Implementation of strong form meshfree methods for solving PDEs

Gregor Kosec
Jožef Stefan Institute, Slovenia

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

**Computational modelling**

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

**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.
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**

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 + 2 PhD 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

\[
\rho \frac{\partial \mathbf{v}}{\partial t} + \rho \nabla \cdot (\mathbf{v} \mathbf{v}) = -\nabla P + \nabla \cdot (\mu \nabla \mathbf{v}) + \mathbf{b}
\]

\[
\rho \frac{\partial (c_p T)}{\partial t} + \rho \nabla \cdot (c_p T \mathbf{v}) = \nabla \cdot (\lambda \nabla T)
\]

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

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

Implement and execute it on different computer architectures

Traditional mesh-based methods FDM, FVM, FEM

Meshless methods MLPG, DAM, LRBFCM, SPH
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]


- 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.


- 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


- 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

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

\[
(Lu)(x_i) \approx \sum_{x_j \in N(x_i)} w_j^i u(x_j)
\]

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

\[
\begin{bmatrix}
\varphi(||x_{j1} - x_{j1}||) & \cdots & \varphi(||x_{jn} - x_{j1}||) \\
\vdots & \ddots & \vdots \\
\varphi(||x_{j1} - x_{jn}||) & \cdots & \varphi(||x_{jn} - x_{jn}||)
\end{bmatrix}
\begin{bmatrix}
w_{j1}^i \\
\vdots \\
w_{jn}^i
\end{bmatrix}
= 
\begin{bmatrix}
(L\varphi_{j1})(x_i) \\
\vdots \\
(L\varphi_{jn})(x_i)
\end{bmatrix}
\]

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

\[
\begin{bmatrix}
A & P \\
P^T & 0
\end{bmatrix}
\begin{bmatrix}
w \\
\lambda
\end{bmatrix}
= 
\begin{bmatrix}
\ell_{\varphi} \\
\ell_p
\end{bmatrix}
\]

\[
P = 
\begin{bmatrix}
p_1(x_1) & \cdots & p_s(x_1) \\
\vdots & \ddots & \vdots \\
p_1(x_n) & \cdots & p_s(x_n)
\end{bmatrix},
\ell_p = 
\begin{bmatrix}
(Lp_1)(x=x^*) \\
\vdots \\
(Lp_s)(x=x^*)
\end{bmatrix}
\]

Several strong form methods that can be described with this approach:

- **Finite Differences Method**
  \[ n=3, \ W(p)=1, \ b=\{1, x, x^2\} \text{ on regular nodes} \]

- **Local Radial Basis Function Collocation Method**
  \[ n, \ W(p)=1, \ b=\{\sqrt{1+(p/c)^2}\}, \ m=n \]

- **Generalized FDM - Finite pointset method**
  \[ n=20-50, \ W(p)=\exp(-p/s)^2, \ b=\{1, x, x^2, \ldots\} \]

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

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

...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

\[\begin{bmatrix} U_1 & V_1 \\ U_2 & V_2 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\]

Approximation of differential operator

\[(L\mathbf{u})(x_i) \approx \sum_{x_j \in N(x_i)} w_j^i \mathbf{u}(x_j)\]

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

- **Abstract PDE description**
  - fields, operators
  - manual discretization
  - conventional discretization
  - automatic discretization (library)

- **Discrete description**
  - arrays, indexes

- **Abstract program**

- **Domain definition**

- **Node positioning algorithm**

- **Position vector**
  - Defines dimensionality

- **Type of weight function**

- **Type of basis functions**

- **Support size**
  - size_t

- **Basis pool**
  - vector<function>

- **Weight function**
  - function W

- **Stencil selection Engine**

- **Differential Operator approximation**

- **Approximation object**

- **Approximation of governing PDEs**

- **Solution of the problem**

- **Explicit operators**

- **Implicit operators**

- **Vec_t laplace** (container<vec_t> &u, int node)

- **void laplace** (matrix_t &M, int node, scalar_t alpha)
**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)

[MediWiki](http://e6.ijs.si/medusa/wiki/index.php/Medusa)
\[ \nabla^2 u(x) = -d \pi^2 \prod_{i=1}^{d} \sin(\pi x_i) \]

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

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

\[ \Omega = \left\{ x, \left\| x - \frac{1}{2} \right\| < \frac{1}{2} \right\}, \]

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

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

// 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)) = neumann_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);
ScalarFielddd u = solver.solve(rhs);
Examples :: basic tests

\[ m = 2 \quad m = 4 \quad m = 6 \quad m = 8 \]

\[ d = 1 \quad d = 2 \quad d = 3 \]

\[ \epsilon_\infty \]

\[ \frac{1}{\sqrt{N}} \quad \frac{2}{\sqrt{N}} \quad \frac{3}{\sqrt{N}} \]

\[ k = -1.89 \quad k = -4.01 \quad k = -6.03 \quad k = -7.67 \]

\[ k = -1.91 \quad k = -3.98 \quad k = -6.33 \quad k = -7.94 \]

\[ O(h^8) \]
Global sparse matrices of discretized PDE and theirs spectra.

1D

\[ \lambda_{\text{max}} = -9.9 + i0.0 \]

2D

\[ \lambda_{\text{max}} = -23.9 + i0.0 \]

3D

\[ \lambda_{\text{max}} = -39.8 + i0.0 \]
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

Analysis of low level CPU counters

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.
Examples :: performance tests

Comparison with FreeFem++

2D

3D
**Examples :: Adaptive linear elasticity**

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

```java
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

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

```java
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
\[ \nabla \cdot \mathbf{v} = 0 \]

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

\[ \rho \frac{\partial (c_p T)}{\partial t} + \rho \mathbf{v} \cdot (c_p T \mathbf{v}) = \nabla \cdot (\lambda \nabla T) \]

```cpp
    for (int step = 0; step <= 0.5 * steps; ++step) {
        // Explicit Navier-Stokes computed on whole domain, including boundaries
        // without pressure
        for (int c = all) {
            v_2[c] = v_1[c] + 0.5 * dt *
                0.5 * rho / 0.5 * op.lap(v_1, c)
                - op.grad(v_1, c) * v_1[c]
                + 0.5 * (1 - 0.5 * beta * (T_1[c] - T_{\text{ref}}));
        }
        // Pressure correction
        VecXd rhs_pressure(N + 1, 0); // Note N+1, +1 stands for regularization equation
        rhs_pressure(N) = 0; // = 0 part of the regularization equation
        for (int i = interior) rhs_pressure(c) = 0.5 * rho / 0.5 * op.div(v_2, c);
        for (int i = boundary) rhs[boundary] = 0; // force boundary conditions
        VecXd solution = solver_p.solve(rhs_pressure);
        alpha = solution[N];
        VecXd p_c = solution.head(N);
        for (int i = interior) v_2[c] -= 0.5 * (1 - 0.5 * beta * (T_1[c] - T_{\text{ref}}));
        v_2[boundary] = 0; // force boundary conditions
        // heat transport
        for (auto c = interior) {
            T2[c] = T1[c] + 0.5 * dt * 0.1 * 0.5 * rho / op.lap(T1, c) -
                0.5 * op.grad(T1, c);
        }
        for (auto c = top) T2[c] = op.neumann(T2, c, vec_t{0, -1}, 0.0);
        for (auto c = bottom) T2[c] = op.neumann(T2, c, vec_t{0, 1}, 0.0);
    }
```
Temperature contour and velocity quiver plots of solutions in 2D and 3D domains

Comparison of results with published data

<table>
<thead>
<tr>
<th>Ra</th>
<th>$v_{max}(x, 0.5)$</th>
<th>$x$</th>
<th>$u_{max}(0.5, y)$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^6$</td>
<td>0.2628</td>
<td>0.2604</td>
<td>0.2627</td>
<td>0.037</td>
</tr>
<tr>
<td>$10^7$</td>
<td>0.2633</td>
<td>0.2580</td>
<td>0.2579</td>
<td>0.022</td>
</tr>
<tr>
<td>$10^8$</td>
<td>0.2557</td>
<td>0.2587</td>
<td>0.2487</td>
<td>0.010</td>
</tr>
<tr>
<td>3D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^5$</td>
<td>0.2295</td>
<td>0.2218</td>
<td>0.2252</td>
<td>0.850</td>
</tr>
<tr>
<td>$10^6$</td>
<td>0.2564</td>
<td>0.2556</td>
<td>0.2588</td>
<td>0.961</td>
</tr>
</tbody>
</table>
Examples :: Natural convection

Generalisation to irregular domains

```cpp
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

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

with boundary conditions

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

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

---

```cpp
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  double complex incident = std::exp(1.0i * k0 * (x * std::cos(theta) + y *
13      std::sin(theta)));
14  op_inner.value(i) + (-1) * op_outer.value(inner_bnd[i], N_outer + 1) = incident;
15  // Neumann boundary condition on the inside circle
16  auto n0 = norm[n0];
17  double n0 = norm[n0];
18  double n1 = norm[n1];
19  // Anisotropic normal derivative
20  ((n0*mxx+n1*myx)*v_x + (n0*mxy+n1*myy)*v_y + du/dn = d(incident)/dn
21  op_outer.neumann(inner_bnd[i], outer.normal(inner_bnd[i])
22  + (n0 * mxx + n1 * myx) * op_iner.der1(i, 0, -N_outer + inner_bnd[i])
23  + (n0 * mxy + n1 * myy) * op_iner.der1(i, 1, -N_outer + inner_bnd[i])
24  = 1.0i * k0 * std::cos(theta - t0) *
25      std::exp(1.0i * k0 * (x * std::cos(t0) + y * std::sin(t0)));
26 }
27 for (int i : outer.interior()) {
28  // wave equation for the outer region
29  op_outer.lap(i) + k0 * k0 * op_outer.value(i) = 0;
30 }
31 for (int i : outer_bnd) {
32  // Sommerfeld boundary condition
33  op_outer.neumann(i, outer.normal(i)) + (k0 * 1.0i + 1 / (2 * r2)) *
34  op_outer.value(i) = 0.0;
35 }```

Examples :: Electromagnetic scattering
Examples: heat transfer in complex domain

Steady state heat equation:

\[-\alpha \nabla^2 u = q_x\]

where \( q \) is the volumetric heat source and \( \alpha \) 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

\[ \nabla^2 \Psi = -\frac{q}{\varepsilon} (p - n + D) \]

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

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

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

**Macroscopic (macro) level**

- One phase continuum model
- Typical RSM simulation

**Microscopic (micro) level**

- C-T phase diagram
- Typical RSM simulation

Governing PDEs:

\[
\langle \mathbf{v} \rangle = f_L \mathbf{v}_L \\
\nabla \cdot \langle \mathbf{v} \rangle = 0 \\
\rho \frac{\partial \langle \mathbf{v} \rangle}{\partial t} + \rho \mathbf{f} \cdot \nabla \langle \mathbf{v} \rangle = -f_L \mathbf{P} + \nabla \cdot (\mu \nabla \langle \mathbf{v} \rangle) - f_L \frac{\mu}{K} \langle \mathbf{v} \rangle + f_L \mathbf{b} \\
\rho \frac{\partial \langle h \rangle}{\partial t} + \rho \langle \mathbf{v} \rangle \cdot \nabla \langle h \rangle = \nabla \cdot (\lambda \nabla T) \\
\frac{\partial \langle C \rangle}{\partial t} + \langle \mathbf{v} \rangle \cdot \nabla C_L = 0
\]

Macrosegregation map measured in a similar case experiment (Sn-5%Pb) in 10x6 cm domain

Yves Fautrelle et al., SIMAP - EPM, Grenoble
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.
Aggressive adaptivity
smallest internodal distance is $2^{-7}$ times smaller than the initial one.

Final solution in terms of von Mises stress
SIMULATION OF OVERHEAD POWER LINE COOLING

\[ \rho \frac{\partial \mathbf{v}}{\partial t} + \rho \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla P + \nabla \cdot (\mu \nabla \mathbf{v}) + \mathbf{b} \]

\[ \nabla \cdot \mathbf{v} = 0 \]

\[ \rho c_p \frac{\partial T}{\partial t} + \rho c_p \mathbf{v} \cdot \nabla T = \nabla \cdot (\kappa \nabla T) \]

\[ \mathbf{b} = \rho c_p \left( 1 - \beta (T - T_{\text{ref}}) \right) \]

\[ q_R = \sigma_{\text{eff}} \left( T^R_{\text{in}}(r) - T^R_{\text{in}} \right) \left[ \frac{W}{m^2} \right] \]

\[ q_b = \frac{\alpha E}{\pi} \left[ \frac{W}{m^2} \right] \]

Steel part – heat transport

Aluminium part – heat transport and heat generation

\[ T^\text{al} (r) = T^\text{al} (r) \]

\[ T^\text{al} (r) = T^\text{al} (r) \]

Radiation Boundary condition

\[ A_{\text{al}} \frac{\partial T^\text{al}}{\partial n} - \lambda_{\text{al}} \frac{\partial T^\text{al}}{\partial n} = q_s + q_j \]

\[ \frac{I^2 R}{S^\text{al}} \left[ \frac{W}{m^3} \right] \]

\[ \lambda^\text{al} \frac{\partial T^\text{al}}{\partial n} = \lambda^\text{al} \frac{\partial T^\text{al}}{\partial n} \]

The project is funded by:

- ELES: transmitting energy, maintaining balance.
Dynamic thermal rating operational module

**DTRi**

- DTRi back end C++14
- Spdlog
- Eigen
- Protocol Buffer
- Middle end python
- NGINX
- Django
- SQL
- DTRi front end web app
- Bootstrap
- Plotly
- Rivets
- HTML5

**SUMO BUS**

- Checks for icing alarms
- Supply minimal current

Analyses
- Model setup
- Case studies

**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

\[ Q_i = I^2 R(T) \quad [W/m] \]

Heat exchange with surrounding

**Convection**

\[ Q_c = -\pi D h(T_i - T_s) \quad [W/m] \]

**Radiation**

\[ Q_r = -\pi D \sigma \epsilon (T_i^4 - T_s^4) \quad [W/m] \]

**Symmetry**

\[ \frac{\partial T}{\partial r} \bigg|_{r=0} = 0 \]

**Solar**

\[ Q_s = a_s I_r D \quad [W/m] \]

**Impinging**

\[ Q_{im} = C E D c p \frac{d \rho}{dt} (T_i - T_s) \quad [W/m] \]

**Evaporation**

\[ Q_e = -h \left( \frac{L_m A_c}{\rho c_p} \right) (e_i - e_s) \quad [W/m] \]

**Coupling with ALADIN model**

**Validation**

![Validation Chart]
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

**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

---

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

Thank you for your attention