Scientific Computation Project 3

Andreea Patra — CID 01365348

March 27, 2020

Part 1

1.1)

For this part, we have been given sea-surface data for one instance in time. However, as the data was being processed, it became corrupted and a value of -1000 had been placed at all locations where the data was lost. We will use the recommender system approach to try and recover the lost data. The main approach is to fill in the blanks without introducing new trends or creating new information. We will do this by minimising the rank using an approach based on the SVD. Therefore, we define the approximate matrix $\tilde{R} = AB$ and the problem becomes an optimisation one. We construct A and B by minimizing the following cost

$$c = \sum \sum_{i,j \in K} (R_{ij} - \tilde{R}_{ij})^2 + \lambda \Big[\sum_{i=1}^{q} \sum_{j=1}^{p} (A_{ij})^2 + \sum_{i=1}^{p} \sum_{j=1}^{q} (B_{ij})^2 \Big]$$

The magnitudes of A and B are constrained using a l2-regularization term. The cost is minimised using an iterative approach by setting the gradient with respect to each matrix element to zero. Therefore, we get the following equations for updating A and B.

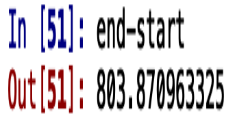
$$\frac{\partial c}{\partial A_{mn}} = -2 \sum \sum_{i,j \in K} (R_{ij} - \tilde{R}_{ij}) \frac{\partial \tilde{R}_{ij}}{\partial A_{mn}} + 2\lambda A_{mn} = 0 \tag{1}$$

$$A_{mn}\Big(\lambda + \sum_{j,(m,j) \in K} (B_{nj})^2\Big) = \sum_{j,(m,j) \in K} \Big(R_{mj} - \sum_{k \neq n} A_{mk} B_{kj}\Big) B_{nj} \tag{2}$$

$$\frac{\partial c}{\partial B_{mn}} = -2 \sum \sum_{i,j \in K} (R_{ij} - \tilde{R}_{ij}) \frac{\partial \tilde{R}_{ij}}{\partial B_{mn}} + 2\lambda B_{mn} = 0 \tag{3}$$

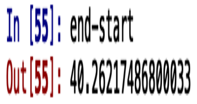
$$B_{mn}\Big(\lambda + \sum_{i,(i,n) \in K} (A_{im})^2\Big) = \sum_{i,(i,n) \in K} \Big(R_{in} - \sum_{k \neq m} A_{ik} B_{kn}\Big) A_{im} \tag{4}$$

Repair1 is the initial function which calculates A and B. For each iteration, we loop through each element of A and B in a random way and we update them with respect to the equations 2) and 4). With each iteration we make sure to update the elements in a stochastic way so that we would not induce overfiting and introduce new information. The if statements checks which matrix to update, more specifically if $n < p$, we update A and if $m < p$, we update B. That is because, when developing the formulas, we assume that $n < p$ for A and that $m < p$ for B. After each 10 iterations, we print the resulting error and as expected, the error decreases. Repair1 has initially a for loop which depends on the number of missing values. The operations inside this loop have complexity O(1). The inefficiency lies in the next for loop. We set N as the number of iterations and (A,B) as the shape of the matrix R. Inside the second for loop we have 2 if statements. Inside each if statement, we have a for loop with 2 O(1) operations and a for loop for each iteration. Therefore, the time complexity for one if statement depends on p and the number of columns,resp. rows, with valid data. In order to make it more efficient, we will try and vectorise it so that we can delete the for loops. Therefore, we can realise that $\sum_{j,(m,j)\in K}(B_{nj})^2$ and $\sum_{i,(i,n)\in K}(A_{im})^2$ are norms of vectors of A and B. As well as, $\sum_{k\neq m} A_{ik}B_{kn}$ and $\sum_{k\neq n} A_{mk}B_{kj}$ is the matrix multiplication. By doing this, we were able to remove the for loops that were inside the if statements.



```
In [51]: end-start
Out[51]: 803.870963325
```

Figure 1



```
In [55]: end-start
Out[55]: 40.26217486800033
```

Figure 2

We can see that the time to run the algorithm for 10 iterations have reduced from approx 803 seconds to approx 40 seconds (Figure 1 and 2).

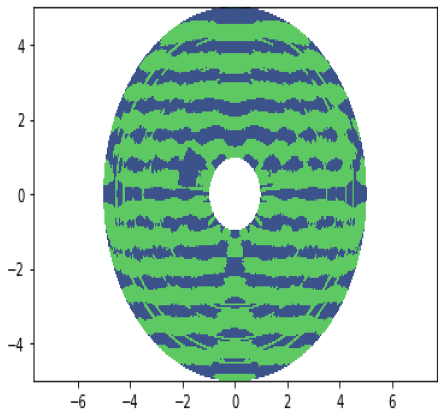


Figure 3

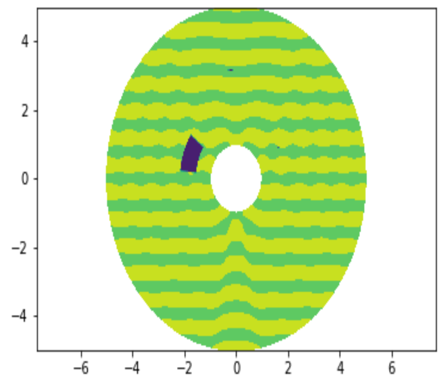


Figure 4: Initial image

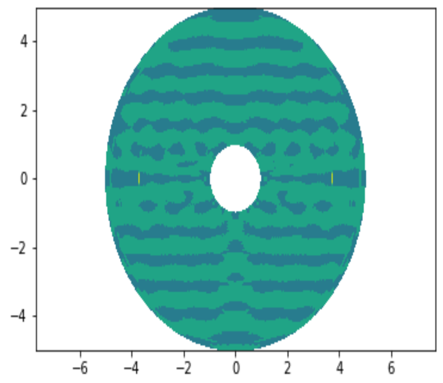


Figure 5: p=10 and λ=2

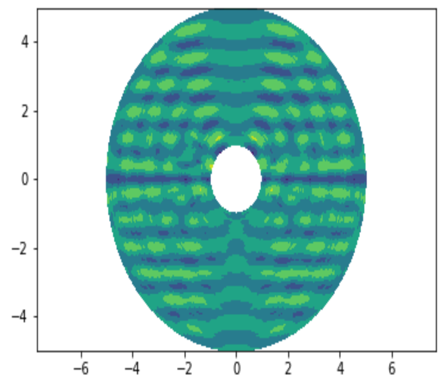


Figure 6: p=15 and λ=2

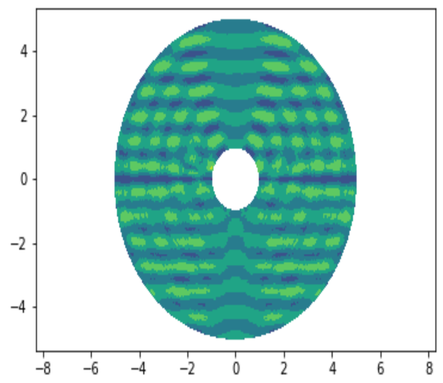


Figure 7: p=20 and λ=2

In figure 4, we can see the initial image of our data. In the next 3 figures, I have set p=10,15,20 and $\lambda = 2$ for 150 iterations. As we can see, by increasing the value for p we managed to make the image clearer. For these images, the levels for hfield plotting function was set to 5. In figure 3, there is the image of the data for p=20,$\lambda = 2$ and levels=0. Moreover, if we increase p, we can get more yellow in our image which will resemble the pattern from the initial image. One improvement, could be that we could use a l1 regularization. Using a l1 regularization would make the updated terms more closer to 0 as the cost would be redefined as $c = \sum\sum_{i,j\in K}(R_{ij} - \tilde{R}_{ij})^2 + \lambda\Big[\sum_{i=1}^{q}\sum_{j=1}^{p} \mid A_{ij} \mid + \sum_{i=1}^{p}\sum_{j=1}^{q} \mid B_{ij} \mid\Big]$. Having updated terms more close to zero, we can make sure that new information would not be added, but at the same time some information can be lost as well. A thorough investigation of the effect of the values of λ would therefore be needed.

1.2i))

1.2ii)

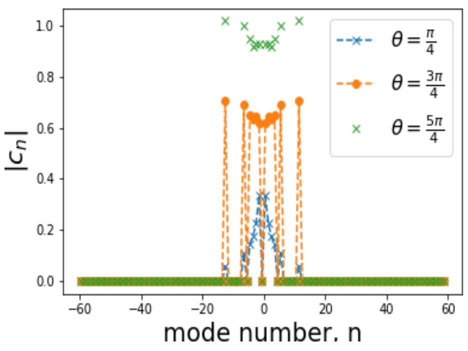


Figure 8

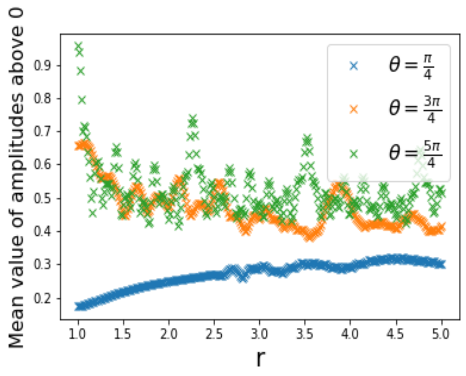


Figure 9

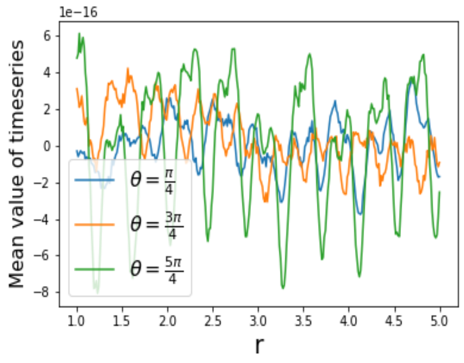


Figure 10

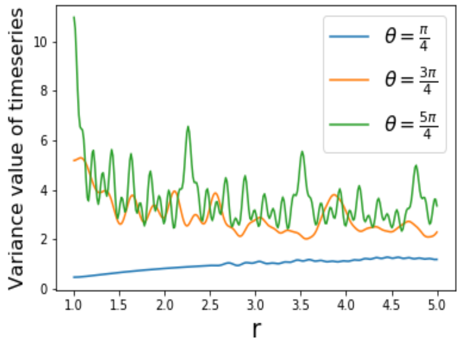


Figure 11

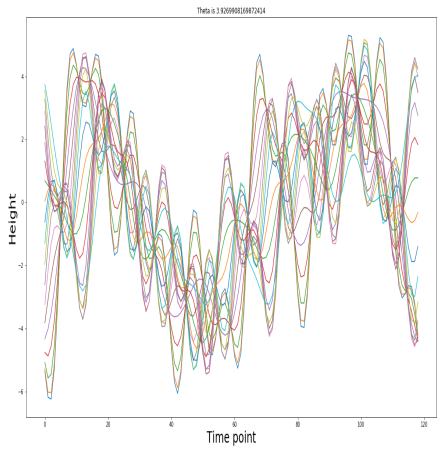


Figure 12: $\theta = \frac{5\pi}{4}$

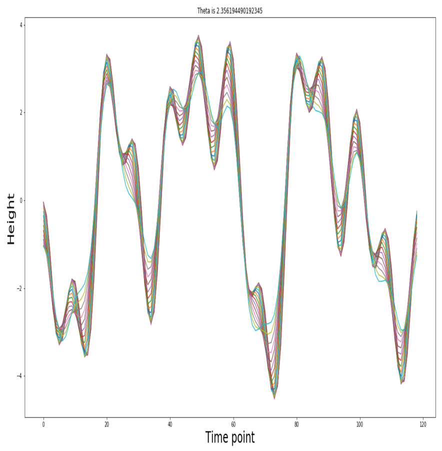


Figure 13: $\theta = \frac{3\pi}{4}$

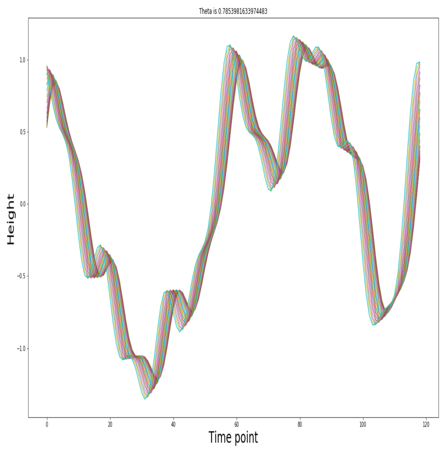


Figure 14: $\theta = \frac{\pi}{4}$

For this part, we will investigate a 3-D numpy array that contains 119 equally spaced points in time. We will investigate the height dynamics with respect to $\theta = \frac{\pi}{4}$, $\theta = \frac{3\pi}{4}$ and $\theta = \frac{5\pi}{4}$. When looking at the index in the theta array for these values, we discovered that these are 38,108 and 180, respectively. Therefore, in our analysis we will look at these columns in 'data3.npy'. We will use a Fourier Analysis approach for our time series. In figure 12,13 and 14 you can see how the values oscillates across time for the first 20 values of r. As we can see, for $\theta = \frac{\pi}{4}$ and $\theta = \frac{3\pi}{4}$ the waves follows the same pattern for different values of r, but for $\theta = \frac{5\pi}{4}$ we have random oscillations. Moreover, for the first 2 values of θ it appears as it is periodic as the end points of the timeseries coincide. When looking at the Fourier coefficients of the time series for r=1 for each value of theta, figure 8, we can see that the signal is a superposition of 6 sine waves with different frequencies and with random amplitudes and phases $Y = \sum_{m=1}^{6} a_m \sin(2\pi f_m t + \phi_m$. For our data, Nt=119 and T=$\frac{20\pi}{80}$. For higher values of θ, we have a higher amplitude for the fourier coefficients. On another level, if we look at all values for r, we can see that we will always have a superposition of 6 sine waves as expected. In figure 9, we have plotted the mean value of $| c_n |> 0.01$ for each value of r. In figure 10 and 11, we have plotted the mean and variance of the time series for each value of r. If we compare figure 11 and figure 9, we can see that the pattern is kept. This checks our conclusion of the superposition of 6 sine waves for the signal.

1.3)

In this part, we will implement PCA algorithm so that we can reduce the memory that the 'data3.npy' requires for storing. The function 'reduce' gives the arrays that can be used to reconstruct the matrix. In order to apply PCA, we will split it in 119 2-D matrices and apply PCA on each one of them. The PCA algorithm relies on the SVD approach. Using the svd function in numpy.linalg, we can get U,Σ and W such that $A = U\Sigma W^T$ and our transformed data is $G = V^T A$. When choosing the number of principal components we want to keep, we set a percentage through the per variable in reduce which denotes how much variance we want to preserve. The shape of the matrix has been preserved. The initial matrix was of the form 300 by 289 by 119 and because the type of float, we can estimate that it needs approx 8 bytes per number. Therefore the estimated memory is 82538400 bytes. Using the PCA method, we can reduce the memory usage. For a percentage of preservations of 30% we would keep 30 principal components and we would reconstruct the matrix only using these 3 arrays. Therefore, the estimate of the memory is $119 \times (300 + 30 + 300) = 73661$.

Part 2

2.1ii)

For this part, we will implement the compact finite difference scheme so that we can compute the second derivative. The function 'newdiff' follows the equations given. Putting them in matrix form, we are able to get the following output for $A \times \mathbf{f}" = B \times \mathbf{f}$ with $a_1 = \frac{c}{9h^2}$, $a_2 = \frac{b}{4h^2}$, $a_3 = \frac{a}{h^2}$ and s=$-2(a_1 + a_2 + a_3)$

$$A = \begin{pmatrix} 1 & 10 & 0 & 0 & \cdots & \cdots & 0 \\ \alpha & 1 & \alpha & 0 & \cdots & \cdots & 0 \\ 0 & \alpha & 1 & \alpha & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & \alpha & 1 & \alpha \\ 0 & 0 & \cdots & \cdots & \cdots & 10 & 1 \end{pmatrix}$$

$$
\mathrm{B}=\begin{pmatrix}
\frac{145}{12h^2} & \frac{-76}{3h^2} & \frac{29}{2h^2} & \frac{-4}{3h^2} & \frac{1}{12h^2} & 0 & \cdots & \cdots & 0 & 0 & 0 \\
a_3 & \mathrm{s} & a_3 & a_2 & a_1 & 0 & \cdots & \cdots & 0 & a_1 & a_2 \\
a_2 & a_3 & \mathrm{s} & a_3 & a_2 & a_1 & 0 & \cdots & 0 & 0 & a_1 \\
a_1 & a_2 & a_3 & \mathrm{s} & a_3 & a_2 & a_1 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
a_1 & 0 & 0 & \cdots & \cdots & a_1 & a_2 & a_3 & \mathrm{s} & a_3 & a_2 \\
a_2 & a_1 & 0 & \cdots & \cdots & 0 & a_1 & a_2 & a_3 & \mathrm{s} & a_3 \\
0 & 0 & 0 & \cdots & a_1 & a_2 & a_3 & \mathrm{s} & a_3 & a_2 & a_1 \\
0 & 0 & 0 & \cdots & \cdots & 0 & \frac{1}{12h^2} & \frac{-4}{3h^2} & \frac{29}{2h^2} & \frac{-76}{3h^2} & \frac{145}{12h^2}
\end{pmatrix}
$$

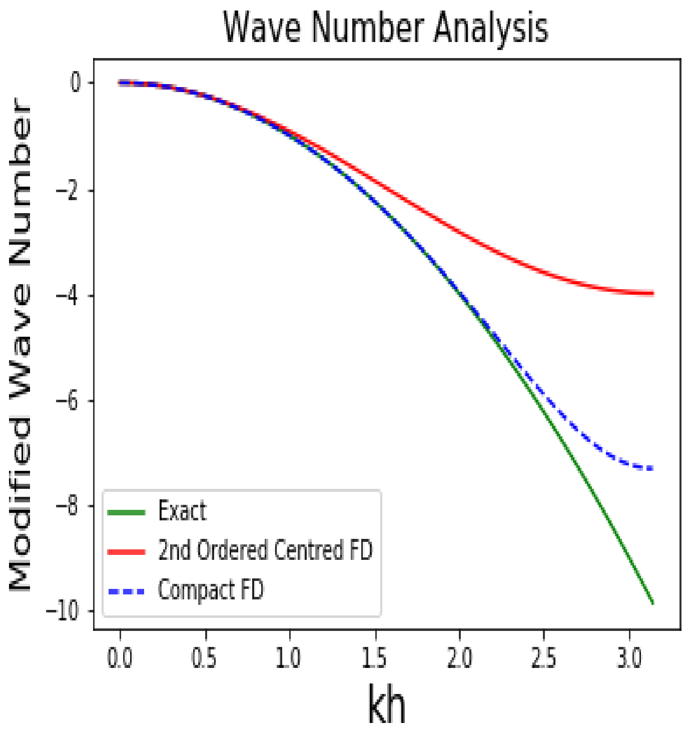


Figure 15

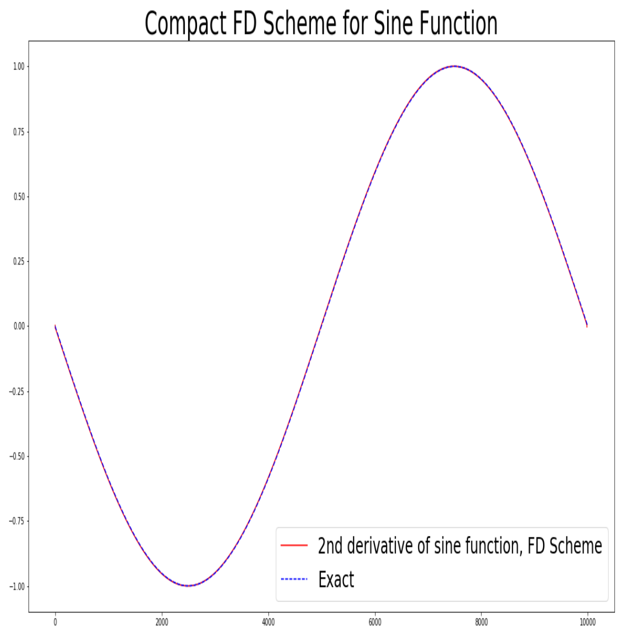


Figure 16

The compact difference scheme has a few advantages, such as the possibility of using built in efficient numpy functions as well as sparse matrices. Using scipy.sparse.diags function, we can easily construct sparse matrices based on the diagonals. Then using scipy.sparse.linalg.spsolve, we can solve the system of equations and find the second derivative. This will reduce the memory by using sparse matrices. In order to critically assess the accuracy of the algorithm, we are going to initially test it on $\sin(x)$ for x in $(0, 2\pi)$. An import thing to note is the choice of the gridspacing. In order to have an accurate approximation, we would need an uniform space between mesh points, therefore, $h = \frac{x_N - X0}{N}$. In figure 16, we can see that the compact finite difference scheme offers a good approximation of the second derivative when we have removed the first and last 8 elements. Therefore, we can see that this method has higher error at the boundaries.

We are going to compare the compact finite difference with the 2^{nd} order centered difference scheme. The form of this one is $f" = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}$. We use the wave number analysis to understand how accurate are the methods with respect to the wave number. For compact finite difference, we compare 'modified wave number', $-(kh)^2$ to $\frac{\frac{2c}{9}(\cos(3kh)-1)+\frac{b}{2}(\cos(2kh)-1)+2a(\cos(kh)-1)}{1+2\alpha\cos(kh)}$. This was constructed knowing that any periodic function can be decomposed into their Fourier components which are in the form e^{ikx} where k is the wave number. The exact second derivative of the components is $-k^2 e^{ikx}$. By applying the formula from the compact finite difference for $0 < i < N-1$, we get that $-k^2 e^{ikx}(2\alpha\cos(kh)+1) = \left(\frac{2c}{9}(\cos(3kh)-1) + \frac{b}{2}(\cos(2kh)-1) + 2a(\cos(kh)-1)\right)\frac{1}{h^2}e^{ikx}$ and therefore our reasons for analysis. Our modified wave number is $\frac{\frac{2c}{9}(\cos(3kh)-1)+\frac{b}{2}(\cos(2kh)-1)+2a(\cos(kh)-1)}{(1+2\alpha\cos(kh))h^2}$. By doing a series expansion of the modified wave number, we get that it is a second order approximation as well. For second order centered difference, we compare 'modified wave number', $-(kh)^2$ to $2\cos(kh)$. By applying the formula for second order centred scheme, we get that $f^{"} = \frac{2e^{ikx}}{h^2}(cos(kh)-1)$ with modified wave number $\frac{2}{h^2}(cos(kh)-1)$ In figure 15, we can see that for the compact finite difference there is 1% error when $kh \simeq 1.8$ and as $kh = \frac{2\pi h}{\lambda}$, we would need approx 3.5 points per wavelength and for second order there is 1% error when $kh \simeq 0.4$ and we would need approx 15 points per wavelength. We can choose the centered scheme when we have a small wave number as we can ensure an accurate approximation, but when the wave number is big we need to switch to compact scheme and sacrifice time for accuracy. The mean error for the compact scheme when analysed with respect to sine function is approx 0.33. As it can be seen, the compact scheme is more accurate than the second order scheme, but it requires more operations. The second order requires roughly N multiplications and 3N additions per point and and the compact scheme has 7N additions and 8N multplications, both having a linear time complexity O(N). However, considering that the compact method is not behaving well at the end points, then it would mean that it would not work well for PDEs with strict boundary conditions.

2.2)

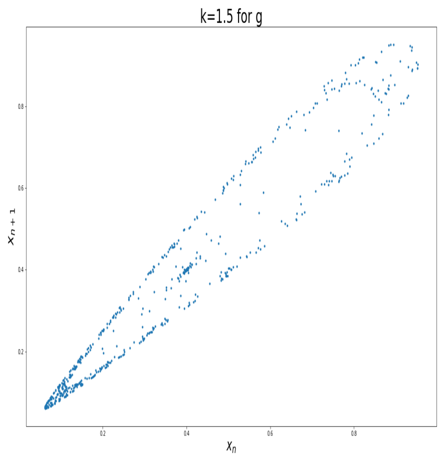


Figure 17

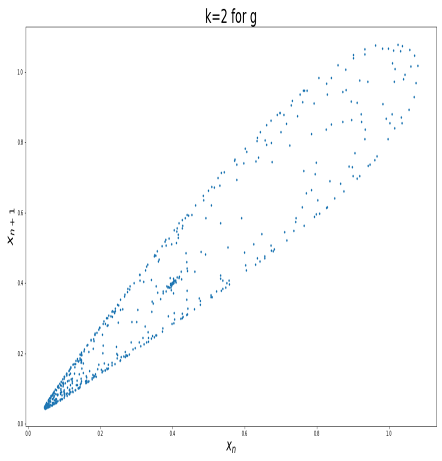


Figure 18

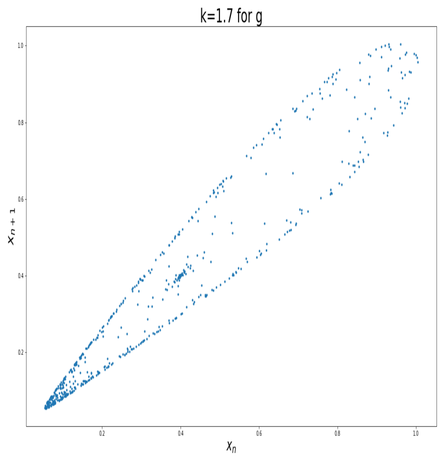


Figure 19

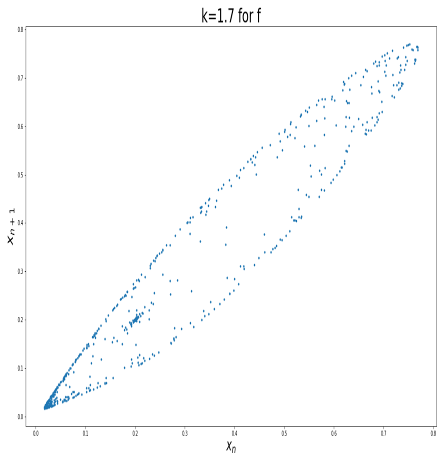


Figure 20

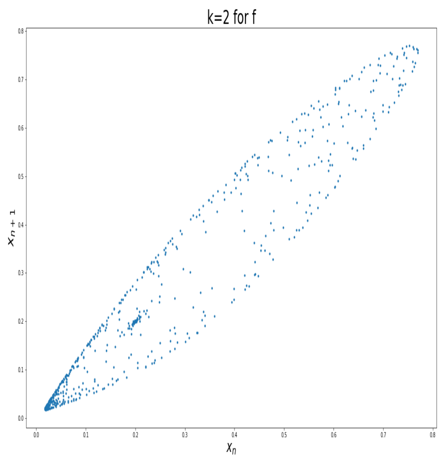


Figure 21

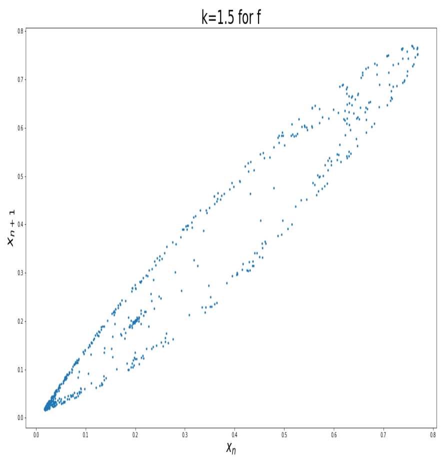


Figure 22

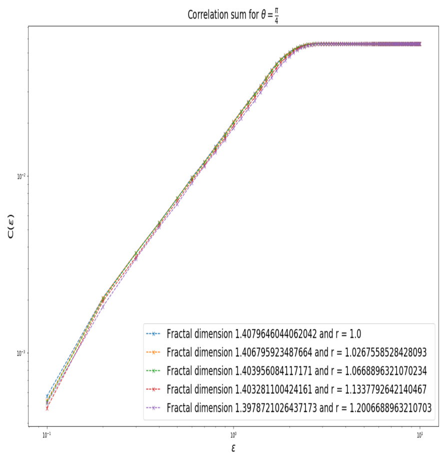


Figure 23

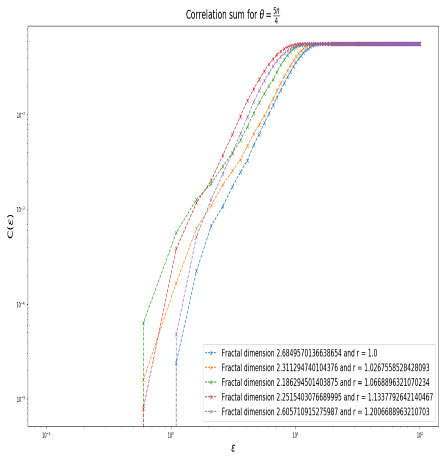


Figure 24

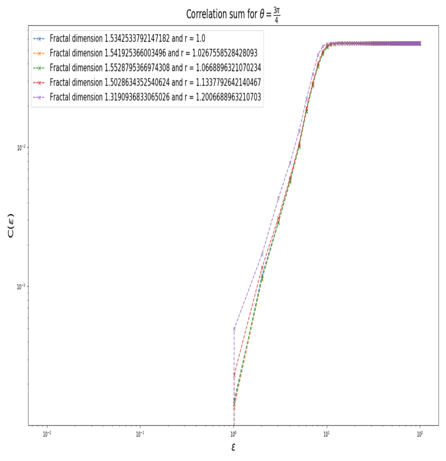


Figure 25

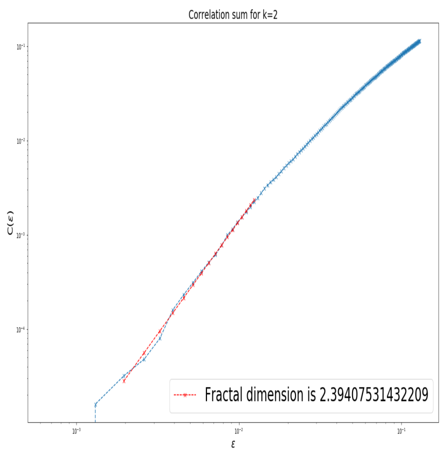


Figure 26

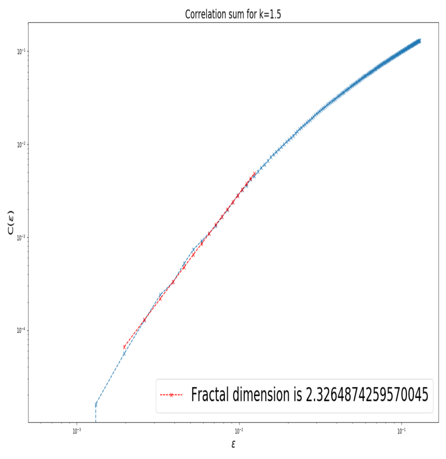


Figure 27

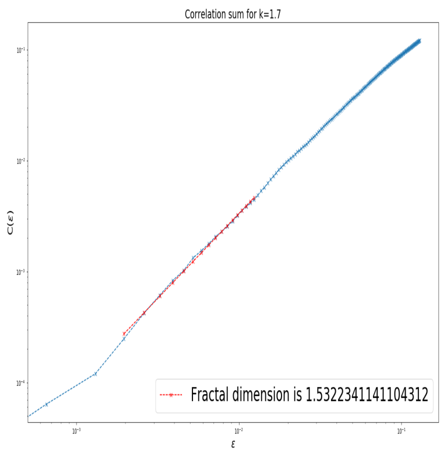


Figure 28

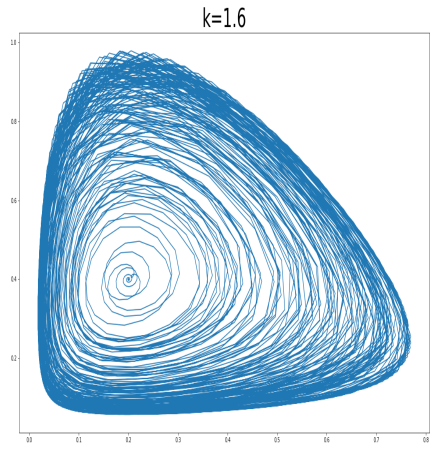


Figure 29: Phase space

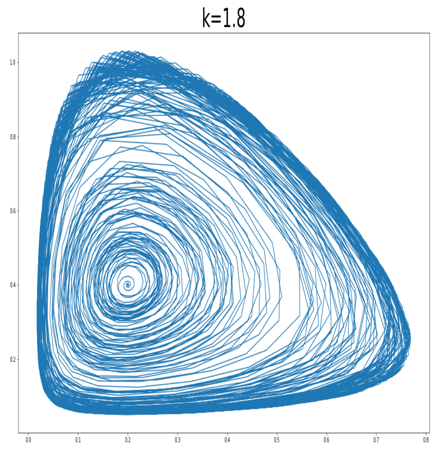


Figure 30: Phase space

For this part, we are going to set $\phi = 0.3$, $L = 1024$, $Nx = 1024$ and $\frac{\mu}{k} = 0.4$ with k in range $1.5 \leq k \leq 2$. First of all, for values of 1.5, 1.7 and 2, we will investigate to what degree the dynamics correspond to chaos. To determine the chaotic behaviour, we are going to calculate the fractal dimension using the correlation sum. The correlation sum is defined as $C(\epsilon) = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} H(\epsilon - \|x_i - x_j\|)$. We use the pdist function in the scipy package to calculate the pairwise distances. Then, the fractal dimension will be the slope when plotting $\log(\epsilon)$ against $\log(C(\epsilon))$. Specifically, when looking at the 100th component of the f and g function along the x axis, we get a fractal dimension higher than 2 for k=1.5 and k=2 which suggests chaotic behaviour(figures 26 and 27). When plotting x_n against x_{n+1} for each f and g, we have seen than it does not suggest a periodic behaviour, thus we were expecting a fractal dimension greater than 2 (Figure 17-22). As well, in figure 23 and 25, we can see that for data3, when $\theta = \frac{\pi}{4}$ and $\theta = \frac{3\pi}{4}$, we have a fractal dimension lower than 2 which will suggest periodic behaviour. This ties back to the analysis of the height dynamics done in part 1.2ii) where we have observed that the time series specific for $\theta = \frac{\pi}{4}$ and $\theta = \frac{3\pi}{4}$ has amplitudes lower for the Fourier coefficients. We can see that the fractal dimension usually decreases as r increases. To calculate the fractal dimension of the data3, we have constructed a m-dimensional vector at t_i using time delays. The value for m is usually chosen to be bigger than the fractal dimension. The value of τ was calculated using the Welch method. Calculating the power spectrum, we were able to find a dominant frequency and set τ as a fifth of this. If we look at the phase space, figures 29 and 30 , we can observe that it looks as a repelling spiral which starts at 0.2 and with time it gets further away from the initial condition. As k increases, we can see that the trajectories become more centered around the convergent point. For this analysis we have increased T to 2000 and Nt to 2402 so that we can encompass more information.

2.3)

In this part, we will consider methods of analysing the change in be the dynamics behaviour when small sinusoidal perturbations are introduced to the steady state. One approach for both finite time span and infinite is to use Fourier transform to analyse the change in the number of sine waves. We can see if the dominant frequency has been affected using the Welch method. We can see if the phase space keeps its shape and if it is converging towards to the same orbit.