Scientific Computation Project 3

Matthew Cowley, CID 1059624

March 27, 2020

Part 1

1.1)

Repair1 was fairly inefficient within the loops of iterations,n and m (which are all necessary). I have vectorised repair1 where possible, using built in numpy operations. Rsum is calculated as a vector of elements of mlist (and nlist for updating a value of B), with Asum (and Bsum) using this with a dot product and summing out of the vector, to update a value of A (and B respectively). I used a Boolean matrix instead of mlist and nlist, to access appropriate values, this produces a significant speed up. Calling random.permutation every loop is quite inefficient, but it is hard too replace, e.g precomputing it for 20 random seeds and iterating randomly over these, caused an increased runtime. So I left this alone.

p dictates the size of the rectangular matrices A and B, which are used to reconstruct the data, generally bigger p leads to higher quality reconstruction, up to a point. Large p requires far more computations and generally takes more iterations to converge to a solution. Since repair2 is effectively a low rank approximation, we can use the SVD of data1, to indicate what the size of p should be (although the true SVD of data1 will have some error due to the missing values, hence why it is an indicator). The singular values are relatively insignificant after 42nd (all of size 10^{-12}), so a value of p=50 should be sufficient for a good approximation of data1.

λ limits the magnitude of values in A and B, for a given p. Larger λ permits the optimum solution to have more error for known values of $\hat{R}$, with the condition that it limits values of A and B, which limits the maximum magnitude of approximated values for missing data. A problem often encountered for low rank approximations, is that approximated values are too large or small, so the penalty parameter reduces this problem (in this case both as the range of the data is roughly symmetric about 0). So larger values of λ can lead to better approximations of missing values, but incur more error in known values. In the case of p=50, $\lambda > 2$, gives poor results, with $\hat{R}$ values being to small (in magnitude). I used $\lambda = 0.5$ as this seems to lead to good results for p=50.

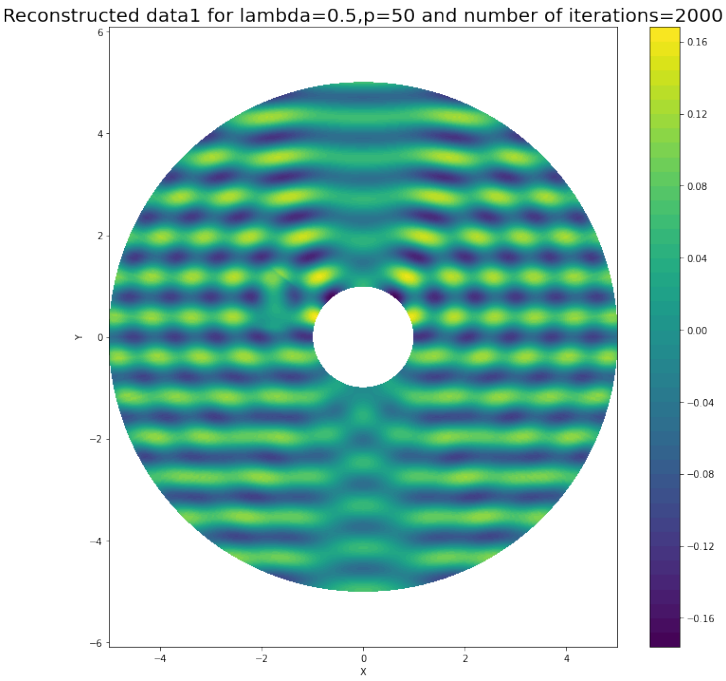

Figure 1: A partially successful reconstruction of data1, A,B converged before the 2000th iteration

For better results you could add more constraints on the cost e.g maybe separate penalty parameters λ_+, λ_- for positive and negative values. You could use a different method to find the optimum cost e.g Stochastic Gradient decent. You could also just replace known values of R in $\hat{R}$, to improve the quality of the final result. From the image it looks as if the data needs smoothing aswell.

1.2.1)

The following shows what form the solution should $h(r, \theta, t)$ should take. The wave equation in radial coordinates takes the form:

$$h_{tt} - (h_{rr} + \frac{h_r}{r} + \frac{h_{\theta\theta}}{r^2}) = 0 \tag{1}$$

with b.c $h(r = 1, \theta, t) = f(\theta, t)$. Now you can assume the solution for h can solved using separation of variables. Assume it can be written in $h = \phi(r, \theta)T(t)$ which wrewritting (1) gives:

$$\frac{T_{tt}}{T} = -\lambda = \frac{\phi_{rr}r^2 + \phi_r r + \phi_{\theta\theta}}{\phi r^2} \tag{2}$$

We assume $\lambda > 0$ to give solutions of T appropriate for problem. Solutions of T are of the form:

$$T(t) = e^{-i\sqrt{\lambda}t} \tag{3}$$

The right side when multiplied out gives:

$$\phi_{rr}r^2 + \phi_r r + \phi_{\theta\theta} + \lambda\phi r^2 = 0 \tag{4}$$

This we can split further letting $\phi(r,\theta) = R(r)\Theta(\theta)$ which gives:

$$\frac{R_{rr}r^2 + R_r r + R + \lambda R r^2}{R} = \mu = \frac{\Theta_{\theta\theta}}{\Theta} \tag{5}$$

now the right side gives:

$$\Theta_{\theta\theta} + \mu\Theta = 0 \tag{6}$$

Θ must be periodic with period 2π, so we get that $\mu = 0, 1, 4, \ldots, n^2, \ldots$ with:

$$\Theta = a_n \sin(n\theta) + b_n \cos(n\theta) \tag{7}$$

Now using a change of variables $x = \sqrt{\lambda}r$ and rearranging the left hand side of (4), the left hand side becomes the Bessel equation:

$$R_{xx}x^2 + R_x x + (x^2 - n^2)R = 0 \tag{8}$$

Which is satisfyed by the Hankel functions $H_n^{(1)}(x)$. So solutions are of the following form:

$$h(r,\theta,t) = \sum_{n=0}^{\infty}\sum_{m=1}^{\infty} H_n^{(1)}(\sqrt{\lambda_{nm}}r)e^{-i\sqrt{\lambda_{nm}}t}(a_{nm}\sin(n\theta) + b_{nm}\cos(n\theta)) \tag{9}$$

My method was not successfully implemented but it involved the following, from looking at size of 2d Discrete Fourier series coefficients for $r = 1$ and you identify only significant coefficients, retaining the corresponding frequency's index's m and n. Along with frequency's, here they are $\omega_m = \sqrt{\lambda_{nm}}$ and n. You discarding large n as will result in NaNs in the Hankel function although will reduce quality of results. Outwave should then return the sum all over n,m and scipy's Hankel, with Θ and T being specified (the same as t and theta in data2):

$$h(r_0,\theta,t) = \sum_{n=0}^{N}\sum_{m=1}^{M} H_n^{(1)}(\omega_m r_0)c_{nm}\langle e^{-i\omega_m T}, e^{-in\Theta}\rangle_{\text{outer}} \tag{10}$$

The coefficients c_{nm} might need adjustment to fit initial data better. Solutions of the 2nd kind might need to be consider, as well at the effect of imaginary values in the above sum. To remedy this you could potentially just use Bessel functions(in scipy.special as well), instead of the Hankel functions of the first kind.

1.2.2)

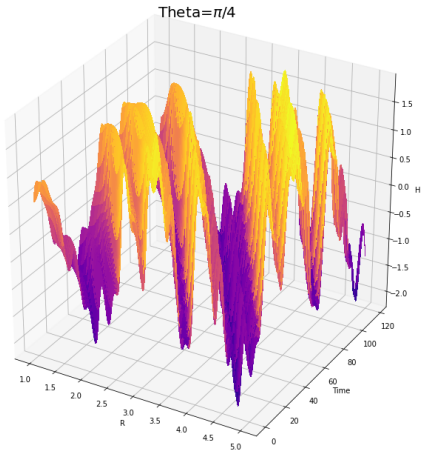

Figure 2: 3D contour plot of h(r,θ) for fixed $\theta = \pi/4$

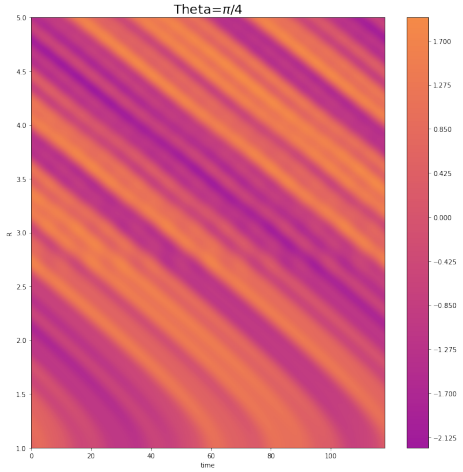

Figure 3: contour plot of h(r,θ) for fixed $\theta = \pi/4$

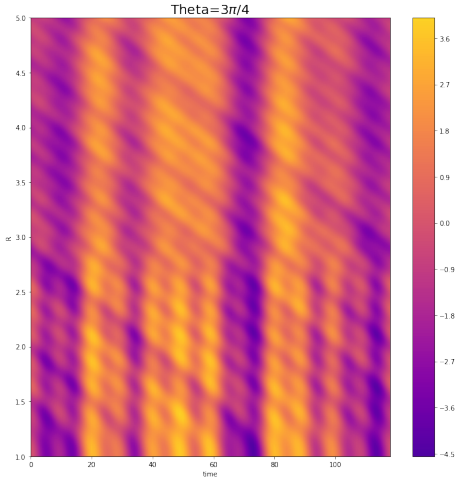

Figure 4: contour plot of h(r,θ) for fixed $\theta = 3\pi/4$

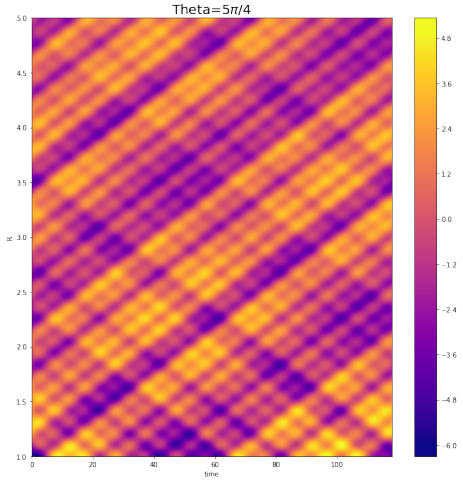

Figure 5: contour plot of h(r,θ) for fixed $\theta = 5\pi/4$

Figure 2 although pretty, is hard to interpret and compare with other θ values, But there is clearly a wavelike structure. However figures 3,4,5 are very informative. In these you can clearly see 2 waves oscillating at 2 different angles for different θ,each with different amplitudes and frequency's. For example in $\theta = \pi/4$ waves seem at the same angle, where as for $\theta = 5\pi/4$ they are perpendicular. Where as for $\theta = 3\pi/4$, it is somewhere in between, maybe $\pi/4$.

For large r in $\theta = 3\pi/4$ they seem to be less well defined (the blurryness), suggesting the effect of another wave or amplitude dying out. Likewise for small r on $\theta = \pi/4$ the waves are not distinct, which could be the effect of the wave hitting the island and bouncing back suggesting wave interference.

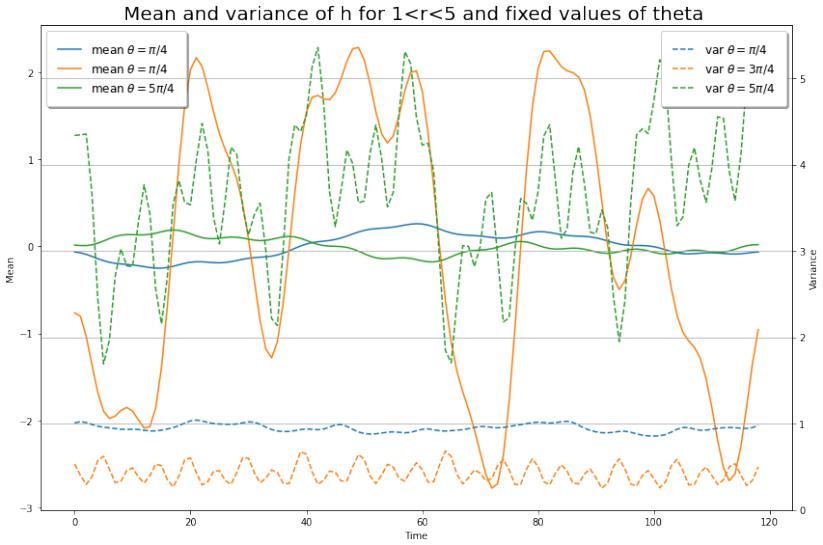

Figure 6: How mean and variance of h(t) change for time, for fixed theta

Interesting the fixed values of θ all produce different results, the mean is fairly constant producing wavelike structure(sinusoidal), apart from $\theta = 3\pi/4$ which seems very sporadic but has a constant oscillating variance. Wheres as $\theta = 3\pi/4$ and $\theta = \pi/4$ have nice wave like mean, but different variances.

On the next page is figure 7, which shows the same results but with respect to radius, it clearly shows overall sinusoidal behavior for all theta values, but they differ in phase and amplitude, but similar frequency. The variance is different for all of them and the mean is not smooth, due to the effects of waves of much smaller amplitude and higher frequency.

To summarise for different fixed θ you get different results of how 2 wave sinusoidal waves interact at different amplitudes, phases and angles, this can be linked to the form of the solution in (10).

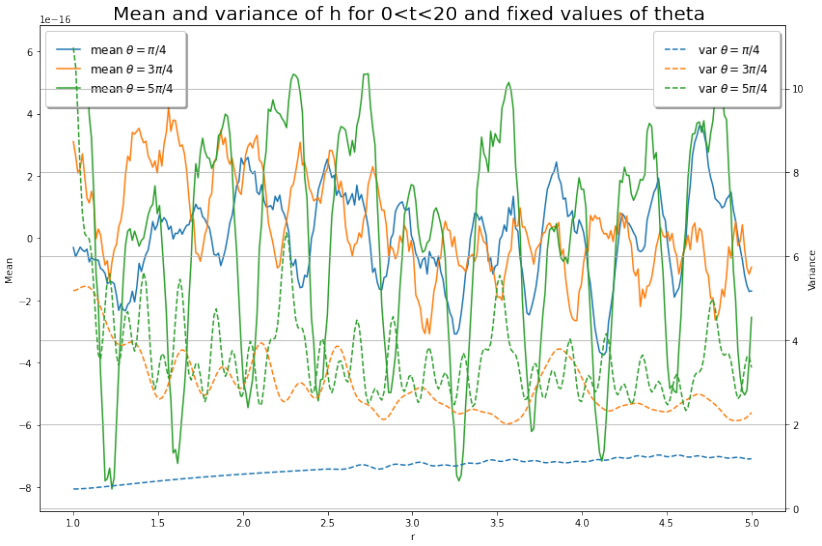

Figure 7: How mean and variance of h(t) change for r, for fixed theta

1.3)

My reduce works by splitting the 3D array into several 2D arrays (you can choose whether to split with time or r), it then uses SVD on these on these 2D arrays $A_{n\times p}$ to decompose them into $U_{n\times n}, \Sigma^{\text{diag}}_{n\times p}, V_{p\times p}$ (in reality Σ is 1d array of length min(n,p), techniquely it is V^T). It selects the biggest k singular values of Σ, where k is specified by the user in inputs, and corresponding k columns/rows of U and V. You could send these arrays as 3D arrays (extra dimension being the split direction) e.g $A_{n\times p\times j} = U_{n\times k\times j}, \Sigma_{k\times j}, V^T_{k\times p\times j}$. You can then reconstructs $A_{n\times p}$ them using outer products:

$$A_{npl} = \sum_{i=1}^{k} \langle U_{nil}, V_{ipl} \rangle_{\text{outer}} \Sigma_{il}, \qquad \forall l \in \{1, 2, ..., j\} \tag{11}$$

, which creates a rank k approximation of each 2D array. In my reduce I then break down U and V further, using SVD as above but Slicing in the direction of k. I do this for a specified k_u and k_v, reduce sends these arrays: $U^u_{nk_uk}, \Sigma^u_{k_uk}, V^u_{k_ujk}, \Sigma_{kj}, U^V_{pk_vk}, \Sigma^u_{k_vk}, V^V_{k_vjk}$. Reconstruct work by using these arrays to first construct U and V above, which are then used with Σ_{kj} to reconstruct original data.

Another method which could be used is to reshape the 3D data into a large 2D and then perform SVD, however the data in this case is too large and my kernel crashed every-time I attempted this, also using SVD on very large matrix's becomes very computationally expensive, e.g will take roughly $O(4n^2p + 22p^3)$ in time, for an algorithm called R-SVD. So it does not make sense/not possible to perform SVD on all the data at once. However you get different performances depending on how you split the 3D array and what order i.e split it with respect to time or to radius (r) or theta (θ)? theta and radius give pretty much identical results, so I have just shown r, the graph summarises the results, after the first split:

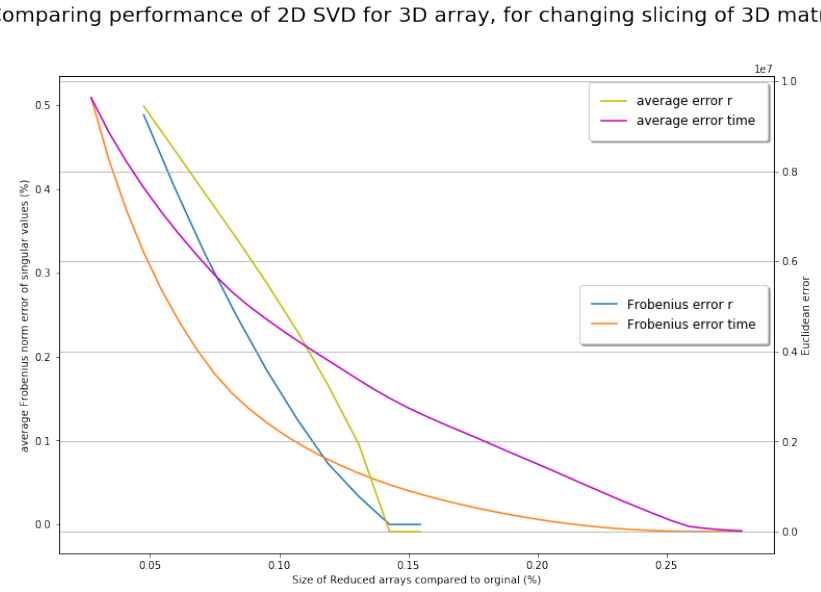

Figure 8: How Low rank approximations perform for different splittings of 3D matrix, The euclidean error is the euclidean distance between the reconstructed and original data, like wise the Frobenius error is calculated as the average of the Frobenius norms of reconstructed arrays over the original arrays

As you can see splitting with time gives more gradual results, and if you want to save more than 90% memory it gives the better results. But if you want to retain all features, you do this with splitting with r and using k=12, you save 86% memory and have a Euclidean error of 10^{-12}. So in my reduce I split with with r first using k=12, I then split the V (slicing across r) using $k_v = 12$, giving similarly good results (virtually no error, with singular values not used being 10^{-12}). Then I split U with time, which gives similar performances too what is shown in the graph, if you want virtually no error you use $k_u = 80$, but you can use $k_u = 40$ and still capture most key features. You alter this k_u to alter the number of key features captured and how much memory is saved.

If your original data is dimension mnj, then splitting it for example in r, then splitting it's U and V, with k, k_u, k_v being the number of singular value entries you use respectively, requires the following number of floating points:

$$k((m+n+1)k_u + (m+j+1)k_v + m)$$

So with data3 this gives memory savings of:

$$\frac{k(590k_u + 420k_v + 300)}{300 \times 289 \times 119}$$

I recommend that $k, k_v = 12$ (k any lower and there is drastic loss in capturing key features and no higher is necessary), with k_u being varied between 10 and 80, (greater than 40 is recommended), the table below shows some k values and their memory savings.

k-values	Memory savings	Quality of Reconstructed array
$k, k_v, k_u = 12, 12, 80$	93.9%	captures all features
$k, k_v, k_u = 12, 12, 40$	96.7%	captures key features
$k, k_v, k_u = 12, 12, 20$	98.0%	captures very basic features
$k, k_v, k_u = 10, 10, 10$	99.0%	captures very little, with lots of error (more than 80% Frobenius error of decomp of U)

Part 2

2.1.ii)

The 2nd order centered difference scheme is very efficient it just takes 3N add(itions), plus 2N mult(iplications), giving it 5N operations overall (per time step). The compact scheme is far less efficient, after doing several precomputations and treating the efficeny at the edges as the same as the center, constructing the R.H.S ($f_i's$) requires 7N add and 4N mult. Then solving the resulting tridiagonal system with scipy.linalg.diags is of O(N), it uses the gbsv algorithm from lapack, which I was unable to find the complexity of. You could also solve the system using the Thomas algorithm. But $11N + O(N)$ compared to 5N is quite substantial e.g when tested on a periodic f, for large N and Nt it was 3 times slower (when implemented in the microbes function without precomputations it produced similar slow down).

The compact scheme also has a truncation error of $O(h^2)$, so is likewise a second order scheme. The graph below shows the error of a known periodic function $f(x) = (3\cos(8\pi x) - 2\sin(20\pi x))/200$.

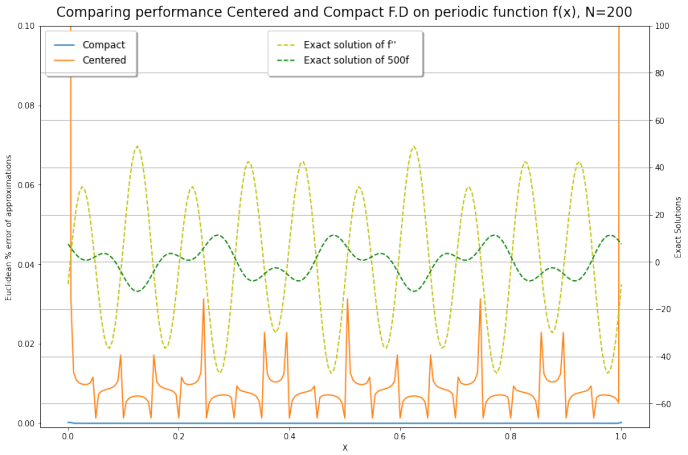

Figure 9: Comparison of centered and compact 2nd order F.D approximations.

It appears the compact scheme in this case is outperforming centered scheme, for this fairly oscillatory function. To investigate this, I have done some wavenumber analysis, which assumes solutions are of form $f(x) = e^{ikx}$, leading to the modified wave numbers for centered (12) and compact (13) schemes, which produce the graphs on the next page.

$$(kh)^2 e^{ikx} = 2(1 - \cos(kh))e^{ikx} \tag{12}$$

$$(kh)^2 e^{ikx} = \frac{2(a(1 - \cos(kh)) + b(1 - \cos(2kh)) + c(1 - \cos(3kh)))}{1 + 2\alpha\cos(kh)} e^{ikx} \tag{13}$$

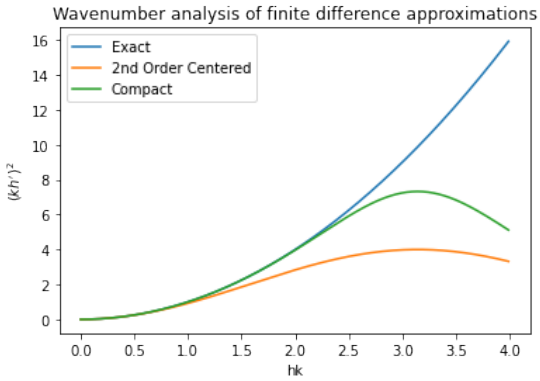

Figure 10: Modified Wave-number plot

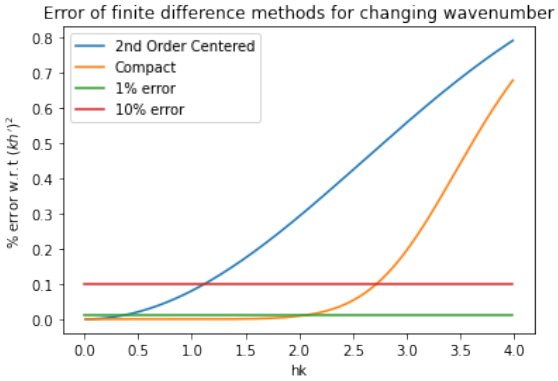

Figure 11: % Error w.r.t exact solution

The compact scheme appears to be performing far better than the centered in figure 5, for high wave number problems. High wave number problems are ones where the solution is highly oscillatory. The approximation of 1% error occurs at $kh \approx 0.36$ and $kh \approx 2.04$. From this you can calculate that you need $\approx$17 points per wavelength for the compact scheme and $\approx$3 for the centered scheme. So if you are using the centered scheme you have to use 5.5 times the amount of points you would need for a compact scheme, to achieve the same accuracy if the solutions of the problem are highly oscillatory.

So if increasing the number of grid point makes the overall all solver far more expensive, and the solutions are highly oscillatory(high wave number), the compact scheme is preferred. If the system has low wave number solutions then the centered scheme is preferred. So although the centered scheme is not as efficient it is more accurate for higher wave number problems.

To summarise centered difference is far more efficient, but not as accurate for high wave number problems.

2.2)

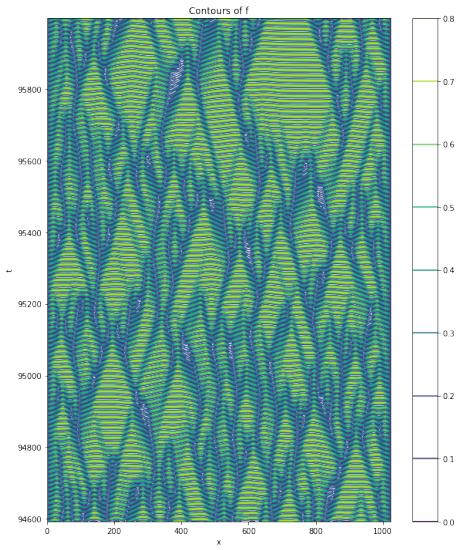

Figure 12: contour plot for ϕ=0.3,κ = 1.5,$\mu/\kappa = 0.4$,timestep=2

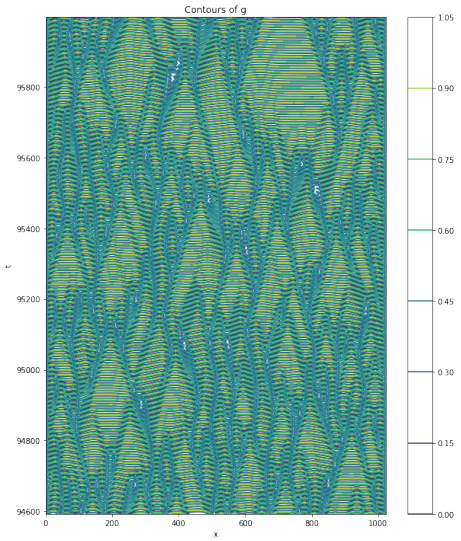

Figure 13: contour plot for ϕ=0.3,κ = 1.5,$\mu/\kappa = 0.4$,timestep=2

$\kappa = 1.7, 2.0$ give similar results to the above. In the coupled system when ever f increases, this causes g to increases at rate proportional to κfg (but limited by ϕ), at the expense of f. f will always stay between 0 and 1, as result of f(1-f) term. g also decays at a rate μ. Both f and g diffusive with time. The resulting system produce with the correct initial conditions results in a system where f and g, where g is dependent on f for survival at the expense of f. This means f is constantly increasing, but the g starts increasing and f is quickly reduced, this process is are represented by the horizontal lines of both contours. When looking into the dynamics of the system we only have to look at f or g, as the behaviour of one implies the other.

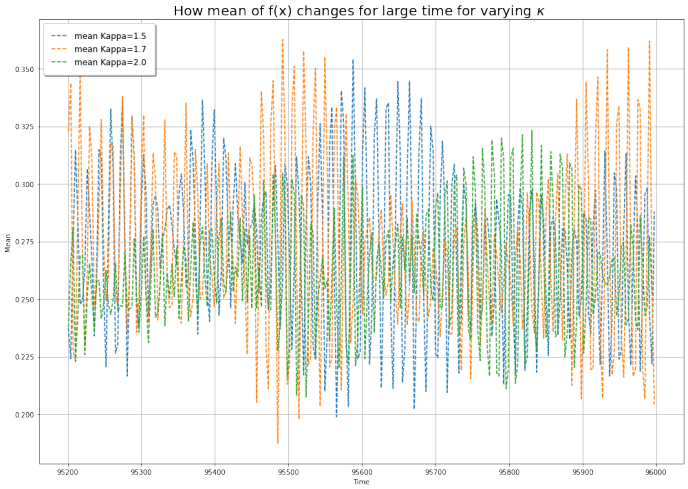

Figure 14: ϕ=0.3,$\mu/\kappa = 0.4$,timesteps ≈ 0.75

From this it can be seen the oscillatory behavior of f, with the spreading of f due to diffusion represented by the changing amplitude. The lack of uniformity suggests chaotic like behavior. With perhaps $\kappa = 1.7$ give the most chaotic results, as it has the most changing structure.

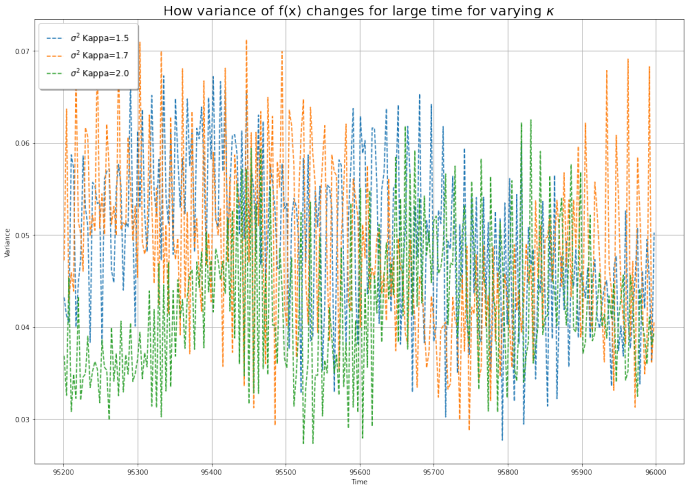

Figure 15: ϕ=0.3,$\mu/\kappa = 0.4$,timesteps ≈ 0.75

Similar to the mean the highly oscillatory behavior is present, with also the low wave number oscillations of changing amplitude. All kappa values are suggesting chaotic behavior, with the variance constantly changing with no clear structure. $\kappa = 1.7$ appears to have the most varied structure, so most chaotic.

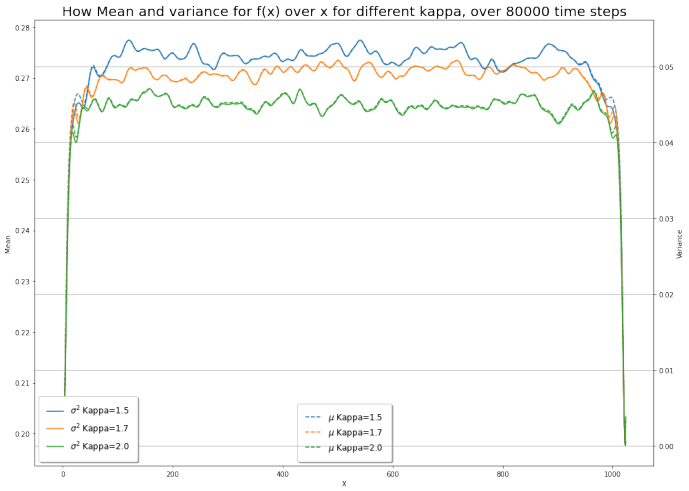

Figure 16: ϕ=0.3,$\mu/\kappa = 0.4$,timesteps ≈ 0.75

This graph didn't begin to stabilise until over 20,000 time steps were used, suggesting that the behavior is fairly chaotic. It is surprising to see that the shape of variance match's the mean, suggesting high values have greater rate of change which makes sense in the context of the problem. It is interesting to see than high kappa leads to higher mean and variance with respect to time. At the edges the solutions stable, but the not very smooth random nature of inner mean and variance suggest that the systems is chaotic for all values.

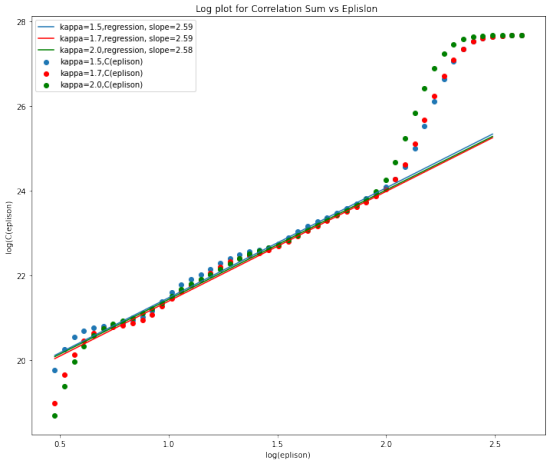

Figure 17: for f with ϕ=0.3,$\mu/\kappa = 0.4$, over 2000 steps in time, for large t

The following graphs used the scaled correlated sum to asses how chaotic the system is. In the microbes case m is the number of grid points. The first graph assumes the a period of the system is around 10,000, where as the second graph increases this to 100,000, with no change in results. Fitting the regression to the graph was difficult due to the unclear trend of the graph, perhaps as result of the wrong m and n. However the figure 17 indicates the system, is in fact very chaotic for all values of kappa, as the slope is greater than 2 (2.58). The larger time frame results again agree with this, but suggest that kappa=2 is ever slight more chaotic, but this a negligible result.

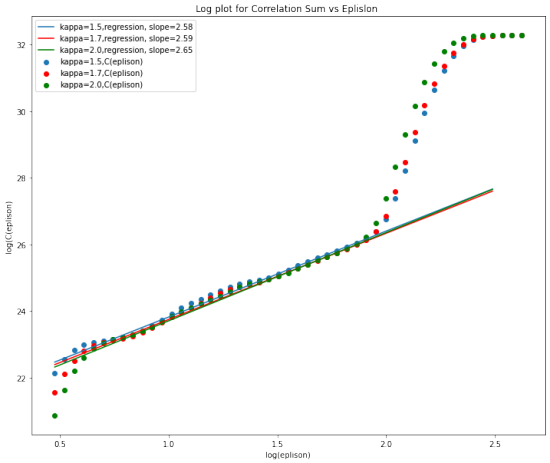

Figure 18: for f with ϕ=0.3,$\mu/\kappa = 0.4$, over 20000 steps in time, for large t

From part 1 it does not seem as if the data is very chaotic, if it would have been hard to reduce so much using SVD and it's singular values being quite low would suggest otherwise. Also the fact that clear patterns can be seen contour plots, but the randomness of some mean and variance plots, do suggest that it could be chaotic. The graph is for a fixed theta, but my initial method flattened the non time components, however calculating pdist on that size of matrix caused my computer to crash. However even for a singular value theta value, it does not seem as if the data is very chaotic for a value of 1.31 (less than 2). Further investigation is required.

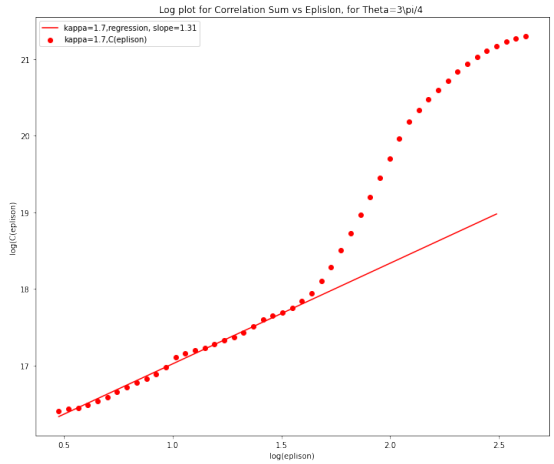

Figure 19: data 3 for fixed theta, ignore fist 10 time components of data3.

2.3)

Assume the perturbation is of the following form, where a is a constant (generally interested in $a < L$ e.g $a = 20\pi/L$):

$$f_0 = \frac{\mu\phi}{\kappa - \mu} \tag{14}$$

$$g_0 = (1 - f_0)(\phi + f_0) + \epsilon \sin(ax) \tag{15}$$

You could add more sin perturbations of different wavelengths and also perturbate f's initial condition instead.

For short time spans you would use the Lyapunov exponents. So you would define a distance metric to compare them e.g d(t) the average sum of distance of g (or f):

$$d(t) = \frac{1}{N} \sum_{i=0}^{i=N} (g_i^{sted}(t) - g_i^{perb}(t))^2 \tag{16}$$

You would then plot d(t) against time until d(t) appears to stabilise (stop increasing). You could then use a least square fit on the data, up to the point in time when the data stabilises. To calculate its Lyapunov exponent you use the slope of the least squares fit and divide it by 2. A higher value of the exponent indicates that the system is sensitive to perturbations and suggests potentially chaotic behavior. You could vary ϵ and investigate how it changes the exponent. Instead of calculating an average distance, you could analyse the exponent at every point x_i of g (or f).

For Long time spans you would analyse both the perturbed and steady state system in the same way as question 2.2, focusing on the differences of the solutions, not the variation of κ (this would still be interesting to investigate), with a big focus on the correlation sum analysis. It might be useful to compare variance (and mean) of f values for both systems through time, seeing if there are any similarities, in the patterns they produce.
