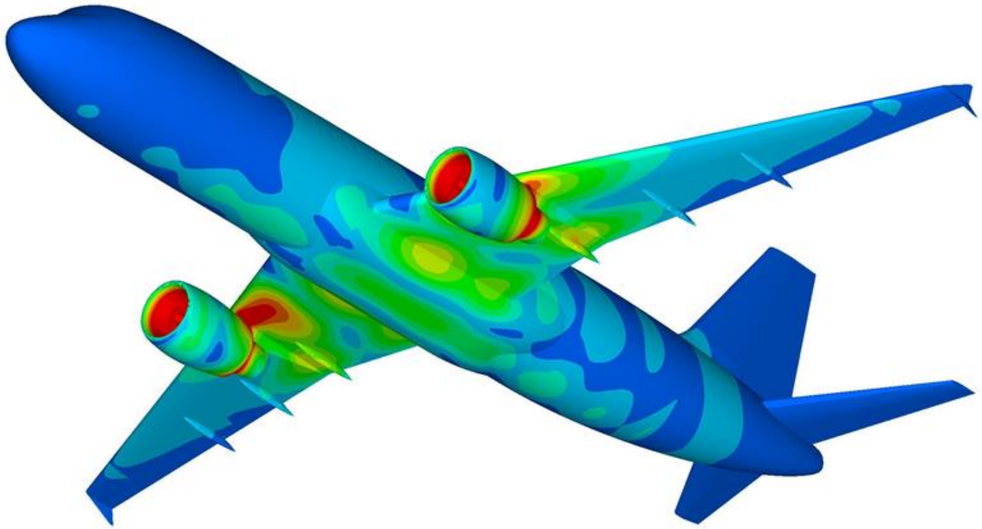
Worked examples solving steady and time-evolving PDEs in FEniCS

Jakub Köry
Research Associate
jakub.koery@glasgow.ac.uk

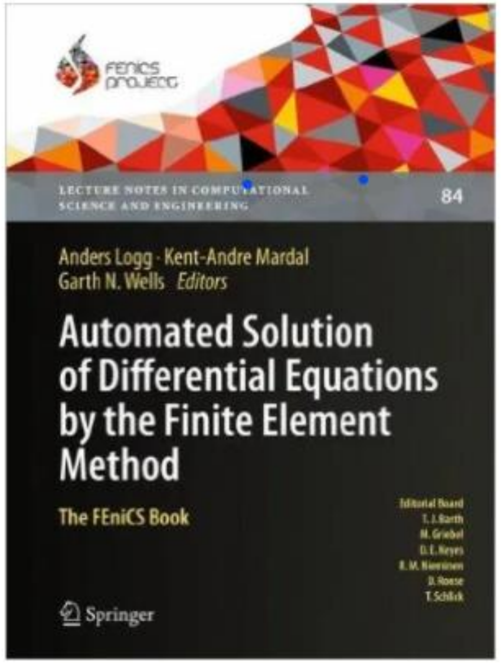
What is FEniCS?

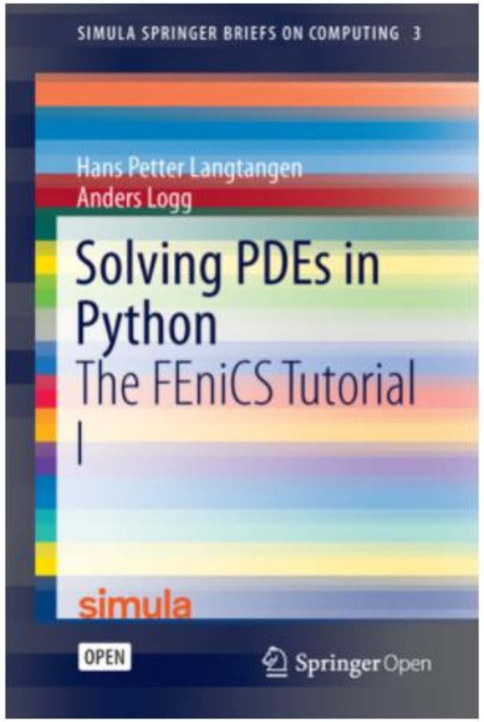
- Freely-available, open-source, automated computing environment for solving PDEs using FE method
  - https://fenicsproject.org/
- Organized as a collection of components
  - Dolfin, FFC, UFL, FIAT, UFC
  - https://fenicsproject.org/citing/
- Runs on a multitude of platforms
  - Windows, **Linux**, Anaconda, Docker
  - https://fenicsproject.org/download/
- High-level **Python** and C++ interfaces
- Very intuitive
- Large community of users



FEniCS documentation

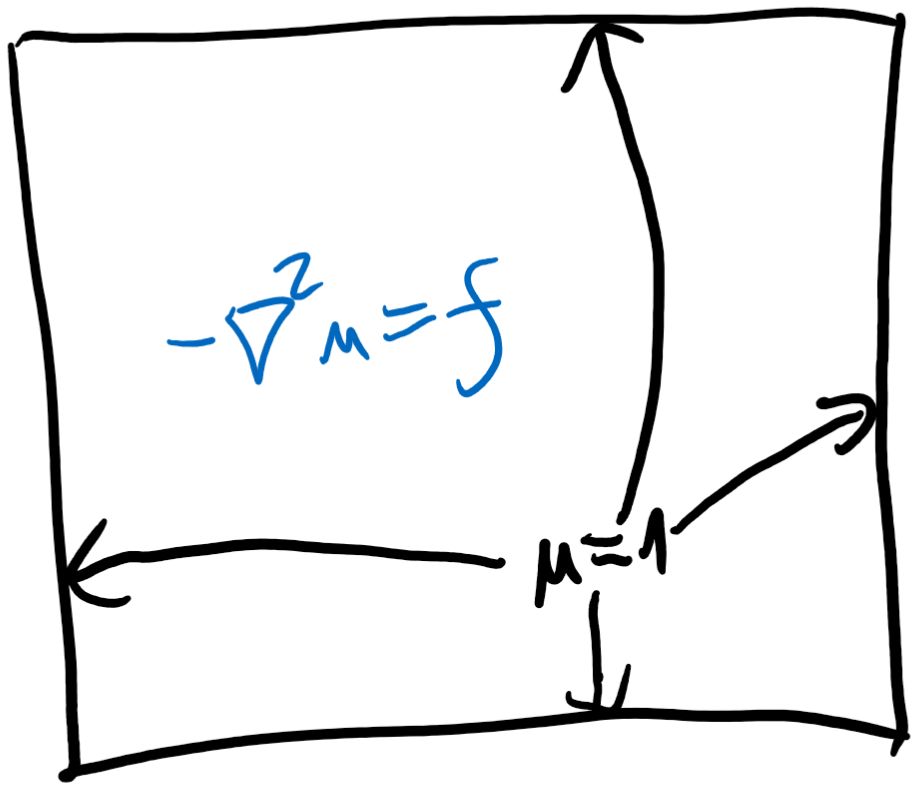
- Well-documented
  - https://fenicsproject.org/documentation/
- The FEniCS book
  - "Automated solution of Differential Equations by the Finite Element Method" (2012)
  - Very detailed (~700 pages) presentation of the theory, software components and applications
  - Some examples might use outdated interfaces
  - First chapter still worth reading (~70 pages)
- **FEniCS Tutorial**
  - "Solving PDEs in Python" (2017)
  - Great for new users who prefer Python
  - Explains
    - installation
    - the fundamentals of FEM
    - FEniCS programming
    - how to quickly solve a range of PDEs





Simple example: Poisson equation in 2D

- Based on https://github.com/hplgit/fenics-tutorial/blob/master/pub/python/vol1/ft01_poisson.py
- One can choose arbitrary (e.g. polynomial) functional form u_D for the Dirichlet BC and then select the forcing f so that u_D satisfies the Poisson equation
- Analytic solution is then at hand which allows one to test the accuracy of the numerical solution
- Instead, we choose Dirichlet BC ($u_D = 1$) on the entire boundary of a unit square
- f=10 for heating, f=-10 for cooling



Poisson equation in 2D (implementation)

PoissonSquare.py (in Notepad++)

- Import all necessary functionality
- Create a mesh and define the appropriate function space (without considering BCs)
  - ‘P’ for standard Lagrange family of elements
  - 1 for the degree of the finite element
- Define the Dirichlet BC
- Set up the variational problem
- Weak formulation (Peter, Namshad): $\int_{\Omega} \nabla u \cdot \nabla v \, dx = \int_{\Omega} f v \, dx$
- Solve the problem
- Post-processing
- *In terminal: “python PoissonSquare.py”*

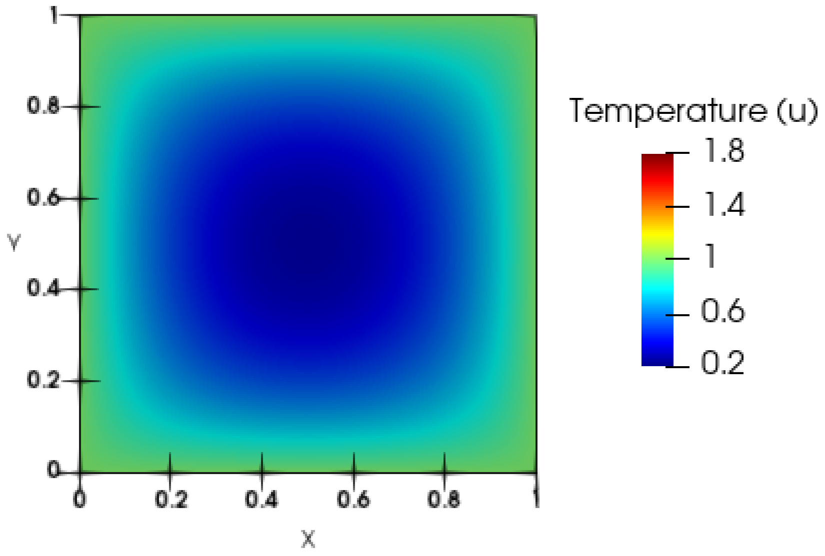
```
from fenics import *
import matplotlib.pyplot as plt

# Create mesh and define function space
mesh = UnitSquareMesh(50, 50)
V = FunctionSpace(mesh, 'P', 1)

# Define boundary condition
u_D = Expression('1', degree=1)

def boundary(x, on_boundary):
    return on_boundary

bc = DirichletBC(V, u_D, boundary)

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(10.0) # -10.0 for cooling
a = dot(grad(u), grad(v))*dx
L = f*v*dx

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

# Plot solution and mesh
plot(u)
plot(mesh)

# Save solution to file in VTK format
vtkfile = File('poissonSimplest/solution.pvd')
vtkfile << u

# Hold plot
plt.show()
```

Post-processing

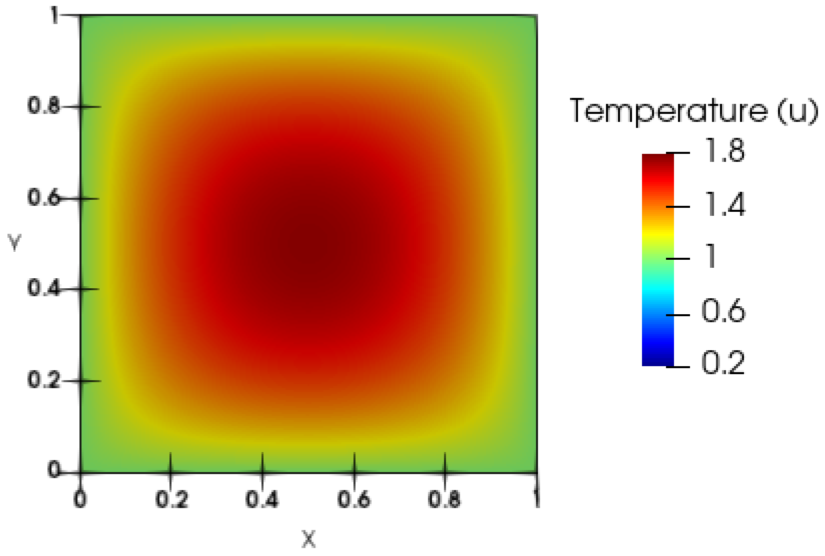
- Direct plotting using matplotlib.pyplot
- Export Vtk file
  - use Paraview/Visit to visualize it

```python
# Plot solution and mesh
plot(u)
plot(mesh)

# Save solution to file in VTK format
vtkfile = File('poissonSimplest/solution.pvd')
vtkfile << u

# Hold plot
plt.show()
```

Cooling (f=-10)



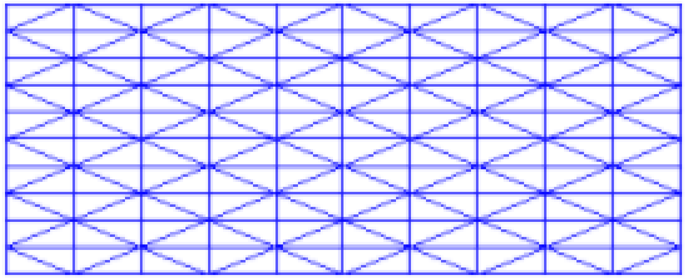
Heating (f=10)



Mesh generation on simple domains

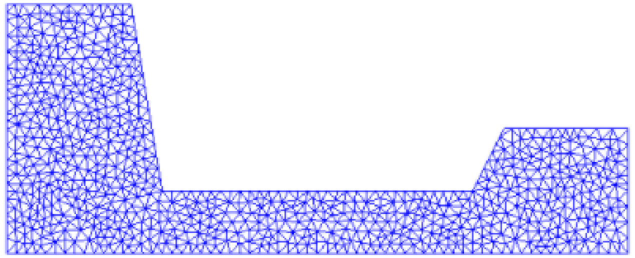
- Simple built-in meshes
  - https://fenicsproject.org/olddocs/dolfin/1.3.0/python/demo/documented/built-in_meshes/python/documentation.html
  - UnitIntervalMesh, UnitSquareMesh, RectangleMesh, UnitCircleMesh, UnitCubeMesh, BoxMesh

```
mesh = RectangleMesh(-3.0, 2.0, 7.0, 6.0, 10, 10, "right/left")
plot(mesh, title="Rectangle (right/left)")
interactive()
```



- Generate polygonal (2D) and polyhedral (3D) meshes
  - https://fenicsproject.org/olddocs/dolfin/1.3.0/python/demo/documented/mesh-generation/python/documentation.html
  - PolygonalMeshGenerator, PolyhedralMeshGenerator

```
# Create empty Mesh
mesh = Mesh()

# Create list of polygonal domain vertices
domain_vertices = [Point(0.0, 0.0),
                   Point(10.0, 0.0),
                   Point(10.0, 2.0),
                   Point(8.0, 2.0),
                   Point(7.5, 1.0),
                   Point(2.5, 1.0),
                   Point(2.0, 4.0),
                   Point(0.0, 4.0),
                   Point(0.0, 0.0)]

# Generate mesh and plot
PolygonalMeshGenerator.generate(mesh, domain_vertices, 0.25);
plot(mesh, interactive=True)
```



Advanced meshing: Importing a mesh

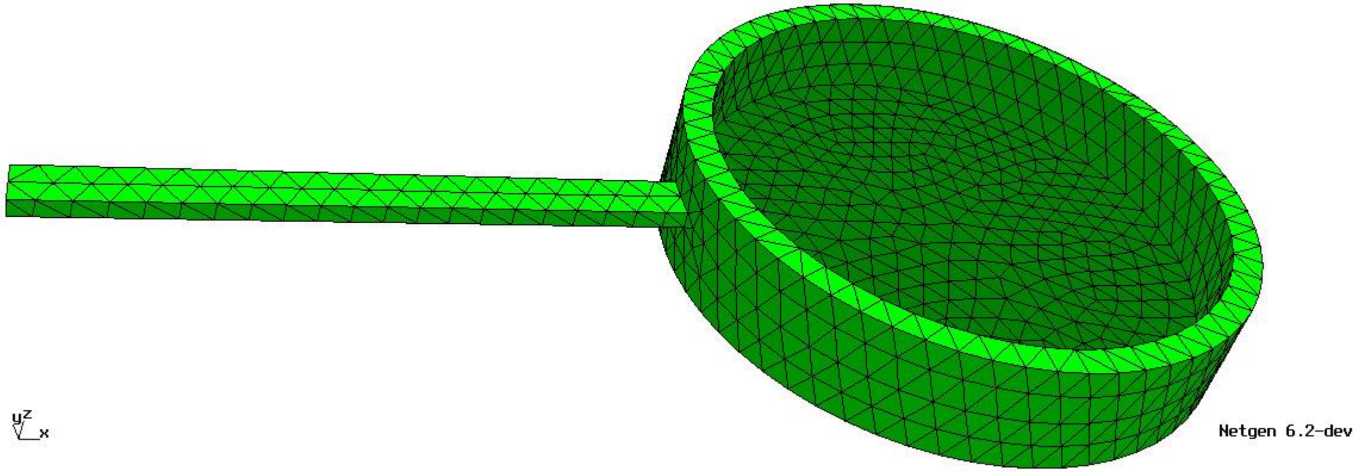
- NetGen
  - Write a .geo file (describing the geometry of our domain using simple primitives - e.g. cylinders - and Boolean operations - complement, union and intersection)
  - Use [NetGen] to generate the meshed domain with a specified maximal mesh size, in the form of .msh file
  - Use [DOLFIN-CONVERT] (a python script) to convert the .msh file into an .xml file, which can be used as an input for FEniCS

"In terminal: dolfin-convert Pan.msh Pan.xml"

- Other external mesh generators (Liuyang)
  - [Gmsh], [TetGen], …

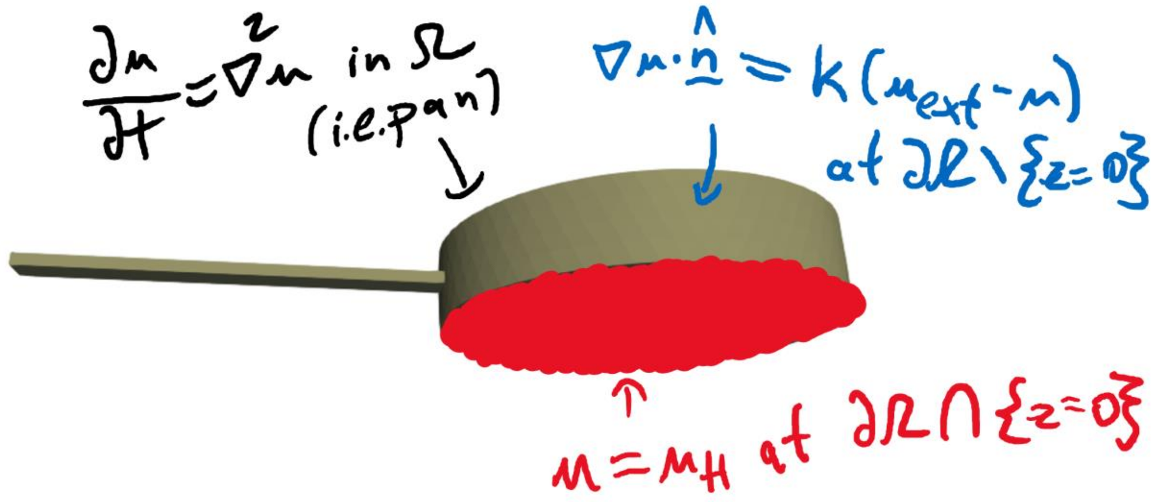
Pan.geo

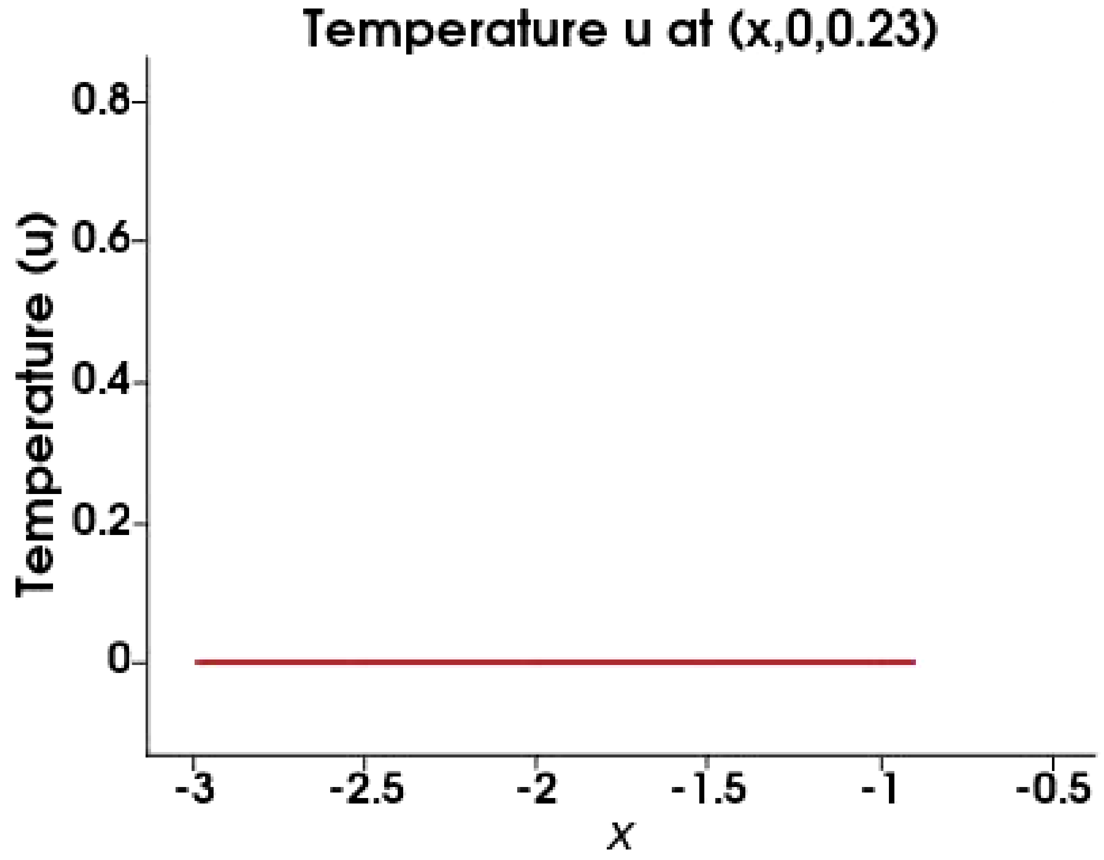
```
algebraic3d

#
# Generates a pan/pot
#

solid BiggerCylinder = cylinder ( 0, 0, 0.4; 0, 0, 0; 1.0 )
            and plane (0, 0, 0.4; 0, 0, 1) and  plane (0, 0, 0; 0, 0, -1);

solid SmallerCylinder = cylinder ( 0, 0, 0.4; 0, 0, 0.1; 0.9 )
            and plane (0, 0, 0.4; 0, 0, 1) and  plane (0, 0, 0.1; 0, 0, -1);

solid Handle = orthobrick (-3,-0.08,0.17;-0.5,0.08,0.23);

solid Pan = (BiggerCylinder or Handle) and not SmallerCylinder -maxh = 0.1;

tlo Pan;
```



Time-evolving PDE

- Thermal conduction in a 3D pan
    - Based on https://github.com/hplgit/fenics-tutorial/blob/master/pub/python/vol1/ft03_heat.py
- Demonstrates advantages of FE over FD
    - domain geometry is nontrivial (irregular)
    - possible non-smoothness at the joint
- Mixed boundary conditions
    - Dirichlet BC at the bottom of the pan
    - Fourier’s law (heat transfer) – Robin (third type) BC – at remaining boundary parts



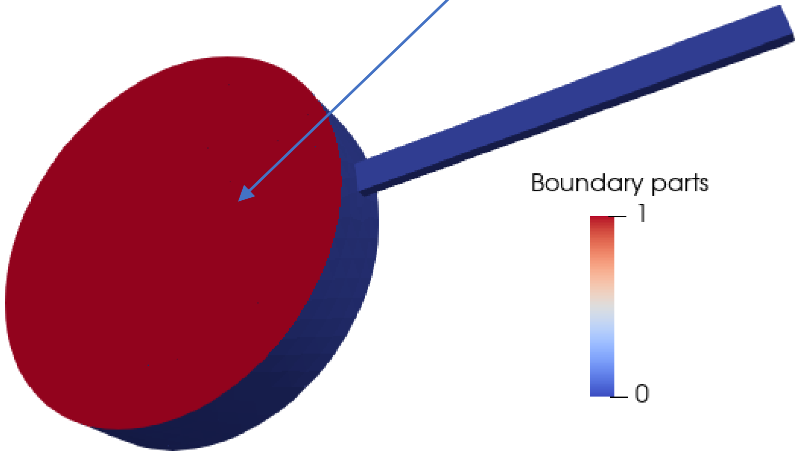


Importing the mesh and identifying the boundary parts

HeatOnPan.py (1st part)

- Import all necessary functionality and define the key parameters
- Import mesh and define the appropriate function space
- Define and mark distinct boundary parts
  - Set Dirichlet BC at the bottom

```python
from fenics import *

T = 0.5              # final time
num_steps = 100      # number of time steps
dt = T / num_steps # time step size
K=1.0          # parameter K; try 0.1 too
uExt = 0 # external temperature u_{ext}

# Create mesh and define function space
mesh = Mesh('pan.xml')
V = FunctionSpace(mesh, 'P', 1)
# Define boundary condition
boundary_parts = MeshFunction("size_t", mesh, mesh.topology().dim()-1)
file0= File("boundary_parts.pvd")
boundary_parts.set_all(0)
tol = 1e-10 # tolerance for coordinate comparisons

class BottomDirichletBoundary(SubDomain):
  def inside(self, x, on_boundary):
    return on_boundary and abs(x[2]) < tol
# add "and (x[0]*x[0]+x[1]*x[1]<0.25)" if flame is smaller than pan

BottomDirichletBoundary().mark(boundary_parts, 1)
file0 << boundary_parts
u_D = Constant(1.0) # boundary value u_H imposed at z=0
bc = DirichletBC(V, u_D, boundary_parts, 1)
# Redefine boundary integration measure
dss = Measure('ds', domain=mesh, subdomain_data=boundary_parts)
```



Heat equation with mixed BCs: weak formulation

- Centering at time t_{n+1}, manually discretize the time derivative (backwards Euler):

$$\frac{u_{n+1} - u_n}{\Delta t} = \nabla^2 u_{n+1}$$

- Rearrange, multiply by test function and integrate to get:

$$\int_\Omega u_{n+1} v\, dx \; - \Delta t \int_{\partial\Omega} (\nabla u_{n+1} \cdot \hat{\boldsymbol{n}}) v\, dS + \Delta t \int_\Omega (\nabla u_{n+1} \cdot \nabla v)\, dx - \int_\Omega u_n v\, dx = 0$$

- Use Robin BC:

$$\int_\Omega u_{n+1} v\, dx + K\Delta t \int_{\partial\Omega \backslash \{z=0\}} (u_{n+1} - u_{ext}) v\, dS + \Delta t \int_\Omega (\nabla u_{n+1} \cdot \nabla v)\, dx - \int_\Omega u_n v\, dx = 0$$

- From known u_n, calculate u_{n+1}

- *In terminal: “python HeatOnPan.py”*

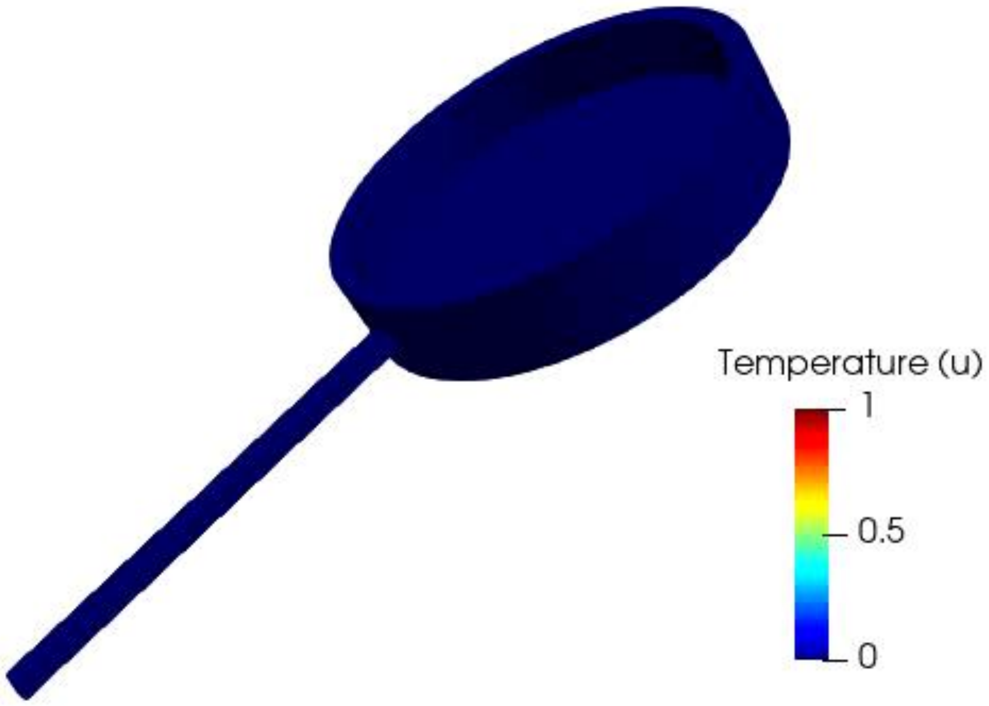
- *One can easily compute mean temperature, flux through boundary,…*

HeatOnPan.py (2nd part)

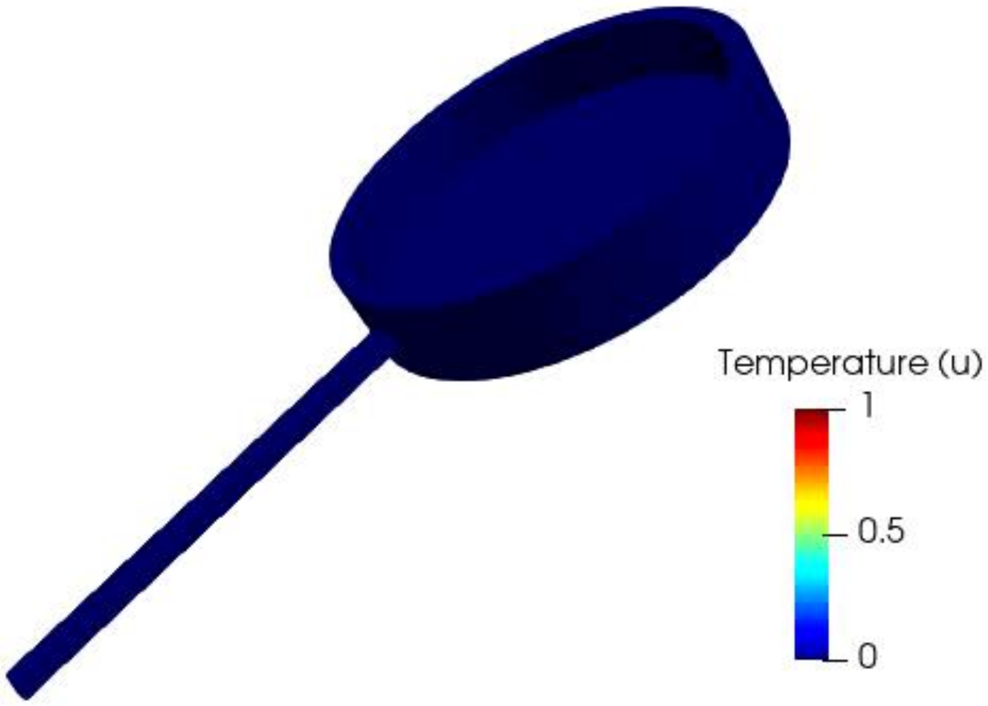
```

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
u_n = interpolate(Constant(0.0), V) # Define initial value
F = u*v*dx + dt*K*(u-uExt)*v*dss(0) + dt*dot(grad(u), grad(v))*dx - u_n*v*dx
a, L = lhs(F), rhs(F)
# Create VTK file for saving solution
vtkfile = File('HeatedPanWithExternalCooling/solution.pvd')
u = Function(V)
t = 0 # Time-stepping

for n in range(num_steps):
    # Compute solution
    solve(a == L, u, bc)
    # Save solution at this time, update previous solution and time
    vtkfile << (u_n, t)
    u_n.assign(u)
    t += dt
```

Post-processing in Paraview – the effect of K

K=1

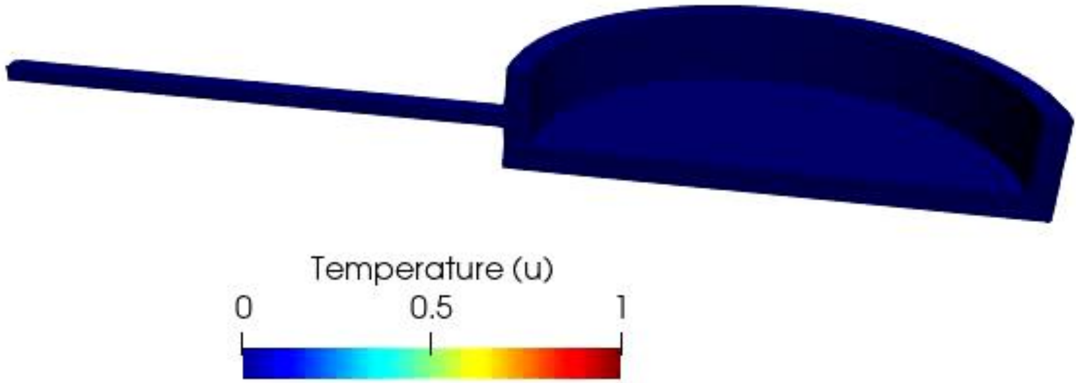


K=0.1

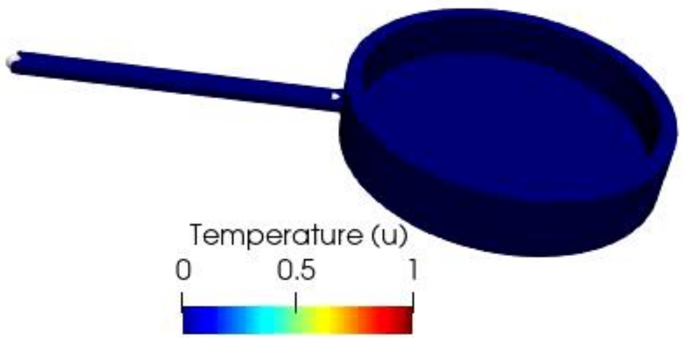


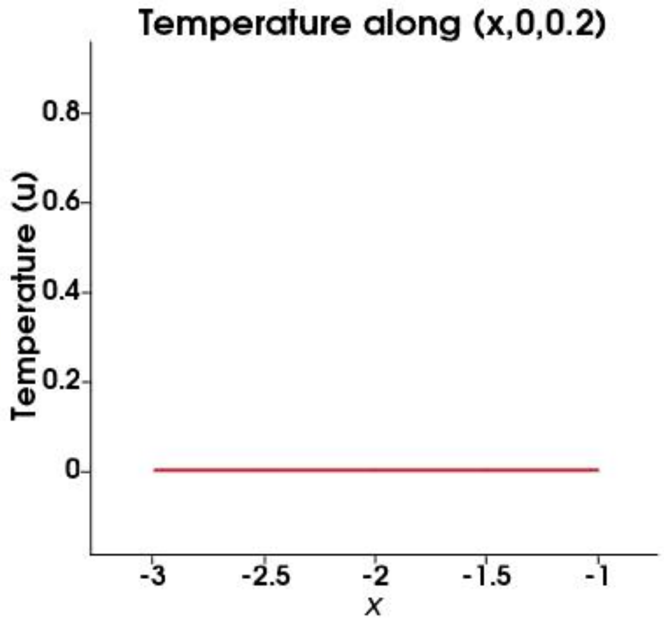
Clipped domain and plotting along a line in Paraview

Clipped domain



Plot along a line





Easy extensions (references w.r.t. *Solving PDEs in Python*)

- Our heat equation example was based on Section 3.1
- Two materials with different (thermal) properties - Section 4.3
- Nonlinear problems – Section 3.2
- Advection-reaction-diffusion equations – Section 3.5
- Systems of PDEs
  - Elasticity – Section 3.3 (Hao)
  - Navier-Stokes – Section 3.4
- Many others on https://fenicsproject.org/tutorial/

Thank you for your attention

If you have any questions, please do not hesitate to contact me at

jakub.koery@glasgow.ac.uk