Lab9: Physical Processes	Aliya Aibarkyzy Tursynzhanova

Laboratory Work 9: Projection Method for the Navier-Stokes + Temperature (2D)

Solve the boundary problem of given PDE. Write a code for the approximation and compare different iteration results of numerical solutions.

Your goal is to learn how to visualize two dimensional equations.

The deadline of given Lab9 is the W13.
No delays, no mercy.

YOU SHOULD SUBMIT YOUR COMPLETED REPORD IN PDF ON TEAMS' ASSIGNMENT SECTION!

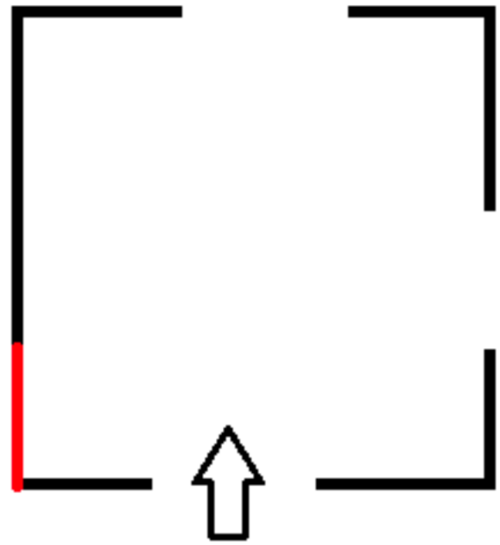
ONLY IN PDF!

Max – 4 POINTS.

Example of a correctly completed laboratory work 8

1. Your mathematical model (eq + BC + IG)
2. Numerical approximation by using any methods to solve Navier-Stokes
3. Python/Matlab code
4. !!! At least three different iterations!!! for u and v components, P and T
5. Final iterations, and how number of iterations changes with the change of epsilon.
6. Conclusion.

The system of Navier – Stokes equation	$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$ $$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right),$$ $$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \alpha^2\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right),$$ $$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$
Initial condition:	$u(t = 0, x, y) = 0,$ $v(t = 0, x, y) = 0$ $P(t = 0, x, y) = 0, i\ guess\ ...$ For Temperature $T(t = 0, x, y) = 0\ or\ also\ you\ can\ say\ T(t = 0, x, y) = 16$

Boundary condition	 arrows -> Dirichlet = 1 outlet (empty space, hole)-> Neumann = 0 Red line is Boundary condition Dirichlet for the Temperature, it can be equal to 1 or to 25, for example Don't forget about the P
Reynolds number	Consider any value
Density	It also can be any value, for our convenience let's do 1
Diffusion coef	It also can be any value, for our convenience let's do 1
Projection method	$$\frac{\partial u}{\partial t} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + L_x$$ $$\frac{\partial v}{\partial t} = -\frac{1}{\rho}\frac{\partial P}{\partial y} + L_y$$ Where $L_x and\ L_y$ are operators that consider remaining convection and diffusion part $$L_x = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$ $$L_y = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$ Now we should add our splitting parameter (параметр расщепления) to get accurate results $$\frac{\partial u}{\partial t} = \frac{u^{n+1} - u^n + u^* - u^*}{dt}$$ $$\frac{\partial v}{\partial t} = \frac{v^{n+1} - v^n + v^* - v^*}{dt}$$

	Since we added splitting parameter u_star and v_star, we will SPLIT our equation into two: for parameter and for convection and diffusion. Let's do it, brothers: $$\begin{cases} \dfrac{u^* - u^n}{dt} = L_x \\ \dfrac{u^{n+1} - u^*}{dt} = -\dfrac{1}{\rho}\dfrac{\partial P}{\partial x} \end{cases}$$ *and the same for v* $$\begin{cases} \dfrac{v^* - v^n}{dt} = L_y \\ \dfrac{v^{n+1} - v^*}{dt} = -\dfrac{1}{\rho}\dfrac{\partial P}{\partial y} \end{cases}$$
Firstly, we will find u_star and v_star. You can use any preferable numerical method, that you would like to use.	$$\frac{u^* - u^n}{dt} = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$ u^* *is uknown, we should find it*. By rewriting given equation $$\frac{u^* - u^n}{dt} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$ What do we see? Exactly! Burger's equation, that you already know how to solve by using different methods. I assume, easiest will be Simple Iteration Method. $$\frac{v^* - v^n}{dt} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$ v^* *is uknown, we should find it*.
Secondly, we will edit our Pressure field by expressing u^(n+1) and plugging it into Continuity equation	$$\frac{u^{n+1} - u^*}{dt} = -\frac{1}{\rho}\frac{\partial P}{\partial x} \rightarrow u^{n+1} = u^* - \frac{dt}{\rho}\frac{\partial P}{\partial x}$$ $$\frac{v^{n+1} - v^*}{dt} = -\frac{1}{\rho}\frac{\partial P}{\partial y} \rightarrow v^{n+1} = v^* - \frac{dt}{\rho}\frac{\partial P}{\partial y}$$
Poisson's equation -> You can solve it by using any method	$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} = \frac{\rho}{dt}\left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y}\right)$$
Last step, to solve u^(n+1) and v^(n+1)	$$u_{ij}^{n+1} = u_{ij}^* - \frac{dt}{\rho}\left(\frac{P_{ij}^n - P_{i-1j}^n}{\Delta x}\right)$$ $$v_{ij}^{n+1} = v_{ij}^* - \frac{dt}{\rho}\left(\frac{P_{ij}^n - P_{ij-1}^n}{\Delta y}\right)$$

Transport equation for the temperature, you can apply any methods	$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \alpha^2\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)$ Simple Iteration Method: $\frac{T_{ij}^{n+1} - T_{ij}^{n}}{dt} + u_{ij}^{n}\frac{T_{ij}^{n} - T_{i-1j}^{n}}{dx} + v_{ij}^{n}\frac{T_{ij}^{n} - T_{ij-1}^{n}}{dy} = \alpha^2\left(\frac{T_{i+1j}^{n} - 2T_{ij}^{n} + T_{i-1j}^{n}}{dx^2} + \frac{T_{ij+1}^{n} - 2T_{ij}^{n} + T_{ij-1}^{n}}{dy^2}\right)$

Pseudo-Code (It's your work to complete given code) + add to this code Temperature transport equation

phys_proc_25 > proj_method_sample.py > ...

```python
import matplotlib.pyplot as plt
import numpy as np
import copy

def burgers_star(u, v, dy, dt, dx, Re, n):
    iter = 0
    while True:
        diff = 0
        un = copy.deepcopy(u)
        vn = copy.deepcopy(v)
        for i in range(1, n):
            for j in range(1, n):

                u_star and v_star =

        u, v = un, vn
        iter += 1
        if diff <= 0.001:
            break

    print("Burgers", iter, diff)
    return u, v

def poisson_p(P, u, v, p, dx, dy, dt, n):
    iter = 0
    while True:
        diff = 0
        Pn = copy.deepcopy(P)
        for i in range(1, n):
            for j in range(1, n):
                if i == 1 and j == 1:
                    Pn[i][j] = 1 / 2 * (P[i + 1][j] + P[i][j + 1] - p * ((u[i][j] - u[i - 1][j]) / dx + (v[i][j] - v[i][j - 1]) / dy))
                elif i == 1:
                    Pn[i][j] = 1 / 3 * (P[i + 1][j] + P[i][j + 1] + Pn[i][j - 1] - p * ((u[i][j] - u[i - 1][j]) / dx + (v[i][j] - v[i][j - 1]) / dy))
                elif j == 1 and not (0.4 * n <= i <= 0.6 * n):
                    Pn[i][j] = 1 / 3 * (P[i + 1][j] + Pn[i - 1][j] + P[i][j + 1] - p * ((u[i][j] - u[i - 1][j]) / dx + (v[i][j] - v[i][j - 1]) / dy))
                else:
                    Pn[i][j] = 1 / 4 * (P[i + 1][j] + Pn[i - 1][j] + P[i][j + 1] + Pn[i][j - 1] - p * ((u[i][j] - u[i - 1][j]) / dx + (v[i][j] - v[i][j - 1]) / dy))
                diff = max(diff, abs(Pn[i][j] - P[i][j]))
```

```python
def poisson_p(P, u, v, p, dx, dy, dt, n):
        for i in range(n + 1):
            Pn[i][0] = Pn[i][1]
            Pn[i][n] = Pn[i][n - 1]
            Pn[n][i] = Pn[n - 1][i]
            Pn[0][i] = Pn[1][i]

        Pn[0][0] = P[1][1]
        Pn[0][n] = Pn[1][n - 1]
        Pn[n][0] = Pn[n - 1][1]
        Pn[n][n] = Pn[n - 1][n - 1]

        for i in range(int(0.4 * n), int(0.6 * n) + 1):
            Pn[i][0] = 0
            Pn[i][n] = 1
            Pn[0][i] = 0
        P = Pn

        iter += 1

        if diff <= 0.001:
            break
    print("Gauss_seidel", iter, diff)
    return P


n = 100
dx = dy = 1 / n
dt = dx ** 2
Re, p = 2, 4
iter = 0

xlist = [i * dx for i in range(n + 1)]
ylist = [j * dy for j in range(n + 1)]

u = np.zeros((n + 1, n + 1))
v = np.zeros((n + 1, n + 1))
P = np.zeros((n + 1, n + 1))
```

phys_proc_25 > proj_method_sample.py > ...

```python
for i in range(n + 1):
    if int(0.4 * n) <= i <= int(0.6 * n):
        P[i][n] = 1
        u[i][n] = -1

plt.contourf(xlist, ylist, P)
plt.show()

plt.contourf(xlist, ylist, u)
plt.show()

plt.contourf(xlist, ylist, v)
plt.show()

while True:
    diff = 0

    un = copy.deepcopy(u)
    vn = copy.deepcopy(v)

    us, vs = burgers_star(u, v, dy, dt, dx, Re, n)

    P = poisson_p(P, us, vs, p, dx, dy, dt, n)

    for i in range(1, n):
        for j in range(1, n):
            un[i][j] = us[i][j] - dt / (p * dx) * (P[i][j] - P[i - 1][j])
            diff = max(diff, abs(un[i][j] - u[i][j]))

            vn[i][j] = vs[i][j] - dt / (p * dy) * (P[i][j] - P[i][j - 1])
            diff = max(diff, abs(vn[i][j] - v[i][j]))

    for i in range(n + 1):
        if int(0.4 * n) <= i <= int(0.6 * n):
            un[i][0] = un[i][1]
            vn[0][i] = vn[1][i]

    u, v = un, vn

    iter += 1
```

```
    iter += 1

    print("the end", iter, diff)

    if diff <= 0.0001:
        break

print(iter)

plt.contourf(xlist, ylist, u)
plt.show()

plt.contourf(xlist, ylist, v)
plt.show()

plt.contourf(xlist, ylist, P)
plt.show()
```

Graphs should be printed for different iterations for u and v components, for P and T.

Conclusion: I understand that time is a valuable thing, watch it fly by as the pendulum swings.

You can find your boundary condition in attached lab9.pdf