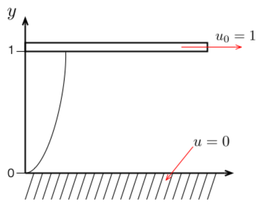
PDE python exercise 6

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 number of points along the y axis and the number of time steps. The diffusion number D will therefore be computed inside this function. This function should return the complete history of the velocity u (i.e. a matrix). The initial and boundary conditions will be enforced inside this function. columns of u.
3. Implement a function which solve the PDE using the Dufort-Frankel scheme. As input we want the number of points along the y axis and the number of time steps. The diffusion number D will therefore be computed inside this function. This function should return the complete history of the velocity u (i.e. a matrix). The initial and boundary conditions will be enforced inside this function. columns of u.
4. 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 using both the FTCS (question 3) and the Dufort-Frankel scheme (question 4).
5. 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 using the the Dufort-Frankel scheme (question 4). What can you observe?

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:

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

We first import our mandatory python modules.

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

We implement a function which computes the solution of the proposed PDE problem using the FTCS scheme.

In [ ]:
```
def compute_u_FTCS(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 implement another function which computes the solution using the Dufort-Frankel scheme.

In [ ]:
```
def compute_u_Dufort_Frankel(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):
            if n < 2:
                U[n,j] = (1.0/(1.0+2*D))*(2.0*D*(U[n-1,j+1]+U[n-1,j-1])+(1.0-2*D)*U[0,j])
            else:
                U[n,j] = (1.0/(1.0+2*D))*(2.0*D*(U[n-1,j+1]+U[n-1,j-1])+(1.0-2*D)*U[n-2,j])
    return U
```

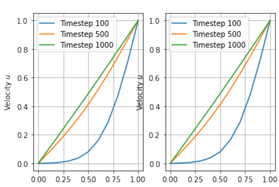
We then plot the solution obtained using the two numerical schemes for different timesteps. We observe that the two schemes are yielding the same results.

In [ ]:
```
y = np.linspace(0,1,11)
U_FTCS = compute_u_FTCS(11,1001)
U_DF = compute_u_Dufort_Frankel(11,1001)

fig,axs = plt.subplots(1,2)
axs[0].plot(y,U_FTCS[100,:],label='Timestep 100')
axs[0].plot(y,U_FTCS[500,:],label='Timestep 500')
axs[0].plot(y,U_FTCS[1000,:],label='Timestep 1000')
axs[0].grid()
axs[0].legend()
axs[0].set_xlabel(r'$y$ coordinate')
axs[0].set_ylabel(r'Velocity $u$')

axs[1].plot(y,U_DF[100,:],label='Timestep 100')
axs[1].plot(y,U_DF[500,:],label='Timestep 500')
axs[1].plot(y,U_DF[1000,:],label='Timestep 1000')
axs[1].grid()
axs[1].legend()
axs[1].set_xlabel(r'$y$ coordinate')
axs[1].set_ylabel(r'Velocity $u$')
```

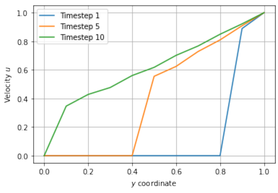
Out[ ]:
```
Text(0, 0.5, 'Velocity $u$')
```



We then re-run the simulation with 11 timesteps and 11 grid points using the Dufort-Frankel scheme. The solution with the FTCS scheme was shown earlier (Python exercise 5) to be unstable. Here, we do not see large oscillations using the Dufort-Frankel scheme but the results are still different from the ones of the previous questions. Even though the applied numerical scheme is stable we still need enough timesteps to discretize the problem properly.

In [ ]:
```
U_DF_2 = compute_u_Dufort_Frankel(11,11)
plt.plot(y,U_DF_2[1,:],label='Timestep 1')
plt.plot(y,U_DF_2[5,:],label='Timestep 5')
plt.plot(y,U_DF_2[10,:],label='Timestep 10')
plt.grid()
plt.legend()
plt.xlabel(r'$y$ coordinate')
plt.ylabel(r'Velocity $u$')
```

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



In [ ]:
```
```