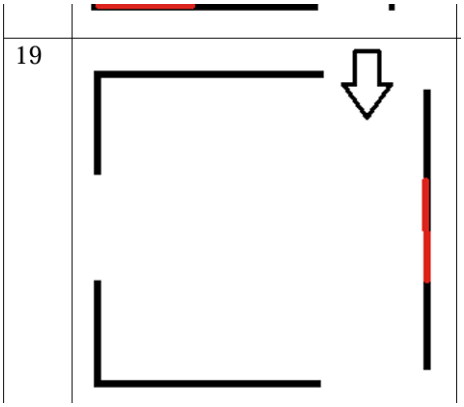
Assignment

Shahnoza Nurubloeva 22B051055

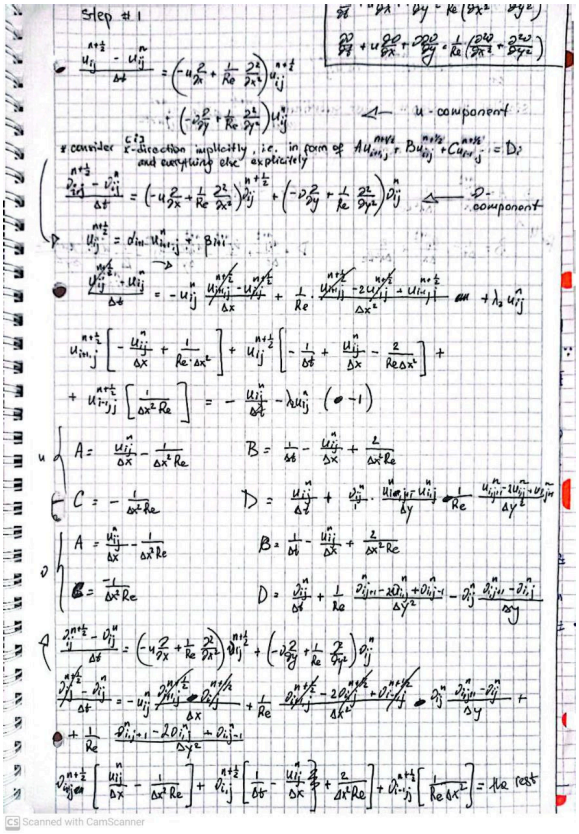
Problem

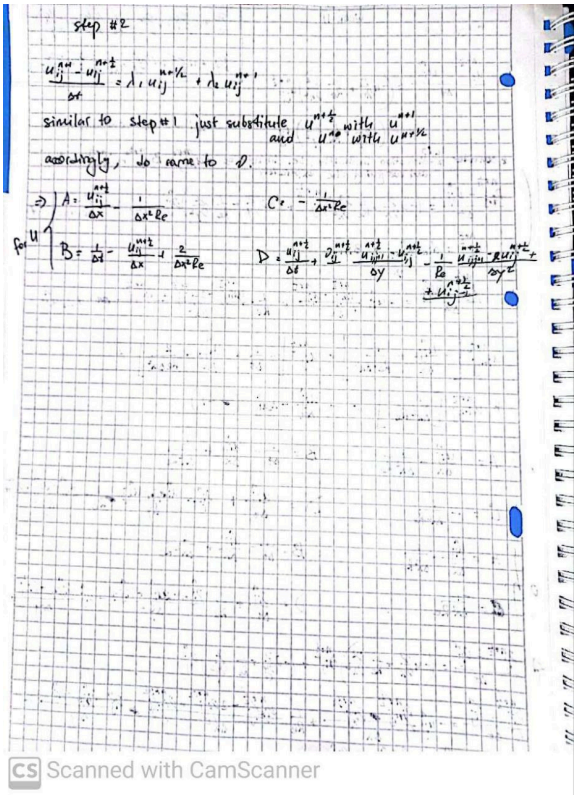


Math Method

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{\partial P}{\partial x}+\frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right),$$
$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\rho}\frac{\partial P}{\partial y}+\frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right),$$
$$\frac{\partial T}{\partial t}+u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}=\alpha^2\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right),$$
$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0$$

$$u(t=0,x,y)=0,$$
$$v(t=0,x,y)=0$$





Code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from mpl_toolkits.mplot3d import Axes3D
import copy

n = 50
dx = dy = 1.0 / n
dt = dx * 0.5
Re = 100
rho = 1.0
alpha = 0.1
max_iter = 500
epsilon = 1e-4
```

```
u = np.zeros((n+1, n+1))  # x-velocity
v = np.zeros((n+1, n+1))  # y-velocity
p = np.zeros((n+1, n+1))  # pressure
T = np.zeros((n+1, n+1))  # temper.

# boundary conditions
# inlet condition
inlet_start = int(0.7*n)
inlet_end = n
u[-1, inlet_start:inlet_end] = 1.0

# velocity BCs
u[int(0.3*n):int(0.7*n), 0] = 1
# v[int(0.3*n):int(0.7*n), 0] = 1
# u[0, inlet_start:inlet_end]=1
v[0, inlet_start:inlet_end]=1



# temperature BCs
T[0, :] = 0.0
T[-1, :] = 0.0
T[:, 0] = 0.0
T[int(0.4*n):int(0.7*n), -1] = 25.0

x = np.linspace(0, 1, n+1)
y = np.linspace(0, 1, n+1)
X, Y = np.meshgrid(x, y)

def burgers_star(u, v, dt, dx, dy, Re):
    u_star = np.zeros_like(u)
    v_star = np.zeros_like(v)

    for i in range(1, n):
        for j in range(1, n):
            conv_u = u[i,j] * (u[i,j] - u[i-1,j])/dx + v[i,j] * (u[i,j] - u[i,j-1])/dy
            conv_v = u[i,j] * (v[i,j] - v[i-1,j])/dx + v[i,j] * (v[i,j] - v[i,j-1])/dy

            diff_u = (u[i+1,j] - 2*u[i,j] + u[i-1,j])/dx**2 + (u[i,j+1] - 2*u[i,j] + u[i,j-1])/d
```

```
            diff_v = (v[i+1,j] - 2*v[i,j] + v[i-1,j])/dx**2 + (v[i,j+1] - 2*v[i,j] + v[i,j-1])/d

            u_star[i,j] = u[i,j] + dt*(-conv_u + (1/Re)*diff_u)
            v_star[i,j] = v[i,j] + dt*(-conv_v + (1/Re)*diff_v)

    u_star[:, 0] = 0; u_star[:, -1] = 0; u_star[0, :] = 0; u_star[-1, :] = 0
    v_star[:, 0] = 0; v_star[:, -1] = 0; v_star[0, :] = 0; v_star[-1, :] = 0

    return u_star, v_star

def solve_poisson(p, u_star, v_star, dt, dx, dy, rho, max_iter=1000, eps=1e-4):
    p_new = np.copy(p)
    b = np.zeros_like(p)

    for i in range(1, n):
        for j in range(1, n):
            b[i,j] = (rho/dt) * ((u_star[i+1,j] - u_star[i-1,j])/(2*dx) + (v_star[i,j+1] - v_s

    # Jacobi iteration
    for iter in range(max_iter):
        p_old = np.copy(p_new)
        max_diff = 0

        for i in range(1, n):
            for j in range(1, n):
                p_new[i,j] = 0.25 * (p_old[i+1,j] + p_old[i-1,j] + p_old[i,j+1] + p_old[i,j-

                diff = abs(p_new[i,j] - p_old[i,j])
                if diff > max_diff:
                    max_diff = diff

        if max_diff < eps:
            print(f"Poisson converged in {iter} iterations")
            break

    # pressure BCs (Neumann)
    p_new[0, :] = p_new[1, :]    # Left
    p_new[-1, :] = p_new[-2, :]  # Right
```

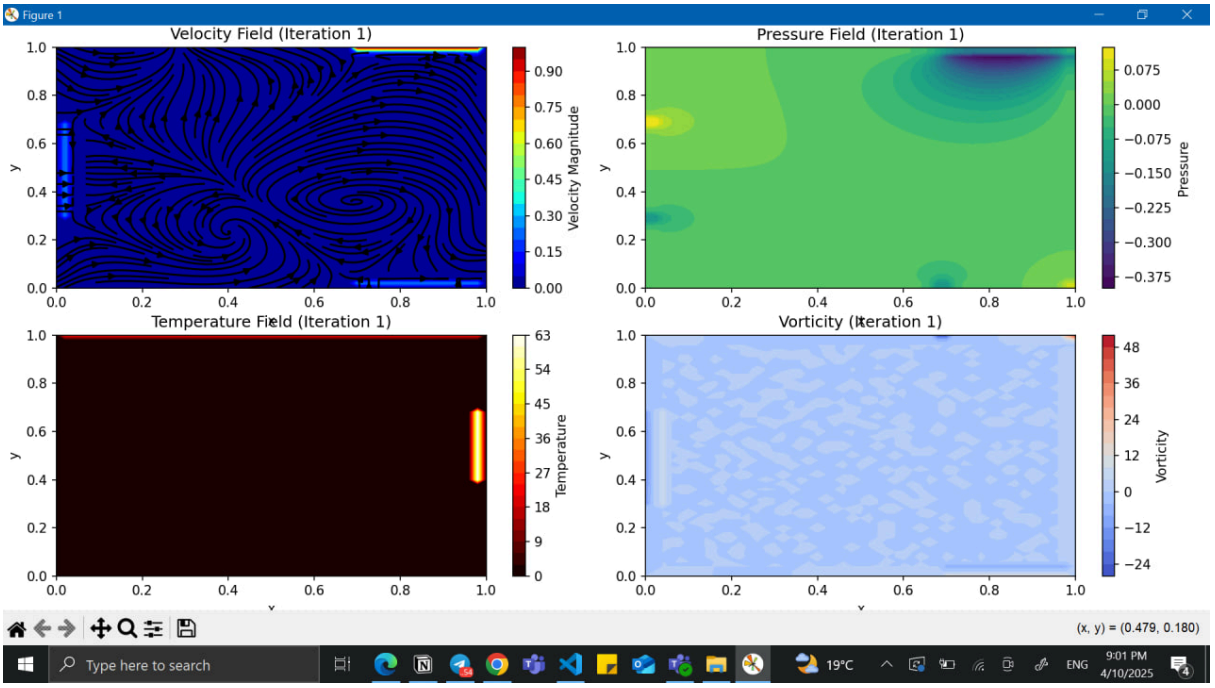
```
    p_new[:, 0] = p_new[:, 1]    # Bottom
    p_new[:, -1] = p_new[:, -2]  # Top

    return p_new

def update_temperature(T, u, v, dt, dx, dy, alpha):
    T_new = np.copy(T)

    for i in range(1, n):
        for j in range(1, n):
            adv_T = u[i,j] * (T[i,j] - T[i-1,j])/dx + v[i,j] * (T[i,j] - T[i,j-1])/dy

            diff_T = alpha * ((T[i+1,j] - 2*T[i,j] + T[i-1,j])/dx**2 + (T[i,j+1] - 2*T[i,j] +

            T_new[i,j] = T[i,j] + dt*(-adv_T + diff_T)

    T_new[0, :] = 0.0  # Left wall (cold)
    T_new[-1, :] = 25.0  # Right wall (hot)
    T_new[:, 0] = 0.0   # Bottom wall (cold)
    T_new[:, -1] = 0.0  # Top wall (cold)

    return T_new

for step in range(max_iter):
    print(f"Step {step+1}/{max_iter}")

    u_old = np.copy(u)
    v_old = np.copy(v)
    p_old = np.copy(p)
    T_old = np.copy(T)

    u_star, v_star = burgers_star(u, v, dt, dx, dy, Re)

    p = solve_poisson(p, u_star, v_star, dt, dx, dy, rho)

    # velocities → divergence-free
    for i in range(1, n):
        for j in range(1, n):
```

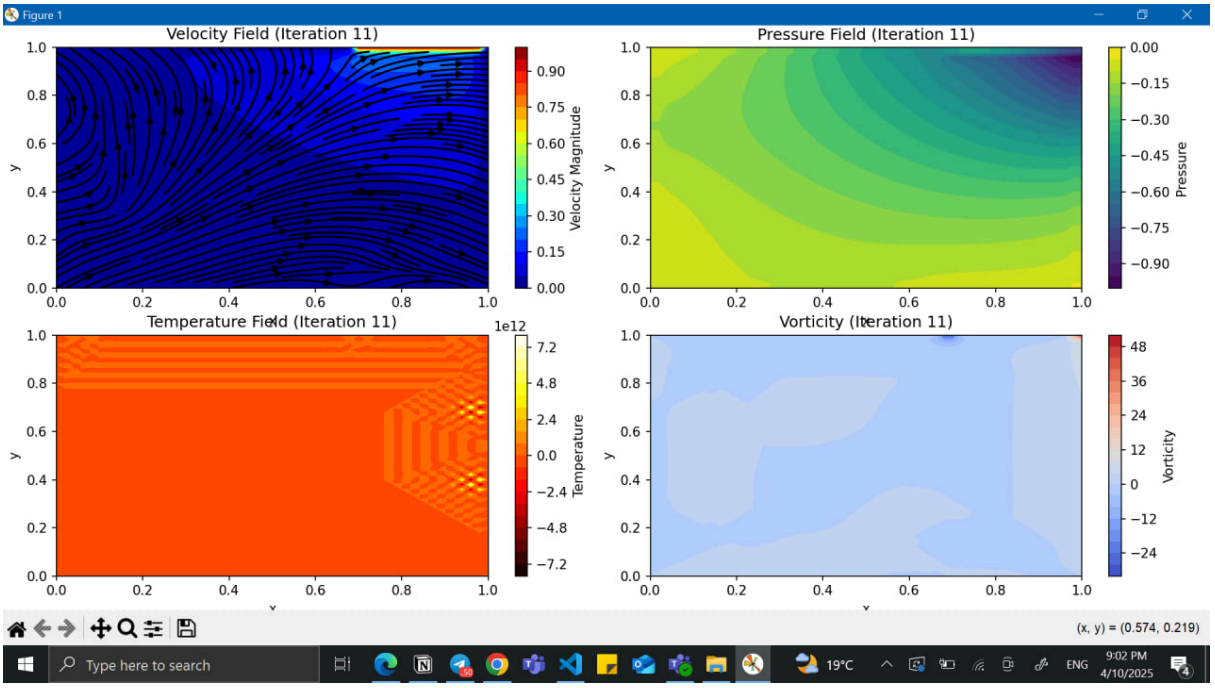
```
        u[i,j] = u_star[i,j] - (dt/rho) * (p[i+1,j] - p[i-1,j])/(2*dx)
        v[i,j] = v_star[i,j] - (dt/rho) * (p[i,j+1] - p[i,j-1])/(2*dy)

T = update_temperature(T, u, v, dt, dx, dy, alpha)

u[:, 0] = 0; u[:, -1] = 0; u[0, :] = 0; u[-1, :] = 0
v[:, 0] = 0; v[:, -1] = 0; v[0, :] = 0; v[-1, :] = 0

u[-1, inlet_start:inlet_end] = 1.0

diff_u = np.max(np.abs(u - u_old))
diff_v = np.max(np.abs(v - v_old))
diff_p = np.max(np.abs(p - p_old))
diff_T = np.max(np.abs(T - T_old))

max_diff = max(diff_u, diff_v, diff_p, diff_T)
print(f"Max difference: {max_diff:.2e}")

if max_diff < epsilon:
    print(f"Solution converged at step {step+1}")
    break

if step in [0, 10, 50, 100, 200, 300, 400, 499]:
    plt.figure(figsize=(15, 10))

    plt.subplot(2, 2, 1)
    vel_mag = np.sqrt(u**2 + v**2)
    plt.contourf(X, Y, vel_mag, levels=20, cmap='jet')
    plt.colorbar(label='Velocity Magnitude')
    plt.streamplot(X, Y, u, v, color='k', density=1.5)
    plt.title(f'Velocity Field (Iteration {step+1})')
    plt.xlabel('x'); plt.ylabel('y')

    plt.subplot(2, 2, 2)
    plt.contourf(X, Y, p, levels=20, cmap='viridis')
    plt.colorbar(label='Pressure')
    plt.title(f'Pressure Field (Iteration {step+1})')
    plt.xlabel('x'); plt.ylabel('y')
```

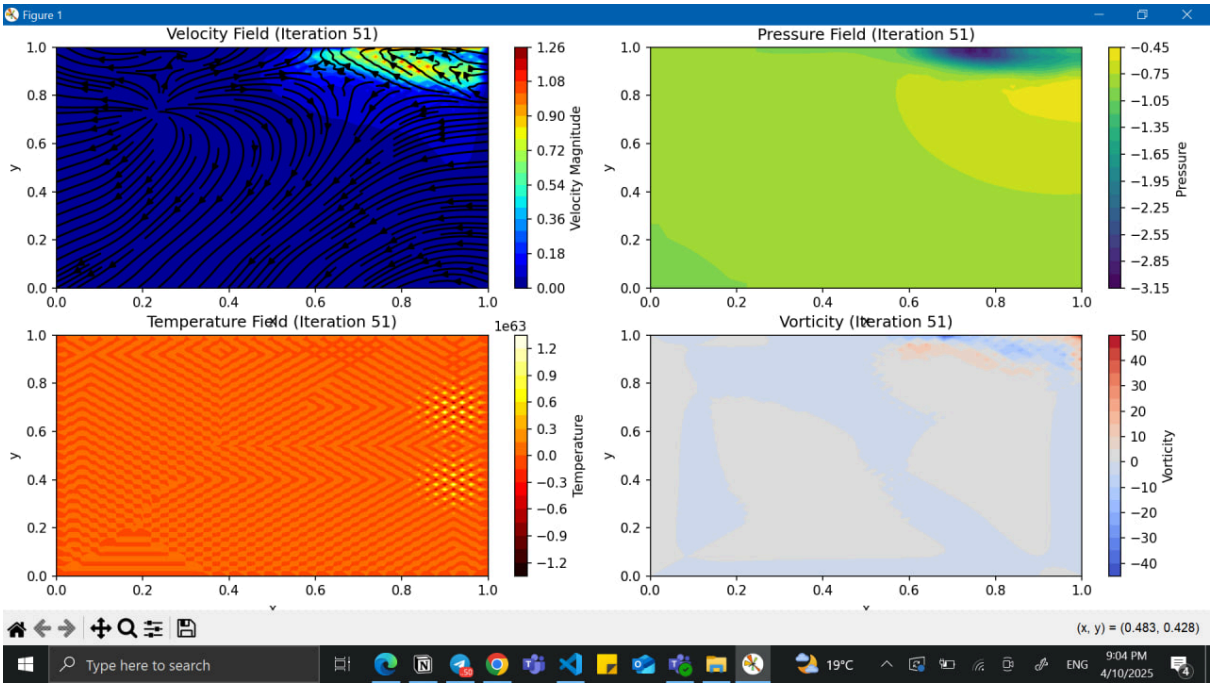
```
        plt.subplot(2, 2, 3)
        plt.contourf(X, Y, T, levels=20, cmap='hot')
        plt.colorbar(label='Temperature')
        plt.title(f'Temperature Field (Iteration {step+1})')
        plt.xlabel('x'); plt.ylabel('y')

        plt.subplot(2, 2, 4)
        vorticity = (np.gradient(v, axis=0) - np.gradient(u, axis=1)) / dx
        plt.contourf(X, Y, vorticity, levels=20, cmap='coolwarm')
        plt.colorbar(label='Vorticity')
        plt.title(f'Vorticity (Iteration {step+1})')
        plt.xlabel('x'); plt.ylabel('y')

        plt.tight_layout()
        plt.show()

fig = plt.figure(figsize=(18, 12))

ax1 = fig.add_subplot(2, 2, 1)
vel_mag = np.sqrt(u**2 + v**2)
cf1 = ax1.contourf(X, Y, vel_mag, levels=20, cmap='jet')
plt.colorbar(cf1, ax=ax1, label='Velocity Magnitude')
ax1.streamplot(X, Y, u, v, color='k', density=1.5)
ax1.set_title('Final Velocity Field')
ax1.set_xlabel('x'); ax1.set_ylabel('y')

ax2 = fig.add_subplot(2, 2, 2)
cf2 = ax2.contourf(X, Y, p, levels=20, cmap='viridis')
plt.colorbar(cf2, ax=ax2, label='Pressure')
ax2.set_title('Final Pressure Field')
ax2.set_xlabel('x'); ax2.set_ylabel('y')

ax3 = fig.add_subplot(2, 2, 3)
cf3 = ax3.contourf(X, Y, T, levels=20, cmap='hot')
plt.colorbar(cf3, ax=ax3, label='Temperature')
ax3.set_title('Final Temperature Field')
ax3.set_xlabel('x'); ax3.set_ylabel('y')
```

```
ax4 = fig.add_subplot(2, 2, 4)
vorticity = (np.gradient(v, axis=0) - np.gradient(u, axis=1)) / dx
cf4 = ax4.contourf(X, Y, vorticity, levels=20, cmap='coolwarm')
plt.colorbar(cf4, ax=ax4, label='Vorticity')
ax4.set_title('Final Vorticity Field')
ax4.set_xlabel('x'); ax4.set_ylabel('y')

plt.tight_layout()
plt.show()
```

Graph







Conclusion

his simulation shows how heat and fluid flow behave under the given conditions. The hot wall heats up the fluid, while the flow carries this heat through the domain. The fluid moves fastest at the inlet and slows down near solid walls due to friction. Pressure is highest where the fluid enters and drops as it flows out. The graphs will clearly show these patterns—temperature

changes, flow direction, and pressure differences. By tweaking the setup, like adjusting wall temperatures or flow speeds, we can see how these patterns shift. Overall, it helps us understand the physics of heat and fluid movement in the system.