Scientific Computation Project 3

01847210

December 15, 2023

Part 1

To analyze non trivial trends in the dataset we should try to extract important features and statistics from u. To begin with a simple feature lets plot the average wind speed over time for each geographical location. My function part1 will plot the yearly average speeds if called with input averagespeed = True.

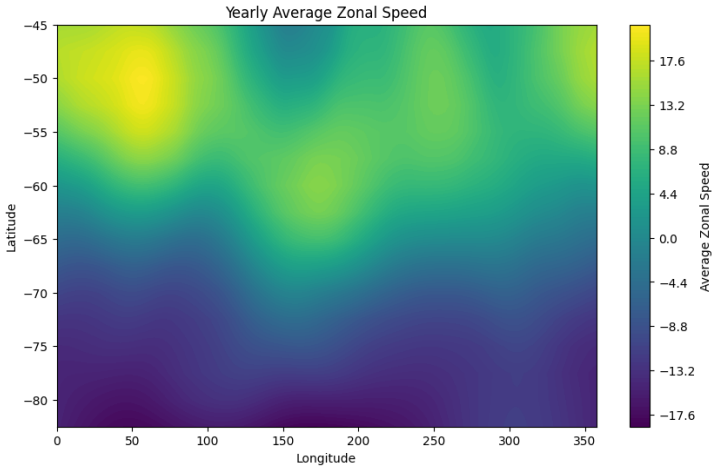

Figure 1: Contour plot of average zonal wind speed over time

Clearly from the picture we can see that on average higher latitudes will have a higher zonal wind speed, and lower latitudes will have lower (westward flow).
To analyse more non trivial spacial and temporal trends we should turn to principal component analysis. I have flattened u from a three dimensional tensor into a matrix such that time is indexed by the columns and the rows are indexed by lat-lon pairs. Now we can apply PCA to try and find either spatial or temporal trends in the data.
I have coded a function pca which returns useful insights to the data, such as the principal components and a 'good' estimate of the original dataset (depending on the p value which corresponds to the number of components to keep), the singular values corresponding to each PC which will give insight to how many components describe most of the variance in the dataset, and the scores of the PCs which tell us how relevent our spatial/temporal PCs are over time/space respectively.
What I will show below are the PCs that can be extracted from the dataset, this method is to transform the original space/time variables (potentially correlated) into uncorrelated variables that represent different features of the system.
Spatial Principal components
To Calculate the spacial principal components I have centralised my reshaped u and passed it through my pca function. Calling part1 with explained_variance_spatial=True plots the percentage of variance explained by the first p principal components against p. From this we see that the first 50 principal components explain approximately 90% of the total variance. By calling the function with no_components = 4 and spatialPCs=True then we can see the first few principal components reshaped into interpretable contour plots:

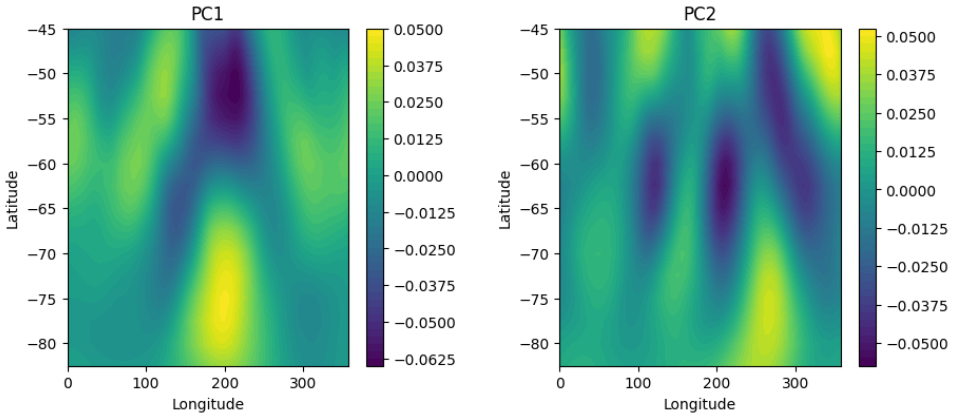

Figure 2: First two spatial PCs

These PCs represent spatial patterns in the data. For example in the first few PCs we can see like-colour zones zigzagging as longitude increases. We can also reconstruct the original data using only the first few PCs, this new data will reflect the prominent features of the system without any of the noise we see in the original system. Calling part1 with no_components = 25 and animation=True will show the reduced data, a similar less 'noisy' system. Similarly no_components can be left blank and the function will show the original animation of zonal wind speed over time.

Finally it is also possible to plot the scores of the PCs, by toggling spatial_scores to True which represent how prevelant each PC is over time. However the scores seem to remain constant and noisy, implying that these spatial patterns are generally seen for all times.

Temporal Principal components

To find temporal patterns in the data we can conduct PCA on reshaped u transposed, as the pca function will now read the days as variables and the points on the plane observations. Similar to the spatial pcs toggling temporalPCs=True and giving a number of components plots the first few temporal PCs, and explained_variances_temporal gives the proportion of variance explained by p components:

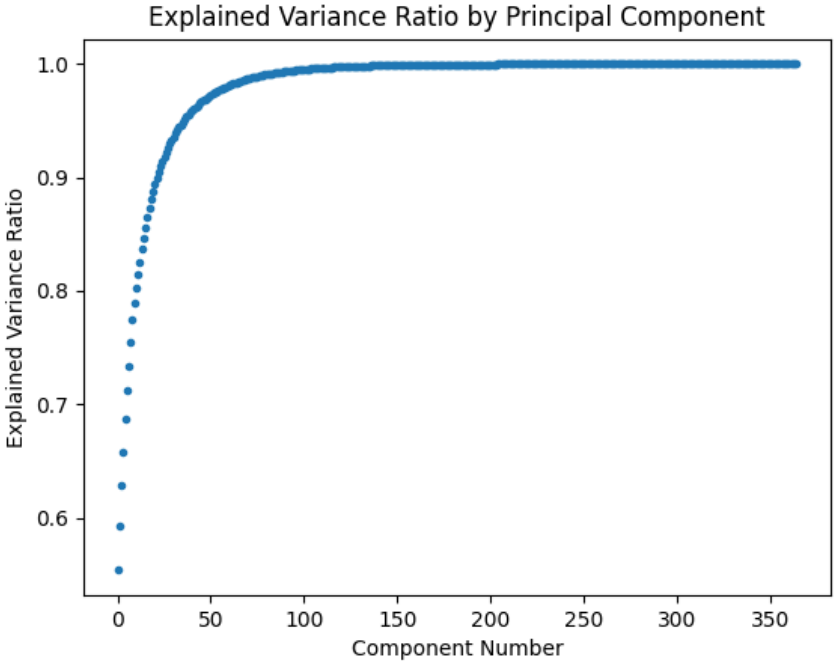

Figure 3: Explained variance for the first p temporal PCs

We can see that the first PC explains about 55% of the total variance, making this a very prominent

pattern in the dataset.

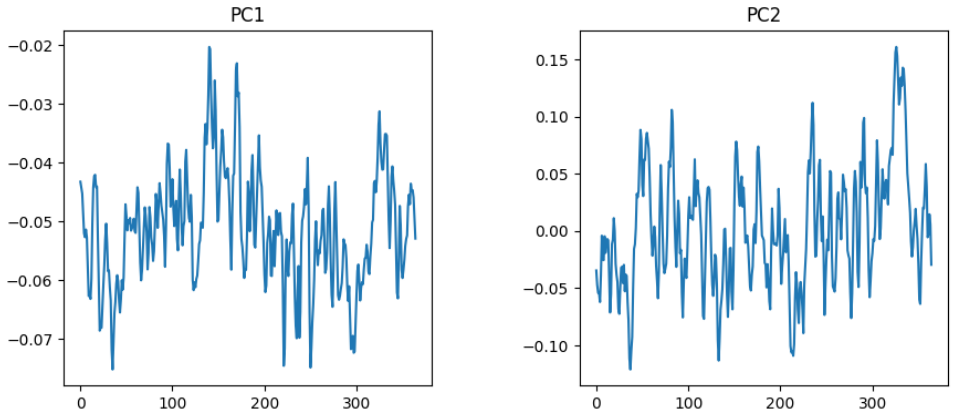

Figure 4: First two temporal PCs

Whether there are strong patterns in time is not immediately obvious however PC1 appears to show some noisy periodic behaviour. To establish more exact patterns in time we can analyse the fourier spectra of these PCs to find dominating periodicities.

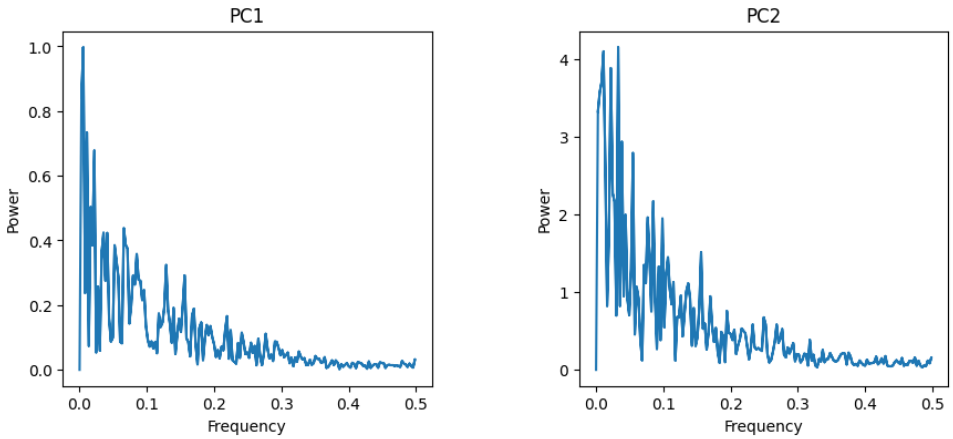

Figure 5: First two temporal PC spectrum

Clearly the sharp peaks in power decay as frequency increases, implying periodic nature in our temporal PCs.
By plotting the scores of the temporal PCs we can see the points in space most affected by these temporal patterns.

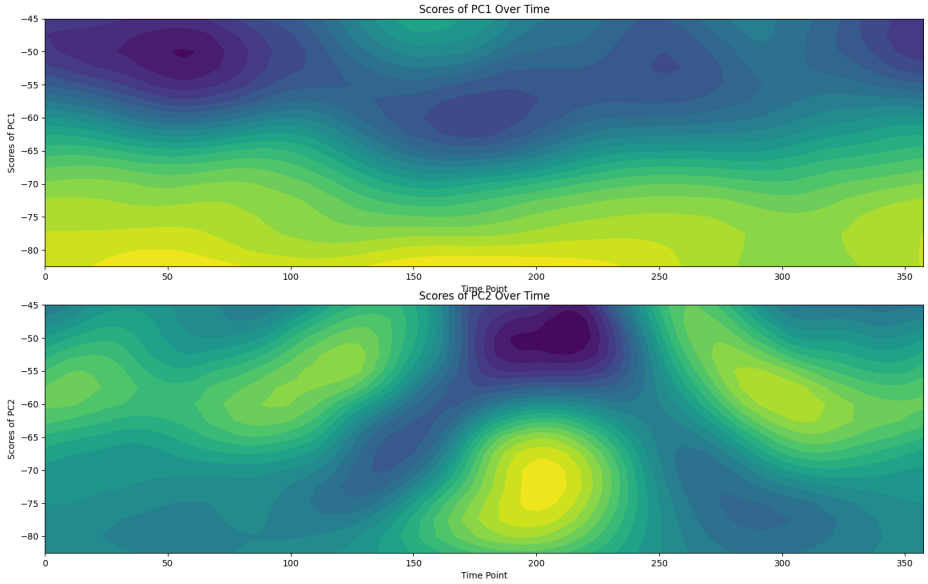

Figure 6: Score of first two temporal PCs over space

Summary
From our spatial PCs we can deduce patterns in space, such as how geographical features affect zonal wind speed like whether the wind is over sea or land, or around mountains, the cause of higher zonal wind speed at higher latitudes could be down to the feature of mapping globes to planes: a circle around a globe at greater latitude has a greater circumference (in the southern hemisphere).
We may also find features of fluid dynamics in these principal components, hinted at by the sinusoidal patterns with varying frequencies in our first few principal components.
From our temporal PCs and their scores we can find temporal patterns and features, such as seasonal change in zonal wind speed due to temperature change.
It is also worth noting that a good approximation of the data takes a large number principle components to construct, meaning the nature of how zonal wind speed evolves over time and space is not easily described by a small number of features likely due to the chaotic and noisy nature of wind.

Part 2

2.

To examine the effectiveness for the two methods I have defined a test function $f = cos(kx)sin(ky)$ and evaluated it on a grid of size 50x40. The wavenumber k is variable as we would like to see how effective each method is for varying frequencies. Part2_analyze with error = True will plot the absolute differences between each method and the real data across the grid:

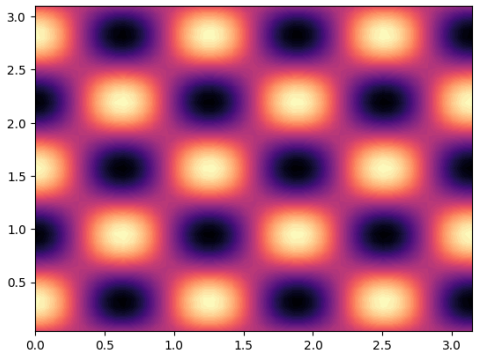

Figure 7: True values for k=5

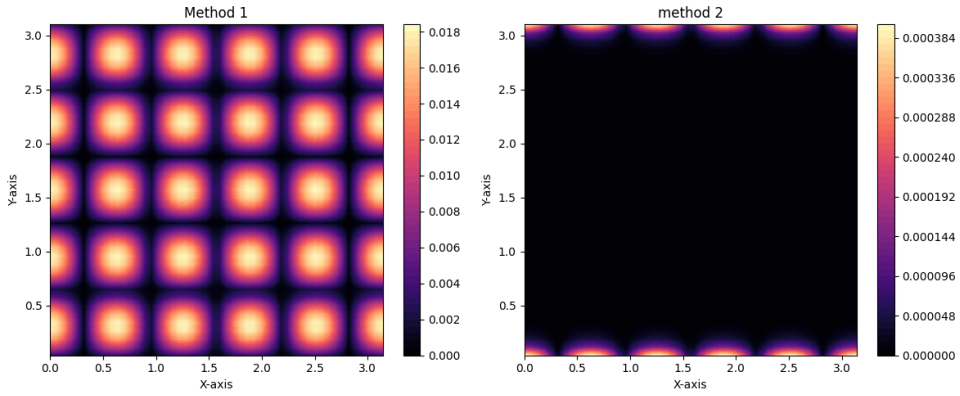

Figure 8: Errors for k=5

Immediately we can see that the implicit method, method 2, is far better at interpolating data for this specific wavenumber, with the exception of the boundarys at low and high y values. We can also see that method 1 struggles to correctly interpolate when the gradient in the y direction is changing, however does a good job when the function is locally linear.
To look deeper we can quantify the total error of a method by taking the frobenius norm across the grid, then plot the total error for each method across a broad spectrum of k.

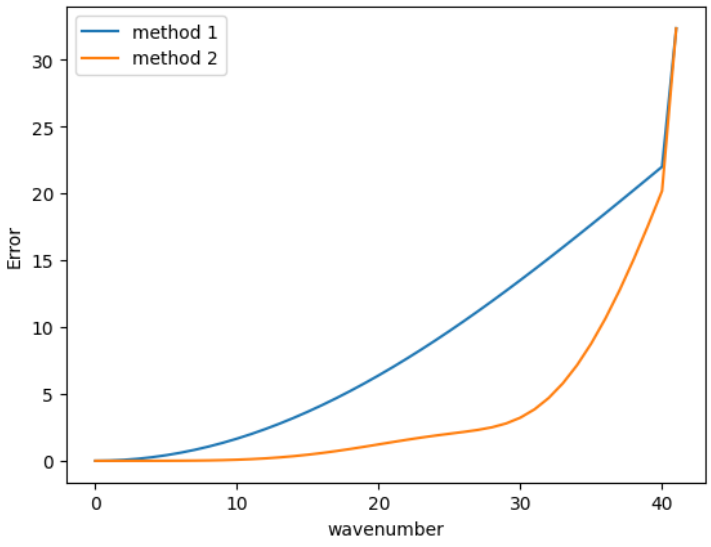

Figure 9: Error for each method for each k

Unsurprisingly method 2 will beat method one when the wavenumber is less than the number of y points.
Wavenumber analysis: To analyse why method 1 does worse than method two we can look at when it will be a good approximation with wavenumber analysis. If we consider a generic wave function $f(x) = e^{ikx}$ then we can write:

$$\tilde{f}_i(x) = e^{ikx}, f_i(x) = e^{ik(x-\frac{h}{2})}$$

where h is the grid spacing, $\tilde{f}$ is the interpolated data and f is the data.
From this we can deduce that for method one to give a good approximation we must have

$$\tilde{f} = \frac{1}{2}(e^{ik(x-\frac{h}{2})} + e^{ik(x+\frac{h}{2})})$$

Or equivalently:

$$\tilde{f} = e^{ikx}cos(\frac{kh}{2})$$

, since $\tilde{f} = e^{ikx}$ this implies method one is a good approximation when the equation

$$cos(\frac{kh}{2}) \approx 1$$

is satisfied.
For method 2 we can apply the same method to find a condition for the interpolation to be accurate:

$$\alpha(e^{ik(x-h)} + e^{ik(x+h)}) + e^{ikx} = \frac{b}{2}(e^{ik(x-\frac{3h}{2}} + e^{ik(x+\frac{3h}{2})}) + \frac{a}{2}(e^{ik(x+\frac{h}{2}} + e^{ik(x-\frac{h}{2})}$$

which is equivalent to

$$1 \approx acos(\frac{kh}{2}) + bcos(\frac{3kh}{2}) - 2\alpha cos(kh)$$

multiplying our conditions for method 1 and method 2 by kh on both sides gives a relationship between our wavenumber kh and a modified wavenumber, to see which method is more accurate we can plot these against each other:

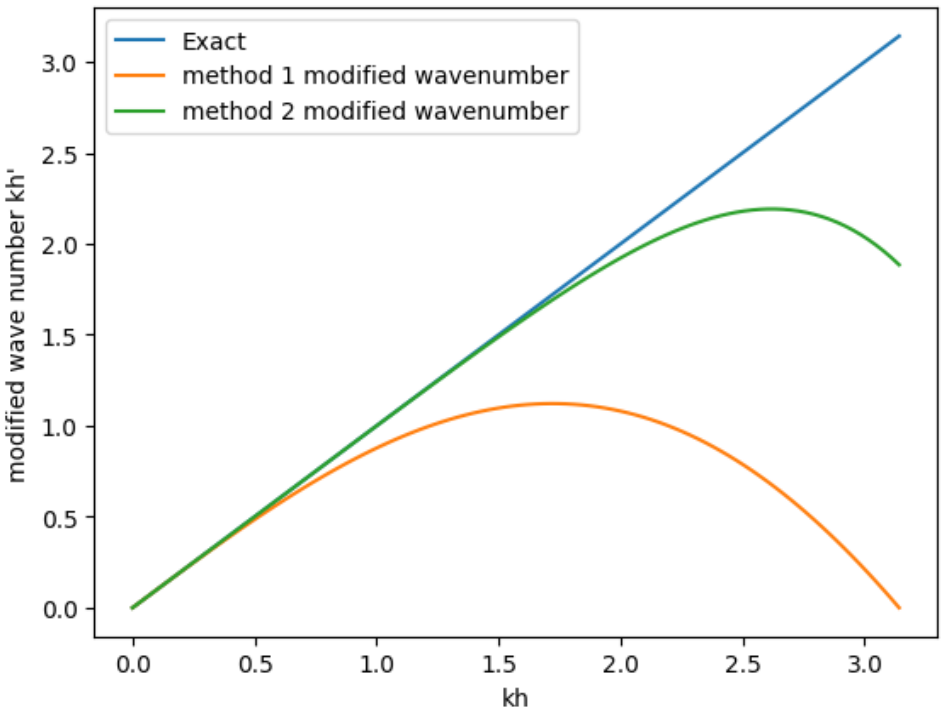

Figure 10: Wave number analysis

From the equations we can see that we get a 1% error from method 1 at about 0.283, implying the number of points needed per wavelength is about $\frac{2\pi}{0.283} \approx 22$ since the wavelength is $\frac{2\pi}{k}$. And for method

2 similarly: 1% error occurs at abount 1.5 so we would need approximately 4 points per wavelength, a stark improvement from method 1.
To see these graphs for arbitrary k part2_analyze can be called with arbitrary k, and wavernumbers = True or k_errors = True or error = True.
Since method two is much more accurate than method 1 and involves solving matrix equations one might expect the time complexity to be greater, so counting the operations we obtain:
Method 1
Method 1 adds two matricies together and multiplies by a constant, since the matrcies are mxn this method is O(mn), the question states n and m are of similar order so this is equivalent to $O(n^2)$
Method 2
Method 2 initialises a banded matrix, constructs c and solves the banded matrix equation. Breaking this down we have O(m) for initialising 3 length m vectors to describe the banded matrix, O(nm) to create c as it is created from all elements of the nxm matrix f, and O(nm) to solve the banded equation. This is because there is one O(m) banded equation to solve for each column of f which is length n.
Overall with O(n)=O(m) we see that both methods are actually $O(n^2)$, however method 1 has much fewer operations so will have a lower wall time. Both methods have a similar error when the number of y points is near the number of waves, so in this case method one should be preferred however the error is large so interpolation using either method will be inaccurate and should be used with caution.

Part 3

1.

To analyse the impact of c on the solutions for the system of ODEs we should plot the solution for varying c:

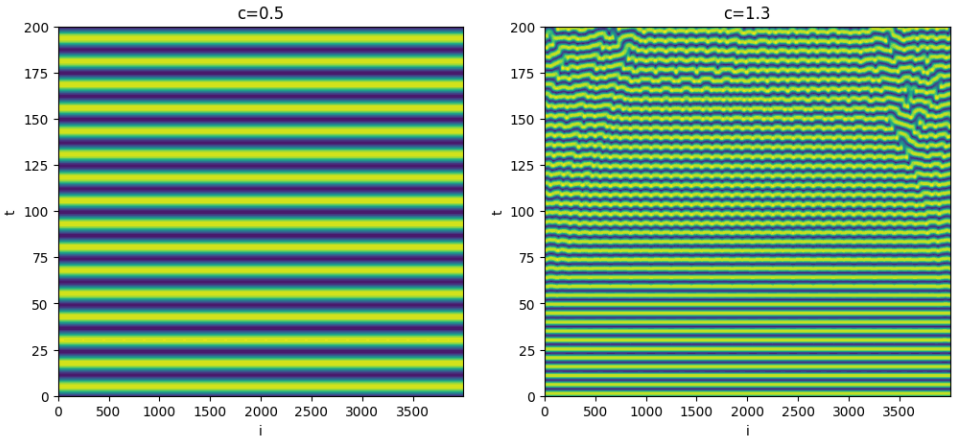

Figure 11: Contour map of u(t), varying c

We see that for each c value our solution for u follows a sine curve with regular period, and for c = 1.3 and 1.5 the period for each u_i becomes unpredictable and seemingly chaotic after a short time. Plots for v show similar graphs, with the initial state being cosine. We want to analyse the settled state of the dynamical system, so it makes sense to discard the transient initial state as done in part3_analyze, where I have changed the construction of the initial conditions to be the state after tf=200 as opposed to 20, this is to ensure we are analyzing the settled state of the system.
phase plot
Plotting u against v is difficult since the dimensions of u and v are arbitrary, however each dimension has similar dynamics in the settled state. Toggling phaseplot=True in part3_analyze and choosing c=1.3 will show the phase plot of one dimension, after discarding the effect of initial conditions. We see that unlike the circle mapped out by the fixed period transient state, less predictable trajectories are seen due to the changing periods of v and u.

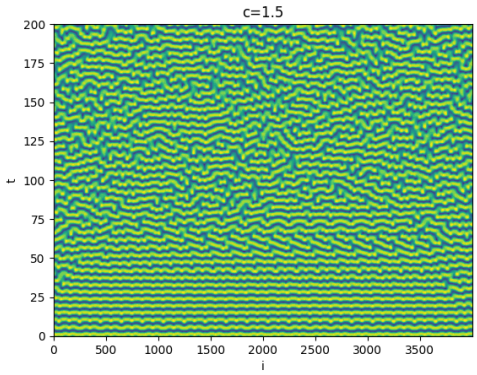

Figure 12: Contour map of u(t), varying c

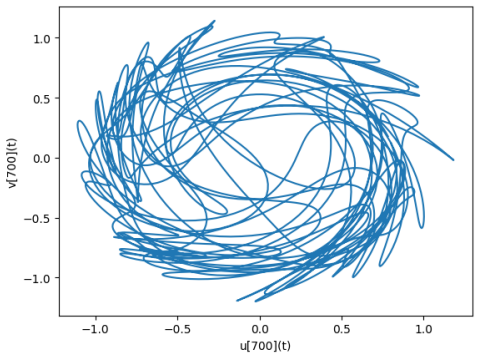

Figure 13: one dimension of u against v

Fractal dimension

It is unclear whether the settled state behaviour of u for c=1.3 is actually periodic, so we should analyze the spectral density of each u_i with Welches method to find the frequencies that best describe a full oscillation around this n dimensional sphere. First we should discard the transient states so we only get the settled state we want to analyze, the state being analyzed is below. Spectrum = True will give a contour plot of the spectral values, giving insight into the dominating period.

I have included the spectrum of the settled system for c=0.5 (where we observe periodic behaviour) as

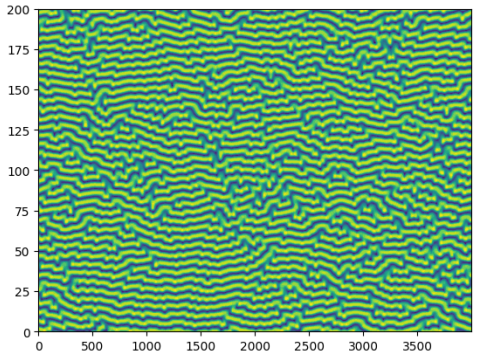

Figure 14: Settled state for c=1.3

a reference, and we see that each u has an indentical spectrum and a lower frequency than in the c=1.3 system.

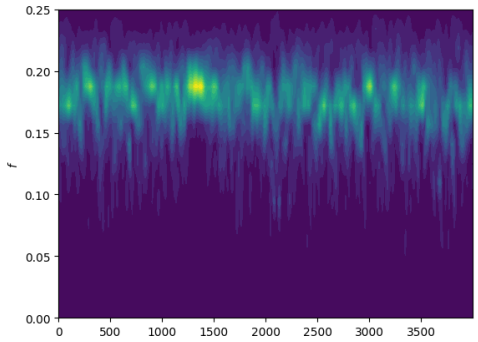

Figure 15: Spectral analysis of each u (c=1.3)

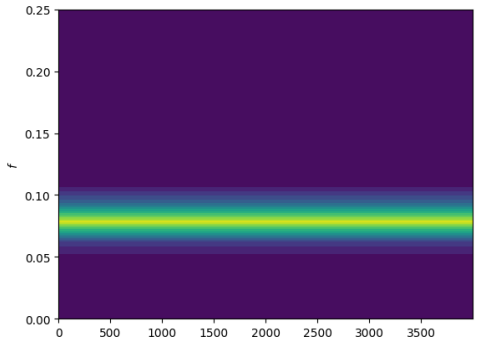

Figure 16: Spectral analysis of each u (c=0.5)

As expected we see a broad range of frequencies, indicating noisy and chaotic behaviour, and the range of frequencies is slightly different for each u_i.
To find a good estimate of the fractal dimension of the final settled state we calculate the correlation sum as a good estimate. To compute an accurate estimate it is important that the vectors we include are not correlated. However in this system it can be seen with np.corrcoef(u[0],u[1]) that each obervation is correlated with the last.
To avoid this we can use time delays, it is clear from Welches method that the averge peak period for each dimension is approximately f=0.2. Setting the time delay to be tau=1/5*f = 1 we can conveniently slice u in steps of four to find minimally correlated vectors. Sure enough np.corrcoef(u[0],u[4]) yields a much lower correlation.
To calculate the correlation dimension the function correlation_sum calculates the distance between every vector in the sample, then for a range of epsilon counts the number of distances less than epsilon. It then plots the log of this count against log(eps) where we expect the gradient of the linear part of this graph to be our correlation dimension. This is because $C(\epsilon)$ scales with ϵ^d. I chose my range of epsilon to avoid capturing non trivial counts such as 0 distances $\leq \epsilon$ and all distances $\leq \epsilon$ by ranging from the minimum distance to the maximum distance.
Toggling correlation_dimension to True shows the graph and the correlation dimension which is found to be approximately 15.3. Since this is the correlation dimension of a settled state for a system of autonomous ODEs and $15.3 \geq 2$ we see there is chaotic behaviour.

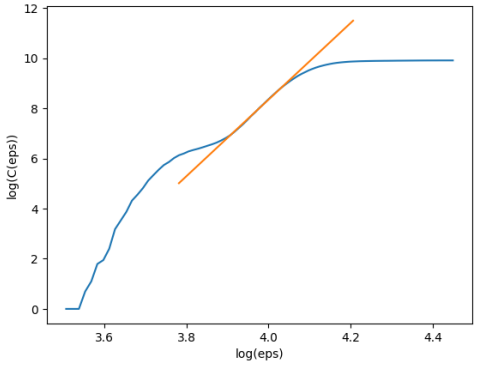

Figure 17: Calculating the correlation dimension

2.

The four lines of code are for finding a reduced rank approximation to the solution A by the means of projecting the data onto the first x principal components of A. This is a principal component analysis method to capture most of the explained variance in the first x principal components.
Line 1 computes A.T@A, the covariance matrix of A which is real symmetric, then extracts the eigenvalues and eigenvectors of this matrix. These are the principal components of A.
Line 2 multiplies A and the eigenvector matrix found in line 1. This projects A onto the basis of principal components, or equivalently the eigenvectors of the covariance matrix.
Line 3 reconstructs an estimate of the data in A in a matrix A2, by taking the first x variables in the projected data and taking the outer product with the first x eigenvectors, projecting back into the original variables.
Line 4 computes the square difference between each component in the approximated data and the real data, then sums over every component creating a cumulative error value.
Considering the covariance matrix method was used to compute the estimate of the data this code is efficient. Line 1 takes advantage of A.T@A being real symmetric and uses linalg.eigh which is designed to be more efficient for real symmetric matricies. The last three lines make use of slicing, numpy operations and matrix operations which are optimised in numpy.
The code may be made more efficient considering we know A is non square, singular value decomposition is optimised for non square matricies and can yield the same principal components and singular values as the covariance matrix approach. This would avoid calculating A.T@A which is expensive when A is large.

Figure 18: Add figure description here
