2.3.1 – 2nd Order Accurate Central Differencing (20 points)

1. Add the second order accurate central two-point scheme of equation 2.1 to the script and plot its results on the same plot as the first order backward difference two-point scheme at time t = 0.75. (10 points).

   The python code provided for the midterm was modified to add second order central differencing. This was then plotted alongside first order backward differencing, as can be seen in Figure 1.

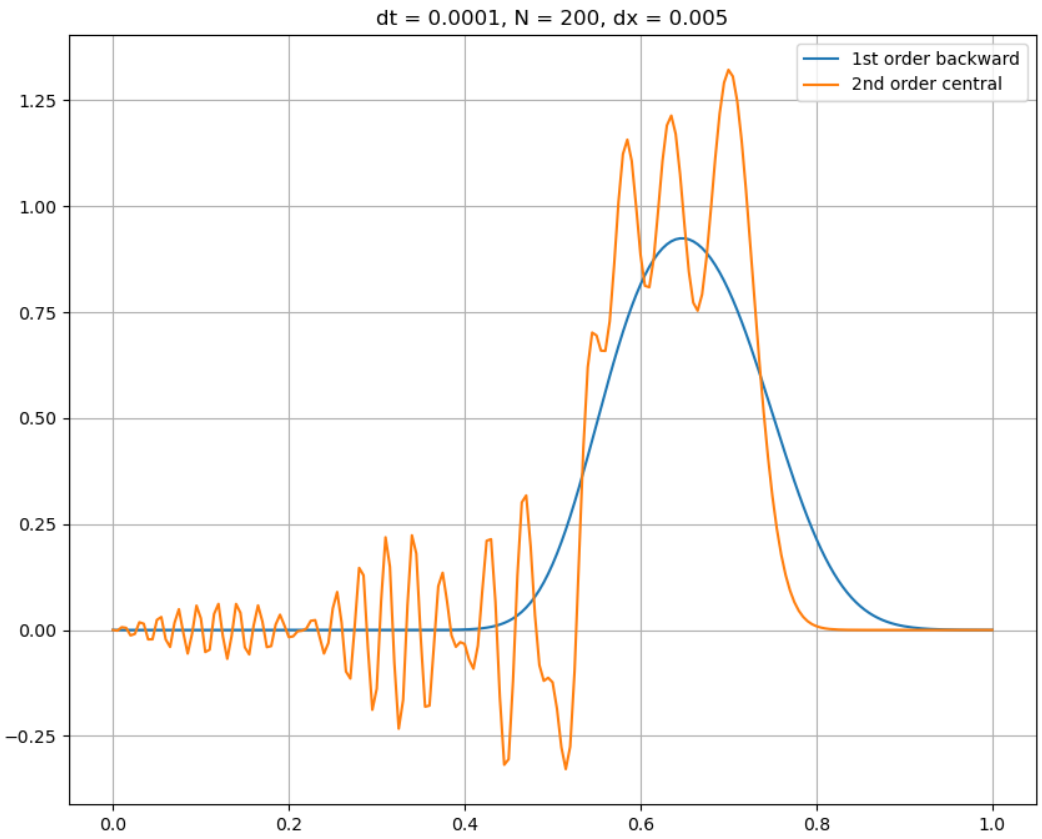

Figure 1: 2nd Order Central Vs. 1st Order Backward

2. Keeping the Courant number limitations of both schemes in mind, can you refine the time step and mesh spacing until your results stop varying at time t = 0.75? This is called time step and mesh independence. (10 points)

A new function was added to the code provided in the midterm which would only calculate the values for the second order central scheme. This allowed looping to be used to iteratively compare multiple time and mesh spacings. This was done by first running for a range of different mesh spacings, and then reducing the timestep until suitable results were achieved. The results can be seen in Figure 2.

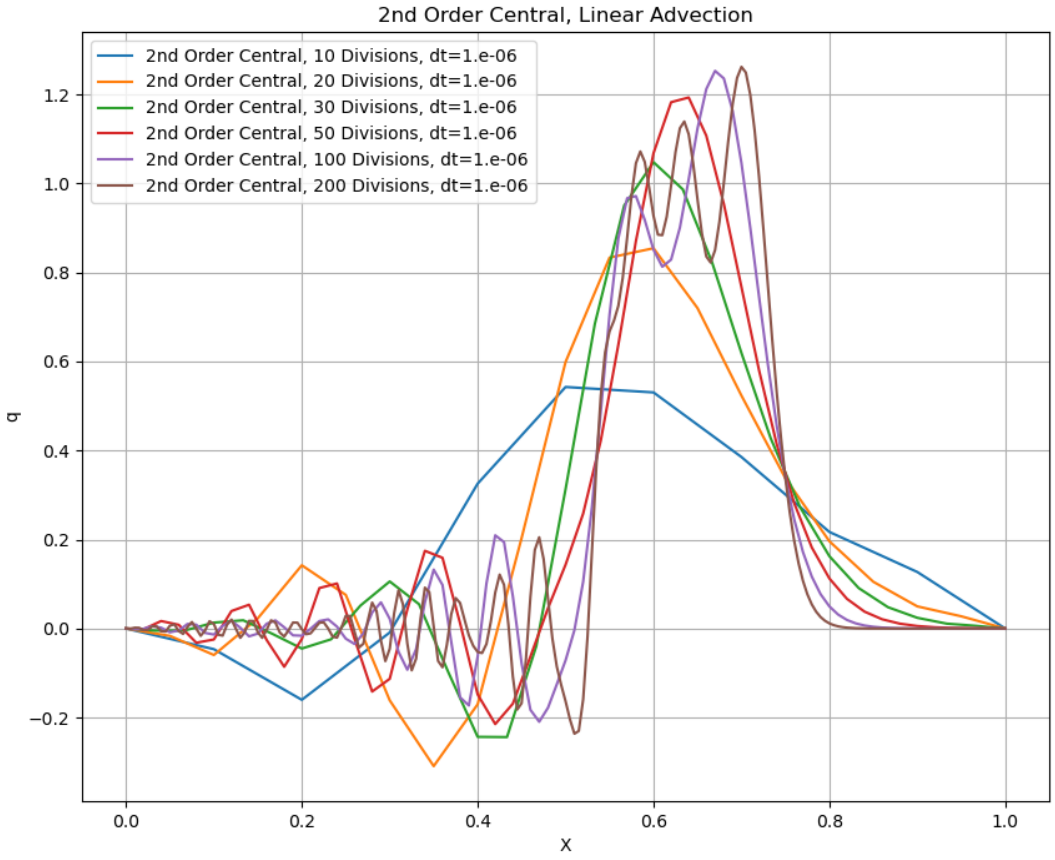

Figure 2: 2nd Order Central Mesh and Time Independence

In this case the best results were obtained with a timestep of 1e-6, and a mesh step as low as 20 divisions. With these values, the courant number is 2.5e-8. This lies within the range of values for which the central scheme is expected to be stable.

2.3.2 – 2nd Order Accurate Backward Differencing (25 Points)

1. Add the second order accurate backward three-point scheme of equation 2.2 to the script and plot its results on the same plot as the first order scheme of the first derivative at time t = 0.75. (10 points)

   Similar to question 2.3.1.1, the provided midterm code was modified to add the second order backward differencing. These results were then plotted alongside first order backward differencing, which can be seen in Figure 3.

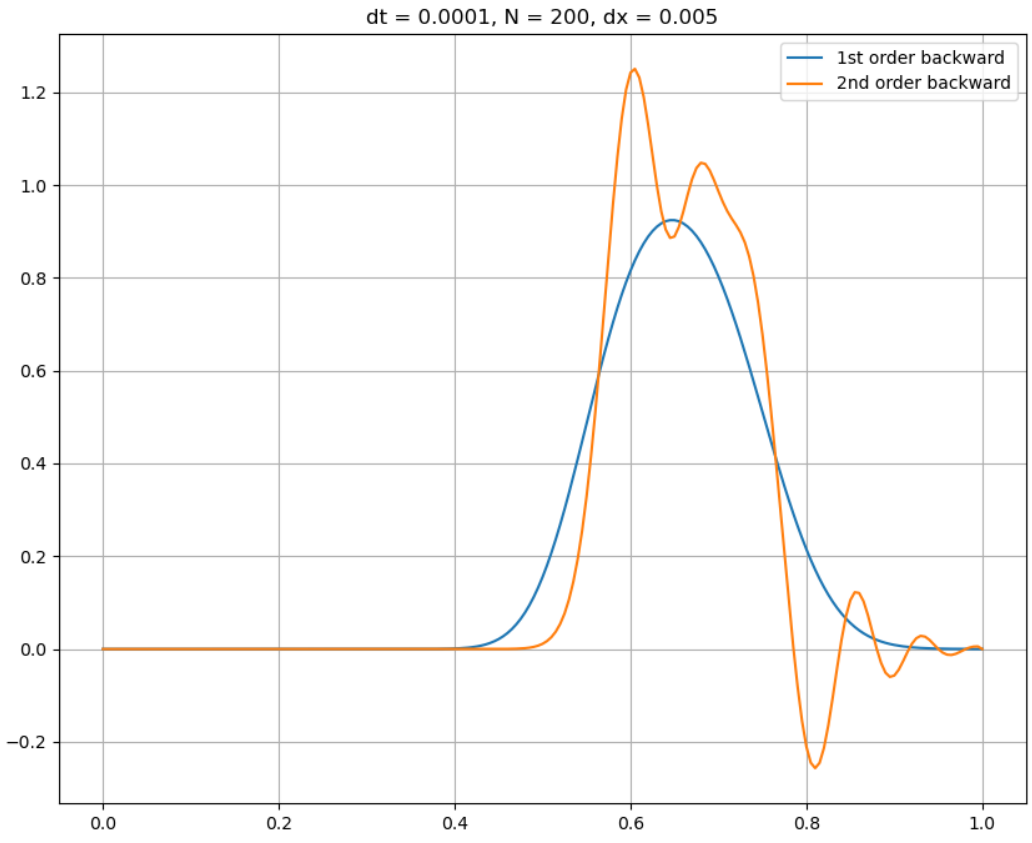

Figure 3: 2nd Order Backward Vs. 1st Order Central

2. Keeping the Courant number limitations of both schemes in mind, can you refine the time step and mesh spacing until your results stop varying at time t = 0.75? This is called time step and mesh independence. (10 points)

Similar to question 2.3.1.2, a separate function was created which would only calculate values for the second order central backwards. This makes it easy to loop through ranges of values and test for mesh independence and time independence. Instead of testing a variety of different timesteps in this case however, the courant number was set to be a constant value of 0.05, and the timestep was calculated based on the number of divisions and the courant number. The results were then plotted, and can be seen in Figure 4.

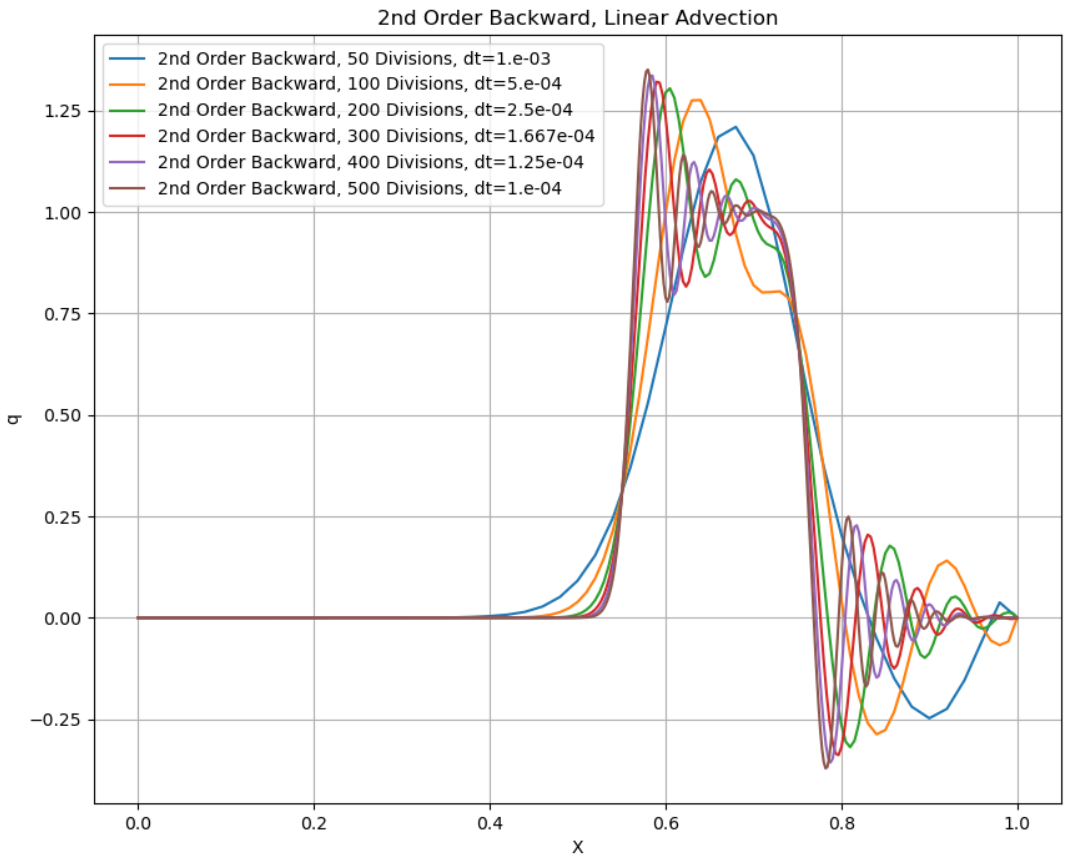

Figure 4: 2nd Order Backward Mesh and Time Independence

In this case the best results were determined to be 200 divisions. This led to a timestep of 2.5e-4.

2.3.3 – Comparing All 3 Schemes (5 Points)

Plot the solutions of all three schemes on the same plot at time t = 0.75 and answer the following questions.

In order to directly compare the 3 methods, the code was run 3 times. The first time involved running the code with a timestep of 1e-4, and 200 divisions. The code was then run again using the respective maximum timestep and number of divisions found in 2.3.1 and 2.3.2. The first order backward scheme was kept constant at 200 divisions with a timestep of 1e-4. Finally, the code was run using the timesteps found in 2.3.1 and 2.3.2, but with 200 divisions for each scheme. The results for these can be found in Figure 5, Figure 6, and Figure 7.

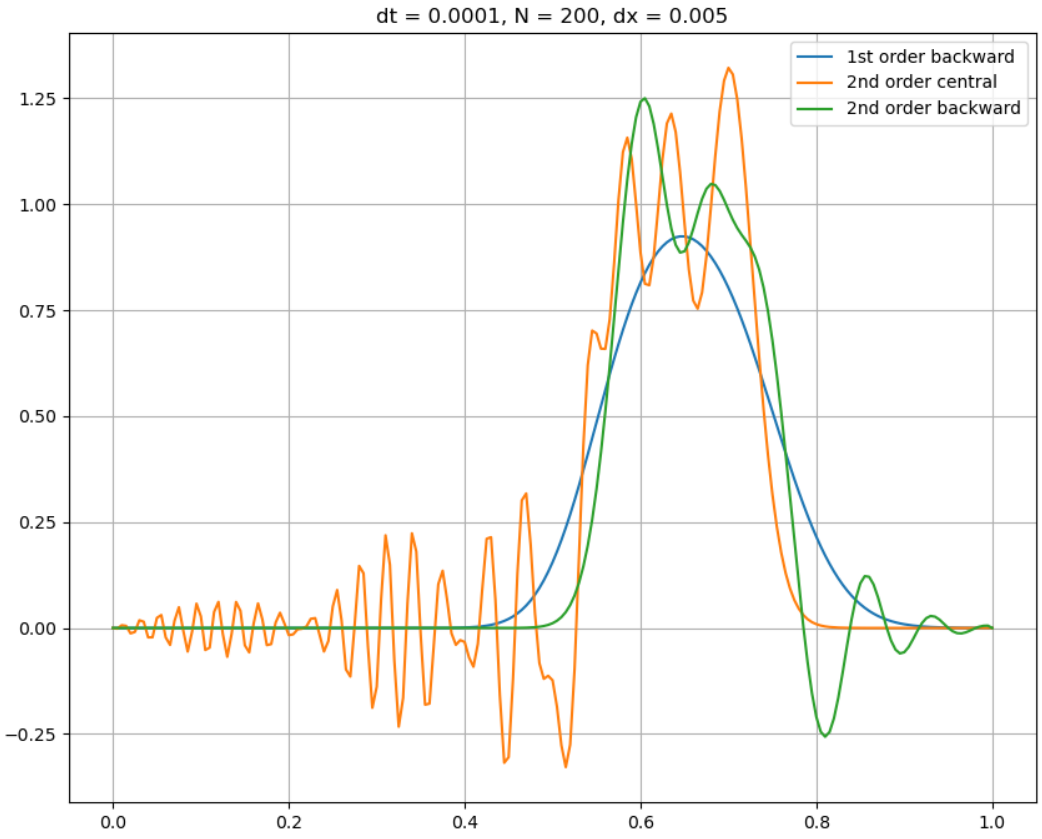

Figure 5: Comparison of all 3 schemes with dt = 1e-4, and 200 divisions

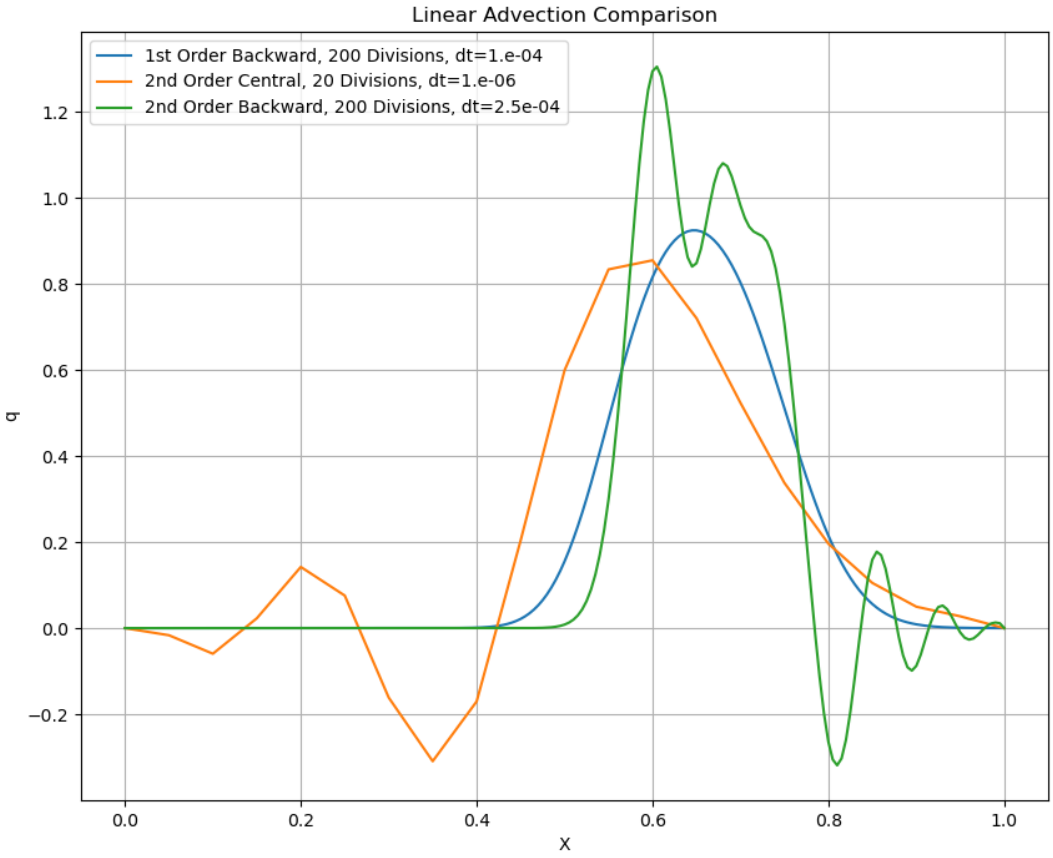

Figure 6: Comparison of all 3 schemes with ideal timestep and mesh spacing

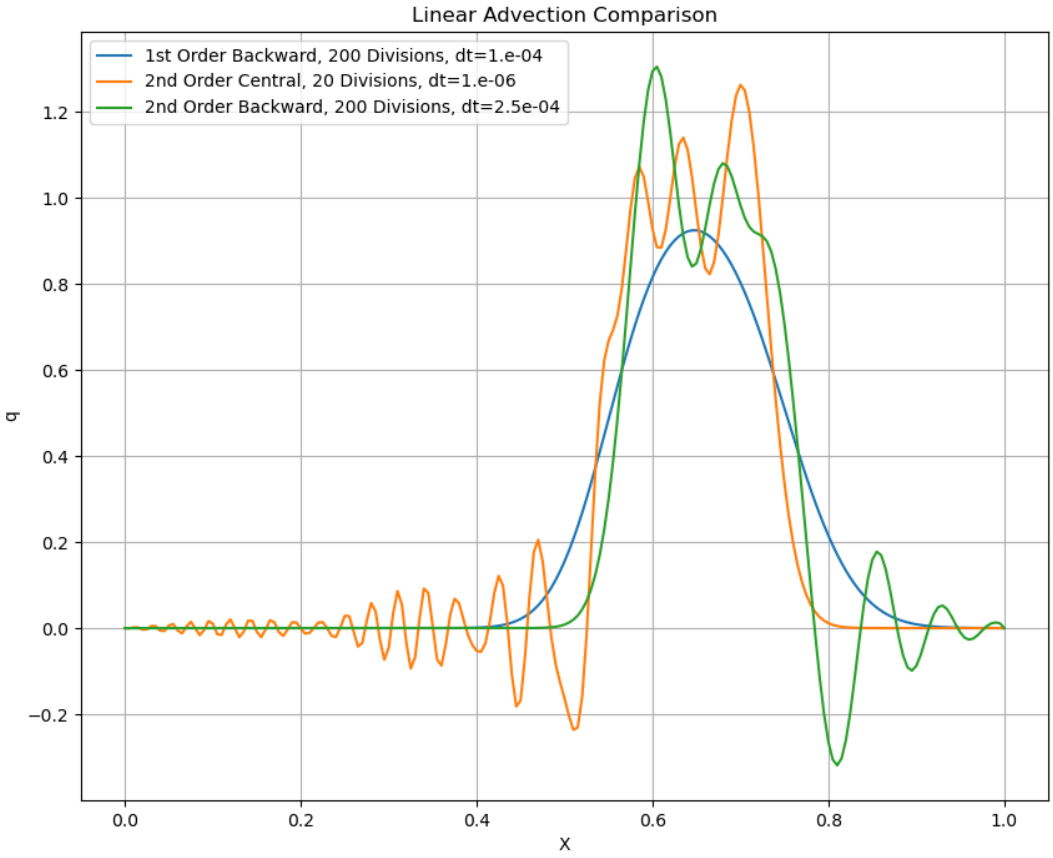

Figure 7: Comparison of all 3 schemes with ideal timestep and 200 divisions

1. Which of the 3 schemes exhibits the most oscillations and why? (2.5 points)
   Referring to Figure 5, Figure 6, and Figure 7 it becomes clear that the second order central scheme exhibits the most oscillations. This is expected, as the central scheme has the highest accuracy. Both the forward and backward schemes converge linearly, while central scheme converges quadratically to the value. This can be seen when looking at the equations.

$$\frac{\partial u_i}{\partial x} = \frac{(u_i - u_{i-1})}{\Delta x} + O(\Delta x) - Backward\ difference\ scheme$$

$$\frac{\partial u_i}{\partial x} = \frac{(u_{i+1} - u_i)}{\Delta x} + O(\Delta x) - Forward\ difference\ scheme$$

$$\frac{\partial u_i}{\partial x} = \frac{(u_{i+1} - u_{i-1})}{2\Delta x} + O(\Delta x^2) - Central\ difference\ scheme$$

The central difference scheme has an order of accuracy proportional to Δx^2, while the other two schemes only have an order of accuracy proportional to Δx. Schemes with higher of orders of accuracy are subject to higher oscillatory behaviour, and as such it is expected that the central scheme will have more oscillations. It is for this reason that a common tactic used in CFD is to run simulations initially using a lower order of accuracy scheme, and then switching to a higher order of accuracy scheme once the solution begins to converge. This allows the solution to stabilize quickly initially, while obtaining the desired accuracy afterwards.

2. Which well-known stability theorem about the order of accuracy of a stable linear scheme applies to your results? (2.5 points)

   The Neumann stability analysis theory applies to the results obtained. This can be seen in the derivations obtained in questions 2.1.1 and 2.1.3.

3.1 – Bonus Problem

Modify the linear advection script from problem 2.3 by adding the following 2nd order accurate three-point central difference approximation of the second derivative,

$$\frac{\partial^2 q}{\partial x^2} = \frac{q_{i-1} - 2q_i + q_{i+1}}{\Delta x^2} + O(\Delta x^2) \quad (3.2)$$

Since the diffusion operator is a second derivative you will need to add another boundary condition at the outlet of the domain. Use a zero-gradient boundary condition, which can be implemented by setting the solution value of the last point of the domain equal to that of the preceding point. Obviously, you will not need to solve for the solution at the last point anymore. (15 points)

For the bonus problem, a new code was created which included the diffusion part described in equation 3.1 on the midterm. A 3-point 2nd order accurate central difference approximation of the second derivative was added to the linear advection equation found in question 2. This allowed for the solving of the linear advection-diffusion equation. This code will also be included in the final package handed in, and can be verified. It will be named linear_advection_diffusion.py.

3.2 - Using the 2nd order accurate scheme of equation 2.2 for the advection term, simulate the transport of the same step function as in problem

2.3. Keep refining the time step and mesh spacing until you reach time step and mesh independence at time t = 0.75. Show it by plotting the results of successive time and space refinements on the same plot and mention which time-step and mesh spacing achieved time step and mesh independence. (5 points)

The time step and mesh independence analysis were performed similarly to that which was done in question 2.3.2. That is, a function was created which would output the linear advection-diffusion results, and the courant number was set to be a constant value. In this case the courant number was set to be 0.05. The timestep was then calculated based on the courant number and the mesh spacing. Finally multiple mesh spacings were iterated through. The results can be seen in Figure 8. It can be seen that after 400 divisions with a timestep of 1.25e-4 the results begin to converge.

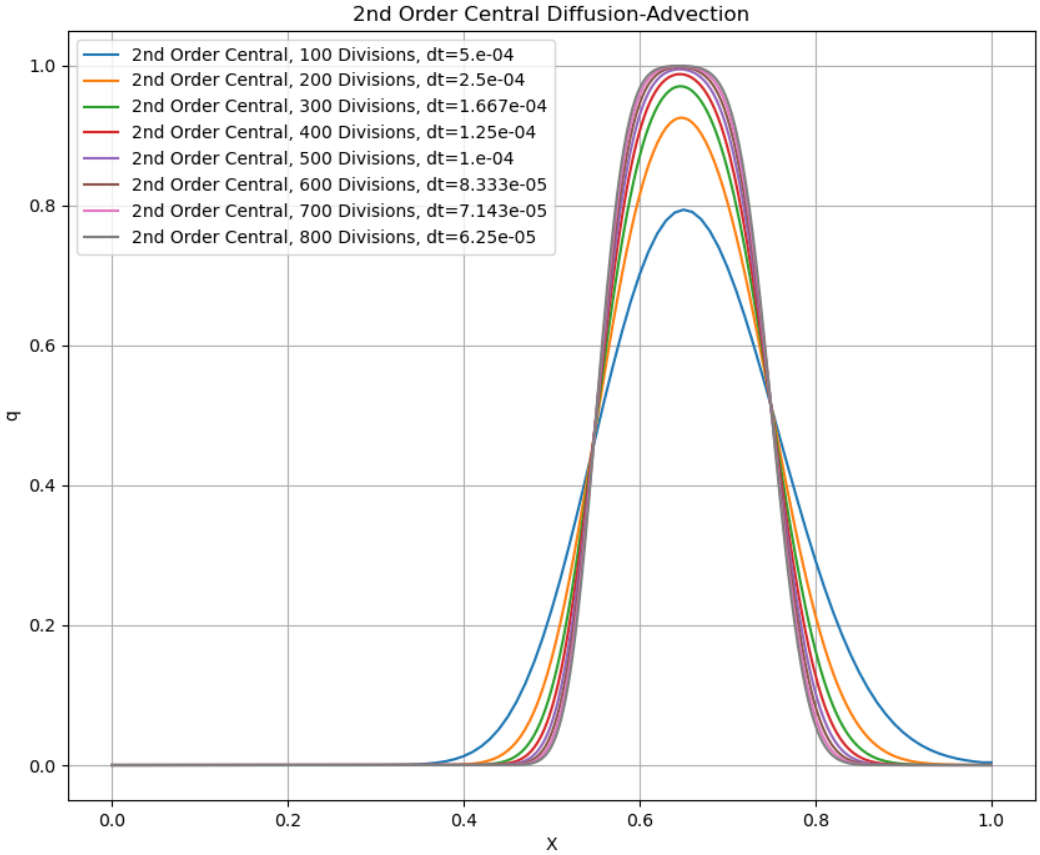

Figure 8: 2nd Order Central Diffusion-Advection

The results found in Figure 8 demonstrate the linear advection diffusion equation using a first order backward scheme for the first derivative. The same thing was coded for both the second order central scheme as well as the second order backward scheme for the second derivative. The results for this can be seen in Figure 9.

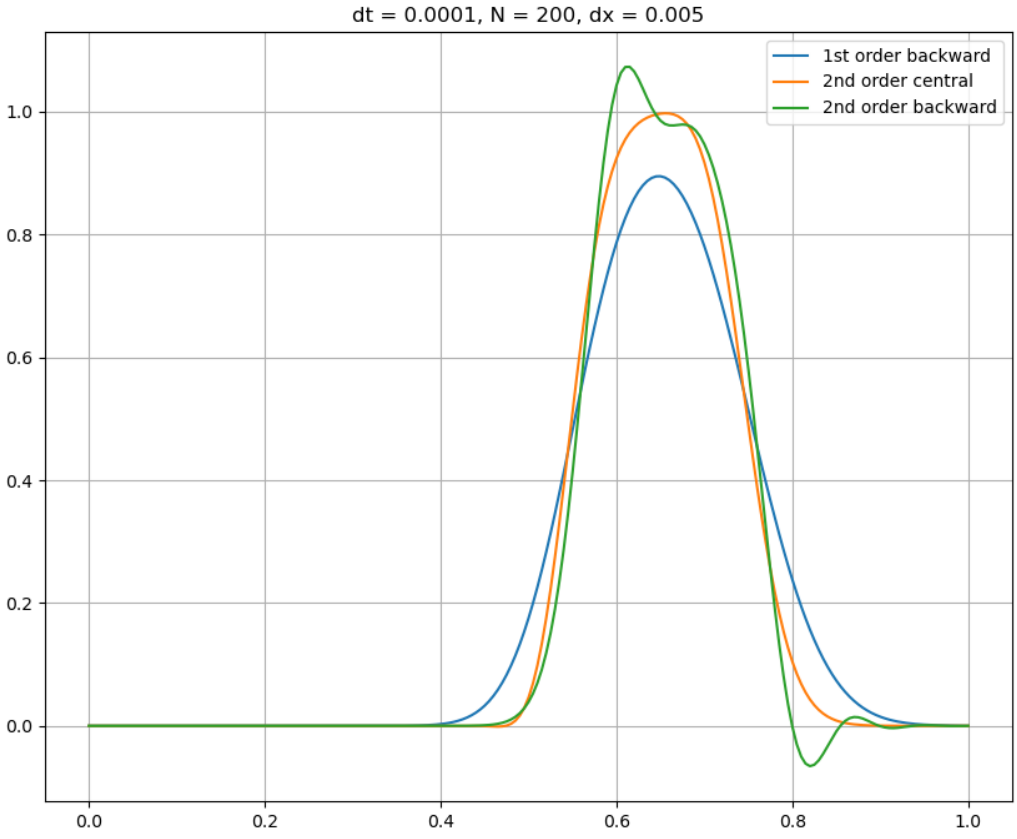

Figure 9: Linear Advection-Diffusion Comparison

3.3 - Once you have reached time step and mesh independence, run the script again for β = 10e−5, 10e-4, 10−2, 10−1 and plot your results on the same plot for t = 0.75 with the baseline solution at β = 10−3. (5 points)

The following plots were created for the varying beta values.

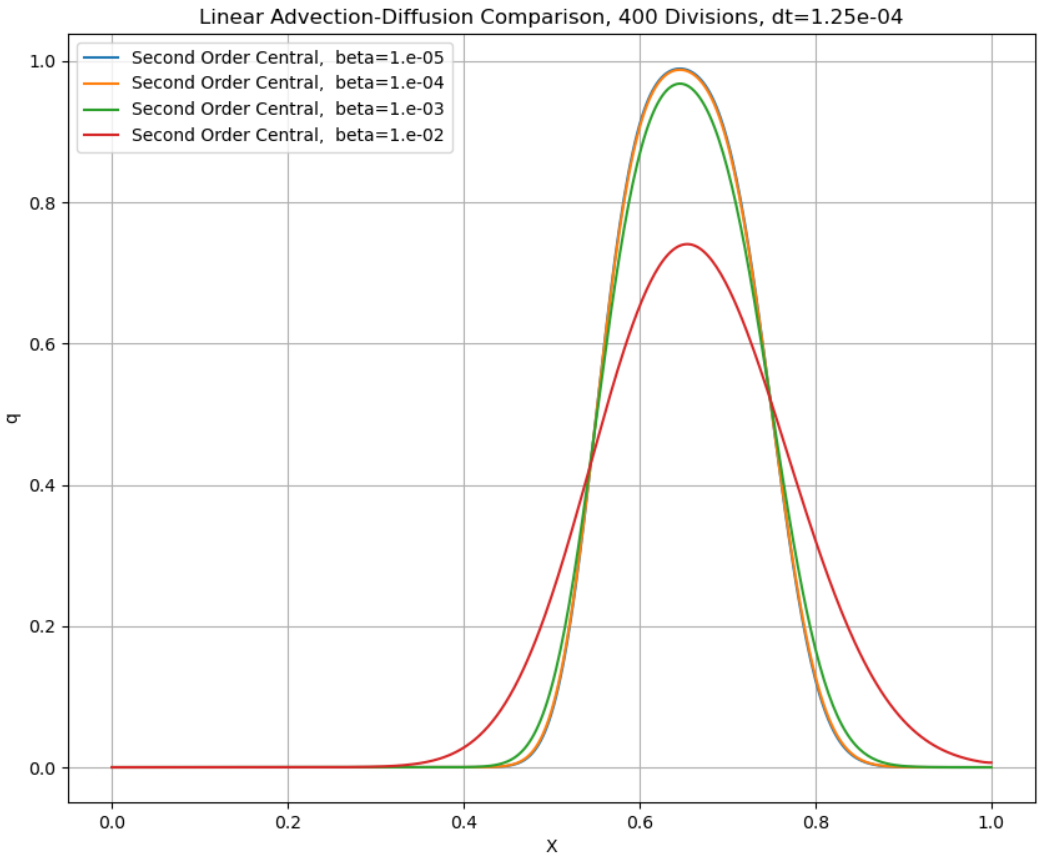

Figure 10: Beta value comparison with first order backward method for the first derivative

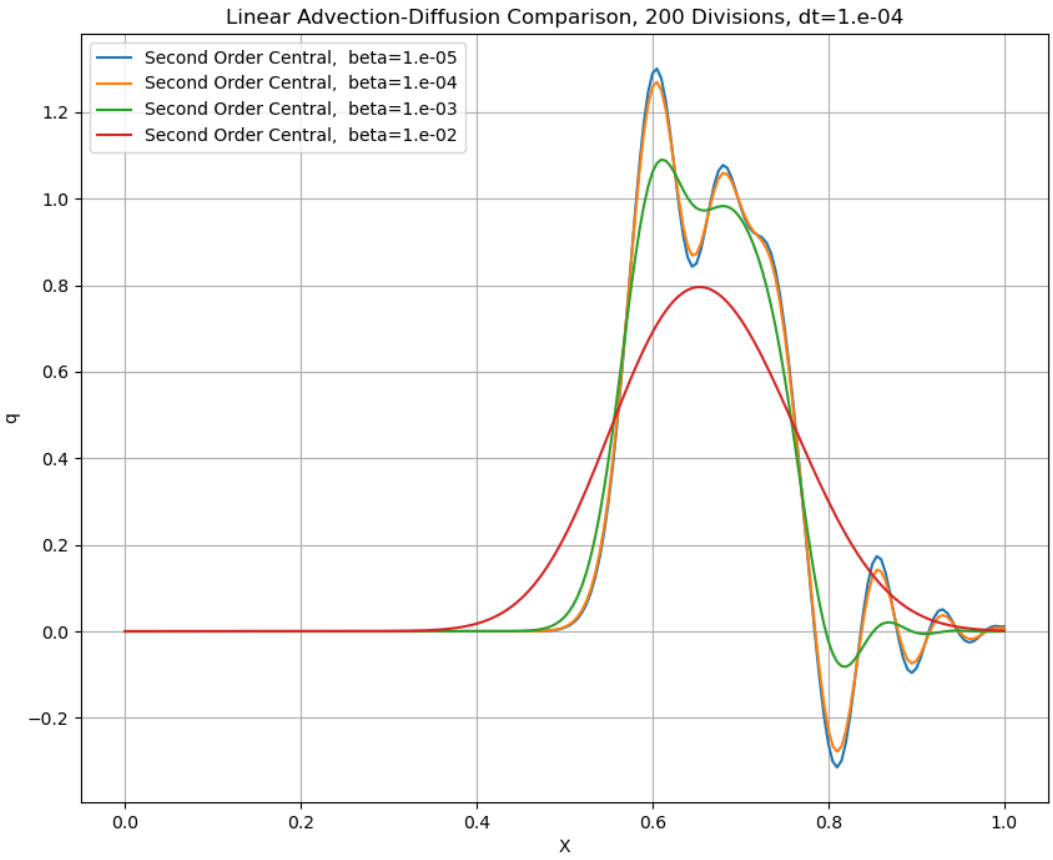

Figure 11: Beta value comparison with second order backward method for the first derivative

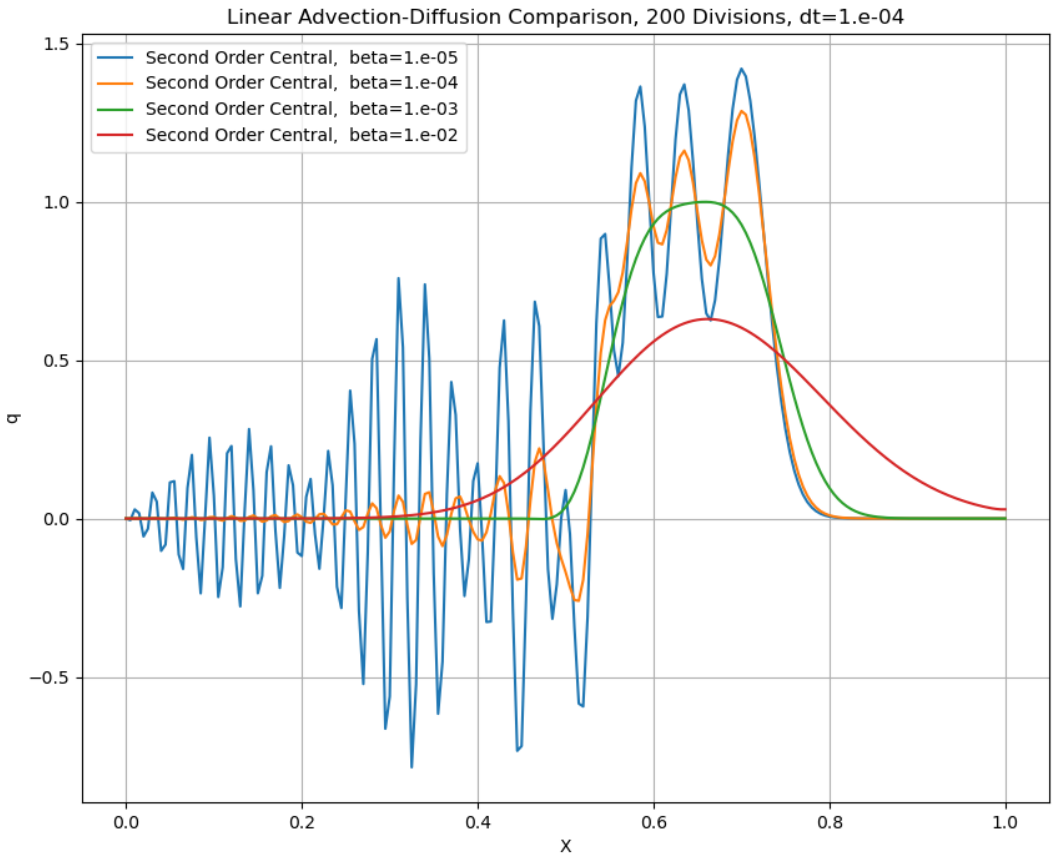

Figure 12: Beta value comparison with second order central method for the first derivative

3.4 - How do your results for each β value compare to the baseline solution at β = 10−3? Which solution has the most oscillations and what do you attribute that to? (5 points)

Figure 10, Figure 11, and Figure 12 show how the results obtained with varying beta for each different method used. Each method used the second order central difference scheme for the second derivative, but varied the method used for the first derivative. Unfortunately, when beta was set to 1e-1, the code consistently crashed. It was due to overflow, indicating the values obtained blew up with this value for beta. Overflow simply means that the values were too large to be captured by numpy. In this case, numpy double precision was used, and as such the value was outside of the range [-1.79769313486e+308, 1.79769313486e+308]. It is unclear as to why this occurred. However, for the results obtained the most oscillations were found with the central second order method for the first derivative, when beta was 1e-5. It was explained in question 2.3.1 why using the

central second order method will cause the most oscillations. However, what is interesting is the fact that the diffusion coefficient changing also caused oscillations to occur. This may have occurred, as in the equation beta is the coefficient which the second derivative is multiplied by. Changing beta will change the value of what gets added for the second derivative term, and induce oscillations.