Generic implementation of Meshless Local Strong Form Method

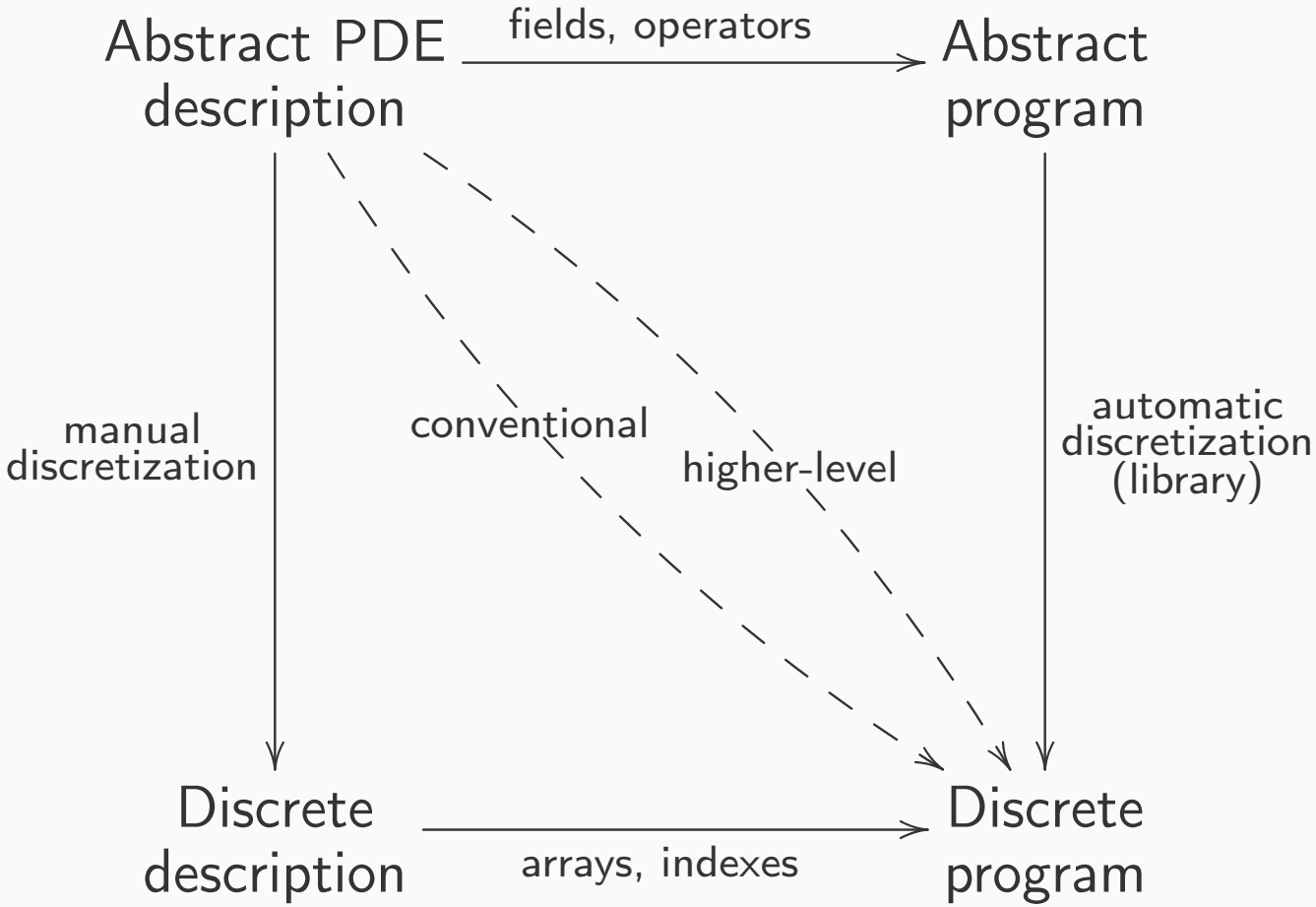
Jure Slak, Gregor Kosec

ECT 2018, 5. 9. 2018

"Jožef Stefan" Institute

Two approaches to programming

JSI



- Abstract programs are easier to understand and modify

A sample boundary value problem

JSI

$$u''(x) = \sin(x), \quad x \in [0, 1]$$
$$u(0) = 0, u'(1) = 0$$

Conventional:

```
vector<Triplet<scalar_t>> ts;
for (int i = 1; i < N-1; ++i) {
    ts.emplace_back(i, i, -2/h/h);
    ts.emplace_back(i, i-1, 1/h/h);
    ts.emplace_back(i, i+1, 1/h/h);
    rhs(i) = std::sin(i*h);
}
rhs(0) = 0;
ts.emplace_back(0, 0, 1);
ts.emplace_back(N-1, N-3, -1./2/h);
ts.emplace_back(N-1, N-2, 2/h);
ts.emplace_back(N-1, N-1, -3./2/h);
rhs(N-1) = 0;
M.setFromTriplets(ts.begin(), ts.end());
VectorXd solution = M.lu().solve(rhs);
```

High-level:

```
auto op = sh.implicitOperators(M, rhs);
for (int i : domain.interior()) {
    double x = domain.pos(i, 0);
    op.lap(i) = std::sin(x);
}
op.value(0) = 0;
op.neumann(N-1, {1}) = 0;
VectorXd solution = M.lu().solve(rhs);
```

General boundary value problem:

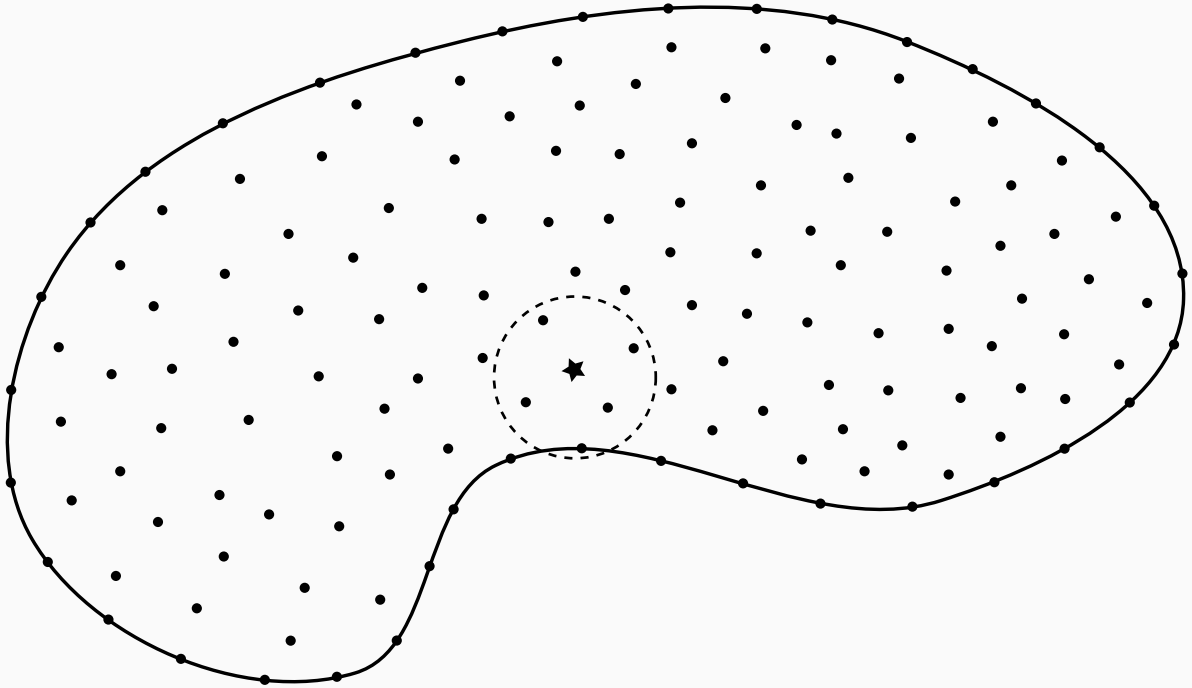
$$\mathcal{L}u = f, \text{ in } \Omega \tag{1}$$

$$\mathcal{R}u = g, \text{ on } \partial\Omega, \tag{2}$$

where $\Omega \subseteq \mathbb{R}^d$ is a domain, u is an unknown scalar or vector field, $\mathcal{L}$ and $\mathcal{R}$ are linear partial differential operators and f and g are known functions.

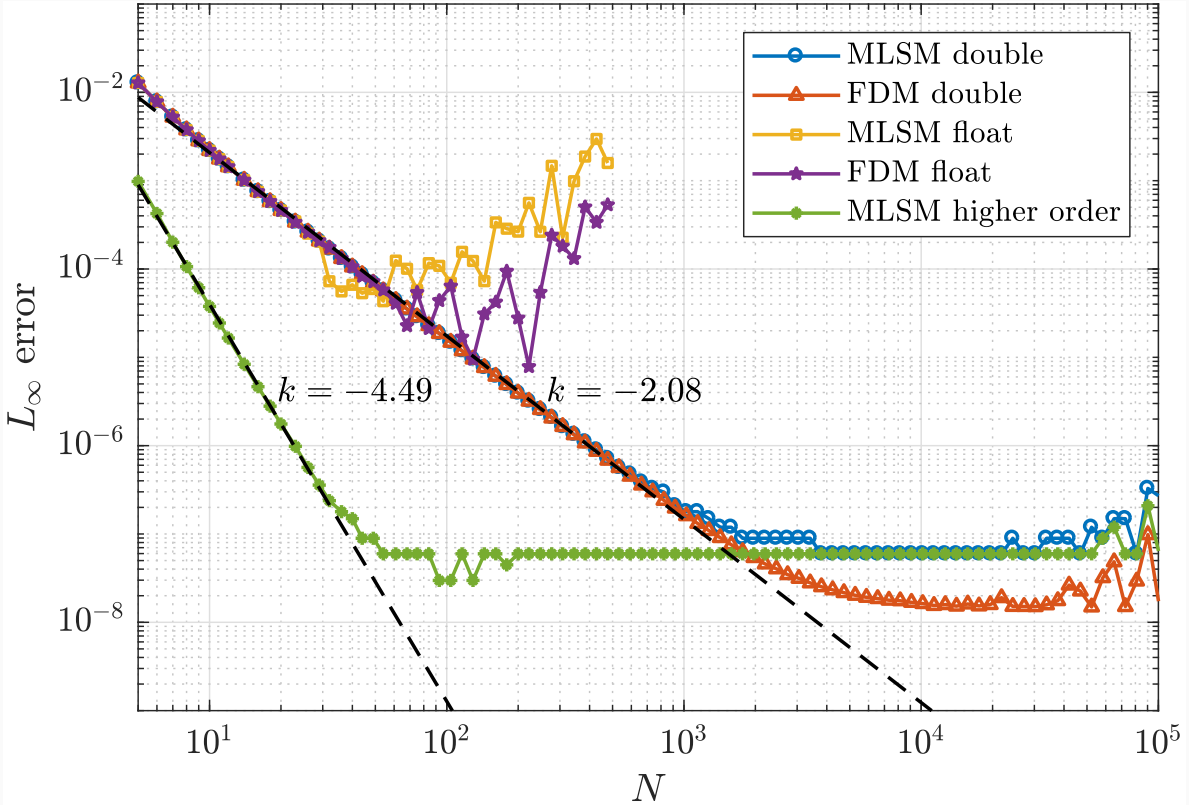
- Discretize by choosing N points in the domain.
- Some on the boundary and some in the interior.
- Find neighborhoods for all points.

Approximation of spatial partial differential operators:

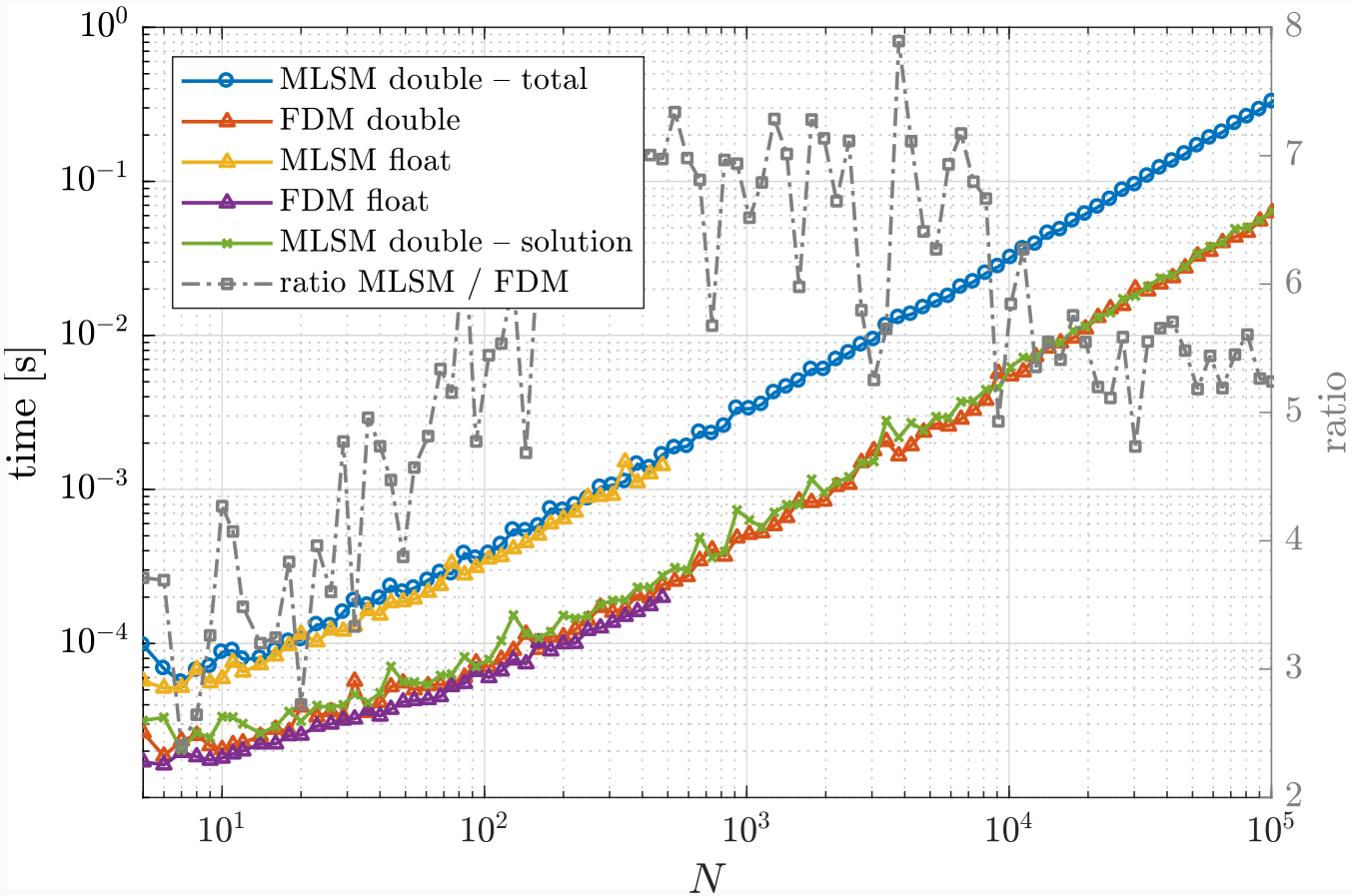


- Approximate $\mathcal{L}$ at a point p with function values in the neighborhood of p
- Example: $u''(x_i) = \frac{1}{h^2}u(x_{i-1}) - \frac{2}{h^2}u(x_i) + \frac{1}{h^2}u(x_{i+1})$
- $(\mathcal{L}u)p = \sum_{i=1}^{n} \chi_i u_i = \boldsymbol{\chi} \cdot \boldsymbol{u}$

Comparison of Meshless Local Strong Form method and Finite Difference Method: different orders and numerical precision



Comparison of Meshless Local Strong Form method and Finite Difference Method



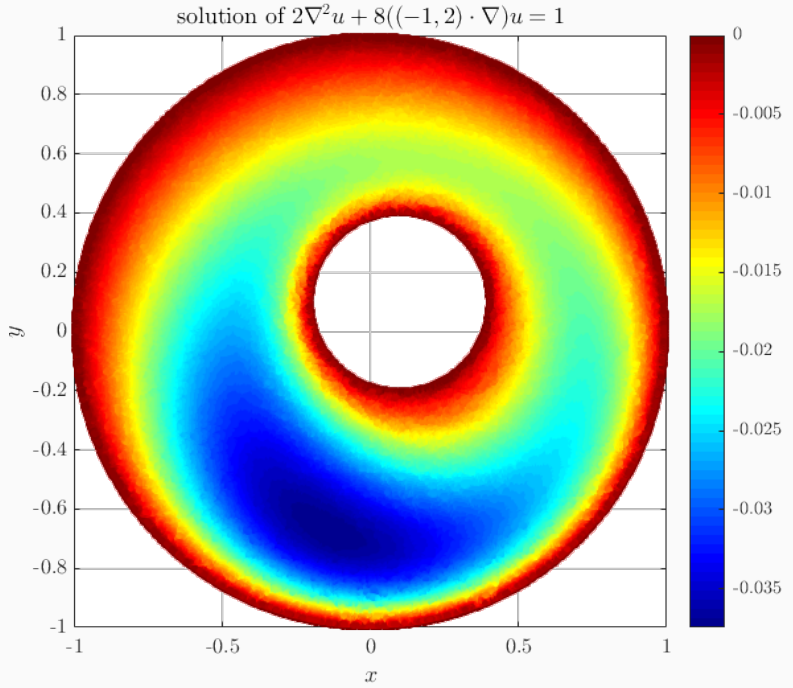
- Time difference due to shape precomputation.
- Shape computation is trivially parallelizable.

Steady state advection-diffusion:

$$8(-1,2)\cdot\nabla u + 2\nabla^2 u = 1 \qquad \text{in } \Omega \qquad (3)$$

$$u = 0 \qquad \text{on } \partial\Omega \qquad (4)$$

```
for (int i : domain.interior()) {
    8.0*op.grad(i, {-1.0, 2.0})
      + 2.0*op.lap(i) = 1.0;
}
for (int i : domain.boundary()) {
    op.value(i) = 0.0;
}
```

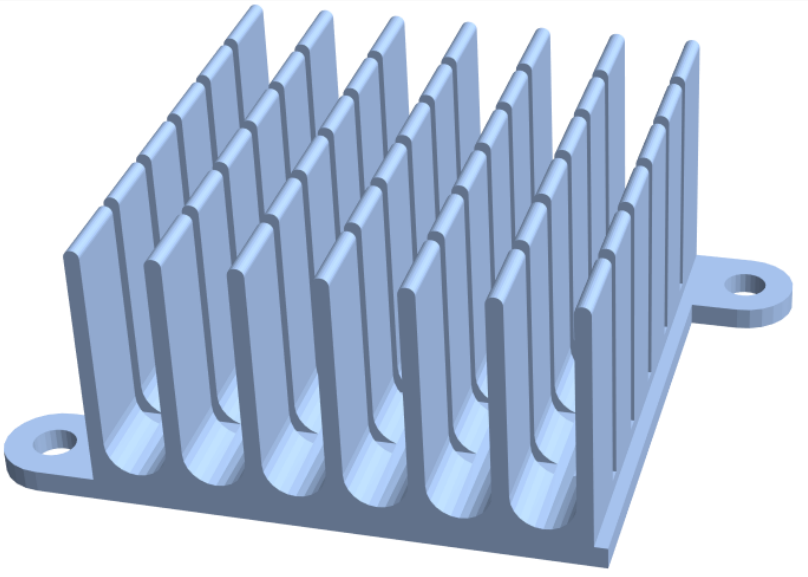


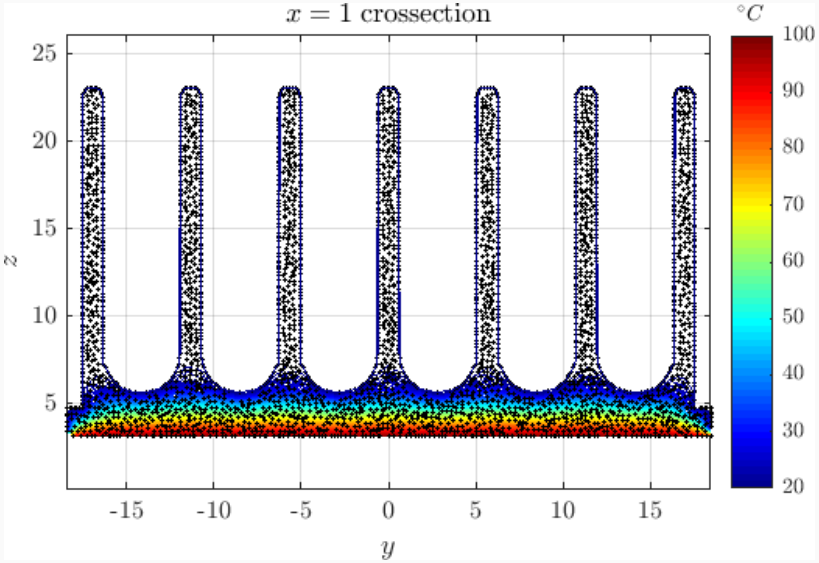
Steady state heat equation:

$$-\alpha \nabla^2 u = q, \tag{5}$$

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

Heatsink:





Setup: $q = 0$, bottom set to $100^\circ C$, remainder to $20^\circ C$.

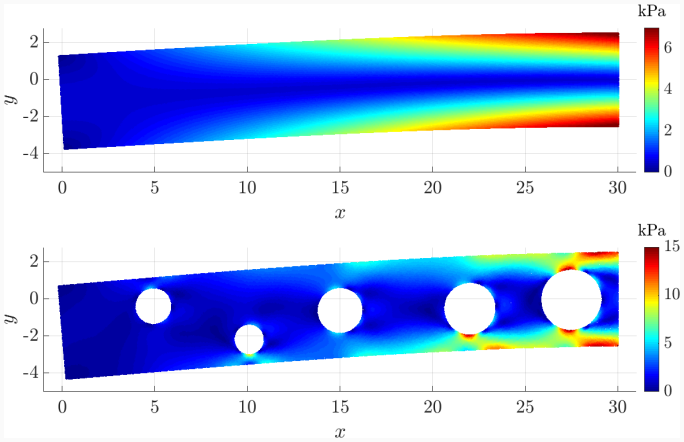
Lamé-Navier equation

$$(\lambda + \mu)\nabla(\nabla \cdot \vec{u}) + \mu\nabla^2\vec{u} = \vec{f} \tag{6}$$

describing displacements $\vec{u} = (u, v)$ and stresses σ.

Cantilever beam:

Beam bent at left end with a force P, along with "drilled" variant



$$u = \frac{Py\left(3D^2(\nu+1) - 4\left(3L^2 + (\nu+2)y^2 - 3x^2\right)\right)}{24EI}$$

$$v = -\frac{P\left(3D^2(\nu+1)(L-x) + 4(L-x)^2(2L+x) + 12\nu x y^2\right)}{24EI}$$

Cantilever beam

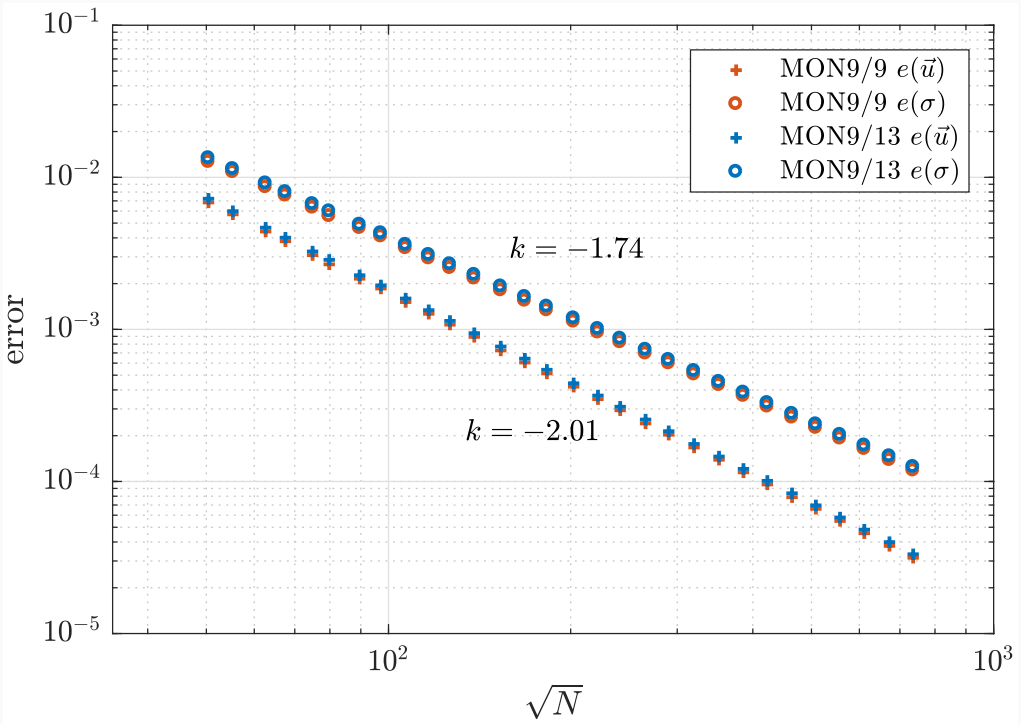
```
for (int i : domain.interior()) {
    (lam+mu)*op.graddiv(i) + mu*op.lap(i) = 0;
}
for (int i : domain.types() == BOTTOM) {
    op.traction(i lam, mu, {0, -1}) = 0;
}
for (int i : domain.types() == TOP) {
    op.traction(i, lam, mu, {0, 1}) = 0;
}
for (int i : domain.types() == LEFT) {
    double 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() == RIGHT) {
    double y = domain.pos(i, 1);
    op.value(i) = {(P*y*(3*D*D*(1+v)-4*(2+v)*y*y))/ 24.*E*I),
                   -(L*v*P*y*y) / (2.*E*I))};
}
```

Cantilever beam - convergence

Errors are measured using analogues of the ℓ_∞ norms

$$e(\vec{u}) = \frac{\max_{x\in X}\{\max\{|u(x)-\hat{u}(x)|,|v(x)-\hat{v}(x)|\}\}}{\max_{x\in X}\{\max\{|u(x)|,|v(x)|\}\}},$$

$$e(\sigma) = \frac{\max_{x\in X}\{\max|\sigma(x)-\hat{\sigma}(x)|}{\max_{x\in X}\{\max|\sigma(x)|\}}.$$



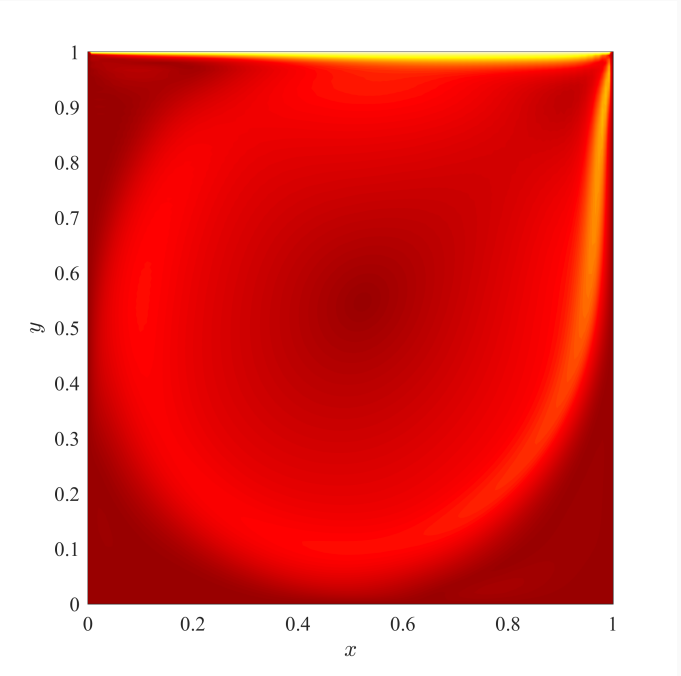
The Navier-Stokes equations for incompressible flow are

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\boldsymbol{v} = -\frac{1}{\rho}\nabla P + \frac{\mu}{\rho}\nabla^2 \boldsymbol{v}$$

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

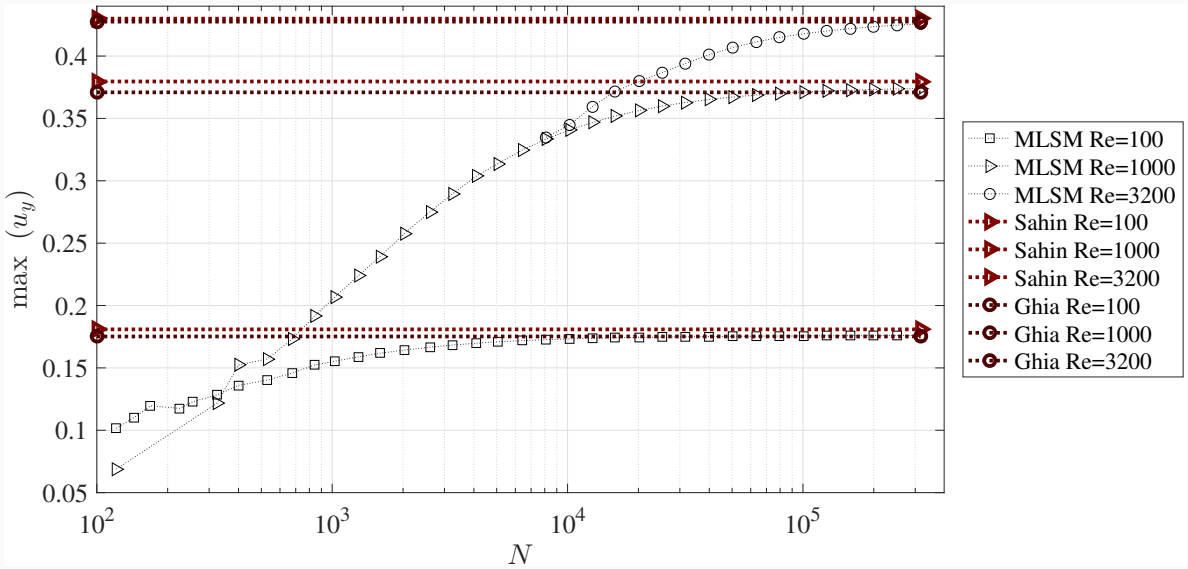
where $\boldsymbol{v} = (u, v)$ is the flow velocity, μ is the viscosity, ρ is the fluid density and P is the pressure.

Lid-driven cavity: Upper boundary has $\boldsymbol{v} = (0, 1)$, other boundaries fixed.

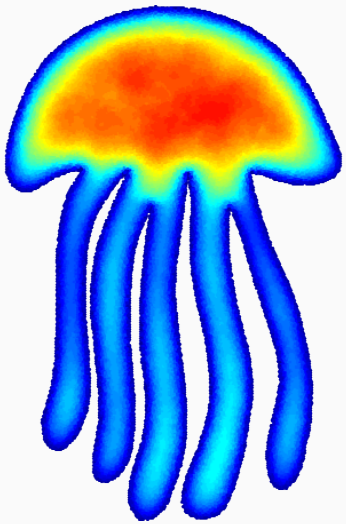


```
for (int step = 0; step <= max_steps; ++step) {
    for (int i : interior) {
        v2[i] = v1[i] + dt * (- 1/rho * op.grad(P1, i) + mu/rho *
                                op.lap(v1, i) - op.grad(v1, i) * v1[i]);
    }
    for (auto i : interior) {
        P2[i] = P1[i] - dl * dt * rho * op.div(v2, i) +
                    dl2 * dl * dt * dt * op.lap(P1, i);
    }
    v1.swap(v2);
    P1.swap(P2);
}
```

Comparison to reference data:



All computations were done using the open source Medusa library.



Medusa

Coordinate Free Mehless Method implementation

`http://e6.ijs.si/medusa/`

Thank you for your attention!

Acknowledgments: FWO Lead Agency project: G018916N Multi-analysis of fretting fatigue using physical and virtual experiments, and the ARRS research core funding No. P2-0095.