Physics Informed Neural Networks for Fluid Dynamics

NAPDE Project

Giulia Mescolini, Luca Sosta

Politecnico di Milano

23/04/2021

1. Poiseuille Flow
2. Pressure as Datum
3. Pressure as Unknown

The problem

In order to consider a case in which an analytical solution is known, we have firstly considered the laminar flux between two parallel plates (at distance 2δ) in the fully developed region.

- $\delta = 0.05$ 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:

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

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

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:

$$u_{exact}(y) = -Re\frac{dp}{dx}y(2 - \frac{y}{\delta})\frac{\delta}{2} \tag{3}$$

$$v_{exact} = 0 \tag{4}$$

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.

Definition of the model

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(20, input_shape=(2,), activation=tf.nn.tanh),
    tf.keras.layers.Dense(20, activation=tf.nn.tanh),
    tf.keras.layers.Dense(20, activation=tf.nn.tanh),
    tf.keras.layers.Dense(2)
])
```

Loss Creation

```python
def PDE(x, k, force):  # k is the coordinate of the vectorial equation
    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)
    return (u * deqx + v * deqy) - (lapl_eq) / Re - force
```

Loss Creation

```python
def BC_D(x, k, g_bc = None):
    with ns.GradientTape(persistent = True) as tape:
        if g_bc is None:
            samples = x.shape[0]
            g_bc = tf.zeros(shape = [samples,1], dtype = ns.config.get_dtype())
        tape.watch(x)
        uk = model(x)[:,k]
        return tf.math.abs(uk - g_bc)

def BC_N(x, k, j, g_bc = None): #j is the direction in which we want the derivative
    with ns.GradientTape(persistent = True) as tape:
        if g_bc is None:
            samples = x.shape[0]
            g_bc = tf.zeros(shape = [samples,1], dtype = ns.config.get_dtype())
        tape.watch(x)
        uk = model(x)[:,k]
        uk_j = operator.gradient_scalar(tape, uk, x)[:,j]
        return tf.math.abs(uk_j - g_bc)
```

Loss Creation

```python
def Hints():
    with ns.GradientTape(persistent = True) as tape:
        tape.watch(x_hint)
        u_vect = model(x_hint)
        u = u_vect[:,0]
        v = u_vect[:,1]
    return (u - u_hint) * (u - u_hint) + (v - v_hint) * (v - v_hint)

def test_loss():
    u_vect = model(x_test)
    u = u_vect[:,0]
    v = u_vect[:,1]
    return (u - u_test) * (u - u_test) + (v - v_test) * (v - v_test)
```

Losses Defintion

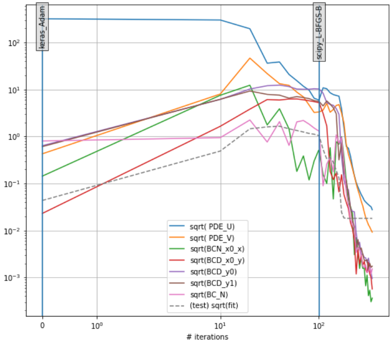
```
losses = [ns.LossMeanSquares(' PDE_U', lambda: PDE(x_PDE,0,f_1), weight = 1.0),
          ns.LossMeanSquares(' PDE_V', lambda: PDE(x_PDE,1,f_2), weight = 1.0),
          ns.LossMeanSquares('BCN_x0_x', lambda: BC_N(x_BC_x0,0,0), weight = 5.0),
          ns.LossMeanSquares('BCD_x0_y', lambda: BC_D(x_BC_x0,1), weight = 5.0),
          ns.LossMeanSquares('BCD_y0', lambda: BC_D(x_BC_y0,0) + BC_D(x_BC_y0,1),
                             weight = 10.0),
          ns.LossMeanSquares('BCD_y1', lambda: BC_D(x_BC_y1,0) + BC_D(x_BC_y1,1),
                             weight = 10.0),
          ns.LossMeanSquares( 'BC_N', lambda: BC_N(x_BC_x1,0,0)
                             + BC_N(x_BC_x1,1,0), weight = 10.0),
          #ns.LossMeanSquares('Hints', lambda: Hints(), weight = 15.0)
          ]
loss_test = ns.LossMeanSquares('fit', test_loss, normalization = num_test)
```

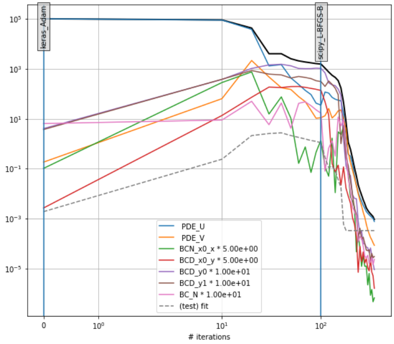
1. Poiseuille Flow

2. Pressure as Datum
   - Theoretical Model
   - Model Initialization
   - Losses Definition
   - Numerical Solution
   - Critical Issues

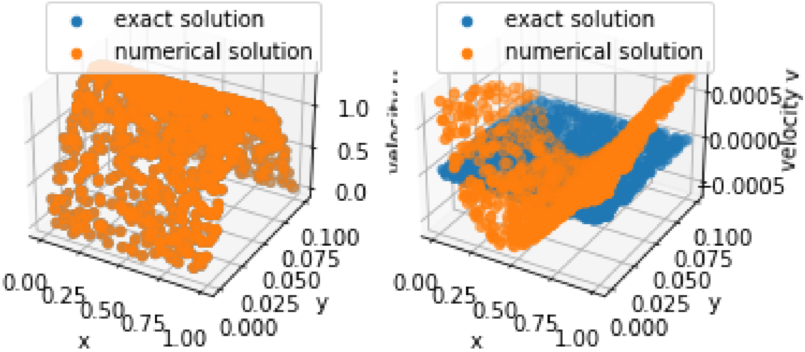
3. Pressure as Unknown

Losses History



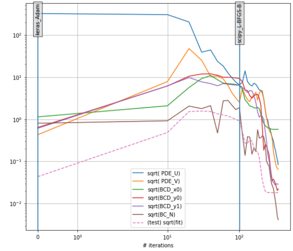


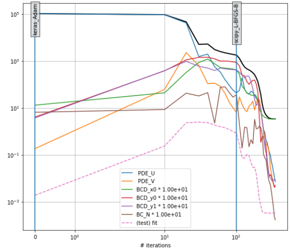
Numerical Solution vs Exact Solution



Dirichlet Boundary condition for u at x0

When imposing the exact solution for u at x0, we have an unpleasant behaviour of the corresponding loss:





Addition of boundary conditions

We need to add boundary conditions for pressure: which ones?

- Surely, the two Dirichlet BCs:
  - p $= p_{str} =$ 1e6 at $0 \times (0, 2\delta)$
  - p $= p_{end} =$ 0 at $1 \times (0, 2\delta)$
- Our trials for y=0 and y=2δ
  1. Dirichlet condition at y=0 (imposing the exact solution) and Neumann condition at y=2δ
  2. Neumann conditions for both borders

Exact solution

The exact solution for pressure is:

$$p(x) = \frac{(p_{end} - p_{str})}{L}x + p_{str} \tag{5}$$

Loss Creation

```
def PDE(x, k, force):  # k is the coordinate of the vectorial equation
    with ns.GradientTape(persistent=True) as tape:
        tape.watch(x)
        u_vect = model(x)
        u = u_vect[:,0]
        v = u_vect[:,1]
        p = u_vect[:,2]
        u_eq = u_vect[:,k]
        grad_eq = operator.gradient_scalar(tape, u_eq, x)
        dp    = operator.gradient_scalar(tape, p, x)[:,k]
        deqx = grad_eq[:,0]
        deqy = grad_eq[:,1]
        lapl_eq = operator.laplacian_scalar(tape, u_eq, x, dim)
    return (u * deqx + v * deqy) - (lapl_eq) / Re + dp - force
```

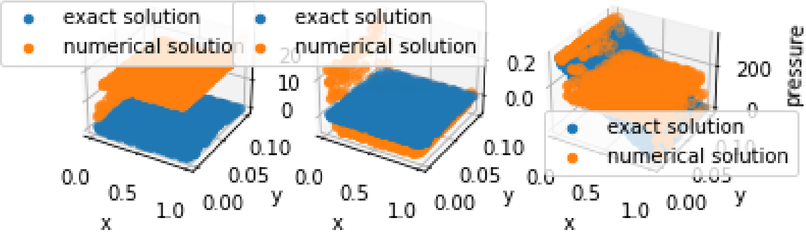
Loss Creation

```
def test_loss():
    u_vect = model(x_test)
    u = u_vect[:,0]
    v = u_vect[:,1]
    p = u_vect[:,2]
    return (u - u_test) * (u - u_test) + (v - v_test) * (v - v_test) +
           (p - p_test) * (p - p_test)
```

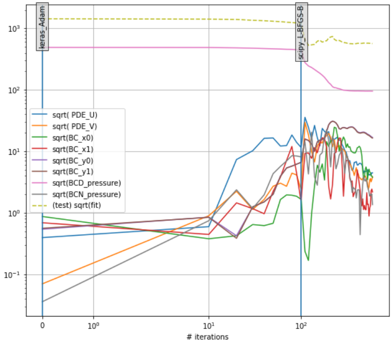
Loss Definition (case 1)

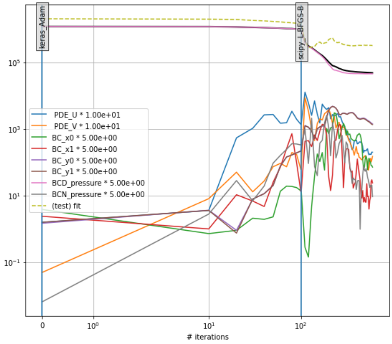
```
losses = [ns.LossMeanSquares('␣PDE_U', lambda: PDE(x_PDE, 0, f_1), weight = 10.0),
          ns.LossMeanSquares('␣PDE_V', lambda: PDE(x_PDE, 1, f_2), weight = 10.0),
          ns.LossMeanSquares( 'BC_x0', lambda: BC_N(x_BC_x0,0,0) + BC_D(x_BC_x0,1),
                              weight = 5.0),
          ns.LossMeanSquares( 'BC_x1', lambda: BC_N(x_BC_x1,0,0) +
                              BC_N(x_BC_x1,1,0), weight = 5.0),
          ns.LossMeanSquares( 'BC_y0', lambda: BC_D(x_BC_y0,0  ) + BC_D(x_BC_y0,1
),
                              weight = 5.0),
          ns.LossMeanSquares( 'BC_y1', lambda: BC_D(x_BC_y1,0  ) + BC_D(x_BC_y1,1
),
                              weight = 5.0),
          ns.LossMeanSquares( 'BCD_pressure', lambda: BC_D(x_BC_x0,2, p_inlet)
                              + BC_D(x_BC_x1,2, p_outlet) + BC_D(x_BC_y0, 2, p_y0),
                              weight = 5.0 ),
          ns.LossMeanSquares( 'BCN_pressure', lambda: BC_N(x_BC_y1, 2, 1),
                              weight = 5.0),
         #ns.LossMeanSquares('Hints', Hints, weight = 15.0)
          ]
```

Numerical Solution vs Exact Solution in case 1

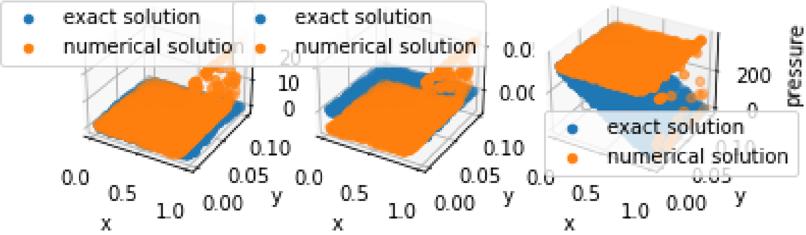


History Loss in case 1





Numerical Solution vs Exact Solution in case 2



History Loss in case 2