HPC Python Tutorial: Introduction to FEniCS 4/23/2012

Instructor:
Yaakoub El Khamra, Research Associate, TACC
yaakoub@tacc.utexas.edu

What is FEniCS

"The FEniCS Project is a collection of free software with an extensive list of features for automated, efficient solution of differential equations."

http://fenicsproject.org/

But what is it really

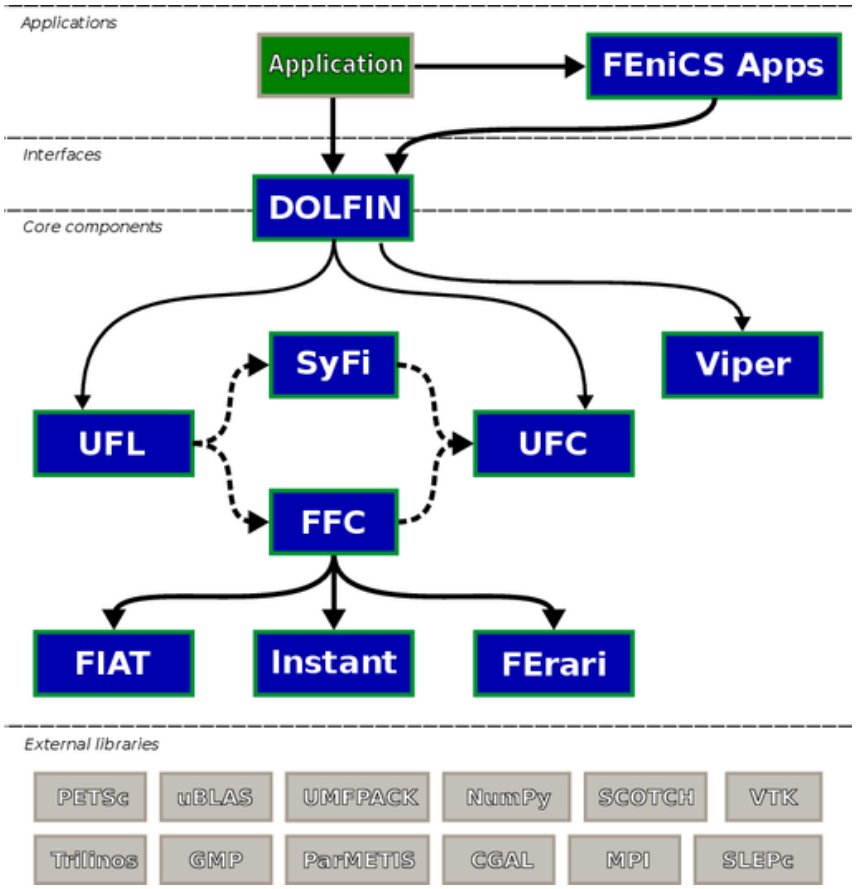
"The FEniCS Project is a collaborative project for the development of innovative concepts and tools for automated scientific computing, with a particular focus on automated solution of differential equations by finite element methods."

FEniCS Features

- Automated solution of variational problems
- Automated error control and adaptivity
- An extensive library of finite elements
- High performance linear algebra: PETSc, Trilinos/Epetra, uBlas, MTL4
- Computational Meshes: adaptive refinement, mesh partitioning (parmetis, scotch)
- Visualization and plotting
- Extensive documentation: It has its own book!
- Easy to install (it is installed on your laptops)

FEniCS Components

- DOLFIN: Problem solving environment
- FFC: FEniCS Form compiler
- FIAT: FInite element Automatic Tabulator
- UFC: Unified Form-assembl Code Code generation interface
- UFL: the Unified Form-assembly Code
- JIT compiler: instant



A bit more detail: DOLFIN

DOLFIN is a C++/Python library that functions as the main user interface of FEniCS. A large part of the functionality of FEniCS is implemented as part of DOLFIN. It provides a problem solving environment for models based on partial differential equations and implements core parts of the functionality of FEniCS, including data structures and algorithms for computational meshes and finite element assembly

A bit more detail: FIAT

FIAT (FInite element Automatic Tabulator) supports generation of arbitrary order instances of the Lagrange elements on lines, triangles, and tetrahedra. It is also capable of generating arbitrary order instances of Jacobi-type quadrature rules on the same element shapes. Further, H(div) and H(curl) conforming finite element spaces such as the families of Raviart-Thomas, Brezzi-Douglas-Marini and Nedelec are supported on triangles and tetrahedra. Upcoming versions will also support Hermite and nonconforming elements.

A bit more detail: UFC

UFC (Unified Form-assembly Code) is a unified framework for finite element assembly. More precisely, it defines a fixed interface for communicating low level routines (functions) for evaluating and assembling finite element variational forms. The UFC interface consists of a single header file ufc.h that specifies a C++ interface that must be implemented by code that complies with the UFC specification.

A bit more detail: FFC

One of the key features of FEniCS is automated code generation for the general and efficient solution of finite element variational problems. FFC (FEniCS Form Compiler) is a compiler for variational forms. It generates efficient low-level C++ code (UFC) from a high-level mathematical description (UFL) of a finite element variational problem.

A bit more detail: UFL

UFL (Unified Form Language) is a domain specific language for declaration of finite element discretizations of variational forms.

More precisely, it defines a flexible interface for choosing finite element spaces and defining expressions for weak forms in a notation close to mathematical notation

Let's work with examples

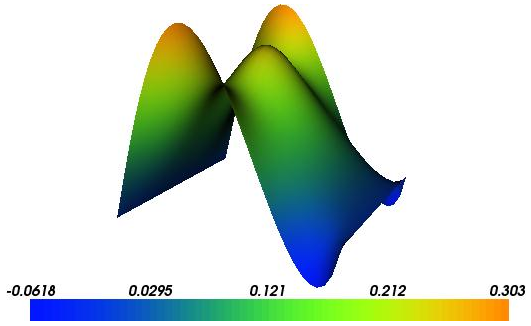
```
from dolfin import *

# Create mesh and define function space
mesh = UnitSquare(32, 32)
V = FunctionSpace(mesh, "Lagrange", 1)

# Define Dirichlet boundary (x = 0 or x = 1)
def boundary(x):
    return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EPS

# Define boundary condition
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundary)

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)")
g = Expression("sin(5*x[0])")
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds

# Compute solution
u = Function(V)
solve(a == L, u, bc)

# Save solution in VTK format
file = File("poisson.pvd")
file << u

# Plot solution
plot(u, interactive=True)
```

$$
\begin{aligned}
-\nabla^2 u &= f \quad \text{in } \Omega, \\
u &= 0 \quad \text{on } \Gamma_D, \\
\nabla u \cdot n &= g \quad \text{on } \Gamma_N.
\end{aligned}
$$

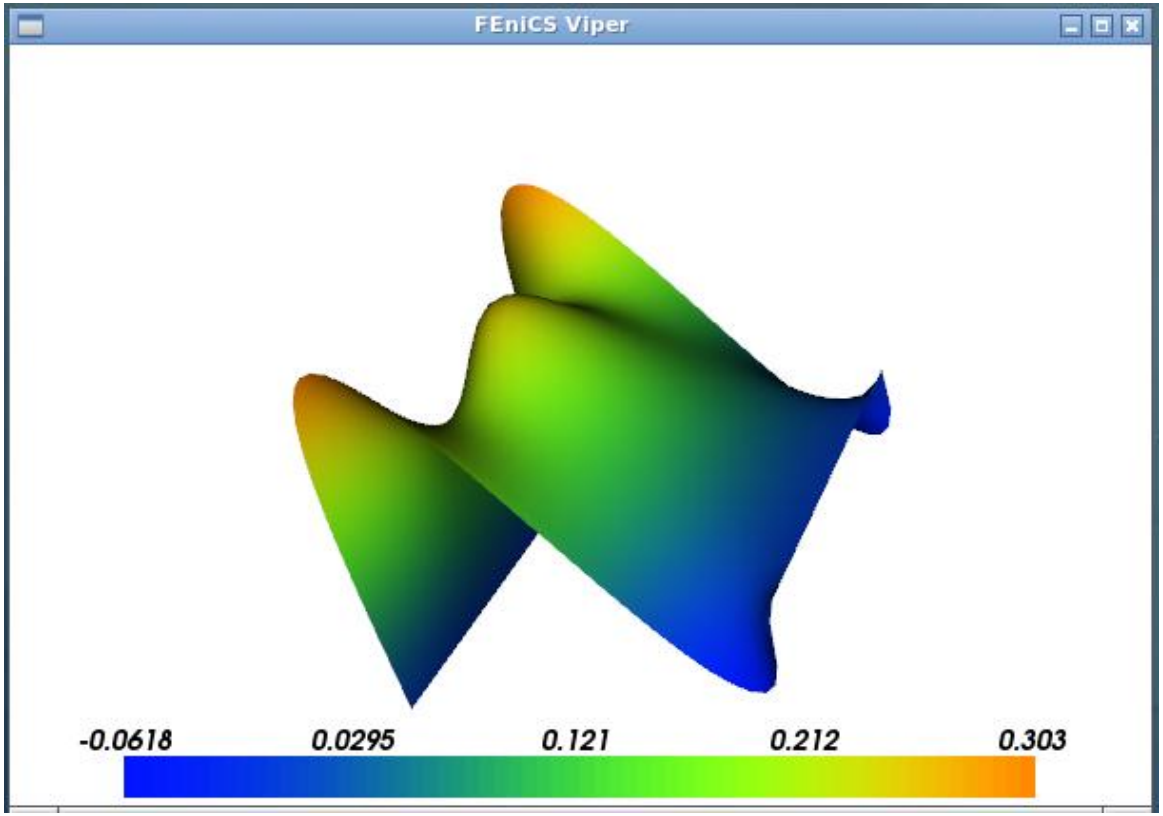
$$a(u, v) = L(v) \quad \forall\, v \in V,$$

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x,$$

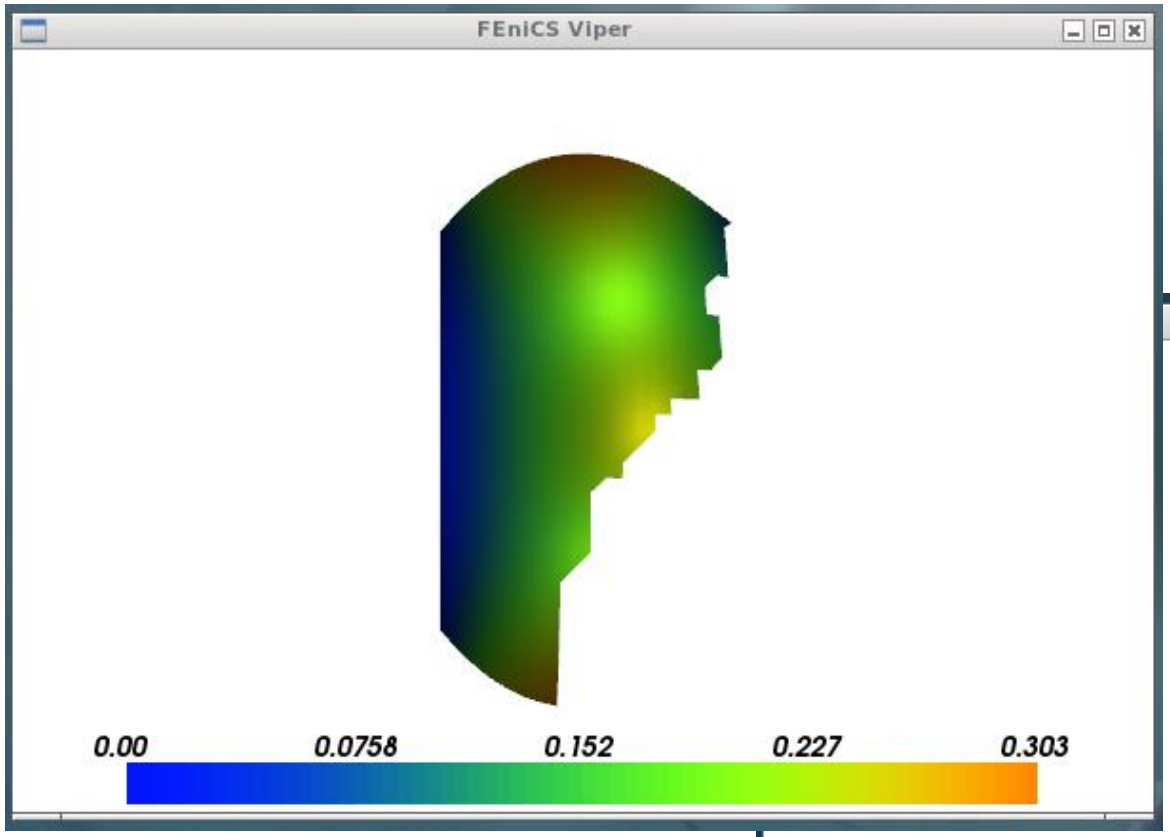
$$L(v) = \int_\Omega f v \, \mathrm{d}x + \int_{\Gamma_N} g v \, \mathrm{d}s.$$

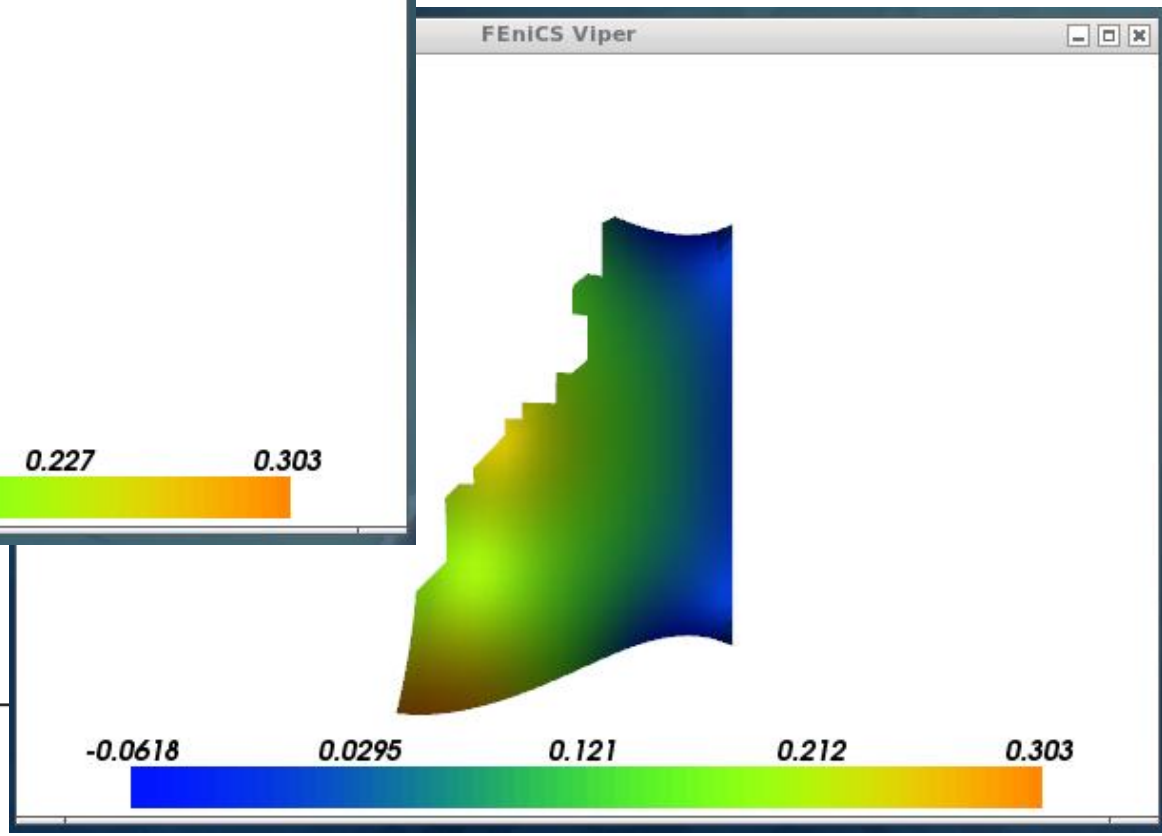


Running on 1 core



Running with 2 cores



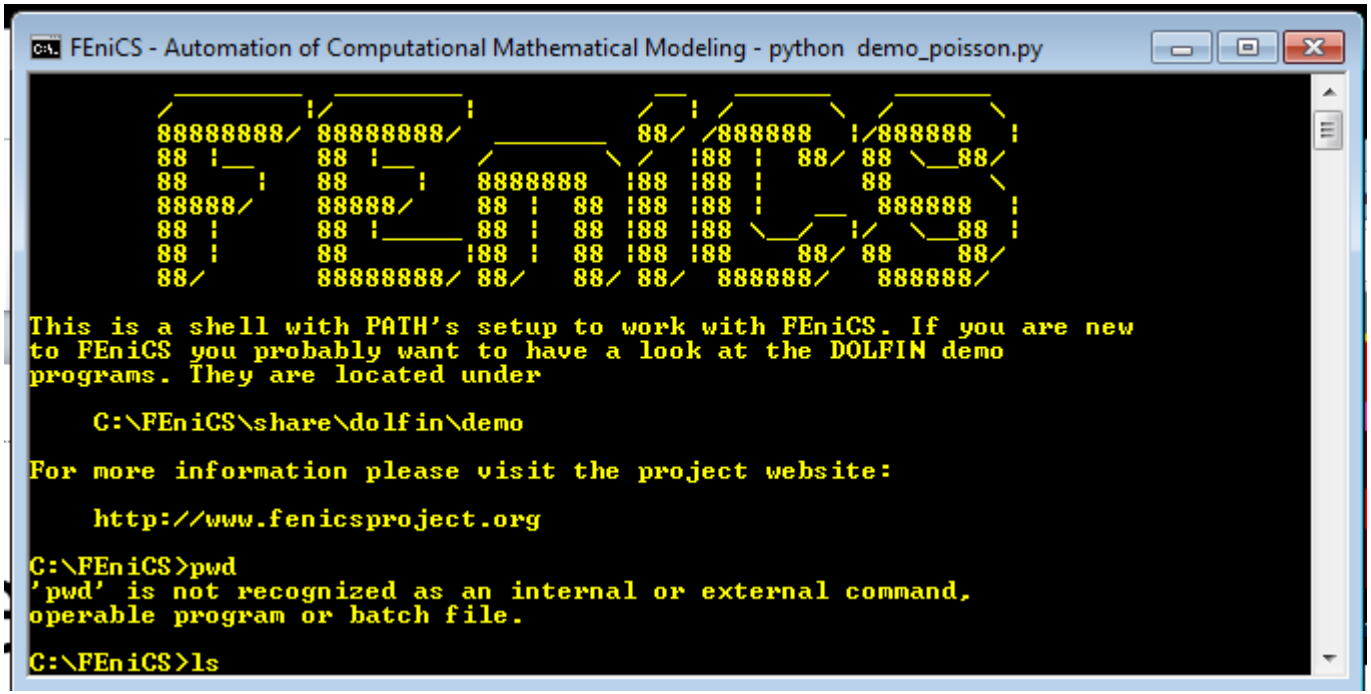


TACC

Try it yourself

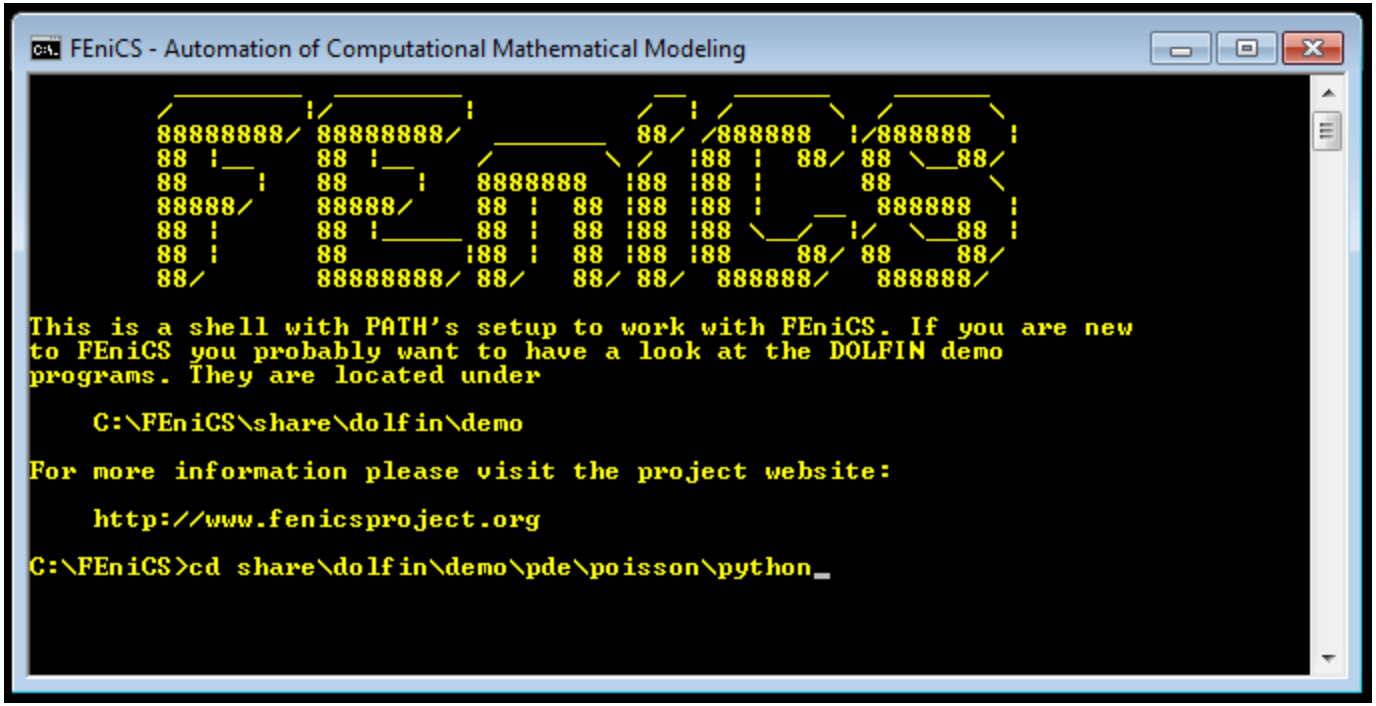
- FEniCS is installed on your laptops
- You can install it from this page: http://fenicsproject.org/download/
- On windows:
  - Use your file manager and navigate to C:\FEniCS
  - Double click on the executable "runme"

FEniCS in DOS



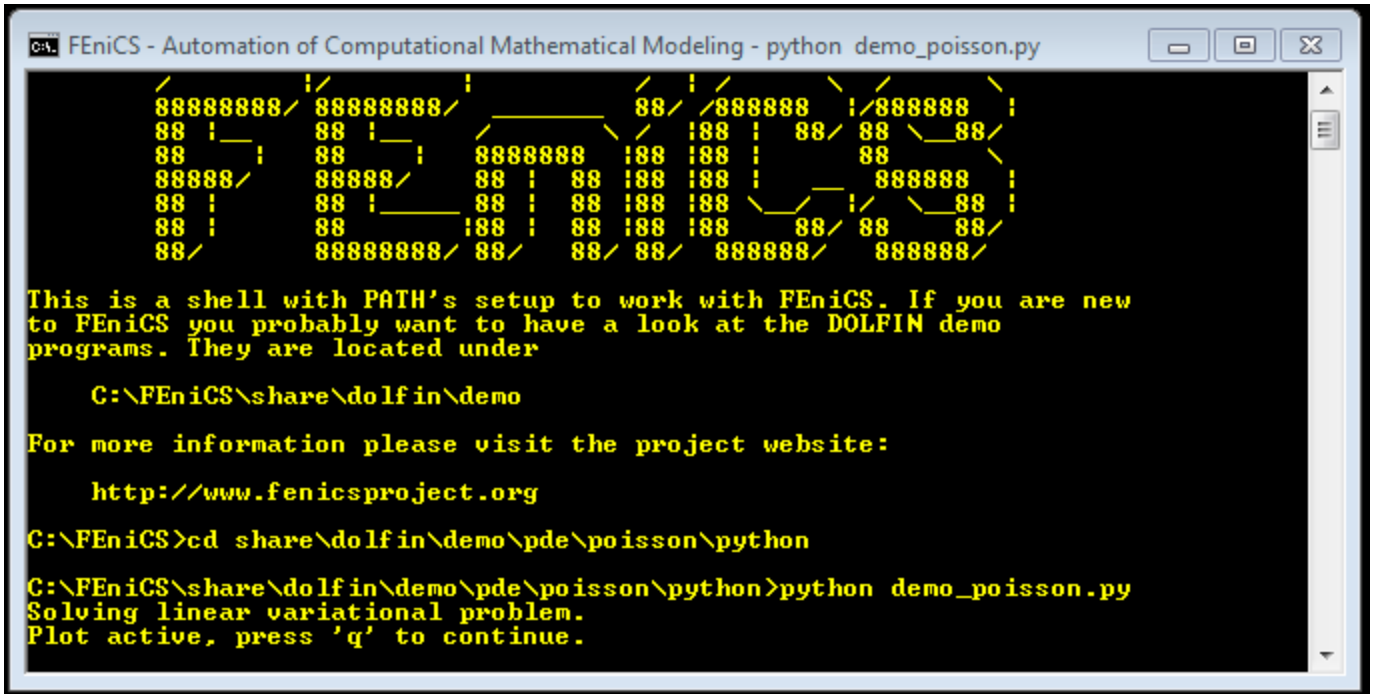
Try the Poisson Example

cd share\dolfin\demo\pde\poisson\python

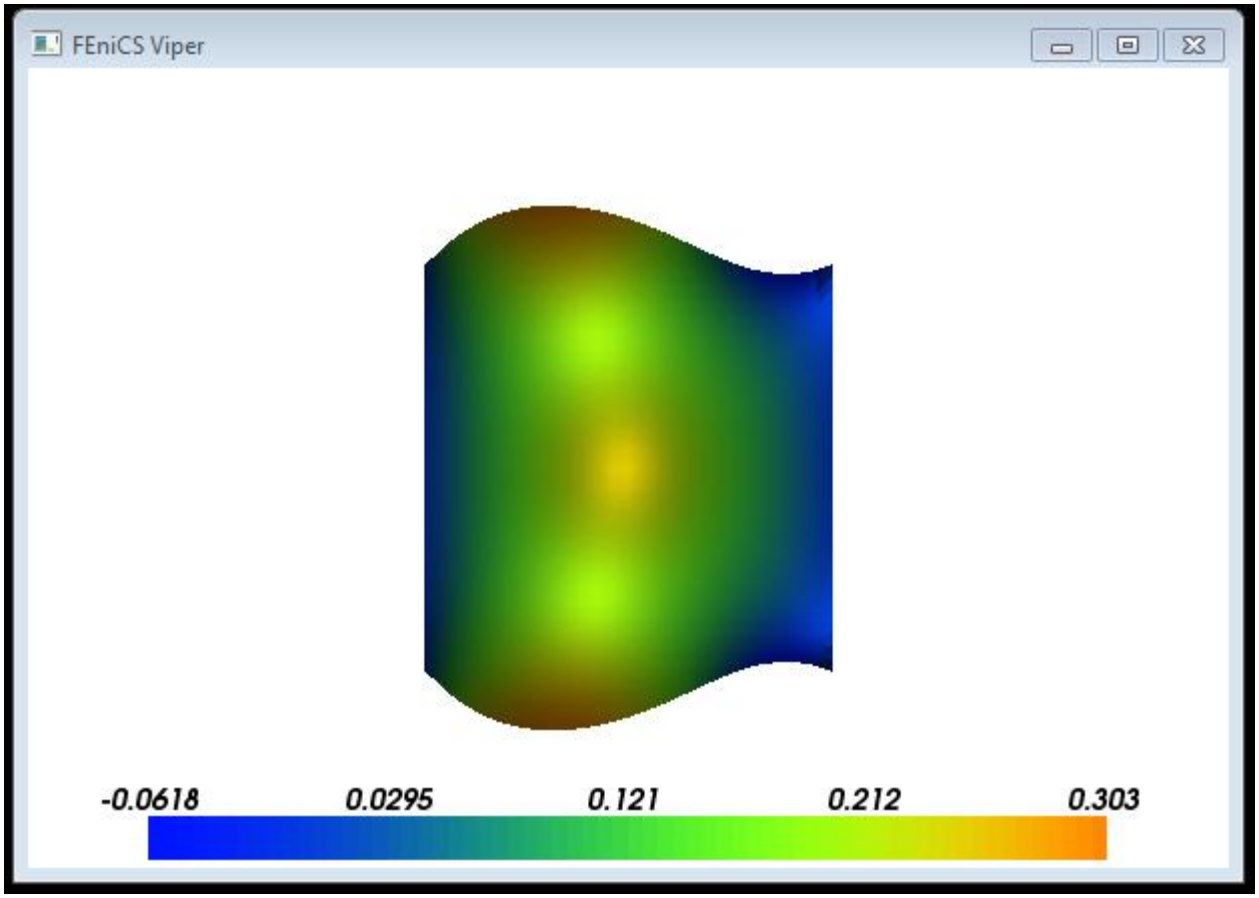


Run with python!

Python demo_poisson.py



3D Viz



What next?

- You can open up all the documented and undocumented examples

  C:\FEniCS\share\dolfin\demo\pde
- You can edit those files with your new-found understanding of HPC python
- You can use PETSc underneath FEniCS
- You can run the code in parallel under Linux
- You can change the code from 2D to 3D (1 line change)
- You can change the mesh resolution (1 line change)

Where to get more information

- FEniCS webpage: http://fenicsproject.org/
- FEniCS online tutorial: http://fenicsproject.org/documentation/tutorial/fundamentals.html
- FEniCS Demos:
- http://fenicsproject.org/documentation/dolfin/1.0.0/python/demo/index.html
- FEniCS Book: http://fenicsproject.org/book/
- Ask questions on launchpad: https://launchpad.net/fenics-project