MATH 352 - Spring 2021
Homework 2

Dr. Alessandro Veneziani

Kai Chang, Kevin Hong, Rodrigo Gonzalez

- **Verify the exact solution.**

  First, we check that it satisfies the PDE:

$$\frac{d}{dx}u_{ex} = P\frac{e^{Px}}{e^P-1} = \frac{\beta}{\mu}\frac{e^{Px}}{e^P-1}$$
$$\frac{d^2}{dx^2}u_{ex} = P^2\frac{e^{Px}}{e^P-1} = \frac{\beta^2}{\mu^2}\frac{e^{Px}}{e^P-1}$$

  Substituting:

$$-\mu\left(\frac{\beta^2}{\mu^2}\frac{e^{Px}}{e^P-1}\right) + \beta\left(\frac{\beta}{\mu}\frac{e^{Px}}{e^P-1}\right) = -\frac{\beta^2}{\mu}\frac{e^{Px}}{e^P-1} + \frac{\beta^2}{\mu}\frac{e^{Px}}{e^P-1} = 0$$

  Then, we check the BC's:

$$u_{ex}(0) = \frac{e^0-1}{e^P-1} = \frac{1-1}{e^P-1} = 0$$
$$u_{ex}(1) = \frac{e^P-1}{e^P-1} = 1$$

- **Finite difference second order discretization.**

  At a point x_i in the domain, we approximate $u_i = u(x_i)$ with the following second order approximations:

$$\frac{d}{dx}u_i \approx \frac{u_{i+1}-u_{i-1}}{2\Delta x}$$
$$\frac{d^2}{dx^2}u_i \approx \frac{u_{i+1}-2u_i+u_{i-1}}{\Delta x^2}$$

  Substituting into the PDE, we obtain, for $i = 1, \ldots, m-1$, the following linear equation

$$-\mu\frac{u_{i+1}-2u_i+u_{i-1}}{\Delta x^2}+\beta\frac{u_{i+1}-u_{i-1}}{2\Delta x}=0$$

Given the boundary conditions, $u_0 = 0$, $u_m = 1$. Therefore, we pay special attention to the $i = 1$ and $i = m - 1$ cases:

$$-\mu\frac{u_2-2u_1}{\Delta x^2}+\beta\frac{u_2}{2\Delta x}=0$$

$$-\mu\frac{-2u_{m-1}+u_{m-2}}{\Delta x^2}+\beta\frac{-u_{m-2}}{2\Delta x}=\frac{\mu}{\Delta x^2}-\frac{\beta}{2\Delta x}$$

Thus, we may formulate the problem in a matrix form $A\vec{u} = \vec{b}$, where $A \in \mathbb{R}^{m-1\times m-1}$ and $\vec{b}, \vec{u} \in \mathbb{R}^{m-1}$ are given by:

$$A=\begin{bmatrix} \mathrm{a} & \mathrm{b} & 0 & 0 & ... & 0 \\ \mathrm{c} & \mathrm{a} & \mathrm{b} & 0 & ... & 0 \\ 0 & \mathrm{c} & \mathrm{a} & \mathrm{b} & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & ... & \mathrm{c} & \mathrm{a} \end{bmatrix}$$

$$a=\frac{2\mu}{\Delta x^2},\ b=-\frac{\mu}{\Delta x^2}+\frac{\beta}{2\Delta x},\ c=-\frac{\mu}{\Delta x^2}-\frac{\beta}{2\Delta x}$$

$$u=\begin{bmatrix} u_1 \\ u_2 \\ ... \\ ... \\ u_{m-1} \end{bmatrix} b=\begin{bmatrix} 0 \\ 0 \\ ... \\ ... \\ b_{m-1} \end{bmatrix},\ \text{where m} = \lfloor\frac{1}{\Delta x}\rfloor \text{ and } b_{m-1}=\frac{\mu}{\Delta x^2}-\frac{\beta}{2\Delta x}.$$

- **Test the expected accuracy for $\beta > 0$.**

  Using $\mu = 10$, and letting $\Delta x = h$ we obtained the following results:

beta	h	error	Pe
1.0	0.100	1.041032e-07	0.00500
1.0	0.050	2.602549e-08	0.00250
1.0	0.025	6.506356e-09	0.00125
1.0	0.010	1.041295e-09	0.00050
1.0	0.005	2.603148e-10	0.00025
10.0	0.100	1.006860e-04	0.05000
10.0	0.050	2.514359e-05	0.02500
10.0	0.025	6.291757e-06	0.01250
10.0	0.010	1.006797e-06	0.00500
10.0	0.005	2.516964e-07	0.00250
100.0	0.100	3.452870e-02	0.50000
100.0	0.050	7.874142e-03	0.25000
100.0	0.025	1.927742e-03	0.12500
100.0	0.010	3.066741e-04	0.05000
100.0	0.005	7.660603e-05	0.02500
1000.0	0.100	6.961247e-01	5.00000
1000.0	0.050	4.353094e-01	2.50000
1000.0	0.025	1.931961e-01	1.25000
1000.0	0.010	3.454611e-02	0.50000
1000.0	0.005	7.879441e-03	0.25000

We notice that as long as $\mathbb{P}\mathrm{e} < 1$, the error scales with $(\Delta x)^2$, as expected.

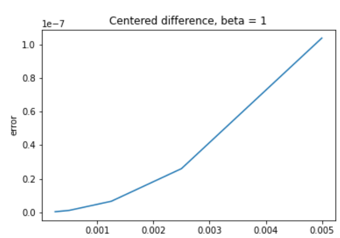


Figure 1: error against the Peclet numbe

- **Repeat for $\beta < 0$.**

  Similarly, using $\mu = 10$, and letting $\Delta x = h$ we obtained the following results:

beta	h	error	Pe
-1.0	0.100	1.041032e-07	0.00500
-1.0	0.050	2.602550e-08	0.00250
-1.0	0.025	6.506355e-09	0.00125
-1.0	0.010	1.041294e-09	0.00050
-1.0	0.005	2.603306e-10	0.00025
-10.0	0.100	1.006860e-04	0.05000
-10.0	0.050	2.514359e-05	0.02500
-10.0	0.025	6.291757e-06	0.01250
-10.0	0.010	1.006797e-06	0.00500
-10.0	0.005	2.516964e-07	0.00250
-100.0	0.100	3.452870e-02	0.50000
-100.0	0.050	7.874142e-03	0.25000
-100.0	0.025	1.927742e-03	0.12500
-100.0	0.010	3.066741e-04	0.05000
-100.0	0.005	7.660603e-05	0.02500
-1000.0	0.100	6.961247e-01	5.00000
-1000.0	0.050	4.353094e-01	2.50000
-1000.0	0.025	1.931961e-01	1.25000
-1000.0	0.010	3.454611e-02	0.50000
-1000.0	0.005	7.879441e-03	0.25000

  We notice that as long as $\mathbb{P}\mathrm{e} < 1$, the error scales with $(\Delta x)^2$, as expected.

- **Coarsest discretization to avoid oscillations.**

  The condition to avoid oscillations is

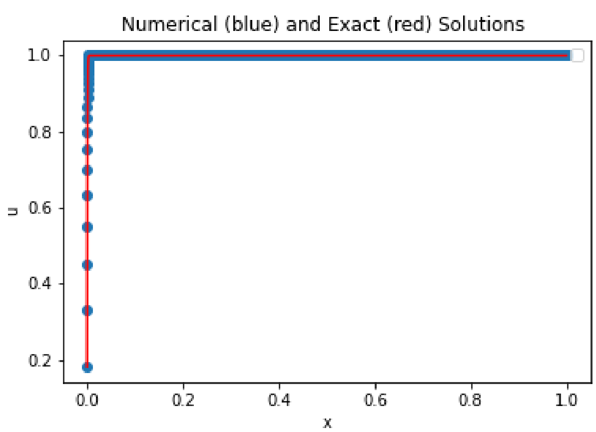
$$\mathbb{P}e = \frac{|\beta|\Delta x}{2\mu} < 1$$

  .

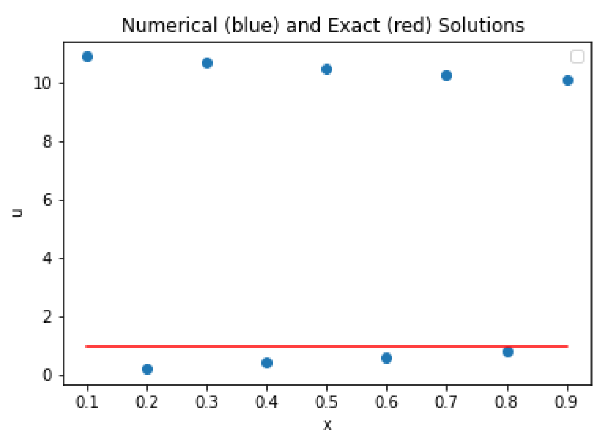
  Using $\beta = -200$, and $\mu = 0.1$, we obtain an upper bound for Δx:

$$\Delta x < \frac{2\mu}{|\beta|} = 1 \times 10^{-3}$$

  To test this, we may use $(\Delta x)_1 = 0.0001 < 1 \times 10^{-3}$, and obtain the following graph with no oscillations



  Now, we use $(\Delta x)_2 = 0.1 > 1 \times 10^{-3}$, and obtain the following graph with clear oscillations



- **Upwind method for $\beta > 0$.**

  We prioritize data coming from the upwind direction. Given that $\beta > 0$, this direction corresponds to lower values of x. Therefore, we can replace the first derivative with the following first-order approximation.

  $$\frac{d}{dx}u_i \approx \frac{u_i - u_{i-1}}{\Delta x}$$

  .

  Note that we include the point in the upwind direction, u_{i-1}.

  The scheme becomes:

  $$-\mu\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + \beta\frac{u_i - u_{i-1}}{\Delta x} = 0$$

  Which we write in matrix form $A\vec{u} = \vec{b}$, with

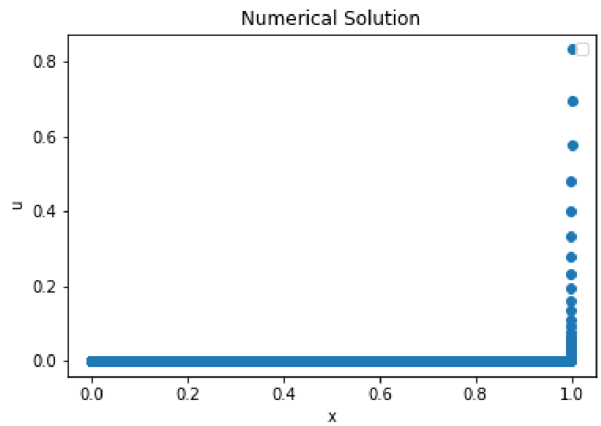
  $$A = \begin{bmatrix} a & b & 0 & 0 & ... & 0 \\ c & a & b & 0 & ... & 0 \\ 0 & c & a & b & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & ... & c & a \end{bmatrix}$$

  $$a = \frac{2\mu}{\Delta x^2} + \frac{\beta}{\Delta x},\ b = -\frac{\mu}{\Delta x^2},\ c = -\frac{\mu}{\Delta x^2} - \frac{\beta}{\Delta x}$$

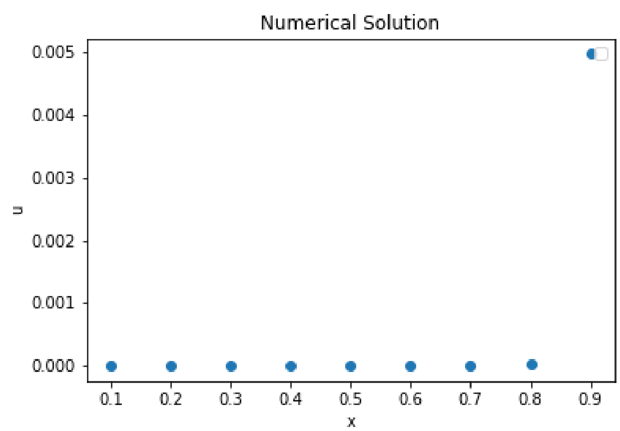
  $$u = \begin{bmatrix} u_1 \\ u_2 \\ ... \\ ... \\ u_{m-1} \end{bmatrix} b = \begin{bmatrix} 0 \\ 0 \\ ... \\ ... \\ b_{m-1} \end{bmatrix},\ \text{where m} = \lfloor\frac{1}{\Delta x}\rfloor \text{ and } b_{m-1} = \frac{\mu}{\Delta x^2}.$$

We let $\beta = 200$ and $\mu = 0.1$, which were the same values used before with the second order method which showed oscillations.

For $(\Delta x)_1 = 0.0001 < 1 \times 10^{-3}$, we obtain the following graph.



For $(\Delta x)_2 = 0.1 > 1 \times 10^{-3}$, we obtain the following graph.



Note that now we avoid oscillations in any case.

- **Upwind method for $\beta < 0$**

  We prioritize data coming from the upwind direction. Given that $\beta < 0$, this direction corresponds to higher values of x. Therefore, we can replace the first derivative with the following first-order approximation.

$$\frac{d}{dx}u_i \approx \frac{u_{i+1} - u_i}{\Delta x}$$

  .

  Note that we include the point in the upwind direction, u_{i+1}.

  The scheme becomes:

$$-\mu\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + \beta\frac{u_{i+1} - u_i}{\Delta x} = 0$$

  Which we write in matrix form $A\vec{u} = \vec{b}$, with

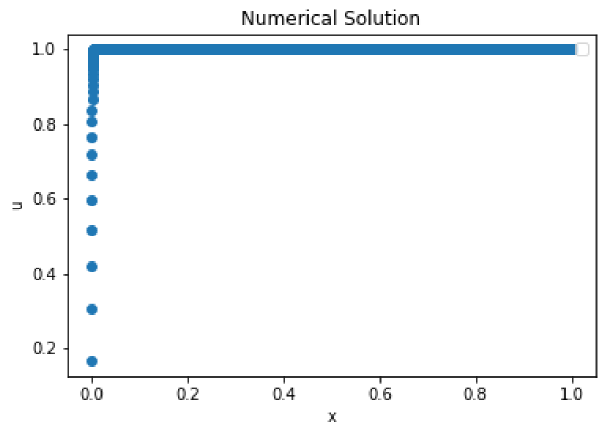
$$A = \begin{bmatrix} a & b & 0 & 0 & ... & 0 \\ c & a & b & 0 & ... & 0 \\ 0 & c & a & b & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & ... & c & a \end{bmatrix}$$

$$a = \frac{2\mu}{\Delta x^2} - \frac{\beta}{\Delta x},\ b = -\frac{\mu}{\Delta x^2} + \frac{\beta}{\Delta x},\ c = -\frac{\mu}{\Delta x^2}$$

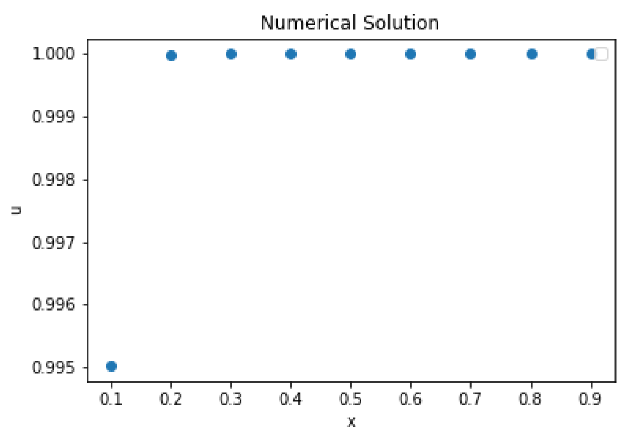
$$u = \begin{bmatrix} u_1 \\ u_2 \\ ... \\ ... \\ u_{m-1} \end{bmatrix} b = \begin{bmatrix} 0 \\ 0 \\ ... \\ ... \\ b_{m-1} \end{bmatrix},\ \text{where m} = \lfloor\frac{1}{\Delta x}\rfloor \text{ and } b_{m-1} = \frac{\mu}{\Delta x^2} - \frac{\beta}{\Delta x}.$$

We let $\beta = -200$ and $\mu = 0.1$, which were the same values used before with the second order method which showed oscillations.

For $(\Delta x)_1 = 0.0001 < 1 \times 10^{-3}$, we obtain the following graph.



For $(\Delta x)_2 = 0.1 > 1 \times 10^{-3}$, we obtain the following graph.



Note that now we avoid oscillations in any case.

- **Convergence rate for upwind methods**

  Given that for upwind methods we use a first order approximation for the derivative, we expect these methods to be of first order. To test this, we let $\mu = 1$, $\beta = 20$, and compute the errors for $\Delta x = 0.001$, $\Delta x/2$, and $\Delta x/4$. The results were as follows.

beta	h	error	Pe
20.0	0.00100	0.003648	0.0100
20.0	0.00050	0.001832	0.0050
20.0	0.00025	0.000918	0.0025

  We notice that the error scales with Δx, as expected.

- **Second order upwind method**

  It is possible as long as we use a one-sided, second-order approximation for $\frac{d}{dx}u_i$.

  For $\beta > 0$, we use the following second-order approximation

$$\frac{d}{dx}u_i \approx \frac{u_{i-2} - 4u_{i-1} + 3u_i}{2\Delta x}$$

.

The scheme becomes:

$$-\mu\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + \beta\frac{u_{i-2} - 4u_{i-1} + 3u_i}{2\Delta x} = 0$$

Which we write in matrix form $A\vec{u} = \vec{b}$, where $A \in \mathbb{R}^{m-1\times m-1}$ with

$$A = \begin{bmatrix} a_{11} & b_{12} & 0 & 0 & 0 & ... & 0 \\ c & a & b & 0 & 0 & ... & 0 \\ d & c & a & b & 0 & ... & 0 \\ 0 & d & c & a & b & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & ... & d & c & a \end{bmatrix}$$

$$a_{11} = \frac{2\mu}{\Delta x^2},\ b_{12} = -\frac{\mu}{\Delta x^2} + \frac{\beta}{2\Delta x}$$

$$a = \frac{2\mu}{\Delta x^2} + \frac{3\beta}{2\Delta x},\ b = -\frac{\mu}{\Delta x^2},\ c = -\frac{\mu}{\Delta x^2} - \frac{2\beta}{\Delta x},\ d = \frac{\beta}{2\Delta x}$$

$$u = \begin{bmatrix} u_1 \\ u_2 \\ ... \\ ... \\ u_{m-1} \end{bmatrix} b = \begin{bmatrix} 0 \\ 0 \\ ... \\ ... \\ b_{m-1} \end{bmatrix},\ \text{where m} = \lfloor\frac{1}{\Delta x}\rfloor \text{ and } b_{m-1} = \frac{\mu}{\Delta x^2}.$$

a_{11} and b_{12} do not follow the pattern because we do not have enough data to use the second-order upwind scheme in the corner case, i.e. the case of i = 1. We use the centered discretization instead. This does not affect the accuracy because the centered scheme is also of second-order. This modification yields two distinctive entries a_{11} and b_{12}.

For $\beta < 0$, we use the following second-order approximation

$$\frac{d}{dx}u_i \approx \frac{-u_{i+2} + 4u_{i+1} - 3u_i}{2\Delta x}$$

.

This was deducted by using the Taylor's Formula

The scheme becomes:

$$-\mu \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + \beta \frac{-u_{i+2} + 4u_{i+1} - 3u_i}{2\Delta x} = 0$$

Which we write in matrix form $A\vec{u} = \vec{b}$, where $A \in \mathbb{R}^{m-1 \times m-1}$ with

$$A = \begin{bmatrix} a & b & c & 0 & 0 & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & ... & d & a & b & c & 0 \\ 0 & ... & 0 & d & a & b & c \\ 0 & ... & 0 & 0 & d & a & b \\ 0 & ... & 0 & 0 & 0 & d_{-12} & a_{-11} \end{bmatrix}$$

$$a_{-11} = \frac{2\mu}{\Delta x^2}, \; d_{-12} = -\frac{\mu}{\Delta x^2} - \frac{\beta}{2\Delta x}$$

$$a = \frac{2\mu}{\Delta x^2} - \frac{3\beta}{2\Delta x}, \; b = -\frac{\mu}{\Delta x^2} + \frac{2\beta}{\Delta x}, \; c = -\frac{\beta}{2\Delta x}, \; d = -\frac{\mu}{\Delta x^2}$$

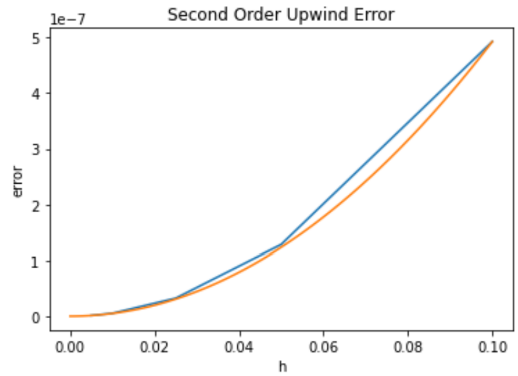
$$u = \begin{bmatrix} u_1 \\ u_2 \\ ... \\ ... \\ u_{m-1} \end{bmatrix} b = \begin{bmatrix} 0 \\ 0 \\ ... \\ ... \\ b_{m-1} \end{bmatrix}, \text{ where m} = \lfloor \frac{1}{\Delta x} \rfloor \text{ and } b_{m-1} = \frac{\mu}{\Delta x^2} - \frac{\beta}{2\Delta x}.$$

a_{-11} and d_{-12} do not follow the pattern because we do not have enough data to use the second-order upwind scheme in the corner case, i.e. the case of i = m-1. We use the centered discretization instead. This does not affect the accuracy because the centered scheme is also of second-order. This modification yields two distinctive entries a_{-11} and d_{-12}.

For the coding implementation and explanation, please refer to the Appendix.

Using $\beta = 10$, and different values of Δx, we see that the error scales with $(\Delta x)^2$, as expected.

	beta	h	error	Pe
0	10.0	0.000100	1.523571e-05	0.005000
0	10.0	0.000050	3.820538e-06	0.002500
0	10.0	0.000025	9.565708e-07	0.001250
0	10.0	0.000013	2.392636e-07	0.000625
0	10.0	0.000006	5.959596e-08	0.000313


Second Order Upwind Error
1e−7
5
4
3
2
1
0
error
0.00
0.02
0.04
0.06
0.08
0.10
h

Appendix

Codes

We chose to write our codes in python.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse as sp
import scipy.sparse.linalg as spla
from numpy import linalg as LA
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd

# returns the exact solution
def u_ex(x,mu,beta): return (np.exp(beta*x/mu) - 1)/(np.exp(beta/mu) - 1)

# return an array of zeros of size len(x)
def f(x): return [0]*len(x)

# the centered-discretization solver
def f_center(a, b, mu, beta, h):
    # check if mu is positive
    if mu < 0:
      print('negative mu')

    # a and b are boundary points; h is the step change of x
    a, b, mu, beta, h = a, b, mu, beta, h

    # number of steps in the interval
    N = int((b-a)/h)

    # discretize the space
    x = np.linspace(a,b,N+1)

    # omit the end points since they are not considered in our system of equations
    y = x[1:-1]

    # we use ff to represent the vector b we referred to in our analysis
    # initialize ff, which should have the same length as y
    ff = f(y)

    # split the matrix A into two parts
    # in the centered solver, the sign of beta does not affect the matrix A
    Ad = -(mu/h**2)*sp.diags([1., -2., 1.], [-1, 0, 1], shape=[N-1, N-1], format = 'csr')
    Ac = beta/(2*h)*sp.diags([-1., 0., 1.], [-1, 0, 1], shape=[N-1, N-1], format = 'csr')

    # integrate to get A
    A = Ad + Ac

    # per our analysis, the last entry of ff is not empty and has the following value
    ff[-1] = mu/h**2 - beta/(2*h)
```

```
    # return the solution of Ax = ff
    return sp.linalg.spsolve(A, ff)

# 1st-order upwind solver
def f_upwind(a, b, mu, beta, h):
    # check if mu is positive
    if mu < 0:
      print('negative mu')

    # all the parameters are defined in the same way as those in the centered solver
    a, b, mu, beta, h = a, b, mu, beta, h
    N = int((b-a)/h)
    x = np.linspace(a,b,N+1)
    y = x[1:-1]
    ff = f(y)

    # the sign of beta matters in the upwind scheme
    # beta > 0, the upwind discretization is (u_i - u_{i-1})/(\Delta x)
    if beta > 0:
        Ad = -(mu/h**2)*sp.diags([1., -2., 1.], [-1, 0, 1], shape=[N-1, N-1], format = 'csr')
        Ac = beta/(h)*sp.diags([-1., 1., 0], [-1, 0, 1], shape=[N-1, N-1], format = 'csr')
        ff[-1] = mu/h**2

    # beta < 0, the upwind discretization is (u_{i+1} - u_i)/(\Delta x)
    else:
        Ad = -(mu/h**2)*sp.diags([1., -2., 1.], [-1, 0, 1], shape=[N-1, N-1], format = 'csr')
        Ac = beta/(h)*sp.diags([0, -1., 1.], [-1, 0, 1], shape=[N-1, N-1], format = 'csr')
        ff[-1] = mu/h**2 - beta/h

    A = Ad + Ac

    return sp.linalg.spsolve(A, ff)

# second-order upwind
def f_O2upwind(a,b,mu,beta,h):
    # check if mu is positive
    if mu<0:
      print('negative mu')

    # all the parameters are defined in the same way as those in the centered solver
    a,b,mu,beta,h = a,b,mu,beta,h

    N = int((b-a)/h)

    x = np.linspace(a,b,N+1)
    y = x[1:-1]

    ff = f(y)

    if beta > 0:
    Ad = -(mu/h**2)*sp.diags([1., -2., 1.], [-1, 0, 1], shape = [N-1,N-1], format = 'csr')
    Ac = (beta/h)*sp.diags([1/2, -2, 3/2], [-2, -1, 0], shape = [N-1, N-1], format = 'csr')
    ff[-1] = mu/h**2
    A = Ad+Ac
```

```
A[0,0] = 2*mu/h**2
A[0,1] = -mu/h**2 + beta/(2*h)
else:
Ad = -(mu/h**2)*sp.diags([1., -2., 1.], [-1, 0, 1], shape = [N-1,N-1], format = 'csr')
Ac = -(beta/h)*sp.diags([3/2, -2, 1/2], [0, 1, 2], shape = [N-1, N-1], format = 'csr')
A = Ad+Ac
ff[-1] = mu/h**2 - beta/(2*h)
A[-1,-1] = 2*mu/h**2
A[-1, -2] = -mu/h**2 - beta/(2*h)

return sp.linalg.spsolve(A, ff)
```

We coded everything exactly as how it is presented in our report. a and b stand for the boundary points; h is the step of change; mu and beta represent the constants in the problem. We discretize the space with N+1 points and store them in the array x. Since we don't need the end-point cases in our system of equations, we omit them by defining a new array y. The array ff stores the vector b which should have the same length as the array y. We construct the matrix A by splitting it into Ad and Ac. After the computation, we integrate them into A. However, there is a corner case that needs consideration: when beta ¿ 0, the corner case is i = 1; when beta ¡ 0, the corner case is i = N-1. In the corner cases, since we do not have enough data, we change to the centered discretization which end up with different coefficients. That's why we modify 2 entries of A in each case. Finally, we return the solution to the equation Ax = ff.