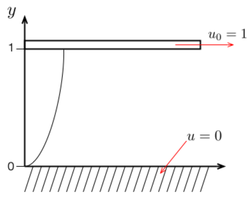
PDE python exercise 5

In this exercise we are looking for the solution of a confined, unsteady couette flow. This problem is controlled by a parabolic PDE and is illustrated below. We consider the unsteady Couette flow, confined by two walls, which has the following governing equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial y^2}$$

The system we are looking at is described in the following picture, we are looking a fluid constrained between two rigid walls, one fixed and one moving with a constant velocity u_0. We are interested by the distribution of the velocity along the y axis within the time frame [0,0.4].



where the velocity u:

1. $u = 0$ when $y = 0$
2. $u = u_0$ when $y = 1$
3. driven by the PDE when $0 < y < 1$

As initial conditions we have $u = 0$ for $0 < y < 1$

In this exercise, you have to:

1. Discretize the PDE using the FTCS scheme (i.e forward in time and centered in space)
2. Implement a function which solve the PDE using the FTCS scheme. As input we want the initial and boundary conditions as well as the number of points along the y axis and the number of time step. The diffusion number D will be computed inside this function. This function should return the complete history of the velocity u (i.e. a matrix). In this function you can use `for` loops to fill out the columns of u.
3. Run the problem with 10 grid points along y and 1001 timesteps Δt and plot the distribution of the velocity along y for the timesteps 100, 500 and 1000
4. Run the problem with 10 grid points along y and 11 timesteps Δt and plot the distribution of the velocity along y for the timesteps 1, 5 and 10. What can you observe?
5. Define a new function where you solve the PDE problem using array slicing instead of for loops and repeat question 3.

Question 1

For the FTCS scheme, we make use of centered finite difference for the space term:

$$\left.\frac{\partial^2 u}{\partial y^2}\right|_j^n \approx \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta y)^2}$$

and a forward difference for the time term:

$$\left.\frac{\partial u}{\partial t}\right|_j^n \approx \frac{u_j^{n+1} - u_j^n}{\Delta t}$$

where the index j indicates the space coordinate and the index n represents the time.

Once combined we expressed the FTCS scheme as:

$$u_j^{n+1} = D\left(u_{j+1}^n + u_{j-1}^n\right) + (1 - 2D)\,u_j^n$$

where D is the diffusion number and is defined as:

$$D = \frac{\Delta t}{(\Delta y)^2}$$

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In the next cell we define a function which computes the complete history of the velocity u with as input the discretization along the y axis and the time t. We have defined the initial and boundary conditions into the function.

In [ ]:
```
def compute_u(Ny,Nt):
    y    = np.linspace(0, 1, Ny)
    time = np.linspace(0,0.4,Nt)
    dy   = y[1] - y[0]
    dt   = time[1]-time[0]
    # Compute the diffusion number
    D = dt/dy**2.0
    U = np.zeros([Nt,Ny])
    # Enforce boundary and initial conditions
    U[:,0]  = 0
    U[:,-1] = 1
    # Loop over time
    for n in range(1,Nt):
        # Loop over space
        for j in range(1,Ny-1):
            U[n,j] = D*(U[n-1,j+1]+U[n-1,j-1])+(1-2*D)*U[n-1,j]
    return U
```

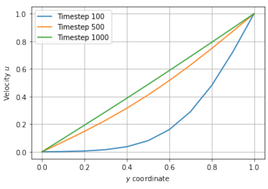
We first run the simulations with 1001 timesteps for 11 points along the y axis and plot the results.

In [ ]:
```
U = compute_u(11,1001)
```

In [ ]:
```
y = np.linspace(0,1,11)

plt.plot(y,U[100,:],label='Timestep 100')
plt.plot(y,U[500,:],label='Timestep 500')
plt.plot(y,U[1000,:],label='Timestep 1000')
plt.grid()
plt.legend()
plt.xlabel(r'$y$ coordinate')
plt.ylabel(r'Velocity $u$')
```

Out[ ]:
```
Text(0, 0.5, 'Velocity $u$')
```

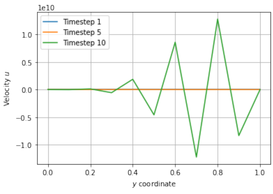


We then re-run the same simulation using 11 timesteps and 11 points along the y coordinate. The results are very different from the previous run showing an unstable behaviour from the numerical scheme.

In [ ]:
```
U = compute_u(11,11)
y = np.linspace(0,1,11)

plt.plot(y,U[1,:],label='Timestep 1')
plt.plot(y,U[5,:],label='Timestep 5')
plt.plot(y,U[10,:],label='Timestep 10')
plt.grid()
plt.legend()
plt.xlabel(r'$y$ coordinate')
plt.ylabel(r'Velocity $u$')
```

Out[ ]:
```
Text(0, 0.5, 'Velocity $u$')
```



For the last question we re-code our function using array slicing instead of a `for` loop to compute the space part of our numerical scheme.

In [ ]:
```
def compute_u_array(Ny,Nt):
    y    = np.linspace(0, 1, Ny)
    time = np.linspace(0,0.4,Nt)
    dy   = y[1] - y[0]
    dt   = time[1]-time[0]
    # Compute the diffusion number
    D = dt/dy**2.0
    U = np.zeros([Nt,Ny])
    # Enforce boundary and initial conditions
    U[:,0]  = 0
    U[:,-1] = 1
    # Loop over time
    for n in range(1,Nt):
        U[n,1:-1] = D*(U[n-1,2:]+U[n-1,:-2])+(1-2*D)*U[n-1,1:-1]
    return U
```

In [ ]:
```
y = np.linspace(0,1,11)
U = compute_u(11,1001)
plt.plot(y,U[100,:],label='Timestep 100')
plt.plot(y,U[500,:],label='Timestep 500')
plt.plot(y,U[1000,:],label='Timestep 1000')
plt.grid()
plt.legend()
plt.xlabel(r'$y$ coordinate')
plt.ylabel(r'Velocity $u$')
```

Out[ ]:
```
Text(0, 0.5, 'Velocity $u$')
```