(Multiscale) Modelling With SfePy

Random Remarks . . .

Robert Cimrman & Eduard Rohan & others

Department of Mechanics & New Technologies Research Centre
University of West Bohemia
Plzeň, Czech Republic

PANM 14, June 4, 2008, Dolní Maxov

Outline

Introduction

- SfePy = general finite element analysis software
- 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

Introduction

- SfePy = general finite element analysis software
- 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

Programming Languages

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

Pros

- speed, speed, speed, . . . , did I say 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, . . .)

Programming Languages

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

Pros

- speed, speed, speed, . . . , did I say 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
- lack of speed (very naturally so!)

Programming Languages

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

Pros

- speed, speed, speed, . . . , did I say 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
- lack of speed, or even utterly slow!!

Programming Languages

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

Pros

- speed, speed, speed, . . . , did I say 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
- lack of speed, or even utterly slow!

Programming Languages

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

Pros

- speed, speed, speed, . . . , did I say 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
- lack of speed, or even utterly slow!

Programming Languages

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

Pros

- speed, speed, speed, . . . , did I say 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
- lack of speed, or even utterly slow!

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"

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"

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"

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

Python
References

NASA uses Python. . .

. . . so does Rackspace, Industrial Light and Magic, AstraZeneca, Honeywell, and many others.

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

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

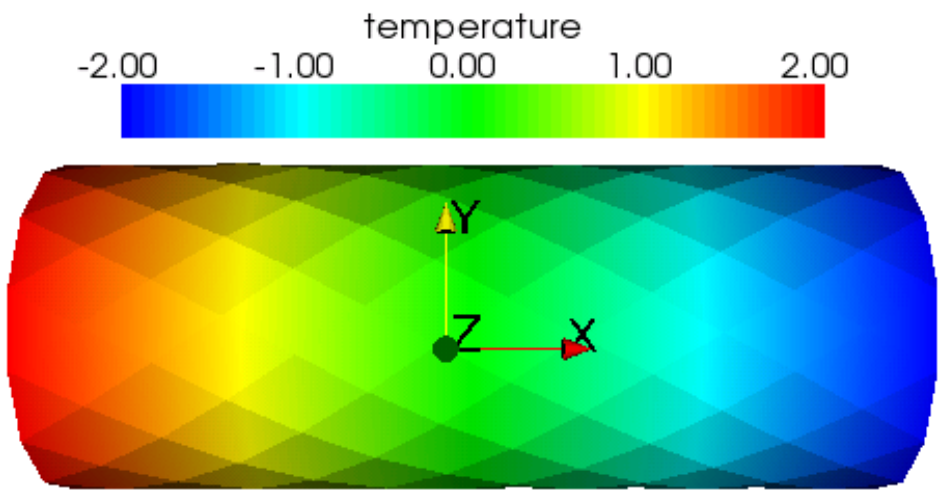
Software Dependencies

- 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 — 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 — 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 $\longrightarrow$ 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 $\longrightarrow$ 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 — Boundary Conditions

- regions → 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 → 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'
              }
```

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'
}
```

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',
               }
```

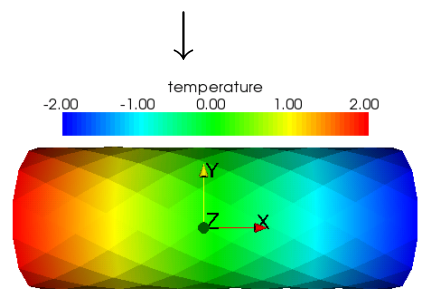
- equations → define what and where should be solved:

```
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)



Optimal Flow Problem

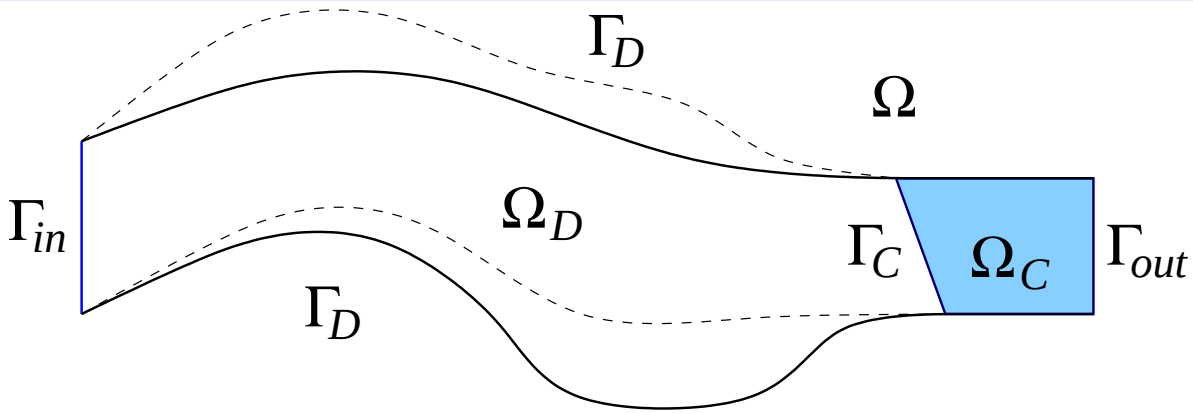
Problem Setting

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 $\mathcal{V}$

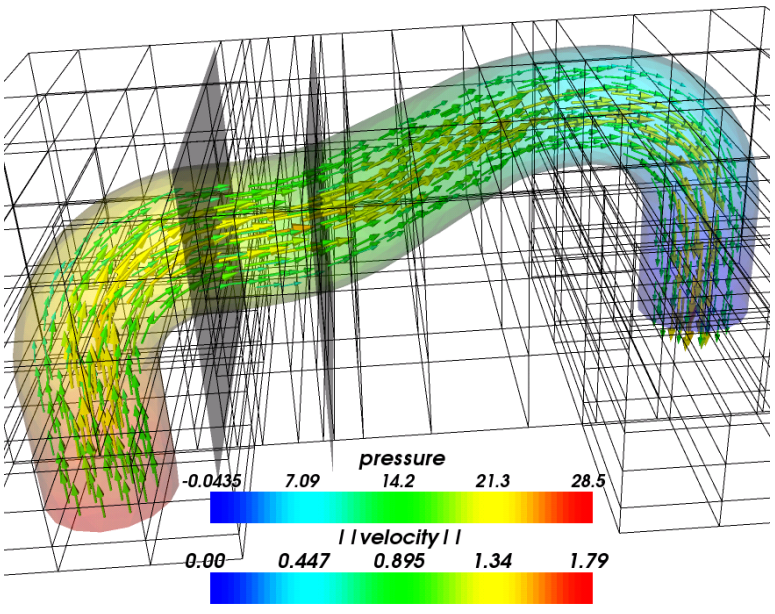
$$\Omega(t) = \Omega + \{t\mathcal{V}(x)\}_{x\in\Omega} \quad \text{where } \mathcal{V} = 0 \text{ in } \bar{\Omega}_c \cup \partial\Omega \setminus \Gamma$$

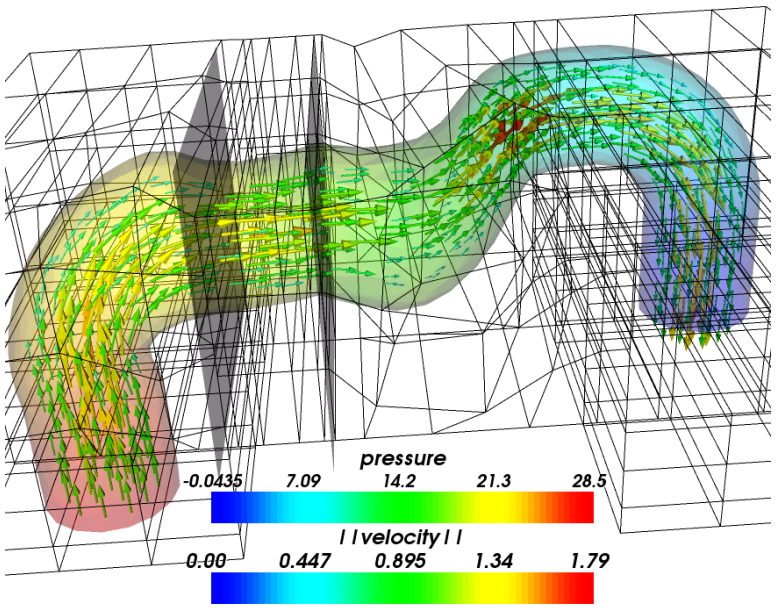


Optimal Flow Problem

Example Results

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





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

Direct Problem

. . . paper $\leftrightarrow$ input file

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

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

- 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""",
}
```

Direct Problem

. . . paper $\leftrightarrow$ input file

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

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

- 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:

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

- 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""",
}
```

Adjoint Problem

. . . paper $\leftrightarrow$ input file

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

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

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

- 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""",
}
```

Yes, the final slide!

What is done

- basic FE element engine
  - approximations up to P2 on simplexes (possibly with bubble)
  - Q1 tensor-product approximation on rectangles
- fields, variables, boundary conditions
- FE assembling
- equations, terms, regions
- materials, material caches
- various solvers accessed via abstract interface
- unit tests, automatic documentation generation

What is not done

- general FE engine, possibly with symbolic evaluation (SymPy)
- good documentation
- fast problem-specific solvers (!)
- adaptive mesh refinement (!)
- parallelization of both assembling and solving (PETSc?)

What will not be done (?)

- GUI
- real symbolic parsing/evaluation of equations

http://sfepy.org

Yes, the final slide!

What is done

- basic FE element engine
  - approximations up to P2 on simplexes (possibly with bubble)
  - Q1 tensor-product approximation on rectangles
- fields, variables, boundary conditions
- FE assembling
- equations, terms, regions
- materials, material caches
- various solvers accessed via abstract interface
- unit tests, automatic documentation generation

What is not done

- general FE engine, possibly with symbolic evaluation (SymPy)
- good documentation
- fast problem-specific solvers (!)
- adaptive mesh refinement (!)
- parallelization of both assembling and solving (PETSc?)

What will not be done (?)

- GUI
- real symbolic parsing/evaluation of equations

http://sfepy.org

Yes, the final slide!

What is done

- basic FE element engine
  - approximations up to P2 on simplexes (possibly with bubble)
  - Q1 tensor-product approximation on rectangles
- fields, variables, boundary conditions
- FE assembling
- equations, terms, regions
- materials, material caches
- various solvers accessed via abstract interface
- unit tests, automatic documentation generation

What is not done

- general FE engine, possibly with symbolic evaluation (SymPy)
- good documentation
- fast problem-specific solvers (!)
- adaptive mesh refinement (!)
- parallelization of both assembling and solving (PETSc?)

What will not be done (?)

- GUI
- real symbolic parsing/evaluation of equations

http://sfepy.org

Yes, the final slide!

What is done

- basic FE element engine
  - approximations up to P2 on simplexes (possibly with bubble)
  - Q1 tensor-product approximation on rectangles
- fields, variables, boundary conditions
- FE assembling
- equations, terms, regions
- materials, material caches
- various solvers accessed via abstract interface
- unit tests, automatic documentation generation

What is not done

- general FE engine, possibly with symbolic evaluation (SymPy)
- good documentation
- fast problem-specific solvers (!)
- adaptive mesh refinement (!)
- parallelization of both assembling and solving (PETSc?)

What will not be done (?)

- GUI
- real symbolic parsing/evaluation of equations

`http://sfepy.org`

This is not a slide!