SfePy - Simple Finite Elements in Python

Short Introduction . . .

Robert Cimrman[1] Ondřej Čertík[2]

[1]Department of Mechanics & New Technologies Research Centre
University of West Bohemia in Plzeň, Czech Republic

[2]Institute of Physics, Academy of Sciences of the Czech Republic
& Charles University in Prague, Czech Republic

EuroSciPy 2008, July 27, Leipzig, Germany

Outline

Introduction

- SfePy = general finite element analysis software
  - solving systems of PDEs
- BSD open-source license
- available at
  - `http://sfepy.org` (developers)
    - mailing lists, issue (bug) tracking
    - we encourage and support everyone who joins!
  - `http://sfepy.kme.zcu.cz` (project information)
- selected applications:
  - homogenization of porous media (parallel flows in a deformable porous medium)
  - acoustic band gaps (homogenization of a strongly heterogenous elastic structure: phononic materials)
  - shape optimization in incompressible flow problems
  - finite element formulation of Schrödinger equation

Notes on Programming Languages

Rough Division

- compiled (fortran, C, C++, Java, . . .)

Pros

- speed
- large code base (legacy codes)
- tradition

Cons

- (often) complicated build process, recompile after any change
- low-level ⇒ lots of lines to get basic stuff done
- code size ⇒ maintenance problems
- static!

- interpreted or scripting (sh, tcl, matlab, perl, ruby, python, . . .)

Pros

- no compiling
- (very) high-level ⇒ a few of lines to get (complex) stuff done
- code size ⇒ easy maintenance
- dynamic!
- (often) large code base

Cons

- many are relatively new
- not known as useful in many scientific communities
- lack of speed

Mixing Languages — Best of Both Worlds

- low level code (C or fortran): element matrix evaluations, costly mesh-related functions, ...
- high level code (Python): logic of the code, particular applications, configuration files, problem description files

www.python.org

SfePy = Python + C (+ fortran)

- notable features:
    - small size (complete sources are just about 1.3 MB, June 2008)
    - fast compilation
    - problem description files in pure Python
    - problem description form similar to mathematical description "on paper"

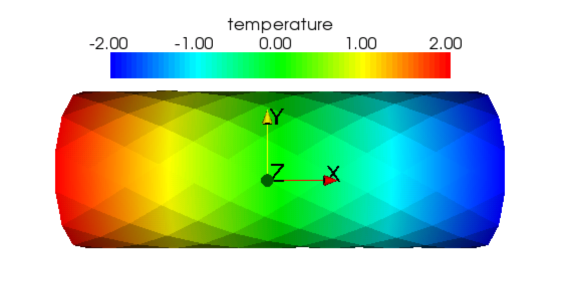
Software Dependencies

- to install and use SfePy, several other packages or libraries are needed:
    - NumPy and SciPy: free (BSD license) collection of numerical computing libraries for Python
        - enables Matlab-like array/matrix manipulations and indexing
    - other: UMFPACK, Pyparsing, Matplotlib, Pytables (+ HDF5), swig
    - visualization of results: ParaView, MayaVi2, or any other VTK-capable viewer
- missing:
    - free (BSD license) 3D mesh generation and refinement tool
    - ...can use netgen, tetgen

Introduction

- problem description file is a regular Python module, i.e. all Python syntax and power is accessible
- consists of entities defining:
  - fields of various FE approximations, variables
  - equations in the weak form, quadratures
  - boundary conditions (Dirichlet, periodic, "rigid body")
  - FE mesh file name, options, solvers, ...
- simple example: the Laplace equation:

$$c\Delta u = 0 \text{ in } \Omega, \quad u = \bar{u} \text{ on } \Gamma, \text{ weak form: } \int_\Omega c\, \nabla u \cdot \nabla v = 0, \quad \forall v \in V_0$$



Problem Description File

Solving Laplace Equation — FE Approximations

- mesh → define FE approximation to Ω:

```
fileName_mesh = 'simple.mesh'
```

- fields → define space V_h:

```
field_1  = {
           'name'    :  'temperature',
           'dim'     :  (1,1),
           'domain'  :  'Omega',
           'bases'   :  'Omega' :  '3_4_P1'
}
```

'3_4_P1' means P1 approximation, in 3D, on 4-node FEs (tetrahedra)

- variables → define u_h, v_h:

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

Problem Description File

Solving Laplace Equation — Boundary Conditions

- regions $\rightarrow$ define domain Ω, regions Γ_{left}, Γ_{right}, $\Gamma = \Gamma_{\text{left}} \cup \Gamma_{\text{right}}$:
  - $_h$ omitted from now on ...

```
regions  = {
          'Omega'           :  ('all', {}),
          'Gamma_Left'      :  ('nodes in (x < 0.0001)', {}),
          'Gamma_Right'     :  ('nodes in (x > 0.0999)', {}),
}
```

- Dirichlet BC $\rightarrow$ define $\bar{u}$ on Γ_{left}, Γ_{right}:

```
ebcs  = {
        't_left'    :  ('Gamma_Left', 'u.0' :  2.0),
        't_right'   :  ('Gamma_Right', 'u.all' :  -2.0),
}
```

Problem Description File

Solving Laplace Equation — Equations

- materials → define c:

```
material_1 = {
    'name'   : 'm',
    'mode'   : 'here',
    'region' : 'Omega',
    'c'      : 1.0,
}
```

- integrals → define numerical quadrature:

```
integral_1 = {
    'name'       : 'i1',
    'kind'       : 'v',
    'quadrature' : 'gauss_o1_d3',
}
```

- equations → define what and where should be solved:

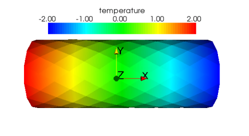
```
equations = {
    'eq' : 'dw_laplace.i1.Omega( m.c, v, u ) = 0'
}
```

Running SfePy

```
$ ./simple.py input/poisson.py
sfe: reading mesh...
sfe: ...done in 0.02 s
sfe: setting up domain edges...
sfe: ...done in 0.02 s
sfe: setting up domain faces...
sfe: ...done in 0.02 s
sfe: creating regions...
sfe:     leaf Gamma_Right region_4
sfe:     leaf Omega region_1000
sfe:     leaf Gamma_Left region_03
sfe: ...done in 0.07 s
sfe: equation "Temperature":
sfe: dw_laplace.i1.Omega( coef.val, s, t ) = 0
sfe: describing geometries...
sfe: ...done in 0.01 s
sfe: setting up dof connectivities...
sfe: ...done in 0.00 s
sfe: using solvers:
                nls: newton
                 ls: ls
sfe: matrix shape: (300, 300)
sfe: assembling matrix graph...
sfe: ...done in 0.01 s
sfe: matrix structural nonzeros: 3538 (3.93e-02% fill)
sfe: updating materials...
sfe:     coef
sfe: ...done in 0.00 s
sfe: nls: iter: 0, residual: 1.176265e-01 (rel: 1.000000e+00)
sfe:   rezidual:    0.00 [s]
sfe:      solve:    0.01 [s]
sfe:     matrix:    0.00 [s]
sfe: nls: iter: 1, residual: 9.921082e-17 (rel: 8.434391e-16)
```

- top level of SfePy code is a collection of executable scripts tailored for various applications
- `simple.py` is dumb script of brute force, attempting to solve any equations it finds by the Newton method
- ...exactly what we need here (solver options were omitted in previous slides)

↓

→ 

Verification of Numerical Results

- to verify numerical results we use method of manufactured solutions: for example, for Poisson's equation $\mathrm{div}(\mathrm{grad}(u)) = f$:
  1. make up a solution, e.g. $\sin 3x \cos 4y$
  2. compute corresponding f, here $f = 25 \sin 3x \cos 4y$, and boundary conditions
  3. solve numerically and compare
- manual derivation of f tedious $\longrightarrow$ SymPy
  - each term class annotated by a corresponding symbolic expression
  - example: anisotropic diffusion term

```
symbolic = { 'expression': 'div( K * grad( u ) )',
             'map' : {'u' : 'state', 'K' : 'material'}}
```

  - f is built by substituting the manufactured solution into the expressions and subsequent evaluation in FE nodes
  - work in progress

Optimal Flow Problem

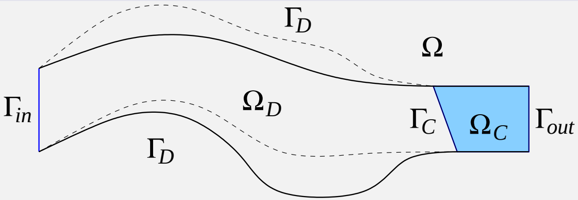
Problem Setting

Objective Function

$$\Psi(u) \equiv \frac{\nu}{2} \int_{\Omega_c} |\nabla u|^2 \longrightarrow \min$$

- minimize gradients of solution (e.g. losses) in $\Omega_c \subset \Omega$
- by moving design boundary $\Gamma \subset \partial\Omega$
- perturbation of Γ by vector field $\mathcal{V}$

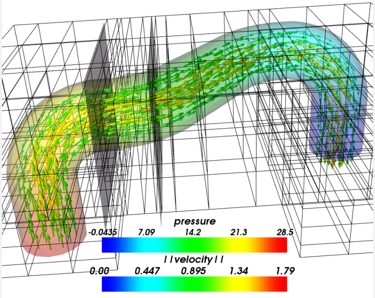
$$\Omega(t) = \Omega + \{t\mathcal{V}(x)\}_{x\in\Omega} \quad \text{where } \mathcal{V} = 0 \text{ in } \bar{\Omega}_c \cup \partial\Omega \setminus \Gamma$$

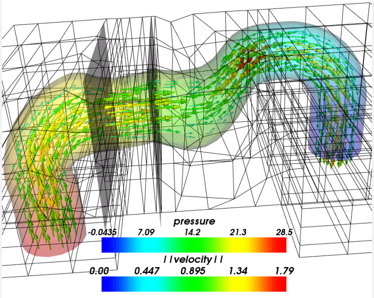


Optimal Flow Problem

Example Results

- flow and domain control boxes, left: initial, right: final





- Ω_C between two grey planes
- work in progress . . .

Direct Problem

. . . paper ↔ input file

- weak form of Navier-Stokes equations: ? $\mathbf{u} \in \mathbf{V}_0(\Omega)$, $p \in L^2(\Omega)$ such that

$$\begin{aligned} a_\Omega\left(\mathbf{u}, \mathbf{v}\right) + c_\Omega\left(\mathbf{u}, \mathbf{u}, \mathbf{v}\right) - b_\Omega\left(\mathbf{v}, p\right) &= g_{\Gamma_{\rm out}}\left(\mathbf{v}\right) \quad \forall \mathbf{v} \in \mathbf{V}_0 \ , \\ b_\Omega\left(\mathbf{u}, q\right) &= 0 \quad \forall q \in L^2(\Omega) \ . \end{aligned} \tag{1}$$

- in SfePy syntax:

```
equations = {
'balance'              :  """
                       dw_div_grad.i2.Omega( fluid.viscosity, v, u )
                       + dw_convect.i2.Omega( v, u )
                       - dw_grad.i1.Omega( v, p ) = 0""",
'incompressibility'    :  """
                       dw_div.i1.Omega( q, u ) = 0""",
}
```

Adjoint Problem

...paper $\leftrightarrow$ input file

- KKT conditions $\delta_{\mathbf{u},p}\mathcal{L} = 0$ yield adjoint state problem for $\mathbf{w}$, r:

$$\delta_{\mathbf{u}}\mathcal{L} \circ \mathbf{v} = 0 = \delta_u \Psi(\mathbf{u}, p) \circ \mathbf{v} + a_\Omega\left(\mathbf{v}, \mathbf{w}\right) + c_\Omega\left(\mathbf{v}, \mathbf{u}, \mathbf{w}\right) + c_\Omega\left(\mathbf{u}, \mathbf{v}, \mathbf{w}\right) + b_\Omega\left(\mathbf{v}, r\right) ,$$

$$\delta_p\mathcal{L} \circ q = 0 = \delta_p \Psi(\mathbf{u}, p) \circ q - b_\Omega\left(\mathbf{w}, q\right) , \forall \mathbf{v} \in \mathbf{V}_0, \text{ and } \forall q \in L^2(\Omega).$$

- in SfePy syntax:

```
equations = {
'balance'              :  """
                       dw_div_grad.i2.Omega( fluid.viscosity, v, w )
                       + dw_adj_convect1.i2.Omega( v, w, u )
                       + dw_adj_convect2.i2.Omega( v, w, u )
                       + dw_grad.i1.Omega( v, r )
                       = - ''δ_u Ψ(u,p) ∘ v''""",
'incompressibility'    :  """
                       dw_div.i1.Omega( q, w ) = 0""",
}
```

Finite Element Formulation of Schrödinger Equation

One particle Schrödinger equation:

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + V\right)\psi = E\psi\,.$$

FEM:

$$(K_{ij} + V_{ij})\,q_j = EM_{ij}q_j + F_i\,,$$

$$\begin{aligned}
V_{ij} &= \int \phi_i V \phi_j \,\mathrm{d}V\,,\\
M_{ij} &= \int \phi_i \phi_j \,\mathrm{d}V\,,\\
K_{ij} &= \frac{\hbar^2}{2m}\int \nabla\phi_i \cdot \nabla\phi_j \,\mathrm{d}V\,,\\
F_i &= \frac{\hbar^2}{2m}\oint \frac{\mathrm{d}\psi}{\mathrm{d}n}\phi_i \,\mathrm{d}S\,.
\end{aligned}$$

Usually we set $F_i = 0$.

Particle in the Box

$$V(x) = \begin{cases} 0, & \text{inside the box} \quad a \times a \times a \\ \infty, & \text{outside} \end{cases}$$

Analytic solution:

$$E_{n_1 n_2 n_3} = \frac{\pi^2}{2a^2}\left(n_1^2 + n_2^2 + n_3^2\right)$$

where $n_i = 1, 2, 3, \dots$ are independent quantum numbers. We chose $a = 1$, i.e.: $E_{111} = 14.804$, $E_{211} = E_{121} = E_{112} = 29.608$, $E_{122} = E_{212} = E_{221} = 44.413$, $E_{311} = E_{131} = E_{113} = 54.282$ $E_{222} = 59.217$, $E_{123} = E_{\text{perm.}} = 69.087$.

Numerical solution ($a = 1$, 24702 nodes):

E	1	2-4	5-7	8-10	11	12-
theory	14.804	29.608	44.413	54.282	59.217	69.087
FEM	14.861	29.833	44.919	55.035	60.123	70.305
		29.834	44.920	55.042		70.310
		29.836	44.925	55.047		. . .

3D Harmonic Oscillator

$$V(r) = \begin{cases} \frac{1}{2}\omega^2 r^2, & \text{inside the box} \quad a \times a \times a \\ \infty, & \text{outside} \end{cases}$$

Analytic solution in the limit $a \to \infty$:

$$E_{nl} = \left(2n + l + \frac{3}{2}\right)\omega$$

where $n, l = 0, 1, 2, \ldots$. Degeneracy is $2l + 1$, so: $E_{00} = \frac{3}{2}$, triple $E_{01} = \frac{5}{2}$, $E_{10} = \frac{7}{2}$, quintuple $E_{02} = \frac{7}{2}$ triple $E_{11} = \frac{9}{2}$, quintuple $E_{12} = \frac{11}{2}$:

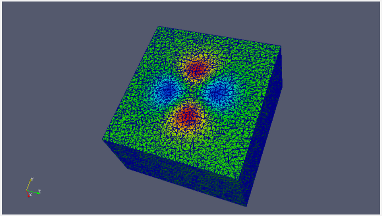
Numerical solution ($a = 15$, $\omega = 1$, 290620 nodes):

E	1	2-4	5-10	11-
theory	1.5	2.5	3.5	4.5
FEM	1.522	2.535	3.554	4.578
		2.536	3.555	4.579
		2.536	3.555	4.579
			3.555	. . .
			3.556	
			3.556	

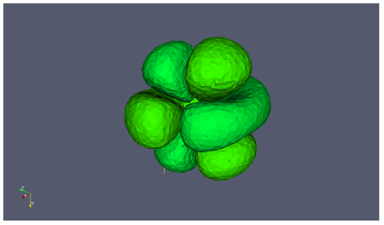
3D Harmonic Oscillator

Eigenvectors:

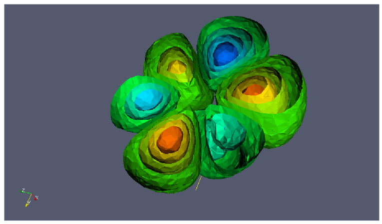
0th



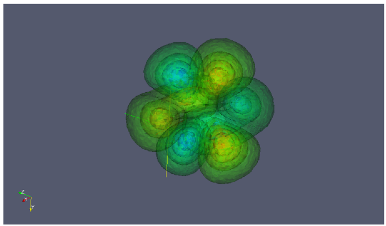
12th



10th



12th



Hydrogen Atom

$$V(r) = \begin{cases} -\frac{1}{r}, & \text{inside the box} \quad a \times a \times a \\ \infty, & \text{outside} \end{cases}$$

Analytic solution in the limit $a \to \infty$:

$$E_n = -\frac{1}{2n^2}$$

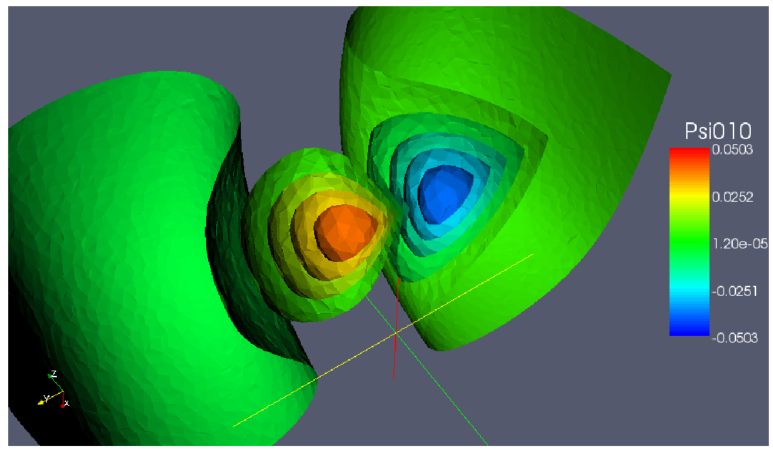
where $n = 1, 2, 3, \ldots$. Degeneracy is n^2, so: $E_1 = -\frac{1}{2} = -0.5$, $E_2 = -\frac{1}{8} = -0.125$, $E_3 = -\frac{1}{18} = -0.055$, $E_4 = -\frac{1}{32} = -0.031$.

Numerical solution ($a = 15$, 160000 nodes):

E	1	2-5	6-14	15-
theory	-0.5	-0.125	-0.055	-0.031
FEM	-0.481	-0.118	-0.006	. . .

Hydrogen Atom

11th eigenvalue (calculated: -0.04398532, exact: -0.056), on the mesh with 976 691 tetrahedrons and 163 666 nodes, for the hydrogen atom (V=-1/r).



Conclusion

What is done

- basic FE element engine:
  - finite-dimensional approximations of continuous fields
  - variables, boundary conditions, FE assembling
  - equations, terms, regions
  - materials, material caches
- various solvers accessed via abstract interface
- unit tests, automatic documentation generation
- mostly linear problems, but multiphysical

What is not done

- general FE engine, possibly with symbolic evaluation (SymPy)
- good documentation
- fast problem-specific solvers (!)
- adaptive mesh refinement (!)
- parallelization (petsc4py)

What will not be done (?)

- GUI
- real symbolic parsing/evaluation of equations

`http://sfepy.org`

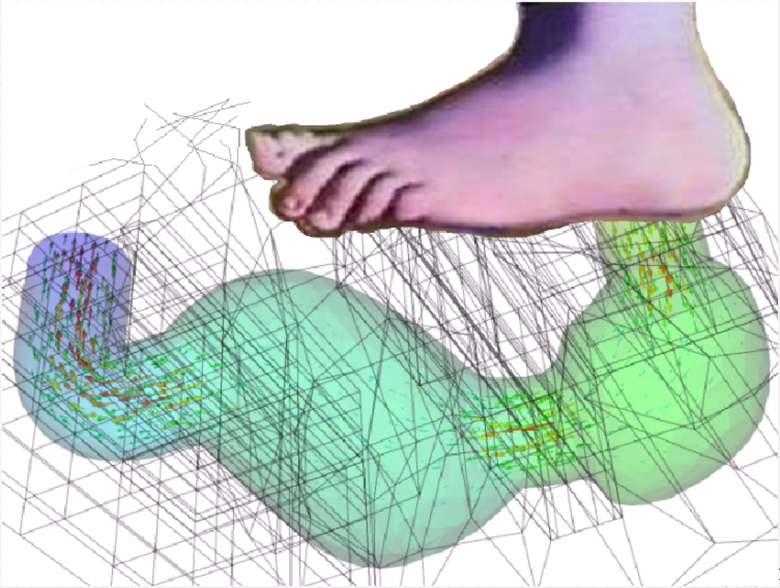
Yes, the final slide!

Acknowledgements

The work on various parts of SfePy has been supported by the following grants and research projects funded by several agencies in the Czech Republic:

- Robert Cimrman:
  - grant project GAČR 101/07/1471, entitled "Finite element modelling of linear, non-linear and multiscale effects in wave propagation in solids and heterogeneous media"
  - research project MŠMT 1M06031
  - research project MŠMT 4977751303
- Ondřej Čertík:
  - research center project LC06040
  - grant project GAČR IAA100100637

This is not a slide!


[1]

[1]Do you like Monty Python's Flying Circus? It helps! (Python FAQ 1.1.17)