Physics Informed Neural Networks for Fluid Dynamics

NAPDE Project

Giulia Mescolini, Luca Sosta

Politecnico di Milano

02/11/2021

Poisson Problem with Neumann BCs

For the stationary Poisson problem:

$$-\Delta u = f \qquad \Omega = (0, 2\pi) \times (0, 2\pi) \tag{1}$$

We implemented the Neumann conditions:

```python
def BC_N():
    with ns.GradientTape(persistent = True) as tape:
        tape.watch(x_BC_N)
        u = model(x_BC_N)
        u_x = operator.gradient_scalar(tape, u, x_BC_N)[:,0]
return u_x - g
```

The problem

We considered a first test case in which an analytical solution is known; we have considered the laminar flux between two parallel plates (at distance 2δ), in the fully developed region.

Data of the problem:

- $\delta = 0.05$ m
- L $= 1$ m
- Fluid chosen: Lava
  - $\rho = 3100$ kg/m^3
  - $\mu = 890$ Pa $\cdot$ s

The problem

The data have been chosen in such a way that fully developed conditions are achieved, as we have verified with a simulation on the software PHOENICS. Moreover, the flow is laminar and the Reynolds Number is ~ 3.

Analytical Formulation

We have used the dimensionless formulation of the Navier Stokes equations:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = -\frac{\partial p}{\partial x} \qquad \Omega = (0,1)\times(0,2\delta) \quad (2)$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = -\frac{\partial p}{\partial y} \qquad \Omega = (0,1)\times(0,2\delta) \quad (3)$$

Analytical Formulation

With boundary conditions:

- $u = v = 0$ on $(0, 1) \times \{0, 2\delta\}$
- $\frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = 0$ on $\{0, 1\} \times (0, 2\delta)$

The exact solution is:

$$u_{exact}(y) = -Re\frac{dp}{dx}y(2 - \frac{y}{\delta})\frac{\delta}{2} \tag{4}$$

$$v_{exact} = 0 \tag{5}$$

Analytical Formulation

For this first attempt, we used the known linear exact pressure; therefore, we put:

- $\frac{dp}{dx} = P_{end} - P_{str}$
- $\frac{dp}{dy} = 0$

Where P_{end} and P_{str} are the pressures at the outlet and at the inlet, suitably made dimensionless.

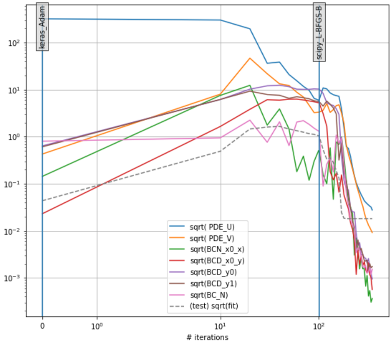
Loss Creation

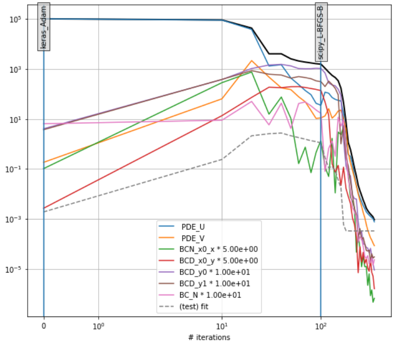
```
def PDE_MOM(x, k, force):
    with ns.GradientTape(persistent=True) as tape:
        tape.watch(x)

        u_vect = model(x)
        u = u_vect[:,0]
        v = u_vect[:,1]
        u_eq = u_vect[:,k]

        grad_eq = operator.gradient_scalar(tape, u_eq, x)
        deqx = grad_eq[:,0]
        deqy = grad_eq[:,1]
        lapl_eq = operator.laplacian_scalar(tape, u_eq, x, dim)

        rhs = create_rhs(x, force)
    return (u * deqx + v * deqy) - (lapl_eq) / Re - rhs
```

Loss Creation

```
def BC_D(x, k, g_bc = None):
    uk = model(x)[:,k]
    rhs = create_rhs(x, g_bc)
    return uk - rhs

def BC_N(x, k, j, g_bc = None):
    with ns.GradientTape(persistent = True) as tape:
        tape.watch(x)
        uk = model(x)[:,k]
        uk_j = operator.gradient_scalar(tape, uk, x)[:,j]
        rhs = create_rhs(x, g_bc)
        return uk_j - rhs

def exact_value(x, k, sol = None):
    uk = model(x)[:,k]
    rhs = create_rhs(x, sol)
    return uk - rhs
```

Losses History





Addition of the mass conservation equation

Since the number of unknowns is now 3, we need to use the mass equation as well:

```
def PDE_MASS(x):
    with ns.GradientTape(persistent=True) as tape:
        tape.watch(x_PDE)
        u_vect = model(x_PDE)[:,0:2]
        div = operator.divergence_vector(tape, u_vect, x_PDE, dim)
    return div
```

Note that we have modified PDE_MOM as well, since it involves pressure which is now unknown.

Changes in boundary conditions

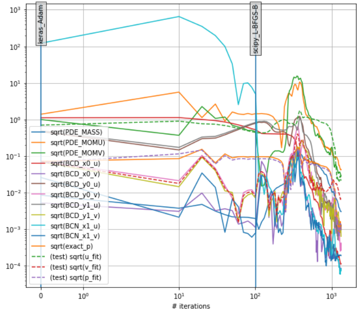
Pressure appears in the classical Neumann condition for fluid-dynamics problems:

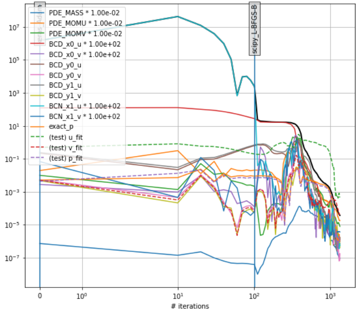
$$\frac{1}{Re}\frac{\partial u_k}{\partial n} - p_k = g \qquad \Gamma_N = \{1\} \times (0, 2\delta)$$

```
def BC_N(x, k, j, pr = None):
    with ns.GradientTape(persistent = True) as tape:
        tape.watch(x)
        uk = model(x)[:,k]
        p = model(x)[:,2] * (k == j) * rho
        uk_j = operator.gradient_scalar(tape, uk, x)[:,j]
        rhs = create_rhs(x, pr) * (k == j)
        return 1/Re * uk_j - p - rhs
```

Loss trend

The imposition of this boundary conditions, together with the imposition of the exact value of pressure for few points (10 in our last simulation), guarantees a good performance of the PINN.





The problem

A slightly more complex case, in which an analytical solution is known, is the following:

$$\begin{cases} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 & \Omega = (-1,1) \times (-1,1) \\ \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial p}{\partial x} & \Omega = (-1,1) \times (-1,1) \\ \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = \frac{\partial p}{\partial y} & \Omega = (-1,1) \times (-1,1) \\ u = 20xy^3 & \partial\Omega \\ v = 5x^4 - 5y^4 & \partial\Omega \end{cases} \tag{6}$$

The exact solution is:

$$\begin{cases} p_{exact}(x,y) = 60x^2y - 20y^3 + C \\ u_{exact}(x,y) = 20xy^3 \\ v_{exact}(x,y) = 5x^4 - 5y^4 \end{cases}$$

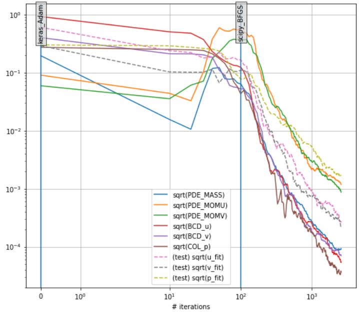
Uniquely determination of pressure

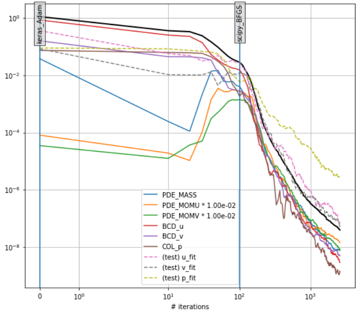
In the equations, pressure appears only through its derivatives; therefore, it is known only up to an additive constant. We considered the case in which $C = 0$, and developed two strategies to reconstruct the solution for pressure with the Neural Network.

- **Imposition of pressure value at some collocation points**
- **Imposition of the mean value for a random sample of points**

Pressure with collocation

We applied the same strategy of the previous test case, using 20 hint points.

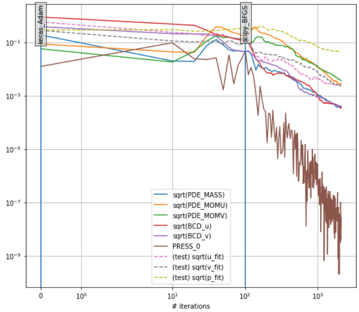


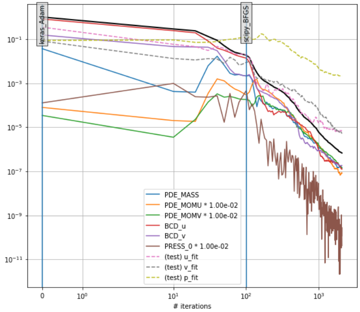


Pressure with mean imposition

We extract num_pres $= 20$ points in the square in our case, and impose $C = 0$ as mean.

```
def PRESS_0(x):
    uk = model(x)[:,2]
    uk_mean = tf.abs(tf.math.reduce_mean(uk))
    return uk_mean
```



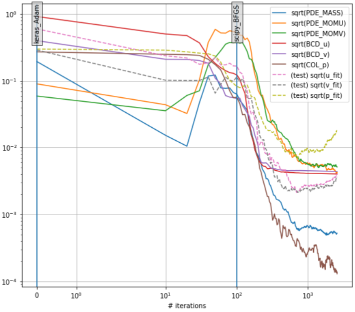


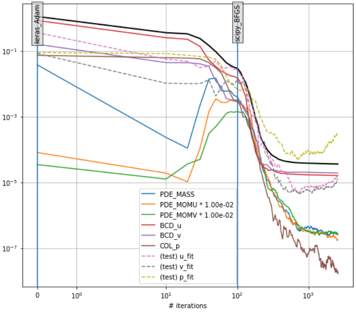
Noise

In this test case, we started considering noisy situations; we included to our model an option which performs the same task, but with Gaussian noise on boundary data. Note that, in order to generate noise, we should fix width, mean and standard deviation (in our case, $w = 0.1$, $\mu = 0$, $\sigma = 1$).

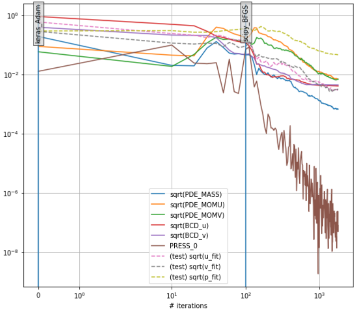
```
    def generate_noise(x, factor = 0, sd = 1.0, mn = 0.0):
        shape = x.shape[0]
        noise = tf.random.normal([shape], mean=mn, stddev=sd, \\
        dtype= ns.config.get_dtype())
        return noise * factor
```

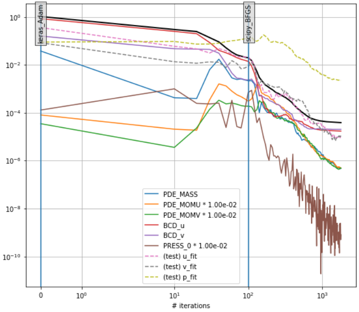
Pressure with collocation - noisy





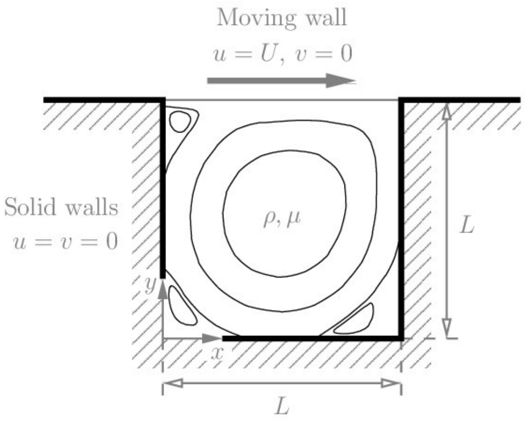
Pressure with mean imposition - noisy





Lid-driven Cavity

The next problem considered is the flow in cavity, both in the stationary and in the non-stationary case.



Stationary problem

The analytical formulation of the problem is the following:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \qquad \Omega = (0,1) \times (0,1) \tag{7}$$

$$-(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} = 0 \qquad \Omega = (0,1) \times (0,1) \tag{8}$$

$$-(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} = 0 \qquad \Omega = (0,1) \times (0,1) \tag{9}$$

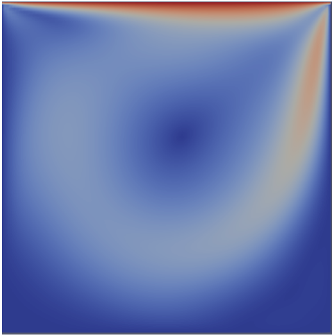
With the following Dirichlet boundary conditions for velocities:

$$u = v = 0 \qquad \Gamma_{D1} = \{0,1\} \times (0,1) \cup (0,1) \times \{0\}$$

$$u = 500, v = 0 \qquad \Gamma_{D2} = (0,1) \times \{1\}$$

What is the solution?

In this case, there is no analytical solution, so we needed a numerical one, generated through the FEniCS script provided. With Paraview we can visualize its trend:



(a) Velocity Magnitude

(b) Pressure

Remarkable modifications

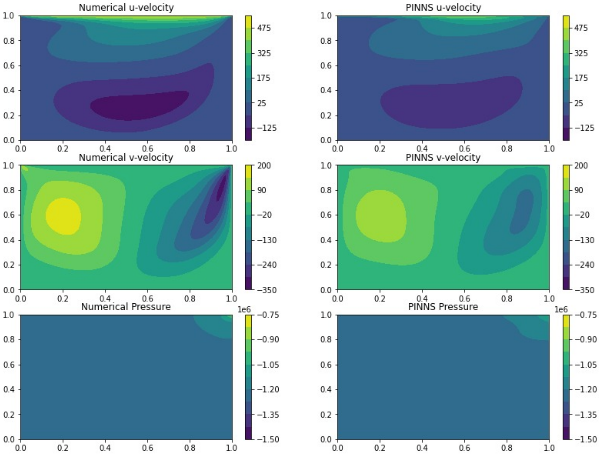
First of all, we had to extract the point locations for training, testing and collocation from the .csv file of the numerical solution.

```
x_num   = pd.DataFrame(df, columns= ['x','y']).to_numpy()
x_PDE   = tf.convert_to_tensor(x_num[:num_PDE,:])
x_col   = tf.convert_to_tensor(x_num[num_PDE:num_PDE+num_col,:])
...
```

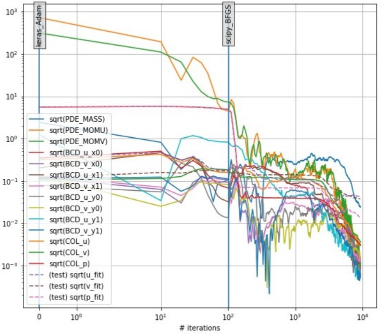
Note that the order of the point is already random, so we can take consecutive elements for each group without renouncing to have points spread in the whole domain.

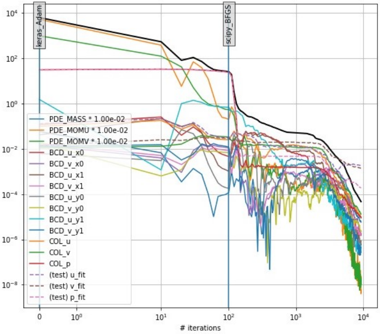
Plots

Here we show a comparison between the numerical and the PINNs solution:



Losses


keras_Adam
scipy_BFGS
sqrt(PDE_MASS)
sqrt(PDE_MOMU)
sqrt(PDE_MOMV)
sqrt(BCD_u_x0)
sqrt(BCD_v_x0)
sqrt(BCD_u_x1)
sqrt(BCD_v_x1)
sqrt(BCD_u_y0)
sqrt(BCD_v_y0)
sqrt(BCD_u_y1)
sqrt(BCD_v_y1)
sqrt(COL_u)
sqrt(COL_v)
sqrt(COL_p)
(test) sqrt(u_fit)
(test) sqrt(v_fit)
(test) sqrt(p_fit)
iterations


keras_Adam
scipy_BFGS
PDE_MASS * 1.00e-02
PDE_MOMU * 1.00e-02
PDE_MOMV * 1.00e-02
BCD_u_x0
BCD_v_x0
BCD_u_x1
BCD_v_x1
BCD_u_y0
BCD_v_y0
BCD_u_y1
BCD_v_y1
COL_u
COL_v
COL_p
(test) u_fit
(test) v_fit
(test) p_fit
iterations

Unsteady case

In this case, the analytical formulation of the problem is:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \qquad \Omega = (0,1) \times (0,1) \tag{10}$$

$$\frac{du}{dt} - \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} = 0 \qquad \Omega = (0,1) \times (0,1) \tag{11}$$

$$\frac{dv}{dt} - \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} = 0 \qquad \Omega = (0,1) \times (0,1) \tag{12}$$

Boundary and Initial Conditions

$$\begin{cases} u = v = 0 \quad \text{on } \{0,1\} \times (0,1) \times [0,T) \cup (0,1) \times \{0\} \times [0,T) \\ u = 1 \quad \text{on } (0,1) \times \{1\} \times [0,T) \\ u = v = p = 0 \quad \text{on } \Omega \times \{0\} \end{cases}$$

Loading the numerical solution

We generated a mesh with 101 points on each edge and divided the time interval $[0, 1]$ into 100 subintervals.
FEniCS generates a .h5 file with the solution for each time instant, and we read them in sequence, storing the numerical solutions for p, u and v at point (i, j) at instant n into the component $k = (101^2)n + 101j + i$.

A trick for pressure

We had difficulties for pressure even for the collocation-only case; the reason was that the pressure has a different mean value for each time instant.
By subtracting the mean value at each time instant, the problem was fixed.

```
# Pressure Mean Equal to 0
p.append(pp-np.mean(pp))
```

The variables' grid

We stored each triple (x, y, t) in the `var` tensor, by fixing time, y and x in this order. This choice had an impact on the formulation of the PDE losses; for example, now the first component of the gradient computed with `nisaba` is the time derivative.
In the following slides, we report our implementation.

PDE Losses

```
def PDE_MASS(x):
    with ns.GradientTape(persistent=True) as tape:
        tape.watch(x)
        u_vect = model(x)[:,0:2] * vel_max
        du_x = operator.gradient_scalar(tape, u_vect[:,0], x)[:,1]
        dv_y = operator.gradient_scalar(tape, u_vect[:,1], x)[:,2]
    return du_x + dv_y
```

PDE Losses

```
def PDE_MOM(x, k, force):
    with ns.GradientTape(persistent=True) as tape:
        tape.watch(x)

        u_vect = model(x)
        p = u_vect[:,2] * p_max
        u_eq = u_vect[:,k] * vel_max

        dp   = operator.gradient_scalar(tape, p, x)[:,k+1]

        du_t = operator.gradient_scalar(tape, u_eq, x)[:,0]
        du_x = operator.gradient_scalar(tape, u_eq, x)[:,1]
        du_y = operator.gradient_scalar(tape, u_eq, x)[:,2]
        du_xx = operator.gradient_scalar(tape, du_x, x)[:,1]
        du_yy = operator.gradient_scalar(tape, du_y, x)[:,2]

        conv1 = tf.math.multiply(vel_max * u_vect[:,0], du_x)
        conv2 = tf.math.multiply(vel_max * u_vect[:,1], du_y)

        rhs = create_rhs(x, force)

    return du_t - du_xx - du_yy + dp + conv1 + conv2 - rhs
```

Loss related to the Initial Condition

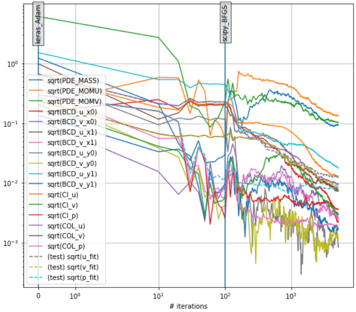
This is the first time-dependent case that we have analyzed; therefore we added also a loss to impose the inititial condition.

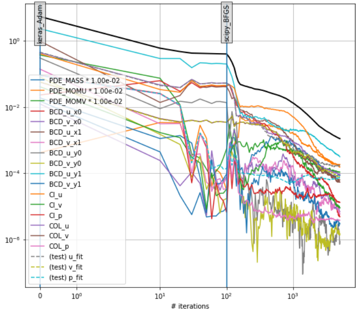
```
def BC_IN(x, k, f, norm = 1, noise = None):
    uk = model(x)[:,k]
    rhs = create_rhs(x, f, noise)
    norm_rhs = rhs/norm
    return uk - norm_rhs

IN_losses = [ns.LossMeanSquares('CI_u', lambda: BC_IN(x_CI, 0, 0, vel_max),\\
             weight = 1e0),
             ns.LossMeanSquares('CI_v', lambda: BC_IN(x_CI, 1, 0, vel_max),\\
             weight = 1e0),
             ns.LossMeanSquares('CI_p', lambda: BC_IN(x_CI, 2, 0, p_max),\\
             weight = 1e0)]
```

Solution of the case without noise (5000 epochs)

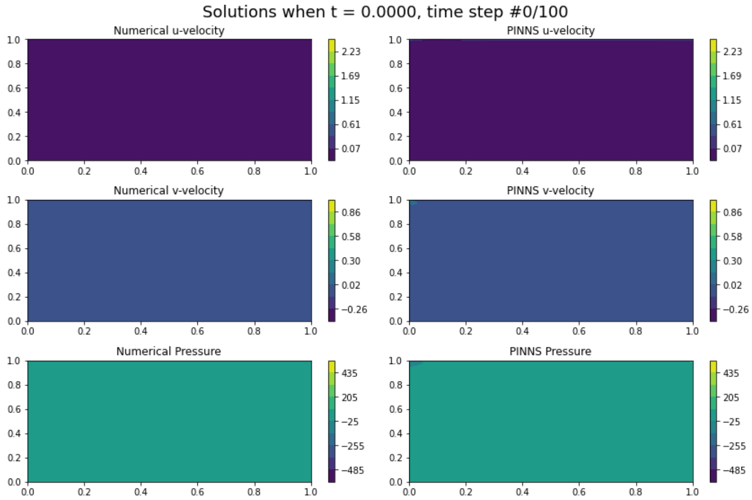
With the following numerical options:

```
num_PDE = 10000
num_BC = 5000
num_CI = 9000
num_col = 1000
num_pres = 2500
num_test = 7500
```

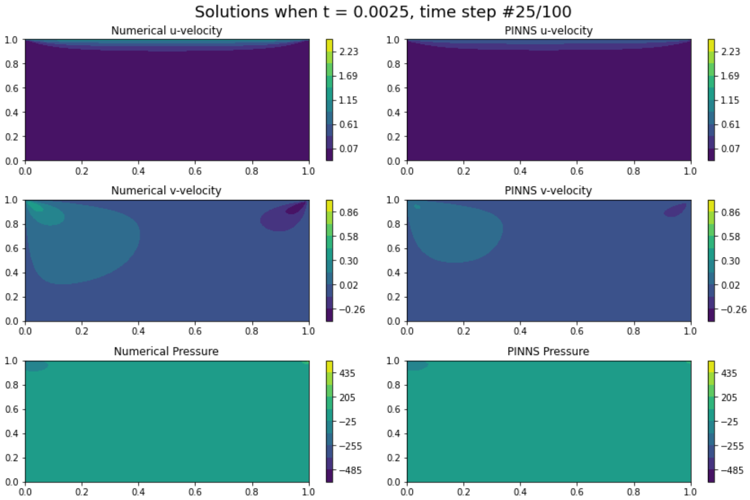




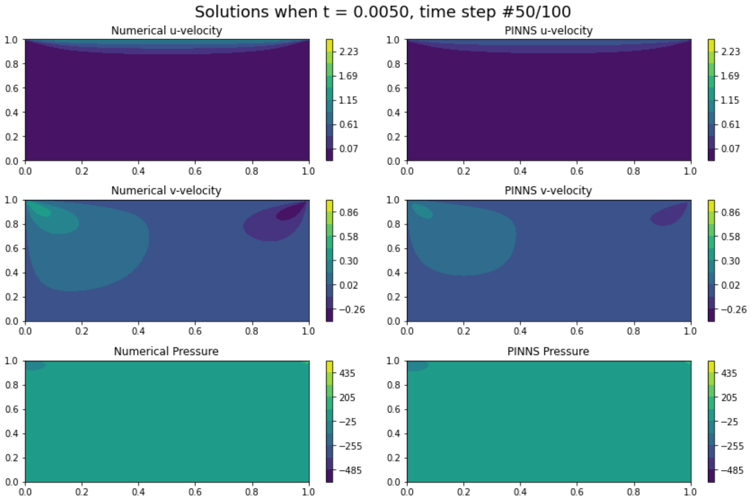
Plots


Solutions when t = 0.0000, time step #0/100
Numerical u-velocity
PINNS u-velocity
2.23
1.69
1.15
0.61
0.07
Numerical v-velocity
PINNS v-velocity
0.86
0.58
0.30
0.02
−0.26
Numerical Pressure
PINNS Pressure
435
205
−25
−255
−485
0.0
0.2
0.4
0.6
0.8
1.0

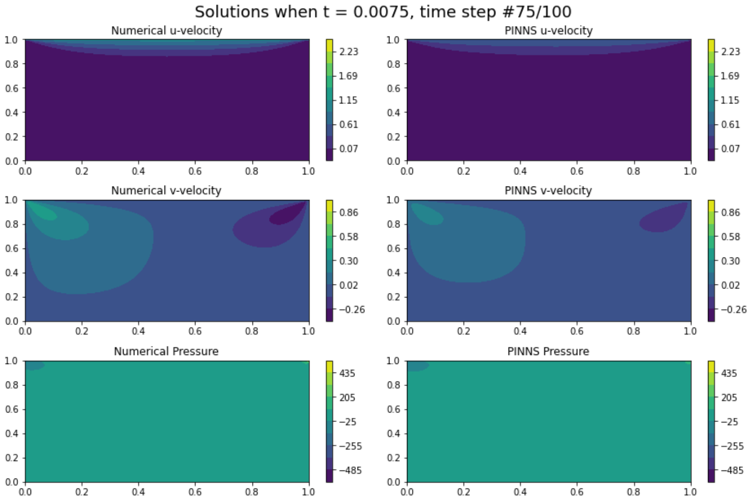
Plots


Solutions when t = 0.0025, time step #25/100
Numerical u-velocity
PINNS u-velocity
2.23
1.69
1.15
0.61
0.07
Numerical v-velocity
PINNS v-velocity
0.86
0.58
0.30
0.02
−0.26
Numerical Pressure
PINNS Pressure
435
205
−25
−255
−485
0.0
0.2
0.4
0.6
0.8
1.0

Plots

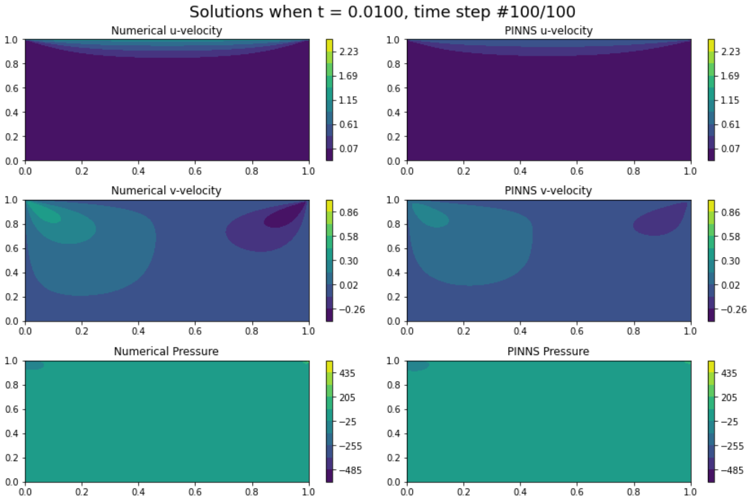

Solutions when t = 0.0050, time step #50/100
Numerical u-velocity
PINNS u-velocity
2.23
1.69
1.15
0.61
0.07
Numerical v-velocity
PINNS v-velocity
0.86
0.58
0.30
0.02
−0.26
Numerical Pressure
PINNS Pressure
435
205
−25
−255
−485
0.0
0.2
0.4
0.6
0.8
1.0

Plots



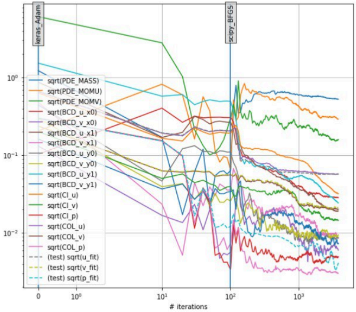
Solutions when t = 0.0075, time step #75/100
Numerical u-velocity
PINNS u-velocity
2.23
1.69
1.15
0.61
0.07
Numerical v-velocity
PINNS v-velocity
0.86
0.58
0.30
0.02
−0.26
Numerical Pressure
PINNS Pressure
435
205
−25
−255
−485
0.0
0.2
0.4
0.6
0.8
1.0

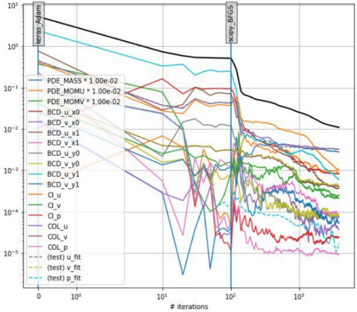
Plots


Solutions when t = 0.0100, time step #100/100
Numerical u-velocity
PINNS u-velocity
Numerical v-velocity
PINNS v-velocity
Numerical Pressure
PINNS Pressure
2.23
1.69
1.15
0.61
0.07
0.86
0.58
0.30
0.02
−0.26
435
205
−25
−255
−485
0.0
0.2
0.4
0.6
0.8
1.0

Noisy case

We added some noise on the boundary conditions; then we analyzed the loss trend while varying the number of boundary and collocation points. With the same options of the non-noisy case, we obtained this trend:

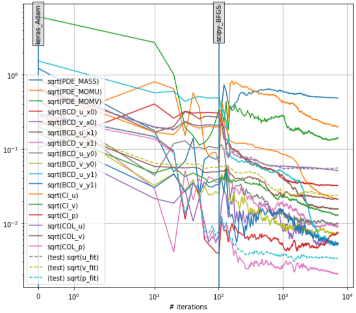


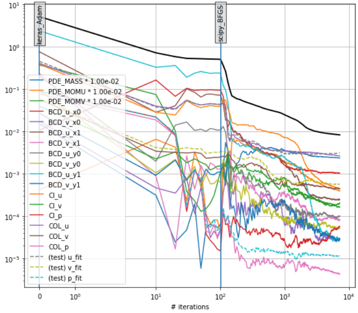


Noisy case

Then we divided by 10 each number of points:

```
num_BC = 500
num_CI = 900
num_col = 100
num_pres = 250
```

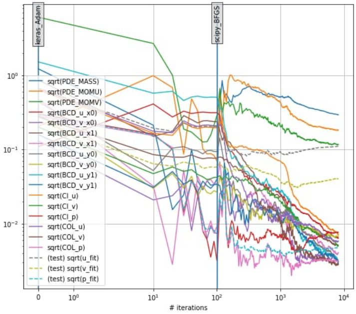


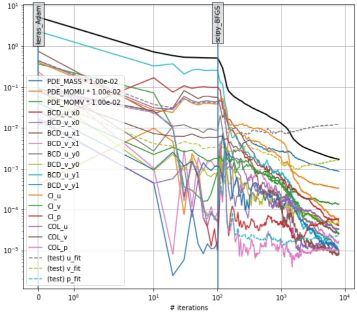


Noisy case

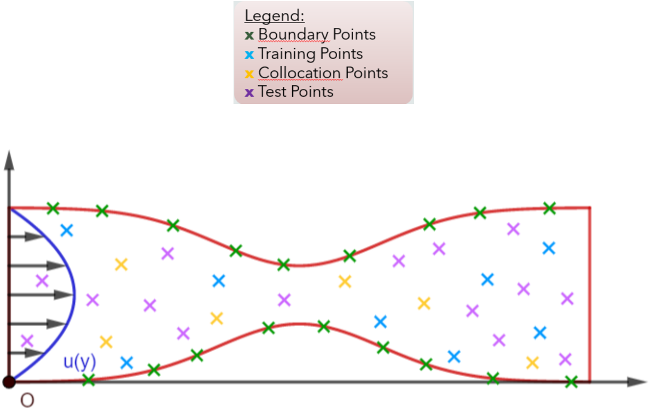
And then by 100:

```
num_BC = 50
num_CI = 90
num_col = 10
num_pres = 25
```



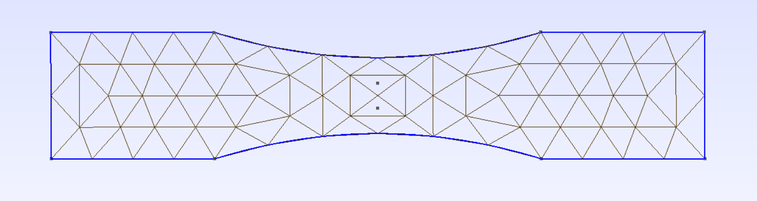


The next case we will analyze is the blood flow in an unhealthy coronary.



Generation of the mesh with GMSH

We generated a mesh with GMSH (with BSpline); should we use a particular function to model the upper and lower edge?



Problem Setup

- **Equations:** Navier-Stokes 2D, stationary/non-stationary
- **Boundary Conditions:** $u = v = 0$ on upper and lower boundary, Neumann on the vertical edges?
- **Uniqueness of Pressure:** through imposition of numerical solution in collocation points