Scientific Computation Project 3

02027072

December 15, 2023

Part 1

We begin by reshaping u into a 2304×365 *NumPy* array $\mathbf{A}$, with each day as a datapoint in the columns, and locations (longitude and latitude combinations) as the attributes in the rows, and we apply PCA to the flattened 2-dimensional data. This allows us to spot temporal trends in wind speed.

When applying PCA, we note that the number of non-zero singular values is 364, which shows that the matrix $\mathbf{A}$ is not full rank as we have 365 days.

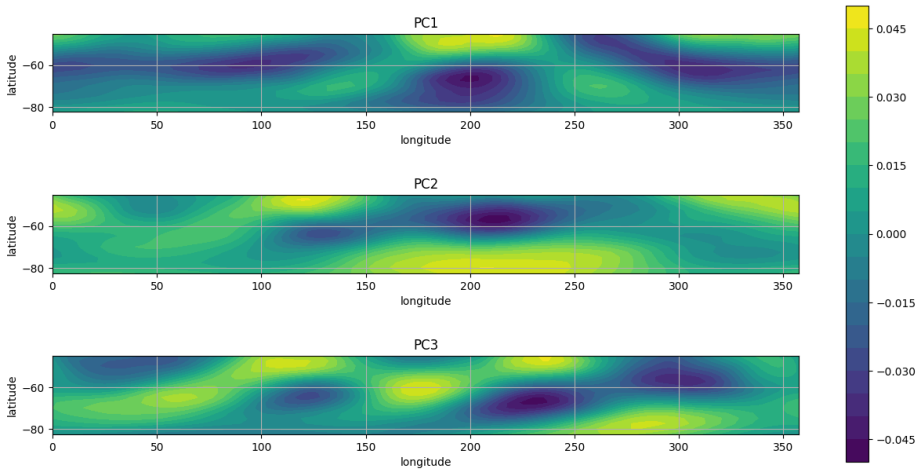

Figure 1: First 3 PCs

Figure 1 shows the first 3 principal components, i.e. the directions in which the highest variance in wind speed is captured. The maps show distinct zones which exhibit similar behaviour over time.

From Figure 2, we observe that 80% of the total variance is retained at around 20 principal components and that under 20% of the total variance is captured by the first PC.

Figure 3 shows our transformed data values as projections onto the first 2 principal components. There are no clear patterns in this so we do further analysis on this time series.

From Figure 4, we observe there are none of the frequencies significantly dominate the others in both spectral density estimates, suggesting there are no consistent periodic patterns in wind speed over the year. The frequency density also appears to decrease as frequency increases, indicating that trends appear to be gradual and seasonal, and there are fewer high-frequency variations (rapid changes in wind speed).

We see from Figure 5 that there is a general positive correlation in wind speed between subsequent days. This also supports our earlier observation that there are few high-frequency variations.

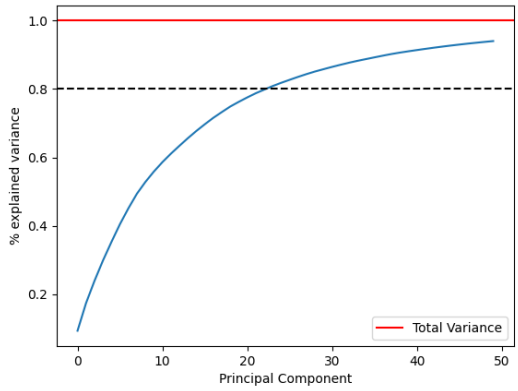

Figure 2: Cumulative retained variance of PCs

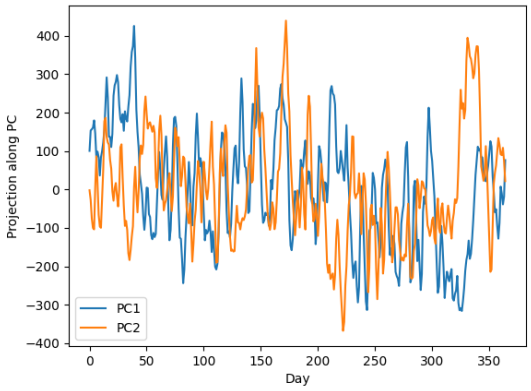

Figure 3: Projections onto first 2 PCs

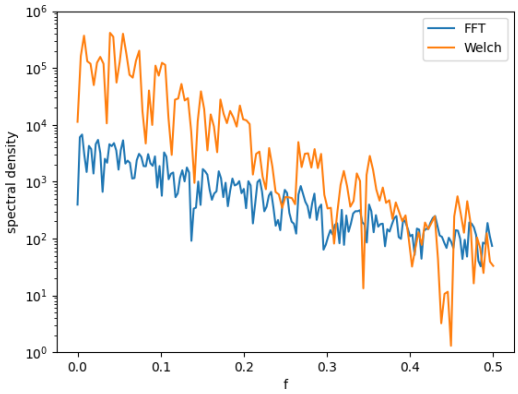

Figure 4: FFT and Welch's on projection onto PC1

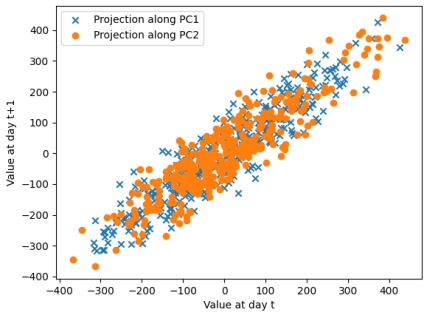

Figure 5: Subsequent days of PC1 and PC2

We then transpose our array $\mathbf{A}$ into a 365×2304 *NumPy* array, now treating the locations as the datapoints, and each day as an attribute.

Figure 6 shows the first 2 principal components, i.e. the values in time that capture the most variance.

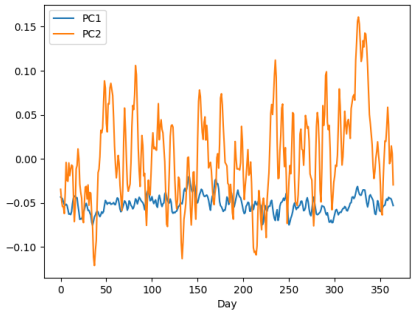

Figure 6: First 2 Principal Components

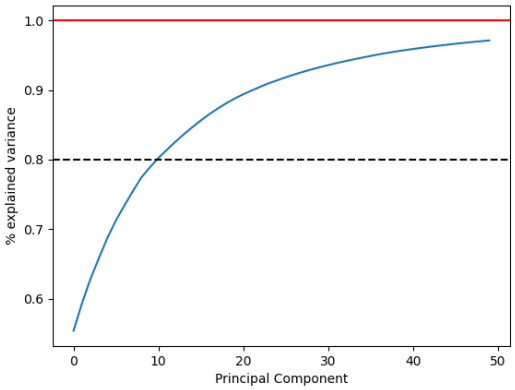

Figure 7: Cumulative retained variance of PCs

From Figure 7, we observe that 80% of the total variance is retained at around 10 principal components, and that the first principal component captures over 50% of the total variance.

From Figure 8, we see a large variation in scores between the higher and lower latitudes, indicating a clear split in wind speed patterns between the upper and lower halves of the map. The second principal

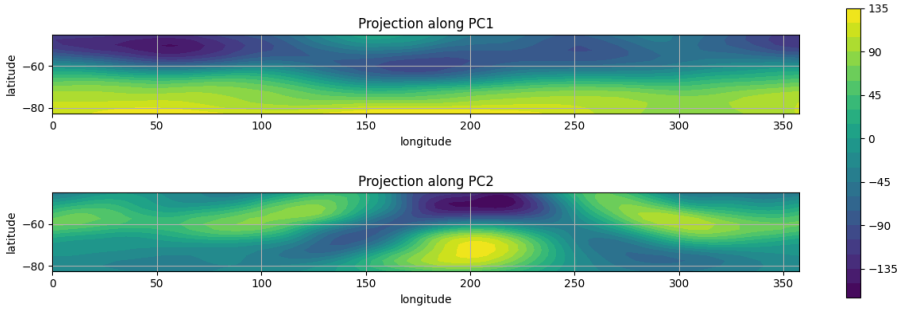

Figure 8: Projected values of first 2 principal components

component also shows distinct zones where the scores are similar, indicating the presence of spatial clusters and patterns. There appears to be two distinct zones within around 175 to 250 longitude in both upper and lower halves that share similar wind speed patterns.

Part 2

2.

For method 2, we solve the system of linear equations

$$\mathbf{A}\tilde{\mathbf{f}} = \mathbf{B}\mathbf{f}$$

for $\tilde{\mathbf{f}}$ with columns $\tilde{\mathbf{f}}_j = (\tilde{f}_{1/2,j}, \ldots, \tilde{f}_{m-3/2,j})^\top$, where

$$\mathbf{A} = \begin{pmatrix} 1 & & & & & \\ \alpha & 1 & \alpha & & & \\ & \ddots & \ddots & \ddots & & \\ & & \alpha & 1 & \alpha & \\ & & & & 1 & \end{pmatrix}$$

is a banded square matrix,

$$\mathbf{B} = \begin{pmatrix} \tilde{a} & \tilde{b} & \tilde{c} & \tilde{d} & & \\ \frac{b}{2} & \frac{a}{2} & \frac{a}{2} & \frac{b}{2} & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & \frac{b}{2} & \frac{a}{2} & \frac{a}{2} & \frac{b}{2} \\ & & \tilde{d} & \tilde{c} & \tilde{b} & \tilde{a} \end{pmatrix}$$

is a rectangular banded matrix, and the columns of $\mathbf{f}$ are $\mathbf{f}_j = (f_{0,j}, \ldots, f_{m-1,j})^\top$.

To solve this linear equation efficiently, we convert matrix $\mathbf{A}$ to "matrix diagonal ordered form" and then apply *scipy.linalg.solve_banded*.

We apply both methods to the following functions on a 2-dimensional grid:

- *smooth oscillating function*

$$f(x,y) = \sin(2\pi x)\cos(2\pi y)$$

- *Franke's function - used as a test function for interpolation problems*

$$\begin{aligned} f(x,y) =& \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right) \end{aligned}$$

To determine the accuracy, cost and efficiency of both methods, we first apply them to each function and plot the errors as the absolute difference between the interpolated value and the actual value. We also measure the average wall times (over 1000 runs) and calculate the mean error (mean absolute difference between interpolated and calculated values). We test the function on a grid with sizes $n = 50$ and $m = 40$.

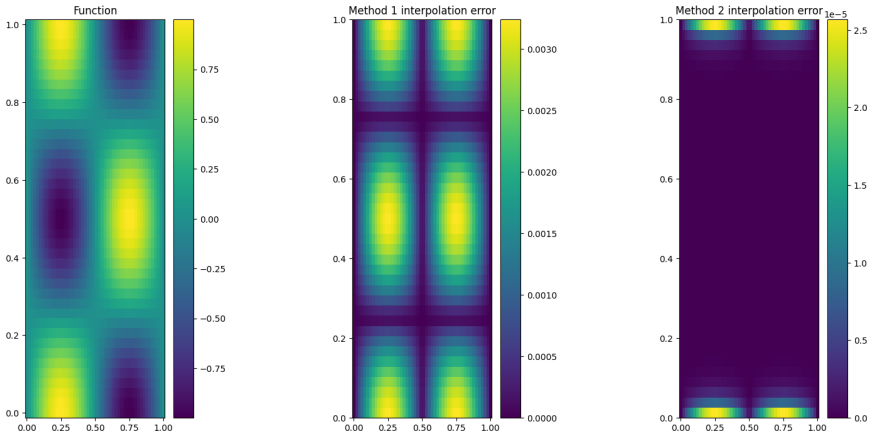

Figure 9: Results for oscillating function

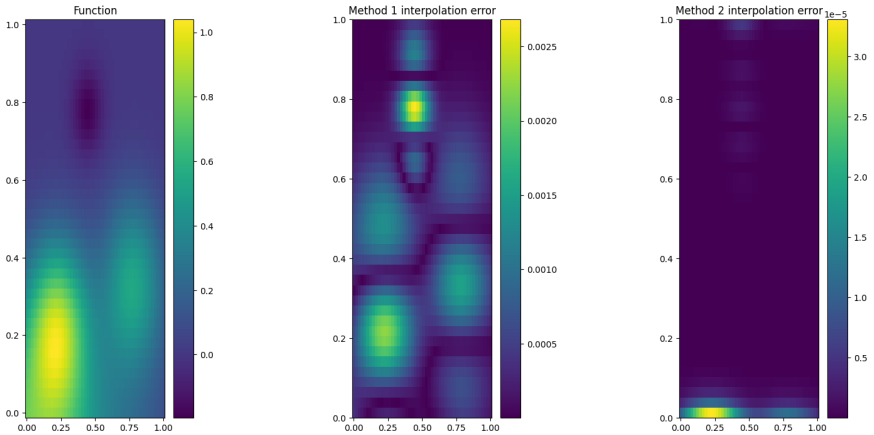

Figure 10: Results for Franke's function

From figures 9 and 10, we observe that method 1 produces large errors at the peaks and troughs of both functions. Method 2 has a much more stable error value throughout both functions, except at the boundaries, where there is a significantly larger error value.

	Error Method 1	Error Method 2
Oscillating function	1.288e-03	1.234e-06
Franke's function	5.093e-04	7.182e-07

Table 1: Mean Error

From Table 1, we note that method 2 has a significantly lower error for both functions, hence it is more accurate overall despite the higher error on the boundaries.

From Table 2, we see that method 2 has significantly longer wall times, as the method has a higher computational cost due to the use of matrix multiplication and solving linear equations, in comparison to the simpler method 1.

	Mean Wall Time (s) Method 1	Mean Wall Time (s) Method 2
Oscillating function	7.026e-06	2.693e-04
Franke's function	6.220e-06	3.413e-04

Table 2: Wall Times

We then test both methods on Franke's function for a range of different grid sizes (fixing $n = m + 10$ for consistency).

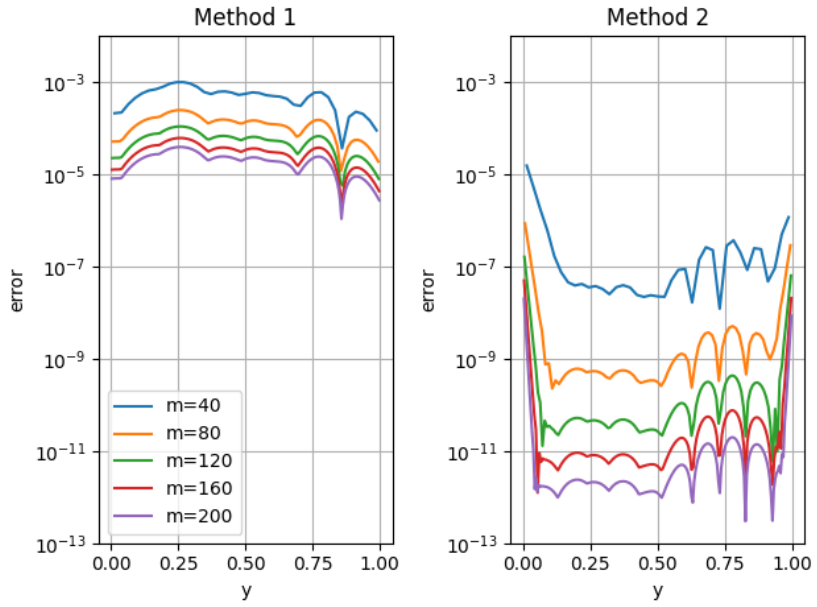

Figure 11: Mean Error for each y

In Figure 11, we test $m = 40, 80, \ldots, 160, 200$. We observe that the error is more consistent for different y with method 1, whereas method 2 is significantly worse at the boundaries. Both methods are more accurate at higher m values, and method 2 is consistently more accurate than method 1.

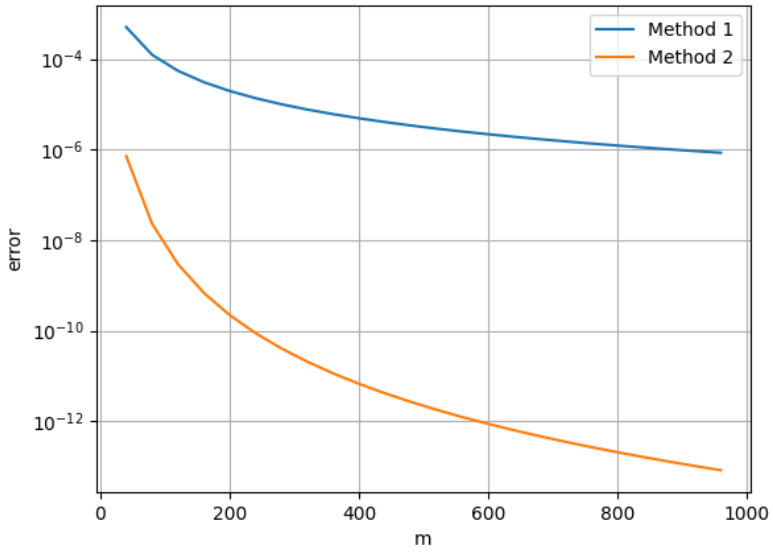

Figure 12: Mean Error against m

In Figure 12, we test for higher m values. The error for increasing y is almost exponential for both methods, with the error for method 2 decreasing at a higher rate than method 1. Both errors appear to gradually converge to a fixed error value across m.

Part 3

1.

For this question, we perform analysis on $c = 0.5, 1.2, 1.3, 1.5$, and we let $x_i = u_{i-100}$ for $i = 100$ to $n - 100$.

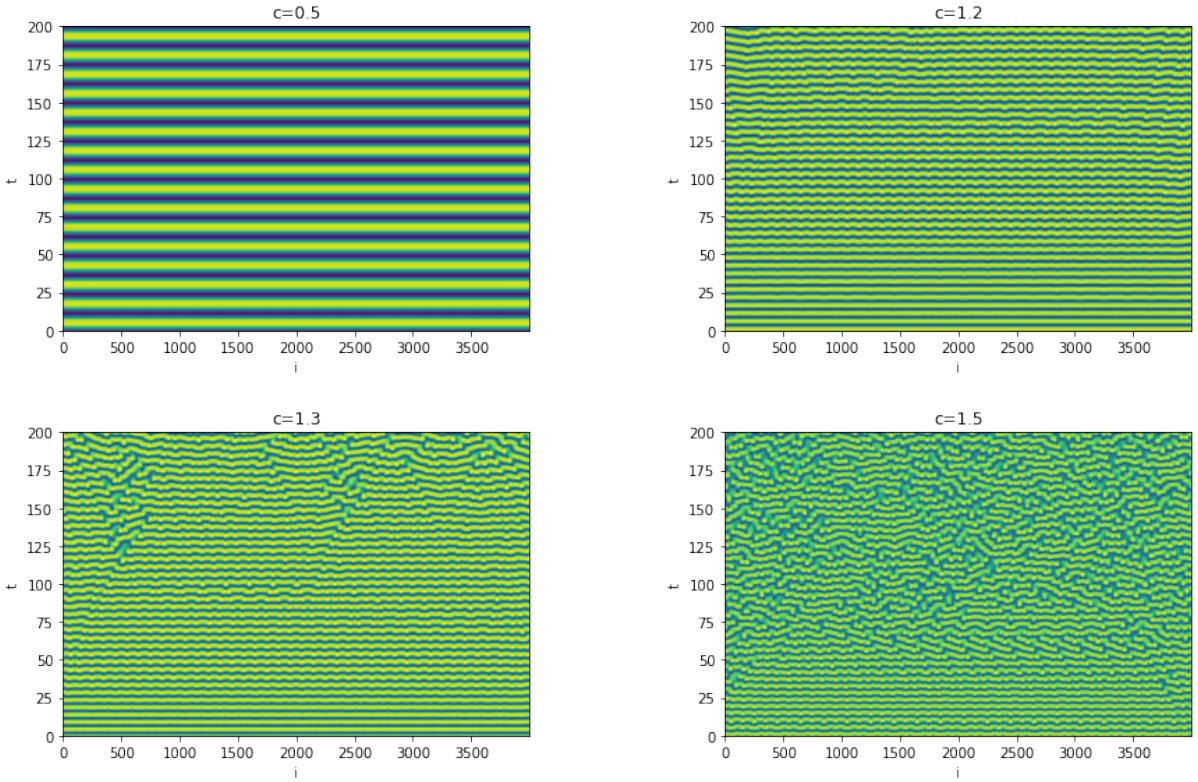

Figure 13: Contour plots with time against i

We observe from Figure 13 that the solutions are oscillating initially for all c, and remain oscillating for $c = 0.5$ and $c = 1.2$, but the periodicity appears to break down gradually for $c = 1.3$ at around $t = 110$, and very quickly for $c = 1.5$ at around $t = 40$.

Figure 14 makes the sinusoidal oscillations very clear for $c = 1.3$, and we observe the waves breaking down over time.

We calculate the correlation sum $C(\epsilon)$ (as seen in lecture slides) for each c. Figure 15 shows the correlation sum plots and the corresponding least squares fit in log-log scale. The slope of this fit is the correlation dimension. The dimension is less than 2 for $c = 0.5$ and $c = 1.2$, and more than 2 for $c = 1.3$ and $c = 1.5$, suggesting the system is stable when $c \leq 1.2$.

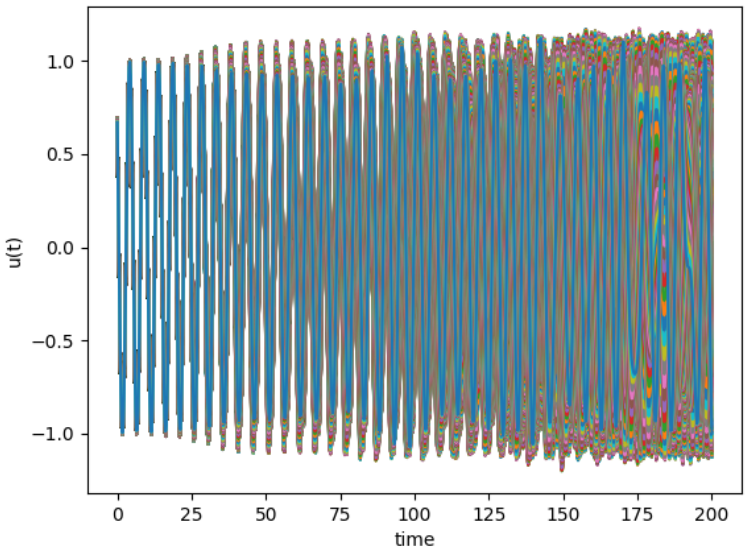

Figure 14: Solution for $c = 1.3$ against time

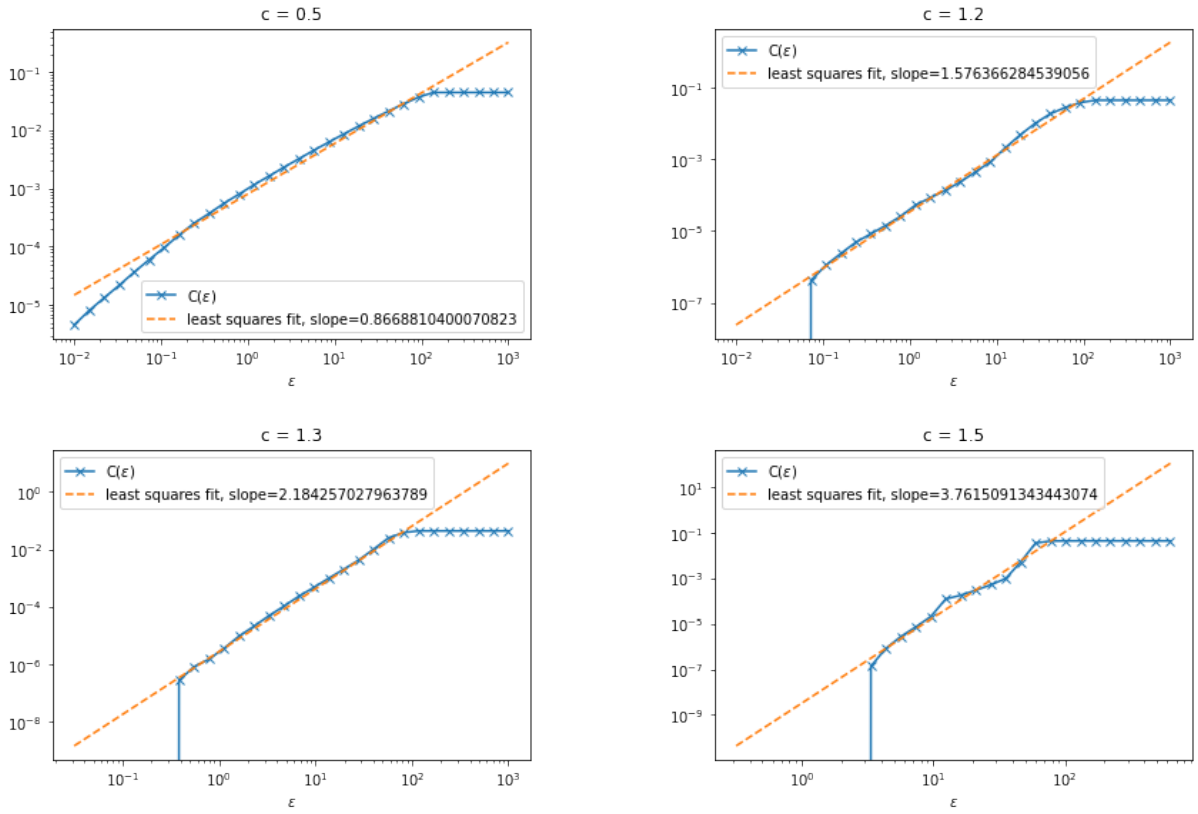

Figure 15: Correlation sum plots

We then apply PCA to x and use the first component of the transformed data to spot patterns.

Figure 16 shows the frequency spectrum at $c = 1.3$ for the first component. We apply Welch's to all values of c to estimate their dominant frequencies and time scales, these can be seen in Table 3. The time periods should approximately correspond to the period of the sinusoidal oscillations seen in Figure 13.

Figure 17 shows the dynamics for each c. For the phase plots, we plot the scores of the second

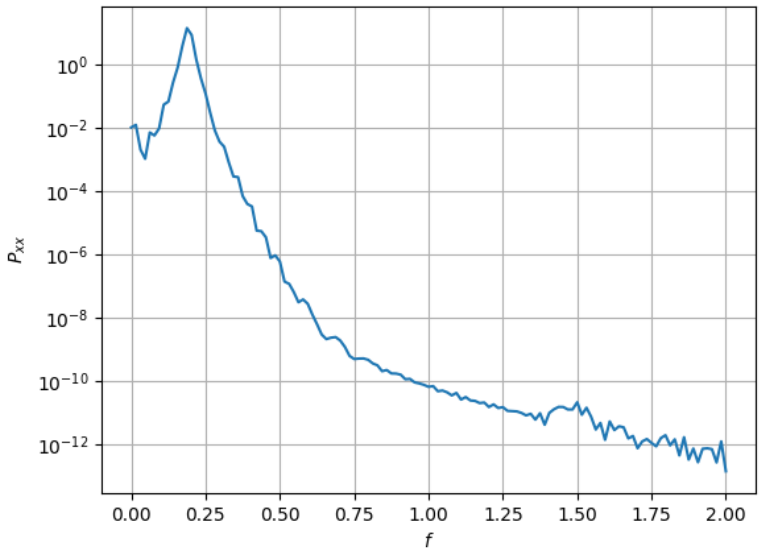

Figure 16: Power spectrum on first component of PCA-transformed data using Welch's method for $c = 1.3$

	Dominant Time Period (1/f)
$c = 0.5$	12.8
$c = 1.2$	5.33
$c = 1.3$	5.33
$c = 1.5$	4.92

Table 3: Dominant Time Period

principal component $\tilde{x}_1$ against the scores of the first component $\tilde{x}_0$. For the transition maps, we take τ as $\frac{1}{5}$ of the dominant time scale derived from Table 3 and use this to plot $\tilde{x}_0(t+\tau)$ against $\tilde{x}_0(t)$. These plots support our conclusion that the systems are stable for $c \leq 1.2$, and show that the systems exhibit chaotic behaviour for $c = 1.3$ and $c = 1.5$.

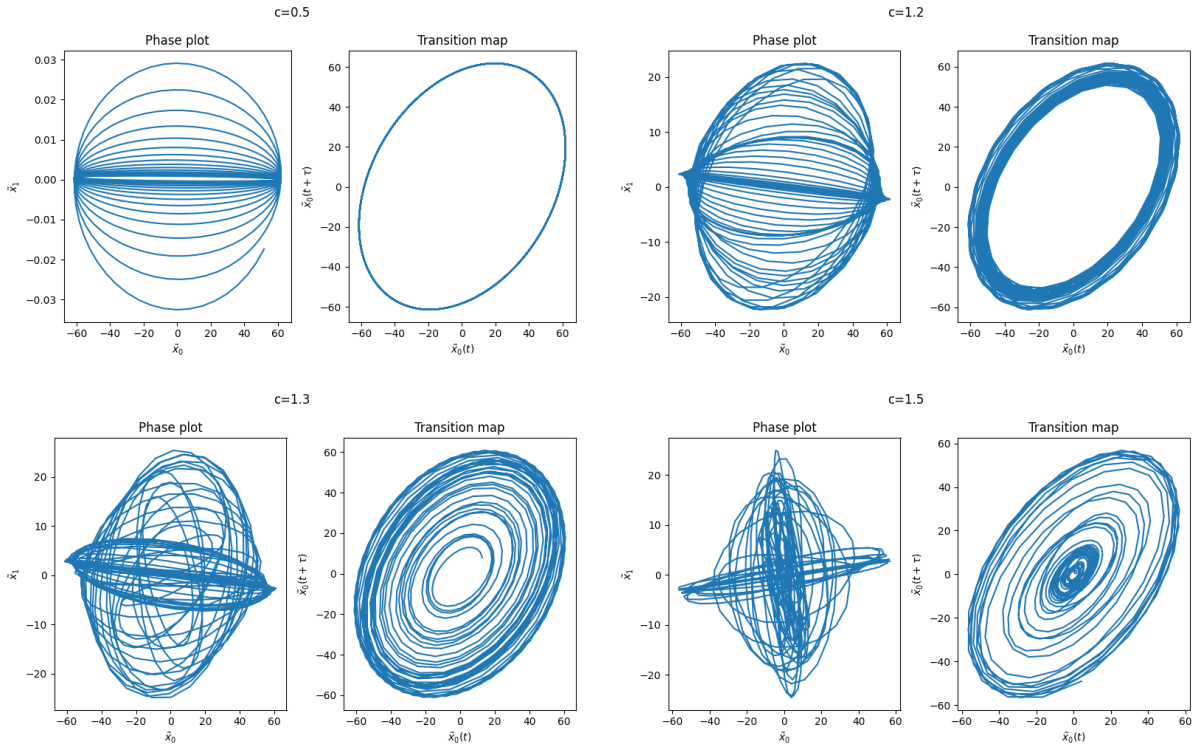

Figure 17: Phase plots and transition maps of the PCA-transformed data

2.

The 4 lines of code in function *part3q2* apply principal component analysis to the solution matrix $\mathbf{A}$, then return a lower x-dimensional approximate reconstruction of $\mathbf{A}$, along with the error of the reconstruction.

1. The first line retrieves the principal components of $\mathbf{A}$ efficiently by finding $\mathbf{v_1}$, the eigenvectors of the covariance matrix $\mathbf{A}^\top\mathbf{A}$, using *np.linalg.eigh*. Calculating the covariance matrix and its eigenvalues is computationally faster than the SVD algorithms we use in *part1* for PCA, however, it is more memory-intensive and can be numerically unstable.

2. The second line calculates $\mathbf{v_2}$, the transformed data matrix, by transforming $\mathbf{A}$ into the space spanned by the principal components using matrix multiplication.

3. The third line uses the first x columns of the transformed data matrix, along with the first x principal components, to reconstruct the rank-x approximation of A. This again uses matrix multiplication

4. The last line calculates the square error of the reconstruction efficiently using *NumPy* vectorised operations.
