# Short introduction of SfePy

and its scientific applications

Robert Cimrman[1] & others[2]

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

[2]other guys from around the world, notably:
Eduard Rohan (KME),
Vladimír Lukeš (KME),
Ondřej Čertík (Reno University)

June 8, 2009

## Outline

## Introduction

- SfePy = simple finite elements in Python
    - 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)
    - acoustic waves in thin perforated layers
    - shape optimization in incompressible flow problems
    - finite element formulation of Schrödinger equation

# Notes on Programming Languages

## Rough Division

- compiled (fortran, C, C++, Java, . . . )
  - + speed
  - + large code base (legacy codes)
  - + tradition
  - - recompile after any change (often slow)
  - - low-level ⇒ lots of lines
  - - code size ⇒ maintenance problems
  - - static!
- interpreted or scripting (sh, tcl, matlab, perl, ruby, python, . . . )
  - + no compiling
  - + (very) high-level ⇒ “short” programs
  - + code size ⇒ easy maintenance
  - + dynamic!
  - + (often) large code base
  - - 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): element matrix evaluations, ...
- high level code (Python): logic of the code, particular applications, configuration files, problem description files

`www.python.org`

python™

**SfePy = Python + C (+ fortran)**

- notable features:
  - small size (complete sources + examples with meshes are just about 4.6 MB, May 2009)
  - problem description files in pure Python
  - problem description form similar to mathematical description "on paper"
  - `isfepy` ...interactive shell based in IPython

## Python
Batteries Included

*Python® is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code.*

...

"batteries included"

References: NASA uses Python, so does Rackspace, Industrial Light and Magic, AstraZeneca, Honeywell, and many others.

`http://wiki.python.org/moin/NumericAndScientific`

## Python
Origin of the Name

1.1.16 Why is it called Python?

*At the same time he began implementing Python, Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus" (a BBC comedy series from the seventies, in the unlikely case you didn't know). It occurred to him that he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.*

1.1.17 Do I have to like "Monty Python's Flying Circus"?

*No, but it helps. :)*

. . . General Python FAQ

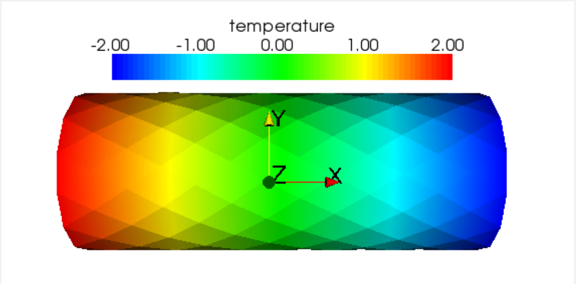
## Software Dependencies

- 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
  - isfepy requires IPython
- missing:
  - free (BSD license) 3D mesh generation and refinement tool
  - . . . can use netgen, tetgen
- but otherwise there are completely free tools for everything!

## Setting

- 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 $\longrightarrow$ define FE approximation to $\Omega$:

  ```
  filename_mesh = 'simple.mesh'
  ```

- regions $\longrightarrow$ define domain $\Omega$, regions $\Gamma_{\mathrm{left}}$, $\Gamma_{\mathrm{right}}$, $\Gamma = \Gamma_{\mathrm{left}} \cup \Gamma_{\mathrm{right}}$:
  - $_h$ omitted from now on ...

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

- fields $\longrightarrow$ define space $V_h$:

  ```
  fields  = {
          'temperature'  :  ((1,1), 'real', 'Omega',
                              {'Omega' :  '3_4_P1'})
  }
  ```

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

- variables $\longrightarrow$ define $u_h$, $v_h$:

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

## Problem Description File

Solving Laplace Equation — Equations

- materials $\rightarrow$ define $c$:

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

- integrals $\rightarrow$ define numerical quadrature:

```
integrals   = {
            'i1'   :  ('v', 'gauss_o1_d3')
}
```

- equations $\rightarrow$ define what and where should be solved:

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

- Dirichlet BCs $\rightarrow$ define $\bar{u}$ on $\Gamma_{\mathrm{left}}, \Gamma_{\mathrm{right}}$:

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

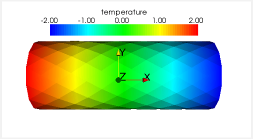
# Running SfePy

```
$ ./simple.py input/poisson.py
sfepy: reading mesh (database/simple.mesh)...
sfepy: ...done in 0.00 s
sfepy: setting up domain edges...
sfepy: ...done in 0.02 s
sfepy: setting up domain faces...
sfepy: ...done in 0.03 s
sfepy: creating regions...
sfepy:     leaf Gamma_Right region_Gamma_Right
sfepy:     leaf Omega region_Omega
sfepy:     leaf Gamma_Left region_Gamma_Left
sfepy: ...done in 0.04 s
sfepy: equation "Temperature":
sfepy: dw_laplace.i1.Omega( coef.val, s, t ) = 0
sfepy: setting up dof connectivities...
sfepy: ...done in 0.00 s
sfepy: describing geometries...
sfepy: ...done in 0.00 s
sfepy: using solvers:
              nls: newton
               ls: ls
sfepy: matrix shape: (300, 300)
sfepy: assembling matrix graph...
sfepy: ...done in 0.01 s
sfepy: matrix structural nonzeros: 3538 (3.93e-02% fill)
sfepy: updating materials...
sfepy:     coef
sfepy: ...done in 0.00 s
sfepy: nls: iter: 0, residual: 1.176265e-01 (rel: 1.000000e+00)
sfepy:   rezidual:    0.00 [s]
sfepy:      solve:    0.00 [s]
sfepy:     matrix:    0.00 [s]
sfepy: nls: iter: 1, residual: 1.173819e-16 (rel: 9.979198e-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)

$\downarrow$

$\longrightarrow$



## Time-dependent Problem

- adding time to the Laplace equation:

$$\frac{\partial u}{\partial t} - c\Delta u = 0 \text{ in } \Omega, \quad u(\cdot, x) = \bar{u} \text{ on } \Gamma, \quad u(0, \cdot) = 0,$$

$$\text{weak form: } \int_\Omega v \cdot \frac{\partial u}{\partial t} + \int_\Omega c \, \nabla u \cdot \nabla v = 0, \quad \forall v \in V_0$$

- variables

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

- equations

```
equations   = {
            'eq'    :  """dw_mass_scalar.i1.Omega( v, du/dt )
                         + dw_laplace.i1.Omega( m.c, v, u ) = 0"""
}
```

- + define time-stepping solver options

$$\downarrow$$

## Top-level Scripts

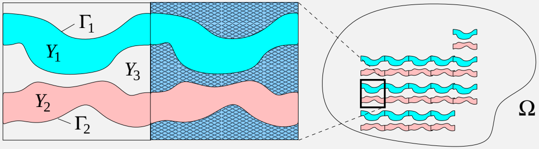
Main scripts / applications (version 2009.2):

- `runTests.py` . . . run all/selected unit tests
- `simple.py` . . . generic problem solver (stationary, time-dependent)
- `isfepy` . . . interactive SfePy interpreter
- `postproc.py` . . . quick mayavi2-based visualization of results
- `eigen.py` . . . application: acoustic band gaps in strongly heterogenous media
- `schroedinger.py` . . . Schrödinger equation solver
- `probe.py` . . . data probes plotter

Auxiliary:

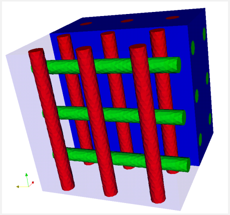
- `extractor.py` . . . HDF5 $\rightarrow$ VTK data extractor
- `findSurf.py` . . . mesh surface extractor
- `gen` . . . LaTeX documentation generator, requires pexpect, lxml
- `genPerMesh.py` . . . scale and periodically repeat a reference volume mesh

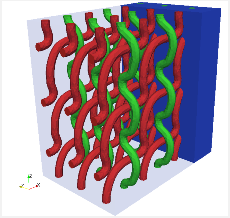
## Porous Media



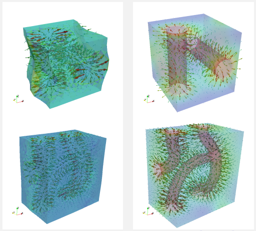
- modeling blood perfusion in muscles, brain
- compact bone poromechanics
- fading memory effects
- stress recovery at micro-level

microstructure: two channels + porous matrix





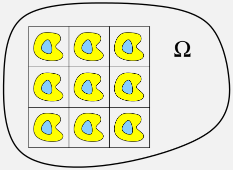
microscopic corrector functions



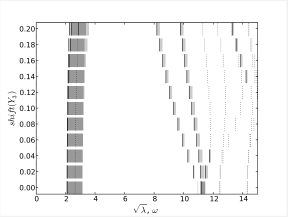
## Prediction of Acoustic Band Gaps

- phononic materials (similar to photonic crystals but for sound)
- forbidden frequency ranges: acoustic band gaps
- example applications: smart materials, wave guides, silencers
- strongly heterogeneous materials
  - constitutive parameters (elasticity, density) depend on scale
  - much easier band gap determination in low frequency range than classical methods

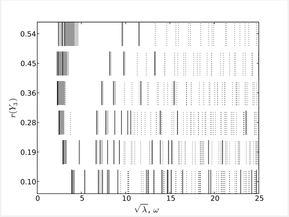
strong heterogeneity: rigid inclusion $Y_R$ coated by compliant material



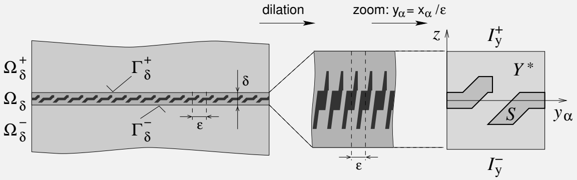
influence of $Y_R$ excentricity



influence of $Y_R$ size

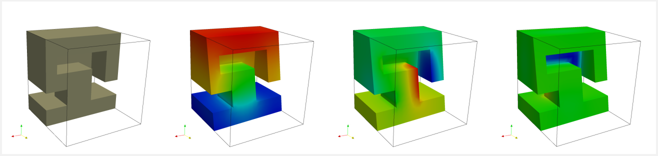


## Acoustic Waves in Thin Perforated Layer



- goal: to replace the perforated layer by a "homogenized surface"
- arbitrary geometry in the layer
- systematic computation of the impedance properties
- allows for optimal design of perforations

reference volume element, examples of microscopic corrector functions



## 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)
  - GUI
  - real symbolic parsing/evaluation of equations

`http://sfepy.org`

## This is not a slide!