Scientific Computation Project 3

01724711

March 24, 2023

Part 1

1.

We begin by visualising the blood-flow velocity across the whole time period, $t = (0, 3)$, at regular intervals. As expected we see that the blood moves quickest in the centre of the vessel. We will be interested in analysing how the velocity changes depending on the radius from the centre, r, and the polar angle, θ. We focus specifically on $r = 0.1$ and $r = 0.5$ across all $\theta \in (0, 2\pi)$. For $r = 0.1$ and

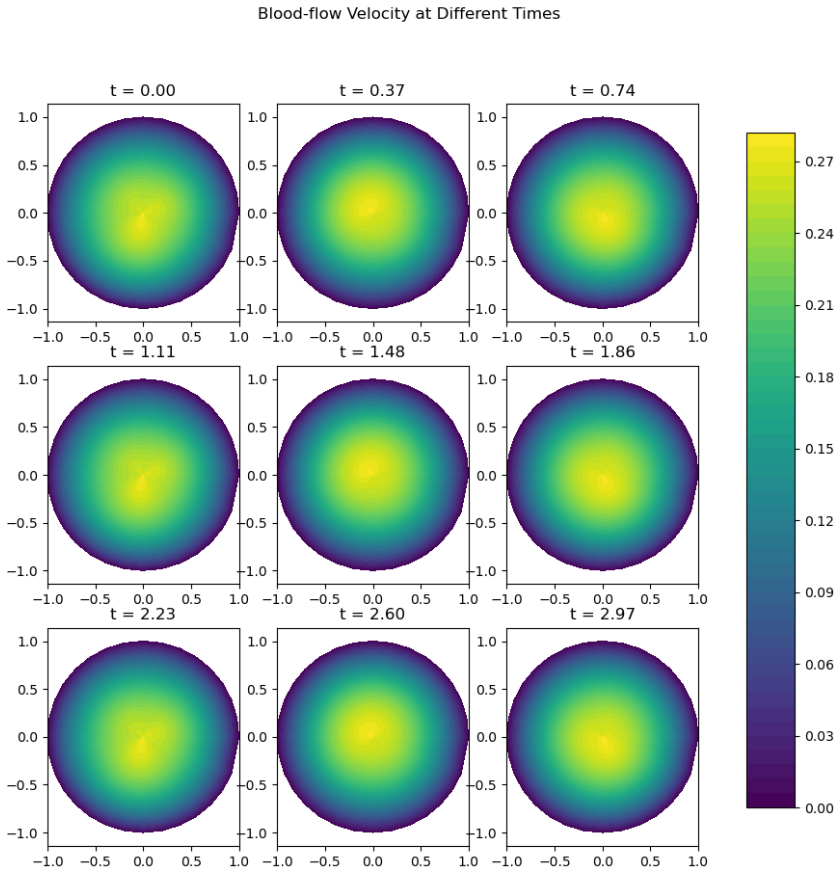

Figure 1: Blood-flow velocity over all time

$r = 0.5$ we observe a periodic oscillatory behaviours across all angles. In the two top plots of figure 2 we see approximate sinusoidal behaviours with some repeating perturbations over a period of roughly 1.1 seconds. We see that at both radii we have the same time period for all angles and that the time period is equal for both radii. The plots for all the angles seem to vary very little from each other in terms of overall shape as we observe a sort of a merge pattern with all the curves quite close to one another. We

now examine the velocity at both radii against the angle in the two plots below for all times. Again, we see that the pattern in the two plots follow the same behaviour. We of course have spacial period of 2π from the problem set up but in these plots we are able to observe that there is no smaller spacial period. We observe smooth but irregular oscillatory behaviour for both $r = 0.1, 0.5$ for each time step. Combining these plots we can show a surface 3D plot at each radius over all time and angles which is shown in figure 3. We are able to observe that the velocity at $r = 0.1$ has almost identical behaviour in time and angle to the $r = 0.5$ plot except that the velocity magnitudes are shifted downward from the $(0.21, 0.28)$ range to $(0.15, 0.21)$. We observe that the amplitude decreases from 0.35 at $r = 0.1$ to 0.3 at $r = 0.5$. The periodic oscillatory behavior observed in the blood-flow velocity data can be related to the regular contraction and relaxation of the heart muscles during each heartbeat. The heartbeat generates a pressure wave that travels through the blood vessels and causes the blood to move in a pulsating manner. This can lead to the observed periodicity in the blood-flow velocity data. As blood flows away from the heart, it encounters resistance from the walls of the blood vessels, causing the blood flow velocity to decrease. This decrease in velocity leads to a decrease in the amplitude of the blood flow waves. Additionally, the further away from the heart the blood flows, the more the blood flow will be influenced by other factors such as branching and narrowing of the vessels, which can further reduce the amplitude of the blood flow waves.

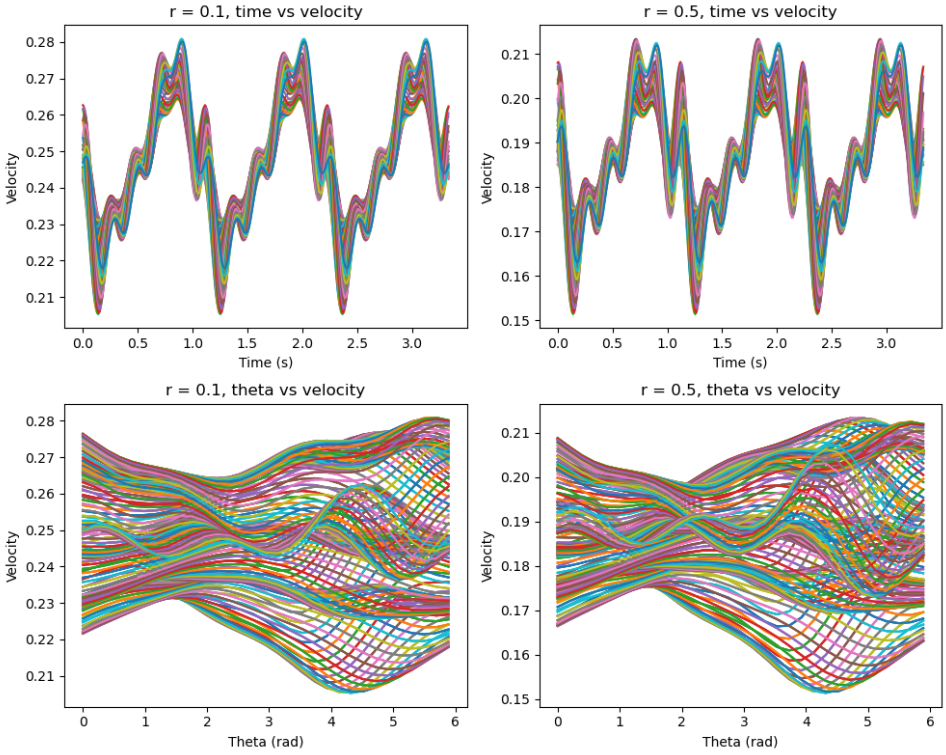

Figure 2: Blood-flow velocity over all time and angles

2.

To construct a compressed version of the dataset, we can use a technique called truncated Singular Value Decomposition (SVD), which is a mathematical technique that can be used to reduce the dimensionality of a dataset while retaining the key features. In the function part1q2C, the first step is to reshape the 3D data (which has dimensions (N_r, N_θ, N_t)) into a 2D matrix $(N_r N_\theta, N_t)$. Next, the centered data matrix is computed by subtracting the mean of each row from each element in that row. After computing the centered data matrix, the SVD is performed on it. The resulting SVD components are three matrices: U, S, and V transpose. Here, U is a matrix containing the left singular vectors, S is a diagonal matrix containing the singular values, and V is the matrix containing the right singular vectors. The next step is to plot the cumulative variance explained and singular values. The cumulative variance explained is a measure of how much of the total variance in the data is explained by each mode. It is computed by summing the squares of the singular values up to each mode and dividing by the total sum of squares.

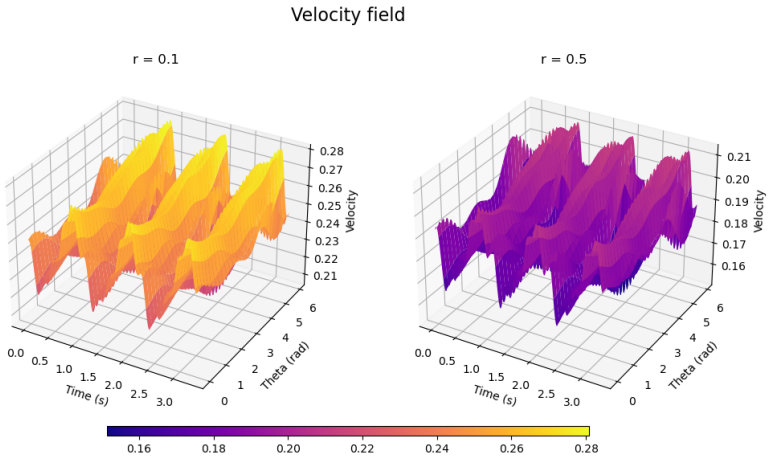

Figure 3: Surface plots of blood-flow velocity over all time and angles

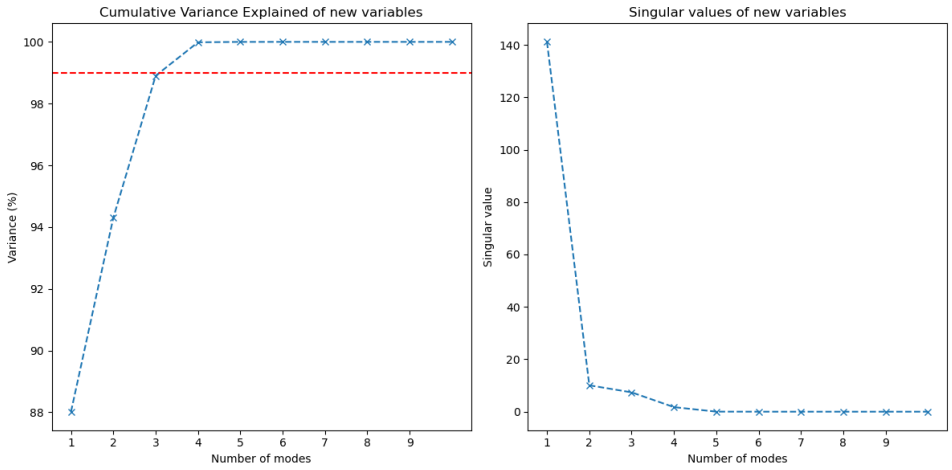

Figure 4: Left: Cumulative Variance Explained of new variables, Right: singular values

The resulting values are plotted against the mode number. The singular values are also plotted against the mode number. These plots are used to determine the number of modes to keep. The number of modes to keep is typically determined by choosing the number of modes that capture most of the variance in the data. This is often done by choosing a threshold value (we choose 99% of the variance), and then selecting the number of modes that explain at least that much of the variance. We observe this threshold to be 4 modes which explain 99.98663737% of the total variance. Finally, a tuple containing the truncated SVD components, row means, original data array, and dimensions of the original data array is returned. The truncated SVD components are obtained by keeping only the first 4 singular values, where k is the number of modes to keep. The row means are the means of each row in the original data array, which were subtracted to obtain the centered data matrix. The original data array and its dimensions are also returned for reference. In the function part1q2E, we aim to compress the dataset using the components obtained in the previous step. To construct the compressed dataset, we first truncate the SVD components by keeping only the first 4 columns of the U matrix and the first 4 rows of the transpose of the V matrix, which is denoted as WT in the code. We also keep only the first 4 singular values. Next, we reconstruct the centered data using the truncated SVD components and the row means. This is achieved by multiplying the truncated U and WT matrices, along with the singular values, to obtain an approximate version of the centered data. We then add back the row means to obtain an approximate version of the original data. Finally, we reshape the approximate original data back into the original 3D shape by using the dimensions of the original data array. This gives us the compressed dataset, which contains the same information as the original data but uses fewer components/modes to represent it. The key features retained in the reconstructed array depend on the number of modes kept during the

SVD truncation. By retaining the first 4 columns of U and first 4 rows of WT, we are retaining the most important patterns in the data. By retaining the first 4 singular values, we are retaining the most important information about the magnitude of the patterns. Therefore, the compressed version of the dataset retains the most important patterns and magnitude of patterns, which are the key features of the dataset. The memory usage of the compressed data array is significantly lower than the memory usage of the original data array, as we only store a subset of the SVD components, row means, and singular values. In our implementation, we calculate the reduction in memory usage to be from 9039600 bytes in the orginal data array to 115480 bytes in the compressed array. This is a percentage reduction in memory usage of 98.72% which we calculate via $100 * (1 - \text{memory usage compressed/memory usage original})$. When we plot the same 3 by 3 set of images as in figure 1 for the compressed data we see the exact same images to the naked eye, so we can confirm that we have indeed maintained most of the important information from the original.

3.

The three repaired matrices (R1, R2, R3) are different because they were generated using the same low-rank matrix factorization algorithm with different rank parameters. Specifically, R1 was generated using a rank of 2, R2 using a rank of 4, and R3 using a rank of 5.

The rank parameter controls the maximum number of factors (or "latent features") used to approximate the original matrix. In other words, a higher rank allows the algorithm to capture more complex patterns in the data, but at the cost of a larger number of factors. Conversely, a lower rank results in a simpler approximation with fewer factors.

When we apply the low-rank matrix factorization algorithm to repair the matrix, the resulting completed matrix will depend on the rank we choose. Specifically, a lower rank will lead to a simpler approximation with fewer factors, which may not capture all the complexity of the data, resulting in a larger approximation error. On the other hand, a higher rank can capture more of the complexity of the data, but may overfit the data and lead to a larger reconstruction error.

In summary, the choice of rank in low-rank matrix factorization can have a significant impact on the resulting completed matrix. A lower rank will result in a simpler approximation with fewer factors, but may not capture all the complexity of the data, while a higher rank can capture more of the complexity of the data, but may overfit the data and lead to a larger reconstruction error.

MSE, or mean squared error, is a common metric for evaluating the quality of an image or data reconstruction. It measures the average of the squared differences between the original and reconstructed data. A lower MSE value indicates a better reconstruction.

Structural similarity index (SSIM) is another metric for comparing images or in this case, matrices. It takes into account the perceived change in structural information, luminance, and contrast of two images. SSIM values range from -1 to 1, with 1 indicating identical images, and 0 indicating completely dissimilar images.

In the context of comparing the repaired matrices, a higher SSIM value between the original matrix and a repaired matrix indicates that the repaired matrix better preserves the structural information of the original matrix. This means that the missing values have been filled in a way that does not significantly alter the underlying patterns or structure in the data.

Since SSIM provides a measure of structural similarity, it can be useful in situations where the underlying structure of the data is important. In this case, the velocity data likely has a structure that is important for analysis, and therefore the repaired matrix with the highest SSIM value could be considered the best choice.

PSNR, or Peak Signal-to-Noise Ratio, is a measure of the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. In the context of image processing, PSNR is used as an objective metric to evaluate the quality of a reconstructed image compared to its original.

In the case of the velocity data, we can consider the missing data as corrupting noise and the reconstructed matrices as the signal. A higher PSNR indicates that the reconstructed matrix is closer to the original matrix in terms of mean squared error and is therefore a better quality reconstruction.

PSNR is especially useful for detecting small differences between high-quality images, where the differences may not be apparent to the human eye. In the case of the velocity data, we can use PSNR to compare the three repaired matrices and choose the one with the highest PSNR as the best quality reconstruction.

We add the code to calculate these metrics in part1q3 and summarise the results in the table below.

	R1	R2	R3
Rank	2	4	5
MSE	0.0010065432957402413	1.3245943796110803e-06	3.080613210676337e-09
SSIM	0.9740874545076164	0.9997910847605184	0.9999976751689165
PSNR	32.38	61.19	87.52

From these results, we can see that the rank of R1 is the smallest, indicating that it has the lowest complexity and may be the most computationally efficient to use. However, its MSE value is the highest among the three repaired matrices, indicating that it has the highest level of error in comparison to the original matrix.

On the other hand, R3 has the highest rank and the lowest MSE, indicating that it has the most information retained and the lowest level of error. However, it also has the highest complexity, which may make it more computationally expensive to use.

In terms of SSIM, both R2 and R3 have very high values, indicating that they have a high level of similarity with the original matrix. However, R1 has a lower SSIM value, indicating that it may have a higher level of dissimilarity with the original matrix.

Finally, from the PSNR values, we can see that R3 has the highest value, indicating that it has the highest level of similarity with the original matrix, while R1 has the lowest value, indicating that it has the lowest level of similarity with the original matrix.

Based on these results, we can recommend using R3 as the best repaired matrix, as it has the lowest level of error and the highest level of similarity with the original matrix, despite its higher complexity. However, further calculations may still be necessary, depending on the specific application and performance requirements.

Part 2

1.

We observe that f exhibits fairly complicated dynamics from the top left plot of f(x=0.5,t). The spatiotemporal behaviour is shown in the top right plot. We see a repeating yet not regular triangular fractal-looking pattern in the oscillatory behaviour in both space and time. We see clear symmetry in the behaviours about the midpoint in space, x=2.5. A more insightful view of the spatial fluctuations is obtained by computing the DFT of f(x,t=3000). The results are shown in the middle left figure. The spectrum decays relatively slowly and the energy is distributed fairly broadly across wavenumbers. The lower left figure shows the power spectra for the first 3 non-zero wavenumbers where DFTs are applied for f(x,t) at every t and then Welch's method is applied to ther result for the three wavenumbers. There appears to be one peak and examining the figure shows that for smaller wavenumbers the first peak occurs at lower frequencies. The large drop in power as frequency and wavenumber increases suggests that the dynamics may be more simple than we initially think. We can see that the results for k=0 (no variation in x) are just about the most energetic. The middle right plot shows the local maxima vs minima at x=0,1,2. We see a range of points but they all appear to lie on straight line segment between (0.2,0.2) and (1,1). This may suggest that the dynamics we observe are not completely simple periodic oscillations but also not complete chaos as then we would expect to see a broad range of patternless points. The correlation sum result in the lower right figure shows that the correlation dimension to be around 2. For an autonomous system of ODEs, a dimension greater than two is necessary for chaos, and our dimension estimate for our system indicates weak or low-dimensional chaos.

2.ii

We implement the explicit 2nd order centred FD scheme.

$$f_{xx}(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \mathcal{O}(h^4)$$

Where $f_{xx}(x)$ is the second derivative of the function $f(x)$ with respect to x, h is the grid spacing, and $f(x+h)$ and $f(x-h)$ are the values of the function $f(x)$ evaluated at $x+h$ and $x-h$, respectively.

Figure 6 shows results for the error and speed of the implicit and explicit finite- difference (fd) methods. The test function is $f(x) = cos(x)sin(y)$, and the top-left plot shows the average error as the

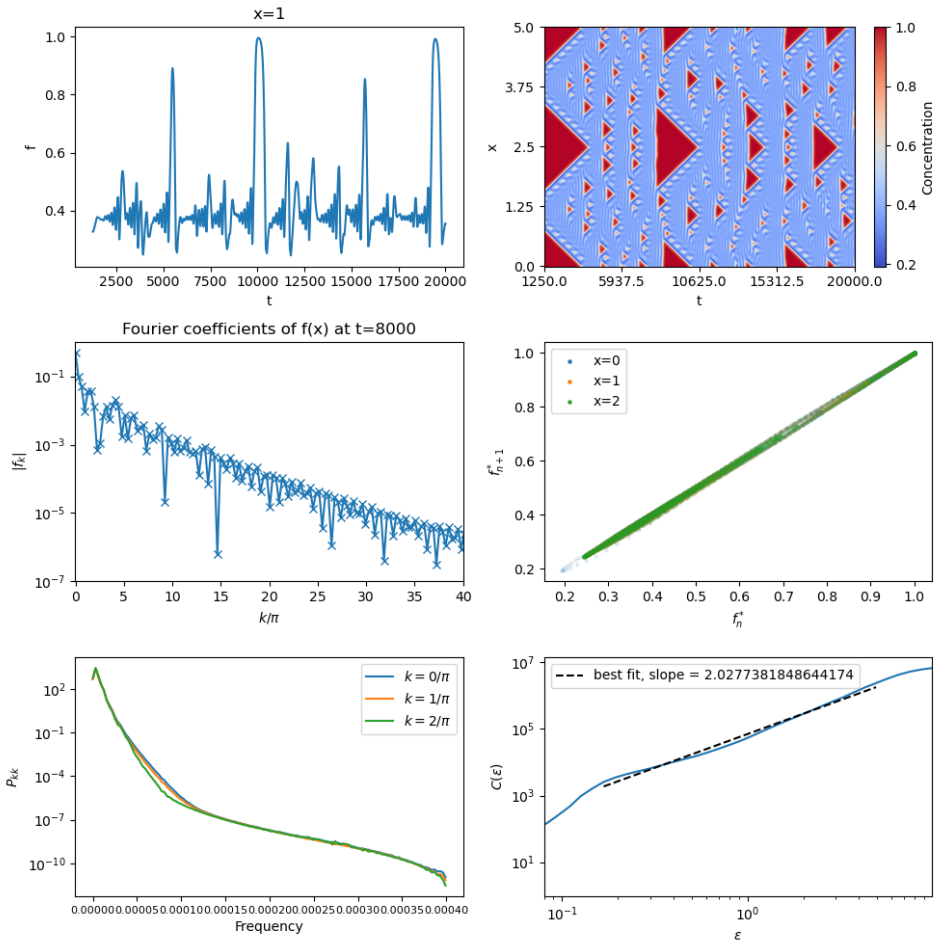

Figure 5: Top left: variation of f with time at x=1; top right: contours of f(x,t); middle left: spacial spectra at t=8000; middle right: extrema at x=0,1,2; lower left: power spectra; lower right: correlation sum results

grid spacing (h) is varied. Both the FD methods follow an approximate 4th order trend for small h. However, the DFT method shows improvements in MAE as the step size is increased. The DFT method calculates the derivative of a function using its Fourier series. The Fourier series requires that the function being differentiated is periodic, and the DFT method approximates this periodicity by assuming that the input function repeats itself outside the range of the input data. As the step size increases, the number of sample points used to compute the Fourier series decreases. This means that the assumption of periodicity becomes more and more accurate as the step size increases, resulting in a better approximation of the derivative. However, there is a trade-off between accuracy and computational cost. As the step size increases, the number of sample points decreases, but the size of the Fourier transform increases. This can lead to longer computation times and may not be practical for very large datasets. The top-right figure shows the computed modified wavenumber for the explicit and implicit method theoretical results. We can clearly see that the implicit method is much more accurate for large wavenumbers. For the DFT we treat the modified wave number exactly as the wavenumber by construction. The walltime for the methods are shown in the lower-left plot. Here, both N and h are varied. The DFT method involves a lot of complex arithmetic operations and requires the computation of the Fast Fourier Transform (FFT) for each row and column of the input array. The FFT algorithm has a computational complexity of O(N log N), where N is the number of elements in the array. Therefore, as the size of the input array increases, the computation time for the DFT method also increases significantly. On the other hand, the explicit and implicit methods only involve simple arithmetic operations such as addition, subtraction, multiplication, and division. The computation time for these methods depends on the number of elements in the input array, but the increase is not as significant as in the case of the DFT method. Additionally, the explicit and implicit methods do not require the computation of the FFT, which is a time-consuming operation.

The bottom right plot shows the ratio of the walltimes for all the methods. The implicit method is much slower than the explicit method due, primarily, to the cost of solving a tridiagonal system of equations. Figure 7 compares the efficiency of the two methods by examining the variation of the average error with the walltime. The question to consider is, which method is faster for a given error. We see a clear trend that for a given error there is only one possible method to use since they do not overlap.

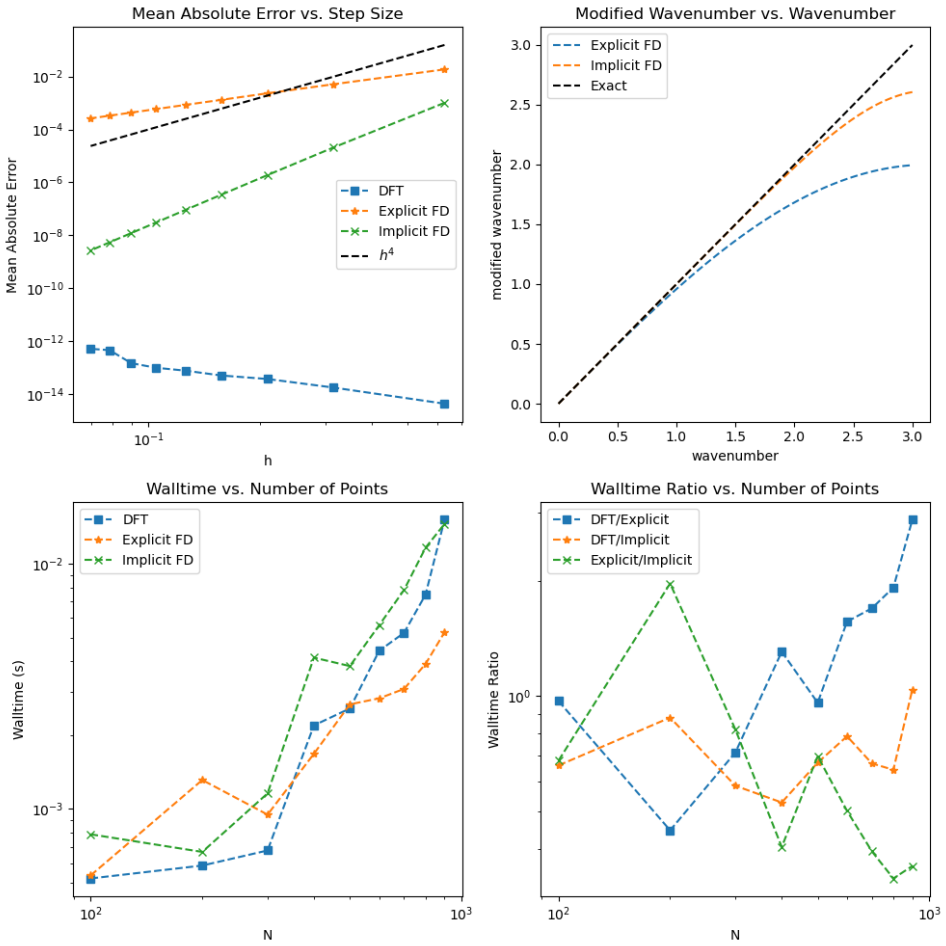

Figure 6: Top left: MAE; top right: modified wavenumber; lower left: walltime; lower right: ratio of walltimes

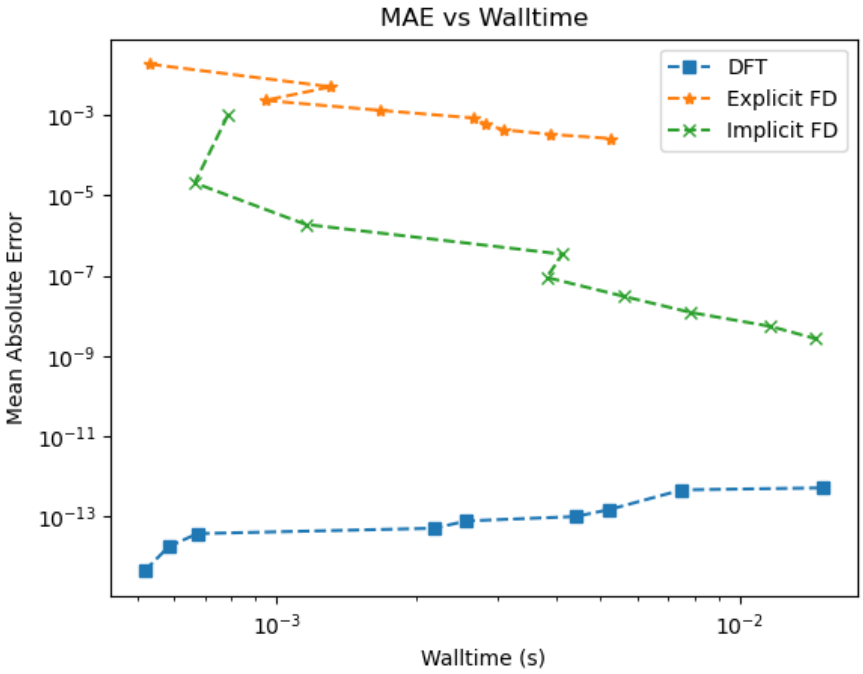

Figure 7: MAE vs walltime