Implementation of strong form meshfree methods for solving PDEs

open source meshless project
https://gitlab.com/e62Lab/medusa
http://e6.ijs.si/medusa/

Gregor Kosec
Jožef Stefan Institute, Slovenia

Scope

Computational modelling

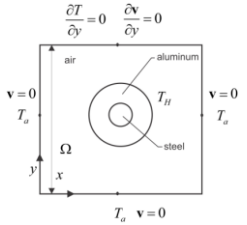
Discretization of domain, approximation of differential operator, approximation of partial differential equations.

$$\rho^{\alpha}\frac{\partial \mathbf{v}}{\partial t}+\rho^{\alpha}\nabla\cdot(\mathbf{vv})=-\nabla P+\nabla\cdot\left(\mu^{\alpha}\nabla\mathbf{v}\right)+\mathbf{b}$$

$$\nabla\cdot\mathbf{v}=0$$

$$\rho^{\alpha}c_{p}^{\alpha}\frac{\partial T}{\partial t}+\rho^{\alpha}c_{p}^{\alpha}\nabla\cdot(T\mathbf{v})=\nabla\cdot\left(\lambda^{\alpha}\nabla T\right)$$

$$\mathbf{b}=\rho_{ref}\left[1-\beta_{T}(T-T_{ref})\right]\mathbf{g}$$



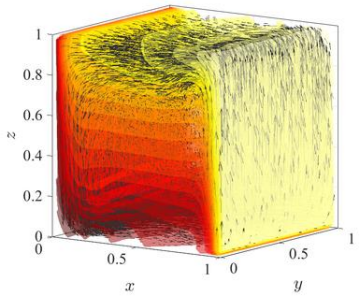
Implementation

Implementation of the solution procedure in open source generic C++ library.

open source meshless project
Medusa: Coordinate Free Mehless Method implementation (MM)
https://gitlab.com/e62Lab/medusa
http://e6.ijs.si/medusa/

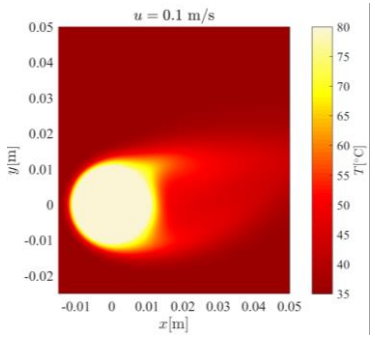
Examples

Simple test cases, linear elasticity, thermo-fluid flow, EM scattering, solidification of a binary alloy, simulation of semiconductors.



Application in projects

Simulation of fretting fatigue, simulation of overhead power line cooling.



JOŽEF STEFAN INSTITUTE

Basic info

Leading Slovenian research organization named after the most recognized Slovenian scientist Jožef Stefan

radiation of a black body ($j = \sigma T^4$)

Stefan's problems (study of ice growth)

JSI employs approximately 780 employees

150 doctoral students

350 professional scientists with Ph. D.

There are three main research branches at JSI

Physics:

Theoretical Physics, Low and Medium Energy Physics, Thin Films and Surfaces, Surface Engineering and Optoelectronics, Condensed Matter Physics, Complex Matter, Reactor Physics, Experimental Particle Physics

Chemistry and Biochemistry:

Biochemistry and Molecular Biology, Molecular and Biomedical Sciences, Biotechnology , Inorganic Chemistry and Technology, Electronic Ceramics, Engineering Ceramics, Nanostructured Materials, Synthesis of Materials, Advanced Materials. Environmental Sciences

Electronics and Information Technologies

Automation, Bio-cybernetics and Robotics, Systems and Control, Artificial Intelligence, Open Computer Systems and Networks, Communication Systems, Computer Systems, Knowledge Technologies, Intelligent Systems

The principal funding of JSI

60 % national programme – 10 % applied research - 10 % international projects – 20 % industrial contracts

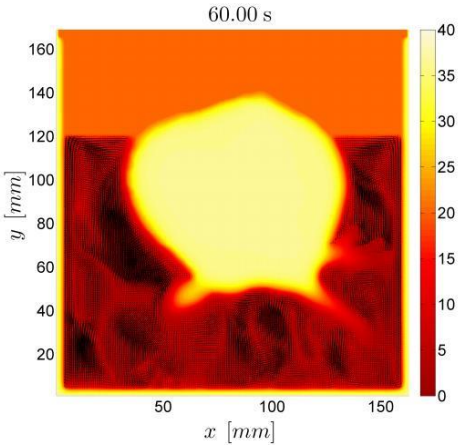
Parallel and distributed systems laboratory

8 full time researchers + 2PhD students + 7 MSc students

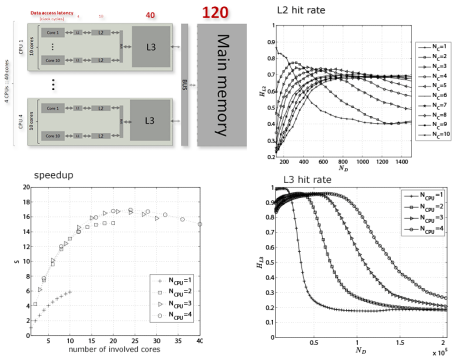
Interested in:

Computational modelling, numerical methods, computer simulations, Parallelization and performance analysis, optimization, processor architectures, interconnection networks, clusters, grid and cloud computing, and processing of Bio-signals.

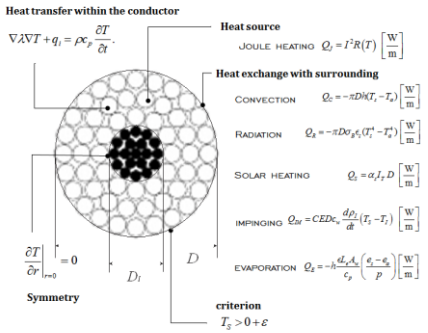
Smartphone application for live ECG monitoring



Simulated temperature distribution of a heart cross-section.



Memory architecture of a test computer, L2 and L3 hit rates, and speedup.



Physical model of dynamic thermal rating

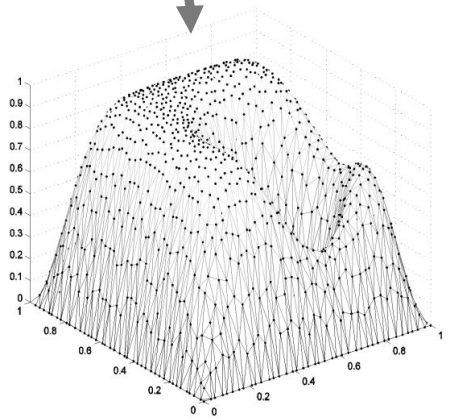
Computational modelling

Prepare a physical model in terms of PDEs

$$\rho\frac{\partial \mathbf{v}}{\partial t}+\rho\nabla\cdot(\mathbf{vv})=-\nabla P+\nabla\cdot\left(\mu\nabla\mathbf{v}\right)+\mathbf{b}$$

$$\rho\frac{\partial\left(c_pT\right)}{\partial t}+\rho\nabla\cdot(c_pT\mathbf{v})=\nabla\cdot\left(\lambda\nabla T\right)$$

Usually PDEs do not have closed form solution



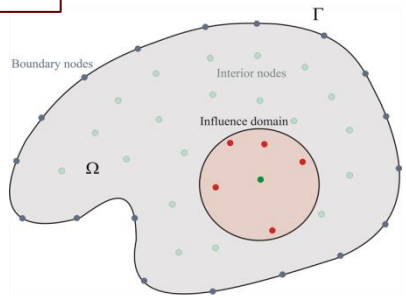
Employ numerical method to transform PDE to the system of algebraic equations

Discretize the domain into final number of elements or nodes.

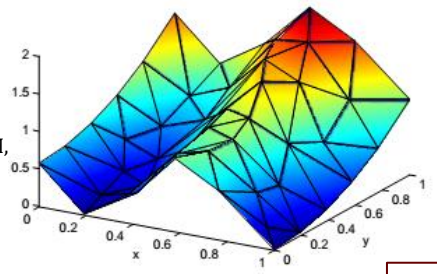
Approximate partial differential operator

Approximate PDE resulting in a linear system

Solve the linear system



Meshless methods
MLPG, DAM, LRBFCM, SPH



Traditional mesh-based methods FDM, FVM, FEM

Implement and execute it on different computer architectures

```
class trivial2D : public optimObjBase{
      typedef trivial2D TT;
public:
      ///PARAMETERs AND CRITERIA definition
      enum CriteriaName{R1,endCrit};
      enum ParamName{X,Y,endParam};
      trivial2D(){
            params.resize(endParam);
            criteria.resize(endCrit);
      }
};
```

Final goal is to create expressive, robust and computationally efficient implementation of numerical solution procedure

Discretization of the domain

The most basic way to generate node sets is to employ existing tools and algorithms for mesh generation, use the generated nodes and simply discard the connectivity relations

(G.-R. Liu, Mesh free methods: moving beyond the finite element method, CRC press, 2002)

- Conceptually flawed
- Expensive
- Inappropriate for meshless

A common iterative approach is to position nodes by simulating free charged particles, obtaining so-called minimal energy nodes, Other iterative methods include bubble simulation, Voronoi relaxation or a combination of both. [17, 24, 1]

(D. P. Hardin and E. B. Saff, Discretizing manifolds via minimum energy points, Notices of the AMS, 51 (2004), pp. 1186-1194

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 (2010), p. 89)

- Iterative methods are computationally expensive and require an initial distribution
- can be used as a post processing

Advancing front methods, which usually begin at the boundary and advance towards the domain interior, filling it in the process.

(R. Lohner and E. Onate , A general advancing front technique for filling space with arbitrary objects, Int. J. Numer. Methods Eng., 61 (2004), pp. 1977)

- Often limited to 2D

Circle or sphere packing methods

(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, Citeseer, 2000.)

- Expensive
- Good quality of nodal distribution

Poisson Disk sampling based algorithms

(R. L. Cook, Stochastic sampling in computer graphics, ACM Trans. Graphics, 5 (1986), pp. 51

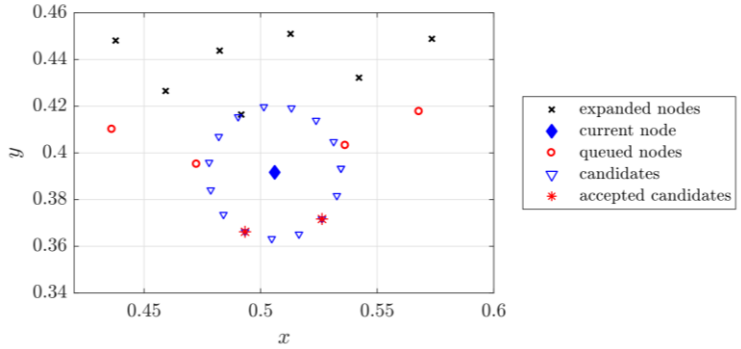
V. Shankar, R. M. Kirby, and A. L. Fogelson, Robust node generation for meshfree discretizations on irregular domains and surfaces, SIAM J. Sci. Comput., 40 (2018), pp. 2584

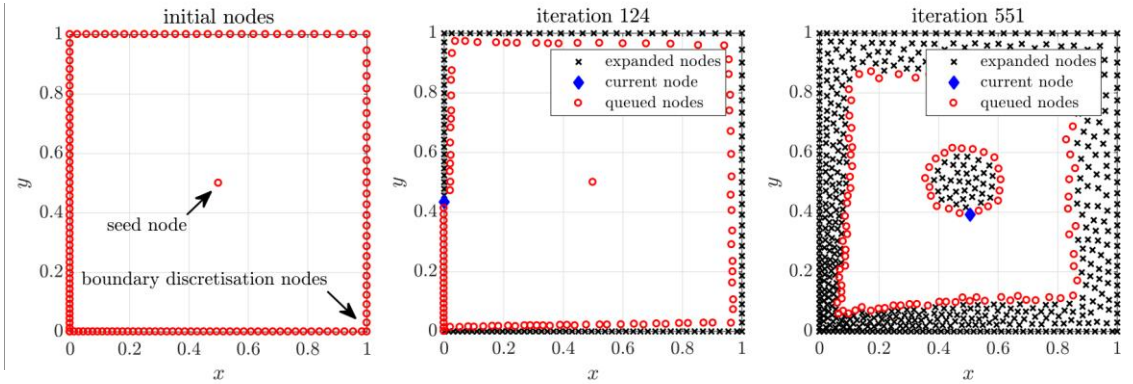
Slak J., Kosec G., On Generation of Node Distributions for Meshless PDE Discretizations, SIAM Journal on Scientific Computing, 41(5))

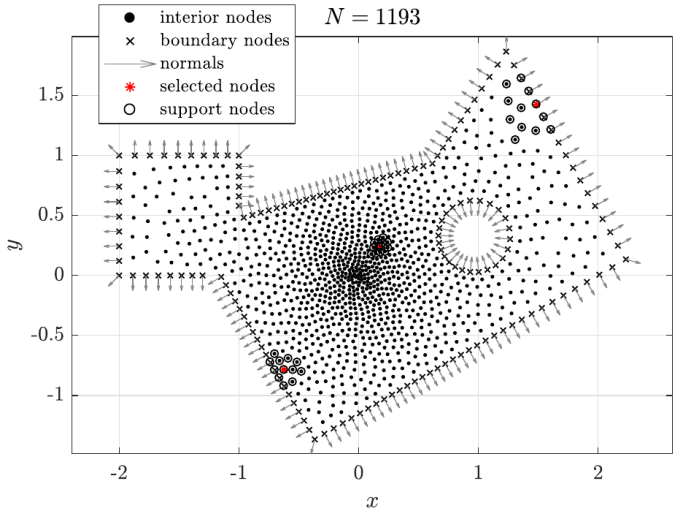
- Computationally effective
- Dimension independent
- Good quality of nodal distribution

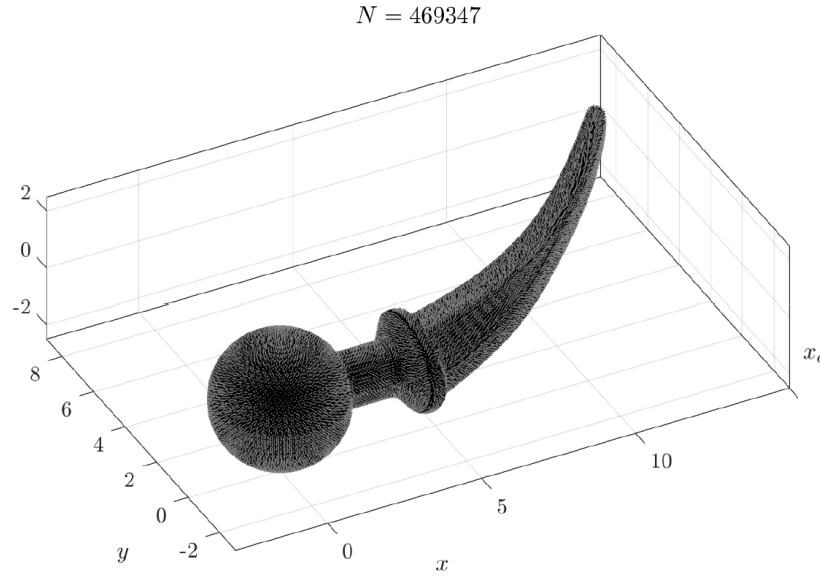
Poisson disk sampling based fill

- Local regularity with minimal spacing guaranteed
- Spatially variable density
- Scalable
- Compatible with irregular domains
- Dimension independent







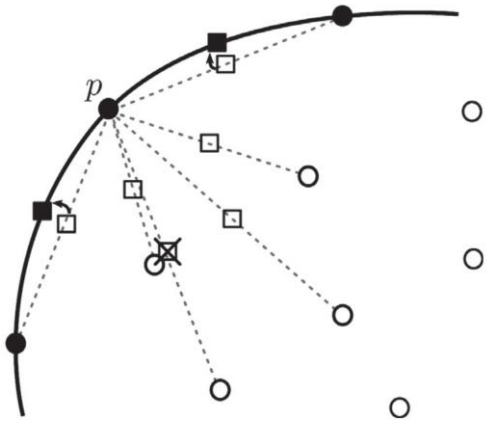


$$x_1 = r\cos(\phi_1)$$
$$x_2 = r\sin(\phi_1)\cos(\phi_2)$$
$$x_3 = r\sin(\phi_1)\sin(\phi_2)\cos(\phi_3)$$
$$\vdots$$
$$x_{d-1} = r\sin(\phi_1)\cdots\sin(\phi_{d-2})\cos(\phi_{d-1})$$
$$x_d = r\sin(\phi_1)\cdots\sin(\phi_{d-2})\sin(\phi_{d-1})$$

H-ADAPTIVITY AND RELAX ALGORITHMS

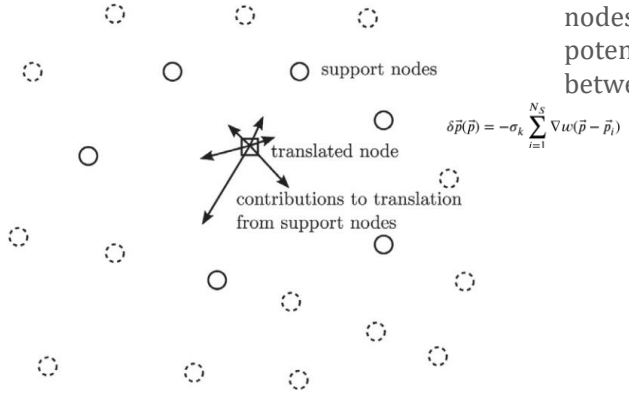
Simple dimension independent h-adaptive algorithm

In each node to be refined, new nodes are added on the half distances between the node itself and its support nodes.

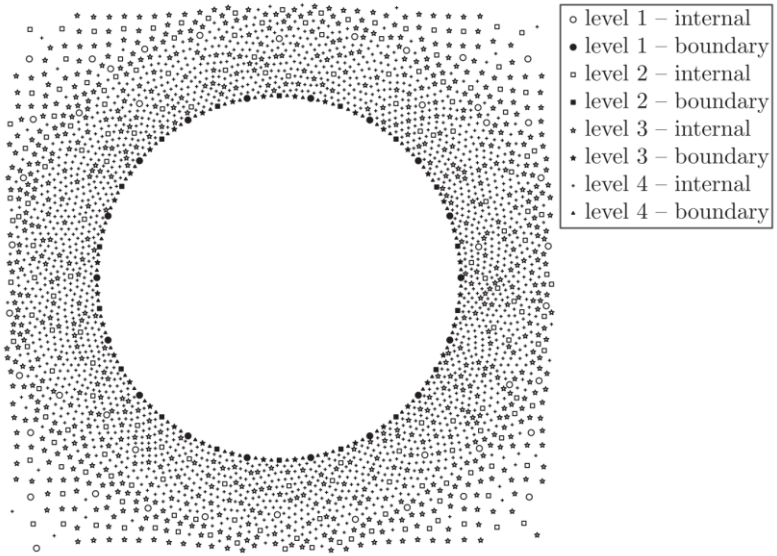


Dimension independent relax post-process algorithm

The basic idea is to "relax" the nodes based on a potential between them.

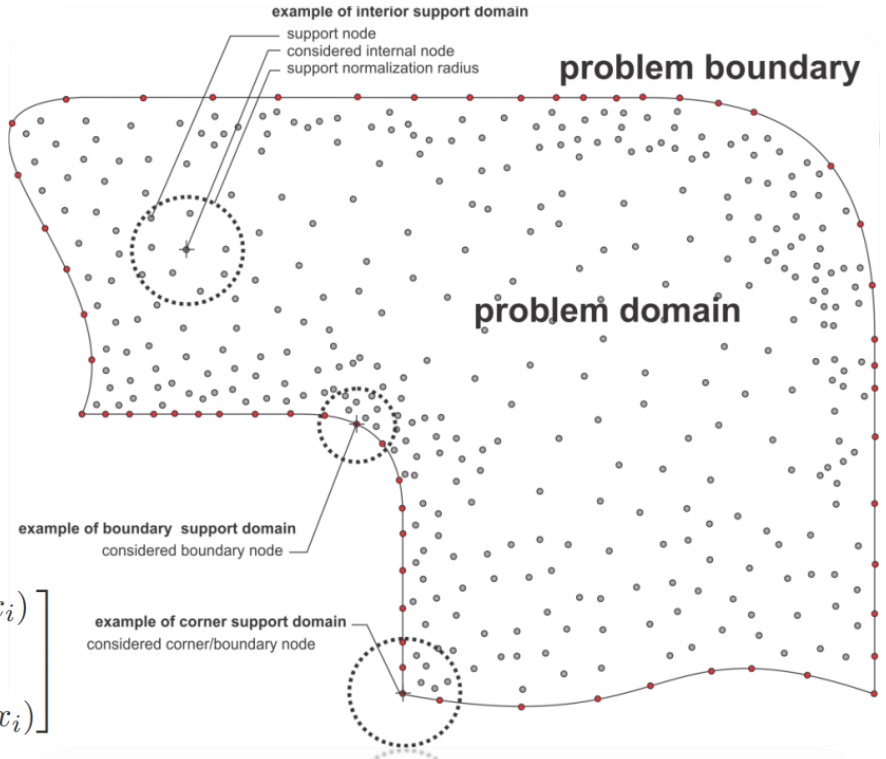


Four levels of the refinement algorithm applied around a hole in a domain after relaxation.



Differential operator approximation

The main idea behind meshless method is the approximation of differential operators over the local cluster of nodes, in many cases simply *n* closest nodes



Differential operator is approximated as

$$(\mathcal{L}u)(x_i) \approx \sum_{x_j \in N(x_i)} w_j^i u(x_j)$$

Imposing exactness for a certain set of basis functions, e.g. monomials, MQs, Gaussians, etc., results in a system

$$\begin{bmatrix} \varphi(\|x_{j_1} - x_{j_1}\|) & \cdots & \varphi(\|x_{j_{n_i}} - x_{j_1}\|) \\ \vdots & \ddots & \vdots \\ \varphi(\|x_{j_1} - x_{j_{n_i}}\|) & \cdots & \varphi(\|x_{j_{n_i}} - x_{j_{n_i}}\|) \end{bmatrix} \begin{bmatrix} w_{j_1}^i \\ \vdots \\ w_{j_{n_i}}^i \end{bmatrix} = \begin{bmatrix} (\mathcal{L}\varphi_{j_1})(x_i) \\ \vdots \\ (\mathcal{L}\varphi_{j_{n_i}})(x_i) \end{bmatrix}$$

That can be subjected to additional weighting (W), when working with more nodes in support than basis functions (n>m) [1]

To enforce consistency of the approximation system can be augmented with monomials up to a certain order

$$\begin{bmatrix} A & P \\ P^{\mathsf{T}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\ell}_\varphi \\ \boldsymbol{\ell}_p \end{bmatrix} \qquad P = \begin{bmatrix} p_1(\boldsymbol{x}_1) & \cdots & p_s(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ p_1(\boldsymbol{x}_n) & \cdots & p_s(\boldsymbol{x}_n) \end{bmatrix}, \quad \boldsymbol{\ell}_p = \begin{bmatrix} (\mathcal{L}p_1)|_{\boldsymbol{x}=\boldsymbol{x}^*} \\ \vdots \\ (\mathcal{L}p_s)|_{\boldsymbol{x}=\boldsymbol{x}^*} \end{bmatrix}$$

[1] http://e6.ijs.si/medusa/wiki/index.php/Computation_of_shape_functions

Differential operator approximation

Several strong form methods that can be described with this approach

Finite Differences Method → n=3, W(p)=1, b={1, x, x2} on regular nodes

Local Radial Basis Function Collocation Method → n, W(p)=1, b={sqrt(1+(p/c)2)}, m=n

Šarler, B. (2007): From global to local radial basis function collocation method for transport phenomena, Advances in Meshfree Techniques, Springer, Berlin, pp. 257-282.

Generalized FDM - Finite pointset method → n=20-50, W(p)=exp(-(p/s)2), b={1, x, x^2, …}

S Tiwari, J Kuhnert - Meshfree methods for partial differential equations, 2003 - Springer

Diffuse Approximate Method → n=13, W(p)=exp(-(p/s)2), b ={1, x, y, x^2, y^2, xy}

Diffuse approximation method for solving natural convection in porous media,C Prax, H Sadat, P Salagnac - Transport in Porous Media, 1996 – Springer

Radial basis function generated finite differences (RBF-FD) → n=12, basis = r^3, augmentation ={1, x, y, x^2, y^2, xy}

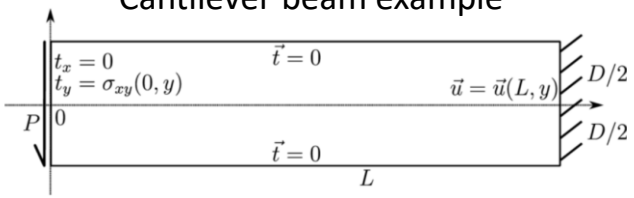
Accuracy of radial basis function interpolation and derivative approximations on 1-D infinite grids, B Fornberg, N. Flyer - Advances in Computational Mathematics, 2005 – Springer

… and many more

More details on differential operator approximation such as complexity analysis, ghost nodes, boundary conditions, implementation notes, stability, … can be found at out discussion wiki page -- http://e6.ijs.si/medusa/wiki/index.php/Medusa

Approximation of PDE

Cantilever beam example



Governing model

$$(\lambda+\mu)\nabla(\nabla\cdot\mathbf{u})+\mu\nabla^2\mathbf{u}=\mathbf{b}$$

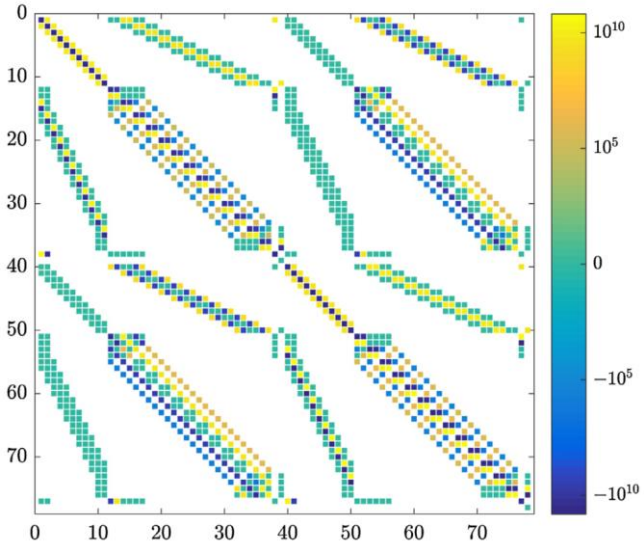
(Navier Cauchy equation)

Assembly of linear system

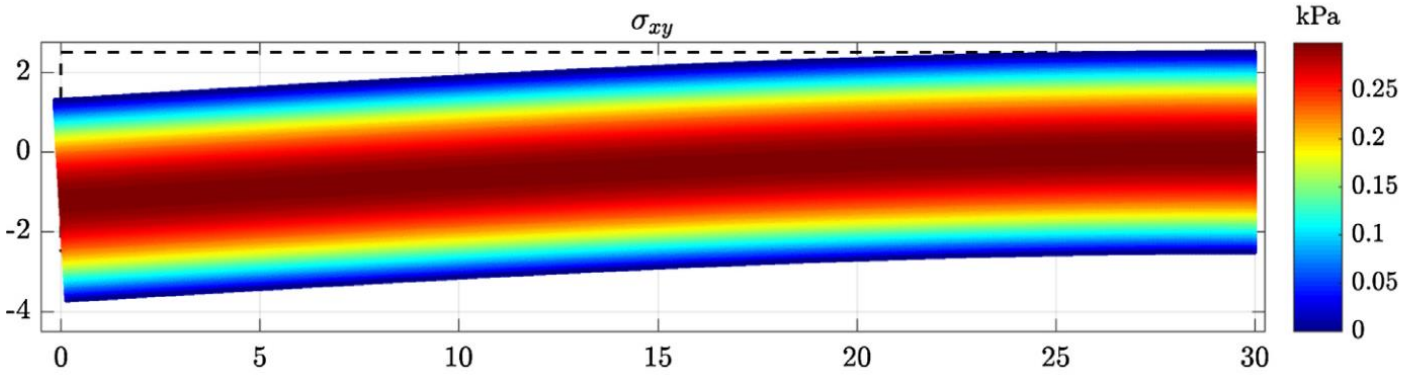
$$\begin{bmatrix} U1 & V1 \\ U2 & V2 \end{bmatrix}\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \end{bmatrix}$$

Approximation of differential operator

$$(\mathcal{L}u)(x_i) \approx \sum_{x_j \in N(x_i)} w_j^i u(x_j)$$

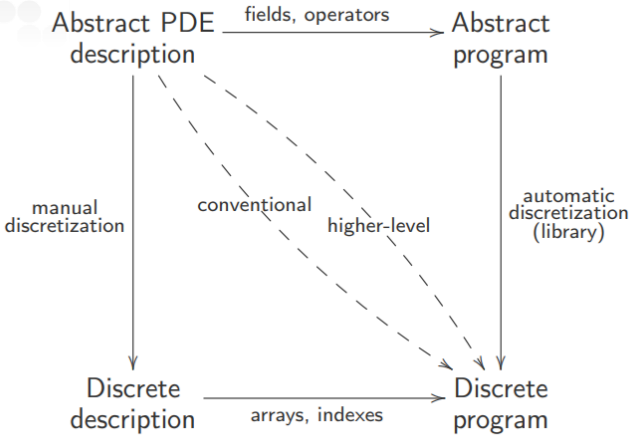


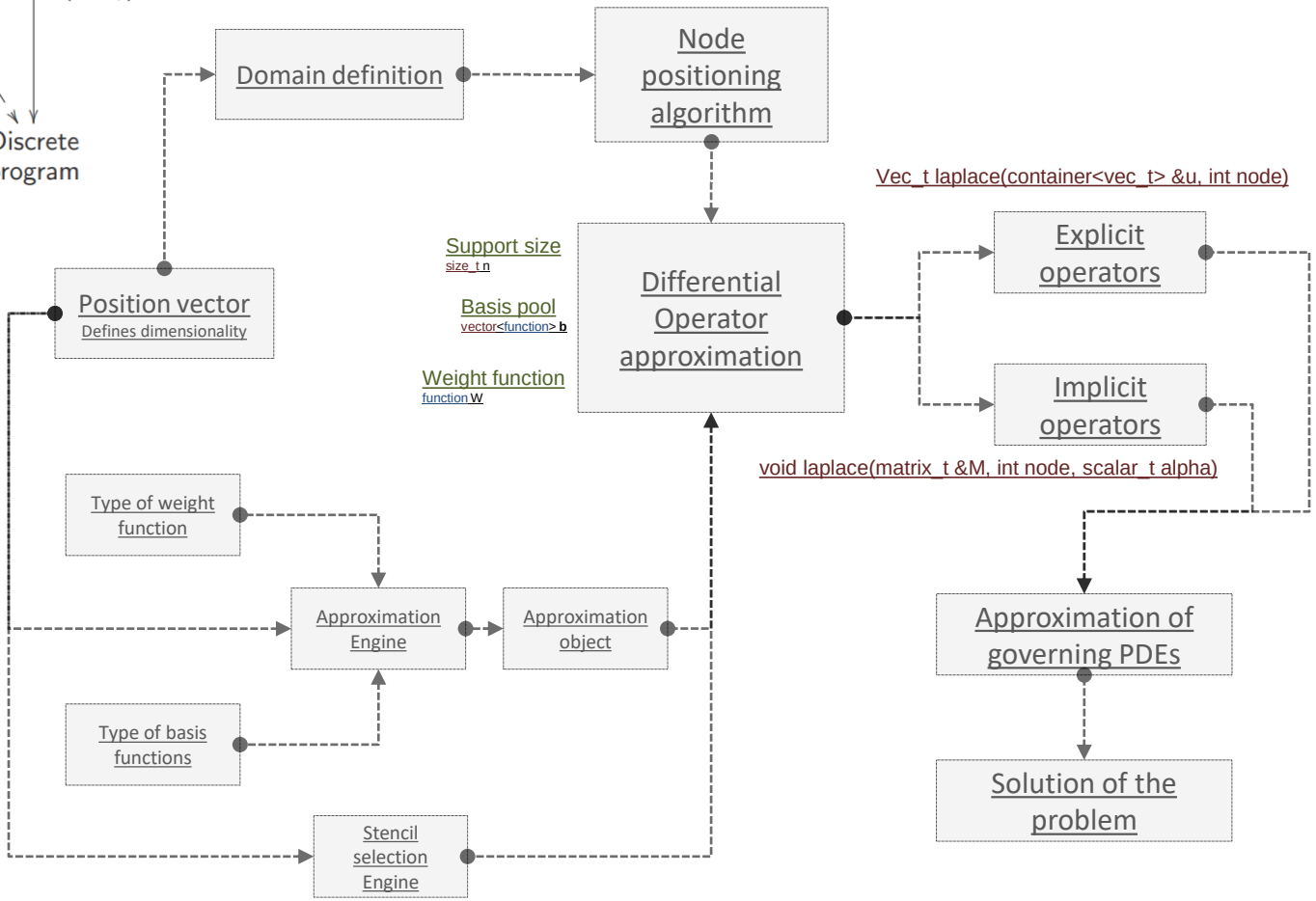
whose solution stands for discrete solution of governing problem



. Numerical solution of cantilever beam case. Note that for the sake of visibility the displacements are multiplied by factor 10^5

Concept of the Medusa Implementation





Basic information of Medusa

Main features:

- Coordinate free and dimension independent implementation
- Support different strong form meshless methods
- Explicit transformation of equations into code
- Minimal overheads due to the programming abstraction
- Tested code

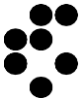
Medusa is C++ template library using

- Eigen library for linear algebra
- Google test testing framework with approximately 500 tests
- XML and HDF5 support for IO
- Nanoflann for spatial-search structures

Doxygen http://e6.ijs.si/medusa/docs/html

MediaWiki http://e6.ijs.si/medusa/wiki/index.php/Medusa

```
[ RUN      ] MLSM_Operators.explicit3DLaplace
[       OK ] MLSM_Operators.explicit3DLaplace (3 ms)
[ RUN      ] MLSM_Operators.explicit3DGradient
[       OK ] MLSM_Operators.explicit3DGradient (3 ms)
[ RUN      ] MLSM_Operators.explicit3DDivergence
[       OK ] MLSM_Operators.explicit3DDivergence (3 ms)
[ RUN      ] MLSM_Operators.explicit3DLaplaceOfVector
[       OK ] MLSM_Operators.explicit3DLaplaceOfVector (2 ms)
[ RUN      ] MLSM_Operators.explicit3DGradientOfVector
[       OK ] MLSM_Operators.explicit3DGradientOfVector (3 ms)
[ RUN      ] MLSM_Operators.implicit1DLaplaceDense
[       OK ] MLSM_Operators.implicit1DLaplaceDense (0 ms)
[ RUN      ] MLSM_Operators.implicit2DLaplaceSparse
[       OK ] MLSM_Operators.implicit2DLaplaceSparse (1 ms)
[ RUN      ] MLSM_Operators.implicit2dLaplaceOfvector
[       OK ] MLSM_Operators.implicit2dLaplaceOfvector (108 ms)
[ RUN      ] MLSM_Operators.implicit2dGradOfScalar
[       OK ] MLSM_Operators.implicit2dGradOfScalar (20 ms)
[ RUN      ] MLSM_Operators.implicit2dGradOfVector
[       OK ] MLSM_Operators.implicit2dGradOfVector (21 ms)
[ RUN      ] MLSM_Operators.ScalarNeumannLinear
[       OK ] MLSM_Operators.ScalarNeumannLinear (9 ms)
[ RUN      ] MLSM_Operators.ScalarNeumannSin
[       OK ] MLSM_Operators.ScalarNeumannSin (0 ms)
[ RUN      ] MLSM_Operators.ScalarBoundaryDeathTest
[       OK ] MLSM_Operators.ScalarBoundaryDeathTest (58 ms)
[ RUN      ] MLSM_Operators.VectorNeumannSin
[       OK ] MLSM_Operators.VectorNeumannSin (1 ms)
[ RUN      ] MLSM_Operators.VectorBoundaryDeathTest
[       OK ] MLSM_Operators.VectorBoundaryDeathTest (54 ms)
[ RUN      ] MLSM_Operators.Div3DSimple
[ RUN      ] MLSM_Operators.explicit3DLaplace
[       OK ] MLSM_Operators.explicit3DLaplace (3 ms)
[ RUN      ] MLSM_Operators.explicit3DGradient
[       OK ] MLSM_Operators.explicit3DGradient (3 ms)
[ RUN      ] MLSM_Operators.explicit3DDivergence
[       OK ] MLSM_Operators.explicit3DDivergence (3 ms)
[ RUN      ] MLSM_Operators.explicit3DLaplaceOfVector
[       OK ] MLSM_Operators.explicit3DLaplaceOfVector (2 ms)
[ RUN      ] MLSM_Operators.explicit3DGradientOfVector
[       OK ] MLSM_Operators.explicit3DGradientOfVector (3 ms)
[ RUN      ] MLSM_Operators.implicit1DLaplaceDense
[       OK ] MLSM_Operators.implicit1DLaplaceDense (0 ms)
[ RUN      ] MLSM_Operators.implicit2DLaplaceSparse
[       OK ] MLSM_Operators.implicit2DLaplaceSparse (1 ms)
[ RUN      ] MLSM_Operators.implicit2dLaplaceOfvector
[       OK ] MLSM_Operators.implicit2dLaplaceOfvector (108 ms)
[ RUN      ] MLSM_Operators.implicit2dGradOfScalar
[       OK ] MLSM_Operators.implicit2dGradOfScalar (20 ms)
[ RUN      ] MLSM_Operators.implicit2dGradOfVector
[       OK ] MLSM_Operators.implicit2dGradOfVector (21 ms)
[ RUN      ] MLSM_Operators.ScalarNeumannLinear
[       OK ] MLSM_Operators.ScalarNeumannLinear (9 ms)
[ RUN      ] MLSM_Operators.ScalarNeumannSin
[       OK ] MLSM_Operators.ScalarNeumannSin (0 ms)
[ RUN      ] MLSM_Operators.ScalarBoundaryDeathTest
[       OK ] MLSM_Operators.ScalarBoundaryDeathTest (58 ms)
[ RUN      ] MLSM_Operators.VectorNeumannSin
[       OK ] MLSM_Operators.VectorNeumannSin (1 ms)
[ RUN      ] MLSM_Operators.VectorBoundaryDeathTest
[       OK ] MLSM_Operators.VectorBoundaryDeathTest (54 ms)
[ RUN      ] MLSM_Operators.Div3DSimple
```



Examples :: basic tests

$$\nabla^2 u(\boldsymbol{x}) = -d\pi^2 \prod_{i=1}^{d} \sin(\pi x_i)$$

$$u(\boldsymbol{x}) = \prod_{i=1}^{d} \sin(\pi x_i)$$

$$\frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x}) = \pi \sum_{i=1}^{d} n_i \cos(\pi x_i) \prod_{j \neq i} \sin(\pi x_j)$$

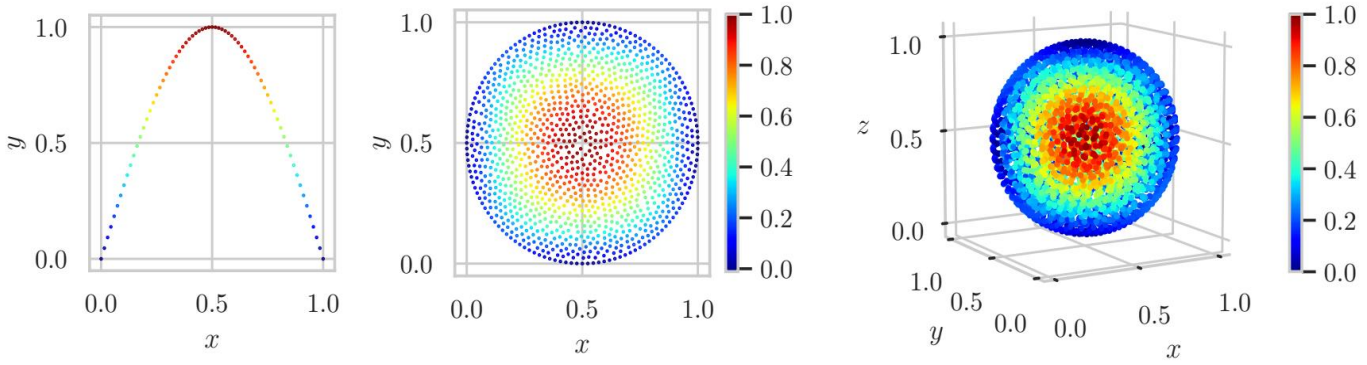
$$\Omega = \left\{ \boldsymbol{x},\ \left\| \boldsymbol{x} - \frac{\mathbf{1}}{\mathbf{2}} \right\| < \frac{1}{2} \right\},$$

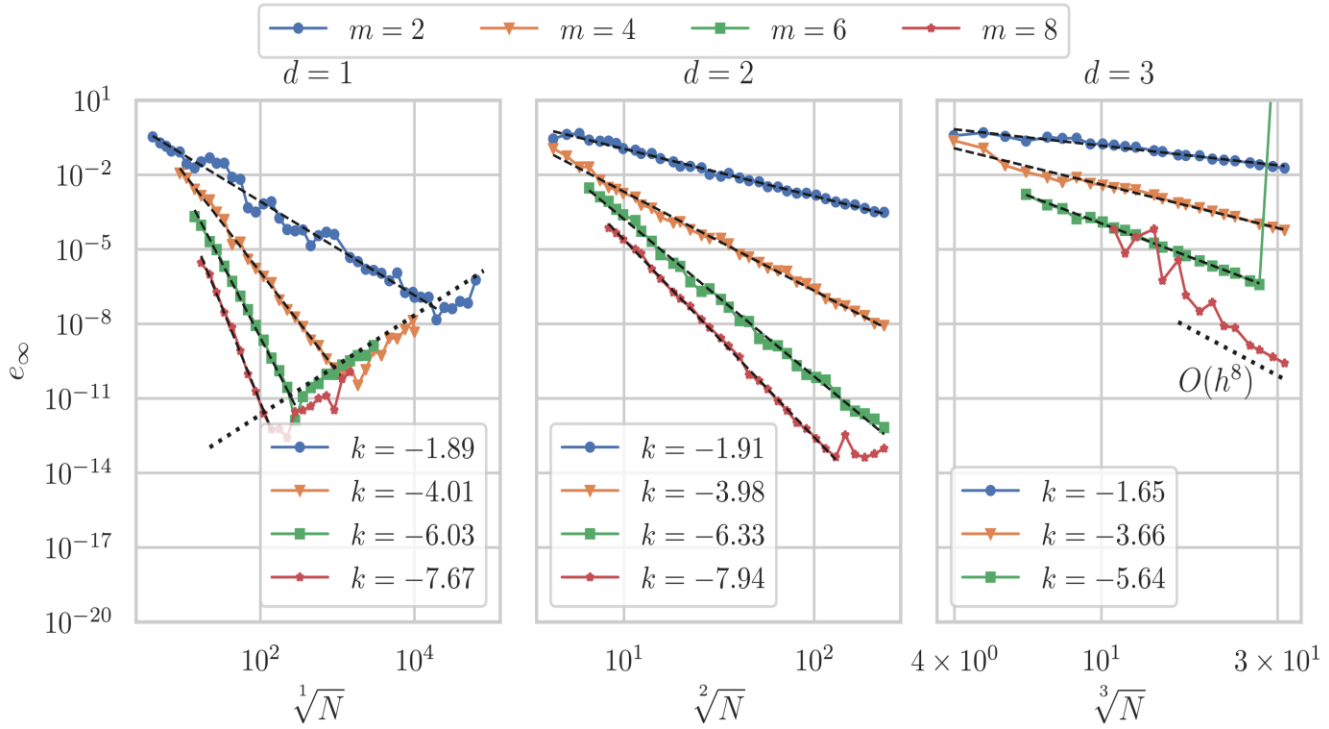
$$\Gamma_d = \left\{ \boldsymbol{x} \in \partial\Omega,\ x_1 < \frac{1}{2} \right\},$$

$$\Gamma_n = \left\{ \boldsymbol{x} \in \partial\Omega,\ x_1 \geq \frac{1}{2} \right\}.$$

```
// Domain definition and discretization
BallShape<vec> b1(origin, radius);
DomainDiscretization<vec> domain = b1.discretizeWithStep(dx);

// Define differential operator approximation
Monomials<vec> mon(m);
Polyharmonic<double, k> ph;
RBFFD<decltype(ph), vec, ScaleToFarthest> appr(ph, mon);

// Compute stencil weights (shapes) with RBF-FD
auto storage = domain.computeShapes<sh::lap|sh::d1>(appr);
Eigen::SparseMatrix<double, Eigen::RowMajor> M(N, N);
M.reserve(storage.supportSizes());
Eigen::VectorXd rhs(N); rhs.setZero();

// Prepare "operators" abstraction
auto op = storage.implicitOperators(M, rhs);

// PDE discretization
// Interior
for (int i : interior) {
    op.lap(i) = -dim * PI * PI * sin_product(domain.pos(i));
}
// Dirichlet boundary
for (int i : dir) {
    double sinp = sin_product(domain.pos(i));
    op.value(i) = sinp;
    op.lap(i, gh[i]) = -dim * PI * PI * sinp;
}
// Neumann boundary
for (int i : neu) {
    op.neumann(i, domain.normal(i)) = neummann_bc(domain.pos(i), domain.normal(i));
    op.lap(i, gh[i]) = -dim * PI * PI * sin_product(domain.pos(i));
}

// Solve system
Eigen::BiCGSTAB<decltype(M), Eigen::IncompleteLUT<double>> solver;
solver.compute(M);
ScalarFieldd u = solver.solve(rhs);
```

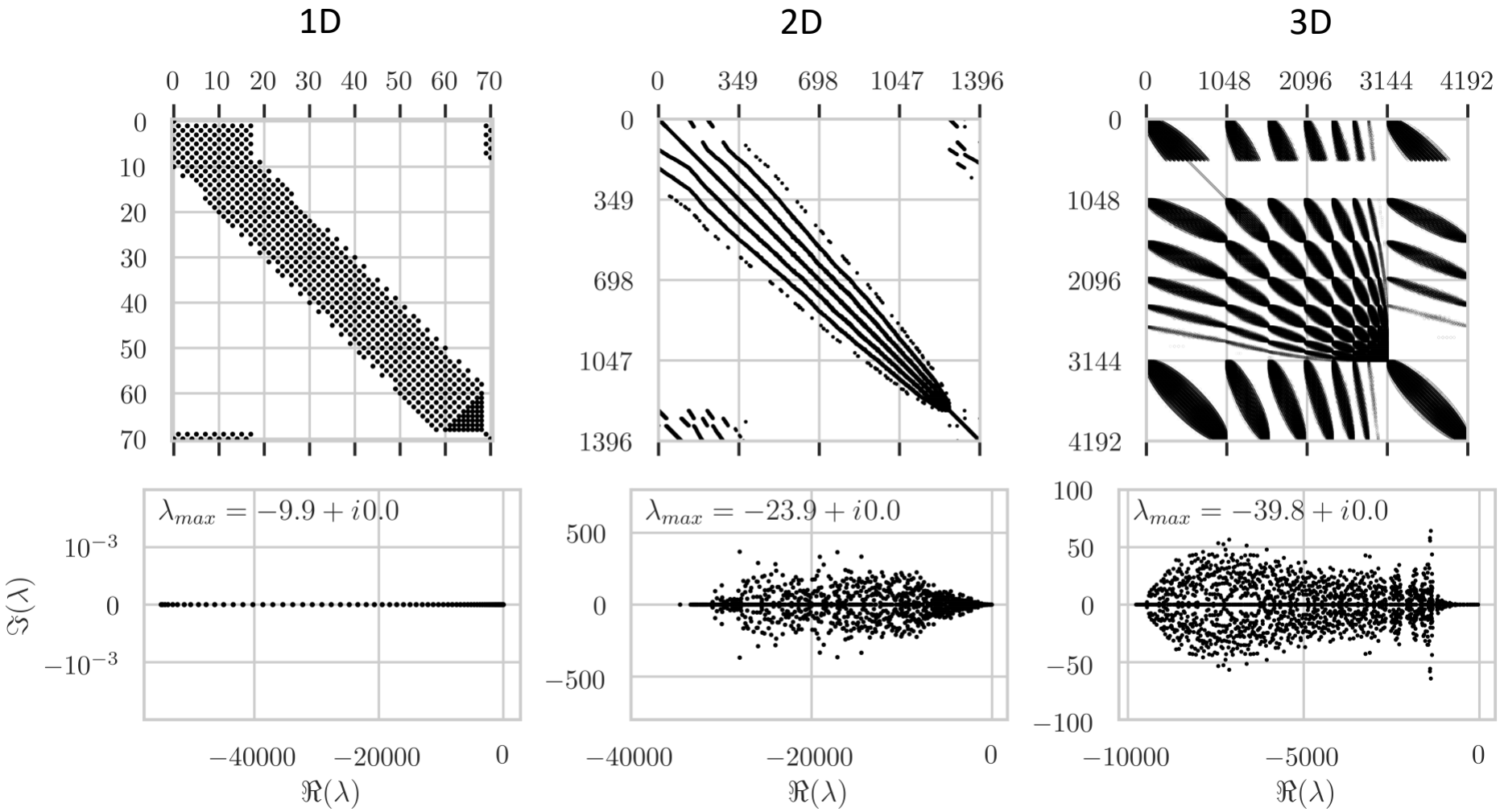
Examples :: basic tests





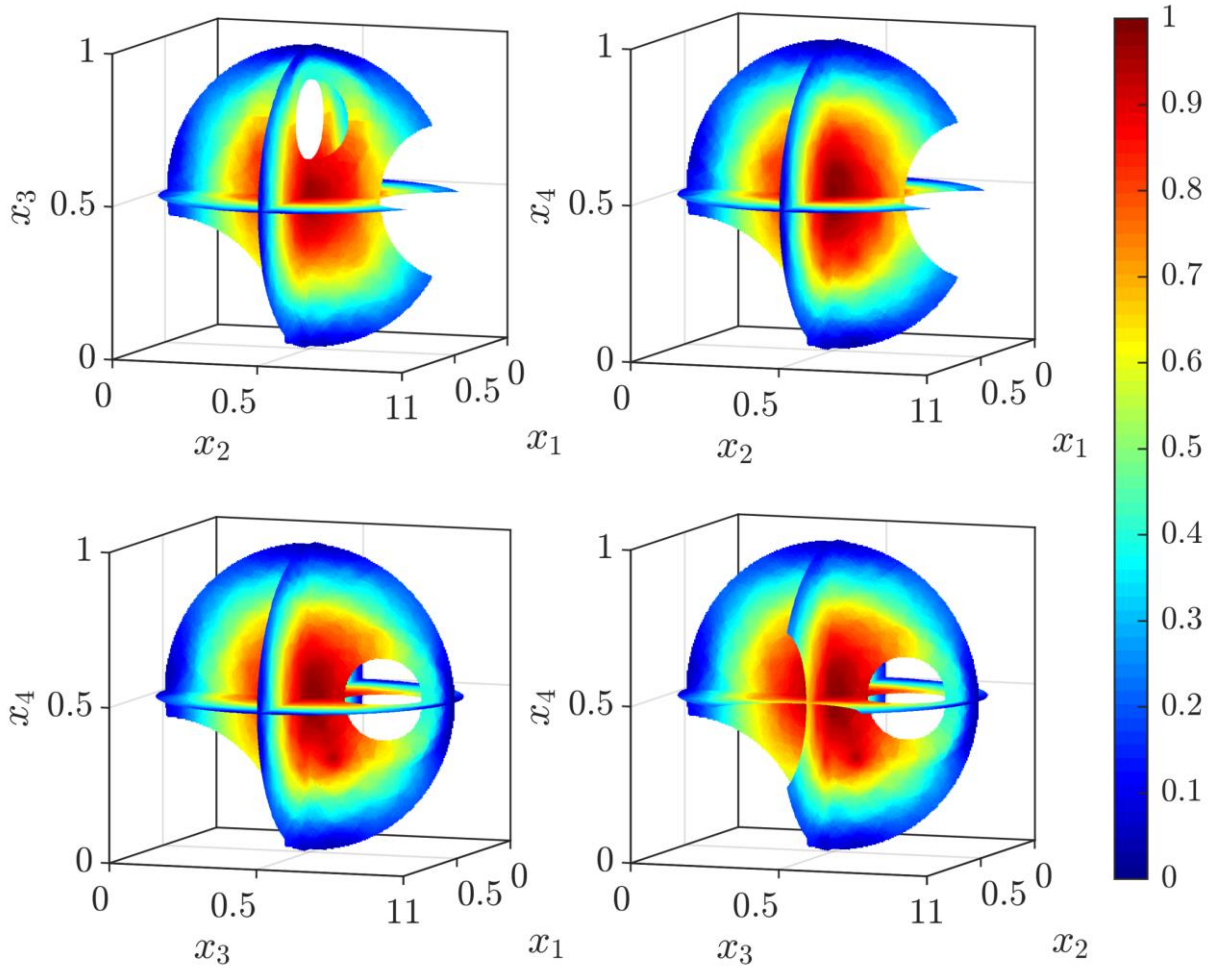
Examples :: basic tests

Global sparse matrices of discretized PDE and theirs spectra.



Examples :: basic tests

Extension to 4D

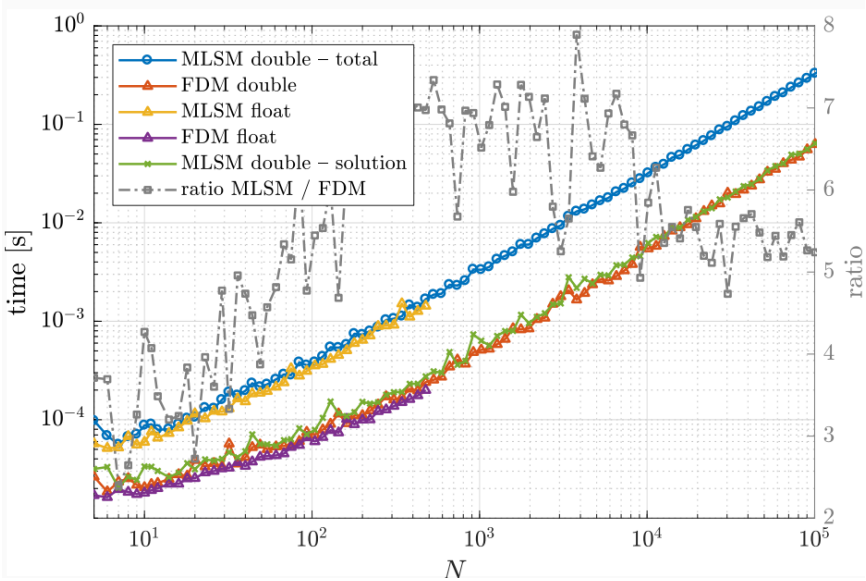


3-dimensional cross sections of a solution to 4-dimensional Poisson problem

Examples :: performance tests

Analysis of potential overheads due to the heavily abstracted code

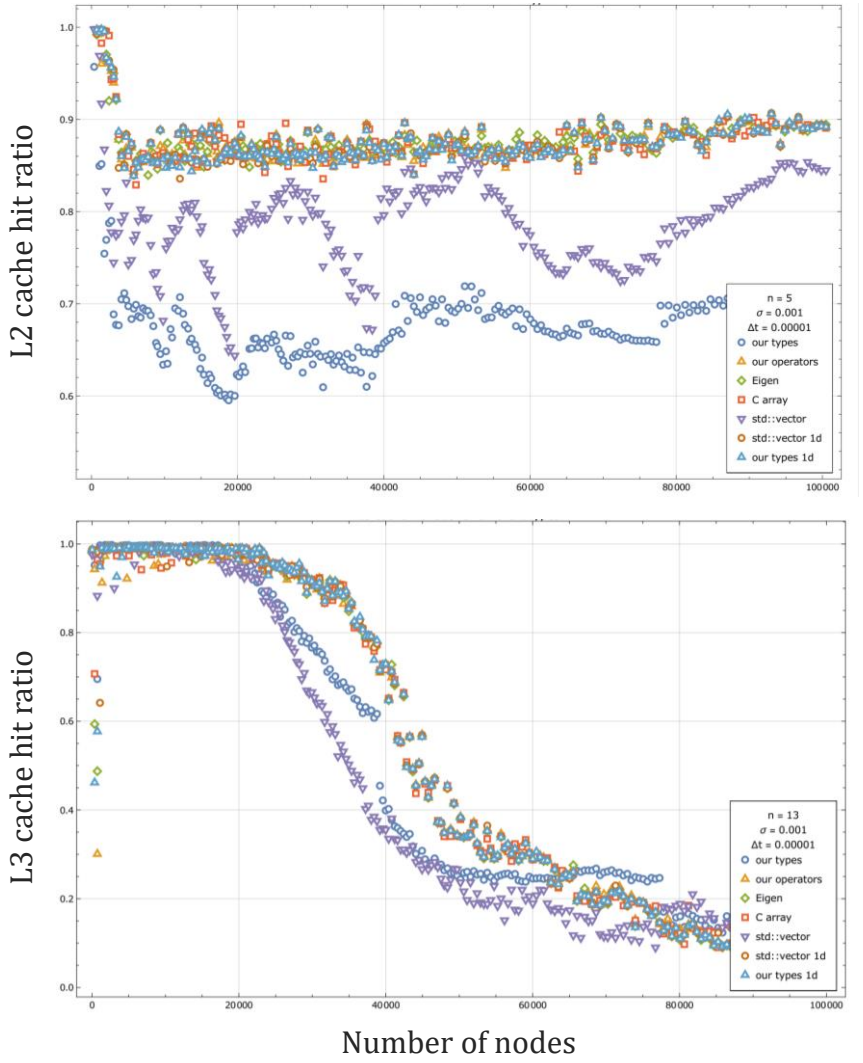
Comparison with primitive FDM implementation



There are no measurable overheads in abstracted code in comparison with plain old data based code.

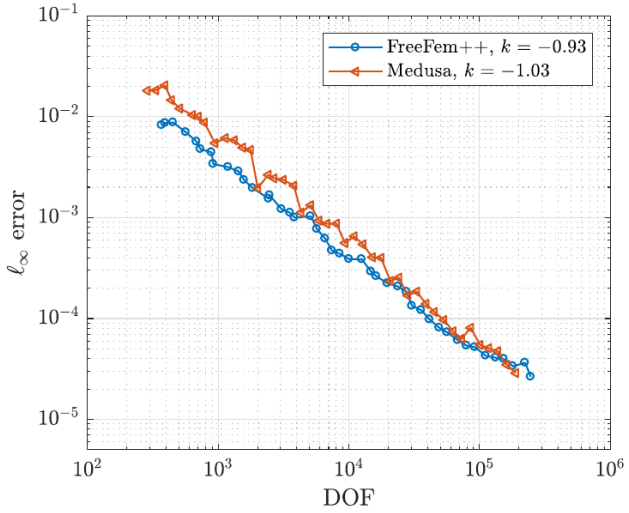
Cache utilization of abstracted code is of the same order in comparison to plain old data based code

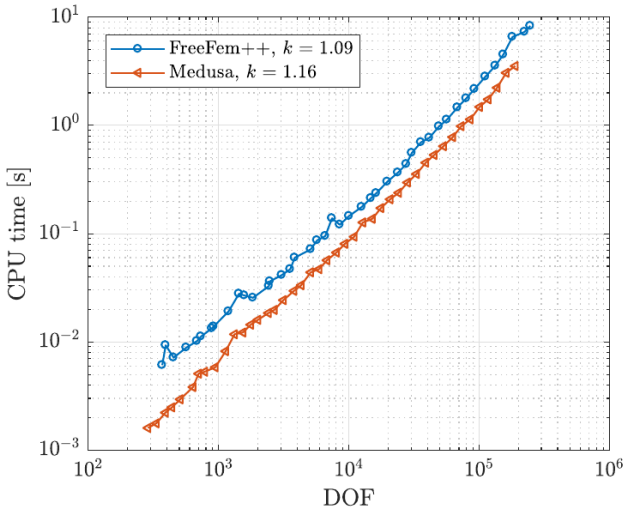
Analysis of low level CPU counters



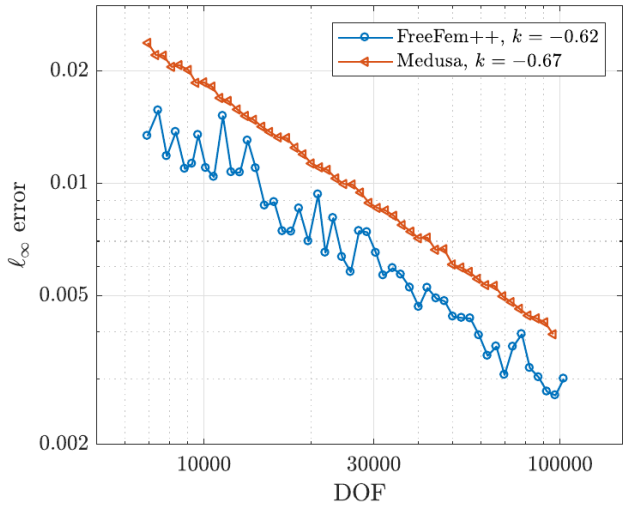
Examples :: performance tests

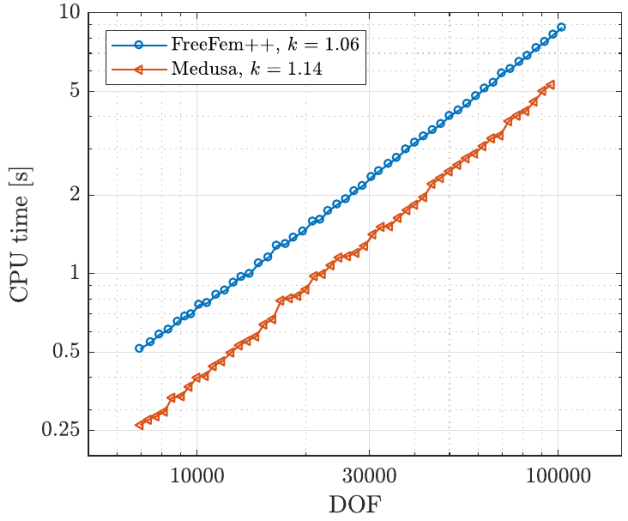
Comparison with FreeFem++





2D





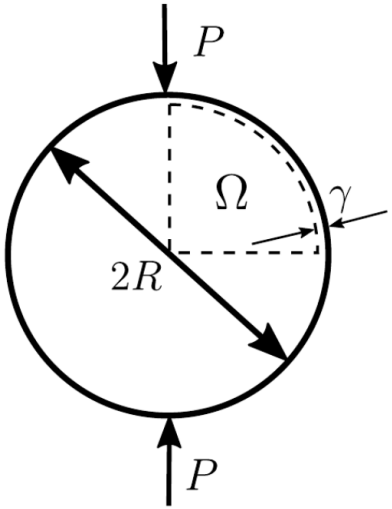
3D

Examples :: Adaptive linear elasticity

Disk under pressure case

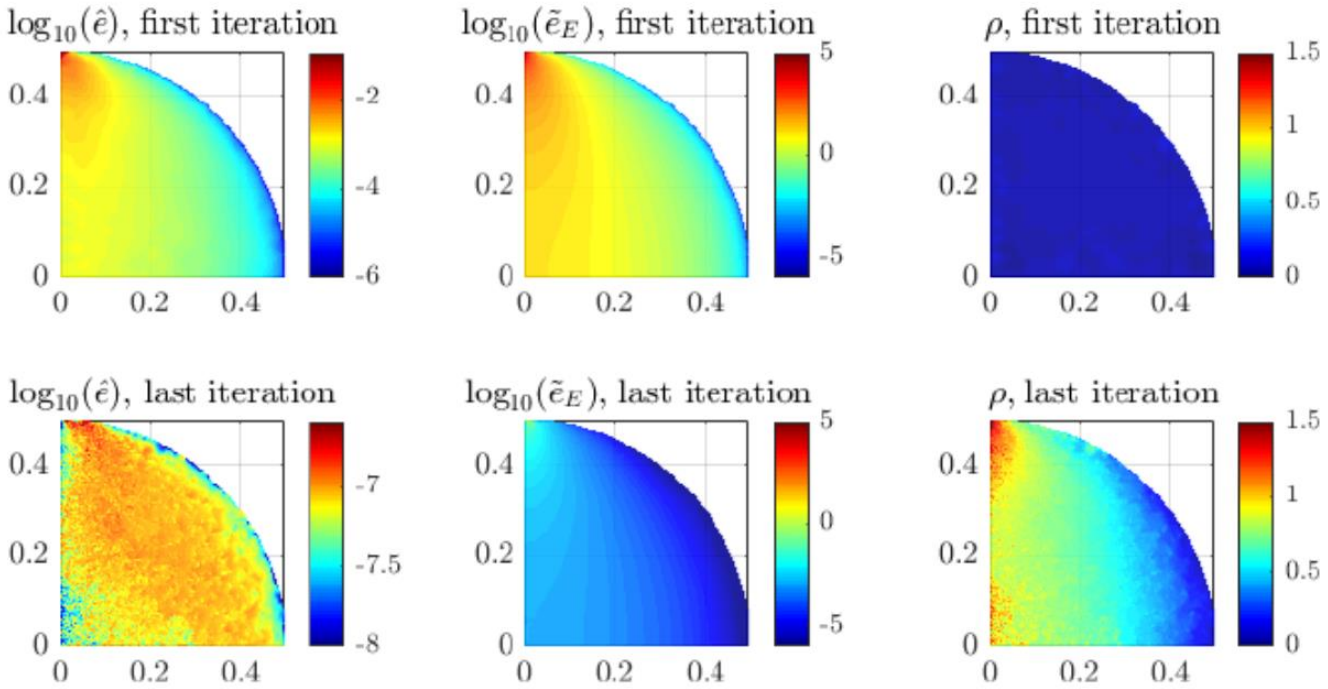
$$(\lambda+\mu)\nabla(\nabla\cdot\vec{u})+\mu\nabla^2\vec{u}=\vec{f}$$

```
for (int i : domain.interior()) {
    (lam+mu)*op.graddiv(i) + mu*op.lap(i) = 0.0;
}
```



Principal "re-mesh" adaptive solution procedure

- Solve governing PDE with uniform node distribution.
- Estimate error with simple ad-hoc error indicator.
- Based on computed error construct new density function for fill algorithm
- Re-position nodes
- repeat.

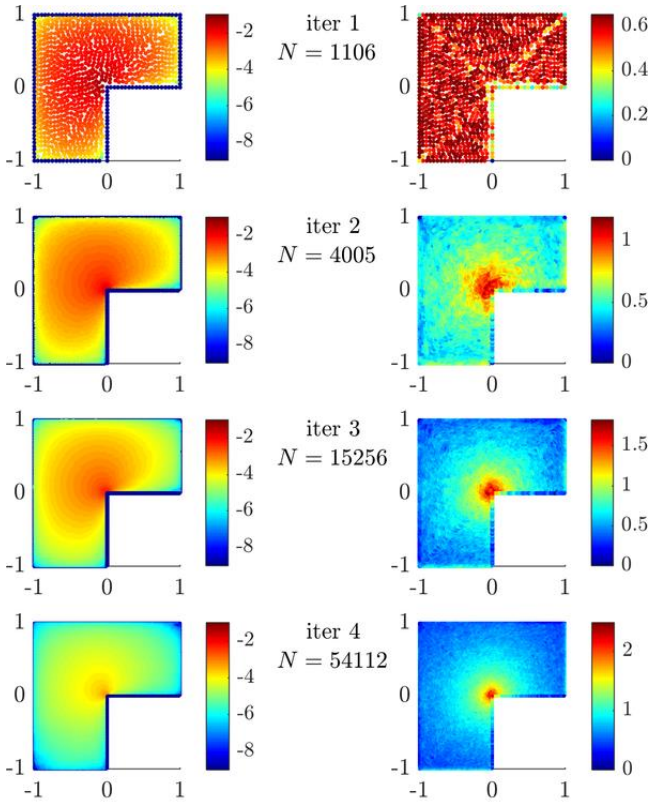


Examples :: Adaptive linear elasticity

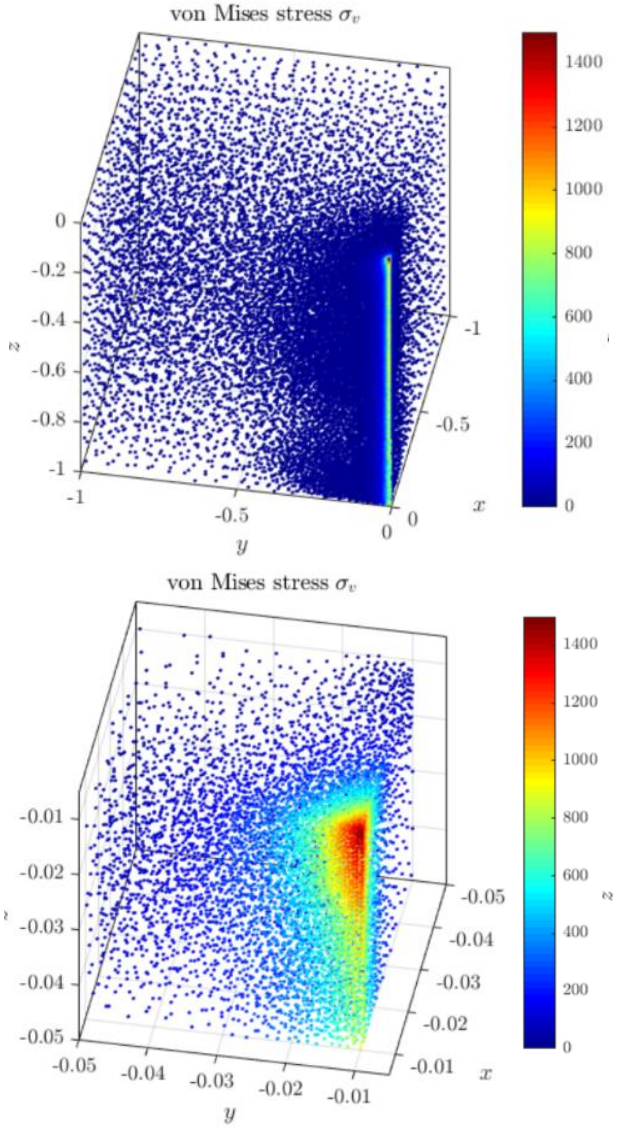
$$(\lambda+\mu)\nabla(\nabla\cdot\vec{u})+\mu\nabla^2\vec{u}=\vec{f}$$

```
for (int i : domain.interior()) {
    (lam+mu)*op.graddiv(i) + mu*op.lap(i) = 0.0;
}
```

Demonstration of adaptive solution on L-shaped domain



Adaptive solution of 3D Boussinesq problem



Examples :: Thermo fluid transport

$$\nabla \cdot \mathbf{v} = 0$$

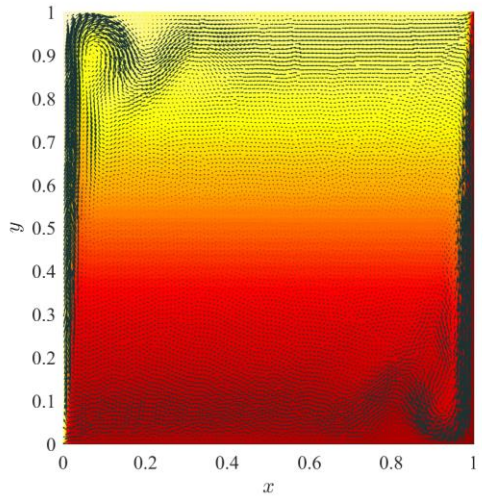
$$\rho \frac{\partial \mathbf{v}}{\partial t} + \rho \nabla \cdot (\mathbf{v}\mathbf{v}) = -\nabla P + \nabla \cdot \left(\mu \nabla \mathbf{v}\right) + \rho \left[1 - \beta_T (T - T_{\text{ref}})\right] \mathbf{g}$$

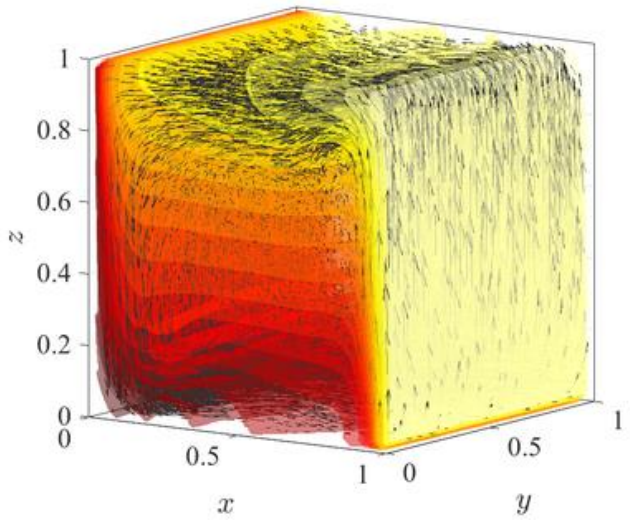
$$\rho \frac{\partial \left(c_p T\right)}{\partial t} + \rho \nabla \cdot (c_p T \mathbf{v}) = \nabla \cdot \left(\lambda \nabla T\right)$$

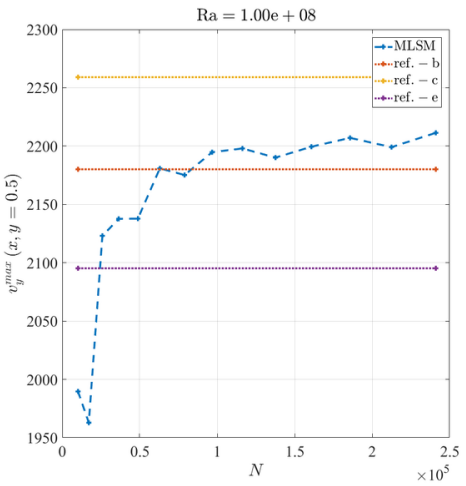
```
1     for (int step = 0; step <= O.t_steps; ++step) {
2         // Explicit Navier-Stokes computed on whole domain, including boundaries
3         // without pressure
4         for (int c:all) {
5             v_2[c] = v_1[c] + O.dt (
6                                     O.mu / O.rho * op.lap(v_1, c)
7                                      - op.grad(v_1, c) * v_1[c]
8                                      + O.g * (1 - O.beta * (T_1[c] - O.T_ref)));
9         }
10        // Pressure correction
11        VecXd rhs_pressure(N + 1, 0); //Note N+1, +1 stands for regularization equation
12        rhs_pressure(N) = 0; // = 0 part of the regularization equation
13        for (int i:interior) rhs_pressure(c) = O.rho / O.dt * op.div(v_2, c);
14        for (int i: boundary) rhs_pressure(c) = O.rho / O.dt * v_2[c].dot(domain.normal(c));
15        VecXd solution = solver_p.solve(rhs_pressure);
16        alpha = solution[N];
17        VecXd P_c = solution.head(N);
18        for ( int i = interior) v_2[c] -=  O.dt / O.rho * op.grad(P_c, c);
19        v_2[boundary] = 0; // force boundary conditions
20        //heat transport
21        for (auto c : interior) {
22            T2[c] = T1[c] + O.dt * O.lam / O.rho / O.c_p * op.lap(T1, c) -
23                    O.dt * v1[c].transpose() * op.grad(T1, c);
24        }
25        for (auto c : top) T2[c] = op.neumann(T2, c, vec_t{0, -1}, 0.0);
26        for (auto c : bottom) T2[c] = op.neumann(T2, c, vec_t{0, 1}, 0.0);
27    }
```

Examples :: Natural convection

Temperature contour and velocity quiver plots of solutions in 2D and 3D domains







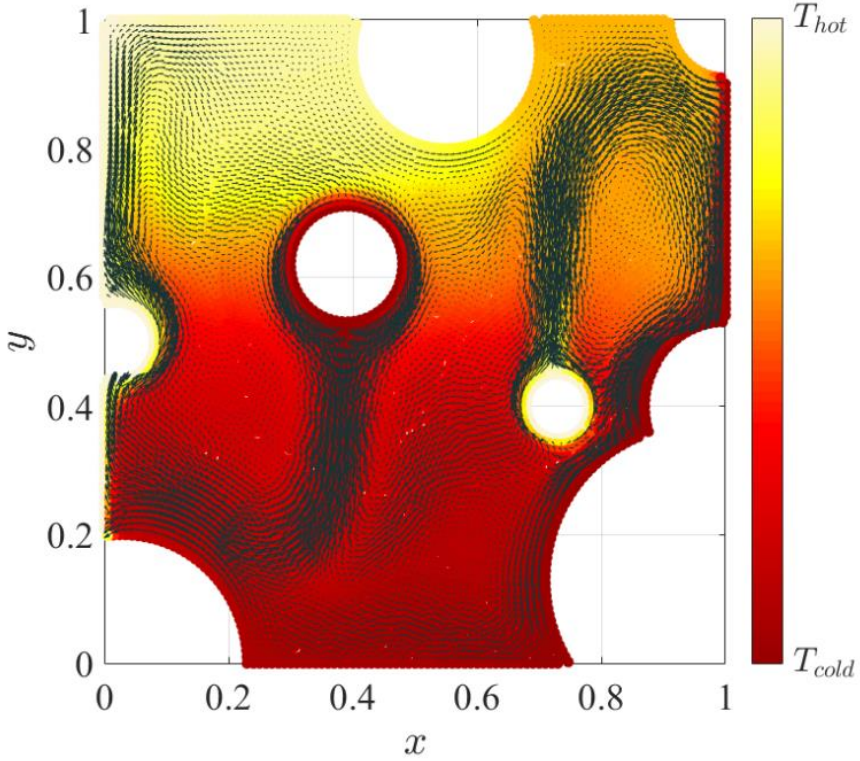
Comparison of results with published data

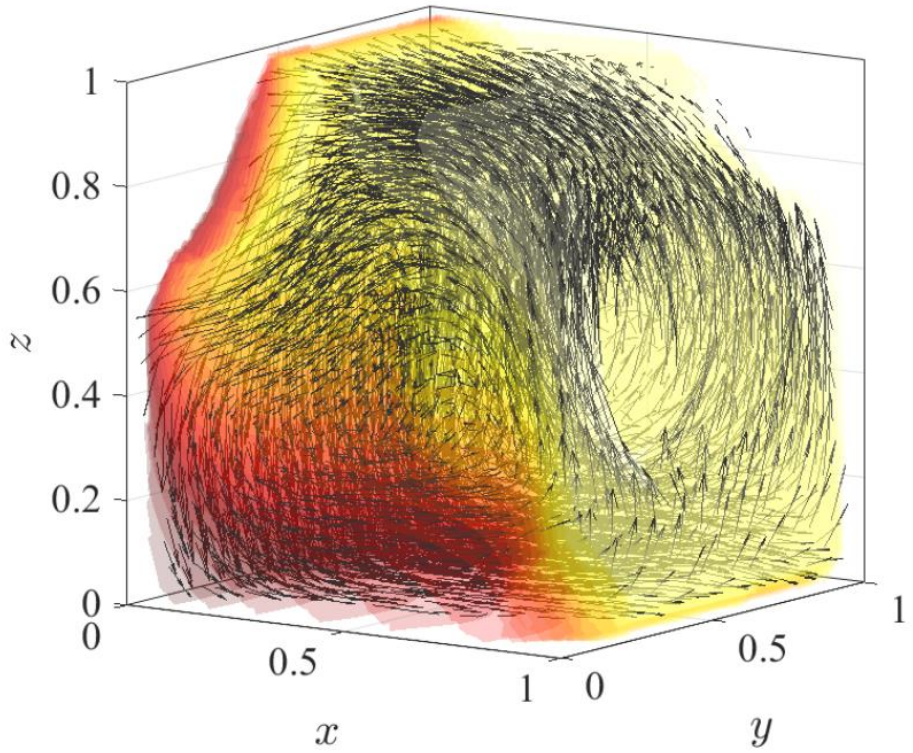
	Ra	$v_{max}(x, 0.5)$			x			$u_{max}(0.5, y)$			y		
		present	[8]	[20]	present	[8]	[20]	present	[8]	[20]	present	[8]	[20]
2D	10^6	0.2628	0.2604	0.2627	0.037	0.038	0.039	0.0781	0.0765	0.0782	0.847	0.851	0.861
	10^7	0.2633	0.2580	0.2579	0.022	0.023	0.021	0.0588	0.0547	0.0561	0.870	0.888	0.900
	10^8	0.2557	0.2587	0.2487	0.010	0.011	0.009	0.0314	0.0379	0.0331	0.918	0.943	0.930
	Ra	$w_{max}(x, 0.5, 0.5)$			x			$u_{max}(0.5, 0.5, z)$			z		
		present	[37]	[14]	present	[37]	[14]	present	[37]	[14]	present	[37]	[14]
3D	10^4	0.2295	0.2218	0.2252	0.850	0.887	0.883	0.2135	0.1968	0.2013	0.168	0.179	0.183
	10^5	0.2545	0.2442	0.2471	0.940	0.931	0.935	0.1564	0.1426	0.1468	0.144	0.149	0.145
	10^6	0.2564	0.2556	0.2588	0.961	0.965	0.966	0.0841	0.0816	0.0841	0.143	0.140	0.144

Examples :: Natural convection

Generalisation to irregular domains

```
BoxShape<vec_t> box(0.0, 1.0);
DomainDiscretization<vec_t> domain = box.discretizeBoundaryWithStep(dx);

for (auto i=0; i<o_t.size() ; ++i) {
    BallShape<vec_t> ball(o_c[i], o_r[i]);
    DomainDiscretization<vec_t> obstacle = ball.discretizeBoundaryWithStep(dx);
    obstacle.types() = o_t[i];
    domain.subtract(obstacle);
}

fill(domain, dx);
```





Examples :: Electromagnetic scattering

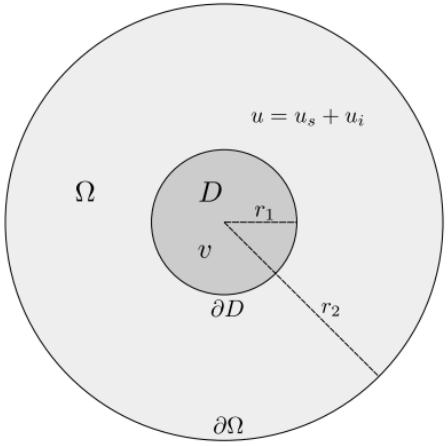
Governing PDEs

$$\nabla \cdot A \nabla v + \epsilon_r k^2 v = 0 \quad \text{in} \quad D$$
$$\nabla^2 u^s + k^2 u^s = 0 \quad \text{in} \quad \Omega \setminus D$$

with boundary conditions

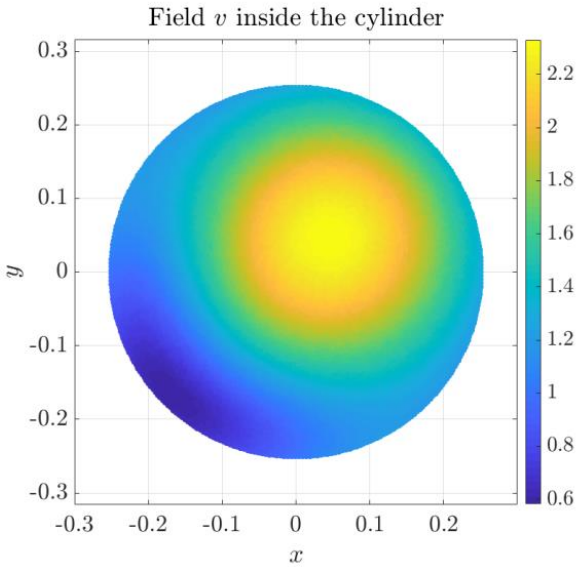
$$v - u^s = u^i \quad \text{on} \quad \partial D$$
$$\frac{\partial v}{\partial n_A} - \frac{\partial u^s}{\partial n} = \frac{\partial u^i}{\partial n} \quad \text{on} \quad \partial D$$
$$\frac{\partial u^s}{\partial n} + \left(ik + \frac{1}{2r_2} \right) u^s = 0 \quad \text{on} \quad \partial \Omega$$

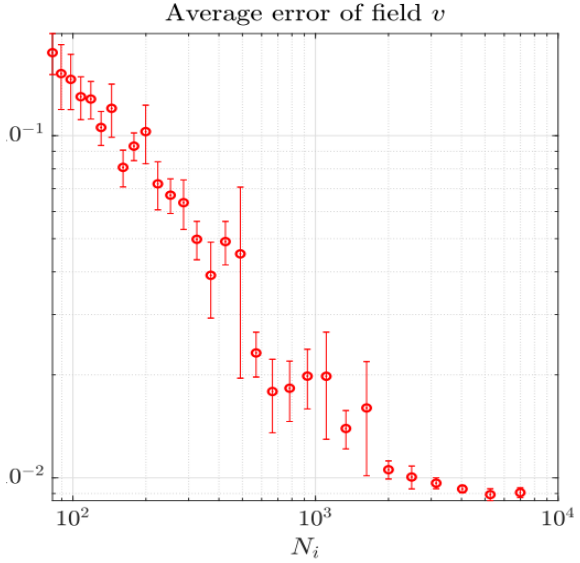
Anisotropic cylindrical scatterer. Let v be the (complex-valued) field inside the scatterer and $u = u^s + u^i$ outside.

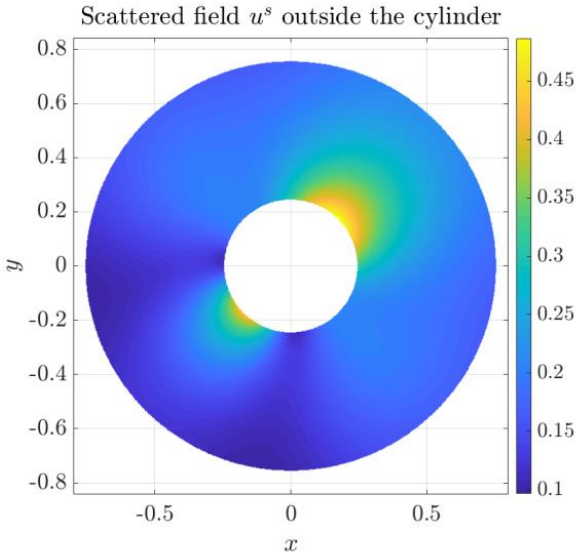


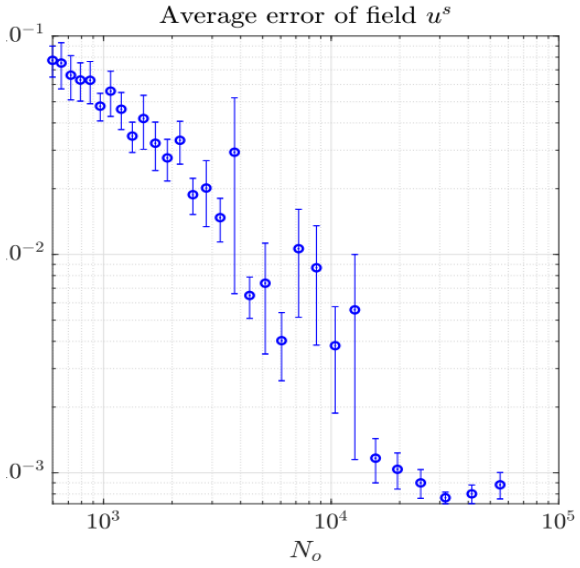

```
1     for (int i : inner.interior()) {
2         //Eq on the inner circle
3         mxx * op_inner.der2(i, 0, 0) + myy * op_inner.der2(i, 1, 1) +
4         (mxy + myx) * op_inner.der2(i, 0, 1)
5         + k0 * k0 * er * op_inner.value(i) = 0;
6     }
7     for (int i : inner.boundary()) {
8         //dirichlet, u_inside - u_scattered = u_incident <- incoming wave from t0 angle
9         double x = inner.pos(i, 0);
10        double y = inner.pos(i, 1);
11        double theta = atan2(y, x); // get phase
12
13        // in the upper side of the matrix
14        std::complex<double> incident = std::exp(1.0i * k0 * (x * std::cos(t0) + y *
std::sin(t0)));
15        op_inner.value(i) + (-1) * op_outer.value(inner_bnd[i], N_outer + i) = incident;
16
17        // Neumann boundary condition on the inside circle
18        auto norm = inner.normal(i);
19        double n0 = norm[0];
20        double n1 = norm[1];
21
22        // Anisotropic normal derivative
23        // (n0*mxx+n1*myx)*v_x + (n0*mxy+n1*myy)*v_y + du/dn = d(incident)/dn
24        op_outer.neumann(inner_bnd[i], outer.normal(inner_bnd[i]))
25        + (n0 * mxx + n1 * myx) * op_inner.der1(i, 0, -N_outer + inner_bnd[i])
26        + (n0 * mxy + n1 * myy) * op_inner.der1(i, 1, -N_outer + inner_bnd[i])
27                = 1.0i * k0 * std::cos(theta - t0) *
28                  std::exp(1.0i * k0 * (x * std::cos(t0) + y * std::sin(t0)));
29    }
30    for (int i : outer.interior()) {
31        // wave equation for the outer region
32        op_outer.lap(i) + k0 * k0 * op_outer.value(i) = 0;
33    }
34    for (int i : outer_bnd) {
35        // Sommerfeld boundary condition
36        op_outer.neumann(i, outer.normal(i)) + (k0 * 1.0i + 1 / (2 * r2)) *
op_outer.value(i) = 0.0;
37    }
```

Examples :: Electromagnetic scattering









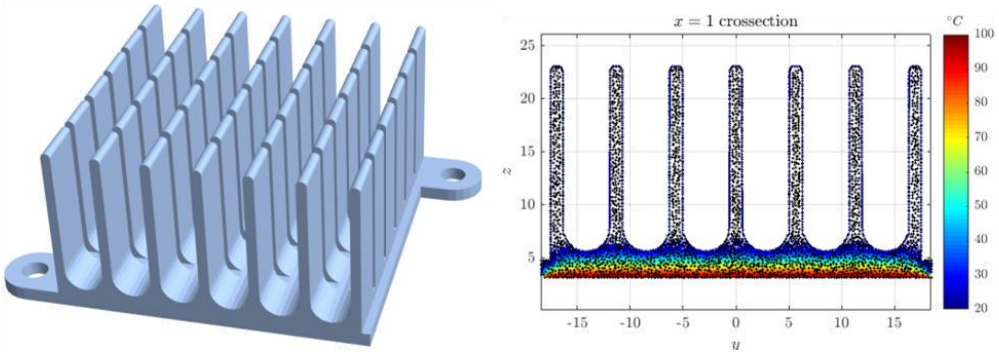
Examples: heat transfer in complex domain

Steady state heat equation:

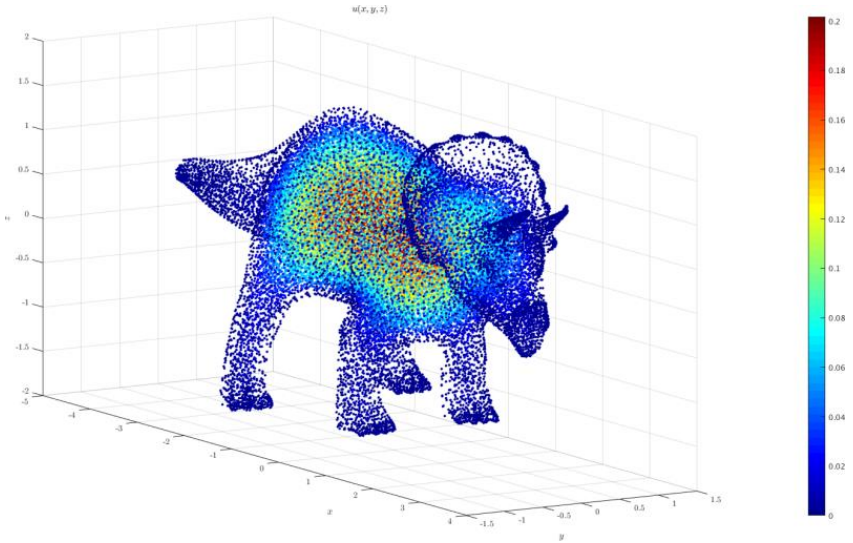
$$-\alpha \nabla^2 u = q,$$

where q is the volumetric heat source and α is thermal diffusivity.

Heat sink created from the Mathematica's heat sink model.



Triceratops created from the Mathematica's triceratops model.



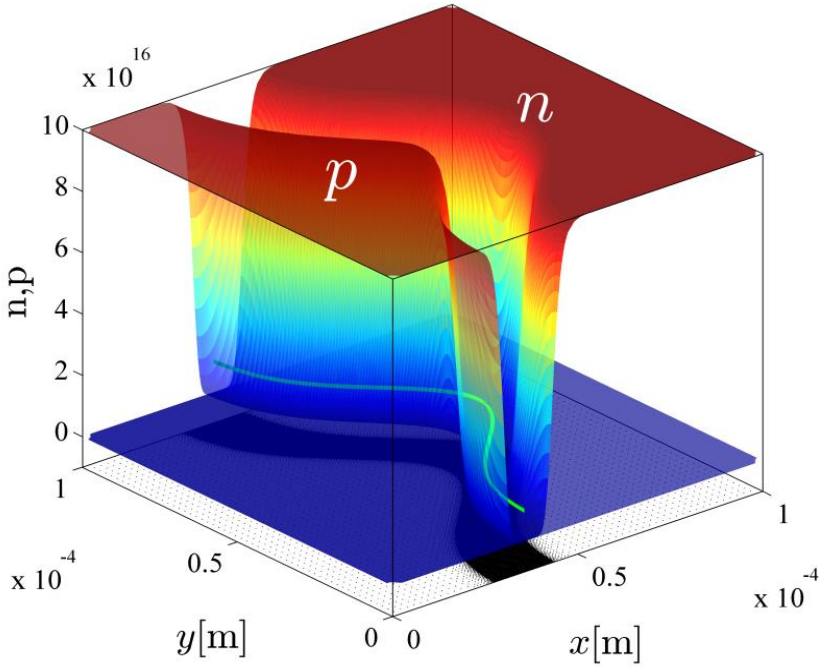
Examples: Simulation of PN junction

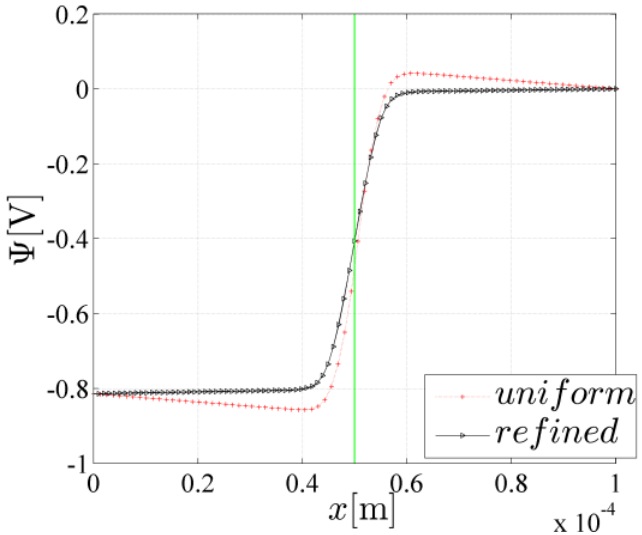
$$\nabla^2\Psi = -\frac{q}{\varepsilon}\left(p - n + D\right)$$

$$D_n\nabla^2 n - \mu_n\left(\nabla n\cdot\nabla\Psi + n\nabla^2\Psi\right) = 0$$

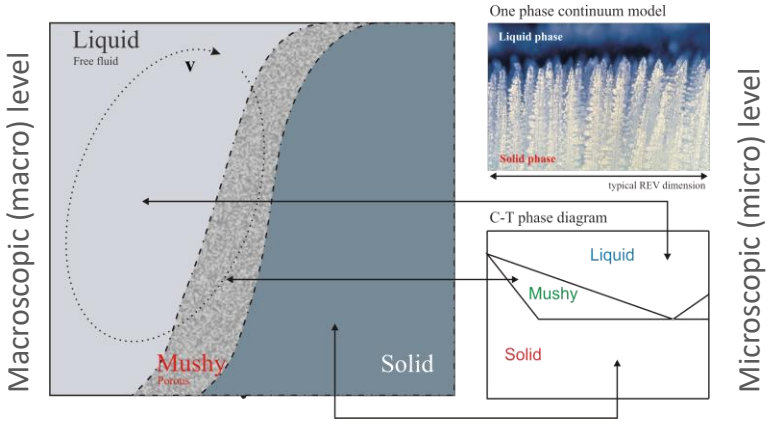
$$D_p\nabla^2 p + \mu_p\left(\nabla p\nabla\Psi + p\nabla^2\Psi\right) = 0$$

After joining the P and N doped semiconductors, the electrons and holes start to diffuse. The diffusing carriers leave charged ions behind, which induce the electric field that counteracts the diffusion. In the equilibrium, the junction and its local surrounding is depleted of all carriers.





Examples: Solidification of binary alloy



Governing PDEs

$$\langle \mathbf{v} \rangle = f_L \mathbf{v}_L$$

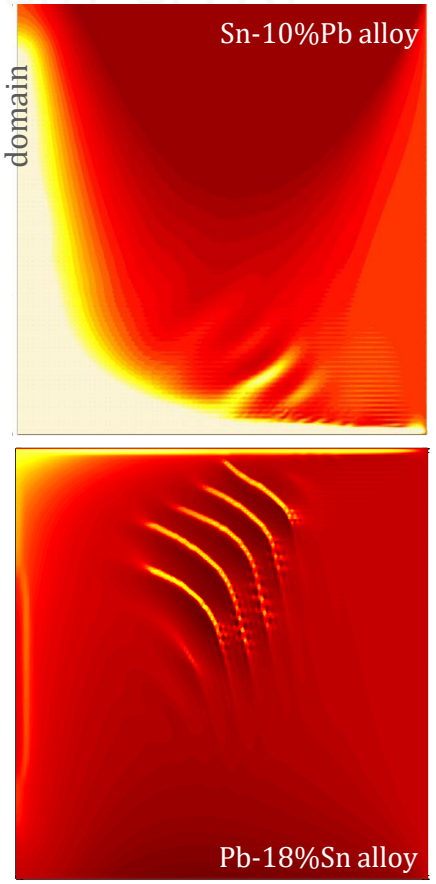
$$\nabla \cdot \langle \mathbf{v} \rangle = 0$$

$$\rho \frac{\partial \langle \mathbf{v} \rangle}{\partial t} + \frac{\rho}{f_L} \nabla \cdot \left(\langle \mathbf{v} \rangle \langle \mathbf{v} \rangle \right) = - f_L \nabla P + \nabla \cdot \left(\mu \nabla \langle \mathbf{v} \rangle \right) - f_L \frac{\mu}{K} \langle \mathbf{v} \rangle + f_L \mathbf{b}$$

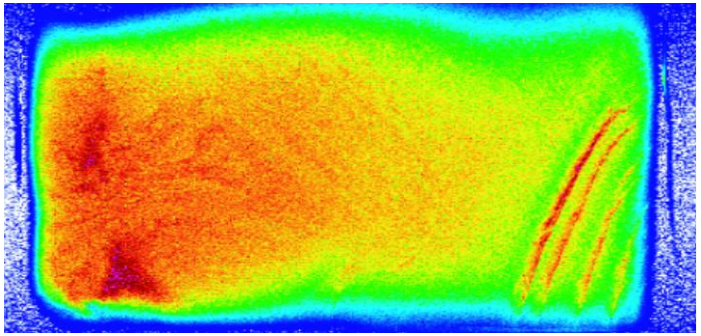
$$\rho \frac{\partial \langle h \rangle}{\partial t} + \rho \langle \mathbf{v} \rangle \cdot \nabla \langle h \rangle = \nabla \cdot \left(\lambda \nabla T \right)$$

$$\frac{\partial \langle C \rangle}{\partial t} + \langle \mathbf{v} \rangle \cdot \nabla C_L = 0$$

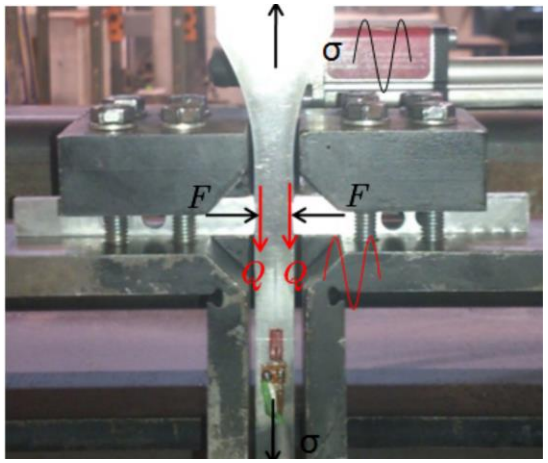
material composition after solidification in 5x6 cm domain



Macrosegregation map measured in a similar case experiment (Sn-5%Pb) in 10x6 cm
Yves Fautrelle et al., SIMAP - EPM, Grenoble



Fretting Fatigue simulation



The project is funded by:

- Research Foundation – Flanders – (FWO)

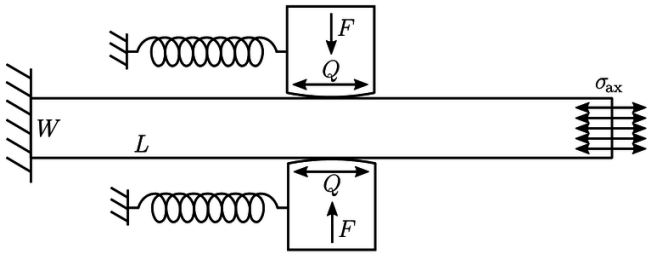
fwo Opening new horizons

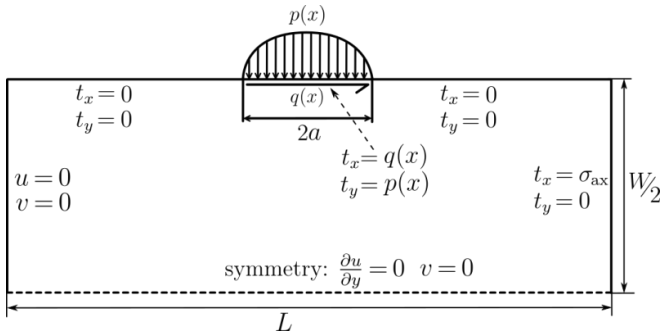
- The Luxembourg National Research Fund (FNR)

Luxembourg National Research Fund

- Slovenian Research Agency (ARRS)

Slovenian Research Agency





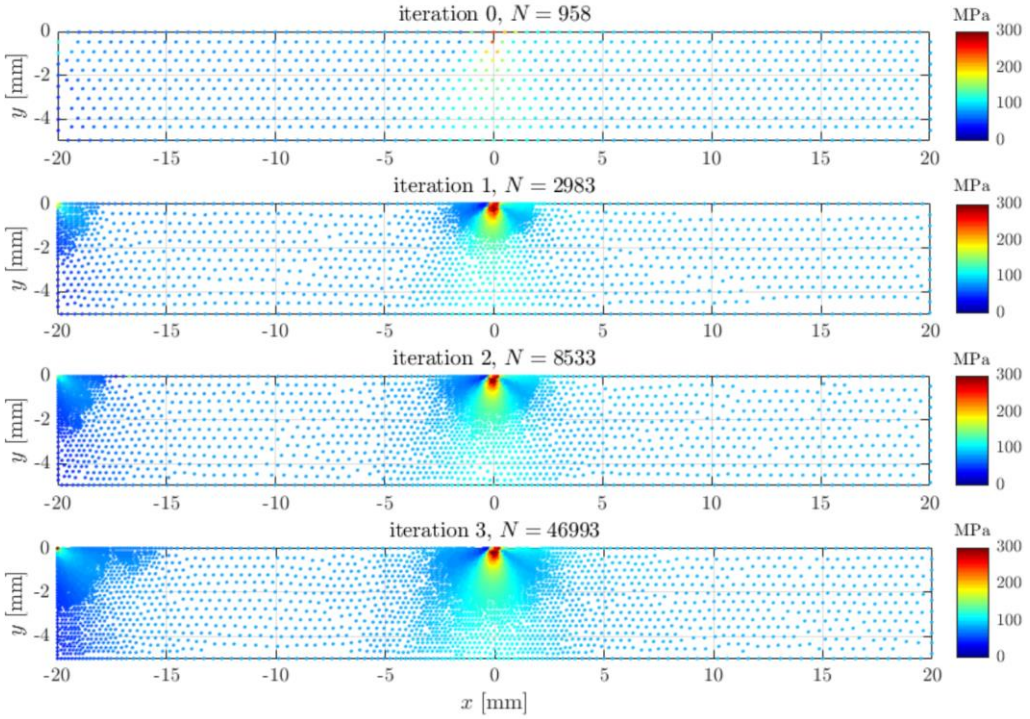
$$(\lambda+\mu)\nabla(\nabla\cdot\mathbf{u})+\mu\nabla^2\mathbf{u}=\mathbf{b}$$

The goal of the project was to determine stress distribution within the specimen as fast as possible in order to simulate fretting fatigue and predict the crack initiation and propagation.

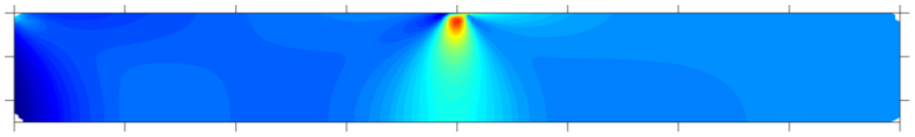
Fretting Fatigue simulation

Aggressive adaptivity

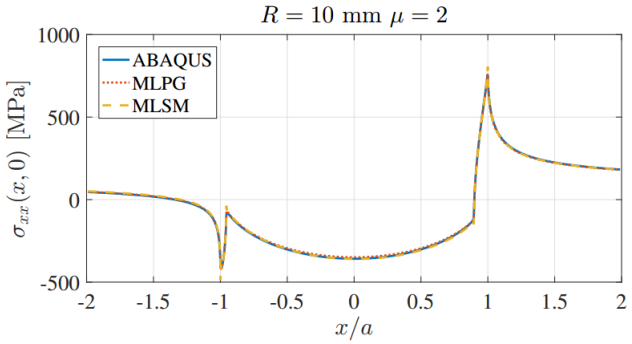
smallest internodal distance is 2^{17} times smaller than the initial one.

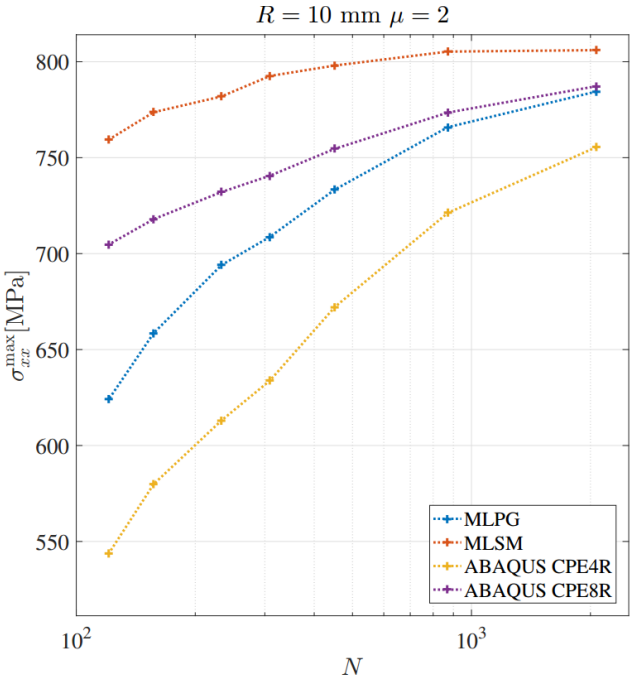


...



Final solution in terms of von Mises stress





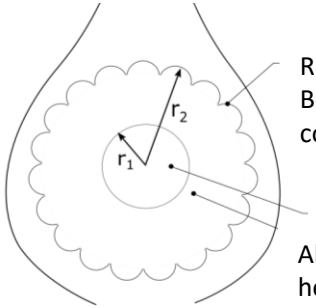
SIMULATION OF OVERHEAD POWER LINE COOLING

$$\rho^a \frac{\partial \mathbf{v}}{\partial t} + \rho^a \nabla \cdot (\mathbf{vv}) = -\nabla P + \nabla \cdot \left(\mu^a \nabla \mathbf{v}\right) + \mathbf{b}$$

$$\nabla \cdot \mathbf{v} = 0$$

$$\rho^a c_p^a \frac{\partial T^a}{\partial t} + \rho^a c_p^a \nabla \cdot (T^a \mathbf{v}) = \nabla \cdot \left(\lambda^a \nabla T^a\right)$$

$$\mathbf{b} = \rho_{ref}\left[1 - \beta_T (T^a - T_{\text{ref}})\right]\mathbf{g}$$



Radiation Boundary condition

$$q_R = \sigma_B \epsilon_s \left(T_{al}^4(r_2) - T_a^4\right)\left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]$$

$$q_S = \frac{\alpha_s I_T}{\pi}\left[\frac{\mathrm{W}}{\mathrm{m}^2}\right],$$

Steel part – heat transport

$$T^{al}(r_1) = T^{st}(r_1)$$

$$c_p^{st}\rho^{st}\frac{\partial T^{st}}{\partial t} = \lambda^{st}\nabla^2 T^{st}$$

Aluminium part – heat transport and heat generation

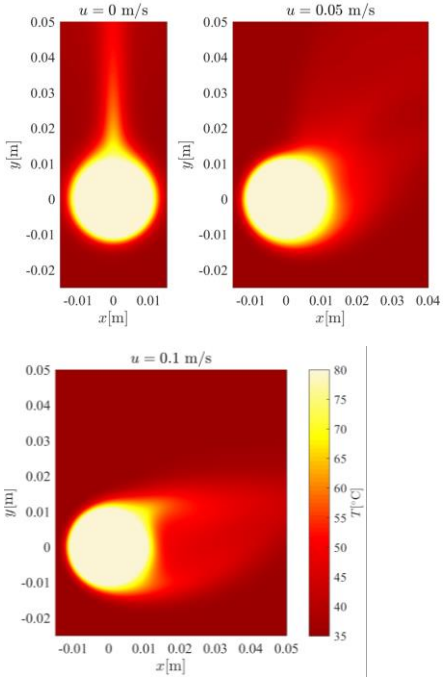
$$c_p^{al}\rho^{al}\frac{\partial T^{al}}{\partial t} = \lambda^{al}\nabla^2 T^{al} + q_J$$

$$q_j = \frac{I^2 R\left(T^{al}\right)}{S^{al}}\left[\frac{\mathrm{W}}{\mathrm{m}^3}\right],$$

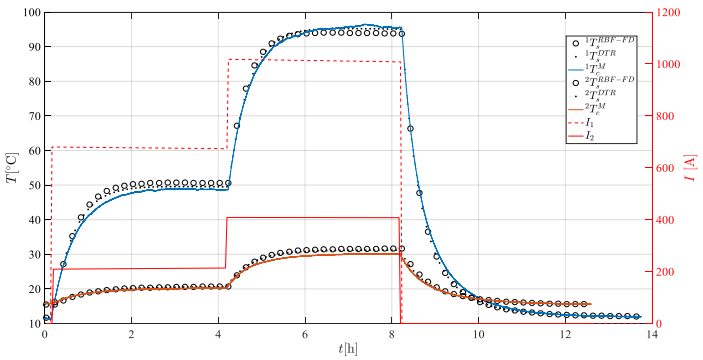
$$\lambda^{al}\left.\frac{\partial T^{al}}{\partial \mathbf{n}}\right|_{r_2} - \lambda^a \left.\frac{\partial T^a}{\partial \mathbf{n}}\right|_{r_2} = q_s + q_r$$

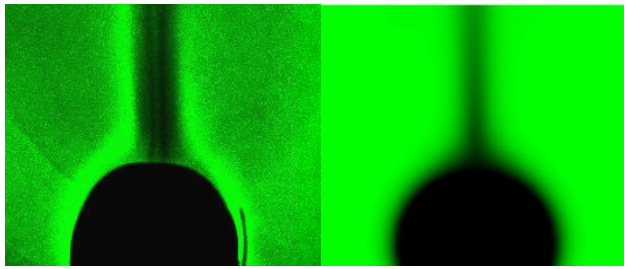
$$T^{al}(r_2) = T^a(r_2)$$

$$\lambda^{al}\left.\frac{\partial T^{al}}{\partial \mathbf{n}}\right|_{r_1} = \lambda^{st}\left.\frac{\partial T^{st}}{\partial \mathbf{n}}\right|_{r_1}$$



RBF-FD – numerical simulation, M - measurement





Result of Schlieren photography

Simulated temperature

Experimental setup for measurements of conductor temperature

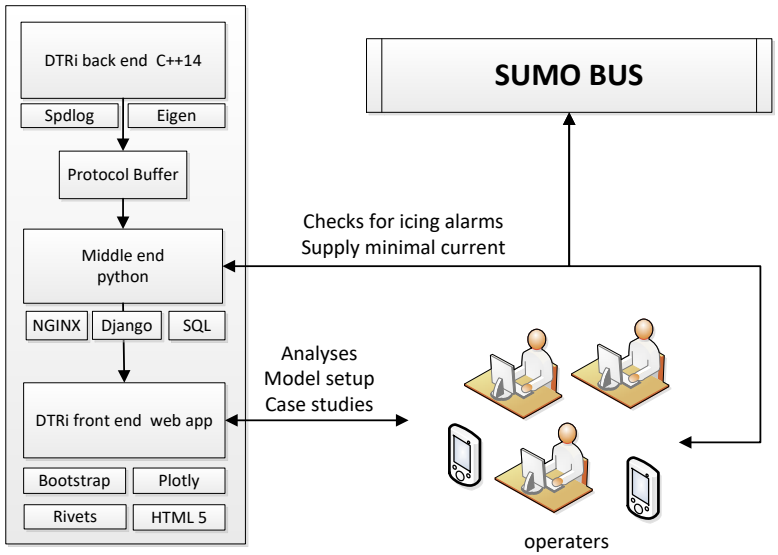
The project is funded by:

- ELES: transmitting energy, maintaining balance.

ELES

Dynamic thermal rating operational module

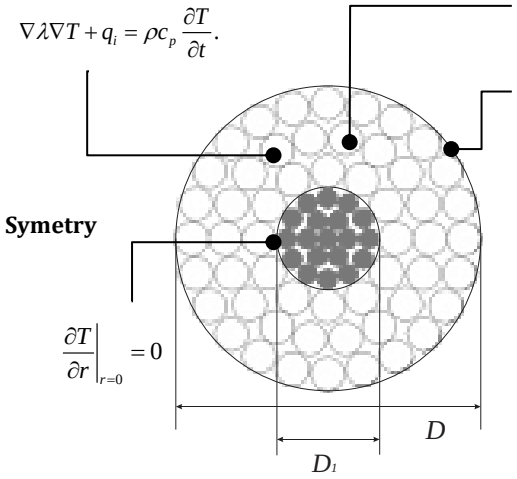
DTR*i*



Extension and simplification of the model

Heat transfer within the conductor

$$\nabla \lambda \nabla T + q_i = \rho c_p \frac{\partial T}{\partial t}.$$



Heat generation

$$Q_J = I^2 R(T) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$$

Heat exchange with surrounding

CONVECTION

$$Q_C = -\pi D h (T_s - T_a) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$$

RADIATION

$$Q_R = -\pi D \sigma_B \epsilon_s (T_s^4 - T_a^4) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$$

SOLAR

$$Q_S = \alpha_s I_T D \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$$

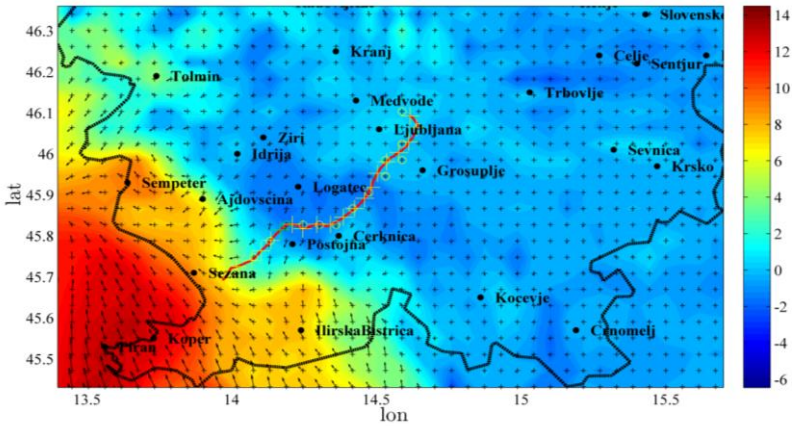
IMPINGING

$$Q_{IM} = CED c_w \frac{d\rho_I}{dt} (T_S - T_I) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$$

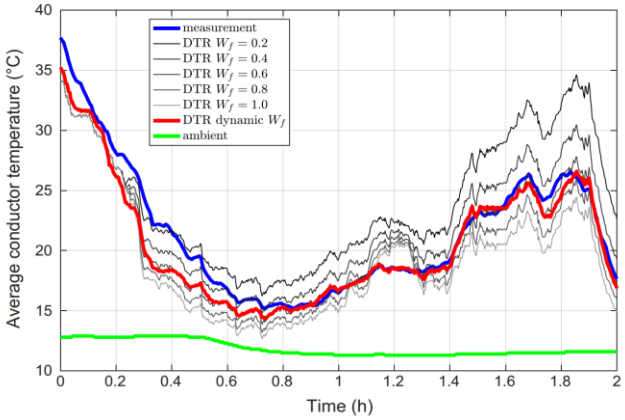
EVAPORATION

$$Q_E = -h \frac{\epsilon L_e A_w}{c_p} \left(\frac{e_s - e_a}{p}\right) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$$

Coupling with ALADIN model

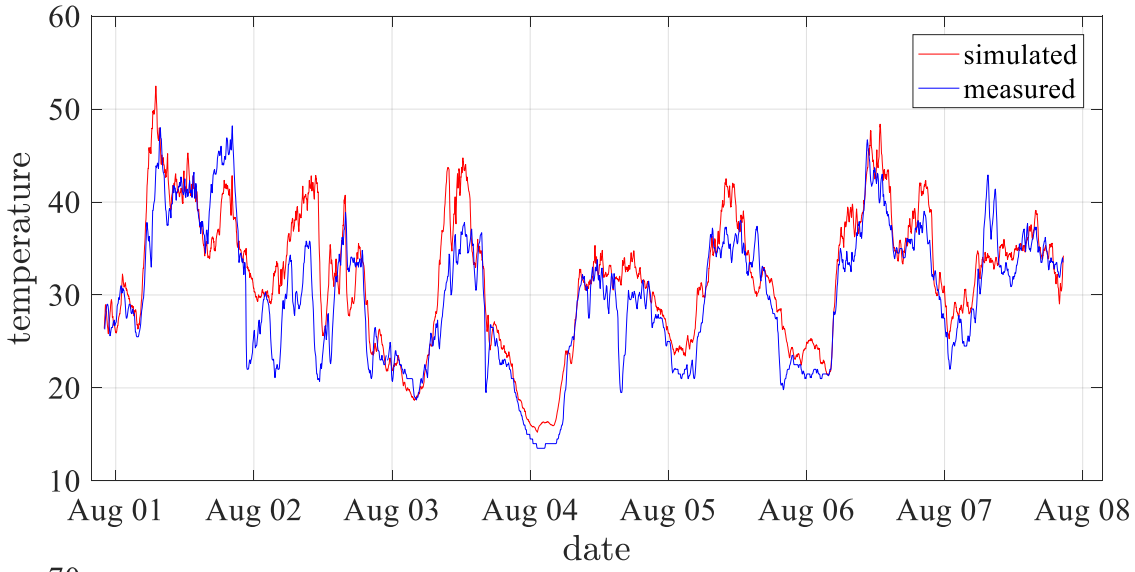


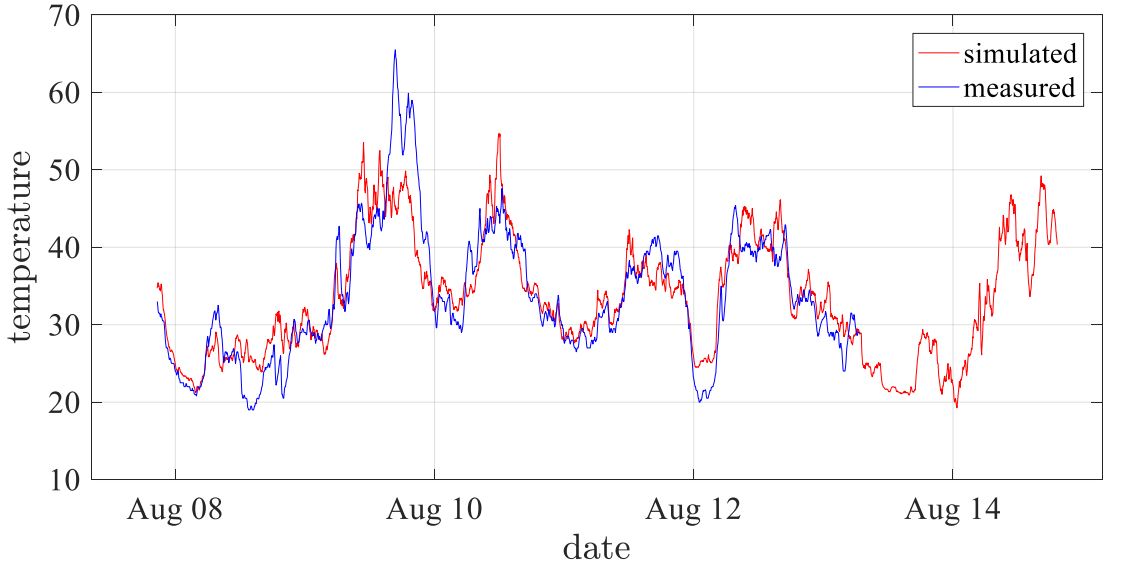
Validation



Dynamic thermal rating operational module

Comparison of simulated and measured line temperature at Podlog power line in August 2019.





Conclusions

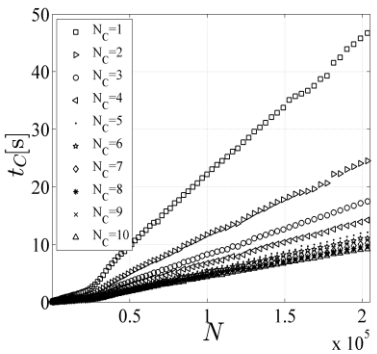
Medusa

- tested modular open source library for mesh-free simulations
- dimension independent
- explicit transformation of equations into code
- minimal overheads due to the introduced abstractions

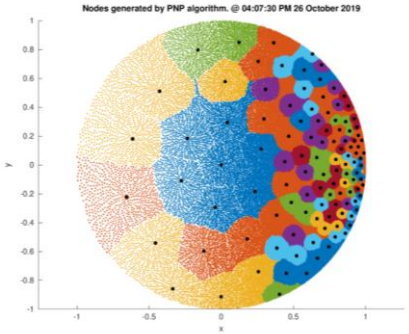
open source meshless project
Medusa: Coordinate Free Mehless Method implementation (MM)
https://gitlab.com/e62Lab/medusa
http://e6.ijs.si/medusa/

Future work

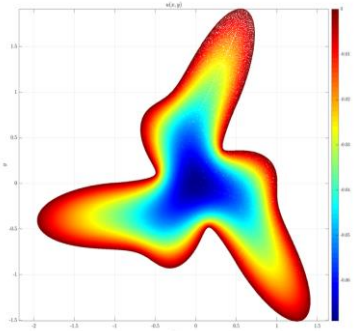
- Further development of features for solving different problems
- Parallelization of fill algorithm
- Domain decomposition
- Discretization of surfaces – coupling of Meshfree and CAD
- Solve more problems starting with Ionospheric model



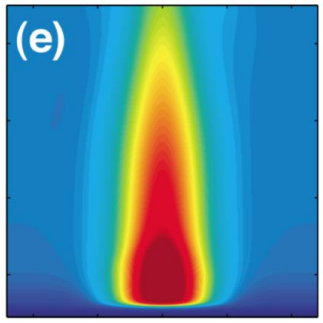
Parallelization



Domain decomposition



Parametric domains



Ionospheric model

Thank you for your attention