiquadric:* $\phi(r) = \sqrt{1 + (\varepsilon r)^2)}$,
- *Inverse multiquadric:* $\phi(r) = (1 + (\varepsilon r)^2)^{-\frac{1}{2}}$,
- *Polyharmonic:* $\phi(r) = \begin{cases} r^k, & k \text{ odd} \\ r^k \log r, & k \text{ even} \end{cases}$.

The parameter ε is called a shape parameter and is a positive real number that determines the shape of the RBF.

Definition 1.3.3 (Radial basis functions)**.** Let X be a set of points in $\mathbb{R}^d$ and $\phi\colon [0, \infty) \to \mathbb{R}$ a function. The set of radial functions

$$\{\phi_{\boldsymbol{x}};\ \boldsymbol{x} \in X\} \tag{1.3.13}$$

is called the set of *radial basis functions* on the centers X, generated by ϕ.

Remark 1.3.4. The name *basis* is not always justified, as the functions $\phi_{\boldsymbol{x}_i}$ are not always linearly independent, as shown in the following example. However, the previously listed radial basis function do indeed form a basis and satisfy nonsingularity interpolation theorems, justifying the name.

Example 1.3.5. Radial basis functions in $\mathbb{R}$ generated by $\phi(r) = r^2$ on centers $\{-2, -1, 0, 1, 2\}$ are not linearly independent.

The obtained radial basis functions are $\phi_{-2}(x) = (x+2)^2$, $\phi_{-1}(x) = (x+1)^2$, $\phi_0 = x^2$, $\phi_1(x) = (x-1)^2$ and $\phi_2(x) = (x-2)^2$. These are 5 polynomials of 2nd degree and cannot be linearly independent, for example $\phi_{-2} = -3\phi_0 + 3\phi_{-1} + \phi_1$ holds.

An RBF interpolant for nodes X is of the form

$$\hat{u}(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) \tag{1.3.14}$$

and the interpolation conditions $\hat{u}(\boldsymbol{x}_i) = f_i$ form a system of linear equations

$$\begin{bmatrix} \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_n\|) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \tag{1.3.15}$$

compactly written as $A_\phi \boldsymbol{\alpha} = \boldsymbol{f}$. This can be recognized as a special form of a kernel interpolant with a radial kernel $\mathcal{K}_\phi(\boldsymbol{x}, \boldsymbol{y}) = \phi(\|\boldsymbol{x} - \boldsymbol{y}\|)$. Figure 1.7 shows interpolation of u_{ex} with Gaussian and inverse multiquadric RBFs.

Naturally, we are interested in the well-posedness of such interpolation problems in general, which translates to (non)singularity of the interpolation matrix A_ϕ. We will in fact be able to prove that for many functions ϕ, the matrix A_ϕ even is positive definite. In fact, most of the remainder of this section is dedicated to the task of proving (conditional)

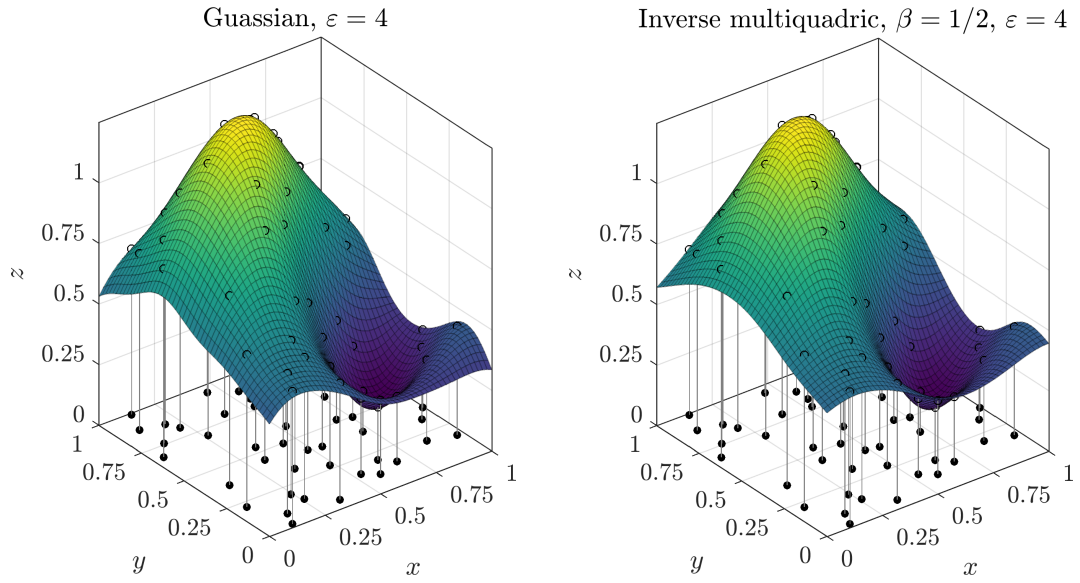


Figure 1.7: RBF interpolant on data $(\boldsymbol{x}_i, u_i)$ using Gaussians (left) and inverse multiquadrics (right).

positive definiteness of the 4 RBF types listed in examples 1.3.2 as well as a famous characterization of positive definite functions, so that proving positiveness on all $\mathbb{R}^d$ is reduced to a simple derivative check.

To this end, we first define the notion of a positive definite function.

Definition 1.3.6 (Positive definite function)**.** A function $\Phi\colon \mathbb{R}^d \to \mathbb{C}$ is positive semi-definite iff the kernel $\mathcal{K}_\Phi(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x} - \boldsymbol{y})$ is positive semi-definite, i.e. the quadratic form

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \overline{\alpha_j} \Phi(\boldsymbol{x}_i - \boldsymbol{x}_j) \tag{1.3.16}$$

is nonnegative for any $n \in \mathbb{N}$, any n distinct points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^d$ and any coefficients $(\alpha_1, \ldots, \alpha_n) \in \mathbb{C}^n$. If the form is equal to 0 only if all α_i are 0, then the function is called positive definite.

Definition 1.3.7 (Positive definite radial function)**.** A univariate function $\phi\colon [0, \infty) \to \mathbb{R}$ is positive (semi-)definite on $\mathbb{R}^d$ if the function $\Phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|)$ is positive (semi-)definite.

To derive the aforementioned derivative check, we will first introduce the canonical example of a positive definite function, the exponential. This will allow us to connect positive definite functions to Fourier transforms.

Example 1.3.8 (Positive semi-definitness of exponentials)**.** The function

$$\Phi(\boldsymbol{x}) = \exp(i\boldsymbol{x}^\mathsf{T}\boldsymbol{y}) \tag{1.3.17}$$

is positive semi-definite for any fixed $\boldsymbol{y} \in \mathbb{R}^d$ for any d. Indeed, if we observe the

condition for positive semi-definitness and compute

$$\sum_{j=1}^{n}\sum_{k=1}^{n}\alpha_j\overline{\alpha_k}\exp(i(\boldsymbol{x}_j-\boldsymbol{x}_k)^{\mathsf{T}}\boldsymbol{y}) = \left(\sum_{j=1}^{n}\alpha_j e^{i\boldsymbol{x}_j^{\mathsf{T}}\boldsymbol{y}}\right)\left(\sum_{k=1}^{n}\overline{\alpha_k}e^{-i\boldsymbol{x}_k^{\mathsf{T}}\boldsymbol{y}}\right) \tag{1.3.18}$$

$$= \left|\sum_{j=1}^{n}\alpha_j e^{i\boldsymbol{x}_j^{\mathsf{T}}\boldsymbol{y}}\right|^2 \geq 0, \tag{1.3.19}$$

we can see that it is always nonnegative.

What follows is the famous characterization of positive semi-definite functions, originally due to Bochner [Boc33] and described in modern measure-theoretic language in [Wen04, p. 70]. Positive semi-definite functions are characterized somewhat analogously to Example 1.3.8 as Fourier transforms of finite nonnegative Borel measures.

Theorem 1.3.9 (Bochner)**.** *A continuous function $\Phi\colon \mathbb{R}^d \to \mathbb{C}$ is positive semi-definite iff it is the Fourier transform of a finite nonnegative Borel measure μ on $\mathbb{R}^d$, i.e.*

$$\forall \boldsymbol{x}\in\mathbb{R}^d: \quad \Phi(\boldsymbol{x}) = \widehat{\mu}(\boldsymbol{x}) = (2\pi)^{-d/2}\int_{\mathbb{R}^d} e^{-i\boldsymbol{x}^{\mathsf{T}}\boldsymbol{\omega}}d\mu(\boldsymbol{\omega}). \tag{1.3.20}$$

Proof. We will prove only the easy direction, i.e. that Fourier transforms of nonnegative finite Borel measures are positive definite, since that will suffice to prove that many important RBFs are positive definite.

As in the Example 1.3.8 we choose distinct $\boldsymbol{x}_j \in \mathbb{R}^d$ and $\alpha_j \in \mathbb{C}$ and compute

$$\sum_{j=1}^{n}\sum_{k=1}^{n}\alpha_j\overline{\alpha_k}\Phi(\boldsymbol{x}_j-\boldsymbol{x}_k) = \sum_{j=1}^{n}\sum_{k=1}^{n}\alpha_j\overline{\alpha_k}(2\pi)^{-d/2}\int_{\mathbb{R}^d} e^{i(\boldsymbol{x}_j-\boldsymbol{x}_k)^{\mathsf{T}}\boldsymbol{\omega}}d\mu(\boldsymbol{\omega}) \tag{1.3.21}$$

$$= (2\pi)^{-d/2}\int_{\mathbb{R}^d}\left(\sum_{j=1}^{n}\alpha_j e^{-i\boldsymbol{x}_j^{\mathsf{T}}\boldsymbol{\omega}}\right)\left(\sum_{k=1}^{n}\overline{\alpha_k}e^{i\boldsymbol{x}_k^{\mathsf{T}}\boldsymbol{\omega}}\right)d\mu(\boldsymbol{\omega}) =$$

$$= (2\pi)^{-d/2}\int_{\mathbb{R}^d}\left|\sum_{j=1}^{n}\alpha_j e^{-i\boldsymbol{x}_j^{\mathsf{T}}\boldsymbol{\omega}}\right|^2 d\mu(\boldsymbol{\omega}) \geq 0.$$

The result is nonnegative due to integrating a nonnegative function with respect to a nonnegative measure. □

The assumptions of Theorem 1.3.9 can be strengthened slightly to obtain a sufficient condition for positive definiteness.

Theorem 1.3.10. *If the carrier of a finite nonnegative Borel measure μ contains an open subset, then its Fourier transform is positive definite.*

Proof. We only present the main ideas of the proof. Let U be an open subset, so that $\mu(U) > 0$. Then, for distinct $\boldsymbol{x}_j \in U$, we have from the final step in computation (1.3.21) that

$$\sum_{j=1}^{n}\alpha_j e^{-i\boldsymbol{x}_j^{\mathsf{T}}\boldsymbol{\omega}} = 0 \tag{1.3.22}$$

must hold for all $\boldsymbol{\omega} \in U$. By the identity theorem from complex analysis, this can be extended to all $\boldsymbol{\omega} \in \mathbb{R}^d$, where we can use the linear independence of exponentials. □

Corollary 1.3.11. *Let $f \in L_1(\mathbb{R}^d)$ be continuous, nonvanishing and nonnegative. Then*

$$\Phi(\boldsymbol{x}) = \int_{\mathbb{R}^d} f(\boldsymbol{\omega}) e^{-i\boldsymbol{x}^\mathsf{T}\boldsymbol{\omega}} d\boldsymbol{\omega} \tag{1.3.23}$$

is positive definite.

Proof. Define the measure μ as $\mu(B) = \int_B f(\boldsymbol{x})d\boldsymbol{x}$. The conditions on f imply that μ is a finite nonnegative Borel measure, with carrier of μ equal to support of f, which must contain an interior point. Thus, its Fourier transform is positive definite by Theorem 1.3.10. □

Equipped with the last corollary, we can prove that Gaussians are positive definite on any $\mathbb{R}^d$.

Theorem 1.3.12 (Gaussians are positive definite)**.** *Gaussian RBF $\phi(r) = \exp(-(\varepsilon r)^2)$ is positive definite on $\mathbb{R}^d$ for all d.*

Proof. The function $\Phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|_2) = \exp(-(\varepsilon\|\boldsymbol{x}\|_2)^2)$ can be written as

$$\Phi(\boldsymbol{x}) = (2\varepsilon)^{-d}\pi^{-\frac{d}{2}} \int_{\mathbb{R}^d} e^{-\|\boldsymbol{\omega}\|_2^2/(4\varepsilon^2)} e^{-i\boldsymbol{x}^\mathsf{T}\boldsymbol{\omega}} d\boldsymbol{\omega}. \tag{1.3.24}$$

Since $\boldsymbol{\omega} \to e^{-\|\boldsymbol{\omega}\|_2^2/(4\varepsilon^2)}$ is in $L_1(\mathbb{R}^d)$, continuous, nonvanishing and nonnegative, ϕ is positive definite by corollary 1.3.11. □

This result will be useful to obtain the famous characterization of radial basis functions that are positive definite for every dimension d in terms of completely monotone functions.

Definition 1.3.13 (Completely monotone function)**.** A function $\psi \in C((0,\infty))$ is called completely monotone on $(0,\infty)$ if

$$(-1)^\ell \psi^{(\ell)}(r) \geq 0 \tag{1.3.25}$$

for all $r > 0$ and all $\ell \in \mathbb{N}$. Function ϕ is additionally called completely monotone on $[0,\infty)$ if it is in $C([0,\infty))$.

We will borrow the following fact from analysis about completely monotone functions.

Theorem 1.3.14 (Haussdorff-Bernstein-Widder)**.** *A function $\psi\colon [0,\infty) \to \mathbb{R}$ is completely monotone on $[0,\infty)$ if and only if it is the Laplace transform of a nonnegative finite Borel measure ν, i.e.*

$$\psi(r) = \int_0^\infty e^{-rt} d\nu(t). \tag{1.3.26}$$

The fact that functions ψ of the form (1.3.26) are indeed completely monotone follows directly from successive differentiation. For the proof in the other direction, see for Example [Wen04, p. 91].

Remark 1.3.15. The assumption on finitness of ν is necessary to get continuity of ψ at 0. Monotonicity on $(0, \infty)$ is guaranteed by only having a nonnegative Borel ν on $[0, \infty)$.

The Haussdorff-Bernstein-Widder theorem allows us to connect completely monotone functions to positive semi-definite radial basis functions, using the following characterization by Schoenberg.

Theorem 1.3.16 (Shoenberg)**.** *A function ψ is completely monotone on $[0, \infty)$ if and only if ϕ, given by $\phi(r) = \psi(r^2)$, is positive semi-definite on $\mathbb{R}^d$ for all d.*

Proof. Once again, we shall prove only the simple direction, which will allow for checking that given functions are positive definite. If ψ is completely monotone, then by Haussdorff-Bernstein-Widder theorem it allows the representation

$$\psi(r) = \int_0^\infty e^{-rt} d\nu(t) \tag{1.3.27}$$

for a nonnegative finite Borel measure ν. We can now check that ϕ is positive semi-definite on $\mathbb{R}^d$ by choosing arbitrary $(\boldsymbol{x}_j)_{j=1}^n$ and $\boldsymbol{\alpha} \in \mathbb{C}$ and computing

$$\sum_{j=1}^n \sum_{k=1}^n \alpha_j \overline{\alpha_k} \phi(\|\boldsymbol{x}_j - \boldsymbol{x}_k\|) = \sum_{j=1}^n \sum_{k=1}^n \alpha_j \overline{\alpha_k} \int_0^\infty e^{-t\|\boldsymbol{x}_j - \boldsymbol{x}_k\|^2} d\nu(t) = \tag{1.3.28}$$

$$= \int_0^\infty \sum_{j=1}^n \sum_{k=1}^n \alpha_j \overline{\alpha_k} e^{-t\|\boldsymbol{x}_j - \boldsymbol{x}_k\|^2} d\nu(t) \geq 0, \tag{1.3.29}$$

The final integral is nonnegative since ν is nonnegative and because we have already proven in Theorem 1.3.12 that Gaussians are positive definite. □

We will now rewrite the previous theorem from the point of view where we would want to check ϕ for positive definitness and amend it with characterization conditions for that.

Theorem 1.3.17 (Characterization of positive definitness for all $\mathbb{R}^d$)**.** *The following statements are equivalent for $\phi : [0, \infty) \to \mathbb{R}$:*

(i) function ϕ is positive definite on every $\mathbb{R}^d$

(ii) function ψ, given by $\psi(r) = \phi(\sqrt{r})$, is completely monotone on $[0, \infty)$ and not constant

(iii) ϕ has the form

$$\phi(r) = \int_0^\infty e^{-tr^2} d\nu(t) \tag{1.3.30}$$

for a finite nonnegative Borel measure ν that is not concentrated as zero.

Proof. Equivalence for semi-definitness follows from Shoenberg's theorem. To test for positive definitness, we again observe the expression

$$\int_0^\infty \sum_{j=1}^n \sum_{k=1}^n \alpha_j \overline{\alpha_k} e^{-t\|\boldsymbol{x}_j - \boldsymbol{x}_k\|^2} d\nu(t), \tag{1.3.31}$$

for $\boldsymbol{\alpha} \neq 0$. Positive definitness of Gaussians implies strict positivity of the integrand, as long as all $\boldsymbol{x}_i$ are distinct and $t > 0$. The value of $t = 0$ is not a problem, unless ν is concentrated at zero, in which case the sum and the integral can be zero.

The property that ν is concentrated at zero (i.e. $\nu = c\,\delta_0$, where δ_0 is the Dirac measure at zero) is in fact equivalent to the fact that ϕ is constant, as in that case

$$\phi(r) = \int_0^\infty e^{-tr^2} d(c\delta_0) = c. \tag{1.3.32}$$

□

Property (ii) is extremely useful to check positive definitness for a desired function ϕ, such as in the following corollary.

Corollary 1.3.18 (Inverse multiquadrics are positive definite)**.** *The inverse multiquadrics*

$$\phi(r) = (1 + (\varepsilon r)^2)^{-\beta} \tag{1.3.33}$$

are positive definite for $\beta > 0$ and $\varepsilon > 0$ on any $\mathbb{R}^d$.

Proof. We define $\psi(r) = (1 + \varepsilon^2 r)^{-\beta}$. It is not constant and it is completely monotone, since

$$(-1)^\ell \psi^{(\ell)}(r) = (-1)^\ell \varepsilon^{2\ell} (1 + \varepsilon^2 r)^{-\beta-\ell} (-\beta)(-\beta - 1) \cdots (-\beta - \ell + 1) \geq 0. \tag{1.3.34}$$

Additionally it can be verified, that the measure realizing the representation (1.3.27) is given by $d\nu(t) = \frac{\varepsilon^{-2\beta}}{\Gamma(\beta)} t^{\beta-1} e^{-t/\varepsilon^2} dt$. □

Positive definitness of other radial basis functions can be verified similarly, for example Laguerre-Gaussian, Poisson and Matérn functions [Fas07, p. 40].

Is is also important to note that Theorem 1.3.17 only characterizes RBFs that are positive definite for all $\mathbb{R}^d$, and quite some time passed before examples of RBFs that are positive definite only on R^d for $d \leq d_0$ were discovered. Nowadays, compactly supported RBFs are of interest, especially when large clouds of points are considered, since they yield a sparse interpolation matrix A_ϕ. One technique for generating a compactly supported positive definite function ϕ from a continuous compactly supported nonzero $\psi : [0, \infty) \to \mathbb{R}$ is by using convolution to define

$$\phi(\|\boldsymbol{x}\|) = \int_{\mathbb{R}^d} \psi(\|\boldsymbol{y}\|)\, \psi(\|\boldsymbol{x} - \boldsymbol{y}\|)\, d\boldsymbol{y}. \tag{1.3.35}$$

Positive definitness follows from

$$\sum_{j=1}^n \sum_{k=1}^n \alpha_j \overline{\alpha_k} \phi(\|\boldsymbol{x}_j - \boldsymbol{x}_k\|) = \int_{\mathbb{R}^d} \left| \sum_{j=1}^n \alpha_j \psi(\|\boldsymbol{x}_j - \boldsymbol{y}\|) \right|^2 d\boldsymbol{y} \tag{1.3.36}$$

and linear independence of translations of ψ, which is compactly supported and continuous. Schaback [Sch95a] presents this idea in more detail and Wendland [Wen04, chapter 9] proved nice overview of compactly supported RBFs in general.

The cost of computing the RBF interpolant is $O(n^3)$, with $O(n)$ for each evaluation. If compactly supported RBFs are used, this cost can be reduced to the number of nodes present in the support.

Augmentation with monomials

Some popular choices of radial basis functions, such as polyharmonic RBFs are not positive definite. Furthermore, RBF interpolants do not in general reproduce polynomials. However, polynomial reproduction is often desirable, and the RBF interpolant (1.3.14) can be augmented with monomials by introducing it in the form

$$\hat{u}(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) + \sum_{\ell=1}^{q} \beta_\ell p_\ell(\boldsymbol{x}), \tag{1.3.37}$$

where p_ℓ are polynomials that form a basis of $\mathbb{P}_m(\mathbb{R}^d)$, the space of polynomials of (total) degree lower or equal to m. This means that q must be equal to the dimension of the space of polynomials, i.e. $q = \dim \mathbb{P}_m(\mathbb{R}^d) = \binom{m+d}{m}$. Such an interpolant will necessarily reproduce monomials up to the order m.

Interpolation conditions $\hat{u}(\boldsymbol{x}_i) = f_i$ give n equations for $n + q$ variables, and q additional constraints are necessary to compensate for the added degrees of freedom. It is advantageous to add the conditions of the form

$$\sum_{j=1}^{n} \alpha_j p_\ell(\boldsymbol{x}_j) = 0, \quad \forall \ell = 1, \ldots, q, \tag{1.3.38}$$

which yield the $(n + q) \times (n + q)$ interpolation system

$$\begin{bmatrix} A_\phi & P \\ P^\mathsf{T} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix}, \tag{1.3.39}$$

where P is the $n \times q$ matrix of evaluated polynomials

$$P = \begin{bmatrix} p_1(\boldsymbol{x}_1) & \cdots & p_q(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ p_1(\boldsymbol{x}_n) & \cdots & p_q(\boldsymbol{x}_n) \end{bmatrix}. \tag{1.3.40}$$

The added conditions retain the symmetry of the matrix and will prove useful in further analysis. Similarly as before, we will be able to prove that the system is uniquely solvable for some important classes of RBFs. An example of an interpolant obtained this way is shown in Figure 1.8.

First, let us define the key property of that will ensure solvability of the interpolation system.

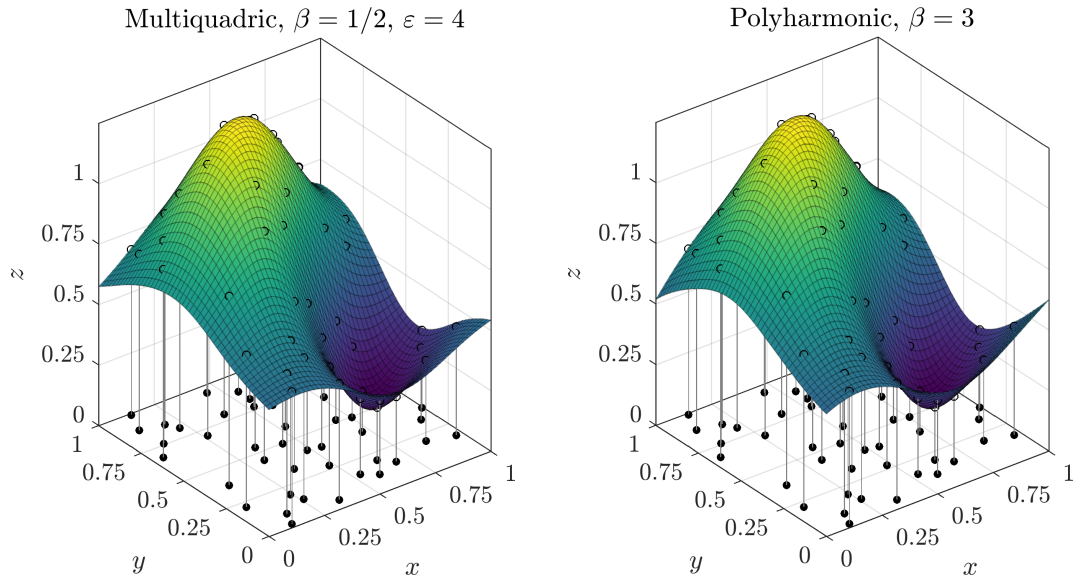


Figure 1.8: RBF interpolant on data $(\boldsymbol{x}_i, u_i)$ using multiquadrics (left) and polyharmonics (right). The interpolation was augmented with monomials of order 0 and 1, respectively.

Definition 1.3.19 (Conditionally positive definite function)**.** A continuous function $\Phi\colon \mathbb{R}^d \to \mathbb{C}$ is positive semi-definite of order $m+1$ iff its associated quadratic form is nonnegative, i.e.

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \overline{\alpha_j} \Phi(\boldsymbol{x}_i - \boldsymbol{x}_j) \geq 0 \tag{1.3.41}$$

for any $n \in \mathbb{N}$, any n distinct points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^d$ and any coefficients $(\alpha_1, \ldots, \alpha_n) \in \mathbb{C}^n$, which satisfy

$$\sum_{j=1}^{n} \alpha_j p(\boldsymbol{x}_j) = 0 \tag{1.3.42}$$

for all complex polynomials of degree not greater than m, i.e. $p \in \mathbb{P}_m(\mathbb{C}^d)$. If the form is equal to 0 only if all α_i are 0, then the function is called conditionally positive definite.

We are often interested in minimal order of a conditionally positive definite function ϕ, since any conditionally positive (semi-)definite function of order m is also of order $\ell \geq m$, due to the fact that the space of admissible $\boldsymbol{\alpha}$ narrows down with larger ℓ, since more conditions are added. Conditional positive (semi-)definiteness of order zero is equivalent to ordinary positive (semi-)definitness.

The condition in Definition 1.3.19 is equivalent to the fact that the matrix A_ϕ is positive definite on the space of vectors α satisfying

$$\sum_{j=1}^{n} \alpha_j p_\ell(\boldsymbol{x}_j) = 0 \tag{1.3.43}$$

for a selected basis p_ℓ of $p \in \mathbb{P}_m(\mathbb{C}^d)$.

Definition 1.3.19 is also extended to radial functions.

Definition 1.3.20 (Conditionally positive definite radial function)**.** A univariate function $\phi\colon [0,\infty) \to \mathbb{R}$ is conditionally positive (semi-)definite of order m on $\mathbb{R}^d$ if the function $\Phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|)$ is conditionally positive (semi-)definite of order m.

We first show that Definition 1.3.19 of conditional positive definitness is good, in the sense that the interpolation system is solvable for conditionally positive definite Φ, and uniquely solvable if unisolvency of X is assumed.

Theorem 1.3.21 (Well-posedness of RBF interpolation augmented with monomials)**.** *Let Φ be conditionally positive definite of order $m+1$ and X be a $\mathbb{P}_m(\mathbb{R}^d)$-unisolvent set of centers. Then the interpolation system of RBFs augmented with monomials $(n+q)\times(n+q)$ interpolation system*

$$\begin{bmatrix} A_\phi & P \\ P^\mathsf{T} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \end{bmatrix}, \tag{1.3.44}$$

is uniquely solvable. Furthermore, if X is not assumed to be unisolvent, the system is still solvable.

Proof. For unique solvability, we will prove that the matrix is invertible. Given the homogeneous equations of the interpolation system

$$A_\phi \boldsymbol{\alpha} + P\boldsymbol{\beta} = \mathbf{0} \tag{1.3.45}$$

$$P^\mathsf{T}\boldsymbol{\alpha} = \mathbf{0}. \tag{1.3.46}$$

we wish to prove that $\boldsymbol{\alpha} = \mathbf{0}$ and $\beta = \mathbf{0}$ is the only solution. The latter of these equations gives exactly the conditions on $\boldsymbol{\alpha}$ used in conditional positive definitness. Multiplying the first equation by $\boldsymbol{\alpha}^\mathsf{T}$ from the left gives $\boldsymbol{\alpha}^\mathsf{T} A_\phi \boldsymbol{\alpha} = \mathbf{0}$, since $\boldsymbol{\alpha}^\mathsf{T} P\boldsymbol{\beta}$ is zero, because of the second equation (1.3.46). The conditional positive definitness is exactly the condition that ensures $\boldsymbol{\alpha} = 0$. What remains of the first equation is $P\boldsymbol{\beta} = 0$, which can be used to deduce $\boldsymbol{\beta} = 0$ due to unisolvency of the node set X.

If unisolvency of X is not assumed, the system is still solvable, albeit not uniquely. To see this, we must se that the span of the left hand side of (1.3.45) is the whole space $\mathbb{R}^n$, i.e. so that for any $\boldsymbol{f}$ we can find at least one such pair of $\boldsymbol{\alpha}$ and β, that $A_\phi\boldsymbol{\alpha} + P\boldsymbol{\beta} = \boldsymbol{f}$. Formally, we wish to prove that $A(\ker(P^\mathsf{T})) + \operatorname{im} P = \mathbb{R}^n$. First, we note that this sum is direct: if $\boldsymbol{x} \in A(\ker(P^\mathsf{T})) \cap \operatorname{im} P$, then $\boldsymbol{x}$ can be written as $\boldsymbol{x} = A_\phi\boldsymbol{\alpha} = P\boldsymbol{\beta}$ for some $\boldsymbol{\alpha} \in \ker(P^\mathsf{T})$ and $\boldsymbol{\beta} \in \mathbb{R}^q$. This implies that

$$\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{x} = \boldsymbol{\alpha}^\mathsf{T} A_\phi \boldsymbol{\alpha} = \boldsymbol{\alpha}^\mathsf{T} P\boldsymbol{\beta} = (P^\mathsf{T}\boldsymbol{\alpha})^\mathsf{T}\boldsymbol{\beta} = \mathbf{0}^\mathsf{T}\boldsymbol{\beta} = 0, \tag{1.3.47}$$

since $\boldsymbol{\alpha} \in \ker(P^\mathsf{T})$. Because A_ϕ is positive definite on $\ker(P^\mathsf{T})$, we can deduce that $\boldsymbol{\alpha} = 0$ from $\boldsymbol{\alpha}^\mathsf{T} A_\phi \boldsymbol{\alpha} = 0$. Consequently, $\boldsymbol{x} = A\alpha = \mathbf{0}$ and the sum is indeed direct. We can now use a simple dimension argument to deduce solvability. We can estimate

$$n \geq \dim\left[A(\ker(P^\mathsf{T})) + \operatorname{im} P\right] = \dim\left[A(\ker(P^\mathsf{T})) \oplus \operatorname{im} P\right] \tag{1.3.48}$$

$$= \dim A(\ker(P^\mathsf{T})) + \dim \operatorname{im} P = \dim \ker(P^\mathsf{T}) + \dim \operatorname{im} P = \tag{1.3.49}$$

$$= \dim\left[\operatorname{im}(P)^\perp\right] + \dim \operatorname{im} P = n, \tag{1.3.50}$$

where we used the fact that A_ϕ restricted to $\ker(P^\mathsf{T})$ is bijective, that $\ker(P^\mathsf{T}) = \operatorname{im}(P)^\perp$ and that sum of orthogonal complements is direct. □

The method of interpolation can be extended to other linearly independent functions other than monomials, with a similar proof.

To find examples of conditionally positive definite functions (other than positive definite functions), we overview the basic theoretical foundations. Many results analogous to those obtained for positive definite functions can be proven for conditionally positive definite functions. This includes a theorem analogous to Bochner's characterization in terms of generalized Fourier transforms [Wen04, chapter 8.2] and a theorem analogous to Schoenberg's characterization in terms of completely monotone functions. The latter is named after Micchelli and provides a very useful tool for checking conditional positive definitness.

Theorem 1.3.22 (Micchelli)**.** *Let $\psi \in C[0,\infty) \cap C^\infty(0,\infty)$ be given. The function ϕ, given by $\phi(r) = \psi(r^2)$, is conditionally positive semi-definite of order $m \geq 0$ on every $\mathbb{R}^d$ if and only if $(-1)^m \psi^{(m)}$ is completely monotone on $(0,\infty)$.*

The proof of the theorem reduces to the representation theorem for completely monotone functions and the fact that Gaussians are positive definite. Full details are given in e.g. [Wen04, p. 114]. It also includes the following test for conditional positive definitness, which is a generalization of Theorem 1.3.17.

Theorem 1.3.23 (Test for conditional positive definitness for all $\mathbb{R}^d$)**.** *Suppose that $\psi \in C[0,\infty) \cap C^\infty(0,\infty)$ is not a polynomial of degree m. Then ϕ, given by $\phi(r) = \psi(r^2)$ is conditionally positive definite of order m on every $\mathbb{R}^d$.*

Using this, we can easily check two major families.

Proposition 1.3.24 (Multiquadrics are conditionally positive definite)**.** *The multiquadrics $\phi(r) = (-1)^{\lceil \beta \rceil}(1+(\varepsilon r)^2)^\beta$, for $\beta > 0$, $\beta \notin \mathbb{N}$, $\varepsilon > 0$ are conditionally positive definite of order $m = \lceil \beta \rceil$.*

Proof. We define $\psi(r) = \phi(\sqrt{r}) = (-1)^{\lceil \beta \rceil}(1+\varepsilon^2 r)^\beta$ and compute

$$\psi^{(k)}(r) = (-1)^{\lceil \beta \rceil}\beta(\beta-1)\cdots(\beta-k+1)\varepsilon^{2k}(1+\varepsilon^2 r)^{\beta-k}. \tag{1.3.51}$$

The choice $m = \lceil \beta \rceil$ is the smallest one such that all subsequent derivatives alternate in signs, giving a completely monotone function. □

Specifically, that means that famous Hardy's multiquadrics ($\beta = 1/2$) [Har71] are conditionally positive definite of order 1, and should be augmented with at least a constant.

Proposition 1.3.25 (Polyharmonics are conditionally positive definite)**.** *The polyharmonics $\phi(r) = (-1)^{\lceil \beta/2 \rceil} r^\beta$, for $\beta > 0$, $\beta \notin 2\mathbb{N}$, are conditionally positive definite of order $m = \lceil \beta/2 \rceil$.*

Proof. Once again we define $\psi(r) = (-1)^{\lceil \beta/2 \rceil} r^{\beta/2}$ and compute

$$\psi^{(k)}(r) = (-1)^{\lceil \beta/2 \rceil}\frac{\beta}{2}\left(\frac{\beta}{2}-1\right)\cdots\left(\frac{\beta}{2}-k+1\right) r^{\beta/2-k}. \tag{1.3.52}$$

and the same argument about changing signs applies as before, giving the order $m = \lceil \beta/2 \rceil$. □

In particular, for $\phi(r) = r^{2k+1}$ the order is $k+1$, meaning that monomials of at least order k should be added to interpolation.

The shape parameter ε

Many of the basis function described feature a shape parameter $\varepsilon > 0$, which controls the flatness of the basis functions. The value of ε influences the accuracy of the interpolant and the condition number of the interpolation matrix. Decreasing ε often leads to higher accuracy, but also high condition number of the interpolation matrix, and a lot of research was dedicated to the analysis of the effect of the shape parameter and choosing the optimal one [FZ07a; FZ07b; BMK12].

Interestingly, even though the matrices A_ϕ often turn singular as $\varepsilon \to 0$, it was shown (using Taylor expansions of RBFs) that the limit $\varepsilon \to 0$ leads to polynomial interpolation for many common RBF types, including Gaussians, multiquadrics and inverse multiquadics [LF05]. This lead to other algorithms for stably computing RBF interpolants for small ε, such as Contour-Padé integration [FW04], or RBF-QR algorithms [FP08].

1.4 Error estimates and condition numbers for kernel based interpolation

For one-dimensional interpolation of scattered data (i.e. non-uniformly spaced points), the error estimates are often specified as $h := \max_{i=1,\dots,n} \Delta x_i = \max_{i=1,\dots,n}(x_i - x_{i-1})$ tends towards zero. Estimates of errors of scattered data interpolants will take a similar form, but, we first need to define the appropriate generalization for quantities related to nodal spacing.

1.4.1 Basic quality measures for a node set

In numerical methods, where subdivisions T of a domain Ω are used for discretization, such as the finite element method, the properties of elements, such as volumes, diameters and aspect ratios are used to define the quality of T and are used in convergence and stability theorems. It is commonly assumed in convergence theorems that the sequences of meshes are *quasi uniform*, which represents the idea that all elements of the subdivision are of comparable size and the difference in element sizes stays approximately constant [BS07]. Similar notions are used in meshless methods, with the additional benefit that there are no conditions on the shape of the elements but only on the distances between nodes. Some terminology also needs to be adapted to the meshless setting, as expressions such as “mesh size”, or “mesh ratio” are no longer appropriate. We will use the following definitions throughout this work, following [Wen04; HSW12].

Definition 1.4.1 (Fill distance)**.** Let $\Omega \subseteq \mathbb{R}^d$ be bounded and $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subseteq \Omega$ a set of points. The fill distance $h_{X,\Omega}$ of X in Ω is defined to be

$$h_{X,\Omega} = 2 \sup_{\boldsymbol{x} \in \Omega} \min_{j=1,\dots,n} \|\boldsymbol{x} - \boldsymbol{x}_j\|. \tag{1.4.1}$$

The definition is well formed, because the supremum exists due to boundedness of Ω. The half of fill distance is also called *fill radius*, *covering radius*, or *mesh norm of* X. It measures how well data sites in X cover Ω in the sense that the largest ball with a center in Ω that contains no data site, can have the diameter at most $h_{X,\Omega}$.

The complementing property to fill distance is the separation distance of a set of data sites.

Definition 1.4.2 (Separation distance)**.** Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^d$ be a set of points. The separation distance s_X of X is defined to be

$$s_X = \min_{1 \leq i < j \leq n} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|. \tag{1.4.2}$$

The separation distance is simply the distance between two data sites that are closest together. Note that we implicitly assumed $n \geq 2$, which will remain our assumption when discussing quantities related to separation distances.

Very loosely, one can think of the fill distance as a measure of how far and how close apart neighboring data sites can be. Illustration of a node set on a domain with marked separation and fill distances is shown in Figure 1.9.

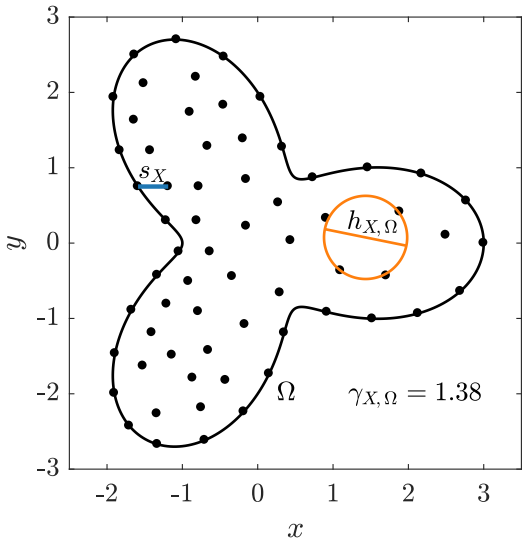


Figure 1.9: Fill and separation distance of a node set.

Different authors use different multiplicative constants for definitions of fill and separation distances [Wen04; HSW12]. These differences are irrelevant in light of the following definitions.

Definition 1.4.3 (Node set ratio)**.** The ratio of fill distance and separation distance for a given node set $X \subseteq \Omega$ is called node set ratio and defined as

$$\gamma_{X,\Omega} = \frac{h_{X,\Omega}}{s_X}. \tag{1.4.3}$$

Definition 1.4.4 (Quasi-uniform node sets)**.** Let Ω be bounded. A sequence of node sets $\{X_\lambda\}_{\lambda \in \Lambda} \subset \Omega$ is called quasi-uniform, if the node set ratio is uniformly bounded, i.e. if there exist constants $c_l, c_u > 0$ independent of λ, such that for all λ

$$c_l \leq \gamma_{X_\lambda,\Omega} \leq c_u. \tag{1.4.4}$$

Formally, quasi-uniformity of a single node set is vacuous, but a general goal is to keep c_l large and c_u small. What the notion of quasi uniformity represents is that we can assume that the distances between neighboring points will be approximately uniform throughout the domain, in the sense that the distance between closest and farthest neighbors can differ for a ratio of at most $\max(c_u, c_l)$, which we desire to be small. Additionally, the lower bound c_l is often non-problematic, as demonstrated by the next proposition.

Proposition 1.4.5 (Separation distance is smaller than fill distance)**.** *For any points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^d$ and any bounded Ω, such that $\Omega \setminus \bigcup_{i=1}^n B(\boldsymbol{x}_i, s_X/2)$ is nonempty, it holds that*

$$s_X \leq h_{X,\Omega}. \tag{1.4.5}$$

Proof. Let c be a point in $\Omega \setminus \bigcup_{i=1}^n B(\boldsymbol{x}_i, s_X/2)$. The fill distance can be bounded as

$$h_{X,\Omega} = 2 \sup_{\boldsymbol{x} \in \Omega} \min_{i=1,\ldots,n} \|\boldsymbol{x} - \boldsymbol{x}_i\| \geq 2 \min_{i=1,\ldots,n} \|\boldsymbol{c} - \boldsymbol{x}_i\| \geq 2 s_X/2 = s_X. \tag{1.4.6}$$

□

Remark 1.4.6. For the requirements of the Proposition 1.4.5 to be satisfied, it is sufficient to assume that Ω contains all midpoints of all pairs of data sites, or that Ω contains the convex hull of X.

Moreover, it is enough that Ω is connected. We can easily show that connectedness implies nonemptiness of $\Omega \setminus \bigcup_{i=1}^n B(\boldsymbol{x}_i, s_X/2)$ by contraposition. If $\Omega \setminus \bigcup_{i=1}^n B(\boldsymbol{x}_i, s_X/2)$ is empty, then $\Omega \subseteq \bigcup_{i=1}^n B(\boldsymbol{x}_i, s_X/2)$ and we can easily find a separation

$$U = B(\boldsymbol{x}_1, s_X/2) \quad \text{and} \quad V = \bigcup_{i=2}^{n} B(\boldsymbol{x}_i, s_X/2). \tag{1.4.7}$$

Both U and V are open, nonempty, since they contain at least $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively, and $U \cap V = \emptyset$, because all centers of the balls are at least s_X apart.

This means that we can most of the time safely assume that $c_l = 1$.

Example 1.4.7 (Equidistant intervals)**.** Consider a set X of $n+1$ equispaced points on an interval $[0, 1]$, $X_n = \{i/n, i = 0, \ldots, n\}$. Their separation distance is $h_{X_n,[0,1]} = 1/n$, which is realized if the center of a ball is placed in the midpoint between two data sites. The minimal separation distance is also $s_X = 1/n$, and X_n are quasi-uniform on $[0, 1]$, since $\gamma_{X_n,[0,1]} = 1$ for all n.

If we keep the set of points the same, but increase the domain to $[-1, 2]$, the fill distance stays at 1 no matter the n, and $\gamma_{X_n,[-1,2]}$ is no longer bounded.

Example 1.4.8 (Regular grid)**.** A regular grid of points G_h on $[0, 1]^d$ with spacing h has separation distance $s_{G_h} = h$ and fill distance $\sqrt{d}h$, if the center of the ball is positioned in a center of any grid cell. Regular grids are quasi-uniform, since $\gamma_{G_h,[0,1]^d}$ is bounded with $c_u = \sqrt{d}$.

Another important feature of quasi-uniform node sets is that the nodal spacing is proportional to the number of nodes in the usual way.

Proposition 1.4.9 (Relation between nodal spacing and number of nodes)**.** *Let $\Omega \subseteq \mathbb{R}^d$ be bounded and measurable and let $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ be quasi-uniform with constant $c_u > 0$. Then, there exist constants $c_1, c_2 > 0$, independent of X, such that*

$$c_1 n^{-1/d} \leq h_{X,\Omega} \leq c_2 n^{-1/d}. \tag{1.4.8}$$

Proof. Since $h := h_{X,\Omega}$ is the fill distance, $h/2$ is the covering radius and we have $\Omega \subseteq \bigcup_{i=1}^n B(\boldsymbol{x}_i, h/2)$. Measuring the volumes gives

$$|\Omega| \leq \left| \bigcup_{i=1}^n B(\boldsymbol{x}_i, h/2) \right| \leq \sum_{i=1}^n |B(\boldsymbol{x}_i, h/2)| = n \frac{\pi^{d/2}}{\Gamma(d/2+1)} (h/2)^d, \tag{1.4.9}$$

which proves the lower bound, as $2\sqrt[d]{|\Omega|} \frac{\sqrt[d]{\Gamma(d/2+1)}}{\sqrt{\pi}} n^{-1/d} \leq h$.

For the upper bound we use the fact that Ω is bounded. There exists a radius R and a point $\boldsymbol{x}_0$, such that $\Omega \subseteq B(\boldsymbol{x}_0, R)$. By extending the radius further by $s_X/2$, we ensure that this ball also includes all the empty balls centered at $\boldsymbol{x}_i$:

$$\bigcup_{i=1}^n B(\boldsymbol{x}_i, s_X/2) \subseteq B(x_0, R + s_X/2). \tag{1.4.10}$$

Because $B(\boldsymbol{x}_i, s_X/2)$ are disjoint, we can make the following comparison of volumes

$$n \frac{\pi^{d/2}}{\Gamma(d/2+1)} (s_X/2)^d \leq \frac{\pi^{d/2}}{\Gamma(d/2+1)} (R + s_X/2)^d, \tag{1.4.11}$$

which gives $n \leq (1 + 2R/s_X)^d$. Since $R \geq s_x/2$ (otherwise X would only have one point), we can estimate $n \leq (1 + 2R/s_X)^d \leq (4R/s_X)^d$. This can be further estimated by using $h \leq c_u s_X$, as it follows from quasi uniformity of X. This gives us

$$n \leq (4Rc_u)^d h^{-d} \quad \text{or} \quad h \leq 4Rc_u n^{-1/d}. \tag{1.4.12}$$

□

The last proposition shows that s_X, $h_{X,\Omega}$ and $n^{-1/d}$ are all equivalent measures of nodal spacing. Furthermore, if we look at a sequence of node sets $\{X_\lambda\}_{\lambda \in \Lambda}$ such that $h_{X_\lambda,\Omega} \to 0$, we can say that $|X_\lambda| = \Theta(h_{X_\lambda,\Omega}^{-d})$.

1.4.2 Error estimates

An important tool for error estimation of kernel-based interpolants are the *cardinal* or *Lagrange* basis functions and are defined by the Kronecker delta property:

Definition 1.4.10 (Cardinal functions)**.** Given a positive definite kernel $\mathcal{K}$ and a node set X, the functions $u_j^* \in \operatorname{span}\{\mathcal{K}(\cdot, \boldsymbol{x}_1), \dots, \mathcal{K}(\cdot, \boldsymbol{x}_n)\}$ that satisfy

$$u_j^*(\boldsymbol{x}_i) = \delta_{i,j}, \quad i, j = 1, \dots, n, \tag{1.4.13}$$

are called *cardinal* functions.

The asterisk in the name is due to the fact that cardinal functions can be seen as the dual basis to the evaluation functionals $\delta_{\boldsymbol{x}_i}$.

Cardinal functions trivialize the interpolation problem. Given data f_j, the interpolant $\hat{u}$ can be simply written as

$$\hat{u}(\boldsymbol{x}) = \sum_{j=1}^{n} f_j u_j^*(\boldsymbol{x}), \tag{1.4.14}$$

and since cardinal functions do not depend on f_j, having them known beforehand would be a major advantage. They are available beforehand only for specific point configurations, but can be computed pointwise. The following propositions offers such a way and also proves their existence.

Proposition 1.4.11 (Existence, uniqueness and computation of cardinal functions)**.** *For a given positive definite kernel $\mathcal{K}$ and a node set X, a cardinal basis exists, is unique, and can be computed pointwise as the solution of*

$$K\boldsymbol{u}^*(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x}), \tag{1.4.15}$$

where $\boldsymbol{u}^*(\boldsymbol{x}) = [u_j^*(\boldsymbol{x})]_{j=1}^n$ and $\boldsymbol{k}(\boldsymbol{x}) = [\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_i)]_{i=1}^n$.

Proof. One proof of existence and uniqueness is to view the Kronecker delta property as n interpolation problems, one for each u_j. All of this problems are uniquely solvable for a positive definite kernel $\mathcal{K}$, proving both. This approach is not computationally the most efficient.

Instead, consider an interpolation problem $\hat{u}(\boldsymbol{x}_i) = f_i$. Seeking the interpolant in the standard basis as $\hat{u}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^\mathsf{T}\boldsymbol{\alpha}$ gives rise to a linear system (1.2.3) for unknowns $\boldsymbol{\alpha}$, written as $K\boldsymbol{\alpha} = \boldsymbol{f}$. The solution can be written as $\boldsymbol{\alpha} = K^{-1}\boldsymbol{f}$, which gives the interpolant as $\hat{u}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^\mathsf{T}K^{-1}\boldsymbol{f}$. This form is exactly the cardinal form of interpolation (1.4.14) written in vector notation, giving $\boldsymbol{u}_j^*(\boldsymbol{x})^\mathsf{T} = \boldsymbol{k}(\boldsymbol{x})^\mathsf{T}K^{-1}$.

This also enables them to be computed for a given $\boldsymbol{x}$ as the solution of a system $K\boldsymbol{u}^*(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})$, due to symmetry of K. □

The cardinal basis can be used to estimate the pointwise error of the kernel interpolant, due to its ability to separate the values f_i from data sites x_i. The error analysis is done in the native space of the kernel $\mathcal{K}$, described in Section 1.2.

Theorem 1.4.12 (Error bound for kernel interpolation)**.** *For a given positive definite kernel $\mathcal{K}$, a node set X and a function f from the native space $\mathcal{N}_{\mathcal{K},X}$, we can bound the pointwise error as*

$$|f(\boldsymbol{x}) - \hat{u}(\boldsymbol{x})| \leq \|f\|_{N_{\mathcal{K},X}} P_{\mathcal{K},X}(\boldsymbol{x}), \tag{1.4.16}$$

where the power function $P_{\mathcal{K},X}$ *is equal to*

$$P_{\mathcal{K},X}(\boldsymbol{x}) = \sqrt{\mathcal{K}(\boldsymbol{x},\boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^\mathsf{T}K\boldsymbol{k}(\boldsymbol{x})}. \tag{1.4.17}$$

Proof. All the inner products and norms are from the native space $N_{\mathcal{K},X}$. Using the representation $\boldsymbol{u}_j^*(\boldsymbol{x})^\mathsf{T} = \boldsymbol{k}(\boldsymbol{x})^\mathsf{T}K^{-1}$, the reproducing properties, and Cauchy-Schwartz

inequality we get

$$\begin{aligned}
|f(\boldsymbol{x}) - \hat{u}(\boldsymbol{x})| &= \left| \langle f, \mathcal{K}(\cdot, \boldsymbol{x}) \rangle - \sum_{j=1}^{n} f_j u_j^*(\boldsymbol{x}) \right| = \\
&= \left| \langle f, \mathcal{K}(\cdot, \boldsymbol{x}) \rangle - \sum_{j=1}^{n} \langle f, \mathcal{K}(\cdot, \boldsymbol{x}_j) \rangle u_j^*(\boldsymbol{x}) \right| \\
&= \left| \langle f, \mathcal{K}(\cdot, \boldsymbol{x}) - \sum_{j=1}^{n} \mathcal{K}(\cdot, \boldsymbol{x}_j) u_j^*(\boldsymbol{x}) \rangle \right| \\
&= \left| \langle f, \mathcal{K}(\cdot, \boldsymbol{x}) - \boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} \boldsymbol{k}(\cdot) \rangle \right| \leq \|f\| \|\mathcal{K}(\cdot, \boldsymbol{x}) - \boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} \boldsymbol{k}(\cdot)\|.
\end{aligned} \tag{1.4.18}$$

The final term in the above inequality can be simplified further to obtain

$$\begin{aligned}
&\|\mathcal{K}(\cdot, \boldsymbol{x}) - \boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} \boldsymbol{k}(\cdot)\|^2 \\
&= \langle \mathcal{K}(\cdot, \boldsymbol{x}), \mathcal{K}(\cdot, \boldsymbol{x}) \rangle - 2 \langle \mathcal{K}(\cdot, \boldsymbol{x}), \boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} \boldsymbol{k}(\cdot) \rangle + \langle \boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} \boldsymbol{k}(\cdot), \boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} \boldsymbol{k}(\cdot) \rangle \\
&= \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) - 2 \boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} \boldsymbol{k}(\boldsymbol{x}) + \left\langle \sum_{j=1}^{n} \mathcal{K}(\cdot, \boldsymbol{x}_j) u_j^*(\boldsymbol{x}), \sum_{j=1}^{n} \mathcal{K}(\cdot, \boldsymbol{x}_j) u_j^*(\boldsymbol{x}) \right\rangle \\
&= \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) - 2 \boldsymbol{k}(\boldsymbol{x})^\mathsf{T} K^{-1} \boldsymbol{k}(\boldsymbol{x}) + \sum_{i=1}^{n} \sum_{j=1}^{n} u_i^*(\boldsymbol{x}) u_j^*(\boldsymbol{x}) \langle \mathcal{K}(\cdot, \boldsymbol{x}_i), \mathcal{K}(\cdot, \boldsymbol{x}_j) \rangle \\
&= \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) - 2 \boldsymbol{k}(\boldsymbol{x})^\mathsf{T} K^{-1} \boldsymbol{k}(\boldsymbol{x}) + \sum_{i=1}^{n} \sum_{j=1}^{n} u_i^*(\boldsymbol{x}) u_j^*(\boldsymbol{x}) \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
&= \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) - 2 \boldsymbol{k}(\boldsymbol{x})^\mathsf{T} K^{-1} \boldsymbol{k}(\boldsymbol{x}) + \boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} K \boldsymbol{u}^*(\boldsymbol{x}) \\
&= \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) - 2 \boldsymbol{k}(\boldsymbol{x})^\mathsf{T} K^{-1} \boldsymbol{k}(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^\mathsf{T} K^{-1} K K^{-1} \boldsymbol{k}(\boldsymbol{x}) \\
&= \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^\mathsf{T} K^{-1} \boldsymbol{k}(\boldsymbol{x}) = P_{\mathcal{K},X}(\boldsymbol{x})^2
\end{aligned}$$

□

Similar estimates can be made for derivatives of interpolants, and for augmented kernel interpolation. This type of interpolation uses conditionally positive definite kernels, which are analogous to conditionally positive RBFs. The following theorem states the general result.

Theorem 1.4.13 (Error bound for derivatives using augmented kernel interpolation)**.** *Let $\Omega \subseteq \mathbb{R}^n$ be open and assume that $\mathcal{K}$ is $2k$-continuous conditionally positive definite kernel on Ω with respect to $\mathcal{P} \subseteq C^k(\Omega)$. Suppose that the node set X is $\mathcal{P}$-unisolvent, and denote the interpolant of $f \in \mathcal{N}_{\mathcal{K},X}$ with $\hat{u}$. Then for every $x \in \Omega$ and every $\boldsymbol{\alpha} \in \mathbb{N}_0^d$ with $|\boldsymbol{\alpha}| \leq k$, the interpolation error is bounded by*

$$|D^{\boldsymbol{\alpha}} f(\boldsymbol{x}) - D^{\boldsymbol{\alpha}} \hat{u}(\boldsymbol{x})| \leq P_{\mathcal{K},X}^{(\boldsymbol{\alpha})}(\boldsymbol{x}) \|f\|_{\mathcal{N}_{\mathcal{K},X}}, \tag{1.4.19}$$

where the power function $P_{\mathcal{K},X}^{(\boldsymbol{\alpha})}(\boldsymbol{x})$ is given as

$$\begin{aligned}
P_{\mathcal{K},X}^{(\boldsymbol{\alpha})}(\boldsymbol{x})^2 = D_1^{\boldsymbol{\alpha}} D_2^{\boldsymbol{\alpha}} \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) - 2 \sum_{j=1}^{n} D^{\boldsymbol{\alpha}} u_j^*(\boldsymbol{x}) D_1^{\boldsymbol{\alpha}} \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_j) \\
+ \sum_{i=1}^{n} \sum_{j=1}^{n} D^{\boldsymbol{\alpha}} u_i^*(\boldsymbol{x}) D^{\boldsymbol{\alpha}} u_j^*(\boldsymbol{x}) \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j).
\end{aligned} \tag{1.4.20}$$

The bounds in terms of fill distance are typically obtained by bounding the power function $P_{\mathcal{K},X}$ or $P_{\mathcal{K},X}(\boldsymbol{x})$. These bounds are of the form

$$P_{\mathcal{K},X}(\boldsymbol{x})^2 \leq cF(h_{X,\Omega}), \tag{1.4.21}$$

where c is some constant independent of X. Many bounds for specific kernels, especially kernels generated by RBFs have been found over the years. A list of bounds is specified in Table 1.1 and additional theorems specifying bounds for the generalized power functions involving derivatives are proven in e.g. [Wen04, ch. 11].

1.4.3 Stability

Scattered data interpolant is defined by knowing its coefficients $\boldsymbol{\alpha}$, which can be computed from the interpolation system (1.2.3), (1.3.15) or (1.3.39), in case of augmentation. In all of these, the coefficients $\boldsymbol{\alpha}$ satisfy the relation

$$\boldsymbol{\alpha}^\mathsf{T} K \boldsymbol{\alpha} = \boldsymbol{\alpha}^\mathsf{T} \boldsymbol{f}, \tag{1.4.22}$$

where K is the interpolation matrix of the kernel. Knowing the lower and upper bounds on the above quadratic form as

$$\lambda \|\boldsymbol{\alpha}\|^2 \leq \boldsymbol{\alpha}^\mathsf{T} K \boldsymbol{\alpha} \leq \Lambda \|\boldsymbol{\alpha}\|^2 \tag{1.4.23}$$

enables us to estimate the relative perturbation of the solution in case of a perturbation in function values. Given this perturbation as $\boldsymbol{f} + \Delta\boldsymbol{f}$, we can estimate the resulting perturbation $\boldsymbol{\alpha} + \Delta\boldsymbol{\alpha}$ as

$$\frac{\|\Delta\boldsymbol{\alpha}\|}{\|\boldsymbol{\alpha}\|} \leq \frac{\Lambda}{\lambda} \frac{\|\Delta\boldsymbol{f}\|}{\|\boldsymbol{f}\|}. \tag{1.4.24}$$

If we focus on the case when $\mathcal{K}$ is positive definite, then optimal Λ and λ are the largest and the smallest eigenvalues of K, respectively. A more useful upper bound on Λ can be obtained using one of the classical matrix estimates: for a matrix $A \in \mathbb{R}^{n\times n}$ it holds that $\|A\|_2 \leq n \max_{i,j} |a_{i,j}|$, and thus

$$\Lambda \leq n \max_{i,j} |\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)| \leq n \|\mathcal{K}\|_\infty, \tag{1.4.25}$$

which shows that growth of Λ can be at most linear with n. Sadly, this is not the case for λ.

We will prove a famous result in RBF approximation theory: the trade-off principle by Schaback, which says that we cannot have both accuracy and stability at the same time, when computing $\boldsymbol{\alpha}$ in the standard basis (but it is possible in other basis, see the remarks at the end of Section 1.3.3). When reading the theorem it is good to keep in mind that $P^2_{\mathcal{K},X}$ is a factor in upper bounds for accuracy (1.4.16) and $1/\lambda$ is a factor in stability bounds.

Theorem 1.4.14 (Trade-off principle [Sch95b]). *Let u^*_j be cardinal functions for kernel $\mathcal{K}$ on a node set $X \subseteq \Omega \subseteq \mathbb{R}^n$. Then for all $\boldsymbol{x} \in \Omega \setminus X$ we have*

$$1 \leq 1 + \sum_{j=1}^{n} u^*_j(\boldsymbol{x})^2 \leq P_{\mathcal{K},X}(\boldsymbol{x})^2 / \lambda(K_{\boldsymbol{x}}), \tag{1.4.26}$$

where

$$\lambda(K_{\boldsymbol{x}}) = \inf_{\boldsymbol{\alpha}\neq\mathbf{0}} \frac{\boldsymbol{\alpha}^{\mathsf{T}} K_{\boldsymbol{x}} \boldsymbol{\alpha}}{\boldsymbol{\alpha}^{\mathsf{T}}\boldsymbol{\alpha}} \tag{1.4.27}$$

and $K_{\boldsymbol{x}}$ is the interpolation matrix of the kernel $\mathcal{K}$ on node set $X \cup \{\boldsymbol{x}\}$.

Proof. The proof is surprisingly short. Knowing that the power function $P_{\mathcal{K},X}$ is expressed as in the proof of Theorem 1.4.12, we get

$$P^2_{\mathcal{K},X}(\boldsymbol{x}) = \mathcal{K}(\boldsymbol{x},\boldsymbol{x}) - 2\boldsymbol{u}^*(\boldsymbol{x})^{\mathsf{T}}\boldsymbol{k}(\boldsymbol{x}) + \boldsymbol{u}^*(\boldsymbol{x})^{\mathsf{T}} K \boldsymbol{u}^*(\boldsymbol{x}) \tag{1.4.28}$$

If we add an additional point $\boldsymbol{x}_0 = \boldsymbol{x}$ to X and define $u^*_0(\boldsymbol{x}) = -1$, we can write the power function as a quadratic form and bound it

$$P^2_{\mathcal{K},X}(\boldsymbol{x}) = \sum_{i=0}^{n}\sum_{j=0}^{n} u^*_i(\boldsymbol{x}) u^*_j(\boldsymbol{x}) \mathcal{K}(\boldsymbol{x}_i,\boldsymbol{x}_j) \geq \lambda(K_x) \sum_{j=0}^{n} u^*_j(\boldsymbol{x})^2. \tag{1.4.29}$$

Writing the term at $j = 0$ separately gives the result. □

The trade-off principle says that both $P^2_{\mathcal{K},X}$ and $\lambda(K_x)^{-1}$ cannot be small together. The theorem also gives a bound on cardinal functions u^*_j and in turn on the Lebesgue function $\Lambda(\boldsymbol{x}) = \sum_{i=1}^{n} |u^*_j(\boldsymbol{x})|$.

Lower bounds for $\lambda(\mathcal{K})$ on an arbitrary set X are related to the separation distance s_X. They typically have the form

$$\lambda(\mathcal{K}) \geq c\, G(s_X), \tag{1.4.30}$$

where c is some constant, independent of X. Similarly to upper bounds of $P_{X,\Omega}$, many bounds for specific kernels, especially kernels generated by RBFs have been found over the years. A list of bounds is specified in Table 1.1 and additional theorems specifying bounds for the generalized power functions involving derivatives are proven in e.g. [Wen04, ch. 12]. The table is adapted from [Sch95b] and [Wen04], where the bounds are also referenced or proven.

Table 1.1: Accuracy and stability bounds for commonly used RBFs. When present, both parameters β and ε are assumed to be real and positive. The value c denotes a constant, and d denotes the number of dimensions.

Abbr.	Name	$\mathcal{K}(\boldsymbol{x},\boldsymbol{y}) = \phi(r)$, $r = \|\boldsymbol{x}-\boldsymbol{y}\|$	$F(h)$	$G(s)$
GA	Gaussians	$\exp(-(\varepsilon r)^2)$	$e^{-c\lvert\log h\rvert/h}$	$s^{-d}e^{-cd^2/(\varepsilon s)^2}$
MQ	Multiquadrics	$(-1)^{\lceil\beta\rceil}(1+(\varepsilon r)^2)^{\beta}, \beta\notin\mathbb{N}$	$e^{-c/h}$	$s^{\beta-(d-1)/2}e^{-cd/(\varepsilon s)}$
IMQ	Inverse MQ	$(1+(\varepsilon r)^2)^{-\beta}$	$e^{-c/h}$	$s^{-\beta-(d-1)/2}e^{-cd/(\varepsilon s)}$
PH	Polyharmonics	$(-1)^{\lceil\beta/2\rceil} r^{\beta}, \beta\notin 2\mathbb{N}$	h^{β}	s^{β}

These bounds also show the effect that the scaling of the shape parameter ε can have on the condition number. Scaling the parameter ε proportional to $1/s$ ($\approx 1/h$ for quasi-uniform node sets) will keep the product εs constant. The effects of this for operator approximation are illustrated in Example 2.2.11 (page 47).

Chapter 2

RBF-FD and similar methods

Radial-basis-function-generated finite differences (RBF-FD) is a strong-form meshless method for solving PDEs. More specifically, RBF-FD is a method of approximating partial differential operators on scattered nodes, which can than be canonically used to solve PDEs. To understand its place among other meshless methods and its distinctive features, we briefly review the history of meshless in Section 2.1. In Section 2.2 we describe how RBF-FD and similar methods approximate partial differential operators. Notably, this includes establishing an equivalence between derivation from function approximation and from the method of unknown coefficients. This is also generalized to least-squares based methods. Additional properties of stencil weights such as linearity and in some cases, the Kronecker delta property are established. With the outlook to using the approximations in practice, we review some important computational aspects. Finally, we describe how the computed approximations can be used to solve PDEs in Section 2.3, along with other commonly used techniques for handling boundary conditions.

2.1 A brief review of the history of meshless methods

One of the first developed meshless methods is the Smoothed Particle Hydrodynamics (SPH) [GM77], used initially for astronomical simulations, and today best known for its usage in computational fluid dynamics. The first meshless methods for boundary value problems were generalizations of FEM to a mesh-free setting, starting with the Diffuse Element Method (DEM) [NTV92] in 1992. DEM was soon extended by Belytschko into the Element Free Galerkin (EFG) method [BLG94] which has similar methodology to FEM in the sense that the problem is solved in its weak form using test and trial functions, but uses Moving Least Squares (MLS) based shape functions instead of (piecewise) monomials, since they can be defined without a mesh. It also uses a background grid for integration, and since its shape functions do not satisfy the Kronecker delta property, there is some additional work needed to enforce the boundary conditions, often via Lagrange multipliers. The use of a background grid led to some authors classifying the method as not "truly meshless", even though the precise definition has not been agreed upon [Bel+96]. Any methods that use background grids or cells of some kind are not seen as "truly meshless". There has also been a drive to develop "truly meshless" data pre- and post-processors [MKX07]. Additionally, while there may have been

a distinction between *meshless* and *mesh-free* methods, the terms are nowadays used interchangeably [CB15]. Another interesting class of methods developed in that time was based on reproducing kernels, which aimed to achieve higher order consistency than SPH [Liu+95]. The search for additional desirable properties also drove the development of partition of unity methods (PUM) [MB96; BM97]. A more thorough overview of the mentioned methods and other methods developed at the same time is given by Belytschko [Bel+96].

A few years after EFG, the Meshless Local Petrov Galerkin (MLPG) method was published by Atluri and Zhu [AZ98]. MLPG also solves the problem in its weak form, but locally, by enforcing the equation for selected subdomains of the whole computational domain. Similarly to FEM, the Boundary Element Method has been generalized to a meshless setting both with global weak forms as the Boundary Node Method [MM97] and local weak forms as the Local Boundary Integral Method (LBIM) [ZZA98]. One of the first books on the topic of meshless methods, which focuses mostly on weak form methods with applications to elasticity, was by Liu [Liu02] and an overview of more recent developments in the weak-form meshless community is given in the review paper by Nguyen et al. [Ngu+08], or a shorter article by Viana and Lai [VRL07].

Along with the development of weak-form methods, the development was also ongoing in the area of strong form methods. The finite difference method (FDM) was generalized to scattered grids [PK75; LO80] to allow for more flexible geometry, often called FDM methods on irregular or arbitrary grids. In general, the strong form methods were unified under the name "collocation methods" and can be viewed as a special case of weighted residual methods, where the test functions are chosen to be the Dirac delta distributions [Ngu+08]. Among the most notable strong-form collocation methods was the the Finite Point Method developed in 1996 [Oña+96], which approximates a partial differential operator using either least squares, weighted least squares or moving least squares method. The method is well researched and has been used with a wide variety of problems, including in 3D [OOI07]. Many other ideas that were used in weak-form methods were also applied to strong-form methods, such as reproducible kernel approximations, resulting in a point collocation method based on reproducing kernel approximations [Alu00]. Another notable method if the Finite Pointset Method, an improvement upon SPH, which is still actively used in practical applications [Jef+15]. It is described in great detail in a recent PhD thesis by Suchde [Suc18]. For more details on meshless methods, the reader can refer to textbooks by Fasshauser [Fas07] or the brief review by Chen and Belytschko [CB15].

2.1.1 Meshless methods and radial basis functions

Meshless methods based on radial basis function have been around nearly from the beginning. Kansa was the first to suggest in 1990 that RBFs could be used for derivative approximation [Kan90a] and, consequently, for development of numerical methods for solving PDEs [Kan90b]. This led to the development of the global collocation methods [FS98] (nowadays called "the Kansa method"), which exhibit good convergence properties (even exponential in appropriate circumstances [Che+03]), but with the same conditioning problem as the scattered interpolation, with various proposed solutions, such as better algorithms to avoid working in standard basis [FLF11] or using multi-

precision packages [KH17]. Moreover, the cost of the global approach scales as $O(N^3)$, where N represents the number of nodes, making it infeasible for large node sets.

As the answer to the shortcomings of the global methods, the local methods emerged. Along with already mentioned local methods, Tolstykh suggested the use of RBFs to generate finite difference-like formulas in his conference presentation "On using RBF-based differencing formulas for unstructured and mixed structured-unstructured grid calculations" [Tol00]. Afterwards, similar ideas were suggested a few times [WL02; SDY03], including in a PhD thesis by Wright [Wri03]. One benefit of RBF-based finite difference formulas is that they do not, contrary to polynomials, suffer from Haar-Mairhuber-Curtis theorem and the possibility of singular point configurations, as discussed in Chapter 1. The new method was eventually named RBF-generated finite differences (RBF-FD) and was and continues to be used in various applications, from elasticity [TS03], diffusion problems [ŠV06], fluid-flow [CS07; KŠ08], and more recently also flame propagation [KB13], dynamic thermal rating [KS18c] and option pricing [MS18]. The state and usages of global and local RBF-based methods are described in more detail in a review paper by Fornberg and Flyer [FF15c]. One of the main criticisms of the RBF-FD when compared to the least-squares based local methods is that the choice of the RBF and its shape parameter play a large role in accuracy and stability of the method [Suc18, sec. 2.2]. This is true, and plenty of research has been done on approximation properties of RBF-FD [Bay+10], the choice of the shape parameter [BMK11] and the scaling of the shape parameter [BMK12]. It was also suggested to scale the shape parameter in a way to keep the condition number below a certain threshold [Fly+12].

Recently, the role of polynomial augmentation has been investigated in a series of papers "On the role of polynomials in RBF-FD approximations" [Fly+16; Bay+17; BFF19] where polyharmonic (also called power [Wen04]) RBFs $\phi(r) = r^k$ augmented with monomials were suggested as a promising alternative to classical RBFs. Polyharmonics do not have a shape parameter and avoid bad conditioning under refinement, while keeping consistency due to inclusion of monomials. Preliminary comparison with least-squares methods have been performed [Bay19], but further investigation is needed. From out personal experience RBF-FD method with polyharmonic RBFs augmented with monomials was more stable with respect to the nodal positions than least-squares based methods, which is why it was also used in this work. Additional recent trends related to RBF-FD include solving PDEs on (possibly evolving) surfaces [SNK18; Pet+19], stabilization techniques for transport equations [Jav+19; Sok+19] and adaptivity (see review on page 92). Some parallelization efforts have also been made [BFE12], but more work in this area is also needed.

2.2 Approximation of partial differential operators

Most local strong form methods approximate a partial differential operator $\mathcal{L}$ applied to u at a point $\boldsymbol{p}$ in the form

$$(\mathcal{L}u)(\boldsymbol{p}) \approx \sum_{i=i}^{n} w_i u(\boldsymbol{x}_i) = \boldsymbol{w}^\mathsf{T} \boldsymbol{u}, \tag{2.2.1}$$

where $X = \{\boldsymbol{x}_1, \dots, x_n\}$ are nodes around $\boldsymbol{p}$, called the *stencil* of $\boldsymbol{p}$. The weights w_i depend on the operator $\mathcal{L}$, the point $\boldsymbol{p}$, and the neighboring nodes $\boldsymbol{x}_i$, but not on the

function u. When important, we will write the dependence explicitly as $w_i^{\mathcal{L},X}(\boldsymbol{p})$.

This type of an approximation can be viewed as approximating $\mathcal{L}$ at p with a functional that maps tuples of functional values in neighboring nodes to values of $\mathcal{L}$ applied to the function at $\boldsymbol{p}$. Assembling the weights in a vector $\boldsymbol{w} = [w_i]_{i=1}^n$ gives us the Riesz' representation of this functional, and the approximated value can be obtained by using just a dot product with $\boldsymbol{u} = [u(\boldsymbol{x}_i)]_{i=1}^n$.

2.2.1 Using scattered data interpolation

A general technique to obtain an approximation of an operator $\mathcal{L}$ at $\boldsymbol{p}$ is to construct an approximation $\hat{u} \approx u$ on a neighborhood $\Omega \subseteq \mathbb{R}^d$ of $\boldsymbol{p}$ and use $(\mathcal{L}\hat{u})(\boldsymbol{p})$ as an approximation of $(\mathcal{L}u)(\boldsymbol{p})$. We will assume that $\boldsymbol{p}$ has a stencil consisting of n points $X \subseteq \Omega$, $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. If the approximation $\hat{u}$ is of the form

$$\hat{u}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i b_i(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x})^\mathsf{T} \boldsymbol{\alpha}, \tag{2.2.2}$$

for some unknown coefficients $\boldsymbol{\alpha}$ and some basis $= (b_1, \ldots, b_n)$, $b_i \colon \Omega \to \mathbb{R}$, then the coefficients $\boldsymbol{\alpha}$ can be obtained by solving a system of linear equations

$$B\boldsymbol{\alpha} = \boldsymbol{u}, \tag{2.2.3}$$

where $\boldsymbol{u}$ is the vector of function values in stencil nodes $\boldsymbol{u} = [u(\boldsymbol{x}_i)]_{i=1}^n$ and B is the interpolation matrix $B = [b_j(\boldsymbol{x}_i)]_{i,j=1}^n$.

We can express $\boldsymbol{\alpha}$ as $\boldsymbol{\alpha} = B^{-1}\boldsymbol{u}$ and write the approximant as

$$\hat{u}(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x})^\mathsf{T}\boldsymbol{\alpha} = \boldsymbol{b}(\boldsymbol{x})^\mathsf{T} B^{-1}\boldsymbol{u}. \tag{2.2.4}$$

The approximation of $\mathcal{L}$ at $\boldsymbol{p}$ is now given as

$$(\mathcal{L}u)(\boldsymbol{p}) \approx (\mathcal{L}\hat{u})(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T} B^{-1}\boldsymbol{u} =: \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}\boldsymbol{u}. \tag{2.2.5}$$

The expression for $(\mathcal{L}\hat{u})(\boldsymbol{p})$ can be viewed in two ways

$$(\mathcal{L}\hat{u})(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T}\boldsymbol{\alpha} = \overbrace{(\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T} B^{-1}}^{\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}} \underbrace{\boldsymbol{u}}_{\boldsymbol{\alpha}} = \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}\boldsymbol{u}. \tag{2.2.6}$$

The first expression $(\mathcal{L}\hat{u})(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T}\boldsymbol{\alpha}$ gives the coefficients $\boldsymbol{\alpha}$ from known function values $\boldsymbol{u}$, offering us the way to obtain values of $\hat{u}$ for any $\boldsymbol{p}$, as well as any operator $\mathcal{L}$, by applying it to the basis functions. The second option $(\mathcal{L}\hat{u})(\boldsymbol{p}) = \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}\boldsymbol{u}$ allows us to compute $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}$, for a fixed point $\boldsymbol{p}$ and operator $\mathcal{L}$, but independently of function values (which may even be unknown at the time) and the same values of $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}$ can be used for different functions u.

We can compute $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}$ as $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T} = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T} B^{-1}$, which is equivalent to solving the system

$$B^\mathsf{T}\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}), \tag{2.2.7}$$

where the transpose of matrix B is used.

This can also be used to obtain the cardinal functions in a pointwise fashion, as described already in Section 1.4.2, by using $\mathcal{L} = \mathrm{id}$, i.e.

$$\boldsymbol{u}^*(\boldsymbol{x})^\mathsf{T} = b(\boldsymbol{x})^\mathsf{T} B^{-1}. \tag{2.2.8}$$

2.2.2 Using the method of undefined coefficients

The final system

$$B^{\mathsf{T}}\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}), \tag{2.2.9}$$

obtained above can also be viewed in a different light, as the method of undefined coefficients. Once again, we seek the approximation in form

$$(\mathcal{L}u)(\boldsymbol{p}) \approx \sum_{i=i}^{n} w_i u(\boldsymbol{x}_i) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}, \tag{2.2.10}$$

but instead of obtaining $\boldsymbol{w}$ from approximant $\hat{u}$, we require that approximation (2.2.9) is exact for a certain basis $\boldsymbol{b}$. This leads to n linear equations, obtained by substituting b_j for u:

$$(\mathcal{L}b_j)(\boldsymbol{p}) = \sum_{i=i}^{n} w_i b_j(\boldsymbol{x}_i), \tag{2.2.11}$$

which written as a linear system give

$$B^{\mathsf{T}}\boldsymbol{w} = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}), \tag{2.2.12}$$

which is the same system as derived before. This equivalence of these two derivations is helpful to understand the behavior of the approximations: on the one hand, the interpolation formulation is useful, as error analyses for interpolants are often available, while this formulation directly gives the reproduction properties by construction and also immediately derives the final system as needed in the computation.

2.2.3 RBF-FD with augmentation

If radial basis functions are used as the basis to obtain the stencil weights, the resulting method is called RBF-generated finite differences (RBF-FD). In case of positive definite RBFs, or *pure* RBF-interpolation, the procedure used to generate the weights is exactly the same as above, using $b_i = \phi_{\boldsymbol{x}_i} = \phi(\|\cdot - \boldsymbol{x}_j)$ as basis functions. This gives the system

$$\begin{bmatrix} \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_n\|) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} (\mathcal{L}\phi_{\boldsymbol{x}_1})(\boldsymbol{p}) \\ \vdots \\ (\mathcal{L}\phi_{\boldsymbol{x}_n})(\boldsymbol{p}) \end{bmatrix}, \tag{2.2.13}$$

which can be written more compactly as $A_\phi \boldsymbol{w} = \boldsymbol{\ell}_\phi$. The matrix A_ϕ is the same as the interpolation matrix in (1.3.15), due to its symmetry, and the solvability and stability analysis is exactly the same as in the interpolation case. This also generalizes to other positive definite kernels, and the properties derived in Chapter 1 can be applied.

Augmentation with monomials

For conditionally positive RBFs, we have to augment RBFs with monomials to ensure solvability. Recall from Section 1.3.3 that the interpolant is sought in the form

$$\hat{u}(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) + \sum_{\ell=1}^{q} \beta_\ell p_\ell(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x})^{\mathsf{T}} & \boldsymbol{\pi}(\boldsymbol{x})^{\mathsf{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix}, \tag{2.2.14}$$

where we used the $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\pi}(\boldsymbol{x})$ to denote the RBF and polynomial bases. The unknown coefficients of the approximation namely $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, are still expressed as solutions of a linear system

$$\begin{bmatrix} A_\phi & P \\ P^\mathsf{T} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \boldsymbol{u} \\ \mathbf{0} \end{bmatrix}, \tag{2.2.15}$$

where A_ϕ is the RBF collocation matrix and P the monomial collocation matrix, as defined in 1.3.39. An operator $\mathcal{L}$ applied to $\hat{u}$ can be expressed as

$$(\mathcal{L}\hat{u})(\boldsymbol{p}) = \begin{bmatrix} (\mathcal{L}\boldsymbol{\phi})(\boldsymbol{p}) \\ (\mathcal{L}\boldsymbol{\pi})(\boldsymbol{p}) \end{bmatrix}^\mathsf{T} \begin{bmatrix} A_\phi & P \\ P^\mathsf{T} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{u} \\ \mathbf{0} \end{bmatrix} := \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\lambda} \end{bmatrix}^\mathsf{T} \begin{bmatrix} \boldsymbol{u} \\ \mathbf{0} \end{bmatrix} = \boldsymbol{w}^\mathsf{T}\boldsymbol{u}. \tag{2.2.16}$$

Similarly to before, we see that the final expression remains a weighted sum of function values in stencil nodes, but the weights $\boldsymbol{w}$ are computed as a solution of the $(n+q) \times (n+q)$ system

$$\begin{bmatrix} A_\phi & P \\ P^\mathsf{T} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\ell}_\phi \\ \boldsymbol{\ell}_\pi \end{bmatrix}. \tag{2.2.17}$$

The vectors $\boldsymbol{\ell}_\phi = [(\mathcal{L}\phi_{\boldsymbol{x}_j})(p)]_{j=1}^n$ and $\boldsymbol{\ell}_\pi = [(\mathcal{L}p_k)(p)]_{k=1}^q$ contain the values of $\mathcal{L}$ applied to basis functions. The values $\boldsymbol{\lambda}$ are discarded after computing the solution.

This procedure can also be seen in the same light as the method of undefined coefficients. We would like to ensure that the approximation

$$(\mathcal{L}u)(\boldsymbol{p}) \approx \sum_{i=i}^{n} w_i u(\boldsymbol{x}_i) = \boldsymbol{w}^\mathsf{T}\boldsymbol{u}, \tag{2.2.18}$$

is exact for all RBFs $\phi_{\boldsymbol{x}_j}$ and all monomials p_ℓ. However, that is impossible, as the system would be over-determined. Instead, exactness is required for monomials,

$$(\mathcal{L}p_k)(\boldsymbol{p}) = \boldsymbol{w}^\mathsf{T}\boldsymbol{u}, \quad \forall k = 1, \dots, q \tag{2.2.19}$$

giving the second equation $P^\mathsf{T}\boldsymbol{w} = \boldsymbol{\ell}_\pi$ of (2.2.17). The meaning of the whole system is summarized in the following proposition, where we can se that the discarded values λ act as Lagrange multipliers.

Proposition 2.2.1 (RBF-FD weights as constrained minimization [Fly+16])**.** *The RBF-FD weights $\boldsymbol{w}$ with monomial augmentation are determined as a solution to the equality-constrained minimization problem*

$$\min J(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^\mathsf{T} A_\phi \boldsymbol{w} - \boldsymbol{w}^\mathsf{T}\boldsymbol{\ell}_\phi \quad \textit{subject to} \quad P^\mathsf{T}\boldsymbol{w} = \boldsymbol{\ell}_\pi. \tag{2.2.20}$$

Proof. The Lagrangian is given by

$$L(\boldsymbol{w}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{w}^\mathsf{T} A_\phi \boldsymbol{w} - \boldsymbol{w}^\mathsf{T}\boldsymbol{\ell}_\phi - \boldsymbol{\lambda}^\mathsf{T}(P^\mathsf{T}\boldsymbol{w} - \boldsymbol{\ell}_\pi), \tag{2.2.21}$$

and the conditions $\nabla_{\boldsymbol{w}} L = \mathbf{0}$ and $\nabla_{\boldsymbol{\lambda}} L = \mathbf{0}$ give exactly the system (2.2.17). The fact that the solution is a global minimum follows from the fact that A_ϕ is positive definite on the null space of P^T. Indeed, let $(\boldsymbol{w}, \boldsymbol{\lambda})$ be the solution of (2.2.17) and $\tilde{\boldsymbol{w}}$ some other

vector satisfying the constraints. The difference $\Delta\boldsymbol{w} = \tilde{\boldsymbol{w}} - \boldsymbol{w} \neq \mathbf{0}$ satisfies $P^\mathsf{T}\Delta\boldsymbol{w} = \mathbf{0}$ and the minimized expression is equal to

$$J(\tilde{\boldsymbol{w}}) = \frac{1}{2}(\boldsymbol{w} + \Delta\boldsymbol{w})^\mathsf{T} A_\phi(\boldsymbol{w} + \Delta\boldsymbol{w}) - (\boldsymbol{w} + \Delta\boldsymbol{w})^\mathsf{T}\boldsymbol{\ell}_\phi \tag{2.2.22}$$

$$= J(\boldsymbol{w}) + \frac{1}{2}\Delta\boldsymbol{w}^\mathsf{T} A_\phi \Delta\boldsymbol{w} + \Delta\boldsymbol{w}^\mathsf{T} A_\phi \boldsymbol{w} - \Delta\boldsymbol{w}^\mathsf{T}\boldsymbol{\ell}_\phi \tag{2.2.23}$$

$$= J(\boldsymbol{w}) + \frac{1}{2}\Delta\boldsymbol{w}^\mathsf{T} A_\phi \Delta\boldsymbol{w} + \Delta\boldsymbol{w}^\mathsf{T}(\boldsymbol{\ell}_\phi - P\boldsymbol{\lambda}) - \Delta\boldsymbol{w}^\mathsf{T}\boldsymbol{\ell}_\phi \tag{2.2.24}$$

$$= J(\boldsymbol{w}) + \frac{1}{2}\Delta\boldsymbol{w}^\mathsf{T} A_\phi \Delta\boldsymbol{w} > J(\boldsymbol{w}), \tag{2.2.25}$$

because A_ϕ is positive definite on the null space of P^T. □

2.2.4 Least-squares based methods

Another way of obtaining strong form approximations for $\mathcal{L}$ is by not using interpolants over local neighborhoods but only approximants. We will derive the procedure for computation of stencil weights, based on weighted least squares approximation. We can recall from Section 1.3.2 that the approximant is of the form

$$\hat{u}(\boldsymbol{x}) = \sum_{j=1}^{m} \alpha_j b_j(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x})^\mathsf{T}\boldsymbol{\alpha} \tag{2.2.26}$$

for some basis b_j (usually monomials) and that coefficients $\boldsymbol{\alpha}$ are obtained as a solution of a least squares problem

$$WB\boldsymbol{\alpha} = W\boldsymbol{u}, \tag{2.2.27}$$

where $B = [b_j(x_i)]_{i=1,j=1}^{n,\ell}$ is the $n \times \ell$ collocation matrix and $W_{ii} = \sqrt{\omega(\boldsymbol{x}, \boldsymbol{x}_i)}$ is the diagonal matrix of weights.

Expressing the coefficients $\boldsymbol{\alpha}$ as $\boldsymbol{\alpha} = (B^\mathsf{T}W^2B)^{-1}B^\mathsf{T}W^2\boldsymbol{u}$ allows us to express the value of $\mathcal{L}$ applied to $\hat{u}$ at $\boldsymbol{p}$ as

$$(\mathcal{L}\hat{u})(\boldsymbol{p}) = (\mathcal{L}b)(\boldsymbol{p})^\mathsf{T}(B^\mathsf{T}W^2B)^{-1}B^\mathsf{T}W^2\boldsymbol{u} =: \boldsymbol{w}^\mathsf{T}\boldsymbol{u}, \tag{2.2.28}$$

where the stencil weights $\boldsymbol{w}$ are computed as

$$\boldsymbol{w} = [(\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T}(B^\mathsf{T}W^2B)^{-1}B^\mathsf{T}W^2]^\mathsf{T} = W^2B(B^\mathsf{T}W^2B)^{-1}(\mathcal{L}\boldsymbol{b})(\boldsymbol{p}). \tag{2.2.29}$$

Similarly to how we developed alternative views on computations of $\boldsymbol{w}$ in case of interpolation or augmented interpolation, we will find an interpretation of $\boldsymbol{w}$ as a solution of an under-determined weighted problem.

Proposition 2.2.2 (Alternative derivation of WLS-based stencil weights)**.** *The WLS stencil weights $\boldsymbol{w}$ are a solution of*

$$\min_{\boldsymbol{w}\in\mathbb{R}^n} \sum_{i=1}^{n} (w_i/W_{ii})^2, \quad \textit{subject to} \quad B^\mathsf{T}\boldsymbol{w} = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}). \tag{2.2.30}$$

Proof. The constraints of the problem are obtained by requiring the stencil weights to exactly reproduce $\mathcal{L}$ for the basis functions b_j:

$$(\mathcal{L}b_j)(\boldsymbol{p}) = \sum_{i=1}^{n} w_i b_j(x_i), \quad \forall j = 1, \dots, \ell. \tag{2.2.31}$$

This is similar to the method of undetermined coefficients, but the number of equations ℓ is less (or equal to) than the number of unknowns n. The remaining degrees of freedom are determined by minimization instead of adding more basis functions.

The weighted minimization problem can be transformed to an ordinary least-norm under-determined problem, by substituting $\boldsymbol{w} = W\tilde{\boldsymbol{w}}$ and solving the problem

$$\min_{\tilde{\boldsymbol{w}} \in \mathbb{R}^n} \sum_{i=1}^{n} \tilde{\boldsymbol{w}}^2, \quad \text{subject to} \quad B^\mathsf{T} W \tilde{\boldsymbol{w}} = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}). \tag{2.2.32}$$

The solution of least-norm under-determined problem is expressed as the pseudoinverse, which takes the form

$$\tilde{\boldsymbol{w}} = (B^\mathsf{T} W)^\mathsf{T} (B^\mathsf{T} W (B^\mathsf{T} W)^\mathsf{T})^{-1} (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}) = WB(B^\mathsf{T} W^2 B)^{-1} (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}), \tag{2.2.33}$$

giving the final weights as

$$\boldsymbol{w} = W\tilde{\boldsymbol{w}} = W^2 B(B^\mathsf{T} W^2 B)^{-1} (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}). \tag{2.2.34}$$

□

The computation of the weights in this case is best done by first solving the under-determined system (2.2.32), using e.g. normal equations, QR, SVD, and then multiplying the final result by W.

2.2.5 Properties of stencil weights

So far, we derived three different procedures for computing stencil weights $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})$ and two views for each of them: one derived by applying $\mathcal{L}$ to a suitable approximant and another by requiring exactness for a certain set of functions and determining the remaining coefficients using suitable minimization. The three options will be denoted as follows.

- Interpolation:
$$\boldsymbol{w}^{\mathcal{L},X}_{I(\boldsymbol{b})}(\boldsymbol{p}) = B^{-\mathsf{T}} (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}), |X| = |\boldsymbol{b}| = n \tag{2.2.35}$$
- Polynomial augmented RBF:
$$\boldsymbol{w}^{\mathcal{L},X}_{\mathrm{RBF}(\phi,m)}(\boldsymbol{p}) = \begin{bmatrix} I_n & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} A_\phi & P \\ P^\mathsf{T} & 0 \end{bmatrix}^{-1} \begin{bmatrix} (\mathcal{L}\boldsymbol{\phi})(\boldsymbol{p}) \\ (\mathcal{L}\boldsymbol{\pi})(\boldsymbol{p}) \end{bmatrix} \tag{2.2.36}$$
- WLS:
$$\boldsymbol{w}^{\mathcal{L},X}_{\mathrm{WLS}(\boldsymbol{b},\,\omega)}(\boldsymbol{p}) = W^2 B(B^\mathsf{T} W^2 B)^{-1} (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}), |X| = n, |\boldsymbol{b}| = \ell \tag{2.2.37}$$

The subscript $I(\boldsymbol{b})$ denotes interpolation with basis functions $\boldsymbol{b}$, subscript RBF(ϕ, m) denotes RBF interpolation augmented with monomials up and including degree m, and subscript WLS$(\boldsymbol{b}, \omega)$ WLS approximation with basis $\boldsymbol{b}$ and weight function ω.

The following prepositions restates the equality of cardinal functions and stencil weights for interpolation, proven in 2.2.1, into above notation.

Proposition 2.2.3 (Interpolation stencil weights evaluated at stencil nodes)**.** *The stencil weights computed using interpolation satisfy the Kronecker-delta property:*

$$(\boldsymbol{w}^{\mathrm{id},X}_{I(\boldsymbol{b})}(\boldsymbol{x}_i))_j = \delta_{ij}. \tag{2.2.38}$$

Furthermore, the derivatives of stencil weights are easy to compute, regardless of the approximation procedure used to obtain the weights.

Proposition 2.2.4 (Differentiation of stencil weights)**.**

$$(\mathcal{L}\boldsymbol{w}^{\mathrm{id},X})(\boldsymbol{p}) = \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p}) \tag{2.2.39}$$

Proof. All three weights can be computed as $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p}) = M(\mathcal{L}\boldsymbol{b})(\boldsymbol{p})$, for some constant matrix M. Thus,

$$\mathcal{L}\boldsymbol{w}^{\mathrm{id},X}(\boldsymbol{p}) = \mathcal{L}M(\boldsymbol{b})(\boldsymbol{p}) = M(\mathcal{L}\boldsymbol{b})(\boldsymbol{p}) = \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p}). \tag{2.2.40}$$

□

Both RBF and WLS stencil weights reduce to interpolation stencil weights for special cases.

Proposition 2.2.5 (Special cases reducing to interpolation)**.** *The following special cases hold:*

- *If RBF is used without augmentation, it reduces to RBF interpolation:*

$$\boldsymbol{w}^{X,\mathcal{L}}_{\mathit{RBF}(\phi,0)} = \boldsymbol{w}^{X,\mathcal{L}}_{I((\phi_{\boldsymbol{x}_j})_{j=1}^n)}, \tag{2.2.41}$$

- *If the number of basis functions is the same as the number of stencil nodes in WLS approximation, it reduces to interpolation:*

$$\boldsymbol{w}^{X,\mathcal{L}}_{\mathit{WLS}((b_j)_{j=1}^n,\,\omega)} = \boldsymbol{w}^{X,\mathcal{L}}_{I((b_j)_{j=1}^n)}. \tag{2.2.42}$$

- *If the number of stencils nodes equals the number of augmenting monomials, i.e.* $|X| = \binom{m+d}{d}$*, then stencil weights reduce to polynomial interpolation:*

$$\boldsymbol{w}^{X,\mathcal{L}}_{\mathit{RBF}(\phi,m)} = \boldsymbol{w}^{X,\mathcal{L}}_{I(\boldsymbol{\pi})}, \tag{2.2.43}$$

In particular, the second point also means that for $\ell = n$ the WLS shapes are independent of the weight ω.

Proof. The first point is obvious from the definition of RBF interpolation. The second point is obvious from definition of weighted least squares, since the system is not overdetermined and can be solved exactly. Alternatively, using the interpretation in the spirit of Proposition 2.2.2, there is nothing to minimize, since constraints leave no degrees of freedom. Finally, this can also be made clear directly from the expression for stencil weights. The matrix B is square if $n = \ell$ and the basis and points are assumed to be such that B is invertible. This gives

$$\boldsymbol{w}^{X,\mathcal{L}}_{\mathrm{WLS}((b_j)_{j=1}^n,\,\omega)} = W^2 B (B^\mathsf{T} W^2 B)^{-1} (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}) \tag{2.2.44}$$

$$= W^2 B B^{-1} W^{-2} B^{-\mathsf{T}} (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}) = B^{-\mathsf{T}} (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}) = \boldsymbol{w}^{X,\mathcal{L}}_{I((\phi_{\boldsymbol{x}_j})_{j=1}^n)}. \tag{2.2.45}$$

Similarly, if the number of points in RBF interpolation is equal to the number of constraints, the constraints in (2.2.20) consume all the degrees of freedom. Alternatively, since P is invertible in this case, the equation $P^\mathsf{T}\boldsymbol{\alpha} = \mathbf{0}$ implies $\boldsymbol{\alpha} = 0$ reducing it to polynomial interpolation. □

Proposition 2.2.6 (Reduction to FDM)**.** *The interpolation stencil weights reduce to standard finite difference weights if the nodes X form a regular grid around $\boldsymbol{p}$, i.e. $X = G_h^k = \prod_{i=1}^d \{p_i + jh, j = -k, \dots, k\}$ and the basis $\boldsymbol{b}$ is the tensor basis of monomials, $\boldsymbol{b}(\boldsymbol{x}) = \bigotimes_{i=1}^d \{x_i^j, j = 0, \dots, 2k\}$.*

Proof. In the setup as described above, the interpolating function $\hat{u}$ becomes a tensor product of Lagrange interpolating polynomials on a grid, which is exactly the setup used to derive stencil weights in the finite difference method. □

A useful property of all stencil weights is linearity in $\mathcal{L}$.

Proposition 2.2.7 ($\mathcal{L}$-linearity of stencil weights)**.** *All three types of stencil weights $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})$ are linear in $\mathcal{L}$, i.e.*

$$\boldsymbol{w}^{\alpha\mathcal{L}_1+\beta\mathcal{L}_2,X}(\boldsymbol{p}) = \alpha \boldsymbol{w}^{\mathcal{L}_1,X}(\boldsymbol{p}) + \beta \boldsymbol{w}^{\mathcal{L}_2,X}(\boldsymbol{p}). \tag{2.2.46}$$

Proof. All three weights can be computed as $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p}) = M(\mathcal{L}\boldsymbol{b})(\boldsymbol{p})$, for some matrix M independent of $\mathcal{L}$ and some basis functions $\boldsymbol{b}$. Linearity follows directly:

$$\boldsymbol{w}^{\alpha\mathcal{L}_1+\beta\mathcal{L}_2,X}(\boldsymbol{p}) = M((\alpha\mathcal{L}_1 + \beta\mathcal{L}_2)\boldsymbol{b})(\boldsymbol{p}) \tag{2.2.47}$$

$$= M\left[(\alpha(\mathcal{L}_1\boldsymbol{b}) + \beta(\mathcal{L}_2\boldsymbol{b})\right](\boldsymbol{p})) \tag{2.2.48}$$

$$= M(\alpha(\mathcal{L}_1\boldsymbol{b})(p) + \beta(\mathcal{L}_2\boldsymbol{b})(\boldsymbol{p})) \tag{2.2.49}$$

$$= \alpha M(\mathcal{L}_1\boldsymbol{b})(p) + \beta M(\mathcal{L}_2\boldsymbol{b})(\boldsymbol{p}) \tag{2.2.50}$$

$$= \alpha \boldsymbol{w}^{\mathcal{L}_1,X}(\boldsymbol{p}) + \beta \boldsymbol{w}^{\mathcal{L}_2,X}(\boldsymbol{p}). \qquad \square$$

The consequences of linearity is that stencil weights for operators of the form

$$\mathcal{L} = \sum_{|\boldsymbol{\alpha}|\leq k} a_{\boldsymbol{\alpha}} D^{\boldsymbol{\alpha}}, \quad D^{\boldsymbol{\alpha}} = \frac{\partial^{|\boldsymbol{\alpha}|}}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}}, \tag{2.2.51}$$

can be easily computed knowing only finitely many "basis" stencil weights $\boldsymbol{w}^{D^{\boldsymbol{\alpha}},X}(\boldsymbol{p})$.

Note that this property is useful only if $\mathcal{L}$ is known in such form, and even if it is, applying $\mathcal{L}$ to basis functions directly might be easier or more numerically stable then computing the weights using linearity.

2.2.6 Computational aspects

All of the formulas (2.2.35), (2.2.36) and (2.2.37) for stencil weights contain a matrix inverse. However, the formulas should not be used as written, but instead by solving an appropriate linear system and then possibly transforming the solution. This ensures better control over the stability properties and the execution time with a wide variety of matrix decompositions to choose from, depending on the properties of the matrix. For positive definite RBF or for WLS with normal equations Cholesky decomposition can be used, otherwise general purpose LU or QR solvers can be used, or even SVD. This is also useful in situations where weights for multiple operators $\mathcal{L}$ have to be computed on the same set of nodes X. As only the right hand side of the system depends on $\mathcal{L}$, the matrix can be computed and decomposed only once (in cubic time), the decomposition can be stored and subsequent weights can be computed in quadratic time.

Another important aspect that is often only noted in passing (e.g. [Ngu+08]) is the importance of evaluation in the local scaled coordinate frame. The following example illustrates the importance of this recommendation when computing weights numerically.

Example 2.2.8 (FDM weight computation – shifting)**.** Consider a classical one-dimensional setup with $\boldsymbol{b} = \{1, x, x^2\}$, compute the stencil weights $\boldsymbol{w}^{X,\mathcal{L}}_{I(\boldsymbol{b})}$ around point x^* with stencil $X = \{x_0 - h, x_0, x_0 + h\}$. In a typical situation we expect x^* to be close to x_0, eg. $x_0 - x^* = \Theta(h)$, while x_0 and h can be arbitrary.

Without translation and scaling, the matrix interpolation matrix B_1 is equal to

$$B_1 := \begin{bmatrix} 1 & x_0 - h & (x_0 - h)^2 \\ 1 & x_0 & x_0^2 \\ 1 & x_0 + h & (x_0 + h)^2 \end{bmatrix}. \tag{2.2.52}$$

Evaluated in a local frame, by shifting the origin into point x^*, we obtain

$$B_2 := \begin{bmatrix} 1 & x_0 - x^* - h & (x_0 - x^* - h)^2 \\ 1 & x_0 - x^* & (x_0 - x^*)^2 \\ 1 & x_0 - x^* + h & (x_0 - x^* + h)^2 \end{bmatrix}. \tag{2.2.53}$$

Closed form expressions are available for Frobenius condition numbers of B_i:

$$\kappa_F(B_i) = \frac{\sqrt{\left(2h^4+2h^2\left(6\chi_i^2+1\right)+3\left(\chi_i^4+\chi_i^2+1\right)\right)\left(4h^4+h^2\left(2-6\chi_i^2\right)+6\left(\chi_i^4+4\chi_i^2+1\right)\right)}}{2h^2} \tag{2.2.54}$$

$$= \frac{3}{\sqrt{2}}\frac{1}{h^2} + \frac{15}{2\sqrt{2}}\frac{\chi_i^2}{h^2} + \frac{3}{2\sqrt{2}} + \frac{21\chi_i^2}{4\sqrt{2}} + O(h^2, \chi_i^2), \tag{2.2.55}$$

where $\chi_1 = x_0$ and $\chi_2 = x_0 - x^*$. The Taylor expansion shows that B_1 becomes ill-conditioned if $|x_0| \gg 1$ or $|h| \ll 1$. Case 2 fixes the first problem, as $\chi = \Theta(h)$, but still suffers from instabilities when $|h| \ll 1$.

To avoid these unnecessary numerical inaccuracies, all expressions should be evaluated as functions of local dimensionless coordinates $\frac{\boldsymbol{x}_i - \boldsymbol{x}^*}{\delta}$, where $\boldsymbol{x}^*$ is a chosen point, usually near the center of X and δ is a local measure of distance, such as the diameter of X or distance between the two nodes closest to $\boldsymbol{x}^*$. This makes the computational procedure independent of the choice of origin and the units of the coordinate system. We can also view this as choosing a local basis $\{b_j((\cdot - \boldsymbol{x}^*)/\delta)\}$ instead of $\{b_j\}$. However, when computing $\mathcal{L}$ applied to scaled b_j, one has to take care to not omit additional factors of δ that might appear due to scaling.

Example 2.2.9 (FDM weight computation – scaling)**.** Continuing with the same notation from example above, we compute B_3 using the approach described above. We take $\delta = h$ and the matrix B_3 is

$$B_3 := \begin{bmatrix} 1 & \chi_3 - h & (\chi_3 - h)^2 \\ 1 & \chi_3 & \chi_3^2 \\ 1 & \chi_3 + h & (\chi_3 + h)^2 \end{bmatrix}, \tag{2.2.56}$$

where $\chi_3 = \frac{x_0 - x^*}{h}$. Again, closed form expression for Frobenius condition number is available,

$$\kappa_F(B_3) = \sqrt{\frac{3}{2}}\sqrt{\chi_3^4 + 3\chi_3^2 + 2}\sqrt{3\left(\chi_3^2 + 5\right)\chi_3^2 + 7} \tag{2.2.57}$$

$$= \sqrt{21} + \frac{51}{4}\sqrt{\frac{3}{7}}\chi_3^2 + \mathcal{O}\left(\chi_3^4\right), \tag{2.2.58}$$

In this case, the condition number is bounded under initial assumption $\chi_3 = O(1)$ and the problem is well conditioned regardless of the choice of x_0 and h.

As a concrete example for choice of $x_0 = 100$, $h = 0.02$ and $x^* = 100.01$, the following condition numbers are computed:

$$\kappa_F(B_1) \approx 5.30463 \cdot 10^{11}, \tag{2.2.59}$$

$$\kappa_F(B_2) \approx 5305.69, \tag{2.2.60}$$

$$\kappa_F(B_3) \approx 6.79283, \tag{2.2.61}$$

further demonstrating the differences in described approaches.

Remark 2.2.10. All the matrices B_i appearing in examples 2.2.8 and 2.2.9 were Vandermonde matrices, and despite being unstable, linear systems involving Vandermonde matrices can be solved un a stable way using QR decomposition [BNT19]. However, despite being able to overcome the issues outlined in these particular cases with a more involved method, shifting and scaling the coordinate system beforehand is still beneficial in general, as we can ensure that the same stencil will result in the same matrix, regardless of its absolute position, the choice of units and the linear solver being used.

The points discussed in this chapter are all reflected in the implementation of the approximation, as described in Section 5.2.2.

2.2.7 Examples

Apart from the example coinciding with FDM discussed in the previous section, we will show a few other illustrative examples. Some regular examples can be computed in closed form, and the convergence properties of various RBFs have been investigates in such cases [Bay+10].

Since all commonly used computational procedures for stencil weights are translation invariant, we will always choose the stencil around the origin.

Example 2.2.11 (Gaussian RBF-FD weights on a regular stencil)**.** Consider the RBF-FD weights on a stencil $X = \{(0,0),(-h,0),(h,0),(0,-h),(0,h)\}$ for the Laplacian operator using Gaussian RBFs. The system for weights $\boldsymbol{w}^{X,\nabla^2}(\mathbf{0})$ is

$$\begin{bmatrix} 1 & e^{-\varepsilon^2h^2} & e^{-\varepsilon^2h^2} & e^{-\varepsilon^2h^2} & e^{-\varepsilon^2h^2} \\ e^{-\varepsilon^2h^2} & 1 & e^{-4\varepsilon^2h^2} & e^{-2\varepsilon^2h^2} & e^{-2\varepsilon^2h^2} \\ e^{-\varepsilon^2h^2} & e^{-4\varepsilon^2h^2} & 1 & e^{-2\varepsilon^2h^2} & e^{-2\varepsilon^2h^2} \\ e^{-\varepsilon^2h^2} & e^{-2\varepsilon^2h^2} & e^{-2\varepsilon^2h^2} & 1 & e^{-4\varepsilon^2h^2} \\ e^{-\varepsilon^2h^2} & e^{-2\varepsilon^2h^2} & e^{-2\varepsilon^2h^2} & e^{-4\varepsilon^2h^2} & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = \begin{bmatrix} -4\varepsilon^2 \\ 4e^{-\varepsilon^2h^2}\varepsilon^2\left(\varepsilon^2h^2-1\right) \\ 4e^{-\varepsilon^2h^2}\varepsilon^2\left(\varepsilon^2h^2-1\right) \\ 4e^{-\varepsilon^2h^2}\varepsilon^2\left(\varepsilon^2h^2-1\right) \\ 4e^{-\varepsilon^2h^2}\varepsilon^2\left(\varepsilon^2h^2-1\right) \end{bmatrix}, \tag{2.2.62}$$

which can be solved to obtain

$$\boldsymbol{w}^{X,\nabla^2}(\mathbf{0})^\mathsf{T} = \left[-4\varepsilon^2\left(1+\frac{\varepsilon^2h^2}{\sinh^2(\varepsilon^2h^2)}\right), w_2, w_2, w_2, w_2\right], w_2 = \frac{4\varepsilon^4h^2e^{3\varepsilon^2h^2}}{\left(e^{2\varepsilon^2h^2}-1\right)^2}. \tag{2.2.63}$$

The weights do not sum to zero, which means that the approximation does not even reproduce constants. However, the approximation is of second order, as we can compute

$$\boldsymbol{u}^\mathsf{T}\boldsymbol{w}^{X,\nabla^2} - \nabla^2 u = 2\varepsilon^4h^2u + \varepsilon^2h^2\nabla^2u + \frac{1}{12}h^2\left(\frac{\partial^4u}{\partial x^4}+\frac{\partial^4u}{\partial y^4}\right) + O(h^4), \tag{2.2.64}$$

where all the expressions are evaluated at $(0,0)$. If ε is constant, this approximation converges with order 2, as shown in Figure 2.1. While the theoretical and computed errors match initially, the computed approximation eventually breaks down due to bad conditioning of the interpolation matrix. But, if we scale $\varepsilon = \varepsilon_0/h$ to avoid conditioning problems, the leading error terms change to

$$2\varepsilon_0^2/h^2u + \varepsilon_0^2\nabla^2u + \frac{1}{12}h^2\left(\frac{\partial^4u}{\partial x^4}+\frac{\partial^4u}{\partial y^4}\right) + O(h^4,h^{-4}), \tag{2.2.65}$$

which is of the form $c_1h^{-2}+c_2+c_3h^2$, where the constants depend on u and ε_0. This is not convergent and gives a classical error shape with two local minima, as shown in Figure 2.1. The deviation between the theoretical and predicted errors for scaled version is due to the fact the theoretical version only includes the initial terms of the Taylor expansion as derived in (2.2.64).

This lack of convergence is important enough that it has its own name, and has been called divergence due to "stagnation" or "saturation" errors [Fly+16]. The shape ε is often chosen as small as possible due to conditioning, and the divergent terms come into play at much smaller h than in our example, when the domain is already "saturated" with nodes, and the error "stagnates". However, the true behavior of this phenomenon is simply that the approximation with scaled ε is not convergent with the truncation error as described in (2.2.65).

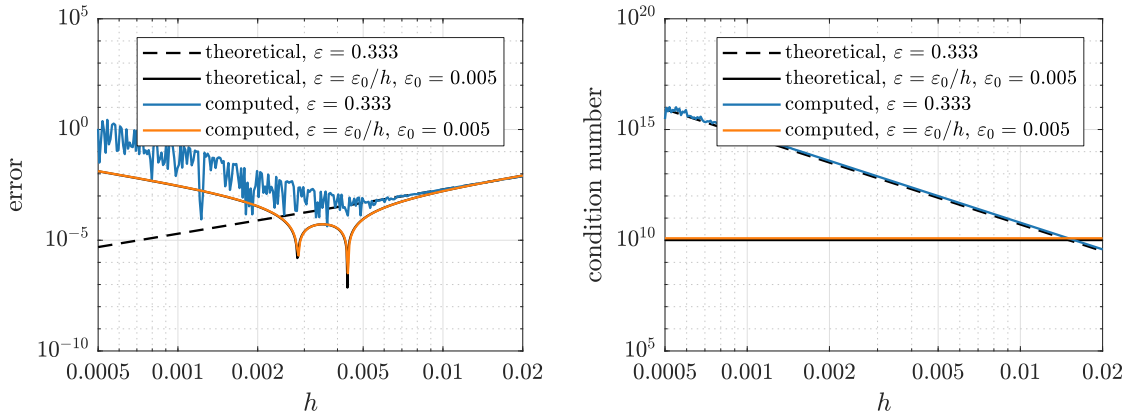


Figure 2.1: Error and condition number of the approximation of the Laplacian on regular grid with constant and scaled shape parameter ε using Gaussian RBFs. The theoretical and computed lines in the right plot overlap completely, so they are drawn at a slight offset to keep both visible.

The scaling of ε is still often used, because the interpolation matrix is a function of εh, and is constant, if ε is scaled proportionally to $\frac{1}{h}$, meaning that its condition number is constant as well. This behavior is similar to other RBFs that include a shape parameter. The Frobenius condition number of the interpolation matrix can be computed in (rather long) closed form, but we give its Taylor expansion instead:

$$\kappa_F = \frac{5}{2}\sqrt{\frac{13}{2}}\chi^{-4} - \frac{37}{\sqrt{26}}\chi^{-2} + \frac{75313}{780\sqrt{26}} - \frac{3343049}{25350\sqrt{26}}\chi^2 + O\left(\chi^4\right), \tag{2.2.66}$$

where $\chi = \varepsilon h$. We can see that the condition number grows with the 4th power as $\varepsilon \to 0$ with a constant h, or as $h \to 0$ with a constant ε. The condition number is also shown in Figure 2.1 for both the scaled and the non-scaled version.

This example is also a demonstration of the trade-off described in Section 1.4.3, since scaled parameter gives us stability without accuracy, and constant parameter gives accuracy, but is unstable.

One way to avoid the problems with space parameters is to use a RBF that does not have one – the polyharmonics $\phi(r) = r^{2k+1}$. These are only conditionally positive definite and need to be augmented with monomials of at least up to order k to ensure solvability. The properties of polyharmonic RBFs with monomial augmentation were investigated in PhD thesis by Barnett [Bar15].

Example 2.2.12 (Polyharmonic RBF-FD weights with monomial augmentation)**.** As a contrast to the previous example, we include another example of weights that can be computed in closed form. We use a regular 9-noded stencil $X = \{\{0,0\},\{0,-h\},\{0,h\},\{-h,0\},\{h,0\},\{-h,-h\},\{-h,h\},\{h,-h\},\{h,h\}\}$ and compute the weights for Laplacian around point $(0,0)$ using $\phi(r) = r^3$ and monomial augmentation of 2nd order.

Note that we can compute approximations using no or lower augmentation, but it turns out that the resulting approximation does not converge. For 2nd order augmenta-

tion, the interpolation matrix is

$$\begin{bmatrix}
0 & h^3 & h^3 & h^3 & h^3 & 2\sqrt{2}h^3 & 2\sqrt{2}h^3 & 2\sqrt{2}h^3 & 2\sqrt{2}h^3 & 1 & 0 & 0 & 0 & 0 & 0 \\
h^3 & 0 & 8h^3 & 2\sqrt{2}h^3 & 2\sqrt{2}h^3 & h^3 & 5\sqrt{5}h^3 & h^3 & 5\sqrt{5}h^3 & 1 & 0 & -h & 0 & 0 & h^2 \\
h^3 & 8h^3 & 0 & 2\sqrt{2}h^3 & 2\sqrt{2}h^3 & 5\sqrt{5}h^3 & h^3 & 5\sqrt{5}h^3 & h^3 & 1 & 0 & h & 0 & 0 & h^2 \\
h^3 & 2\sqrt{2}h^3 & 2\sqrt{2}h^3 & 0 & 8h^3 & h^3 & h^3 & 5\sqrt{5}h^3 & 5\sqrt{5}h^3 & 1 & -h & 0 & 0 & h^2 & 0 \\
h^3 & 2\sqrt{2}h^3 & 2\sqrt{2}h^3 & 8h^3 & 0 & 5\sqrt{5}h^3 & 5\sqrt{5}h^3 & h^3 & h^3 & 1 & h & 0 & 0 & h^2 & 0 \\
2\sqrt{2}h^3 & h^3 & 5\sqrt{5}h^3 & h^3 & 5\sqrt{5}h^3 & 0 & 8h^3 & 8h^3 & 16\sqrt{2}h^3 & 1 & -h & -h & h^2 & h^2 & h^2 \\
2\sqrt{2}h^3 & 5\sqrt{5}h^3 & h^3 & h^3 & 5\sqrt{5}h^3 & 8h^3 & 0 & 16\sqrt{2}h^3 & 8h^3 & 1 & -h & h & -h^2 & h^2 & h^2 \\
2\sqrt{2}h^3 & h^3 & 5\sqrt{5}h^3 & 5\sqrt{5}h^3 & h^3 & 8h^3 & 16\sqrt{2}h^3 & 0 & 8h^3 & 1 & h & -h & -h^2 & h^2 & h^2 \\
2\sqrt{2}h^3 & 5\sqrt{5}h^3 & h^3 & 5\sqrt{5}h^3 & h^3 & 16\sqrt{2}h^3 & 8h^3 & 8h^3 & 0 & 1 & h & h & h^2 & h^2 & h^2 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -h & h & -h & -h & h & h & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -h & h & 0 & 0 & -h & h & -h & h & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & h^2 & -h^2 & -h^2 & h^2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & h^2 & h^2 & h^2 & h^2 & h^2 & h^2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & h^2 & h^2 & 0 & 0 & h^2 & h^2 & h^2 & h^2 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{2.2.67}$$

and the right hand side is

$$\begin{bmatrix} 0 & 9h & 9h & 9h & 9h & 9\sqrt{2}h & 9\sqrt{2}h & 9\sqrt{2}h & 9\sqrt{2}h & 0 & 0 & 0 & 0 & 2 & 2 \end{bmatrix}^\mathsf{T}. \tag{2.2.68}$$

The block structure of the system is nicely visible in this example. Only first 9 elements of the solution vector are kept, and are equal to

$$\boldsymbol{w} = \begin{bmatrix} -\frac{8.971}{h^2}, \frac{3.330}{h^2}, \frac{3.330}{h^2}, \frac{3.330}{h^2}, \frac{3.330}{h^2}, -\frac{1.087}{h^2}, -\frac{1.087}{h^2}, -\frac{1.087}{h^2}, -\frac{1.087}{h^2} \end{bmatrix}^\mathsf{T}, \tag{2.2.69}$$

where the constants can be computed exactly, but have no special meaning. The weights sum to zero, and are consistent with up to 2nd order monomials by construction. Indeed, the error is of order h^2 with the constant proportional to a combination of 4th derivatives of u.

Ultimately, we decided to use polyharmonic splices $\phi(r) = r^3$ with monomials augmentation of the second order. This allows us to avoid convergence and stability issued that are present when using pure Gaussian (or other) RBFs, removes any shape parameter tuning and reproduces monomials up to order 2. The choice of power 3 was arbitrary, and does not have much effect on the overall performance [Fly+16]. While being more expensive to compute than WLS-based stencils ($O((n+\ell)^3)$ vs. $O(n\ell^2)$), this setup also exhibited better stability with respect to nodal positions than WLS-based methods, which is important for adaptivity.

2.3 PDE discretization

After obtaining the computational weights (in any of the ways described in the previous section), we are finally equipped to solve the PDE-governed problems. The procedure to actually obtain the solution of a PDE depends greatly on the type of the problem being solved. The two most common patterns are solving an elliptic boundary value problem, by assembling a sparse linear system, and solving an initial value problem with explicit time iteration.

Both of these patterns are described in the following sections and share the same setup: a domain Ω, with boundary $\partial\Omega = \Gamma_d \cup \Gamma_n$ which represents the parts of the boundary where Dirichlet and Neumann boundary conditions are applied, respectively.

Furthermore, we will use the following discretization of Ω, the construction of which will be discussed in greater detail in Chapter 3. The discretization consists of N nodes $X = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$ placed in the domain, of which some lie in the interior, some on the Neumann boundary and some on the Dirichlet boundary. Each node $\boldsymbol{x}_i \in X$ also has a corresponding stencil S_i of size $n_i \leq N$ consisting of neighboring nodes with indices $I_i = (I_{i,1}, \dots, I_{i,n_i})$. Thus, $I_{i,j}$ is the index of the j-th stencil node of i-th node and we additionally define $S_{i,j} = \boldsymbol{x}_{I_{i,j}}$ to denote the coordinates of this node.

With the above discretization, we can compute and store stencil weights $\boldsymbol{w}^{\mathcal{L},S_i}(\boldsymbol{x}_i)$ for all nodes $\boldsymbol{x}_i$ and operators $\mathcal{L}$ that appear in the problem. These weights, domain nodes and the stencil indices I_i are needed for PDE discretization.

2.3.1 Explicit evaluation

Consider a sample time-dependent initial value problem on domain Ω:

$$\frac{\partial u}{\partial t}(\boldsymbol{x}, t) = (\mathcal{L}u)(\boldsymbol{x}, t) \qquad \text{in } \Omega, \tag{2.3.1}$$
$$u(\boldsymbol{x}, t) = f(\boldsymbol{x}, t) \qquad \text{at } t = 0, \tag{2.3.2}$$
$$u(\boldsymbol{x}, t) = g_d(\boldsymbol{x}, t) \qquad \text{on } \Gamma_d, \tag{2.3.3}$$
$$\frac{\partial u}{\partial \vec{n}}(\boldsymbol{x}, t) = g_n(\boldsymbol{x}, t) \qquad \text{on } \Gamma_n, \tag{2.3.4}$$

where Γ_d and Γ_n are Dirichlet and Neumann boundaries, respectively, and f, g_d and g_n are known functions.

The spatial part of the PDE can be discretized into a system of ODEs by introducing $u_i(t) := u(\boldsymbol{x}_i, t)$ and replacing spatial operators with their discrete approximations as per (2.2.1), to obtain

$$\frac{\partial u_i}{\partial t}(t) = \sum_{j=1}^{n_i} (\boldsymbol{w}^{\mathcal{L},S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}}(t) \qquad \text{in } \Omega, \tag{2.3.5}$$
$$u_i(t) = f(\boldsymbol{x}_i, t) \qquad \text{at } t = 0, \tag{2.3.6}$$
$$u_i(t) = g_d(\boldsymbol{x}, t) \qquad \text{on } \Gamma_d \tag{2.3.7}$$
$$\sum_{\ell=1}^{d} (\vec{n})_\ell \sum_{j=1}^{n_i} (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}}(t) = g_n(\boldsymbol{x}, t) \qquad \text{on } \Gamma_n. \tag{2.3.8}$$

The notation $\partial_\ell := \frac{\partial}{\partial \boldsymbol{x}_\ell}$ was used to denote the ℓ-th coordinate derivative.

This scheme can then be further discretized in time to obtain a suitable stepping scheme. Using explicit Euler scheme in time, starting at $t = 0$ with time step Δt, we

define $u_i^k = u_i(k\Delta t)$. The iteration then proceeds as follows:

$$u_i^0 = f(\boldsymbol{x}_i), \text{ for all nodes } \boldsymbol{x}_i \tag{2.3.9}$$

$$u_i^{k+1} = u_i^k + \Delta t \left(\sum_{j=1}^{n_i} (\boldsymbol{w}^{\mathcal{L},S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}}(t) \right), \text{ for internal nodes } \boldsymbol{x}_i, \tag{2.3.10}$$

$$u_i^{k+1} = g_d(\boldsymbol{x}_i, (k+1)\Delta t), \text{ for Dirichlet nodes } \boldsymbol{x}_i, \tag{2.3.11}$$

$$u_i^{k+1} = \frac{g_n(\boldsymbol{x}_i, (k+1)\Delta t) - \sum_{j=2}^{n_i} u_{I_{i,j}}^k \sum_{\ell=1}^{d} (\vec{n})_\ell (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j}{\sum_{\ell=1}^{d} (\vec{n})_\ell (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_1}, \text{ for Neumann } \boldsymbol{x}_i. \tag{2.3.12}$$

The iteration on the Neumann boundary is obtained by expressing u_i from the discretized version (2.3.8) of the Neumann boundary conditions. Additionally, to express the iteration explicitly, we need to assume that each node $\boldsymbol{x}_i$ is a member of its own stencil. For the sake of simplicity, we can assume that it is the first stencil node, meaning that $S_{i,1} = \boldsymbol{x}_i$ and $I_{i,1} = i$. This allows us to express it out of the discretized Neumann conditions as follows:

$$\frac{\partial u}{\partial \vec{n}}(\boldsymbol{x}_i, t) = \sum_{\ell=1}^{d} (\vec{n})_\ell (\partial_\ell u)(\boldsymbol{x}_i) \approx \sum_{\ell=1}^{d} (\vec{n})_\ell \boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i)^\mathsf{T} u_{I(i)} \tag{2.3.13}$$

$$= \sum_{\ell=1}^{d} (\vec{n})_\ell \sum_{j=1}^{n_i} (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}} = \sum_{\ell=1}^{d} (\vec{n})_\ell \sum_{j=1}^{n_i} (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}} \tag{2.3.14}$$

$$= \sum_{j=1}^{n_i} u_{I_{i,j}} \sum_{\ell=1}^{d} (\vec{n})_\ell (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j = \tag{2.3.15}$$

$$= u_i \sum_{\ell=1}^{d} (\vec{n})_\ell (\boldsymbol{w}_{\partial_\ell,S_i}(\boldsymbol{x}_i))_1 + \sum_{j=2}^{n_i} u_{I_{i,j}} \sum_{\ell=1}^{d} (\vec{n})_\ell (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j. \tag{2.3.16}$$

The expression $u_{I(i)}$ represents a vector of function values in stencil nodes $u_{I(i)} = (u(\boldsymbol{x}_j))_{j \in I(i)}$.

Similar expressions to equations (2.3.9–2.3.12) can be developed for other time iterations schemes. Additionally, same techniques can be used to efficiently explicitly differentiate an already known field if needed.

2.3.2 Implicit solution

In this case, we consider a boundary value problem

$$\mathcal{L}u = f \qquad \text{in } \Omega, \tag{2.3.17}$$

$$u = g_d \qquad \text{on } \Gamma_d, \tag{2.3.18}$$

$$\frac{\partial u}{\partial \vec{n}} = g_n \qquad \text{on } \Gamma_n, \tag{2.3.19}$$

where, as before, Γ_d and Γ_n are Dirichlet and Neumann boundaries, and f, g_d and g_n are known functions. Similarly to before, each of the operators in above equations is approximated with stencil weights using (2.2.1). The unknown values $u_i := u(\boldsymbol{x}_i)$ are treated as unknown variables. This gives us the following system of equations:

$$\sum_{j=1}^{n_i} (\boldsymbol{w}^{\mathcal{L},S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}} = f(\boldsymbol{x}_i) \qquad \text{for internal nodes } \boldsymbol{x}_i, \tag{2.3.20}$$

$$u_i = g_d(\boldsymbol{x}_i) \qquad \text{for Dirichlet nodes } \boldsymbol{x}_i, \tag{2.3.21}$$

$$\sum_{\ell=1}^{d} (\vec{n})_\ell \sum_{j=1}^{n_i} (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}} = g_n(\boldsymbol{x}_i) \qquad \text{for Neumann nodes } \boldsymbol{x}_i. \tag{2.3.22}$$

This system of N linear equations can be written as $Mu = r$, where i-th row of the system corresponds to the equation that holds for node $\boldsymbol{x}_i$. The matrix M is called the *differentiation matrix*, because it contains the *differentiation weights*, i.e. the stencil weights of the differential operators in the problem. The $N \times N$ matrix M and the right-hand side r are given by

$$M_{i,I_{i,j}} = (\boldsymbol{w}_{\mathcal{L},\boldsymbol{x}_i})_j, \qquad r_i = f(\boldsymbol{x}_i), \qquad \text{for internal nodes } \boldsymbol{x}_i, \tag{2.3.23}$$

$$M_{i,i} = 1, \qquad r_i = g_d(\boldsymbol{x}_i), \qquad \text{for Dirichlet nodes } \boldsymbol{x}_i, \tag{2.3.24}$$

$$M_{i,I_{i,j}} = \sum_{\ell=1}^{d} (\vec{n})_\ell (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j, \qquad r_i = g_n(\boldsymbol{x}_i), \qquad \text{for Neumann nodes } \boldsymbol{x}_i, \tag{2.3.25}$$

where index j runs from 1 to n_i. The differentiation matrix M is sparse with at most $\sum_{i=1}^{N} n_i$ nonzero entries. The nodal values for Dirichlet nodes are explicitly known, and can in practice be substituted in the other equations and moved into the right hand side, which reduces the number of degrees of freedom by the number of nodes on the Dirichlet boundary.

The equations (2.3.23–2.3.25) define the unknown field u implicitly by using stencil weights and the solution of the system $Mu = r$ is the numerical approximation of the unknown function u. Similar approximations can be obtained for vector equations, or in implicit time stepping schemes.

2.3.3 Ghost nodes

Another technique that can be useful when applying boundary conditions is the use of *ghost* or *fictitious* nodes. This technique is often used in FDM to handle Neumann boundary conditions with symmetric differences, and it is used similarly in strong form meshless methods. The stencils near the boundary are usually one-sided, and to avoid that, one or more layers of nodes can be added outside of Ω. For a boundary node $\boldsymbol{x}_i$ with local spacing $h(\boldsymbol{x}_i)$ the ghost node is usually added in the outside normal direction at position $\boldsymbol{x}_i + h(\boldsymbol{x}_i)\vec{n}_i$, where $\vec{n}$ is the outside unit normal. If more than one layer is desired, further layers can be added at positions $\boldsymbol{x}_i + 2h(\boldsymbol{x}_i)\vec{n}_i$, $\boldsymbol{x}_i + 3h(\boldsymbol{x}_i)\vec{n}_i$, …

Additional nodes introduce additional degrees of freedom, which are filled by requiring that the PDE itself holds in the boundary node besides also requiring the boundary conditions. In case of a boundary value problem (2.3.17–2.3.19), besides requiring $\frac{\partial u}{\partial \vec{n}}(\boldsymbol{x}_i) = g_n(\boldsymbol{x}_i)$, we add another equation $(\mathcal{L}u)(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)$. If more than one

layer was added, then we can fill the additional degrees of freedom with equations $(\mathcal{L}\mathcal{L}u)(\boldsymbol{x}_i) = (\mathcal{L}f)(\boldsymbol{x}_i)$, $(\mathcal{L}\mathcal{L}\mathcal{L}u)(\boldsymbol{x}_i) = (\mathcal{L}\mathcal{L}f)(\boldsymbol{x}_i)$, …

While the ghost nodes in FDM help with the derivation of the equations and do not necessarily appear in the final equations, in strong form meshless methods they do in fact add additional degrees of freedom and we also obtain "function values" corresponding to these nodes. However, note that neither $\mathcal{L}$ for f are ever evaluated in the ghost nodes and the obtained function values only represent a possible extension of the solution u.

2.3.4 Special cases

Finite difference method

Similarly to how local stencil weights coincide with stencil weights for FDM, the whole strong-form solution procedure reduces to FDM if the same setup as in Proposition 2.2.6 is used. If the nodes are arranged in a rectangular grid, stencils of 9 closest nodes and tensor monomial basis in 2D give exactly the assumptions for Proposition 2.2.6, which means that every interior stencil is the same as the FDM stencil. The assembly of the differentiation matrices is exactly the same as in FDM, and so is the handling of the boundary conditions (with or without ghost nodes).

A direct FDM implementation is of course faster, since stencil weights can be computed analytically beforehand. However, the meshless setup has an advantage of being easy to generalize to higher order methods, where the manual computation and case-analysis of stencil weights near the boundary becomes tedious.

Kansa method

One of the first meshless methods for solving PDEs in strong form was the collocation method developed by Edward Kansa [Kan90b], who generalized the RBF-based methods for scattered data interpolation to solving PDEs.

To illustrate the method, we will describe how to solve a simple boundary value problem

$$\mathcal{L}u = f \qquad \text{in } \Omega, \tag{2.3.26}$$

$$u = g_d \qquad \text{on } \partial\Omega. \tag{2.3.27}$$

The method assumes that N collocation nodes $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ are present in the domain, and that the solution is of the form $\hat{u}(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|)$. This is substituted in the equation $(\mathcal{L}\hat{u})(\boldsymbol{x}) = f(\boldsymbol{x})$ for all internal nodes and into $\hat{u}(\boldsymbol{x}) = g_d(\boldsymbol{x})$ for all boundary nodes. Thus, assuming nodes 1 to I are in the interior and $I+1$ to N are on the boundary, we get the system

$$\begin{bmatrix} \mathcal{L}\phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_1\|) & \cdots & \mathcal{L}\phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_N\|) \\ \vdots & \ddots & \vdots \\ \mathcal{L}\phi(\|\boldsymbol{x}_I - \boldsymbol{x}_1\|) & \cdots & \mathcal{L}\phi(\|\boldsymbol{x}_I - \boldsymbol{x}_N\|) \\ \phi(\|\boldsymbol{x}_{I+1} - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_{I+1} - \boldsymbol{x}_N\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\boldsymbol{x}_N - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_N - \boldsymbol{x}_N\|) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix} = \begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_I) \\ g(\boldsymbol{x}_{I+1}) \\ \vdots \\ g(\boldsymbol{x}_N) \end{bmatrix}. \tag{2.3.28}$$

The solution of this system gives the coefficients $\boldsymbol{\alpha}$ which in turn give the solution $\hat{u}$.

RBF-FD method coincides with the Kansa method in an extreme case.

Proposition 2.3.1 (Kansa method as a special case of RBF-FD)**.** *If the stencil of each node includes all nodes, i.e. $S(i) = X$ for all $i = 1, \dots, N$, then RBF-FD is equivalent to the Kansa method.*

Proof. Instead of seeking the approximation in form $u(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|)$, we can seek the approximation from the same space using cardinal basis

$$\hat{u}(\boldsymbol{x}) = \sum_{i=1}^{N} u_i \boldsymbol{w}^{\mathrm{id},X}(\boldsymbol{x}), \tag{2.3.29}$$

where we used the fact that cardinal basis corresponds to stencil weights for $\mathcal{L} = \mathrm{id}$. Substituting $\hat{u}$ for u in the PDE and in the boundary conditions and evaluating at the collocation points, gives the RBF-FD differentiation matrix due to 2.2.4 and 2.2.3, the unknowns are the function values u_i and the right hand side remains the same. □

Of course, using Kansa method directly is more efficient than using RBF-FD with full stencils. The most expensive part of the Kansa method is the $O(N^3)$ solution of the dense system, while RBF-FD would require N computations of stencil weights, which cost $O(N^3)$ each, and an additional $O(N^3)$ for the system solution.

Chapter 3

Domain discretization

One of the main advantages of mesh-free methods is that they do not rely on polygonization of the computational domain to solve PDEs but instead approximate the solution using scattered computational nodes. Classical methods, such as FEM use computational nodes and connectivity relation as the basis of the discretization, and grid-based methods such as FDM can also be viewed as having an implicit rectangular mesh. Most meshless methods only use computational nodes and substitute connectivity relations with stencils of neighboring nodes. The discretizations used by these three types of methods are shown in Figure 3.1.

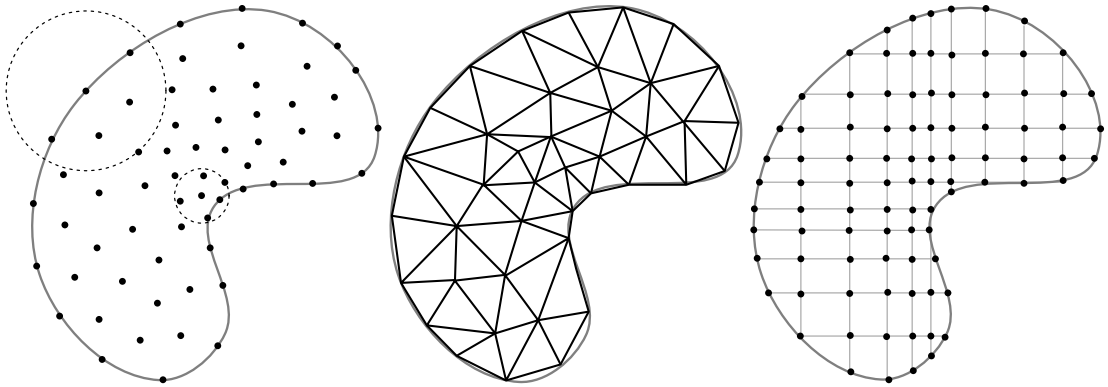


Figure 3.1: From left to right: meshless, FEM and FDM discretization. Only two stencils of closest 6 nodes are shown in meshless discretization for clarity.

Meshless methods discretize the domain by positioning N points, called *computational nodes*, or simply *nodes* in the domain and on its boundary. Each point is also assigned a set of neighboring nodes, as shown for a boundary and an interior node in Figure 3.1. The details of meshless discretizations and the terminology that is used differs on a method-by-method basis. Weak form methods, such as MLPG [AZ98], often use FEM terminology which is sometimes adapted to strong form methods as well [Ngu+08; TK15]. Thus the ball with a large enough radius to include all the neighboring nodes of a given node is called the *local subdomain* [AZ98], the *support domain* [TK15] or the *domain of influence* [Ngu+08]. The nodes themselves are often called *support nodes*. For strong form methods, FDM terminology is often transferred into meshless context, and neighboring nodes are often called *stencils*, despite not having the same shape for each

node and thus deviating from the original meaning. Some authors also call the neighboring nodes *local point clouds* [Oña+96] or *stars* [LO80] and the center node the *star node*, inspired by the star shape that is created if all neighboring nodes are connected to the center node. We will use the FDM terminology and use the term *stencil*.

The desired structure of domain discretization has been briefly described in Section 2.3. The next section gives the formal definition of a domain discretization used in this work and reviews state of the art algorithms for generating computational nodes. A new algorithm for generating nodes on the interior is presented in Section 3.2 and its generalization to surfaces, including domain boundaries, is presented in Section 3.3. These two algorithms together present a complete algorithm for generating computational nodes and will form the foundations of the adaptive solution procedure. Analyses of both algorithms and the generated node sets are described in Section 3.4. Finally, we discuss stencil selection algorithms in Section 3.5 to fully define how domain discretizations are constructed.

3.1 Basic definitions and state of the art

Consider a domain $\Omega \subseteq \mathbb{R}^d$, which we will assume is a connected open subset with a piecewise smooth boundary. The notation $[N]$ will be used to denote the set $\{1, \dots, N\}$.

Definition 3.1.1 (Domain discretization)**.** A (meshless, mesh-free) domain discretization or a (meshless, mesh-free) representation of Ω is a tuple $\mathcal{D} = (X, I, \tau, \vec{n})$, where

- X represents a node set $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$, such that all the nodes are contained in the closure of Ω, i.e. $\boldsymbol{x}_i \in \overline{\Omega}$. When working with nodes, it will be worthwhile to have them ordered, and thus we formally define X as an injective function $X\colon [N] \to \overline{\Omega}$, $X(i) = x_i$.

- I represents stencil indices. The stencil of i-th node has size n_i and the indices will be denoted as $\{I_{i,1}, \dots, I_{i,n_i}\}$. Formally, n is a function $n\colon [N] \to \mathbb{N}$ giving the stencil size and $I\colon \bigcup_{i=1}^{N} \{(i, j),\ j \in [n(i)]\} \to [N]$ is the function giving the stencil indices. We will write n_i and $I_{i,j}$ instead of $n(i)$ and $I(i, j)$.

  Furthermore, we will define the stencil function S as $S(i, j) = X(I(i, j))$, and similarly adopt the notation $S_{i,j}$ for j-th stencil node of the i-th node. Moreover, the one-argument versions $I_i = I(i) := (I_{i,1}, \dots, I_{i,n_i})$ and $S_i = S(i) := (S_{i,1}, \dots, S_{i,n_i})$ will also be used when referring to the tuple of all stencil indices or nodes, respectively.

- τ represents different node types. The two most common types are boundary and interior nodes, but these can be further refined into subgroups. We will allow each node to be assigned a whole number, formally making τ a function $\tau\colon [N] \to \mathbb{Z}$, with a general agreement that negative values of τ represent boundary nodes and positive values represent interior nodes. Thus, $X(\tau^{-1}(\mathbb{N})) \subseteq \Omega$ and $X(\tau^{-1}(-\mathbb{N})) \subseteq \partial\Omega$.

- For boundary nodes, we also require the outer unit normals $\vec{n}_i$ for boundary nodes $\boldsymbol{x}_i$ to be known. This is usually needed only on the Neumann boundary, but

if the domain is discretized without a particular problem in mind, the normals are assumed to be computed for all boundary nodes. Formally, $\vec{n}$ is a function $\vec{n}\colon \tau^{-1}(-\mathbb{N}) \to \mathbb{R}^d$, $\|\vec{n}_i\| = 1$, where we once again adopt the notation $\vec{n}_i = \vec{n}(i)$.

Sometimes it is convenient to refer to stencils directly from the node instead of its index. This is possible, because the nodes are distinct, and the left inverse X^{-1} exists. With slight abuse of notation we will write I instead of $I \circ X^{-1}$, and similarly, S instead of $S \circ X^{-1}$, for one-argument versions of I and S. This allows us to refer to node indices as $I(\boldsymbol{x}_i)$ and stencil nodes as $S(\boldsymbol{x}_i)$.

The Definition 3.1.1 does not account for ghost nodes. Ghost nodes can be thought of as members of X and can be contained as stencil nodes of other non-ghost nodes, but they themselves have neither stencils nor normals.

The construction of domain discretizations is often divided into three steps: generation of boundary nodes, generation of interior nodes and computation of stencils. The first to develop were the algorithms for interior node generation, as nodes in the interior are needed even for 2D examples, where boundary is one-dimensional.

3.1.1 Existing algorithms for interior node generation

In the beginning of the development of meshless methods positioning of computational nodes was considered simple, and the advantage that nodes can be scattered was sometimes stretched to the point that it was advertised that arbitrarily positioned nodes can be used (see [Liu02, p. 14]). This would make node generation seemingly trivial, but it soon turned out that such simplification leads to unstable results.

Without dedicated algorithms to generated scattered node sets, researchers often turned to existing mesh generators, with the idea to generate the mesh with desired properties and discard the connectivity information. This rationale was also explained by Liu [Liu02, p. 14]: “There are very few dedicated node generators available commercially; thus, we have to use preprocessors that have been developed for FEM.”. Such an approach has a few problems. Firstly, it is conceptually flawed, since one major point of using meshless methods is to avoid meshing and all the difficulties that come with it. Secondly, it is computationally wasteful, as we also compute the unneeded connectivity relations. Finally, some authors have reported that node layouts obtained by meshing yielded unstable operator approximations [SKF18], making them unable to obtain a solution.

Other approaches were researched, with the goal to cater to the strengths of meshless methods, such as simple generalizations to three and higher dimensions and the ability to use highly spatially variable node distributions. Some specialized algorithms for meshless node generation of layouts have been developed in recent years and most of them can generally be categorized into either mesh-based, refinement-based, iterative, advancing front or sphere-packing algorithms.

The downsides of mesh-based approaches have already been discussed. Their upside is that they are well developed, mature, easy to use and familiar. Most popular mesh generation algorithms (e.g. algorithms in the computational geometry algorithms library CGAL[1] [FP09]) also cannot be easily altered to only generate the nodes, as mesh

[1] https://doc.cgal.org/latest/Mesh_3/index.html

generation is tied together with generation of new nodes. Additionally, mesh-based approaches also include filling the domain with points on a rectangular, hexagonal or any other repeating grid pattern, which causes trouble with spatially variable density or irregular domains. Such approaches can be useful on a case-per-case basis, but not in general.

Refinement based algorithms start from a small node set and adds nodes to produce a larger one, repeating until some criterion is satisfied. This idea is commonly used in mesh-generation as well. An example of this are the 2D Ruppert's algorithm [Rup95] and its generalization to 3D [She98], which both use Delaunay refinement. This approach is not directly generalizable to meshless node generation, but others are. For example, Voronoi diagram related techniques are used by Mu et al. [DGJ02] and a quadtree refinement-based mesh generator [FG08] has been adapted by Zamolo and Nobile [ZN18] into a variable density 2D meshless node generator.

Existing node sets can be improved using iterative approaches. A common approach is to move the nodes locally by simulating free charged particles, which results in a type of node configuration called minimal energy nodes [HS04]. Other approaches include Voronoi relaxation [BSD09], bubble simulation [Liu+10] or a combination of above [CK99]. Iterative methods require an initial distribution and thus do not really count as algorithms for node generation. The initial distribution can be chosen to be random one of the simple grid-based node sets, but the number of iterations to obtain a good node set can be large and the overall procedure can be very expensive. A user is thus often required to consider a trade-off between node quality (i.e. number of iterations) and execution time. Furthermore, not many of the iterative methods are designed to handle spatially variable densities.

One approach to node generation are the circle or sphere packing methods [LTU00b], which generate densely packed high quality node distributions, at the expense of computational time. However, such methods are not only useful for domain discretization, but have uses in the graphics community as well. Poisson disk sampling, a supersampling method used also in image anti-aliasing [Coo86] has been repurposed as a node generation algorithm by Shankar et al. [SKF18] in 2018. Poisson disk sampling ensures a required minimal inter-point distance and has efficient implementation in arbitrary dimension due to Bridson [Bri07]. It has been successfully used with high order RBF-FD approximations, but does not support variable nodal spacing.

Another approach to node generation are the advancing front methods. These methods begin node generation in a certain location (e.g. the domain boundary) and advance towards non-discretized parts of the domain, generating nodes along the way. These methods are also common approaches for mesh generation but are often restricted to two dimensions [PS04], with some 3D techniques also available [LP88], that can be generalized to produce point sets as well [LO98]. Drumm, Tiwari and Kuhnert [Dru+08] also briefly describe an background-grid advancing front-based technique, which is similar to Poisson disk sampling, but with applying additional corrections after the node set has been generated, to remove nodes that are too close and additionally fill too large gaps. One of the first papers dedicated to algorithms for generating node sets suitable for strong-form meshless methods was published in 2015 by Fornberg and Flyer [FF15b] and included an advancing font based methods. The method is inspired by the process of dropping variable-sized grains into a bucket and yields quality variable density node

distributions in 2D. It is also reasonably computationally efficient in practice and has since been improved to run in log-linear time [SK18a].

Historically, the algorithm by Fornberg and Flyer was one of the first to outline the need for dedicated meshless node generation, and solved the problem fairly efficiently in 2D. The algorithm based on Poisson disk sampling with Bridson's implementation as used by Shankar was the next in historic succession and supports generation of uniformly spaced node distribution in arbitrary dimensions. Also in 2018, Zamolo and Nobile [ZN18] published two new algorithms for 2D variable density node generation. A pre-print uploaded to arXiv in late 2018 by Slak and Kosec presented a new efficient algorithm for node generation, which is able to generate variably spaced distributions in arbitrary dimensions. The paper was published in 2019 [SK19d]. In the meantime, a pre-print authored by van der Sande and Fornberg describing a generalization of [FF15b] that also supports generating variable density node distributions in arbitrary dimensions was uploaded to arXiv [SF19], but has not been published as of writing this. Research in related areas, such as parallel implementations and node generation on surfaces has also a lot of activity lately. This is reviewed at the beginning of Section 3.3.

3.1.2 Requirements for node generation algorithms

As partially discussed in the previous section there exist requirements for algorithms and for the generated node sets that make them suitable for strong-form meshless discretizations. They were often discussed implicitly, and have since been explicitly stated in [SK19d]. A very similar list is given below along with the rationale for the property and how existing algorithms behave. We will be using the definitions of fill distance, separation distance and quasi-uniformity introduced in Section 1.4.1 to state some of the requirements formally. The requirements are loosely ordered by decreasing importance.

1. *Domain coverage* The nodes should cover the whole domain, and not leave any large areas without nodes. Formally, this means that the fill distance $h_{X,\Omega}$ should tend towards zero if the desired maximal nodal spacing tends toward zero as well. This statement is vacuous when considering a single distribution, but even for a single distribution $h_{X,\Omega}$ should not be grater then the maximal desired spacing by a large factor.

2. *Minimal spacing guarantees.* Too closely positioned nodes can severely impact stability. Thus, provable minimal spacing guarantees are desirable. The simplest guarantee for constant spacing h, is of the form

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \geq h, \quad i \neq j. \tag{3.1.1}$$

   For variable nodal spacing, lower bound for the distance $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ is still desired. It is best if the bound is local, and that can be obtained in several different ways, as discussed later, in Section 3.2.3.

3. *Local regularity.* Nodal distributions should be locally regular, i.e. the distances between a node and its nearest neighbors should be approximately equal. This

requirement is based on the fact that local strong form meshless methods are often sensitive to nodal irregularities, such as large discrepancies in distances to the nearest neighbors. Such irregularities can can cause ill-conditioned approximations, making the distribution inappropriate for solving PDEs.

This requirement together with the requirement for domain coverage and minimal spacing is best formally expressed as quasi-uniformity for nodal distributions where uniform spacing is desired. For distributions with variable spacing, a generalization of quasi-uniformity is defined in Section 3.4.1 that captures the same notion locally.

Besides the formal definition of quasi-uniformity, other definitions of local regularity exist, as introduced e.g. in [LTU00a]. However, in practice local regularity means that the node sets produced by the algorithm should yield quality PDE solutions when using local strong form methods.

4. *Spatially variable densities.* The algorithm should be able to produce node sets with spatially variable density while conforming to the previous points, as spatially variable nodal distributions are often required when dealing with irregular domains or adaptivity. The desired nodal spacing can be assumed to be given as a function $h\colon \mathbb{R}^d \to (0, \infty)$. The changes in spatial density should be gradual and relatively smooth in order to satisfy the requirement of local regularity. The algorithm should work without any continuity assumptions for reasonable h (see remarks in 3.2.2) and should see a constant step h as a special case of variable step $h(\boldsymbol{x})$, not the other way around. Additional assumptions on h may be required to prove additional properties of the algorithm (e.g. its commonly used that h is Lipschitz).

5. *Computational efficiency and scalability.* Time complexity of the algorithm should ideally be linear in number of generated nodes an not directly dependent on Ω and h. The choice of Ω and h obviously affects the expected number of nodes in the domain, but the time complexity should be the same regardless of which combination of h and Ω produced this many nodes.

   Time complexity that is similar to linear in practice, e.g. $O(N \log N)$ is acceptable, while time complexity that is $\Omega(N^\alpha)$, for $\alpha > 1$, is undesirable. The algorithm should also be computationally efficient in practice, making it feasible to use as a node generation algorithm in an adaptive setting. A reasonable goal on today's laptop architectures is to generate 10^6 nodes per second (in a compiled language, such as C, C++ or Fortran).

6. *Compatibility between discretizations of interior or boundary.* Boundary and interior discretizations are often constructed separately, or the boundary or interior discretization is known beforehand. It is assumed that such discretizations are already conforming to spacing h. In that case the generated node set should seamlessly join with the existing (boundary) discretization to form a complete discretization of Ω. This helps to prevent problems often encountered when enforcing boundary conditions (see [SKF18, sec. 3.5] and references therein).

7. *Compatibility with irregular domains.* The algorithms should handle irregular domains (i.e. not only regular shapes such as rectangles and circles). It can assume that a characteristic function

$$\chi_\Omega \colon \mathbb{R}^d \to \{0, 1\},$$
$$\chi_\Omega(p) = \begin{cases} 1, & p \in \Omega, \\ 0, & p \notin \Omega \end{cases} \tag{3.1.2}$$

   is known, but can use other properties of Ω, as long as the performance is independent of the particular shape, as discussed in point 5.

   Algorithms that fill bounding boxes of domains are are seen as impaired in this aspect. Desirably, as the volume of Ω decreases, so should the number of operations required by the algorithm.

8. *Dimension independence.* The same algorithm should work in all (low) dimensions d without special cases. Furthermore, it should ideally also share the same general implementation in all dimensions.

9. *Direction independence.* The properties of the produced distributions and running time of the algorithm should be independent of the spatial orientation of the domain Ω or the coordinate system used.

10. *Few free parameters.* The number of free or tuning parameters of the algorithm should be minimal. It should work well for all domains and density functions, without any user intervention, making the algorithm robust and work "out of the box". Any potential free parameters should be well understood with recommended default values and explained effects.

11. *Simplicity.* Algorithms with simpler formulations and implementations are preferred.

The algorithm proposed in [SK19d] was developed to support efficient generation of quality variable density distributions in arbitrary domains and dimensions, which was not possible with the current algorithms. The satisfaction of these requirements for the proposed algorithms and some of the existing algorithms are discussed in Section 3.4.

3.2 Node generation in domain interiors

The proposed node generation algorithm is similar to Poisson disk sampling [Coo86], a type of stochastic sampling, which is used in graphical applications to generate points that are uniformly distributed over a given region. Poisson disk sampling chooses samples in a domain Ω with the property that no two samples can be closer than a specified distance h. This is done in a random fashion, by "dart throwing", i.e. uniformly selecting a point $\boldsymbol{p} \in \Omega$ and accepting it, if it is not too close to already accepted candidates. The naive approach described above has time complexity of $O(N^2)$ for N generated points, but an implementation by Bridson [Bri07] generates such distributions in $O(N)$

time in box-shaped domains and was successfully used as node generation algorithm by Shankar et al. [SKF18].

Node sets for strong form mesh-free discretizations do not need the stochastic properties that Poisson disk sampling possesses. Instead, node distributions with variable spacing are often desired. The Poisson disk sampling algorithms was adjusted to fit these new objectives. The new algorithm keeps the same idea of generating new candidates from existing nodes in an advancing front fashion, but adjusts the advancing to the domain shape and desired spacing function h, while decreasing the effect of randomness.

3.2.1 Algorithm

The algorithm takes as input a region Ω, given by a characteristic function $\chi_\Omega \colon \Omega \subseteq \mathbb{R}^d \to \{0, 1\}$, a nodal spacing function $h \colon \Omega \subset \mathbb{R}^d \to (0, \infty)$ and a list of starting nodes X, called "seed nodes". These nodes are put in a queue, and will be used to generate new nodes. During the course of the algorithm, each node can be either *expanded*, *active* or a *candidate*. Active nodes are the ones currently in the queue. Expanded nodes are those that have already been processed and dequeued. Candidates are nodes that have been newly created and not yet enqueued. Both expanded and active nodes are *accepted* and will appear in the resulting node set.

A spatial search structure S that supports insertions and nearest neighbor queries is build on X. The algorithm processes points in a fasion similar to a breath-first search. In i-th iteration of the algorithm, a node $\boldsymbol{p}_i$ is dequeued. A set of candidates $\{c_{i,j}\}_j$ is generated around this node, in such a way that the candidates are spaced approximately uniformly on the sphere with radius $h(\boldsymbol{p}_i)$ and center $\boldsymbol{p}_i$. The candidates are processed in order. If a candidate $\boldsymbol{c}_{i,j}$ lies outside of Ω, it is immediately rejected. It it lies in Ω, then its nearest neighbor $\boldsymbol{n}_{i,j}$ among already accepted nodes is found using S. If the distance between $\boldsymbol{c}_{i,j}$ and $\boldsymbol{n}_{i,j}$ is lower than $h(\boldsymbol{p}_i)$, then $\boldsymbol{c}_{i,j}$ is rejected, otherwise it is accepted. An accepted candidate is inserted into the queue and in S. When there are no more nodes left in the queue, the set of nodes in S is returned as the result. The illustration of the run time progress of the algorithm is shown in Figure 3.2 and the pseudocode of the algorithm is listed as Algorithm 3.1. The algorithms uses an implicit queue in the initial list X.

An important part of the algorithm is the candidate generation. Candidates are obtained by choosing a set of unit vectors $\vec{s}_{i,j}$ that are distributed in all directions, and define $\boldsymbol{c}_{i,j} = \boldsymbol{p}_i + \vec{s}_{i,j} h(\boldsymbol{p}_i)$. Thus we have reduced the problem of candidate generation to an approximately uniform distribution of points on a unit sphere. One option is to select a number of randomly uniformly distributed points each time. Another is to have a preset fixed discretization of a sphere. Yet another is to have a fixed pattern, that is slightly randomized each time. Figure 3.3 shows an example of these methods. Fixed candidates method is the faster, because it can be precomputed, but can produce repeating patterns, gaps and other artifacts in the node set, which are not desired in general. The node set produced by using random candidates exhibits no random structure, but produces more loosely packed nodes. The randomized option is faster at candidate generation and covers all directions more uniformly, while still avoiding regular patterns in the resulting node set. Further analysis of different candidate generation schemes is reported in [SK19d]. The final chosen candidate generation method is the random-

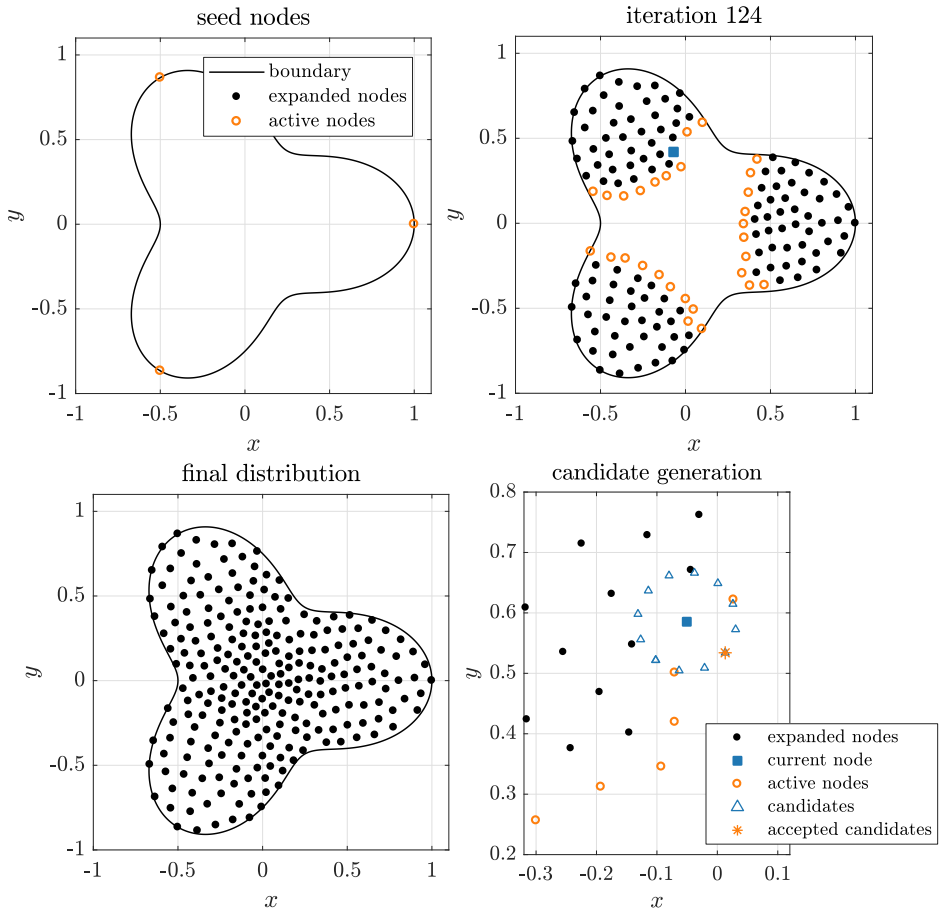


Figure 3.2: Progress of the node generation algorithm on a region defined by a polar curve $r(\vartheta) = \frac{1}{4}(3 + \cos(3\vartheta))$. The seed nodes lie on the boundary at ϑ values of $0, 2\pi/3$ and $4\pi/3$. The nodal spacing function is equal to $h(\boldsymbol{x}) = 0.05(1 + \|\boldsymbol{x}\|_1)$.

ized option, which is implemented as Algorithm 3.2 in d dimensions using generalized spherical coordinates.

Remark 3.2.1 (Numerical precision)**.** When the algorithm checks that a candidate node is not too close to any existing nodes on line 11, the comparison should be done with some care. If a candidate $\boldsymbol{c}$ was generated from $\boldsymbol{p}$ at the distance $h(\boldsymbol{p})$, it often happens that the closest node $\boldsymbol{n}$ to $\boldsymbol{c}$ is $\boldsymbol{p}$ itself at exactly distance $h(\boldsymbol{p})$. However, when comparing this numerically, the check $\|\boldsymbol{c} - \boldsymbol{n}\| \geq h(\boldsymbol{p})$ should be substituted for

$$\|\boldsymbol{c} - \boldsymbol{n}\| \geq (1 - \epsilon)h(\boldsymbol{p}), \tag{3.2.1}$$

for some small positive ϵ, to avoid $\boldsymbol{c}$ from being rejected as being too close to $\boldsymbol{p}$ due to numerical errors.

Remark 3.2.2 (Points on the boundary)**.** Algorithm 3.1 does not generate points on the domain boundaries. However, as seen in Figure 3.2, some of the generated points can be placed near the domain boundary. Joining this discretization with an already existing boundary discretization can be troublesome, as simply superimposing both discretization can lead to certain points being too close together. A generic solution in

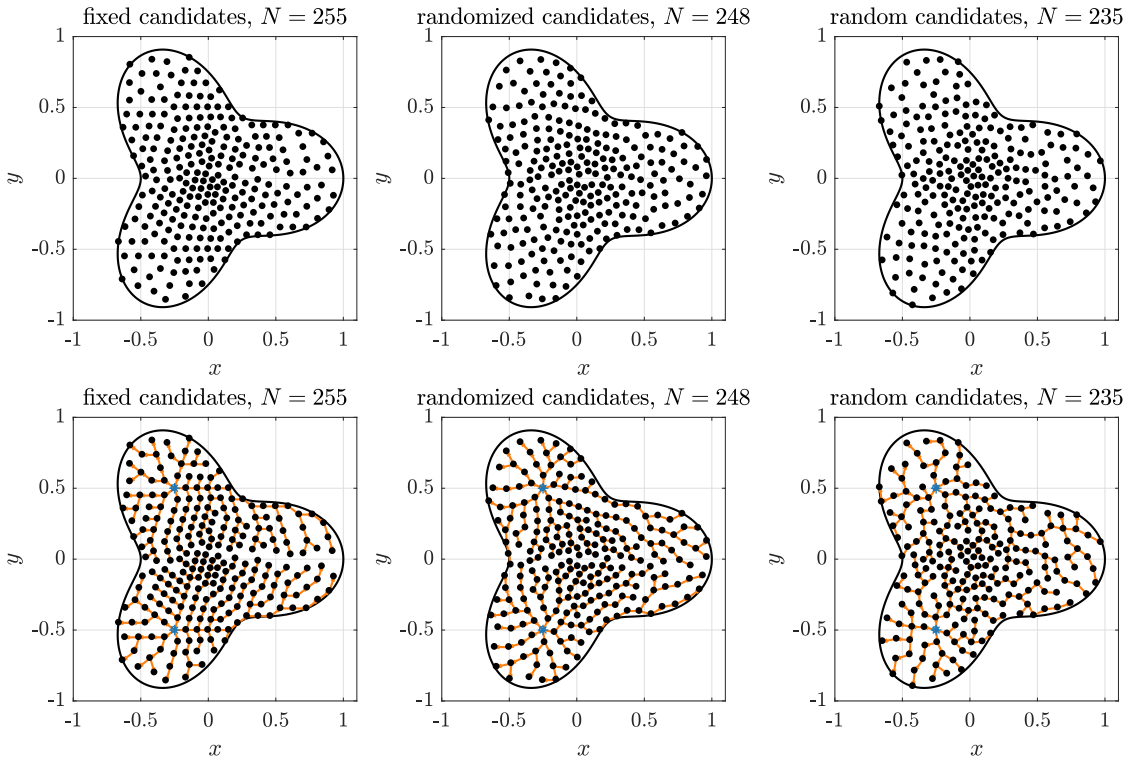


Figure 3.3: Different options for candidate generation displayed on the same domain as in Figure 3.2, but with seed nodes $X = \{(-\frac{1}{4}, -\frac{1}{2}), (-\frac{1}{4}, \frac{1}{2})\}$. Three candidate generation methods are fixed, where 10 candidate angles are equispaced on $[0, 2\pi)$, randomized, where the fixed angles are added a uniformly random shift from $[0, 2\pi)$, and finally, random, where 10 angles are picked uniformly at random from $[0, 2\pi)$. The bottom row shows an arrow from each node to all the candidates that it generated, with seed nodes marked with stars.

this case is to remove all the points of the interior discretization that are close to any boundary points (e.g. for each boundary point p, removing any interior points in radius $h(p)$), as presented in [FF15b; SKF18]. Blindly removing the points might cause gaps to arise between the boundary and interior, which can then be mitigated by post-processing techniques, such as locally performing a few iterations of a charged particle simulation [FF15b].

However, in our case, a simpler solution exists. If the boundary discretization is generated first, or known beforehand, it can be used as a list of seed nodes for the interior node generation algorithm. Alternatively, if we do not want to use all of the nodes of the boundary discretization as the seed nodes, but still want the algorithm to not place nodes too close to them, the boundary nodes can be pre-inserted into the spatial search data structure. This way, they will be taken into account when placing new nodes, without counting as seed nodes.

3.2.2 Time and space complexity

It is the easiest to state the time and space complexity in an output-sensitive manner, depending on the number of nodes N. We will also denote with $P_S(N)$ the precompu-

Algorithm 3.1 Node generation in domain interiors.

Input: A set Ω, given by a characteristic function $\chi_\Omega \colon \Omega \subseteq \mathbb{R}^d \to \{0, 1\}$.
Input: A nodal spacing function $h \colon \Omega \subset \mathbb{R}^d \to (0, \infty)$.
Input: A nonempty list of starting nodes $X \subseteq \Omega$.
Input: Maximal number of nodes generated $N_{\max} \geq 0$.
Input: A whole number $k > 0$, influencing the number of generated candidates.
Output: A list of nodes in Ω distributed according to spacing function h.

function FILL(Ω, h, X, $N_{\max}$, k **default** 10)
  $S \leftarrow$ INIT(X) ▷ Initialize spatial search structure on points X.
  $i \leftarrow 0$ ▷ Current node index.
  while $i < |X|$ **and** $i < N_{\max}$ **do** ▷ Until the queue is not empty.
    $\boldsymbol{p}_i \leftarrow X[i]$ ▷ Dequeue current point.
    $h_i \leftarrow h(p_i)$ ▷ Compute its nodal spacing.
    for each $\vec{s}_{i,j}$ **in** CANDIDATES(d, k) **do** ▷ Loop through directions.
      $\boldsymbol{c}_{i,j} \leftarrow \boldsymbol{p}_i + h_i \vec{s}_{i,j}$ ▷ Generate a new candidate.
      if $\boldsymbol{c}_{i,j} \in \Omega$ **then** ▷ Discard candidates outside the domain.
        $\boldsymbol{n}_{i,j} \leftarrow$ FINDNEAREST($S, \boldsymbol{c}_{i,j}$) ▷ Find nearest node for proximity test.
        if $\|\boldsymbol{c}_{i,j} - \boldsymbol{n}_{i,j}\| \geq h_i$ **then** ▷ Test that $\boldsymbol{c}_{i,j}$ is not too close to other nodes.
          APPEND($X, \boldsymbol{c}_{i,j}$) ▷ Enqueue $\boldsymbol{c}_{i,j}$ as the last element of X.
          INSERT($S, \boldsymbol{c}_{i,j}$) ▷ Insert $\boldsymbol{c}_{i,j}$ into the spatial search structure.
        end if
      end if
    end for
    $i \leftarrow i + 1$ ▷ Move to the next non-expanded node.
  end while
  return X
end function

tation/initialization time used by the data structure S on N nodes, $Q_S(N)$ is the time spent on a radius query and $I_S(N)$ is the time spent for element insertion. Additionally, the time complexity of the evaluation of χ_Ω is denoted by T_Ω, and the time complexity of evaluation of h with T_h.

Let us denote the number of seed nodes in X with $N_s = |X|$. Initialization of the search structure takes $I(N_s)$ time, and initialization of other variables takes $O(1)$ time.

The number of iterations of the main loop is equal to the number of generated points, denoted by N. This is also the number of times h is evaluated. Denote the number of candidates generated in each iteration with n_c. In the worst case, we perform n_c positive evaluations of χ_Ω, costing $n_c T_\Omega$ time, n_c queries on the search structure with at most N nodes and n_c insertions in the search structure with at most N nodes. While the number of evaluations of χ_Ω and queries might be close to the $n_c N$ bound, this is too pessimistic for insertions. While one single candidate loop can cause multiple insertions, there can be at most $N - N_s$ in total. All other operations are (amortized) constant.

The total time complexity of the algorithm is therefore equal to

$$T_S(N) = I_S(N_s) + O(1) + N(T_h + n_c(T_\Omega + Q_S(N) + O(1))) + N I_S(N) \tag{3.2.2}$$
$$= I_S(N) + O(N(T_h + n_c(T_\Omega + Q_S(N)))) + N I_S(N) \tag{3.2.3}$$

Many search structures, usually tree-based, that support desired behavior in logarith-

Algorithm 3.2 Randomized candidate generation.

Input: Spatial dimension $d \in \mathbb{N}, d > 0$.
Input: Number of candidates $k \in \mathbb{Z}, k \geq 0$.
Input: Radius of the sphere $r \in (0, \infty)$.
Output: A set of points (position vectors) on a d-dimensional sphere with radius r.

```
1: function CANDIDATES(d, k, r default 1)
2:     if d = 1 then                                   ▷ Base case for recursion.
3:         return {−r, r}
4:     end if
5:     Δφ ← 2π/k
6:     offset ← RANDOM(0, π)
7:     C ← ∅
8:     for i ← 0 to ⌊k/2⌋ − 1 do   ▷ Discretize d − 1 dimensional spherical slices at angles φ_i.
9:         φ_i ← offset + iΔφ
10:        if φ_i ≥ π then                              ▷ Loop back into [0, π).
11:            φ_i ← φ_i − π
12:        end if
13:        r_i ← r sin(φ_i)                             ▷ The slice has a smaller radius.
14:        k_i ← ⌈k sin(φ_i)⌉          ▷ The slice has less nodes to maintain constant density.
15:        C_i ← CANDIDATES(d − 1, r_i, k_i)            ▷ Discretize recursively.
16:        C ← C ∪ {(r cos φ_i, c) for c ∈ C_i}         ▷ Add one coordinate to the front.
17:    end for
18:    return C
19: end function
```

mic time exist. Among the most common are ball trees [Omo89], k-d trees [Moo91] and cover trees [BKL06], with a comparison of this and many other techniques for nearest neighbor searching given by Kibriya and Frank [KF07]. In our general implementation we use a k-d tree, and in this case the time complexity is

$$T_{k\text{-d tree}} = O(NT_h + Nn_c(T_\Omega + \log N)) \tag{3.2.4}$$

If nodal spacing h is assumed to not vary too much, the algorithm could be sped up by using a uniform-grid based spatial search structure, usually with spacing $h/\sqrt{d}$, so that there is at most one point per grid cell, similar to Bridson in his implementation of Poisson disk sampling [Bri07]. When constructed on the (oriented) bounding box $\operatorname{obb}\Omega$, the time complexity of its allocation and initialization is proportional to the number of cells, which leads to time complexity

$$O\left(\frac{|\operatorname{obb}\Omega|}{(h/\sqrt{d})^d}\right) = O\left(\frac{|\operatorname{obb}\Omega|}{|\Omega|}N\right), \tag{3.2.5}$$

using the fact that for constant h the number of nodes is $N = \Theta(|\Omega|/h^d)$.

The subsequent insertions and queries in the grid are all $O(1)$, thus improving the time complexity of the algorithm for constant h to

$$T_{\text{grid}} = O\left(\frac{|\operatorname{obb}\Omega|}{|\Omega|}N + NT_h + Nn_cT_\Omega\right). \tag{3.2.6}$$

Furthermore, the factor $\frac{|\operatorname{obb}\Omega|}{|\Omega|}$ can be eliminated by using a hash map of cells instead of a grid; however, the practical benefit of that approach shows only with very irregular domains.

Using a background cell structure is feasible even with moderately variable h, by allowing more than one point per cell. For even higher variability, hierarchical cell grids could be used, eventually degrading back into tree-like structures.

Most often, the number of candidates n_c is a relatively small dimension dependent constant, like 10 in 2D or 30 in 3D, and can be omitted in O-notation. The time complexity of T_Ω varies greatly with representation of Ω, but is often constant or logarithmic (e.g. for balls, boxes, polygons). In many practical cases, when h is not too variable and Ω not too irregular, the algorithm thus runs in linear time with a simple background grid structure. Otherwise, it can be implemented to always run in log-linear time, independent of Ω and h.

The spatial complexity is simple to deduce. The algorithm uses additional $O(N)$ space to generate the output set, $O(n_c)$ for candidate generation and additional memory used by the space structure, usually another $O(N)$ or $O(N \log N)$.

Remarks on the finiteness

Algorithm 3.1 has a parameter $N_{\max}$ which limits the maximal number of nodes generated and is therefore forced to finish in finitely many steps. In practice this is usually set to a higher number than the expected number of nodes, and acts as a fail-safe to have a bounded running time and memory usage. Without it, the algorithm does not necessarily finish if Ω and h are chosen badly.

Example 3.2.3 (Infinite node generation)**.** The algorithms naturally runs forever if Ω is unbounded. However, even if it is bounded and h is positive, the algorithm does not necessarily finish.

If we choose $\Omega = (0, 1)$, $h(x) = \frac{1-x}{2}$, set the seed node to be $x = 0$, and run the algorithm, we obtain a sequence of nodes $x_1 = \frac{1}{2}$, $x_2 = \frac{1}{2} + \frac{1}{4}$, ... The general recurrence relation is

$$x_{n+1} = x_n + h(x_n) \tag{3.2.7}$$

and the general solution for this case is $x_n = 1 - 2^{-n}$. Thus, all nodes are contained in Ω and the algorithm continues forever (at least in theory).

Such a thing cannot happen if h is constant. Every new node contains no previously generated nodes in the ball with radius $h/2$. The volume covered by these balls grows by a constant number (the volume of this sphere) each time a new node is added. Due to boundedness of Ω, this must stop eventually. Furthermore, the algorithm is also finite if h is variable but is bounded away from zero, i.e. $h(\boldsymbol{x}) > q > 0$, using the same argument as before with q instead of h.

3.2.3 Minimal spacing requirements

For constant spacing h, it is easy to state and prove the minimal spacing requirements for the algorithm. For any distinct nodes $\boldsymbol{x}_i, \boldsymbol{x}_j \in X$, such that at least one of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is not a seed node, we have

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \geq h, \tag{3.2.8}$$

as the check on line 11 of Algorithm 3.1 ensures that (3.2.8) holds up to numerical precision. The reason that at least one of the nodes must not be a seed node is that there are no restrictions on the user input. However, if only one seed node is supplied, the bound will hold for all nodes, otherwise we assume that the user has to take care that the seed nodes are not positioned too closely together.

For spatially variable h, the same argument gives a global bound

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \geq \min_{\boldsymbol{x}\in\Omega} h(\boldsymbol{x}) \tag{3.2.9}$$

for $i \neq j$, but it can be very coarse. Mitchell et al. [Mit+12] define more precise, local bounds when considering spatially variable node sets. The *empty disk property* for a sequence of nodes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ is satisfied if

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \geq f(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{3.2.10}$$

for $1 \leq i < j \leq N$, for some function f, which is evaluated at a previously accepted node $\boldsymbol{x}_i$ and a new candidate $\boldsymbol{x}_j$. Four choices of f were proposed, based on which point's spacing is taken into account when positioning new candidates:

- *Prior-disks:* $f(\boldsymbol{x}_i, \boldsymbol{x}_j) = h(\boldsymbol{x}_i)$,
- *Current-disks:* $f(\boldsymbol{x}_i, \boldsymbol{x}_j) = h(\boldsymbol{x}_j)$,
- *Bigger-disks:* $f(\boldsymbol{x}_i, \boldsymbol{x}_j) = \max\{h(\boldsymbol{x}_i), h(\boldsymbol{x}_j)\}$,
- *Smaller-disks:* $f(\boldsymbol{x}_i, \boldsymbol{x}_j) = \min\{h(\boldsymbol{x}_i), h(\boldsymbol{x}_j)\}$.

By changing the comparison on the line 11 of Algorithm 3.1, we could satisfy any of these properties, but it could cause the algorithm to terminate prematurely in some cases, if e.g. $h(\boldsymbol{x}_j) > h(\boldsymbol{x}_i)$.

The proposed node placing algorithm as stated satisfies neither of this variations, but the property is instead established in the following proposition.

Proposition 3.2.4. *Let the nodes $\boldsymbol{x}_i$, $i = 1, \ldots, N$, be generated by Algorithm 3.1, where first N_b nodes were given as seed nodes. Then*

$$\|\boldsymbol{x}_k - \boldsymbol{x}_j\| \geq h(\boldsymbol{x}_{\beta(j)}) \tag{3.2.11}$$

holds for all $1 \leq k < j \leq N$, $j > N_b$, where β represents the predecessor function.

Proof. The algorithm begins with N_b seed nodes, and each new candidate is generated from a unique existing node, thus giving rise to a predecessor-successor relation. This relation is illustrated in the bottom row of Figure 3.3. We can define the predecessor function $\beta\colon \{N_b + 1, \ldots, N\} \to \{1, \ldots N\}$ for an accepted candidate $\boldsymbol{x}_j$ that was generated from $\boldsymbol{x}_i$ as $\beta(j) = i$. Seed nodes have no predecessor defined.

A node $\boldsymbol{x}_j$, generated from $\boldsymbol{x}_i$, is at a distance $h(\boldsymbol{x}_i)$ from $\boldsymbol{x}_i$, thus satisfying the equality

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = h(\boldsymbol{x}_i) = h(\boldsymbol{x}_{\beta(j)}), \tag{3.2.12}$$

the so called *prior-disks* property for predecessor-successor pairs. The condition for node acceptance is that the distance to already accepted nodes is not larger than $h(\boldsymbol{x}_i)$. This means that for all $k < j$, we have

$$\|\boldsymbol{x}_k - \boldsymbol{x}_j\| \geq h(\boldsymbol{x}_i) = h(\boldsymbol{x}_{\beta(j)}), \tag{3.2.13}$$

establishing the desired property. □

3.3 Node generation on parametric surfaces

The counterpart to generating nodes in domain interiors is to generate nodes on domain boundaries. Nodes on the boundary are necessary to solve boundary-value problems but they are often easier to generate ad-hoc, especially for 2D domains. For specific boundaries of 3D domains, such as spheres or tori, specialized algorithms for point placement can also be used, and quite a few might exists. Hardin et al. offer a recent review of popular point configurations on the sphere [HMS16].

To obtain a general algorithm we must first decide on a representation of the boundary. Different surface representations are possible and the most commonly used is parametric representation (possibly split into patches), such as produced by geometric modeling with non-uniform rational B-splines (NURBS) [PT12] or Radial Basis Functions (RBFs) [Car+03]. Other representations include level-sets [ZOF01] or subdivision surfaces [LLS01] which each have their own advantages and disadvantages (see [Str06]).

Parametric representation is most suitable for point placement and most existing algorithms also assume that domain boundary is available as a parametric surface, given by a parametrization $\boldsymbol{r}\colon \Lambda \to \mathbb{R}^d$. Existing techniques for point generation on parametric surfaces vary and are often generalizations of algorithms for spatial node generation. The most elementary technique is the naive sampling, which samples the parametric domain uniformly and then maps the points to the surface without any additional processing. This results in a non-uniform distribution of nodes on the surface, depending of the space-distorting properties of the parametric map. To counteract this distortions, the spacing of points in the parametric space can be defined as

$$h_{\text{scaled}}(\boldsymbol{\lambda}) = \frac{h}{\sqrt{\det G(\boldsymbol{\lambda})}}, \quad G(\boldsymbol{\lambda}) = \left[\left\langle \frac{\partial \boldsymbol{r}}{\partial x_i}(\boldsymbol{\lambda}), \frac{\partial \boldsymbol{r}}{\partial x_j}(\boldsymbol{\lambda}) \right\rangle\right]_{i,j}. \tag{3.3.1}$$

This means that a sampling algorithm that supports variable density distributions is needed. This scaling technique works well for conformal surface mappings as suggested by Fornberg and Flyer [FF15b], but conformal mapping are not often easily available. They can be computed numerically [Gu+12], but the computation is expensive and not worth it in general, as better techniques exist.

Point sampling on surfaces is often treated in the context of random sampling, probabilistic approaches that generate uniformly distributed points (in the sense of probability) are available [DHS13; KM15]. These are not suitable as node generators for PDE discretizations due to the potentially high irregularity of generated node sets. However, both naive and probabilistic approaches can be useful to generate initial distributions for iterative discretization improvement schemes, such as minimal energy nodes [HS04].

Shankar et al. [SKF18] also offered a boundary discretization algorithms along with the algorithm for interior generation. Their approach relies on supersampling and decimation: the parametric space is sampled with e.g. 5 times smaller spacing, the points are mapped to the surface and the mapped points are subsequently decimated to conform to the required nodal spacing. However, the algorithm only deals with cases where the surface is homeomorphic to a sphere $\mathbb{S}^1$ or $\mathbb{S}^2$, where parametric domain is a rectangle and nodal spacing is constant.

The surface placing algorithm proposed in the next section works in arbitrary dimensions with variable nodal spacing, with irregular surfaces and parametric domains,

but at the cost of requesting the user to supply the Jacobian $\nabla \boldsymbol{r}$ of the surface map. This can be problematic when compared to the simplicity of conformal mapping techniques or supersampling, but offers robustness, and is also often available in NURBS or RBF based geometric models. The algorithm and the analyses are also included in the pre-print [DKS20].

3.3.1 Algorithm

The algorithm is a generalization of Algorithm 3.1 used to fill domain interiors. Assume the boundary of Ω is parametrized by a regular parametrization $\boldsymbol{r}\colon \Lambda \subseteq \mathbb{R}^{d_\Lambda} \to \partial\Omega \subseteq \mathbb{R}^d$. The space Λ will be called the parametric space and its elements will be called parameters. The dimension of the parametric space is usually $d-1$, but can be lower, when describing e.g. a curve in 3D space.

We wish to run the algorithm for discretizing domain interiors in the parametric space Λ, and map the obtained parameters to the surface. However, the distances in parametric space and on $\partial\Omega$ differ and simply mapping the points will not produce points distributed with desired spacing.

When generating a candidate parameter direction $\vec{s}$ from $\boldsymbol{\lambda} \in \Lambda$ as $\boldsymbol{\mu} = \boldsymbol{\lambda} + \alpha\vec{s}$, for $\alpha > 0$, we wish to choose α in such a way that

$$\|\boldsymbol{r}(\boldsymbol{\lambda}) - \boldsymbol{r}(\boldsymbol{\mu})\| = h(\boldsymbol{r}(\boldsymbol{\lambda})), \tag{3.3.2}$$

where the norm is measured in $\mathbb{R}^d$. The illustration of this is shown in Figure 3.4.

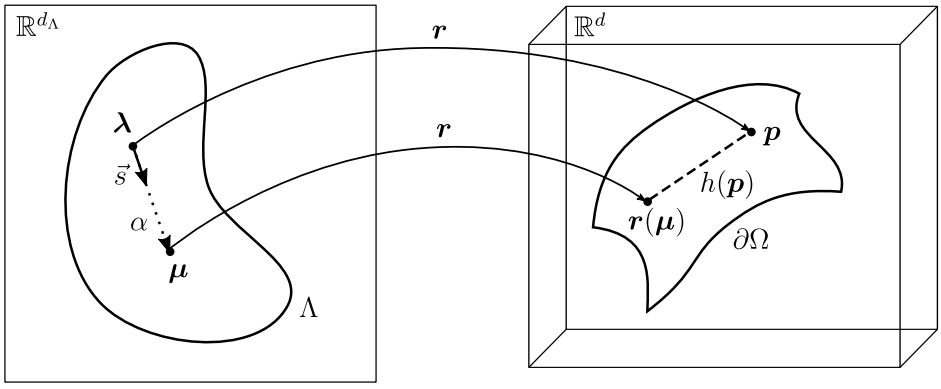


Figure 3.4: Placing a candidate $\boldsymbol{\mu}$ from $\boldsymbol{\lambda}$ in the parametric domain and the corresponding nodes on the surface.

To achieve this equality exactly, we would in general need to solve a nonlinear equation involving $\boldsymbol{r}$, but we can approximate it using linear Taylor's expansion.

$$\boldsymbol{r}(\boldsymbol{\mu}) = \boldsymbol{r}(\boldsymbol{\lambda} + \alpha\vec{s}) = \boldsymbol{r}(\boldsymbol{\lambda}) + \alpha\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s} + \boldsymbol{R}(\boldsymbol{\lambda}, \alpha, \vec{s}) \approx \boldsymbol{r}(\boldsymbol{\lambda}) + \alpha\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}, \tag{3.3.3}$$

where the remainder term $\boldsymbol{R}(\boldsymbol{\lambda}, \alpha, \vec{s})$ was dropped. Using this approximation in (3.3.2), we obtain

$$h(\boldsymbol{r}(\boldsymbol{\lambda})) = \|\boldsymbol{r}(\boldsymbol{\lambda}) - \boldsymbol{r}(\boldsymbol{\lambda}) - \alpha\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\| = \alpha\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|, \tag{3.3.4}$$

due to positivity of α. This can be used to obtain the candidate $\boldsymbol{\mu}$ as

$$\boldsymbol{\mu} = \boldsymbol{\lambda} + \frac{h(\boldsymbol{r}(\boldsymbol{\lambda}))}{\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|}\vec{s}, \quad \alpha = \frac{h(\boldsymbol{r}(\boldsymbol{\lambda}))}{\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|}. \tag{3.3.5}$$

Algorithm 3.3 Node generation on parametric surfaces.

Input: A set Λ, given by a characteristic function $\chi_\Lambda \colon \Lambda \subseteq \mathbb{R}^{d_\Lambda} \to \{0, 1\}$.
Input: A parametrization $\boldsymbol{r} \colon \Lambda \subseteq \mathbb{R}^{d_\Lambda} \to \partial\Omega$ and its Jacobian $\nabla\boldsymbol{r}$.
Input: A nodal spacing function $h \colon \partial\Omega \subset \mathbb{R}^d \to (0, \infty)$.
Input: A nonempty list of starting parameters $L \subseteq \Lambda$.
Input: Maximal number of nodes generated $N_{\max} \geq 0$.
Input: A whole number $k > 0$, influencing the number of generated candidates.
Output: A list of nodes in $\partial\Omega$ distributed according to spacing function h.

function FILL(Λ, $\boldsymbol{r}$, $\nabla\boldsymbol{r}$, h, L, $N_{\max}$, k **default** 10)
  $S \leftarrow$ INIT($\boldsymbol{r}(L)$) ▷ Initialize spatial search structure on points with parameters L.
  $i \leftarrow 0$ ▷ Current node index.
  while $i < |L|$ **and** $i < N_{\max}$ **do** ▷ Until the queue is not empty.
    $\boldsymbol{\lambda}_i \leftarrow L[i]$ ▷ Dequeue current parameter.
    $\boldsymbol{p}_i \leftarrow \boldsymbol{r}(\boldsymbol{\lambda}_i)$ ▷ Compute the point in $\partial\Omega$.
    $h_i \leftarrow h(\boldsymbol{p}_i)$ ▷ Compute its local nodal spacing.
    for each $\vec{s}_{i,j}$ **in** CANDIDATES(d_Λ, k) **do** ▷ Loop through directions in Λ.
      $\boldsymbol{\mu}_{i,j} \leftarrow \lambda_i + (h_i / \|\nabla\boldsymbol{r}(\boldsymbol{\lambda}_i)\vec{s}_{i,j}\|)\vec{s}_{i,j}$ ▷ Generate a new candidate parameter.
      if $\boldsymbol{\mu}_{i,j} \in \Lambda$ **then** ▷ Discard parameters outside Λ.
        $\boldsymbol{c}_{i,j} \leftarrow \boldsymbol{r}(\boldsymbol{\mu}_{i,j})$ ▷ Map the candidate parameter to $\partial\Omega$.
        $\boldsymbol{n}_{i,j} \leftarrow$ FINDNEAREST($S, \boldsymbol{c}_{i,j}$) ▷ Find nearest node for proximity test.
        $\hat{h}_{i,j} \leftarrow \|\boldsymbol{c}_{i,j} - \boldsymbol{p}_i\|$ ▷ Compute the actual spacing.
        if $\|\boldsymbol{c}_{i,j} - \boldsymbol{n}_{i,j}\| \geq \hat{h}_{i,j}$ **then** ▷ Test that $\boldsymbol{c}_{i,j}$ is not too close to other nodes.
          APPEND($L, \boldsymbol{\mu}_{i,j}$) ▷ Enqueue $\boldsymbol{\mu}_{i,j}$ as the last element of L.
          INSERT($S, \boldsymbol{c}_{i,j}$) ▷ Insert $\boldsymbol{c}_{i,j}$ into the spatial search structure.
        end if
      end if
    end for
    $i \leftarrow i + 1$ ▷ Move to the next non-expanded node.
  end while
  return $\boldsymbol{r}(L)$
end function

Note that the linear equation for α can always be solved, as $\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}$ must be nonzero due to regularity of $\boldsymbol{r}$. Due to truncation of $\boldsymbol{R}$ the spacing between $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ is not exactly $h(\boldsymbol{r}(\boldsymbol{\lambda}))$. Instead, we define the actual spacing

$$\hat{h}(\boldsymbol{\lambda}, \vec{s}) = \left\| \boldsymbol{r}(\boldsymbol{\lambda}) - \boldsymbol{r}\left(\boldsymbol{\lambda} + \frac{h(\boldsymbol{r}(\boldsymbol{\lambda}))}{\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|}\vec{s}\right) \right\|. \tag{3.3.6}$$

Higher orders of Taylor's approximation could be used to obtain α as well, but we would have to solve a quadratic (or higher order) equation and it would require the user to know higher order derivatives of the surface, which is often not the case.

Other than the generation of candidate nodes, the algorithm works very similarly to the interior fill algorithm. A list of initial parameters is provided and the parameters are processed one by one. A spatial search structure containing already accepted nodes in $\mathbb{R}^d$ is kept. When new candidate parameters are generated from a candidate $\boldsymbol{\lambda}$, the candidate nodes are positioned according to (3.3.5). The spatial search structure is used to determine if any of the nodes are too close. However, we cannot use the exact spacing

$h(\boldsymbol{r}(\boldsymbol{\lambda}))$, because it might happen that even $\boldsymbol{r}(\boldsymbol{\lambda})$ is too close to $\boldsymbol{r}(\boldsymbol{\mu})$, due to the errors in linear approximation of α. Instead, we use the distance $\hat{h}$, and if no other nodes are present, the candidate is accepted.

The described algorithm is implemented in pseudocode as Algorithm 3.3. It returns a list of nodes on the surface $\partial\Omega$, but can be trivially modified to return the list of accepted parameters $\boldsymbol{\lambda}$ as well if desired. Figure 3.10 shows an example of node set on a parametric surface as well as the corresponding parameters in the parametric domain.

3.3.2 Possible generalizations

The algorithm as described requires a regular parametrization of a domain boundary $\partial\Omega$. In general, it is not needed that the parametrized set is a boundary of some domain, but can be an arbitrary surface of any dimension embedded in $\mathbb{R}^d$. The surface can also be non-orientable, self-intersecting, or have a boundary itself. Figure 3.5 shows a discretized Möbius strip, parametrized by

$$\boldsymbol{r}(u,v) = \left[\cos(u)\left(\frac{1}{2}v\cos(u/2)+1\right), \sin(u)\left(\frac{1}{2}v\cos\left(\frac{u}{2}\right)+1\right), \frac{1}{2}v\sin\left(\frac{u}{2}\right)\right]^{\mathsf{T}}, \quad (3.3.7)$$

whose boundary and interior were discretized using Algorithm 3.3.

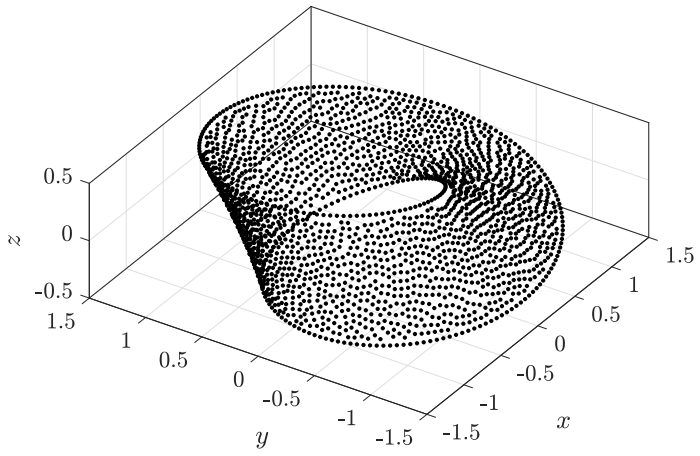


Figure 3.5: Discretization of a Möbius strip.

The algorithm works the same way, if the spatial structure S already contains a few nodes at the beginning. If the representation of the surface is made of patches, the algorithm can be run consequently on each patch with S storing the discretization nodes so far.

3.3.3 Time and space complexity

The analysis of time complexity is very similar to Algorithm 3.1. The only difference is that we need to evaluate χ_Λ instead of χ_Ω and that we need to evaluate the parametric mapping $\boldsymbol{r}$. In the worst case, this happens n_c times per candidate. Additionally, we also need to evaluate $\nabla\boldsymbol{r}$ once per generated node. If we denote the time complexities

of evaluating χ_Λ, $\boldsymbol{r}$ and $\nabla\boldsymbol{r}$ with T_Λ, $T_{\boldsymbol{r}}$ and $T_{\nabla\boldsymbol{r}}$, and assume we are using a k-d tree as a spatial search structure, we can write the total time complexity as

$$T_{\text{surface}} = O(N(T_h + T_{\nabla\boldsymbol{r}}) + Nn_c(T_\Omega + T_{\boldsymbol{r}} + \log N)). \tag{3.3.8}$$

The spatial complexity remains the same as for Algorithm 3.1. However, the use of a background-grid based search structure, even though faster, presents an even greater memory overhead, as the points on the surface will not fully fill the space that the surface occupies. However, if the interior fill algorithm is run directly after the boundary fill algorithm, the spatial search structure generated at the end of Algorithm 3.3 can be reused as the first step of Algorithm 3.1 to avoid construction of a new one.

3.3.4 Minimal spacing requirements

The surface node placing algorithm inherits minimal spacing requirements from the underlying node placing algorithm in the parametric space. The only difference is that $\hat{h}$ is used as a spacing criterion instead of h, and we wish to quantify this error.

We denote this difference as $\Delta h(\lambda, \vec{s}) = h(\boldsymbol{r}(\boldsymbol{\lambda})) - \hat{h}(\boldsymbol{\lambda}, \vec{s})$ and estimate it. First, we will need the following lemma.

Lemma 3.3.1. *Let $a \in \mathbb{R}$ and $b, c \in \mathbb{R}^n$. Then*

$$|a - \|b + c\|| \leq |a - \|b\|| + \|c\|. \tag{3.3.9}$$

Proof. We differentiate two cases

(i) $a \leq \|b + c\|$:

$$|a-\|b+c\|| = \|b+c\|-a \leq \|b\|+\|c\|-a = \|b\|-a+\|c\| \leq |\|b\|-a|+\|c\| \tag{3.3.10}$$

(ii) $a \geq \|b + c\|$:

$$|a - \|b + c\|| = a - \|b + c\| \leq a - (\|b\| - \|c\|) \leq a - \|b\| + \|c\| = |a - \|b\|| + \|c\| \tag{3.3.11}$$

We used the fact that $\|b\| - \|c\| \leq \|b + c\| \leq \|b\| + \|c\|$ and $a \leq |a|$ hold for all a, b and c. □

This can be used to derive the following bounds on Δh. We need to assume that $\boldsymbol{r}$ is differentiable on Λ to use Taylor's theorem and the appropriate topological prerequisites on Λ must be assumed, e.g. either Λ is open or $\boldsymbol{r}$ is defined on a neighborhood of Λ.

Proposition 3.3.2. *The following estimates hold for Δh:*

$$|\Delta h(\boldsymbol{\lambda}, \vec{s})| \leq \frac{\sqrt{d_\Lambda}}{2} h(\boldsymbol{p})^2 \frac{\max\limits_{i\in[d_\Lambda]} \max\limits_{\theta\in[0,\alpha]} \left|\vec{s}^{\mathsf{T}} (\nabla\nabla r_i)(\boldsymbol{\lambda} + \theta\vec{s})\vec{s}\right|}{\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|^2}, \alpha = \frac{h(\boldsymbol{p})}{\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|}, \tag{3.3.12}$$

$$|\Delta h(\boldsymbol{\lambda}, \vec{s})| \leq \frac{\sqrt{d_\Lambda}}{2} h(\boldsymbol{p})^2 \frac{\max\limits_{i\in[d_\Lambda]} \sup\limits_{\boldsymbol{\zeta}\in B(\boldsymbol{\lambda},\rho_{\boldsymbol{\lambda}})\cap\Lambda} \sigma_1((\nabla\nabla r_i)(\boldsymbol{\zeta}))}{\sigma_{d_\Lambda}(\nabla\boldsymbol{r}(\boldsymbol{\lambda}))^2}, \ \rho_{\boldsymbol{\lambda}} = \frac{h(\boldsymbol{p})}{\sigma_{d_\Lambda}(\nabla\boldsymbol{r}(\boldsymbol{\lambda}))}, \tag{3.3.13}$$

$$|\Delta h(\boldsymbol{\lambda}, \vec{s})| \leq \frac{\sqrt{d_\Lambda}}{2} h_M^2 \frac{\sigma_{1,M}(\nabla\nabla\boldsymbol{r})}{\sigma^2_{d_\Lambda,m}(\nabla\boldsymbol{r})}, \tag{3.3.14}$$

where

$$h_M^2 = \sup_{\boldsymbol{\lambda}\in\Lambda} h(\boldsymbol{r}(\boldsymbol{\lambda}))^2, \tag{3.3.15}$$

$$\sigma_{1,M}(\nabla\nabla\boldsymbol{r}) = \max_{i=1,\dots,d_\Lambda} \sup_{\boldsymbol{\lambda}\in\Lambda} \sigma_1((\nabla\nabla r_i)(\boldsymbol{\lambda})), \tag{3.3.16}$$

$$\sigma_{d_\Lambda,m}(\nabla\boldsymbol{r}) = \inf_{\boldsymbol{\lambda}\in\Lambda} \sigma_{d_\Lambda}(\nabla\boldsymbol{r}(\boldsymbol{\lambda})), \tag{3.3.17}$$

and $\sigma_i(A)$ denotes the i-th largest singular value of A.

Remark 3.3.3. We included above three estimates instead of only the final one, because they differ in their locality. The first estimate depends on $\boldsymbol{\lambda}$ and $\vec{s}$, the second depends on $\boldsymbol{\lambda}$ only, and the third is independent of $\boldsymbol{\lambda}$ and $\vec{s}$.

In particular, the final estimate proves that that the relative error in spacing $|\Delta h|/h$ decreases linearly with h for well behaved $\boldsymbol{r}$, and therefore the algorithm for placing points on surfaces asymptotically retains the minimal spacing of the underlying algorithm for flat space.

Proof. The difference between the desired and the actual nodal spacing is first shown to be bounded by the remainder of the Taylor series. We can estimate

$$\begin{aligned}
|\Delta h(\boldsymbol{\lambda},\vec{s})| &= |h(\boldsymbol{p}) - \hat{h}(\boldsymbol{\lambda},\vec{s})| && (3.3.18)\\
&= |h(\boldsymbol{p}) - \|\boldsymbol{r}(\boldsymbol{\lambda}) - \boldsymbol{r}\left(\boldsymbol{\lambda} + \alpha(\boldsymbol{\lambda},\vec{s})\vec{s}\right)\|| && (3.3.19)\\
&= |h(\boldsymbol{p}) - \|\alpha(\boldsymbol{\lambda},\vec{s})\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s} + \boldsymbol{R}(\boldsymbol{\lambda},\vec{s})\|| && (3.3.20)\\
&\leq |h(\boldsymbol{p}) - \|\alpha(\boldsymbol{\lambda},\vec{s})\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|| + \|\boldsymbol{R}(\boldsymbol{\lambda},\vec{s})\| && (3.3.21)\\
&= \left|h(\boldsymbol{p}) - \frac{h(\boldsymbol{p})}{\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|}\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|\right| + \|\boldsymbol{R}(\boldsymbol{\lambda},\vec{s})\| && (3.3.22)\\
&= \|\boldsymbol{R}(\boldsymbol{\lambda},\vec{s})\|, && (3.3.23)
\end{aligned}$$

where the inequality holds due to the lemma 3.3.1 proven above and $\boldsymbol{R}(\boldsymbol{\lambda},\vec{s})$ is the remainder of the Taylor approximation as introduced in (3.3.3). It is only dependent on $\boldsymbol{\lambda}$ and $\vec{s}$ and not α, since α itself is a function of $\boldsymbol{\lambda}$ and $\vec{s}$.

Standard bounds for reminders only hold on each component separately, so we estimate

$$\|\boldsymbol{R}(\boldsymbol{\lambda},\vec{s})\| \leq \sqrt{d_\Lambda} \max_{i=1,\dots,d_\Lambda} \|R_i(\boldsymbol{\lambda},\vec{s})\|. \tag{3.3.24}$$

and use the Lagrange form of remainder $R_i(\boldsymbol{\lambda},\vec{s}) = \frac{1}{2}\alpha^2\vec{s}^{\mathsf{T}}\nabla\nabla r_i(\boldsymbol{p}+\theta\vec{s})\vec{s}$ on each component. This gives us

$$R_i(\boldsymbol{\lambda},\vec{s}) = \frac{1}{2}\frac{h(\boldsymbol{p})^2}{\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|^2}\vec{s}^{\mathsf{T}}(\nabla\nabla r_i)(\boldsymbol{\lambda}+\theta\vec{s})\vec{s}, \tag{3.3.25}$$

for some $\theta\in[0,\alpha]$, where $\nabla\nabla r_i$ is the Hessian matrix of the i-th component of $\boldsymbol{r}$. The remainders can be bounded as

$$|R_i(\boldsymbol{\lambda},\vec{s})| \leq \frac{1}{2}h(\boldsymbol{p})^2\frac{\max_{\theta\in[0,\alpha]}|\vec{s}^{\mathsf{T}}(\nabla\nabla r_i)(\boldsymbol{\lambda}+\theta\vec{s})\vec{s}|}{\|\nabla\boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\|^2}, \tag{3.3.26}$$

which is the first error bound.

To estimate the error around $\boldsymbol{p}$ for all directions $\vec{s}$, we can further bound (3.3.26) and eliminate some occurrences of $\vec{s}$ by bounding matrix products with singular values, making use of the fact that $\|\vec{s}\| = 1$:

$$\|\nabla \boldsymbol{r}(\boldsymbol{\lambda})\vec{s}\| \geq \sigma_{d_\Lambda}(\nabla \boldsymbol{r}(\boldsymbol{\lambda})) > 0 \tag{3.3.27}$$

$$\left|\vec{s}^{\mathsf{T}}(\nabla\nabla r_i)(\boldsymbol{\lambda}+\theta\vec{s})\vec{s}\right| \leq \sigma_1((\nabla\nabla r_i)(\boldsymbol{\lambda}+\theta\vec{s})) \tag{3.3.28}$$

where σ_{d_Λ} is the smallest singular value of the Jacobian, which is positive as $\boldsymbol{r}$ is regular, and σ_1 is the largest singular value of the Hessian. Following that, we can bound R_i as

$$|R_i(\boldsymbol{\lambda},\vec{s})| \leq \frac{1}{2}h(\boldsymbol{p})^2 \frac{\max_{\theta\in[0,\alpha]} \sigma_1((\nabla\nabla r_i)(\boldsymbol{\lambda}+\theta\vec{s}))}{\sigma_{d_\Lambda}(\nabla \boldsymbol{r}(\boldsymbol{\lambda}))^2} \tag{3.3.29}$$

$$\leq \frac{1}{2}h(\boldsymbol{p})^2 \frac{\sup\limits_{\boldsymbol{\zeta}\in B(\boldsymbol{\lambda},\rho_{\boldsymbol{\lambda}})\cap\Lambda} \sigma_1((\nabla\nabla r_i)(\boldsymbol{\zeta}))}{\sigma_{d_\Lambda}(\nabla \boldsymbol{r}(\boldsymbol{\lambda}))^2} \tag{3.3.30}$$

where

$$\rho_{\boldsymbol{\lambda}} = \frac{h(\boldsymbol{p})}{\sigma_{d_\Lambda}(\nabla \boldsymbol{r}(\boldsymbol{\lambda}))} \geq \alpha(\boldsymbol{\lambda},\vec{s}) \tag{3.3.31}$$

is the radius of the ball $B(\boldsymbol{\lambda},\rho_{\boldsymbol{\lambda}})$ centered at $\boldsymbol{\lambda}$. The inequality above holds, as maximum over $[0,\alpha]$ is equal to supremum over $(0,\alpha)$ and since $(0,\alpha) \subseteq B(\boldsymbol{\lambda},\rho_{\boldsymbol{\lambda}}) \cap \Lambda$, the supremum is sought over a larger domain. This gives the second estimate

$$|\Delta h(\boldsymbol{\lambda},\vec{s})| \leq \frac{\sqrt{d_\Lambda}}{2}h(\boldsymbol{p})^2 \frac{\max\limits_{i=1,\dots,d_\Lambda} \sup\limits_{\boldsymbol{\zeta}\in B(\boldsymbol{\lambda},\rho_{\boldsymbol{\lambda}})\cap\Lambda} \sigma_1((\nabla\nabla r_i)(\boldsymbol{\zeta}))}{\sigma_{d_\Lambda}(\nabla \boldsymbol{r}(\boldsymbol{\lambda}))^2}. \tag{3.3.32}$$

To obtain a global estimate, we take the maximum of (3.3.32) over all $\boldsymbol{\lambda}$:

$$|\Delta h(\boldsymbol{\lambda},\vec{s})| \leq \sup_{\boldsymbol{\lambda}\in\Lambda} \left(\frac{\sqrt{d_\Lambda}}{2}h(\boldsymbol{p})^2 \frac{\max\limits_{i=1,\dots,d_\Lambda} \sup\limits_{\boldsymbol{\zeta}\in B(\boldsymbol{\lambda},\rho_{\boldsymbol{\lambda}})\cap\Lambda} \sigma_1((\nabla\nabla r_i)(\boldsymbol{\zeta}))}{\sigma_{d_\Lambda}(\nabla \boldsymbol{r}(\boldsymbol{\lambda}))^2} \right) \tag{3.3.33}$$

$$\leq \frac{\sqrt{d_\Lambda}}{2} \sup_{\boldsymbol{\lambda}\in\Lambda}(h(\boldsymbol{p})^2) \frac{\max\limits_{i=1,\dots,d_\Lambda} \sup\limits_{\boldsymbol{\lambda}\in\Lambda} \sup\limits_{\boldsymbol{\zeta}\in B(\boldsymbol{\lambda},\rho_{\boldsymbol{\lambda}})\cap\Lambda} \sigma_1((\nabla\nabla r_i)(\boldsymbol{\zeta}))}{\inf\limits_{\boldsymbol{\lambda}\in\Lambda} \sigma_{d_\Lambda}^2(\nabla \boldsymbol{r}(\boldsymbol{\lambda}))} \tag{3.3.34}$$

$$\leq \frac{\sqrt{d_\Lambda}}{2} \sup_{\boldsymbol{\lambda}\in\Lambda}(h(\boldsymbol{p})^2) \frac{\max\limits_{i=1,\dots,d_\Lambda} \sup\limits_{\boldsymbol{\lambda}\in\Lambda} \sigma_1((\nabla\nabla r_i)(\boldsymbol{\lambda}))}{\inf\limits_{\boldsymbol{\lambda}\in\Lambda} \sigma_{d_\Lambda}^2(\nabla \boldsymbol{r}(\boldsymbol{\lambda}))} \tag{3.3.35}$$

$$= \frac{\sqrt{d_\Lambda}}{2} h_M^2 \frac{\sigma_{1,M}(\nabla\nabla \boldsymbol{r})}{\sigma_{d_\Lambda,m}^2(\nabla \boldsymbol{r})}. \tag{3.3.36}$$

The innermost supremum over local balls in (3.3.34) is superfluous when the supremum is sought over the whole domain. The final equality simply denotes the resulting quantities as given in the proposition. □

3.4 Analysis of node generation algorithms

The analysis of the algorithm will be mainly done on two test domains, Ω_2 and Ω_3.

The test domain Ω_2 in 2D will be the set, enclosed by the polar curve

$$\boldsymbol{r}_2(\varphi) = \frac{1}{4}(3 + \cos(3\varphi)), \quad \varphi \in [0, 2\pi). \tag{3.4.1}$$

Since Ω_2 is star-shape with respect to 0, we can define it as

$$\Omega_2 = \{(x, y);\ \|(x, y)\|_2 < \boldsymbol{r}_2(\arctan(y, x))\} \tag{3.4.2}$$

which also gives a fast and easy way of computing the characteristic function χ_{Ω_2}. The function $\arctan$ is the two-argument form of inverse tangent function, which computes $\arctan(y/x)$, but also takes into account the signs of x and y to return an angle in the correct quadrant.

The test domain Ω_3 in 3D will be the set of points enclosed by the following curve, given in spherical coordinates as

$$\boldsymbol{r}_3(\varphi, \vartheta) = \frac{1}{4}\left(3 + \frac{\vartheta^2(\pi - \vartheta)^2}{4}(2 + \cos(3\varphi))\right), \quad (\varphi, \vartheta) \in [0, 2\pi) \times [0, \pi). \tag{3.4.3}$$

Once again, Ω_3 is star-shaped with respect to 0, and we can define it as

$$\Omega_3 = \{(x, y, z);\ \|(x, y, z)\|_2 < \boldsymbol{r}_3(\arctan(y, x), \arccos(z/\|(x, y, z)\|_2)\}. \tag{3.4.4}$$

Both Ω_2 and Ω_3 are shown in Figure 3.6.

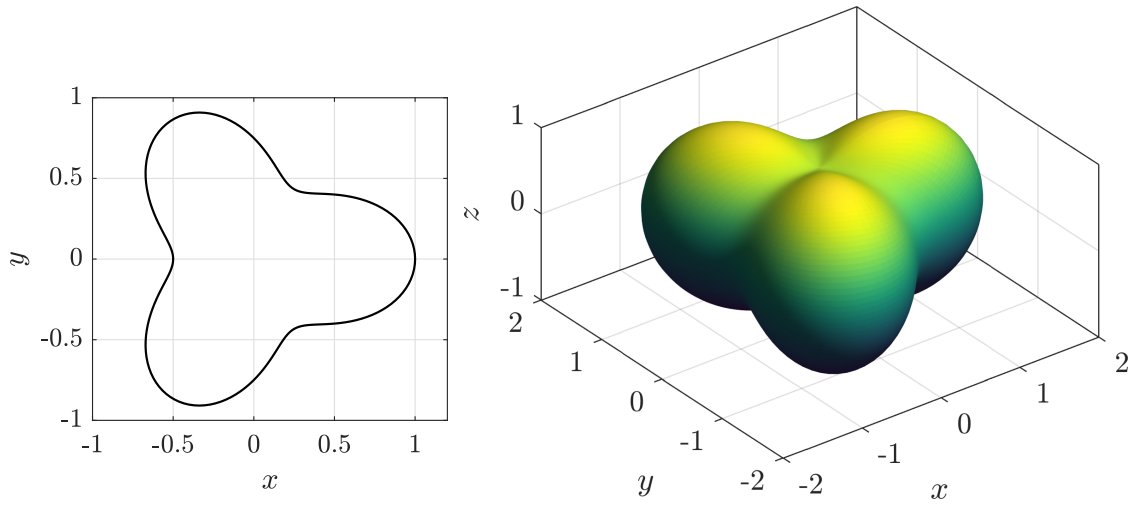


Figure 3.6: Example domains in 2D and 3D, given by (3.4.1) and (3.4.3), respectively.

When appropriate, the algorithm will be compared with two other node positioning algorithms: the Poisson disk sampling algorithm by Bridson [Bri07], which was used in [SKF18] as an interior node positioning algorithm and to the yet unpublished algorithm by van der Sande and Fornberg [SF19] which supports variable density sampling in 3D and uses a different technique analogous to dropping grains of sand in a bucket. The present algorithm will be denoted with PA, and we will use PDS for Poisson disk sampling and GD for the grain dropping algorithm.

Implementation considerations

All three algorithms were implemented in C++ and, when needed, use the same implementation of background search grids and k-d trees. The implementation of background search grids is our own, but the nanoflann library [BR14] was used when k-d trees were required.

Best effort was put into the implementation of all three algorithms and they were in the same environment as described in the introduction.

3.4.1 Quasi-uniformity

We will compare the proposed algorithm for generation of interior nodes with other available algorithms. Test domains Ω_2 and Ω_3 were filled with nodes with constant nodal spacing h, for decreasing values of h. The present algorithm was used with $k = 15$ and so was Poisson disc sampling. A random starting seed node was used. The grain drop algorithm started with a background grid of nodes with density $h/10$, as recommended by the authors. Separation distance was computed exactly, and fill distance was computed by sampling points in Ω with a few times higher density than h and computing the maximal distance to the closest node.

The results are shown in Figure 3.7. The separation distance agrees with the prescribed minimal nodal spacing h and is closely reproduced with all three algorithms. The fill distance also decreases proportionally the separation distance, as shown by the node set rations plotted on the right. All three algorithms produced node sets with bounded ratios and GD had the smallest mesh ratio.

The experiment was re-run only with the proposed algorithm, but the boundary node generation algorithm was used to first discretize the boundary and the resulting nodes were used as seed node for the interior fill algorithm. This is the standard use of the algorithms and is the one most relevant for generating node sets for PDE discretizations. The results are shown in Figure 3.8. The results are similar to before, with the difference that the minimal spacing can be lower than the target spacing. However, this is caused by the boundary nodes and is asymptotically approaching the target spacing, as proven in Proposition 3.3.2.

3.4.2 Variable density and local regularity

When considering discretizations with variable nodal spacing, quasi-uniformity is no longer an appropriate criterion for nodal spacing. We define a concept of quasi uniformity with respect to a spacing function h, that extends the notion of classical quasi-uniformity to variable distributions. Additionally, distances to nearest neighbors will also be analyzed to ensure that they are approximately equal in all directions.

To define quasi uniformity with respect to h, we need to define the local versions of fill and separation distances.

Definition 3.4.1 (Local fill distance)**.** The local fill distance $h_{X,\Omega}(\boldsymbol{x}_i)$ at node x_i is the diameter of the largest ball with a center in Ω, such that $\boldsymbol{x}_i$ lies on the boundary and that the ball contains no nodes in the interior:

$$h_{X,\Omega}(\boldsymbol{x}_i) = \sup_{\boldsymbol{x}\in\Omega}\{2\|\boldsymbol{x} - \boldsymbol{x}_i\|; B(\boldsymbol{x}, \|\boldsymbol{x} - \boldsymbol{x}_i\|) \cap X = \emptyset\}. \tag{3.4.5}$$

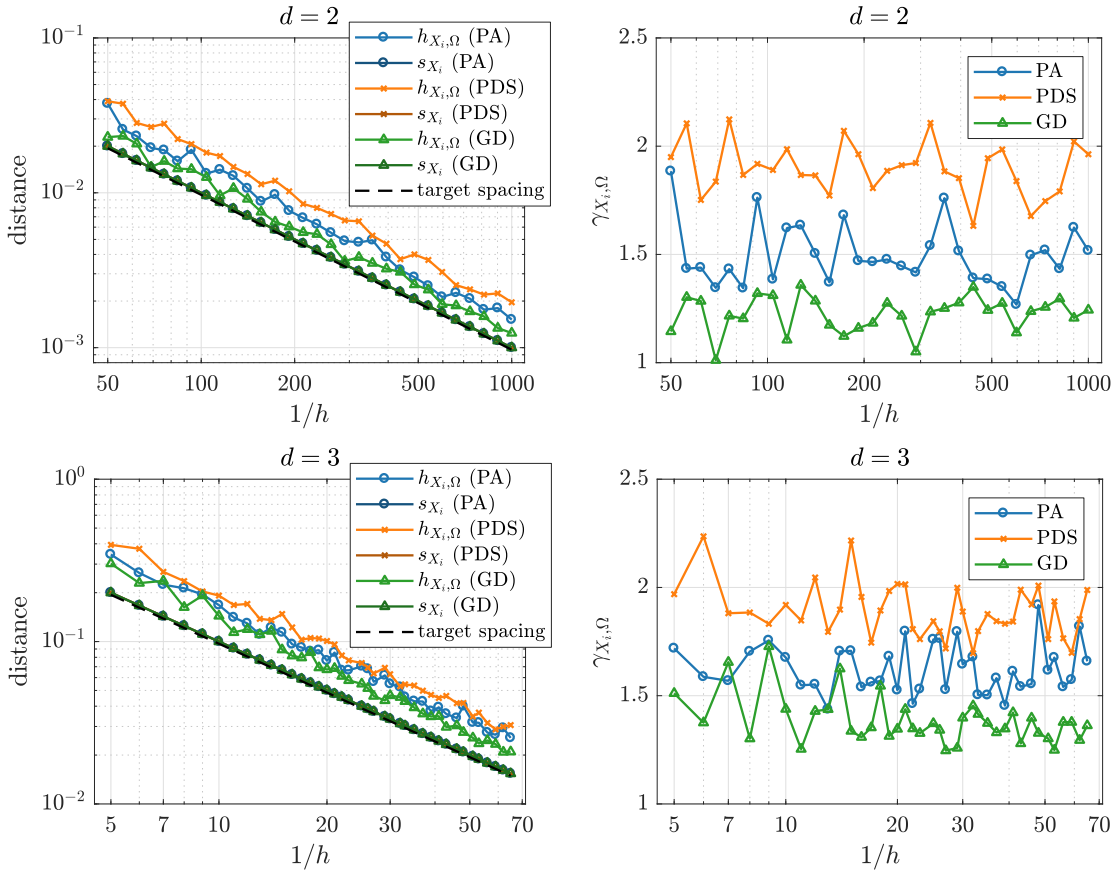


Figure 3.7: Fill and separation distance of node sets with constant density (left) and the node set ratio of generated node sets (right). The node sets were generated with three different interior node generation algorithms. The number of nodes at the smallest h surpassed 10^6 in both 2D and 3D. The s_{X_i} graphs in the left plot for all three algorithms overlap with the target spacing line.

The local fill distance can be straightforwardly used to compute the overall fill distance.

Proposition 3.4.2 (Fill distance as a maximum of local fill distances)**.** *The fill distance $h_{X,\Omega}$ can be expressed as a maximum of local fill distances:*

$$h_{X,\Omega} = \max_{\boldsymbol{x}_i \in X} h_{X,\Omega}(\boldsymbol{x}_i). \tag{3.4.6}$$

Proof. The fill distance is expressed as $h_{X,\Omega} = 2 \sup_{\boldsymbol{x} \in \Omega} \min_{\boldsymbol{x}_j \in X} \|\boldsymbol{x} - \boldsymbol{x}_j\|$. Local fill distance is clearly smaller because it only considers empty balls touching a certain node instead of all empty balls. However, any $\boldsymbol{x} \in \overline{\Omega}$ that realizes the supremum will touch at least one node $\boldsymbol{x}_i$ (otherwise the radius could be extended) and the local fill distance at that $\boldsymbol{x}_i$ will be equal to overall fill distance. □

Definition 3.4.3 (Local separation distance)**.** The local separation distance of the node $\boldsymbol{x}_i$ is the closest distance to any other node, i.e.

$$s_X(\boldsymbol{x}_i) = \min_{\boldsymbol{x}_j \in X \setminus \{\boldsymbol{x}_i\}} \|\boldsymbol{x}_j - \boldsymbol{x}_i\|. \tag{3.4.7}$$

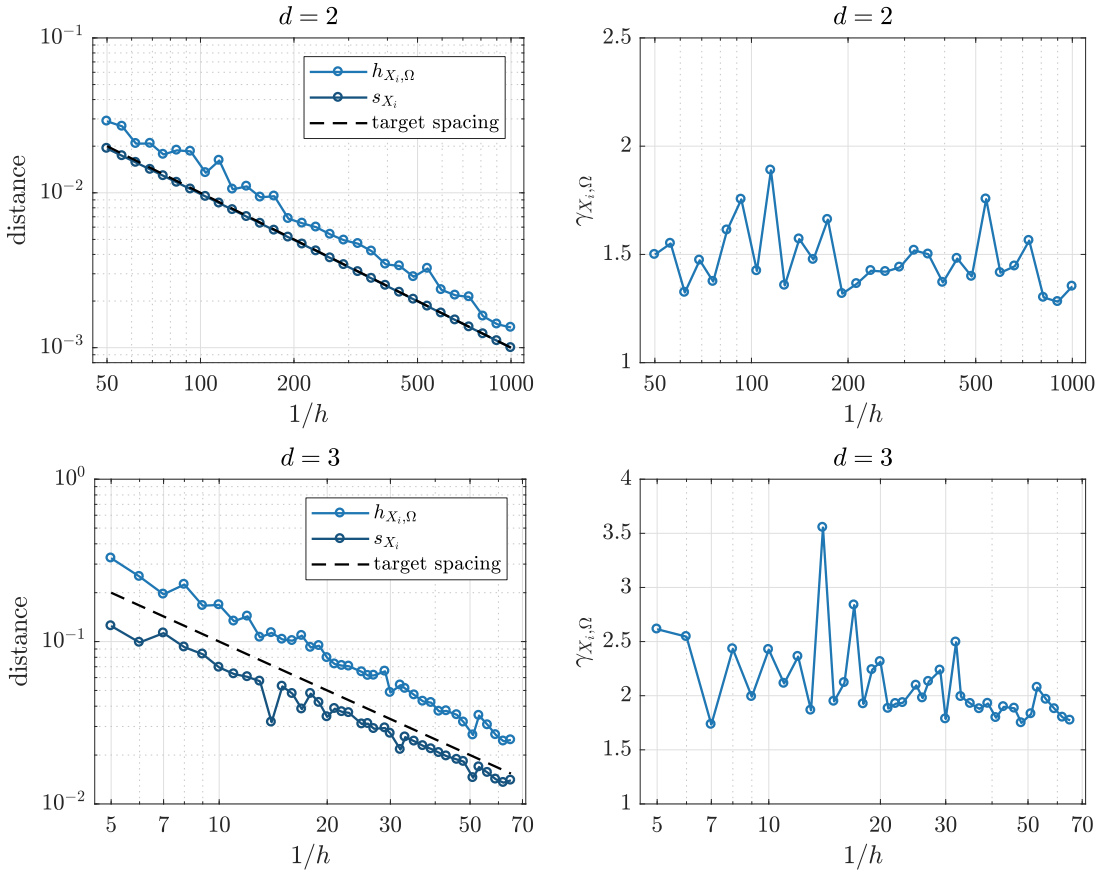


Figure 3.8: Fill and separation distance of node sets with constant density (left) and the node set ratio of generated node sets (right). The node sets were generated with the proposed boundary and interior node generation algorithms. The number of nodes at the smallest h surpassed 10^6 in both 2D and 3D.

Remark 3.4.4. The overall separation can be obtained by taking the minimum over local separation distance $s_X = \min_{\boldsymbol{x}_i \in X} s_X(\boldsymbol{x}_i)$.

Definition 3.4.5 (Quasi-uniformity with respect to prescribed nodal spacing)**.** When the desired nodal spacing is known in advance as a function $h\colon \Omega \to (0, \infty)$, we can define the node set ratio as

$$\gamma_{X,\Omega,h} = \frac{\max_{\boldsymbol{x}_i \in X} h_{X,\Omega}(\boldsymbol{x}_i)/h(\boldsymbol{x}_i)}{\min_{\boldsymbol{x}_i \in X} s_X(\boldsymbol{x}_i)/h(\boldsymbol{x}_i)}. \tag{3.4.8}$$

Then a sequence of node sets $\{X_n\}$ is called quasi-uniform with respect to h_n if the sequence γ_{X_n,Ω,h_n} is bounded independently of h_n.

Remark 3.4.6. If h_n are constants, then quasi-uniformity with respect to h_n reduces to ordinary quasi-uniformity. Indeed, the factor $h(\boldsymbol{x}_i)$ in the node set ratio (3.4.8) can be put before the maximum and minimum and cancels out, and the resulting maximization and minimization are equal to the fill and separation distances, per propositions 3.4.2 and 3.4.4, respectively.

As a demonstration, we fill the two test domains Ω_2 and Ω_3 using variable nodal spacing. The spacing function in our case will be constructed by selecting three points

$\boldsymbol{p}_1$, $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$ of interest at angles $\varphi_1 = \pi/3$, $\varphi_2 = \pi$ and $\varphi_3 = 5\pi/3$ in 2D and the same points in $z = 0$ plane in 3D. To construct h, we will need the distance

$$d(\boldsymbol{x}) = \min\{\|\boldsymbol{x} - \boldsymbol{p}_1\|, \|\boldsymbol{x} - \boldsymbol{p}_2\|, \|\boldsymbol{x} - \boldsymbol{p}_3\|\} \tag{3.4.9}$$

to these three points. The spacing h is then expressed as

$$h(\boldsymbol{x}) = h_{\min} + \frac{d(\boldsymbol{x})}{d_{\max}}(h_{\max} - h_{\min}), \tag{3.4.10}$$

where $h_{\min}$ and $h_{\max}$ are the minimal and maximal desired nodal spacing in the domain. The value $d_{\max}$ represents $d_{\max} = \sup_{\boldsymbol{x}\in\Omega} d(\boldsymbol{x})$, but does not need to be known exactly and can be simply a reference distance. We will use $d_{\max} = 1$.

Figure 3.10 shows node sets with variable spacing in 2D and 3D. Both node sets have nodes on the boundary and in the interior, generated with algorithms 3.3 and 3.1, respectively. The left side shows consecutive close-ups of the node set to visually asses the gradual change in density.

To test quasi uniformity with respect to h, domains Ω_2 and Ω_3 were filled with boundary and interior node filling algorithms with spatial density (3.4.10). The values $1/h_{\min}$ varied from 50 to 5000 in 2D and from 10 to 500 in 3D. The values $1/h_{\max}$ varied from 10 to 250 in 2D and from 5 to 30 in 3D. Both $1/h_{\min}$ and $1/h_{\max}$ were distributed equidistantly in logarithmic space between the respective lower and upper bounds. The resulting node set ratios with respect to h are shown in Figure 3.9. The local separation distance $s_X(\boldsymbol{x}_i)$ was computed exactly, and the local fill distance $h_{X,\Omega}(\boldsymbol{x}_i)$ was computed numerically by using a fine uniform grid of sample points in Ω.

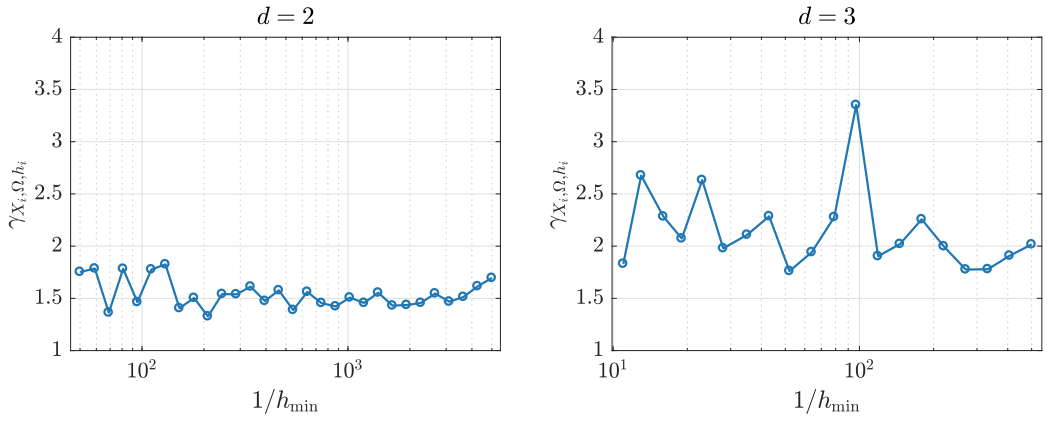


Figure 3.9: Quasi uniformity of variable density node sets with nodal spacing (3.4.10). The number of nodes in the final case surpassed 10^6 in both 2D and 3D.

Another way to measure local regularity is to look at the distances to nearest neighbors. For each node $\boldsymbol{x}_i$, we find n nearest neighbors (excluding $\boldsymbol{x}_i$ itself) denoted $\boldsymbol{x}_{i,j}$, $j = 1, \ldots, n$ and compute the distances $d_{i,j} = \|\boldsymbol{x}_i - \boldsymbol{x}_{i,j}\|$. The distances are normalized by local spacing $h(\boldsymbol{x}_i)$ to obtain the normalized distances $d'_{i,j} = d_{i,j}/h(\boldsymbol{x}_i)$. Additionally, we also compute the average normalized neighbor distance $\bar{d}'_i = \frac{1}{k}\sum_{j=1}^{k} d'_{i,j}$, the maximal and minimal normalized distances

$$(d_i^{\max})' = \max_{j=1,\ldots,k} d'_{i,j}, \qquad (d_i^{\min})' = \min_{j=1,\ldots,k} d'_{i,j}. \tag{3.4.11}$$

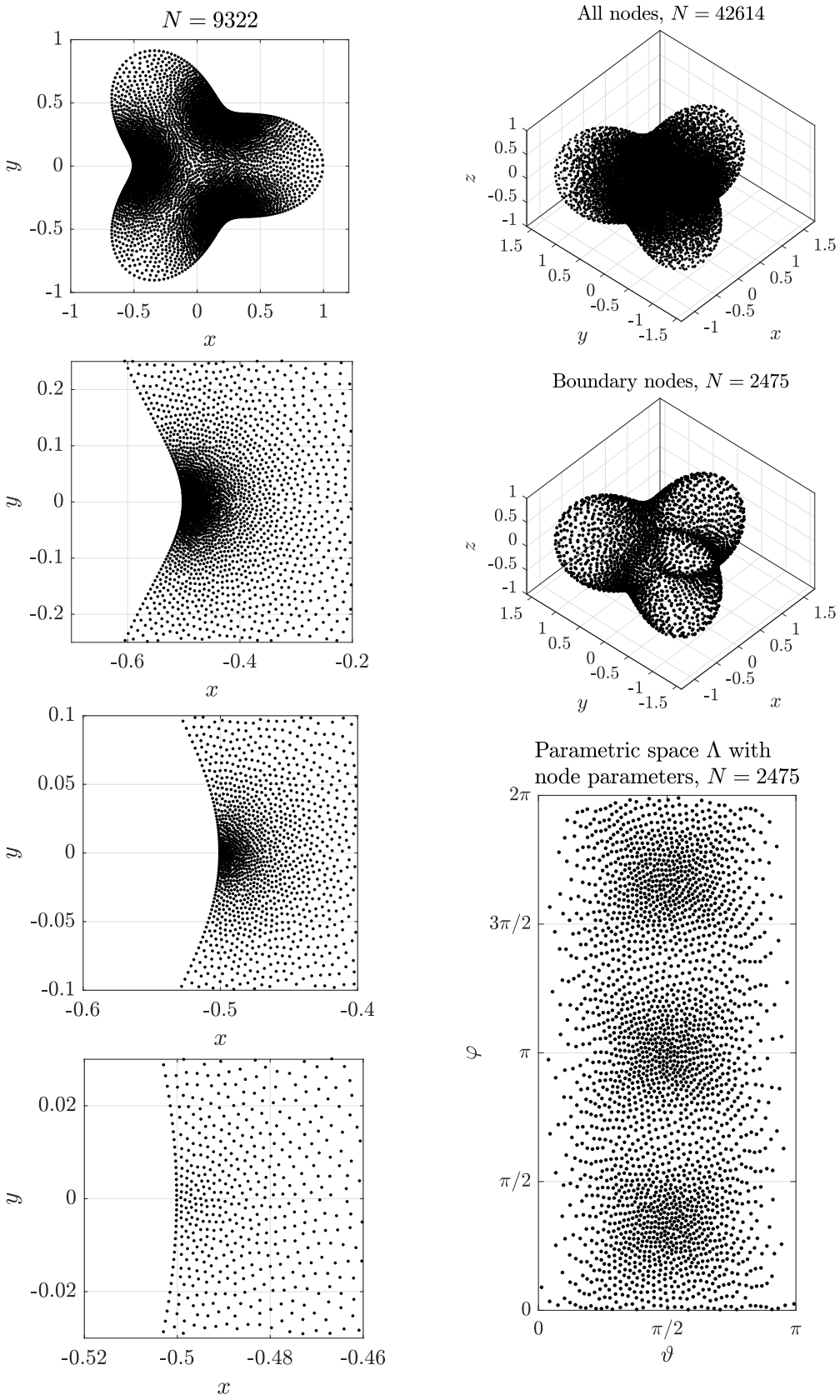


Figure 3.10: Node sets with variable density (3.4.10) in 2 and 3 spatial dimensions. The left side shows consecutive enlargement of the initial figure towards the point $(-0.5, 0)$. The right hand side shows, from top to bottom, the complete node set X, the node set on the boundary, and the parametric space with generated parameters.

The difference between these two quantities will be denoted with $(d_i^{\Delta})' = (d_i^{\max})' - (d_i^{\min})'$. Histograms of $d'_{i,j}$ are shown in Figure 3.11 for the uniformly distributed node set, generated as the last data point of Figure 3.8 in 2D and 3D, and for the variably spaced node set, generated as the last data point of Figure 3.9. The number of nearest neighbors used was $n = 6$ in 2D and $n = 12$ in 3D, which roughly corresponds to the first "layer" of neighboring nodes.

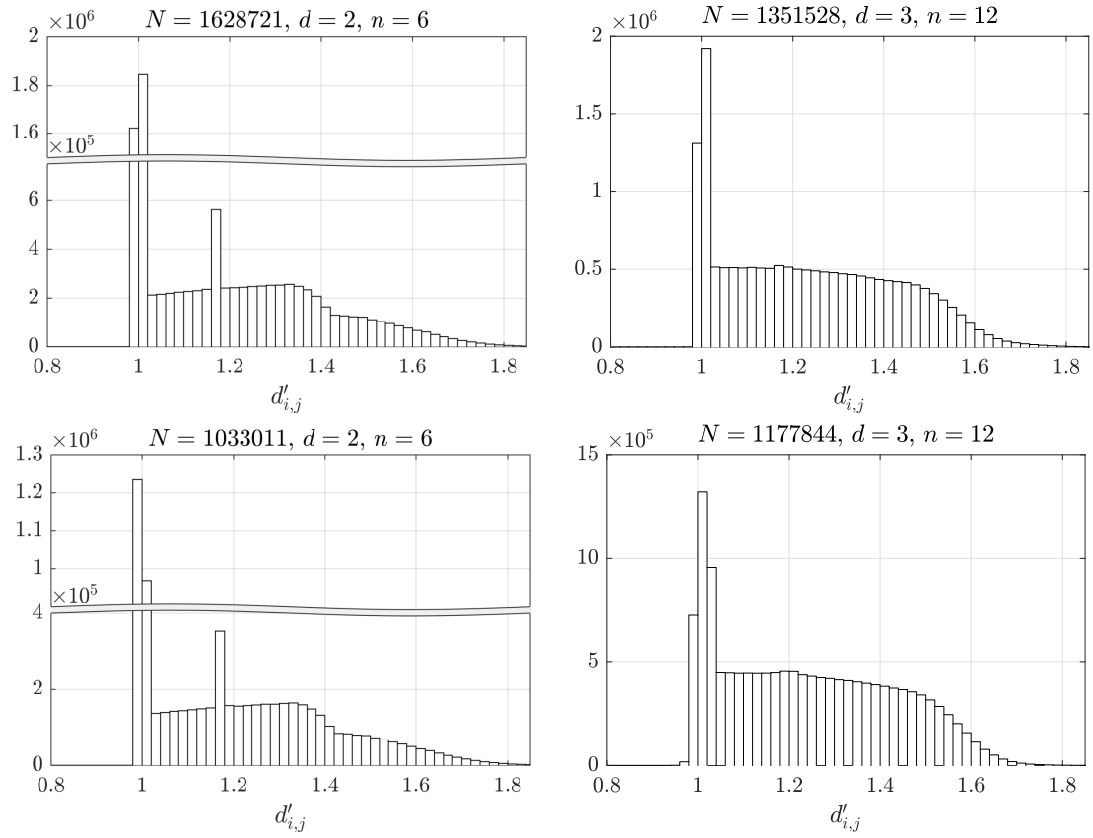


Figure 3.11: Histograms of normalized distances $d'_{i,j}$ to n nearest neighbors. Top row represents data for node sets with constant spacing and the bottom for node sets with variable spacing.

Both in 2D and in 3D, the majority of distances corresponds to local spacing h. The second peak at around 1.2 in 2D is due to boundary nodes, which have one-sided stencils and their distances follow a different distribution. Some values of $d'_{i,j}$ lower than 1 are possible, due to errors in spacing on the boundary and, in case of variable h due to not following the max-disks variant of minimal spacing criteria. Various additional statistics are reported in Table 3.1.

3.4.3 Time complexity and execution time

We compare the time complexity and execution time of PA, PDS and GA algorithms. Two variants of the proposed algorithm are tested, the variant with a k-d tree implementation with time complexity $O(N \log N)$ (called just PA) and the variant called PA-grid, which produces exactly the same nodes, but uses a background grid as a spatial search

Table 3.1: Statistics of relative distances to nearest neighbors for cases corresponding to histograms in Figure 3.11.

h	d	N	mean($\bar{d}_i$)	std($\bar{d}_i$)	mean($(d_i^\Delta)'$)	std($(d_i^\Delta)'$)
constant	2	1628721	1.1858	0.0494	0.4828	0.1354
constant	3	1351528	1.2314	0.0386	0.5300	0.0818
variable	2	1033011	1.1851	0.0495	0.4835	0.1360
variable	3	1177844	1.2395	0.0369	0.5344	0.0764

structure, reducing the time complexity to $O(N)$. The GD and PDS algorithms also use the same background grid structure and also have $O(N)$ running time. However, all the algorithms using background structure have an additional cost factor of $|\operatorname{bb}\Omega|/|\Omega|$ due to enclosing the domain in its bounding box. Additionally, if h is spatially variable, the grid needs to be reasonably fine, and the running time and memory are better expressed as $O((\min_{\boldsymbol{p}\in\Omega} h(\boldsymbol{p}))^d$, which can be significantly more than $O(N)$.

Execution time for PDS and GD algorithms as well as the two variants of PA was measured for the node sets produced during the interior quasi uniformity test from Section 3.4.1, Figure 3.7. Measured execution times are shown in Figure 3.12.

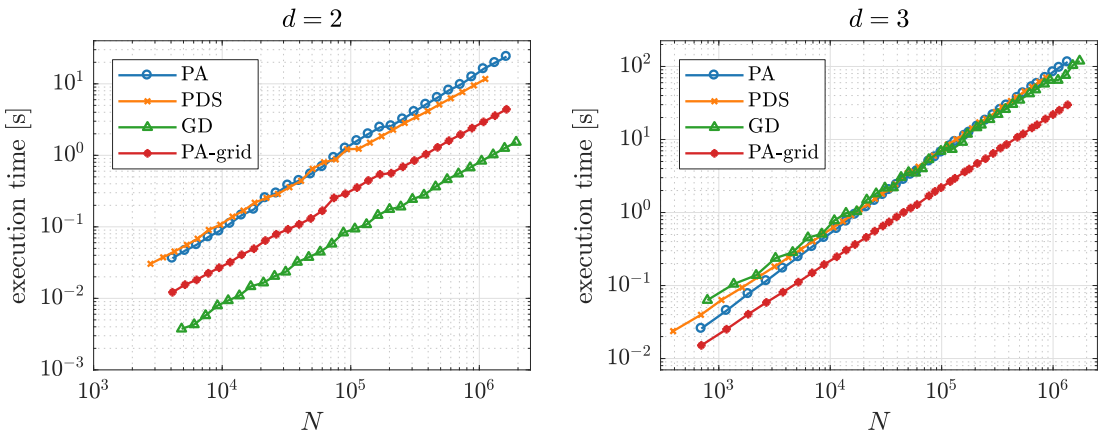


Figure 3.12: Execution time for generation of node sets in Figure 3.7. Median execution time of 15 runs is shown for each point, with absolute deviation being around 1%.

The measured execution times agree with the theoretical time complexity. In both considered cases PDS and PA have very similar execution times. PA-grid is better than both of them in 2D and in 3D, while GA performed best of all in 2D and similarly to PA and PDS in 3D.

The execution time of the combined boundary and interior fill algorithms with k-d tree spatial search structure is shown in Figure 3.13. The execution time agrees with $O(N \log N)$ time complexity and the results for constant and spatially variable h have very similar running times. This agrees with the theoretical complexity, which is not directly dependent on h (only on the number of nodes). The algorithm is slower in 3D, due to more candidates, higher cost of k-d tree operations and more complicated computation of the characteristic function.

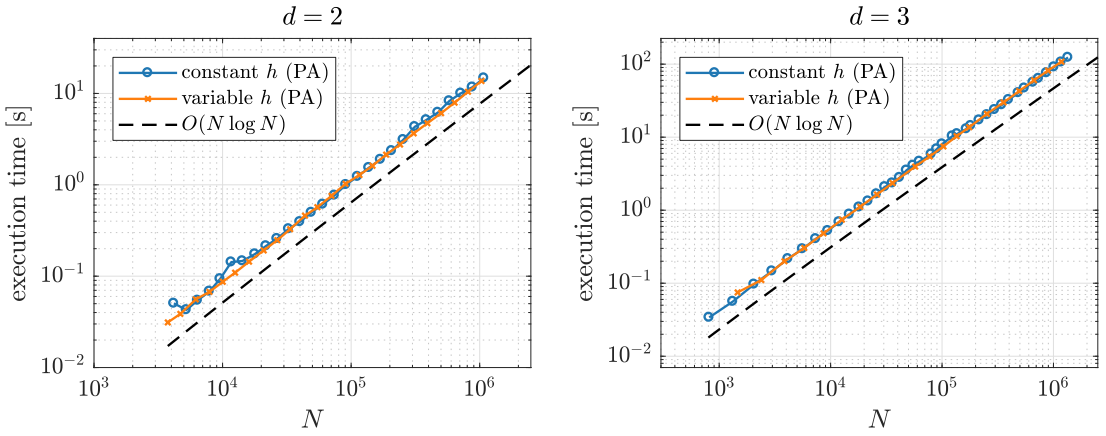


Figure 3.13: Combined execution time of boundary and interior fill algorithm for cases shown in figures 3.8 and 3.9.

3.4.4 Miscellaneous aspects

We now discuss the remaining requirements for node positioning algorithms as outlined in Section 3.1.2.

Dimension and direction independence

The proposed surface and interior fill algorithms are dimension independent, in the sense that the same algorithm can be used in any dimension d. This is also true for the implementation, the spatial dimension, and in case of boundary fill algorithm, also the parametric domain dimension are template parameters, meaning that they can be changed at will, while keeping the same implementation.

Similarly to that, the specific position and rotation of the domain also does not play any role in the algorithm. The node distributions are effectively the same in all cases. The only problematic part of the algorithm could be candidate generation, but due to random rotation applied to each batch of candidates, the starting rotation does not matter. However, if the candidate generation setup changed to a fixed pattern, the algorithm would be direction dependent.

Poisson disk sampling and the grain-drop technique are also dimension independent. Poisson Disk sampling is also direction independent, apart from the bounding box construction, but the grain-drop algorithm is intrinsically directional, as it fills the domain advancing along the last coordinate (or any other direction).

Irregular domains

The proposed algorithm handles irregular domains quite well with reasonable limitations. The geometric shape of the domain is not really relevant, but does affect the discretization, and the algorithms discovers the areas in the domain in a breadth-first search manner as the advancing front moves forwards. Note that this approach might sometimes fail to discover the whole domain, if domain includes parts narrower than local spacing h. For example in an hourglass shape with a seed node in the top part, the algorithm might stop, because it would fail to generate any candidates in the bottom part. This can be avoided, if the discretization starts from the boundary, but cases where

the algorithm fails to discretize some parts are still possible to construct. However, in practice this is rarely a problem, since the discretization usually needs to be a few times denser than the geometric features of the domain if we aim to adequately capture the geometry in the PDE discretization.

Algorithms such as GD and PDS do not suffer from these kinds of problems, as they discretize the bounding box of the domain, and only stamp out the domain shape after the discretization is complete. In this aspect, the discretization is independent of the domain shape, as long as it has the same bounding box, and problems may arise near the edges. These are discussed in more detail in the next section. Filling the bounding box causes these algorithms to generate $|\operatorname{bb}\Omega|/|\Omega|$ too many nodes, and the running time does not scale proportionally to $|\Omega|$. The proposed algorithm fares better in this aspect, and only generates nodes inside Ω, only evaluates h for points in Ω and the number of nodes actually scales proportionally to $|\Omega|$. A demonstration and further discussion of this fact is included in [SK19d].

Compatibility with boundary discretizations

Many algorithm for interior discretization do not take into account the existence of the boundary discretization. In fact, the recommended procedure is often [FF15b; SKF18] to generate interior nodes in the bounding box, superimpose the boundary discretization and remove any interior nodes outside the domain or too close to boundary nodes. This can cause gaps in spacing near the boundary, which can be additionally smoothed out by some post-processing step, such as local movement of nodes based on charged particle simulation.

The proposed algorithm has the option of supplying the boundary discretization as seed nodes, and thus the nodes near the boundary fit well next to the boundary nodes. When the advancing fronts from different sides meet, gaps can form, but they form in the middle of the domain and not on the boundary.

Free parameters

It is desirable that the algorithms do not have too many free or tuning parameters, but instead work "out of the box" for many domain shapes and spacing functions. The only remaining free parameter in the presented fill algorithms is k, which controls the number of candidates and consequently how densely the neighborhood is searched for possible positions for new nodes. This has a direct effect on the running time, and some effect on the quality of the node distribution. Values of k from around 10 to 20 give node distributions of sufficient quality. Anything larger unnecessarily increases the running time, and values lower than 6 can adversely affect the quality of the generated node sets.

Parallelization

The algorithm is not immediately parallelizable, but parallel variants of the interior node placing algorithm for multiple CPU cores are an ongoing effort. Some initial results have been presented at the ParNum 2019 and Mirpo 2020 conferences [DKS19; Duh+20]. Additionally, any parallel version of the interior fill algorithm will probably generalize to the boundary fill algorithm as well, even though that is not as important, since the number of interior nodes (and consequently the time needed to generate them) is usually orders of magnitude higher than the number of boundary nodes.

3.4.5 Behavior of RBF-FD on generated nodes

One of the most important aspects of the node generation algorithms is that meshless methods can successfully and accurately solve PDEs on the generated nodes.

Sensitivity to node positioning

To explore the sensitivity of the methods with respect to node positioning, the node set with the same constant spacing h was generated R times, each time starting from a random seed node on the boundary. The boundary set was generated first, followed by the interior nodes, where the boundary nodes were used as seeds.

This was repeated for 20 values of h chosen equidistantly in logarithmic space so that the total number of generated nodes ranged from a few thousand to over 10^5 in both 2D and 3D. Our problem of choice was a Poisson boundary value problem

$$\nabla u = f \qquad \text{in } \Omega, \tag{3.4.12}$$

$$u = g \qquad \text{on } \partial\Omega, \tag{3.4.13}$$

where the solution $u(\boldsymbol{x}) = \prod_{i=1}^{d} \sin(\pi x_i)$ was used as the closed form solution and g and f were computed from u. The Ω was chosen to be the test domain Ω_2 in 2D and Ω_3 in 3D. Stencils of closest $n = 13$ nodes were used in 2D and $n = 25$ was used in 3D. RBF-FD with Polyharmonic splines $\phi(r) = r^3$ with 2nd order monomial augmentation was used to construct the stencil weights and obtain the numerical solution u_h as described in Section 2.3.2.

The error between the analytical solution u and the numerical solution u_h was measured as relative discrete p-norm errors:

$$e_1 = \|u_h - u\|_{X,1} / \|u\|_{X,1}, \quad \|y\|_{X,1} = \sum_{\boldsymbol{x}_i \in X} |y(\boldsymbol{x}_i)|, \tag{3.4.14}$$

$$e_2 = \|u_h - u\|_{X,2} / \|u\|_{X,2}, \quad \|y\|_{X,2} = \sqrt{\sum_{\boldsymbol{x}_i \in X} |y(\boldsymbol{x}_i)|^2}, \tag{3.4.15}$$

$$e_\infty = \|u_h - u\|_{X,\infty} / \|u\|_{X,\infty}, \quad \|y\|_{X,\infty} = \max_{\boldsymbol{x}_i \in X} |y(\boldsymbol{x}_i)|. \tag{3.4.16}$$

As the solution procedure was repeated R times for each spacing h_i each time with a randomized node set and consequently random errors, it makes sense to compute statistics of various quantities. For a quantity y which resulted in R realizations $y_1, \ldots, y_R$, we will use $\operatorname{med}(y)$ to denote the median, and we will measure the absolute and relative deviation about the median using

$$\Delta(y) = \max_{i=1,\ldots,R} |y_i - \operatorname{med}(y)|, \quad \Delta_r(y) = \Delta(y) / \operatorname{med}(y). \tag{3.4.17}$$

The whole procedure was run 100 times and the obtained node counts N_i and errors $(e_\infty)_i$ are shown in Figure 3.14 as gray dots. The computed statistics for every second run are recorded in Table 3.2. The median, maximal and minimal error are also shown in Figure 3.14 for each batch of runs. We can observe that the deviation in the number of generated nodes in much smaller than the deviation of errors. While the individual error curves can be quite ragged, the median convergence is very smooth and obeys $O(h^2) = O(N^{-2/d})$ convergence rate.

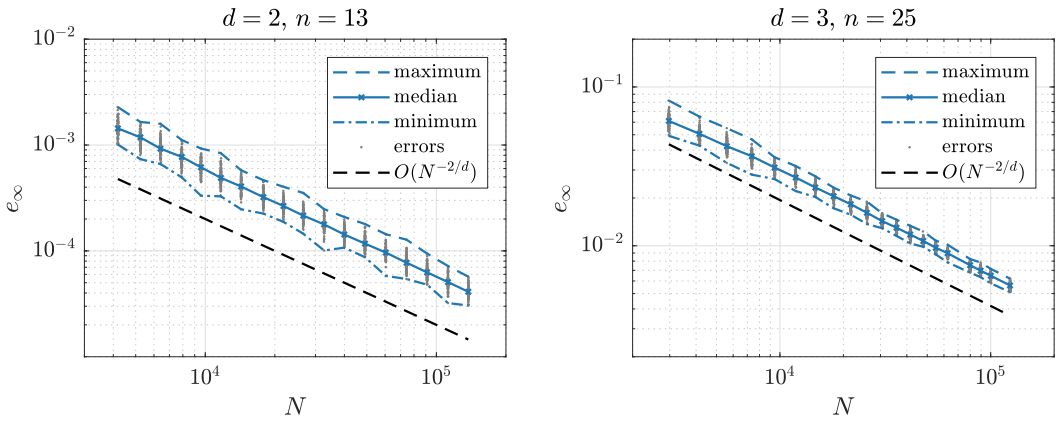


Figure 3.14: Sensitivity of RBF-FD to node positioning. Grey dots show the actual number of nodes and the error of the numerical solution on those nodes. The lines show average, minimal and maximal errors, and the expected convergence order.

Table 3.2: Medians and deviations of the number of nodes and errors obtained on randomized node sets used to solve (3.4.12).

$d = 2, n = 13$							
$\mathrm{med}(N)$	$\Delta_r(N)$	$\mathrm{med}(e_1)$	$\Delta_r(e_1)$	$\mathrm{med}(e_2)$	$\Delta_r(e_2)$	$\mathrm{med}(e_\infty)$	$\Delta_r(e_\infty)$
4188	$7.9 \cdot 10^{-3}$	$8.7 \cdot 10^{-4}$	0.44	$9.8 \cdot 10^{-4}$	0.42	$1.4 \cdot 10^{-3}$	0.59
6404	$4.4 \cdot 10^{-3}$	$5.8 \cdot 10^{-4}$	0.46	$6.6 \cdot 10^{-4}$	0.42	$9.2 \cdot 10^{-4}$	0.72
9583	$3.9 \cdot 10^{-3}$	$3.9 \cdot 10^{-4}$	0.45	$4.3 \cdot 10^{-4}$	0.44	$6.2 \cdot 10^{-4}$	0.50
14294	$2.9 \cdot 10^{-3}$	$2.6 \cdot 10^{-4}$	0.37	$2.9 \cdot 10^{-4}$	0.38	$4.1 \cdot 10^{-4}$	0.42
21796	$3.3 \cdot 10^{-3}$	$1.7 \cdot 10^{-4}$	0.45	$1.9 \cdot 10^{-4}$	0.45	$2.7 \cdot 10^{-4}$	0.52
32677	$2.1 \cdot 10^{-3}$	$1.2 \cdot 10^{-4}$	0.38	$1.3 \cdot 10^{-4}$	0.38	$1.8 \cdot 10^{-4}$	0.43
49106	$2.4 \cdot 10^{-3}$	$7.9 \cdot 10^{-5}$	0.35	$8.6 \cdot 10^{-5}$	0.37	$1.2 \cdot 10^{-4}$	0.53
74320	$1.8 \cdot 10^{-3}$	$5.2 \cdot 10^{-5}$	0.37	$5.7 \cdot 10^{-5}$	0.33	$7.7 \cdot 10^{-5}$	0.65
112279	$1.5 \cdot 10^{-3}$	$3.4 \cdot 10^{-5}$	0.36	$3.7 \cdot 10^{-5}$	0.31	$5.1 \cdot 10^{-5}$	0.42
$d = 3, n = 25$							
$\mathrm{med}(N)$	$\Delta_r(N)$	$\mathrm{med}(e_1)$	$\Delta_r(e_1)$	$\mathrm{med}(e_2)$	$\Delta_r(e_2)$	$\mathrm{med}(e_\infty)$	$\Delta_r(e_\infty)$
2972	$9.6 \cdot 10^{-3}$	$3.1 \cdot 10^{-2}$	0.19	$3.7 \cdot 10^{-2}$	0.19	$6.1 \cdot 10^{-2}$	0.34
5598	$9.1 \cdot 10^{-3}$	$2.3 \cdot 10^{-2}$	0.14	$2.7 \cdot 10^{-2}$	0.15	$4.2 \cdot 10^{-2}$	0.30
9432	$7.9 \cdot 10^{-3}$	$1.7 \cdot 10^{-2}$	0.12	$2.0 \cdot 10^{-2}$	0.11	$3.1 \cdot 10^{-2}$	0.16
14718	$6.9 \cdot 10^{-3}$	$1.3 \cdot 10^{-2}$	0.09	$1.5 \cdot 10^{-2}$	0.10	$2.3 \cdot 10^{-2}$	0.18
21686	$6.5 \cdot 10^{-3}$	$1.0 \cdot 10^{-2}$	0.10	$1.2 \cdot 10^{-2}$	0.10	$1.8 \cdot 10^{-2}$	0.14
30539	$5.5 \cdot 10^{-3}$	$8.4 \cdot 10^{-3}$	0.09	$9.5 \cdot 10^{-3}$	0.09	$1.4 \cdot 10^{-2}$	0.12
41526	$5.6 \cdot 10^{-3}$	$6.9 \cdot 10^{-3}$	0.08	$7.8 \cdot 10^{-3}$	0.08	$1.2 \cdot 10^{-2}$	0.13
54880	$5.5 \cdot 10^{-3}$	$5.7 \cdot 10^{-3}$	0.08	$6.5 \cdot 10^{-3}$	0.08	$9.7 \cdot 10^{-3}$	0.11
79886	$5.3 \cdot 10^{-3}$	$4.5 \cdot 10^{-3}$	0.06	$5.1 \cdot 10^{-3}$	0.07	$7.5 \cdot 10^{-3}$	0.10
100119	$6.0 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	0.06	$4.4 \cdot 10^{-3}$	0.07	$6.5 \cdot 10^{-3}$	0.10

The variability of the errors is largest in e_∞ error and decreases on denser node sets. This can be observed both for the number of nodes and the error. In 2D, it can be well observed in e_1 and e_2 errors, but in 3D the decrease is also nicely visible on the e_∞ error

plot. The magnitude of this variability also varies with the stencil size n. Stencils of size 9 in 2D and 15 in 3D gave similar results, but with larger deviations.

Spectra of the discretized Laplacian

Another way to test the quality of the node set and the method used is to observe the spectrum of the differentiation matrix for the Laplace operator [Bay+17; SKF18; SK19d].

We used the same setup as before when testing sensitivity to node positioning. The differentiation matrix as constructed in Section 2.3.2 was restricted to only the indices corresponding to internal nodes. The structure of the 2D and 3D matrices is shown in the top row of Figure 3.15. The spectra of these matrices are expected to resemble the spectrum of the continuous Laplacian, i.e. the eigenvalues should have negative real parts and small imaginary parts. The bottom row of Figure 3.15 shows the spectra of the differentiation matrices obtained in our case, which also posses the desired properties.[2]

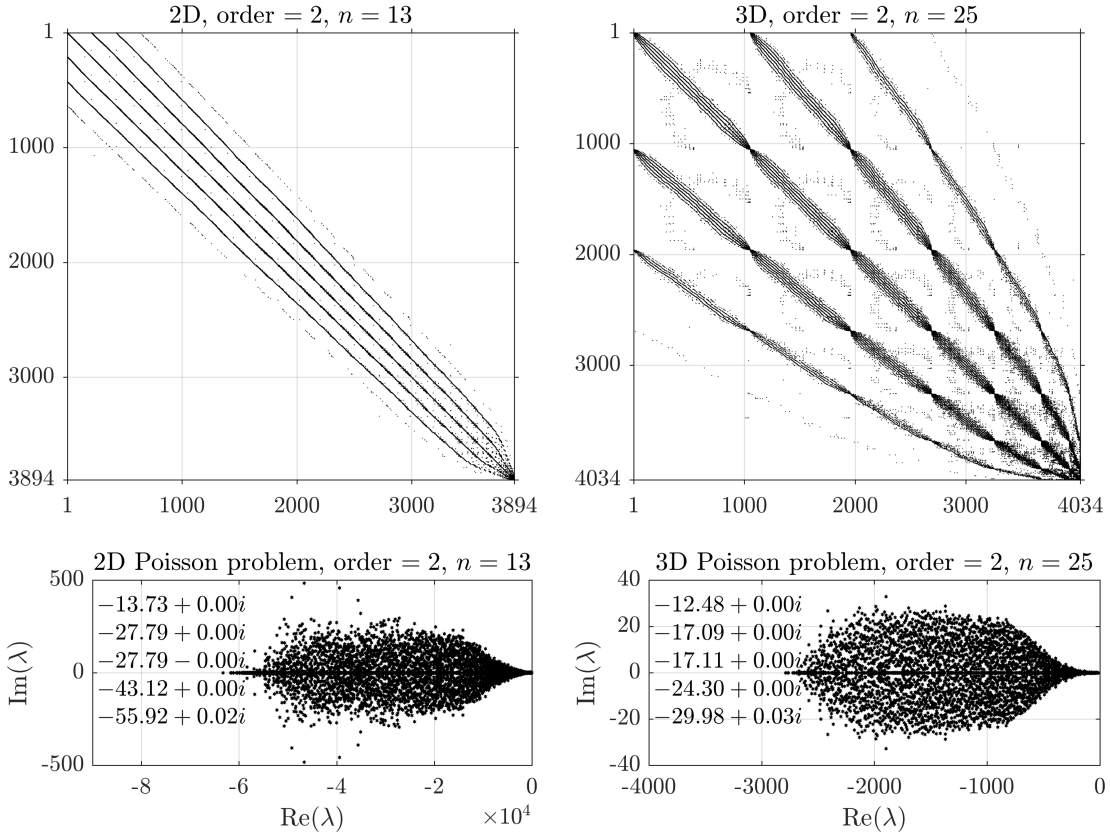


Figure 3.15: Differentiation matrices for Laplacian operator and their spectra in the complex plane. Five eigenvalues with the largest real parts are written out on the spectral plot. Note the different units on the real and imaginary axes.

[2] A similar picture of a differentiation matrix its lemon-shaped spectrum to the one shown in the right part of Figure 3.15 was selected as a part of a public photo exhibition commemorating University of Ljubljana's centennial, where researchers and artists from the Faculty of Mathematics and Physics showed visually interesting parts of their research work.

3.5 Stencil selection

The final step of constructing the discretization is the stencil selection. Choosing the stencils is sometimes analogized to meshing, as both involve "connecting" the nodes with their neighbors. However, this is not a good analogy for at least two reasons: firstly, the structure of a mesh is much more restrictive, as neighboring cells must not overlap, while the stencils can and usually do, and secondly, high quality meshes cannot always be constructed automatically, while stencil construction is considered an automatic part of the solution procedure, without the need for manual intervention.

There are several options for stencil construction. The first one is to choose all the nodes in a given radius. Especially for spatially variable node sets, this is usually expressed as a factor of the local spacing. Formally, the radial stencils for a node set $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ are defined as

$$I_{r,h}(i) = \{j; \|\boldsymbol{x}_i - \boldsymbol{x}_j\| \leq rh(\boldsymbol{x}_i)\}. \tag{3.5.1}$$

One downside of these types of stencils is that they vary in size. If the node set is quasi-uniform with respect to h, the variation is bounded, but we still need to ensure a minimum number of nodes in each stencil for local approximations to work. Another even simpler option is to use stencils of n closest nodes. This naturally adapts to the local spacing and ensures that stencils are of uniform size. However, the radii of the stencils can vary, and we have to somehow break potential ties for the n-th closest node. Both of these approaches are efficient to compute with a static spatial search structure such as a k-d tree or a search grid. Stencils of n closest nodes can be computed in $O(nN \log N)$ time.

A common addition to both stencil selection methods is to only consider nodes that are visible from the center node. Node $\boldsymbol{x}_j$ is (straight-line-)visible from $\boldsymbol{x}_i$ if the line segment from $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$ needs to be fully contained in Ω [BLG94]. This is often important for correctness of the PDE solution, as many physical quantities can only transfer through the domain, and having unrelated nodes in the stencils is equivalent to allowing the transfer of quantities.

More sophisticated stencil selection algorithms have also been developed [DO11]. One notable method is to use so called *balanced* stencils. Closest node stencils can become heavily one-sided especially in adaptive applications, where nodes can be denser on one side of the central node that the other [Mil12]. Balanced stencils ensure a certain number of nodes is present in all main direction, which helps with stability of the method. Symmetrization of stencils is also sometimes used, i.e. to update the stencils so that $\boldsymbol{x}_j \in S(\boldsymbol{x}_i)$ implies $\boldsymbol{x}_i \in S(\boldsymbol{x}_j)$ for all i, j, or at least for all interior nodes [PLP08].

As defined in the domain discretization, stencil indices need to be ordered. They can be ordered arbitrarily, although it is common to sort the nodes according to the distance from the central node – this also ensures that the central node itself is first. In this work, we will use stencils of n closest nodes, and we will assume that the indices are ordered according to the distance of stencil nodes from the central node. In particular, this means that the first stencil node is the center node itself.

Chapter 4

Adaptivity

Adaptive PDE solution procedures are indispensable in problems where the solution greatly varies within the computational domain, where only a particular region is of special interest or where varying precision in different parts of the domain is desired, and where using uniform discretization through the domain would be computationally wasteful or even completely unfeasible. A typical example is following a shock wave, where finer discretization is required to accurately model the more violent parts and coarser discretization can be used everywhere else. In time-depended problems, the discretization usually evolves in time, but adaptivity is very useful for steady-state problems as well, e.g. in presence of large solution gradients, singularities, or when modeling phenomena concentrated on a small part on the domain. These situations appear often in linear elastostatics, with a primer example being a contact of two bodies, where extremely high stress concentrations are present around the contact area.

Solving such problems using uniform discretization is intractable due to the amount of time and computational resources required to obtain a numerical solution with satisfactory precision. Instead, many different refinement techniques were developed to help solve such problems. When the problematic areas are known beforehand, the discretization can be constructed accordingly before solving the PDE. However, this is often not the case, and the refinement needs to be a part of the solution procedure, along with an indicator of solution quality to guide the refinement. A typical adaptive procedure for elliptic problems proceeds in iterations. The problem is solved on an initial discretization, the quality of the solution is evaluated and the discretization is refined where necessary, with the ultimate goal to ensure sufficient quality of the solution throughout the domain. The best version of this approach is known as *fully automatic adaptivity*, where the goal is to solve the problem without any human intervention. Research in this area has been ongoing for decades for more established methods, such as FEM [RPD06]. Fully automatic adaptivity is difficult to achieve and it requires a robust numerical method for solving PDE, good error indicators, algorithms for generating refined discretizations and is also difficult to implement and analyze. Strong form meshless methods have matured enough over the years and enough tools in the related research areas have been developed to make first attempts at fully automatic adaptivity possible.

The rest of this chapter is structured as follows: in Section 4.1 we discuss possible refinement techniques. Following that, in Section 4.2 we review previous work on error indicators. These two sections will serve as a basis to decide on the type of refinement

and the error indicator. Our adaptive procedure for elliptic problems will be discussed in Section 4.3. Using the described adaptive procedure, we will solve two classical problems for testing adaptive methods, the L-shaped domain in 2D and the Fichera corner in 3D. Finally, various 2D and 3D contact problems will also be solved in 4.5.

4.1 Types of refinement

Two conceptually different types of refinement are primarily in use when solving PDEs with classical methods, namely *h*-refinement and *p*-refinement, which are named accordingly to the variables used to represent the quantity they refine: *h*-refinement modifies the nodal spacing h and *p*-refinement modifies the approximation order p. When used together, they give rise to a popular method, called hp-FEM [BG92]. Isogeometric analysis supports both *h*-refinement via knot insertion and *p*-refinement via degree elevation, but also introduced a new type of refinement specific to isogeometric analysis, called *k*-refinement that combines knot insertion and degree elevation in one operation [HCB05].

In general, the solution accuracy is influenced by either the order of approximation or by modifying the underlying discretization. Varying the approximation order, i.e. *p*-refinement, can be performed in different ways. The two most common are to increase the stencil size [Ste+09] or to increase the number of basis function [Mil12], which gives rise to high order methods. For RBF-FD, high orders of accuracy are obtained relatively easily by increasing both the stencil size and the order of monomial augmentation [Bay+17]. As a demonstration of this, the Poisson problem (3.4.12) was solved on test domains Ω_2 and Ω_3 under uniform refinement with RBF-FD with $\phi(r) = r^3$ and with different orders of monomial augmentation m, similarly to [SSK19a]. Stencil sizes for $m = 2$ coincided with the case presented in 3.4.5 and the sizes for higher augmentation orders are listed in 4.1.

Table 4.1: Stencil sizes for higher order augmentation used to obtain results in Figure 4.1.

$d = 2$		$d = 3$	
m	n	m	n
2	13	2	25
4	32	4	78
6	60	6	173
8	185	8	345

The errors were measured in the same way as described in Section 3.4.5, as relative point-wise errors. The e_1, e_2 and e_∞ errors had the same general behavior so only the worst e_∞ error is shown in Figure 4.1. The order of convergence matches the augmentation order m. In 2D, $m = 6$ and $m = 8$ curves level off due to numerical precision, consistent with observations in [Fly+16]. The trade-off of high accuracy vs. computation efficiency is investigated in [JSK19]. The idea to use different orders locally to perform *p*-refinement in RBF-FD has been realized in a pre-print by Mishra et al. [Mis+20]. A related type of adaptivity for RBF-based method is also shape-based adaptivity, where the shape parameter of RBFs is varied [JDX14].

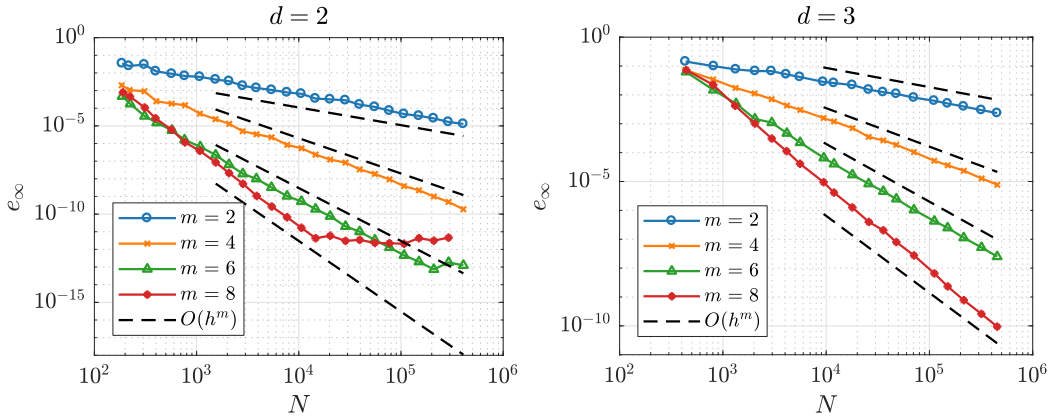


Figure 4.1: Errors with different orders of monomials augmentation.

The second approach is to modify the discretization of the domain. This includes *h*-refinement, where parts of the domain discretization are coarsened or made finer, or other approaches, such as *r*-refinement, which changes the nodal positions without changing the total number of nodes. This is often useful for evolutionary PDEs [Zeg98], but can also be used with other types. Successful *r*-refinement has been demonstrated in a meshless solution of elasticity problems with discrete least square method [ANA11].

Contrary to *r*-refinement, *h*-refinement in general changes the total number of nodes by either adding/removing nodes from the areas where so required or by creating an entirely new distribution of nodes. These techniques are known in mesh-based methods as *mesh enrichment* and *mesh regeneration* (or *re-meshing*), respectively [ZHZ93]. The enrichment *h*-refinement scheme is the most popular and has been successfully used in meshless solutions of linear elasticity problems with the meshless finite volume method [EFK15], the global RBF collocation method [Lib+08], RBF-FD method [SK19c], the point interpolation method [Tan+11]. It has also been successfully applied to transient problems [Ben+03; KŠ11; Jac+16].

Even though *h*-refinement is widely popular, its usage for scattered meshless node sets is rare, especially in 3D, as care needs to be taken when adding or removing nodes to avoid degrading the node set. This becomes even more apparent with fully automatic adaptivity, and can lead to typical complications associated with irregular nodal distributions [FZ07b; DH07]. Many authors opt for simpler refinement techniques, such as quad-tree based [KŠ11; Jac+16], or adding new nodes at midpoints between center and stencil nodes [Ben+03], or they revert back to mesh-based algorithms, using e.g. Voronoi diagrams [APP09; EFK15] or local Delaunay triangulations [PLP08].

In the context of RBF-FD, Davydov and Oanh [DO11] proposed a pure meshless *h*-refinement enrichment scheme and tested it on a 2D Poisson equation. The new nodes were added at the midpoints between a node and its stencil nodes (if appropriate), but the distributions degraded over time. The issues were addressed in a later paper, where a special stencil selection algorithm was introduced to improve stability [ODP17]. A similar *h*-refinement algorithm was used in [SK19b] to test the behavior of RBF-FD under extreme refinement with a ratio of more than 10^5 between the densest and coarsest parts of the discretization. Li et al. [Li+17] presented three-dimensional adaptive *h*-refinement examples using RBF-FD, but they used graded grids of regular nodes.

To avoid the problems caused by enrichment and to still be able to test the method on scattered data sets, we opted to use *h*-refinement with regeneration. This way, the generated node set is always of sufficient quality and fully automatic adaptivity can be achieved even in 3D [SK19c]. The algorithms for local enrichment can be developed after feasibility is demonstrated, and can be used to lower the computational costs.

4.2 Error indicators

An important part of the adaptive procedure is an indicator of solution quality, called an *error indicator*, that gives information about the regions where refinement or derefinement is needed, with the idea to refine where error is high and derefine where error is low. If the regions of high errors are known *a priori*, such as in the presence of known singularities, an appropriately spaced nodal distribution can be generated. However, this is not the case in general, and *a posteriori* error indicators that estimate the error of an already obtain solution are of interest. As a test, the difference between a closed form solution and the computed solution can be used, but an a posteriori error indicator should not rely on other knowledge, apart from the computed solution and the discretization that was used. A substitute for the closed form solution is to compute a higher order approximation, if possible, and compare the obtained solution to this higher order approximation instead [Ben+03]. However, these methods can be quite expensive. More primitive error indicators are the gradient-based indicators, which use the idea that high variations in the solution are good candidates for refinement. These indicators can be very simple and effective, but are not necessarily robust enough for practical applications. They can still be very useful for research purposes, when error indicators are not the main focus, such as in [DO11], which used the indicator based on the difference of solution values between the center and support node:

$$\hat{e}(\boldsymbol{x}_i, \boldsymbol{x}_j) = |u_h(\boldsymbol{x}_i) - u_h(\boldsymbol{x}_j)|, \tag{4.2.1}$$

where u_h is the numerical solution and $\boldsymbol{x}_j \in S(\boldsymbol{x}_i)$. An edge $\boldsymbol{x}_i\boldsymbol{x}_j$ was refined if $\hat{e}(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq \eta e_{\max}$, where $e_{\max} = \max_{\boldsymbol{x}_i \in X} \max_{\boldsymbol{x}_j \in S(x_i)} \hat{e}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for some $\eta \in (0, 1]$. Other error indicators with the same idea have also been used to compute errors in nodes instead of in edges, such as

$$\hat{e}(\boldsymbol{x}_i) = \left(\sum_{\boldsymbol{x}_j \in S(\boldsymbol{x}_i)} |u_h(\boldsymbol{x}_j) - \bar{u}(\boldsymbol{x}_i)|^2 \right)^{\frac{1}{2}}, \qquad \bar{u}(\boldsymbol{x}_i) = \frac{1}{n_i} \sum_{\boldsymbol{x}_j \in S(\boldsymbol{x}_i)} u_h(\boldsymbol{x}_j), \tag{4.2.2}$$

used in [KŠ11], that we will also use unless specified otherwise. This indicator can also be appropriately scaled or weighted so that stencil size and the magnitude of u_h do not have immediate influence on its values. In general, indicators that refine based on the magnitude of the solution gradient can produce undesirable results, such as over-refining relatively flat regions and under-refining the regions of high curvature [ODP17].

A posteriori error indicators are well developed for FEM and the most popular error indicator is the recovery-based Zienkiewicz and Zhu error indicator [ZZ87], also known as a ZZ indicator. The idea of a recovery based indicator is to “recover” a more accurate solution from the already computed one using appropriate post-processing, and then

use the difference between the obtained and the recovered solution as an indicator of an error between the obtained and the true solution. This approach has been extended especially to weak form meshless methods [RB05; Tan+11; EFK15], but also to some strong form methods [APP09; Hu+19], as well as to RBF-FD [ODP17]. The indicator presented in [Hu+19] can be directly generalized to RBF-FD, as it is based on the same phenomenon that causes discontinuities in k-nearest neighbors scattered interpolation, as shown e.g. in Figure 1.6. The error indicator is based on the solution gradient ∇u_h and is computed as

$$\hat{e}(\boldsymbol{x}_i) = \left(\sum_{\boldsymbol{x}_j \in S(\boldsymbol{x}_i) \setminus \{\boldsymbol{x}_i\}} (R[\nabla u_h](\boldsymbol{x}_j) - \nabla u_h(\boldsymbol{x}_j; \boldsymbol{x}_i)) \right)^{\frac{1}{2}}, \tag{4.2.3}$$

$$R[\nabla u_h](\boldsymbol{x}_j) = \frac{1}{n_j} \sum_{\boldsymbol{x}_k \in S(\boldsymbol{x}_j) \setminus \{\boldsymbol{x}_j\}} \nabla u_h(\boldsymbol{x}_k; \boldsymbol{x}_j), \tag{4.2.4}$$

where $\nabla u_h(\boldsymbol{x}_j; \boldsymbol{x}_i)$ is the value of the gradient of the RBF interpolant constructed over points $S(\boldsymbol{x}_i)$ evaluated at $\boldsymbol{x}_j$. The recovered gradient at $\boldsymbol{x}_j$ is the average of all the gradients as seen from the stencil nodes of $\boldsymbol{x}_j$. The error at $\boldsymbol{x}_i$ is the norm of all differences between the recovered gradient at $\boldsymbol{x}_j$ and the value as seen from $\boldsymbol{x}_i$. Contrary to indicators (4.2.1) and (4.2.2) which have time complexity $O(1)$ and $O(n)$, this indicator has time complexity $O(n^5)$ per node if computed naively or $O(n^4)$, if all values of $\nabla u_h(\boldsymbol{x}_k; \boldsymbol{x}_j)$ are precomputed. Instead of node-based indicators, Oahn et al. [ODP17] use RBF-FD with edge based indicator based on the directional derivatives:

$$\hat{e}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left| (u_h(\boldsymbol{x}_i) - u_h(\boldsymbol{x}_j)) - (\ell(\boldsymbol{x}_i; \boldsymbol{x}_i) - \ell(\boldsymbol{x}_j; \boldsymbol{x}_i)) \right|, \tag{4.2.5}$$

where $\ell(\cdot, \boldsymbol{x}_i)$ is a linear polynomial of the form $\ell(\boldsymbol{x}, \boldsymbol{x}_i) = a + \boldsymbol{b}^\mathsf{T}(\boldsymbol{x} - \boldsymbol{x}_i)$ that minimizes

$$\sum_{\boldsymbol{x}_j \in S(\boldsymbol{x}_i)} |u_h(\boldsymbol{x}_j) - \ell(\boldsymbol{x}_j, \boldsymbol{x}_i)|^2 \tag{4.2.6}$$

in the least squares sense. The time complexity of this indicator is $O(n)$ per node or $O(1)$ per edge.

A different type of error indicators are the residual-based error indicators, also used in FEM [BR78], which are more commonly generalized to a meshless setting in the context of least squares-based methods. This type of indicators is based on the residual of weighted least square computation, as given by (2.2.27). An estimator of this type is described by Sang-Hoon et al. [SKS03] for first order PDEs, and also by Afshar et al. [ANA11] for use in linear elasticity.

4.3 Adaptive solution procedure for elliptic problems

Our adaptive procedure will rely heavily on the node discretization algorithms developed in Chapter 3 that have the option to discretize domains according to an arbitrary spacing function h. This means that instead of directly adapting the domain discretization, the spacing function h will be adapted and a new domain discretization can be generated from it with. The usual procedure of directly modifying the domain discretization

essentially does the same thing, except that the spacing function h is never changed explicitly, but is instead implicitly changed when the domain is modified. This decoupling of adaptation and modification of domain discretizations allows us to test adaptivity strategies easier and without the need to develop algorithms for modification of domain discretizations.

As the adaptive procedure is iterative, the quantities will be denoted with an added superscript (j) corresponding to the iteration number j. Quantities with iteration number (0) are given as initial data or computed from the initial data before the first adaptive modification. It is possible that the initial solution is already of sufficient quality, in which case we will say that 0 adaptive iteration were needed. The adaptive solution procedure with this notation is described in pseudo-code as Algorithm 4.1.

Algorithm 4.1 Adaptive solution procedure.

Input: A boundary value problem $\mathcal{P}$.
Input: Computation domain $\Omega \subseteq \mathbb{R}^d$.
Input: Initial density function $h^{(0)} : \Omega \to (0, \infty)$.
Input: Global error tolerance $\tau \geq 0$.
Input: Maximal number of iterations $J_{\max}$.
Output: The numerical solution of the problem.

function ADAPTIVE_SOLVE($\mathcal{P}, \Omega, h^{(0)}, J_{\max}, \tau$)
  for $j \leftarrow 0$ **to** $J_{\max}$ **do**
    $\mathcal{D}^{(j)} \leftarrow$ DISCRETISE$(\Omega, h^{(j)})$ ▷ Discretizes domain Ω (see Chapter 3).
    $u^{(j)} \leftarrow$ SOLVE$(\mathcal{P}, \mathcal{D}^{(j)})$ ▷ Solves the problem (see Chapter 2).
    $\hat{e}^{(j)} \leftarrow$ ESTIMATE_ERROR$(\mathcal{D}^{(j)}, u^{(j)})$ ▷ Error indicator computation (see 4.2).
    if NORM$(\hat{e}^{(j)}) < \tau$ **then** ▷ Other quality measures can be used.
      return $u^{(j)}$
    end if
    $h^{(i+1)} \leftarrow$ ADAPT$(h^{(j)}, \mathcal{D}^{(j)}, \hat{e}^{(j)})$ ▷ Adapt the nodal spacing (see sec. 4.3.1).
  end for
  warning "Maximal number of iterations reached."
  return $u^{(J_{\max})}$
end function

The procedure is not the most efficient, as the whole domain discretization, solution and error indicator are computed anew each time. If modifications to h are restricted to some local regions (as is usually the case), nodes in those areas could be removed from the discretization and the empty space could be filled with the same fill algorithms, using the neighboring existing nodes as seed nodes. Then, the stencils of all new node should be computed, and the stencils of nearby nodes should be updated. New stencil weights should be computed for nodes whose stencils changed and a new solution must be computed, where the solution from the previous iteration may be used as an initial guess.

All these modification mainly affect the running time and not the final result. That is why we first decided to test the feasibility and behavior of adaptivity in RBF-FD before implementing any of the aforementioned improvements.

4.3.1 Spacing function modification

The spacing function $h^{(j)}$ is adapted based on the error indicator values $\hat{e}^{(j)}$ for the solution $u^{(j)}$ computed on discretization $\mathcal{D}^{(j)}$ with the node set $X^{(j)}$. The adaptation will be subject to additional parameters that limit the rate and amount that nodal spacing can change in each iteration:

- $h_{\mathrm{r}}, h_{\mathrm{d}} \colon \Omega \to (0, \infty)$: the lower and upper bounds on nodal spacing h,
- $\varepsilon_{\mathrm{r}}, \varepsilon_{\mathrm{d}}$: nonnegative real numbers that represent upper and lower bounds for local error thresholds,
- $\alpha_{\mathrm{r}}, \alpha_{\mathrm{d}}$: real numbers greater than one that represent bounds for density increase factors.

All quantities with subscript 'r' are related to refinement and quantities with subscript 'd' are related to derefinement.

The nodal spacing function is adapted by locally increasing or decreasing the spacing by a certain factor. Define $h_i^{(j)} := h^{(j)}(\boldsymbol{x}_i^{(j)})$ as the local nodal spacing around node $\boldsymbol{x}_i^{(j)}$. This spacing is modified by defining new local nodal spacing values as

$$h_i^{(j+1)} := \max\{\min\{h_i^{(j)}/f_i^{(j)}, h_{\mathrm{d}}(\boldsymbol{x}_i^{(j)})\}, h_{\mathrm{r}}(\boldsymbol{x}_i^{(j)})\}, \tag{4.3.1}$$

where the density increase factor $f_i^{(j)}$ is defined as

$$f_i^{(j)} = \begin{cases} 1 + \frac{\varepsilon_{\mathrm{d}} - \hat{e}_i^{(j)}}{\varepsilon_{\mathrm{d}} - m^{(j)}}(\frac{1}{\alpha_{\mathrm{d}}} - 1), & \hat{e}_i^{(j)} \leq \varepsilon_{\mathrm{d}}, \quad \text{i.e. decrease the density} \\ 1, & \varepsilon_{\mathrm{d}} < \hat{e}_i^{(j)} < \varepsilon_{\mathrm{r}}, \quad \text{i.e. no change in density} \\ 1 + \frac{\hat{e}_i^{(j)} - \varepsilon_{\mathrm{r}}}{M^{(j)} - \varepsilon_{\mathrm{r}}}(\alpha_{\mathrm{r}} - 1), & \hat{e}_i^{(j)} \geq \varepsilon_{\mathrm{r}}, \quad \text{i.e. increase the density} \end{cases}. \tag{4.3.2}$$

The values $\hat{e}_i^{(j)}$ are indicator values at nodes $\boldsymbol{x}_i^{(j)}$ and $M^{(j)}$ and $m^{(j)}$ represent the extrema of the error indicator values:

$$M^{(j)} = \max_{\boldsymbol{x}_i^{(j)} \in X^{(j)}} \hat{e}_i^{(j)}, \quad m^{(j)} = \min_{\boldsymbol{x}_i^{(j)} \in X^{(j)}} \hat{e}_i^{(j)}. \tag{4.3.3}$$

The values $f_i^{(j)}$ are defined so that $\frac{1}{\alpha_{\mathrm{d}}} \leq f_i^{(j)} \leq \alpha_{\mathrm{r}}$ always holds, as $\frac{\hat{e}_i^{(j)} - \varepsilon_{\mathrm{r}}}{M^{(j)} - \varepsilon_{\mathrm{r}}} \in [0, 1]$ and $\alpha_{\mathrm{r}} \geq 1$. If the node is on the upper error threshold, i.e. $\hat{e}_i^{(j)} = \varepsilon_{\mathrm{r}}$, then the factor $f_i^{(j)}$ equals 1 and the density will stay the same, ensuring compatibility with the case when $\hat{e}_i^{(j)} < \varepsilon_{\mathrm{r}}$. Additionally, if the node has the highest indicated error, i.e. $\hat{e}_i^{(j)} = M^{(j)}$, the density increase factor will be maximal, i.e. α_{r}. Symmetric observations hold for the derefine case. Setting $\alpha_{\mathrm{r}} = 1$ or $\alpha_{\mathrm{d}} = 1$ disables refinement or derefinement, respectively. The same can also be achieved by setting $\varepsilon_{\mathrm{d}} = 0$ or $\varepsilon_{\mathrm{r}} = \infty$.

The value α_{r} limits that the spacing at a given node can be reduced for at most a factor of α_{r}, and symmetrically for α_{d}. The factors α_{r} and α_{d} are called refinement and derefinement *aggressiveness*. They limit the rate of change of spacing h between two iterations, but still allow for exponential decrease of the form $h_i^{(j)} = h_i^{(0)}/\alpha_{\mathrm{r}}^j$ if needed in certain areas. The impact and role of the adaptivity parameters introduced in this section is further investigated later, in Section 4.4.3.

Figure 4.2 shows the plot of the density increase factor f_i, and the behavior of the adaptive procedure on sample value of the error indicator. The nodes in the discretization are split into three categories: where the density should be increased due to high indicated error, where the density should stay the same, because error is in the acceptable range, and where the nodes should be sparser, because the error is unnecessarily low.

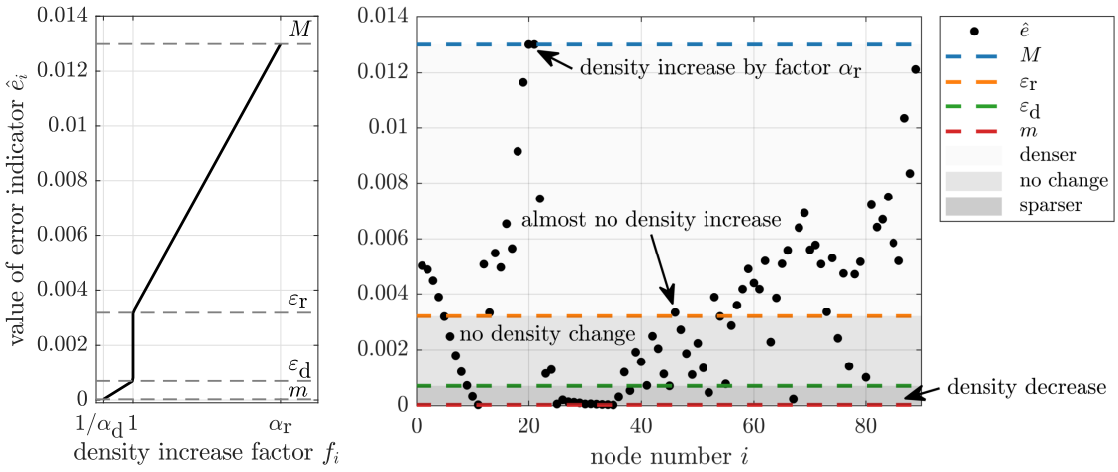


Figure 4.2: Construction of the new spacing function.

Besides limits on the rates of change of h during iterations, global refinement and derefinement limits h_r and h_d are also enforced in (4.3.1). The initial spacing $h^{(0)}$ is often chosen as the derefinement limit, i.e. $h_d = h^{(0)}$, while the refinement limit is often just a small constant spacing, 100 or 1000 times smaller than the minimal initial spacing. It can also be set to 0 to impose no limit on refinement.

Until now, we only defined the new values $h_i^{(j+1)}$, not the whole function $h^{(j+1)}$, which is needed to run the node positioning algorithms. However, obtaining $h^{(j+1)}$ is a matter of scattered data interpolation. We opted to use k-Sheppard's interpolation (see Section 1.3.1 on page 12) as a simple and computationally efficient method. The number of nearest neighbors k was equal to the number of nodes included in the stencils.

The domain discretization was constructed by first discretizing the boundary and then the interior, using the boundary nodes as seed nodes. Additionally, the algorithm was first run with spacing $10h$ and then with spacing h to achieve a more uniform nodal distribution on a local level.

4.3.2 Minimal adaptive example

We will first demonstrate the adaptive procedure described in Algorithm 4.1 on a simpler problem of function approximation. The function

$$g(x) = 3(1-x)^2 \exp(-x^2) + 3\exp(-4(x-1)^2) \tag{4.3.4}$$

is approximated on $\Omega_1 = [-3, 3]$ using 12-MLS (see Section 1.3.2 on page 13) with basis $\{1, x\}$, 12 neighboring nodes and a Gaussian weight $\exp(-r^2/d^2)$, where d is the distance to the farthest support node. The resulting approximation on j-th iteration is denoted by $\hat{g}^{(j)}$.

The initial spacing function is

$$h^{(0)}(x) = h_0(1 + 25|3 + x|), \quad h_0 = 0.005, \tag{4.3.5}$$

which results in a dense distribution of nodes on the negative part of the x-axis and a sparse distribution on the positive part, as shown in 4.3, iteration 0.

To demonstrate the effect of refinement and derefinement limits, we set $h_\mathrm{r} = 0.02$ and $h_\mathrm{d} = 0.05$. Values $\alpha_\mathrm{r} = \alpha_\mathrm{d} = 5$ will be used for refine and derefine aggressiveness. For scattered interpolation of the new spacing values, 12-Sheppard's interpolation was used.

The aim of the adaptive procedure is to adaptively refine the distribution to reduce the true error $e^{(j)} = \|g - g^{(j)}\|_1$ below a certain threshold τ. During the iteration, the error $e^{(j)}$ will be approximated using

$$e^{(j)} = \int_{\Omega_1} |g(x) - g^{(j)}(x)|dx \approx \sum_{x_i^{(j)} \in X^{(j)}} |g(x_i^{(j)}) - g^{(j)}(x_i^{(j)})|\, h^{(j)}(x_i^{(j)}) \tag{4.3.6}$$

To make sure that error is below the threshold τ, we can require that $|g(x_i^{(j)}) - g^{(j)}(x_i^{(j)})|h(x_i^{(j)}) < \tau/N^{(j)}$. This means that we can use

$$\hat{e}_i^{(j)} = N^{(j)}|g(x_i^{(j)}) - g^{(j)}(x_i^{(j)})|h(x_i^{(j)}) \tag{4.3.7}$$

as the error indicator and the condition $\frac{1}{N^{(j)}} \sum_{i=1}^{N^{(j)}} \hat{e}_i^{(j)} < \tau$ on line 6 of Algorithm 4.1 as the threshold check. The error indicator (4.3.7) is only used for demonstration purposes, as it includes the analytical solution g and would be unfeasible in practice.

We will use $\tau = 10^{-1}$, $\varepsilon_\mathrm{r} = \tau$ and $\varepsilon_\mathrm{d} = 10^{-2}$. If error indicators in all nodes are below ε_r, then the total error must be below τ as well. However, as the error used in the exit condition is an approximation itself, the true error might be different. It is also common to set ε_r lower than τ, as keeping it the same might cause the procedure to converge towards it rather slowly, but this was not a problem in this simple case.

Figure 4.3 shows the progress of the adaptive iteration. During the progress of the iteration, the number of total nodes $N^{(j)}$ was tracked, along with the action taken for each node. We will use $N_r^{(j)}$, $N_{r,\mathrm{lim}}^{(j)}$, $N_s^{(j)}$, $N_d^{(j)}$ and $N_{d,\mathrm{lim}}^{(j)}$ to denote the number of nodes that were successfully refined; refined, but hit limit h_r; kept the same; derefined; and derefined, but hit limit h_d, respectively. This categorizes all the nodes, so that $N^{(j)} = N_r^{(j)} + N_{r,\mathrm{lim}}^{(j)} + N_s^{(j)} + N_d^{(j)} + N_{d,\mathrm{lim}}^{(j)}$ must hold. Additionally, we also kept track of the current approximation for the error $e^{(j)}$, as well as the true error, computed on a much denser independent node set. The recorded data is shown in Table 4.2.

The initial node distribution is constructed in such a way, that it is too dense on the negative part of the x-axis and too sparse on the positive part. Consequently, the nodes on the left are derefined, and the nodes on the right are refined, with the density factor proportional to the local error. In the first iteration, 20 nodes are derefined, of which 14 hit the lower bound h_d, and 17 nodes were refined. In the next iteration, more nodes were refined and less derefined, and in the final iteration some nodes even hit the refinement limit h_r. Throughout the iteration, both the estimate of the error and the error itself decrease. Figure 4.4 shows the final nodal spacing and how it adapted to the absolute

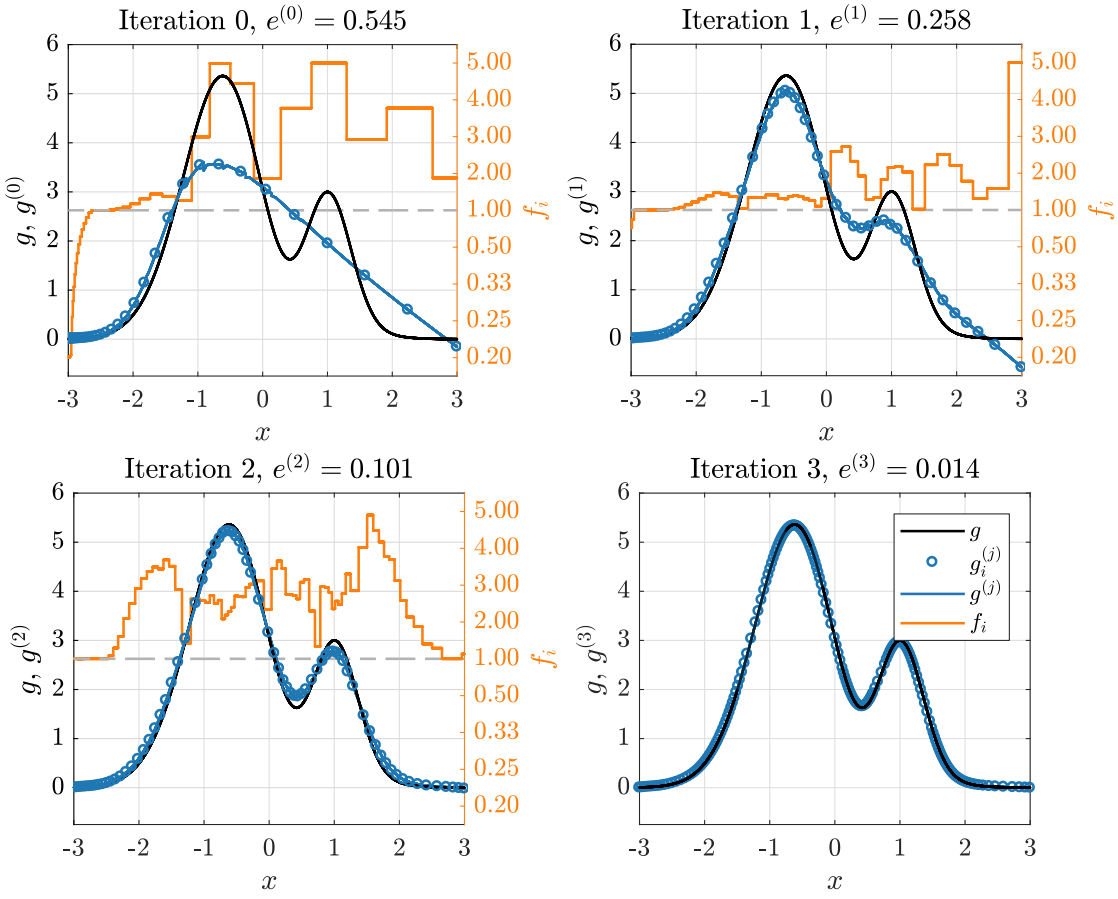


Figure 4.3: Adaptive function approximation. The left axis shows the approximated function and the approximant, and the right axis shows the density increase factors f_i. Values above the dashed line represent refinement and values below represent derefinement.

value of the second derivative. This is due to the fact that the approximation was linear, and the second derivative is the leading term in local truncation error. Besides the second derivative, nodal spacing is also increased near the boundary, where approximation errors are also usually larger due to one-sided stencils.

4.4 Classical problems

We will first solve two classical Poisson problems designed to test adaptivity, the L-shape problem and the Fichera corner. Different variations of these problems can be found in the literature, all using the same domain shape, but with different boundary conditions. We will use the versions where the solution is known in closed form and both Dirichlet and Neumann boundary conditions will be used. Some preliminary results on the RBF-FD adaptivity and the L-shape problem were presented at the international conference on Boundary Elements and other Mesh Reduction Methods (BEM/MRM 42) [SK19b]. A similar problem was solved recently using local strong form methods with MLS-based stencils by Hu, Trask, Hu and Pan [Hu+19].

Table 4.2: Number of nodes and errors during the course of adaptive iteration shown in Figure 4.3.

j	$N^{(j)}$	$\frac{1}{N^{(j)}}\sum_{i=1}^{N^{(j)}}\hat{e}_i^{(j)}$	$e^{(j)}$	$N_r^{(j)}$	$N_{r,\text{lim}}^{(j)}$	$N_s^{(j)}$	$N_d^{(j)}$	$N_{d,\text{lim}}^{(j)}$
0	41	3.2700	0.5454	17	0	4	6	14
1	54	1.6352	0.2576	42	0	7	1	4
2	78	0.6090	0.1010	46	18	12	0	2
3	190	0.0862	0.0141	/	/	/	/	/

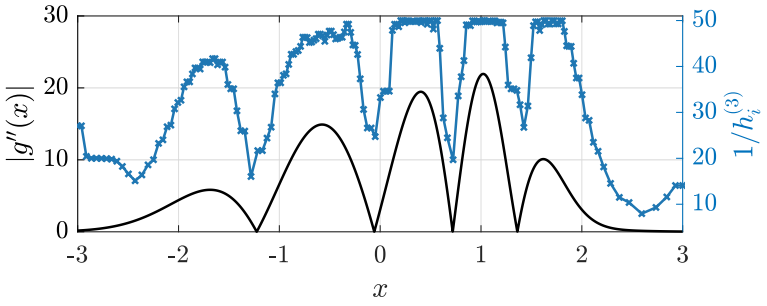


Figure 4.4: Final nodal spacing values and adaptation to the second derivative of the approximated function. The imposed refinement limit of $50 = 1/h_\text{r}$ is also clearly visible.

4.4.1 *L*-shape domain

The 2D test problem is the L-shape problem, described in e.g. [Dem06, p. 234]. Formally, the problem reads

$$\begin{aligned} \nabla u &= 0 && \text{in } \Omega_L, \\ u &= u_L && \text{on } \Gamma_d, \\ \frac{\partial u}{\partial \vec{n}} &= \vec{n} \cdot \nabla u_L && \text{on } \Gamma_n, \end{aligned} \tag{4.4.1}$$

where $\Omega_L = ([-1,1] \times [-1,-1]) \setminus ([0,1] \times [-1,0])$, the boundaries Γ_d and Γ_n are shown in Figure 4.5 and u_L is given in polar coordinates as $u_L(r,\theta) = r^{\frac{2}{3}} \sin(\frac{2}{3}\theta)$, where $\theta \in [0, 2\pi)$. It can be easily verified that $\nabla^2 u_L = 0$ and u_L is indeed the solution of problem (4.4.1). This problem is interesting for adaptivity, because despite the solution being relatively tame, its gradient has a singularity at the origin, which can cause problems with convergence. The solution and its gradient are shown in Figure 4.5.

Initially, we try to solve the problem with uniform refinement, using RBF-FD with polyharmonic RBFs $\phi(r) = r^3$ and monomials augmentation of order 2. Stencils of 13 closest nodes were used.

The error between the analytical solution u and the numerical solution u_h was measured on an independent grid of points G, denser than the densest spacing used. The

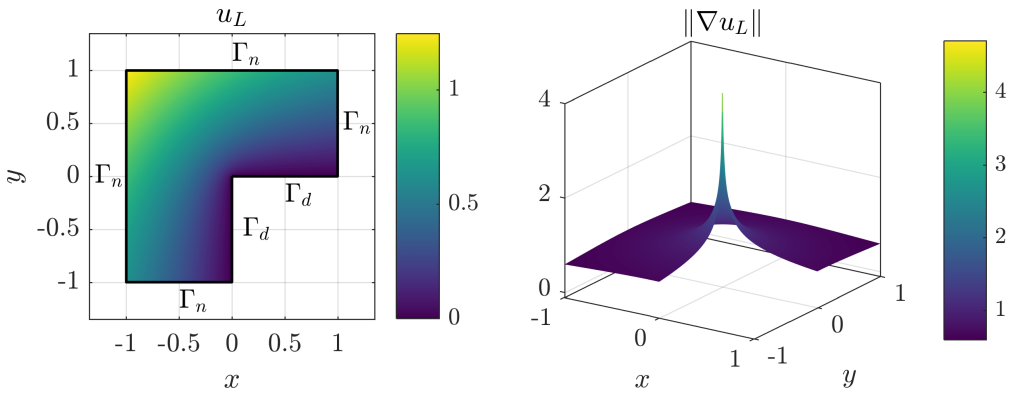


Figure 4.5: The solution of the L-shape problem (4.4.1) and the norm of its gradient.

same error norms as in Section 3.4.5 were used:

$$e_1 = \|u_h - u\|_{G,1} / \|u\|_{G,1}, \quad \|y\|_{G,1} = \sum_{\boldsymbol{p}_i \in G} |y(\boldsymbol{p}_i)|, \tag{4.4.2}$$

$$e_2 = \|u_h - u\|_{G,2} / \|u\|_{G,2}, \quad \|y\|_{G,2} = \sqrt{\sum_{\boldsymbol{p}_i \in G} |y(\boldsymbol{p}_i)|^2}, \tag{4.4.3}$$

$$e_\infty = \|u_h - u\|_{G,\infty} / \|u\|_{G,\infty}, \quad \|y\|_{G,\infty} = \max_{\boldsymbol{p}_i \in G} |y(\boldsymbol{p}_i)|. \tag{4.4.4}$$

As the solution u_h is known only in computational nodes, RBF interpolation is used to get intermediate values at $\boldsymbol{p} \in G$. At $p \in G$, a RBF interpolant with cubic polyharmonics and linear augmentation was constructed over a stencil of 13 closest nodes and its value at p was used as $u_h(\boldsymbol{p})$.

Figure 4.6 shows errors of numerical solution to the L-shape problem under uniform refinement. The convergence order is not the expected $O(h^2)$ but instead looks more like $O(h)$.

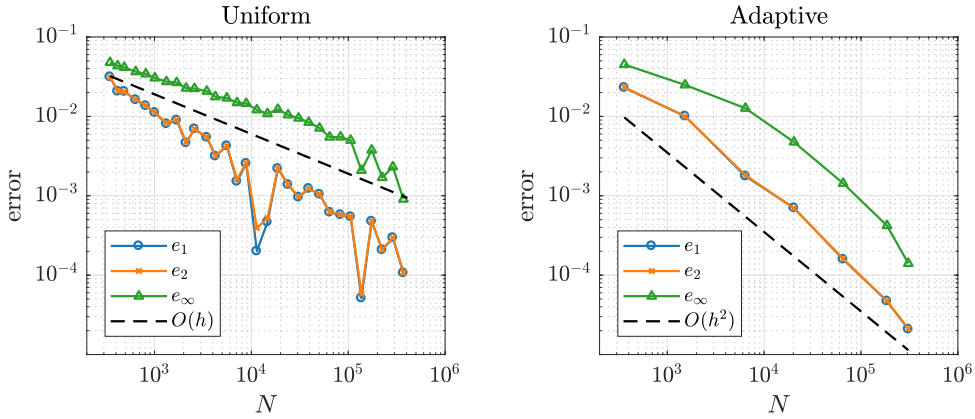


Figure 4.6: Errors of numerical solution to the L-shape problem (4.4.1) under uniform and adaptive refinement.

The problem was also solved adaptively with the same setup, starting with the same uniform density as in the uniform case, $h^{(0)}(\boldsymbol{x}) = 0.09$. We used $\varepsilon_\mathrm{r} = 10^{-2}, \alpha_\mathrm{r} =$

$3, h_{\mathrm{r}} = 0, \varepsilon_{\mathrm{d}} = 0, h_{\mathrm{d}} = 0.05$ to adapt the nodal density and the global tolerance was set to $\tau = 10^{-2}$. The gradient based error indicator (4.2.2) was used. The errors during the course of adaptive iteration are shown in Figure 4.6, and the results are decidedly better than under uniform refinement. The iteration stopped after 5 iterations.

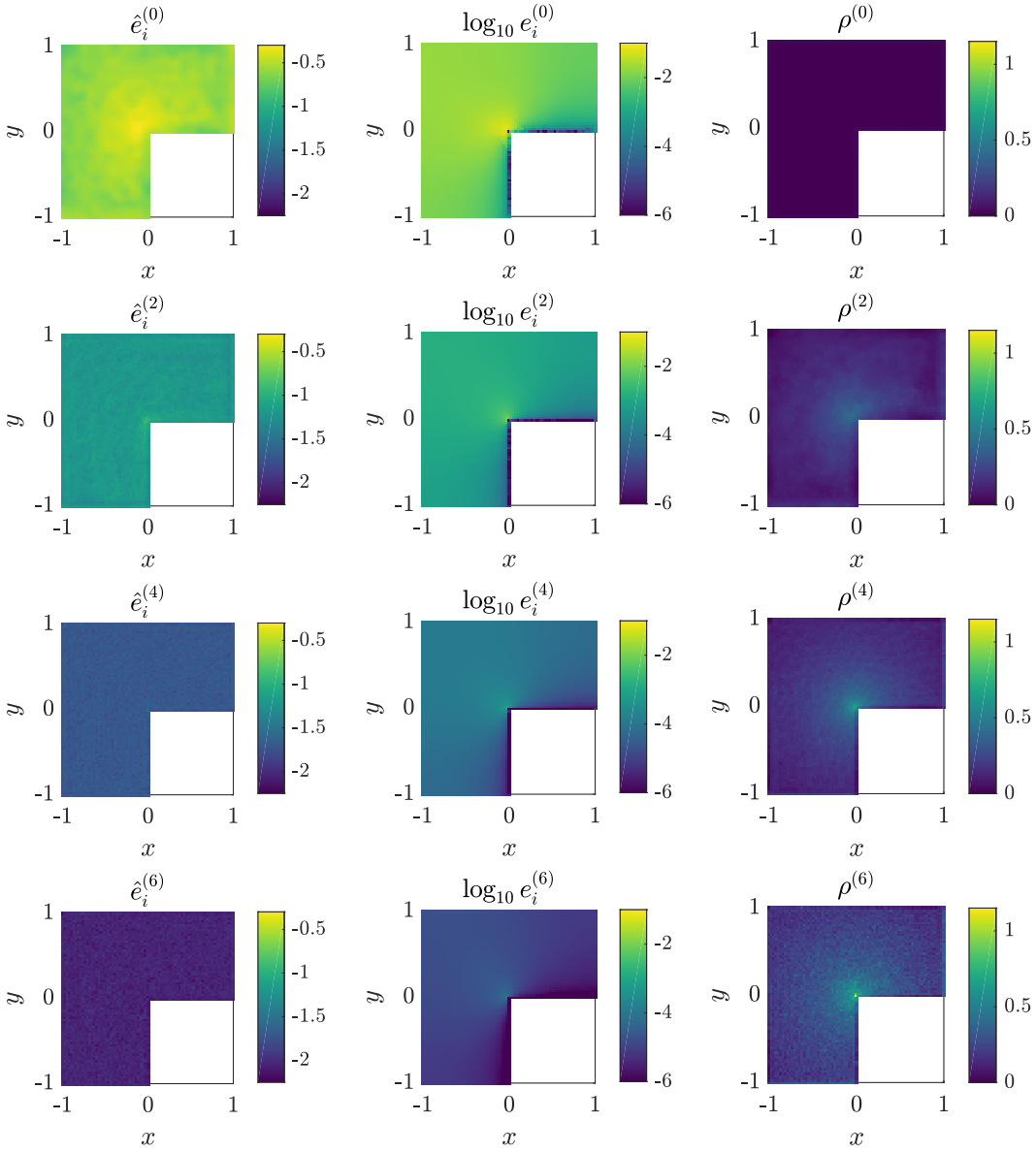


Figure 4.7: Values of the error indicator $\hat{e}_i^{(j)}$, true error values $e_i^{(j)}$ and nodal density $\rho_i^{(j)}$ during the course of adaptive iteration while solving the L-shape problem (4.4.1). Rows from top to bottom are for iterations $j = 0, 2, 4, 6$.

Besides the overall error, the errors $e_i^{(j)} = |u(\boldsymbol{x}_i^{(j)}) - u^{(j)}(\boldsymbol{x}_i^{(j)})|$ and the values of the error indicator $\hat{e}_i^{(j)}$ were also recorded. Additionally, local nodal density $\rho^{(j)}$ was also recorded, defined as

$$\rho^{(j)}(\boldsymbol{x}) = -\log_{10} \frac{h^{(j)}(\boldsymbol{x})}{\max_{\boldsymbol{y} \in \Omega_L} h^{(j)}(\boldsymbol{y})}. \tag{4.4.5}$$

This helps us visualize the variability in nodal spacing. Value $\rho(\boldsymbol{x}) = 0$ represents that

spacing around $\boldsymbol{x}$ is the coarsest in the domain, and value of $\rho(\boldsymbol{x}) = 1$ means that spacing at $\boldsymbol{x}$ is 10 times as dense as in the coarsest part. Plots of $\hat{e}_i^{(j)}$, $e_i^{(j)}$ and $\rho^{(j)}$ for even iterations are shown in Figure 4.7. Each row corresponds to one iteration, for $j = 0, 2, 4, 6$ with the first row showing the results on the initial uniform nodal distribution.

We can see that the nodal spacing adapts to the error distribution and the nodal density is highest near $(0, 0)$. As the nodal spacing adapts, both the error indicator and the error becomes more and more uniform throughout the domain.

4.4.2 Fichera's corner

The classical 3D domain is the Fichera's corner, described in e.g. [Kur+08, p. 112]. Formally, the problem is defined as

$$\begin{aligned} \nabla u &= 0 && \text{in } \Omega_F, \\ u &= u_F && \text{on } \Gamma_d, \\ \frac{\partial u}{\partial \vec{n}} &= \vec{n} \cdot \nabla u_F && \text{on } \Gamma_n, \end{aligned} \tag{4.4.6}$$

where $\Omega_F = [-1, 1]^3 \setminus [0, 1]^3$, the boundaries Γ_d and Γ_n are shown in Figure 4.8 and u_F is given in spherical coordinates as $u_L(r, \theta, \varphi) = r^{\frac{1}{2}}$. It can be easily verified that $\nabla^2 u_L = -\frac{3}{4} r^{-\frac{3}{2}}$ and $\vec{n} \cdot \nabla u_L = \frac{1}{2} r^{-\frac{3}{2}} \vec{r} \cdot \vec{n}$. Similarly to the L-shape problem, this problem is interesting for adaptivity due to high gradients around the origin. The solution and its gradient are shown in Figure 4.8.

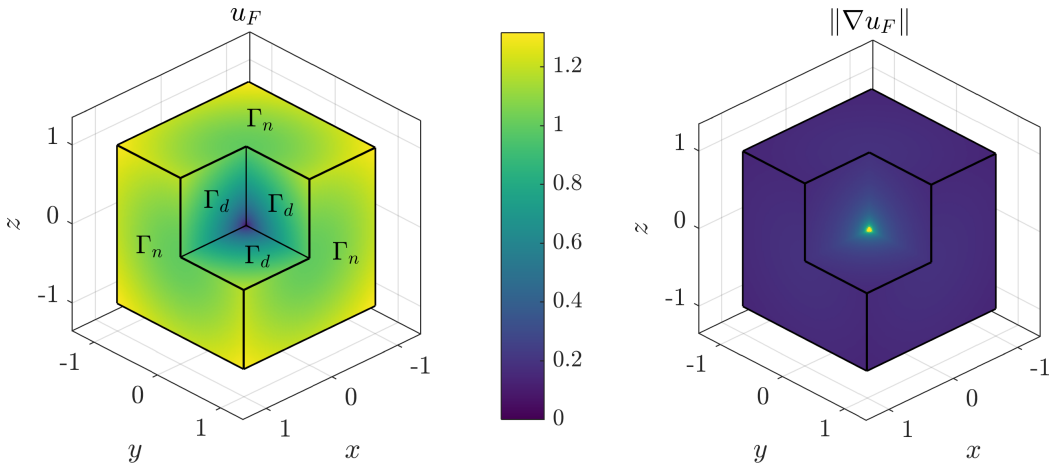


Figure 4.8: Geometry of the Fichera's corner with marked Dirichlet and Neumann boundaries. The hidden boundaries are also Neumann. The left plot shows the solution and the right the norm of the gradient.

The problem is initially solved with uniform node distributions using PHS RBF-FD with $\phi(r) = r^3$ and augmentation of order 2 with stencils of 25 closest nodes. The errors were measured similarly to those in the L-shape problem, but to ensure high enough density of sample evaluation points, they were only sampled around the origin on $[-\frac{1}{20}, \frac{1}{20}]^3 \cap \Omega_F$. Figure 4.9 shows errors under uniform refinement. The e_∞ error converges even slower than $O(h)$ and the other errors show convergence of order around $O(h)$.

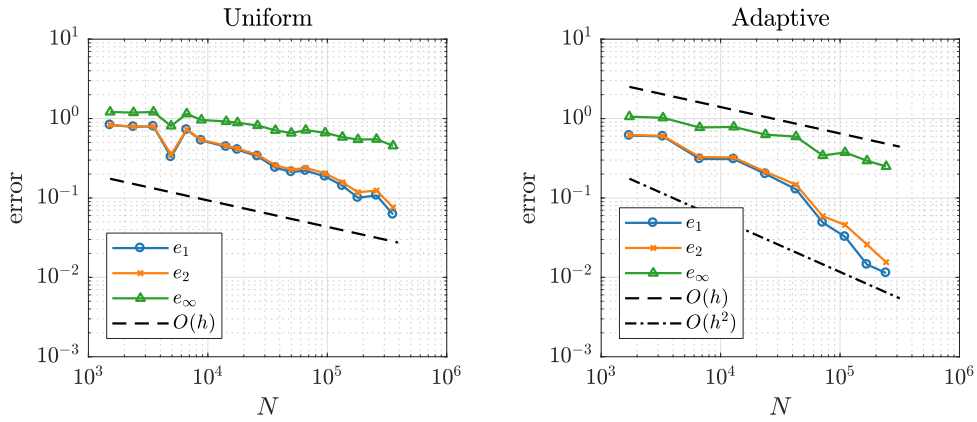


Figure 4.9: Errors of numerical solution to the Fichera problem (4.4.6) under uniform and adaptive refinement.

The problem was then solved adaptively with the same numerical setup, with initial uniform spacing $h^{(0)}(\boldsymbol{x}) = 0.16$. We used $\varepsilon_\mathrm{r} = 0.05, \alpha_\mathrm{r} = 1.5, h_\mathrm{r} = 0, \varepsilon_\mathrm{d} = 0, h_\mathrm{d} = 0.16$ to adapt the nodal density and the global tolerance was set to $\tau = 0.075$. The errors during the course of adaptive iteration are shown in Figure 4.9. The iteration stopped after 10 iterations. The results initially agree with uniform refinement, but start to improve as the number of nodes increases.

4.4.3 Analysis of adaptivity parameters

The adaptive procedure as described in Section 4.3 has free parameters $h_\mathrm{r}, h_\mathrm{d}, \alpha_\mathrm{r}, \alpha_\mathrm{d}, \varepsilon_\mathrm{r}$, and ε_d. The upper and lower limits on local spacing h_r and h_d serve mostly practical purposes, similar to the iteration limit J_max. Setting $h_\mathrm{d} > 0$ is important to not derefine regions near Dirichlet conditions indefinitely, where the error is usually small, and $h^{(0)}$ is often a good choice. Parameter h_r can often be set to 0, if there are no problems with internodal distances getting too small. In fact, setting it to a nonzero value without a good reason can in fact harm the adaptive procedure, as shown in the model example in Figure 4.4, where it would be beneficial to have a denser distribution of nodes in high-error areas, and the nodal density hit the bound h_r.

Choosing the values of ε_r and ε_d is related to the specific error indicator and the problem at hand. It is recommended to not set the values too close together, as this may cause the adaptive procedure to oscillate between refinement and derefinement. It is also advised to set the value of ε_r in such a way, that the chosen norm of the indicator would tend towards a value lower than τ. Setting it in so that it goes to τ specifically might unnecessarily increase the number of iterations required as the solution will try to asymptotically approach τ instead just going past it quickly.

The most interesting are the roles of α_r and α_d, which are symmetric in nature. To analyze the role of α_r, we solve the L-shape problem from Section 4.4.1 with the same parameters as before, and only vary α_r from just above 1 to 20. This allows us to observe the effect of the aggressiveness on the adaptive procedure. The maximal number of iterations was set to $J_\mathrm{max} = 500$ and was never reached. Figure 4.10 shows errors, node counts, maximal density, execution time and number of iterations for each analyzed α_r.

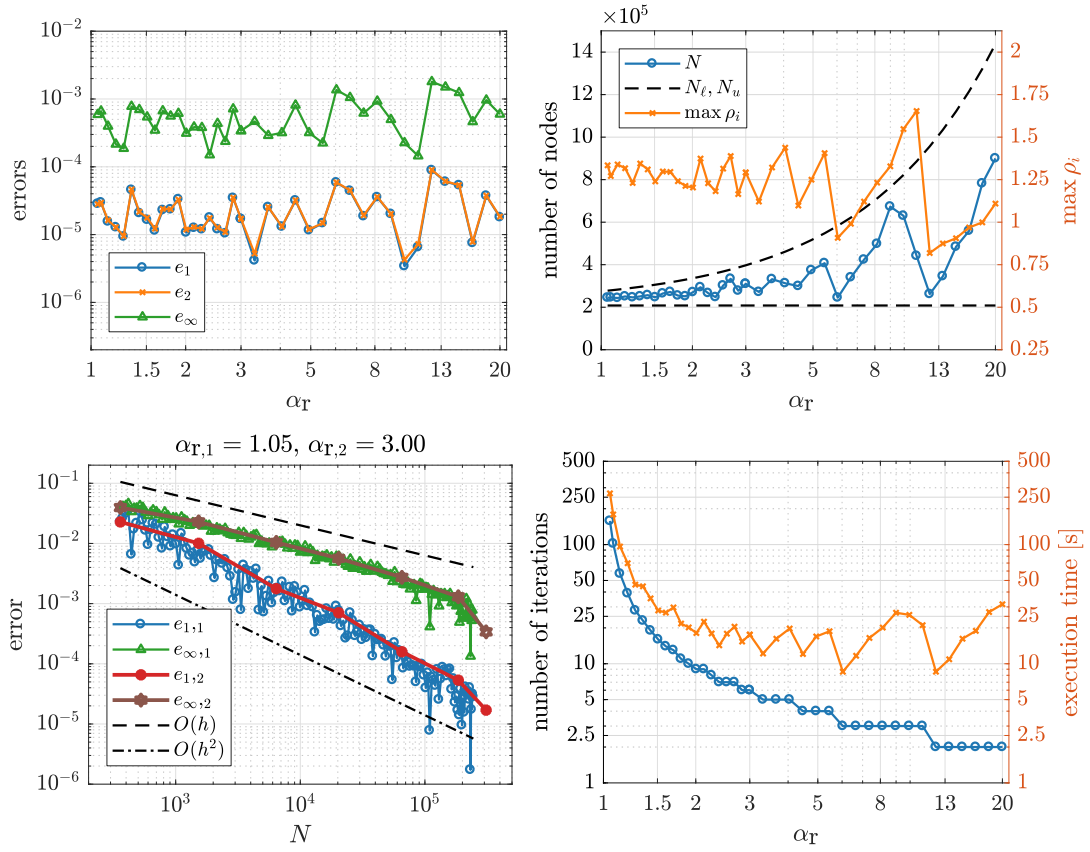


Figure 4.10: Behavior of the adaptive procedure with respect to α_r. The plot in the top left corner shows the errors of the final solutions obtained with adaptive iterations for different α_r. Similarly, the plot in the top right shows the final number of nodes and maximal density, as well as the upper and lower bounds N_ℓ and N_u, as defined in further analysis. The bottom right plot shows the number od adaptive iterations required and the total computational time. The bottom left plot shows the errors during the course of the adaptive iterations for two different values of α_r.

The errors and nodal densities of the obtained solutions remain roughly constant as α_r is varied, meaning that different choices of α_r only affect the path we choose to get to the solution of high enough quality, and does not impact the final quality in general. This is further illustrated by showing two adaptive iterations with α_r very close to 1 and another with larger α_r. They both obtain sequences of solutions with similar errors, but the first one needs 158 iterations and the second one needs 6. We can see that the number of necessary iterations decreases with increasing α_r (note that the both axes are logarithmic). The execution time follows the same general pattern, especially for smaller α_r, where the number of nodes does not change that rapidly and the total time is dominated by the number of iterations. However, it is not advisable to keep α_r too large, as indicated by the plot showing the final number of nodes. Large α_r can cause a significant increase on the number of nodes generated. Depending on the specific values of ε_r and α_r this increase may be just enough to pass the error threshold τ, making the iteration successful with almost minimal number of nodes. However, if the number of nodes falls just short of reaching the threshold, the next iteration overshoots the number

of nodes significantly. The chances and the magnitude of such overshoots increase with large α_r, as evidenced by higher spikes in the plot.

The expected number of nodes can be estimated after this analysis. With very small α_r we come very close to the number of nodes required to achieve the desired tolerance, and this is taken as the lower bound N_ℓ. The upper bound is dependent on α_r, and considering the unlucky case, where the second-to-last iteration is just below the required number of nodes N_ℓ, the next iteration can generate up to $\alpha_\mathrm{r} N_\ell$ nodes. In practice however, not all nodes get refined with factor α_r, but this instead depends on the distribution of the error. The upper bound in Figure 4.10 is obtained by assuming that the number of nodes increases as if only one fourth of the nodes was fully refined, plotting $N_u = N_\ell(1 + (\alpha_\mathrm{r} - 1)/4)$. Both bounds are also plotted with slight offset to not overlap the original data points. The spikes in the number of nodes also affect the running time and cause larger deviations in nodal density, which is otherwise around 1.25, similar to the final iteration in Figure 4.7, meaning that the densest region is around 17 times denser than the maximal internodal distance.

Based on this analysis the recommendation is to choose the initial α_r somewhere around 3 and then adjust it to the specific problem. A more important guideline is that smaller α_r can be used to test the behavior more reliably, and the α_r can be increased to improve computational time. Increasing α_r too much can lead to unnecessary large nodal distributions, and can even cause the adaptive procedure to fail by too eagerly refining the wrong regions [SK19c].

4.5 Contact problems

As the focus of this section is not contact modeling, but testing adaptivity, the contact will not be simulated with a full iterative simulation of two deforming bodies, but instead as a suitable analytically computed traction distribution, concentrated on a small portion of the boundary.

This will allow us to compare our results to the closed form solutions when available, but also to model some cases of practical interest. The results presented in this section are based on our paper [SK19c], but are computed with a different numerical setup.

4.5.1 Linear elasticity

The linear theory of elasticity describes the deformation of bodies under stress [Sla12; Sad14]. The main quantities of interest are the displacement $\vec{u}$ and the stress σ. The body is described as a set Ω and a displacement is a vector field $\vec{u}\colon \Omega \to \mathbb{R}^3$, describing that the point $\boldsymbol{x}$ moves to $\boldsymbol{x} + \vec{u}(\boldsymbol{x})$ under load. The stress is represented by a symmetric 2nd order tensor, so that $\sigma(\boldsymbol{x})\vec{n}(\boldsymbol{x})$ represents the traction on the (possibly virtual) surface with normal $\vec{n}$ at a point $\boldsymbol{x}$.

The equations of motion are derived from the Cauchy momentum equation

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} = \nabla \cdot \sigma + \vec{f}, \tag{4.5.1}$$

where ρ is the material density and $\vec{f}$ is the body loading force. The first linearization

used by the theory comes from the expression used to compute strains ε from displacements, namely

$$\varepsilon(u) = \frac{1}{2}(\nabla\vec{u} + (\nabla\vec{u})^{\mathsf{T}}), \tag{4.5.2}$$

which is only valid for small $\vec{u}$. What remains is to connect σ and ε using a constitutive relation, on this case the Hooke's law, which reads

$$\sigma = C : \varepsilon \tag{4.5.3}$$

in its general form, where C is a fourth order tensor. This is also a linearization, and is only valid for small strains.

The materials under question are often homogeneous and isotropic, in which case ρ is constant and the Hooke's law simplifies to

$$\sigma = \lambda \operatorname{tr}(\varepsilon) I + 2\mu\varepsilon, \tag{4.5.4}$$

where λ and μ are the Lamé parameters and I is the identity tensor.

Substituting (4.5.2) into (4.5.4) and then (4.5.4) into (4.5.1), we can derive the Navier-Cauchy equations of linear elasticity

$$\rho\frac{\partial^2\vec{u}}{\partial t^2} = (\lambda + \mu)\nabla(\nabla \cdot \vec{u}) + \mu\nabla^2\vec{u} + \vec{f}. \tag{4.5.5}$$

When solving steady-state problems of elastostatics, we assume the left hand side of (4.5.5) to be equal to zero. This gives an elliptic problem, where two types of boundary conditions are most often used. The first are Dirichlet, geometric or *essential* boundary conditions,

$$\vec{u}(\boldsymbol{x}) = \vec{u}_0(\boldsymbol{x}), \tag{4.5.6}$$

where the displacement $\vec{u}$ is specified along a portion of the boundary. Setting $\vec{u}_0 = 0$ means that a certain part of the body is kept fixed.

The second kind are the traction, force or *natural* boundary conditions,

$$\sigma(\boldsymbol{x})\vec{n}(\boldsymbol{x}) = \vec{t}_0(\boldsymbol{x}), \tag{4.5.7}$$

where the surface traction is specified along a portion of the boundary with normal $\vec{n}(\boldsymbol{x})$. Setting $\vec{t}_0 = 0$ means that the movement on that surface is unrestricted.

Another option for boundary conditions are the boundary conditions on $\vec{u}$ or ∇u which can sometimes be derived from geometrical symmetry of the body and the forces acting on it.

Simplification to two dimensions

A three-dimensional problem from linear elasticity can sometimes be reduced to two dimensions using one of the two most common methods: plane stress or plane strain. In the plane strain formulation we assume that the strain ε has no nonzero components in one of the cardinal directions, and similarly in the plane stress formulation we assume that the stress tensor σ has no nonzero components in one of the cardinal directions. In both cases, the original equation can be reduced to only two variables, with further

details found in e.g. [Sla12, pp. 260–278] In plane strain case, the values of the Lamé parameters remain unchanged, but in plane stress formulation a reduced first Lamé parameter $\hat{\lambda}$ needs to be used instead of λ in (4.5.4) and (4.5.5), defined as

$$\hat{\lambda} = \frac{2\mu\lambda}{\lambda + 2\mu}. \tag{4.5.8}$$

We will use $\vec{u} = (u, v)$ and $\sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{bmatrix}$ when referring to the individual components.

Other quantities of interest

Often, instead of the Lamé parameters λ and μ, the elastic properties are expressed in terms of other elastic moduli, such as the Young's modulus E and the Poisson ratio ν. Conversions between various moduli are given in e.g. [Sla12, p. 215], and we will use the relations

$$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu}, \qquad \nu = \frac{\lambda}{2(\lambda + \mu)}, \tag{4.5.9}$$

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)} \tag{4.5.10}$$

to define the problem either in terms of λ and μ and later use E and ν, or vice versa.

Another quantity of interest is the von Mises stress [Sad14, ch. 3.5], which is related to yielding [Mis13], and is defined as

$$\sigma_v = \sqrt{\frac{2}{3}} \|\sigma^{\text{dev}}\|_F, \quad \sigma^{\text{dev}} = \sigma - \frac{1}{3}\operatorname{tr}(\sigma) I, \tag{4.5.11}$$

where $\|\cdot\|_F$ is the Frobenius norm and I is the identity tensor.

4.5.2 Disk under stress

Point contact is a first contact example, and a case of disk under under stress from two opposing point forces is used to test adaptive approaches [ANA11]. The case considers a disk of radius R compressed by two diametrical point forces of magnitude P, as illustrated in Fig. 4.11.

Assuming plane stress conditions, the closed form solution for stresses is given by e.g. Sadd [Sad14, p. 197] as:

$$\sigma_{xx} = -\frac{2P}{\pi}\left(\frac{x^2(R-y)}{r_1^4} + \frac{x^2(R+y)}{r_2^4} - \frac{1}{2R}\right) \tag{4.5.12}$$

$$\sigma_{yy} = -\frac{2P}{\pi}\left(\frac{(R-y)^3}{r_1^4} + \frac{(R+y)^3}{r_2^4} - \frac{1}{2R}\right) \tag{4.5.13}$$

$$\sigma_{xy} = \frac{2P}{\pi}\left(\frac{x(R-y)^2}{r_1^4} - \frac{x(R+y)^2}{r_2^4}\right), \tag{4.5.14}$$

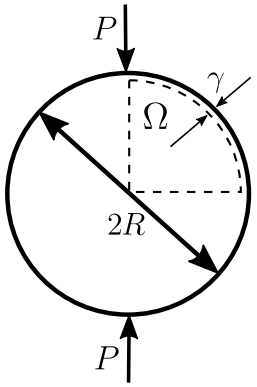


Figure 4.11: Disk under diametrical compression.

where $r_{1,2} = \sqrt{x^2 + (R \mp y)^2}$ are the distances to the points of contact. The stress distribution is singular under the contact points.

Numerically, we will consider only a quarter of the disk due to symmetry and we will move away from the boundary by γ to avoid the stress singularity. Formally, the problem is defined as follows:

$$(\lambda + \mu)\nabla(\nabla \cdot \vec{u}) + \mu\nabla^2\vec{u} = 0 \quad \text{in } \Omega, \tag{4.5.15}$$

where

$$\Omega = \{(x, y) \in \mathbb{R}^2;\ x \geq 0, y \geq 0, x^2 + y^2 \leq (R - \gamma)^2\} \tag{4.5.16}$$

is a quarter of a circle with radius $R - \gamma$, shown in Figure 4.11. The curved part of the boundary is part of the original circle and traction boundary conditions are imposed there, as defined by equations (4.5.12–4.5.14). Symmetry boundary conditions $v = 0$, $\frac{\partial u}{\partial y} = 0$ and $u = 0$, $\frac{\partial v}{\partial x} = 0$ are imposed on the bottom and left boundary, respectively.

The stress concentration is located around the point $(0, R - \gamma)$ and the parameter γ determines the peak stress magnitude, which is of order $\frac{P}{2\pi\gamma}$. This in turn controls the difficulty of the problem. The problem was solved for two different values of γ. The first was $\gamma = 0.01$ which represents a case of medium difficulty, where uniform refinement still obtains a solution, but has very erratic error behavior. The second case is $\gamma = 0.001$ where uniform refinement completely fails. Values $P = 1$, $R = 0.5$ and material parameters $E = 1$ and $\nu = 0.33$ were used for the physical and geometric parameters.

RBF-FD method with $\phi(r) = r^3$ and augmentation of order 2 with stencils of 13 closest nodes was used. Three different error measures were used to measure the difference between the closed form stress field σ and the numerically computed σ_h:

$$e_1 = \|\sigma_h - \sigma\|_{X,1} / \|\sigma\|_{X,1}, \quad \|y\|_{X,1} = \sum_{\boldsymbol{x}_i \in X} \sum_{1 \leq j \leq k \leq d} |\sigma_{x_j x_k}(\boldsymbol{x}_i)|, \tag{4.5.17}$$

$$e_\infty = \|\sigma_h - \sigma\|_{X,\infty} / \|\sigma\|_{X,\infty}, \quad \|y\|_{X,\infty} = \max_{\boldsymbol{x}_i \in X} \max_{1 \leq j \leq k \leq d} |\sigma_{x_j x_k}(\boldsymbol{x}_i)|, \tag{4.5.18}$$

$$e_E = \|\sigma_h - \sigma\|_{X,E} / \|\sigma\|_{X,E}, \quad \|y\|_{X,E} = \sqrt{\sum_{\boldsymbol{x}_i \in X} y(\boldsymbol{x}_i) : C^{-1} : y(\boldsymbol{x}_i)}. \tag{4.5.19}$$

The final expression computes the energy error, and C is the stiffness tensor from the Hooke's law (4.5.3). For a given solution u, the expression $\sigma(u(\boldsymbol{x}_i)) : C^{-1} : \sigma(u(\boldsymbol{x}_i))$ can also be written as $\sigma(u(\boldsymbol{x}_i)) : \varepsilon(u(\boldsymbol{x}_i))$, where colon : denotes the double contraction, $a : b = \sum_{i,j} a_{ij} b_{ij}$.

Adaptive refinement was used with $\varepsilon_\mathrm{r} = 0.05$, $\alpha_\mathrm{r} = 3$, $\varepsilon_\mathrm{d} = 0$, $h_\mathrm{r} = 0$, $h_\mathrm{d} = h^{(0)}$. The initial distribution was $h^{(0)} = 0.02$ in $\gamma = 0.01$ case with error tolerance $\tau = 1$. For the $\gamma = 0.001$ a denser initial distribution was required and we used $h^{(0)} = 0.005$, with a higher global tolerance $\tau = 10$. The gradient-based error indicator given by (4.2.2) was adapted for a vector-valued u by adding the indicator values for each component of $\vec{u}$. This adaptation was made for all the following cases as well.

The errors under uniform adaptive refinement are shown in Figure 4.12. Uniform refinement is not sufficient in any of the cases. In the $\gamma = 0.01$ case, the error does not follow a smooth downwards pattern, and is very sensitive and jagged, but still with a general downwards trend. In the $\gamma = 0.001$ case, uniform refinement is unable to produce a solution of desirable quality.

Adaptive solutions behave better. The initial distributions is still uniform and must be dense enough to put a few nodes in the high stress area near the contact. Initially, the error declines sharply, as the nodes are positioned near the contact area to accommodate the high gradients present there. After that, the error decreases at a slower rate, but still regularly, compared to the uniform case.

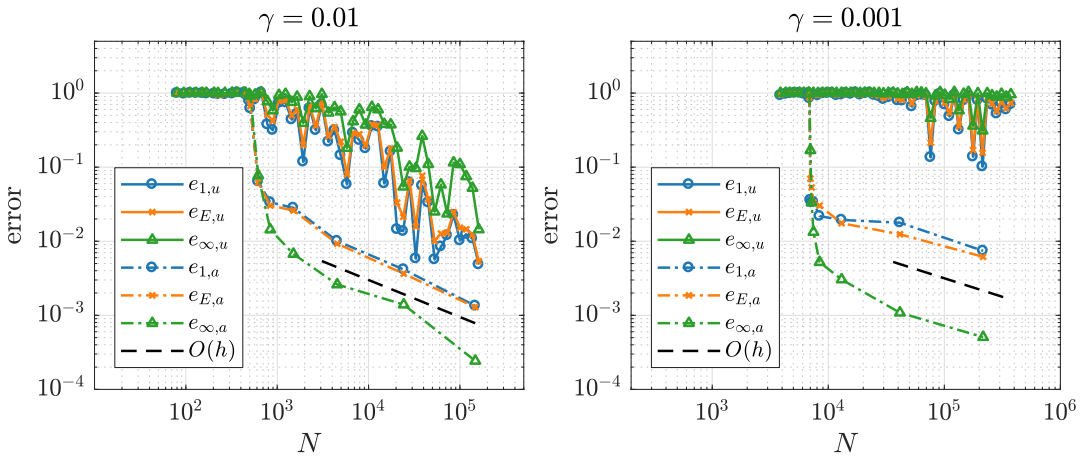


Figure 4.12: Errors under uniform and adaptive refinement while solving the compressed disk problem (4.5.15).

Figure 4.13 shows the stress components σ_{xy} on the anti-diagonal from $(R - \gamma, 0)$ to $(0, R - \gamma)$. The analytical solution is shown in black in the background. The top row of plots shows the solutions on a normal scale, to get a sense of how sharp the stress peaks are and how bad the initial approximations are. The bottom row of plots show the approximations of the stress peak and represent a very zoomed-in version of the top plots. It can be seen how subsequent iterations improve and refine the solution, getting closer to the analytical values.

The number of nodes and errors during the adaptive iterations are shown in Figure 4.14. It is interesting to note, that initially, the number of nodes does not increase much, with quite a significant decrease in error, as nodes are only added in the high

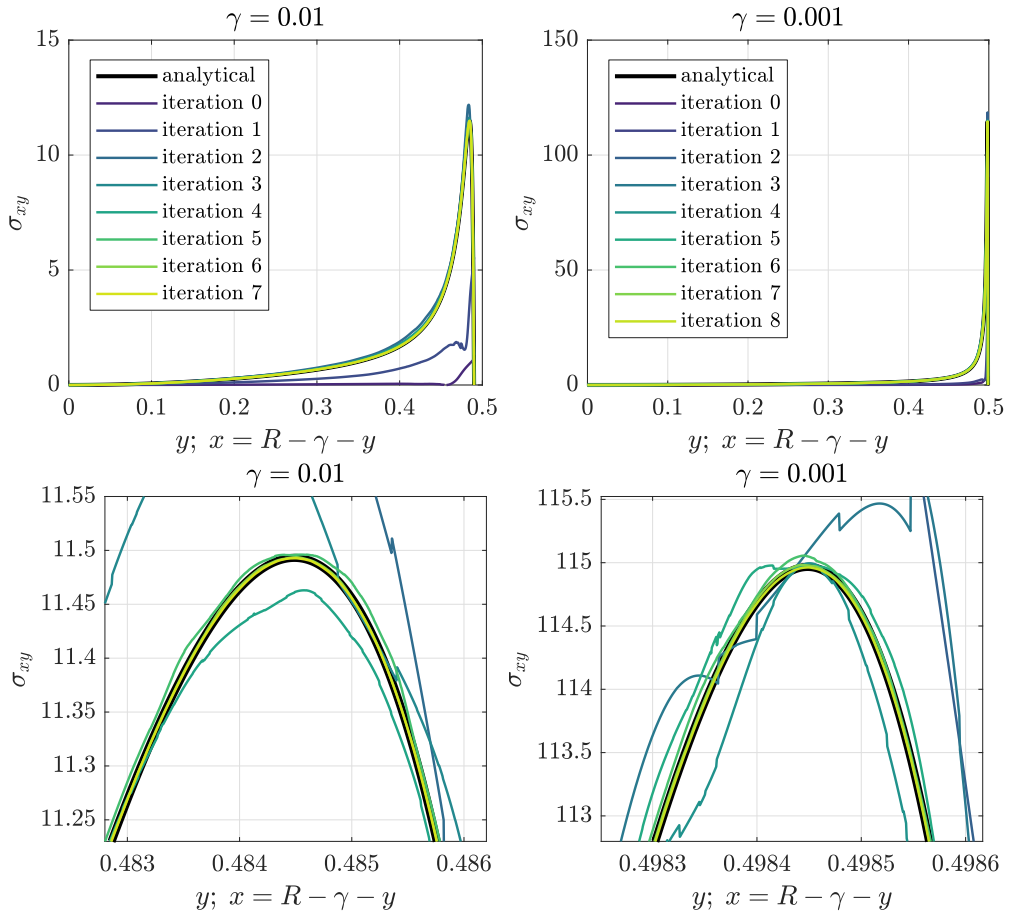


Figure 4.13: Computed stress profiles during adaptive solving of the compressed disk problem (4.5.15).

stress areas. This is the main mechanism responsible for the sharp drop in error shown in Figure 4.12 when error is plotted against the number of nodes. After the nodes are distributed so that the error indicator is distributed more uniformly, the refinement is more uniform as well, leading to a larger increase in the number of nodes, which in turn slows down the convergence curve in Figure 4.12. This effect is more pronounced in the $\gamma = 0.001$ case, which has sharper peaks.

Figure 4.15 shows the error and node density plots in first (0-th) and last (J-th) iteration of the adaptive procedure in the $\gamma = 0.01$ case. The top row shows the value of the energy error kernel $e_{E,i} = \sigma(\boldsymbol{x}_i) : \varepsilon(\boldsymbol{x}_i)$ at note $\boldsymbol{x}_i$ in first and last iterations. The error distribution in the first iteration is uneven and it becomes much more uniform in the last iterations. Note that the scales on the color axes are different on the plots to accommodate the decrease in error, but show a similar span of orders of magnitude.

The middle row shows the nodal density $\rho_i = -\log_{10}(d_i/\max_j(d_j))$, where d_i is the distance from $\boldsymbol{x}_i$ to its nearest neighbor. The nodal density values are shown on the actual nodal distributions to also show what the node distributions look like. Once again, note the different limits on the color axes. The initial distribution is uniform and very coarse with $N = 449$ nodes. The final distribution is quite strongly refined with a maximal density of approximately 3.14, meaning that the minimal closest distance is

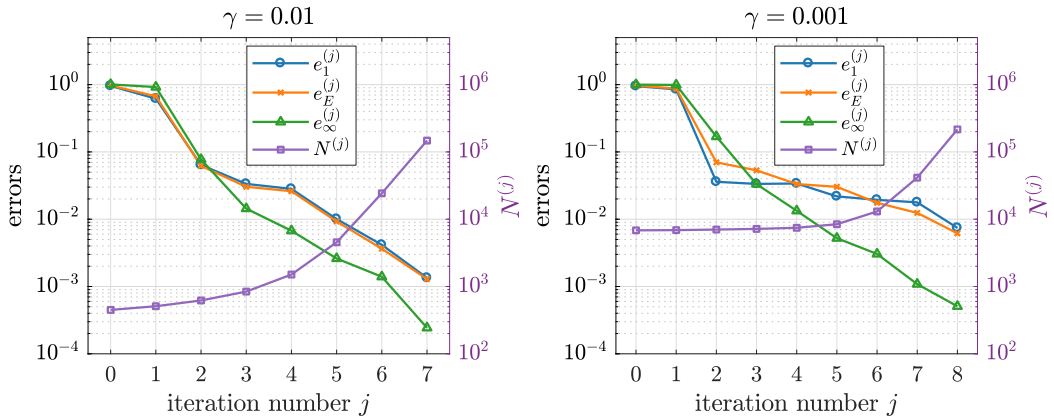


Figure 4.14: Errors and number of nodes during the adaptive iteration while solving the compressed disk problem (4.5.15).

about 1300 times smaller than the largest distance to the closest neighbor. The plots in the bottom row show 10 and 100 times zoomed versions of the final distribution (note the different x, y and color axes) to better visualize how the nodal distribution looks. The node markers are kept the same size in all four plots of density to further illustrate the nodal density.

4.5.3 3D point contact

A classical contact problem in three dimensions with a closed form solution is the Boussinesq's problem of a point force acting on a half-space. The problem is described by Slaughter [Sla12, p. 352] and illustrated in Figure 4.16. The closed form solution for a concentrated force of magnitude P pressing on a material with Poisson's ratio ν and second Lamé parameter μ is given in cylindrical coordinates r, θ and z as

$$\begin{aligned}
u_r &= \frac{Pr}{4\pi\mu}\left(\frac{z}{R^3} - \frac{1-2\nu}{R(z+R)}\right), \qquad u_\theta = 0, \qquad u_z = \frac{P}{4\pi\mu}\left(\frac{2(1-\nu)}{R} + \frac{z^2}{R^3}\right), \\
\sigma_{rr} &= \frac{P}{2\pi}\left(\frac{1-2\nu}{R(z+R)} - \frac{3r^2 z}{R^5}\right), \qquad \sigma_{\theta\theta} = \frac{P(1-2\nu)}{2\pi}\left(\frac{z}{R^3} - \frac{1}{R(z+R)}\right), \\
\sigma_{zz} &= -\frac{3Pz^3}{2\pi R^5}, \qquad \sigma_{rz} = -\frac{3Prz^2}{2\pi R^5}, \qquad \sigma_{r\theta} = 0, \qquad \sigma_{\theta z} = 0,
\end{aligned} \tag{4.5.20}$$

where $R = \sqrt{r^2 + z^2}$ is the distance of a point (x, y, z) from the origin. This solution has a singularity at the origin and similarly to how it was done in the compressed disk problem in Section 4.5.2, we will consider behavior some distance γ away from the singularity. Numerically, we will solve the Navier-Cauchy equations on $\Omega = [-1, -\gamma]^3$ with Dirichlet boundary conditions given by (4.5.20).

The 3D point contact problem is solved numerically for $\gamma = 10^{-3}$ in Cartesian coordinates using RBF-FD with $\phi(r) = r^3$ and augmentation of order 2 on stencils of 30 closest nodes. The domain was initially filled with spacing $h^{(0)} = 0.05$ and the adaptivity parameters were $\varepsilon_\mathrm{r} = 100, \alpha_\mathrm{r} = 3, h_\mathrm{r} = 0, \varepsilon_\mathrm{d} = 10, \alpha_\mathrm{d} = 2, h_\mathrm{d} = 0.05$. Values $P = -1$, $E = 1$ and $\nu = 0.33$ were used for the physical parameters.

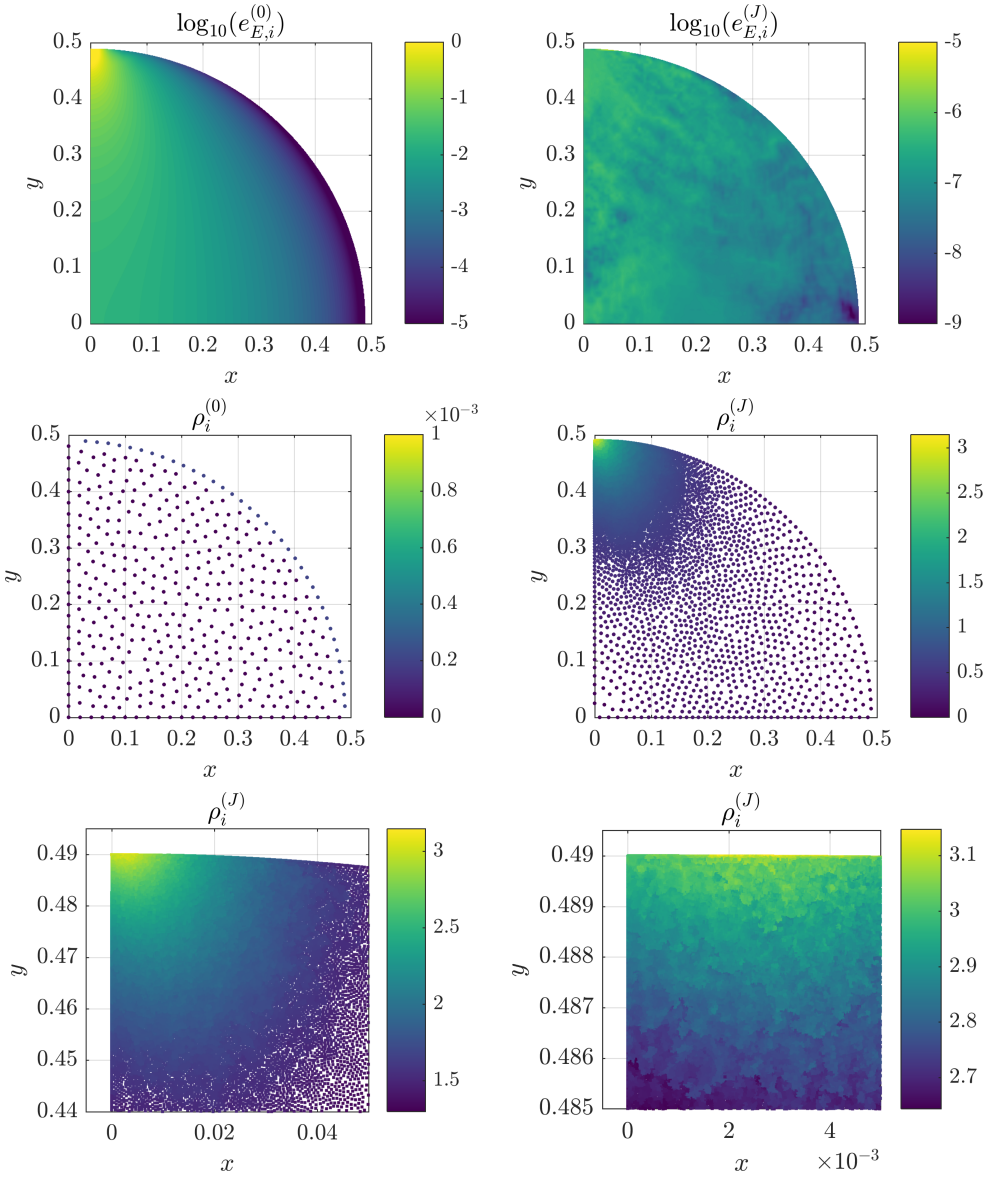


Figure 4.15: Error field and nodal distributions in the initial and final iteration while solving the compressed disk problem (4.5.15) with $\gamma = 0.01$. The top two plots show the initial and final energy error distribution, the middle row shows plots of initial and final nodal distribution, colored by nodal density, and the bottom row shows 10 and 100 times zoom of the final nodal distribution near the top corner.

The errors e_1, e_∞ and e_E were computed as in the compressed disk case using (4.5.17–4.5.19). Since displacements are also known in closed form, we also computed e_1 and e_∞ errors for displacements. To achieve a high enough density of sample nodes, the errors are only measured in the area $[-3\gamma, \gamma]^3$. These errors are shown in Figure 4.17.

Initially, the discretization is not dense enough to recognize the large stresses in the contact area and the error rises after the contact is first populated with a decent number of nodes. After that, the behavior is very similar as seen in Figure 4.12 in the compressed disk case: the error first sharply declines and then slowly decreases further. This is further illustrated in Figure 4.18 which shows the von Mises stress profiles along

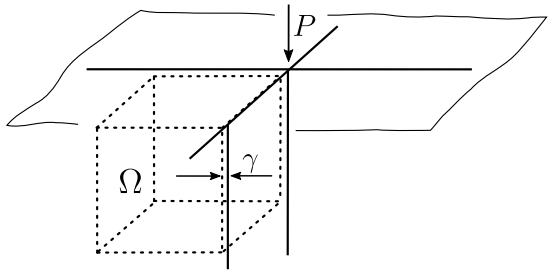


Figure 4.16: Illustration of the 3D point contact problem and the numerical domain Ω.

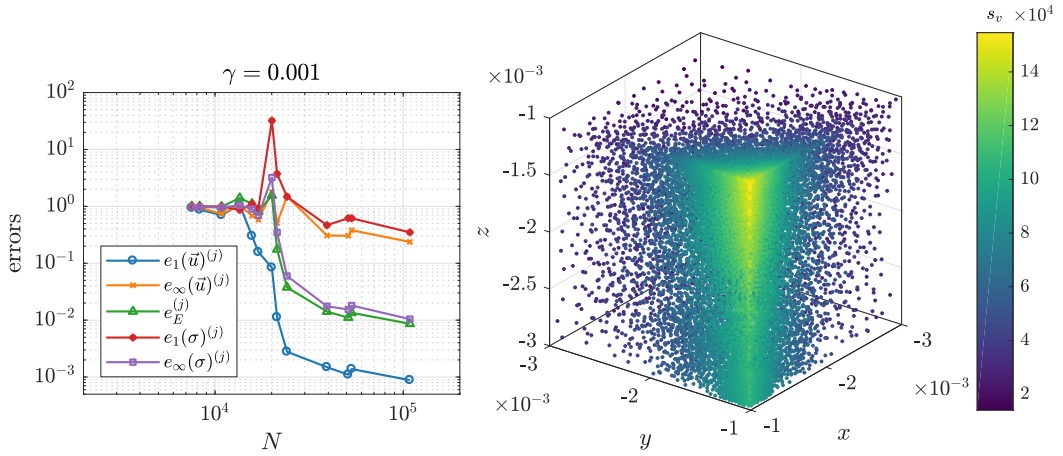


Figure 4.17: Errors during solving the 3D point contact problem (left) and the solution around the corner $(-\gamma, -\gamma, -\gamma)$ in the final iteration colored by von Mises stress (right).

the body diagonal of Ω from $(-2.5\gamma, -2.5\gamma, -2.5\gamma)$ to $(-\gamma, -\gamma, -\gamma)$.

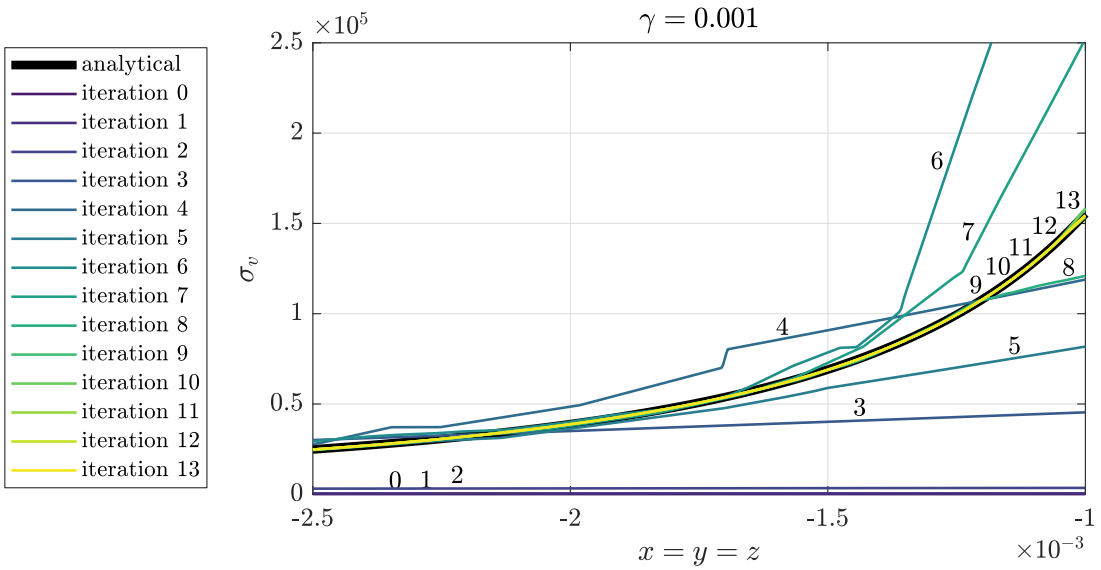


Figure 4.18: Von Mises stress along the body diagonal during the adaptive iteration while solving the 3D point contact problem.

The non-enlarged stress profiles are even sharper than the ones in top right plot

of Figure 4.13 and only the enlarged versions are shown. In the first three iterations the profiles are effectively flat. Then, the solutions starts to behave correctly, but with large deviations from the correct profile, which corresponds to the rise and subsequent sharp drop in error in Figure 4.17. The final 5 iterations which are overlapping in this picture correspond to the final 5 iterations, where error slowly improves and further enlargement of Figure 4.13 would show the trend consistent with the reported errors.

Values of $e_1(\sigma)$ errors, detailed node counts and maximal nodal densities are reported in Table 4.3, using the same notation as in Table 4.2.

Table 4.3: Errors and detailed node counts during the adaptive iteration for the solution of the 3D point contact problem.

j	$e_1^{(j)}(\sigma)$	$N^{(j)}$	$N_r^{(j)}$	$N_{r,\text{lim}}^{(j)}$	$N_s^{(j)}$	$N_d^{(j)}$	$N_{d,\text{lim}}^{(j)}$	$\max_i \rho_i^{(j)}$
0	1.0001	7524	219	0	398	0	6907	0.04
1	0.9994	8283	880	0	654	0	6749	0.47
2	0.9819	10829	3044	0	913	54	6818	0.93
3	1.3985	13521	5620	0	1057	24	6820	1.24
4	1.1114	15735	9111	0	3128	0	3496	1.63
5	0.8105	17012	8205	0	2000	16	6791	1.72
6	1.5566	20052	11151	0	2128	25	6748	2.19
7	0.1754	21379	12165	0	2393	41	6780	2.64
8	0.0377	24136	14788	0	2621	39	6688	3.32
9	0.0142	39349	29572	0	2798	89	6890	3.28
10	0.0110	51094	41384	0	2838	77	6795	3.29
11	0.0134	53256	43299	0	3996	61	5900	3.27
12	0.0086	108476	98388	0	3609	121	6358	3.57

The solution near corner $(-\gamma, -\gamma, -\gamma)$, colored by von Mises stress, is shown in Figure 4.17 on the obtained nodal distribution.

4.5.4 Hertzian contact

Hertzian contact theory is a classical theory of non-adhesive contact which began with Hertz in 1882 [Her82]. In some cases, the contact forces can be computed analytically, such as in the case of contact between two infinitely long cylinders, shown in Figure 4.19.

The elastic contact and the analytical expressions for the pressure distribution are described in [WD00, p. 122]. We will assume that the cylinders have radii R_1 and R_2, Young's moduli E_1 and E_2 and Poisson's ratios ν_1 and ν_2. The contact is predicted to be of width $2a$, with

$$a = 2\sqrt{\frac{FR^*}{\pi E^*}}, \tag{4.5.21}$$

where F is force per unit length, R^* is the combined radius, and E^* the combined Young's module, which are given by

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}, \quad \frac{1}{E^*} = \frac{1-\nu_1^2}{E_1} + \frac{1-\nu_2^2}{E_2}. \tag{4.5.22}$$

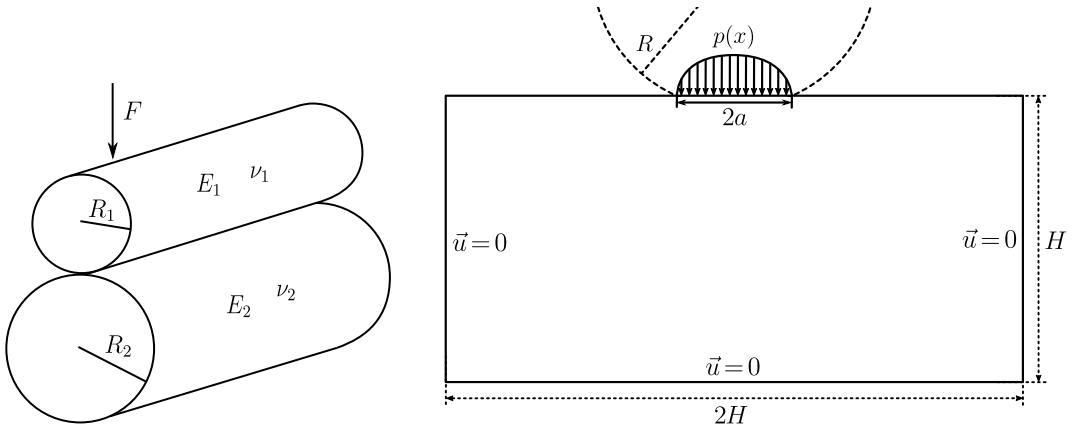


Figure 4.19: Hertzian contact between two cylinders (left) and the numerical setup for computing the contact between a cylinder and a plane adaptively.

The normal traction in the area of contact is given as

$$p(x) = p_0\sqrt{1 - \frac{x^2}{a^2}}, \quad p_0 = \sqrt{\frac{FE^*}{R^*\pi}} \tag{4.5.23}$$

and there is no tangential traction.

We will solve the limit case when R_1 tends to infinity, resulting in a contact between a cylinder and a half-space, which can be reduced to 2D using plane strain conditions [WD00]. The prescribed traction conditions hold in the contact area, with zero traction conditions outside the contact area on the top boundary. The remaining boundary conditions are the zero-displacement boundary conditions at infinity.

Closed form solutions for stresses in this (and more general) case have been constructed using the method of complex potentials by McEwen [McE49]. Stresses in the half-plane at a point (x, y) are expressed with expressions m and n,

$$\sigma_{xx} = -\frac{p_0}{b}\left[m\left(1 + \frac{y^2+n^2}{m^2+n^2}\right) + 2y\right], \tag{4.5.24}$$

$$\sigma_{yy} = -\frac{p_0}{b}m\left(1 - \frac{y^2+n^2}{m^2+n^2}\right), \tag{4.5.25}$$

$$\sigma_{xy} = \sigma_{yx} = \frac{p_0}{b}n\left(\frac{m^2-y^2}{m^2+n^2}\right), \tag{4.5.26}$$

where

$$m^2 = \frac{1}{2}\left(\sqrt{(b^2 - x^2 + y^2)^2 + 4x^2y^2} + b^2 - x^2 + y^2\right), \tag{4.5.27}$$

$$n^2 = \frac{1}{2}\left(\sqrt{(b^2 - x^2 + y^2)^2 + 4x^2y^2} - (b^2 - x^2 + y^2)\right), \tag{4.5.28}$$

and $m = \sqrt{m^2}$ and $n = \operatorname{sgn}(x)\sqrt{n^2}$. One can verify (using a computer algebra system) that $\nabla \cdot \sigma = 0$ holds on the lower half plane and that the traction conditions on the top boundary are satisfied.

To solve this problem numerically, the infinite half-plane is substituted for a finite computational domain $\Omega = (-H, H) \times (-H, 0)$ on the basis of Saint-Venant's principle, as shown in Figure 4.19. The specific problem we are solving is

$$\begin{aligned} (\lambda + \mu)\nabla(\nabla \cdot \vec{u}) + \mu\nabla^2\vec{u} &= 0 \ \text{ in } \Omega, \\ \vec{u}(-H, y) = \vec{u}(H, y) = \vec{u}(x, -H) &= 0, \\ \sigma(x, 0)\vec{n} = \vec{n} &\begin{cases} p(x), & |x| \leq a \\ 0, & |x| > a \end{cases}. \end{aligned} \tag{4.5.29}$$

We will choose $R_1 = 1\,\mathrm{m}$, $F = 500\,\mathrm{N/m}$, $E_1 = E_2 = 72.1\,\mathrm{GPa}$ and $\nu_1 = \nu_2 = 0.33$ for the physical parameters. This means that the contact width a is approximately 0.13 mm. We choose the domain size H to be $H = 0.25\,\mathrm{m}$, which is approximately 1923 times larger than the contact area. Based on the analysis done in [Sla17; SK19b] this is (well in the range of being) large enough so that the truncation error does not present a significant contribution to the overall error.

We used RBF-FD with $\phi(r) = r^3$ and monomial augmentation of order 2 with stencils of $n = 19$ nearest neighbors. Larger stencils were needed for increased stability due to more aggressive refinement. The problem was solved adaptively with $\varepsilon_\mathrm{r} = 10^5$, $\alpha_\mathrm{r} = 5$, $\varepsilon_\mathrm{d} = 10^4$, $\alpha_\mathrm{d} = 2$, $h_\mathrm{r} = 0$, $h_\mathrm{d} = 50^{(0)}$. The initial distribution was $h^{(0)} = 0.001$ with error tolerance based on the maximal value of the error indicator equal to $\tau = 50\,000$.

The large ratio between the domain size and the contact area in combination with initial uniform discretization means that a large number of nodes is needed initially to cover the contact, even though the initial discretization is only $h^{(0)} \approx 7.7a$, i.e. one node per 7.7 contact widths. The error was measured the same way as in the compressed disk case using (4.5.17–4.5.19), computed in the area $[-10a, 10a] \times [-10a, 0]$. Figure 4.20 shows the errors and node counts during the adaptive iteration. Detailed node counts and errors are reported in Table 4.4, using the same notation as in Table 4.2.

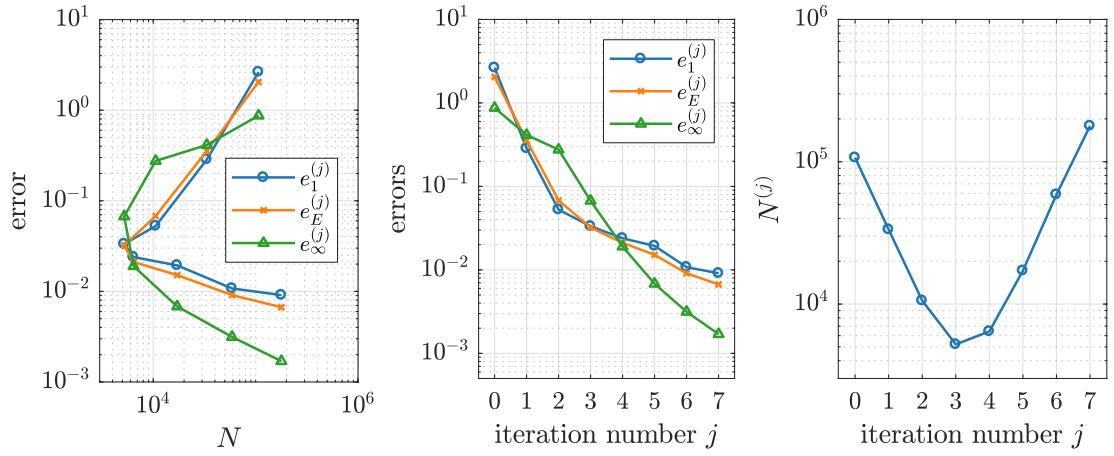


Figure 4.20: Errors and node counts during the adaptive iteration for the solution of the Hertzian contact problem (4.5.29).

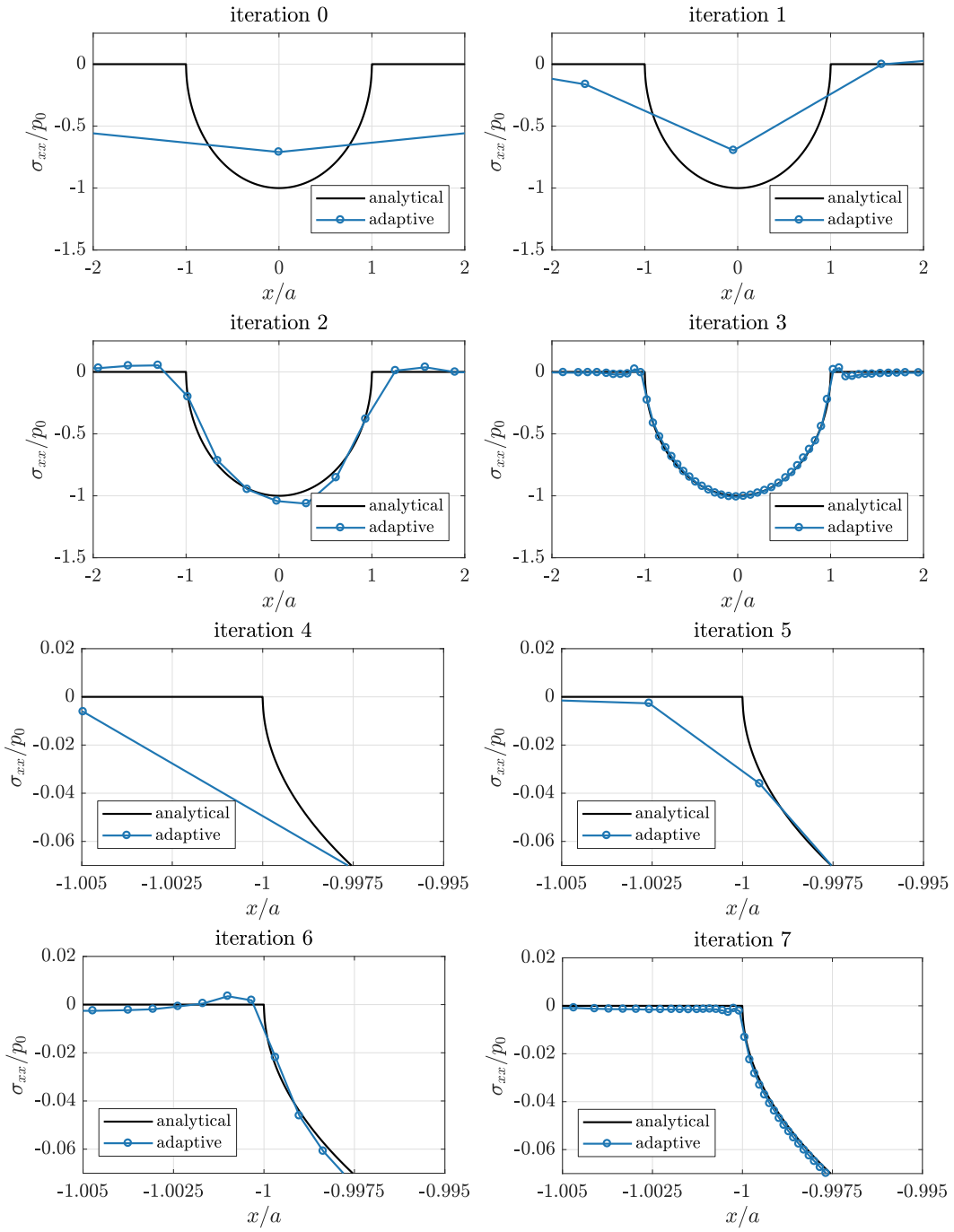


Figure 4.21: Normal stress profiles in contact area during adaptive iteration for the solution of Hertzian contact problem (4.5.29). Last 4 plots show enlargements of the boundary of the contact area.

Table 4.4: Errors and detailed node counts during the adaptive iteration for the solution of the Hertzian contact problem (4.5.29).

j	$e_\infty^{(j)}$	$N^{(j)}$	$N_r^{(j)}$	$N_{r,\mathrm{lim}}^{(j)}$	$N_s^{(j)}$	$N_d^{(j)}$	$N_{d,\mathrm{lim}}^{(j)}$	$\max_i \rho_i^{(j)}$
0	0.8720	106858	218	0	1837	104803	0	0.00
1	0.4132	33397	288	0	583	32526	0	1.00
2	0.2755	10571	731	0	576	9264	0	2.00
3	0.0672	5207	1543	0	658	3006	0	3.00
4	0.0190	6424	4477	0	603	1344	0	3.99
5	0.0068	17150	15785	0	715	613	37	4.96
6	0.0031	58790	57571	0	847	314	58	5.81
7	0.0017	178475	174047	0	4100	272	56	6.48

Initially, derefinement is substantial in most of the domain, but refinement is present in the contact area, which causes the number of nodes to decrease while also increasing the accuracy. This trend continues and less and less nodes are derefined in each iteration, while refinement around the contact area is increased. The same trend can be observed also in Figure 4.21 where profiles of σ_{xx} on the top boundary are shown. The strength of the refinement and the solution accuracy under contact and on its edge can be observed in more detail. The density growth is limited by $j \log(\alpha_\mathrm{r}/\alpha_\mathrm{d})$ and is initially closely followed, but then deviates slightly from this upper bound as derefinement slows down.

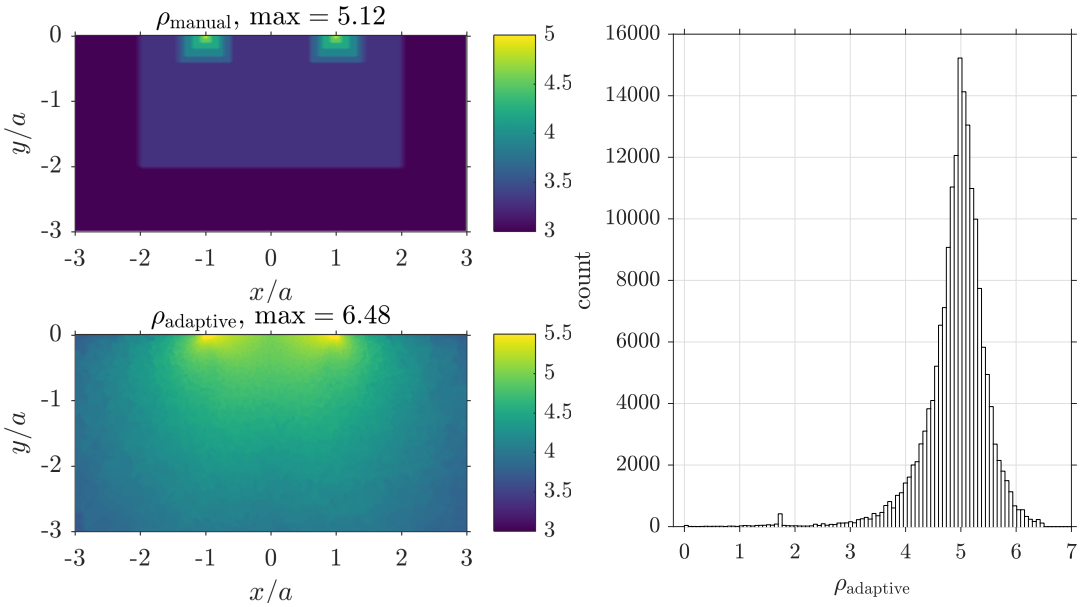


Figure 4.22: Comparison between the manual and adaptively obtained density (left) and the histogram of local densities in the adaptive case (right).

It is also interesting to observe the final nodal density. A histogram of densities ρ_i for all nodes i in the final iteration is shown in Figure 4.22. The maximal density is $\rho = 6.48$, which means that the ration between the smallest and the largest local nodal spacing is around 3 million. We can see that most nodes in fact have density between 3 and 6. However, these nodes cover a very small proportion of the whole domain. The nodes included with $0 \leq \rho_i < 1$ cover 97.12% of the whole domain, the nodes

with $1 \leq \rho < 2$ cover around 2.85% and all the other nodes are concentrated on an area representing around 0.03% of the domain. In fact, 95% percent of all nodes are located in $[-3a, 3a] \times [-3a, 0]$ rectangle, which represents 0.000027% of the domain area. It is also interesting to compare adaptively obtained density to the density of the manually refined nodal distribution used in [SK19b]. Both nodal distributions are dense in the contact area, specifically near the contact edges, but the adaptive density is much smoother.

4.5.5 Fretting fatigue contact

A contact problem from the study of fretting fatigue, described by Pereira et al. [Per+16] is chosen as the next example. The choice of this example is motivated by collaboration in the international project "Multi-analysis of fretting fatigue using physical and virtual experiments" (G018916N) between the University of Luxembourg, the University of Ghent and the Jožef Stefan Institute, where some results presented here were also used. The example is a part of a standard laboratory test in the study of fretting fatigue of materials [Hoj+14], where a small rectangular specimen of width W, length L and thickness t is compressed width-wise by two oscillating pads with radius of curvature R and axially stretched with tension σ_{ax}. The pads compress with force F and cause maximal tangential traction Q. The test is schematically shown in Figure 4.23.

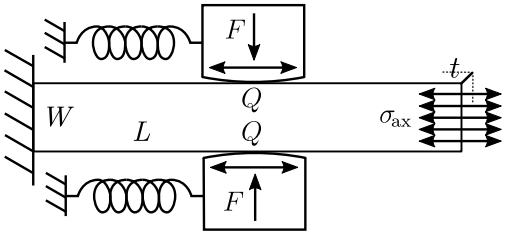


Figure 4.23: Schema of a fretting fatigue laboratory test.

The goal is to compute the stresses in the specimen. The traction induced by the pads will be approximated analytically, derived from an extension of Hertzian contact theory and used before in [Per+16; Kos+19]. The problem is reduced to two dimensions using plane strain. The contact width and the normal traction remain the same as in the Hertzian theory, except that the force per unit length F in (4.5.21) and (4.5.23) is computed as F/t, giving the contact half-width a as

$$a = 2\sqrt{\frac{FR}{t\pi E^*}} \tag{4.5.30}$$

and the normal traction as

$$p(x) = p_0\sqrt{1 - \frac{x^2}{a^2}}, \quad p_0 = \sqrt{\frac{FE^*}{t\pi R}}. \tag{4.5.31}$$

Tangential traction additionally depends on the coefficient of friction μ_f and divides the contact area into stick and slip zones. The half width of the stick zone is expressed as $c = a\sqrt{1 - \frac{Q}{\mu_f F}}$ (under the assumption that $Q \leq \mu_f F$). Furthermore, the axial traction

σ_{ax} causes that the stick zone is not in the center of the contact area, but is instead offset by eccentricity $e = \operatorname{sgn}(Q)\frac{a\sigma_{ax}}{4\mu_f p_0}$. This is only valid if $\sigma_{ax} \leq 4\left(1 - \sqrt{1 - \frac{Q}{\mu_f F}}\right)$, which will hold in our case.

The tangential traction in the contact area is then defined piece-wise for stick and slip zones separately:

$$q(x) = -\mu_f p_0 \begin{cases} \left(\sqrt{1-\frac{x^2}{a^2}} - \frac{c}{a}\sqrt{1-\frac{(x-e)^2}{c^2}}\right), & |x-e| < c \\ \sqrt{1-\frac{x^2}{a^2}}, & c \leq |x-e| \text{ and } |x| \leq a \end{cases}, \tag{4.5.32}$$

but it can be easily checked for continuity when $|x - e| = c$. Sample stick and slip zone division and the tractions p and q are shown in Figure 4.24.

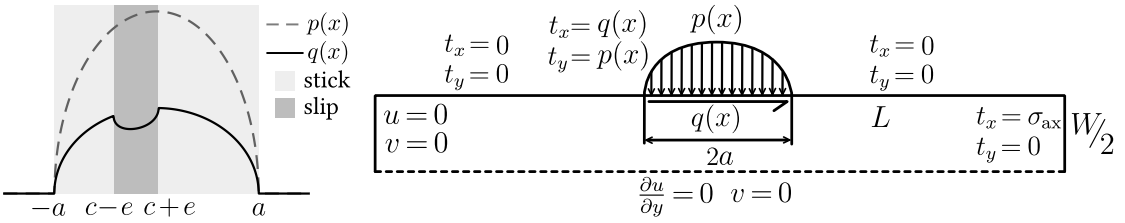


Figure 4.24: Computational domain with boundary conditions and surface tractions with stick and slip zones shown. Computational domain is not to scale.

The problem is further reduced by exploiting the symmetry along the horizontal axis. The computational domain is thus a rectangle $\Omega = [-L/2, L/2] \times [-W/2, 0]$, where the coordinate system has been positioned in such a way that $(0, 0)$ is the center point of the contact. The problem is formally given by

$$\begin{aligned} (\lambda+\mu)\nabla(\nabla\cdot\vec{u}) + \mu\nabla^2\vec{u} &= 0 \text{ in } \Omega, \\ \vec{u}(-L/2, y) &= \vec{0}, \\ \sigma(x,0)\vec{n} &= \begin{cases} (p(x), q(x)), & |x| \leq a \\ (0,0), & |x| > a \end{cases} \\ \sigma(L/2, y)\vec{n} &= \sigma_{ax}\vec{n} \\ v(x, -W/2) &= 0 \\ \frac{\partial u}{\partial y}(x, -W/2) &= 0, \end{aligned} \tag{4.5.33}$$

and the domain with boundary conditions is shown also in Figure 4.24 using notation $\vec{u} = (u_x, u_y), \sigma\vec{n} = \vec{t} = (t_x, t_y)$.

The stresses in the specimen will be computed in the state of oscillation maximum. The specimen dimensions are $L = 40\,\mathrm{mm}$, $W = 10\,\mathrm{mm}$ and $t = 4\,\mathrm{mm}$, and material parameters used are $E_1 = E_2 = 72.1\,\mathrm{GPa}$, $\nu_1 = \nu_2 = 0.33$, coinciding with aluminum 2420-T3. Values $F = 543\,\mathrm{N}$, $Q = 155\,\mathrm{N}$, $\sigma_{ax} = 100\,\mathrm{MPa}$, $R = 10\,\mathrm{mm}$ and $\mu_f = 0.3$ were chosen for the model parameters. The half-contact width a in this case equals $0.2067\,\mathrm{mm}$, which is approximately 200 times smaller than the domain length L, and adaptivity is needed.

Similarly to Hertzian contact case, we use RBF-FD with $\phi(r) = r^3$ and augmentation of second order on stencils of 19 closest nodes. The initial spacing was selected as $h^{(0)} = 0.5\,\mathrm{mm}$ and adaptivity parameters were $h_\mathrm{r} = 0$, $h_\mathrm{d} = 1\,\mathrm{mm}$, $\alpha_\mathrm{r} = 10$, $\alpha_\mathrm{d} = 1.5$, $\varepsilon_\mathrm{r} = 10^5$, $\varepsilon_\mathrm{d} = 10^4$. The adaptive procedure was run for 4 iterations. Table 4.5 shows node counts during the adaptive iteration as well as maximal nodal density. We see that no derefinement occurs and thus density growth is limited by $j \log_{10}(\alpha_\mathrm{r})$, which is closely followed, i.e. the refinement is the maximal possible.

Table 4.5: Detailed node counts during the adaptive iteration for the solution of the fretting fatigue contact problem (4.5.33).

j	$N^{(j)}$	$N_r^{(j)}$	$N_{r,\mathrm{lim}}^{(j)}$	$N_s^{(j)}$	$N_d^{(j)}$	$N_{d,\mathrm{lim}}^{(j)}$	$\max_i \rho_i^{(j)}$
0	772	716	0	56	0	0	0.00
1	2423	2340	0	83	0	0	1.00
2	5856	5778	0	78	0	0	2.00
3	31271	31193	0	78	0	0	2.95

The adaptive solution is compared to a solution obtained with the Finite Element Method (FEM) using the freely available FreeFem++ [Hec12] and its built-in adaptivity via `adaptmesh`, as well as to another FEM solution, on denser manually refined mesh with more than 10^5 DOFs, using commercial ABAQUS® software for finite element analysis [Per+16].

The obtained surface traction σ_{xx} is shown in Figure 4.25. This traction is of particular interest, because the location and value of the σ_{xx} peak can be used to determine the location of crack initiation [Per+16]. A paper by Kosec et al. [Kos+19] specifically compares strong and weak form methods when computing such peaks. The adaptive solution initially only has one node in the contact area, but it gets aggressively refined.

The error of surface σ_{xx} is measured against both FEM solutions as

$$e_1 = \|\hat{\sigma}_{xx} - \sigma_{xx}\|_1 / \|\sigma_{xx}\|_1, \quad \|\sigma_{xx}\|_1 = \sum_{x \in G} |\sigma_{xx}(x)|, \tag{4.5.34}$$

where G is a set of 10^5 independent equidistant points on $[-2a, 2a]$. The error is shown in Figure 4.26, as well as the error between the reference solutions themselves. We can see the adaptive error decrease with every iteration and achieving satisfactory accuracy in the end, agreeing with ABAQUS more than FreeFem++.

The total computational time of the adaptivity procedure was around 2.5 s and varies from 2 s to 10 s depending on the choice of adaptivity parameters.

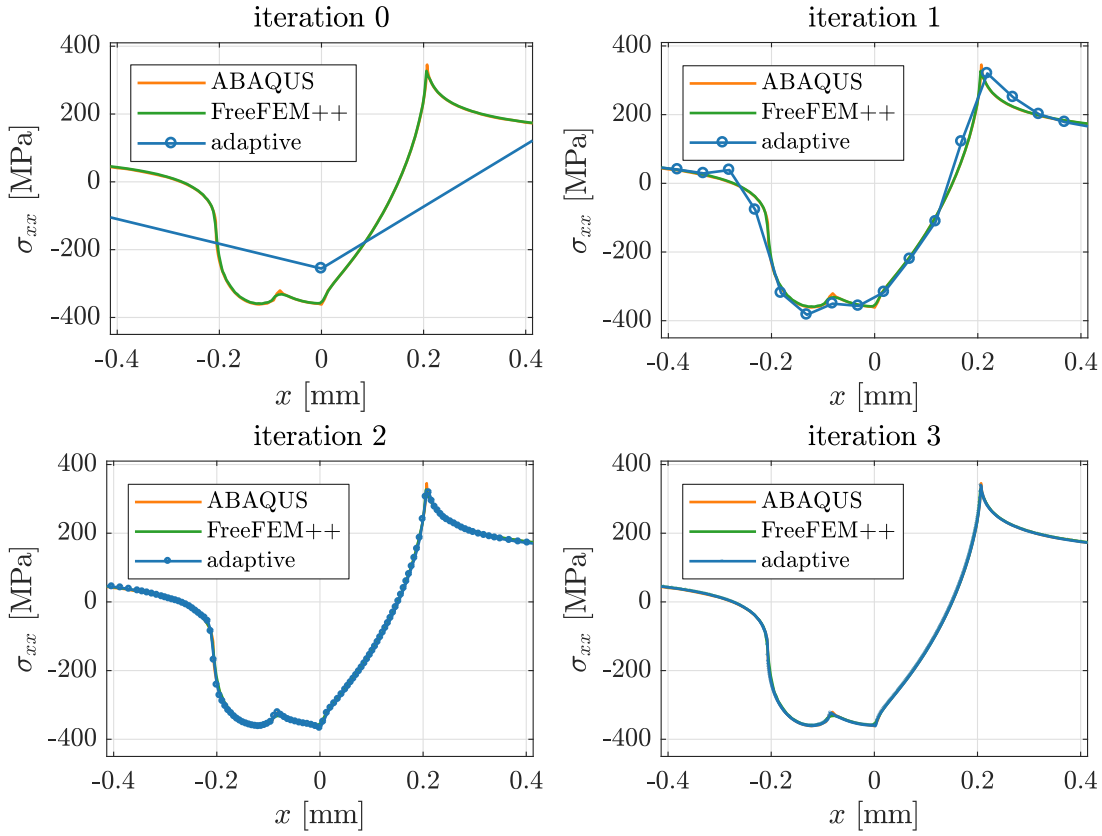


Figure 4.25: Surface traction σ_{xx} obtained during the adaptive iteration compared with two other FEM solutions.

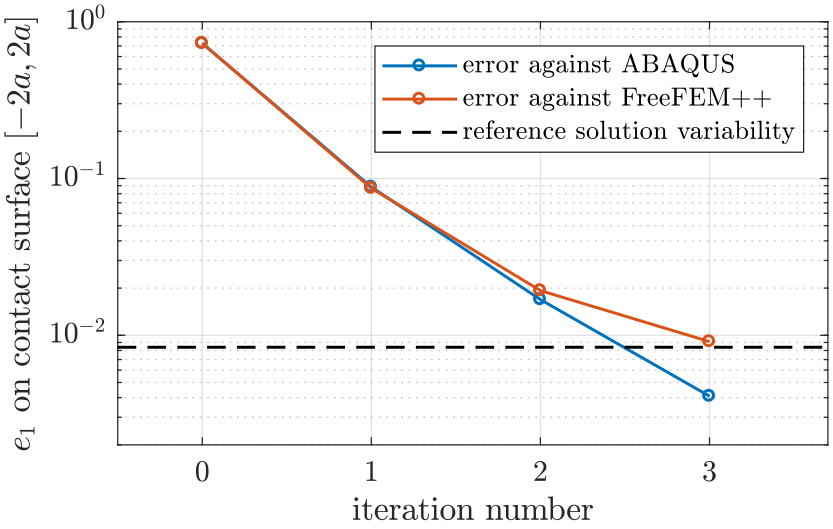


Figure 4.26: Error of the surface traction σ_{xx} obtained adaptively compared to both reference solutions.

Chapter 5

Implementation

Implementations of numerical methods for solving PDEs are often long, complex and poorly readable. The solution procedure is usually intertwined with the discretization technique, which makes the code hard to debug and very error prone and unmaintainable. This issue is pronounced enough that is has been studied on its own, along with the identification of design patterns that help avoid it [HMÅ02; RAX10; Ara+14]. Another common issue that PDE solvers suffer from is that the code for solving 2D problems and 3D problems is often completely separate. The idea to use proper (dimension independent) abstractions and think in terms of vector fields and operators instead of only arrays and indices was called "Coordinate free numerics" [MH96] and the main idea is illustrated in Figure 5.1.

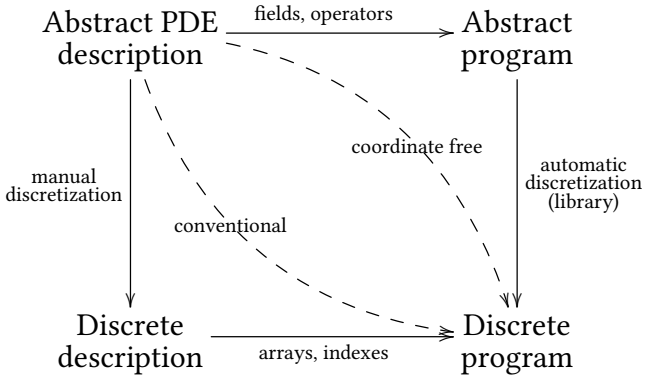


Figure 5.1: The idea of coordinate free numerics.

While this naming never got wider recognition, the same idea is still relevant. The importance that well-designed software projects have on increasingly complex physical simulations was outlined also in the "Physics today" article by Post and Votta titled "Computational science demands a new paradigm" [PV05] as well as in "Nature" by Meralli in the paper titled "Computational science: ...Error: why scientific programming does not compute" [Mer10].

The usefulness of proper abstraction can be seen in the fact that many such libraries were developed for FEM, including deal.II [BHK07], DOLFIN (part of the FEniCS Project [Aln+15]) [LW10] and FreeFem++ [Hec12]. Sadly, in the field of strong form meshless methods there are no open source general purpose libraries available. Many

papers come with their own set of scripts, such as MFDMtool [Mil13], GEC_RBFFD [Bay+15], MFree2D [Liu02], and even the review paper [Ngu+08]. For specific established methods, such as SPH, there are software packages available, e.g. DualSPHyiscs [Cre+15]. A notable newer implementation is the MESHFREE software [KMM17], which implements the Finite Pointset, but it focuses on applications and not on the development of methods themselves and is not open source. Another package to note is the RBF python package [Hin15] (although not present in the standard Python Package Index), which implements basic RBF interpolation and RBF-based PDE solution techniques.

DOLFIN, deal.II and DualSPHyiscs use the C++ programming language, and FreeFem++ implements its own extended language on top of a C++-based core. The language is popular due to a combination of its speed and the abstractions it offers, mainly the template system, which allows for writing efficient generic code. As no general purpose open-source libraries for strong-form meshless methods were available, we decided to implement our own, to support our research in the field. The development began in 2015 and the library, called Medusa, is available at `http://e6.ijs.si/medusa`. The library has been a good investment and is one of the reasons we can attempt more challenging tasks, such as automatic adaptivity.

The main design goals of the library were to have reusable and reliable building blocks that can be used to quickly test many different meshless methods, without tedious, error-prone and unmaintainable repetition of implementation. The general workflow of solving a problem was divided into: preparing the domain discretization, defining the discretization, computing the stencil weights, and finally, solving the PDE. We wanted the design to be modular, so that underlying algorithms can easily be swapped, and extensible, so that new can be added when needed. Together with the desire for reasonable efficiency and readability, C++ was the language of choice. Additionally, it offers a good way to achieve dimension independence through non-type template parameters. The library has no dependencies, apart from the C++ standard library and the HDF5 C library [Fol+11] for reading and writing binary HDF5 files. As stated in the introduction, Medusa includes the `Eigen`, `nanoflann`, `tinyformat` and RapidXML libraries. The source code of the library is available on Gitlab[1]. Reliability is ensured using continuous integration running over 300 tests and the documentation of the library, generated with Doxygen, is also available[2].

To achieve dimension independence, nearly all core classes have a template parameter `vec_t`, which contains the spatial dimension and the floating point type. This means that the same implementation is used when solving a 2D problem with single precision or a 3D problem with complex numbers in double precision. The remainder of this chapter presents the more interesting parts of the library. We start with a minimal working dimension independent example in Section 5.1, followed by a description of the main modules in Section 5.2. Finally, some execution time benchmarks are presented in Section 5.3.

[1] `https://gitlab.com/e62Lab/medusa`
[2] `http://e6.ijs.si/medusa/docs`

5.1 Minimal working example

We will borrow an example from the Medusa library test suite to show a minimal working example. Consider a steady state advection-diffusion problem

$$8(-\mathbf{1} \cdot \nabla u) + 2\nabla^2 u = -1 \ \text{ in } \ \Omega \tag{5.1.1}$$

$$u = 0 \ \text{ on } \ \partial\Omega, \tag{5.1.2}$$

where $\Omega = B(\mathbf{0}, 1) \setminus B(\mathbf{0.1}, 0.3) \subseteq \mathbb{R}^d$ is a ball with a hole. The implementation of the numerical solution of this problem is shown as listing 5.1. The dimension of the problem was never specified, neither in the mathematical formulation nor in the implementation, but was instead kept an arbitrary d or as a template parameter.

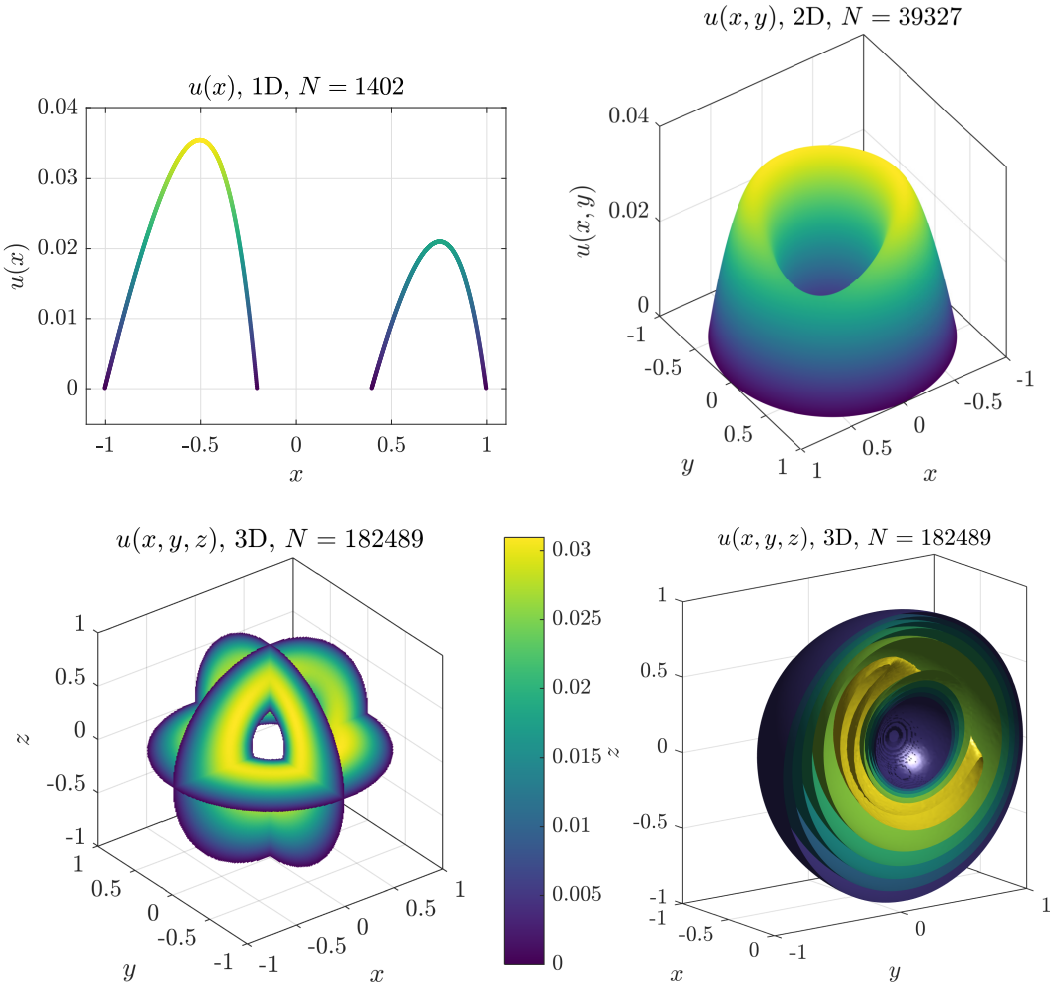


Figure 5.2: Solutions of a steady state advection-diffusion equation in 1D, 2D (top row) and 3D (bottom row), showing slices along the coordinate planes and isosurfaces on a half of the domain.

The steps taken in the code follow the general procedure described above. First, we construct the domain shape, which is then discretized to produce the domain discretization, stored in the variable `domain`.

```cpp
template <int dim>
void solve() {
    typedef Vec<double, dim> vec;  // Define the point type.
    BallShape<vec> c(0.0, 1.0), hole(0.1, 0.3); // Define the domain shape.

    // Discretize the domain. The nodal spacing is increased in higher dimensions
    // to keep the number of nodes small enough.
    double dx = 0.001*dim*dim*dim;
    DomainDiscretization<vec> domain = (c-hole).discretizeBoundaryWithStep(dx);
    GeneralFill<vec> fill; fill(domain, dx);
    int N = domain.size(); prn(N);
    Monomials<vec> mon(2);  // Monomial basis.
    domain.findSupport(FindClosest(2*mon.size()));  // Stencil size = 2*(number of mon.)

    // Construct the approximation.
    RBFFD<Polyharmonic<double, 3>, vec, ScaleToClosest> approx({}, mon);
    auto storage = domain.template computeShapes<sh::lap|sh::d1>(approx);

    // Define matrices and construct the operators interface.
    Eigen::SparseMatrix<double, Eigen::RowMajor> M(N, N);
    Eigen::VectorXd rhs(N); rhs.setZero();
    auto op = storage.implicitOperators(M, rhs);
    M.reserve(storage.supportSizes());

    // Write the discretized equations into M and rhs.
    for (int i : domain.interior()) {
        8.0*op.grad(i, -1) + 2.0*op.lap(i) = -1.0;
    }
    for (int i : domain.boundary()) {
        op.value(i) = 0.0;
    }

    // Solve the system.
    Eigen::BiCGSTAB<decltype(M), Eigen::IncompleteLUT<double>> solver;
    solver.preconditioner().setDroptol(1e-4);
    solver.preconditioner().setFillfactor(20);
    solver.compute(M);
    ScalarFieldd u = solver.solve(rhs);   // The solution as a scalar field of doubles.

    // Save the solution.
    std::ofstream out_file(format("example_%dd_data.m", dim));
    out_file << "positions = " << domain.positions() << ";" << std::endl;
    out_file << "solution = " << u << ";" << std::endl;
    out_file.close();
```

Listing 5.1: Solving the advection-diffusion equation.

As we will be using RBF-FD with polyharmonic RBFs and monomial augmentation, we prepare the monomial basis. The stencil size was twice the number of monomials in the basis and closest neighbor stencils were used. RBF-FD approximation is declared with $\phi(r) = r^3$ and monomials of second order. The stencil weights are computed for 1st order derivatives and for the Laplacian operator.

Then the sparse matrix and the right hand side are constructed with appropriate reserved space, and the implicit operators interface. The discretized equations are written in the matrix and the right hand side. It is worthwhile to compare the lines 25–30 to equations (5.1.1) and (5.1.2) and observe the expressiveness. The sparse system is then solved and the solution is saved. The solutions produced by calling `solve<`d`>()` as defined in listing 5.1, for $d = 1, 2, 3$ are shown in Figure 5.2. The domain in 1D case is in fact disconnected, i.e. $\Omega = (-1, -0.2) \cup (0.4, 1)$, which is not a problem.

5.2 Library modules

We give a brief overview of the most important library modules. The full list is available in the documentation[3] and includes modules for I/O, integrators for time stepping, types for vectors, scalar fields and vector fields, and other utilities.

5.2.1 Domains

The implementation of domain discretization closely follows the Definition 3.1.1. We store nodal positions, types and stencil indices for all nodes, as well as unit normals for boundary nodes. This discrete description of the domain is all that is needed for further computation, as such it can be pre-computed and stored (e.g. in a file) and then loaded without knowing the actual analytical shape.

The library also includes tools to generate domain discretizations. A rudimentary system of basic analytical shapes, such as balls, boxes and polygons is supported as well as affine transformations of the shapes and Boolean operations between them. The surface and interior fill algorithms described in Section 3.2 and 3.3 are implemented, as well as grid-based fill algorithms and the algorithm described in [SF19]. There are also different methods for stencil searching, currently only stencils of closest nodes and balanced stencils are supported.

This design enables modularity, as the user can choose from any supported shape, node generation algorithm or stencil selection algorithm, as well as extensibility, as it is simple to define custom classes that can be plugged in indistinguishably from the existing ones.

5.2.2 Approximations

Computing stencil weights is one of the core functionalities of the library. This is done by an *approximation engine*, which represents a technique for operator approximation as described in Chapter 2. The operator approximation is performed in two steps: first, the

[3] http://e6.ijs.si/medusa/docs/html/modules.html

engine is set up at a point $\boldsymbol{c}$ with its stencil nodes. This involves computing and decomposing the approximation matrix, which is a relatively costly operation, taking cubic time. Then, stencil weights are computed for each of the operators, taking quadratic time for each operator. The procedure is described as Algorithm 5.1. The two-step computation allows us to only do the costly setup once. Furthermore, the for loop on line 3 of Algorithm 5.1 has independent iterations, and can thus be trivially parallelized.

Algorithm 5.1 Computation of stencil weights.

Input: A domain discretization $\mathcal{D}$.
Input: An approximation engine $\mathcal{A}$.
Input: A list of nodes $\mathcal{I}$.
Input: A list of operators $\mathcal{O}$.
Output: Stencil weights as defined by $\mathcal{A}$ for all operators in $\mathcal{O}$ for all node indices in $\mathcal{I}$.

1: **function** COMPUTE_WEIGHTS($\mathcal{D}$, $\mathcal{A}$, $\mathcal{I}$, $\mathcal{O}$)
2: $R \leftarrow$ STORAGE
3: **for each** i **in** $\mathcal{I}$ **do**
4: $\boldsymbol{c} \leftarrow$ GET_NODE($\mathcal{D}, i$) ▷ Get the i-th node.
5: $s \leftarrow$ GET_STENCIL_NODES($\mathcal{D}, i$) ▷ Get the stencil nodes of the i-th node.
6: STORE_INDICES($R, i, \mathcal{L}$, GET_STENCIL_INDICES($\mathcal{D}, i$))
7: $S \leftarrow$ SETUP($\mathcal{A}, \boldsymbol{c}, s$) ▷ Setup the approximation only once.
8: **for each** $\mathcal{L}$ **in** $\mathcal{O}$ **do**
9: $\boldsymbol{w} \leftarrow$ COMPUTE($S, \mathcal{L}$) ▷ Compute stencil weights.
10: STORE_WEIGHTS($R, i, \mathcal{L}, \boldsymbol{w}$) ▷ And store them.
11: **end for**
12: **end for**
13: **return** R
14: **end function**

The approximation engines included in Medusa include `RBFFD` and `WLS` methods. The `WLS` engine is constructed from a basis, a weight function, a scaling function and a solver. Similarly, the `RBFFD` engine is constructed from an RBF, possible monomials augmentation, a scaling function and a solver. The scaling function and the solver are useful for computational purposes as described in Section 2.2.6.

The infrastructure was once again designed to be modular and extensible, which was achieved with C++'s template engine. Users can define their own approximation engines, their own bases, weights, RBF's and operators, as long as they conform to the prescribed concepts, which define what properties such objects should implement and how they should behave[4]. For example, to define a custom operator, the user needs to specify how the operator is applied to any of the functions in the basis and evaluated at the origin, by implementing a method `applyAt0(basis, index, stencil, scale)`. The requirements will be able to become more explicit with the introduction of Concepts in C++20 and their wider use. The ability to freely swap whole approximation engines or their parts is one of the strongest features of the library that allows for quick testing of many different combinations.

[4] http://e6.ijs.si/medusa/docs/html/concepts.html

5.2.3 Operators

This module implements an interface layer on top of the computed stencil weights that enables the user to write discretized PDE equations without interfacing with the discretization too much. This is extremely helpful when solving more complex PDEs where the physical model or the general solution procedure is the main focus instead of the approximation methods itself. It is not obligatory to use this interface, as stencil weights can be accessed directly, but it is recommended in order to reduce the likelihood of errors and improve readability. This interface is called *operators* as it implements the common operators found in PDEs, such as gradients, Laplacian, divergence, curl, etc. both for explicit evaluation and for use in implicit solving.

The explicit evaluation closely follows the procedure described in Section 2.3.1. Sample code taken from the Medusa test suite using vector and scalar explicit operators solving the lid-driven cavity problem with artificial compressibility method [Kos18] is shown in listing 5.2.

```
auto op_v = storage.explicitVectorOperators();
auto op_s = storage.explicitOperators();
for (int time_step = 0; time_step < max_steps; ++time_step) {
    for (int i : interior) {
        u2[i] = u1[i] + dt * (-1 / rho * op_s.grad(p, i) + mu / rho * op_v.lap(u1, i) -
        ↪   op_v.grad(u1, i) * u1[i]);
    }
    for (int i : all) {
        p[i] = p[i] - dt * dl*dl*rho*op_v.div(u2, i);
    }
    u1.swap(u2);
}
```

Listing 5.2: Explicitly solving the lid-driven cavity problem with artificial compressibility method in Medusa using the explicit operators interface.

The code implements a solution of the Navier-Stokes equations

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\, \vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v} + \vec{f}, \tag{5.2.1}$$

$$\nabla \cdot \vec{v} = 0, \tag{5.2.2}$$

and it only focuses on the implementation of the solution procedure, while hiding most of the (currently irrelevant) discretization. The method for divergence implements a sum

$$(\nabla \cdot \vec{v})(\boldsymbol{x}_k) = \sum_{i=1}^{d} \frac{\partial v_i}{\partial x_i} \approx \sum_{i=1}^{d} \sum_{j \in I(\boldsymbol{x}_k)} (\boldsymbol{w}^{\frac{\partial}{\partial x_i}, S(\boldsymbol{x}_k)}(\boldsymbol{x}_k))_j v_i(\boldsymbol{x}_j), \tag{5.2.3}$$

and others are similar. This is not difficult to implement, but is really error prone and being able to think and look at the operators as operators instead of summation loops is very helpful.

The implicit equations follow a similar philosophy, as seen in the minimal working example. Like the explicit version, implicit operators store a reference to the stencil

weights, but additionally also store a reference to a prepared sparse matrix and a vector representing the right hand side. When called, the combination of weights approximating the desired operator is written in the appropriate place in the matrix and the right hand side, as described in 2.3.2. This has already been used in the minimal working example. Another example is the cantilever beam from linear elasticity, once again taken from the Medusa test suite. The problem is governed by the Navier-Cauchy equation

$$(\lambda + \mu)\nabla(\nabla \cdot \vec{u}) + \mu\nabla^2\vec{u} = 0 \tag{5.2.4}$$

with the boundary conditions

$$\vec{u}|_{\text{right}} = \left(\frac{Py(3D^2(1+\nu) - 4(2+\nu)y^2)}{24EI}, -\frac{L\nu Py^2}{2EI}\right), \tag{5.2.5}$$

$$\sigma|_{\text{top}} = \sigma|_{\text{bottom}} = 0, \tag{5.2.6}$$

$$\sigma|_{\text{left}} = \left(0, \frac{P}{2I}\left(D^2/4 - y^2\right)\right). \tag{5.2.7}$$

The Medusa implementation is shown in listing 5.3.

```
Eigen::SparseMatrix<double, Eigen::RowMajor> M(2*N, 2*N);
Eigen::VectorXd rhs(2*N); rhs.setZero();
M.reserve(shapes.supportSizesVec());

auto op = shapes.implicitVectorOperators(M, rhs);

for (int i : domain.interior()) {
    (lam+mu)*op.graddiv(i) + mu*op.lap(i) = 0.0;
}
for (int i : domain.types() == RIGHT) {
    double y = domain.pos(i, 1);
    op.value(i) = {(P*y*(3*D*D*(1+nu) - 4*(2+nu)*y*y)) / (24.*E*I), -(L*nu*P*y*y) /
    ↪  (2.*E*I)};
}
for (int i : domain.types() == LEFT) {
    double y = domain.pos(i, 1);
    op.traction(i, lam, mu, {-1, 0}) = {0, -P*(D*D - 4*y*y) / (8.*I)};
}
for (int i : domain.types() == TOP) {
    op.traction(i, lam, mu, {0, 1}) = 0.0;
}
for (int i : domain.types() == BOTTOM) {
    op.traction(i, lam, mu, {0, -1}) = 0.0;
}
```

Listing 5.3: Implicitly solving the cantilever beam problem in Medusa using the implicit operators interface.

The implementation of the implicit interface is not so straightforward and it uses a limited version of expression templates [Vel95], supporting the syntax $\sum_{j=1}^{n} \alpha_j L_j = r$ with $n \leq 1$ and some α_j potentially omitted. The call to an operator returns a lazy object

that is written to the matrix only when multiplied with a scalar, summed with another object or assigned a right hand side. Moreover, the addition and assignment serve as a safeguard that checks that all terms refer to the same row. It also forces the user to write the right hand side, so that it is not forgotten. The interface is again optional, parts of the matrix can be filled manually, and the matrix is usable in the usual ways, it has been filled with or without using the implicit operators. Custom explicit and implicit operators can be (for now) implemented as a standalone functions that either wrap other operators or directly apply the stencil weights.

5.3 Benchmarks

The design of the library is mainly focused on modularity and usability, but we still aim for reasonable efficiency of the final solution procedure. The cost of abstractions in performance critical sections was compared with a "bare-bones" implementation in [SK18c], to ensure that their cost is negligible. To get some sense of how Medusa compares to other libraries for solving PDEs, we give a comparison with the FreeFEM++ library.

The aim of this experiment is to give a comparable answer to "How long does it take to solve this problem if I use Medusa?" The following already familiar boundary value problem will be solved:

$$-\nabla u = f \text{ in } \Omega, \qquad u = u_0 \text{ on } \partial\Omega, \tag{5.3.1}$$

for $u_0(x) = \prod_{i=1}^{d} \sin(\pi x_i)$ and $f = -\nabla^2 u_0$ on $\Omega = B(0,1) \setminus B(0,1/2)$ in 2D and 3D.

Note that both libraries implement different methods for solving PDEs and we will be using them in their canonical setup. We used RBF-FD with $\phi(r) = r^3$ with stencils of $n = 9$ and $n = 35$ closest nodes in 2D and 3D, respectively. The problem was solved with quasi-uniform discretization with spacing h produces by the algorithms presented in this work. The FreeFem++ implementation solved the corresponding weak formulation with P1 elements. In fact, the sample domain Ω, the code for its discretization and the code for the solution of a Poisson problem was taken directly from the FreeFem++ website.

Both FreeFem++ with its dependencies and Medusa were compiled from source and run in a single-threaded context, repeating each measurement 9 times. The results are shown in Figure 5.3

The accuracy and convergence rates of both methods are comparable, with the expected convergence rate of $O(h^2) = O(N^{-2/d})$. RBF-FD had slightly worse accuracy and slightly better execution time. The times of the various parts of the solution procedure showed that the execution time difference is almost entirely due to the cost of meshing vs. node placing. Many other factors also influence the execution time, such as the stencil size and the choice of the linear solver. In the presented measurement we used the Conjugate Gradient solver in FreeFem++, and BiCGStab with ILUT$(5, 10^{-2})$ preconditioner for Medusa, as they worked best of the built-in solvers. The solvers took approximately the same amount of time to solve the final system in both cases.

Parts of the Medusa implementation used for just presented measurements were also timed separately. We will take a closer look at the 3D timings of specific parts. The solution procedure was split into 6 parts:

- *domain*, which includes the creation of the domain shape and the node positioning,

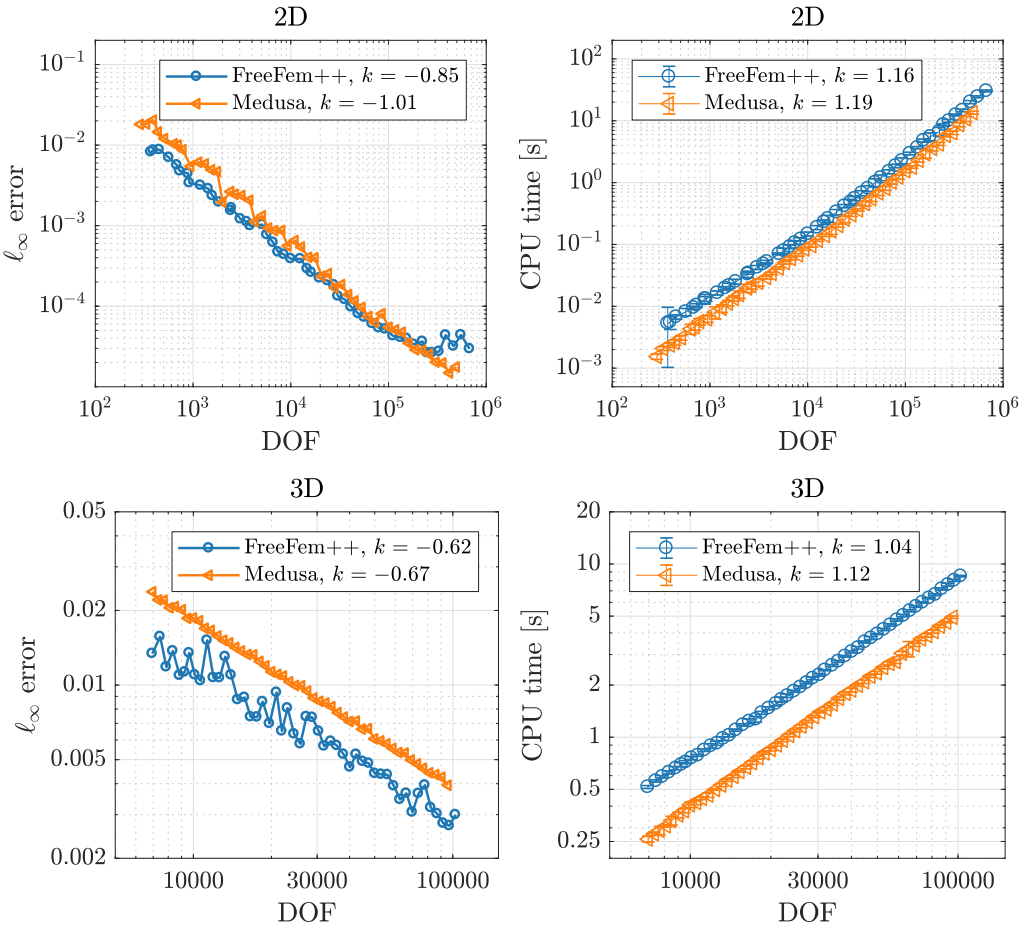


Figure 5.3: Execution time and accuracy of FreeFem++ and Medusa solving the problem (5.3.1). Each time measurement was repeated 9 times with the median value shown as the plot point and the error bars representing the standard deviation. Values of k in the legend represent the slope of the best fit line.

- *stencils*, which includes finding stencils for all nodes,
- *weights*, which includes the definition of the approximation and the computation of stencil weights,
- *assemble*, which includes the sparse matrix and right hand side construction,
- *decompose*, which includes the incomplete LU decomposition used for the preconditioner,
- *solve*, which includes the run time of the iterative solver.

The absolute timings are shown in Figure 5.4, along with the plots showing how execution time is distributed among the parts for different values of N.

In this case, we can see that roughly 30% of the time is spent on constructing the domain discretization (node positioning and stencils), 40% on computing stencil weights and 30% on solving the system. However, at larger N, the solver percentage decreases in favor of domain discretization. Another important thing is that execution time ratio can

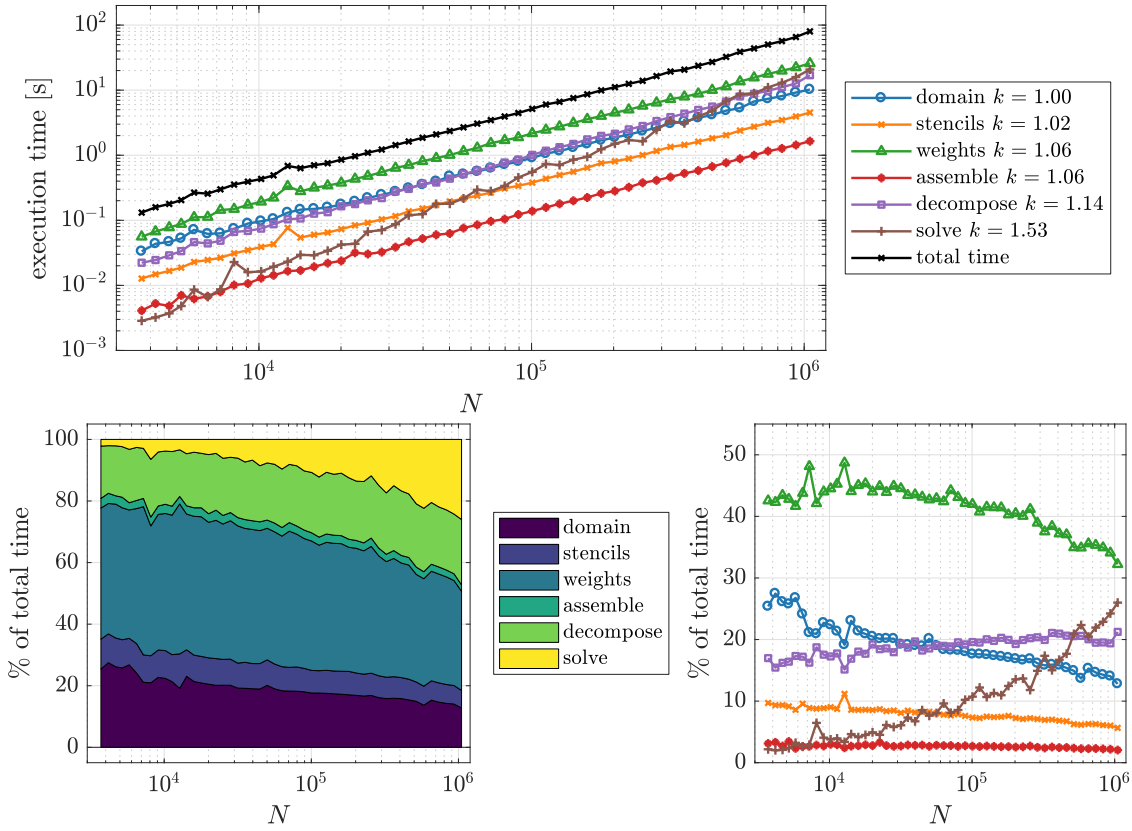


Figure 5.4: Execution times of different stages of computation when solving (5.3.1) with Medusa. Values k represent the slopes of the best fit lines.

vary significantly with different approximation types, stencils sizes, linear solvers, and dimensions. If high-order methods are used, around 80% of the time can easily be spent computing the approximations, but for stencils of 5 nodes in 2D with N around 10^5, the domain discretization can take up to 50%. Furthermore, in some linear elasticity cases, with Gaussian RBF-FD, the linear solver took around 95% of the total time. Separate time measurements are useful to determine which parts are most worthy of optimization and parallelization, and the conclusion is that each (except matrix assembly) is the most important in some cases. Some parallelization efforts are described in [KS19b].

Conclusions and future work

The main goal of the thesis was to develop a fully automatic meshless solution procedure using RBF-FD. To achieve that, a node placing algorithm was developed first, which supports placing nodes in domain interiors and on domain boundaries, when given as parametric surfaces. The algorithm is dimension independent and has provable minimal spacing requirement. It can produce locally regular nodal distributions with variable density in arbitrary-shaped domains and takes $O(N \log N)$ time in general to produce N nodes. The node placing algorithm was combined with the RBF-FD approximation using polyharmonic RBFs $\phi(r) = r^3$ augmented with monomials to achieve stable approximations with local monomial reproduction and to avoid dealing with RBF shape parameters. The approximations were further combined with a gradient-based error indicator and an h-refinement scheme to form an automatic h-adaptive procedure. All parts of the procedure are meshless and dimension independent. The automatic adaptivity was demonstrated on classical 2D and 3D Poisson problems as well as 2D and 3D problems from elastostatics. Both derefinement and refinement were successfully demonstrated, achieving the refinement ratio of $3 \cdot 10^6$ between the coarsest and finest parts of the discretization.

Additionally, reusable implementations of the main building block for strong-form meshless methods were developed along with the adaptive procedure, and published as the Medusa library, an open-source library for solving PDEs with strong-form methods. The library is written in C++ and allows for easy testing of various meshless methods that are also simply generalizable to 3D or higher dimensions.

Many directions are open for future research. Better and more robust error indicators for meshless methods have to be developed, to increase usability in practical cases. For interpolation based methods, such as RBF-FD, the ZZ-type indicators seem most likely. In tandem with that, better refinement strategies have to be developed. Ideally, the nodal distributions should support adding N_{new} nodes and removing N_{old} nodes in $O(N_{\text{new}} + N_{\text{old}})$ time, without changing the rest of the domain. Midpoint strategies and their effect on the quality of the nodal distribution should be further investigated to find possible h-refinement strategies without regeneration.

Further improvements to the node generation algorithm include generalizations to CAD models and analyzing the behavior on multi-patched domains and surfaces. Efforts to parallelize the node placing algorithm should also be made, as node generation can represent a significant portion of the total execution time.

Further research into approximation types is also needed, notably to compare the WLS-based methods to RBF-FD. This includes the behavior on non-uniform node distributions, the effect of stencil size and stability with respect to nodal positioning. The

effects of one-sided stencils that often cause trouble with Neumann boundary conditions should be well understood. Approximations based on k nearest neighbor stencils should be compared with approximation of radius based stencils, where the radius is scaled spatially according to the spacing function h. Judging by the MLS error analysis, such stencil selection could, along with h-quasi-uniformity, lead to better theoretical foundations for approximation errors. Additionally, higher-order monomials augmentation and its relationship with stencil selection should be investigated. The answers to such questions help with the development of p-adaptivity, and can eventually lead to effective hp-adaptivity.

Bibliography

[ANA11] M. H. Afshar, M. Naisipour, and J. Amani, *Node moving adaptive refinement strategy for planar elasticity problems using discrete least squares meshless method*, Finite Elements in Analysis and Design **47**(12):1315–1325, Dec. 2011, DOI: 10.1016/j.finel.2011.07.003 (cited on pp. 93, 95, 109).

[Aln+15] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, *The FEniCS Project Version 1.5*, eng, Archive of Numerical Software **3**, 2015, DOI: 10.11588/ans.2015.100.20553 (cited on p. 125).

[Alt20] Y. Altman, *export_fig: A MATLAB toolbox for exporting publication quality figures*, v3.05, commit a83b4077, May 2020, URL: https://github.com/altmany/export_fig (cited on pp. 4, 160).

[Alu00] N. R. Aluru, *A point collocation method based on reproducing kernel approximations*, International Journal for Numerical Methods in Engineering **47**(6):1083–1121, 2000, DOI: 10.1002/(sici)1097-0207(20000228)47:6<1083::aid-nme816>3.0.co;2-n (cited on p. 36).

[APP09] A. Angulo, L. P. Pozo, and F. Perazzo, *A posteriori error estimator and an adaptive technique in meshless finite points method*, Engineering Analysis with Boundary Elements **33**(11):1322–1338, 2009, DOI: 10.1016/j.enganabound.2009.06.004 (cited on pp. 93, 95).

[Ara+14] S. Arabas, D. Jarecka, A. Jaruga, and M. Fijałkowski, *Formula translation in Blitz++, NumPy and modern Fortran: A case study of the language choice tradeoffs*, Scientific Programming **22**(3):201–222, 2014, DOI: 10.3233/SPR-140379 (cited on p. 125).

[AZ98] S. N. Atluri and T. Zhu, *A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics*, Computational Mechanics **22**(2):117–127, 1998, DOI: 10.1007/s004660050346 (cited on pp. 36, 55).

[BG92] I. Babuška and B. Q. Guo, *The h, p and hp version of the finite element method; basis theory and applications*, Advances in Engineering Software **15**(3-4):159–174, 1992, DOI: 10.1016/0965-9978(92)90097-y (cited on p. 92).

[BM97] I. Babuška and J. M. Melenk, *The partition of unity method*, International Journal for Numerical Methods in Engineering **40**(4):727–758, 1997, DOI: 10.1002/(sici)1097-0207(19970228)40:4<727::aid-nme86>3.0.co;2-n (cited on p. 36).

[BR78] I. Babuška and W. C. Rheinboldt, *A-posteriori error estimates for the finite element method*, International Journal for Numerical Methods in Engineering **12**(10):1597–1615, 1978, DOI: 10.1002/nme.1620121010 (cited on p. 95).

[BSD09] M. Balzer, T. Schlömer, and O. Deussen, *Capacity-constrained point distributions: a variant of Lloyd's method*, ACM Transactions on Graphics (TOG) **28**(3), July 2009, DOI: 10.1145/1531326.1531392 (cited on p. 58).

[BHK07] W. Bangerth, R. Hartmann, and G. Kanschat, *deal.II – a General Purpose Object Oriented Finite Element Library*, ACM Trans. Math. Softw. **33**(4):24/1–24/27, 2007, DOI: 10.1145/1268776.1268779 (cited on p. 125).

[Bar15] G. Barnett, *A robust RBF-FD formulation based on polyharmonic splines and polynomials*, PhD thesis, University of Colorado, 2015, URL: https://scholar.colorado.edu/concern/graduate_thesis_or_dissertations/8623hx72q (cited on p. 48).

[Bay+15] V. Bayona, N. Flyer, G. M. Lucas, and A. J. G. Baumgaertner, *A 3-D RBF-FD solver for modeling the atmospheric global electric circuit with topography (GEC-RBFFD v1. 0)*, Geoscientific Model Development **8**(10):3007, 2015, DOI: 10.5194/gmd-8-3007-2015, URL: https://bitbucket.org/vbayona/gec_rbffd/ (cited on p. 126).

[Bay19] V. Bayona, *Comparison of moving least squares and RBF + poly for interpolation and derivative approximation*, Journal of Scientific Computing **81**(1):486–512, 2019, DOI: 10.1007/s10915-019-01028-8 (cited on p. 37).

[BFF19] V. Bayona, N. Flyer, and B. Fornberg, *On the role of polynomials in RBF-FD approximations: III. Behavior near domain boundaries*, Journal of Computational Physics **380**:378–399, 2019, DOI: 10.1016/j.jcp.2018.12.013 (cited on p. 37).

[Bay+17] V. Bayona, N. Flyer, B. Fornberg, and G. A. Barnett, *On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs*, Journal of Computational Physics **332**:257–273, 2017, DOI: 10.1016/j.jcp.2016.12.008 (cited on pp. 37, 88, 92).

[Bay+10] V. Bayona, M. Moscoso, M. Carretero, and M. Kindelan, *RBF-FD formulas and convergence properties*, Journal of Computational Physics **229**(22):8281–8295, 2010, DOI: 10.1016/j.jcp.2010.07.008 (cited on pp. 37, 46).

[BMK11] V. Bayona, M. Moscoso, and M. Kindelan, *Optimal constant shape parameter for multiquadric based RBF-FD method*, Journal of Computational Physics **230**(19):7384–7399, 2011, DOI: 10.1016/j.jcp.2011.06.005 (cited on p. 37).

[BMK12] V. Bayona, M. Moscoso, and M. Kindelan, *Optimal variable shape parameter for multiquadric based RBF-FD method*, Journal of Computational Physics **231**(6):2466–2481, 2012, DOI: 10.1016/j.jcp.2011.11.036 (cited on pp. 27, 37).

[Bel+96] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl, *Meshless methods: an overview and recent developments*, Computer Methods in Applied Mechanics and Engineering **139**(1):3–47, 1996, DOI: 10.1016/s0045-7825(96)01078-x (cited on pp. 35, 36).

[BLG94] T. Belytschko, Y. Y. Lu, and L. Gu, *Element-free Galerkin methods*, International Journal for Numerical Methods in Engineering **37**(2):229–256, 1994, DOI: 10.1002/nme.1620370205 (cited on pp. 35, 89).

[Ben+03] J. J. Benito, F. Urena, L. Gavete, and R. Alvarez, *An h-adaptive method in the generalized finite differences*, Computer Methods in Applied Mechanics and Engineering **192**(5-6):735–759, 2003, DOI: 10.1016/s0045-7825(02)00594-7 (cited on pp. 93, 94).

[BKL06] A. Beygelzimer, S. Kakade, and J. Langford, *Cover trees for nearest neighbor*, in: Proceedings of the 23rd international conference on machine learning, ACM, 2006, pp. 97–104, DOI: 10.1145/1143844.1143857 (cited on p. 66).

[BR14] J. L. Blanco and P. K. Rai, *nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with KD-trees*, 2014, URL: https://github.com/jlblancoc/nanoflann (cited on pp. 4, 77, 160).

[Boc33] S. Bochner, *Monotone Funktionen, Stieltjessche Integrale und harmonische Analyse*, Mathematische Annalen **108**(1):378–410, 1933, DOI: 10.1007/bf01452844 (cited on p. 19).

[BFE12] E. F. Bollig, N. Flyer, and G. Erlebacher, *Solution to PDEs using radial basis function finite-differences (RBF-FD) on multiple GPUs*, Journal of Computational Physics **231**(21):7133–7151, 2012, DOI: 10.1016/j.jcp.2012.06.030 (cited on p. 37).

[BS07] S. Brenner and R. Scott, *The mathematical theory of finite element methods*, Texts in applied mathematics **15**, Springer, 2007, DOI: 10.1007/978-0-387-75934-0 (cited on p. 27).

[Bri07] R. Bridson, *Fast Poisson disk sampling in arbitrary dimensions*, in: SIGGRAPH sketches, 2007, p. 22, DOI: 10.1145/1278780.1278807 (cited on pp. 58, 61, 66, 76).

[BNT19] P. D. Brubeck, Y. Nakatsukasa, and L. N. Trefethen, *Vandermonde with Arnoldi*, arXiv:1911.09988, 2019 (cited on p. 46).

[Buh03] M. D. Buhmann, *Radial basis functions: theory and implementations*, Cambridge Monographs on Applied and Computational Mathematics **12**, Cambridge University Press, July 2003, DOI: 10.1017/cbo9780511543241 (cited on p. 6).

[Car+03] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell, *Smooth surface reconstruction from noisy range data*, in: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, ACM, 2003, 119–ff, DOI: 10.1145/604471.604495 (cited on p. 69).

[CS07] G. Chandhini and Y. V. S. S. Sanyasiraju, *Local RBF-FD solutions for steady convection–diffusion problems*, International Journal for Numerical Methods in Engineering **72**(3):352–378, 2007, DOI: 10.1002/nme.2024 (cited on p. 37).

[CB15] J.-S. Chen and T. Belytschko, *Meshless and meshfree methods*, in: Encyclopedia of Applied and Computational Mathematics, ed. by B. Engquist, Springer Berlin Heidelberg, 2015, pp. 886–894, DOI: 10.1007/978-3-540-70529-1_531 (cited on p. 36).

[Che+03] A. H.-D. Cheng, M. A. Golberg, E. J. Kansa, and G. Zammito, *Exponential convergence and H-c multiquadric collocation method for partial differential equations*, Numerical Methods for Partial Differential Equations: An International Journal **19**(5):571–594, 2003, DOI: 10.1002/num.10062 (cited on p. 36).

[CK99] Y. Choi and S. Kim, *Node generation scheme for meshfree method by Voronoi diagram and weighted bubble packing*, in: Fifth US national congress on computational mechanics, Boulder, CO, 1999 (cited on p. 58).

[CR73] P. G. Ciarlet and P.-A. Raviart, *Maximum principle and uniform convergence for the finite element method*, Computer Methods in Applied Mechanics and Engineering **2**(1):17–31, 1973, DOI: 10.1016/0045-7825(73)90019-4 (cited on p. 1).

[Coo86] R. L. Cook, *Stochastic sampling in computer graphics*, ACM Transactions on Graphics (TOG) **5**(1):51–72, 1986, DOI: 10.1145/7529.8927 (cited on pp. 58, 61).

[Cou43] R. Courant, *Variational methods for the solution of problems of equilibrium and vibrations*, Bulletin of the American Mathematical Society **49**(1):1–24, Jan. 1943, DOI: 10.1090/s0002-9904-1943-07818-4 (cited on pp. 1, 157).

[Cre+15] A. J. C. Crespo, J. M. Domínguez, B. D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, and O. García-Feal, *DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH)*, Computer Physics Communications **187**:204–216, 2015, DOI: 10.1016/j.cpc.2014.10.004 (cited on p. 126).

[DO11] O. Davydov and D. T. Oanh, *Adaptive meshless centres and RBF stencils for Poisson equation*, Journal of Computational Physics **230**(2):287–304, 2011, DOI: 10.1016/j.jcp.2010.09.005 (cited on pp. 89, 93, 94).

[Dem06] L. Demkowicz, *Computing with hp-adaptive finite elements: volume I: One and two dimensional elliptic and Maxwell problems*, Chapman & Hall/CRC Applied Mathematics & Nonlinear Science, CRC Press, 2006, DOI: 10.1201/9781420011685 (cited on p. 101).

[DKS19] M. Depolli, G. Kosec, and J. Slak, *Parallelizing a node positioning algorithm for meshless method*, in: Book of abstracts: ParNum 2019, Dubrovnik, Croatia, October 28–30, 2019, URL: https://www.fsb.unizg.hr/parnum2019/abs_book_web.pdf (cited on pp. 4, 85, 159).

[DHS13] P. Diaconis, S. Holmes, and M. Shahshahani, *Sampling from a manifold*, in: Advances in modern statistical theory and applications: a Festschrift in honor of Morris L. Eaton, Institute of Mathematical Statistics Collections **10**, Institute of Mathematical Statistics, 2013, pp. 102–125, DOI: 10.1214/12-imscoll1006 (cited on p. 69).

[DH07] T. A. Driscoll and A. R. H. Heryudono, *Adaptive residual subsampling methods for radial basis function interpolation and collocation problems*, Computers & Mathematics with Applications **53**(6):927–939, 2007, DOI: 10.1016/j.camwa.2006.06.005 (cited on p. 93).

[Dru+08] C. Drumm, S. Tiwari, J. Kuhnert, and H.-J. Bart, *Finite pointset method for simulation of the liquid–liquid flow field in an extractor*, Computers & Chemical Engineering **32**(12):2946–2957, 2008, DOI: 10.1016/j.compchemeng.2008.03.009 (cited on p. 58).

[DGJ02] Q. Du, M. Gunzburger, and L. Ju, *Meshfree, probabilistic determination of point sets and support regions for meshless computing*, Computer Methods in Applied Mechanics and Engineering **191**(13-14):1349–1366, 2002, DOI: 10.1016/s0045-7825(01)00327-9 (cited on p. 58).

[Duh+20] U. Duh, M. Depolli, J. Slak, and G. Kosec, *Parallel point sampling for 3D bodies*, in: MIPRO 2020: 43rd International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, 2020 (cited on pp. 4, 85, 159).

[DKS20] U. Duh, G. Kosec, and J. Slak, *Fast variable density node generation on parametric surfaces with application to mesh-free methods*, arXiv:2005.08767, May 2020, URL: https://arxiv.org/abs/2005.08767 (cited on p. 70).

[EFK15] M. Ebrahimnejad, N. Fallah, and A. R. Khoei, *Adaptive refinement in the meshless finite volume method for elasticity problems*, Computers & Mathematics with Applications **69**(12):1420–1443, 2015, DOI: 10.1016/j.camwa.2015.03.023 (cited on pp. 93, 95).

[FP09] A. Fabri and S. Pion, *CGAL: The computational geometry algorithms library*, in: Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems, 2009, pp. 538–539, DOI: 10.1145/1653771.1653865 (cited on p. 57).

[Fas07] G. E. Fasshauer, *Meshfree approximation methods with MATLAB*, Interdisciplinary Mathematical Sciences **6**, World Scientific, 2007, DOI: 10.1142/6437 (cited on pp. 22, 36).

[Fas11] G. E. Fasshauer, *Positive definite kernels: past, present and future*, Dolomite Research Notes on Approximation **4**:21–63, 2011, URL: http://www.math.iit.edu/~fass/PDKernels.pdf (cited on p. 8).

[FZ07a] G. E. Fasshauer and J. G. Zhang, *On choosing "optimal" shape parameters for RBF approximation*, Numerical Algorithms **45**(1-4):345–368, 2007, DOI: 10.1007/s11075-007-9072-8 (cited on p. 27).

[Fly+16] N. Flyer, B. Fornberg, V. Bayona, and G. A. Barnett, *On the role of polynomials in RBF-FD approximations: I. Interpolation and accuracy*, Journal of Computational Physics **321**:21–38, Sept. 2016, DOI: 10.1016/j.jcp.2016.05.026 (cited on pp. 37, 40, 47, 49, 92).

[Fly+12] N. Flyer, E. Lehto, S. Blaise, G. B. Wright, and A. St-Cyr, *A guide to RBF-generated finite differences for nonlinear transport: Shallow water simulations on a sphere*, Journal of Computational Physics **231**(11):4078–4095, 2012, DOI: 10.1016/j.jcp.2012.01.028 (cited on p. 37).

[Fol+11] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, *An overview of the HDF5 technology suite and its applications*, in: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, ACM, 2011, pp. 36–47, DOI: 10.1145/1966895.1966900 (cited on pp. 4, 126, 160).

[FF15a] B. Fornberg and N. Flyer, *A primer on radial basis functions with applications to the geosciences*, CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Sept. 2015, DOI: 10.1137/1.9781611974041 (cited on pp. 2, 158).

[FF15b] B. Fornberg and N. Flyer, *Fast generation of 2-D node distributions for mesh-free PDE discretizations*, Computers & Mathematics with Applications **69**(7):531–544, 2015, DOI: 10.1016/j.camwa.2015.01.009 (cited on pp. 2, 58, 59, 64, 69, 85, 158).

[FF15c] B. Fornberg and N. Flyer, *Solving PDEs with radial basis functions*, Acta Numerica **24**:215–258, 2015, DOI: 10.1017/s0962492914000130 (cited on pp. 2, 37, 158).

[FLF11] B. Fornberg, E. Larsson, and N. Flyer, *Stable computations with Gaussian radial basis functions*, SIAM Journal on Scientific Computing **33**(2):869–892, 2011, DOI: 10.1137/09076756x (cited on p. 36).

[FP08] B. Fornberg and C. Piret, *A stable algorithm for flat radial basis functions on a sphere*, SIAM Journal on Scientific Computing **30**(1):60–80, 2008, DOI: 10.1137/060671991 (cited on p. 27).

[FW04] B. Fornberg and G. Wright, *Stable computation of multiquadric interpolants for all values of the shape parameter*, Computers & Mathematics with Applications **48**(5-6):853–867, 2004, DOI: 10.1016/j.camwa.2003.08.010 (cited on p. 27).

[FZ07b] B. Fornberg and J. Zuev, *The Runge phenomenon and spatially variable shape parameters in RBF interpolation*, Computers & Mathematics with Applications **54**(3):379–398, 2007, DOI: 10.1016/j.camwa.2007.01.028 (cited on pp. 27, 93).

[Fos+11] C. Foster et al., *tinyformat: Minimal, type safe printf replacement library for C++*, 2011, URL: http://rapidxml.sourceforge.net (cited on pp. 4, 160).

[FS98] C. Franke and R. Schaback, *Solving partial differential equations by collocation using radial basis functions*, Applied Mathematics and Computation **93**(1):73–82, 1998, DOI: 10.1016/s0096-3003(97)10104-7 (cited on p. 36).

[FG08] P. J. Frey and P.-L. George, *Quadtree-octree based methods*, in: Mesh Generation: Application to Finite Elements, 2nd ed., Wiley, 2008, chap. 5, pp. 163–199, DOI: 10.1002/9780470611166.ch5 (cited on p. 58).

[GM77] R. A. Gingold and J. J. Monaghan, *Smoothed particle hydrodynamics: theory and application to non-spherical stars*, Monthly Notices of the Royal Astronomical Society **181**(3):375–389, 1977, DOI: 10.1093/mnras/181.3.375 (cited on p. 35).

[GW78] W. J. Gordon and J. A. Wixom, *Shepard's method of "metric interpolation" to bivariate and multivariate interpolation*, Mathematics of Computation **32**(141): 253–264, 1978, DOI: 10.2307/2006273 (cited on p. 12).

[Gu+12] X. D. Gu, W. Zeng, F. Luo, and S.-T. Yau, *Numerical computation of surface conformal mappings*, Computational Methods and Function Theory **11**(2): 747–787, 2012, DOI: 10.1007/BF03321885 (cited on p. 69).

[GJ+10] G. Guennebaud, B. Jacob, et al., *Eigen v3*, 2010, URL: http://eigen.tuxfamily.org (cited on pp. 4, 160).

[HS04] D. P. Hardin and E. B. Saff, *Discretizing manifolds via minimum energy points*, Notices of the AMS **51**(10):1186–1194, 2004 (cited on pp. 58, 69).

[HMS16] D. P. Hardin, T. Michaels, and E. B. Saff, *A comparison of popular point configurations on* $\mathbb{S}^2$, Dolomites Research Notes on Approximation **9**(1), 2016 (cited on p. 69).

[HSW12] D. P. Hardin, E. B. Saff, and J. T. Whitehouse, *Quasi-uniformity of minimal weighted energy points on compact metric spaces*, Journal of Complexity **28**(2):177–191, 2012, DOI: 10.1016/j.jco.2011.10.009 (cited on pp. 27, 28).

[Har71] R. L. Hardy, *Multiquadric equations of topography and other irregular surfaces*, Journal of Geophysical Research **76**(8):1905–1915, 1971, DOI: 10.1029/jb076i008p01905 (cited on pp. 5, 26).

[HMÅ02] M. Haveraaen, H. Mnthe-Kaas, and K. Åhlander, *On object-oriented frameworks and coordinate free formulations of PDEs*, Engineering with Computers **18**(4):286–294, 2002, DOI: 10.1007/s003660200026 (cited on p. 125).

[Hec12] F. Hecht, *New development in FreeFem++*, Journal of numerical mathematics **20**(3-4):251–265, 2012, DOI: 10.1515/jnum-2012-0013, URL: https://freefem.org/ (cited on pp. 123, 125).

[Her82] H. Hertz, *Über die Berührung fester elastischer Körper.*, Journal für die reine und angewandte Mathematik **92**:156–171, 1882 (cited on p. 116).

[Hin15] T. Hines, *RBF: Python package containing the tools necessary for radial basis function (RBF) applications*, 2015, URL: https://github.com/treverhines/RBF (cited on p. 126).

[Hoj+14] R. Hojjati-Talemi, M. A. Wahab, J. De Pauw, and P. De Baets, *Prediction of fretting fatigue crack initiation and propagation lifetime for cylindrical contact configuration*, Tribology International **76**:73–91, 2014, DOI: 10.1016/j.triboint.2014.02.017 (cited on p. 121).

[Hre41] A. Hrennikoff, *Solution of problems of elasticity by the framework method*, Journal of Applied Mechanics **8**(4):169–175, 1941 (cited on pp. 1, 157).

[Hu+19] W. Hu, N. Trask, X. Hu, and W. Pan, *A spatially adaptive high-order meshless method for fluid–structure interactions*, Computer Methods in Applied Mechanics and Engineering **355**:67–93, 2019, DOI: 10.1016/j.cma.2019.06.009 (cited on pp. 95, 100).

[HCB05] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs, *Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement*, Computer Methods in Applied Mechanics and Engineering **194**(39-41):4135–4195, 2005, DOI: 10.1016/j.cma.2004.10.008 (cited on p. 92).

[Ink20] Inkscape Developers, *Inkscape: open source scalable vector graphics editor. Draw freely.* 1.0 (4035a4fb49, 2020-05-01), May 2020, URL: https://inkscape.org/ (cited on pp. 4, 160).

[Int18] Intel developers, *Intel Math Kernel Library*, version 2018.1.63, 2018, URL: http://software.intel.com/en-us/intel-mkl (cited on pp. 4, 160).

[ISO17] ISO, *ISO/IEC 14882:2017 Information technology — Programming languages — C++*, Fifth, Geneva, Switzerland: International Organization for Standardization, Dec. 2017, p. 1605, URL: https://www.iso.org/standard/68564.html (cited on p. 4).

[Jac+16] S. J. Jackson, D. Stevens, D. Giddings, and H. Power, *An adaptive RBF finite collocation approach to track transport processes across moving fronts*, Computers & Mathematics with Applications **71**(1):278–300, 2016, DOI: 10.1016/j.camwa.2015.11.015 (cited on p. 93).

[JSK19] M. Jančič, J. Slak, and G. Kosec, *Analysis of high order dimension independent RBF-FD solution of Poisson's equation*, arXiv:1909.01126, Sept. 2019, URL: https://arxiv.org/abs/1909.01126 (cited on p. 92).

[JSK20] M. Jančič, J. Slak, and G. Kosec, *GPU accelerated RBF-FD solution of Poisson's equation*, in: MIPRO 2020: 43rd International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, 2020 (cited on pp. 4, 159).

[JDX14] A. Javed, K. Djijdeli, and J. T. Xing, *Shape adaptive RBF-FD implicit scheme for incompressible viscous Navier–Stokes equations*, Computers & Fluids **89**:38–52, 2014, DOI: 10.1016/j.compfluid.2013.10.028 (cited on p. 92).

[Jav+19] A. Javed, F. Mazhar, T. A. Shams, M. Ayaz, and N. Hussain, *A stabilized RBF finite difference method for convection dominated flows over meshfree nodes*, Engineering Analysis with Boundary Elements **107**:159–167, 2019, DOI: 10.1016/j.enganabound.2019.07.008 (cited on p. 37).

[Jef+15] A. Jefferies, J. Kuhnert, L. Aschenbrenner, and U. Giffhorn, *Finite pointset method for the simulation of a vehicle travelling through a body of water*, in: Meshfree methods for partial differential equations VII, ed. by M. Griebel and M. A. Schweitzer, Lecture Notes in Computational Science and Engineering **100**, Springer, Nov. 2015, pp. 205–221, DOI: 10.1007/978-3-319-06898-5_11 (cited on p. 36).

[Kal11] M. Kalicinski, *RapidXml*, 2011, URL: https://github.com/c42f/tinyformat (cited on pp. 4, 160).

[KH17] E. J. Kansa and P. Holoborodko, *On the ill-conditioned nature of C^∞ RBF strong collocation*, Engineering Analysis with Boundary Elements **78**:26–30, 2017, DOI: 10.1016/j.enganabound.2017.02.006 (cited on p. 37).

[Kan90a] E. J. Kansa, *Multiquadrics—A scattered data approximation scheme with applications to computational fluid-dynamics—I surface approximations and partial derivative estimates*, Computers & Mathematics with applications **19**(8-9):127–145, 1990, DOI: 10.1016/0898-1221(90)90270-t (cited on p. 36).

[Kan90b] E. J. Kansa, *Multiquadrics—A scattered data approximation scheme with applications to computational fluid-dynamics—II solutions to parabolic, hyperbolic and elliptic partial differential equations*, Computers & mathematics with applications **19**(8-9):147–161, 1990, DOI: 10.1016/0898-1221(90)90271-k (cited on pp. 36, 53).

[KF07] A. M. Kibriya and E. Frank, *An empirical comparison of exact nearest neighbour algorithms*, in: European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2007, pp. 140–151, DOI: 10.1007/978-3-540-74976-9_16 (cited on p. 66).

[KB13] M. Kindelan and V. Bayona, *Application of the RBF meshless method to laminar flame propagation*, Engineering Analysis with Boundary Elements **37**(12):1617–1624, 2013, DOI: 10.1016/j.enganabound.2013.09.004 (cited on p. 37).

[KM15] N. P. Kopytov and E. A. Mityushov, *Uniform distribution of points on hypersurfaces: simulation of random equiprobable rotations*, Vestnik Udmurtskogo Universiteta. Matematika. Mekhanika. Komp'yuternye Nauki **25**(1):29–35, Mar. 2015, DOI: 10.20537/vm150104 (cited on p. 69).

[Kos18] G. Kosec, *A local numerical solution of a fluid-flow problem on an irregular domain*, Advances in engineering software **120**:36–44, June 2018, DOI: 10.1016/j.advengsoft.2016.05.010 (cited on p. 131).

[KŠ08] G. Kosec and B. Šarler, *Local RBF collocation method for Darcy flow*, Computer Modeling in Engineering and Sciences **25**(3):197, 2008, DOI: 10.3970/cmes.2008.025.197 (cited on p. 37).

[KŠ11] G. Kosec and B. Šarler, *h-adaptive local radial basis function collocation meshless method*, Computers, Materials & Continua **26**(3):227–254, 2011, DOI: 10.3970/cmc.2011.026.227 (cited on pp. 93, 94).

[KS18a] G. Kosec and J. Slak, *Numerical simulation of natural convection from a heated cylinder*, in: Proceedings of the International Conference on Computational Methods, ICCM2018, Rome, Italy, August 6–10, ed. by G.-R. Liu and P. Trovalusci, Proceedings of the international conference on computational methods **5**, Scientech Publisher, 2018, pp. 887–896 (cited on pp. 4, 159).

[KS18b] G. Kosec and J. Slak, *Numerical simulation of overhead power line cooling in natural convection regime*, in: ECT2018, The Tenth International Conference on Engineering Computational Technology, Stiges, Barcelona, Spain, August 21–25, Civil-comp proceedings, Elsevier, 2018 (cited on pp. 4, 159).

[KS18c] G. Kosec and J. Slak, *RBF-FD based dynamic thermal rating of overhead power lines*, in: Advances in fluid mechanics XII, ed. by S. Hernández, L. Škerget, and J. Ravnik, WIT Transactions on Engineering Sciences **120**, Wessex institute, WIT press, 2018, pp. 255–262, DOI: 10.2495/afm180261 (cited on pp. 4, 37, 159).

[KS19a] G. Kosec and J. Slak, *Modular implementation of local meshless numerical method*, in: Book of abstracts: ParNum 2019, Dubrovnik, Croatia, October 28–30, 2019, URL: https://www.fsb.unizg.hr/parnum2019/abs_book_web.pdf (cited on pp. 4, 159).

[KS19b] G. Kosec and J. Slak, *Parallel RBF-FD solution of the Boussinesq's problem*, in: Proceedings of the Sixth International Conference on Parallel, Distributed, GPU and Cloud Computing for Engineering, Pécs, Hungary, June 5–6, ed. by P. Iványi and B. H. V. Topping, Civil-comp proceedings, Stirlingshire: Civil-Comp Press, 2019, DOI: 10.4203/ccp.112.7 (cited on pp. 4, 135, 159).

[Kos+19] G. Kosec, J. Slak, M. Depolli, R. Trobec, K. Pereira, S. Tomar, T. Jacquemin, S. P. A. Bordas, and M. A. Wahab, *Weak and strong from meshless methods for linear elastic problem under fretting contact conditions*, Tribology International **138**:392–402, Oct. 2019, DOI: 10.1016/j.triboint.2019.05.041 (cited on pp. 121, 123).

[KMM17] J. Kuhnert, I. Michel, and R. Mack, *Fluid structure interaction (FSI) in the MESHFREE Finite Pointset Method (FPM): theory and applications*, in: International Workshop on Meshfree Methods for Partial Differential Equations, ed. by M. Griebel and M. A. Schweitzer, Lecture Notes in Computational Science and Engineering **129**, Springer, Springer, 2017, pp. 73–92, DOI: 10.1007/978-3-030-15119-5_5 (cited on p. 126).

[Kur+08] J. Kurtz, D. Pardo, M. Paszynski, W. Rachowicz, and A. Zdunek, *Computing with hp-adaptive finite elements: volume II, Frontiers: three dimensional elliptic and Maxwell problems with applications*. Chapman & Hall/CRC Applied Mathematics & Nonlinear Science, CRC Press, 2008 (cited on p. 104).

[Lan79] P. Lancaster, *Moving weighted least-squares methods*, in: Polynomial and spline approximation, ed. by B. N. Sahney, NATO Advanced Study Institute Series C **49**, Reidel, Dordrecht: Springer, 1979, pp. 103–120, DOI: 10.1007/978-94-009-9443-0_7 (cited on p. 13).

[LF05] E. Larsson and B. Fornberg, *Theoretical and computational aspects of multivariate interpolation with increasingly flat radial basis functions*, Computers & Mathematics with Applications **49**(1):103–130, 2005, DOI: 10.1016/j.camwa.2005.01.010 (cited on p. 27).

[Li+17] J. Li, S. Zhai, Z. Weng, and X. Feng, *h-adaptive RBF-FD method for the high-dimensional convection-diffusion equation*, International Communications in Heat and Mass Transfer **89**:139–146, 2017, DOI: 10.1016/j.icheatmasstransfer.2017.06.001 (cited on p. 93).

[LTU00a] X. Y. Li, S. H. Teng, and A. Ungor, *Generating a good quality point set for the mesh-less methods*, Computer Modeling in Engineering Sciences (CMES) **1**(1):10–17, 2000 (cited on p. 60).

[LTU00b] X.-Y. Li, S.-H. Teng, and A. Ungor, *Point placement for meshless methods using sphere packing and advancing front methods*, in: ICCES'00, Los Angeles, CA, 2000 (cited on p. 58).

[Lib+08] N. A. Libre, A. Emdadi, E. J. Kansa, M. Rahimian, and M. Shekarchi, *A stabilized RBF collocation scheme for Neumann type boundary value problems*, Computer Modeling in Engineering & Sciences **24**(1):61–80, 2008, DOI: 10.3970/cmes.2008.024.061 (cited on p. 93).

[LO80] T. Liszka and J. Orkisz, *The finite difference method at arbitrary irregular grids and its application in applied mechanics*, Computers & Structures **11**(1-2):83–95, 1980, DOI: 10.1016/0045-7949(80)90149-2 (cited on pp. 36, 56).

[LLS01] N. Litke, A. Levin, and P. Schröder, *Fitting subdivision surfaces*, in: Proceedings of the conference on Visualization'01, IEEE Computer Society, 2001, pp. 319–324, DOI: 10.1109/visual.2001.964527 (cited on p. 69).

[Liu02] G.-R. Liu, *Mesh free methods: moving beyond the finite element method*, CRC Press, 2002, DOI: 10.1201/9781420040586 (cited on pp. 36, 57, 126).

[Liu+95] W. K. Liu, S. Jun, S. Li, J. Adee, and T. Belytschko, *Reproducing kernel particle methods for structural dynamics*, International Journal for Numerical Methods in Engineering **38**(10):1655–1679, 1995, DOI: 10.1002/nme.1620381005 (cited on p. 36).

[Liu+10] Y. Liu, Y. Nie, W. Zhang, and L. Wang, *Node placement method by bubble simulation and its application*, Computer Modeling in Engineering and Sciences(CMES) **55**(1):89, 2010, DOI: 10.3970/cmes.2010.055.089 (cited on p. 58).

[LW10] A. Logg and G. N. Wells, *DOLFIN: Automated finite element computing*, ACM Transactions on Mathematical Software (TOMS) **37**(2):20, 2010, DOI: 10.1145/1731022.1731030 (cited on p. 125).

[LO98] R. Löhner and E. Onate, *An advancing front point generation technique*, Communications in Numerical Methods in Engineering **14**(12):1097–1108, 1998, DOI: 10.1002/(sici)1099-0887(199812)14:12<1097::aid-cnm183>3.0.co;2-7 (cited on p. 58).

[LP88] R. Löhner and P. Parikh, *Generation of three-dimensional unstructured grids by the advancing-front method*, International Journal for Numerical Methods in Fluids **8**(10):1135–1149, 1988, DOI: 10.1002/fld.1650081003 (cited on p. 58).

[MKX07] J. Ma, P. Krishnaswami, and X. J. Xin, *A truly meshless pre- and post-processor for meshless analysis methods*, Advances in Engineering Software **38**(1):9–30, Jan. 2007, DOI: 10.1016/j.advengsoft.2006.07.001 (cited on p. 35).

[Mak+19] M. Maksić, V. Djurica, A. Souvent, J. Slak, M. Depolli, and G. Kosec, *Cooling of overhead power lines due to the natural convection*, International Journal of Electrical Power & Energy Systems **113**:333–343, Dec. 2019, DOI: 10.1016/j.ijepes.2019.05.005.

[Mat17] Matlab Developers, *MATLAB version 9.2.0.538062 (R2017a)*, The Mathworks, Inc., Natick, Massachusetts, 2017 (cited on pp. 4, 160).

[McE49] E. McEwen, *Stresses in elastic cylinders in contact along a generatrix (including the effect of tangential friction)*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **40**(303):454–459, 1949, DOI: 10.1080/14786444908521733 (cited on p. 117).

[MB96] J. M. Melenk and I. Babuška, *The partition of unity finite element method: Basic theory and applications*, Computer Methods in Applied Mechanics and Engineering **139**(1-4):289–314, Dec. 1996, DOI: 10.1016/s0045-7825(96)01087-0 (cited on p. 36).

[Mer10] Z. Merali, *Computational science: ...Error: why scientific programming does not compute*, Nature **467**(7317):775–777, Oct. 2010, DOI: 10.1038/467775a (cited on p. 125).

[Mil12] S. Milewski, *Meshless finite difference method with higher order approximation—applications in mechanics*, Archives of Computational Methods in Engineering **19**(1):1–49, 2012, DOI: 10.1007/s11831-012-9068-y (cited on pp. 89, 92).

[Mil13] S. Milewski, *Selected computational aspects of the meshless finite difference method*, Numerical Algorithms **63**(1):107–126, 2013, DOI: 10.1007/s11075-012-9614-6 (cited on p. 126).

[MS18] S. Milovanović and L. von Sydow, *Radial basis function generated finite differences for option pricing problems*, Computers & Mathematics with Applications **75**(4):1462–1481, 2018, DOI: 10.1016/j.camwa.2017.11.015 (cited on p. 37).

[Mis13] R. von Mises, *Mechanik der festen Körper im plastisch-deformablen Zustand*, Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse **1913**:582–592, 1913 (cited on p. 109).

[Mis+20] P. K. Mishra, L. Ling, X. Liu, and M. K. Sen, *Adaptive radial basis function generated finite-difference (RBF-FD) on non-uniform nodes using p-refinement*, arXiv:2004.06319, 2020, URL: https://arxiv.org/abs/2004.06319 (cited on p. 92).

[Mit+12] S. A. Mitchell, A. Rand, M. S. Ebeida, and C. Bajaj, *Variable radii Poisson-disk sampling, extended version*, in: Proceedings of the 24th Canadian conference on Computational geometry (CCCG'12), 2012, URL: https://www.sandia.gov/~samitch/_assets/documents/pdfs/VarRadiusPoissonDiskCCCG-bw2.pdf (cited on p. 68).

[Moč+20] J. Močnik Berljavac, P. K. Mishra, J. Slak, and G. Kosec, *RBF-FD analysis of 2D time-domain acoustic wave propagation in heterogeneous Earth's subsurface*, arXiv:2001.01597, Jan. 2020, URL: https://arxiv.org/abs/2001.01597.

[MSK19] J. Močnik Berljavac, J. Slak, and G. Kosec, *Parallel simulation of time-domain acoustic wave propagation*, in: MIPRO 2019: 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, May 20–24, ed. by K. Skala, MIPRO proceedings, IEEE, Croatian Society for Information, Communication Technology, Elec-

tronics, and Microelectronics, 2019, DOI: 10.23919/mipro.2019.8756946 (cited on pp. 4, 159).

[Moo91] A. W. Moore, *An introductory tutorial on kd-trees*, PhD thesis 209, Computer Laboratory, University of Cambridge, 1991, URL: https://www.ri.cmu.edu/pub_files/pub1/moore_andrew_1991_1/moore_andrew_1991_1.pdf (cited on p. 66).

[MM97] Y. U. X. Mukherjee and S. Mukherjee, *The boundary node method for potential problems*, International Journal for Numerical Methods in Engineering **40**(5):797–815, 1997, DOI: 10.1002/(sici)1097-0207(19970315)40:5<797::aid-nme89>3.0.co;2-# (cited on p. 36).

[MH96] H. Munthe-Kaas and M. Haveraaen, *Coordinate free numerics — closing the gap between 'pure' and 'applied' mathematics?*, Zeitschrift für Angewandte Mathematik und Mechanik **76**(suppl. 1):487–488, 1996 (cited on p. 125).

[NTV92] B. Nayroles, G. Touzot, and P. Villon, *Generalizing the finite element method: diffuse approximation and diffuse elements*, Computational Mechanics **10**(5):307–318, 1992, DOI: 10.1007/bf00364252 (cited on p. 35).

[Ngu+08] V. P. Nguyen, T. Rabczuk, S. Bordas, and M. Duflot, *Meshless methods: a review and computer implementation aspects*, Mathematics and Computers in Simulation **79**(3):763–813, 2008, DOI: 10.1016/j.matcom.2008.01.003 (cited on pp. 2, 36, 45, 55, 126, 158).

[ODP17] D. T. Oanh, O. Davydov, and H. X. Phu, *Adaptive RBF-FD method for elliptic problems with point singularities in 2D*, Applied Mathematics and Computation **313**:474–497, 2017, DOI: 10.1016/j.amc.2017.06.006 (cited on pp. 93–95).

[Omo89] S. M. Omohundro, *Five balltree construction algorithms*, tech. rep., Berkeley: International Computer Science Institute, 1989 (cited on p. 66).

[Oña+96] E. Oñate, S. Idelsohn, O. C. Zienkiewicz, and R. L. Taylor, *A finite point method in computational mechanics. Applications to convective transport and fluid flow*, International Journal for Numerical Methods in Engineering **39**(22):3839–3866, 1996, DOI: 10.1002/(sici)1097-0207(19961130)39:22<3839::aid-nme27>3.0.co;2-r (cited on pp. 36, 56).

[OOI07] E. Ortega, E. Oñate, and S. Idelsohn, *An improved finite point method for tridimensional potential flows*, Computational Mechanics **40**(6):949–963, 2007, DOI: 10.1007/s00466-006-0154-6 (cited on p. 36).

[PLP08] F. Perazzo, R. Löhner, and L. Perez-Pozo, *Adaptive methodology for meshless finite point method*, Advances in Engineering Software **39**(3):156–166, 2008, DOI: 10.1016/j.advengsoft.2007.02.007 (cited on pp. 89, 93).

[Per+16] K. Pereira, S. Bordas, S. Tomar, R. Trobec, M. Depolli, G. Kosec, and M. Abdel Wahab, *On the convergence of stresses in fretting fatigue*, Materials **9**(8):639, July 2016, DOI: 10.3390/ma9080639 (cited on pp. 121, 123).

[PK75] N. Perrone and R. Kao, *A general finite difference method for arbitrary meshes*, Computers & Structures **5**(1):45–57, 1975, DOI: 10.1016/0045-7949(75)90018-8 (cited on p. 36).

[PS04] P.-O. Persson and G. Strang, *A simple mesh generator in MATLAB*, SIAM Review **46**(2):329–345, 2004, DOI: 10.1137/s0036144503429121 (cited on p. 58).

[Pet+19] A. Petras, L. Ling, C. Piret, and S. J. Ruuth, *A least-squares implicit RBF-FD closest point method and applications to PDEs on moving surfaces*, Journal of Computational Physics **381**:146–161, 2019, DOI: 10.1016/j.jcp.2018.12.031 (cited on p. 37).

[PT12] L. Piegl and W. Tiller, *The NURBS book*, Monographs in Visual Communication, Springer, 2012 (cited on p. 69).

[PV05] D. E. Post and L. G. Votta, *Computational science demands a new paradigm*, Physics today **58**(1):35–41, 2005, DOI: 10.1063/1.1881898 (cited on p. 125).

[RB05] T. Rabczuk and T. Belytschko, *Adaptivity for structured meshfree particle methods in 2D and 3D*, International Journal for Numerical Methods in Engineering **63**(11):1559–1582, 2005, DOI: 10.1002/nme.1326 (cited on p. 95).

[RPD06] W. Rachowicz, D. Pardo, and L. Demkowicz, *Fully automatic hp-adaptivity in three dimensions*, Computer Methods in Applied Mechanics and Engineering **195**(37-40):4816–4842, July 2006, DOI: 10.1016/j.cma.2005.08.022 (cited on p. 91).

[Ric11] L. F. Richardson, *The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam*, Philosophical Transactions of the Royal Society A **210**(459-470):307–357, Jan. 1911, DOI: 10.1098/rsta.1911.0009 (cited on pp. 1, 157).

[RAX10] D. W. I. Rouson, H. Adalsteinsson, and J. Xia, *Design patterns for multiphysics modeling in Fortran 2003 and C++*, ACM Transactions on Mathematical Software (TOMS) **37**(1):1–30, 2010, DOI: 10.1145/1644001.1644004 (cited on p. 125).

[Rup95] J. Ruppert, *A Delaunay refinement algorithm for quality 2-dimensional mesh generation*, Journal of Algorithms **18**(3):548–585, 1995, DOI: 10.1006/jagm.1995.1021 (cited on p. 58).

[Sad14] M. H. Sadd, *Elasticity: theory, applications, and numerics*, 3rd, Academic Press, 2014 (cited on pp. 3, 107, 109, 158).

[SF19] K. van der Sande and B. Fornberg, *Fast variable density 3-D node generation*, arXiv:1906.00636, 2019, URL: https://arxiv.org/abs/1906.00636 (cited on pp. 59, 76, 129).

[SKS03] P. Sang-Hoon, K. Kie-Chan, and Y. Sung-Kie, *A posteriori error estimates and an adaptive scheme of least-squares meshfree method*, International Journal for Numerical Methods in Engineering **58**(8):1213–1250, 2003, DOI: 10.1002/nme.817 (cited on p. 95).

[ŠV06] B. Šarler and R. Vertnik, *Meshfree local radial basis function collocation method for diffusion problems*, Computers & Mathematics with Applications **51**(8):1269–1282, Apr. 2006, DOI: 10.1016/j.camwa.2006.04.013 (cited on p. 37).

[Sch95a] R. Schaback, *Creating surfaces from scattered data using radial basis functions*, Mathematical methods for curves and surfaces **477**, 1995 (cited on p. 23).

[Sch95b] R. Schaback, *Error estimates and condition numbers for radial basis function interpolation*, Advances in Computational Mathematics **3**(3):251–264, 1995, DOI: 10.1007/bf02432002 (cited on pp. 33, 34).

[SKF18] V. Shankar, R. M. Kirby, and A. L. Fogelson, *Robust node generation for mesh-free discretizations on irregular domains and surfaces*, SIAM Journal on Scientific Computing **40**(4):2584–2608, 2018, DOI: 10.1137/17m114090x (cited on pp. 2, 57, 58, 60, 62, 64, 69, 76, 85, 88, 158).

[SNK18] V. Shankar, A. Narayan, and R. M. Kirby, *RBF-LOI: Augmenting radial basis functions (RBFs) with least orthogonal interpolation (LOI) for solving PDEs on surfaces*, Journal of Computational Physics **373**:722–735, 2018, DOI: 10.1016/j.jcp.2018.07.015 (cited on p. 37).

[She68] D. Shepard, *A two-dimensional interpolation function for irregularly-spaced data*, in: Proceedings of the 1968 23rd ACM national conference, 1968, pp. 517–524, DOI: 10.1145/800186.810616 (cited on p. 12).

[She98] J. R. Shewchuk, *Tetrahedral mesh generation by Delaunay refinement*, in: Proceedings of the fourteenth annual symposium on Computational geometry, 1998, pp. 86–95, DOI: 10.1145/276884.276894 (cited on p. 58).

[SDY03] C. Shu, H. Ding, and K. S. Yeo, *Local radial basis function-based differential quadrature method and its application to solve two-dimensional incompressible Navier–Stokes equations*, Computer Methods in Applied Mechanics and Engineering **192**(7-8):941–954, 2003, DOI: 10.1016/s0045-7825(02)00618-7 (cited on p. 37).

[Sla17] J. Slak, *Solving linear elastostatic problems with meshless methods*, (in slovenian), MA thesis, University of Ljubljana, 2017 (cited on p. 118).

[SK16] J. Slak and G. Kosec, *Detection of heart rate variability from a wearable differential ECG device*, in: MIPRO 2016: 39th International Convention on Information and Communication Technology, Electronics and Microelectronics, May 30–June 3, 2016, Opatija, Croatia, ed. by P. Biljanović, MIPRO proceedings, IEEE, Croatian Society for Information, Communication Technology, Electronics, and Microelectronics, 2016, pp. 430–435, DOI: 10.1109/mipro.2016.7522182 (cited on pp. 4, 159).

[SK18a] J. Slak and G. Kosec, *Fast generation of variable density node distributions for mesh-free methods*, in: Boundary elements and other mesh reduction methods XXXXI, ed. by A. H.-D. Cheng and S. Syngellakis, WIT transactions on engineering sciences **122**, Wessex institute, WIT press, 2018, pp. 163–173, DOI: 10.2495/BE410151 (cited on pp. 4, 59, 159).

[SK18b] J. Slak and G. Kosec, *Generic implementation of meshless local strong form method*, in: ECT2018, The Tenth International Conference on Engineering Computational Technology, Stiges, Barcelona, Spain, September 4–8, Civil-comp proceedings, Elsevier, 2018 (cited on pp. 4, 159).

[SK18c] J. Slak and G. Kosec, *Parallel coordinate free implementation of local meshless method*, in: MIPRO 2018: 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, May 21–25, 2018, Opatija, Croatia, ed. by K. Skala, MIPRO proceedings, IEEE, Croatian Society for Information, Communication Technology, Electronics, and Microelectronics, 2018, pp. 194–200, DOI: 10.23919/mipro.2018.8400034 (cited on pp. 4, 133, 159).

[SK18d] J. Slak and G. Kosec, *Refined RBF-FD solution of linear elasticity problem*, in: Proceedings of the 3rd International Conference on Smart and Sustainable Technologies (SpliTech), Split, Croatia, June 26–29, ed. by T. Perković, FESB, University of Split, 2018, pp. 393–398, URL: https://ieeexplore.ieee.org/abstract/document/8448351 (cited on pp. 4, 159).

[SK19a] J. Slak and G. Kosec, *Adaptive radial basis function-generated finite differences method for contact problems*, International Journal for Numerical Methods in Engineering **119**(7):661–686, Aug. 2019, DOI: 10.1002/nme.6067 (cited on pp. 93, 94, 107).

[SK19b] J. Slak and G. Kosec, *Adaptive RBF-FD method for Poisson's equation*, in: Boundary elements and other mesh reduction methods XXXXII, ed. by A. Cheng and A. Tadeu, WIT transactions on engineering sciences **126**, Wessex institute, WIT Press, 2019, pp. 149–157, DOI: 10.2495/be420131 (cited on pp. 4, 100, 159).

[SK19c] J. Slak and G. Kosec, *Medusa: A C++ library for solving PDEs using strong form mesh-free methods*, arXiv:1912.13282, Dec. 2019, URL: https://arxiv.org/abs/1912.13282 (cited on pp. 3, 158).

[SK19d] J. Slak and G. Kosec, *On generation of node distributions for meshless PDE discretizations*, SIAM Journal on Scientific Computing **41**(5):A3202–A3229, Oct. 2019, DOI: 10.1137/18M1231456 (cited on pp. 59, 61, 62, 85, 88).

[SK19e] J. Slak and G. Kosec, *Refined meshless local strong form solution of Cauchy–Navier equation on an irregular domain*, Engineering Analysis with Boundary Elements **100**:3–13, Mar. 2019, DOI: 10.1016/j.enganabound.2018.01.001 (cited on pp. 93, 118, 121).

[SK20] J. Slak and G. Kosec, *Dynamic thermal rating in icing conditions*, in: 10th International Conference on Power, Energy and Electrical Engineering (CPEEE 2020), virtual, 2020 (cited on pp. 4, 159).

[SSK19a] J. Slak, B. Stojanovič, and G. Kosec, *High order RBF-FD approximations with application to a scattering problem*, in: Proceedings of the 4th International Conference on Smart and Sustainable Technologies (SpliTech), Bol, island of Brač and Split, Croatia, June 18–21, ed. by T. Perković, FESB, University of Split, 2019, DOI: 10.23919/splitech.2019.8782918 (cited on pp. 4, 92, 159).

[Sla12] W. S. Slaughter, *The linearized theory of elasticity*, New York: Springer, 2012 (cited on pp. 3, 107, 109, 113, 158).

[Sok+19] A. Sokolov, O. Davydov, D. Kuzmin, A. Westermann, and S. Turek, *A flux-corrected RBF-FD method for convection dominated problems in domains and on manifolds*, Journal of Numerical Mathematics **27**(4):253–269, 2019, DOI: 10.1515/jnma-2018-0097 (cited on p. 37).

[Ste+09] D. Stevens, H. Power, M. Lees, and H. Morvan, *The use of PDE centres in the local RBF Hermitian method for 3D convective-diffusion problems*, Journal of Computational Physics **228**:4606–4624, 2009, DOI: 10.1016/j.jcp.2009.03.025 (cited on p. 92).

[SSK19b] B. Stojanovič, J. Slak, and G. Kosec, *RBF-FD solution of electromagnetic scattering problem*, in: MIPRO 2019: 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, May 20–24, ed. by K. Skala, MIPRO proceedings, IEEE, Croatian Society for Information, Communication Technology, Electronics, and Microelectronics, 2019, DOI: 10.23919/mipro.2019.8756943 (cited on pp. 4, 159).

[SF73] G. Strang and G. J. Fix, *An analysis of the finite element method*, Prentice-Hall series in automatic computation, Englewood Cliffs, NJ: Prentice-Hall, 1973 (cited on p. 1).

[Str06] I. Stroud, *Boundary representation modelling techniques*, Springer, 2006 (cited on p. 69).

[Suc18] P. Suchde, *Conservation and accuracy in meshfree generalizedfinite difference method*, PhD thesis, University of Kaiserslautern, 2018 (cited on pp. 36, 37).

[Tan+11] Q. Tang, G. Y. Zhang, G. R. Liu, Z. H. Zhong, and Z. C. He, *A three-dimensional adaptive analysis using the meshfree node-based smoothed point interpolation method (NS-PIM)*, Engineering Analysis with Boundary Elements **35**(10):1123–1135, 2011, DOI: 10.1016/j.enganabound.2010.05.019 (cited on pp. 93, 95).

[TS03] A. I. Tolstykh and D. A. Shirobokov, *On using radial basis functions in a "finite difference mode" with applications to elasticity problems*, Computational Mechanics **33**(1):68–79, 2003, DOI: 10.1007/s00466-003-0501-9 (cited on p. 37).

[Tol00] A. I. Tolstykh, *On using RBF-based differencing formulas for unstructured and mixed structured-unstructured grid calculations*, in: 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation: Lausanne, Switzerland, August 21–25, ed. by M. Deville, 2000, pp. 4606–4624 (cited on p. 37).

[TK15] R. Trobec and G. Kosec, *Parallel scientific computing: theory, algorithms, and applications of mesh based and meshless methods*, SpringerBriefs in Computer Science, Springer, 2015, DOI: 10.1007/978-3-319-17073-2 (cited on p. 55).

[Vel95] T. Veldhuizen, *Expression templates*, C++ Report **7**(5):26–31, 1995 (cited on p. 132).

[VRL07] S. A. Viana, D. Rodger, and H. C. Lai, *Overview of meshless methods*, ICS Newsletter **14**(2), July 2007 (cited on p. 36).

[WL02] J. G. Wang and G. R. Liu, *A point interpolation meshless method based on radial basis functions*, International Journal for Numerical Methods in Engineering **54**(11):1623–1648, 2002, DOI: 10.1002/nme.489 (cited on p. 37).

[Wen04] H. Wendland, *Scattered data approximation*, Cambridge Monographs on Applied and Computational Mathematics **17**, Cambridge University Press, 2004, DOI: 10.1017/cbo9780511617539 (cited on pp. 2, 6, 11, 19, 20, 23, 26–28, 33, 34, 37, 158).

[WD00] J. A. Williams and R. S. Dwyer-Joyce, *Modern tribology handbook*, in: ed. by B. Bhushan, Mechanics & Materials Science **1**, Boca Raton: CRC Press, 2000, chap. Contact between solid surfaces, pp. 121–162, DOI: 10.1201/9780849377877.ch3 (cited on pp. 3, 116, 117, 158).

[Wri03] G. B. Wright, *Radial basis function interpolation: numerical and analytical developments*, PhD thesis, College of Engineering Boulder, CO, United States: University of Colorado at Boulder, 2003 (cited on p. 37).

[ZN18] R. Zamolo and E. Nobile, *Two algorithms for fast 2D node generation: Application to RBF meshless discretization of diffusion problems and image halftoning*, Computers & Mathematics with Applications **75**(12):4305–4321, June 2018, DOI: 10.1016/j.camwa.2018.03.031 (cited on pp. 58, 59).

[Zeg98] P. A. Zegeling, *r-refinement for evolutionary PDEs with finite elements or finite differences*, Applied Numerical Mathematics **26**(1-2):97–104, 1998, DOI: 10.1016/s0168-9274(97)00086-x (cited on p. 93).

[ZOF01] H.-K. Zhao, S. Osher, and R. Fedkiw, *Fast surface reconstruction using the level set method*, in: Proceedings IEEE Workshop on Variational and Level Set Methods in Computer Vision, IEEE, IEEE Computer Soc, 2001, pp. 194–201, DOI: 10.1109/vlsm.2001.938900 (cited on p. 69).

[ZHZ93] J. Z. Zhu, E. Hinton, and O. C. Zienkiewicz, *Mesh enrichment against mesh regeneration using quadrilateral elements*, Communications in Numerical Methods in Engineering **9**(7):547–554, 1993, DOI: 10.1002/cnm.1640090702 (cited on p. 93).

[ZZA98] T. Zhu, J.-D. Zhang, and S. N. Atluri, *A local boundary integral equation (LBIE) method in computational mechanics, and a meshless discretization approach*, Computational Mechanics **21**(3):223–235, 1998, DOI: 10.1007/s004660050297 (cited on p. 36).

[ZZ87] O. C. Zienkiewicz and J. Z. Zhu, *A simple error estimator and adaptive procedure for practical engineering analysis*, International Journal for Numerical Methods in Engineering **24**(2):337–357, 1987, DOI: 10.1002/nme.1620240206 (cited on p. 94).

Razširjeni povzetek v slovenščini

Uvod

Enega prvih člankov o numeričnem reševanju parcialnih diferencialnih enačb (PDE) je objavil Richardson leta 1911 [Ric11], kjer je uporabil sedaj klasično aproksimacijo končnih diferenc za opis izračuna napetosti v jezovih. Geometrijske omejitve regularnih mrež so bile prehude in v 40ih letih 20. stoletja je bila razvita metoda končnih elementov (FEM) [Cou43; Hre41], ki s pomočjo konstrukcije mreže v domeni rešuje robne probleme v šibki obliki. FEM je postala popularna in komercialno uspešna metoda zaradi njene robustnosti in geometrijske fleksibilnosti, ki ji je omogočila reševanje problemov iz prakse. Slaba stran FEM je, da za reševanje potrebuje mrežo, ki lahko nenatančno opiše geometrijo problema, poleg tega pa je kvalitetno mrežo tudi težko konstruirati, sploh v treh ali več dimenzijah. Druge popularne metode, kot npr. metoda končnih volumnov, prav tako pogosto potrebujejo mrežo. Konstrukcija kvalitetnih mrež je od nekdaj težak problem, sploh v treh dimenzijah, kjer so pogosto potrebni ročni popravki. Dodatne težave povzročajo želje po dinamičnem prilagajanju mreže problemu, ki ga rešujemo, za kar je potrebno imeti dobre algoritme za avtomatsko konstrukcijo.

Metode, ki uporabljajo mreže, imajo poleg problemov, povezanih s konstrukcijo mrež samih, tudi težave pri simulaciji materialov, ki se lahko gibljejo prosto ali kako drugače močno deformirajo. Poleg tega so lahko začetni podatki ali druge izmerjene vrednosti znane v razpršenih točkah brez vnaprej znane strukture, in metode, ki znajo take podatke obravnavati neposredno, imajo prednost. Zgoraj navedeni razlogi so bili povod za razvoj metod, ki uporabljajo reducirano mrežo, kot npr. metoda robnih elementov (BEM), ali pa je sploh ne. Slednje se imenujejo brezmrežne metode. Namesto povezovanja točk v mrežo, brezmrežne metode hranijo pri vsaki točki le seznam njenih sosedov. Kljub enostavnejši diskretizaciji pa točke še vedno ne smejo biti razporejene poljubno; ne smejo namreč biti preblizu skupaj, saj to lahko povzroči težave pri stabilnosti, niti ne sme obstajati večji del domene, kjer ni nobene točke, saj to lahko povzroči težave pri konvergenci. Eden izmed ciljev te disertacije je razvoj algoritma za konstrukcijo brezmrežnih diskretizacij s spremenljivo gostoto v neregularnih domenah v poljubni dimenziji.

Ena izmed zadnje čase bolj obetavnih brezmrežnih metod je metoda končnih diferenc, generiranih z radialnimi baznimi funkcijami (RBF-FD). Ta metoda je posplošitev klasične metode končnih diferenc na razpršene točke in izkazuje dobro konvergenčno in stabilnostno obnašanje. Slabost te metode, kot tudi veliko drugih brezmrežnih metod, je šibko matematično ozadje, saj globalne ocene napak ali teoretične stabilnostne analize niso poznane ali izdelane. Kljub temu se uporaba RBF-FD v praktičnih problemih povečuje, kar pospešuje tudi bolj temeljne raziskave o metodi. Drugi cilj te disertacije je raziskati potencial metode v področju izboljševanja diskretizacije in jo razviti v polno

avtomatsko adaptivno metodo, ki diskretizacijo prilagaja zahtevam problema.

Organizacija dela

Disertacija po vrsti predstavi potrebna orodja za razvoj avtomatske adaptivnosti.

- Poglavje 1 je posvečeno aproksimaciji funkcij na razpršenih podatkih, kar je predpogoj za razvoj in analizo vseh brezmrežnih metod, obravnavanih v tem delu.
- V poglavju 2 predstavimo RBF-FD metodo za reševanje PDE. Primerjamo jo z drugimi podobnimi metodami na nekaj ilustrativnih primerih, kar nam pomaga pri odločitvi za uporabo poliharmoničnih baznih funkcij, obogatenih z monomi.
- Poglavje 3 se ukvarja z brezmrežnimi diskretizacijami domen. Natančno definiramo pojem diskretizacije in zahteve, ki jih narekuje uporaba v RBF-FD. Za konstrukcijo diskretizacij razvijemo potrebne algoritme, ki generirajo točke v notranjosti in na robovih domen, in dokažemo več ugodnih lastnosti teh algoritmov. To nam bo služilo kot osnova za razvoj adaptivnosti.
- V poglavju 4 je predstavljen polno avtomatski adaptiven postopek za reševanje eliptičnih problemov. Njegovo obnašanje je analizirano na klasičnih problemih, z njim pa rešimo tudi več kontaktnih problemov iz linearne elastostatike.
- V poglavju 5 je podan kratek opis knjižnice Medusa in idej, ki so bile potrebne za njen razvoj.

Literatura

Najpomembnejši viri za glavne teme v delu so sledeči.

- Teme iz aproksimacije na razpršenih podatkih so povzete po knjigi [Wen04].
- RBF-FD in druge brezmrežne metode so opisane v knjigi [FF15a] in preglednih člankih [Ngu+08] ter [FF15c].
- Algoritmi za diskretizacijo domen, s katerimi primerjamo naše algoritme, so objavljeni v člankih [FF15b] in [SKF18].
- Teme iz linearne elastostatike so v glavnem povzete po [Sla12], z dodatnim materialom iz [Sad14].
- Pri kontaktnih problemih je bila v veliko pomoč knjiga [WD00].

Doprinos

Poglavji 3 in 4 predstavljata izvirno delo avtorja, Jureta Slaka. Knjižnica Medusa, predstavljena v poglavju 5, je rezultat dela Gregorja Kosca in Jureta Slaka, skupaj z drugimi sodelavci, ki so prispevali h kodi in sorodnim materialom, kot navedeno v [SK19c].

Določen del vsebine tega dela je bil objavljen v naslednjih znanstvenih člankih, ali pa je vsebovan v verzijah člankov pred objavo:

[SK19b] J. Slak in G. Kosec, *Refined meshless local strong form solution of Cauchy–Navier equation on an irregular domain*, Engineering Analysis with Boundary Elements **100**:3–13, mar. 2019, DOI: 10.1016/j.enganabound.2018.01.001 (cit. na str. 93, 118, 121).

[SK19c] J. Slak in G. Kosec, *Adaptive radial basis function-generated finite differences method for contact problems*, International Journal for Numerical Methods in Engineering **119**(7):661–686, avg. 2019, DOI: 10.1002/nme.6067 (cit. na str. 93, 94, 107).

[SK19d] J. Slak in G. Kosec, *On generation of node distributions for meshless PDE discretizations*, SIAM Journal on Scientific Computing **41**(5):A3202–A3229, okt. 2019, DOI: 10.1137/18M1231456 (cit. na str. 59, 61, 62, 85, 88).

[SK19e] J. Slak in G. Kosec, *Medusa: A C++ library for solving PDEs using strong form mesh-free methods*, arXiv:1912.13282, dec. 2019, URL: https://arxiv.org/abs/1912.13282 (cit. na str. 3, 158).

[DKS20] U. Duh, G. Kosec in J. Slak, *Fast variable density node generation on parametric surfaces with application to mesh-free methods*, arXiv:2005.08767, maj 2020, URL: https://arxiv.org/abs/2005.08767 (cit. na str. 70).

Poleg prej naštetih, sem soavtor tudi v naslednjih člankih, od katerih so nekateri še pred objavo.

[JSK19] M. Jančič, J. Slak in G. Kosec, *Analysis of high order dimension independent RBF-FD solution of Poisson's equation*, arXiv:1909.01126, sep. 2019, URL: https://arxiv.org/abs/1909.01126 (cit. na str. 92).

[Kos+19] G. Kosec, J. Slak, M. Depolli, R. Trobec, K. Pereira, S. Tomar, T. Jacquemin, S. P. A. Bordas in M. A. Wahab, *Weak and strong from meshless methods for linear elastic problem under fretting contact conditions*, Tribology International **138**:392–402, okt. 2019, DOI: 10.1016/j.triboint.2019.05.041 (cit. na str. 121, 123).

[Mak+19] M. Maksić, V. Djurica, A. Souvent, J. Slak, M. Depolli in G. Kosec, *Cooling of overhead power lines due to the natural convection*, International Journal of Electrical Power & Energy Systems **113**:333–343, dec. 2019, DOI: 10.1016/j.ijepes.2019.05.005.

[Moč+20] J. Močnik Berljavac, P. K. Mishra, J. Slak in G. Kosec, *RBF-FD analysis of 2D time-domain acoustic wave propagation in heterogeneous Earth's subsurface*, arXiv:2001.01597, jan. 2020, URL: https://arxiv.org/abs/2001.01597.

Poleg tega sem svoje delo predstavil v sledečih konferenčnih člankih na mednarodnih konferencah [SK16], [SK18b], [SK18a], [SK18d], [KS18c], [KS18b], [SK19b], [SSK19a], [KS19b], [KS19a], [SK20] in bil soavtor pri sledečih konferenčnih člankih: [KS18a], [MSK19], [SK18c], [SSK19b], [DKS19], [Duh+20], [JSK20].

Programska in strojna oprema

Vsi numerični izračuni so bili opravljeni na prenosnem računalniku s procesorjem Intel® Core™ i7-7700HQ CPU @ 2.80GHz in z 16 GB DDR4 SODIMM delovnega pomnilnika s hitrostjo 2400 MT/s.

Opisani algoritmi so bili implementirani v programskem jeziku C++ in prevedeni s prevajalnikom `g++` (`verzija Arch Linux 9.3.0-1`) na Linux jedru verzije `4.19.121-1-MANJARO`. Privzeto so bile uporabljene zastavice

```
-std=c++17 -fopenmp -O3 -Wall -Wextra -Wfloat-conversion
-Wno-deprecated-copy -Wno-maybe-uninitialized -pedantic -DNDEBUG
```

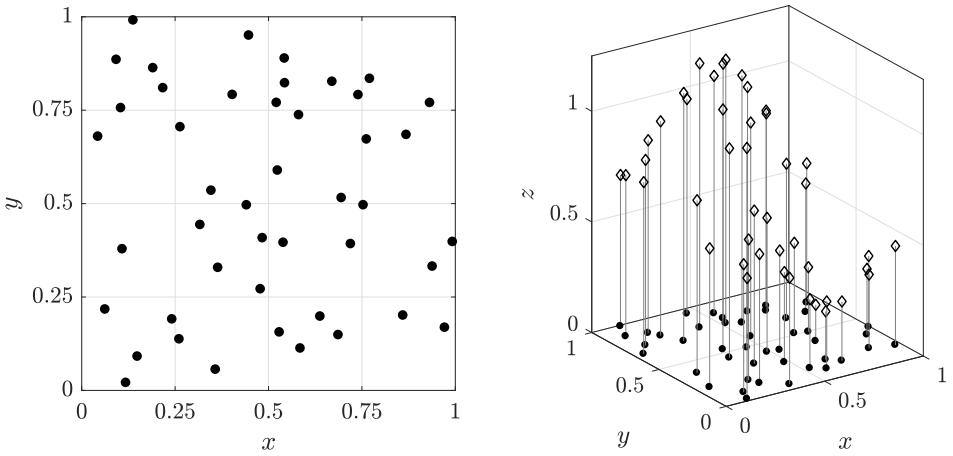
Pri izdelavi programov je bila uporabljena knjižnica `Eigen` skupaj z Intel®Math Kernel library (MKL) [Int18] in knjižnico Pardiso za reševanje sistemov linearnih enačb. Sistem `cmake` (v. 3.17.2) je bil uporabljen za organizirano grajenje izvršnih datotek. Vsa programska oprema, uporabljena v tem delu, vključujoč vse skripte za risanje slik in `.tex` izvorne datoteke za to besedilo, je na voljo na `https://gitlab.com/jureslak/phd`.

Izvorna koda uporablja knjižnico Medusa (v. `3e7409f64`), ki je na voljo na `https://gitlab.com/e62Lab/medusa`. Medusa vključuje knjižnico `Eigen` za delo z matrikami (verzija 3.3.7 z ročno dodano podporo za multi-indeksiranje) [GJ+10], knjižnico `nanoflann` za k-d drevesa [BR14], knjižnico `RapidXml` [Kal11] za delo z XML datotekami, paket `HDF5` za delo s HDF5 datotekami [Fol+11] in knjižnico `tinyformat` [Fos+11] za enostavno oblikovanje izpisov na standardni izhod.

Večina analize izračunov je bilo opravljene z Matlabom [Mat17]. To vključuje pripravo vseh slik, razen tistih, izrecno omenjenih spodaj. Pri izvozu visokokvalitetnih slik je bil v veliko pomoč dodaten paket `export_fig` [Alt20]. Risbe na slikah 1.3, 3.1, 3.4, 4.11, 4.16, 4.19, 4.23 in 4.24 so bile ustvarjene z grafičnim urejevalnikom Inkscape [Ink20].

S.1 Aproksimacija funkcij na razpršenih podatkih

V tem poglavju se posvetimo problemu aproksimacije funkcij na razpršenih podatkih. Za dane točke $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$ in funkcijske vrednosti $u_i = u(\boldsymbol{x}_i)$ želimo najti funkcijo $\hat{u}$, tako da velja $\hat{u}(\boldsymbol{x}_i) \approx u_i$. Za točke X predpostavljamo, da so *razpršene*, brez regularne strukture. Primer vhodnih podatkov za problem je prikazan na sliki S.1.



Slika S.1: Množica razpršenih točk $X = \{\boldsymbol{x}_i = (x_i, y_i)\}_{i=1}^{45}$ (levo) skupaj s pripadajočimi funkcijskimi vrednostmi u_i (desno). Vrednosti so dobljene iz funkcije $u_{\text{ex}}(x, y) = \cos(5x(1-y)) \exp(-6\left((x-0.5)^2 + (y-0.6)^2\right)) + 0.5$.

Za reševanje problema obstaja več načinov. Pomanjkanje regularne strukture lahko rešimo tako, da sami na točkah ustvarimo umetno strukturo, npr. triangulacijo, in uporabimo standardne metode za aproksimacijo v teh primerih. Druga možnost je, da na višje dimenzije posplošimo aproksimacijske sheme iz ene dimenzije, vendar tu velikokrat naletimo na težave. V primeru polinomske interpolacije nam na poti stoji Mairhuber-Curtisov izrek, ki pravi, da bo vedno obstajala singularna konfiguracija točk. Alternativo predstavljajo radialne bazne funkcije (RBF), ki ne potrebujejo nobene dodatne strukture in za katere obstajajo izreki, ki zagotavljajo rešljivost problema pri zelo šibkih predpostavkah. Cilj poglavja 1 je razložiti obnašanje RBF interpolacije. V ta namen najprej dokažemo Mairhuber-Curtisov izrek v razdelku 1.1 in si nato kot možno rešitev pogledamo osnove teorije Hilbertovih prostorov z reproducirajočim jedrom v razdelku 1.2, ki bodo tudi osnova za analizo obnašanja RBF. V razdelku 1.3 si ogledamo obstoječe metode za aproksimacijo razpršenih podatkov, in sicer Shepardovo interpolacijo, metodo premičnih najmanjših kvadratov in interpolacijo z RBF, ki je osnova tudi za RBF-FD metodo.

Definicija S.1.1 (Radialne bazne funkcije)**.** Naj bo dana funkcija $\phi\colon [0,\infty) \to \mathbb{R}$. Radialna bazna funkcija s centrom v $\boldsymbol{c} \in \mathbb{R}^d$ je funkcija

$$\phi_{\boldsymbol{c}}\colon \mathbb{R}^d \to \mathbb{R}, \quad \phi_{\boldsymbol{c}}(\boldsymbol{x}) = \phi(\|\boldsymbol{x} - \boldsymbol{c}\|). \tag{S.1.1}$$

Za dane točke $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$ je množica radialnih baznih funkcij na X generiranih s funkcijo ϕ enaka

$$\{\phi_{\boldsymbol{x}}; \boldsymbol{x} \in X\}. \tag{S.1.2}$$

Ime *bazne* funkcije ni vedno utemeljeno, saj elementi množice $\{\phi_{\boldsymbol{x}}; \boldsymbol{x} \in X\}$ niso vedno linearno neodvisni. Toda v razdelku 1.3.3 dokažemo, da za pogosto uporabljene funkcije ϕ imamo zagotovila o regularnosti.

Pogoste uporabljene funkcije ϕ so naslednje:

- *Gaussove:* $\phi(r) = \exp(-(\varepsilon r)^2)$,
- *Multikvadrične:* $\phi(r) = \sqrt{1 + (\varepsilon r)^2)}$,
- *Inverzne multikvadrične:* $\phi(r) = (1 + (\varepsilon r)^2)^{-\frac{1}{2}}$,
- *Poliharmonične:* $\phi(r) = \begin{cases} r^k, & k \text{ lih} \\ r^k \log r, & k \text{ sod} \end{cases}$.

Parametru ε pravimo *parameter oblike* in je pozitivno realno število, ki določa obliko grafa funkcije.

RBF interpolant za točke X je funkcija oblike

$$\hat{u}(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) \tag{S.1.3}$$

in interpolacijski pogoji $\hat{u}(\boldsymbol{x}_i) = f_i$ tvorijo sistem linearnih enačb

$$\begin{bmatrix} \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_n\|) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \tag{S.1.4}$$

ki ga lahko na kratko zapišemo kot $A_\phi \boldsymbol{\alpha} = \boldsymbol{f}$. V delu dokažemo, da je matrika A_ϕ za Gaussove in multikvadrične RBF pozitivno definitna (in da je posledično sistem rešljiv), čim so točke X paroma različne. Dokažemo tudi znan Shoenbergov izrek, ki povezuje popolnoma monotone funkcije s pozitivno definitnimi funkcijami.

Da bi dobili zagotovila o rešljivosti interpolacijskega problema v primeru inverznih multikvadričnih in poliharmoničnih RBF, je potrebno interpolant obogatiti z monomi. Tako ga iščemo v obliki

$$\hat{u}(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) + \sum_{\ell=1}^{q} \beta_\ell p_\ell(\boldsymbol{x}), \tag{S.1.5}$$

kjer so p_ℓ polinomi, ki tvorijo bazo prostora polinomov skupne stopnje manj ali enako m. Teh polinomov je $q = \binom{n+d}{d}$. V tem primeru interpolacijski pogoji $\hat{u}(\boldsymbol{x}_i) = f_i$ postavijo n enačb za $n + q$ neznank. Da bi zagotovili rešljivost, postavimo dodatnih q vezi oblike

$$\sum_{j=1}^{n} \alpha_j p_\ell(\boldsymbol{x}_j) = 0, \quad \forall \ell = 1, \ldots, q, \tag{S.1.6}$$

kar nam da končni sistem $(n + q)$ enačb z $(n + q)$ neznankami

$$\begin{bmatrix} A_\phi & P \\ P^\mathsf{T} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix}, \tag{S.1.7}$$

kjer je P matrika velikosti $n \times q$ z vrednostmi polinomov,

$$P = \begin{bmatrix} p_1(\boldsymbol{x}_1) & \cdots & p_q(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ p_1(\boldsymbol{x}_n) & \cdots & p_q(\boldsymbol{x}_n) \end{bmatrix}. \tag{S.1.8}$$

Dodane vezi ohranijo simetrijo matrike in so uporabne tudi pri dokazovanju rešljivosti sistema. V nadaljevanju poglavja 1 dokažemo rešljivost za inverzne multikvadrične in poliharmonične funkcije, ter dokažemo tudi Micchellijev izrek, ki povezuje pogojno popolnoma monotone funkcije s pogojno pozitivno semi-definitnimi funkcijami.

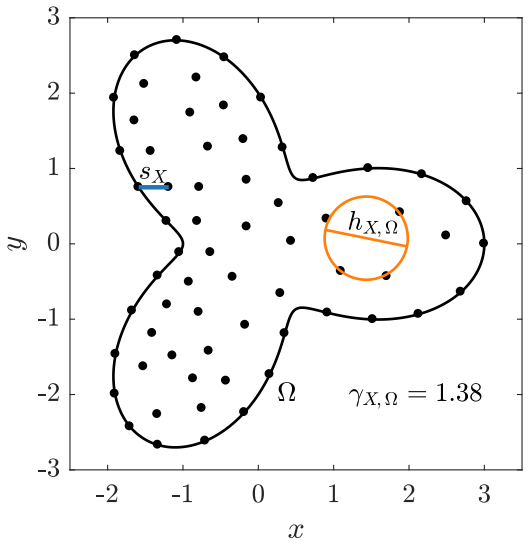
V poglavju 1.4 se posvetimo analizi stabilnosti in napak pri interpolaciji z radialnimi baznimi funkcijami. Pomembni količini, ki vplivata na stabilnost in konvergenco, sta ločitvena razdalja in krovna razdalja, ki ju izračunamo iz množice točk.

Definicija S.1.2 (Krovna razdalja)**.** Naj bo $\Omega \subseteq \mathbb{R}^d$ omejena in naj bo dana množica točk $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \Omega$. Krovna razdalja $h_{X,\Omega}$ množice X znotraj Ω je

$$h_{X,\Omega} = 2 \sup_{\boldsymbol{x} \in \Omega} \min_{j=1,\ldots,n} \|\boldsymbol{x} - \boldsymbol{x}_j\|. \tag{S.1.9}$$

Razdalja je dobro definirana, saj supremum obstaja zaradi omejenosti Ω. Krovna razdalja meri, kako dobro točke X pokrivajo množico Ω, saj ima največja prazna krogla, ki jo lahko najdemo znotraj Ω, premer manjši ali enak $h_{X,\Omega}$.

Komplementarna razdalja h krovni razdalji je ločitvena razdalja množice X.



Slika S.2: Krovna in ločitvena razdalja množice točk ter njuno razmerje $\gamma_{X,\Omega}$.

Definicija S.1.3 (Ločitvena razdalja)**.** Naj bo $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^d$ množica točk. Ločitvena razdalja s_X množice X je

$$s_X = \min_{1 \leq i < j \leq n} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|. \tag{S.1.10}$$

Ločitvena razdalja meri, kako dobro so točke v X med seboj ločene – vsak par točk je namreč vsaj s_X narazen. Krovna in ločitvena razdalja sta prikazani na sliki S.2.

Pri konstrukciji diskretizacij je pomembno gledati razmerje med krovno in ločitveno razdaljo. Kljub goščenju diskretizacije vseeno pogosto želimo, da razmerje med njima ostaja približno enako. Zaporedju množic točk, kjer je to razmerje za vse množice omejeno, bomo rekli zaporedje *kvazi-enakomernih* množic točk.

S.2 RBF-FD in podobne metode

Poglavje 2 opisuje RBF-FD in podobne numerične metode. V razdelku 2.1 je podan zgodovinski opis razvoja in umestitev RBF-FD metode v širše področje brezmrežnih metod. Nato v razdelku 2.2 izpeljemo nekaj aproksimacij, med drugim tudi RBF-FD.

Večina brezmrežnih metod, ki rešujejo PDE v močni obliki, aproksimira parcialne diferencialne operatorje z uteženo vsoto vrednosti funkcije v sosednjih točkah. Natančneje, vrednost operatorja $\mathcal{L}$, apliciranega na funkcijo u v točki $\boldsymbol{p}$, aproksimiramo kot

$$(\mathcal{L}u)(\boldsymbol{p}) \approx \sum_{i=i}^{n} w_i u(\boldsymbol{x}_i) = \boldsymbol{w}^{\mathsf{T}} \boldsymbol{u}, \tag{S.2.1}$$

kjer so $X = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ točke v okolici $\boldsymbol{p}$, ki jih imenujemo *soseščina*, w_i pa so še neznane uteži. Uteži w_i so odvisne le od operatorja $\mathcal{L}$ in točke $\boldsymbol{p}$, ne pa tudi od funkcije u; to odvisnost bomo eksplicitno pisali kot $\boldsymbol{w}^{\mathcal{L},X}$.

Splošna tehnika za izračun uteži w_i je, da jih izračunamo s pomočjo interpolacije. Konstruiramo funkcijo $\hat{u}$, ki v točkah $\boldsymbol{x}_i$ interpolira vrednosti u_i, in namesto točne vre-

dnosti $(\mathcal{L}u)(p)$ vzamemo $(\mathcal{L}\hat{u})(p)$. Denimo, da je interpolant $\hat{u}$ oblike

$$\hat{u}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i b_i(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x})^\mathsf{T} \boldsymbol{\alpha}, \tag{S.2.2}$$

za neznane koeficiente $\boldsymbol{\alpha}$ in neko bazo $= (b_1, \ldots, b_n)$, $b_i \colon \Omega \to \mathbb{R}$. Koeficiente $\boldsymbol{\alpha}$ lahko izračunamo s pomočjo linearnega sistema enačb, ki izhaja iz interpolacijskih pogojev

$$B\boldsymbol{\alpha} = \boldsymbol{u}, \tag{S.2.3}$$

kjer je $\boldsymbol{u}$ vektor funkcijskih vrednosti $\boldsymbol{u} = [u(\boldsymbol{x}_i)]_{i=1}^n$ in B interpolacijska matrika $B = [b_j(\boldsymbol{x}_i)]_{i,j=1}^n$.

Koeficiente $\boldsymbol{\alpha}$ lahko izrazimo kot $\boldsymbol{\alpha} = B^{-1}\boldsymbol{u}$ in zapišemo $\hat{u}$ kot

$$\hat{u}(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x})^\mathsf{T} \boldsymbol{\alpha} = \boldsymbol{b}(\boldsymbol{x})^\mathsf{T} B^{-1} \boldsymbol{u}. \tag{S.2.4}$$

S tem lahko izračunamo aproksimacijo za $\mathcal{L}$ v točki $\boldsymbol{p}$ kot

$$(\mathcal{L}u)(\boldsymbol{p}) \approx (\mathcal{L}\hat{u})(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T} B^{-1} \boldsymbol{u} =: \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T} \boldsymbol{u}. \tag{S.2.5}$$

Izraz za $(\mathcal{L}\hat{u})(\boldsymbol{p})$ lahko gledamo na dva načina:

$$(\mathcal{L}\hat{u})(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T} \boldsymbol{\alpha} = \overbrace{(\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T} \underbrace{B^{-1}\boldsymbol{u}}_{\boldsymbol{\alpha}}}^{\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}} = \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T} \boldsymbol{u}. \tag{S.2.6}$$

Prvi izraz $(\mathcal{L}\hat{u})(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T}\boldsymbol{\alpha}$ nam omogoča, da izračunamo koeficiente $\boldsymbol{\alpha}$ iz znanih funkcijskih vrednosti $\boldsymbol{u}$, in tako dobimo vrednosti $\hat{u}$ za poljubno točko $\boldsymbol{p}$ ali poljuben operator $\mathcal{L}$. Drugi izraz $(\mathcal{L}\hat{u})(\boldsymbol{p}) = \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}\boldsymbol{u}$ pa nam omogoča, da izračunamo $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}$ za določeno točko $\boldsymbol{p}$ in operator $\mathcal{L}$, ampak neodvisno od funkcijskih vrednosti $\boldsymbol{u}$, ki so lahko tudi neznane. Tako lahko enako vrednosti $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T}$ uporabimo za aproksimacijo $\mathcal{L}|_p$ za različne funkcije u. Za izračun $\boldsymbol{w}$ lahko uporabimo izraz $\boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p})^\mathsf{T} = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p})^\mathsf{T} B^{-1}$, vendar je bolje reševati ekvivalenten sistem

$$B^\mathsf{T} \boldsymbol{w}^{\mathcal{L},X}(\boldsymbol{p}) = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}). \tag{S.2.7}$$

Na izračun uteži $\boldsymbol{w}$ lahko pogledamo tudi z drugega zornega kota, z vidika metode nedoločenih koeficientov. Ponovno iščemo aproksimacijo v obliki

$$(\mathcal{L}u)(\boldsymbol{p}) \approx \sum_{i=i}^{n} w_i u(\boldsymbol{x}_i) = \boldsymbol{w}^\mathsf{T} \boldsymbol{u}, \tag{S.2.8}$$

toda namesto, da u zamenjamo z $\hat{u}$, le zahtevajmo, da naj bo aproksimacija točna za neko bazo $\{b_i\}$. Tako dobimo sistem linearnih enačb

$$(\mathcal{L}b_j)(\boldsymbol{p}) = \sum_{i=i}^{n} w_i b_j(\boldsymbol{x}_i), \tag{S.2.9}$$

ki ga lahko zapišemo kot

$$B^\mathsf{T} \boldsymbol{w} = (\mathcal{L}\boldsymbol{b})(\boldsymbol{p}), \tag{S.2.10}$$

kar je enak sistem kot prej.

Osnovna RBF-FD metoda je enaka zgoraj opisani, le da za bazo $\{b_i\}$ uporabimo radialne bazne funkcije, $\{\phi_{\boldsymbol{x}}; \boldsymbol{x} \in X\}$. V tem primeru je matrika linearnega sistema enačb enaka matriki, uporabljeni pri interpolaciji, in zanjo veljajo že dokazane lastnosti. Podobno kot pri interpolaciji lahko tudi RBF-FD metodo obogatimo z monomi, da zagotovimo rešljivost sistema pod ustreznimi predpostavkami in konsistentnost aproksimacije do predpisanega reda monomov. V preostanku razdelka 2.2 pokažemo nekaj lastnosti izpeljanih aproksimacij in analiziramo njihovo obnašanje na preprostih primerih, ki jih je možno izračunati v zaprti obliki.

V razdelku 2.3 opišemo, kako izračunane aproksimacije uporabimo za numerično reševanje problemov. Vzemimo standarden robni problem

$$\mathcal{L}u = f \qquad \text{in } \Omega, \tag{S.2.11}$$

$$u = g_d \qquad \text{on } \Gamma_d, \tag{S.2.12}$$

$$\frac{\partial u}{\partial \vec{n}} = g_n \qquad \text{on } \Gamma_n, \tag{S.2.13}$$

kjer Γ_d predstavlja rob, kjer veljajo Dirichletovi robni pogoji, Γ_n rob z Neumannovimi robnimi pogoji, f, g_d in g_n pa so poznane funkcije. Domeno diskretiziramo z N točkami $X = \{\boldsymbol{x}_1, \dots, x_N\}$, od katerih jih je N_i v notranjosti, N_n na robu Γ_n, N_d pa na robu Γ_d. Točke tudi oštevilčimo in vsaki točki x_i pripišemo njeno soseščino S_i, sestavljeno iz n_i sosednjih točk. Indekse točk v soseščini točke x_i označimo z $\{I_{i,j}\}_{j=1}^{n_i}$.

Neznano funkcijo u bomo aproksimirali v točkah x_i in neznane vrednosti $u_i := u(\boldsymbol{x}_i)$ obravnavali kot neznanke. Enačbe iz robnega problema lahko aproksimiramo z enačbami

$$\sum_{j=1}^{n_i} (\boldsymbol{w}^{\mathcal{L},S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}} = f(\boldsymbol{x}_i) \qquad \text{za } \boldsymbol{x}_i \text{ v notranjosti,} \tag{S.2.14}$$

$$u_i = g_d(\boldsymbol{x}_i) \qquad \text{za } \boldsymbol{x}_i \text{ na } \Gamma_d, \tag{S.2.15}$$

$$\sum_{\ell=1}^{d} (\vec{n})_\ell \sum_{j=1}^{n_i} (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j u_{I_{i,j}} = g_n(\boldsymbol{x}_i) \qquad \text{za } \boldsymbol{x}_i \text{ na } \Gamma_n. \tag{S.2.16}$$

Enačbe zložimo v sistem N linearnih enačb z N neznankami, ki ga lahko zapišemo kot $Mu = r$, kjer i-ta vrstica sistema vsebuje enačbo, ki velja v točki $\boldsymbol{x}_i$. Matrika M in desna stran r imata elemente

$$M_{i,I_{i,j}} = (\boldsymbol{w}_{\mathcal{L},\boldsymbol{x}_i})_j, \qquad r_i = f(\boldsymbol{x}_i), \qquad \text{za } \boldsymbol{x}_i \text{ v notranjosti,} \tag{S.2.17}$$

$$M_{i,i} = 1, \qquad r_i = g_d(\boldsymbol{x}_i), \qquad \text{za } \boldsymbol{x}_i \text{ na } \Gamma_d, \tag{S.2.18}$$

$$M_{i,I_{i,j}} = \sum_{\ell=1}^{d} (\vec{n})_\ell (\boldsymbol{w}^{\partial_\ell,S_i}(\boldsymbol{x}_i))_j, \qquad r_i = g_n(\boldsymbol{x}_i), \qquad \text{za } \boldsymbol{x}_i \text{ na } \Gamma_n, \tag{S.2.19}$$

kjer indeks j teče od 1 do n_i. Matrika M je razpršena z največ $\sum_{i=1}^{N} n_i$ neničelnimi elementi. Rešitev sistema enačb $Mu = r$ nam da numerično aproksimacijo za vrednosti funkcije u.

S.3 Diskretizacija domene

V poglavju 3 definiramo diskretizacijo domene in razvijemo algoritme za generiranje točk v notranjosti in na robu domene. Diskretizacija domene vključuje postavitev točk v notranjosti in na robu domene in določanje soseščin vsem tem točkam, točkam na robu pa je dodatno treba določiti še zunanje enotske normale. V razdelku 3.1 se seznanimo z obstoječimi raziskavami na področju konstrukcije diskretizacij in o pričakovanjih, ki jih o algoritmih imamo. Te med drugim vključujejo generiranje diskretizacij s poljubno gostoto točk, sledenje predpisani medsebojni razdalji, učinkovitost, neodvisnost od dimenzije in orientacije domene, kompatibilnost med diskretizacijami roba in notranjosti in dobro delovanje na neregularnih domenah.

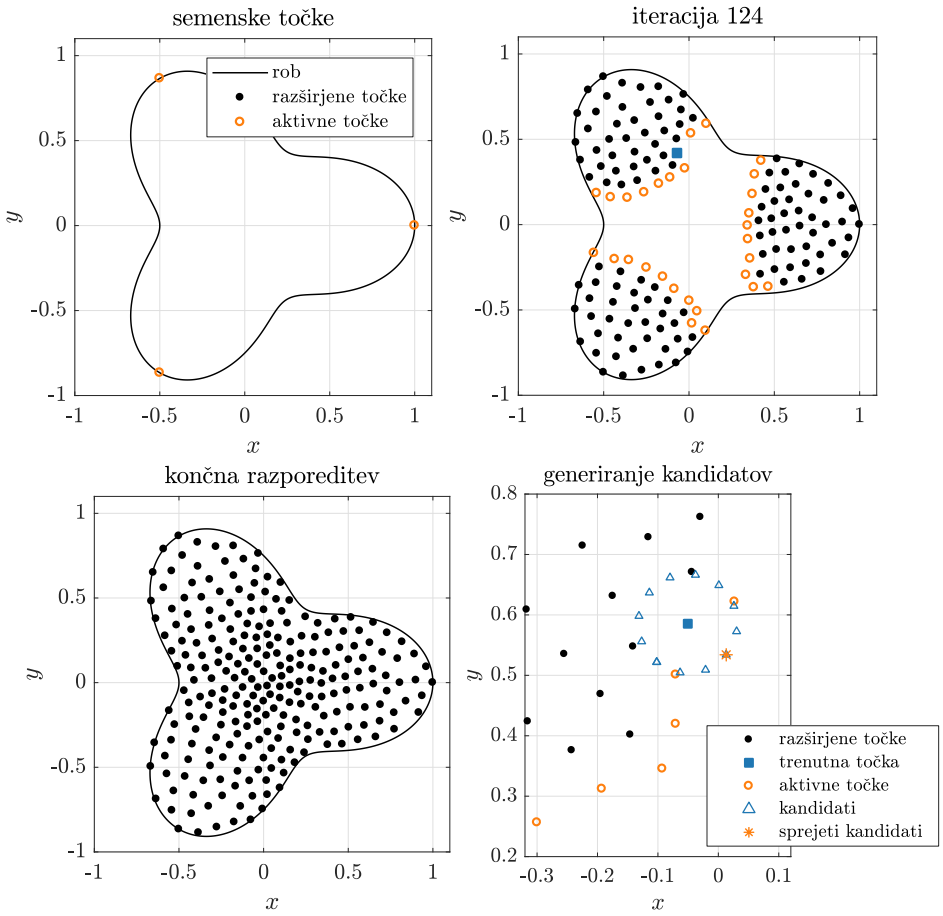
Razviti algoritem za generiranje točk v notranjosti kot vhod prejme domeno Ω, podano z njeno karakteristično funkcijo $\chi_\Omega \colon \Omega \to \{0, 1\}$, funkcijo razmika $h \colon \Omega \to (0, \infty)$ in seznam začetnih *semenskih* točk X. Te točke vstavimo v vrsto in jih uporabimo za generiranje novih točk. Potrebovali bomo tudi iskalno strukturo S, ki podpira vstavljanje točk in iskanje najbližjega soseda. Točkam, ki so v vrsti bomo rekli *aktivne*, točkam, ki smo jih že vzeli iz vrste, pa bomo rekli *razširjene*. Algoritem obravnava točke podobno kot iskanje v širino. V i-ti iteraciji algoritma iz vrste odstranimo točko $\boldsymbol{p}_i$ in jo dodamo na seznam sprejetih točk. Okoli $\boldsymbol{p}_i$ generiramo množico *kandidatov* $\{\boldsymbol{c}_{i,j}\}_j$, tako da so kandidati razporejeni približno enakomerno na sferi z radijem $h(\boldsymbol{p}_i)$ in središčem v $\boldsymbol{p}_i$. Kandidate obravnavamo po vrsti. Če kandidat $\boldsymbol{c}_{i,j}$ leži izven domene Ω, ga zavržemo. Če leži znotraj Ω, s pomočjo S poiščemo njegovega najbližjega soseda $\boldsymbol{n}_{i,j}$. Če je razdalja med $\boldsymbol{n}_{i,j}$ in $\boldsymbol{c}_{i,j}$ manjša od $h(\boldsymbol{p}_i)$, kandidata zavrnemo, sicer pa ga sprejmemo. Sprejeti kandidat je vstavljen v vrsto in v strukturo S. Algoritem se konča, ko je vrsta prazna. Potek algoritma je prikazan na sliki S.3.

Algoritem je bolj podrobno predstavljen in zapisan tudi v psevdokodi v razdelku 3.2, kjer pokažemo, da je časovna zahtevnost generiranja N točk v primeru uporabe k-d dreves za strukturo S enaka $O(N \log N)$. Poleg tega dokažemo tudi izrek o spoštovanju minimalne predpisane razdalje med točkami.

Algoritem za generiranje točk na robovih domen, podanih z regularno parametrizacijo, deluje podobno. V tem primeru izvajamo postavljanje točk v parametričnem prostoru, razdalje pa merimo v ciljnem prostoru. Pri tem za ustrezno merjenje razdalj v parametričnem prostoru aproksimiramo povlek lokalne metrike ciljnega prostora v parametrični prostor. Napako, storjeno pri tej aproksimaciji, tudi ocenimo in oceno uporabimo pri dokazovanju izreka o spoštovanju minimalne razdalje. Algoritem je bolj podrobno opisan v razdelku 3.3.

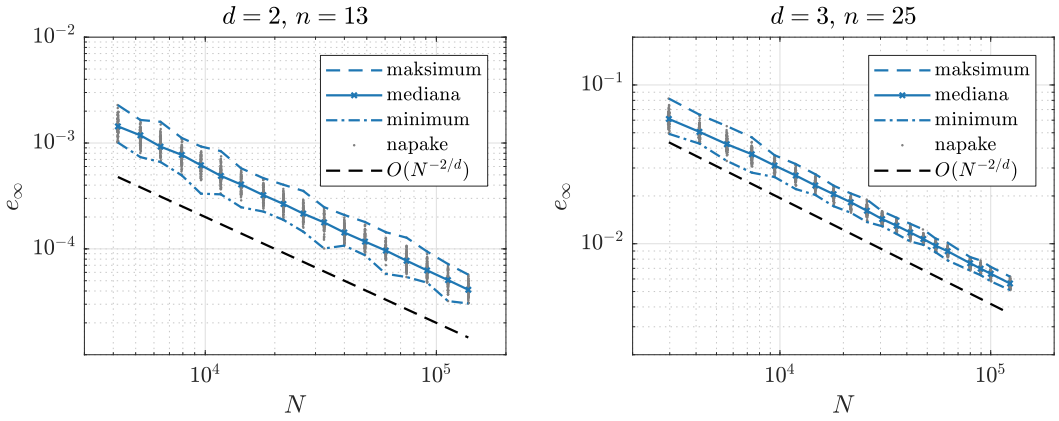
Oba algoritma sta podrobno analizirana z vidika kvalitete točk, kvazi-enakomernosti, hitrosti izvajanja, možnosti generiranja diskretizacij s spremenljivo gostoto, prostih parametrov in možnosti za vzporedno izvajanje v razdelku 3.4. Algoritma sta, ko je možno, tudi primerjana z dvema obstoječima algoritmoma za generiranje točk in končni rezultat analiz je, da je kombinacija algoritmov za generiranje točk na robu in v notranjosti primerna za brezmrežne diskretizacije.

Primer, ki to dobro ilustrira, je sledeč. Na diskretizacijah, dobljenih z razvitima algoritmoma, s pomočjo RBF-FD metode rešimo Poissonov robni problem in opazujemo obnašanje napake pri čedalje gostejših diskretizacijah. Postopek ponovimo stokrat, da dobimo večji vzorec, in napake narišemo na graf, prikazan na sliki S.4. Metoda se obnaša



Slika S.3: Delovanje algoritma za generiranje točk na domeni, definirani s polarno krivuljo $r(\vartheta) = \frac{1}{4}(3 + \cos(3\vartheta))$. Semenske točke ležijo na robu pri vrednostih ϑ enakih $0, 2\pi/3$ in $4\pi/3$. Funkcija razmika je enaka $h(\boldsymbol{x}) = 0.05(1 + \|\boldsymbol{x}\|_1)$.

pričakovano, kar nam daje dobro podlago za adaptivnost.



Slika S.4: Občutljivost RBF-FD metode na postavitev točk, dobljenih z razvitimi algoritmi. Sive pike prikazujejo numerične napake, dobljene pri določenem številu točk. Črte prikazujejo povprečne, najmanjše in največje napake, ter pričakovan red konvergence.

S.4 Adaptivnost

V poglavju 4 razvijemo adaptivno RBF-FD metodo. Najprej si ogledamo možne vrste adaptivnosti v razdelku 4.1 in obstoječe raziskave na področju indikatorjev napake v razdelku 4.2. Na podlagi pregleda literature se odločimo za h-adaptivnost s ponovnim generiranjem diskretizacije in za indikator napake na osnovi velikosti gradienta kot za najbolj smotrni možnosti.

Razviti h-adaptivni rešitveni postopek se močno zanaša na algoritme, opisane v poglavju 3, ki ponujajo možnost konstrukcije diskretizacije s poljubno gostoto, predpisano s pomočjo funkcije razmika h. To pomeni, da lahko namesto diskretizacije prilagajamo funkcijo h, in jo uporabimo za konstrukcijo nove, bolje prilagojene diskretizacije.

Adaptivni rešitveni postopek je iterativen. V prvi iteraciji diskretiziramo domeno z začetno funkcijo razmika $h^{(0)}$, dobimo diskretizacijo $\mathcal{D}^{(0)}$ z diskretizacijskimi točkami $X^{(0)} = \{\boldsymbol{x}_1^{(0)}, \ldots, \boldsymbol{x}_{N^{(0)}}^{(0)}\}$, in s pomočjo RBF-FD metode tudi rešitev $\boldsymbol{u}^{(0)}$. S pomočjo indikatorja napake nato izračunamo vrednosti napak v posameznih točkah, tako da vrednost $\hat{e}_i^{(0)}$ predstavlja oceno napake v točki $\boldsymbol{x}_i^{(0)}$. Če je norma vektorja vseh napak $\hat{e}^{(0)}$ manjša od neke vnaprej izbrane tolerance τ, pravimo, da je rešitev dovolj kvalitetna in nehamo. Sicer glede na ocenjene vrednosti napak $\hat{e}^{(0)}$ prilagodimo $h^{(0)}$ v novo funkcijo razmika $h^{(1)}$ in postopek ponovimo. V praksi je smiselno dodati tudi zgornjo mejo na število iteracij J, tako da je postopek vedno končen. V primeru, da v J iteracijah ne najdemo dovolj kvalitetne rešitve, to tudi sporočimo.

Prilagajanje funkcije $h^{(j)}$ v $h^{(j+1)}$ je odvisno od več parametrov, ki jih nastavimo na začetku postopka:

- $h_\mathrm{r}, h_\mathrm{d} \colon \Omega \to (0, \infty)$: spodnja in zgornja meja za h,
- $\varepsilon_\mathrm{r}, \varepsilon_\mathrm{d}$: pozitivni realni števili, ki predstavljata praga za goščenje in redčenje,
- $\alpha_\mathrm{r}, \alpha_\mathrm{d}$: realni števili, večji od 1, ki predstavljata mejo za faktorje spremembe gostote.

Količine z indeksom ‘r’ so povezane z goščenjem (angl. *refinement*), količine z indeksom ‘d’ pa z redčenjem (angl. *derefinement*).

Funkcijo razmika prilagodimo lokalno, tako da pri vsaki točki $\boldsymbol{x}_i$ prilagodimo lokalno razdaljo $h_i^{(j)} := h^{(j)}(\boldsymbol{x}_i)$ za nek faktor $f_i^{(j)}$. Tako definiramo nove vrednosti

$$h_i^{(j+1)} := \max\{\min\{h_i^{(j)}/f_i^{(j)}, h_\mathrm{d}(\boldsymbol{x}_i)\}, h_\mathrm{r}(\boldsymbol{x}_i)\}, \tag{S.4.1}$$

kjer je faktor spremembe gostote $f_i^{(j)}$ definiran kot

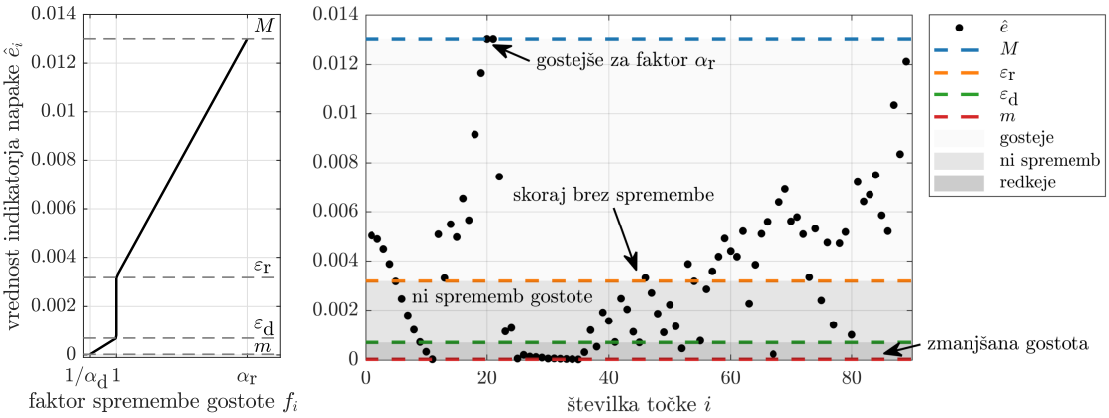
$$f_i^{(j)} = \begin{cases} 1 + \frac{\varepsilon_\mathrm{d} - \hat{e}_i^{(j)}}{\varepsilon_\mathrm{d} - m^{(j)}}\left(\frac{1}{\alpha_\mathrm{d}} - 1\right), & \hat{e}_i^{(j)} \leq \varepsilon_\mathrm{d}, \quad \text{povečana gostota} \\ 1, & \varepsilon_\mathrm{d} < \hat{e}_i^{(j)} < \varepsilon_\mathrm{r}, \quad \text{brez spremembe gostote} \\ 1 + \frac{\hat{e}_i^{(j)} - \varepsilon_\mathrm{r}}{M^{(j)} - \varepsilon_\mathrm{r}}(\alpha_\mathrm{r} - 1), & \hat{e}_i^{(j)} \geq \varepsilon_\mathrm{r}, \quad \text{zmanjšana gostota} \end{cases}. \tag{S.4.2}$$

Pri tem $M^{(j)}$ in $m^{(j)}$ predstavljata ekstrema ocen napak $\hat{e}^{(j)}$:

$$M^{(j)} = \max_{\boldsymbol{x}_i^{(j)} \in X^{(j)}} \hat{e}_i^{(j)}, \quad m^{(j)} = \min_{\boldsymbol{x}_i^{(j)} \in X^{(j)}} \hat{e}_i^{(j)}. \tag{S.4.3}$$

Faktorji $f_i^{(j)}$ so omejeni z $\frac{1}{\alpha_\mathrm{d}} \leq f_i \leq \alpha_\mathrm{r}$, saj velja $\frac{\hat{e}_i^{(j)}-\varepsilon_\mathrm{r}}{M^{(j)}-\varepsilon_\mathrm{r}} \in [0,1]$ in $\alpha_\mathrm{r} \geq 1$. Če je ocenjena napaka točke ravno na pragu goščenja, tj. $\hat{e}_i^{(j)} = \varepsilon_\mathrm{r}$, potem bo faktor $f_i^{(j)}$ enak 1 in se gostota ne bo spremenila, kar je kompatibilno s primerom, ko je $\hat{e}_i^{(j)} < \varepsilon_\mathrm{r}$. Če pa je ocenjena napaka točke najvišja možna, tj. $\hat{e}_i^{(j)} = M^{(j)}$, potem bo faktor spremembe gostote največji možen, α_r. Globalna meja h_r na vrednost $h^{(j+1)}$ pa nam zagotavlja, da diskretizacija ne more postati pregosta. Simetrične lastnosti veljajo tudi v primeru redčenja. Če nastavimo $\alpha_\mathrm{r} = 1$ ali $\alpha_\mathrm{d} = 1$, potem onemogočimo goščenje in redčenje.

Primer obnašanja postopka prilagajanja funkcije $h^{(j)}$ je prikazan na sliki S.5.



Slika S.5: Konstrukcija nove funkcije razmika.

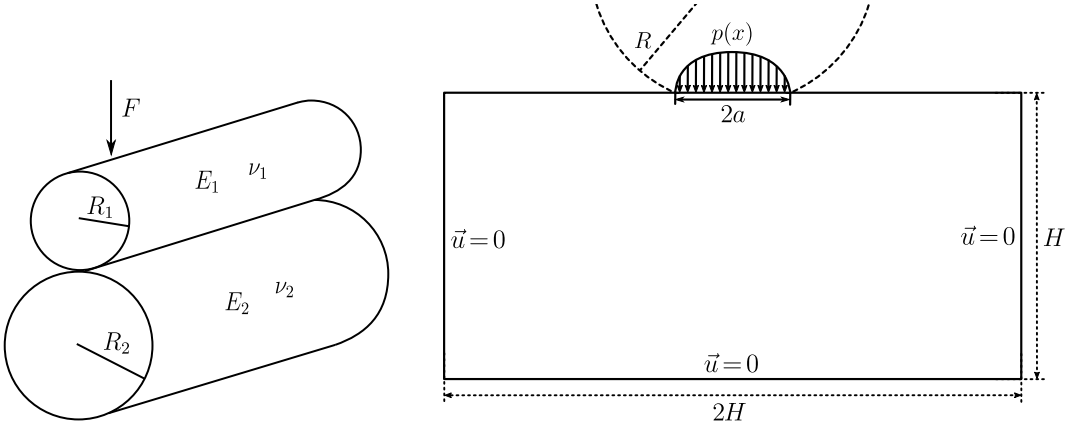
Do sedaj smo definirali le nove vrednosti $h_i^{(j+1)}$, ne pa še cele funkcije $h^{(j+1)}$. Konstrukcija $h^{(j+1)}$ je le vaja iz interpolacije razpršenih podatkov, za kar lahko uporabimo enostavno Sheppardovo interpolacijo na nekaj najbližjih sosedih.

V razdelku 4.4 preizkusimo opisani rešitveni postopek na klasičnih 2D in 3D Poissonovih robnih problemih, kot sta L-domena in Ficherov vogal, in jih tudi uspešno rešimo. Na teh problemih tudi analiziramo obnašanje prostih parametrov postopka. Nato se s postopkom v razdelku 4.5 lotimo kontaktnih problemov iz linearne elastostatike, kot so disk pod napetostjo, točkovni kontakt v 3D, Hertzov kontakt in kontaktni problem, ki izvira iz mehanike utrujanja materialov. Izmed teh problemov zmožnosti adaptivnosti najlepše demonstrira Hertzov kontakt, ki rešuje problem kontakta polprostora z valjem, kot prikazano na sliki S.6.

Numerično problem reduciramo na dve dimenziji in polprostor omejimo na pravokotnik $\Omega = (-H, H) \times (-H, 0)$ za dovolj velik H. Problem, ki ga rešujemo, je

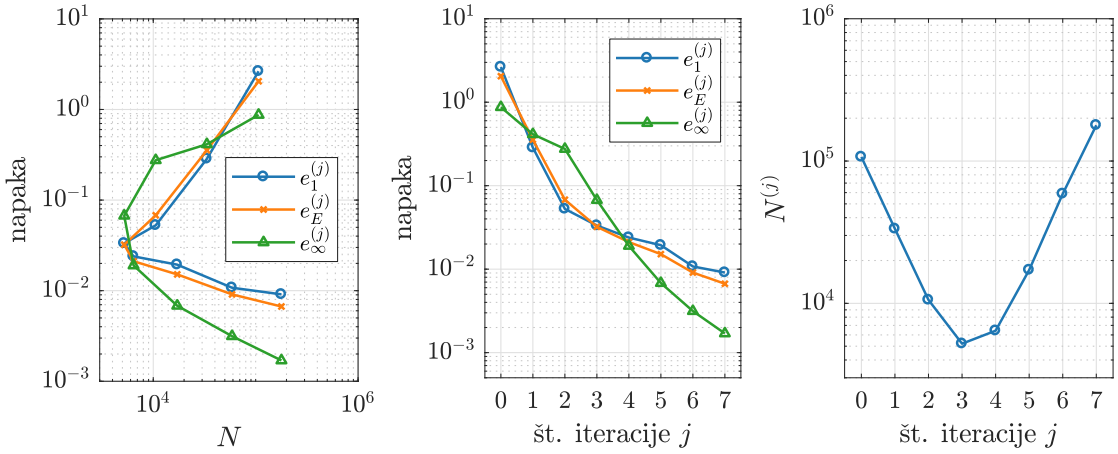
$$\begin{aligned} (\lambda+\mu)\nabla(\nabla\cdot\vec{u}) + \mu\nabla^2\vec{u} &= 0 \ \text{ in } \Omega, \\ \vec{u}(-H,y) = \vec{u}(H,y) = \vec{u}(x,-H) &= 0, \\ \sigma(x,0)\vec{n} = \vec{n}&\begin{cases} p(x), & |x| \leq a \\ 0, & |x| > a \end{cases}, \end{aligned} \tag{S.4.4}$$

kjer sta λ in μ Laméjeva parametra, p je napetost na površini, a pa je širina kontakta. Vrednost H je bila približno $H \approx 1923a$. Domeno na začetku napolnimo s točkami s funkcijo razmika $h^{(0)} \approx 7.7a$, kar pomeni, da imamo približno eno točko na 7.7 širin



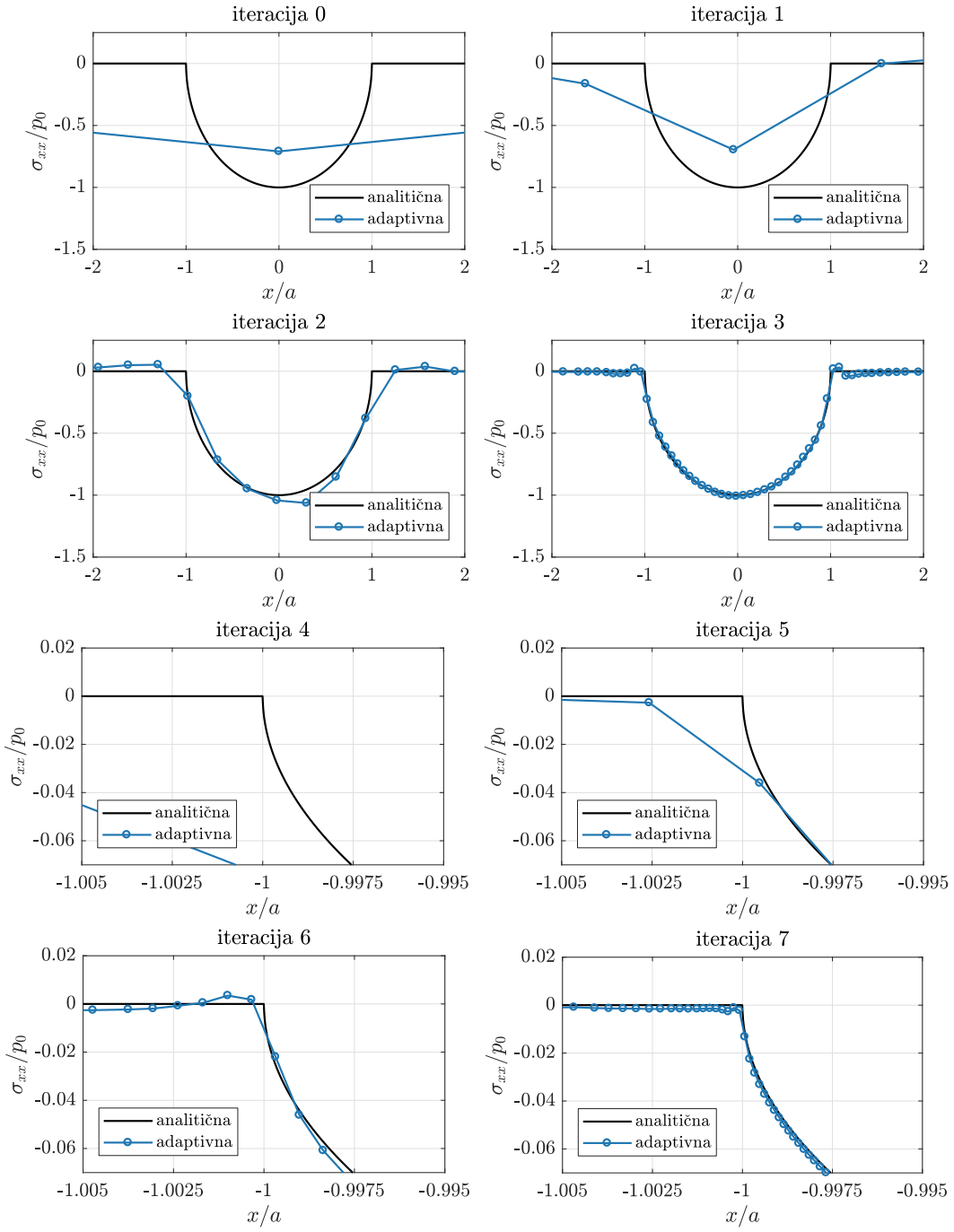
Slika S.6: Hertzov kontaktni problem med dvema valjema (levo) in domena za adaptivno numerično simulacijo kontakta med valjem in polprostorom.

kontakta. Nato zaženemo adaptiven rešitveni postopek, s parametri, opisanimi v razdelku 4.5.4. Tekom postopka se diskretizacija pod kontaktom gosti, drugod po domeni pa se redči. Napake in število točk tekom iteracije so prikazane na sliki S.7.



Slika S.7: Napake in število točk med adaptivnim reševanjem Hertzevega kontaktnega problema.

Na začetku se število točk zmanjšuje, saj po večini domene diskretizacijo redčimo in gostimo le pod kontaktom, sčasoma pa se to redčenje ustavi, pod kontaktom pa se goščenje povečuje, in število točk ponovno naraste. Na koncu je razmerje med najmanjšo in največjo medsebojno razdaljo enako približno $3 \cdot 10^6$ in 95% vseh točk je vsebovanih v pravokotniku $[-3a, 3a] \times [-3a, 0]$, ki predstavlja le 0.000027% površine domene. Dobro ilustracijo o poteku iteracije dajo profili napetosti, prikazani na sliki S.8.



Slika S.8: Profili normalne napetosti v okolici kontakta med adaptivnim reševanjem Hertzovega kontaktnega problema.

S.5 Implementacija

V poglavju 5 je predstavljena knjižnica Medusa za reševanje parcialnih diferencialnih enačb v močni obliki z brezmrežnimi metodami. V razdelku 5.1 je predstavljen minimalni delujoči primer reševanja problema, ki ga lahko rešimo v eni, dveh ali treh dimenzijah s popolnoma enako programsko kodo. Nato v razdelku 5.2 sledi predstavitev glavnih modulov knjižnice s poudarkom na razlogih za tako obliko programskega vmesnika, ki omogoča modularnost, berljivost in učinkovitost kode. Na koncu v razdelku 5.3 pokažemo še časovne meritve, kjer primerjamo implementacijo RBF-FD metode v Medusi z metodo končnih elementov implementirano v FreeFem++.

Zaključki in nadaljnje delo

Glavni rezultat naloge je razvoj avtomatskega postopka za reševanje eliptičnih robnih problemov s pomočjo RBF-FD metode. Za dosego tega smo v delu najprej razvili algoritem za konstrukcijo brezmrežnih diskretizacij, ki omogočajo generiranje točk v notranjosti in na robu domene, ko so robovi podani kot parametrizirane ploskve. Algoritem je neodvisen od dimenzije prostora in dokazano spoštuje minimalno razdaljo med točkami. Generira lokalno regularne množice točk s spremenljivo gostoto in za generiranje N točk potrebuje $O(N \log N)$ časa. Za aproksimacijo linearnih parcialnih diferencialnih operatorjev smo uporabili RBF-FD aproksimacijo, s poliharmoničnimi baznimi funkcijami $\phi(r) = r^3$, obogatenimi z monomi, ter se tako izognili klasičnim težavam RBF-FD metode. Ta aproksimacija je bila združena še z ustrezno shemo za goščenje diskretizacij in indikatorjem napake v končni postopek za polno adaptivnost. Vsi deli razvitega postopka so brezmrežni in posplošljivi na poljubno število dimenzij. Uspešno delovanje postopka je bilo prikazano na klasičnih 2D in 3D Poissonovih robnih problemih ter na 2D in 3D kontaktnih problemih iz linearne elastostatike. Demonstrirali smo tako goščenje kot redčenje in uspešno rešili tudi problem z razmerjem med najredkejšim in najgostejšim delom diskretizacije, večjim od $3 \cdot 10^6$.

Programska koda, napisana med reševanjem zgornjih problemov, je bila razvita z mislijo na ponovno uporabo. Glavni deli postopka so bili izolirani in ločeno objavljeni kot knjižnica za delo z brezmrežnimi metodami. Knjižnica je napisana v C++ in omogoča hitro testiranje različnih idej, povezanih z brezmrežnimi metodami, ki pa jih, če je potrebno, lahko pogosto enostavno posplošimo na tri ali več dimenzij.

Za nadaljnje delo je odprtih več smeri. Razviti je potrebno boljše in robustnejše indikatorje napak, ter boljše strategije goščenja. To zahteva tudi posplošitev algoritmov, da omogočajo dinamično dodajanje ali odvzemanje točk v času, sorazmernim le s številom dodanih in odvzetih točk, ne da bi spremenili preostanek domene. Nadaljnje izboljšave algoritmov za generiranje točk vključujejo posplošitev na ploskve, opisane z zlepki (npr. NURBS) in analizo obnašanja na ploskvah iz več krp.

Potrebne so tudi nadaljnje raziskave na področju brezmrežnih aproksimacij. Primerjava RBF-FD s konkurenčnimi aproksimacijami, npr. aproksimacijami, dobljenimi z uteženimi najmanjšimi kvadrati, bi omogočila jasnejši pogled na množico vseh metod. Za razvoj p-adaptivnosti bi bilo dobro razumeti tudi medsebojni vpliv izbire sosedov na red obogatitve z monomi. Odgovor na to vprašanje bi bil dober korak na poti k razvoju avtomatske hp-adaptivnosti za brezmrežne metode.