Simple Finite Elements in Python
Development Notes and Applications

Robert Cimrman[12]

[1] New Technologies Research Centre, University of West Bohemia
[2] Department of Mechanics, Faculty of Applied Sciences, University of West Bohemia

PANM 2018
June 24–29
Hejnice, Czech Republic

Overview

History

- PhD studies (mathematical modeling of biological tissues): a FEM code in C.
  - Worked reasonably well.
  - Painful refactoring for non-anticipated features.
- Postdoc at INRIA Rocquencourt (human heart modeling, 2002-2003): a mix of matlab (for the code logic) and C (for the actual work).
  - Better for interactive work, refactoring, etc.
  - But the matlab language had drawbacks.

$$\Downarrow$$

Let me write a new code in a new interpreted language I have seen on a web-site, that will solve all those problems once and for all.

Python

- Python
  - It is an interpreted, interactive, object-oriented programming language.
  - It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes.
- http::/python.org:

  Python is a programming language that lets you work more quickly and integrate your systems more effectively.

- Features attractive for scientists (non-IT):
  - clean, easy-to-read syntax;
  - high-level, no manual memory management;
  - huge standard library;
  - talks to other languages (C, fortran);
  - large and friendly scientific computing community.

SfePy in a Nutshell

NEW TECHNOLOGIES
RESEARCH CENTRE
UNIVERSITY
OF WEST BOHEMIA

Simple **f**inite **e**lements in **Py**thon (http://sfepy.org)

- Solves systems of coupled partial differential equations (PDEs) by the FEM or IGA in 1D, 2D and 3D.
- A black-box PDE solver or a Python package which can be used for building custom applications.
- Problem description files have a form of Python modules, with mathematical-like description.
  - declarative API (problem description/definition files)
  - imperative API (interactive commands, scripts)
- It is a free software released under the New BSD License.
- It is a multi-platform software (Linux, Mac OS X, Windows).
- Last release: 2018.2 (19.06.2018).

Some Statistics

- Generated 2018-06-22 12:22:25 (in 12 seconds)
- Generator GitStats (version 2013.12.07), git version 1.9.1
- Report Period: 2007-12-19 14:21:12 to 2018-06-19 12:57:32
- 3836 days, 1696 active days (44.21%)
- Total Files: 890
- Total Lines: 583144 (1400773 added, 817629 removed)
  - Source code: about 120000 lines.
- Total Commits: 6336 (average 3.7 commits per active day, 1.7 per all days)
- Authors: 24 (average 264.0 commits per author)

Author	Commits (%)	+ lines	- lines	First commit	Last commit	Active days
Robert Cimrman	5509 (86.95%)	1301490	815315	2007-12-19	2018-06-19	1548
Vladimir Lukes	412 (6.50%)	83722	15709	2008-07-30	2018-05-25	227

. . . from 14.12.2004 without a VCS

- Languages: 85% Python, 15% C, Cython (+ other).

The Code

- Based on NumPy (ndarray object), SciPy (sparse matrices, solvers, high level algorithms) and other packages.
  - Implements FEM using fast vectorized operations (loops slow in Python).
  - Uses many external solvers (PETSc, Umfpack, MUMPS, ...).
- Domain approximation:
  - 1D line, 2D area (triangle, rectangle) and 3D volume (tetrahedron, hexahedron) finite elements;
  - isogeometric analysis (IGA), single NURBS patch limitation.
- Function spaces:
  - H^1 only.
  - $\boldsymbol{H}(\mathrm{curl})$, $\boldsymbol{H}(\mathrm{div})$ not implemented (yet?).
- Bases or shape functions:
  - the classical nodal (Lagrange) basis can be used with all element types;
  - the hierarchical (Lobatto) basis can be used with tensor-product elements (rectangle, hexahedron);
  - B-splines, NURBS for IGA, implemented using Bézier extraction operators.

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- A mesh can be:
  - given by its name (generated by external tools);
  - generated by the code (simple shapes).
- Examples:
  - declarative

    ```
    filename_mesh = 'meshes/3d/cylinder.mesh'
    ```

  - imperative

    ```
    mesh = Mesh.from_file('meshes/3d/cylinder.mesh')
    domain = FEDomain('domain', mesh)
    ```

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- A region (subdomain) can be defined by:
  - simple conditionals on coordinates;
  - general functions of coordinates;
  - Boolean operations from other regions.
- Examples:
  - declarative

```
regions = {
    'Omega' : 'all',
    'Left' : ('vertices in (x < 0.00001)', 'facet'),
    'Right' : ('vertices in (x > 0.099999)', 'facet'),
}
```

  - imperative

```
omega = domain.create_region('Omega', 'all')
left = domain.create_region('Left',
                            'vertices in x < 0.00001',
                            'facet')
right = domain.create_region('Right',
                             'vertices in x > 0.099999',
                             'facet')
```

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- A field can be defined on
  - the whole domain;
  - a volume (cell) subdomain;
  - a surface (facet) region.
- Examples:
  - declarative

```
fields = {
    'temperature' : ('real', 1, 'Omega', 1),
}
```

  - imperative

```
field = Field.from_args('temperature', nm.float64, 'scalar',
                        omega, approx_order=1)
```

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- Variables have the FE space given by their field and come in three flavors:
  - `unknown field` for state variables;
  - `test field` for test variables;
  - `parameter field` for variables with known values of DOFs.
- Examples:
  - declarative

```
variables = {
    'u' : ('unknown field', 'temperature', 0),
    'v' : ('test field',    'temperature', 'u'),
}
```

  - imperative

```
u = FieldVariable('u', 'unknown', field)
v = FieldVariable('v', 'test', field, primary_var_name='u')
```

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- Material parameters can be defined by:
  - constants;
  - general functions of time and coordinates, evaluated point-wise in quadrature points.
- Examples:
  - declarative

```
materials = {
    'm' : ({'c' : 1.0},),
}
```

  - imperative

```
m = Material('m', c=1.0)
```

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- Dirichlet (essential) boundary conditions can be defined by:
  - constants;
  - general functions of time and coordinates;
  - For the nodal FE basis, the coordinates are nodal coordinates;
  - For the IGA basis, the coordinates are surface quadrature coordinates, l_2 projection is used.
- Examples:
  - declarative

```
ebcs = {
    't1' : ('Left', {'u.0' : 2.0}),
    't2' : ('Right', {'u.0' : -2.0}),
}
```

  - imperative

```
ebc1 = EssentialBC('t1', left, {'u.0' : 2.0})
ebc2 = EssentialBC('t2', right, {'u.0' : -2.0})
```

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- Initial conditions (if applicable) can be defined by:
  - constants;
  - general functions of coordinates.
- Examples:
  - declarative

```
def get_ic(coors, ic):
    x, y, z = coors.T
    return 2 - 40.0 * x + ic_max * nm.sin(4 * nm.pi * x / 0.1)
functions = {
    'get_ic' : (get_ic,),
}
ics = {
    'ic' : ('Omega', {'u.0' : 'get_ic'}),
}
```

  - imperative

```
def get_ic(coors, ic):
    x, y, z = coors.T
    return 2 - 40.0 * x + ic_max * nm.sin(4 * nm.pi * x / 0.1)
ic_fun = Function('ic_fun', get_ic)
ic = InitialCondition('ic', omega, {'u.0' : ic_fun})
```

Problem Description

- (Systems of) PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs.
- Components of a problem description:
  - `Mesh, Domain`: the FE mesh and domain of solution description;
  - `Regions`: subdomain definitions of various topological dimension;
  - `Fields`: the discrete function spaces;
  - `Variables`: the unknown, virtual, or parameter variables for each field;
  - `Materials`: all parameters defined in point-wise in quadrature points;
  - `Boundary Conditions`: Dirichlet (essential), periodic, linear combination;
  - `Initial Conditions`: for time-dependent problem;
  - `Equations, Terms`: PDE definitions.
- Other components:
  - `Solvers`: configuration of time-stepping, nonlinear, linear, eigenvalue problem and optimization solvers;
  - `Options`: various options;
  - ...

Problem Description

- Equations can be built as a linear combination of many predefined terms:
  - each term has its quadrature order and the region of integration;
  - matrices/residuals can be assembled globally or by blocks.
- Examples:
  - declarative

```
integrals = {
    'i' : 2,
}
equations = {
    'Temperature' : """dw_laplace.i.Omega(m.c, v, u)
                     = dw_volume_dot.i.Omega(v, du/dt)"""
}
```

  - imperative

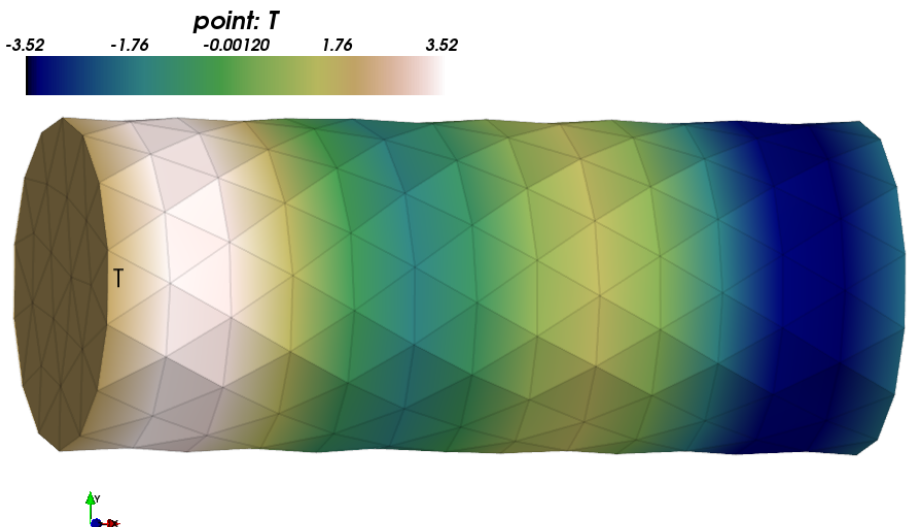
```
integral = Integral('i', order=2)
t1 = Term.new('dw_laplace(m.c, v, u)',
              integral, omega, m=m, v=v, u=u)
t2 = Term.new('dw_volume_dot(v, du/dt)',
              integral, omega, v=v, u=u)
eq = Equation('balance', t1 + t2)
eqs = Equations([eq])
```

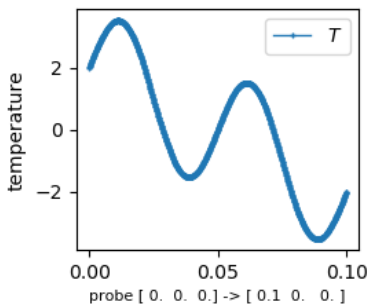
Visualization of Results, Probing

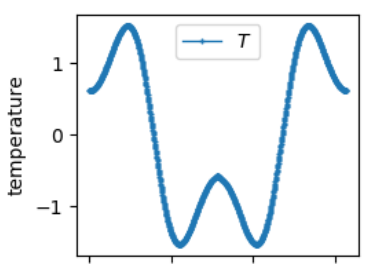
- The problem description snippets in previous slides can be combined to define a time-dependent diffusion problem with non-homogeneous initial conditions:

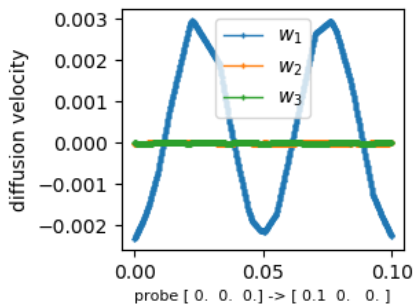
$$\int_\Omega v\frac{\partial u}{\partial t}+\int_\Omega c\nabla v\cdot\nabla u = 0\ ,\forall v\ ,u(x,0)=g(x)\ ,u(x,t)=\begin{cases} -2 & x\in\Gamma_{\text{left}}\ , \\ 2 & x\in\Gamma_{\text{right}}\ . \end{cases}$$

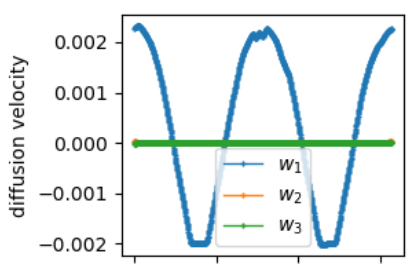
- Results can be stored to VTK files (other formats available).
- Line, circular and other probes can be defined to sample the results along the probe points.











Open Source Development Notes

- Python scientific software ecosystem.
- Tools.
- Lessons learned.

Python Scientific Software Ecosystem

The foundations:

- NumPy (http://www.numpy.org), the fundamental package for numerical computation, defines the numerical nD array type and basic operations.
  - https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html
- SciPy (http://scipy.org), a collection of numerical algorithms and domain-specific toolboxes.
- Matplotlib (http://matplotlib.org), a plotting package, that provides publication-quality 2D plotting as well as rudimentary 3D plotting.

Other widely-used packages:

- Cython (http://cython.org), for Python $\rightarrow$ C translation and C/fortran library calls.
- SymPy (http://sympy.org), for symbolic mathematics and computer algebra.
- pandas (http://pandas.pydata.org), providing high-performance, data structures.
- scikit-image (http://scikit-image.org) is a collection of algorithms for image processing.
- scikit-learn (http://scikit-learn.org) is a collection of machine learning algorithms.

The most disruptive (subjectively):

- Dask (https://dask.readthedocs.io), provides advanced parallelism for analytics, enabling performance at scale.

Tools

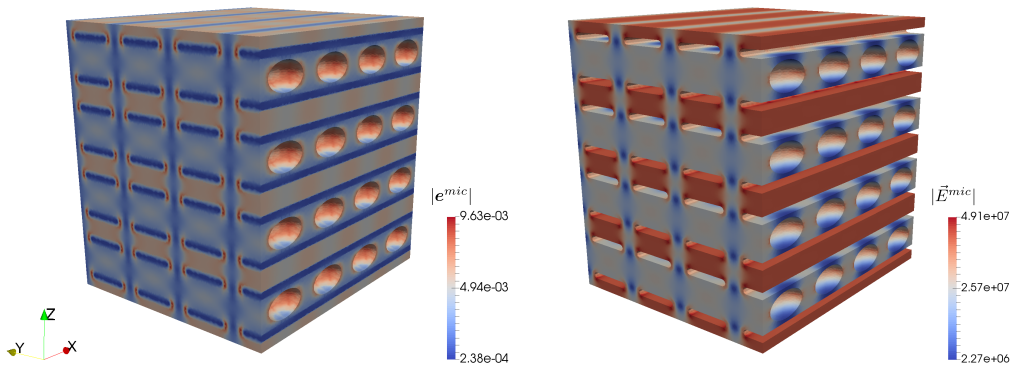
- Git (https://git-scm.com), is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
  - Every repository copy contains all the history, can work off-line!
  - Not only for source code.
  - GitHub (https://github.com), a development platform, allows people to host and review code, manage projects, and build software.
  - GitLab (https://gitlab.com), a non-Microsoft-owned alternative to github.
  - ...
- Continuous integration: Travis CI (via GitHub).
- Sphinx (http://sphinx-doc.org), is a tool that makes it easy to create documentation.
- Collaboration:
  - Issues and pull requests on github.
  - Mailing list: https://mail.python.org/mm3/mailman3/lists/sfepy.python.org/.

Lessons Learned

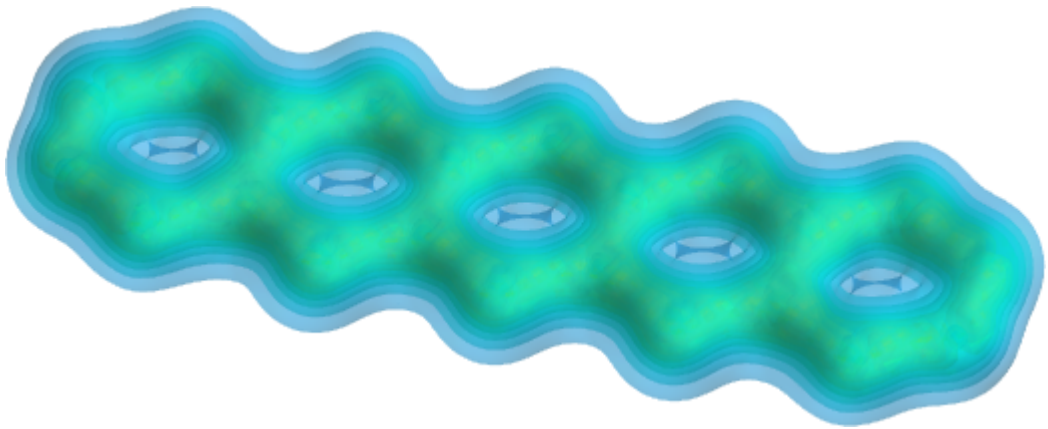
- Use Git even for small projects.
  - Using tools like git, github, Travis CI etc. really helps.
- Reuse existing high-quality packages (with a compatible license).
  - Do not reinvent the wheel (too much - it is OK for learning).
- Automate everything repetitive that can be automated.
  - Find tools, or write your own if needed.
  - Try to streamline and document the maintaining tasks as much as possible, so that things like releases do not take too much time/mental power.
- Complex vs. Complicated.
  - https://www.infoq.com/presentations/Simple-Made-Easy
  - It is not the case of the current SfePy, but I try to steer it that way.

Examples

- Application of the theory of homogenization to modeling of fluid-saturated piezoelectric porous media.



- Convergence of several mixing algorithm in the context of ab-initio electronic structure calculations.



Fluid-Saturated Piezoelectric Porous Media Modeling

Authors:

- Vladimír Lukeš[1] (implementation, simulations, evaluation of results)
- Eduard Rohan[1] (theory)

Extension of Biot model for porous-piezoelectric structures:

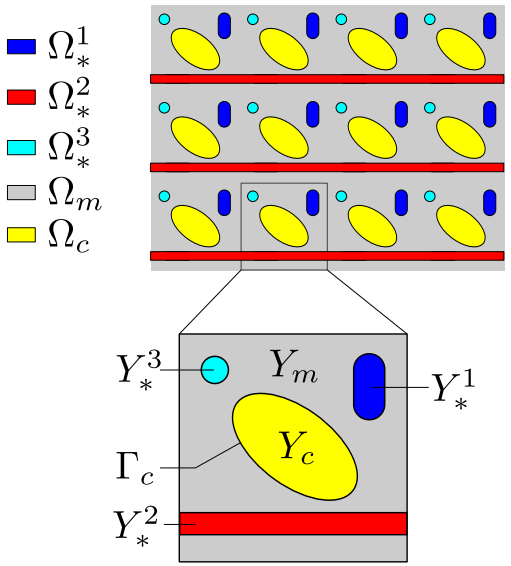
- Applications:
  - biomaterials – scaffolds for bone regeneration, ...
  - metamaterials – electric filed-controlled fluid transport, piezoelectric micropumps, ...

---

[1] Department of Mechanics, Faculty of Applied Sciences, University of West Bohemia

Micromodel of piezoelectric skeleton

- Static response (steady state)
- Domain split: $\Omega = \Omega_m \cup \Omega_c \cup \Omega_*$
  - piezoelectric solid matrix – Ω_m^ε
  - conductors – Ω_*^ε
  - disconnected fluid inclusions – Ω_c^ε



- Several possible configurations ($\Rightarrow$ different homogenized models):
  - connected or disconnected channels;
  - connected or disconnected conductors.
- This example: disconnected channels, connected conductors.
  - For the given potentials $\bar{\varphi}^{k,veps}$ in the conductors $k = 1, 2, \ldots$, compute the piezo-elastic deformation of the matrix and the scalar pressure in each fluid inclusion.

Micromodel of piezoelectric skeleton

- Equilibrium of stress and electric displacement:

$$-\nabla \cdot \boldsymbol{\sigma}^{\varepsilon}(\boldsymbol{u}^{\varepsilon}, \varphi^{\varepsilon}) = \boldsymbol{f}^{\varepsilon} \ , \qquad \text{in } \Omega_m^{\varepsilon} \ ,$$
$$-\nabla \cdot \vec{D}^{\varepsilon}(\boldsymbol{u}^{\varepsilon}, \varphi^{\varepsilon}) = q_E^{\varepsilon} \ , \qquad \text{in } \Omega_m^{\varepsilon} \ ,$$

  - $\boldsymbol{f}^{\varepsilon}$ – volume forces, q_E^{ε} – volume electric charge.
- Mass conservation – fluid filled inclusions:

$$\int_{\partial\Omega_c^{l,\varepsilon}} \boldsymbol{u}^{\varepsilon} \cdot \boldsymbol{n}^{[c]} \, \mathrm{dS} + \gamma p^{l,\varepsilon} |\Omega_c^{l,\varepsilon}| = 0 \ , \quad \forall l \in \{1, \ldots, \bar{l}\} \ ,$$

  - γ – fluid compressibility, $\bar{l}$ – the number of inclusions.
- Boundary and interface conditions:

$$\boldsymbol{n} \cdot \boldsymbol{\sigma}^{\varepsilon} = \boldsymbol{h}^{\varepsilon} \quad \text{on } \Gamma_{\sigma}^{\varepsilon} \ , \qquad \boldsymbol{n} \cdot \vec{D}^{\varepsilon} = \varrho_E^{\varepsilon} \quad \text{on } \Gamma_{\vec{D}}^{\varepsilon} \ ,$$
$$\boldsymbol{u}^{\varepsilon} = \bar{\boldsymbol{u}} \quad \text{on } \Gamma_{u}^{\varepsilon} \ , \qquad \varphi^{\varepsilon} = \bar{\varphi}^{k} \ , \quad \text{on } \Gamma_{\varphi}^{k,\varepsilon} \ ,$$
$$\boldsymbol{n} \cdot \boldsymbol{\sigma}^{\varepsilon} = -p^{\varepsilon} \boldsymbol{n} \quad \text{on } \Gamma_{c}^{\varepsilon} \ , \qquad \int_{\Gamma_*^k} \boldsymbol{n} \cdot \vec{D}^{\varepsilon} \, \mathrm{dS} = 0 \ , \quad k = 1, 2, \ldots k^* \ ,$$

  - $\boldsymbol{h}^{\varepsilon}$ – applied surface forces, ϱ_E^{ε} – surface electric charge.

Micromodel of piezoelectric skeleton

- Constitutive equations:

$$\sigma_{ij}^{\varepsilon}(\boldsymbol{u}^{\varepsilon},\varphi^{\varepsilon}) = A_{ijkl}^{\varepsilon} e_{kl}^{\varepsilon}(\boldsymbol{u}^{\varepsilon}) - g_{kij}^{\varepsilon}\partial_k\varphi^{\varepsilon} \ ,$$
$$D_k^{\varepsilon}(\boldsymbol{u}^{\varepsilon},\varphi^{\varepsilon}) = g_{kij}^{\varepsilon} e_{ij}^{\varepsilon}(\boldsymbol{u}^{\varepsilon}) + d_{kl}^{\varepsilon}\partial_l\varphi^{\varepsilon} \ .$$

$\mathbb{A}^{\varepsilon} = (A_{ijkl}^{\varepsilon})$	the elasticity tensor, $A_{ijkl} = A_{klij} = A_{jilk}$,
$\boldsymbol{g}^{\varepsilon} = (g_{kij}^{\varepsilon})$	piezo-coupling 3rd order tensor, $g_{kij}^{\varepsilon} = g_{kji}^{\varepsilon}$,
$\boldsymbol{d}^{\varepsilon} = (d_{kl}^{\varepsilon})$	electric permitivity tensor, $d_{kl}^{\varepsilon} = d_{lk}^{\varepsilon}$.

- Material scaling:
  - strongly controlled electric field: $\bar{\varphi}^{k,veps} = \bar{\varphi}^{k}$;
  - ε-rescaling of $\boldsymbol{g}^{\varepsilon}$ and $\boldsymbol{d}^{\varepsilon}$ to preserve finite electric field in the $\varepsilon \to 0$ limit:

$$\boldsymbol{g}^{\varepsilon}(x) = \varepsilon\bar{\boldsymbol{g}} \quad \boldsymbol{d}^{\varepsilon}(x) = \varepsilon^2\bar{\boldsymbol{d}} \quad \text{in } \Omega_m^{\varepsilon} \ .$$

- Weak formulation: for given $\bar{\varphi}^{k}$ in $\Omega_*^{k,\varepsilon}$ and $\boldsymbol{f}^{\varepsilon}$, $\boldsymbol{h}^{\varepsilon}$, q_E^{ε}, ϱ_E^{ε}, find $(\boldsymbol{u}^{\varepsilon},\varphi^{\varepsilon},p^{\varepsilon})$ such that: ...

Homogenization

- Homogenization procedure – unfolding method, two scale convergence, ...
  - Asymptotic expansions:
    $\mathcal{T}_\varepsilon(\boldsymbol{u}^\varepsilon) \approx \boldsymbol{u}^0(x) + \varepsilon \boldsymbol{u}^1(x,y)$, $\mathcal{T}_\varepsilon(\varphi^\varepsilon) \approx \varphi^0(x,y)$, $\mathcal{T}_\varepsilon(p^\varepsilon) \approx p^0(x)$.
  - Similar expansions used for the test fields $\boldsymbol{v}, \psi, q$.
  - $\boldsymbol{u}^1$ and φ^0 are Y-periodic in $y \in Y_m$.
- Two-scale functions $\boldsymbol{u}^1(x,y)$, $\varphi^0(x,y)$ expressed (due to the linearity) in terms of the characteristic responses (corrector functions) $\boldsymbol{\omega}, \eta$:

$$\boldsymbol{u}^1(x,y) = \boldsymbol{\omega}^{ij} e^x_{ij}(\boldsymbol{u}^0) - \boldsymbol{\omega}^P p^0 + \boldsymbol{\omega}^\rho \rho_E + \sum_k \hat{\boldsymbol{\omega}}^k \bar{\varphi}^k \ ,$$

$$\varphi^0(x,y) = \eta^{ij} e^x_{ij}(\boldsymbol{u}^0) - \eta^P p^0 + \eta^\rho \rho_E + \sum_k \hat{\varphi}^k \bar{\varphi}^k$$

- To get the corrector functions, several sub-problems must be solved in the reference periodic cell with different boundary conditions, involving:

$$a_Y^{m*}(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{|Y|} \int_{Y_m *} [\mathbb{A} \boldsymbol{e}_y(\boldsymbol{u})] : \boldsymbol{e}_y(\boldsymbol{v}) \ , \quad g_Y^m(\boldsymbol{u}, \psi) = \frac{1}{|Y|} \int_{Y_m} \bar{g}_{kij} e^y_{ij}(\boldsymbol{u}) \partial^y_k \psi \ ,$$

$$d_Y^m(\varphi, \psi) = \frac{1}{|Y|} \int_{Y_m} [\bar{\boldsymbol{d}} \nabla_y \varphi] \cdot \nabla_y \psi \ , \quad \boldsymbol{\Pi}^{ij} = (\Pi^{ij}_k), \quad \Pi^{ij}_k = y_j \delta_{ik} \ .$$

Effective piezo-poroelastic coefficients

Modified Biot poroelastic coefficients:

$$A^H_{klij} = a^{m*}_Y \left(\boldsymbol{\omega}^{ij} + \boldsymbol{\Pi}^{ij}, \boldsymbol{\omega}^{kl} + \boldsymbol{\Pi}^{kl}\right) + d^m_Y \left(\eta^{kl}, \eta^{ij}\right)$$
$$B^H_{ij} = a^{m*}_Y \left(\boldsymbol{\omega}^P, \boldsymbol{\Pi}^{ij}\right) - g^m_Y \left(\boldsymbol{\Pi}^{ij}, \eta^P\right) + \phi\delta_{ij}$$
$$M^H = a^{m*}_Y \left(\boldsymbol{\omega}^P, \boldsymbol{\omega}^P\right) + d^m_Y \left(\eta^P, \eta^P\right) + \phi\delta_{ij}$$

Coefficients related to the prescribed el. potentials and surface charge:

$$H^k_{ij} = a^{m*}_Y \left(\hat{\boldsymbol{\omega}}^k, \boldsymbol{\Pi}^{ij}\right) - g^m_Y \left(\boldsymbol{\Pi}^{ij}, \hat{\varphi}^k\right)$$
$$S^H_{ij} = a^{m*}_Y \left(\boldsymbol{\omega}^\rho, \boldsymbol{\Pi}^{ij}\right) - g^m_Y \left(\boldsymbol{\Pi}^{ij}, \eta^\rho\right)$$
$$R^H = -\oint_{\Gamma_c} \boldsymbol{\omega}^\rho \cdot \boldsymbol{n}^{[c]} \,\mathrm{dS}_y\,, \quad Z^{H,k} = -\oint_{\Gamma_c} \boldsymbol{\omega}^k \cdot \boldsymbol{n}^{[c]} \,\mathrm{dS}_y$$

Homogenized macroscopic problem

- Find $\boldsymbol{u}^0 \in \boldsymbol{U}(\Omega)$, $p^0 \in L^2(\Omega)$ such that:

$$\int_\Omega \left(\mathbb{A}^H \boldsymbol{e}(\boldsymbol{u}^0) - p^0 \boldsymbol{B}^H \right) : \boldsymbol{e}(\boldsymbol{v}^0)\,\mathrm{dV}_x = -\int_\Omega \left(\sum_k \boldsymbol{H}^k \bar{\varphi}^k + \boldsymbol{S}^H \varrho_E \right) : \boldsymbol{e}(\boldsymbol{v}^0)\,\mathrm{dV}_x + \int_\Omega \boldsymbol{f}\cdot\boldsymbol{v}^0\,\mathrm{dV}_x + \int_{\partial\Omega} \boldsymbol{h}\cdot\boldsymbol{v}^0\,\mathrm{dS}_x\ ,$$

$$\int_\Omega \left(\boldsymbol{B}^H : \boldsymbol{e}(\boldsymbol{u}^0) + p^0 M^H \right) q^0\,\mathrm{dV}_x = \int_\Omega \left(\sum_k Z^{H,k} \bar{\varphi}^k + R^H \varrho_E \right) q^0\,\mathrm{dV}_x\ ,$$

  for all $\boldsymbol{v}^0 \in \boldsymbol{U}_0(\Omega)$, $q^0 \in L^2(\Omega)$.

- Fields reconstruction at microlevel:
  - displacement: $\boldsymbol{u}^1(x,y) = \boldsymbol{\omega}^{ij} e^x_{ij}(\boldsymbol{u}^0) - \boldsymbol{\omega}^P p^0 + \boldsymbol{\omega}^\rho \rho_E + \sum_k \hat{\boldsymbol{\omega}}^k \bar{\varphi}^k$
  - el. field: $\varphi^0(x,y) = \eta^{ij} e^x_{ij}(\boldsymbol{u}^0) - \eta^P p^0 + \eta^\rho \rho_E + \sum_k \hat{\varphi}^k \bar{\varphi}^k$
  - gradients – strain and electric fields:

$$\boldsymbol{e}^{mic}(x,y) = \boldsymbol{e}_x(\boldsymbol{u}^0) + \boldsymbol{e}_y(\boldsymbol{u}^1)\ ,$$

$$\nabla\varphi^{mic} \equiv \vec{E}^{mic}(x,y) = \frac{1}{\varepsilon_0}\nabla_y \varphi^0\ .$$

General Algorithm for Linear Problems

- More than two scales allowed.
  - For example, micro-, meso-, macro-scale.
- For each scale level of the microstructure:
  1. Compute **characteristic (corrector) functions** by solving auxiliary corrector problems on a reference periodic cell domain.
  2. Using the corrector functions, evaluate **homogenized coefficients** for the higher level and/or homogenized model of the current level.
  3. Optionally, solve the homogenized model.
  4. Go to an upper level, if any.

$$\Downarrow$$

A way of expressing the relationships and data flow among different sub-problems is needed.

$$\Downarrow$$

Implemented in SfePy: **homogenization engine**.

Homogenization Engine

- Declarative description of complex dependencies in a multiscale simulation.
- Allows defining both the components of FE-discretized PDEs defining the corrector problems . . .
  - domain, regions, materials,
  - {Dirichlet, Neumann, periodic, . . . } boundary conditions,
  - FE fields, variables,
  - (solvers, etc.)
- . . . and corrector-corrector, coefficient-corrector, coefficient-coefficient dependencies.
  - Automatically determines correct evaluation order.
  - **Parallelized** using multiprocessing package.

Illustration of Dependencies


```
requirements = {
    'pis_u': {
        'variables': ['u'],
        'class': cb.ShapeDimDim,
    },
    'corrs_rs': {
        'requires': ['pis_u'],
        'ebcs': ['fixed_u', 'fixed_r'],
        'epbcs': periodic['per_u'] + periodic['per_r'],
        'is_linear': True,
        'equations': {
            'eq1':
                """dw_lin_elastic.i2.Yms(matrix.D, v, u)
                 - dw_piezo_coupling.i2.Ym(piezo.g, v, r)
                = - dw_lin_elastic.i2.Yms(matrix.D, v, Pi_u)""",
            'eq2':
                """
                 - dw_piezo_coupling.i2.Ym(piezo.g, u, s)
                 - dw_diffusion.i2.Ym(piezo.d, s, r)
                 = dw_piezo_coupling.i2.Ym(piezo.g, Pi_u, s)""",
        },
        'set_variables': [('Pi_u', 'pis_u', 'u')],
        'class': cb.CorrDimDim,
        'save_name': 'corrs_rs_%d' % grid0,
        'dump_variables': ['u', 'r'],
        'solvers': {'ls': 'ls', 'nls': 'ns_em1'},
    },
}
```

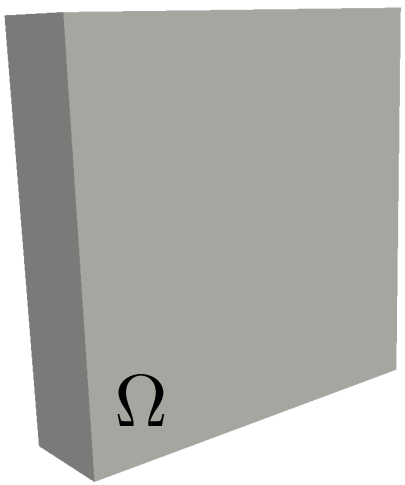
```
coefs = {
    'A1': {
        'status': 'auxiliary',
        'requires': ['pis_u', 'corrs_rs'],
        'expression': 'dw_lin_elastic.i2.Yms(matrix.D, U1, U2)',
        'set_variables': [('U1', ('corrs_rs', 'pis_u'), 'u'),
                          ('U2', ('corrs_rs', 'pis_u'), 'u')],
        'class': cb.CoefSymSym,
    },
    'A2': {
        'status': 'auxiliary',
        'requires': ['corrs_rs'],
        'expression': 'dw_diffusion.i2.Ym(piezo.d, R1, R2)',
        'set_variables': [('R1', 'corrs_rs', 'r'),
                          ('R2', 'corrs_rs', 'r')],
        'class': cb.CoefSymSym,
    },
    'A': {
        'requires': ['c.A1', 'c.A2'],
        'expression': 'c.A1 + c.A2',
        'class': cb.CoefEval,
    },
}
```

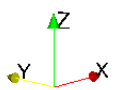
Homogenized model vs. reference model

Homogenized model:

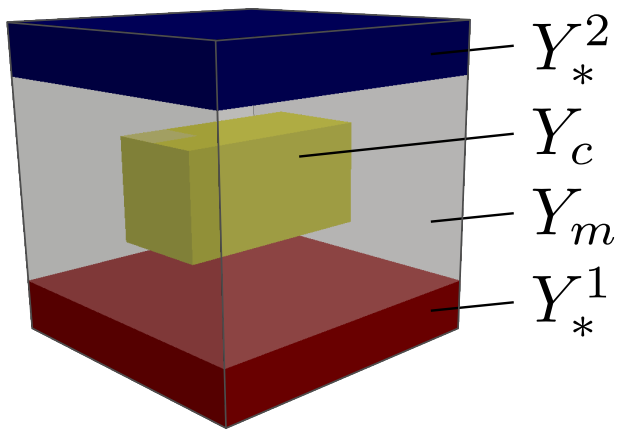
- solution of local subproblems – characteristic responses (correctors) $\longrightarrow$ homogenized coefficients $\longrightarrow$ macroscopic responses $\longrightarrow$ fields reconstruction

Macroscopic domain:





Microscopic domain:

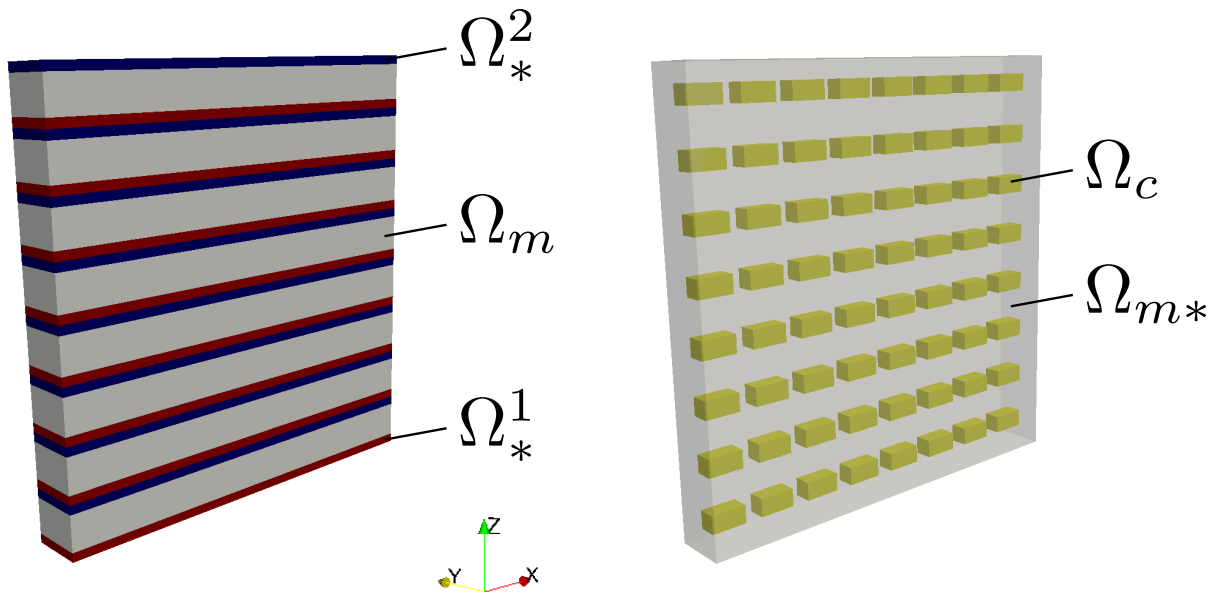


Homogenized model vs. reference model

Reference model:

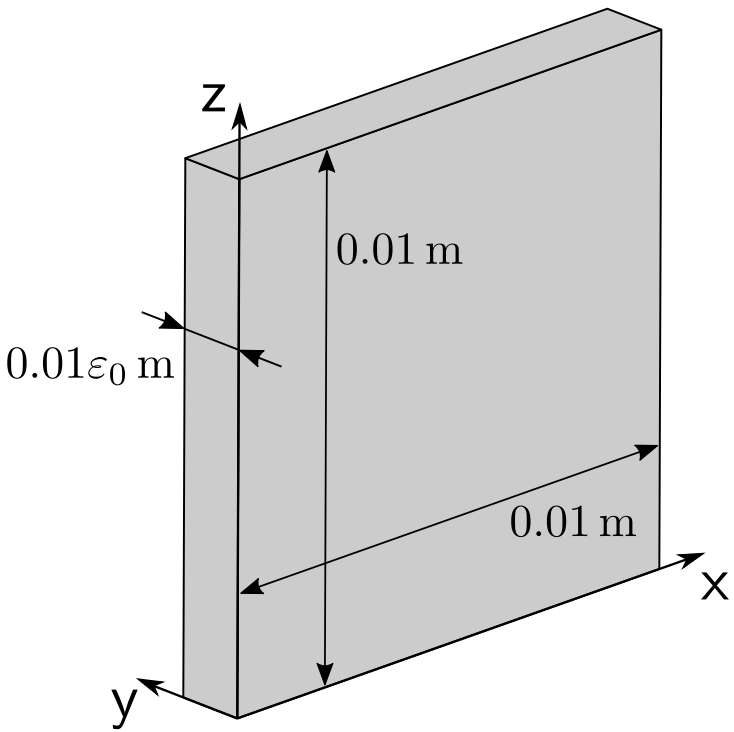
- direct numerical simulation of the heterogeneous periodic structure
- established by copies of the reference cell for a given size $\varepsilon_0 > 0$

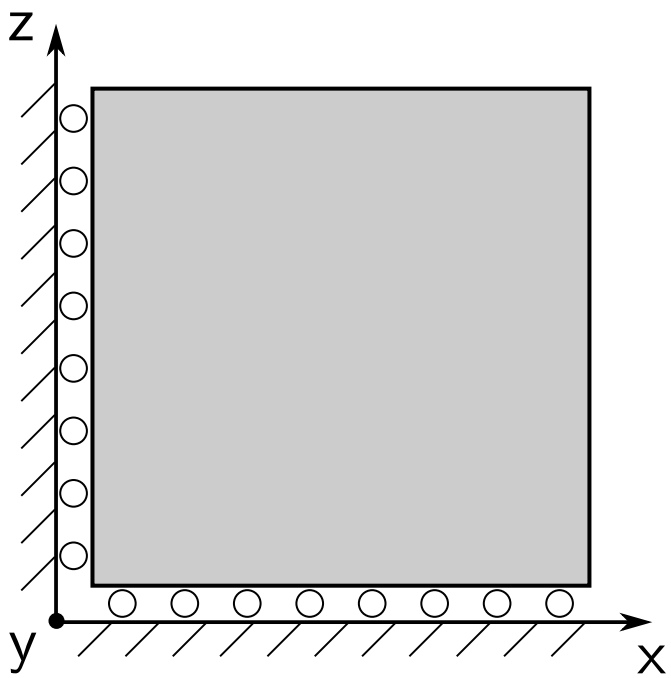
Domain for direct computation:



Validation test

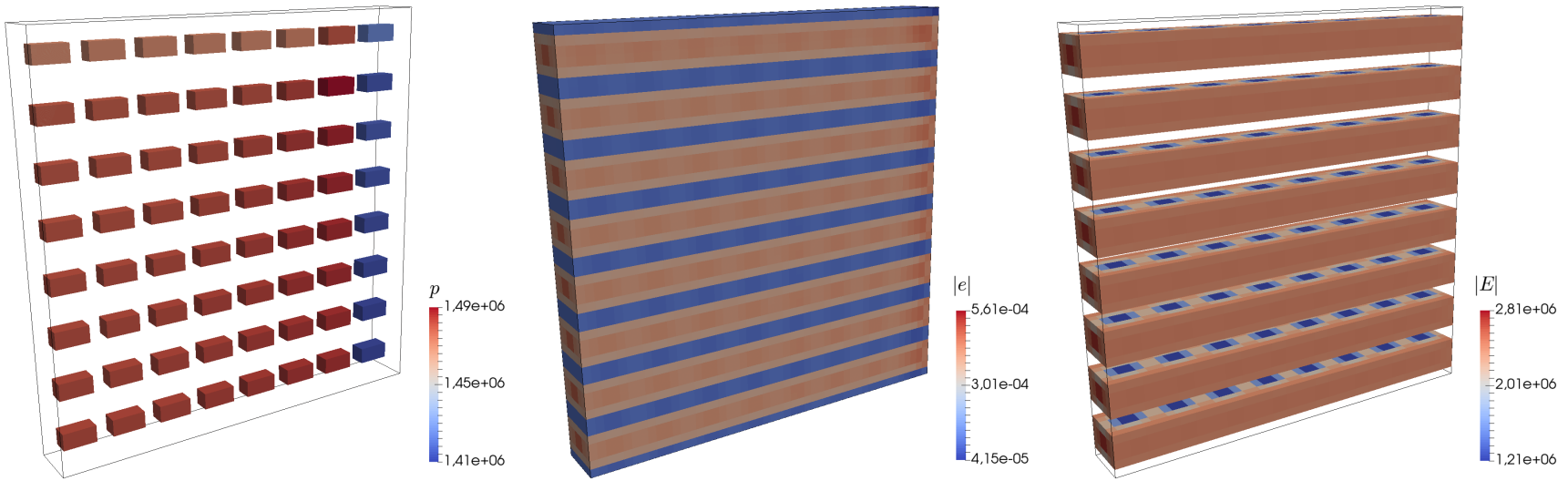
- block sample: $0.01 \times 0.01\varepsilon_0 \times 0.01$
- barium–titanite ($BaTiO_3$) piezoelectric matrix + metallic conductors + fluid inclusions
- no external loads, prescribed potentials $\bar{\varphi}^1 = +1000\,\mathrm{V}$ and $\bar{\varphi}^2 = -1000\,\mathrm{V}$ in conductors $\longrightarrow$ deformation of the sample induced due to the piezoelectric effect



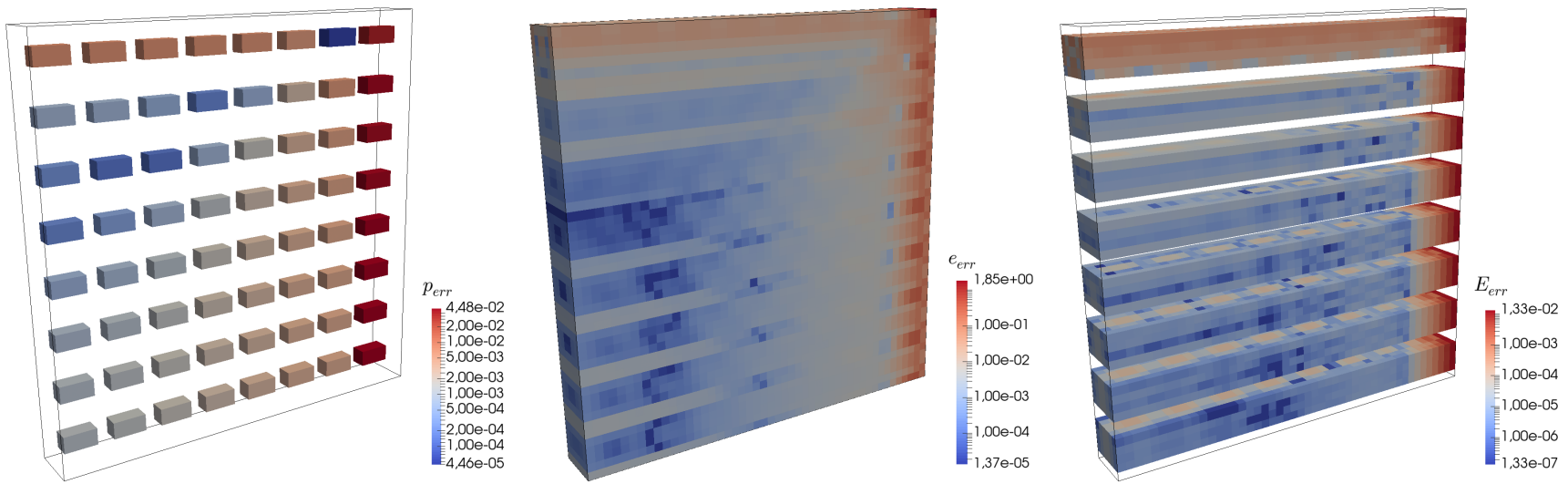


Validation test

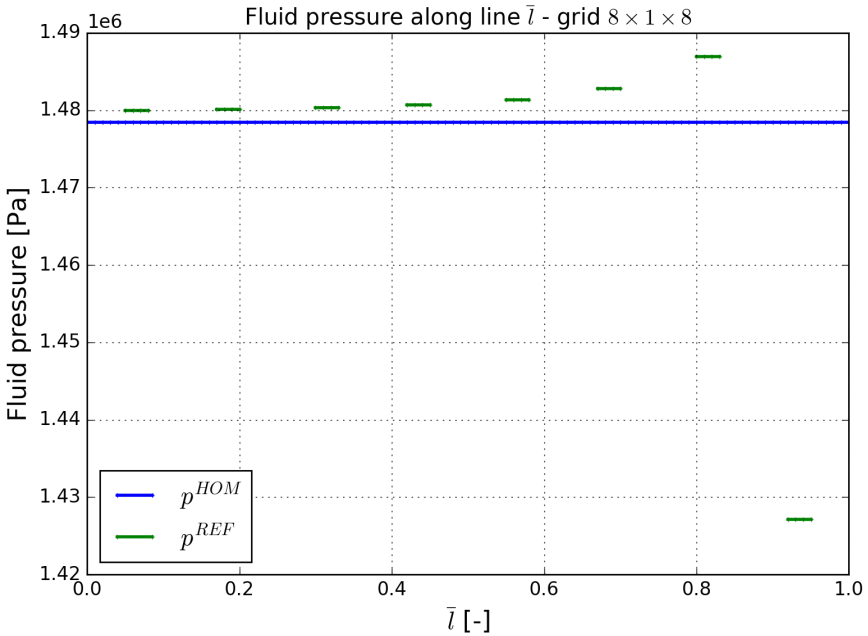
reference model

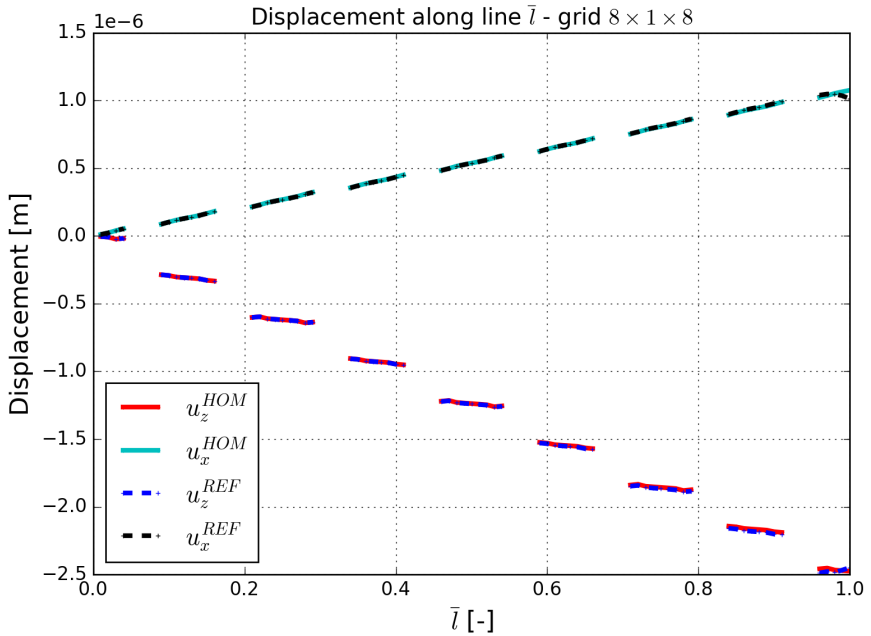


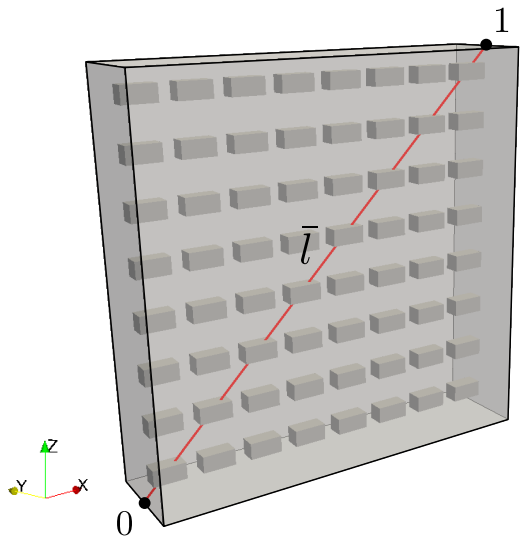
relative errors

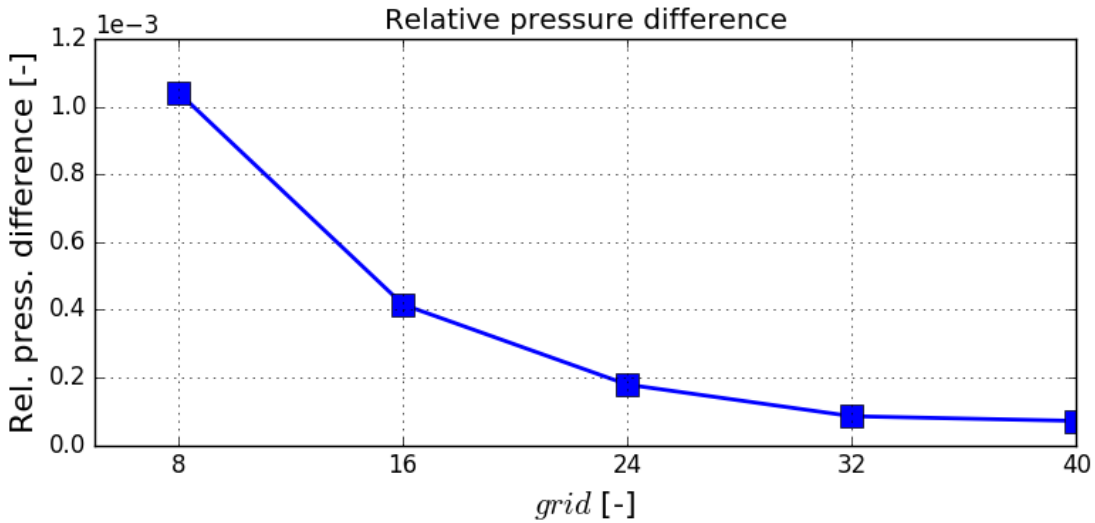


Validation test



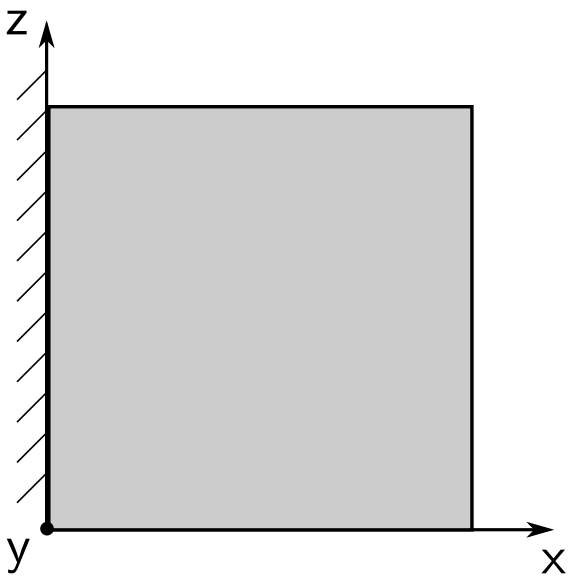


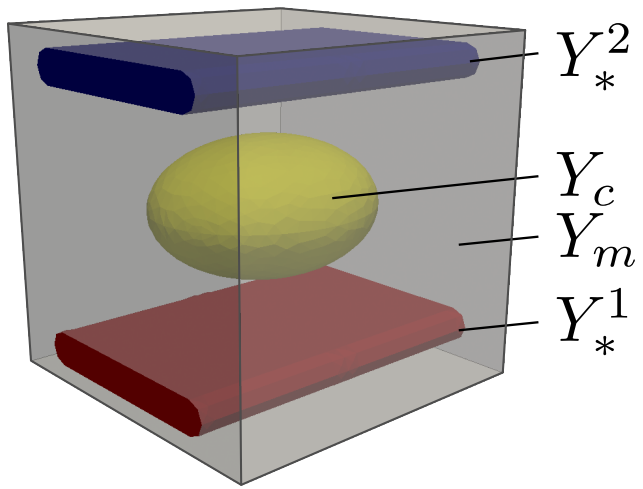




Disconnected Fluid, Connected Conductors I

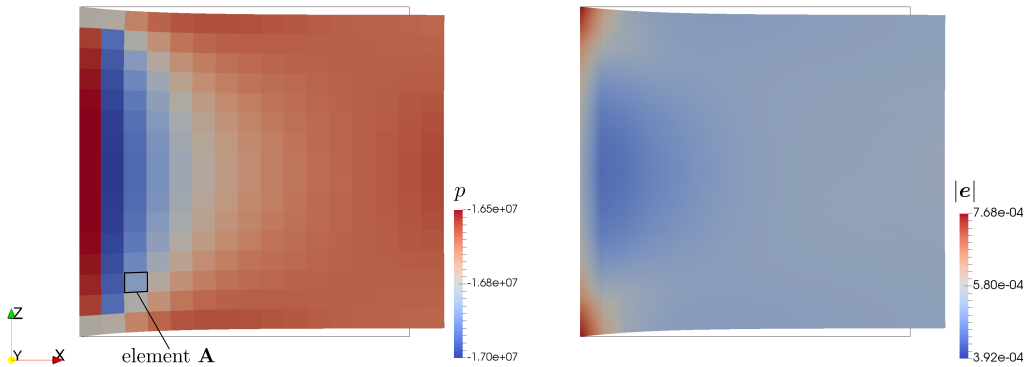
- Macro:
  - Block $0.01 \times 0.0025 \times 0.01$ m is fixed on the left face.
  - Periodic boundary condition is applied in y direction.
  - Deformation is induced by prescribing potentials ± 1000 V in the embedded conductors.
- Micro: disconnected fluid, two connected conductors.



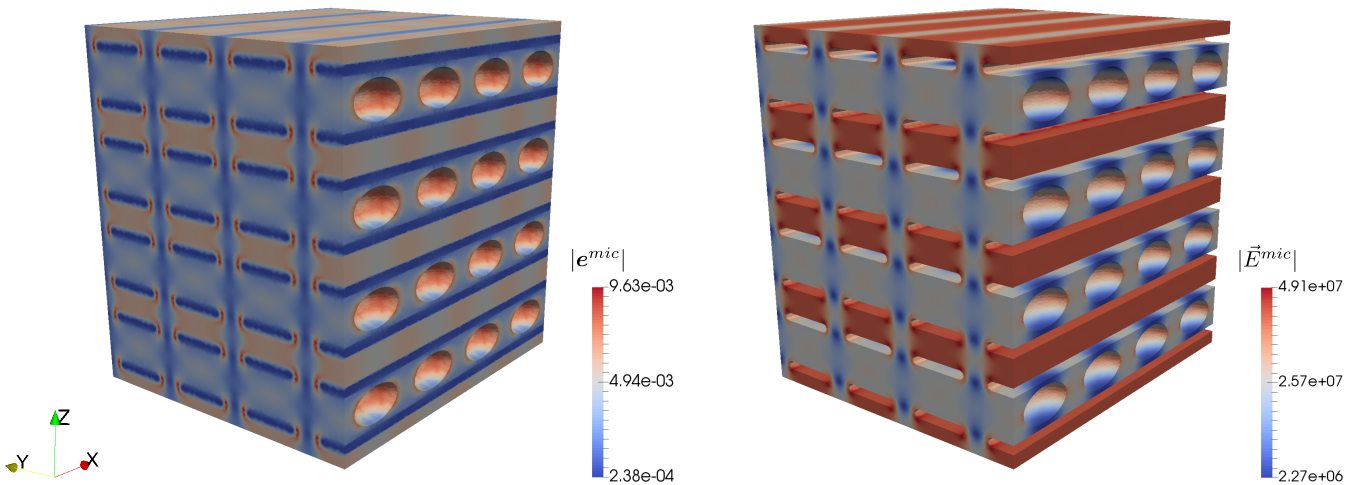


Disconnected Fluid, Connected Conductors II

- Deformed macroscopic (300× magnified) sample and the resulting fields: left: pressure p; right: strain $\boldsymbol{e}(\boldsymbol{u}^0)$.

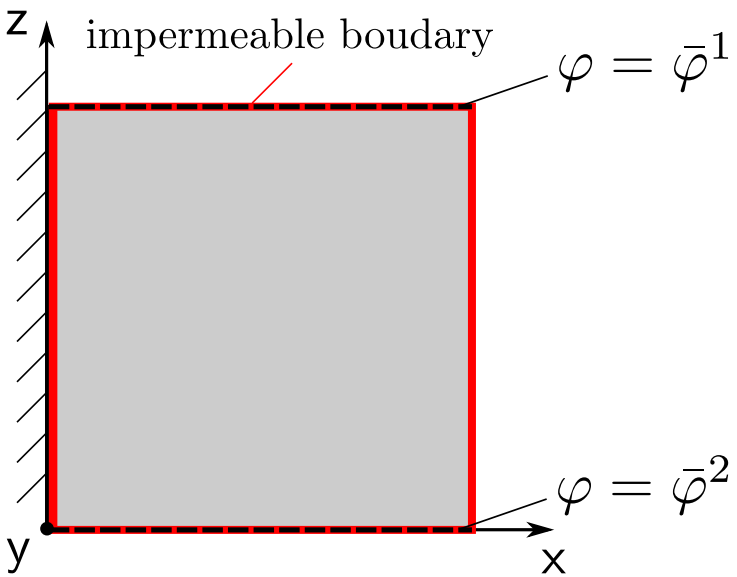


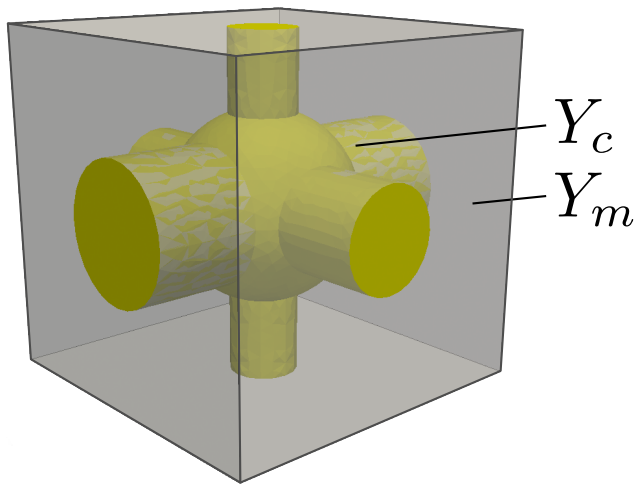
- Magnitudes of reconstructed fields in the macroscopic element A: left: strain $\boldsymbol{e}^{mic}$; right: electric field $\vec{E}^{mic}$.



Connected Fluid, No Conductors I

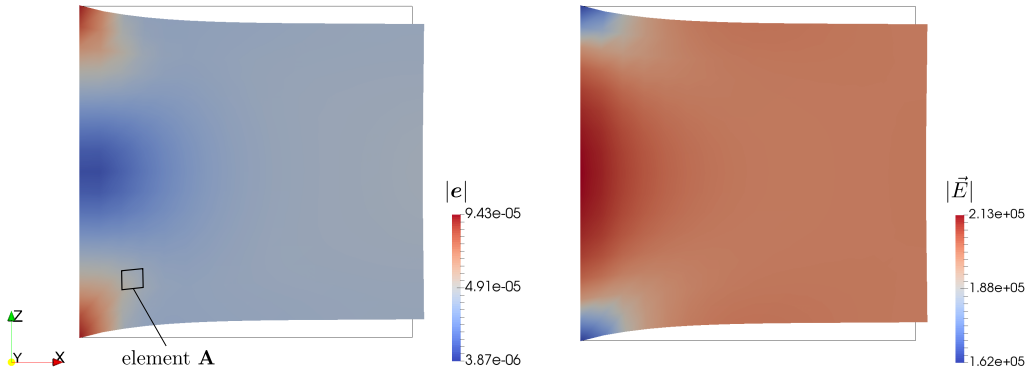
- Macro:
  - Block $0.01 \times 0.0025 \times 0.01$ m is fixed on the left face.
  - Periodic boundary condition is applied in y direction.
  - Deformation is induced by prescribing potentials ± 1000 V on the top and bottom faces of the block.
- Micro: connected fluid, no conductors.



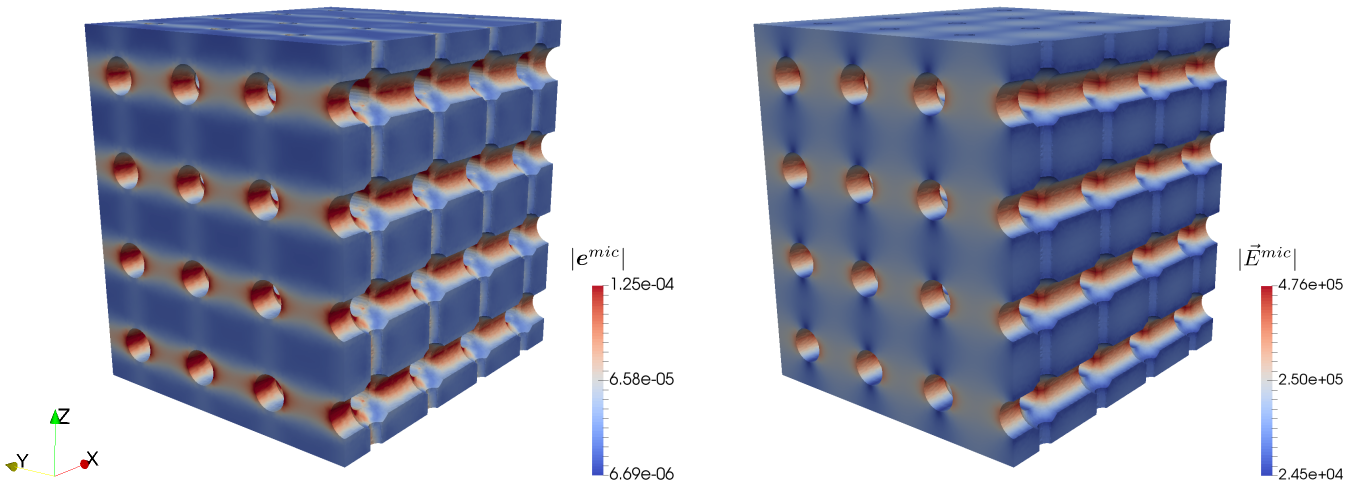


Connected Fluid, No Conductors II

- Deformed macroscopic (3000× magnified) sample and the resulting fields: left: strain $\boldsymbol{e}(\boldsymbol{u}^0)$, right: $\vec{E} = \nabla_x \varphi^0$.



- Magnitudes of reconstructed fields in the macroscopic element A: left: strain $\boldsymbol{e}^{mic}$; right: electric field $\vec{E}^{mic}$.



Discussion

Computational cost – $\varepsilon_0 = 0.01/24$:

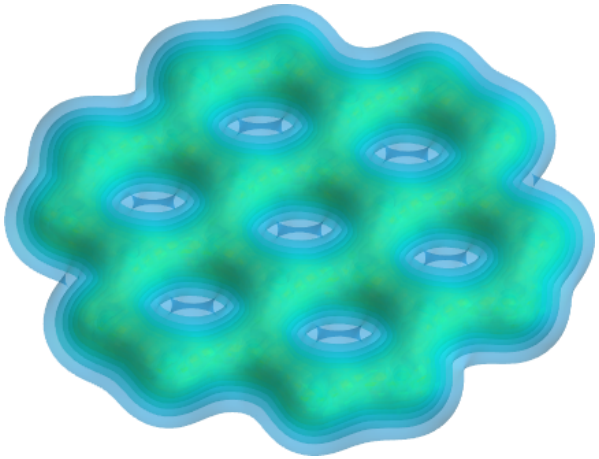
- reference model:
  - solution time $\approx$ 300 seconds
  - FE model $\approx 4.5 \times 10^5$ degrees of freedom (DOFs)
- homogenized model:
  - solution time $\approx$ 20 seconds including reconstructions at the microlevel
  - microscopic FE model $\approx$ 800 DOFs $\times 4$ – corrector subproblems
  - macroscopic FE model $\approx$ 600 DOFs

Conclusion:

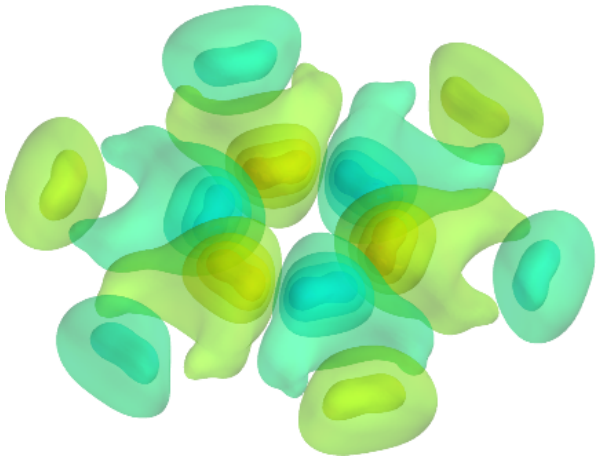
- The presented homogenization method gives, for a sufficiently small ε_0, responses which are in close agreement with the reference model.
- Contrary to the direct numerical computation, the multiscale simulation provides reliable results with a substantially less computational cost.

NEW TECHNOLOGIES
RESEARCH CENTRE
UNIVERSITY
OF WEST BOHEMIA

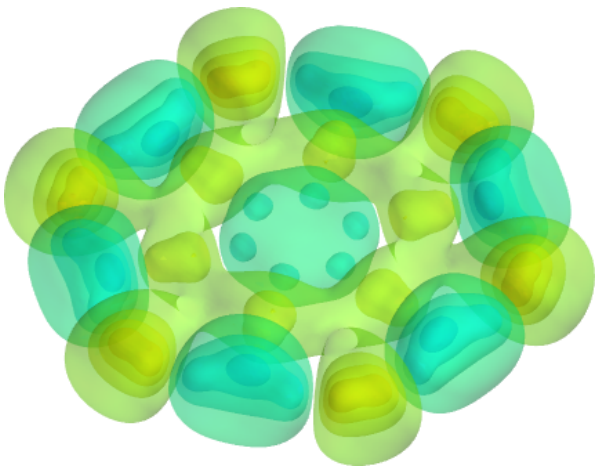
Graphene "Flower"



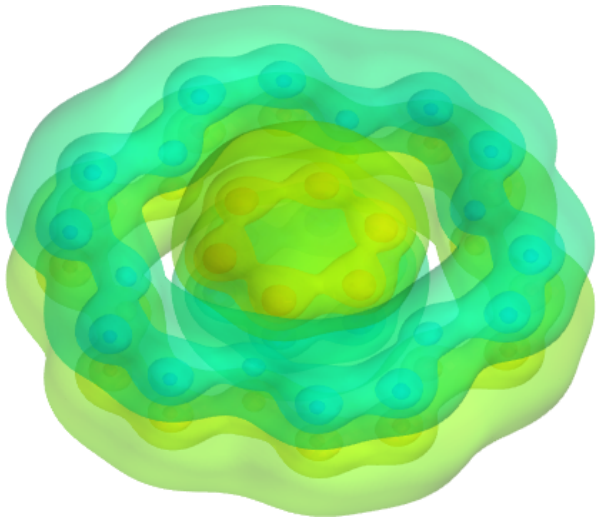
ρ



ψ_{19}



ψ_{21}



ψ_{36}

Team

Collaborators (software):

- Matyáš Novák[1,2]: main QM package developer;
- Jiří Vackář[1]: theoretical physics, pseudopotential generation;
- RC: initial QM package contributor, SfePy, FE-related support'

Domain-specific help:

- Radek Kolman[3]: help with isogeometric analysis implementation;
- Jiří Kopal, Miroslav Rozložník, Miroslav Tůma[4]: linear algebra questions: solvers, preconditioners;
- ...

---

[1] Institute of Physics, Czech Academy of Sciences
[2] Department of Mechanics, Faculty of Applied Sciences, University of West Bohemia
[3] Institute of Thermomechanics, Czech Academy of Sciences
[4] Institute of Computer Science, Czech Academy of Sciences

Context

- Past Project GAP108/11/0853:
  Nanostructures with transition metals: Towards ab-initio material design
- Project GA17-12925S:
  Strength of materials and mechanical components based on iron: Multi-scale approach
- Tools:
  - ab-initio electronic structure calculations $\leftrightarrow$ understanding the structure/material properties,
    - equilibrium atomic positions, stability, etc.
    - elastic constants, cohesive strength, hardness, etc.
    - . . . design of new Finnis-Sinclair type potentials for . . .
  - molecular dynamics simulations,
    - . . . detailed crack resolution for . . .
  - finite element simulations.
- This example: some algorithms accelerating convergence of our electronic structure calculations solver.

Role and Aims

. . . to fill a considerable gap among the existing well-established methods.

Bloch theorem based methods

- translational symmetry,
- Bloch-type basis,
- non-periodic systems: demanding tricks (e.g. supercells).

A plane wave methods:
+ excellent convergence control, orthogonal basis,
− core states: in practice too demanding → pseudopotentials;

B methods using bases derived from atomic orbitals:
+ able to describe core states,
− basis more-or-less restricting wave functions (more basis functions → overdetermined system), limited convergence control.

Real space methods

- **no** translational symmetry,
- arbitrary basis,
- natural for non-periodic systems, no need for tricks.

A our approach:
+ general basis, no wave function shape assumptions/restrictions, excellent convergence control, self-consistent core states,
− ? (early development);

B methods using a non-orthogonal basis related to atomic orbitals (Gaussian, . . .):
+ able to describe core states,
− basis restricting wave functions (more basis functions → overdetermined system), limited convergence control.

Our Solver

Ab-initio electronic structure and total energy calculations.

- **Aim:** to understand and predict material properties from first principles quantum mechanical calculations.
  - We seek a solution to Schrödinger (or Dirac) equation.
- **The solver:** A robust ab-initio real-space code based on:
  - Density functional theory (DFT) [3, 4],
  - Environment-reflecting pseudopotentials [6],
  - Finite element method (FEM).
    - Written (mostly) in Python, built on SfePy code (http::/sfepy.org).
- **Possible applications:**
  - **Non-periodic** substances:
    - clusters,
    - (bio)molecules (possibly with broken charge neutrality),
    - nanocrystalline materials,
    - quantum dots, ...
  - Ab-initio generation of effective potentials for modelling of:
    - defect growth, proteins, ...
      $\rightarrow$ molecular dynamics.

Calculation of Electronic States

The systems of atoms and molecules – the many-particle **Schrödinger equation**

$$H\Psi(e_1, e_2, \ldots, e_n) = \varepsilon\Psi(e_1, e_2, \ldots, e_n) \ .$$

H ... Hamiltonian (energy operator) of the system
e_i ... particles (e.g. electrons)

Too complex to solve!

DFT $\rightarrow$ decompose it into the **Kohn-Sham equations** (in atomic units)

$$\left(-\frac{1}{2}\nabla^2 + V_{\mathsf{H}}(\boldsymbol{r}) + V_{\mathsf{xc}}(\boldsymbol{r}) + \hat{V}(\boldsymbol{r})\right)\psi_i = \varepsilon_i\psi_i \ ,$$

which provide the orbitals Ψ_i that reproduce, with the weights of occupations n_i, the charge density ρ of the original interacting system, as

$$\rho(\boldsymbol{r}) = \sum_i^N n_i|\psi_i(\boldsymbol{r})|^2 \ .$$

strongly nonlinear eigenvalue problem: $\Delta V_{\mathsf{H}} = 4\pi\rho$, $V_{\mathsf{xc}} = V_{\mathsf{xc}}(\rho)$ Note:

Potentials

Electrostatic Potential V_{H}

- Solution of the Poisson problem: $\int_\Omega \nabla v \cdot \nabla V_{\mathrm{H}} = 4\pi \int_\Omega \rho v$.
- (Preconditioned) conjugate gradients work perfectly.

Exchange-Correlation Potential V_{xc}

- Contributions of detailed correlation and exchange to the system energy.
- The actual form of V_{xc} is **not known**! $\Rightarrow$ Local-density approximation (LDA).

Effective Ionic Potential for Electrons $\hat{V}$

- Pseudopotential approach: $\hat{V}$ represents core electrons, separated from valence electrons, together with the nuclear charge.
- A pseudopotential = operator simulating the effect of nucleus + core electrons on electronic states in the energy range of interest.
- Requirements: computational efficiency, accuracy in a wide energy range
- **Environment-reflecting ("all-electron") pseudopotentials**
  - Substantially reduce the number of electrons (i.e. degrees of freedom).
  - Fully relativistic core electrons are "hidden" in pseudopotential ($\Rightarrow$ no need to solve 4-component Dirac equation by FEM).
  - Eliminate high potential gradients.
  - Do not have any additional approximation besides linearization.

Kohn-Sham Eigenvalue Problem

Find functions $\psi_i \in H^1(\Omega)$ such that for all $v \in H_0^1(\Omega)$ holds

$$\underbrace{\int_\Omega \frac{1}{2}\nabla\psi_i \cdot \nabla v \,\mathrm{d}V}_{\boldsymbol{K}\boldsymbol{\psi}_i} + \underbrace{\int_\Omega vV(\psi_i^{\text{previous}})\psi_i \,\mathrm{d}V}_{\boldsymbol{V}(\boldsymbol{\psi}_i^{\text{previous}})\boldsymbol{\psi}_i} = \varepsilon_i \underbrace{\int_\Omega v\psi_i \,\mathrm{d}V}_{\boldsymbol{M}\boldsymbol{\psi}_i} .$$

- Generalized eigenvalue problem with large sparse matrices $\boldsymbol{K}$, $\boldsymbol{M}$.
- Matrix $\boldsymbol{V}$:
  - has a sparse part and a dense part with rank-m update structure $\boldsymbol{U}\boldsymbol{C}_{\texttt{diag}}\boldsymbol{U}^T$;
  - $\boldsymbol{U}$ has about 10 – 30 columns for each atom.
- Number of required ε_i, $\psi_i \gtrapprox$ number of atoms $\times$ number of valence electrons.

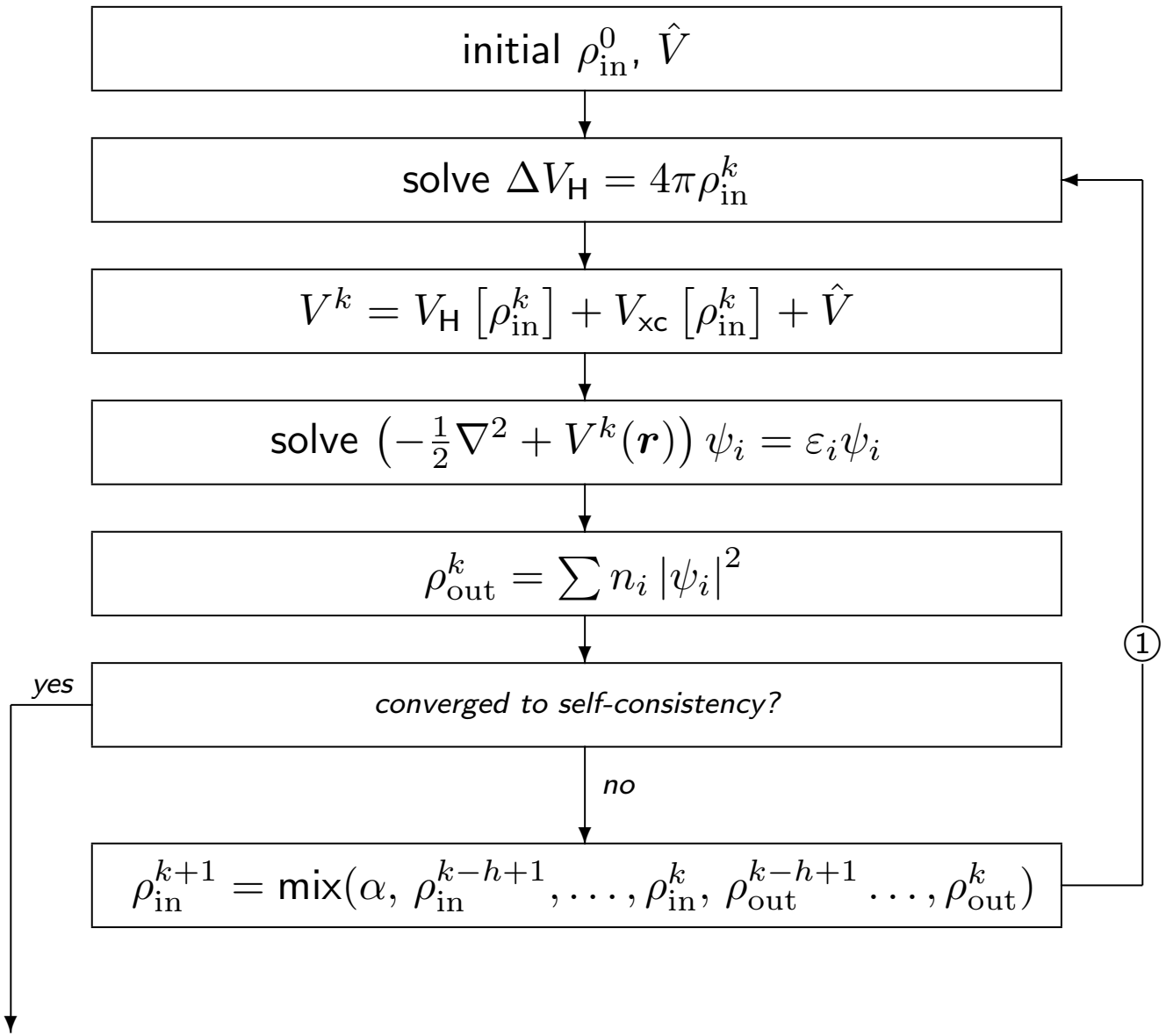
Solvers in use (in various stages):

- BLZPack (block Lanczos), MA57 (LDL matrix decomposition)
- JADAMILU (JAcobi-DAvidson method with Multilevel ILU preconditioning)

Self-consistent solution:

- a fixed point of a function of the charge density ρ;
- Broyden-type quasi-Newton methods, important choice: **mixing algorithms**.

Algorithm


initial ρ_{in}^0, $\hat{V}$
solve $\Delta V_{\text{H}} = 4\pi\rho_{\text{in}}^k$
$V^k = V_{\text{H}}\left[\rho_{\text{in}}^k\right] + V_{\text{xc}}\left[\rho_{\text{in}}^k\right] + \hat{V}$
solve $\left(-\frac{1}{2}\nabla^2 + V^k(\boldsymbol{r})\right)\psi_i = \varepsilon_i\psi_i$
$\rho_{\text{out}}^k = \sum n_i \left|\psi_i\right|^2$
converged to self-consistency?
yes
no
$\rho_{\text{in}}^{k+1} = \text{mix}(\alpha,\ \rho_{\text{in}}^{k-h+1}, \ldots, \rho_{\text{in}}^k,\ \rho_{\text{out}}^{k-h+1} \ldots, \rho_{\text{out}}^k)$
1

Mixing Algorithms

- Mixing: an iterative scheme for obtaining the fixed point of the DFT loop:

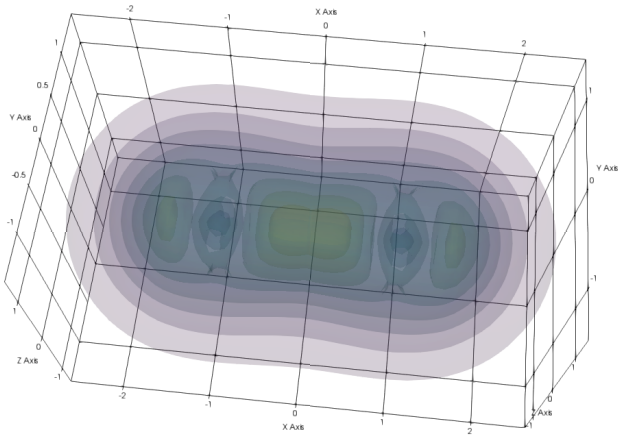
$$DFT(\rho) = \rho, \qquad \rho_{\text{in}}^{k+1} = \text{mix}(\alpha,\, \rho_{\text{in}}^{k-h+1}, \ldots, \rho_{\text{in}}^{k},\, \rho_{\text{out}}^{k-h+1} \ldots, \rho_{\text{out}}^{k})$$

- Linear mixing: $\rho_{\text{in}}^{k+1} = (1-\alpha)\rho_{\text{in}}^{k} + \alpha\rho_{\text{out}}^{k}$
- The following mixing algorithms were numerically tested:
  - **Anderson**: also called Pulay [1, 5]
    - $\rho_{\text{in}}^{k+1} = \sum_{i=k-h+1}^{k} \theta_i \left((1-\alpha)\rho_{\text{in}}^{k} + \alpha\rho_{\text{out}}^{k}\right)$, $\sum \theta_i = 1$,
    - θ_i minimize the linear combination $\sum \theta_i r_i$ of residuals $r^i = \rho_{\text{out}}^{i} - \rho_{\text{in}}^{i}$.
  - **GR-Pulay** (Guaranteed Residual Pulay): [2]
    - Alternates the Anderson step with $\alpha = 0$ and the linear step with $\alpha = 1$.
    - Remembers only Anderson steps densities pairs.
  - **GR-Pulay-LM**: our modification of GR-Pulay
    - Uses the general linear step ($0 \leq \alpha \leq 1$).
  - **hybrid**: newly proposed adaptable hybrid scheme
    - As long as the computation converges, repeat the Anderson step with the given α.
    - If the Anderson step diverges, replace the "diverged densities pair" with the ones obtained by the GR-Pulay step – the Anderson step with $\alpha = 0$.

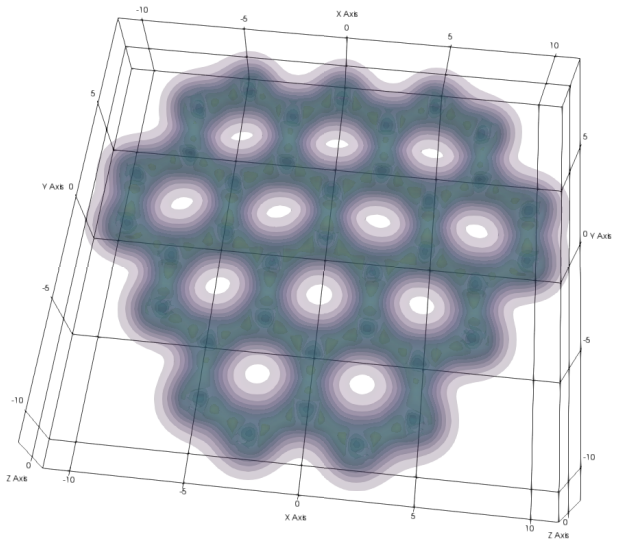
Test Systems

- An easy to solve system: a nitrogen molecule N_2.
  - 10 eigenpairs required.
- A more complex system: a graphene fragment.
  - 84 eigenpairs required.
- The history length for the mixing h was 6.

Charge densities ρ of the test systems:



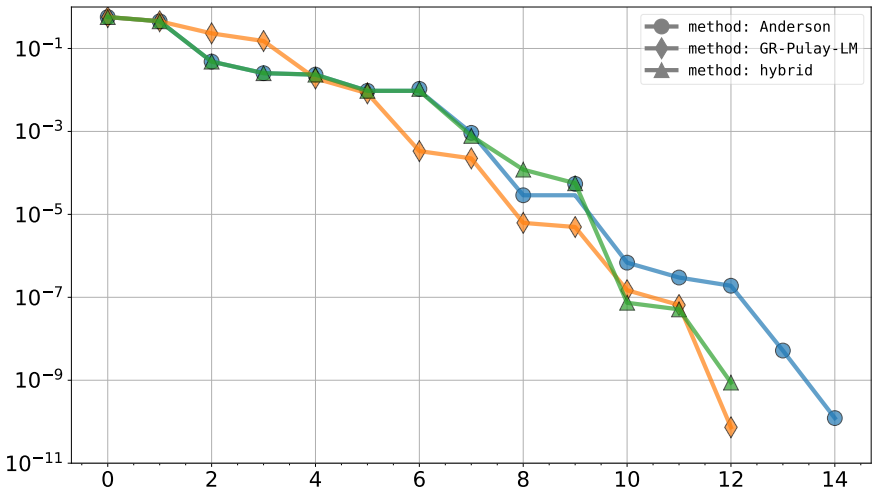
a N_2 molecule



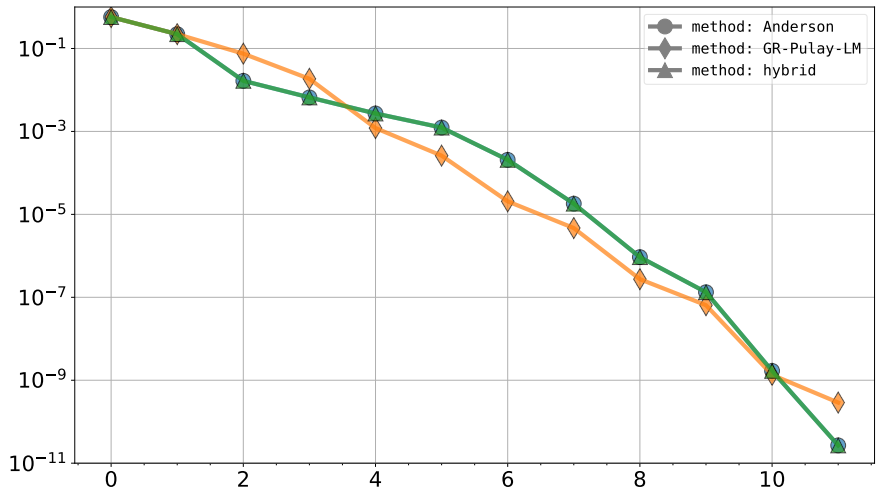
a graphene fragment

Results: Nitrogen Molecule

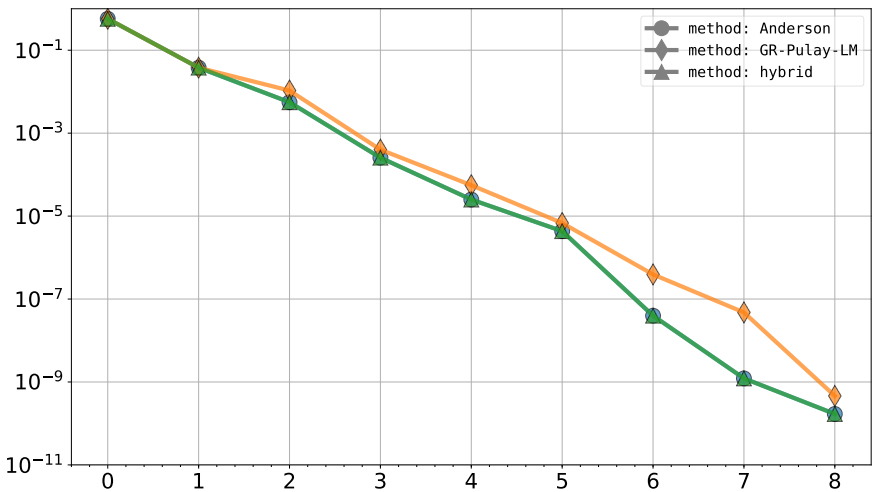
$\alpha = 0.1$



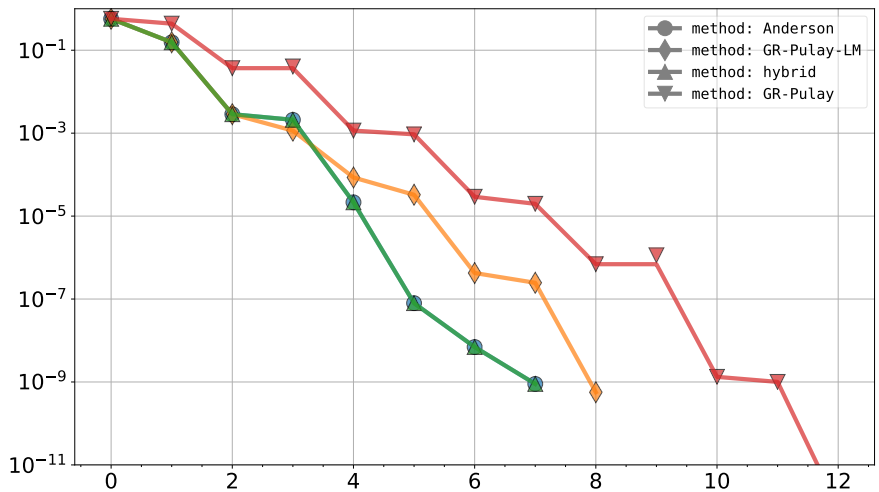
$\alpha = 0.3$



$\alpha = 0.7$



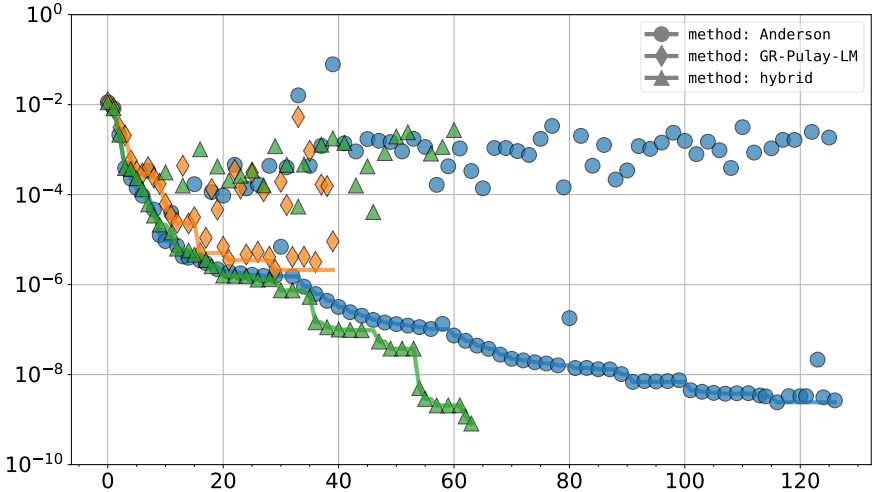
$\alpha = 0.9$



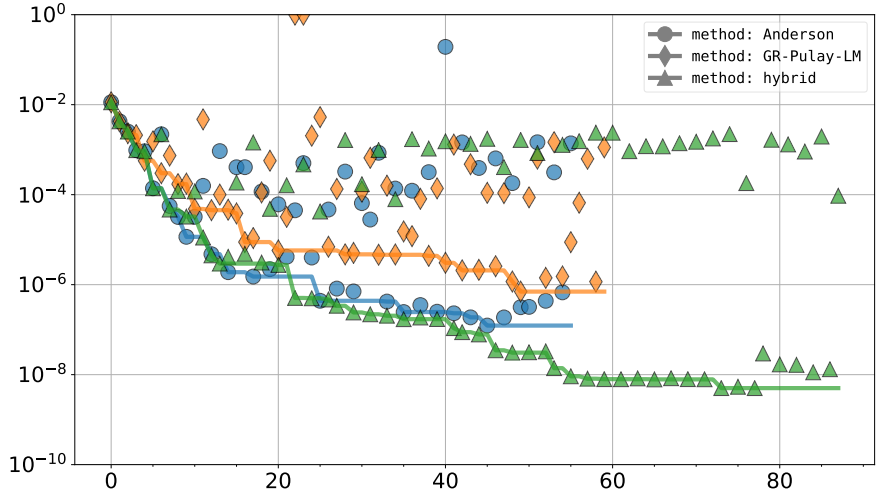
Results: Graphene Fragment

NEW TECHNOLOGIES
RESEARCH CENTRE
UNIVERSITY
OF WEST BOHEMIA

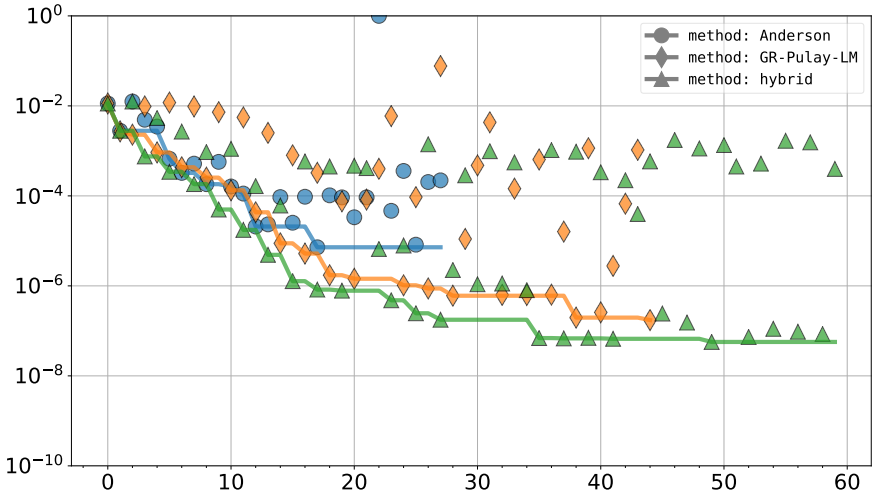
$\alpha = 0.1$



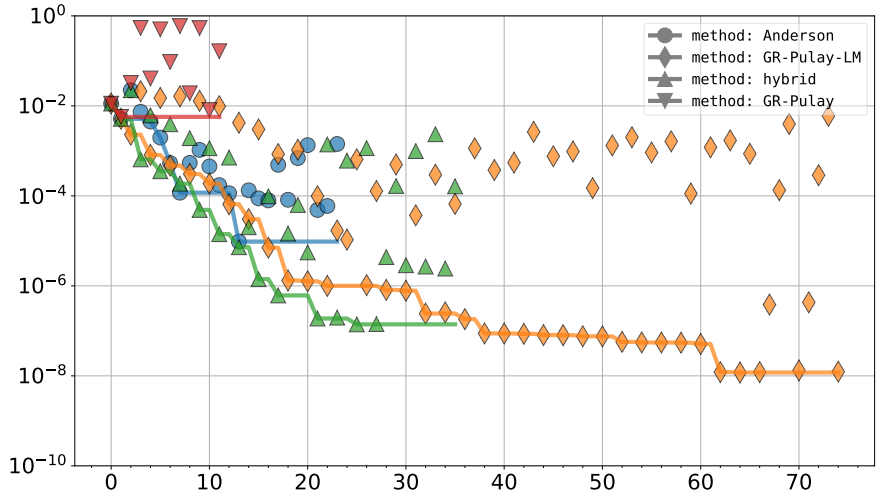
$\alpha = 0.3$



$\alpha = 0.7$



$\alpha = 0.9$



Discussion

- A computer implementation of a new robust ab-initio real-space code:
  - density functional theory + environment-reflecting pseudopotentials + FEM.
- Mixing schemes for fixed-point iterations of the DFT loop:
  - The proposed adaptable hybrid mixing scheme our performs better or as well as the best other scheme in both test problems.
  - Our modification of the GR-Pulay algorithm is competitive especially for the higher mixing parameter values in the complex test problem.
  - Future work: to compare our scheme to other recently published schemes.
- Current work: suitable preconditioning of the eigenvalue problem solver.

Ack.: The work was supported by the Czech Science Foundation, grant project GA17-12925S. The first author acknowledges the support by CEDAMNF project, reg. no. CZ. 02.1.01/0.0/0.0/15_003/0000358, co-funded by the ERDF as part of the Ministry of Education, Youth and Sports OP RDE programme.

Refs:

[1] Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965.

[2] DR Bowler and MJ Gillan. An efficient and robust technique for achieving self consistency in electronic structure calculations. *Chemical Physics Letters*, 325(4):473–476, 2000.

[3] R. M. Martin. *Electronic Structure: Basic Theory and Practical Methods*. Cambridge University Press, 2005.

[4] W. E. Pickett. Pseudopotential methods in condensed matter applications. *Comp. Phys. Reports*, 9:115–198, 1989.

[5] P. Pulay. Improved scf convergence acceleration. *J. Comput. Chem.*, 3:55 6–560, 1982.

[6] J. Vackář and A. Šimůnek. Adaptability and accuracy of all-electron pseudopotentials. *Phys. Rev. B*, 67, 2003. 125113.

http://sfepy.org