Assignment 4 - Catalytic tubular reactor

Group number 9, 1796097, 1732722

1 Summary

This assignment provides a guided set of steps to explore the progress of a reaction A into P inside a tubular catalytic reactor. The techniques used were explicit and implicit time discretization (as well as an exploration of the effects of changing the number of time steps) and Simpson integration and the investigation of the concentration profile's order of convergence. Finally, the effect of an oscillating concentration was investigated.

2 Results and discussion

1. a) To simulate the time evolution of the concentration of A over a 4 seconds operation, one generated a model for the transport of a A along the reactor length to solve the convection-diffusion-reaction equation:

$$\frac{\partial C_A}{\partial t} = D_a x \frac{\partial C_A^2}{\partial x^2} - u \frac{\partial C_A}{\partial x} + R \tag{1}$$

   The reactor's parameters include 100 grid points and 1000 time steps. Based on the equation above, the function can be created as can be seen in Listing 1.

   The dynamic plot of the time evolution against concentration can be observed running Listing 1.

```
def reaction(c):
    return -kR*c

plt.ion()
figure, ax = plt.subplots(figsize = (10, 5))
line = []
line += ax.plot(x, c)
plt.title(f"Time: {t:5.3f} s", fontsize = 16)
plt.xlabel('Axial position', fontsize = 14)
plt.ylabel('Concentration', fontsize=14)
plt.xlim(0, x_end)
plt.ylim(0, max(cL, cR))
plt.grid
iplot = 0

for n in range(Nt):
    t += dt
    c_old = np.copy(c) ## doing c_old = c then modification of c_old
       will also modify c
    c[0] = cL

    rxn = reaction(c_old)
    for i in range(1, Nx):
        if dx*i < 0.1 or dx*i>0.9:
            c[i] = c_old[i] + Fo *c_old[i-1] -2*Fo*c_old[i] + Fo*c_old
               [i+1] - Co*(c_old[i]-c_old[i-1]) # No reaction
        else:
            c[i] = c_old[i] + Fo *c_old[i-1] -2*Fo*c_old[i] + Fo*c_old
               [i+1] - Co*(c_old[i]-c_old[i-1]) + rxn[i]*dt
    c[Nx] = c[Nx-1]

    iplot += 1
    if (iplot % 10 == 0):
        plt.title(f"Time = {t:5.3f} s")
        line[0].set_ydata(c)
        figure.canvas.draw()
        figure.canvas.flush_events()
        iplot = 0
plt.show()
```

Listing 1: Explicit time discretization

Along the time interval, the curve tends to a gradually smaller negative slope and ends in a quasi-linear shape as a steady state is reached.

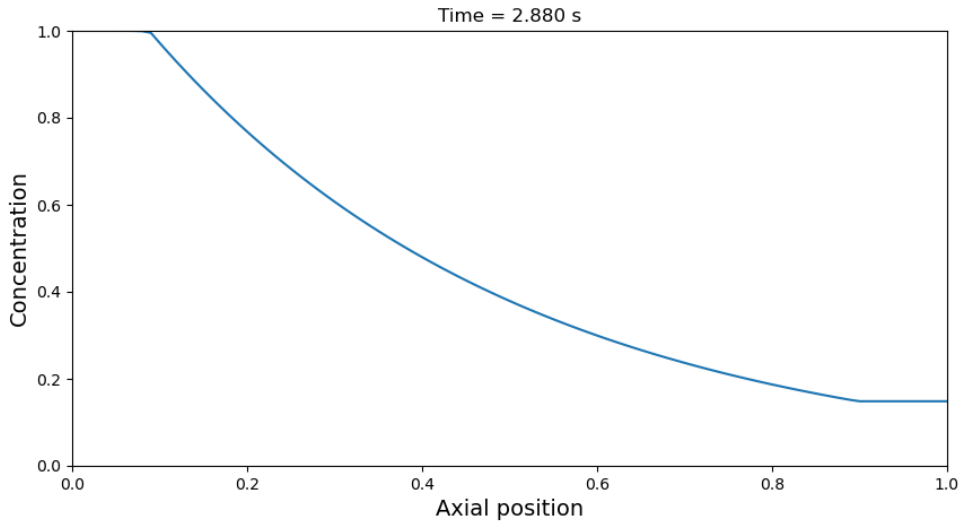

Figure 1: Plot of concentration in the tubular reactor at t=2.880s

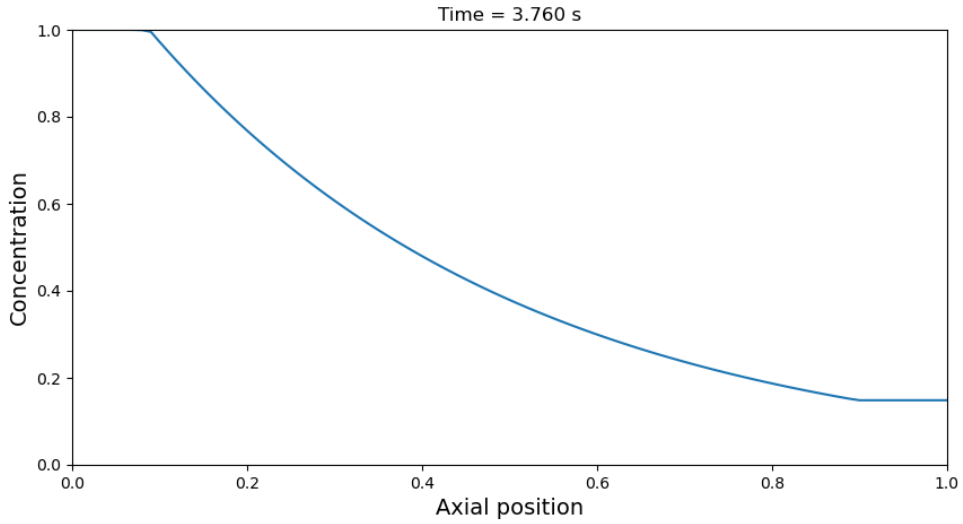

Figure 2: Plot of concentration in the tubular reactor at t=3.760s

As can be seen from Figure 1 and Figure 2, the shape of the concentration curve does not vary between these two times, thus we can say with confidence that a steady state has been reached.

b) By reducing the number of time steps from 1000 to 200 we should expect an unstable solution. This should originate from the fact that the Courant number will be greater than 1.

$$Co = \frac{u \cdot dt}{dx} \tag{2}$$

$$Co = 1.0026761414789407 \tag{3}$$

$$Co > 1 \tag{4}$$

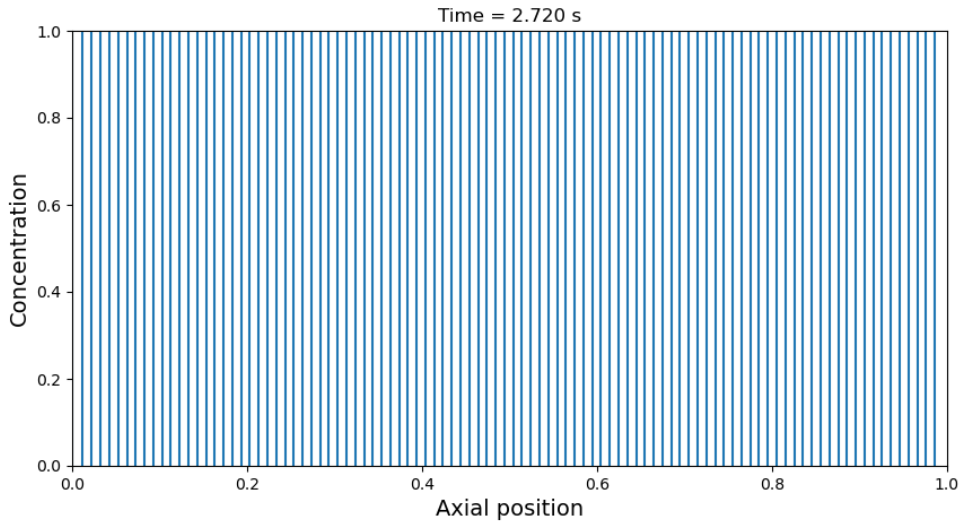

Figure 3: Unstable solution with 200 time steps

As can be seen in Figure 3 the resulting solution is nonsensical, thus, in order to solve the system properly, one must ensure that the Courant number is lower than 1.

c) Applying a central differencing scheme improves the numerical stability of the simulation as, by considering values at both sides of each of the 100 grid points, numerical artifacts are minimized and the convective term is better displayed. Such benefits are more evident at high slopes in the concentration profile.

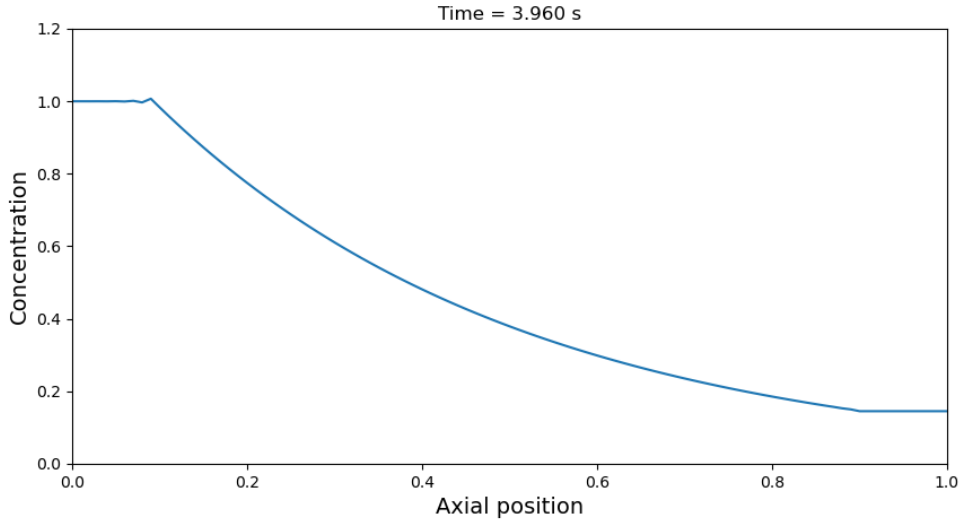

Figure 4: Solution using Central Differencing

As can be seen in Figure 4, the solution appears to be correct, however, some numerical artifacts can be observed particularly at axial position ≈ 0.1 we see that the concentration seemingly increases

and decreases.

2. a) In order to properly implement the implicit solution method, only the convective and reaction term was considered.

```
A = sps.diags([-(Co), (1+Co+kR*dt)], [-1, 0], shape=(Nx+1, Nx+1))
A = sps.csr_matrix(A)
A[0,0]=1
A[0,1] = 0
A[Nx, Nx] = 1
A[Nx, Nx] = -1
for i in range(1, Nx):
    if dx*i < 0.1 or dx*i>0.9:
        A[i, i] = A[i,i] - kR*dt ## Taking away the reaction term
```

Listing 2: Defining diagonals

Listing 2 shows how the matrix was defined for the implicit case. The main diagonal consisted of the previous concentration, Courant number, and reaction term, while the lower off-diagonal only contained the negative Courant number. As the reaction only takes place between in the region of axial position between 0.1 and 0.9, the for loop in the code is used to remove the reaction term for axial position less than 0.1 and greater than 0.9.[1] Listing 3 shows how the system was solved implicitly. The scipy.spares.lingalg (referred to as spss) section of the scipy library was used to solve the system.

```
for n in range(Nt):
        t += dt
        c_old = np.copy(c) ## doing c_old = c then modification of
            c_old will also modify c
        c[0] = cL

        b = c_old
        b[Nx[d]] = 0
        c = spss.spsolve(A, b)
```

Listing 3: Solving system implicitly

[1] The code is used the illustrate how the functions, however, in reality, the code is slightly different as a loop is employed to solve the system with a varying number of grid points.

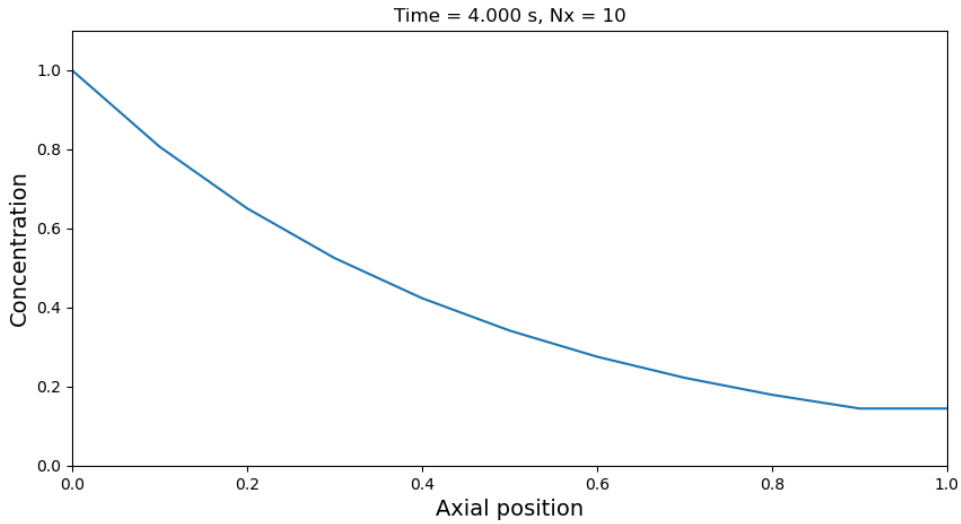

Figure 5: Solution with 10 grid points

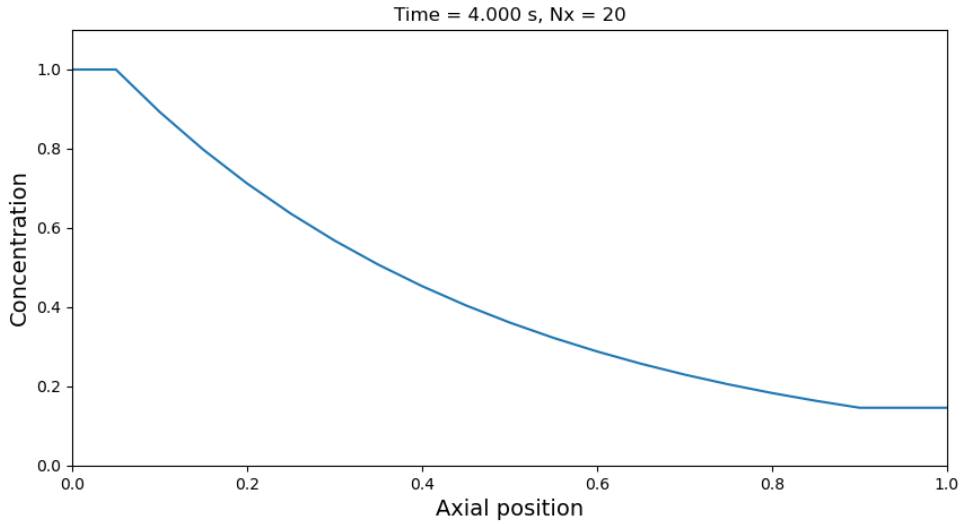

Figure 6: Solution with 20 grid points

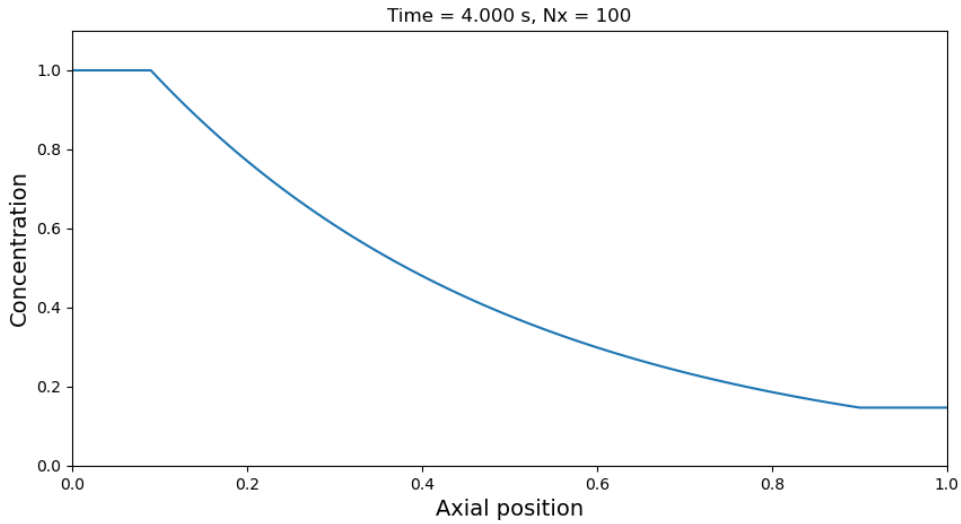

Figure 7: Solution with 100 grid points

As can be seen in Figure 5, Figure 6, and Figure 7 increasing the number of grid points, increases the accuracy of the solution. Using only 10 grid points does not properly illustrate the fact that the reaction only takes when the axial position is between 0.1 and 0.9. Using 20 and 100 grid points yields nearly identical solutions, however, the plateau extends with a higher number of grid points.

b-c) One constructed a sparse tridiagonal matrix A is constructed to represent the discretized system of equations. Boundary conditions were set on the old concentration profile on the first and last rows. The concentration profile at each time step is solved through the sparse linear system with the function 'spss.spsolve'.

```
A = sps.diags([-(Co+Fo), (1+Co+2*Fo+kR*dt), -Fo], [-1, 0, 1], shape=(
    Nx+1, Nx+1))
A = sps.csr_matrix(A)
A[0,0]=1
A[0,1] = 0
A[Nx, Nx] = 1
A[Nx, Nx-1] = -1
for i in range(1, Nx):
    if dx*i < 0.1 or dx*i>0.9:
        A[i, i] = A[i,i] - kR*dt ## Taking away the reaction term
```

Listing 4: Solving system implicitly with all terms

As can be seen in Listing 4, the implementation of the system is nearly identical to the previous task, however, now the Fourier term is included.

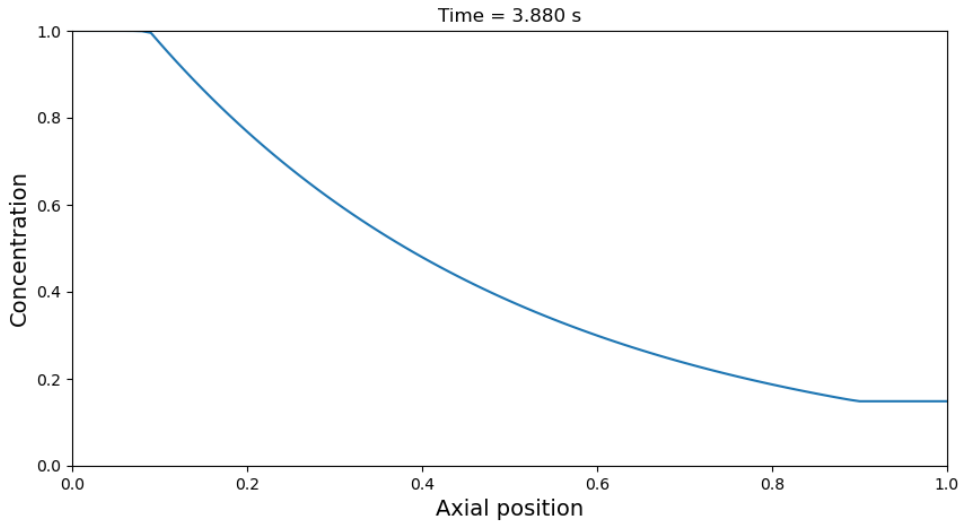

Figure 8: Solution including diffusing and convection solved implicitly

As can be seen in Figure 8, the solution is solved correctly as the figure directly matches the solution in Figure 2. The main difference between solving the system implicitly versus explicitly is that it takes less time to solve implicitly compared to explicitly.

d) Unlike task 1c, by using 200 grid points and solving the system implicitly, the resultant graph is stable. Moreover, the output graph directly matches the solution given in task c where the system was solved with 1000 grid points.

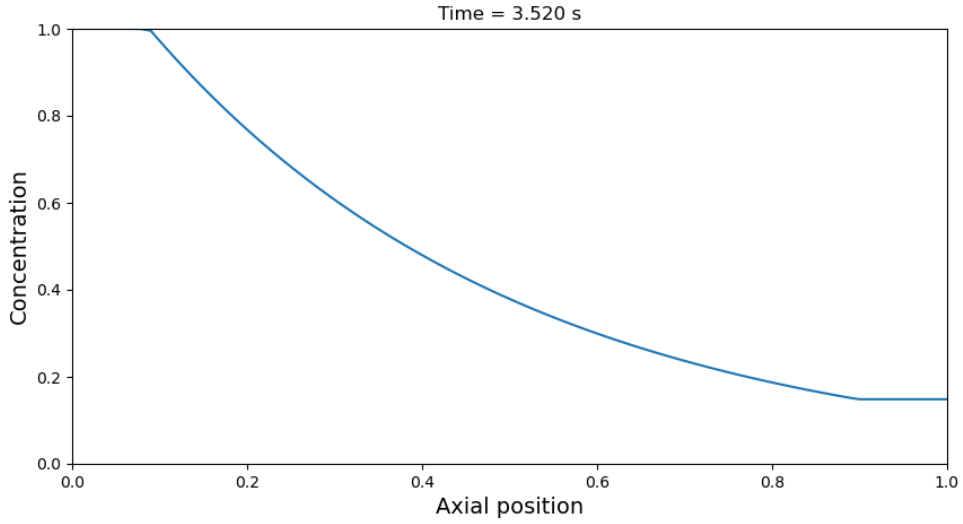

Figure 9: Implicit solution using 200 grid points

3. a) One implemented the Simpson integration scheme to find the amount of A in the reactor at the time the flushing begins. The order of convergence is found for each N ranging between 5 to 200

in 10 steps. From Table 1, it is evident that as the number of intervals increases, the relative error decreases as well as the rate of convergence while the calculated solution tends to the value 2.

Table 1: Analysis of the Simpson Rule method

N_t	Calculated Solution	$\epsilon_{relative}$	Rate of Convergence
5	1.710999	0.144501	N/A
26	1.992117	0.002941	2.1864674
48	1.997767	0.001116	2.057314
70	1.998964	0.000518	2.034659
91	1.999391	0.000305	2.025109
113	1.999607	0.000197	2.019749
135	1.999725	0.000137	2.000559
156	1.999795	0.000103	2.013788
178	1.999842	0.000079	2.012007
200	1.999875	0.000062	2.010603

Following this, a graph was generated with a number of steps varying from 3 to 100. As can be seen in Figure 10, the rate of convergence approaches 2 and the relative error greatly decreases as the number of steps increases. The rate of convergence matches expectations suggesting that the Simpson Rule was implemented correctly. Listing 5 shows how the Simpson rule was implemented. An if statement was employed to make sure that the lists are of equal size.

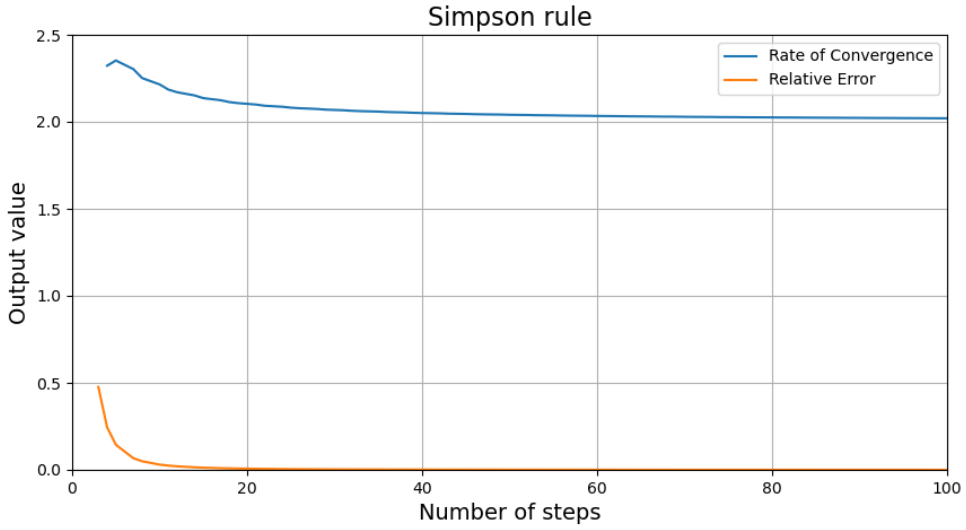

Figure 10: Rate of convergence and relative error

```python
def simpson_rule(xlist, ylist):
    '''Calculates the area of a function using the Simpson rule. \n
    xlist = List of x values \n
    ylist = List of y values'''
    if len(xlist) != len(ylist):
        return "Error, lists must be of equal size"
    sum = 0
    dx = (xlist[1]-xlist[0])
    for i in range(len(ylist)):
        if i>0 and i<len(ylist)-1:
            sum += (dx/6) * (ylist[i-1]+4*ylist[i]+ylist[i+1])
    return sum
```

Listing 5: Implementation of the Simpson rule

b) One calculated the total loss of A in moles via integration of the reactor concentration profile at flush time using the Simpson Rule. This was done by taking the concentration at 4 seconds as the system has already reached steady state meaning it the value of the integral would not change if the simulation was ran for an extended period of time. As can be seen in Listing 6, first the total concentration was calculated using the Simpson rule and subsequently, to get convert the concentration to moles, the concentration was multiplied by $\pi \left(\frac{d}{2}\right)^2 * l$ where d is the diameter of the tubular reactor and l is the length of the reactor. Thus we were able to obtain that the total amount of moles lost by flushing the system was 0.0005769 moles.

```python
sum_conc = simpson_rule(x,c) #Conc. given in mol/m^3
#To get sum of moles we need to multiply by the total volume
sum_moles = sum_conc * np.pi * (diam/2)**2 * x_end
print(f'The total amount of moles lost is {sum_moles}')
```

Listing 6: Calculating the moles lost by flushing the system

4. a) While initially, the solution seemed quite daunting, the solution to this question was easier than it seemed. As was illustrated in the previous tasks, the system reaches steady state quite quickly (at most by 4 seconds) the flow reversal was chosen at t=5 seconds. This was done to briefly simulate steady state. A new coefficient matrix was defined with all the elements of the diagonals multiplied by negative one and the elements of upper diagonal and lower diagonal switched. Moreover, a new Courant number (Co_nit) had to be defined to account for the different flow rate of the nitrogen. The boundary conditions for the last elements were also modified however, no boundary conditions were set for the first elements. Listing Listing 7 shows how this was done within the code. b) In

```python
A_new = sps.diags([(Fo), -(1+Co_nit+2*Fo+kR*dt), Fo+Co_nit], [-1, 0,
    1], shape=(Nx+1, Nx+1))
A_new = sps.csr_matrix(A_new)
A_new[Nx, Nx] = 1
A_new[Nx, Nx-1] = 0
for i in range(1, Nx):
    if dx*i < 0.1 or dx*i>0.9:
        A_new[i, i] = A_new[i,i] + kR*dt
```

Listing 7: Implementing reverse flow

order to plot the concentration as various times, if statements were employed as can be seen in

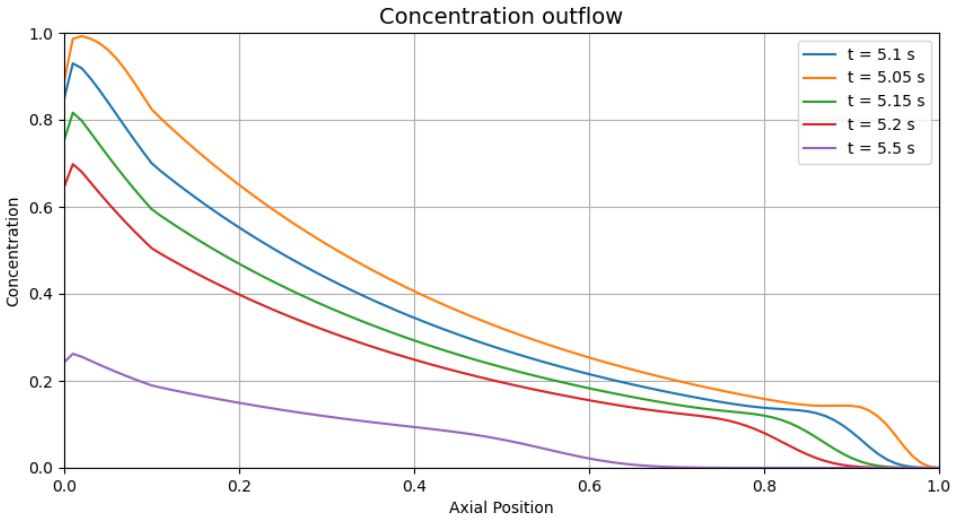

Figure 11: Concentration of A Outflow

Listing 8. The statements check whether the t is within a range close to the value and stores it for plotting. Figure 11 shows the concentration profile at varying times. It is important to recall

```
if t > 5.05 and t < 5.051:
    c_505 = np.abs(np.copy(c))
elif t > 5.1 and t < 5.101:
    c_51 = np.abs(np.copy(c))
elif t > 5.15 and t < 5.151:
    c_515 = np.abs(np.copy(c))
elif t > 5.2 and t < 5.201:
    c_52 = np.abs(np.copy(c))
elif t>5.5 and t < 5.501:
    c_55 = np.abs(np.copy(c))
```

Listing 8: Storing concentration at different times.

that the flow of nitrogen started at t=5 seconds. As can be seen, the concentration decreases quite rapidly with under the flow of nitrogen. As the concentration will should never truly reach zero, the time at which the maximum concentration is below 10^{-6} moles. This was found to be t=6.6906 seconds which corresponds to 1.6906seconds of flushing the system with nitrogen. c) Figure 12 shows the outflow concentration over time. It is important to note that the x axis corresponds to the time from the beginning of the nitrogen flow. The graph was generated by storing the absolute difference between the integral of the previous concentration profile compared to the integral of the following concentration profile. As can be seen, the majority of the compound is removed within one second.

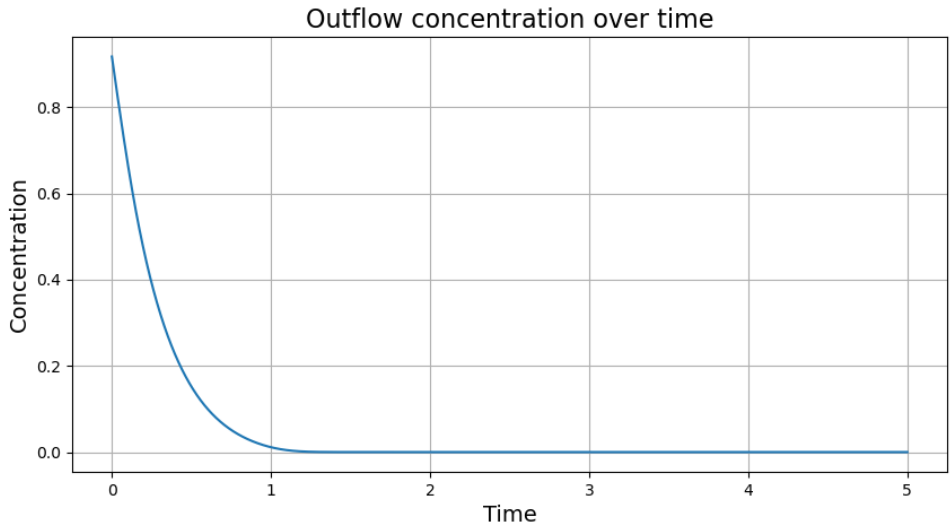

Figure 12: Outflow concentration over time.

d) For this last task, the graph from the previous question was integrated to obtain the total concentration outflow and similarly to task 3b, it was multiplied by $\pi \left(\frac{d}{2}\right)^2 * l$ where d is the diameter of the tubular reactor and l is the length of the reactor. From this we obtained that 0.0003169 moles were lost due to flushing. This value is close to half of the value obtained from task 3b. This aligns with expectations as during the flushing cycle, some of the compound is still able to react.

5. For the last task an oscillating inlet concentration was investigated. The concentration osilated according to the function $c[0] = cL + \sin(t \cdot \pi) \cdot 0.3$ where cL is the original inlet concentration (1 $\mathrm{mol\,m^{-3}}$) and t is the current time. Thus, the concentration varied between 0.7 and 1.3 $\mathrm{mol\,m^{-3}}$.

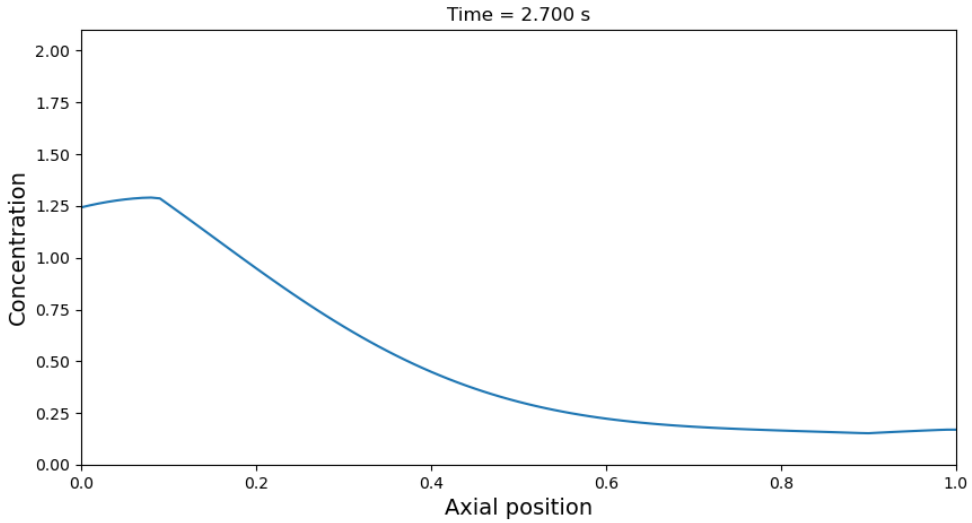

Figure 13: Snippet of the concentration over time for the oscillating inlet concentration

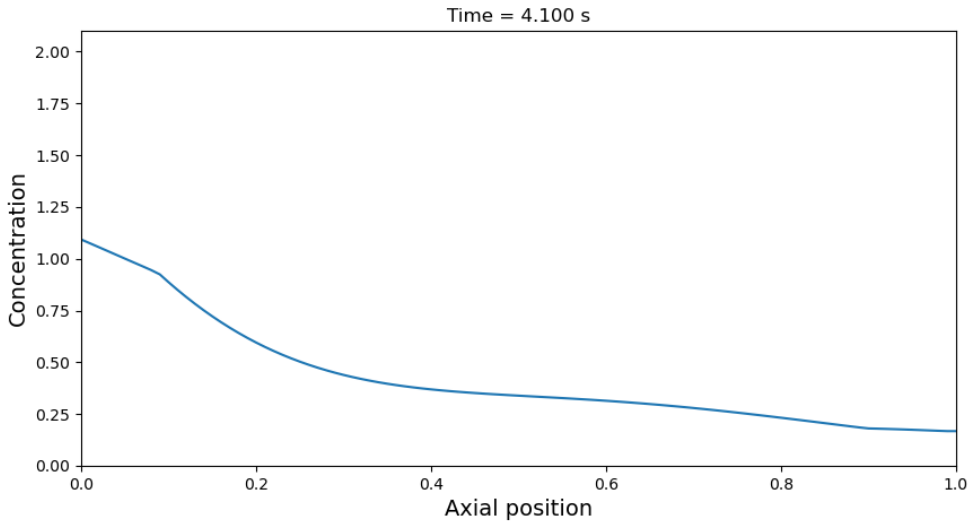

Figure 14: Snippet of the concentration over time for the oscillating inlet concentration

Figure 13 and Figure 14 show two snippets from the simulation with an oscillating inlet concentration. As can be seen in the figures, the effect of the oscillating inlet concentration decreases with increasing axial position. However, we believe that in order to gain a better view, one must run the simulation themselves to see it better.

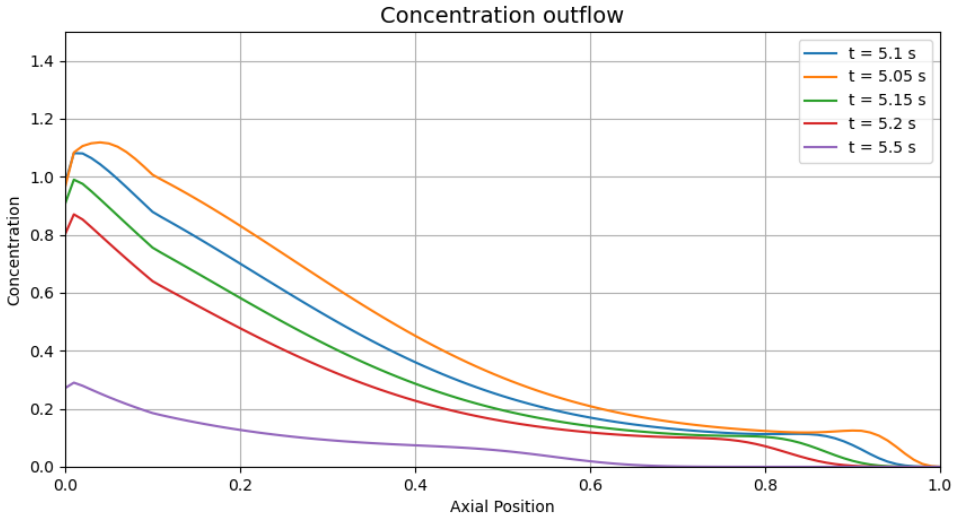

Figure 15: Outflow concentration at various times using an oscillating inlet concentration

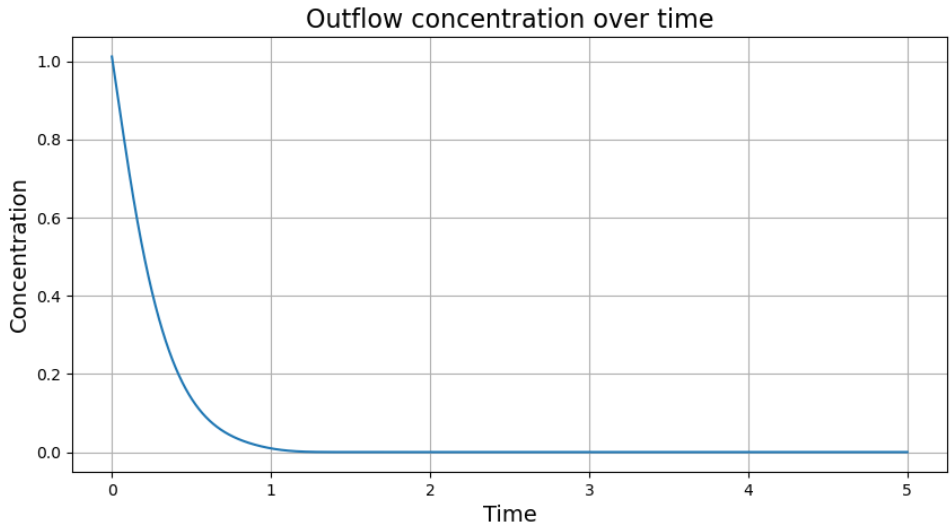

Figure 16: Total concentration outflow with an oscillating inlet concentration

Similar to task 4, the concentration outflow was plotted at various times as well as the total outflow over time. The time needed for the concentration to reach zero was 6.6866s, which was close but slightly lower than the time taken for the steady inlet concentration. Following this, the total molar outflow was found to be 0.0003296 moles which is slightly higher than the total outflow with the constant inlet concentration.

3 Reflection

In conclusion, the assignment was successfully solved. More knowledge was gained on the solution to partial differential equations and the understanding of the functionality of explicit and implicit methods was developed. In particular, we gained a deeper understanding of the functionality of sparse matrices. One thing that remains unclear is that with the flow reversal, the concentration seems to jump around the axial position = 0. We tried using different boundary conditions in order to fix this, however, nothing seemed to work and it was concluded that this is an artifact from solving the system implicitly.

List of symbols

Symbol	Unit	Definition
d	m	Diameter of tubulur reactor
l	m	Length of reactor
ϕ_v	$\mathrm{m^3\,s^{-1}}$	Volumetric flow rate
C_A	$\mathrm{mol\,m^{-3}}$	Concentration of A
k_R	$\mathrm{s^{-1}}$	reaction rate constant
D_{ax}	$\mathrm{m^2\,s^{-1}}$	Dispersion coefficient
Fo	Unitless	Fourrier number
Co	Unitless	Courant number