Building Simulation Software for the Next Decade: Trends and Tools

Hans Petter Langtangen

Center for Biomedical Computing (CBC)
at Simula Research Laboratory

Dept. of Informatics, University of Oslo

September 2, 2013

Center for Biomedical Computing

Problem: scientists are scientists

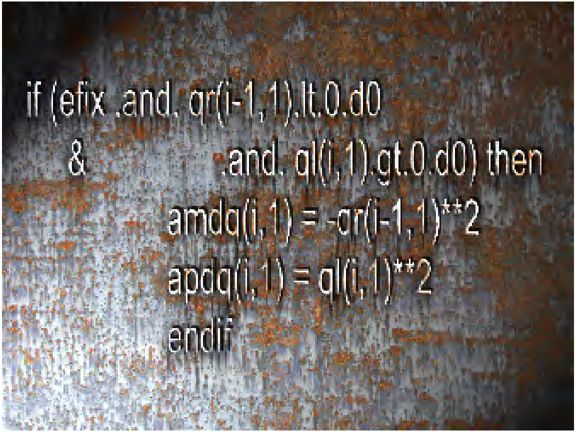
Greg Wilson, Univ. of Toronto, 2010:

Unfortunately, most scientists are never taught how to use computers effectively. After a generic first-year programming course, and possibly a numerical methods or statistics course later on, graduate students and working scientists are expected to figure out for themselves how to build, validate, maintain, and share complex programs. This is about as fair as teaching someone arithmetic and then expecting them to figure out calculus on their own, and about as likely to succeed.

Outline

- Programming technologies
- FEniCS: software for creating mechanistic models
- Ways of dealing with complexity
- Reproducible computational science

Most scientific software is written in Fortran (77) – and faces aging problems



Classical technologies ("low level languages")

- Fortran 77:
  - only primitive data types
  - long argument lists in subroutines
  - easy to interface libraries
- C (and Fortran 90):
  - struct: grouping variables in a new variable
  - short argument lists
  - challenge: different libs use different structs
- C++:
  - class: struct with (local) functions
  - object-oriented programming: extremely popular
  - challenge: too many classes, different libs use different classes
- Java, C#: as C++, popular, but little impact in science

Classical technologies ("low level languages")

- Fortran 77:
  - only primitive data types
  - long argument lists in subroutines
  - easy to interface libraries
- C (and Fortran 90):
  - struct: grouping variables in a new variable
  - short argument lists
  - challenge: different libs use different structs
- C++:
  - class: struct with (local) functions
  - object-oriented programming: extremely popular
  - challenge: too many classes, different libs use different classes
- Java, C#: as C++, popular, but little impact in science

Classical technologies ("low level languages")

- Fortran 77:
  - only primitive data types
  - long argument lists in subroutines
  - easy to interface libraries
- C (and Fortran 90):
  - struct: grouping variables in a new variable
  - short argument lists
  - challenge: different libs use different structs
- C++:
  - class: struct with (local) functions
  - object-oriented programming: extremely popular
  - challenge: too many classes, different libs use different classes
- Java, C#: as C++, popular, but little impact in science

Classical technologies (”low level languages”)

- Fortran 77:
  - only primitive data types
  - long argument lists in subroutines
  - easy to interface libraries
- C (and Fortran 90):
  - struct: grouping variables in a new variable
  - short argument lists
  - challenge: different libs use different structs
- C++:
  - class: struct with (local) functions
  - object-oriented programming: extremely popular
  - challenge: too many classes, different libs use different classes
- Java, C#: as C++, popular, but little impact in science

Classical technologies ("low level languages")

- Fortran 77:
  - only primitive data types
  - long argument lists in subroutines
  - easy to interface libraries
- C (and Fortran 90):
  - struct: grouping variables in a new variable
  - short argument lists
  - challenge: different libs use different structs
- C++:
  - class: struct with (local) functions
  - object-oriented programming: extremely popular
  - challenge: too many classes, different libs use different classes
- Java, C#: as C++, popular, but little impact in science

Last decade: great and growing popularity of "Matlab-style" environments

- Matlab, Maple, Mathematica, R, IDL, Python, Scilab, ... make scientists more productive
- Why? Convenience, *no declaration of variables*, rich libraries, built-in visualization, much less and nicer code
- Downside: not as fast as Fortran, C, C++

(No declaration of variables solves the problem that F90, F2003, C++, Java, C# apply advanced constructs (object-oriented/generic programming) to solve...)

Last decade: great and growing popularity of "Matlab-style" environments

- Matlab, Maple, Mathematica, R, IDL, Python, Scilab, ... make scientists more productive
- Why? Convenience, *no declaration of variables*, rich libraries, built-in visualization, much less and nicer code
- Downside: not as fast as Fortran, C, C++

(No declaration of variables solves the problem that F90, F2003, C++, Java, C# apply advanced constructs (object-oriented/generic programming) to solve...)

Last decade: great and growing popularity of "Matlab-style" environments

- Matlab, Maple, Mathematica, R, IDL, Python, Scilab, ... make scientists more productive
- Why? Convenience, *no declaration of variables*, rich libraries, built-in visualization, much less and nicer code
- Downside: not as fast as Fortran, C, C++

(No declaration of variables solves the problem that F90, F2003, C++, Java, C# apply advanced constructs (object-oriented/generic programming) to solve...)

Combining the best of all worlds...

Wish:

One would like the convenience and high-level code of Matlab combined with the language power of Fortran and C++.

A possible answer: Python

- Supports all major programming styles
  - May look similar to Matlab
  - Has all the advanced flexibility of C++
- Supports large-scale codes
- Emphasizes array-based computing
- A glue of Fortran/C++/Matlab

Combining the best of all worlds...

Wish:

One would like the convenience and high-level code of Matlab combined with the language power of Fortran and C++.

A possible answer: Python

- Supports all major programming styles
  - May look similar to Matlab
  - Has all the advanced flexibility of C++
- Supports large-scale codes
- Emphasizes array-based computing
- A glue of Fortran/C++/Matlab

Combining the best of all worlds...

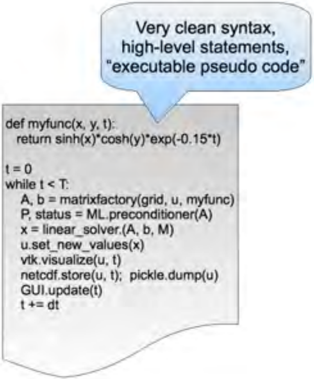
Wish:

One would like the convenience and high-level code of Matlab combined with the language power of Fortran and C++.

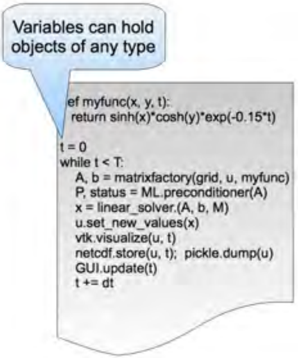
A possible answer: Python

- Supports all major programming styles
  - May look similar to Matlab
  - Has all the advanced flexibility of C++
- Supports large-scale codes
- Emphasizes array-based computing
- A glue of Fortran/C++/Matlab

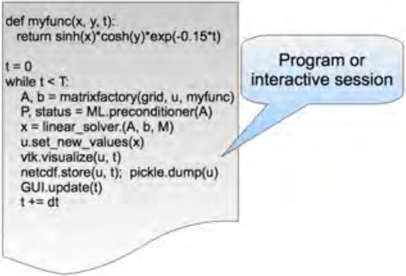
Python is a convenient programming environment



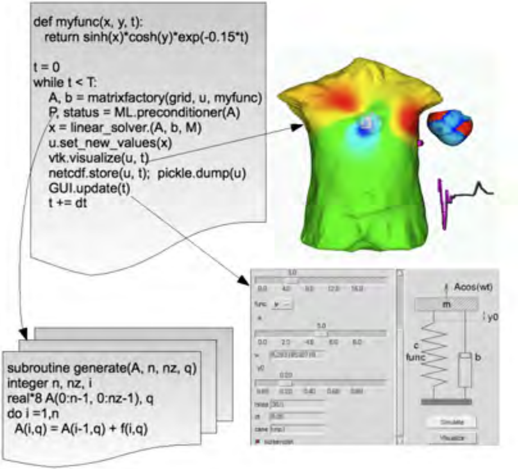
Python is a convenient programming environment



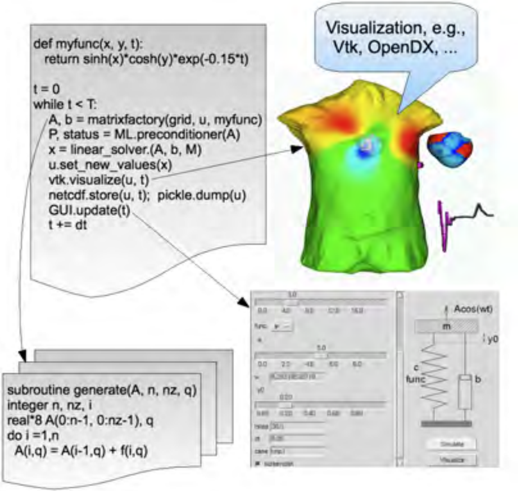
Python is a convenient programming environment



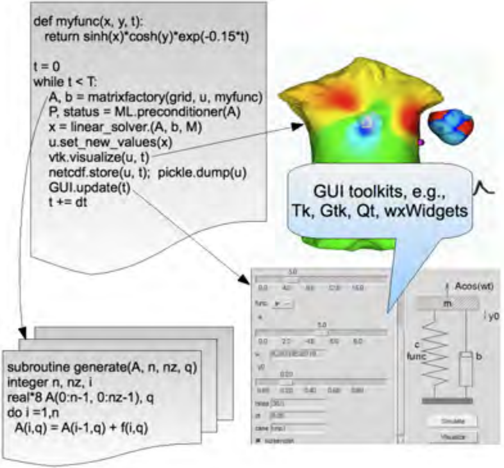
Python is a convenient programming environment



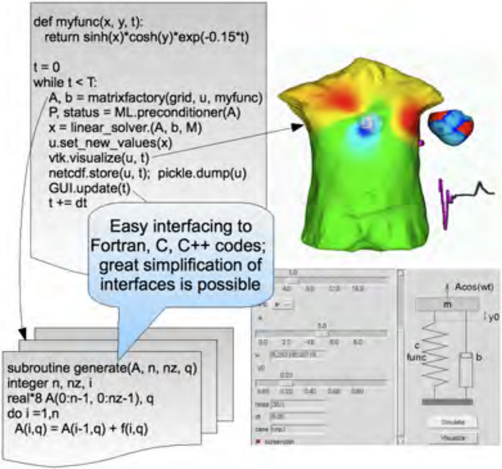
Python is a convenient programming environment



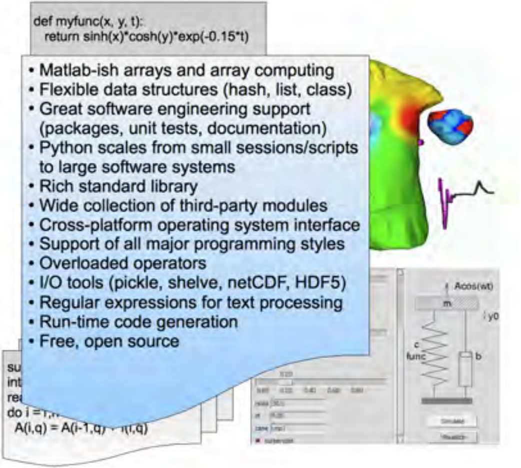
Python is a convenient programming environment



Python is a convenient programming environment



Python is a convenient programming environment



General or specialized programs?

- One specialized program for each equation/model?
- One general program for all equations/models?
- One general program for *generating* specialized programs (Yes! – the FEniCS idea)

General or specialized programs?

- One specialized program for each equation/model?
- One general program for all equations/models?
- One general program for *generating* specialized programs (Yes! – the FEniCS idea)

General or specialized programs?

- One specialized program for each equation/model?
- One general program for all equations/models?
- One general program for *generating* specialized programs
  (Yes! – the FEniCS idea)

General or specialized programs?

- One specialized program for each equation/model?
- One general program for all equations/models?
- One general program for *generating* specialized programs
  (Yes! – the FEniCS idea)

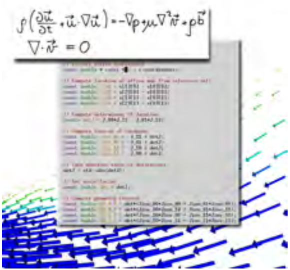
FEniCS solves PDEs by the finite element method

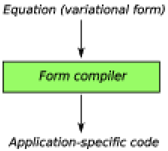
Input: finite element formulation of the PDE problem

$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x + \int_\Omega fv \, \mathrm{d}x$ in Python

Output: C++ code loaded back in Python

Python module with C++ def. of element matrix/vector, linked to finite element and linear algebra libraries





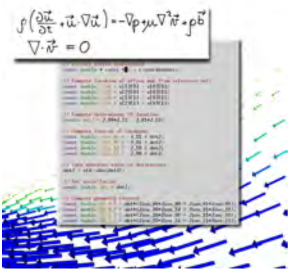
FEniCS solves PDEs by the finite element method

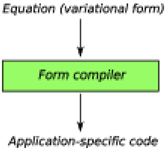
Input: finite element formulation of the PDE problem

$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x + \int_\Omega f v \, \mathrm{d}x$ in Python

Output: C++ code loaded back in Python

Python module with C++ def. of element matrix/vector, linked to finite element and linear algebra libraries





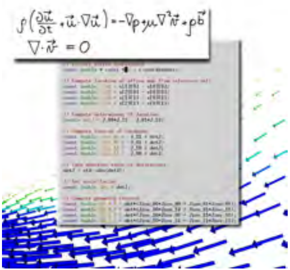
FEniCS solves PDEs by the finite element method

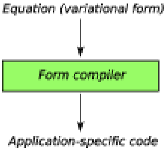
Input: finite element formulation of the PDE problem

$\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x + \int_\Omega f v \,\mathrm{d}x$ in Python

Output: C++ code loaded back in Python

Python module with C++ def. of element matrix/vector, linked to finite element and linear algebra libraries





FEniCS tries to combine four contradictory goals

Simplicity

$\int_\Omega a \nabla u \cdot \nabla v \,\mathrm{d}x \rightarrow$ `inner(a*grad(u), grad(v))*dx`

Generality

Linear $a(u,v) = L(v)$ or nonlinear $F(u;v) = 0$ variational problem

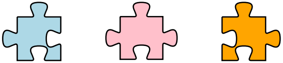
Efficiency

Generated C++ code tailored to the problem + efficient third-party libraries (PETSc, Trilinos, ...)

Reliability

Given a goal $\mathcal{M}(u)$ and tolerance ϵ, compute u such that

$$||\mathcal{M}(u_e) - \mathcal{M}(u)|| \leq \epsilon \quad (u_e\text{: exact sol.})$$



FEniCS tries to combine four contradictory goals

Simplicity

$\int_\Omega a \nabla u \cdot \nabla v \, \mathrm{d}x \rightarrow$ `inner(a*grad(u), grad(v))*dx`

Generality

Linear $a(u, v) = L(v)$ or nonlinear $F(u; v) = 0$ variational problem

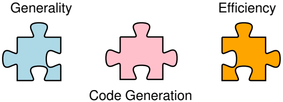
Efficiency

Generated C++ code tailored to the problem + efficient third-party libraries (PETSc, Trilinos, ...)

Reliability

Given a goal $\mathcal{M}(u)$ and tolerance ϵ, compute u such that

$$||\mathcal{M}(u_e) - \mathcal{M}(u)|| \leq \epsilon \quad (u_e\text{: exact sol.})$$



FEniCS tries to combine four contradictory goals

Simplicity

$\int_\Omega a \nabla u \cdot \nabla v \, \mathrm{d}x \rightarrow$ `inner(a*grad(u), grad(v))*dx`

Generality

Linear $a(u, v) = L(v)$ or nonlinear $F(u; v) = 0$ variational problem

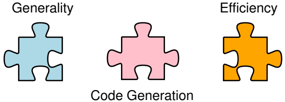
Efficiency

Generated C++ code tailored to the problem + efficient third-party libraries (PETSc, Trilinos, ...)

Reliability

Given a goal $\mathcal{M}(u)$ and tolerance ϵ, compute u such that

$$||\mathcal{M}(u_e) - \mathcal{M}(u)|| \leq \epsilon \quad (u_e\text{: exact sol.})$$



FEniCS tries to combine four contradictory goals

Simplicity

$\int_\Omega a \nabla u \cdot \nabla v \,\mathrm{d}x \rightarrow$ `inner(a*grad(u), grad(v))*dx`

Generality

Linear $a(u, v) = L(v)$ or nonlinear $F(u; v) = 0$ variational problem

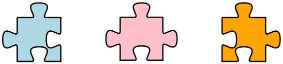
Efficiency

Generated C++ code tailored to the problem + efficient third-party libraries (PETSc, Trilinos, ...)

Reliability

Given a goal $\mathcal{M}(u)$ and tolerance ϵ, compute u such that

$$||\mathcal{M}(u_\mathrm{e}) - \mathcal{M}(u)|| \leq \epsilon \quad (u_\mathrm{e}\text{: exact sol.})$$



FEniCS tries to combine four contradictory goals

Simplicity

$\int_\Omega a \nabla u \cdot \nabla v \, \mathrm{dx} \rightarrow$ `inner(a*grad(u), grad(v))*dx`

Generality

Linear $a(u, v) = L(v)$ or nonlinear $F(u; v) = 0$ variational problem

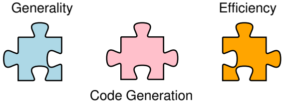
Efficiency

Generated C++ code tailored to the problem + efficient third-party libraries (PETSc, Trilinos, ...)

Reliability

Given a goal $\mathcal{M}(u)$ and tolerance ϵ, compute u such that

$$||\mathcal{M}(u_\mathrm{e}) - \mathcal{M}(u)|| \leq \epsilon \quad (u_\mathrm{e}\text{: exact sol.})$$



"Hello, world!" for PDEs: $-\nabla \cdot (k\nabla u) = f$

$$
\begin{aligned}
-\nabla \cdot (k\nabla u) &= f \text{ in } \Omega \\
u &= g \text{ on } \partial\Omega_D \\
-k\frac{\partial u}{\partial n} &= \alpha(u - u_0) \text{ on } \partial\Omega_R
\end{aligned}
$$

Variational problem: find $u \in V$ such that

$$F = \int_\Omega k\nabla u \cdot \nabla v dx - \int_\Omega f v dx + \int_{\partial\Omega_R} \alpha(u - u_0) v ds = 0 \quad \forall v \in V$$

Implementation:

```
F = inner(k*grad(u), grad(v))*dx - f*v*dx + alpha*(u-u0)*v*ds
```

"Hello, world!" for PDEs: $-\nabla \cdot (k \nabla u) = f$

$$-\nabla \cdot (k \nabla u) = f \text{ in } \Omega$$
$$u = g \text{ on } \partial\Omega_D$$
$$-k \frac{\partial u}{\partial n} = \alpha(u - u_0) \text{ on } \partial\Omega_R$$

Variational problem: find $u \in V$ such that

$$F = \int_\Omega k \nabla u \cdot \nabla v dx - \int_\Omega f v dx + \int_{\partial\Omega_R} \alpha(u - u_0) v ds = 0 \quad \forall v \in V$$

Implementation:

```
F = inner(k*grad(u), grad(v))*dx - f*v*dx + alpha*(u-u0)*v*ds
```

"Hello, world!" for PDEs: $-\nabla \cdot (k\nabla u) = f$

$$-\nabla \cdot (k\nabla u) = f \text{ in } \Omega$$
$$u = g \text{ on } \partial\Omega_D$$
$$-k\frac{\partial u}{\partial n} = \alpha(u - u_0) \text{ on } \partial\Omega_R$$

Variational problem: find $u \in V$ such that

$$F = \int_\Omega k\nabla u \cdot \nabla v dx - \int_\Omega f v dx + \int_{\partial\Omega_R} \alpha(u - u_0) v ds = 0 \quad \forall\, v \in V$$

Implementation:

```
F = inner(k*grad(u), grad(v))*dx - f*v*dx + alpha*(u-u0)*v*ds
```

"Hello, world!" for PDEs: $-\nabla \cdot (k \nabla u) = f$

$$-\nabla \cdot (k \nabla u) = f \text{ in } \Omega$$
$$u = g \text{ on } \partial\Omega_D$$
$$-k \frac{\partial u}{\partial n} = \alpha(u - u_0) \text{ on } \partial\Omega_R$$

Variational problem: find $u \in V$ such that

$$F = \int_\Omega k \nabla u \cdot \nabla v dx - \int_\Omega f v dx + \int_{\partial\Omega_R} \alpha(u - u_0) v ds = 0 \quad \forall\, v \in V$$

Implementation:

```
F = inner(k*grad(u), grad(v))*dx - f*v*dx + alpha*(u-u0)*v*ds
```

The complete "Hello, world!" program

```
from dolfin import *
mesh = Mesh('mydomain.xml.gz')
V = FunctionSpace(mesh, 'Lagrange', degree=1)

dOmega_D = MeshFunction('uint', mesh, 'myboundary.xml.gz')
g = Constant(0.0)
bc = DirichletBC(V, g, 1, dOmega_D)

u = TrialFunction(V)
v = TestFunction(V)
f = Constant(2.0)
k = Expression('A*x[1]*sin(pi*q*x[0])', A=4.5, q=1)
alpha = 10; u0 = 2

F = inner(k*grad(u), grad(v))*dx - f*v*dx + alpha*(u-u0)*v*ds

a = lhs(F); L = rhs(F)
u = Function(V)          # finite element function to compute
solve(a == L, u, bc)
plot(u)
```

Example of an autogenerated element matrix routine

```
void tabulate_tensor(double* A, ...)
{
const double G0_0_0 = det*Jinv00*Jinv00 + det*Jinv00*Jinv00
+ det*Jinv00*Jinv00 + det*Jinv00*Jinv00
+ det*Jinv01*Jinv01 + det*Jinv02*Jinv02
+ det*Jinv01*Jinv01 + det*Jinv02*Jinv02;
const double G0_0_1 = det*Jinv00*Jinv10 + det*Jinv00*Jinv10
+ det*Jinv00*Jinv10 + det*Jinv00*Jinv10
+ det*Jinv01*Jinv11 + det*Jinv02*Jinv12
+ det*Jinv01*Jinv11 + det*Jinv02*Jinv12;
...
const double G8_2_1 = det*Jinv21*Jinv12 + det*Jinv21*Jinv12;
const double G8_2_2 = det*Jinv21*Jinv22 + det*Jinv21*Jinv22;

const real tmp0_13 = 4.166666666666660e-02*G0_0_0;
const real tmp0_38 = 4.166666666666662e-02*G0_2_1;
const real tmp0_1 = -tmp0_13 + -4.166666666666661e-02*G0_1_0
- 4.166666666666660e-02*G0_2_0;
const real tmp0_37 = 4.166666666666661e-02*G0_2_0;
const real tmp0_25 = 4.166666666666661e-02*G0_1_0;
const real tmp0_26 = 4.166666666666662e-02*G0_1_1;

...
const real tmp8_139 = 4.166666666666662e-02*G8_2_2;
const real tmp8_125 = 4.166666666666661e-02*G8_1_0;
const real tmp8_100 = -tmp8_101 + 4.166666666666661e-02*G8_0_1
+ 4.166666666666661e-02*G8_0_2 + 4.166666666666662e-02*G8_1_1
+ 4.166666666666662e-02*G8_1_2 + 4.166666666666662e-02*G8_2_1
+ 4.166666666666662e-02*G8_2_2;
const real tmp8_126 = 4.166666666666662e-02*G8_1_1;
const real tmp8_113 = 4.166666666666660e-02*G8_0_0;
const real tmp8_138 = 4.166666666666662e-02*G8_2_1;
A[0] = tmp0_0;
A[1] = tmp0_1;
A[2] = tmp0_2;
...
A[141] = tmp6_141;
A[142] = tmp6_142;
A[143] = tmp6_143;
}
```

Fluid flow "Hello, world!": Stokes' problem

Stokes' problem for slow viscous flow:

$$-\nabla^2 u + \nabla p = f$$
$$\nabla \cdot u = 0$$

Variational problem: find $(u, p) \in V \times Q$ such that

$$F = \int_\Omega (\nabla v \cdot \nabla u - \nabla \cdot v \, p + v \cdot f) \, \mathrm{d}x +$$
$$\int_\Omega q \nabla \cdot u \, \mathrm{d}x = 0 \quad \forall \, (v, q) \in V \times Q$$

Fluid flow "Hello, world!" code

```
V = VectorFunctionSpace(mesh, 'Lagrange', 2)
Q = FunctionSpace(mesh, 'Lagrange', 1)
W = V * Q    # Taylor-Hood mixed finite element

v, q = TestFunctions(W)
u, p = TrialFunctions(W)

f = Constant((0, 0))

F = (inner(grad(v), grad(u)) - div(v)*p + q*div(u))*dx + inner(v, f)*dx

a = lhs(F);  L = rhs(F)
up = Function(W)
solve(a == L, up, bc)      # solve variational problem

# or

A = assemble(a);  b = assemble(L)
solve(A, up.vector(), b)   # solve linear system

u, p = up.split()
```

Again, code ≈ math

Key mathematical formula:

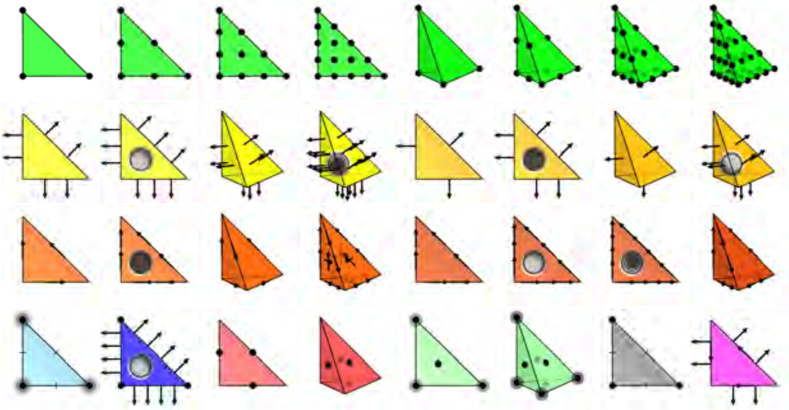
$$F = \int_\Omega (\nabla v \cdot \nabla u - \nabla \cdot v\, p + v \cdot f)\,\mathrm{d}x + \int_\Omega q \nabla \cdot u\,\mathrm{d}x$$

Key code line:

```
F = (inner(grad(v), grad(u)) - div(v)*p + inner(f,v)*dx + q*div(u))*dx
```

FEniCS supports a rich set of finite elements

- Lagrange$_q$ (P_q), DG$_q$, BDM$_q$, BDFM$_q$, RT$_q$, Nedelec 1st/2nd kind, Crouzeix–Raviart, Arnold-Winther, $\mathcal{P}_q\Lambda^k$, $\mathcal{P}_q^-\Lambda^k$, Morley, Hermite, Argyris, Bell, ...



Parallel computing

Distributed computing via MPI:

```
Terminal> mpirun -n 32 python myprog.py
```

Shared memory via OpenMP:

```
# In program
parameters['num_threads'] = Q
```

Automated error control

Input

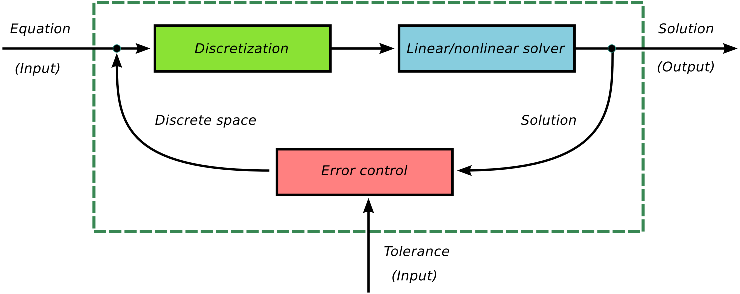
- $a(u, v) = L(v)$ or $F(u; v) = 0$
- Goal $\mathcal{M}(u)$
- $\epsilon > 0$

Output

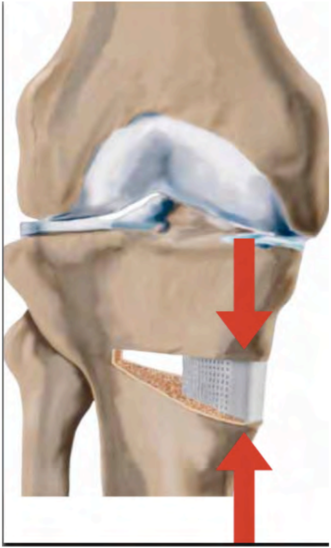
u such that

$$\|\mathcal{M}(u_{\mathrm{e}}) - \mathcal{M}(u)\| \leq \epsilon$$

(u_{e}: exact solution)

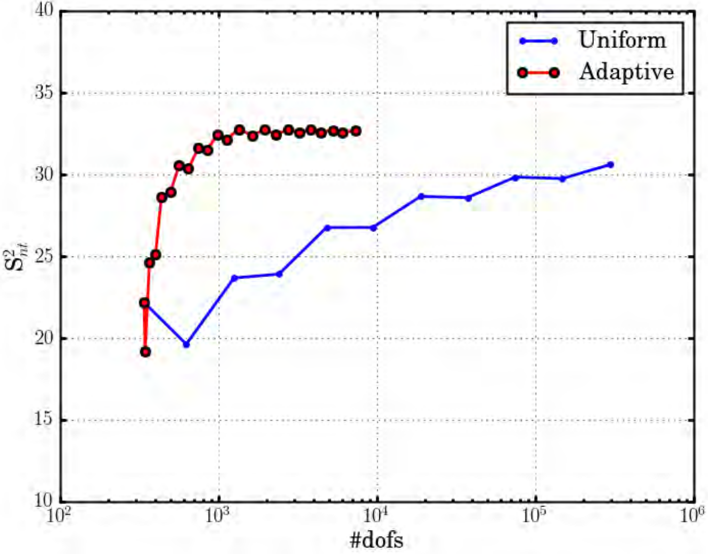


Example: compute shear stress in a bone implant



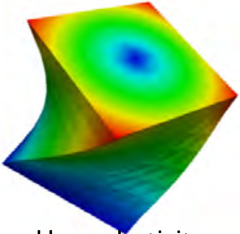
- Polymer-fluid mixture
- Nonlinear hyperelasticity
- Complicated constitutive law
- Novel mixed displacement-stress discretization via Arnold-Winther element

Adaptivity pays off – but would be really difficult to implement by hand in this case

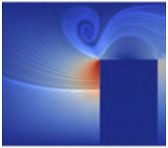


Some applications of FEniCS

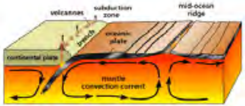
Fluid flow



Hyperelasticity



Fluid-structure



Mantle flow

Electrophysiology

$$\begin{pmatrix} A & B^\top \\ B & 0 \end{pmatrix}$$

Block prec.

How to deal with complexity in a software system

Most scientific packages are monolithic and do "everything" in a huge infrastructure of code

Recall the Unix philosophy (Doug McIlroy):

Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Applied to scientific programming:

Write modules that do one thing and do it well. Write modules to work together. Write modules to handle arrays, text streams + heterogeneous lists and hash tables, because that is a universal interface.

How to deal with complexity in a software system

Most scientific packages are monolithic and do "everything" in a huge infrastructure of code

Recall the Unix philosophy (Doug McIlroy):

Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Applied to scientific programming:

Write modules that do one thing and do it well. Write modules to work together. Write modules to handle arrays, text streams + heterogeneous lists and hash tables, because that is a universal interface.

How to deal with complexity in a software system

Most scientific packages are monolithic and do "everything" in a huge infrastructure of code

Recall the Unix philosophy (Doug McIlroy):

Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Applied to scientific programming:

Write modules that do one thing and do it well. Write modules to work together. Write modules to handle arrays, text streams + heterogeneous lists and hash tables, because that is a universal interface.

How to deal with complexity in a software system

Most scientific packages are monolithic and do "everything" in a huge infrastructure of code

Recall the Unix philosophy (Doug McIlroy):

Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Applied to scientific programming:

Write modules that do one thing and do it well. Write modules to work together. Write modules to handle arrays, text streams + heterogeneous lists and hash tables, because that is a universal interface.

Let code generation deal with complex architectures

- Cluster/multi-core/GPU programming is technically complicated and error-prone
- Let a program read the problem specification and *generate* complicated, low-level code
- Every time you encounter complex syntax: think of a program for generating the syntax
- Code generation makes expert knowledge available to many

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Key issues for making impact with software

- A common software platform accelerates research
- Distribute your software - it gives impact!
- Theory and user documentation gives impact too
- Make a simple/manageable build process
- Or provide a binary executable
  (maybe just for one platform)
- Get attention by regular releases
- Open source with (Git) version control system
- Comprehensive test suite (automatic)
- One-command build and test for developers
- Open design for integration in other systems

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

Is your computational science reproducible?

- Are PhD students' experiments reproducible?
- Can anyone check, rerun and extend your results in 10 years?

- Use a version control system (Git) for all software and paper writing work
- Use a site like GitHub, Bitbucket or Googlecode for collaboration
- Always automate simulation experiments (write scripts)
- Do all research work in virtual machines
- Archive the machines and link any paper to its machine
- Force researchers to document how their work is reproducible
- Goal: Make your computer work as serious as your theory, field and lab work

What have you learned?

- Make a strategy for how you design, develop, maintain and publish software
- Matlab or C++? Consider Python with C++ or Fortran
- Need to solve PDEs? Check out FEniCS at fenicsproject.org
- Parts of your software are better generated by a program
- Do all your work in a version control system
- Link papers to virtual machines