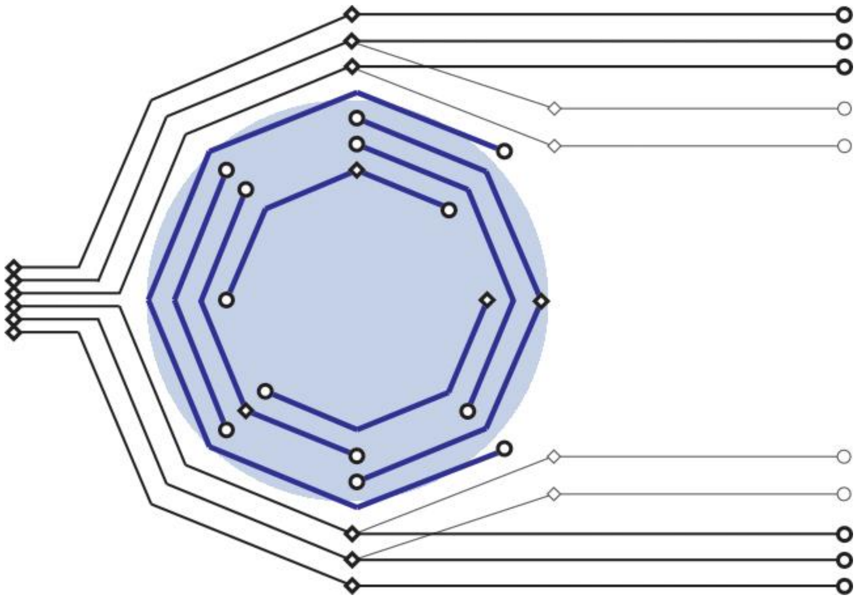


PINN Training

Adhika Satyadharma

Computational Fluid Dynamics Laboratory
National Taiwan University of Science and Technology

Outline

- Tasks
- How to run the code
- How to set the code
- Problem Description

Tasks

- **Code is available at:**
  https://github.com/NTUST-CFDlab/CFD_class_PINN

- **Tasks:**
  1. Burgers Equation (MANDATORY)
  2. Noise Elimination
  3. Error Estimation

  Only 1
  (You can choose only 1 of them)

RUN THE CODE

How to run the code

- You can run the code either in your **own machine** (can be a bit heavy) or **google colab** (https://colab.research.google.com/)(requires a google account)

Run in own machine

- **Install python**

  - Go to https://www.python.org/downloads/

  - Find the version you want to install (current suggestion: 3.12.10)

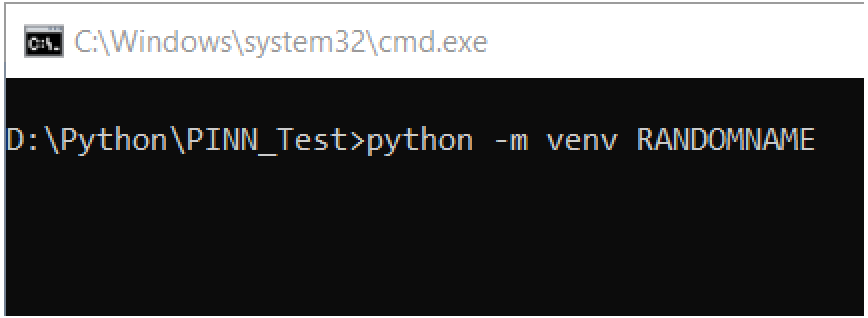
- **Open terminal/command prompt to create a venv (virtual environment)**

Commands:

python -m venv ANYNAME

or

python3 -m venv ANYNAME



*Make sure you are in the folder where you want to place your python script

Run in own machine

- **Activate venv**

```
D:\Python\PINN_Test>cd venv/Scripts

D:\Python\PINN_Test\venv\Scripts>activate.bat

(venv) D:\Python\PINN_Test\venv\Scripts>
```

(Windows)

or

```
D:\Python\PINN_Test\venv\Scripts>source activate
```

(Linux)

- **Install all the libraries:**

Commands:
pip install numpy; pip install matplotlib; pip install scipy; pip install tensorflow

- **Set & Run the code**

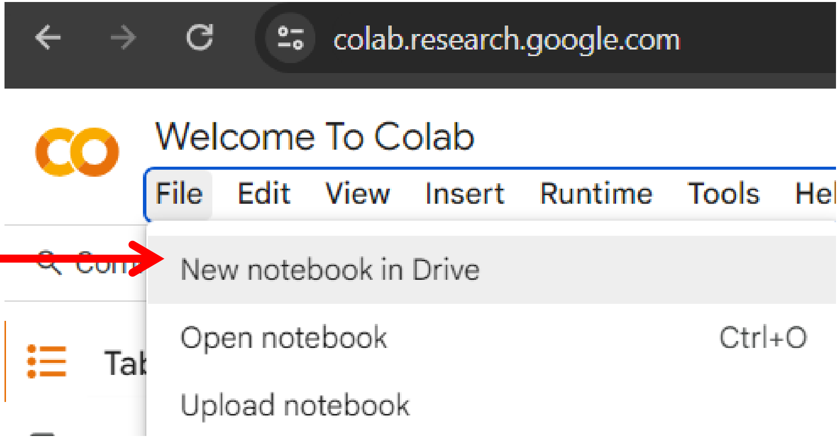
Commands: *python Main.py* or *python3 Main.py*

Run in google colab

- **Note:** It is highly recommended to sync it to a google drive so that you don't have to upload and download every time

- **Upload the files to goggle drive**
- **Create a new notebook**

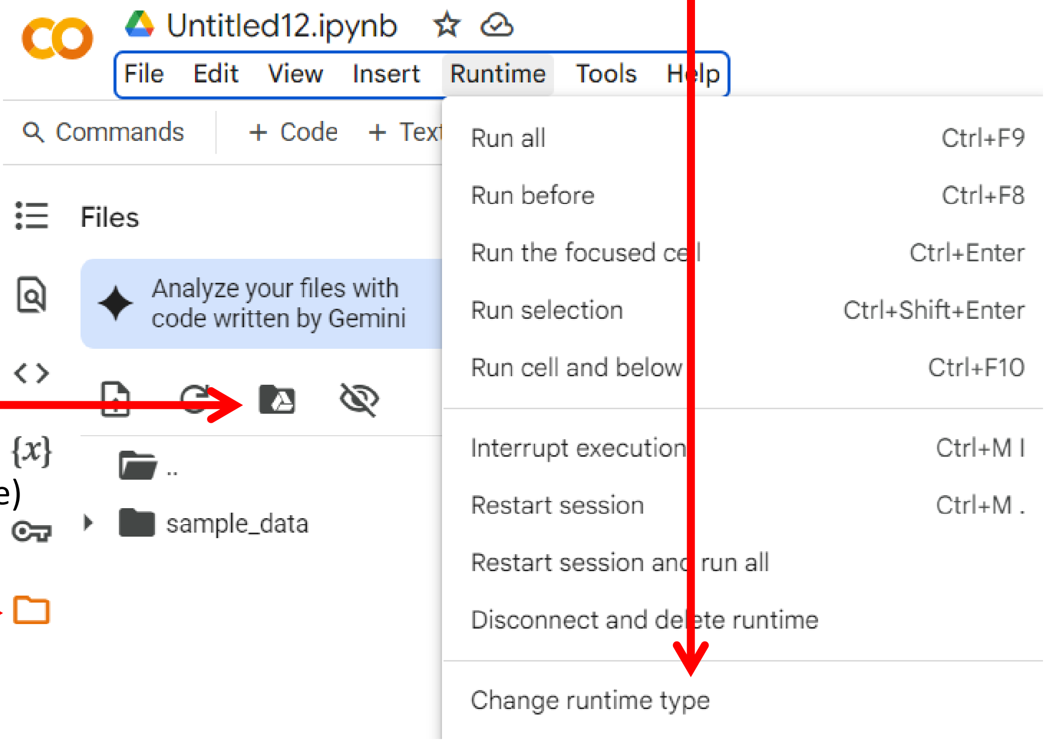
Choose this



Run in goggle colab

- **Sync to gdrive**
- **Change to GPU for run faster**

Change to GPU
(Goggle may limit GPU usage)



Sync with gdrive
(may take a while for the colab to process and let this option available)

See files
(May need to sync with gdrive or upload yourself)

Run in goggle colab

- If you are **NOT** working with google drive, then you must upload the files to colab (just drag and drop)

- If you are working with google drive, then you might need to change your current working directory. Just copy the code below to colab and run it (don’t forget to change the folder name, depending on what you name it in your gdrive)

```
import os
os.chdir('/FOLDER_NAME/')
```

*Depends on your folder name
*example: *os.chdir('/content/drive/My Drive/CFD_Class/PINN/Burgers/')*

Run in goggle colab

- **Set and run the code**

Run

Notes:

- Running is goggle is free, but there are some drawbacks
- If you don’t sync to gdrive, you may need to upload and download the simulation data everytime
- Goggle can limit GPU usage per day per account.
- If you are not active in the colab for about 30 minuets, goggle may stop your simulation or kick you out (files may be lost)
- If you sync with gdrive, you don’t need to be concerned with file lost as it is saved directly in the drive
- You don’t need a lot of storage for PINN, each simulation is mostly 50 MB, mostly due to pictures. The simulation result itself is very small

SET THE CODE

How to set the code

- All settings are located in *Case_Info.py*
- To run, the code, run *Main.py*
- Every other files contains the inner working of the PINN,
  which you can skip if you don’t want to go to the technical details

- **Note:**
  This code is not meant for learning, this version is to allow to test many different settings without the need to modify any part of the code except for *Case_Info.py*. While it does have high flexibility, it can be hard to read or understand how the code works. Also, the code is still messy.

What to set

- Set Governing Equation
- Set Domain size
- Set Neural Network
- Set Loss
- Set Points
- Set What to Plot
- Set Boundary Conditions

What to set

- Set Governing Equation

```
def Load_Equation_Info(self):
    Case_Name = "HELLO_PINN_WORLD"       # Any Name is okay
    Governing_Equation = "Burgers_1D"   # See Equation_Database.py to see the equation in more detail
    Equation_Constants = [0.01]         # Alpha

    return [Case_Name, Governing_Equation, Equation_Constants]
```

- All available equations (in *Equation_Database.py*)

```
def Get_Eq_Class(Equation_Set):
    if Equation_Set == "Burgers_1D":
        return Burgers_1D()
```

```
class Burgers_1D():
    def Equation_Info(self):
        Input_Names    = ["t", "x"]
        Output_Names   = ["u"]
        D1_Names       = ["u_t", "u_x"]
        Residual_Names = ["Residual"]
        Constant_Names = ["alpha"]
```

What to set

- Set domain size

```
def Load_Domain_Size(self):
    Total_Domain = [[0., 1.], [-1., 1.]]
```

The order depends on the governing equation.

```
class Burgers_1D():
    def Equation_Info(self):
        Input_Names    = ["t", "x"]
        Output_Names   = ["u"]
        D1_Names       = ["u_t", "u_x"]
        Residual_Names = ["Residual"]
        Constant_Names = ["alpha"]
```

[[0., 1.], [-1., 1.]]

$[t_{min}, t_{max}], [x_{min}, x_{max}]$

What to set

- Set Neural Network

```python
def Load_NN_Size(self):
    Layers  = 4
    Neurons = 64
```

*Larger network would be more expensive to train

- Set Boundary Conditions

```python
def Calc_BC(self, X, Code):
    # X[0] = t
    # X[1] = x
    if (Code == "Wall"):
        T = 0. * X[1]
    elif (Code == "Burger_Init"):
        T = np.sin(2. * np.pi * X[1])
    return [T]
```

$$T = 0$$

$$T = \sin(2\pi x)$$

What to set

- Set Points

```
def Load_Point_Gen_Info(self):

    #--------------------------------------------------------
    Scr_BP = [[] for i in range(3)] # Boundary
    Scr_DP = [[] for i in range(0)] # Data
    Scr_CP = [[] for i in range(1)] # Collocation Point
    #--------------------------------------------------------

    #["Gen", "Unif_Box", DOMAIN, POINTS, BC]
    Scr_BP[0] = ["Gen", "Unif_Box", [[0., 0.], [-1.,  1.]], [0, 400], "Burger_Init"]
    Scr_BP[1] = ["Gen", "Unif_Box", [[0., 1.], [-1., -1.]], [400, 0], "Wall"]
    Scr_BP[2] = ["Gen", "Unif_Box", [[0., 1.], [ 1.,  1.]], [400, 0], "Wall"]

    Scr_CP[0] = ["Gen", "Unif_Box", [[0., 1.], [-1., 1.]],  [40, 80]]
```

The domain is formatted like $[x_{min}, x_{max}]$, $[y_{min}, y_{max}]$ (Depends on the equation)

The points is formatted like   [points_x, points_y]

What to set

- Set Points

```
def Load_Point_Gen_Info(self):

    #---------------------------------------------------------
    Scr_BP = [[] for i in range(3)] # Boundary
    Scr_DP = [[] for i in range(0)] # Data
    Scr_CP = [[] for i in range(1)] # Collocation Point
    #---------------------------------------------------------

    #["Gen", "Unif_Box", DOMAIN, POINTS, BC]
    Scr_BP[0] = ["Gen", "Unif_Box", [[0., 0.], [-1.,  1.]], [0, 400], "Burger_Init"]
    Scr_BP[1] = ["Gen", "Unif_Box", [[0., 1.], [-1., -1.]], [400, 0], "Wall"]
    Scr_BP[2] = ["Gen", "Unif_Box", [[0., 1.], [ 1.,  1.]], [400, 0], "Wall"]

    Scr_CP[0] = ["Gen", "Unif_Box", [[0., 1.], [-1., 1.]],  [40, 80]]
```

Set the number based on the amount of set of points

This example has 3 set of boundary points and 1 set of collocation points

What to set

- Set Loss

```
def Load_Loss_Function(self):
    #Scr_LF    = ["Name", Weight, Set_of_Points, OutVar]
    Scr_LF     = [[] for i in range(2)]
    Scr_LF[0] = ["BC_D", 1., ["BC", 0, 1, 2], ["M", 0]]
    Scr_LF[1] = ["GE"  , 1., ["C", 0]       , ["R", 0]]
```

Set the number based on the amount of loss settings

- **The loss names:**

  BC_D = Boundary Loss

  GE = Governing Equation Loss

- **Weights** are by default 1

What to set

- Set Loss

```
def Load_Loss_Function(self):
    #Scr_LF    = ["Name", Weight, Set_of_Points, OutVar]
    Scr_LF     = [[] for i in range(2)]
    Scr_LF[0] = ["BC_D", 1., ["BC", 0, 1, 2], ["M", 0]]
    Scr_LF[1] = ["GE"  , 1., ["C", 0]        , ["R", 0]]
```

Set the number based on the amount of loss settings

- **The set of points are based on the points settings**

BC[0]
BC[1]
BC[2]

C[0]

```
Scr_BP[0] = ["Gen", "Unif_Box", [[0., 0.], [-1.,  1.]], [0, 400], "Burger_Init"]
Scr_BP[1] = ["Gen", "Unif_Box", [[0., 1.], [-1., -1.]], [400, 0], "Wall"]
Scr_BP[2] = ["Gen", "Unif_Box", [[0., 1.], [ 1.,  1.]], [400, 0], "Wall"]

Scr_CP[0] = ["Gen", "Unif_Box", [[0., 1.], [-1., 1.]],  [40, 80]]
```

What to set

- Set Loss

```
def Load_Loss_Function(self):
    #Scr_LF    = ["Name", Weight, Set_of_Points, OutVar]
    Scr_LF     = [[] for i in range(2)]
    Scr_LF[0] = ["BC_D", 1., ["BC", 0, 1, 2], ["M", 0]]
    Scr_LF[1] = ["GE"  , 1., ["C", 0]       , ["R", 0]]
```

Set the number based on the amount of loss settings

- The output var depends on the loss type.
  Boundary loss compares the main variable (M)
  GE loss compares the Residual (R)

- The index depends on the GE
  M[0] = *u*
  R[0] = Residual

```
class Burgers_1D():
    def Equation_Info(self):
        Input_Names    = ["t", "x"]
        Output_Names   = ["u"]
        D1_Names       = ["u_t", "u_x"]
        Residual_Names = ["Residual"]
        Constant_Names = ["alpha"]
```

What to set

- Set Output Result (PLOT)

```
def Load_Plot_Point_Info(self):
    Plot_Domain = [[[0., 1.], [-1.1, 1.1]]]
    Img_Size    = [[7, 3]]
```

The domain is formatted like $[x_{min}, x_{max}]$, $[y_{min}, y_{max}]$ (Depends on the equation)
The image size is the size of the image (currently it is 7 inch x 3 inch)

What to set

- Set Output Result (PLOT)

```
def Load_Image_Setting(self, Output_Filter):  # Contour
    # What variable to plot
    # See the ID Numbers in Equation_Database.py
    Main_Var_ID = [0]
    Res_ID      = [0]

    # Domain & Resolution
    Domain_Size   = [[0., 1.], [-1., 1.]]
    Sample_Points = [100,      200]

    # MinMax Values in the plot, you can also just set i
    MainVar_MinMax  = [[-1., 1.]]
    Residual_MinMax = [[-0.1, 0.1]]

    # Figure Size
    Img_Size = [[2.7, 4]]
```

This is like the settings in the loss

Domain size is formatted the same as others

Sampling is the resolution of the image

This is the minimum and maximum values for the image

PROBLEMS

Burgers equation

- This problem is the exact same as homework 2, but conducted in PINN (α_{Default} = 0.01)

- **GOAL:**
  Just solve it.
  *You can use any configuration you want (e.g. any network size, any amount of points), but do not change the loss function. Only the boundary and governing equation loss are allowed.

Consider the 1-D Burgers' equation in [-1, 1]

$$\frac{\partial u}{\partial t} + \frac{\partial uu}{\partial x} = \alpha \frac{\partial^2 u}{\partial x^2}$$

The initial condition is

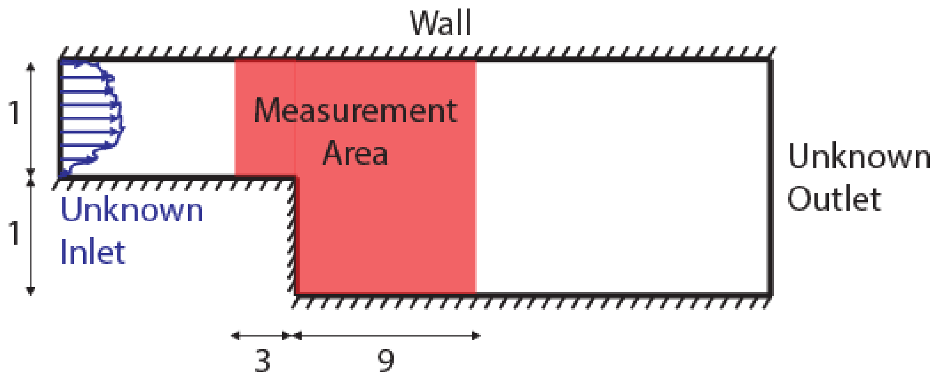
$$u(x, t = 0) = \sin(2\pi x)$$

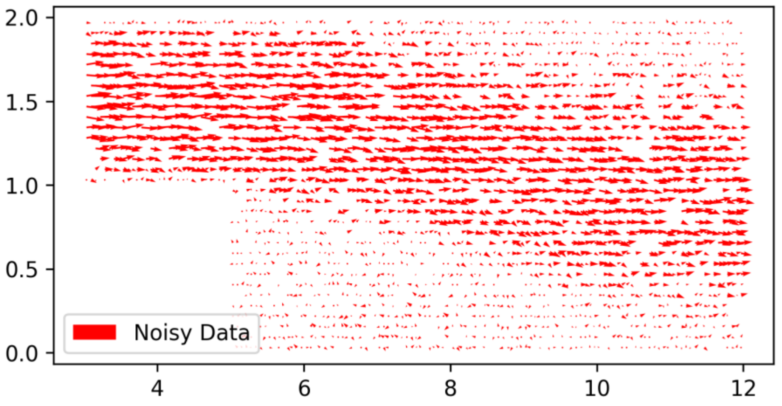
Boundary conditions are

$$u(x = -1, t) = 0, \quad u(x = 1, t) = 0$$

Denoise (Noise Elimination)

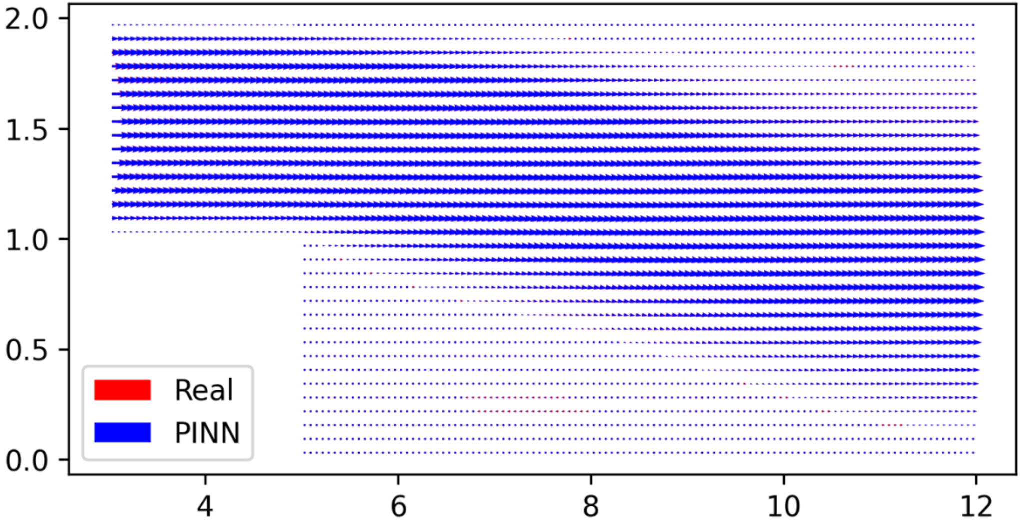
- Assume we have an experimental PIV (particle image velocimetry) data of a flow in a BFS (Backwards facing step) case. The measurement data is noisy and we want to eliminate the noise.





Denoise (Noise Elimination)

- **PINN Settings:**
  GE: NS_2D_SS (2 Dimensional, steady state, Navier-Stokes)
  $\rho = 1$, $\mu = 0.01$
  BC: Only the walls of the measurement area
  Loss: BC and NR (Noise reduction)

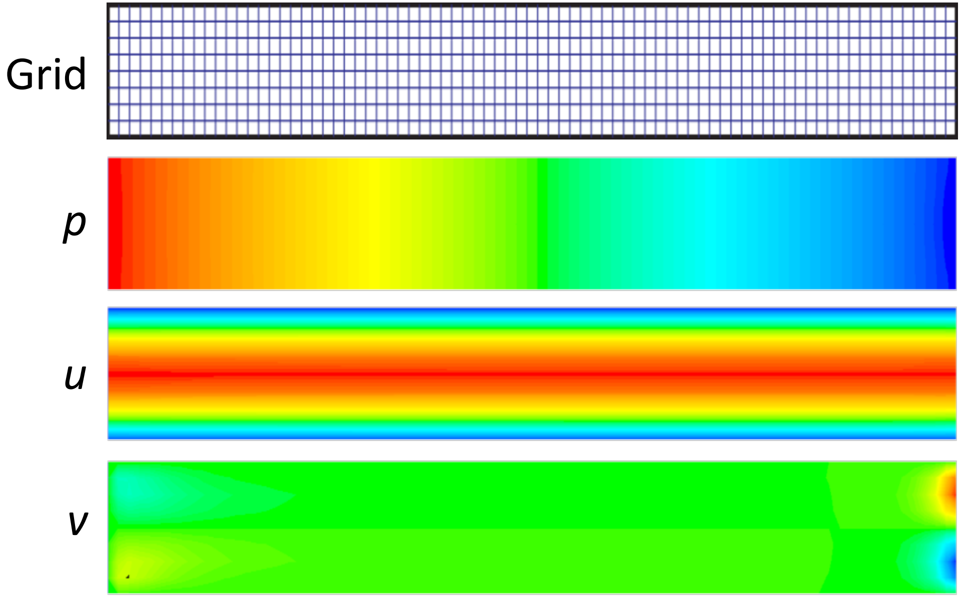


- **GOAL:**
  Set beta to get the best result
  *You can set a lot of beta directly and then compare all of the results.
  *Beta is the tolerance towards noise

Error Estimation

- We have conducted a Hagen-Pousille simulation using a commercial software Fluent.
- The grid is 80 x 8
- $\rho = 1$, $\mu = 0.01$

- **GOAL:**
  Find how accurate or inaccurate this simulation is.
- **Steps:**
  Simulate PINN with many β,
  Extrapolate the loss



Step 2. PINN Simulation

- **Network:**

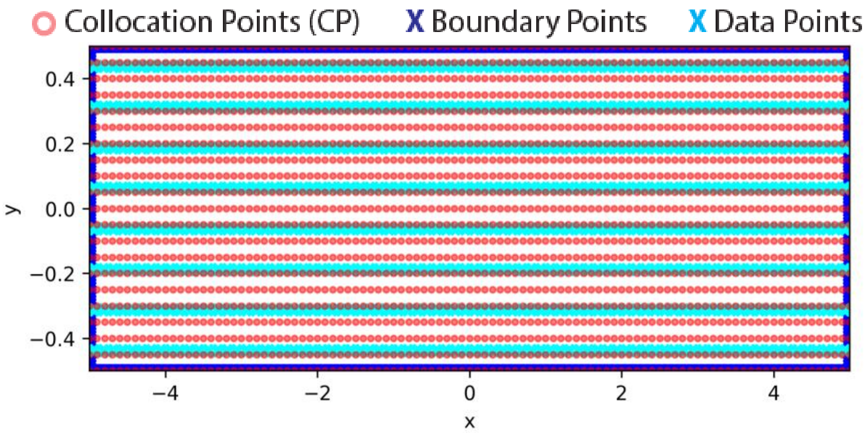
  Feed forward neural network,
  2 layers, each with 16 neurons

- **Loss:**
  Boundary (Dirichlet)
  Boundary (Neumann)
  Data (weighted with β)
  Governing equation

- **β :** {3e-4, 6e-4, 1e-3, 1.8e-3,
  3e-3, 6e-3, 1e-2, 3e-2, 0.1}

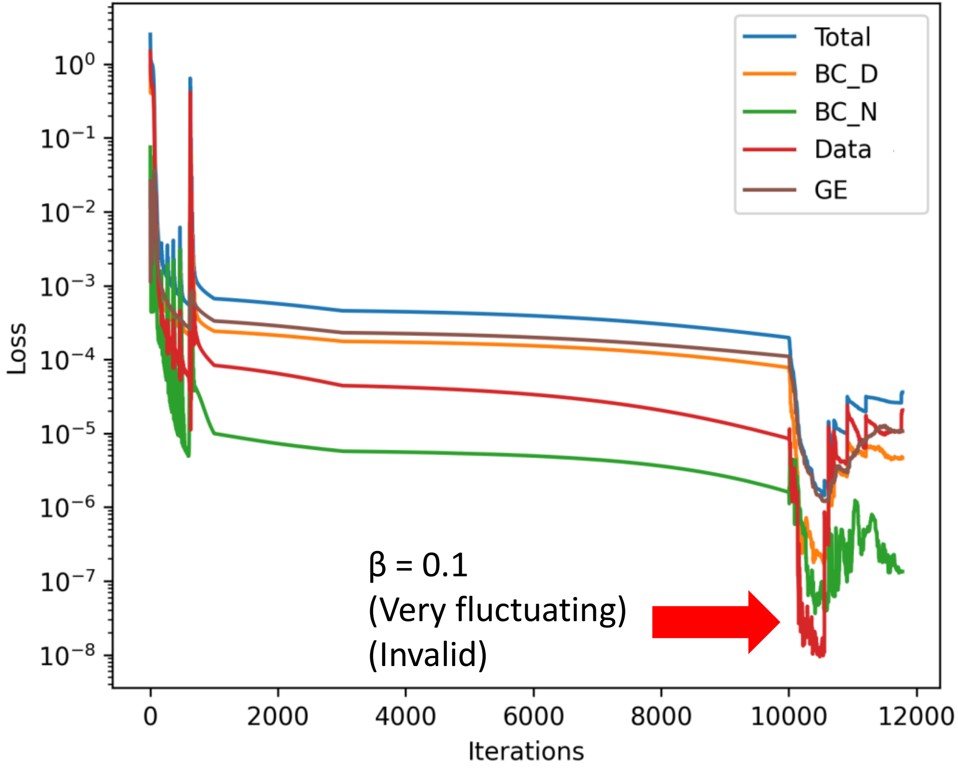
- **Points:**

  Collocation points: 101 x 21 = 2121
  Data points: 80 x 8 = 640
  Boundary points:
    Dirichlet: 500 x 2 + 100 = 1100
    Neumann: 100



Step 3. Analyze Loss

WARNING:
Your training result may look different. There is some randomness in the training that can lead to slightly different results.



- **Minimum estimate:**
  $\beta_{max} = 0.03$
  $L_2 @ \beta_{max} = 3.71 \times 10^{-4}$

L_2 Calculation
- Inverse L_M into L_D (numerically)
- Apply the formula:

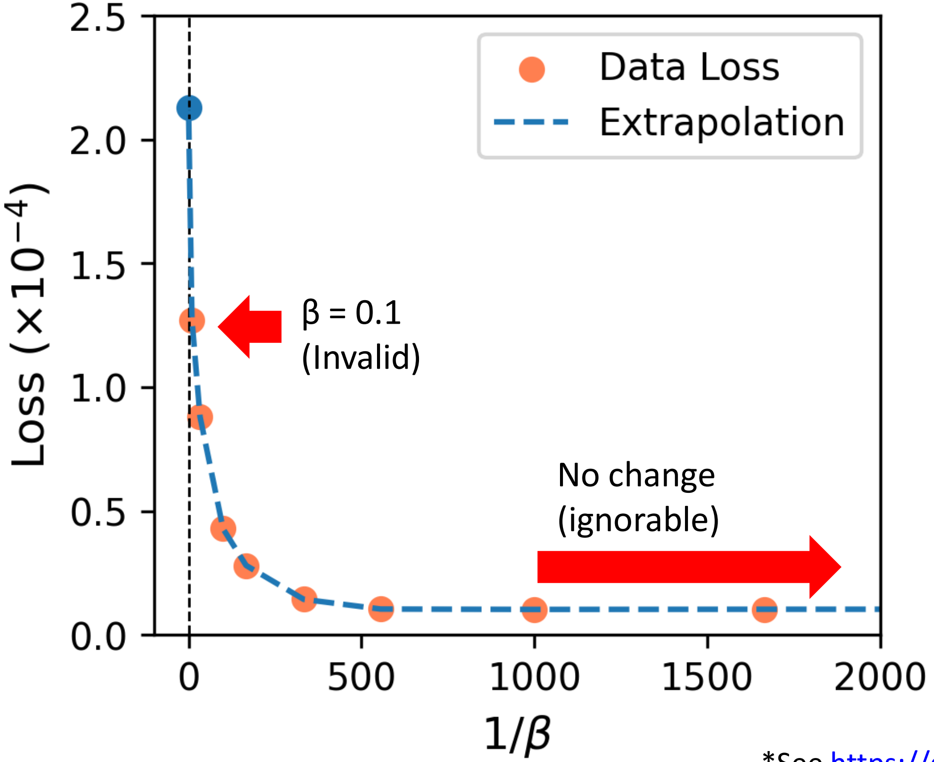
$$L_2 = \sqrt{L_D/n}$$

$$L_2 = \frac{1}{n}\sqrt{\sum(u-u_A)^2 + \sum(v-v_A)^2 + \sum(p-p_A)^2}$$

$$L_D = \frac{1}{n}\left(\sum(u-u_A)^2 + \sum(v-v_A)^2 + \sum(p-p_A)^2\right)$$

*The data loss in the graph is the weighted data loss

Step 3. Analyze Loss



WARNING:
Your training result may look different. There is some randomness in the training that can lead to slightly different results.

- **Minimum estimate:**
  $\beta_{max} = 0.03$
  L_2 @ $\beta_{max} = 3.71 \times 10^{-4}$

- **Maximum estimate:**
  L_D @ $\beta_{inf} = 2.13 \times 10^{-4}$
  L_2 @ $\beta_{inf} = 5.77 \times 10^{-4}$

- **Conclusion**
  $3.71 \times 10^{-4} \leq 5.12 \times 10^{-4} \leq 5.77 \times 10^{-4}$

*See https://doi.org/10.3390/computation9020010 for the extrapolation method

THANK YOU