Scientific Computation Project 3

01854740

March 24, 2023

Part 1

1.

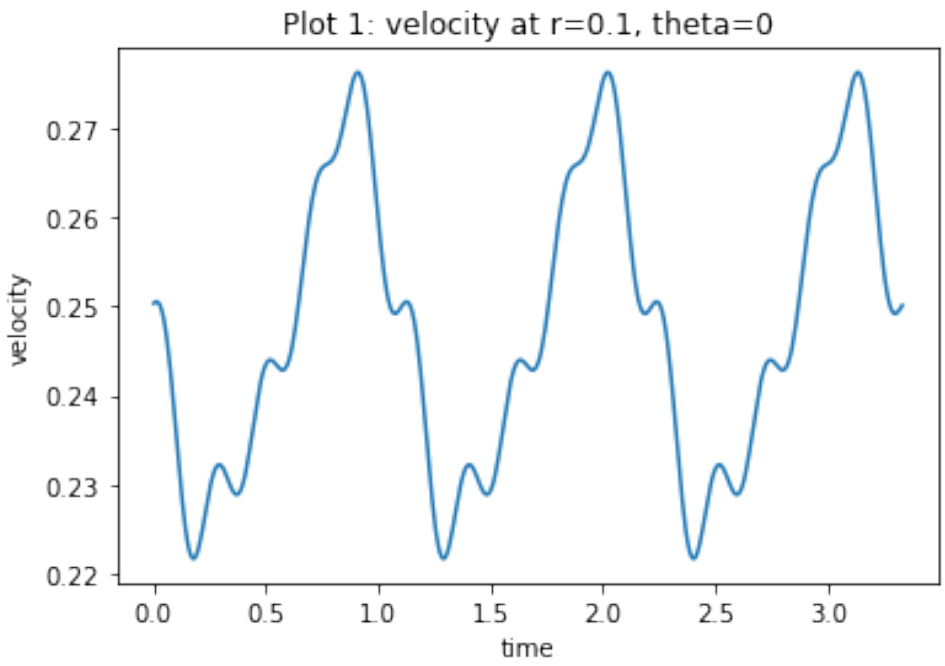
By printing the various sizes of the r, th and t vectors, we see that they correspond to the 3 dimensions of U while U contains the corresponding velocities. Theta values range from 0 to 1.875 pi. The total time T=3.33s and we can see that for all r and theta, the velocity is periodic with period 3 over the time T (the 1st figure shows an example). Hence in the subsequent figures, I plot only the first 1/3 of the time interval.

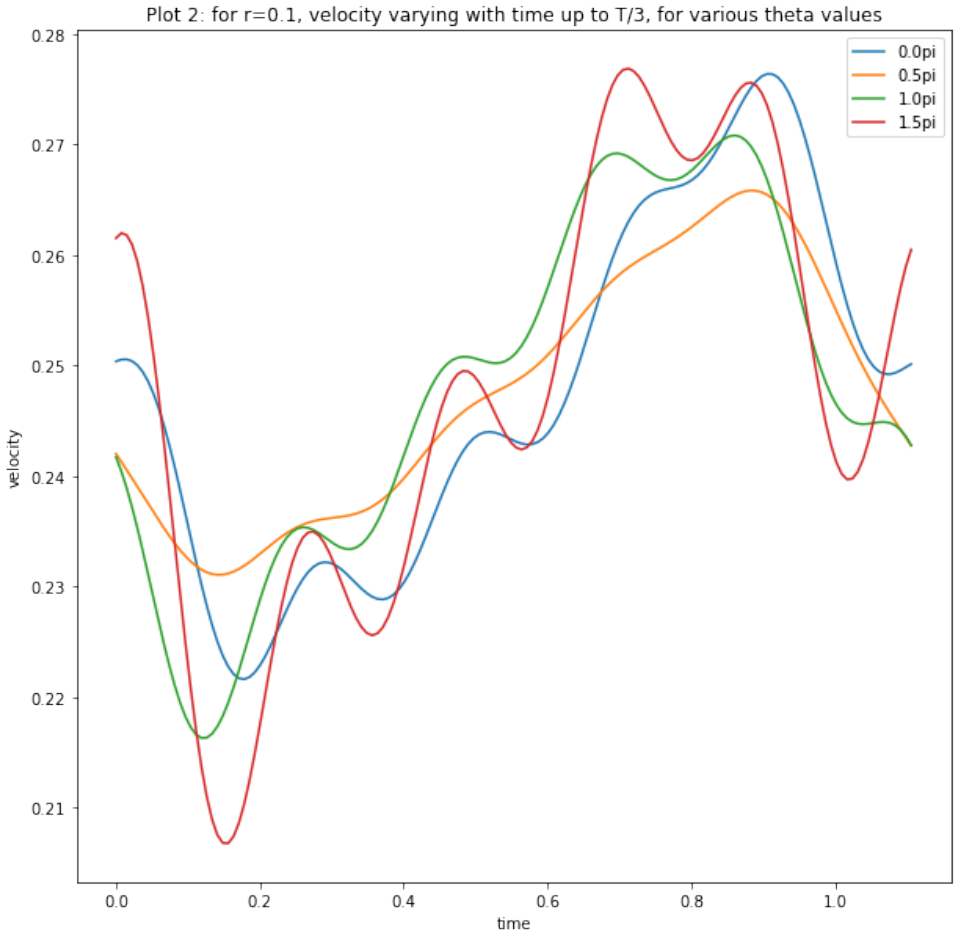
Fixing r=0.1, we can compare the velocity at various angles along the cross section over time in plot 2. The speed of the blood flow is not homoegeneous over a fixed cross sectional distance, it varies with the angle. At angle 0, we can see from the blue curve some bumpy sinusoidal behaviour (acceleration and deceleration of blood flow), whereby the curve starts at 0.25, dips below, rises again and finally dips again to 0.25 by T/3. As the angle increases from 0 towards 2pi, we see that the shapes of the curves remain similar i.e. decrease, then increase and finally decrease again (though not without smaller oscillations too). However, the orange curve corresponding to theta=pi/2 is shifted down at the extremities of the plot compared to theta=0, though it is above the blue curve at most times.

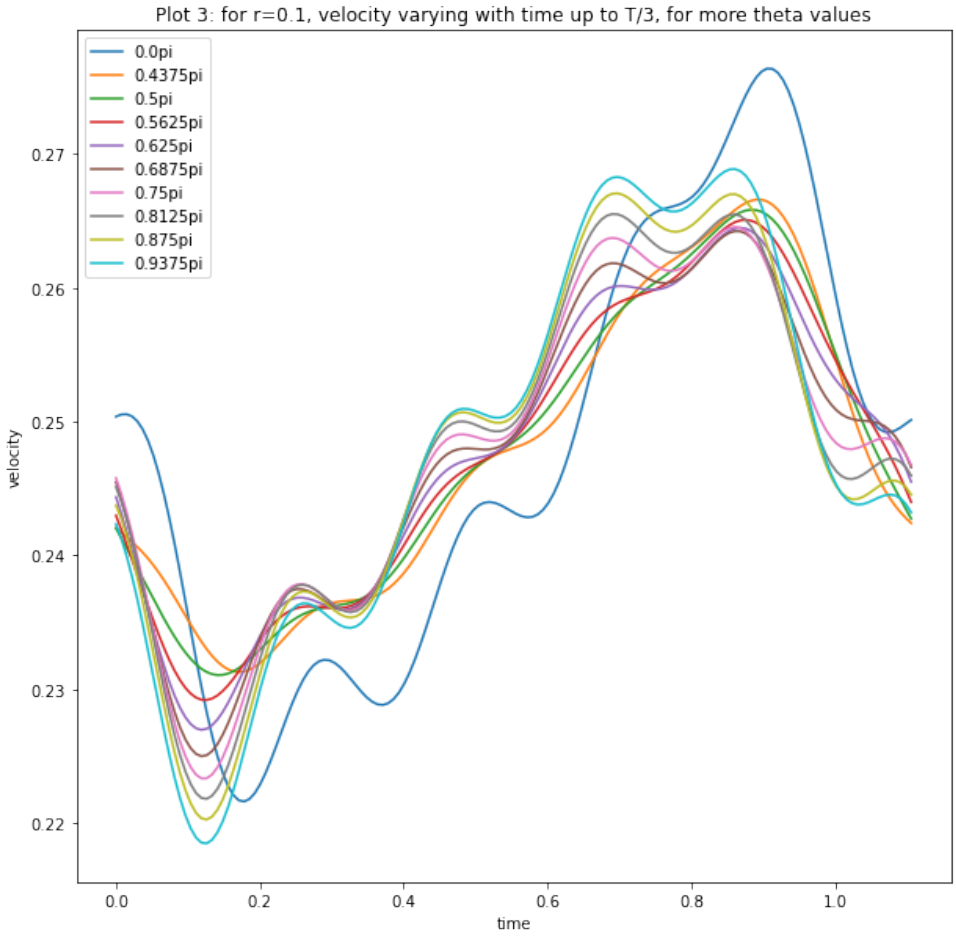
Plot 3 shows the trend in theta more closely, up to pi (for r=0.1). We have the blue line for theta=0 for reference, and several curves around the values 2/3 pi. We can see see some of the curves folding back on themselves (crossing), for example near t=0.25, which is a turning point in velocity as theta is varied for that point in time. From figure 4 we see that for all points in time, the velocity tends to go up and down and back up or vice versa as theta is increased, returning to the starting velocity at 2pi. We can see also from plot 4 that the range in velocities across time points is larger for certain theta values than others, i.e. would have larger amplitude in plot 3. The range is thinnest near pi/2 and largest near 3pi/2.

We notice that for r=0.1, the velocities range between 0.20 and 0.28 at all times and theta, whereas for r=0.5, they range between 0.15 and 0.21 roughly. Hence, the velocities are larger at r=0.1, which is nearer to the centre of the blood vessel than 0.5 which allows for faster blood flow. The maximum difference in velocity over a period is also slightly larger for r=0.1 so acceleration and deceleration of blood is slightly larger near the centre. Also, we see that while for different theta values we get different curves over time, the shape of these curves is consistent for the same theta value at r=0.1 and r=0.5, so the angular trends are consistent across the different radii. Plots 5, 6 and 7 serve to show that all trends described for r=0.1 apply to r=0.5, just with the velocities all being within a range of values which are smaller.

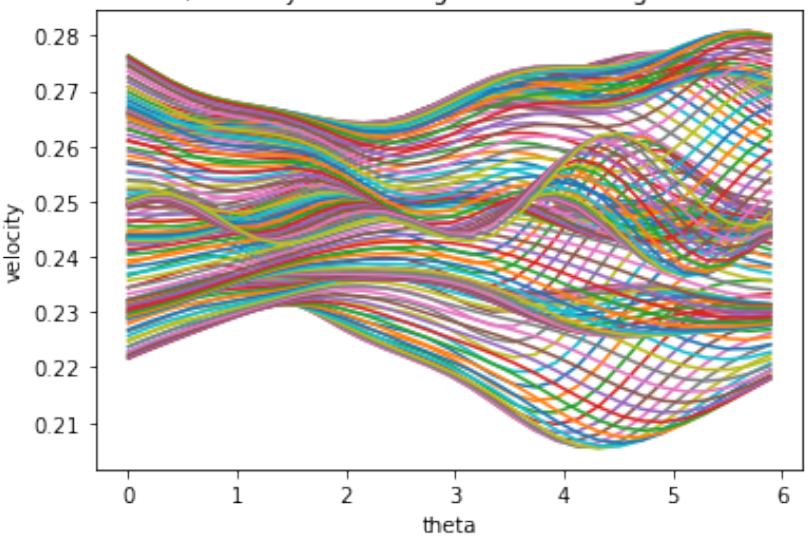
In the final plot 8, I visualise the field at time 0 and we see that velocity increases and we go towards the centre but is different for different angles.

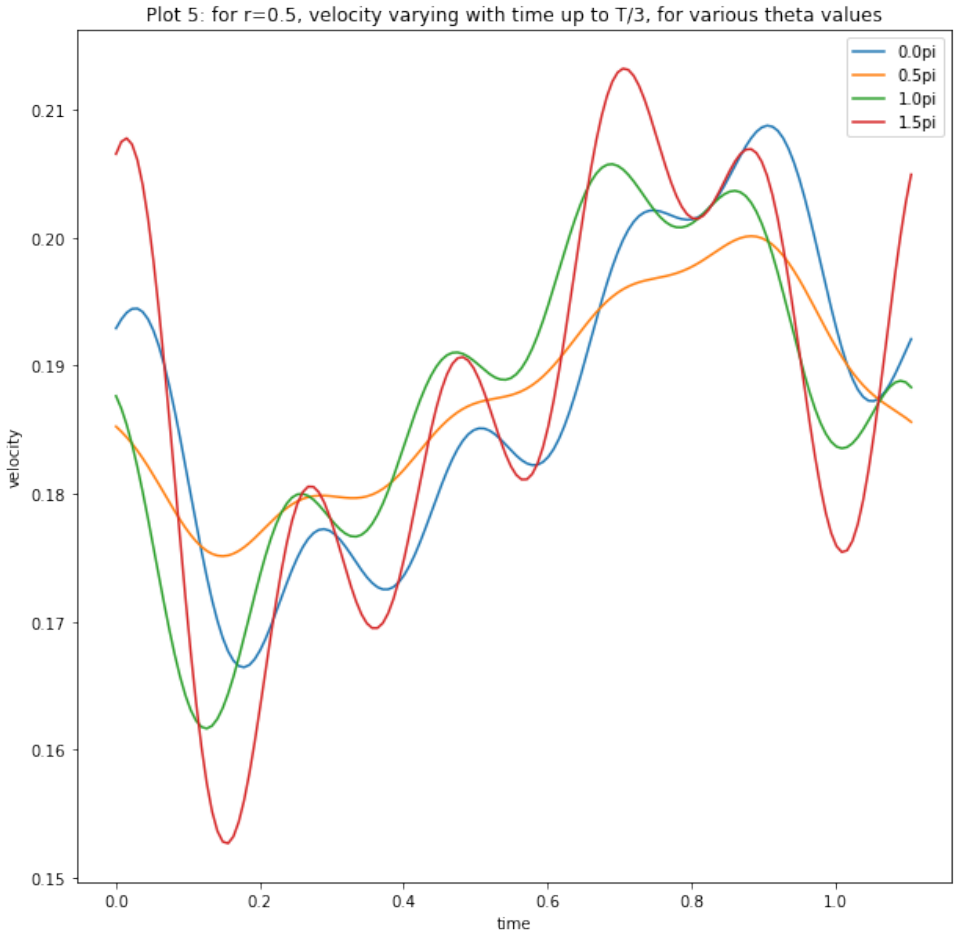

Plot 1: velocity at r=0.1, theta=0
velocity
time
0.22
0.23
0.24
0.25
0.26
0.27
0.0
0.5
1.0
1.5
2.0
2.5
3.0

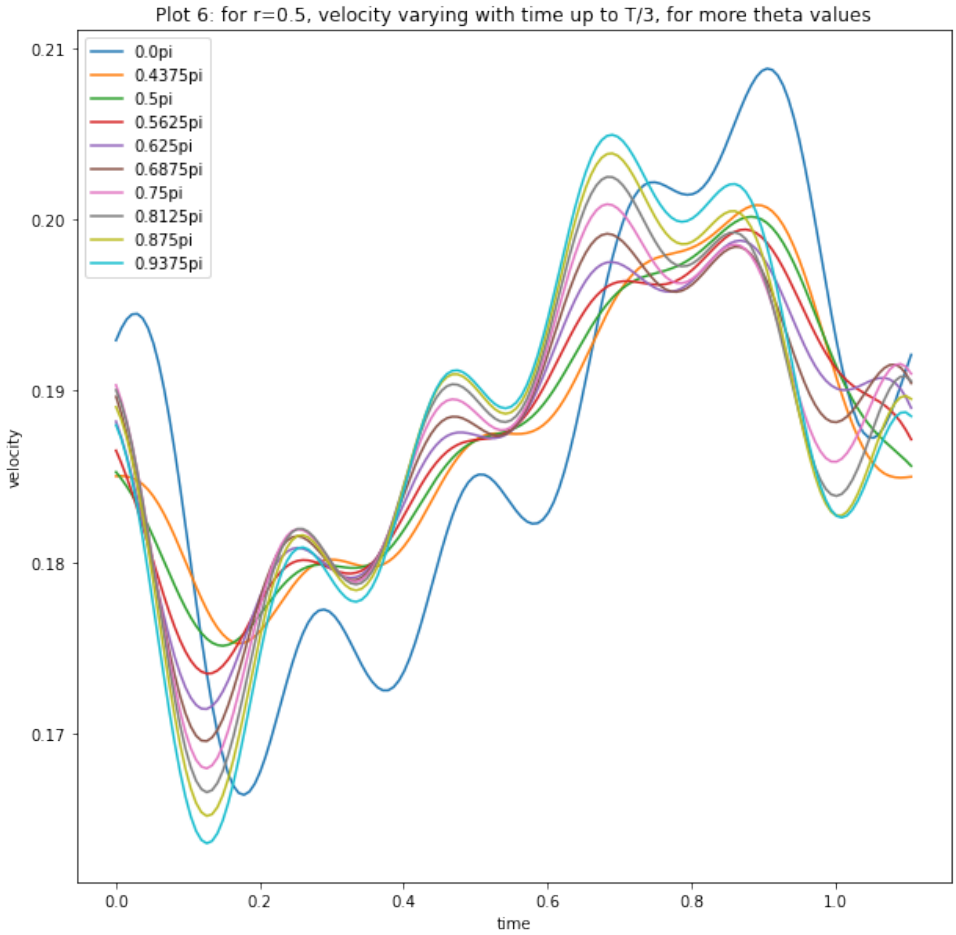

Plot 2: for r=0.1, velocity varying with time up to T/3, for various theta values
0.0pi
0.5pi
1.0pi
1.5pi
velocity
time
0.21
0.22
0.23
0.24
0.25
0.26
0.27
0.28
0.0
0.2
0.4
0.6
0.8
1.0



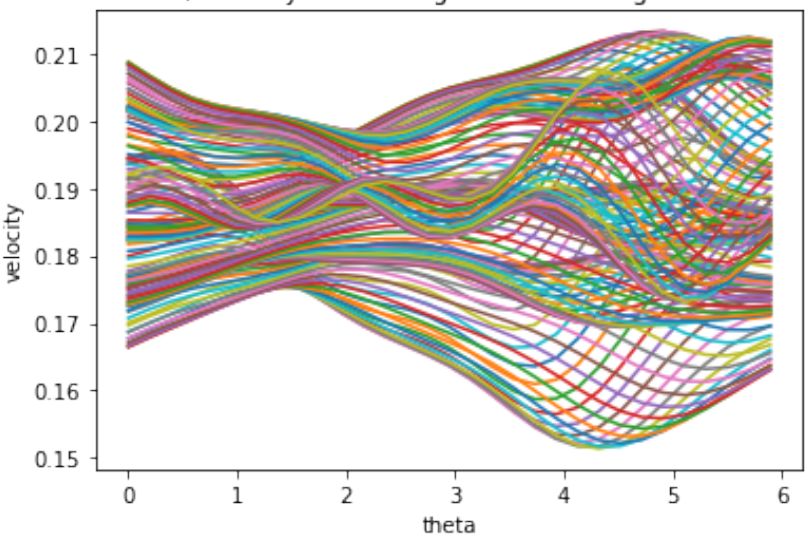
Plot 4: for r=0.1, velocity as it changes over the angles for various times

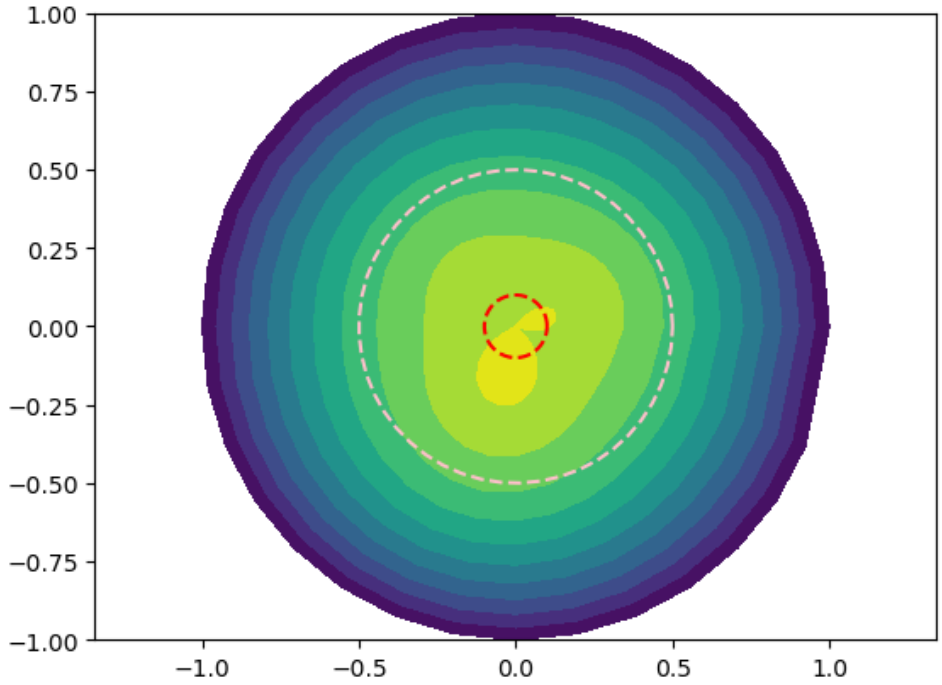



Plot 5: for r=0.5, velocity varying with time up to T/3, for various theta values
0.0pi
0.5pi
1.0pi
1.5pi
velocity
time
0.15
0.16
0.17
0.18
0.19
0.20
0.21
0.0
0.2
0.4
0.6
0.8
1.0



Plot 7: for r=0.5, velocity as it changes over the angles for various times





2.

U is of initial shape (81,31,450), hence the cost in memory is currently of 81x31x450 which is 1,129,950. For the implementation of part1q2C, we first notice that we can make use of the periodicity of the dataset in time. It has periodicity 3, so we can truncate U and keep the first third in time, creating Ucut of shape (81,31,450).

I chose the smallest axis of size 31, and perform singular value decomposition on the other two dimensions, and keeping only singular values above a threshold of 10e-14 (which removes 0 eigenvalues), and storing in lists of length 31: as ks, the number of singular values above the threhsold for every cross-section, as ss, the singular values for all 31 cross-sections, as us the u for each cross-section from SVD up to the kth one where k is the number of singular values being kept, and finally likewise for the w from each SVD in Ws. For the ith k (ki), s has length ki, u has shape (81,ki) and w has shape (ki,150), so the total memory cost for i is kix(1+81+150) = 232ki. I find that across all 31 SVDs, the sum of the ki is 327, hence the memory used to store the us,ss and Ws is of 327x232. Also, I store a list of ks of length 31 so the total memory is 75895. I could not store this and reconstruct it from the ss list and the number of singular values of each element but a list of length 31 is negligeable in memory and makes reconstruction slightly easier to code. The memory saved as a fraction is 75895 by 1,129,950 = 0.0067, or 1/0.0067 = 14.88 times more efficient after compression.

The reconstruction is as follows. I initialise an array of the reduced size (81,31,150) of 0s. For each of the 31 cross-sections, I find the k corresponding to the SVD of that cross-section, and add for each singular value up to the kth one the singular value multiplied by the outer product of u and s corresponding to this corss-section and singular value. Finally, I duplicate Unew 3 times in the time axis making use of the periodicity to simply copy it and retrieve a matrix of dimensions (81,31,450).

I have only removed singular values which are numerically almost equal to 0 via the threshold of 10e-14, and I have kept the largest singular values so I have the most important directions or features of the original matrix. These features are the ones with the largest singular values, which correspond to the directions with the most variance. By keeping these singular values, we are preserving the most significant information about the original matrix. Keeping all the singular values would keep all the information. However, the additional singular values may correspond to noise or other less significant features of the matrix. Also as we drop only singular values near 0, we have almost 100 percent of the explained variance left. Also the eigenvalues and eigenvectors are preserved (for non-zero eigenvalues). We are not preserving the rank however.

3.

Approximately 1 percent of the data is missing. Computing the rank of R1,R2 and R3, we see that they are 2, 4 and 5 respectively and so we can infer that assuming all 3 matrices have filled in U using low-rank factorization methods, that they have been done using different ranks. The rank of the constructed R is at most p, where p is the chosen rank, so we can guess that R3 did LRF with rank 5 and most likely R2 with rank 4 and R1 with rank 2. This makes sense when we print the singular values of R1, R2 and R3: we find that they have 2, 4 and 5 non-zero singular values respectively meaning and we find that the non-zero one are the same. Presumably this means that each method has preserved p of the singular values of the original U.

In order to compare the reconstructions I will consider that the aim of these methods is the fill in the data without introducing new trends, which we can do by aiming for a small rank whilst maintaining and accurate reconstruction. The tradeoff in low-rank factorization for filling in missing data is between the accuracy of the inferred values and the complexity of the model. We can see from calculating the Frobenius norms for the set of known values that R3 minimises the cost function on the data available much better than R2 by orders of magnitude and R2 compared to R1 also by orders of magnitude. However, it is not clear how well this would generalise to the missing data (if we overfit). Looking at the percentage of explained variance (as a fraction of just the first 5 as the singular values after are 0), we can see that R1 has 94 precent of the explained variance of the first 5 values, R2 has 99.8 percent and R3 has all of it. Overall, I would recommend use of matrix R2. R1 has poor cost minimisation and only explains 94 percent of the variance R3 does. But R3 does not do much better than R2 on explained variance (0.2 percent) and the additional part may be introducing new trends. It is best to take the elbow of the curve of singular values which is near 3 or 4, closest to R2.

Further calculations that I could perform would take the rows of data that do not have any missing values as a sort of validation set, get this submatrix, randomly drop values from the submatrix (1 percent for consistency) to create a set to train with each rank value and evaluate of the actually full submatrix with the Frobenius norm. The rank that gives the best performance on the validation set is then selected. This way we could empirically determine the best rank and use validation to test for overfitting from R3. Further calculations of this form are necessary to be sure. I would also investigate the rank of another matrix which I have calculated using rank-3 low-rank approximation, because we cannot be sure that the best tradeoff is not between ranks 2 and 4.

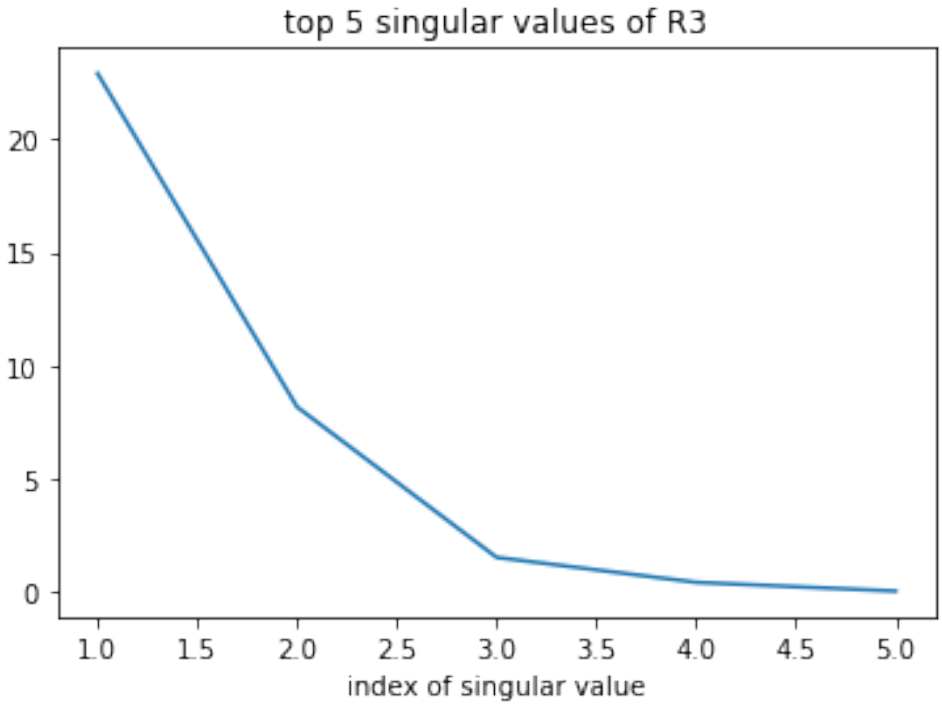


Figure 1: Figure 1: Add figure description here

Part 2

1.

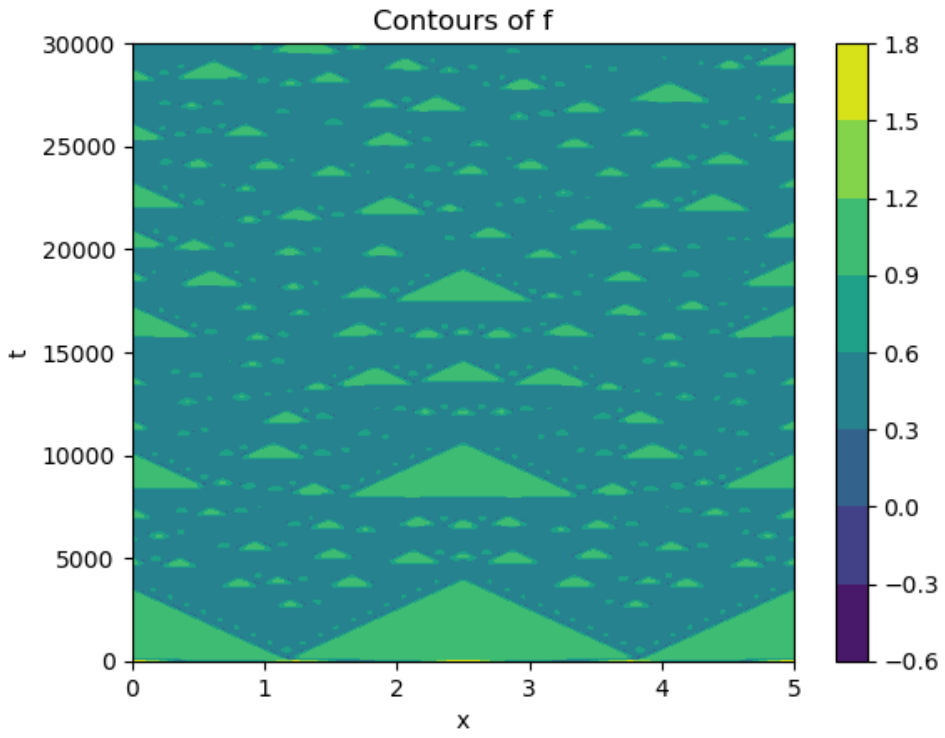
Initially the f value starts at the initial condition near 1.75 and quickly adapts before getting to 1 and staying there for a while. The equilibrium of the system for f at space 0 seems to be somewhere between 0.25 and 0.50 and sometimes increasing up to 1 maximum, and similarly at space index 200 it seems to have equilibrium near 0.4, as the values keeps returning to these value later and oscillating around them. Before time 500 we can also see that the behaviour is not at all sinusoidal but rather more random (see plot 3) hence has not yet reached a settled dynamical state. I will take T=400 as the end of the transient though at different points in space the simulations take different times to adapt to the intial conditions. The transient is over at a point in space either if f is constant at 1 or fluctuating around its equilibrium. I remove data before the transient time. Plot 9 and the first plot show that, after the transient, the reaction starts only at space index 60 and its symmetric opposite and spreads sideways in space until by time around 5000, the reaction has started everywhere. A light green triangle in plot 1 corresponds to the reaction having stopped (or not started) and spreading inwards in space between the 2 points where the reaction is happening to all the points where the triangle is. At space point 5000, the reaction only starts at time 5000, so the concentration is at 1 until then. The colour shows the concentration. Plot 9 shows the initial spreading over the reaction (each plot is more advanced in time).

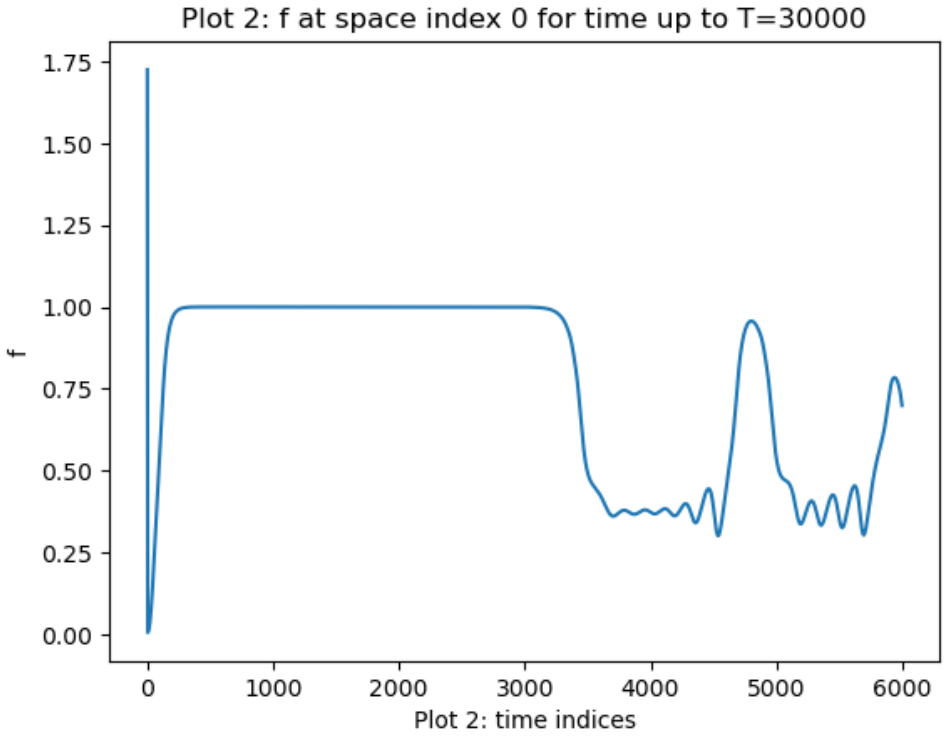
We see in plot 2 that the reaction between f and g does not start at space index 0 for a while as f has constant concentration 1 up to around 3500 time. The reaction starts a lot sooner forplot 3 at space index 200. f fluctuates as expected inversely proportionally to g, i.e. when f increases, g decreases and vice-versa. Also when f is oscillating closely to its equilibrium, g also oscillates closely, and when f rises dramatically above to near 1, g decreases dramatically to go near 0 and comes back to equilibrium at a similar time.

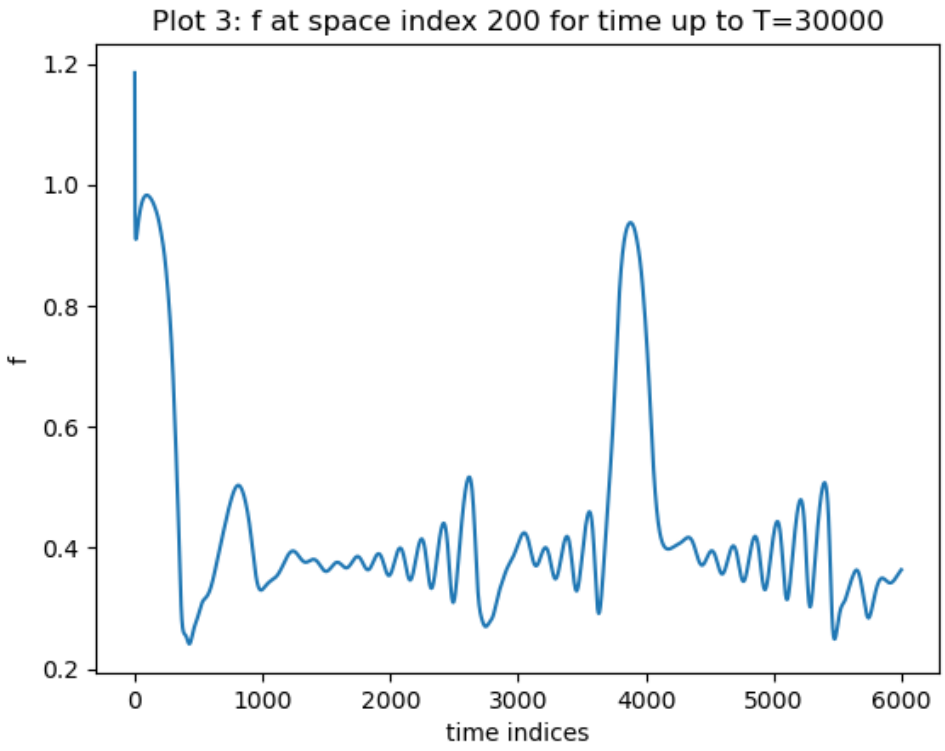
In plot 5, I take a closer look at the fluctuations of f. It seems that there is an equilibrium value that f at this point in space always returns to, oscillating around it sinusoidally with increasing amplitude as time increases until the amplitude becomes very large and the curve shoots off to close to 1, then comes back down to the equilibrium and repeats this trend. Hence the system is time-dependent and has somewhat sinusoidal behaviour but it not periodic - the amplitude can shoot up to 1 at an artibtrary time each time it has come back to near 0.4. Also as amplitude increases, sinusoidal waves can become progressively decentred from the equilibrium.

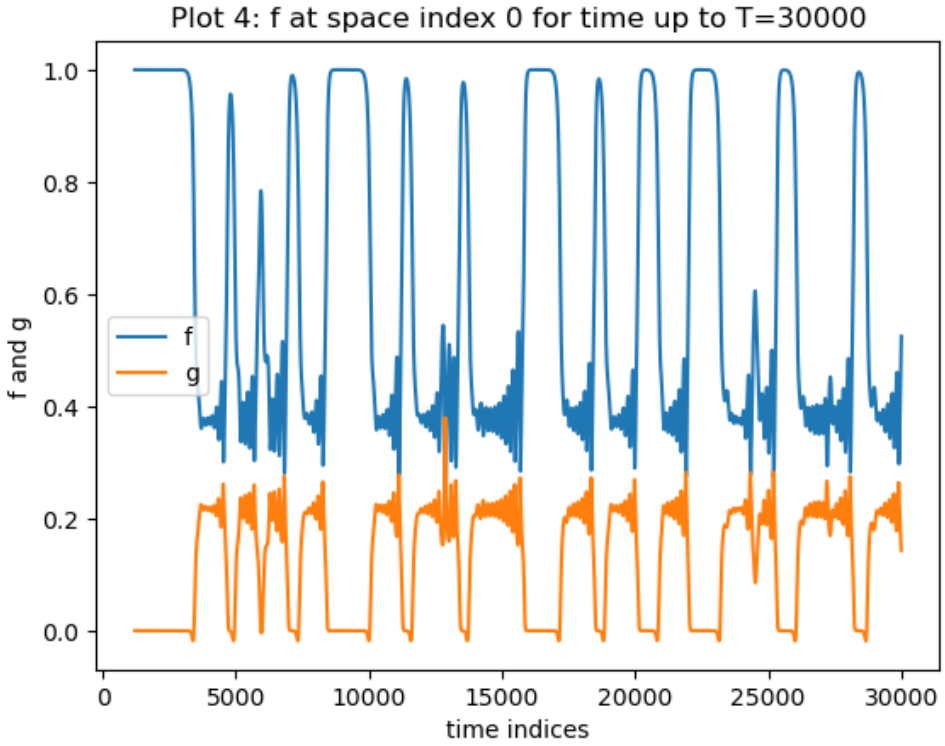
Considering how f fluctuates in space, plot 6 shows an example of how at the start, f is symmetric in space, though there is not observable sinusoidal behaviour. We can see in plot 7 that up to about time 15000 we can only see 3 colours, meaning that the graphs for points 1 and 4 and points 2 and 3 in space have the same values, i.e. there is symmetry. Then, past 15000, we can see both plots, which means that the symmetry in space breaks down. We see an example in plot 8. Still after the symmetry breaks, the behaviour for a fixed point in space is consistent to before in time.

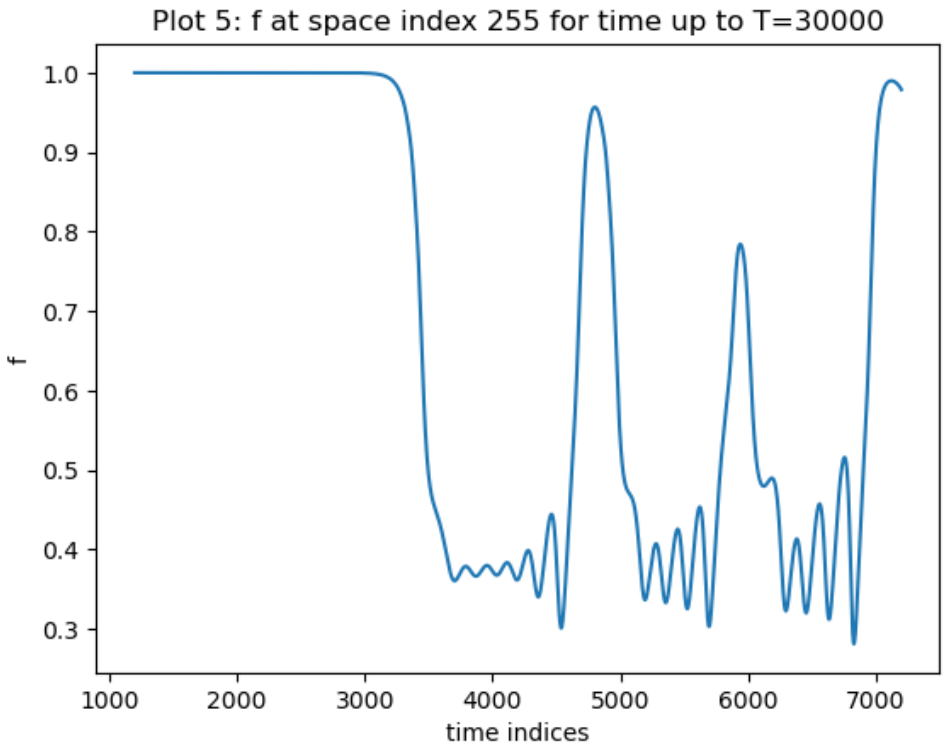
To investigate chaotic dynamics, I have used Welch's method to find tau and m and have found the correlation sum for increasing thresholds at the point in space in the middle where the reaction does not start until time 5000 and at the point where the reaction starts first (index 196). Plotting log log graphs of the correlation sums against the thresholds and fitting the curve where it is linear using polynomial fit to find the slope, we take this as an apporximation of the fractal dimension of the simulations at that point in space. I am only using time up to 5000, which is why the fractal dimension is near 0 in the middle as the reaction has not started, so f is not chaotic at this point in space as it is almost constant. Meanwhile, at the other point, the fractal dimension is near 2, and so we observe that when f is in reactive state most of the time, the system exhibits chaotic behaviour.

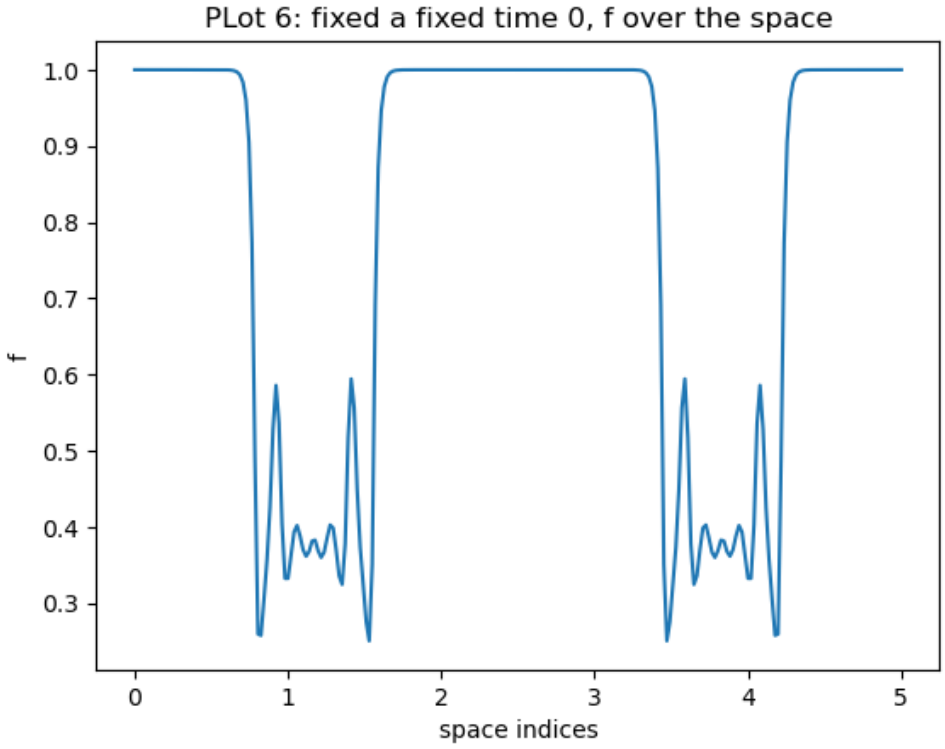

Contours of f
30000
25000
20000
15000
10000
5000
0
t
0
1
2
3
4
5
x
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6

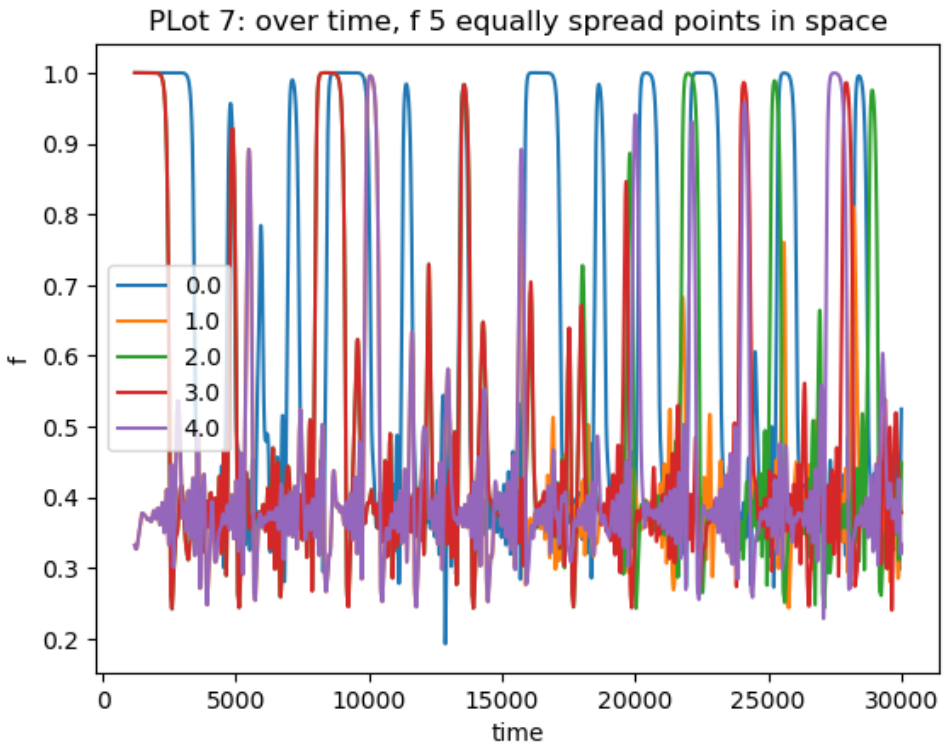

Plot 2: f at space index 0 for time up to T=30000
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
f
0
1000
2000
3000
4000
5000
6000
Plot 2: time indices

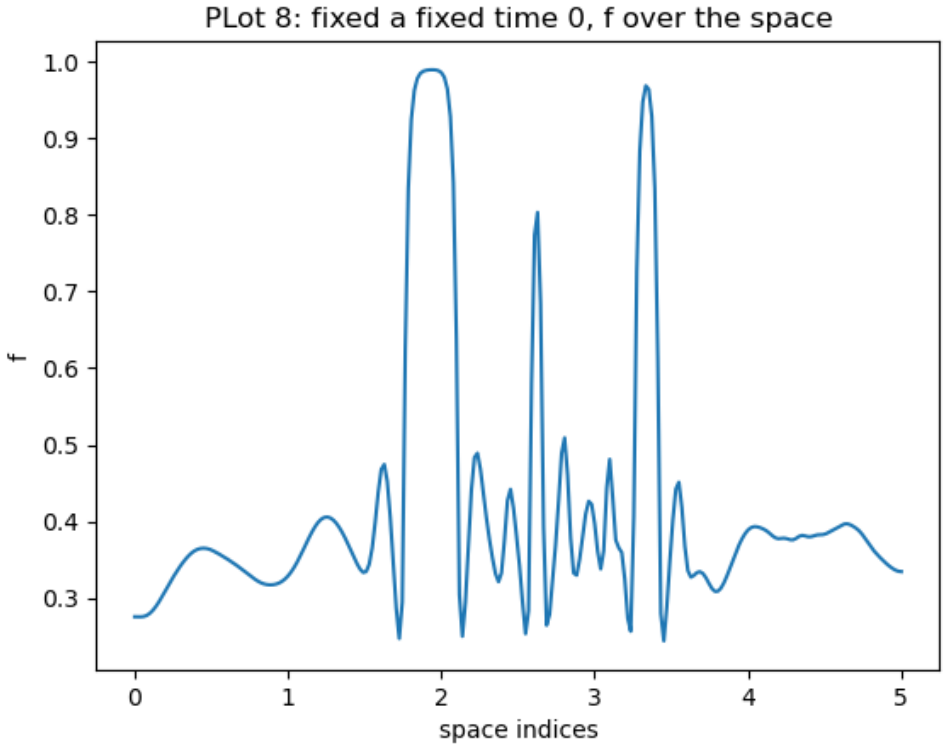

Plot 3: f at space index 200 for time up to T=30000
f
time indices
1.2
1.0
0.8
0.6
0.4
0.2
0
1000
2000
3000
4000
5000
6000

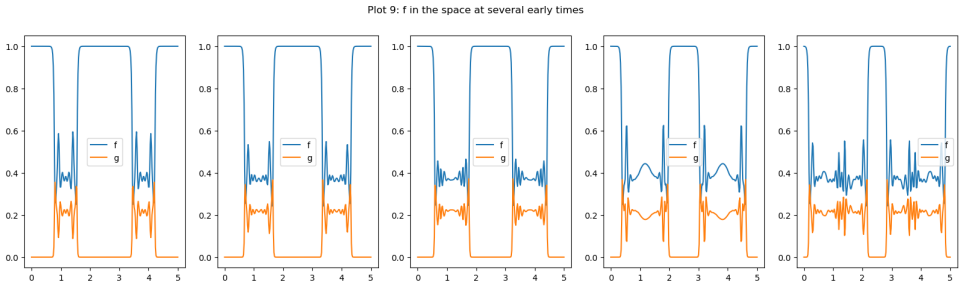

Plot 4: f at space index 0 for time up to T=30000
f and g
time indices
f
g
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000
30000


Plot 5: f at space index 255 for time up to T=30000
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
f
1000
2000
3000
4000
5000
6000
7000
time indices

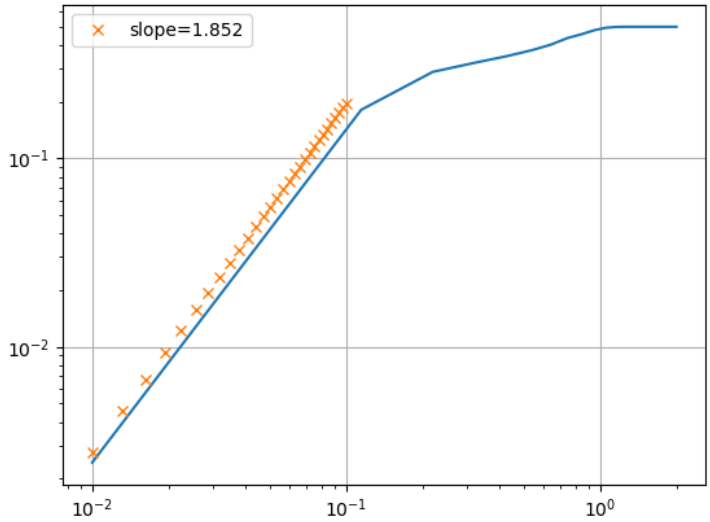

PLot 6: fixed a fixed time 0, f over the space
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
f
0
1
2
3
4
5
space indices


PLot 7: over time, f 5 equally spread points in space
f
time
0.0
1.0
2.0
3.0
4.0

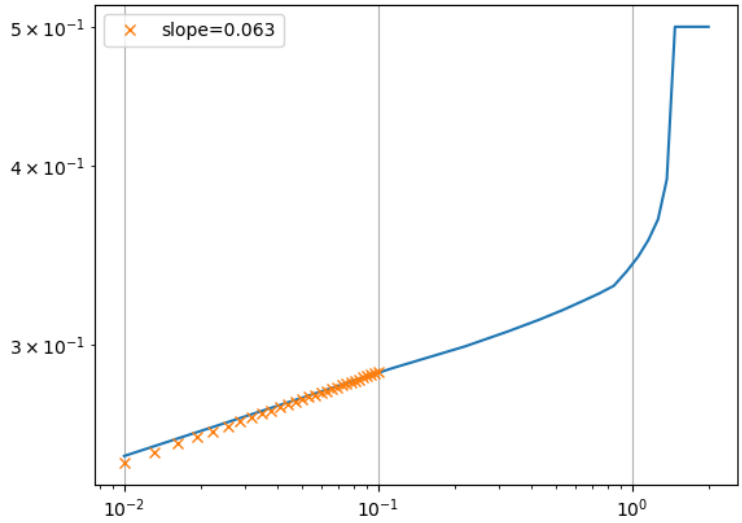

PLot 8: fixed a fixed time 0, f over the space
f
space indices


Plot 9: f in the space at several early times

Plot 10: Fitted log of correlation sum against log of thresholds used at space index 196



Plot 11: Fitted log of correlation sum against log of thresholds used at space index 128



2.ii

When implementing centred finite difference scheme, I adapt the left boundary to use the forward difference and the right boundary to use the backwards difference.

I will do some wavenumber analysis on the various implementations. I consider data corresponding to a complex sinusoidal wave with wavenumber k. Then the second derivative is $-k^2 \exp^{ikx}$. For a second order finite difference scheme where $f_j'' = \frac{f_{j+1}-2f_j+f_{j-1}}{h^2}$, taking out a common exponential factor, we get $f_j'' = \frac{\exp^{ikx_j}(2\cos kh-2)}{h^2}$,hence we can plot k^2h^2 against $2-2\cos kh$ to see how closely the scheme approximates the true values. Similarly for the implicit finite difference scheme, we can use substitution on the left and right hand side to obtain the final expression to compare to k^2h^2 of $\frac{\frac{b}{2}(\cos 2kh-1)+2a(\cos kh-1)}{-(2\alpha\cos kh+1)}$. I plot these all on the same axes and we can observe in plot 1 that both the centered and implicit schemes match the exact curve for small values of kh then both start to diverge as kh is increased but the centered scheme curve diverges faster. Plot 2 shows the percentage error as kh increases, which we observe to be 1 percent at around kh=0.3, meaning $\frac{\lambda}{h} = \frac{2\pi}{kh}$ is roughly 21, so we would need roughly 21 points per wavelength to get 1 percent error. Repeating this for the implicit scheme, kh is roughly 1.75 in plot 3 and so for 1 percent error we require about 4 points per wavelength. We know that the DFT method requires just over 2 points per wavelength so both DFT and implicit are likely maintain higher accuracies on higher wavelength functions (small scale) as they require fewer points when the number of wavelengths blows up.

DFT however has the additional constraint that the function must be periodic hence cannot be used

in many cases.

Looking at the time complexities of the various methods according to the way I have implemented them, we know that the second order scheme is of order O(N) with a small number of multiplications and additions per N. Meanwhile, the implicit scheme implemented is requires solving a linear system with 3 bands along the diagonal, but is also of order N, though we expect the computational cost to be higher as there are more operations per N. Finally, the DFT scheme as in lectures is of order NlogN so we expect this to be slower than the central finite scheme.

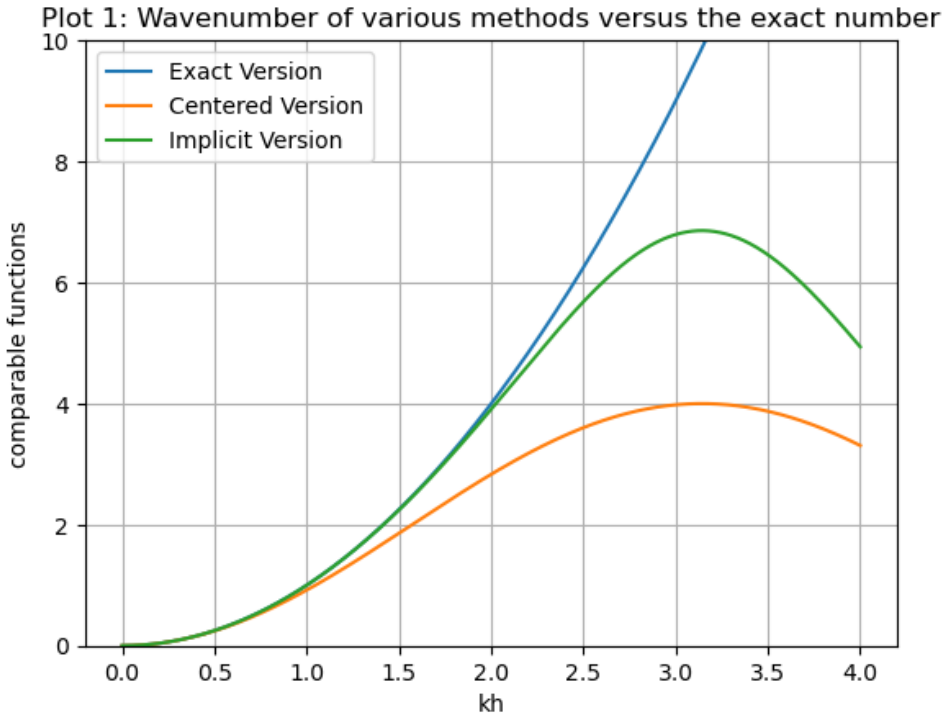
I implement a second derivative function for both x and y using centred second order finite differences in order to be able to make comparisons of accuracy and speed. I test the methods on a function which is sinusoidal in both x and y directions.

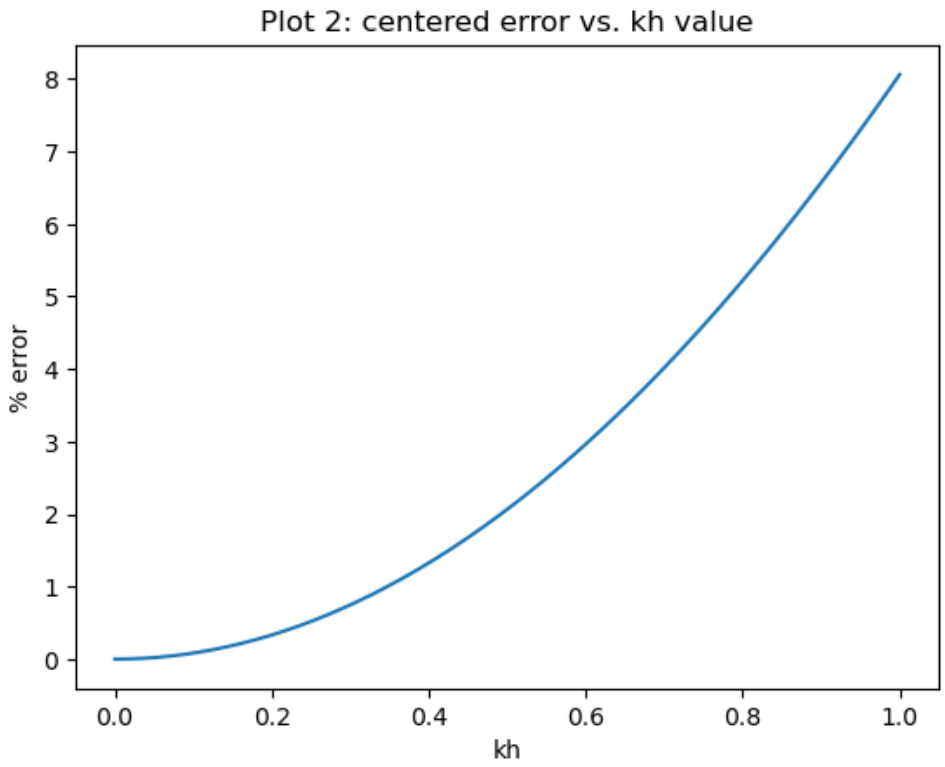
Plots 4 and 5 show that both methods match up to the exact values visibly well. Plot 6 shows that both methods have low MSEs so are very accurate, though the implicit method is more accurate for all N (as it is a more complex method). Both methods have decreasing MSEs as N increases and the accuracy of the methods increase but implicit outperforms.

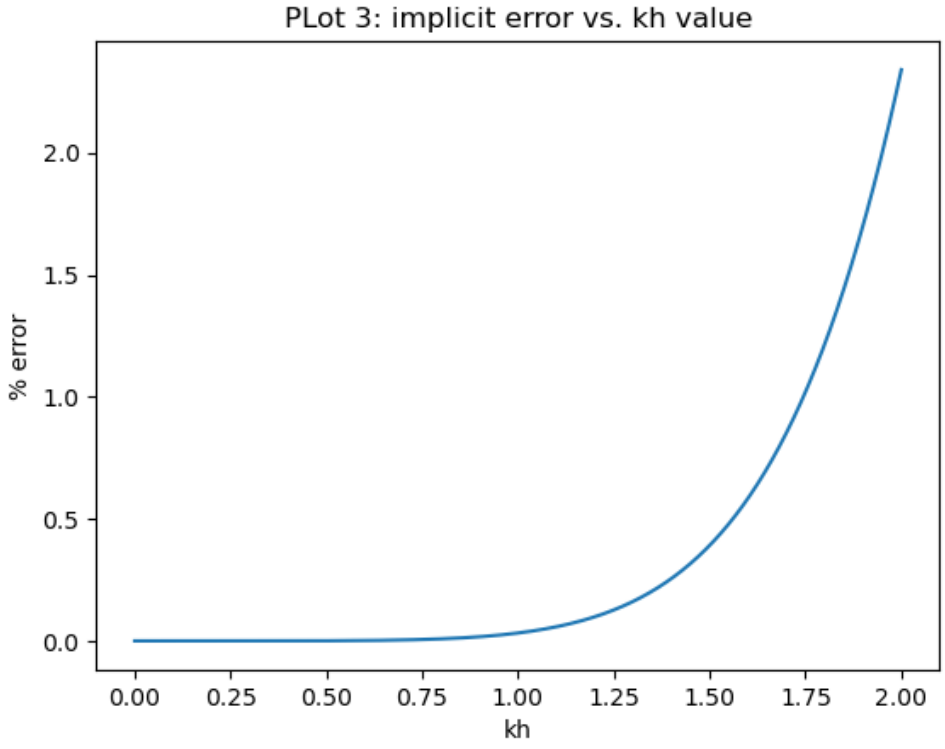
I will also time the functions to empirically compare the costs. Plot 7 shows that both functions linearly increase in cost as N increase as expected, and also, as supported by the theoretical running times, the implicit function is linear with a larger constant i.e. is more costly than the centred method. There is hence a tradeoff between the time taken by implicit compared to centred and the increased accuracy and which to choose depends on the accuracy required.

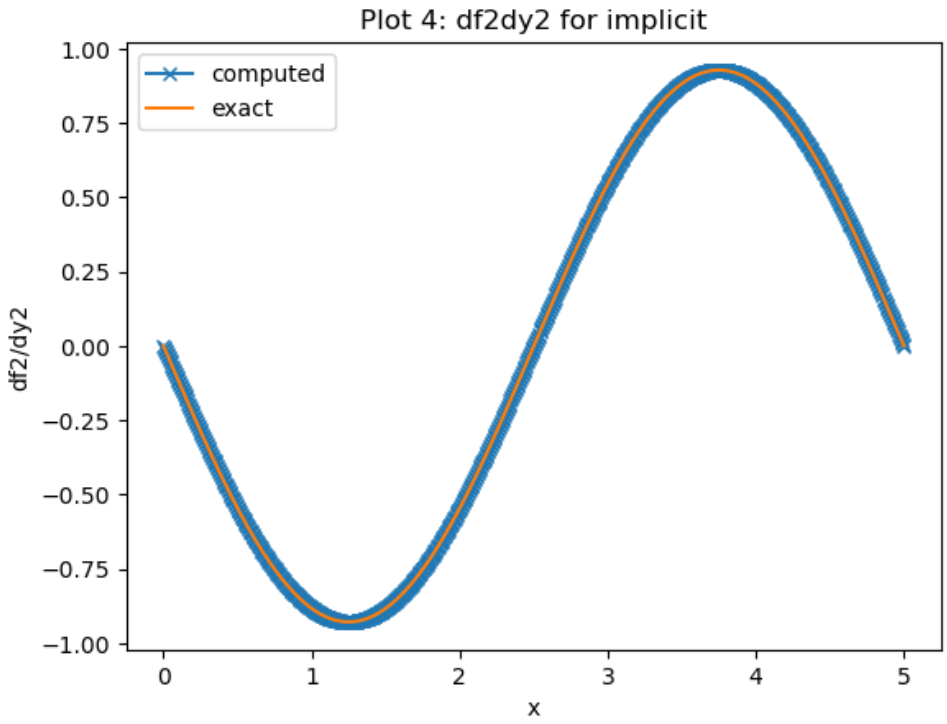
In order to empirically investigate effectiveness of the functions with increased wavenumber, I test the functions on a sinusoid with very high frequency. Plots 8 and 9 show how the implicit and centred methods fit the function with high likelihood, and we can see there are few points per wavelength but that the visual accuracy of the points is high. Measuring the MSEs, we see the error increase quite a bit for the centred method, where quite a bit of accuracy is lost for low number of points per wavelength.

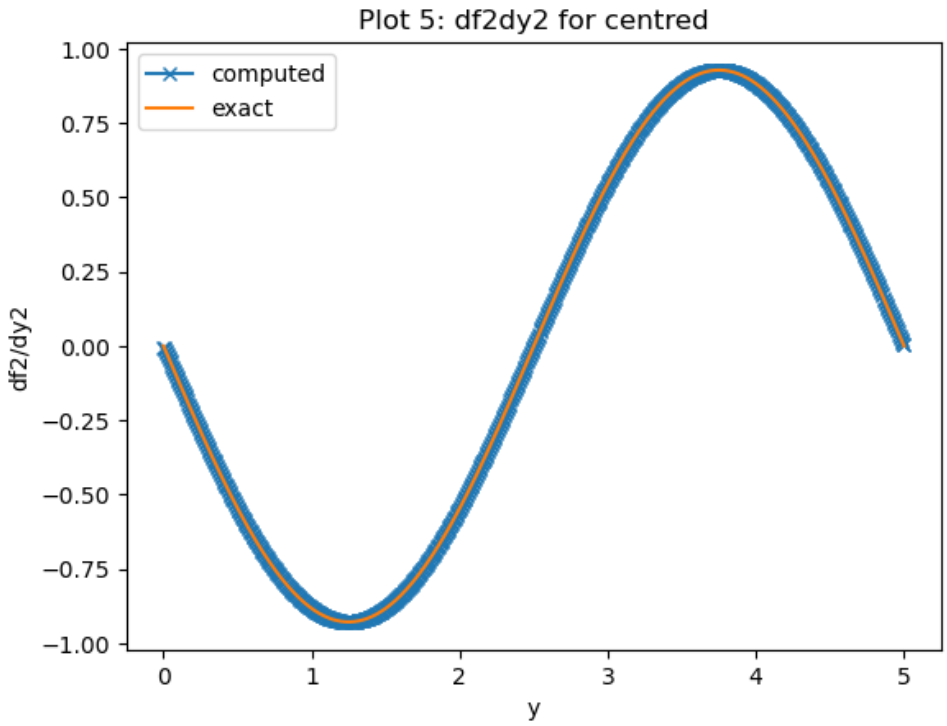
Plot 10 shows that the DFT function is also visually very close to the exact curve. The final 2 plots show that the DFT method takes longer than the centred method as expected for all values of N and that the errors of the DFT method are almost and 0, much smaller than for the centred method though both methods have decreasing MSEs as N increases.



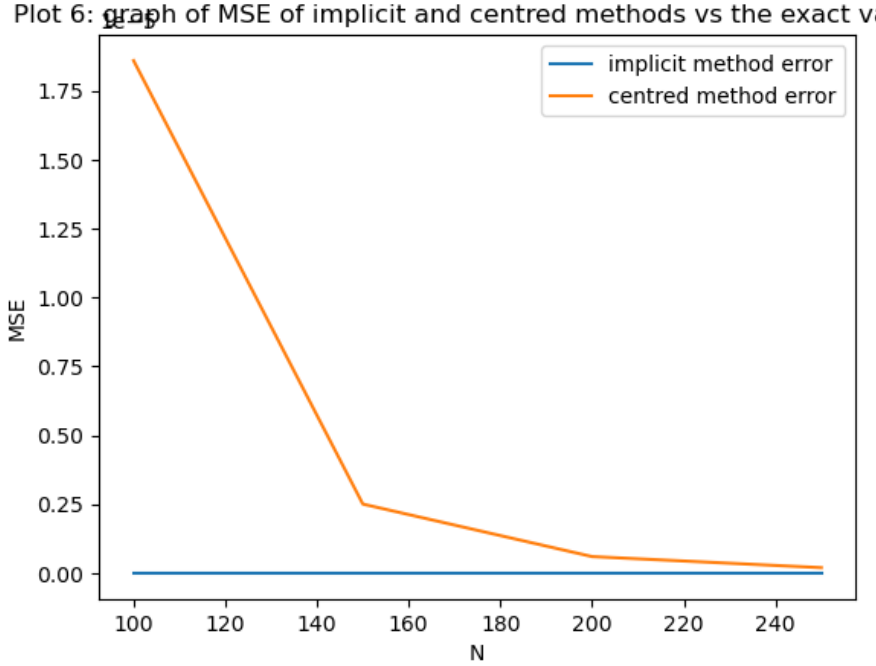

Plot 2: centered error vs. kh value
% error
kh
0
1
2
3
4
5
6
7
8
0.0
0.2
0.4
0.6
0.8
1.0


PLot 3: implicit error vs. kh value
% error
kh
0.0
0.5
1.0
1.5
2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

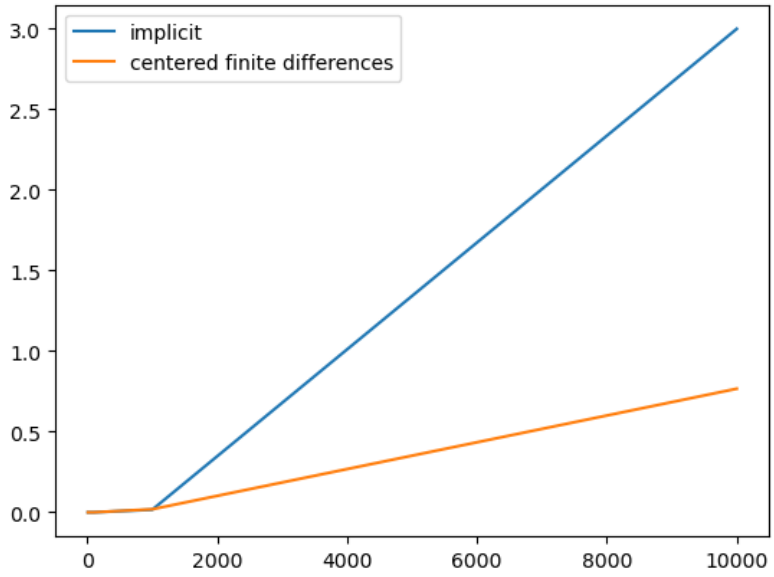

Plot 4: df2dy2 for implicit
computed
exact
df2/dy2
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5

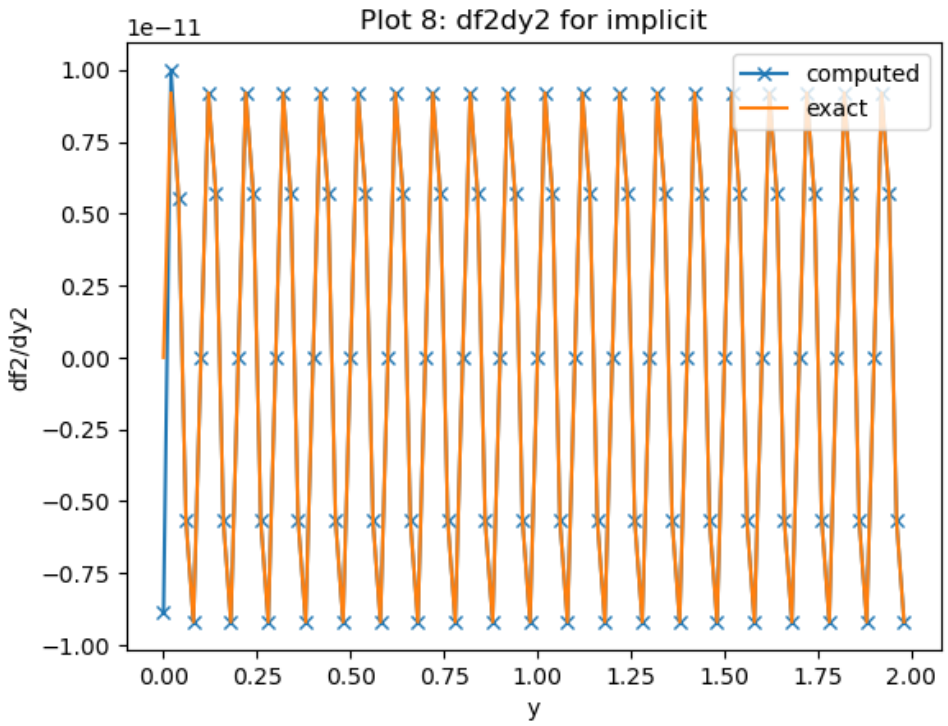

Plot 5: df2dy2 for centred
computed
exact
df2/dy2
y
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5

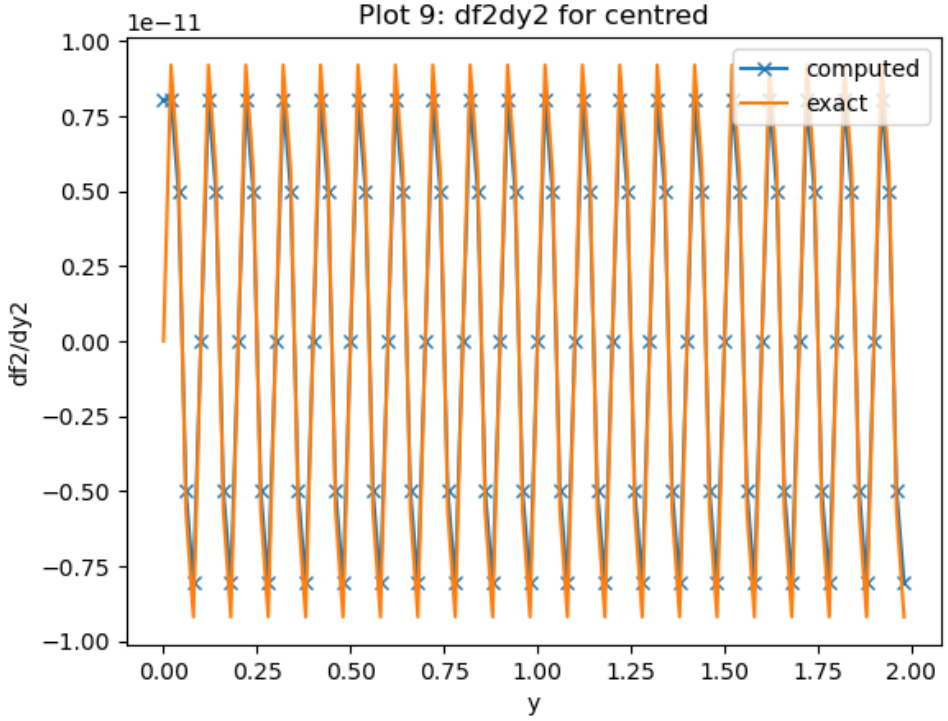
Plot 6: graph of MSE of implicit and centred methods vs the exact values

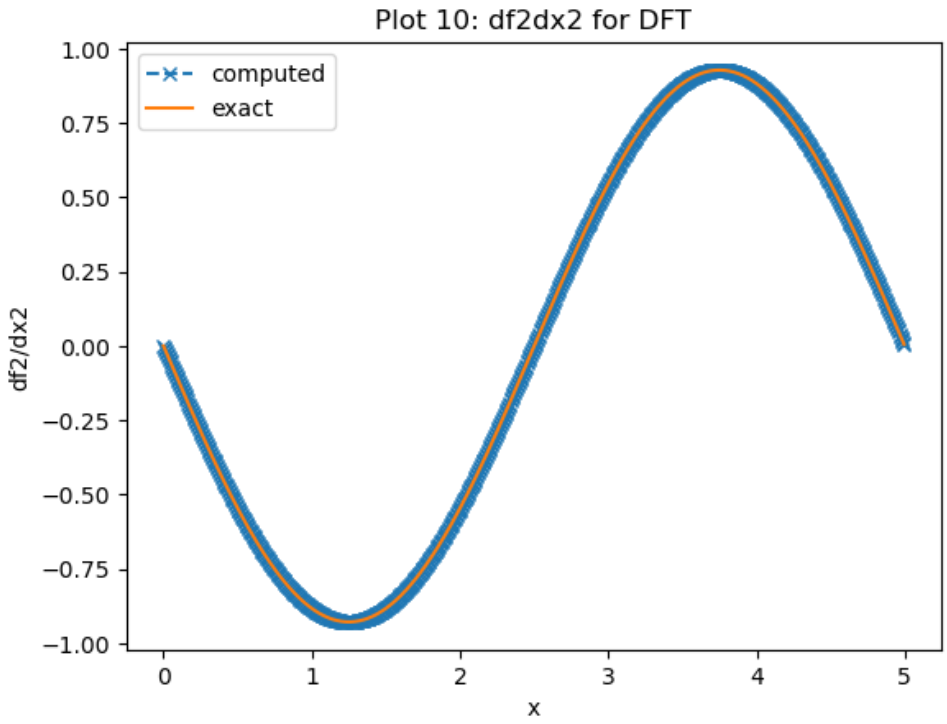


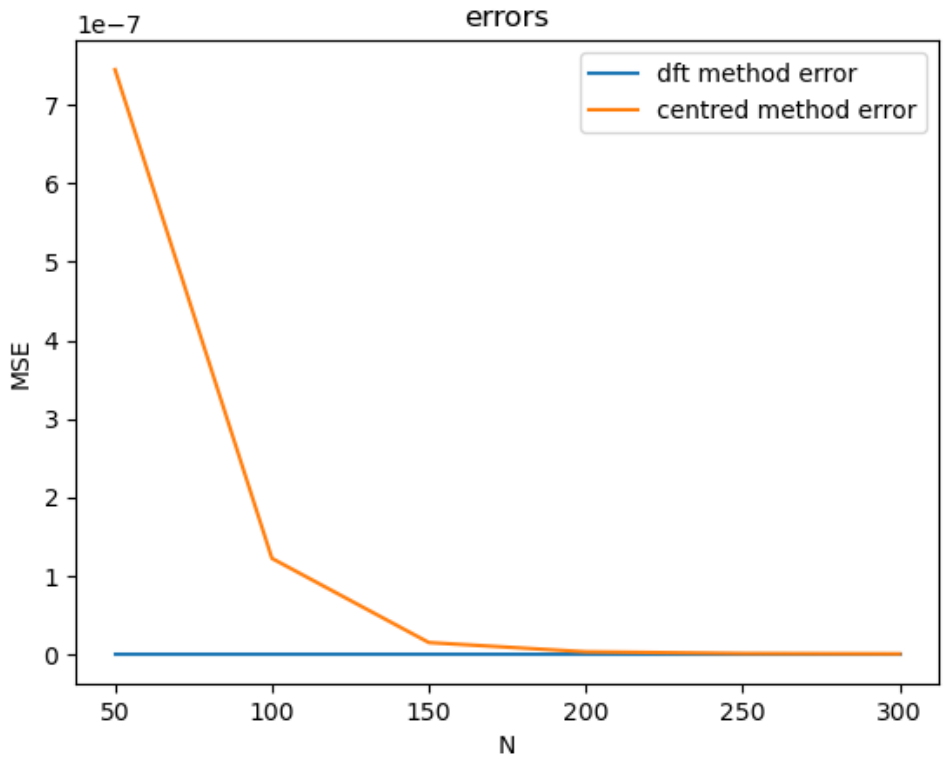
Plot 7: relative speeds of the implicit and centered methods with increasing N




Plot 8: df2dy2 for implicit
1e−11
computed
exact
df2/dy2
y
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Plot 9: df2dy2 for centred
1e−11
computed
exact
df2/dy2
y
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Plot 10: df2dx2 for DFT
computed
exact
df2/dx2
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5


errors
1e−7
dft method error
centred method error
MSE
N
7
6
5
4
3
2
1
0
50
100
150
200
250
300

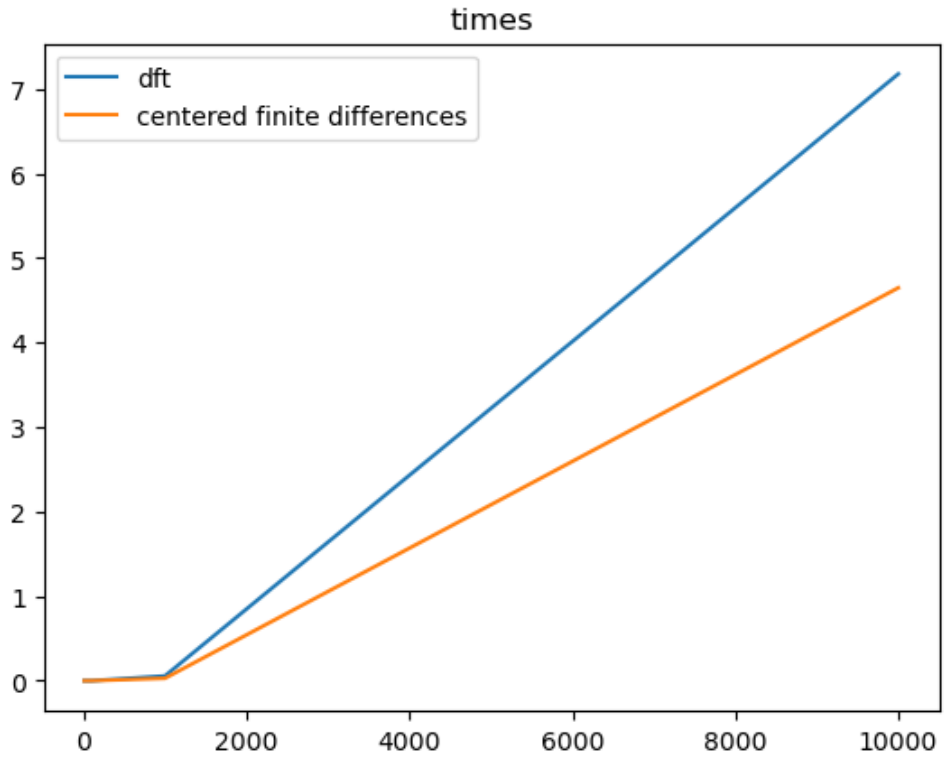


Figure 2: Figure 1: Add figure description here