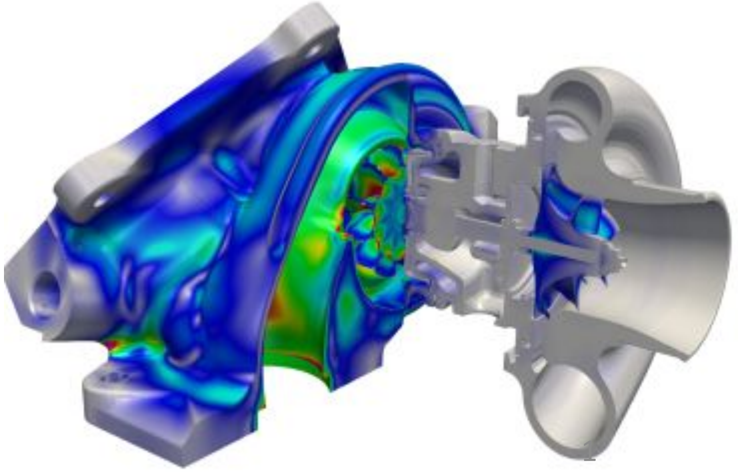
Using FeniCS in Python and C++

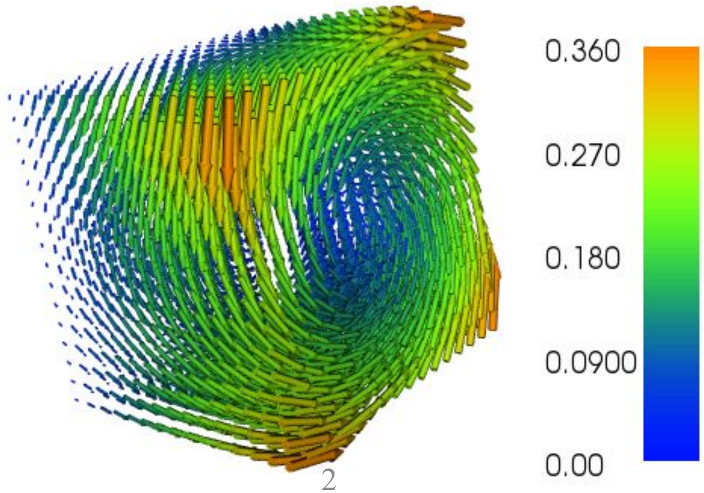
Speaker: Anthony Dowling
Department of Electrical and Computer Engineering
Clarkson University



Introduction

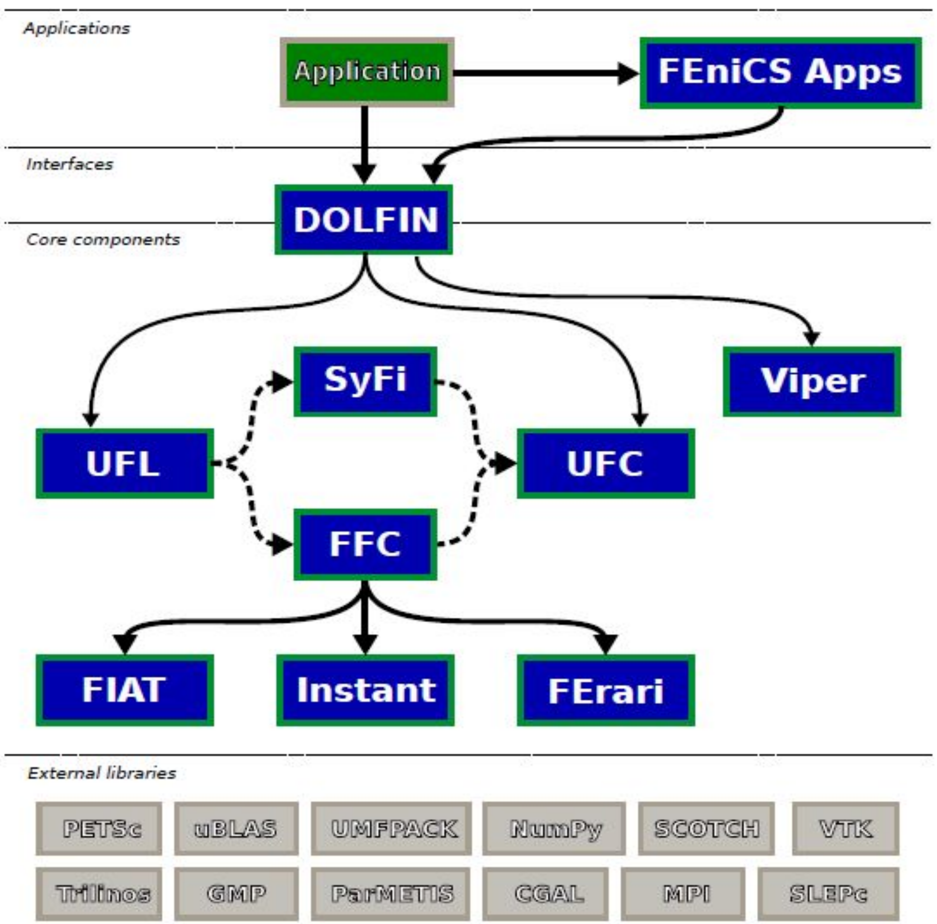
What is FEniCS?

FEniCS is a popular open-source computing platform for solving partial differential equations (PDEs). FEniCS enables users to quickly translate scientific models into efficient finite element code. With the high-level Python and C++ interfaces to FEniCS, it is easy to get started.



Introduction

Overview of components



UFL: unified Form Language for Finite element method.

FFC: FEniCS form compiler to generate low-level c++ code.

SyFi: Symbolic Finite element (C++ library) provide polynomial space and DOF as symbolic expression.

UFC: Unified Form-assembly Code.

FIAT: Finite element automatic Tabulator.

Instant: Instant is a Python module that allows for instant inlining of C and C++ code in Python.

FErari: Finite Element rearrangement to automatically reduce instructions) generates optimized code

Installation

Clarkson

Download

FEniCS on Docker

curl -s https://get.fenicsproject.org | bash

FEniCS on Ubuntu

sudo apt-get install software-properties-common
sudo add-apt-repository ppa:fenics-packages/fenics
sudo apt-get update
sudo apt-get install --no-install-recommends fenics

FEniCS on Anaconda

conda create -n fenicsproject -c conda-forge fenics
source activate fenicsproject

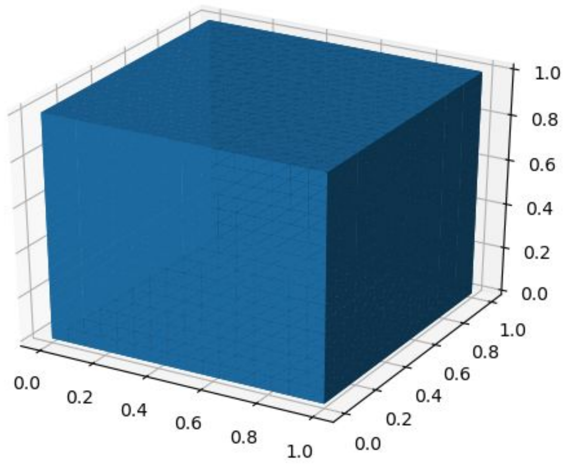
Installation

Test your installation

Simple code:

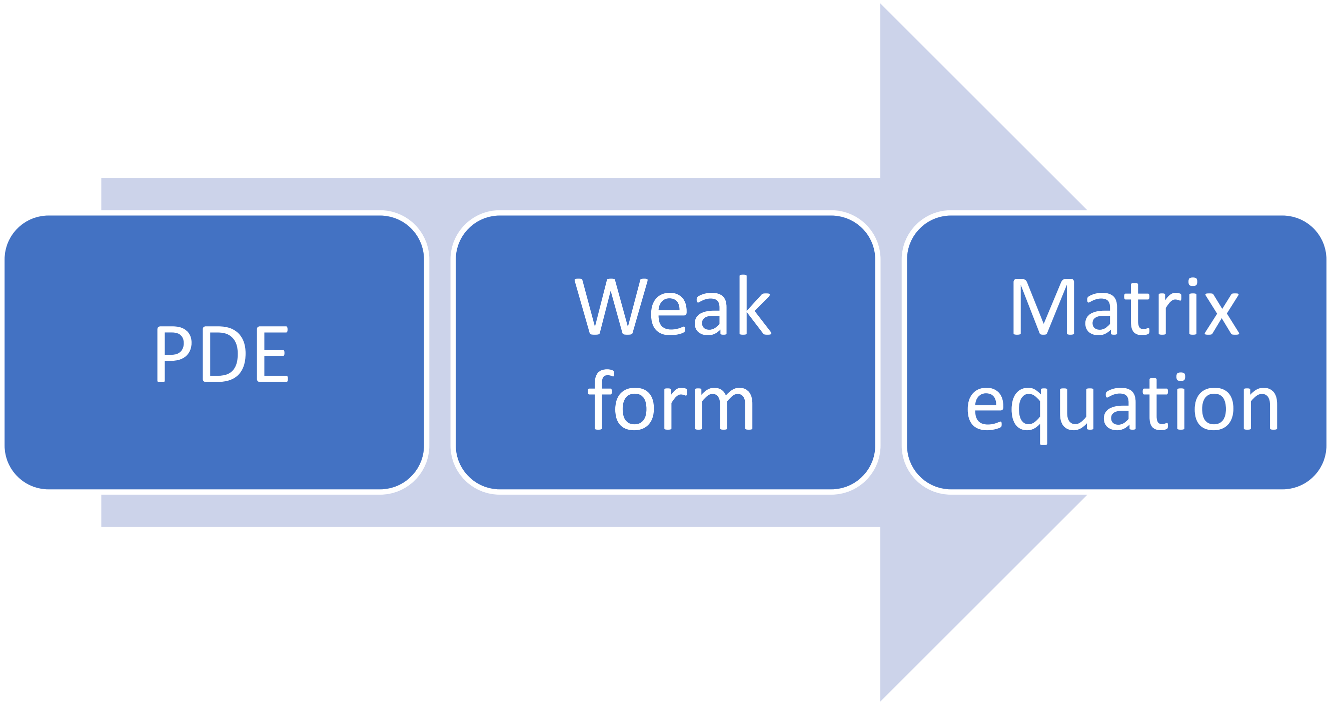
```
from fenics import *
import matplotlib.pyplot as plt
mesh = UnitCubeMesh (16 , 16 , 16)
plot ( mesh )
plt.show()
```

Output:



What is Finite element method?

Finite element method is a framework and a recipe for discretization of mathematical problem for example partial differential equation (PDE).



Clarkson

Mathematical equation

$$
\begin{aligned}
-\nabla^2 u(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad && \boldsymbol{x} \text{ in } \Omega, \\
u(\boldsymbol{x}) &= u_{\mathrm{D}}(\boldsymbol{x}), \quad && \boldsymbol{x} \text{ on } \partial\Omega.
\end{aligned}
$$

Finite element variational formulation

$$-\int_\Omega (\nabla^2 u) v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x.$$

$$-\int_\Omega (\nabla^2 u) v \, \mathrm{d}x = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x - \int_{\partial\Omega} \frac{\partial u}{\partial n} v \, \mathrm{d}s,$$

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x.$$

u is trial function

v is test function

Poisson equation

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

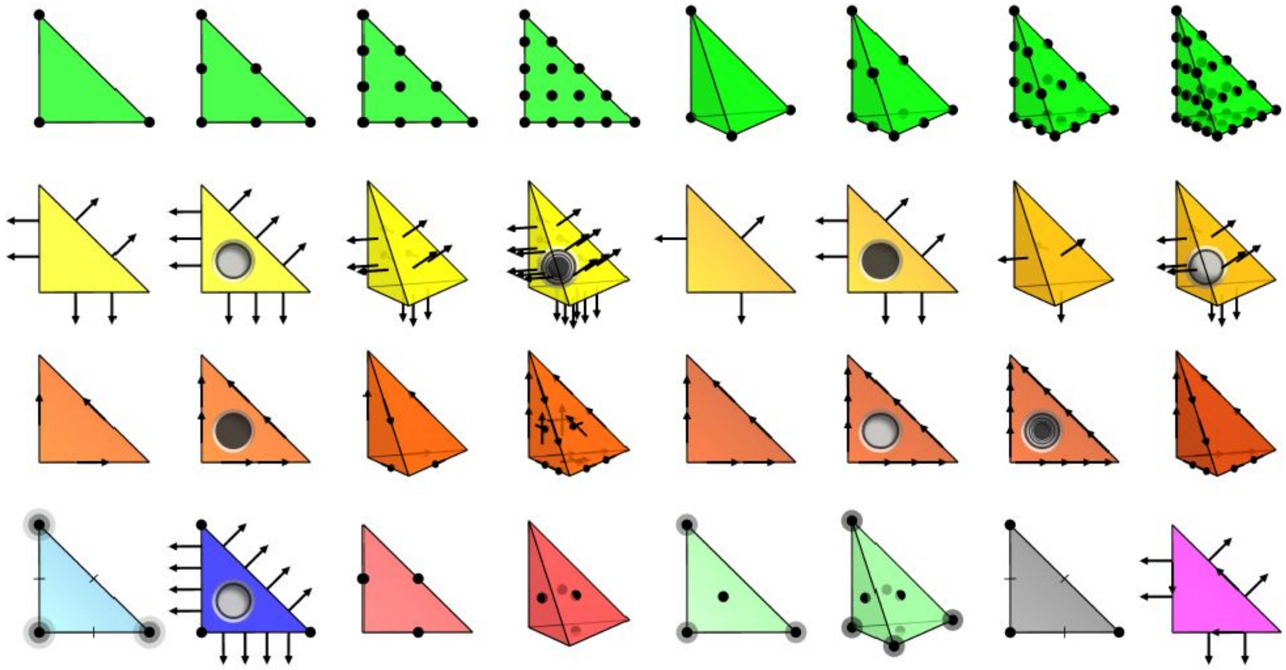
$$L(v) = \int_\Omega f v \, \mathrm{d}x.$$

$$a(u,v) = L(v)$$

Bilinear form

linear form

Families of elements



https://www.femtable.org

FEniCS code – C++ vs. Python

C++	Python
Static Typing	Duck Typing (Dynamic)
Compiled	Interpreted
Lots of excess code	Easier to write

FEniCS code – Setting up the Program

C++

```cpp
#include <dolfin.h>
#include <iostream>
using namespace dolfin;
#include "forms.h"

int main(int argc, char** argv) {
    dolfin::init(argc,argv);
```

Python

```python
#!/usr/bin/env python3

from fenics import *
```

FEniCS code – Defining Your Space

Python

```python
mesh = UnitSquareMesh(128,128)
V = FunctionSpace(mesh, 'P', 1)
```

UFL

```
element = FiniteElement("Lagrange",triangle,1)
```

C++

```cpp
auto mesh = std::make_shared<UnitSquareMesh>(128,128);
auto V = std::make_shared<forms::FunctionSpace>(mesh);
```

FEniCS code – Boundary Condition

Python

```python
u_D = Expression('1+(x[0]*x[0]) - 2*(x[1]*x[1])', degree=1)
```

C++

```cpp
class u_D_expr : public Expression {
    public:
        void eval(
            Array<double>& values,
            const Array<double>& x
        ) const {
            values[0] = 1+(x[0]*x[0]) + 2*(x[1]*x[1]);
        }
};
auto u_D = std::make_shared<u_D_expr>();
```

FEniCS code – Setting the Boundary

C++

```cpp
class Boundary : public SubDomain {
public:
    bool inside(
        const Array<double>& x,
        bool on_boundary
    ) const override {
        return on_boundary;
    }
};
auto boundary = std::make_shared<Boundary>();
DirichletBC bc(V, u_D, boundary);
```

Python

```python
def boundary(x, on_boundary):
    return on_boundary

bc = DirichletBC(V, u_D, boundary)
```

FEniCS code – Defining Your Space

UFL

```
f = Coefficient(element)
u = TrialFunction(element)
v = TestFunction(element)
a = dot(grad(u), grad(v))*dx
L = f*v*dx
```

Python

```
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(-6.0)
a = dot(grad(u), grad(v))*dx
L = f*v*dx
```

C++

```
auto f = std::make_shared<Constant>(-6.0);
forms::LinearForm L(V);
L.f = f;
forms::BilinearForm a(V,V);
```

FEniCS code – Solving

C++

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

Python

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

FEniCS code – Visualizing the Solution

C++

```cpp
std::string ofname = "solution.h5";
auto ofh = HDF5File(mesh->mpi_comm(), ofname, "w");
ofh.write(u,"solution");
ofh.close();
```

Python

```python
#!/usr/bin/env python3

from fenics import *
import sys
if len(sys.argv) != 2:
    print("NEED H5 FILE!")
    exit()

mesh = UnitSquareMesh(128,128)
V = FunctionSpace(mesh, 'triangle', 1)

u=Function(V)
HDF5File(mesh.mpi_comm(),sys.argv[1],'r').read(u,"solution")
```

```python
import matplotlib.pyplot as plt

P = plot(u)
plt.colorbar(P)
plt.show()
```

Slide

Poisson equation

FEniCS code – Running

Cmake Build System

```
cmake_minimum_required(VERSION 3.5)
project(poisson_example)
cmake_policy(SET CMP0004 NEW)

find_package(DOLFIN REQUIRED)
include(${DOLFIN_USE_FILE})

set(SRC_DIR src/)
set(SOURCES
    ${SRC_DIR}main.cpp
    ${SRC_DIR}forms.h
)

add_executable(${PROJECT_NAME} ${SOURCES})
target_link_libraries(${PROJECT_NAME} dolfin)
```

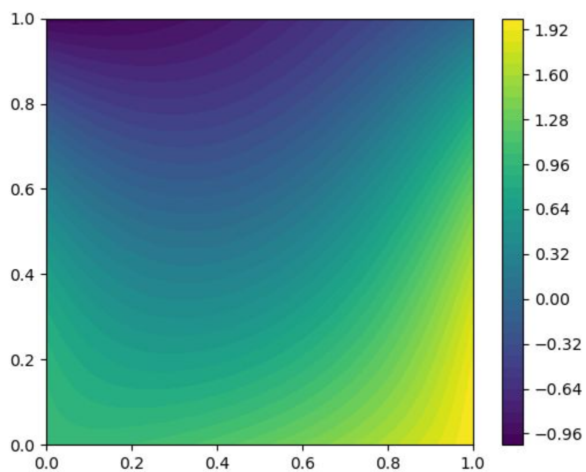
```
cd src
ffc -l dolfin forms.ufl
cd ..
mkdir build/
cd build
cmake ..
make
```

```
poison_cpp
  build
    CMakeFiles
    cmake_install.cmake
    CMakeCache.txt
    Makefile
    poisson_example
    solution.h5
  src
    forms.ufl
    main.cpp
  CMakeLists.txt
  plot_h5.py
  run.sh
```

FEniCS code – Visualizing the Solution

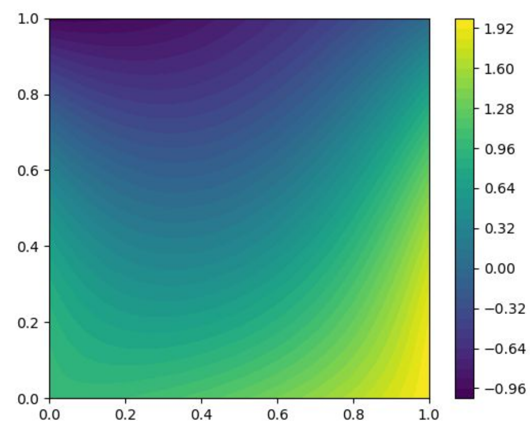
C++

```
$ ./poisson_example
No protocol specified
Solving linear variational problem.
$ ../plot_h5.py solution.h5
No protocol specified
$ 
```



Python

```
$ ./poisson_example.py
No protocol specified
Calling FFC just-in-time (JIT) compiler, this may take some ti
me.
Calling FFC just-in-time (JIT) compiler, this may take some ti
me.
Calling FFC just-in-time (JIT) compiler, this may take some ti
me.
Calling FFC just-in-time (JIT) compiler, this may take some ti
me.
Calling FFC just-in-time (JIT) compiler, this may take some ti
me.
Calling FFC just-in-time (JIT) compiler, this may take some ti
me.
Solving linear variational problem.
$ 
```



FEniCS code – C++ vs. Python Revisited

C++	Python
Static Typing	Duck Typing (Dynamic)
Compiled	Interpreted
Lots of excess code	Easier to write

Mesh: 512x512

C++ ~14% faster