Lecture on FEM in FEniCSx

an introduction

Måns I. Andersson[1]

2022

Table of Contents

FEniCSx

Some syntax has been updated in FEniCSx and most of the code will not run out-of-the-box (but almost).

Table of Contents

Introductory note

- Finite Element Method (FEM) is a method for finding approximated solutions of partial differential equations (PDEs). With some form of optimality of the solution in the given space.

Introductory note

- Finite Element Method (FEM) is a method for finding approximated solutions of partial differential equations (PDEs). With some form of optimality of the solution in the given space.
- FEM is used in many disciplines, such as: Solid Mechanics, Fluid Dynamics, and Electromagnetism. Many different conventions in literature.

Introductory note

- Finite Element Method (FEM) is a method for finding approximated solutions of partial differential equations (PDEs). With some form of optimality of the solution in the given space.
- FEM is used in many disciplines, such as: Solid Mechanics, Fluid Dynamics, and Electromagnetism. Many different conventions in literature.
- FEM has a very mature mathematical background: Error estimation, convergence, ...

Introductory note

- Finite Element Method (FEM) is a method for finding approximated solutions of partial differential equations (PDEs). With some form of optimality of the solution in the given space.
- FEM is used in many disciplines, such as: Solid Mechanics, Fluid Dynamics, and Electromagnetism. Many different conventions in literature.
- FEM has a very mature mathematical background: Error estimation, convergence, ...
- Also mature software stack with commercial and open source contestants:
  - FEniCS / FEniCSx
  - DUNE
  - deal.ii
  - COMSOL
  - ANSYS

Some Notation

Inner product

$$(u, v)_{L^2(\Omega)} = \int_\Omega uv \ d\Omega, \quad \langle u, v \rangle = \int_{\partial\Omega} uv \ dS.$$

Function space:

$$w = \{w \in C^2([a,b]) : (\nabla w, \nabla w) + (w, w) < \infty, w|_{a,b} = 0\} \equiv H_0^2([a,b])$$

Important relations

$$(u, u) = ||u||^2 > 0, \quad ||u + v|| \leq ||u|| + ||v||, \quad |(u, v)| \leq ||v||||u||$$

Read more

(Babuška)–Lax–Milgram theorem gives conditions under which a bilinear form can be "inverted" to show the existence and uniqueness of a weak solution to a given PDE problem.

1D Poisson example

- Given this simple 1D problem of Poisson's eq. with Dirichlet BC.

$$
\begin{aligned}
-\Delta u = f \qquad\qquad\qquad & \text{on} \quad \Omega \\
u(0) = 5, u(1) = 3 \quad & \text{on} \quad \partial\Omega
\end{aligned}
$$

1D Poisson example

- Given this simple 1D problem of Poisson's eq. with Dirichlet BC.

$$\begin{aligned} -\Delta u = f \qquad & \text{on} \quad \Omega \\ u(0) = 5, u(1) = 3 \quad & \text{on} \quad \partial\Omega \end{aligned}$$

- Assume a *test function* v with the properties: zero at boundary points and is differentiable on the domain.

1D Poisson example

- Given this simple 1D problem of Poisson's eq. with Dirichlet BC.

$$
\begin{aligned}
-\Delta u = f \qquad\qquad & \text{on} \quad \Omega \\
u(0) = 5, u(1) = 3 \quad & \text{on} \quad \partial\Omega
\end{aligned}
$$

- Assume a *test function* v with the properties: zero at boundary points and is differentiable on the domain.
- Multiply with test function:

$$
\begin{aligned}
(-\Delta u, v) &= (f, v) \quad \{\text{integration by parts}\} \\
(\nabla u, \nabla v) + \int_0^1 u' v dx &= (f, v) \quad \{\text{by definition of v}\} \\
(\nabla u, \nabla v) &= (f, v)
\end{aligned}
$$

1D Poisson example cont.

- What we found above is the weak formulation of the Poisson equation

$$a(u, v) = (\nabla u, \nabla v), \quad L(v) = (f, v) \tag{1}$$

- Next step is to discretize the problem. We do this by choosing a *test function* and a *trial function* from a discrete space. Such as piecewise linear polynomials.

$$u_h(x) = \sum_{j=0}^{N+1} \xi_j \phi_j(x), \quad v_h(x) = \phi(x) \tag{2}$$

- the "hat functions" are defined in appendix.

1D Poisson example cont. 2

We expand the known BC

$$u_h(x) = \sum_{j=1}^{N} \xi_j \phi_j(x) + 3\phi_0 + 5\phi_{N+1} \text{ and } v(x) = \phi_i \tag{3}$$

Plug in the functions in the weak form

$$a(u_h, v_h) = \sum_{j=1}^{N} \underbrace{\Big(\int_0^1 \phi_j' \phi_i' \, dx \Big)}_{A_{ij}} \underbrace{\xi_j}_{\mathbf{x}_j} \qquad i = 1, \dots, N \tag{4}$$

$$L(v_h) = \underbrace{\int_0^1 f \phi_i \, dx - 3\phi_0' \phi_i' - 5\phi_{N+1}' \phi_i' \, dx,}_{b_i} \qquad i = 1, \dots, N. \tag{5}$$

Note that the basis functions have limited support, and spans the solution with Finite Elements. Here some quadrature rule is needed.

The FEM algorithm for solving PDEs

Write the problem on weak form. Prove existence and uniqueness
Find u such that.

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

Create a Finite Element Space that corresponds to the problem, re-state the problem
Find u_h such that.

$$a(u_h, v_h) = L(v_h) \quad u_h, v_h \in V_h$$

Assemble $a(u_h, v_h) \implies A$ and $L(v_h) \implies b$,

$$A\xi = b.$$

Solve system with some direct or iterative linear solver (CG, GMRES, ...).

Table of Contents

FEniCSx

FEniCS is a software library for automatic solution of PDEs. FEniCSx is a (complete) rewrite designed for modern HPC.

- Python frontend highly integrated with *numpy*, high-performance C++ backend.

FEniCSx

FEniCS is a software library for automatic solution of PDEs. FEniCSx is a (complete) rewrite designed for modern HPC.

- Python frontend highly integrated with *numpy*, high-performance C++ backend.
- High-level near mathematical notation.

FEniCSx

FEniCS is a software library for automatic solution of PDEs. FEniCSx is a (complete) rewrite designed for modern HPC.

- Python frontend highly integrated with *numpy*, high-performance C++ backend.
- High-level near mathematical notation.
- Heavily relies on code generation. Used to be a DSL is turning into a library.

FEniCSx

FEniCS is a software library for automatic solution of PDEs. FEniCSx is a (complete) rewrite designed for modern HPC.

- Python frontend highly integrated with *numpy*, high-performance C++ backend.
- High-level near mathematical notation.
- Heavily relies on code generation. Used to be a DSL is turning into a library.
- Parallelized with MPI: `mpirun -n 4 python program.py`

Components

Internals of FEniCSx and externals [a]

DOLFINx The main code that we interface for building the linear system and handling boundary conditions. The outer most layer of FEniCSx

UFL The *Uniformed Form Language* is the Domain Specific language similar to the mathematical formulation of FEM.

FFCx The *Fenics From Compiler*(x) creates fast C code from the UFL formulations and tabulation from BASix

BASIx Handles parts of the FEM-backend related to elements and basis functions.

[a] numpy, petsc4py and mpi4py are most important ones for building simulations in FEniCSx on a moderate level

Solving Poisson's equation

```
# Define function space
P1 = element("Lagrange",msh.basix_cell(),1)
W = dolfinx.fem.functionspace(msh, P1)
# Define boundary conditions
facets = dolfinx.mesh.locate_entities_boundary(mesh,dim,
    domain)
dofs dolfinx.fem.locate_dofs_topological(W,1,facets)
bc1 = dolfinx.fem.dirichletbc(value, dofs, W)
bcs = [bc1, bc2]
# Define variational problem
u = ufl.TrialFunction(W);
v = ufl.TestFunction(W)
a = inner(grad(u), grad(v)) * dx
L = inner(f, v) * dx
# Compute solution
petsc_opt = {"..."}
solver = dolfinx.fem.petsc.LinearProblem(a, L, bcs,
    petsc_options=petsc_opt)
uh = solver.solve()
```

More Useful FEniCSx

- Maxwell's equations are vector valued:

```
V2 = ufl.VectorElement("Lagrange", ufl.tetrahedron, 2)
```

- Multiple types of elements are supported, for example "Nédélec" type elements[1].
- We can use `ufl.mixed_element()` when having a system depending on multiple variables, such as **E**, **B**.
- `dolfinx.fem.petsc.LinearProblem` can be split up into assembly, bc application, and solving.
- `SubDomains` can be created and used at the stage of meshing.

[1] https://defelement.com/

Mesh creation

Mesh creation in FEniCS GMSH

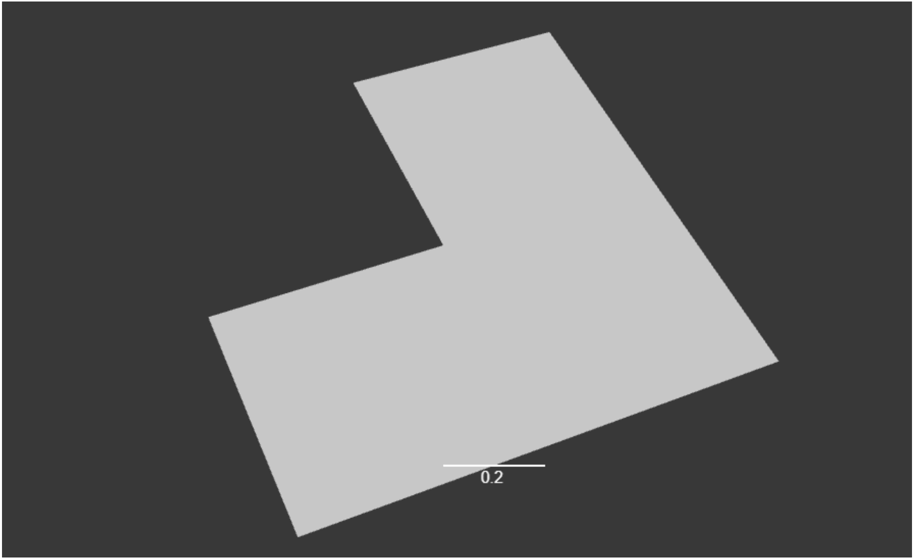
```
gmsh.initialize()
model = gmsh.model()
mesh_comm = MPI.COMM_WORLD
model_rank = 0
if mesh_comm.rank == model_rank:
    small_square = model.occ.addRectangle(0,0,0,0.5,0.5)
    large_square = model.occ.addRectangle(0,0,0,1,1)
    model_dim_tags = model.occ.cut([(2, large_square)],
                                    [(2, small_square)])
    model.add_physical_group(2, [large_square])
    model.occ.synchronize()
    model.mesh.generate(2)
msh, mt, ft =mshio.model_to_mesh(model,mesh_comm,model_rank,2)
```

stand alone GMSH[2], Salome[3].

[2] https://gmsh.info/

[3] https://docs.salome-platform.org/latest/main/index.html

Generated Mesh



Visualization

- I suggest you use PyVista in google Colab

Visualization

- I suggest you use PyVista in google Colab
- Similarly for local use I recommend Paraview
  - Very powerful
  - Scriptable with python
  - use "*xdmf"

```
with io.XDMFFile(msh.comm, "output/poisson.xdmf","w"
    ) as file:
    file.write_mesh(msh)
    file.write_function(uh)

```

Visualization

- I suggest you use PyVista in google Colab
- Similarly for local use I recommend Paraview
  - Very powerful
  - Scriptable with python
  - use "*xdmf"

```
with io.XDMFFile(msh.comm, "output/poisson.xdmf","w"
    ) as file:
    file.write_mesh(msh)
    file.write_function(uh)

```

- A common trick is to project the solution onto a more refined mesh but lower order function space for visualization.

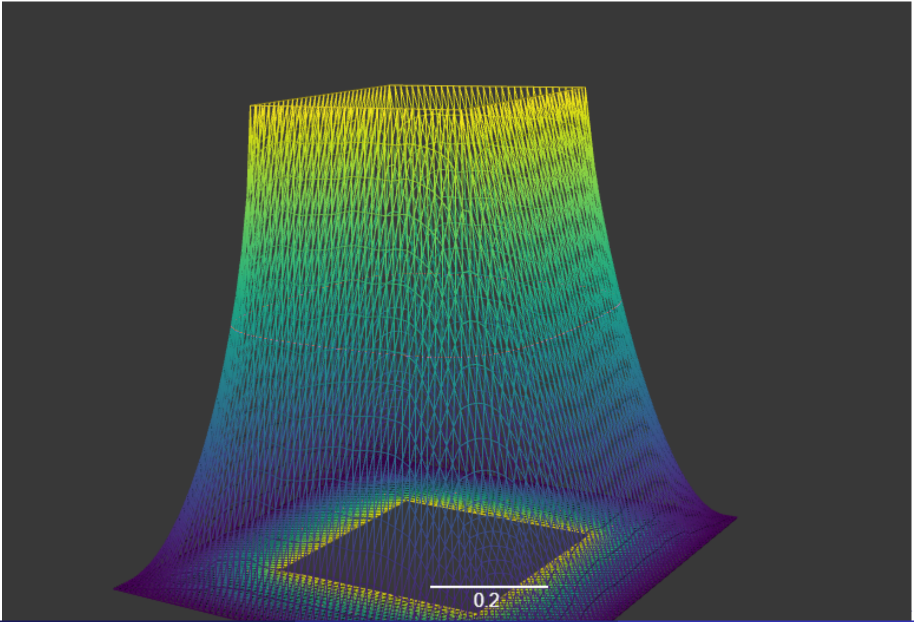
Visualization with PyVista

PyVista plot of Function uh on mesh `msh`.

```
dim = msh.geometry.dim-1
topology, cell_types, geometry = plot.vtk_mesh(msh, dim)
grid = pyvista.UnstructuredGrid(topology, cell_types,
    geometry)
grid.point_data["u"] = uh.x.array.real
grid.set_active_scalars("u")
plotter = pyvista.PlotterITK()
plotter.add_mesh(grid)
warped = grid.warp_by_scalar()
plotter.add_mesh(warped)
plotter.show()

#Generate a HTML file that can be downloaded
plotter.export_html('pyvista.html')
```

Visualization



Viz in Colab

Colab now needs the following solution based on **panel** where panel_plotter is used in the same way as above but rendered in a panel.

```
import pyvista as pv
import numpy as np
import panel as pn

pv.set_jupyter_backend('trame')
pn.extension("vtk")

panel_plotter = pv.Plotter(notebook=True)
panel_plotter._on_first_render_request() #can be buggy
pn.panel(
    panel_plotter.render_window, orientation_widget=
   panel_plotter.renderer.axes_enabled,
    enable_keybindings=False, sizing_mode="stretch_width",
)
```

Links

- FEniCSx docs for python API
  https://docs.fenicsproject.org/dolfinx/v0.5.1/python/

Links

- FEniCSx docs for python API
  https://docs.fenicsproject.org/dolfinx/v0.5.1/python/
- defelemts
  https://defelement.com/

Links

- FEniCSx docs for python API
  https://docs.fenicsproject.org/dolfinx/v0.5.1/python/
- defelemts
  https://defelement.com/
- FEM on Colab : FEniCSx, FEniCS, Firedrake, GMSH, ...
  https://fem-on-colab.github.io/

Appendix A: Hat functions

Linear Lagrange basis functions, P1 element in 1D or Hat functions. For equidistant discretization.

$$\phi_j = \begin{cases} \frac{x-x_{j-1}}{h}, & x \in [x_{j-1}, x_j] \\ \frac{x_{j+1}-x}{h}, & x \in [x_j, x_{j+1}] \\ 0, & \text{otherwise} \end{cases}, \quad \phi_j' = \begin{cases} \frac{1}{h}, & x \in [x_{j-1}, x_j] \\ -\frac{1}{h}, & x \in [x_j, x_{j+1}] \\ 0, & \text{otherwise} \end{cases}. \tag{6}$$

$$A_{ij}\mathbf{x}_j = \mathbf{b}_i \tag{7}$$