PYTOTUBE: DUAL PHSYICS SOLVER FROM SCRATCH

With (Scientific) Python Language

Why Python?

- Extremely powerful
  - Numerical Computation (numpy)
  - Visualizing
  - Parallel processing (including CUDA, and OpenCL)
  - Image processing
  - GUI development
  - Interactive Computing lik (and better than) MATLAB with IPython.

Why Python?

- It is developer (scientist) friendly BECAUSE:
  - You feel very comfortable as developing it MATLAB.
  - Interpreted, NOT compiled. You don’t need to compile and link. **Just code and run.** Developing an extremely computationally intensive programs is better in Assembly but DO NOT worry because **Python is a powerful glue** language to integrate extensions written in other languages.
- Portable (Runs both Windows and Linux without pain.
- Modern
  - Object oriented
  - Well-suited to modern-day application (unit testing, GUI development, readable coding, OOP, etc.)

APPLICATION DESIGN

And planning according to future plans

Programming Requirements

- Modular
- Easy Extensible
- Object oriented
- Reusable so Well documented

Short Term Scientific Requirements

- **2D Transient** Laplace Solver for Velocity and Temperature Fields
- Dual physics (Heat and Flow) and Free/Forced Convection in 2D
- Simple **Optimizations**
- **Polar** / Cylindrical Coordinates Support
- Simple **Radiation** modelling
- Essential heat and flow boundary-initial conditions modelling
- Simple **couplings**
- **Sparse** matrices for performance

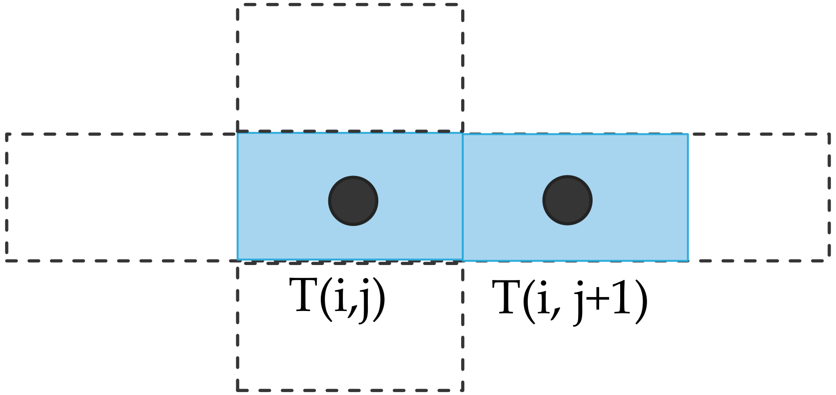
Long Term Requirements

- Geometry/Mesh import from common formats (Paraview, Fluent, Gridgen)
- Automatic meshing with **Netgen Lib**
- Switching from **Finite Differences** to **Finite Elements**.
- Extensive Radiation modelling
- Advanced boundary-initial condition input. (polynomial, function, etc)
- **Turbulent** modelling
- **Topology optimization**

Finite Element Scheme

- Node based approach, finite differences method with node based scheme.
- In **Laplace2d** class, energy approach is used with cell based scheme. Cell based means that one temperature represents a cell.

1D node based approach



2D cell based approach

Application Design

- Classes must separated with the respect of scientific branches.

  Fluid mechanics / Math / Heat Transfer

- *Don't repeat yourself* law.

  For the maintainability and reusability.

```
class Heat1d(Laplace):
```

Heat1d is inherited from **Laplace** class, means **Heat1d** class owns all properties of **Laplace** class. **Laplace** class designed/planned as it would be used as any Partial Differential Equation in the form of Laplacian Equation.

Matrice Solving Method

- At now, we are not considering performance issues. So We solve the matrices by

  solve(A,B) or inv(A)*B

Which Boundary Conditions Implemented in Laplace Eqn.?

Steady State Laplace Equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

Dirichlet BC

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

$$\phi(x) = \phi$$

```
def dirichlet(self, index, Scalar):
```

Call with

```
EXAMPLE.dirichlet(index=0, Scalar=Value)   # or
EXAMPLE.dirichlet(0, Scalar=Value)      # or
EXAMPLE.dirichlet(0, Value)
```

Neumann BC

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

$$\left.\frac{\partial \phi}{\partial x}\right|_x = K$$

```
def neumann(self, index, Value):
```

Call with

```
    EXAMPLE.neumann(index=0, Value=99.0)        # or
    EXAMPLE.neumann(0, Value=99.0)              # or
    EXAMPLE.neumann(0, 99.0)
```

Which Boundary Conditions Implemented in 1D Heat Equation

Steady State 1D Heat Equation

$$\frac{\partial^2 T}{\partial x^2} = \frac{\dot{q}}{k}$$

Symmetry-Adiabatic / Constant Heat Flow

$$\frac{\partial^2 T}{\partial x^2} = \frac{\dot{q}}{k}$$

$$\left.\frac{\partial T}{\partial x}\right|_x = K$$

- If K=0 symmetry or adiabatic line.
- K>0 Fixed heat flow enters the Control Volume
- K<0 Fixed heat flow leaves the Control Volume

Symmetry / Constant Heat Flow

```
def Conduction(self, x, k):
```

Call with

```
    EXAMPLE.Conduction(x=0.0, k=-10.)
```

```
I don’t know why I named this method as ‘Conduction’ ☺
```

Future correct forms would be

```
def Symetry(self, x, k=0): #and
def FixedHeatFlow(self, x, k):
```

- **Heat1d.Conduction** method calls **Laplace1d.Neumann** method

Convection

$$-k\frac{\partial T}{\partial x} = \mathrm{h}(\mathrm{T} - \mathrm{T}_{\mathrm{ambient}})$$

```
def Convection(self, x, k, h, Ta):
```

Call with:

```
EXAMPLE.Convection(x=1.0, k=75., h=125.0,Ta=200.0)
```

Fixed Temperature

$$T(x) = T$$

```
def Temperature(self, x, T):
```

Call with:

```
EXAMPLE.Temperature(x=0.0, T=700.)
```

- **Heat1d.Temperature** method calls **Laplace1d.Dirichlet** method
- **I am planning add a feature that Temperature (Dirichlet) BC** in 2D, can be given a user defined/linear/polynomial/exponential functions of x and y. This should give more flexibility to model/approximate real situations in Thermal Analysis. In addition, **node by node BC** giving also is a good feature to model complex systems as simple problems.

Source

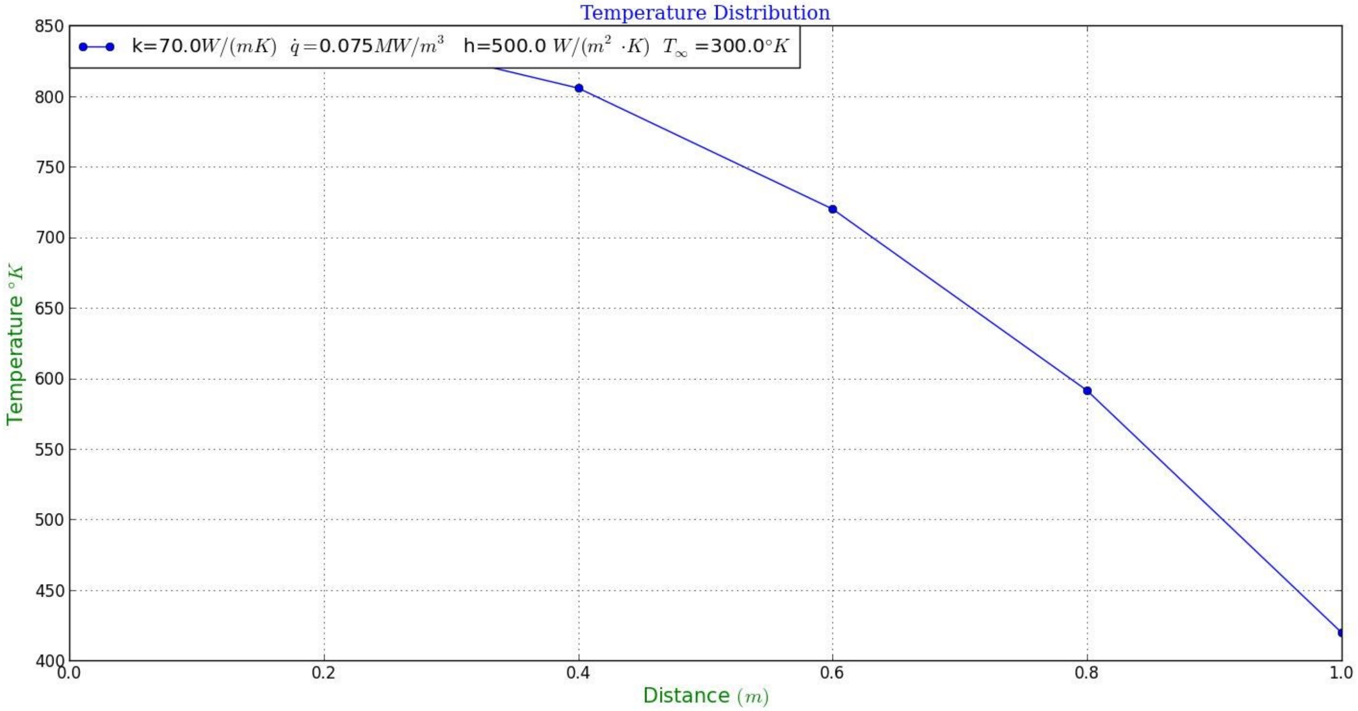
- An now, source method supposes that, heat generation is uniform and all elements generates heat. A feature must be added in 2D such that a source can be given a few nodes. It is more realistic than now.

```
EXAMPLE.source(k=WATER_K, Q=7.5e4)
```

A simple Example

```
from pytotube import Heat1d
Q = Heat1d(L=1.0, N=5)
WATER_K = 70.
Q.source(k=WATER_K, Q=7.5e4)
Q.Conduction(x=0.0,  k=0.)
Q.Convection(x=1.0, k=WATER_K, h=500., Ta=300.)
Q.solve()
Q.plot()
Q.ax.set(title=r'Temperature Distribution')
```

A simple Example



A simple Example

```
>>> Q.A
array([[  1.,  -1.,   0.,   0.,   0.,
0.],
       [  1.,  -2.,   1.,   0.,   0.,   0.],
       [  0.,   1.,  -2.,   1.,   0.,   0.],
       [  0.,   0.,   1.,  -2.,   1.,   0.],
       [  0.,   0.,   0.,   1.,  -2.,   1.],
       [  0.,   0.,   0.,   0.,  70., -
170.]])
```

```
>>> Q.B
array([[    0.       ],
       [  -42.85714286],
       [  -42.85714286],
       [  -42.85714286],
       [  -42.85714286],
       [-30000.      ]])
```

Analogy with Heat Transfer and Irrotational Flow

2d Potential Flow

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}$$

$$\frac{\partial \phi}{\partial x} = u$$

$$\frac{\partial \phi}{\partial y} = v$$

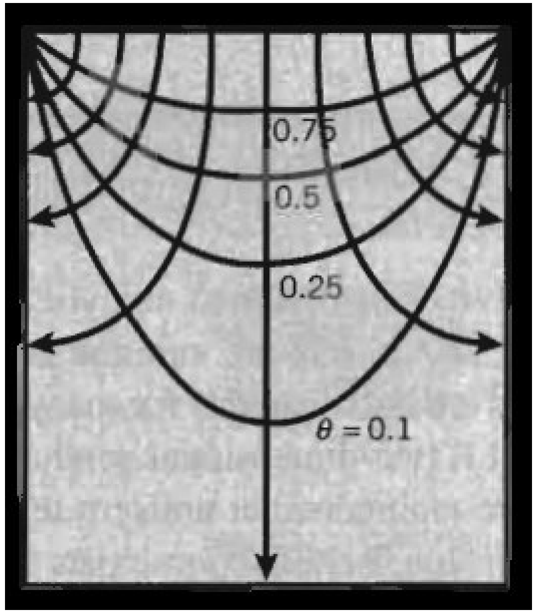
2d Heat Transfer

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

Convection Heat Transfer

$$q_x = -kA\frac{\partial T}{\partial x}$$

$$q_y = -kA\frac{\partial T}{\partial y}$$



Analogy with Heat Transfer and Irrotational Flow

2d Potential Flow

$$d\phi = \frac{\partial \phi}{\partial x} dx + \frac{\partial \phi}{\partial y} dy$$
$$= u\,dx + v\,dy$$

Along a constant ϕ we have $\mathrm{d}\phi = 0$

$$\mathrm{u}\,dx + v\,dy = 0 \text{ and } \frac{\mathrm{dy}}{\mathrm{dx}} = -\frac{u}{v}$$

2d Heat Transfer

This is similar to isothermal contour lines, and conservation of energy.

$$\mathrm{q}_x\,dx + q_y\,dy = 0$$

Analogy with Heat Transfer and Irrotational Flow

2d Potential Flow

These equations seems like $\frac{\partial p}{\partial x} \cong \frac{\Delta \mathrm{p}}{\ell}$

2d Heat Transfer

$$q_x = -kA\frac{dT}{dx} = \frac{kA}{L}(T_{s,1} - T_{s,2})$$

Thermal Resistance

$$R_{t,cond} = \frac{T_{s,1} - T_{s,2}}{q_x} = \frac{L}{kA}$$

Roadmap to Dual Physics

Solving simultaneously heat transfer and fluid flow

Method

1. Conservation of mass
2. Conservation of Energy
3. Conservation of Linear Momentum

Conservation of mass

$$\frac{\partial}{\partial t}\int_{cv}\rho d\mathcal{V} \approx \frac{\partial \rho}{\partial t}\delta x \delta y \delta z$$

$$\frac{\partial \rho}{\partial t}+\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)}{\partial y}+\frac{\partial(\rho w)}{\partial z}=0$$

$$\frac{\partial \rho}{\partial t}+\nabla \cdot \rho\vec{V}=0$$

Conservation of Energy (Energy Equation)

Conservation of Linear Momentum

Comments and Recommendations

- Laplace1d and Heat1d was only simple trial of solving Laplace equation with Finite Differences.
- And experimenting the difficulties developing a general use solver and algorithms with Python/Numpy.
- And picking a numerical scheme, numerical method, building a small application structure and drawing a roadmap for PDE solving.
- And comparing different approaches with engineering and numerical problems
  - Energy Approach / Finite Differences
  - Node based / Cell Based
- 2D solver is serious work because, 2D PDE result are more meaningful than 1D, and often more than 3D analysis.

Comments and Recommendations

To sum up, all of these works in 1D are only a meeting with:

- Numpy/Optimization Tools
- Numerical methods/schemes
- PDE solving Finite Elements/Volumes/Differences
- Problem solving and developing algorithm
- Developing a tested and documented scientific application with a software engineer approach.

2D Laplace Equation

Indexing

X

0	1	2	3	…	NX-1
NX	…	I × NX + J	…	…	2 × NX-1
(NY-1)×NX	…	…	…	…	NY × NX-1

Y