PDEs in FEniCS

Emmanuel Mathioudakis

TECHNICAL UNIVERSITY OF CRETE
APPLIED MATHEMATICS AND COMPUTERS LABORATORY
73100 CHANIA - CRETE - GREECE

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

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/

FEniCS is a multi-institutional project

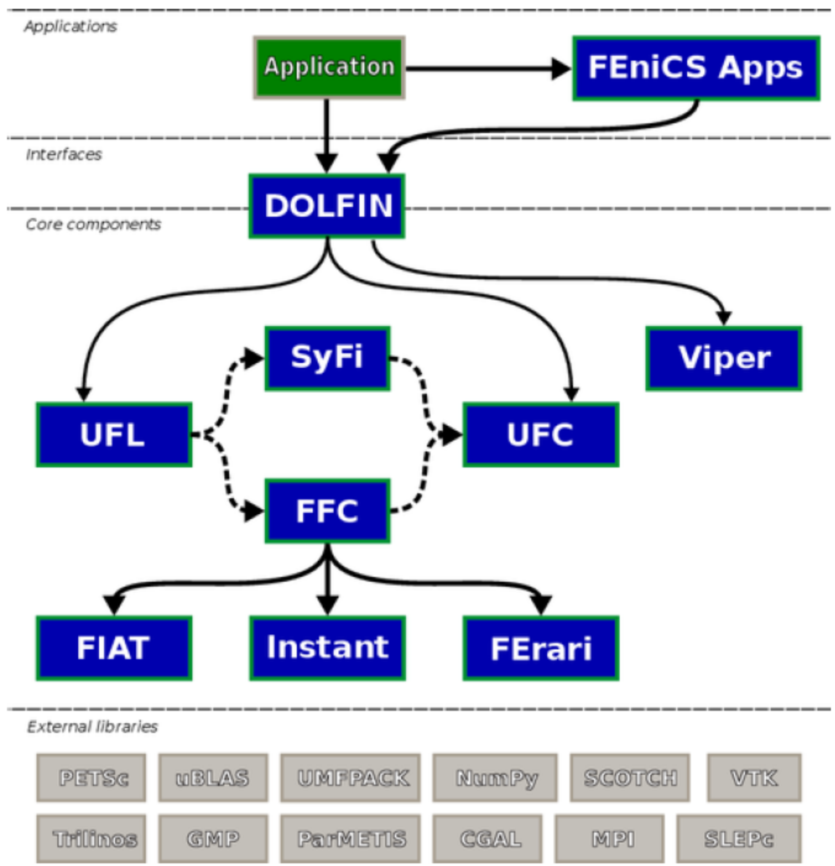
- Initiated 2003 by Univ. of Chicago and Chalmers Univ. of Technology (Ridgway Scott and Claes Johnson)
- Important contributions from
  - Univ.of Chicago (Rob Kirby, Andy Terrel, Matt Knepley, R. Scott)
  - Chalmers Univ. of Technology (Anders Logg, Johan Hoffman, Johan Janson)
  - Delft Univ. of Technology (Garth Wells, Kristian Oelgaard)
- Current key institutions:
  - Simula Research Laboratory (Anders Logg, Marie Rognes, Martin Alnæs, Johan Hake, Kent-Andre Mardal, ...)
  - Cambridge University (Garth Wells, ...)
- About 20 active developers
- Lots of application developers

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!

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



FEniCS is easy to install

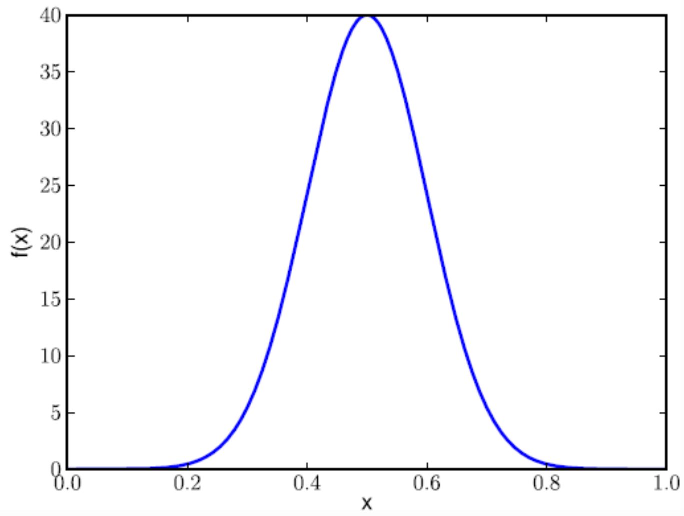
- Easiest on Ubuntu (Debian):
  `sudo apt-get install fenics`
- Mac OS X drag and drop installation (.dmg file)
- Windows binary installer
- Automated installation from source (compile & link)

We will first use FEniCS to solve a stationary diffusion problem, the Poisson equation on the unit interval

Poisson

$$-u'' = f(x); \quad u'(0) = 4; \quad u(1) = 0$$

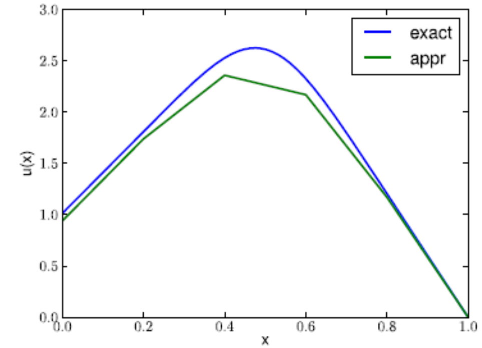
Source, $f(x)$



The problem can be stated, solved and plotted, using a **PyDOLFIN** script written in 16 lines

```
>>> from dolfin import *
>>>
>>> mesh = UnitInterval(5)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> def right(x, on_boundary):
...     return x > 1-DOLFIN_EPS
...
>>> g = Constant(0)
>>> bc = DirichletBC(V, g, right)
>>>
>>> f = Expression("A*exp(-pow(x[0]-0.5,2)/(2*pow(sigma,2)))")
>>> f.A, f.sigma = 40, 0.1
>>>
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> a = inner(grad(v),grad(u))*dx
>>> L = f*v*dx - 4*v*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
>>>
>>> plot(u)
```

$$
\begin{aligned}
-u'' &= f \\
u'(0) &= 4 \\
u(1) &= 0
\end{aligned}
$$

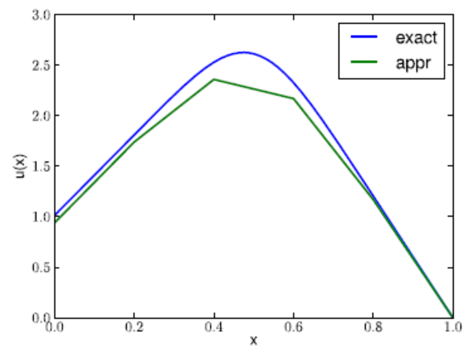


The problem can be stated, solved and plotted, using a **PyDOLFIN** script written in 16 lines

```
>>> from dolfin import *
>>>
>>> mesh = UnitInterval(5)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> def right(x, on_boundary):
...     return x > 1-DOLFIN_EPS
...
>>> g = Constant(0)
>>> bc = DirichletBC(V, g, right)
>>>
>>> f = Expression("A*exp(-pow(x[0]-0.5,2)/(2*pow(sigma,2)))")
>>> f.A, f.sigma = 40, 0.1
>>>
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> a = inner(grad(v),grad(u))*dx
>>> L = f*v*dx - 4*v*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
>>>
>>> plot(u)
```

Domain
Solution space

$$
\begin{aligned}
-u'' &= f \\
u'(0) &= 4 \\
u(1) &= 0
\end{aligned}
$$

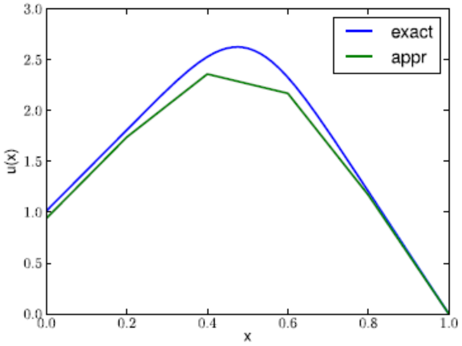


The problem can be stated, solved and plotted, using a **PyDOLFIN** script written in 16 lines

```
>>> from dolfin import *
>>>
>>> mesh = UnitInterval(5)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> def right(x, on_boundary):
...     return x > 1-DOLFIN_EPS
...
>>> g = Constant(0)
>>> bc = DirichletBC(V, g, right)
>>>
>>> f = Expression("A*exp(-pow(x[0]-0.5,2)/(2*pow(sigma,2)))")
>>> f.A, f.sigma = 40, 0.1
>>>
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> a = inner(grad(v),grad(u))*dx
>>> L = f*v*dx - 4*v*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
>>>
>>> plot(u)
```

Boundary condition

$$
\begin{aligned}
-u'' &= f \\
u'(0) &= 4 \\
u(1) &= 0
\end{aligned}
$$

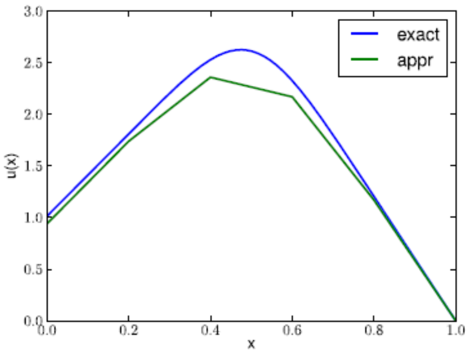


The problem can be stated, solved and plotted, using a **PyDOLFIN** script written in 16 lines

```
>>> from dolfin import *
>>>
>>> mesh = UnitInterval(5)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> def right(x, on_boundary):
...     return x > 1-DOLFIN_EPS
...
>>> g = Constant(0)
>>> bc = DirichletBC(V, g, right)
>>>
>>> f = Expression("A*exp(-pow(x[0]-0.5,2)/(2*pow(sigma,2)))")
>>> f.A, f.sigma = 40, 0.1
>>>
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> a = inner(grad(v),grad(u))*dx
>>> L = f*v*dx - 4*v*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
>>>
>>> plot(u)
```

Source term

$$
\begin{aligned}
-u'' &= f \\
u'(0) &= 4 \\
u(1) &= 0
\end{aligned}
$$

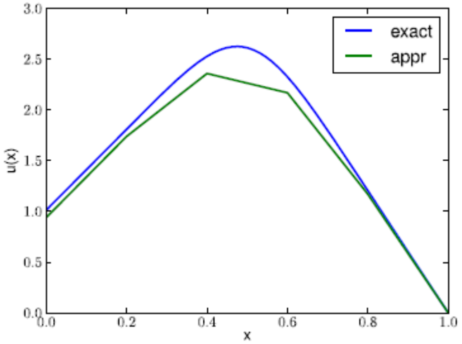


The problem can be stated, solved and plotted, using a **PyDOLFIN** script written in 16 lines

```
>>> from dolfin import *
>>>
>>> mesh = UnitInterval(5)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> def right(x, on_boundary):
...     return x > 1-DOLFIN_EPS
...
>>> g = Constant(0)
>>> bc = DirichletBC(V, g, right)
>>>
>>> f = Expression("A*exp(-pow(x[0]-0.5,2)/(2*pow(sigma,2)))")
>>> f.A, f.sigma = 40, 0.1
>>>
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> a = inner(grad(v),grad(u))*dx
>>> L = f*v*dx - 4*v*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
>>>
>>> plot(u)
```

Variational formulation

$$\begin{aligned} -u'' &= f \\ u'(0) &= 4 \\ u(1) &= 0 \end{aligned}$$

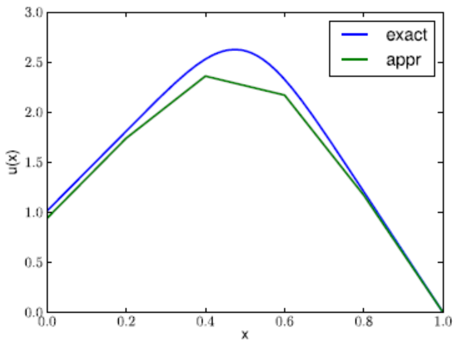


The problem can be stated, solved and plotted, using a **PyDOLFIN** script written in 16 lines

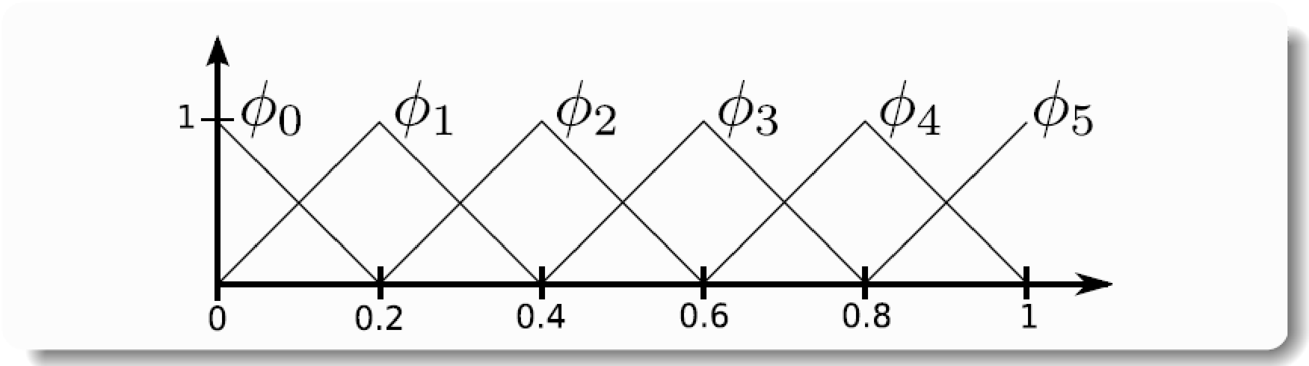
```
>>> from dolfin import *
>>>
>>> mesh = UnitInterval(5)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> def right(x, on_boundary):
...     return x > 1-DOLFIN_EPS
...
>>> g = Constant(0)
>>> bc = DirichletBC(V, g, right)
>>>
>>> f = Expression("A*exp(-pow(x[0]-0.5,2)/(2*pow(sigma,2)))")
>>> f.A, f.sigma = 40, 0.1
>>>
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> a = inner(grad(v),grad(u))*dx
>>> L = f*v*dx - 4*v*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
>>>
>>> plot(u)
```

Solve and plot

$$\begin{aligned} -u'' &= f \\ u'(0) &= 4 \\ u(1) &= 0 \end{aligned}$$



TECHNICAL UNIVERSITY OF CRETE
APPLIED MATHEMATICS AND COMPUTERS LABORATORY



A **FunctionSpace** in PyDOLFIN takes a mesh and a *finite element* as arguments.

```
>>> mesh = UnitInterval(5)
>>> V = FunctionSpace(mesh, "Lagrange", 1)
```

Any scalar function on the domain can be approximated using a linear combination of the basis functions in the **FunctionSpace**

Discrete **Function**

$$u(x) \simeq \sum_{j=0}^{5} u_j \phi_j(x)$$
$$= u_0\phi_0 + u_1\phi_1 + u_2\phi_2 + u_3\phi_3 + u_4\phi_4 + u_5\phi_5$$
$$U = [u_0, u_1, u_2, u_3, u_4, u_5]$$

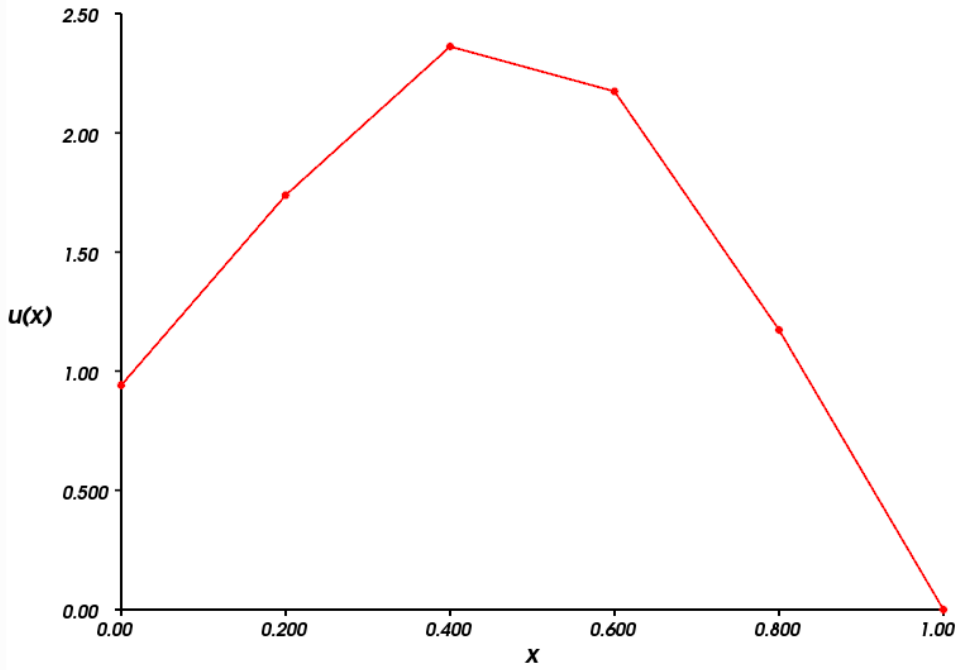
Evaluation is done by interpolation

```
>>> u = Function(V)
>>> u.vector().array()
[ 0.,  0.,  0.,  0.,  0.,  0.]
>>> u(0.1)
0.0
>>> u.vector()[0] = 1
>>> u(0.1)
0.49999999999999967
```

- U is called the vector of *expansion coefficients*

The solution of our problem is a discrete **Function**

Solution: $u(x)$




```
>>> plot(u)
>>> u.vector().array()
array([ 0.94,  1.74,  2.36,  2.17,  1.17,  0.  ])
```

The PDE can be re-written using the *discrete weak* formulation, which eventually will let us describe our problem as a *linear algebraic system*: $Ax = b$

Strong formulation

$$-u'' = f$$

- Should be true for every point (*strong*) in space

Discrete weak formulation

$$\begin{aligned} u(x) &= \sum_{j=0}^{5} u_j \phi_j(x) \\ -\int_0^1 u'' \phi_i dx &= \int_0^1 f \phi_i dx, \ i = 0, ..., 5 \end{aligned}$$

- By weighting the equation with ϕ_i and taking the integral over the whole domain, we solve an approximation of u (*weak*)

Before we simplify the *weak form*, we treat the *dirichlet* boundary condition: $u(1) = 0$

- The *dirichlet* boundary conditions at x=1, is treated by letting $\phi_i(1) = 0$ ($\Rightarrow \phi_5 = 0$)
- Then adding a function $g(x)$ to our *function space* which equals 0 at x=1

$$u(x) = \sum_{j=0}^{5} u_j \phi_j(x) + g(x)$$

```
>>> def right(x, on_boundary):
...     return x > 1-DOLFIN_EPS
...
>>> g = Constant(0)
>>> bc = DirichletBC(V, g, right)
```

To be able to use piecewise linear basis functions, we need to partial integrate the left hand side of the (discrete) weak form as, $\int_0^1 u''\phi_i dx = 0$

Integration by parts

$$-\int_0^1 u'v\,dx = \int_0^1 uv'\,dx - [uv]_0^1$$

To be able to use piecewise linear basis functions, we need to partial integrate the left hand side of the (discrete) weak form as, $\int_0^1 u''\phi_i dx = 0$

Integration by parts

$$
\begin{aligned}
-\int_0^1 u'v\,dx &= \int_0^1 uv'\,dx - [uv]_0^1 \\
-\int_0^1 u''\phi_i dx &= \int_0^1 u'\phi_i' dx + u'(0)\phi_i(0) - u'(1)\phi_i(1)
\end{aligned}
$$

- This includes the derivatives at the boundaries!

To be able to use piecewise linear basis functions, we need to partial integrate the left hand side of the (discrete) weak form as, $\int_0^1 u''\phi_i dx = 0$

Integration by parts

$$
\begin{aligned}
-\int_0^1 u' v \, dx &= \int_0^1 u v' \, dx - [uv]_0^1 \\
-\int_0^1 u'' \phi_i dx &= \int_0^1 u' \phi_i' dx + u'(0)\phi_i(0) - u'(1)\phi_i(1)
\end{aligned}
$$

- This includes the derivatives at the boundaries!
- Recall that our *boundary conditions* implies that $\phi_i(1) = 0$ and $u'(0) = 4$, which gives us:

$$\int_0^1 u' \phi_i' dx = \int_0^1 f \phi_i dx - 4\phi_i(1), \; i = 0, \ldots, 5$$

We are now ready to describe the *weak form* as a *linear algebraic system*: $Ax = b$

$$\text{LHS: } \int_0^1 u'\phi_i' dx = \int_0^1 \left(\sum_{j=0}^{5} u_j \phi_j \right)' \phi_i' dx$$

We are now ready to describe the *weak form* as a *linear algebraic system*: $Ax = b$

$$\text{LHS: } \int_0^1 u' \phi_i' dx = \int_0^1 \left(\sum_{j=0}^{5} u_j \phi_j \right)' \phi_i' dx =$$

$$\sum_{j=0}^{5} \left(\int_0^1 \phi_j' \phi_i' dx \right) u_j = \int_0^1 f \phi_i dx - 4\phi_i(0), \; i = 0, \ldots, 5$$

We are now ready to describe the *weak form* as a *linear algebraic system*: $Ax = b$

$$\text{LHS: } \int_0^1 u' \phi_i' dx = \int_0^1 \left(\sum_{j=0}^{5} u_j \phi_j \right)' \phi_i' dx =$$

$$\sum_{j=0}^{5} \left(\int_0^1 \phi_j' \phi_i' dx \right) u_j = \int_0^1 f \phi_i dx - 4\phi_i(0), \; i = 0, \ldots, 5$$

$$Ax = b, \text{ where :}$$

$$A_{ij} = \int_0^1 \phi_j' \phi_i' dx, \, x_j = u_j \text{ and } b_i = \int_0^1 f \phi_i dx - 4\phi_i(0)$$

PyDOLFIN provides functionality to assemble the *matrix A* and the *vector b* (using numerical integration), and to solve the linear system

Recall that after integration by part we had:

$$\int_0^1 u' \phi_i' dx = \int_0^1 f \phi_i dx - 4\phi_i(0), \; i = 0, \ldots, 5$$

PyDOLFIN provides functionality to assemble the *matrix A* and the *vector b* (using numerical integration), and to solve the linear system

Recall that after integration by part we had:

$$\int_0^1 u'\phi_i' dx = \int_0^1 f\phi_i dx - 4\phi_i(0),\ i = 0, ..., 5$$

We use v instead of ϕ_i and we can write our problem as: find $u \in V$ such that

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

$$a(u,v) = \int_\Omega u'v'dx \quad \text{and} \quad L(v) = \int_\Omega f\,v\,dx - \int_{\partial\Omega} 4v\,ds$$

The *variational formulation*

UFL in **FEniCS** can be used to describe variational forms, and **PyDOLFIN** can be used to solve **VariationalProblems**

Mathematical notation:

find $u \in V$ such that

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

where:

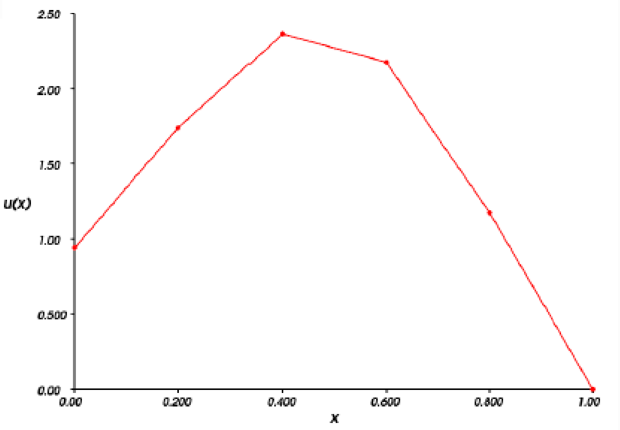
$$a(u, v) = \int_\Omega u' v' dx$$

$$L(v) = \int_\Omega f\, v\, dx - \int_{\partial\Omega} 4v\, ds$$

PyDOLFIN notation:

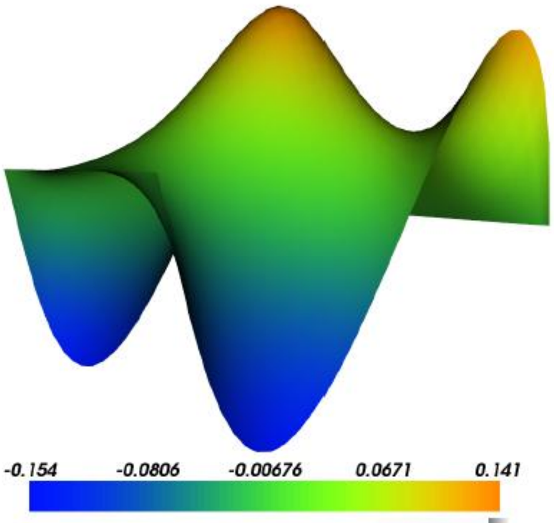
```
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> a = inner(grad(v),grad(u))*dx
>>> L = f*v*dx - 4*v*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
```

```
>>> from dolfin import *
>>>
>>> mesh = UnitInterval(5)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> def right(x, on_boundary):
...     return x > 1-DOLFIN_EPS
...
>>> g = Constant(0)
>>> bc = DirichletBC(V, g, right)
>>>
>>> f = Expression("A*exp(-pow(x[0]-0.5,2)/(2*pow(sigma,2)))")
>>> f.A, f.sigma = 40, 0.1
>>>
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> a = inner(grad(v),grad(u))*dx
>>> L = f*v*dx - 4*v*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
>>>
>>> plot(u)
```



$$-\nabla^2 u = f; \ \frac{\partial u}{\partial n}(:, [0,1]) = g; \ u([0,1], :) = 0$$

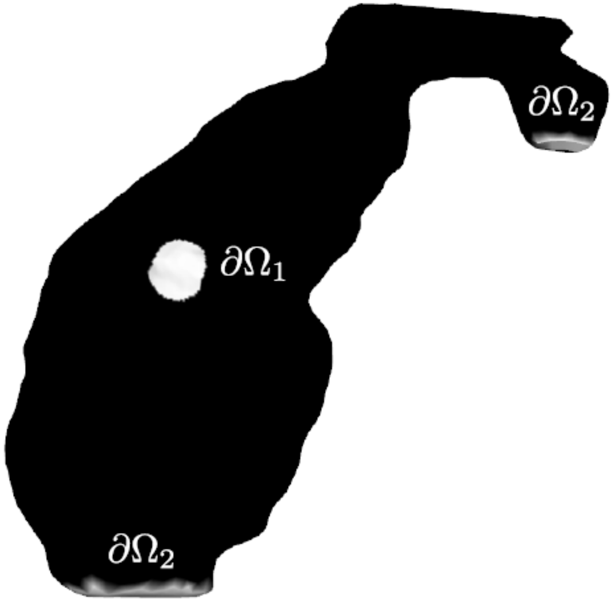
```
>>> from dolfin import *
>>>
>>> mesh = UnitSquare(32, 32)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> def boundary(x):
...     return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EP
...
>>> u0 = Constant(0.0)
>>> bc = DirichletBC(V, u0, boundary)
>>>
>>> v = TestFunction(V)
>>> u = TrialFunction(V)
>>> 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(v), grad(u))*dx
>>> L = v*f*dx + v*g*ds
>>>
>>> problem = VariationalProblem(a, L, bc)
>>> u = problem.solve()
>>> plot(u, interactive=True)
```



Time dependent system, like the diffusion equation, can be solved using the same framework

PDE

$$
\begin{aligned}
\dot{u} &= D\nabla^2 u && \text{in } \Omega \\
D\frac{\partial u}{\partial n} &= J(t) && \text{on } \partial\Omega_1 \\
u &= 1 && \text{on } \partial\Omega_2 \\
u(0,:) &= 1 \\
J(t) &= \begin{cases} 100 & : t \leq J_{stop} \\ 0 & : t > J_{stop} \end{cases}
\end{aligned}
$$



Diffusion equation solved using **PyDOLFIN** (Domain declarations)

```
>>> from dolfin import *
>>>
>>> mesh = Mesh("single-TT.xml.gz")
>>> subdomains = MeshFunction("uint", mesh, 2)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> tstop = 3.0; J_stop = 2.0; dt = 0.02; u0 = 1; J0 = 100; D = 100
>>>
>>> class Outflow(SubDomain):
...     def inside(self, x, on_boundary):
...         return (on_boundary and -110 < x[0] and x[0] < -50 and \
...                 70 < x[1] and x[1] < 130 and \
...                 22 < x[2] and x[2] < 82) or (5 < x[0] and x[0] < 75 and \
...                 -105 < x[1] and x[1] < -35 and \
...                 -210 < x[2] and x[2] < -140)
...
>>> class Inflow(SubDomain):
...     def inside(self, x, on_boundary):
...         return on_boundary and ((x[0]-65)**2+(x[1]+30)**2+x[2]**2 < 16**2)
...
>>> outflow = Outflow()
>>> inflow = Inflow()
>>>
>>> inflow.mark(subdomains, 2)
>>>
>>> out_values = Constant(1)
>>> bc = DirichletBC(V, out_values, outflow)
```

Diffusion equation solved using **PyDOLFIN** (Domain declarations)

```
>>> from dolfin import *
>>>
>>> mesh = Mesh("single-TT.xml.gz")
>>> subdomains = MeshFunction("uint", mesh, 2)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> tstop = 3.0; J_stop = 2.0; dt = 0.02; u0 = 1; J0 = 100; D = 100
>>>
>>> class Outflow(SubDomain):
...     def inside(self, x, on_boundary):
...         return (on_boundary and -110 < x[0] and x[0] < -50 and \
...                 70 < x[1] and x[1] < 130 and \
...                 22 < x[2] and x[2] < 82) or (5 < x[0] and x[0] < 75 and \
...                 -105 < x[1] and x[1] < -35 and \
...                 -210 < x[2] and x[2] < -140)
...
>>> class Inflow(SubDomain):
...     def inside(self, x, on_boundary):
...         return on_boundary and ((x[0]-65)**2+(x[1]+30)**2+x[2]**2 < 16**2)
...
>>> outflow = Outflow()
>>> inflow = Inflow()
>>>
>>> inflow.mark(subdomains, 2)
>>>
>>> out_values = Constant(1)
>>> bc = DirichletBC(V, out_values, outflow)
```

Domain, Subdomains & Solution space

Diffusion equation solved using **PyDOLFIN** (Domain declarations)

Model parameters

```
>>> from dolfin import *
>>>
>>> mesh = Mesh("single-TT.xml.gz")
>>> subdomains = MeshFunction("uint", mesh, 2)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> tstop = 3.0; J_stop = 2.0; dt = 0.02; u0 = 1; J0 = 100; D = 100
>>>
>>> class Outflow(SubDomain):
...     def inside(self, x, on_boundary):
...         return (on_boundary and -110 < x[0] and x[0] < -50 and \
...                 70 < x[1] and x[1] < 130 and \
...                 22 < x[2] and x[2] < 82) or (5 < x[0] and x[0] < 75 and \
...                 -105 < x[1] and x[1] < -35 and \
...                 -210 < x[2] and x[2] < -140)
...
>>> class Inflow(SubDomain):
...     def inside(self, x, on_boundary):
...         return on_boundary and ((x[0]-65)**2+(x[1]+30)**2+x[2]**2 < 16**2)
...
>>> outflow = Outflow()
>>> inflow = Inflow()
>>>
>>> inflow.mark(subdomains, 2)
>>>
>>> out_values = Constant(1)
>>> bc = DirichletBC(V, out_values, outflow)
```

Diffusion equation solved using **PyDOLFIN** (Domain declarations)

```
>>> from dolfin import *
>>>
>>> mesh = Mesh("single-TT.xml.gz")
>>> subdomains = MeshFunction("uint", mesh, 2)
>>> V = FunctionSpace(mesh, "CG", 1)
>>>
>>> tstop = 3.0; J_stop = 2.0; dt = 0.02; u0 = 1; J0 = 100; D = 100
>>>
>>> class Outflow(SubDomain):
...     def inside(self, x, on_boundary):
...         return (on_boundary and -110 < x[0] and x[0] < -50 and \
...                 70 < x[1] and x[1] < 130 and \
...                 22 < x[2] and x[2] < 82) or (5 < x[0] and x[0] < 75 and \
...                 -105 < x[1] and x[1] < -35 and \
...                 -210 < x[2] and x[2] < -140)
...
>>> class Inflow(SubDomain):
...     def inside(self, x, on_boundary):
...         return on_boundary and ((x[0]-65)**2+(x[1]+30)**2+x[2]**2 < 16**2)
...
>>> outflow = Outflow()
>>> inflow = Inflow()
>>>
>>> inflow.mark(subdomains, 2)
>>>
>>> out_values = Constant(1)
>>> bc = DirichletBC(V, out_values, outflow)
```

Define boundaries

Diffusion equation solved using **PyDOLFIN** (linear algebra initialization)

```
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> K = assemble(inner(grad(u),grad(v))*dx)
>>> M = assemble(u*v*dx)
>>> source = assemble(v*ds(2), exterior_facet_domains=subdomains)
>>>
>>> u_n = Function(V)
>>>
>>> A = K.copy()
>>> b = Vector(A.size(1))
>>> b[:] = 0.0
>>> x = u_n.vector()
>>> x[:] = u0
```

Diffusion equation solved using **PyDOLFIN** (linear algebra initialization)

```
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> K = assemble(inner(grad(u),grad(v))*dx)
>>> M = assemble(u*v*dx)
>>> source = assemble(v*ds(2), exterior_facet_domains=subdomains)
>>>
>>> u_n = Function(V)
>>>
>>> A = K.copy()
>>> b = Vector(A.size(1))
>>> b[:] = 0.0
>>> x = u_n.vector()
>>> x[:] = u0
```

Basis functions

Diffusion equation solved using **PyDOLFIN** (linear algebra initialization)

Assemble tensors

```
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> K = assemble(inner(grad(u),grad(v))*dx)
>>> M = assemble(u*v*dx)
>>> source = assemble(v*ds(2), exterior_facet_domains=subdomains)
>>>
>>> u_n = Function(V)
>>>
>>> A = K.copy()
>>> b = Vector(A.size(1))
>>> b[:] = 0.0
>>> x = u_n.vector()
>>> x[:] = u0
```

Diffusion equation solved using **PyDOLFIN** (linear algebra initialization)

```
>>> u = TrialFunction(V)
>>> v = TestFunction(V)
>>>
>>> K = assemble(inner(grad(u),grad(v))*dx)
>>> M = assemble(u*v*dx)
>>> source = assemble(v*ds(2), exterior_facet_domains=subdomains)
>>>
>>> u_n = Function(V)
>>>
>>> A = K.copy()
>>> b = Vector(A.size(1))
>>> b[:] = 0.0
>>> x = u_n.vector()
>>> x[:] = u0
```

Initialise linear algebra

The diffusion equation on *weak form*, using *backward Euler* for the time discretization

Strong form

$$\dot{u} = D\nabla^2 u \text{ in } \Omega; \qquad D\frac{\partial u}{\partial n} = J(t) \text{ on } \partial\Omega_1; \qquad u = 1 \text{ on } \partial\Omega_2$$

Weak form

$$\begin{aligned}
\int_\Omega \dot{u} v \, dx &= \int_\Omega D\nabla^2 u v \, dx \\
\int_\Omega \dot{u} v \, dx &= -D \int_\Omega \nabla u \nabla v \, dx + \int_{\partial\Omega_1} D\frac{\partial u}{\partial n} v \, ds \\
\int_\Omega \frac{u^n - u^{n-1}}{\Delta t} v \, dx &= -D \int_\Omega \nabla u^n \nabla v \, dx + \int_{\partial\Omega_1} J v \, ds \\
\int_\Omega u^n v + \Delta t D \nabla u^n \nabla v \, dx &= \int_\Omega u^{n-1} v \, dx + \Delta t J \int_{\partial\Omega_1} v \, ds
\end{aligned}$$

Instead of using the *variational form*, it is convenient to express the *weak form* on *matrix form*

Matrix form

$$(\mathbf{M} + \Delta t\, D\, \mathbf{K})\, U^n \quad = \quad \mathbf{M}\, U^{n-1} + \Delta t\, J(t)\, source$$

where U^n is the vector of *expansion coefficients* and

$$\left.\begin{aligned} M_{ij} &= \int_\Omega \phi_i \phi_j dx \\ K_{ij} &= \int_\Omega \nabla\phi_i \nabla\phi_j dx; \\ source_i &= \int_{\partial\Omega_1} \phi_i ds \end{aligned}\right\} \text{ for all } \phi_i \text{ and } \phi_j \text{ in } V$$

Assemble of tensors in **PyDOLFIN**

```
>>> K = assemble(inner(grad(u),grad(v))*dx)
>>> M = assemble(u*v*dx)
>>> source = assemble(v*ds(2), exterior_facet_domains=subdomains)
```

Each time step we need to solve a *linear system*

Weak form on *Matrix form*

$$(\mathbf{M} + \Delta t\, D\, \mathbf{K})\, U^n \quad = \quad \mathbf{M}\, U^{n-1} + \Delta t\, J(t)\, source$$

Each time step we need to solve a *linear system*

Weak form on *Matrix form*

$$(\mathbf{M} + \Delta t\, D\, \mathbf{K})\, U^n \quad = \quad \mathbf{M}\, U^{n-1} + \Delta t\, J(t)\, source$$

Time stepping in **PyDOLFIN**

```
>>> t = 0.0
>>> plot(u_n, vmin=1, vmax=5, rescale=False)
>>> while t < tstop:
...     t += float(dt)
...
...     A.assign(K)
...     A *= D*dt
...     A += M
...
...     b[:] = 0.0
...     if t <= J_stop:
...         b[:] = source
...         b *= dt*J0
...
...     b+= M*x
...
...     bc.apply(A, b)
...
...     solve(A, x, b)
...     plot(u_n)
```

```
>>> tstop = 3.0; J_stop = 2.0; dt = 0.02; u0 = 1; J0 = 100; D = 100
```