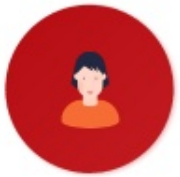
FEniCS Platform

Introduction



How it works

Getting started

Demos

Other Features

Who am I

Tamara Dancheva

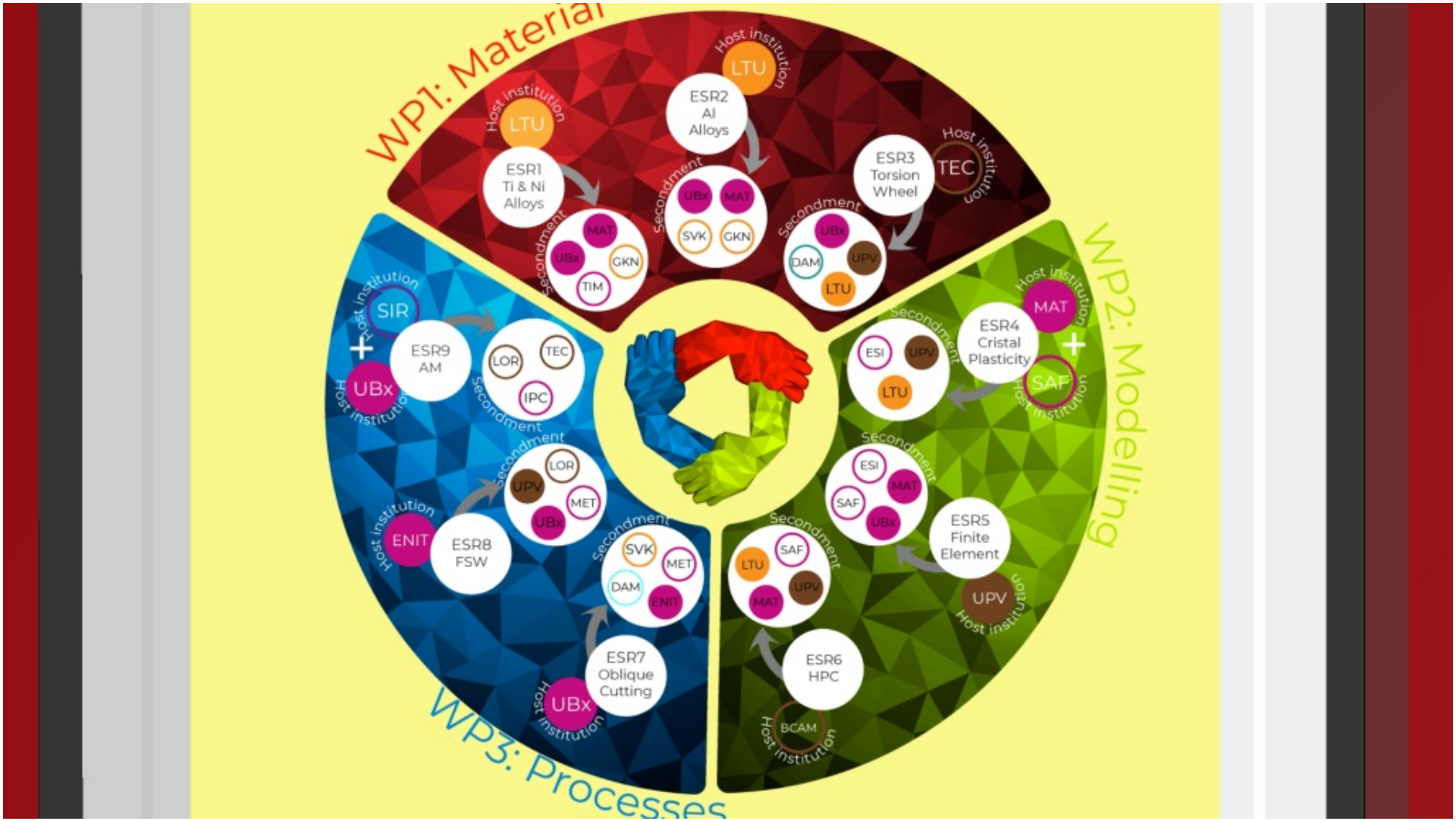
ESR6

Basque Center of Applied Mathematics

tdancheva@bcamath.org

This project received funding from the European Union's Marie Skłodowska-Curie Actions (MSCA) Innovative Training Networks (ITN) H2020-MSCA-ITN-2017 under the grant agreement N°764979.


https://www.enable-project.com
WP1: Material
WP2: Modelling
WP3: Processes


WP1: Material
WP2: Modelling
WP3: Processes
Host institution
LTU
ESR1
Ti & Ni
Alloys
ESR2
Al
Alloys
ESR3
Torsion
Wheel
TEC
Secondment
MAT
UBx
GKN
TIM
SVK
DAM
UPV
SIR
ESR9
AM
LOR
IPC
ENIT
ESR8
FSW
MET
ESR7
Oblique
Cutting
ESR4
Cristal
Plasticity
ESI
SAF
ESR5
Finite
Element
ESR6
HPC
BCAM


Oblique Cutting
UBx
Host institution
HPC
BCAM
Host institution
WP3: Processes
France
Spain
Sweden
Belgium
Denmark
University
Company /research
Safran Tech
ESI Group
Timet
Metallicadour
Innovation Plasturgie Com-
Danish Advanced Manufacturing Research
SIRRIS
GKN Aerospace Sweden
Sandvik Coromant
Tecnalia
Lortek
Basque Center for Applied Mathe-
University of Basque Country
Engineering School of Tarbes
Superior National School of Mines of
Lulea University of Technology
University of Bordeaux

FEniCS Platform

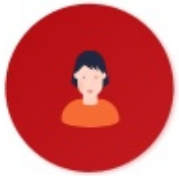
Introduction

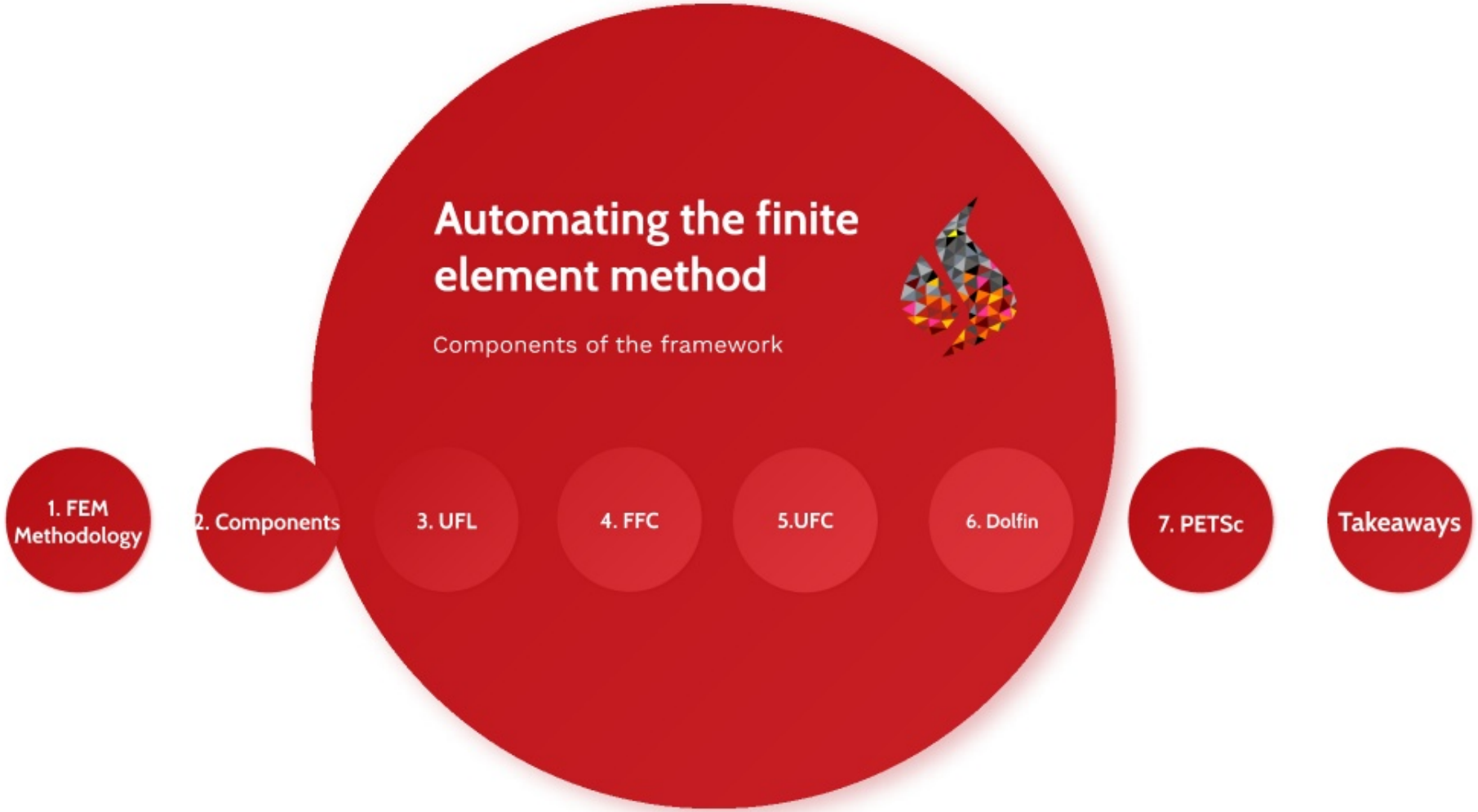
How it works

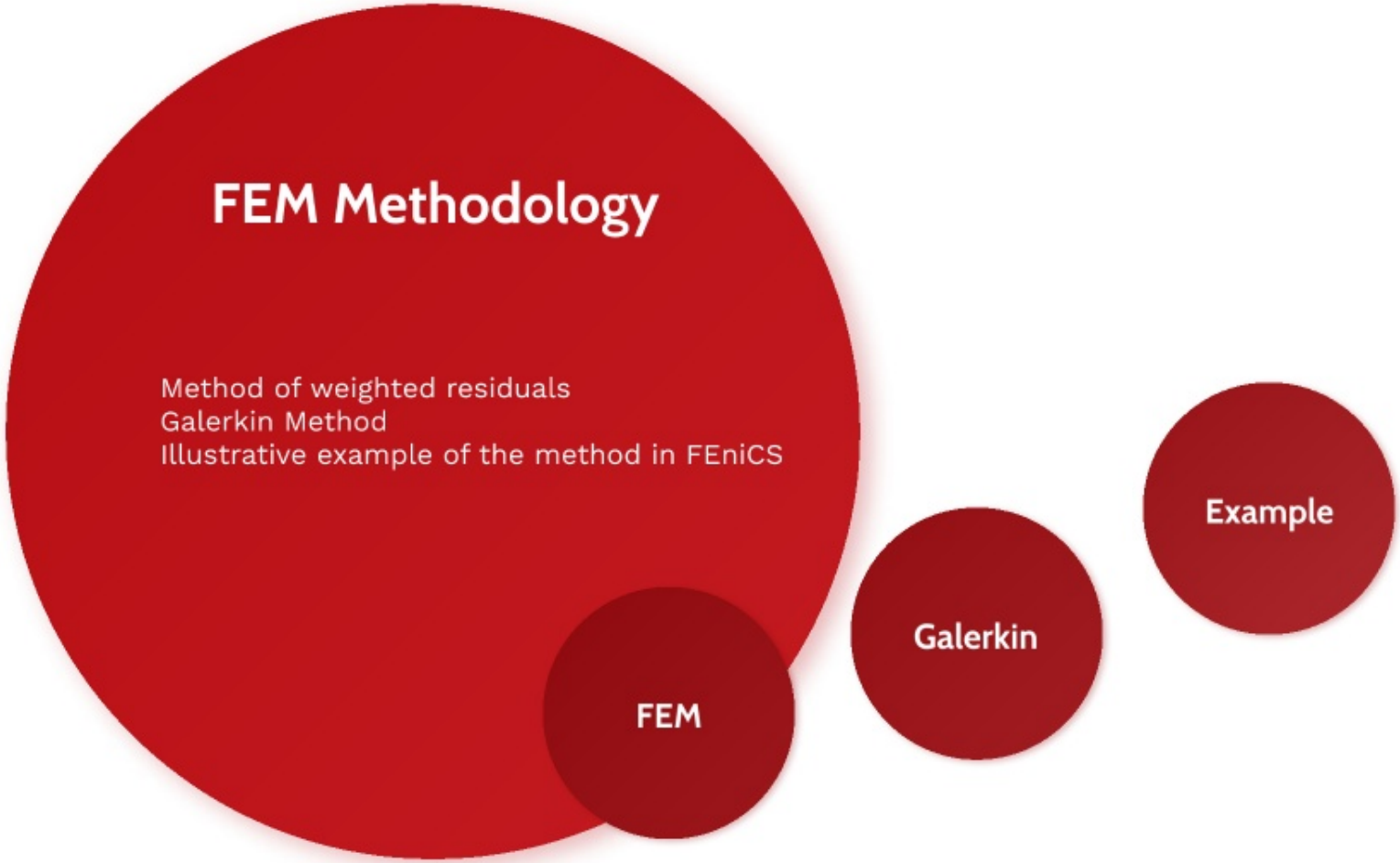
Getting started

Demos

Other Features

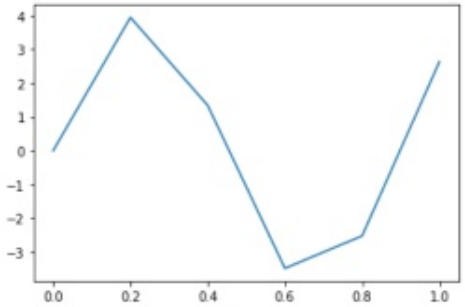



Automating the finite element method
Components of the framework
1. FEM Methodology
2. Components
3. UFL
4. FFC
5.UFC
6. Dolfin
7. PETSc
Takeaways


FEM Methodology
Method of weighted residuals
Galerkin Method
Illustrative example of the method in FEniCS
FEM
Galerkin
Example

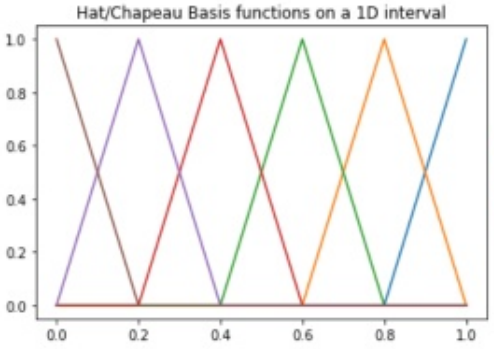
Weighted residual

1. Create a Finite Element mesh, $T = \{T_k\}$, k $\in [1, n]$, where T_k is an element of dimension d
2. Given a governing differential eq. $F(x_1 \dots x_n, u, \frac{\partial u}{\partial x} \dots \frac{\partial^2 u}{\partial x^2}) = 0$ with boundary and initial conditions, assume there exists an approximate solution u_h on T
3. $F(u_h) = R$, where R is the residual
4. Drive the residual to zero



Basis function

$u_h(x) = \sum_{j=1}^{n+1} U_j \Phi_j(x)$, $U \in R^d$ are coefficients



Shape function - a function able to capture the behavior of the quantity we are solving for

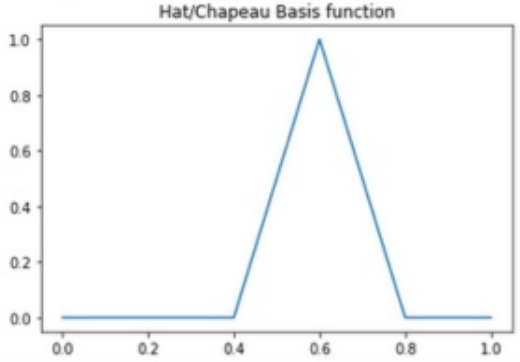
We construct a pair of discrete trial and test spaces V_h and $\hat{V}h$, $\{\Phi_j\}_{j=1}^{n+1}$ and $\{\hat{\Phi}_j\}_{j=1}^{n+1}$

Shape function

$$\phi_i = \begin{cases} 0 & \text{if } x < x_{i-1} \\ \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if } x_{i-1} < x < x_i \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if } x_i < x < x_{i+1} \\ 0 & \text{if } x > x_{i+1} \end{cases}$$

```
1 n = 5
2 mesh = UnitIntervalMesh(n)
3 V = FunctionSpace(mesh, "Lagrange", 1)
4 f2 = Function(V)
5 f2.vector()[2] = 1 # Assigning coefficient 1 at node x=0.6
6 plot(f2, title="Hat/Chapeau Basis function")
```

```
[<matplotlib.lines.Line2D at 0x7f515ffa5e10>]
```



Galerkin
Galerkin orthogonality

$\int_0^L \Phi_i R \, dx = 0$

$\int_0^L \hat{\Phi}_i F(x_1 \dots x_n, u_h, \frac{\partial u_h}{\partial x} \dots \frac{\partial^2 u_h}{\partial x^2}) \, dx = 0,$
u_h is the approximate solution

Using the same basis functions for the trial space basis functions $\hat{\Phi}_i$ and Φ_i in $u_h(x) = \sum_{i=1}^{n+1} U_i \Phi_i(x)$

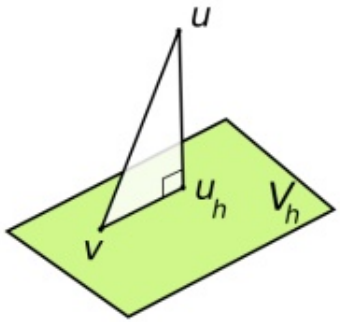
Galerkin orthogonality

$R = u - u_h, \; v \in \hat{V}$

$a(R, v) = a(u, v) - a(u_n, v) = f(v) - f(v) = 0$

Proof of optimality

The approximation is as close to a original solution as any other vector in V_h



Example, Laplace

$$\int_\Omega \nabla u \cdot \nabla \phi = \int_\Omega f\phi.$$

$$\int_\Omega \sum_{j=1}^{n-1} u_j \nabla \phi_j \nabla \phi_i = \int_\Omega f\phi_i.$$

$$A\vec{u} = \vec{b}$$

where matrix A is defined as

$$A_{ij} = \int_\Omega \nabla \phi_i \nabla \phi_j$$

and vector $\vec{b}$ is defined as

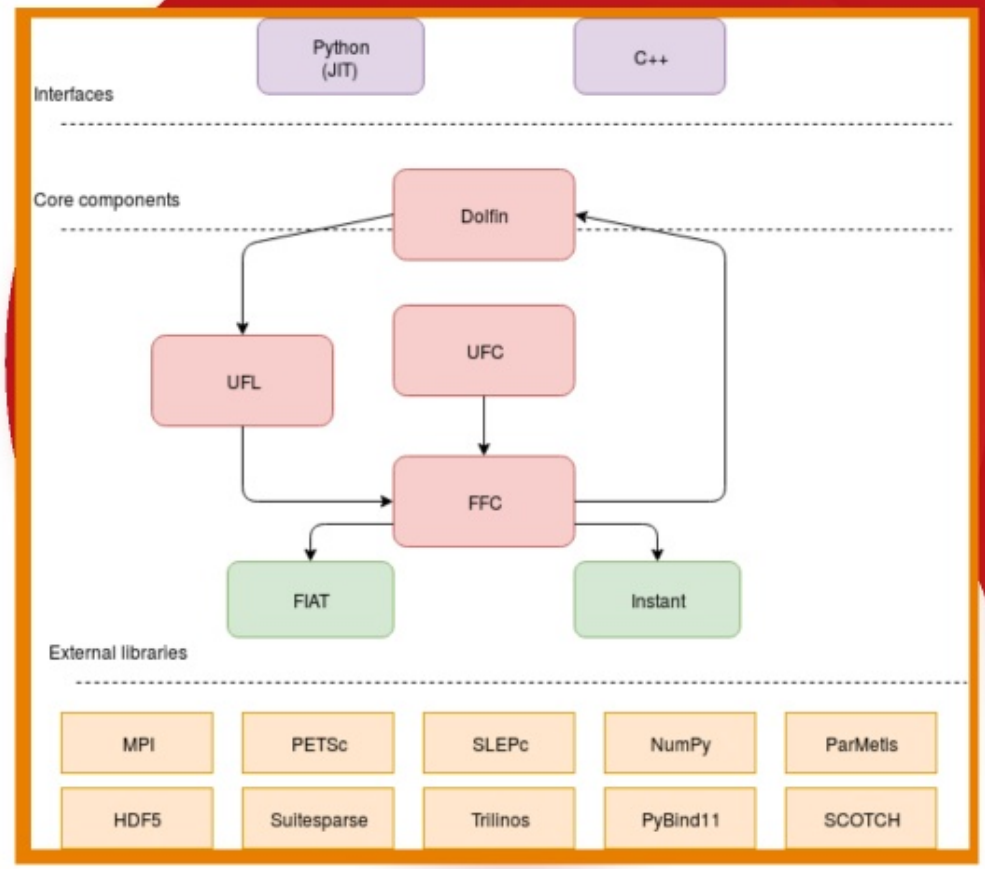
$$\vec{b}_i = \int_\Omega f\phi_i.$$

stiffness matrix

$$A_{ij} = \int \nabla \phi_i \nabla \phi_j = \int_0^1 \phi_i'(x)\phi_j'(x)dx = \begin{cases} 2/h & \text{if } i = j \\ -1/h & \text{if } i = j \pm 1 \\ 0 & \text{else} \end{cases}$$

https://colab.research.google.com/drive/1XFGzBU_Z9RrCIn2x73xgoYb21DG1MeQX?authuser=1#scrollTo=4aEHr5Q8jF-l

Components


Python
(JIT)
C++
Interfaces
Core components
Dolfin
UFL
UFC
FFC
FIAT
Instant
External libraries
MPI
PETSc
SLEPc
NumPy
ParMetis
HDF5
Suitesparse
Trilinos
PyBind11
SCOTCH

Unified Form Language

```
# Compile this form with FFC: ffc -l dolfin TentativeVelocity.ufl.
# Define function spaces (P2-P1)
V = VectorElement("Lagrange", triangle, 2)
Q = FiniteElement("Lagrange", triangle, 1)

# Define trial and test functions
u = TrialFunction(V)
v = TestFunction(V)

# Define coefficients
k  = Constant(triangle)
u0 = Coefficient(V)
f  = Coefficient(V)
nu = 0.01

# Define bilinear and linear forms
eq = (1/k)*inner(u - u0, v)*dx + inner(grad(u0)*u0, v)*dx + \
    nu*inner(grad(u), grad(v))*dx - inner(f, v)*dx
a  = lhs(eq)
L  = rhs(eq)
```

$$\dot{u} + \nabla u \cdot u - \nu \Delta u + \nabla p = f,$$
$$\nabla \cdot u = 0.$$

$$\langle (u_h^\star - u_h^{n-1})/\Delta t_n, v \rangle + \langle \nabla u_h^{n-1} \cdot u_h^{n-1}, v \rangle + \langle \nu \nabla u_h^n, \nabla v \rangle = \langle f, v \rangle.$$

Python

.ufl vs Python interface

```
# Define function spaces (P2-P1)
V = VectorFunctionSpace(mesh, "Lagrange", 2)
Q = FunctionSpace(mesh, "Lagrange", 1)

# Define trial and test functions
u = TrialFunction(V)
p = TrialFunction(Q)
v = TestFunction(V)
q = TestFunction(Q)
```

```
# Create functions
u0 = Function(V)
u1 = Function(V)
p1 = Function(Q)

# Define coefficients
k = Constant(dt)
f = Constant((0, 0))

# Tentative velocity step
F1 = (1/k)*inner(u - u0, v)*dx + inner(grad(u0)*u0, v)*dx + \
     nu*inner(grad(u), grad(v))*dx - inner(f, v)*dx
a1 = lhs(F1)
L1 = rhs(F1)
```

```
# Compile this form with FFC: ffc -l dolfin TentativeVelocity.ufl.

# Define function spaces (P2-P1)
V = VectorElement("Lagrange", triangle, 2)
Q = FiniteElement("Lagrange", triangle, 1)

# Define trial and test functions
u = TrialFunction(V)
v = TestFunction(V)

# Define coefficients
k  = Constant(triangle)
u0 = Coefficient(V)
f  = Coefficient(V)
nu = 0.01

# Define bilinear and linear forms
eq = (1/k)*inner(u - u0, v)*dx + inner(grad(u0)*u0, v)*dx + \
    nu*inner(grad(u), grad(v))*dx - inner(f, v)*dx
a  = lhs(eq)
L  = rhs(eq)
```

.ufl vs Python interface

- Tensor algebra operators:

```
outer, inner, dot, cross, perp,
det, inv, cofac,
transpose, tr, diag, diag_vector,
dev, skew, sym
```

- Elementwise tensor operators:

```
elem_mult, elem_div, elem_pow, elem_op
```

- Differential operators:

```
variable, diff,
grad, div, nabla_grad, nabla_div,
Dx, Dn, curl, rot
```

- Nonlinear functions:

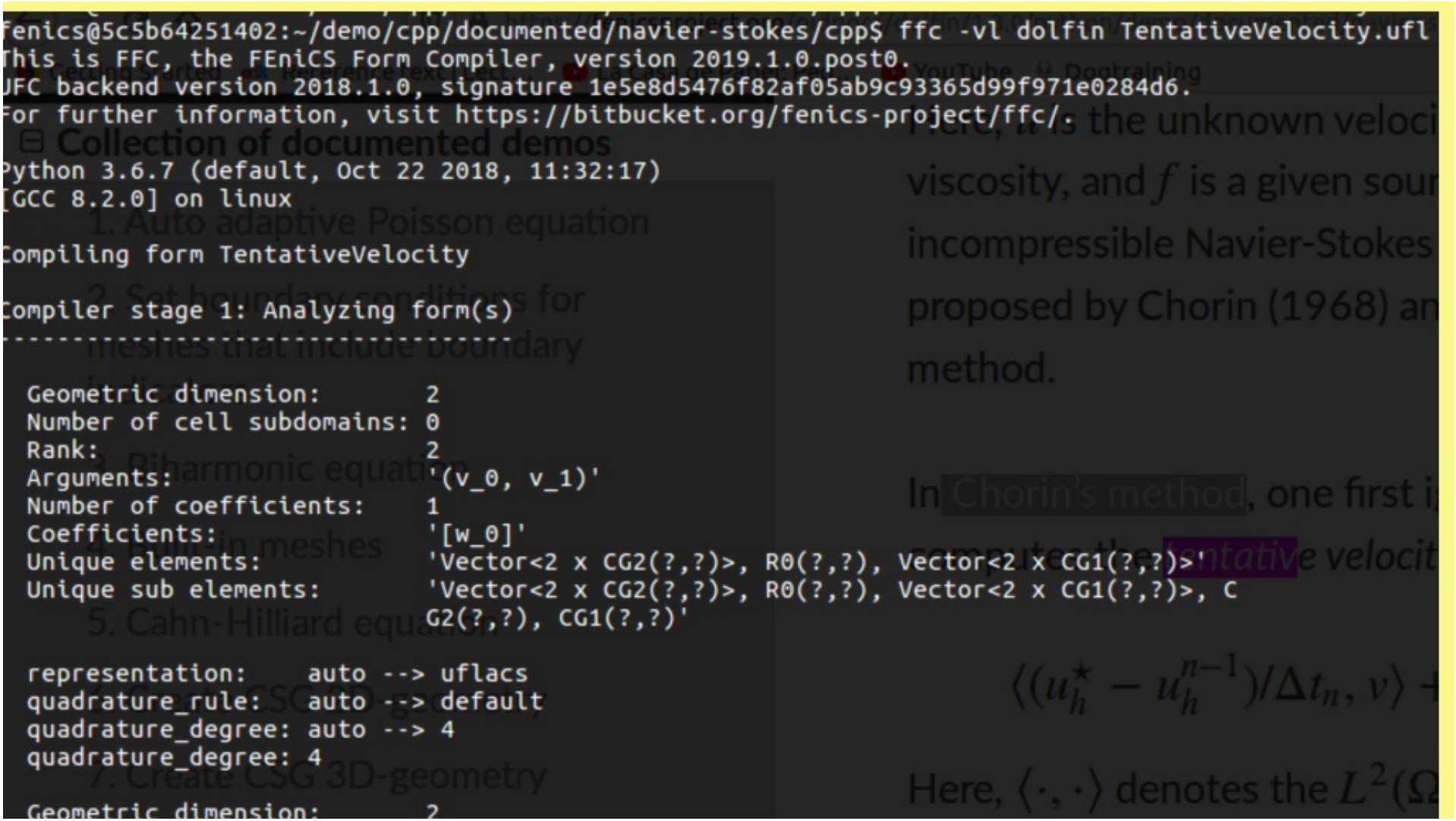
```
max_value, min_value,
abs, sign, sqrt,
exp, ln, erf,
cos, sin, tan,
acos, asin, atan, atan_2,
cosh, sinh, tanh,
bessel_J, bessel_Y, bessel_I, bessel_K
```

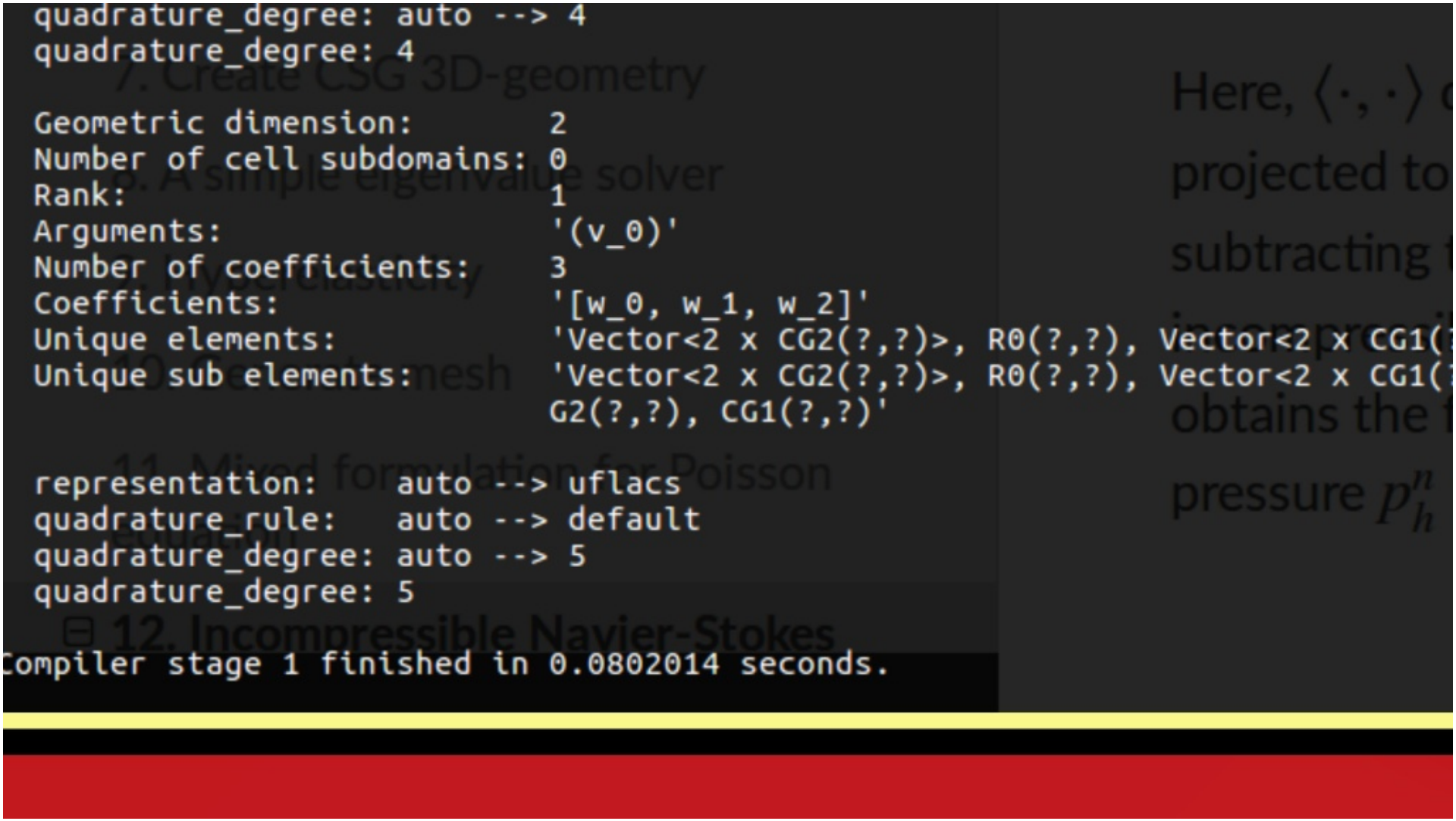
- **Discontinuous Galerkin operators:**
  jump, avg, v('+'), v('-'), cell_avg, facet_avg

FFC

```
fenics@5c5b64251402:~/demo/cpp/documented/navier-stokes/cpp$ ffc -vl dolfin TentativeVelocity.ufl
This is FFC, the FEniCS Form Compiler, version 2019.1.0.post0.
UFC backend version 2018.1.0, signature 1e5e8d5476f82af05ab9c93365d99f971e0284d6.
For further information, visit https://bitbucket.org/fenics-project/ffc/.
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux

Compiling form TentativeVelocity

Compiler stage 1: Analyzing form(s)
--------------------------------

  Geometric dimension:       2
  Number of cell subdomains: 0
  Rank:                      2
  Arguments:                 '(v_0, v_1)'
  Number of coefficients:    1
  Coefficients:              '[w_0]'
  Unique elements:           'Vector<2 x CG2(?,?)>, R0(?,?), Vector<2 x CG1(?,?)>'
  Unique sub elements:       'Vector<2 x CG2(?,?)>, R0(?,?), Vector<2 x CG1(?,?)>, C
                             G2(?,?), CG1(?,?)'

  representation:    auto --> uflacs
  quadrature_rule:   auto --> default
  quadrature_degree: auto --> 4
  quadrature_degree: 4

  Geometric dimension:       2
  Number of cell subdomains: 0
  Rank:                      1
  Arguments:                 '(v_0)'
  Number of coefficients:    3
  Coefficients:              '[w_0, w_1, w_2]'
  Unique elements:           'Vector<2 x CG2(?,?)>, R0(?,?), Vector<2 x CG1(?,?)>'
  Unique sub elements:       'Vector<2 x CG2(?,?)>, R0(?,?), Vector<2 x CG1(?,?)>, C
                             G2(?,?), CG1(?,?)'

  representation:    auto --> uflacs
  quadrature_rule:   auto --> default
  quadrature_degree: auto --> 5
  quadrature_degree: 5

Compiler stage 1 finished in 0.0802014 seconds.
```

```
Compiler stage 2: Computing intermediate representation
--------------------------------------------------------
  Computing representation of 5 elements
  Computing representation of 5 dofmaps
  Computing representation of 1 coordinate mappings
  Computing representation of integrals
  Computing uflacs representation
  Computing uflacs representation
  Computing representation of forms

Compiler stage 2 finished in 0.204529 seconds.

Compiler stage 3: Optimizing intermediate representation
---------------------------------------------------------
  Optimizing uflacs representation
  Optimizing uflacs representation

Compiler stage 3 finished in 0.000309706 seconds.

Compiler stage 4: Generating code
---------------------------------
  Generating code for 5 finite_element(s)
  Generating code for 5 dofmap(s)
  Generating code for 1 coordinate_mapping(s)
  Generating code for integrals
  Generating code from ffc.uflacs representation
  Generating code from ffc.uflacs representation
  Generating code for forms
```


fenics@5c5b64251402:~/demo/cpp/documented/navier-stokes/cpp$ ffc -vl dolfin TentativeVelocity.ufl
This is FFC, the FEniCS Form Compiler, version 2019.1.0.post0.
UFC backend version 2018.1.0, signature 1e5e8d5476f82af05ab9c93365d99f971e0284d6.
For further information, visit https://bitbucket.org/fenics-project/ffc/.
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Compiling form TentativeVelocity
Compiler stage 1: Analyzing form(s)
Geometric dimension: 2
Number of cell subdomains: 0
Rank: 2
Arguments: '(v_0, v_1)'
Number of coefficients: 1
Coefficients: '[w_0]'
Unique elements: 'Vector<2 x CG2(?,?)>, R0(?,?), Vector<2 x CG1(?,?)>'
Unique sub elements: 'Vector<2 x CG2(?,?)>, R0(?,?), Vector<2 x CG1(?,?)>, CG2(?,?), CG1(?,?)'
representation: auto --> uflacs
quadrature_rule: auto --> default
quadrature_degree: auto --> 4
quadrature_degree: 4
Geometric dimension: 2


quadrature_degree: auto --> 4
quadrature_degree: 4
Geometric dimension: 2
Number of cell subdomains: 0
Rank: 1
Arguments: '(v_0)'
Number of coefficients: 3
Coefficients: '[w_0, w_1, w_2]'
Unique elements: 'Vector<2 x CG2(?,?)>, R0(?,?), Vector<2 x CG1(
Unique sub elements: 'Vector<2 x CG2(?,?)>, R0(?,?), Vector<2 x CG1(
G2(?,?), CG1(?,?)'
representation: auto --> uflacs
quadrature_rule: auto --> default
quadrature_degree: auto --> 5
quadrature_degree: 5
Compiler stage 1 finished in 0.0802014 seconds.

Output of FFC using UFC

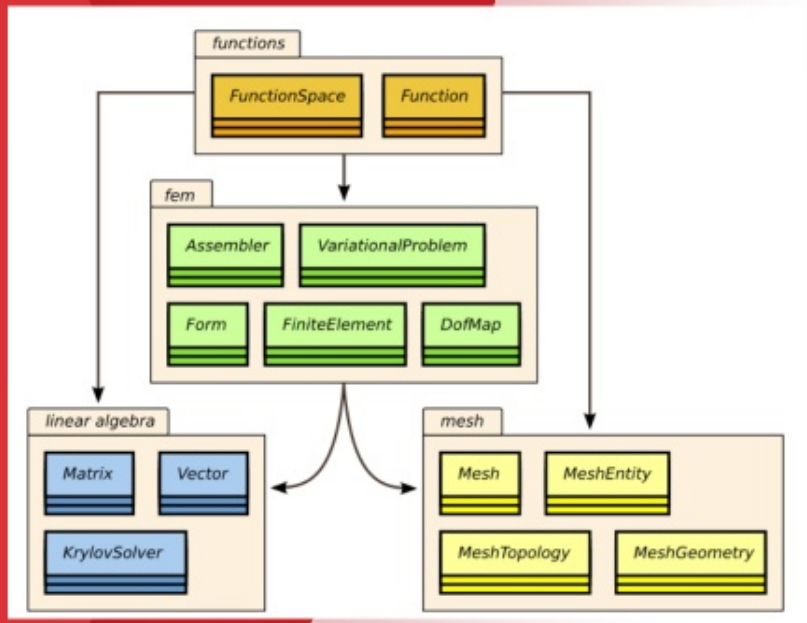
```
// This code conforms with the UFC specification version 2018.1.0
// and was automatically generated by FFC version 2019.1.0.post0.
//
// This code was generated with the option '-l dolfin' and
// contains DOLFIN-specific wrappers that depend on DOLFIN.
//
// This code was generated with the following parameters:
//

//   add_tabulate_tensor_timing:      False
//   convert_exceptions_to_warnings: False
//   cpp_optimize:                    True
//   cpp_optimize_flags:              '-O2'
//   epsilon:                         1e-14
//   error_control:                   False
//   external_include_dirs:           ''
//   external_includes:               ''
//   external_libraries:              ''
//   external_library_dirs:           ''
//   form_postfix:                    True
//   format:                          'dolfin'
//   generate_dummy_tabulate_tensor: False
//   max_signature_length:            0
//   optimize:                        True
//   precision:                       None
//   quadrature_degree:               None
//   quadrature_rule:                 None
//   representation:                  'auto'
//   split:                           False

#ifndef __TENTATIVEVELOCITY_H
#define __TENTATIVEVELOCITY_H
#include <algorithm>
#include <cmath>
#include <iostream>
#include <stdexcept>
#include <ufc.h>

class tentativevelocity_finite_element_0: public ufc::finite_element
{
public:

  tentativevelocity_finite_element_0() : ufc::finite_element()
  {
    // Do nothing
  }
```

```
void tabulate_tensor(double * A,
                     const double * const * w,
                     const double * coordinate_dofs,
                     int cell_orientation) const final override
{
  // This function was generated using 'uflacs' representation
  // with the following integrals metadata:
  //
  // num_cells:          None
  // optimize:           True
  // precision:          16
  // quadrature_degree: 4
  // quadrature_rule:    'default'
  // representation:     'uflacs'
  alignas(32) static const double FE16_C0_D01_Q6[1][1][2] = { { { -1.0, 1.0 } } };
  // Unstructured piecewise computations
  const double J_c0 = coordinate_dofs[0] * FE16_C0_D01_Q6[0][0][0] + coordinate_dofs[2] * FE16_C0_D01_Q6[0][0][1];
  const double J_c3 = coordinate_dofs[1] * FE16_C0_D01_Q6[0][0][0] + coordinate_dofs[5] * FE16_C0_D01_Q6[0][0][1];
  const double J_c1 = coordinate_dofs[0] * FE16_C0_D01_Q6[0][0][0] + coordinate_dofs[4] * FE16_C0_D01_Q6[0][0][1];
  const double J_c2 = coordinate_dofs[1] * FE16_C0_D01_Q6[0][0][0] + coordinate_dofs[3] * FE16_C0_D01_Q6[0][0][1];
  alignas(32) double sp[21];
  sp[0] = J_c0 * J_c3;
  sp[1] = J_c1 * J_c2;
  sp[2] = sp[0] + -1 * sp[1];
  sp[3] = J_c0 / sp[2];
  sp[4] = -1 * J_c1 / sp[2];
  sp[5] = sp[3] * sp[3];
  sp[6] = sp[3] * sp[4];
```

```
void tabulate_tensor(double * A,
                     const double * const * w,
                     const double * coordinate_dofs,
                     int cell_orientation) const final override
{
  // This function was generated using 'uflacs' representation
  // with the following integrals metadata:
  //
  // num_cells:          None
  // optimize:           True
  // precision:          16
  // quadrature_degree: 4
  // quadrature_rule:    'default'
  // representation:     'uflacs'
  alignas(32) static const double FE16_C0_D01_Q6[1][1][2] = { { { -1.0, 1.0 } } };
  // Unstructured piecewise computations
  const double J_c0 = coordinate_dofs[0] * FE16_C0_D01_Q6[0][0][0] + coordinate_dofs[2] * FE16_C0_D0
  const double J_c3 = coordinate_dofs[1] * FE16_C0_D01_Q6[0][0][0] + coordinate_dofs[5] * FE16_C0_D0
  const double J_c1 = coordinate_dofs[0] * FE16_C0_D01_Q6[0][0][0] + coordinate_dofs[4] * FE16_C0_D0
  const double J_c2 = coordinate_dofs[1] * FE16_C0_D01_Q6[0][0][0] + coordinate_dofs[3] * FE16_C0_D0
  alignas(32) double sp[21];
  sp[0] = J_c0 * J_c3;
  sp[1] = J_c1 * J_c2;
  sp[2] = sp[0] + -1 * sp[1];
  sp[3] = J_c0 / sp[2];
  sp[4] = -1 * J_c1 / sp[2];
  sp[5] = sp[3] * sp[3];
  sp[6] = sp[3] * sp[4];
```

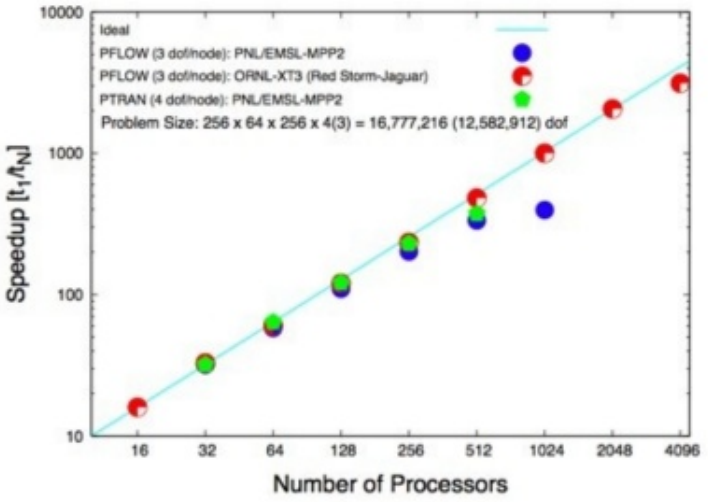
Dolfin

- C++/Python library/interfaces
- Wrappers for other FEniCS components and external libraries simplify the user interface
- Provides classes such as Matrix , Vector , Mesh , FiniteElement , FunctionSpace, Function, Assembler
- Advanced usage of native functionality



PETSc

"PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism."



PETSc

Direct solvers	LU	LU	seqaij, seqbaij	--		X
			seqaij	MATLAB		X
			aij	PaStiX (Inria)	X	X
			aij	SuperLU (BNL) SuperLU Sequential LU solver / SuperLU_DIST Parallel LU solver	X	X
			aij, baij	MUMPS (CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, Mumps Technologies, University of Bordeaux)	X	X
			seqaij	ESSL (IBM)		
			seqaij	UMFPACK Part of SuiteSparse		X
			seqaij	KLU Part of SuiteSparse		X
			seqaij	LUSOL		
			seqaij, seqbaij	MKL Pardiso (Intel)		X
			mpiaij, mpibaij	MKL CPardiso (Intel)	X	X
			dense	--	X	X
	Cholesky	Cholesky	seqaij, seqsbaij	--		X
			sbaij	PaStiX (Inria)	X	X
			sbaij	MUMPS (CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, Mumps Technologies, University of Bordeaux)	X	X
			seqaij, seqsbaij	CHOLMOD Part of SuiteSparse		X
			dense	--	X	X
			seqsbaij	MKL Pardiso (Intel)		
			mpisbaij	MKL CPardiso (Intel)	X	
	QR		matlab	MATLAB		
	XXt and XYt		aij	--	X	

to run copies of single application than containers are arguably t

Direct solvers	LU	LU	seqaij, seqbaij	--		X
			seqaij	MATLAB		X
			aij	PaStiX (Inria)	X	X
			aij	SuperLU (BNL) SuperLU Sequential LU solver / SuperLU_DIST Parallel LU solver	X	X
			aij, baij	MUMPS (CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, Mumps Technologies, University of Bordeaux)	X	X
			seqaij	ESSL (IBM)		
			seqaij	UMFPACK Part of SuiteSparse		X
			seqaij	KLU Part of SuiteSparse		X
			seqaij	LUSOL		
			seqaij, seqbaij	MKL Pardiso (Intel)		X
			mpiaij, mpibaij	MKL CPardiso (Intel)	X	X
			dense	--	X	X
	Cholesky	Cholesky	seqaij, seqsbaij	--		X
			sbaij	PaStiX (Inria)	X	X
			sbaij	MUMPS (CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, Mumps Technologies, University of Bordeaux)	X	X
			seqaij, seqsbaij	CHOLMOD Part of SuiteSparse		X
			dense	--	X	X
			seqsbaij	MKL Pardiso (Intel)		
			mpisbaij	MKL CPardiso (Intel)	X	
	QR		matlab	MATLAB		
	XXt and XYt		aij	--	X	

https://www.mcs.anl.gov/petsca/documentation/linearsolvertable.html

Direct solvers	LU	LU	seqaij, seqbaij	---		X
			seqaij	MATLAB		X
			aij	PaStiX (Inria)	X	X
			aij	SuperLU (BNL) SuperLU Sequential LU solver / SuperLU_DIST Parallel LU solver	X	X
			aij, baij	MUMPS (CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, Mumps Technologies, University of Bordeaux)	X	X
			seqaij	ESSL (IBM)		
			seqaij	UMFPACK Part of SuiteSparse		X
			seqaij	KLU Part of SuiteSparse		X
			seqaij	LUSOL		
			seqaij, seqbaij	MKL Pardiso (Intel)		X
			mpiaij, mpibaij	MKL CPardiso (Intel)	X	X
			dense	---	X	X
	Cholesky	Cholesky	seqaij, seqsbaij	---		X
			sbaij	PaStiX (Inria)	X	X
			sbaij	MUMPS (CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, Mumps Technologies, University of Bordeaux)	X	Direct solv
			seqaij, seqsbaij	CHOLMOD Part of SuiteSparse		
			dense	---	X	
			seqsbaij	MKL Pardiso (Intel)		
			mpisbaij	MKL CPardiso (Intel)	X	
	QR		matlab	MATLAB		
	XXt and XYt		aij	---	X	

to run copies
application tha
are arguably t

ies of single
than containers
y t

Direct solvers	LU	LU	seqaij, seqbaij	---		X
			seqaij	MATLAB		X
			aij	PaStiX (Inria)	X	X
			aij	SuperLU (BNL) SuperLU Sequential LU solver / SuperLU_DIST Parallel LU solver	X	X
			aij, baij	MUMPS (CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, Mumps Technologies, University of Bordeaux)	X	X
			seqaij	ESSL (IBM)		
			seqaij	UMFPACK Part of SuiteSparse		X
			seqaij	KLU Part of SuiteSparse		X
			seqaij	LUSOL		
			seqaij, seqbaij	MKL Pardiso (Intel)		X
			mpiaij, mpibaij	MKL CPardiso (Intel)	X	X
			dense	---	X	X
	Cholesky	Cholesky	seqaij, seqsbaij	---		X
			sbaij	PaStiX (Inria)	X	X
			sbaij	MUMPS (CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, Mumps Technologies, University of Bordeaux)	X	X
			seqaij, seqsbaij	CHOLMOD Part of SuiteSparse		X
			dense	---	X	X
			seqsbaij	MKL Pardiso (Intel)		
			mpisbaij	MKL CPardiso (Intel)	X	
	QR		matlab	MATLAB		
	XXt and XYt		aij	---	X	

Takeaways

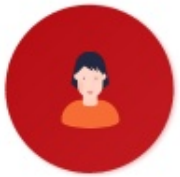
The FEniCS project seeks to automate the solution of differential equations, relying on the following key steps:

- **automation of discretization**
- **automation of discrete solution**
- **automation of error control.**

https://www.minecraft.net/en-us/article/carving-dragons

FEniCS Platform

Introduction

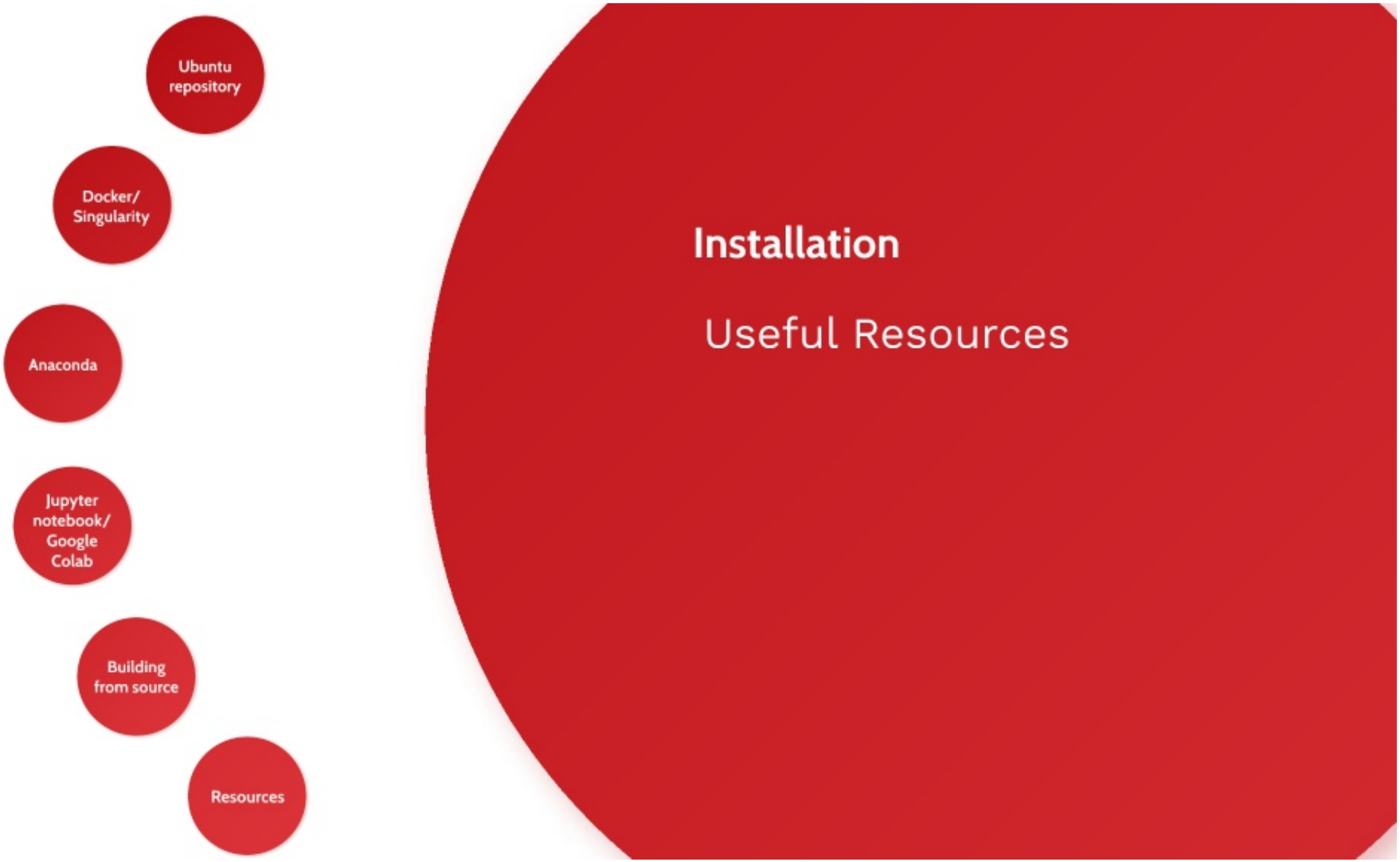


How it works

Getting started

Demos

Other Features


Ubuntu repository
Docker/ Singularity
Anaconda
Jupyter notebook/ Google Colab
Building from source
Resources
Installation
Useful Resources

Ubuntu PPA

Install the latest version of FEniCS from the Ubuntu Personal Package Archives (PPA)

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

Security	The Docker daemon is a service that runs on your host operating system and controls the containers. It requires giving root access to the users.	Runs container as a normal user process using the current user. Note: You can run the container w/wo root, you cannot elevate change user from inside the container
Namespace/Environment	Isolation of the process, network and user space	Transparency, can see the environment and all the processes running on the host. /home, /var/tmp, /tmp, /root are bind mount from host and fully writable by default
MPI/Job scheduling	Swarm mode, other open source alternatives	Can be used with any scheduler. Process Management Interface (PMI/PMIx) calls will pass through the singularity launcher onto ORTED.
Portability	Docker images are stored as filesystem layers. Sharing is safest through private or public registries (default one - Docker Hub)	One file, more convenient
Misc		Direct access to hardware devices ex GPU, optimized MPI, file systems LUSTRE etc. Faster initialization of the container. Can convert Docker images to Singularity images, as well as run Docker containers

Docker, Singularity

On Linux, Mac, Windows

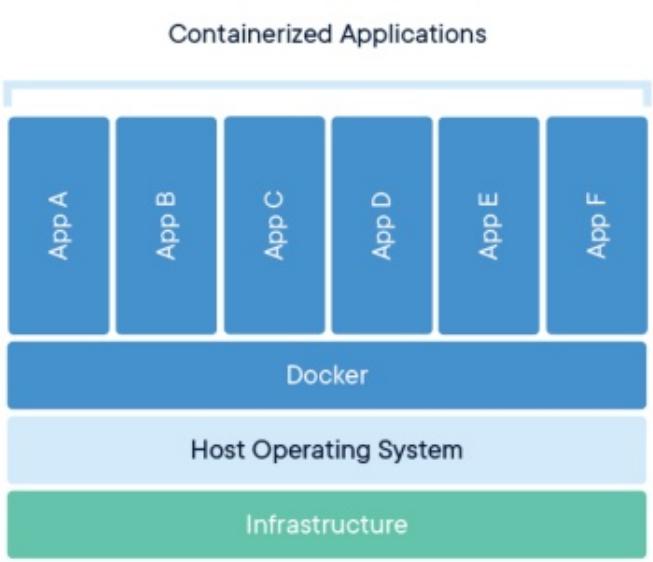
Docker

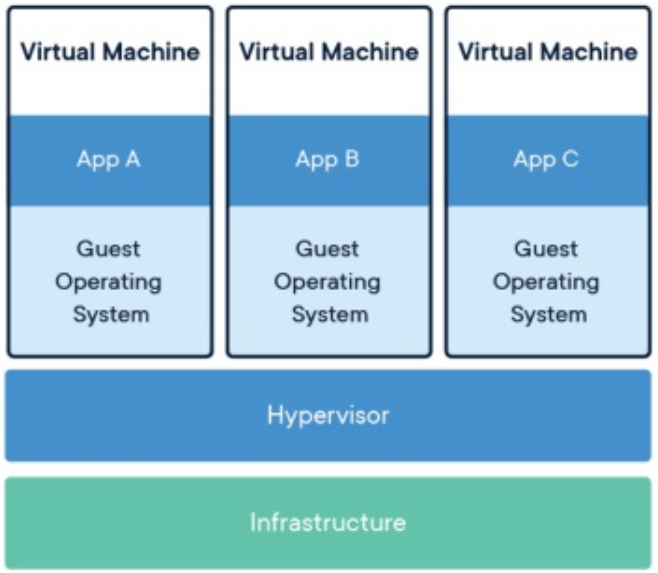
Singularity

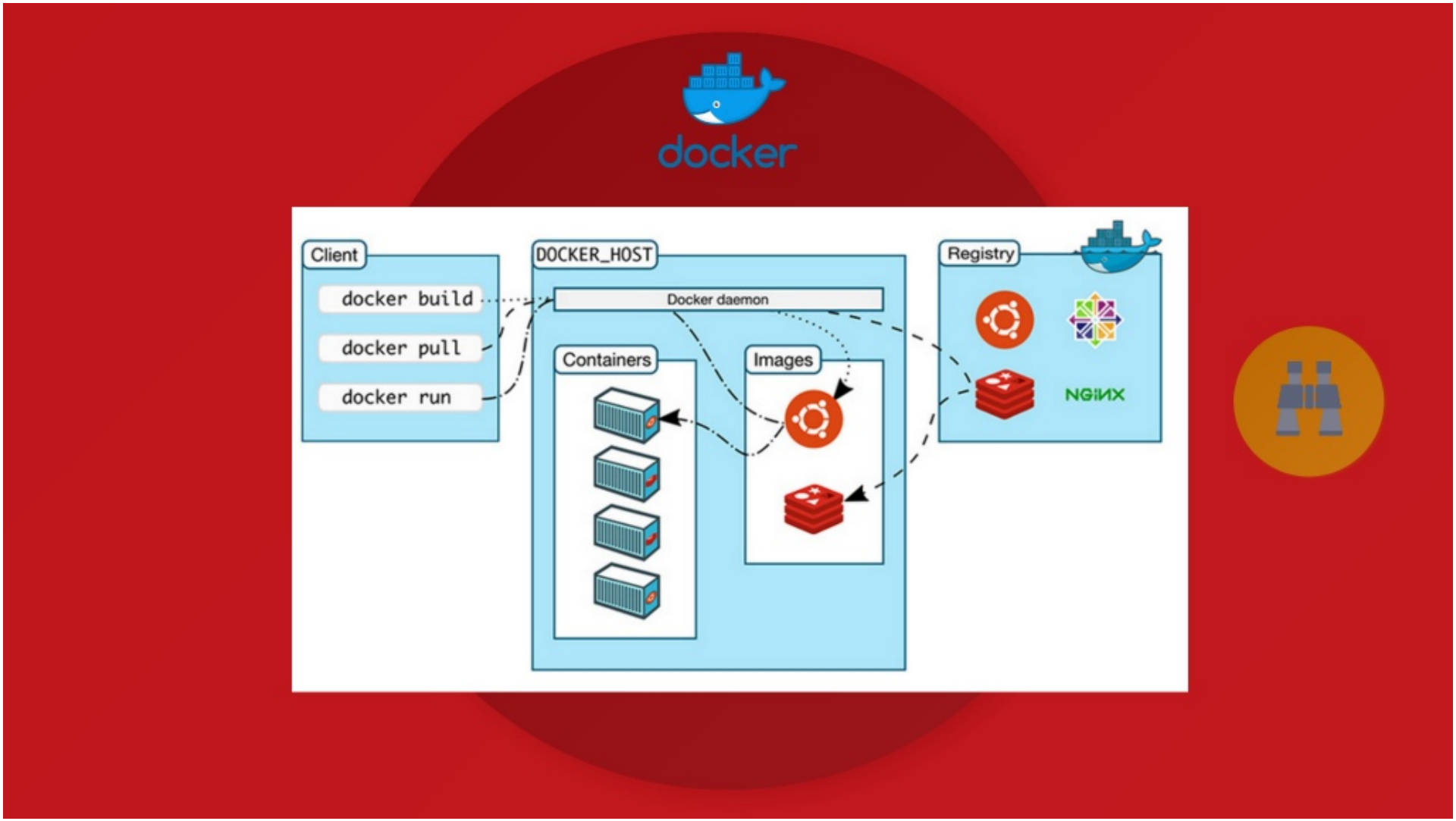
Security	The Docker daemon is a service that runs on your host operating system and controls the containers. It requires giving root access to the users.	Runs container as a normal user process using the current user. Note: You can run the container w/wo root, you cannot elevate change user from inside the container
Namespace/Environment	Isolation of the process, network and user space	Transparency, can see the environment and all the processes running on the host. /home, /var/tmp, /tmp, /root are bind mount from host and fully writable by default
MPI/Job scheduling	Swarm mode, other open source alternatives	Can be used with any scheduler. Process Management Interface (PMI/PMIx) calls will pass through the singularity launcher onto ORTED.
Portability	Docker images are stored as	One file, more convenient

MPI/JOB scheduling	Swarm mode, other open source alternatives	Can be used with any scheduler. Process Management Interface (PMI/PMIx) calls will pass through the singularity launcher onto ORTED.
Portability	Docker images are stored as filesystem layers. Sharing is safest through private or public registries (default one - Docker Hub)	One file, more convenient
Misc		Direct access to hardware devices ex GPU, optimized MPI, file systems LUSTRE etc. Faster initialization of the container. Can convert Docker images to Singularity images, as well as run Docker containers

VM vs. Container






docker
Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

Docker instructions

```
tdancheva@DULX0033:~$ docker volume create fenics-vol
fenics-vol
tdancheva@DULX0033:~$ docker volume inspect fenics-vol
[
    {
        "CreatedAt": "2019-11-13T10:18:05+01:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/fenics-vol/_data",
        "Name": "fenics-vol",
        "Options": {},
        "Scope": "local"
    }
]
```

```
docker run --name=fenics19-ltu --rm -ti -p
127.0.0.1:8000:8000 -v $(pwd):/home/fenics/shared
-w /home/fenics/shared --mount source=fenics-
vol,target=/home/fenics/mydata quay.io/
fenicsproject/stable:2019.1.0.r3

other commands: docker ps -a
                docker stop containerId/Name
                docker rm containerId/Name
```

link to full docker instructions

```
tdancheva@DULX0033:~$  docker run --name=fenics19-ltu -ti --rm -p 127.0.0.1:8000:8000 -v $(pwd):/home/fenics/shared -w /home/fenics/shared --mount source=fenics-vol,targ
et=/home/fenics/mydata quay.io/fenicsproject/stable:2019.1.0.r3
# FEniCS stable version image

Welcome to FEniCS/stable!

This image provides a full-featured and optimized build of the stable
release of FEniCS.

To help you get started this image contains a number of demo
programs. Explore the demos by entering the 'demo' directory, for
example:

    cd ~/demo/python/documented/poisson
    python3 demo_poisson.py
fenics@dda103f5b22f:~/shared$ exit
exit
```

https://bitbucket.org/fenics-project/docker/src/master/

Singularity

https://bitbucket.org/fenics-project/docker/src/master/

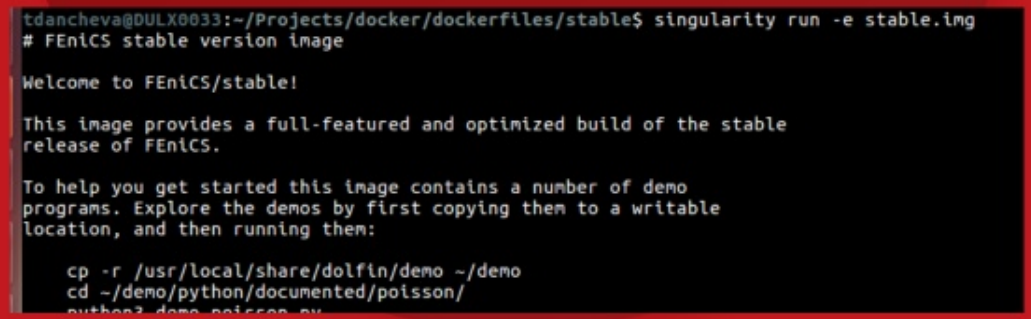
- Building a Singularity image from a docker image pulled from docker hub
- "As of Singularity v2.4 by default build produces immutable images in the squashfs file format. This ensures reproducible and verifiable images."
- Writable containers with:

```
$ sudo singularity build --sandbox ubuntu/ docker://ubuntu
```

- Mount volumes, no need for --writable on mount locations, works on read-only images

```
$ singularity shell --bind /opt,/data:/mnt
my_container.sif
```

- Singularity Hub, Docker Hub



1) Install FEniCS in a new virtual environment fenicsproject

2) Activate the new environment

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

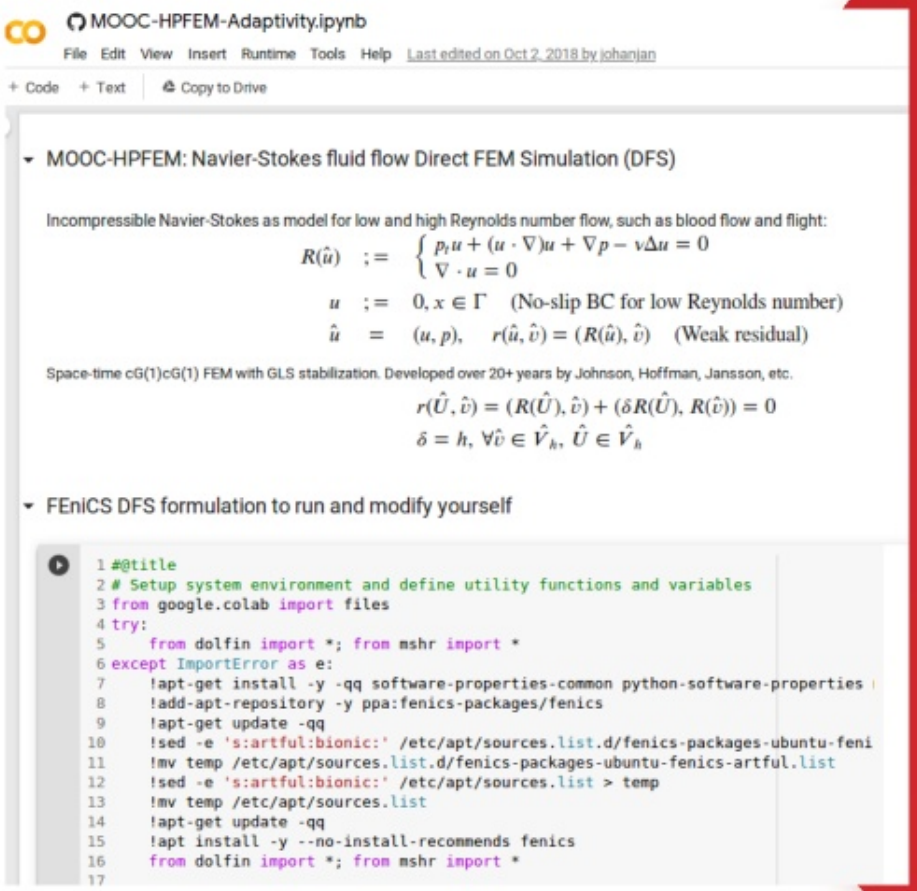
Warning: When installing on clusters, comes with prescribed MPI libraries. Use Singularity or install from source if possible.

Anaconda/Miniconda

https://anaconda.org/conda-forge/fenics

"A community led collection of recipes, build infrastructure and distributions for the conda package manager. Packages."

Package management system conda



Google Colab

Free Jupyter notebook environment that runs entirely in Google cloud.

Jupyter notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and text.

- Operating system: Ubuntu 18.04.2 LTS
- Processors: 2 x Intel(R) Xeon(R) CPU @ 2.30GHz
- RAM: 13 GB
- Storage: 40GB

Google Colab-Hosted runtime

Google Colab-Local runtime

Docker & Jupyter notebooks

Google Cloud connecting to a Local Runtime

This command creates a Docker container locally from the latest stable FEniCS version at the time of writing, given with the fenics_tag variable. Inside this container, we install a Jupyter extension developed by Google Colaboratory's developers and then run a Jupyter notebook from within the container on port 8888.

```
fenics_tag=2019.1.0.r3 # version of FEniCS image
docker run --name notebook-local -w /home/fenics -v
,→ $(pwd):/home/fenics/shared -ti -d -p 127.0.0.1:8888:8888
,→ quay.io/fenicsproject/stable:${fenics_tag} "sudo pip install
,→ jupyter_http_over_ws; sudo apt-get install -y gmsh; jupyter
,→ serverextension enable --py jupyter_http_over_ws;
,→ jupyter-notebook --ip=0.0.0.0
,→ --NotebookApp.allow_origin='https://colab.research.google.com'
,→ --NotebookApp.port_retries=0 --NotebookApp.allow_root=True
,→ --NotebookApp.disable_check_xsrf=True --NotebookApp.token=''
,→ --NotebookApp.password='' --port=8888"
```

In Google Colab: Connect-> Connect to local runtime (enter port 8888)

Google Cloud connecting to Hosted Runtime

Install FEniCS through Ubuntu PPA

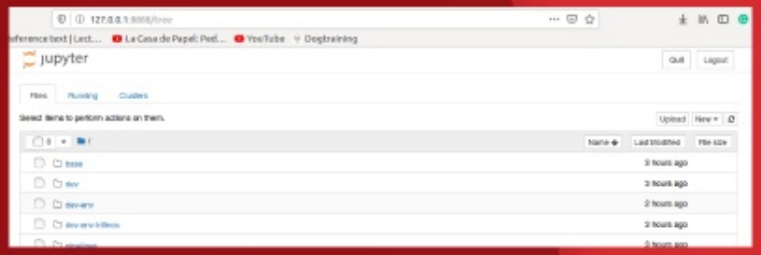
```
#@title
# Setup system environment and define utility functions and variables
from google.colab import files
try:
    from dolfin import *; from mshr import *
except ImportError as e:
    !apt-get install -y -qq software-properties-common python-software-properties
    !add-apt-repository -y ppa:fenics-packages/fenics
    !apt-get update -qq
    !sed -e 's:artful:bionic:' /etc/apt/sources.list.d/fenics-packages-ubuntu-feni
    !mv temp /etc/apt/sources.list.d/fenics-packages-ubuntu-fenics-artful.list
    !sed -e 's:artful:bionic:' /etc/apt/sources.list > temp
    !mv temp /etc/apt/sources.list
    !apt-get update -qq
    !apt install -y --no-install-recommends fenics
    from dolfin import *; from mshr import *

import matplotlib.pyplot as plt;
from IPython.display import clear_output, display, update_display; import time; im
import time
from ufl import replace
```

In Google Cloud: Connect -> Connect to Hosted Runtime

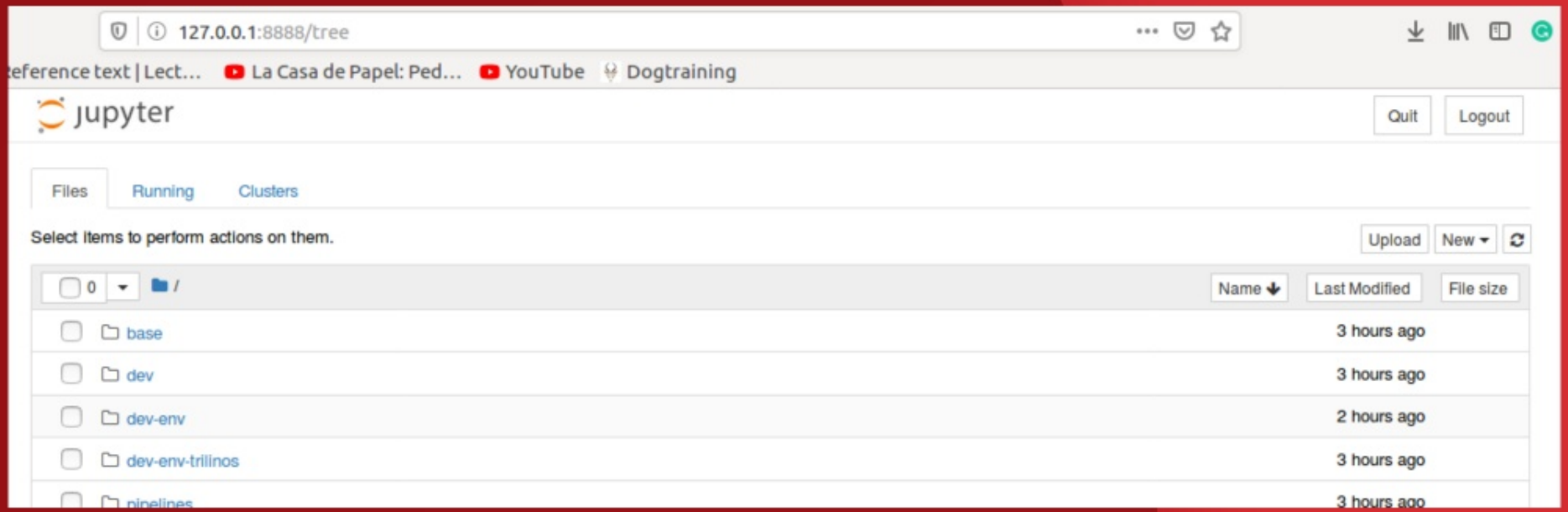
Docker & Jupyter notebooks

This command creates a Docker container locally from the latest stable FEniCS version at the time of writing, given with the fenics_tag variable. Inside this container, we run a Jupyter notebook from within the container on port 8888.

```
$ docker run -ti -p 127.0.0.1:8888:8888 -p
127.0.0.1:8000:8000 -v $(pwd):/home/fenics/shared -w /
home/fenics/shared quay.io/fenicsproject/stable:current
"jupyter-notebook --ip=0.0.0.0 --port=8888"
```




```
tdancheva@DULX0033:~/Projects/docker/dockerfiles$ docker run -ti -p 127.0.0.1:8888:8888 -p 127.0.0.1:8000:8000 -v $(pwd):/home/fenics/shared -w /home/fenics/shared quay.
io/fenicsproject/stable:current "jupyter-notebook --ip=0.0.0.0 --port=8888"
[I 14:09:39.468 NotebookApp] Writing notebook server cookie secret to /home/fenics/.local/share/jupyter/runtime/notebook_cookie_secret
[I 14:09:40.490 NotebookApp] Serving notebooks from local directory: /home/fenics/shared
[I 14:09:40.490 NotebookApp] The Jupyter Notebook is running at:
[I 14:09:40.490 NotebookApp] http://(3733099f5243 or 127.0.0.1):8888/?token=5b9515c3790dfe13d78d710c0ecc5cfdad592ba606af0cb7
[I 14:09:40.490 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 14:09:40.490 NotebookApp] No web browser found: could not locate runnable browser.
[C 14:09:40.490 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(3733099f5243 or 127.0.0.1):8888/?token=5b9515c3790dfe13d78d710c0ecc5cfdad592ba606af0cb7
[I 14:10:00.248 NotebookApp] 302 GET /?token=5b9515c3790dfe13d78d710c0ecc5cfdad592ba606af0cb7 (172.17.0.1) 0.78ms
```


127.0.0.1:8888/tree
eference text | Lect...
La Casa de Papel: Ped...
YouTube
Dogtraining
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/
Name
Last Modified
File size
base
3 hours ago
dev
3 hours ago
dev-env
2 hours ago
dev-env-trilinos
3 hours ago
pipelines
3 hours ago

Start with the following recipe:

https://bitbucket.org/fenics-project/docker/src/master/dockerfiles/dev-env/Dockerfile

```
# Environment variables
ENV PETSC_VERSION=3.11.1 \
    SLEPC_VERSION=3.11.1 \
    PYBIND11_VERSION=2.2.4 \
    MPI4PY_VERSION=3.0.1 \
    PETSC4PY_VERSION=3.11.0 \
    SLEPC4PY_VERSION=3.11.0 \
    TRILINOS_VERSION=12.10.1 \
    OPENBLAS_NUM_THREADS=1 \
    OPENBLAS_VERBOSE=0 \
    FENICS_PREFIX=$FENICS_HOME/local
```

Final steps, install dolfin and mshr

https://bitbucket.org/fenics-project/docker/src/master/dockerfiles/stable/Dockerfile

```
ENV FENICS_PYTHON=python3
ENV DOLFIN_VERSION="2019.1.0.post0"
ENV MSHR_VERSION="2019.1.0"
ENV PYPI_FENICS_VERSION=">=2019.1.0,<2019.2.0"
```

Installing from source

Note: Compiler
Support for C++11 standard

List of resources

FEniCS
Docker
Singularity

- FEniCS documentation: (Python), (C++)
- FEniCS book: Automated Solution of Differential Equations by the Finite Element Method
- FEniCS tutorial

https://github.com/hplgit/fenics-tutorial/tree/master/src

- FEniCS demos
- Slack channel

https://fenicsproject-slack-invite.herokuapp.com/

- FEniCS forum (archived, can view old questions): https://fenicsproject.org/qa/
- FEniCS forum new https://fenicsproject.discourse.group/
- Bitbucket issues

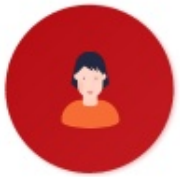
https://bitbucket.org/fenics-project/dolfin/issues?status=new&status=open

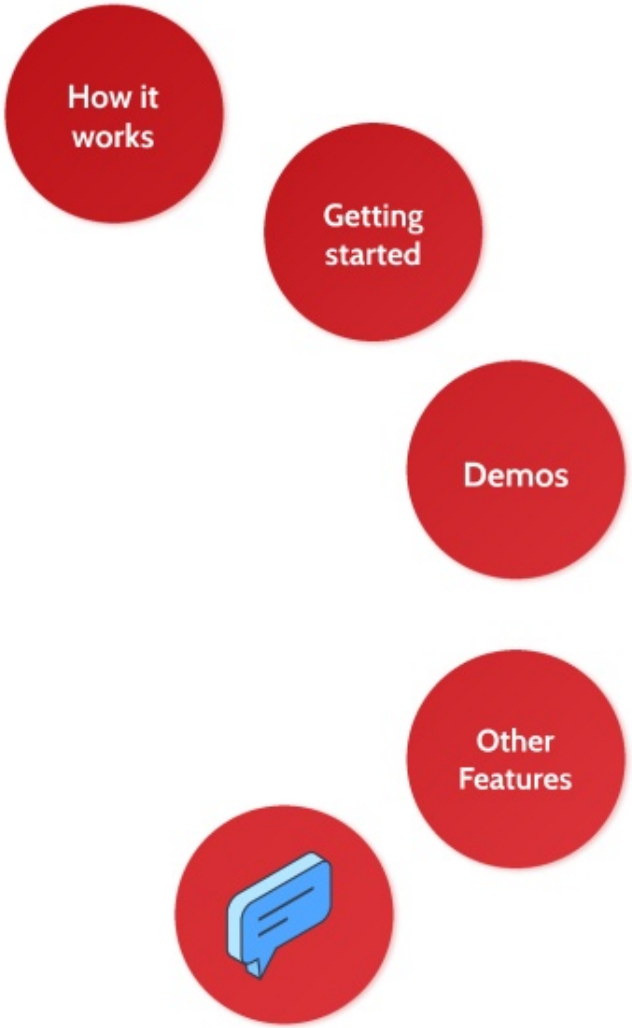
- edX course on "High Performance Finite Element Modeling"
- Docker documentation
- Docker tutorials
- Singularity documentation
- Singularity tutorials

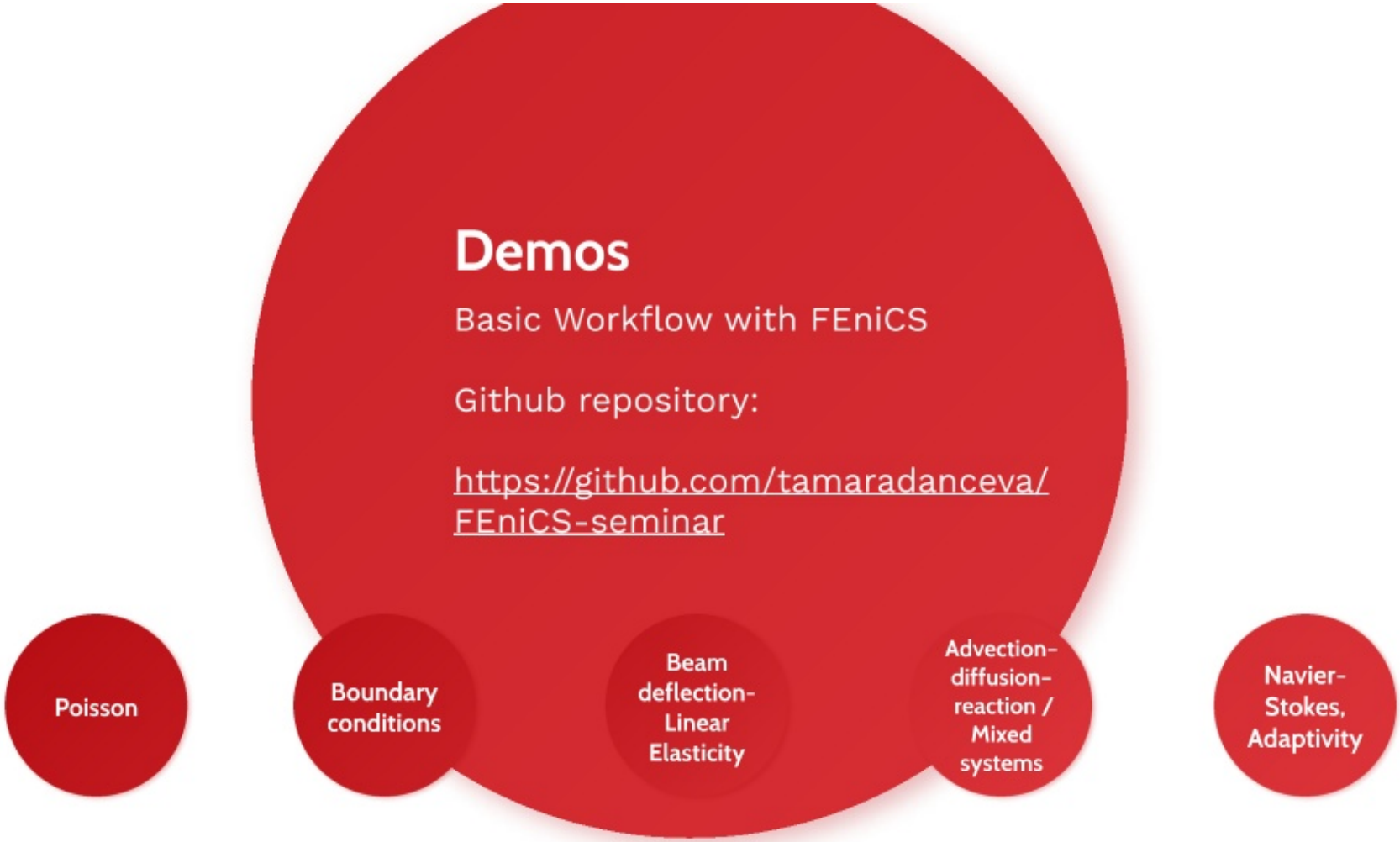
Be careful to run the examples with the right version!

FEniCS Platform

Introduction

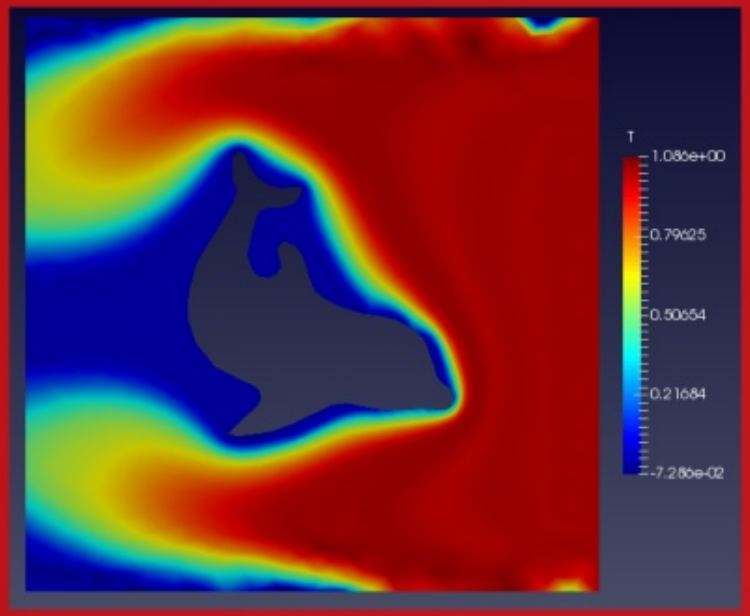





Demos
Basic Workflow with FEniCS
Github repository:
https://github.com/tamaradanceva/FEniCS-seminar
Poisson
Boundary conditions
Beam deflection- Linear Elasticity
Advection-diffusion-reaction / Mixed systems
Navier-Stokes, Adaptivity

Advection, Heat equation

https://colab.research.google.com/drive/1dfhPA_iowk_OEUMGJPsJLpkSa_Jz9Mke



Mesh/ Subdomains

Variational form

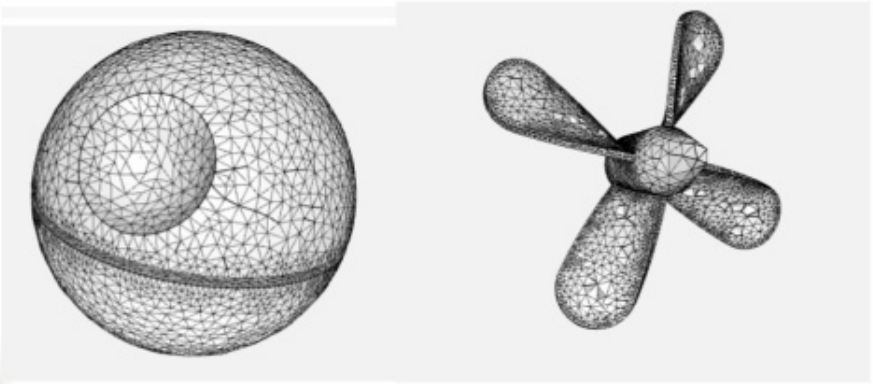
Solution & visualization

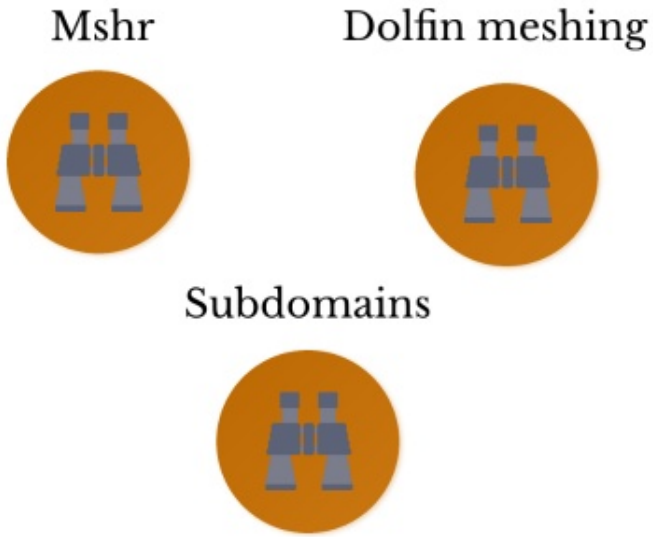
Nonlinear Poisson

Running

Mshr

Simplicial mesh generation component of FEniCS from geometries described by Constructive Solid Geometry (CSG, "Boolean operators") or from surface files, utilizing CGAL and Tetgen as mesh generation backends

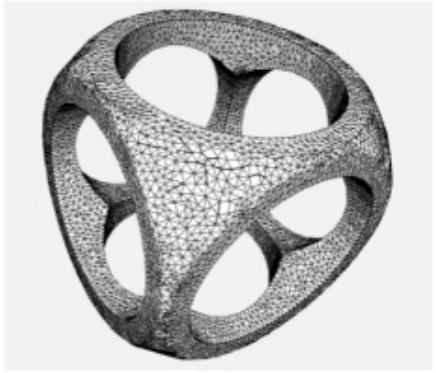


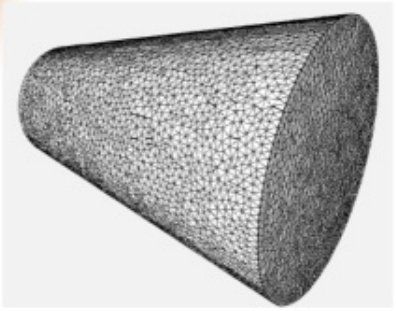


https://bitbucket.org/fenics-project/mshr/src/master/demo/

Generating geometry and mesh using Mshr directly

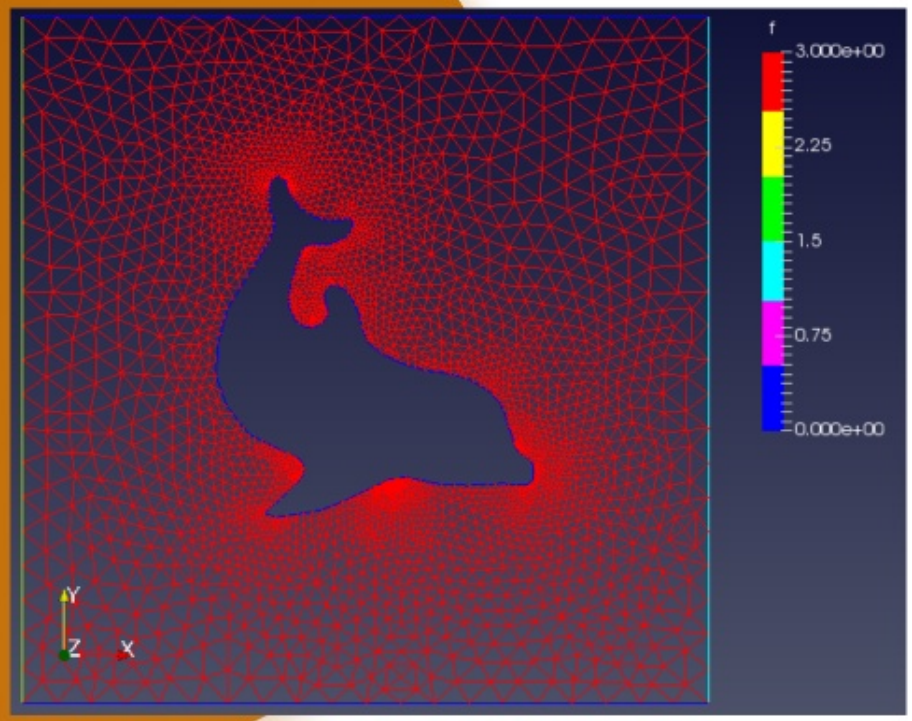
```
s1 = Sphere(Point(0, 0, 0), 1.4)
b1 = Box(Point(-1, -1, -1), Point(1, 1, 1))
c1 = Cylinder(Point(-2, 0, 0), Point(2, 0, 0), 0.8, 0.8)
c2 = Cylinder(Point(0, -2, 0), Point(0, 2, 0), 0.8, 0.8)
c3 = Cylinder(Point(0, 0, -2), Point(0, 0, 2), 0.8, 0.8)
geometry = s1*b1 - (c1 + c2 + c3)

# Create mesh
mesh = generate_mesh(geometry, 64)

c4 = Cylinder(Point(-2,0,0),Point(2,0,0),0.8,1.5)
mesh_cyl = generate_mesh(geometry, 64)
```





Subdomains

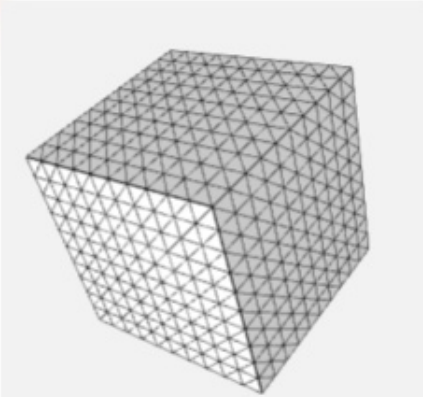
```
class Noslip(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary

# Sub domain for inflow (right)
class Inflow(SubDomain):
    def inside(self, x, on_boundary):
        return x[0] > 1.0 - DOLFIN_EPS and on_boundary

# Sub domain for outflow (left)
class Outflow(SubDomain):
    def inside(self, x, on_boundary):
        return x[0] < DOLFIN_EPS and on_boundary

mesh = Mesh("../dolfin_fine.xml.gz")


# Create mesh functions over the cell facets
sub_domains = MeshFunction("size_t", mesh,
                           mesh.topology().dim() - 1)

noslip = Noslip()
noslip.mark(sub_domains, 0)
inflow = Inflow()
inflow.mark(sub_domains, 1)
outflow = Outflow()
outflow.mark(sub_domains, 2)
```



UnitIntervalMesh
UnitSquareMesh
RectangleMesh
UnitCubeMesh
BoxMesh

Meshing in dolfin

```
mesh_interval = UnitIntervalMesh(20)
mesh = UnitSquareMesh(10, 10)
mesh = UnitSquareMesh(10, 10, "right/left")
mesh = UnitSquareMesh(10, 10, "crossed")
plot(mesh, title="Unit square (crossed)")

mesh = RectangleMesh(Point(-3.0,2.0),
Point(7.0,6.0), 10, 10, "right/left")

mesh = UnitCubeMesh(10, 10, 10)

mesh = BoxMesh(Point(0.0, 0.0, 0.0),
Point(10.0,4.0,2.0), 10, 10, 10)
```



Variational form

SUPG

Plugging in velocity profile obtained by solving the Stokes problem on the domain

Find $T(\boldsymbol{x},t) \in V$, $V = \{T \in H^1(\Omega); T = T_D \text{ on } \partial\Omega\}$ such that:

$$\begin{cases} \frac{\partial T(\boldsymbol{x},t)}{\partial t} + U \cdot \nabla T - \nabla \cdot (c \nabla T) = f \\ T = T_D \text{ on } \Gamma_D \end{cases}$$

, $\forall$ v $\in \hat{V}$, $\hat{V} = \{v \in H^1(\Omega); v = 0 \text{ on } \partial\Omega\}$

$$\int_\Omega [\frac{\partial T(\boldsymbol{x},t)}{\partial t} + U \cdot \nabla T - \nabla \cdot (c \nabla T)] v \, dx = \int_\Omega f v \, dx$$

Using $\int_\Omega (\nabla^2 T) v \, dx = \int_\Omega \nabla T \cdot \nabla v \, dx - \int_{\partial\Omega} \frac{\partial T}{\partial n} v \, ds$ results in:

$$\int_\Omega \frac{\partial T(\boldsymbol{x},t)}{\partial t} v \, dx + \int_\Omega (U \cdot \nabla T) v \, dx - c \int_\Omega \nabla T \cdot \nabla v \, dx = \int_\Omega f v \, dx$$

(Streamline-Upwind Petrov–Galerkin FEM) SUPG introduces artificial diffusion in the streamline direction

$$T = 0.5(T_n + T_{n+1})$$

$$r = T_n - T_{n-1} + dt * (U \cdot \nabla \bar{T} - c * \boldsymbol{\nabla} \cdot \nabla \bar{T} - f)$$

$$\|U\|_2 = \sqrt{U \cdot U}$$

$$\int_\Omega \frac{T_n - T_{n-1}}{dt} v \, dx + \int_\Omega (U \cdot \nabla \bar{T}) v \, dx - c \int_\Omega \nabla \bar{T} \cdot \nabla v \, dx = \int_\Omega f v \, dx \qquad \frac{h}{2.0 * \|U\|_2} * (U \cdot \nabla v) * r * dx$$

```
# Create FunctionSpaces
Q = FunctionSpace(mesh, "CG", 1)
V = VectorFunctionSpace(mesh, "CG", 2)

# Create velocity Function from file
velocity = Function(V);
File("dolfin_fine_velocity.xml.gz") >> velocity
out_vel = File("velocity.pvd")
out_vel << velocity

# Initialise source function and previous solution function
f  = Constant(0.0)
u0 = Function(Q)

# Parameters
T = 5.0
dt = 0.1
t = dt
c = 0.00005

# Test and trial functions
u, v = TrialFunction(Q), TestFunction(Q)

# Mid-point solution
u_mid = 0.5*(u0 + u)

# Residual
r = u - u0 + dt*(dot(velocity, grad(u_mid)) - c*div(grad(u_mid)) - f)

# Galerkin variational problem
F = v*(u - u0)*dx + dt*(v*dot(velocity, grad(u_mid))*dx \
                      + c*dot(grad(v), grad(u_mid))*dx)

# Add SUPG stabilisation terms
vnorm = sqrt(dot(velocity, velocity))
F += (h/(2.0*vnorm))*dot(velocity, grad(v))*r*dx
```

Solution & Visualization

VtkPlotter
https://github.com/marcomusy/vtkplotter

I) Assemble RHS, LHS, apply boundary conditions and solve

```
a = lhs(F)
#print("LHS", repr(a))
#print("LHS", ufl.algorithms.tree_format(a))
L = rhs(F)
#print("RHS:", repr(L))
#print("RHS:", ufl.algorithms.tree_format(L))
```

solve(a == L, u, bc)

II)

```
# Assemble matrix
A = assemble(a)
bc.apply(A)

# Create linear solver and factorize matrix
solver = LUSolver(A)
```

```
while t - T < DOLFIN_EPS:
    # Assemble vector and apply boundary conditions
    b = assemble(L)
    bc.apply(b)

    # Solve the linear system (re-use the already factorized matrix A)
    solver.solve(u.vector(), b)
```

Note: Best to use ParaView or VtkPlotter for more advanced visualization. Examples of internal plotting abilities shown throughout the demos.

Non-linear Poisson

Newton's method

$$F(u;v) = 0 \quad \forall v \in \hat{V},$$

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

$$V = \{v \in H^1(\Omega) : v = u_\mathrm{D} \text{ on } \partial\Omega\},$$
$$\hat{V} = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}.$$

```
def q(u):
  return 1 + u**2

mesh = UnitSquareMesh(8, 8)
V = FunctionSpace(mesh, 'P', 1)
u_D = Expression(u_code, degree=1)

def boundary(x, on_boundary):
  return on_boundary
bc = DirichletBC(V, u_D, boundary)

u = Function(V)
v = TestFunction(V)
f = Expression(f_code, degree=1)
F = q(u)*dot(grad(u), grad(v))*dx - f*v*dx

solve(F == 0, u, bc)
```

Running with FEniCS

Serial, Parallel

Serial execution

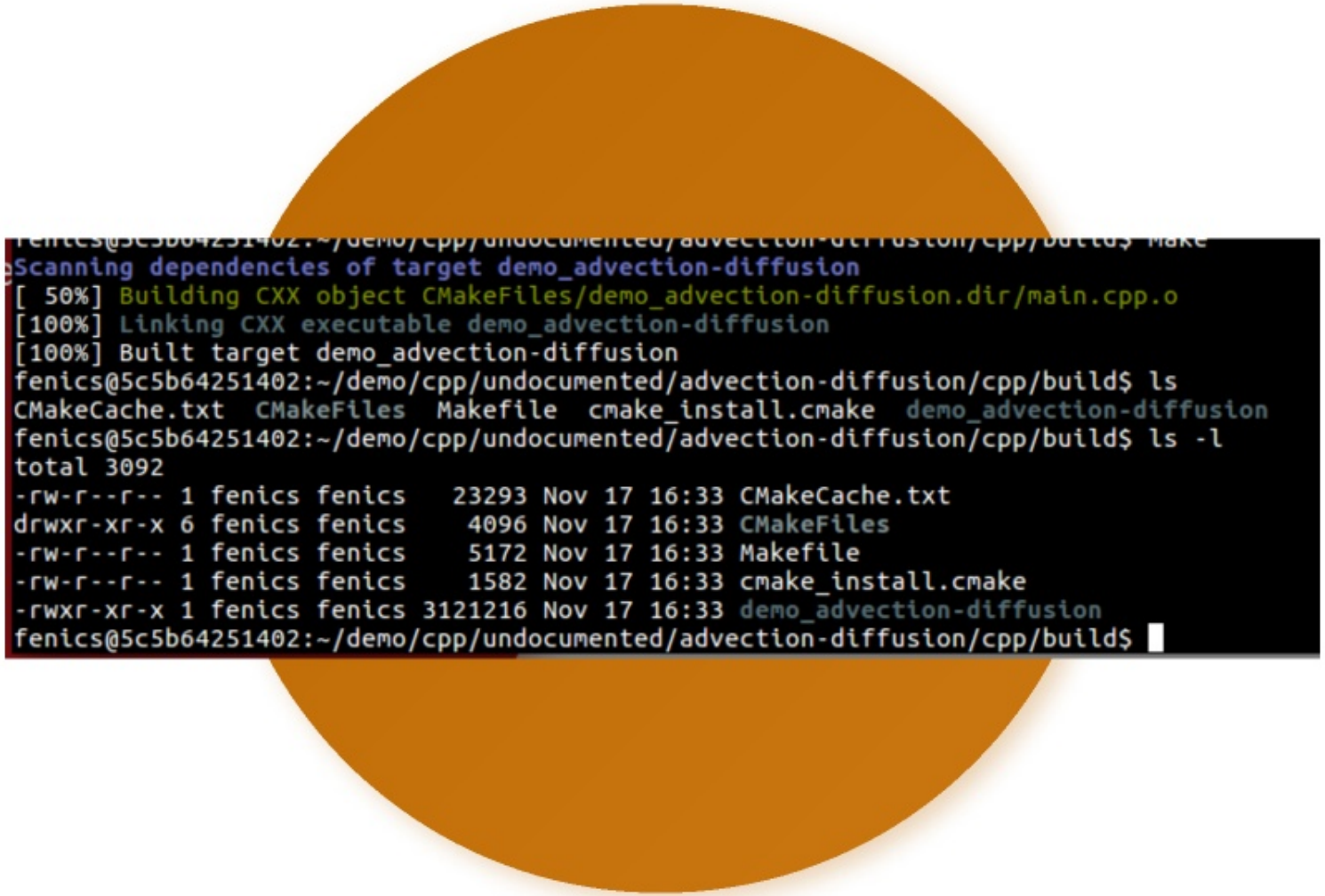
```
$ python3 advection-diffusion.py

 # Make file, see below
 $ g++ ./demo_advection-diffusion
```

Parallel Python

```
$ mpirun -np 4 python3 advection-diffusion.py
```

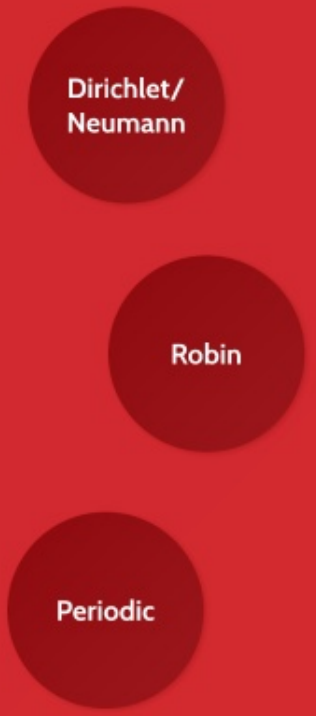
Parallel C++

```
$ mkdir build
$ cd build
$ cmake ..
$ make -j 2
$ cp demo_advection-diffusion ../.
$ cd ..
$ mpirun -np 2 ./demo_advection-diffusion
```


Scanning dependencies of target demo_advection-diffusion
[50%] Building CXX object CMakeFiles/demo_advection-diffusion.dir/main.cpp.o
[100%] Linking CXX executable demo_advection-diffusion
[100%] Built target demo_advection-diffusion
fenics@5c5b64251402:~/demo/cpp/undocumented/advection-diffusion/cpp/build$ ls
CMakeCache.txt CMakeFiles Makefile cmake_install.cmake demo_advection-diffusion
fenics@5c5b64251402:~/demo/cpp/undocumented/advection-diffusion/cpp/build$ ls -l
total 3092
-rw-r--r-- 1 fenics fenics 23293 Nov 17 16:33 CMakeCache.txt
drwxr-xr-x 6 fenics fenics 4096 Nov 17 16:33 CMakeFiles
-rw-r--r-- 1 fenics fenics 5172 Nov 17 16:33 Makefile
-rw-r--r-- 1 fenics fenics 1582 Nov 17 16:33 cmake_install.cmake
-rwxr-xr-x 1 fenics fenics 3121216 Nov 17 16:33 demo_advection-diffusion
fenics@5c5b64251402:~/demo/cpp/undocumented/advection-diffusion/cpp/build$

Boundary Conditions

https://colab.research.google.com/drive/1fuLHMbmXxqxgMO9ueCXnMo4BhXAl5V9v



https://fenicsproject.org/docs/dolfin/2018.1.0/python/demos/subdomains-poisson/documentation.html

Mixed Dirichlet/Neumann Computational Sub-domains/ Interior domains

For illustration purposes, we consider a weighted Poisson equation over a unit square: $\Omega = [0, 1] \times [0, 1]$ with mixed boundary conditions: find u satisfying

$$
\begin{aligned}
-\mathrm{div}(a\nabla u) &= 1.0 \quad \text{in } \Omega, \\
u &= 5.0 \quad \text{on } \Gamma_T, \\
u &= 0.0 \quad \text{on } \Gamma_B, \\
\nabla u \cdot n &= -10.0\, e^{-(y-0.5)^2} \quad \text{on } \Gamma_L. \\
\nabla u \cdot n &= 1.0 \quad \text{on } \Gamma_R,
\end{aligned}
$$

where $\Gamma_T, \Gamma_B, \Gamma_L, \Gamma_R$ denote the top, bottom, left and right sides of the unit square, respectively. The coefficient a may vary over the domain: here, we let $a = a_1 = 0.01$ for $(x, y) \in \Omega_1 = [0.5, 0.7] \times [0.2, 1.0]$ and $a = a_0 = 1.0$ in $\Omega_0 = \Omega \backslash \Omega_1$. We can think of Ω_1 as an obstacle with differing material properties from the rest of the domain.

Variational form

Sub-domains/ Interior domains

Form implementation

Variational form

Poisson with mixed boundary conditions

$$-\frac{\partial u}{\partial n} = g \quad \text{on } \Gamma_{\mathrm{N}}.$$

We can write the above boundary value problem in the standard linear variational form: find $u \in V$ such that

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

where V and $\hat{V}$ are suitable function spaces incorporating the Dirichlet boundary conditions on Γ_T and Γ_B, and

$$a(u, v) = \int_{\Omega_0} a_0 \nabla u \cdot \nabla v \, \mathrm{d}x + \int_{\Omega_1} a_1 \nabla u \cdot \nabla v \, \mathrm{d}x,$$
$$L(v) = \int_{\Gamma_L} g_L v \, \mathrm{d}s + \int_{\Gamma_R} g_R v \, \mathrm{d}s + \int_{\Omega} f v \, \mathrm{d}x.$$

where $f = 1.0$, $g_L = -10.0e^{-(y-0.5)^2}$ and $g_R = 1.0$.

Sub-domains & Interior domains

```
class Left(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[0], 0.0)

class Right(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[0], 1.0)

class Bottom(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[1], 0.0)

class Top(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[1], 1.0)

class Obstacle(SubDomain):
    def inside(self, x, on_boundary):
        return (between(x[1], (0.5, 0.7)) and between(x[0], (0.2, 1.0)))
```

```
# Initialize mesh function for interior domains
domains = MeshFunction("size_t", mesh, mesh.topology().dim())
domains.set_all(0)
obstacle.mark(domains, 1)
```

```
# Initialize mesh function for boundary domains
boundaries = MeshFunction("size_t", mesh, mesh.topology().dim()-1)
boundaries.set_all(0)
left.mark(boundaries, 1)
top.mark(boundaries, 2)
right.mark(boundaries, 3)
bottom.mark(boundaries, 4)
```

```
# Define new measures associated with the interior domains and
# exterior boundaries
dx = Measure('dx', domain=mesh, subdomain_data=domains)
ds = Measure('ds', domain=mesh, subdomain_data=boundaries)
```

Formulation

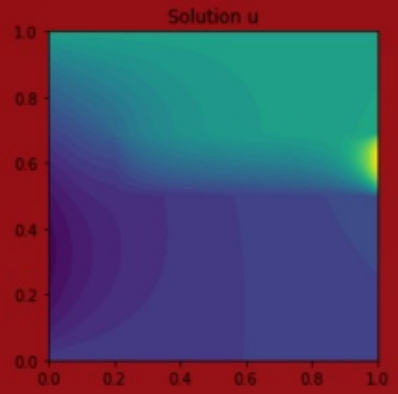
Output quantities

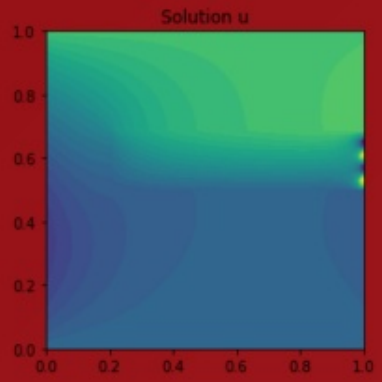
```
# Define variational form
F = (inner(a0*grad(u), grad(v))*dx(0) + inner(a1*grad(u), grad(v))*dx(1)
     - g_L*v*ds(1) - g_R*v*ds(3)
     - f*v*dx(0) - f*v*dx(1))

# Separate left and right hand sides of equation
a, L = lhs(F), rhs(F)

# Solve problem
u = Function(V)
solve(a == L, u, bcs)
```

```
# Evaluate integral of normal gradient over top boundary
n = FacetNormal(mesh)
m1 = dot(grad(u), n)*ds(2)
v1 = assemble(m1)
print "\int grad(u) * n ds(2) = ", v1

# Evaluate integral of u over the obstacle
m2 = u*dx(1)
v2 = assemble(m2)
print "\int u dx(1) = ", v2
```

Robin boundary condition
Mixed

$$-\kappa\frac{\partial u}{\partial n} = r(u-s),$$





Variational form

Form Implementation

Mixed Neumann/Robin boundary conditions

$$
\begin{aligned}
-\int_{\partial\Omega} \kappa \frac{\partial u}{\partial n} v \,\mathrm{d}s &= -\sum_i \int_{\Gamma_{\mathrm{N}}^i} \kappa \frac{\partial u}{\partial n} \,\mathrm{d}s - \sum_i \int_{\Gamma_{\mathrm{R}}^i} \kappa \frac{\partial u}{\partial n} \,\mathrm{d}s \\
&= \sum_i \int_{\Gamma_{\mathrm{N}}^i} g_i \,\mathrm{d}s + \sum_i \int_{\Gamma_{\mathrm{R}}^i} r_i (u - s_i) \,\mathrm{d}s .
\end{aligned}
$$

$$
F = \int_\Omega \kappa \nabla u \cdot \nabla v \,\mathrm{d}x + \sum_i \int_{\Gamma_{\mathrm{N}}^i} g_i v \,\mathrm{d}s + \sum_i \int_{\Gamma_{\mathrm{R}}^i} r_i (u - s_i) v \,\mathrm{d}s - \int_\Omega f v \,\mathrm{d}x = 0 .
$$

Finally yielding,

$$
\begin{aligned}
a(u,v) &= \int_\Omega \kappa \nabla u \cdot \nabla v \,\mathrm{d}x + \sum_i \int_{\Gamma_{\mathrm{R}}^i} r_i u v \,\mathrm{d}s , \\
L(v) &= \int_\Omega f v \,\mathrm{d}x - \sum_i \int_{\Gamma_{\mathrm{N}}^i} g_i v \,\mathrm{d}s + \sum_i \int_{\Gamma_{\mathrm{R}}^i} r_i s_i v \,\mathrm{d}s .
\end{aligned}
$$

Mixed Neumann/Robin boundary conditions

$$
\begin{aligned}
-\int_{\partial\Omega} \kappa \frac{\partial u}{\partial n} v \,\mathrm{d}s &= -\sum_i \int_{\Gamma_{\mathrm{N}}^i} \kappa \frac{\partial u}{\partial n} \,\mathrm{d}s - \sum_i \int_{\Gamma_{\mathrm{R}}^i} \kappa \frac{\partial u}{\partial n} \,\mathrm{d}s \\
&= \sum_i \int_{\Gamma_{\mathrm{N}}^i} g_i \,\mathrm{d}s + \sum_i \int_{\Gamma_{\mathrm{R}}^i} r_i (u - s_i) \,\mathrm{d}s .
\end{aligned}
$$

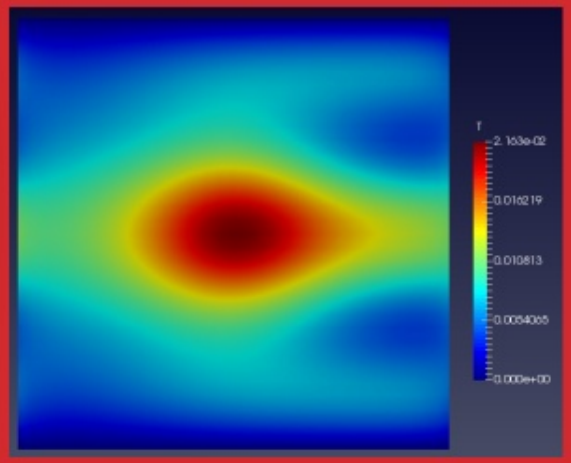
$$
F = \int_\Omega \kappa \nabla u \cdot \nabla v \,\mathrm{d}x + \sum_i \int_{\Gamma_{\mathrm{N}}^i} g_i v \,\mathrm{d}s + \sum_i \int_{\Gamma_{\mathrm{R}}^i} r_i (u - s_i) v \,\mathrm{d}s - \int_\Omega f v \,\mathrm{d}x = 0 .
$$

Finally yielding,

$$
\begin{aligned}
a(u, v) &= \int_\Omega \kappa \nabla u \cdot \nabla v \,\mathrm{d}x + \sum_i \int_{\Gamma_{\mathrm{R}}^i} r_i u v \,\mathrm{d}s , \\
L(v) &= \int_\Omega f v \,\mathrm{d}x - \sum_i \int_{\Gamma_{\mathrm{N}}^i} g_i v \,\mathrm{d}s + \sum_i \int_{\Gamma_{\mathrm{R}}^i} r_i s_i v \,\mathrm{d}s .
\end{aligned}
$$

Periodic boundary conditions

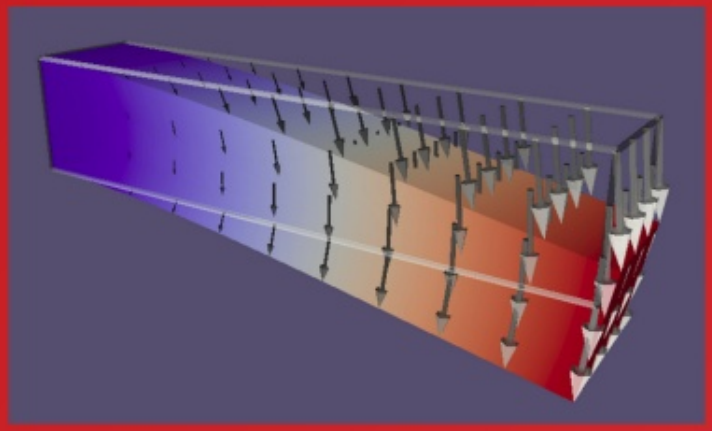
```python
# Sub domain for Periodic boundary condition
class PeriodicBoundary(SubDomain):

    # Left boundary is "target domain" G
    def inside(self, x, on_boundary):
        return bool(x[0] < DOLFIN_EPS and x[0] > -DOLFIN_EPS and on_boundary)

    # Map right boundary (H) to left boundary (G)
    def map(self, x, y):
        y[0] = x[0] - 1.0
        y[1] = x[1]
```



Deflected beam

Linear Elasticity

https://colab.research.google.com/drive/1hCYOEQQQa6cvS5AAt84K_bVemOeKy8zs



Mesh import

Problem

Form Implementation

Import of meshes

There are many workflows to go from Salome to a mesh acceptable for FEniCS, here are just a couple of them:

- The most straightforward one being, export from Salome as .med, use Mesh IO (https://github.com/nschloe/meshio, open source to convert meshes, very easy to install and use) to convert from .med to .xml or .hdmf

- Export from Salome as .inp, import into gmsh, export as msh, use Mesh IO or dolfin-convert (a tool script part of Dolfin) to convert to .xml

Warning: VTK format (PVD) and XDMF will continue to be supported. SVG, XML, X3D, and RAW will no longer be supported.

Static Problem
Variational form
Hooke's law for isotropic materials

$$-\int_\Omega (\nabla \cdot \sigma) \cdot v \, dx = \int_\Omega f \cdot v \, dx .$$

$$-\int_\Omega (\nabla \cdot \sigma) \cdot v \, dx = \int_\Omega \sigma : \nabla v \, dx - \int_{\partial\Omega} (\sigma \cdot n) \cdot v \, ds,$$

$$\begin{aligned} -\nabla \cdot \sigma &= f \text{ in } \Omega, \\ \sigma &= \lambda \operatorname{tr}(\varepsilon) I + 2\mu\varepsilon, \\ \varepsilon &= \frac{1}{2}\left(\nabla u + (\nabla u)^\top\right), \end{aligned}$$

resulting in

$$\begin{aligned} a(u, v) &= L(v) \quad \forall v \in \hat{V}, \\ a(u, v) &= \int_\Omega \sigma(u) : \nabla v \, dx, \\ \sigma(u) &= \lambda(\nabla \cdot u) I + \mu(\nabla u + (\nabla u)^\top), \\ L(v) &= \int_\Omega f \cdot v \, dx + \int_{\partial\Omega_T} T \cdot v \, ds . \end{aligned}$$

$$a(u, v) = \int_\Omega \sigma(u) : \varepsilon(v) \, dx, \quad \varepsilon(v) = \frac{1}{2}\left(\nabla v + (\nabla v)^\top\right)$$

Implementation

`grad(u)` vs. `nabla_grad(u)`. For scalar functions, ∇u has a clear meaning as the vector

$$\nabla u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}\right).$$

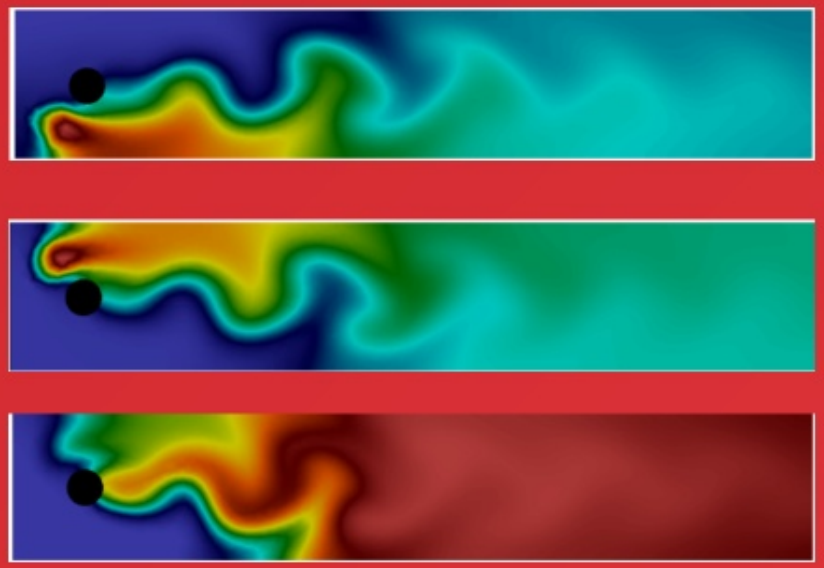
However, if u is vector-valued, the meaning is less clear. Some sources define ∇u as the matrix with elements $\partial u_j/\partial x_i$, while other sources prefer $\partial u_i/\partial x_j$. In FEniCS, `grad(u)` is defined as the matrix with elements $\partial u_i/\partial x_j$, which is the natural definition of ∇u if we think of this as the *gradient* or *derivative* of u. This way, the matrix ∇u can be applied to a differential dx to give an increment $du = \nabla u\ dx$. Since the alternative interpretation of ∇u as the matrix with elements $\partial u_j/\partial x_i$ is very common, in particular in continuum mechanics, FEniCS provides the operator `nabla_grad` for this purpose. For the Navier–Stokes equations, it is important to consider the term $u \cdot \nabla u$ which should be interpreted as the vector w with elements $w_i = \sum_j \left(u_j \frac{\partial}{\partial x_j}\right) u_i = \sum_j u_j \frac{\partial u_i}{\partial x_j}$. This term can be implemented in FEniCS either as `grad(u)*u`, since this is expression becomes $\sum_j \partial u_i/\partial x_j u_j$, or as `dot(u, nabla_grad(u))` since this expression becomes $\sum_i u_i \partial u_j/\partial x_i$. We will use the notation `dot(u, nabla_grad(u))` below since it corresponds more closely to the standard notation $u \cdot \nabla u$.

```
14 # Create mesh and define function space
15 mesh = BoxMesh(Point(0, 0, 0), Point(L, W, W), 10, 3, 3)
16 V = VectorFunctionSpace(mesh, 'P', 1)
17
18 # Define boundary condition
19 tol = 1E-14
20
21 def clamped_boundary(x, on_boundary):
22     return on_boundary and x[0] < tol
23
24 bc = DirichletBC(V, Constant((0, 0, 0)), clamped_boundary)
25
26 # Define strain and stress
27
28 def epsilon(u):
29     return 0.5*(nabla_grad(u) + nabla_grad(u).T)
30     #return sym(nabla_grad(u))
31
32 def sigma(u):
33     return lambda_*nabla_div(u)*Identity(d) + 2*mu*epsilon(u)
34
35 # Define variational problem
36 u = TrialFunction(V)
37 d = u.geometric_dimension()  # space dimension
38 v = TestFunction(V)
39 f = Constant((0, 0, -rho*g))
40 T = Constant((0, 0, 0))
41 a = inner(sigma(u), epsilon(v))*dx
42 L = dot(f, v)*dx + dot(T, v)*ds
43
44 # Compute solution
45 u = Function(V)
46 solve(a == L, u, bc)
```

Advection-diffusion-reaction equations

Mixed systems

https://colab.research.google.com/drive/1H7JIJ2TMr4-jcJcWW_bALXuFne5RTkSU



Problem

Variational form, Mixed elements

Chemical reaction between two species A and B

$$A + B \to C.$$

$$\frac{\mathrm{d}}{\mathrm{d}t}[C] = K[A][B].$$

the concentrations of the three species:

$$u_1 = [A], \quad u_2 = [B], \quad u_3 = [C].$$

$$\varrho\left(\frac{\partial w}{\partial t} + w \cdot \nabla w\right) = \nabla \cdot \sigma(w,p) + f,$$

$$\nabla \cdot w = 0,$$

$$\frac{\partial u_1}{\partial t} + w \cdot \nabla u_1 - \nabla \cdot (\epsilon \nabla u_1) = f_1 - K u_1 u_2,$$

$$\frac{\partial u_2}{\partial t} + w \cdot \nabla u_2 - \nabla \cdot (\epsilon \nabla u_2) = f_2 - K u_1 u_2,$$

$$\frac{\partial u_3}{\partial t} + w \cdot \nabla u_3 - \nabla \cdot (\epsilon \nabla u_3) = f_3 + K u_1 u_2 - K u_3.$$

Variational form

Mixed elements

```
31 # Define function space for velocity
32 W = VectorFunctionSpace(mesh, 'P', 2)
33
34 # Define function space for system of concentrations
35 P1 = FiniteElement('P', triangle, 1)
36 element = MixedElement([P1, P1, P1])
37 V = FunctionSpace(mesh, element)
38
39 # Define test functions
40 v_1, v_2, v_3 = TestFunctions(V)
41
42 # Define functions for velocity and concentrations
43 w = Function(W)
44 u = Function(V)
45 u_n = Function(V)
46
47 # Split system functions to access components
48 u_1, u_2, u_3 = split(u)
49 u_n1, u_n2, u_n3 = split(u_n)
```

```
51 # Define source terms
52 f_1 = Expression('pow(x[0]-0.1,2)+pow(x[1]-0.1,2)<0.05*0.05 ? 0.1 : 0',
53                  degree=1)
54 f_2 = Expression('pow(x[0]-0.1,2)+pow(x[1]-0.3,2)<0.05*0.05 ? 0.1 : 0',
55                  degree=1)
56 f_3 = Constant(0)
57
58 # Define expressions used in variational forms
59 k = Constant(dt)
60 K = Constant(K)
61 eps = Constant(eps)
62
63 # Define variational problem
64 F = ((u_1 - u_n1) / k)*v_1*dx + dot(w, grad(u_1))*v_1*dx \
65   + eps*dot(grad(u_1), grad(v_1))*dx + K*u_1*u_2*v_1*dx  \
66   + ((u_2 - u_n2) / k)*v_2*dx + dot(w, grad(u_2))*v_2*dx \
67   + eps*dot(grad(u_2), grad(v_2))*dx + K*u_1*u_2*v_2*dx  \
68   + ((u_3 - u_n3) / k)*v_3*dx + dot(w, grad(u_3))*v_3*dx \
69   + eps*dot(grad(u_3), grad(v_3))*dx - K*u_1*u_2*v_3*dx + K*u_3*v_3*dx \
70   - f_1*v_1*dx - f_2*v_2*dx - f_3*v_3*dx
71
```

Incompressible Navier-Stokes, Adaptivity, Adjoint problems

I) Chorin's method

II) Space-time cG(1)cG(1) FEM with GLS stabilization

https://colab.research.google.com/drive/1DxKMbISCoJq1MAg_lCvgPwUenOR_Apih

Problem definition

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \nu \Delta \mathbf{u} + \mathbf{g}$$
$$\nabla \cdot \mathbf{u} = 0.$$

$$\mathbf{u} = \mathbf{u_D} \quad on \quad \partial\Omega_D, \quad \text{(Dirichlet)}$$
$$\nu \frac{\partial \mathbf{u}}{\partial \mathbf{n}} - p \cdot \mathbf{n} = \mathbf{h} \quad on \quad \partial\Omega_N, \quad \text{(Neumann)}$$
$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u_0}.$$

$$\overbrace{\underbrace{\frac{\partial \mathbf{u}}{\partial t}}_{\text{Variation}} + \underbrace{(\mathbf{u} \cdot \nabla)\mathbf{u}}_{\text{Convection}}}^{\text{Inertia (per volume)}} = \overbrace{\underbrace{\nu \nabla^2 \mathbf{u}}_{\text{Diffusion}}}^{\text{Divergence of stress}} + \underbrace{-\nabla w}_{\text{Internal source}} + \underbrace{\mathbf{g}}_{\text{External source}}$$

$$\frac{1}{\rho_0} \nabla p = \nabla \left(\frac{p}{\rho_0} \right) \equiv \nabla w$$

1. $\nu \Delta \mathbf{u}$ — elliptic
2. $\frac{\partial \mathbf{u}}{\partial t} - \nu \Delta \mathbf{u}$ — parabolic
3. $\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}$ — nonlinear hyperbolic
4. $\nu \Delta \mathbf{u}$ — constrained problem (Stokes)
   $\nabla \cdot \mathbf{u} = 0$

Chorin's projection method

Operator splitting approach

$$\int_{\partial\Omega}\left(\mu\frac{\partial \boldsymbol{u}}{\partial \hat{\boldsymbol{n}}}-p\hat{\boldsymbol{n}}\right)\cdot\boldsymbol{v}=\underbrace{\int_{\Gamma_D}\left(\mu\frac{\partial \boldsymbol{u}}{\partial \hat{\boldsymbol{n}}}-p\hat{\boldsymbol{n}}\right)\cdot\boldsymbol{v}}_{\boldsymbol{v}=\boldsymbol{0}\text{ on }\Gamma_D}+\int_{\Gamma_N}\underbrace{\left(\mu\frac{\partial \boldsymbol{u}}{\partial \hat{\boldsymbol{n}}}-p\hat{\boldsymbol{n}}\right)}_{=\boldsymbol{h}\text{ on }\Gamma_N}\cdot\boldsymbol{v}=\int_{\Gamma_N}\boldsymbol{h}\cdot\boldsymbol{v}.$$

Having this in mind, the weak formulation of the Navier–Stokes equations is expressed as:[14]

find $\boldsymbol{u}\in L^2(\mathbb{R}^+\ [H^1(\Omega)]^d)\cap C^0(\mathbb{R}^+\ [L^2(\Omega)]^d)$ such that:

$$\begin{cases}\displaystyle\int_\Omega \rho\frac{\partial \boldsymbol{u}}{\partial t}\cdot\boldsymbol{v}+\int_\Omega \mu\nabla\boldsymbol{u}\cdot\nabla\boldsymbol{v}+\int_\Omega \rho(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\cdot\boldsymbol{v}-\int_\Omega p\nabla\cdot\boldsymbol{v}=\int_\Omega \boldsymbol{f}\cdot\boldsymbol{v}+\int_{\Gamma_N}\boldsymbol{h}\cdot\boldsymbol{v}. & \forall\boldsymbol{v}\in V\\ \displaystyle\int_\Omega q\nabla\cdot\boldsymbol{u}=0. & \forall q\in Q.\end{cases}$$

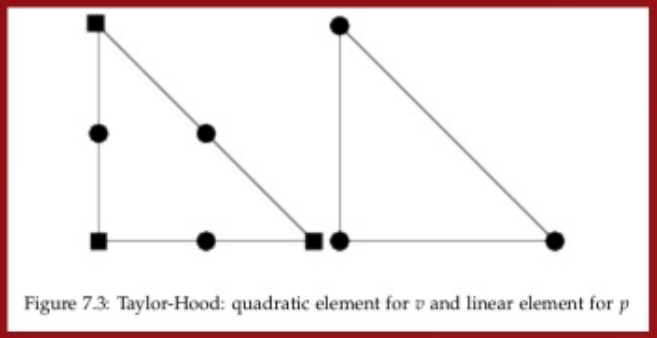


Figure 7.3: Taylor-Hood: quadratic element for v and linear element for p

In Chorin's original version of the projection method, one first computes an intermediate velocity, $\mathbf{u}^*$, explicitly using the momentum equation by ignoring the pressure gradient term:

$$(1)\qquad \frac{\mathbf{u}^*-\mathbf{u}^n}{\Delta t}=-(\mathbf{u}^n\cdot\nabla)\mathbf{u}^n+\nu\nabla^2\mathbf{u}^n$$

where $\mathbf{u}^n$ is the velocity at n^{th} time step. In the second half of the algorithm, the *projection* step, we correct the intermediate velocity to obtain the final solution of the time step $\mathbf{u}^{n+1}$:

$$(2)\qquad \mathbf{u}^{n+1}=\mathbf{u}^*-\frac{\Delta t}{\rho}\nabla p^{n+1}\qquad\qquad \nabla^2 p^{n+1}=\frac{\rho}{\Delta t}\nabla\cdot\mathbf{u}^*$$

One can rewrite this equation in the form of a time step as

$$\frac{\mathbf{u}^{n+1}-\mathbf{u}^*}{\Delta t}=-\frac{1}{\rho}\nabla p^{n+1}$$

$$\int_{\partial\Omega}\left(\mu\frac{\partial \boldsymbol{u}}{\partial \hat{\boldsymbol{n}}}-p\hat{\boldsymbol{n}}\right)\cdot\boldsymbol{v}=\underbrace{\int_{\Gamma_D}\left(\mu\frac{\partial \boldsymbol{u}}{\partial \hat{\boldsymbol{n}}}-p\hat{\boldsymbol{n}}\right)\cdot\boldsymbol{v}}_{\boldsymbol{v}=\boldsymbol{0}\text{ on }\Gamma_D}+\int_{\Gamma_N}\underbrace{\left(\mu\frac{\partial \boldsymbol{u}}{\partial \hat{\boldsymbol{n}}}-p\hat{\boldsymbol{n}}\right)}_{=\boldsymbol{h}\text{ on }\Gamma_N}\cdot\boldsymbol{v}=\int_{\Gamma_N}\boldsymbol{h}\cdot\boldsymbol{v}.$$

Having this in mind, the weak formulation of the Navier–Stokes equations is expressed as:[14]

find $\boldsymbol{u}\in L^2(\mathbb{R}^+\ [H^1(\Omega)]^d)\cap C^0(\mathbb{R}^+\ [L^2(\Omega)]^d)$ such that:

$$\begin{cases}\displaystyle\int_\Omega \rho\frac{\partial \boldsymbol{u}}{\partial t}\cdot\boldsymbol{v}+\int_\Omega \mu\nabla\boldsymbol{u}\cdot\nabla\boldsymbol{v}+\int_\Omega \rho(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\cdot\boldsymbol{v}-\int_\Omega p\nabla\cdot\boldsymbol{v}=\int_\Omega \boldsymbol{f}\cdot\boldsymbol{v}+\int_{\Gamma_N}\boldsymbol{h}\cdot\boldsymbol{v}. & \forall\boldsymbol{v}\in V\\ \displaystyle\int_\Omega q\nabla\cdot\boldsymbol{u}=0. & \forall q\in Q.\end{cases}$$

In Chorin's original version of the projection method, one first computes an intermediate velocity, $\mathbf{u}^*$, explicitly using the momentum equation by ignoring the pressure gradient term:

$$(1) \qquad \frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n + \nu \nabla^2 \mathbf{u}^n$$

where $\mathbf{u}^n$ is the velocity at n^{th} time step. In the second half of the algorithm, the *projection* step, we correct the intermediate velocity to obtain the final solution of the time step $\mathbf{u}^{n+1}$:

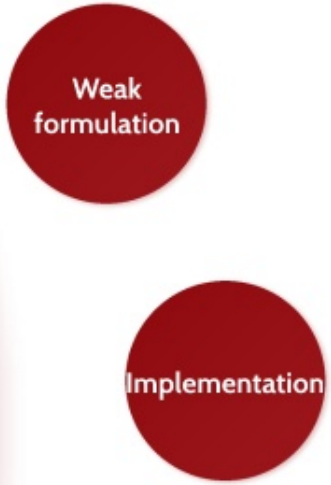
$$(2) \qquad \mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho} \nabla p^{n+1} \qquad\qquad \nabla^2 p^{n+1} = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^*$$

One can rewrite this equation in the form of a time step as

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\frac{1}{\rho} \nabla p^{n+1}$$

Space-time cG(1)cG(1) FEM with GLS stabilization

DOLFIN-HPC and UNICORN (FEniCS-HPC)



Weak formulation

MOOC-HPFEM: Navier-Stokes fluid flow Direct FEM Simulation (DFS)

Incompressible Navier-Stokes as model for low and high Reynolds number flow, such as blood flow and flight:

$$
\begin{aligned}
R(\hat{u}) \quad &:= \quad \begin{cases} p_t u + (u \cdot \nabla)u + \nabla p - \nu \Delta u = 0 \\ \nabla \cdot u = 0 \end{cases} \\
u \quad &:= \quad 0, x \in \Gamma \quad \text{(No-slip BC for low Reynolds number)} \\
\hat{u} \quad &= \quad (u, p), \quad r(\hat{u}, \hat{v}) = (R(\hat{u}), \hat{v}) \quad \text{(Weak residual)}
\end{aligned}
$$

Space-time cG(1)cG(1) FEM with GLS stabilization. Developed over 20+ years by Johnson, Hoffman, Jansson, etc.

$$
\begin{aligned}
& r(\hat{U}, \hat{v}) = (R(\hat{U}), \hat{v}) + (\delta R(\hat{U}), R(\hat{v})) = 0 \\
& \delta = h, \forall \hat{v} \in \hat{V}_h, \hat{U} \in \hat{V}_h
\end{aligned}
$$

Implementation of the primal problem

```
clear_output();
for i in range(0, maxiters):
  # Declare FE spaces
  VE = VectorElement("CG", mesh.ufl_cell(), 1); QE = FiniteElement("CG", mesh.ufl_cell(), 1); h = Cell
  WE = VE * QE; W = FunctionSpace(mesh, WE); V = FunctionSpace(mesh, VE); Q = FunctionSpace(mesh, QE)
  wt = TestFunction(W); (v, q) = (as_vector((wt[0], wt[1])), wt[2]);
  w = Function(W); (u, p) = (as_vector((w[0], w[1])), w[2]); u0 = Function(V)
  phi = Function(W); (phi_u, phi_p) = (as_vector((phi[0], phi[1])), phi[2]);
  w_ = TrialFunction(W)

  ZE = FiniteElement("DG", mesh.ufl_cell(), 0)
  Z = FunctionSpace(mesh, ZE)
  z = TestFunction(Z)

  uin = Expression(("4*(x[1]*(YMAX-x[1]))/(YMAX*YMAX)", "0."), YMAX=YMAX, element = V.ufl_element()) #
  om = Expression("x[0] > XMAX - eps ? 1. : 0.", XMAX=XMAX, eps=eps, element = Q.ufl_element()) # Mark
  im = Expression("x[0] < XMIN + eps ? 1. : 0.", XMIN=XMIN, eps=eps, element = Q.ufl_element())
  nm = Expression("x[0] > XMIN + eps && x[0] < XMAX - eps ? 1. : 0.", XMIN=XMIN, XMAX=XMAX, eps=eps, e
  psimarker = Expression("x[0] > XMIN + eps && x[1] >  YMIN + eps && x[0] < XMAX - eps && x[1] < YMAX
                         XMIN=XMIN, XMAX=XMAX, YMIN=YMIN, YMAX=YMAX, eps=eps, element = Q.ufl_element(

  n = FacetNormal(mesh)
  d = .5*h; # Stabilization parameter
  gamma = 10*1./h # Penalty parameter

  r = ((inner(grad(p) + grad(u)*u, v) + nu*inner(grad(u), grad(v)) + div(u)*q)*dx +
       gamma*(om*p*q + im*inner(u - uin, v) + nm*inner(u, v))*ds + # Weak boundary conditions
      d*(inner(grad(p) + grad(u)*u, grad(q) + grad(u)*v) + inner(div(u), div(v)))*dx) # Stabilization
  solve(r==0, w)  # Solve the Navier-Stokes PDE (stationary)
  plot_compact_static(u, Q, "Velocity") # Plot all quantities (see implementation above)
```

Implementation of the dual problem
Goal function

```
# Define goal functional (quantity of interest)
M = u*dx()
```

```
# Solve equation a = L with respect to u and the given boundary
# conditions, such that the estimated error (measured in M) is less
# than tol
problem = LinearVariationalProblem(a, L, u, bc)
solver = AdaptiveLinearVariationalSolver(problem, M)
solver.parameters["error_control"]["dual_variational_solver"]["linear_solver"] = "cg"
solver.solve(tol)

solver.summary()
```

```
a_dual = adjoint(derivative(r, w)) # Construct the adjoint problem
L_dual = M(mesh, v, q)
solve(a_dual==L_dual, phi)  # Solve the adjoint Navier-Stokes PDE (stationary)

plot_compact_static(phi, Q, "Adjoint Velocity") # Plot all quantities (see implementation above)

Lei = replace(r, {wt:phi*z}); # Construct error indicators
ei = assemble(Lei);
gamma = abs(ei.get_local()) # Get value array (assume serial implementation)

# Mark cells for refinement
cell_markers = MeshFunction("bool", mesh, mesh.topology().dim())
gamma_0 = sorted(gamma, reverse=True)[int(len(gamma)*adapt_ratio) - 1]
for c in cells(mesh):
    cell_markers[c] = gamma[c.index()] > gamma_0

if adaptive: # Refine mesh
    rmesh = refine(mesh, cell_markers)
else:
    rmesh = refine(mesh)
mesh = rmesh # Shift to next adaptive iteration
```

FEniCS Platform

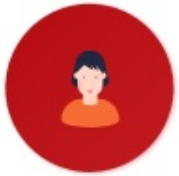
Introduction

How it works

Getting started

Demos

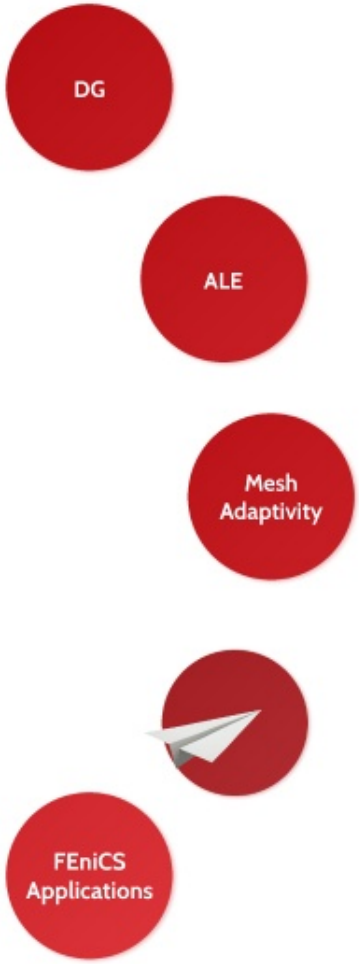
Other Features



Other Features

Roadmap 2019/2020 and HPC Considerations

Various applications



Discontinuous Galerkin method

We introduce jump $[\![\cdot]\!]$ and average $\{ \cdot \}$ of functions at the node x_k:

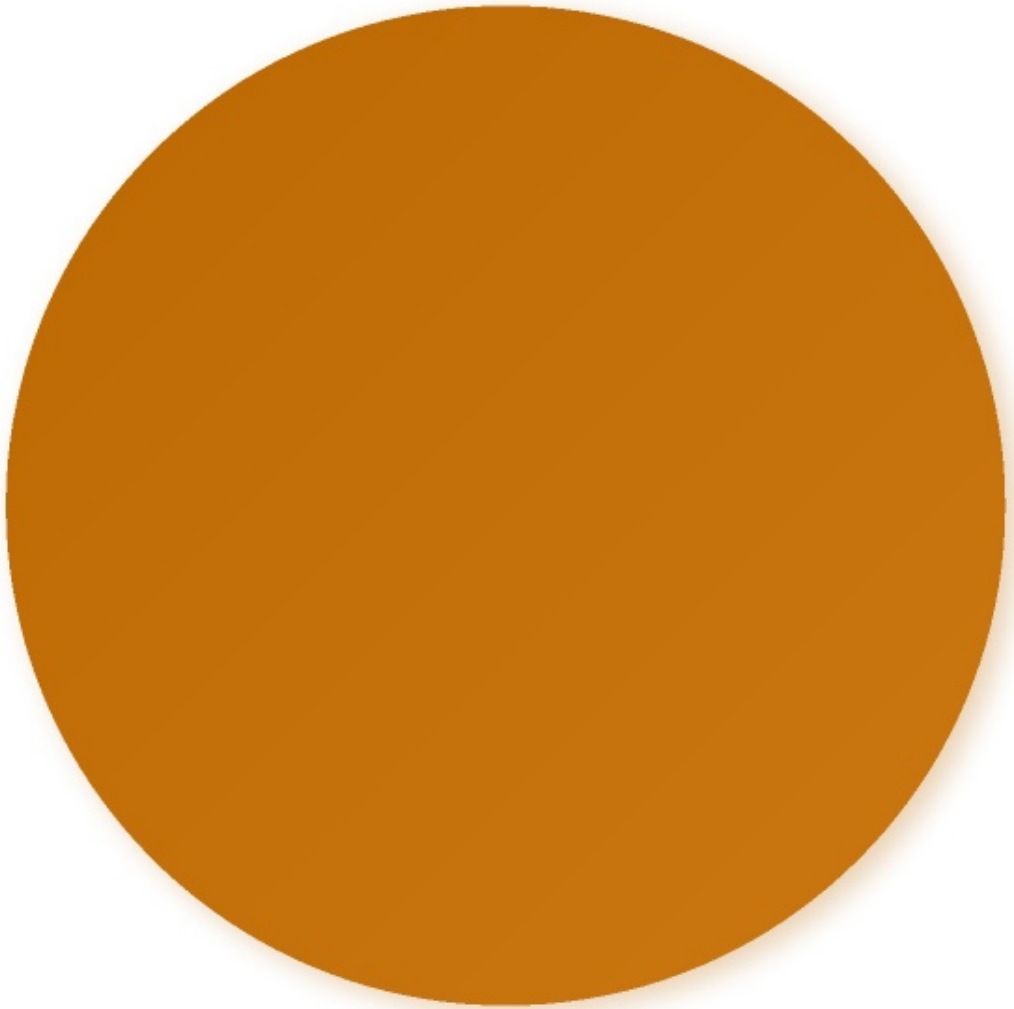
$$[\![v]\!]\Big|_{x_k} = v(x_k^+) - v(x_k^-), \quad \{v\}\Big|_{x_k} = 0.5(v(x_k^+) + v(x_k^-))$$

```
parameters["ghost_mode"] = "shared_facet"

class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return abs(x[0] - 1.0) < DOLFIN_EPS and on_boundary

# Load mesh
mesh = Mesh("../unitsquare_64_64.xml.gz")

# Defining the function spaces
V_dg = FunctionSpace(mesh, "DG", 1)
V_cg = FunctionSpace(mesh, "CG", 1)
V_u  = VectorFunctionSpace(mesh, "CG", 2)

# Create velocity Function
u = Function(V_u, "../unitsquare_64_64_velocity.xml.gz")

# Test and trial functions
v   = TestFunction(V_dg)
```

```
phi = TrialFunction(V_dg)

# Diffusivity
kappa = Constant(0.0)

# Source term
f = Constant(0.0)

# Penalty term
alpha = Constant(5.0)

# Mesh-related functions
n = FacetNormal(mesh)
h = CellDiameter(mesh)
h_avg = (h('+') + h('-'))/2

# ( dot(v, n) + |dot(v, n)| )/2.0
un = (dot(u, n) + abs(dot(u, n)))/2.0

# Bilinear form
a_int = dot(grad(v), kappa*grad(phi) - u*phi)*dx

a_fac = kappa*(alpha/h('+'))*dot(jump(v, n), jump(phi, n))*dS \
      - kappa*dot(avg(grad(v)), jump(phi, n))*dS \
      - kappa*dot(jump(v, n), avg(grad(phi)))*dS

a_vel = dot(jump(v), un('+')*phi('+') - un('-')*phi('-') )*dS  + dot(v, un*phi)*ds

a = a_int + a_fac + a_vel

# Linear form
L = v*f*dx
```

`class dolfin::ALE`

This class provides functionality useful for implementation of ALE (Arbitrary Lagrangian-Eulerian) methods, in particular moving the boundary vertices of a mesh and then interpolating the new coordinates for the interior vertices accordingly.



Mesh adaptivity with Omega_h

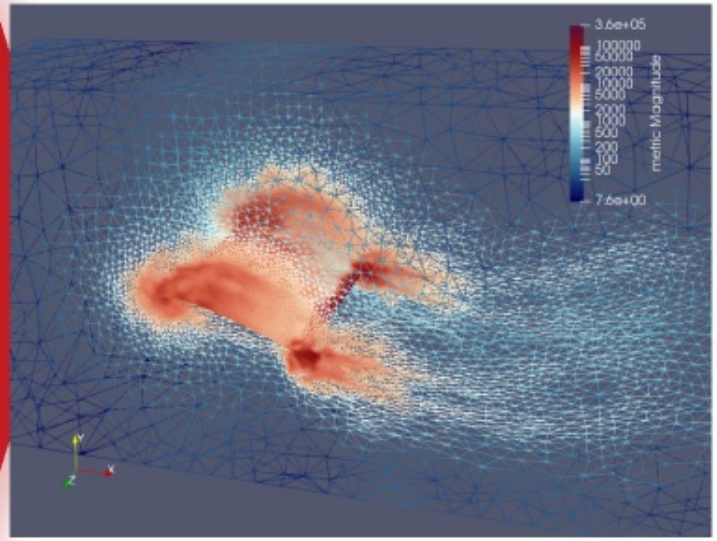
Omega_h is a C++11 library that implements tetrahedron and triangle mesh adaptivity, with a focus on scalable HPC performance using (optionally) MPI and OpenMP or CUDA. It is intended to provided adaptive functionality to existing simulation codes. Mesh adaptivity allows one to minimize both discretization error and number of degrees of freedom live during the simulation, as well as enabling moving object and evolving geometry simulations.

Intergation with FEnICS,

https://github.com/tamaradanceva/omega_h/tree/FEniCS

Demos:
https://github.com/tamaradanceva/FEniCS_Omega_h

Repo:
https://github.com/SNLComputation/omega_h



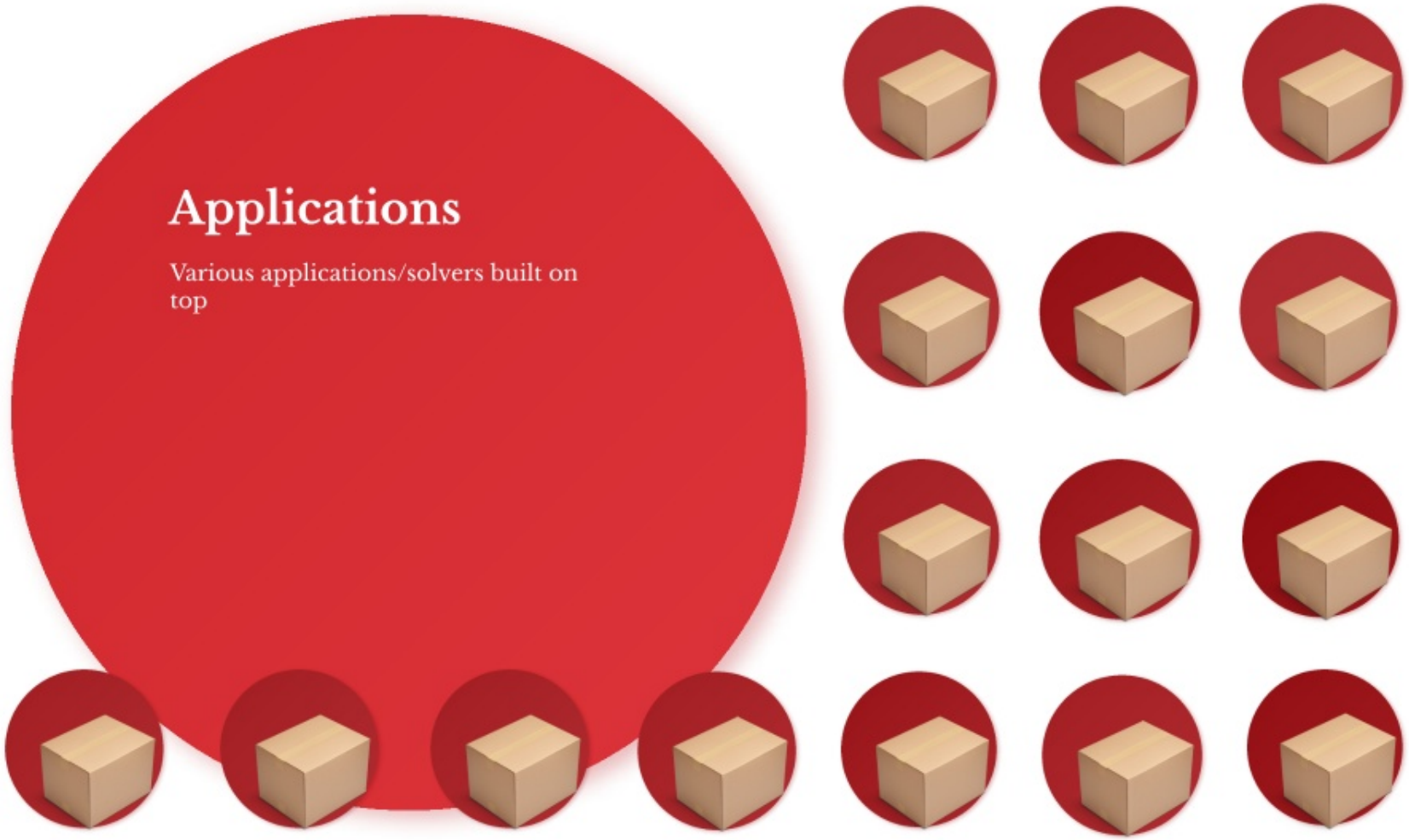
DOLFIN-X

DOLFIN-X is an experimental version of DOLFIN. It is being actively developed, but is not ready for production use.

HPC aspects

Requirements

- Compact, modular core designed to be extensible and to support custom additions
- Reduced public interface that could be kept more stable
- Proper namespacing
- Follow established standards and conventions wherever possible (design, packaging, testing, etc)
- Properly separated C++ and Python interfaces
- Simplified software engineering
- Distributed memory parallel design throughout
- Improved documentation
- Faster just-in-time compilation
- Support for modern Python JIT tools, e.g. numba
- Simple implementation of fast user ‘kernels’ from Python
- Provide just one way to perform an operation wherever possible


Applications
Various applications/solvers built on top

tIGAr

A Python library for isogeometric analysis (IGA) using FEniCS

https://github.com/david-kamensky/tIGAr

Topology Optimization with FEniCS

We present an implementation of topology optimization with a linear elasticity solver using FEniCS. We approach the problem of minimizing compliance using solid isotropic material with penalization (SIMP).

https://github.com/zfergus/fenics-topopt

fem-solvers

- FEniCS (finite element) solvers for 1D/2D Maxwell and Schroedinger problems

https://github.com/obsjames/fem-solvers

Julia wrapper for the FEniCS Finite Element library

Installs FEniCS
Using FEniCS within Julia

https://github.com/JuliaDiffEq/FEniCS.jl

FENaPack

- FEniCS Navier-Stokes preconditioning package

https://github.com/blechta/fenapack

Navier Stokes Solvers

https://github.com/MiroK/ns-fenics/tree/master/solvers

FEniCS-solid-mechanics

The aim of FEniCS Solid Mechanics library (formerly known as FEniCS Plasticity) is to provide functionality which makes it simpler for application developers to create solvers for complex problems

https://bitbucket.org/fenics-apps/fenics-solid-mechanics/src/master/

FEniCS blade

- FEniCS simulation of heat transfer in a turbine blade.

https://github.com/paulcon/fenics-blade

RBniCS

- reduced order modelling in FEniCS

https://github.com/mathLab/RBniCS

phaseflow-fenics

"The governing equations are composed of

- Incompressible flow driven by buoyancy: unsteady Navier-Stokes mas and momentum with Boussinesq approximation
- Convection-diffusion of the enthalpy field, with an enthalpy source term accounting for the laent heat of the phase-change material
- Convection-diffusion of a concentration field, e.g. for salt water or another binary alloy"

https://github.com/geo-fluid-dynamics/phaseflow-fenics

Python FEM and Multiphysics Simulations with FEniCS and FEATool

- Tools for running FEniCS with FEATool

https://www.featool.com/tutorial/2017/06/16/python-fem-and-multiphysics-simulations-with-fenics-and-featool

Diffusion MRI Framework in FEnICS

- Automated solutions of the Bloch-Torrey equations with FEniCS.
- Fast computation with Crank-Nicolson time-step strategy.
- High accuracy with an advanced finite element method.

https://github.com/van-dang/FEniCS-Colab

FEneutroniCS

- This package aims at solving the Boltzmann Transport Equation as used in the computational neutron transport community.

https://github.com/svancri/FEneutroniCS

Maelstroma

maelstrom is a numerical software tool for the solution of magnetohydrodynamics problems in cylindrical coordinates. As such, maelstrom includes time integrators for the heat equation, for the Navier--Stokes equations, and a stationary solver for the Maxwell equations, each in cylindrical coordinates.

https://github.com/nschloe/maelstrom

Eigenvalues

- PyQt GUI for Laplace eigenvalue/eigenfunction computations using FEniCS as finite element backend.

https://github.com/siudej/Eigenvalues

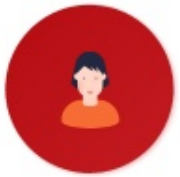
Example Python codes for FEniCS

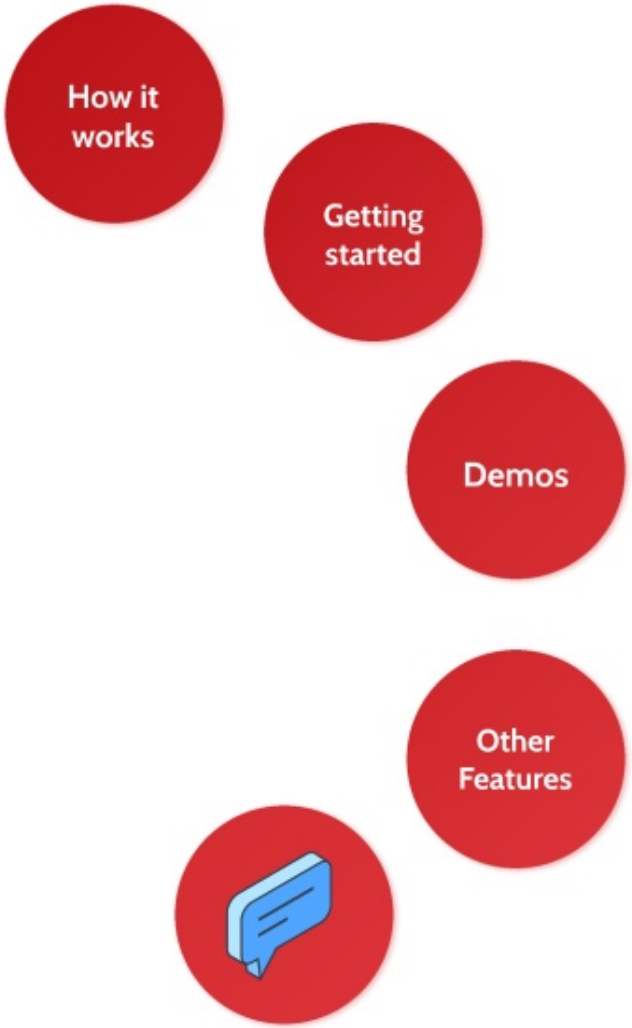
- Various boundary value problems
  Newton
  Eigenvalues and eigenfunctions

https://github.com/gregvw/FEniCS-examples

FEniCS Platform

Introduction





Questions

Discussion

FEniCS Platform

Introduction

How it works

Getting started

Demos

Other Features