MU5MES01
Nonlinear structural mechanics by the finite element method.

Introduction to FEniCS

Corrado Maurini, based on slides from Anders Logg and André Massing

What is FEniCS?

FEniCS is an automated programming environment for differential equations

- C++/Python library
- Initiated 2003 in Chicago
- Licensed under the GNU LGPL

`http://fenicsproject.org/`

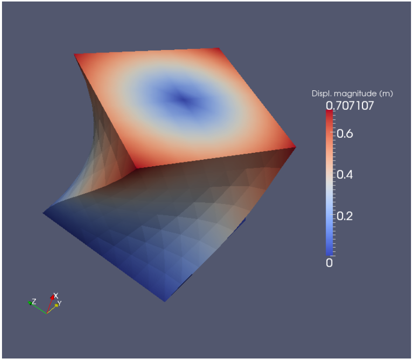
Collaborators

Simula Research Laboratory, University of Cambridge, University of Chicago, Texas Tech University, KTH Royal Institute of Technology, Chalmers University of Technology, Imperial College London, University of Oxford, Charles University in Prague, ...

FEniCS is automated FEM

- Automated generation of basis functions
- Automated evaluation of variational forms
- Automated finite element assembly
- Automated adaptive error control

Hyperelasticity




```python
from fenics import *

mesh = UnitCubeMesh(24, 16, 16)
V = VectorFunctionSpace(mesh, "Lagrange", 1)

left =  CompiledSubDomain("(std::abs(x[0])       < DOLFIN_EPS)
    && on_boundary")
right = CompiledSubDomain("(std::abs(x[0] - 1.0) < DOLFIN_EPS)
    && on_boundary")

c = Expression(("0.0", "0.0", "0.0"), degree=0)
r = Expression(("0.0",
"0.5*(y0+(x[1]-y0)*cos(t)-(x[2]-z0)*sin(t)-x[1])",
"0.5*(z0+(x[1]-y0)*sin(t)+(x[2]-z0)*cos(t)-x[2])"),
y0=0.5, z0=0.5, t=pi/3, degree=3)
bcl = DirichletBC(V, c, left)
bcr = DirichletBC(V, r, right)
bcs = [bcl, bcr]
v  = TestFunction(V)
u  = Function(V)
B  = Constant((0.0, -0.5, 0.0))
T  = Constant((0.1,  0.0, 0.0))
I = Identity(V.cell().d)
F = I + grad(u)
Ic = tr(F.T*F)
J  = det(F)
E, nu = 10.0, 0.3
mu, lmbda = Constant(E/(2*(1 + nu))), Constant(E*nu/((1 +
    nu)*(1 - 2*nu)))
psi = (mu/2)*(Ic - 3) - mu*ln(J) + (lmbda/2)*(ln(J))**2
Pi = psi*dx - dot(B, u)*dx - dot(T, u)*ds
F = derivative(Pi, u, v)

solve(F == 0, u, bcs)
plot(u, interactive=True, mode="displacement")
```

How to use FEniCS?

Hello World in FEniCS: problem formulation

Poisson's equation

$$
\begin{aligned}
-\Delta u &= f \quad \text{in } \Omega \\
u &= 0 \quad \text{on } \partial\Omega
\end{aligned}
$$

Finite element formulation

Find $u \in V$ such that

$$
\underbrace{\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x}_{a(u,v)} = \underbrace{\int_\Omega f\, v \,\mathrm{d}x}_{L(v)} \quad \forall\, v \in V
$$

Hello World in FEniCS: implementation

```
from fenics import *

mesh = UnitSquareMesh(32, 32)

V = FunctionSpace(mesh, "Lagrange", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("x[0]*x[1]", degree=2)

a = dot(grad(u), grad(v))*dx
L = f*v*dx

bc = DirichletBC(V, 0.0, DomainBoundary())

u = Function(V)
solve(a == L, u, bc)
plot(u)
```

Basic API

- `Mesh`, `Vertex`, `Edge`, `Face`, `Facet`, `Cell`
- `FiniteElement`, `FunctionSpace`
- `TrialFunction`, `TestFunction`, `Function`
- `grad()`, `curl()`, `div()`, ...
- `Matrix`, `Vector`, `KrylovSolver`, `LUSolver`
- `assemble()`, `solve()`, `plot()`

- Python interface generated semi-automatically by SWIG
- C++ and Python interfaces almost identical

Function evaluation

`Expression` and `Function` objects `f` can be evaluated at arbitrary points:

```
# 1D
x = 0.5
f(x)
# 2D
x = (0.5, 0.3)  # tuple or list
f(x)
# 3D
x = (0.5, 0.2, 1.0)  # tuple or list
f(x)
print(f(x))
```

Short-hand

```
   f( (0.5, 0.5) )  # Note double parenthesis
```

Exercise: Try it out! Use one of your existing codes and evaluate the solution at some point.

MU5MES01
Nonlinear structural mechanics by the finite element method.

Python programming

Corrado Maurini, based on slides from Anders Logg

What is Python?

The Python programming language is:

- General purpose
- Imperative
- Object-oriented
- High-level
- Slow
- Easy

Computing $1 + 2 + \ldots + 100$ in Python

```
s = 0

for i in range(1, 101):
    s += i

print s
```

Running the program

Bash code

```
$ python sum.py
5050
```

Performance in Python vs C++

Let's compute $\sum_{k=1}^{N} k$ for $N = 100,000,000$.

Bash code

```
$ time python sum.py
5000000050000000

real   0m13.243s
```

Bash code

```
$ time ./sum
5000000050000000

real   0m0.287s
```

Python/FEniCS programming 101

1. Open a file with your favorite text editor (Emacs :-)) and name the file something like `test.py`

2. Write the following in the file and save it:

```
from fenics import *
```

3. Run the file/program by typing the following in a terminal (with FEniCS setup):

```
$ python test.py
```

Python basics

Structure of a Python program

```python
import stuff

def some_function(argument):
    "Function documentation"
    return something

# This is a comment
if __name__ == "__main__":
    do_something
```

Declaring variables

```
a = 5
b = 3.5
c = "hej"
d = 'hej'
e = True
f = False
```

Illegal variable names

`and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try`

Comparison

```
x == y
x != y
x > y
x < y
x >= y
x <= y
```

Logical operators

```
not x
x and y
x or y
```

If

```python
if x > y:
    x += y
```

If / else

```
if x > y:
    x += y
    y += x
elif x < y:
    x += 1
else:
    y += 1
```

For loop

```
for variable in enumerable:
    stuff

for i in range(100):
    stuff

for i in range(100):
    stuff
    morestuff
```

While loop

```
while condition:
    stuff

i = 0
while i < 100:
    stuff
    i++

i = 0
while True:
    stuff
    if i == 99:
        break
```

Functions

```python
def myfunction(arg0, arg1, ...):
    stuff
    ...
    return something # or not, gives None

def sum(x, y):
    return x + y
```

Plotting

Matplotlib gives MATLAB-like plotting

```
from matplotlib import pyplot as plt

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('My figure')
plt.grid(True)
plt.savefig('myfigure.png')
plt.savefig('myfigure.pdf')
```

Python classes

Class structure

```python
class Foo:

    def __init__(self, argument):
        stuff

    def foo(self):
        stuff
        return something

    def bar(self):
        stuff
        return something

f = Foo(argument)
f.foo()
f.bar()
```

Class members

```python
class Foo:

    def __init__(self, argument):
        self.x = 3  # this is a member variable

    def foo(self):  # this is a member function
        stuff
```

Public and private class members

```python
class Foo:

    def __init__(self, argument):
        self.x = 3   # public member variable
        self.__x = 3 # private member variable
```

Python exercises

Exercises

- Write a program that generates the sequence $(x_n)_{n=0}^{100}$ for $x_n = n$.
- Write a program that generates the odd numbers between 1 and 100.
- Write a program that computes the sum $\sum_{n=0}^{100} x_n$ for $x_n = n$.
- Write a program that computes the sum of the odd numbers between 1 and 100.
- Write a program that generates all prime numbers between 2 and 1000.
- Write a program that generates the first 1000 prime numbers.
- Write a program that computes the approximation $\sqrt{2} \approx x_{100}$ for $x_n = (x_{n-1} + 2/x_{n-1})/2$ and $x_0 = 1$.
- Write a program that computes the approximation $\sqrt{2} \approx x_N$ for $x_n = (x_{n-1} + 2/x_{n-1})/2$ and $x_0 = 1$ where N is the smallest number such that $|x_N - x_{N-1}| < 10^{-10}$.
- Write a program that generates the sequence $(x_n)_{n=0}^{N}$ for $N = 10^6$ when

$$x_n = 4 \sum_{k=0}^{n} (-1)^k/(2k+1). \tag{1}$$

  Does it seem to converge to some particular number?