**Report for exercise 5 from group H**


| | |
|---|---|
| Tasks addressed: | 5 |
| Authors: | Hammad Basit ()<br>Antonia Gobillard ()<br>Nayeon Ahn ()<br>Muhammed Yusuf Mermer ()<br>Milena Schwarz () |
| Last compiled: | 2024–05–18 |
| Source code: | https://github.com/Hammad-7/MLCMS_Exercises.git |


The work on tasks was divided in the following way:

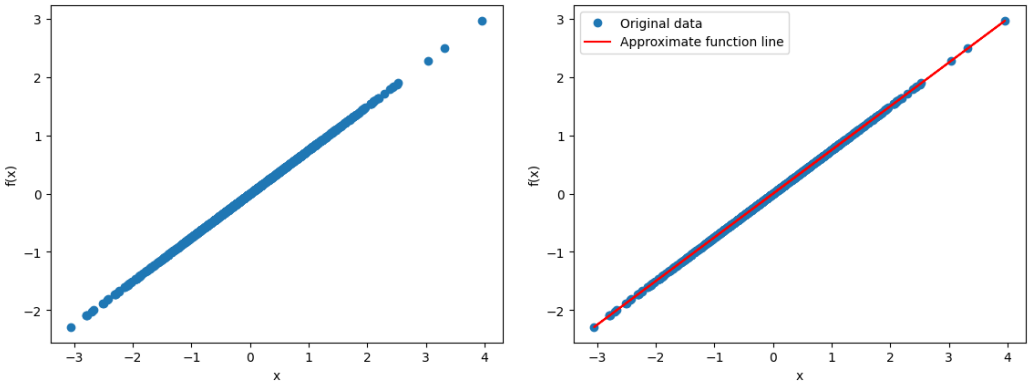
| | | |
|---|---|---|
| Hammad Basit () **Project lead** | Task 1 | 20% |
| | Task 2 | 20% |
| | Task 3 | 20% |
| | Task 4 | 20% |
| | Task 5 | 20% |
| Antonia Gobillard () | Task 1 | 20% |
| | Task 2 | 20% |
| | Task 3 | 20% |
| | Task 4 | 20% |
| | Task 5 | 20% |
| Nayeon Ahn () | Task 1 | 20% |
| | Task 2 | 20% |
| | Task 3 | 20% |
| | Task 4 | 20% |
| | Task 5 | 20% |
| Muhammed Yusuf Mermer () | Task 1 | 20% |
| | Task 2 | 20% |
| | Task 3 | 20% |
| | Task 4 | 20% |
| | Task 5 | 20% |
| Milena Schwarz () | Task 1 | 20% |
| | Task 2 | 20% |
| | Task 3 | 20% |
| | Task 4 | 20% |
| | Task 5 | 20% |

## Report on task 1, Approximating functions

For this task of approximating linear functions using least-squares minimization, we use the two datasets, (A) `linear_function_data.txt` and (B) `nonlinear_function_data.txt` provided to us. Each of them contains 1000 one-dimensional points each, with two columns: $x$ and $f(x)$.

- **First part**: `Approximating function in dataset (A) with a linear function`
  For dataset (A), containing 1000 points, we aim to approximate the underlying linear function. The linear model is represented by $f_{linear} = Ax$, where A is the matrix of coefficients. We used the function `least_square_minimization` employing *scipy.linalg.lstsq* to find the matrix A. The original and the linear approximation of the data is shown in figure 1. It can be seen that our linear function provides a highly accurate approximation of the data.

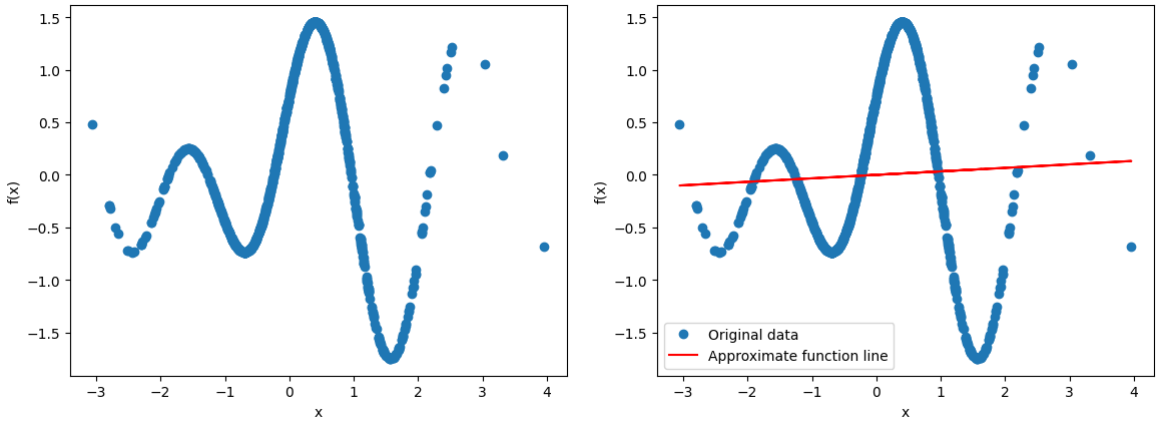
  **Using radial basis functions for approximating dataset (A):** Choosing radial basis functions for linear approximation is generally not a good idea because of the extra parameters and high computational overhead. Instead, linear models are capable, computationally efficient, and provide accurate representations for linear functions without the risk of overfitting.




(a) Original Dataset (A) (b) Linear Approximation of Dataset (A)

Figure 1: Original and approximated Dataset (A)

- **Second part:** `Approximating the function in dataset (B) with a linear function`
  The given dataset (B) is nonlinear as can be seen in figure 2a. Using the same linear approximation as in the previous task, we plotted the approximated function shown in figure 2b. A linear model can not properly approximate a non-linear dataset as can be seen by its inability to capture the dataset patterns.

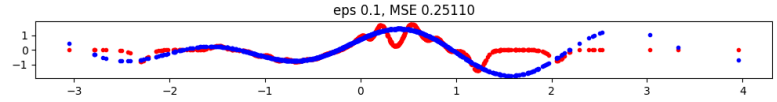



(a) Original Dataset (B) (b) Linear Approximation of Dataset (B)

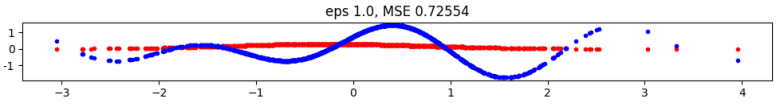
Figure 2: Original and approximated Dataset (B)

- **Third part:** `Approximating the function in dataset(B) with a combination of radial functions`
  For this section, we aim to approximate the non-linear function in dataset (B) with radial basis functions (RBFs). These basis functions provide greater flexibility to capture nonlinear relationships in the data. The challenge now lies in selecting the appropriate $L$ (number of centers or the number of functions) and the $\epsilon$ (width of basis functions).

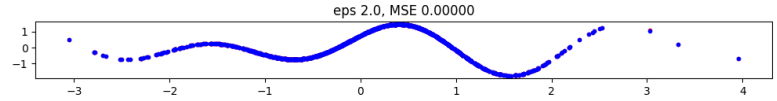
  – **Selecting $\epsilon$:** Selecting $L$ to be equal to the number of data points will give us maximum accuracy but at the same time will be extremely computationally intensive. However, we can use a moderate number of functions to try and narrow down the values of $\epsilon$. So we first set the value of $L$ as 30 and try to fit the function to our data. We see in Fig. 3 that for values of $\epsilon$ greater than 1, the function approximation was pretty good. So we arbitrarily chose 2.0 as the $\epsilon$ value.




(a) $\epsilon = 0.1$



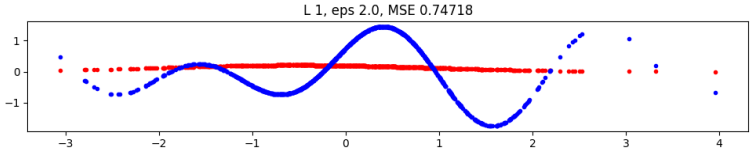

(b) $\epsilon = 1.0$



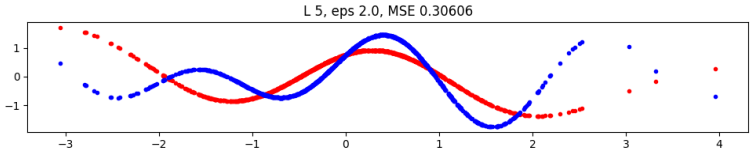

(c) $\epsilon = 2.0$

Figure 3: Function approximation for different values of $\epsilon$

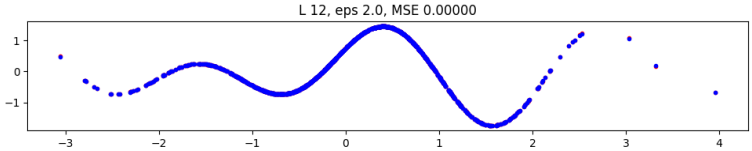
  – **Selecting *L*:** Now, using this $\epsilon$, we again tried to approximate the function by choosing different values of $L$ between 5-20. Looking at the result, we found that a value of 12 (Fig. 4c) captured the function pretty well without overfitting. Thus, we chose the values of $L$ to be 20 and $\epsilon$ to be 2.0.




(a) L=1




(b) L=5




(c) L=12

Figure 4: Function approximation for different values of $L$

The final approximated function can be seen in figure 5 with $L$=10 and $\epsilon$=2.0.

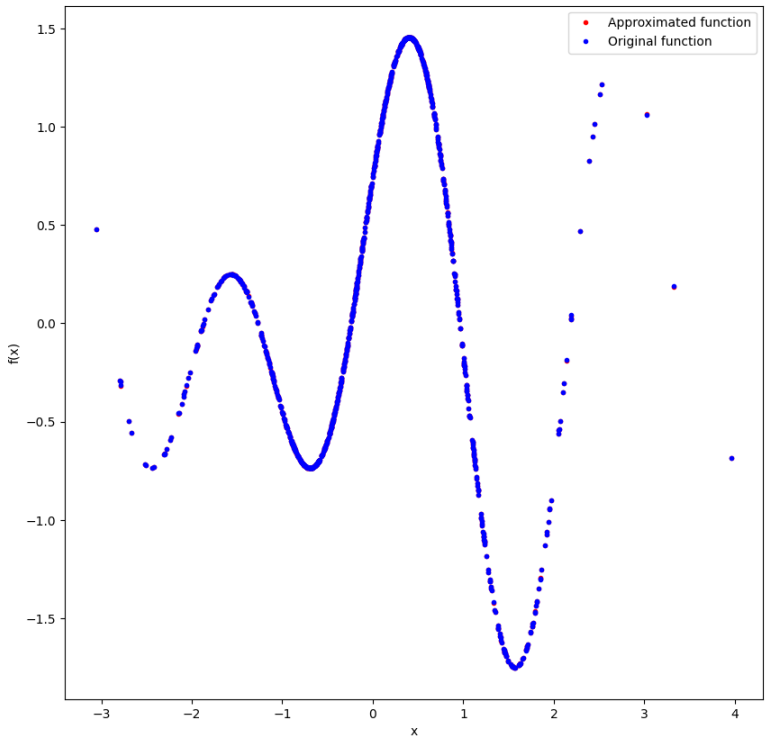



Figure 5: Approximated function for Dataset (B) using radial basis functions

## Report on task 2, Approximating linear vector fields

For this task, we are given two linear vectorfield datasets with 1000 points each. Each row represents a data point, providing the initial position $x_0^{(k)}$ and the corresponding final position $x_1^{(k)}$. The given datasets are plotted in figure 6.

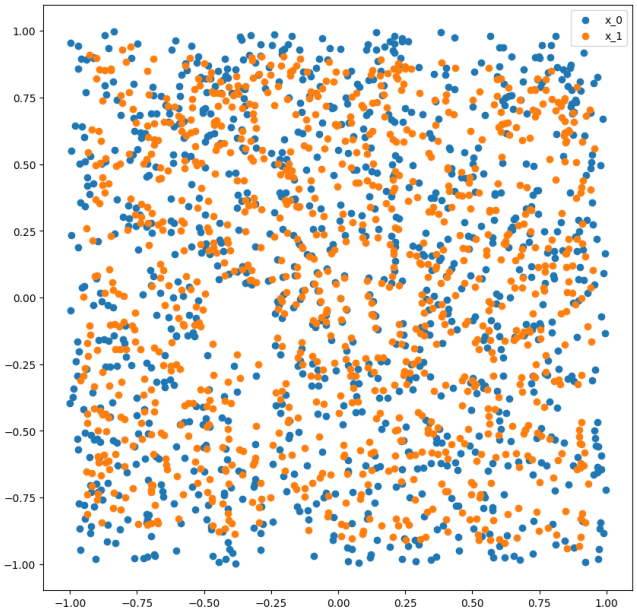



Figure 6: Dataset for task 2

- **Part 1:** For this task, we have to estimate the linear vector field that was used to generate the points $x_1$ from the points $x_0$. Using the finite difference formula with a time step ($\Delta$t=0.1), the vector v_hat is estimated at each point $x_0^{(k)}$. This captures the rate of change in the system. We then employed the least square solution to approximate the matrix A. This matrix captures the linear transformation between the initial positions and the final vectors.

- **Part 2:** For this task, we have to estimate the linear vector field and evaluate the accuracy of our estimated matrix A. We used the `solve_ivp` function to simulate the linear system for each initial point $x_0^{(k)}$ up to the specified end time $T_{end} = \Delta t = 0.1$. We obtained a mean squared error of `0.0030599` providing a measure of the average integration error. The approximated data after $\Delta t = 0.1$ is shown in figure 7.

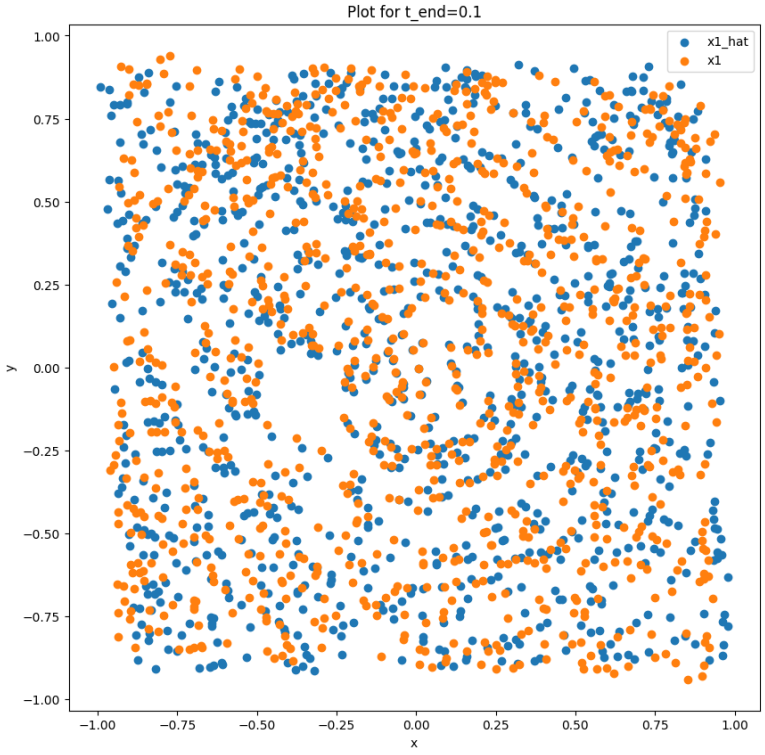



Figure 7: Approximated data after $\Delta t = 0.1$

- **Part 3:** For this task, we first plot the phase portrait of our system and we can see that all values converge to the origin, or in other words we can say that this is the steady point of the system.
We then choose an initial point (10,10), which is far outside our initial data, and solve the linear system with our approximated matrix *A_approximated* for a large time $T_{end} = 100$. We can see that point follows the phase portrait of our system and converges to the steady point at origin.
The phase portrait along with the trajectory can be seen in figure 8.

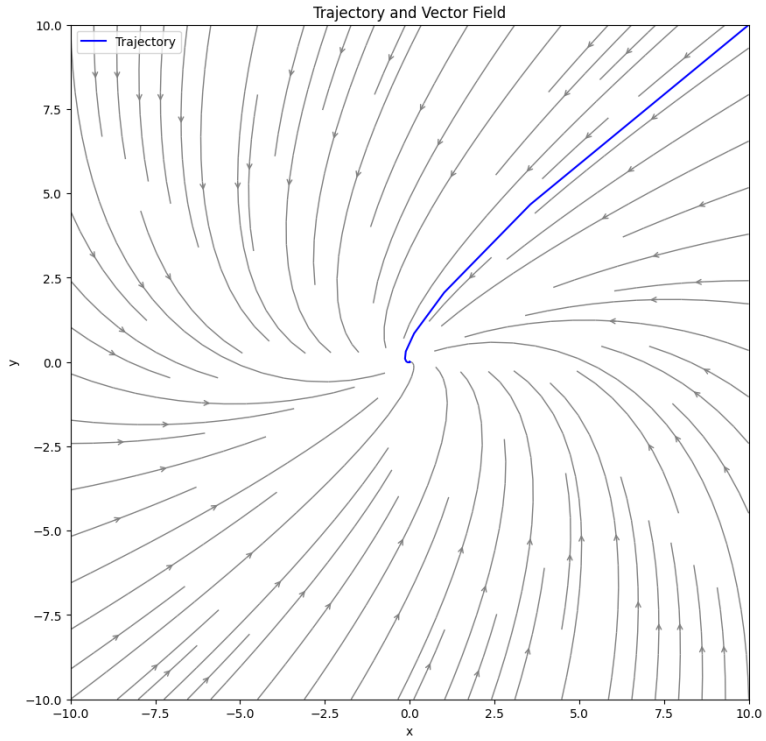



Figure 8: Plotted trajectory for $T_{end} = 100$ and initial point [10,10]

## Report on task 3, Approximating nonlinear vector fields

In task 3, the data files `nonlinear_vectorfield_data_x0.txt` and `nonlinear_vectorfield_data_x1.txt` are used. Both files display the same points. The first file contains the initial points, whereas the second dataset holds the points advanced with an evolution operator. A representation of that data can be found in Fig. 9

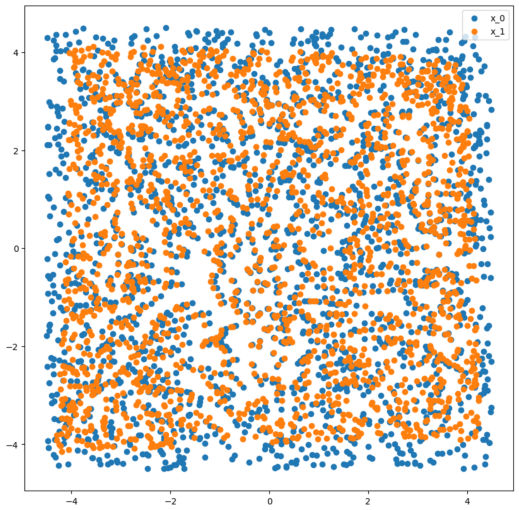
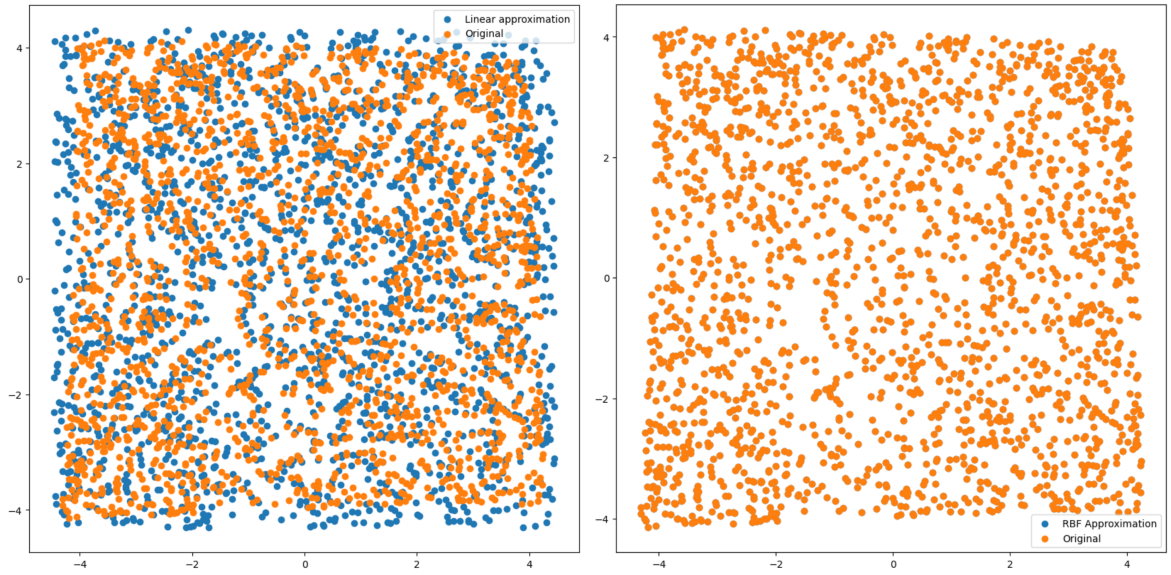



Figure 9: Dataset for task 3

**Part 1: Linear approximation**

In the first part of the task, a linear operator `A` needs to be estimated. This can be done similarly to task 2, using `lstsq` [1]. However, we use a smaller $\Delta t$=0.01. Using the approximated `A` along with the function `linear_approximation`, we can predict values $\hat{x_1}$. In Fig. 10a, the predicted values are plotted against the

original values $x_1$. It is evident that the linear approximation does not reflect the original values well. The image rather resembles $x_0$ and $x_1$ plotted against each other, indicating a weak approximation with almost no effect. The resulting Mean Squared Error (MSE) equals `0.0186`.




(a) Linear approximation (b) Nonlinear approximation

Figure 10: $x_1$ plotted against $\hat{x_1}$

**Part 2: Nonlinear approximation**

To find the optimal MSE for RBF, we looped over multiple $l$ and $\epsilon$ values. The lowest `mse_rbf` value was saved along with the values $l$ and $\epsilon$ that lead to this result:

```
Least mean Squared Error (RBF): 2.898361613586458e-12 reached for:  l=1000, eps=2
```

Additionally, we plotted a graph to showcase how the $l$ and $\epsilon$ values affect the `mse_rbf` outcome. This can be seen in Fig. 11. The z-axis displays the resulting `mse_rbf` values, which all are in the $e^{-12}$ range.

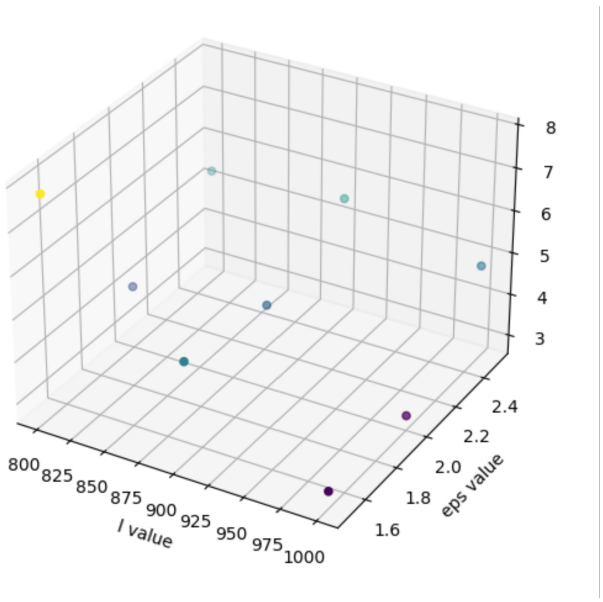



Figure 11: Different $l$ and $\epsilon$ values with their resulting $mse\_rbf$ values

After the optimal values were evaluated, we employed them to calculate a second prediction using the function `nonlinear_approximation`. The plotted result can be seen in Fig. 10b. The approximated points lie virtually in the same position as the original $x_1$ values, indicating a very good prediction.

In comparison to linear approximation, the errors for RBF appear much smaller, down to values of $e^{-12}$. Therefore, it can be assumed that the vector field is nonlinear, just like the nonlinear approximation which yields better results.

**Part 3: Steady States**

For the last part of task 3, we use nonlinear approximation since it proved to establish better predictions. Plotting the phase portrait results in Fig. 12. Four points can be identified where the lines converge to. These points are the steady states of the system.

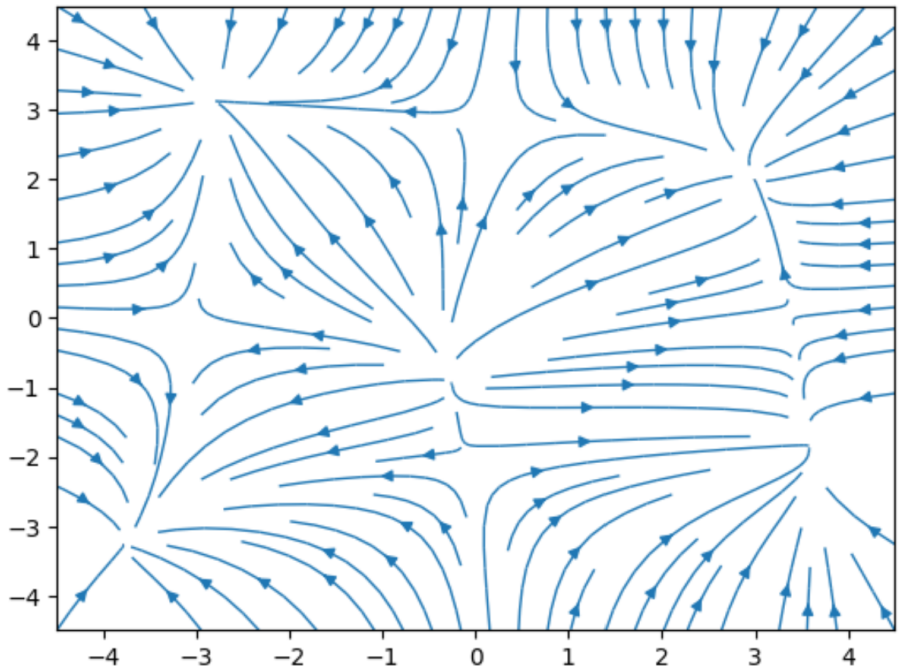
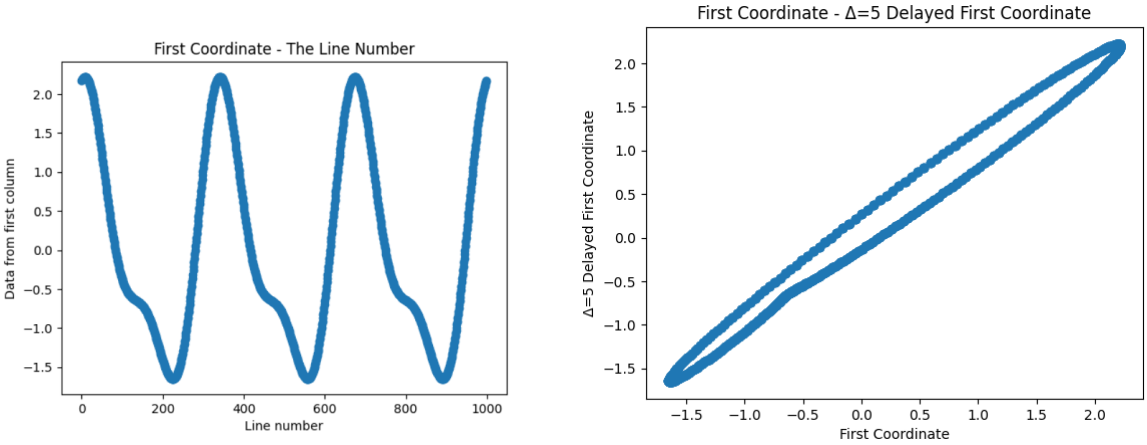



Figure 12: Phase portrait depicting steady states

A linear system with a matrix A of size 2×2 can have 2 steady states at most. Therefore, the nonlinear approximated system cannot be topologically equivalent to a linear system.

## Report on task 4, Time-delay embedding

In Figure 13a, plotting the number of lines against data from the first column reveals a fluctuating curve. In Figure 13b, the data from the first column is graphed against itself with a 5-unit delay.

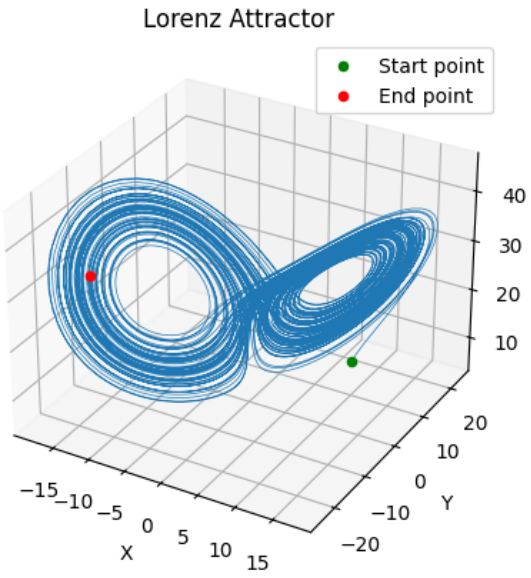



(a) First Coordinate - The Line Number (b) First Coordinate - Δ=5 Delayed First Coordinate

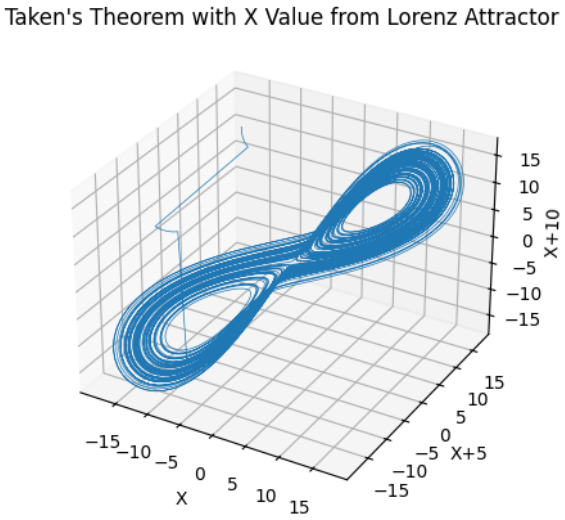
Figure 13

According to the Time-delay embedding explanation in the exercise sheet, for a 1-dimensional manifold, we require **3 coordinates** (d=1 from $2d + 1$) to ensure accurate embedding of the manifold. However, in Figure 13b with our current graph and $\Delta = 5$, there is no collapse, indicating that it can be considered correctly embedded with just 2 coordinates.

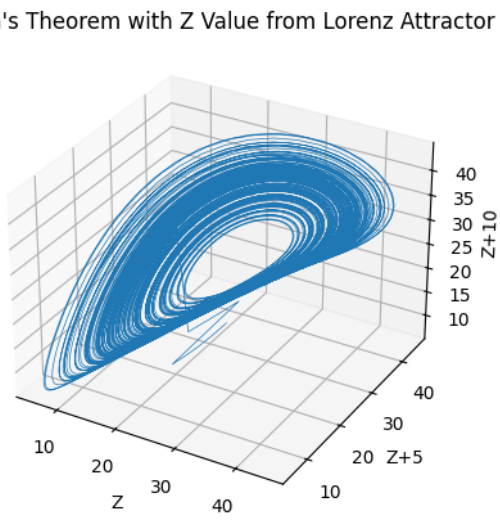
In **part 2** of this exercise, we initiated the Lorenz Attractor as implemented in Exercise 4, with its graph displayed in Figure 14a. The values are stored in the variable `coordinates`. For Figure 14b, we utilized the x coordinate from this variable. Shifting x values by 5 units for the y-axis and 10 units for the z-axis resulted in a similarly shaped curve, affirming the correctness of Takens' theorem. However, when using z values from `coordinates`, the shape diverges from Figure 14a. Collapsation of two circles, originally distinct in 13b, indicates an embedding failure.




(a) Lorenz Attractor which was in the Exercise 4




(b) Using only X values from Lorenz Attractor




(c) Using only Z values from Lorenz Attractor

Figure 14

---

**Report on task 5, Learning crowd dynamics**

In this task, we will analyze the dataset `MI_timesteps.txt`, which contains data from the TUM Garching campus. The dataset named `MI_timesteps.txt` covers seven weekdays and records the count of people in nine different campus areas (PID) at various times. The data, arranged in columns, is depicted in the accompanying figure (15). This time series dataset exhibits cyclical patterns, evident from multiple peaks and recurring trends.

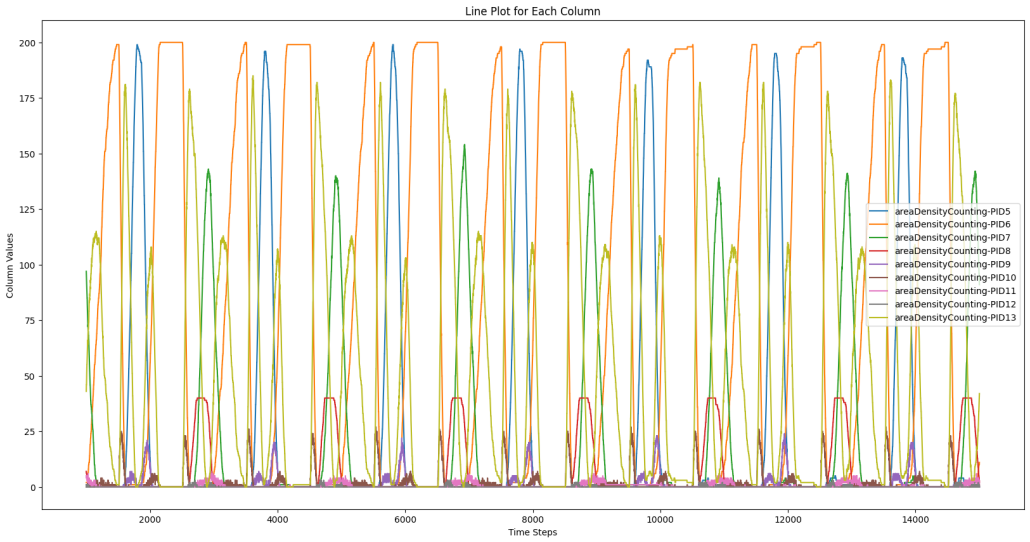


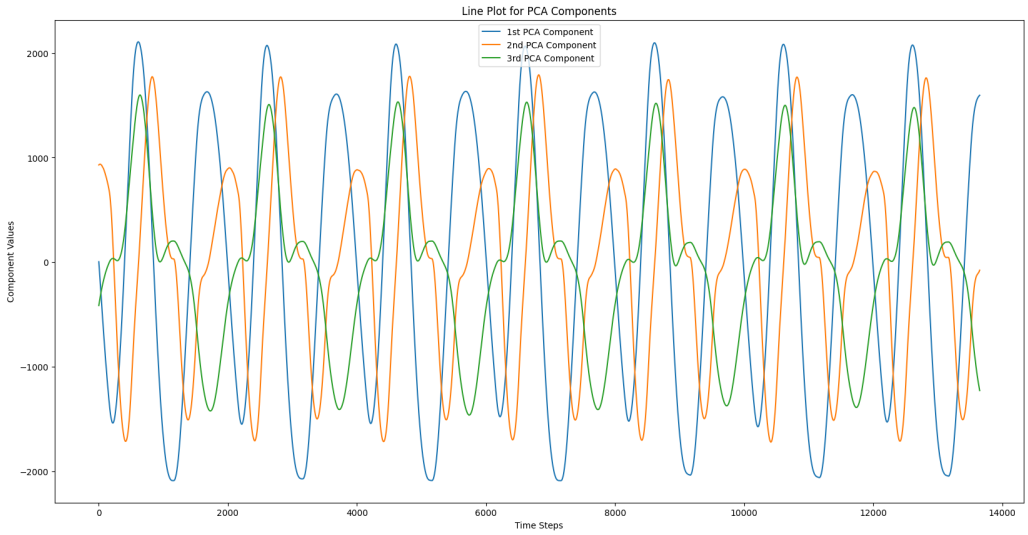

Figure 15

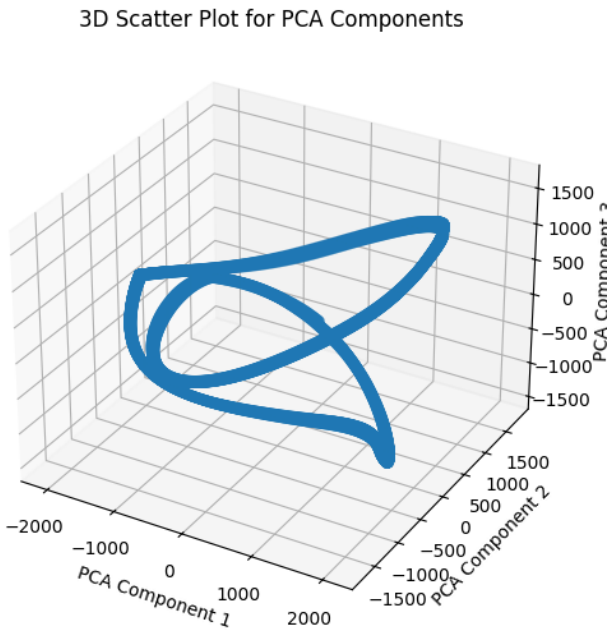
Our goal is to analyze this data to predict the utilization of the MI building. We aim to learn the dynamics of the periodic curve embedded in the principal components and forecast the MI building's utilization for the upcoming 14 days. Also, the first 1000 rows of the dataset have been discarded since they are regarded as a burn-in period by the exercise sheet.

- **Part 1: Takens theorem, delay embedding, PCA.**
  In Part 1, we address the task of determining the appropriate number of dimensions for embedding our dataset according to Takens' Theorem.

  For this, we consider the dataset as a one-dimensional, periodic time series, with no apparent parametric dependencies between different time points. According to Takens' theorem, for a one-dimensional manifold, the embedding dimension required is $2d + 1$, where $d$ is the dimensionality of the original system. Since our system is one dimensional $d = 1$, the embedding dimension needed is $2 \times 1 + 1 = 3$. However, our task specifies using windows of 351 data points across 3 measurements, resulting in vectors of 1053 dimensions. This is far above the minimum required by Takens' theorem and thus provides more detailed insights into the system's dynamics. Consequently, we will use three principal components after performing PCA.

  In the process of creating a delay embedding with 350 delays of the first three measurement areas, we defined a function `create_delay_embedding` that takes the dataset and a delay parameter as inputs. This function constructs a delay embedding matrix by calculating the necessary number of rows for the embedded data, considering the specified delay. An empty matrix `embedded_data` is initialized. The number of columns in this matrix is (delay + 1) * 3, which indicates our consideration of three columns from the original dataset for embedding. Next, a loop runs over the dataset, and for each iteration, a flattened portion of the data (spanning the current index i to i + delay + 1 and considering three columns) is added to the `embedded_data_matrix`. This creates a sequence of delayed observations in the dataset. After that we applied this embedding function to the data with a delay of 351, resulting in a new matrix `window_matrix`. The shape of this matrix indicates the dimensions of the embedded data.




- **Part 2:**
  The 3D scatter plots in figure 16 are generated by coloring the points by the first coordinate of the delay embedding. Each point on the scatter plot represents a data sample with its corresponding values on the first three principal components resulting from a PCA on the delay embedded data.

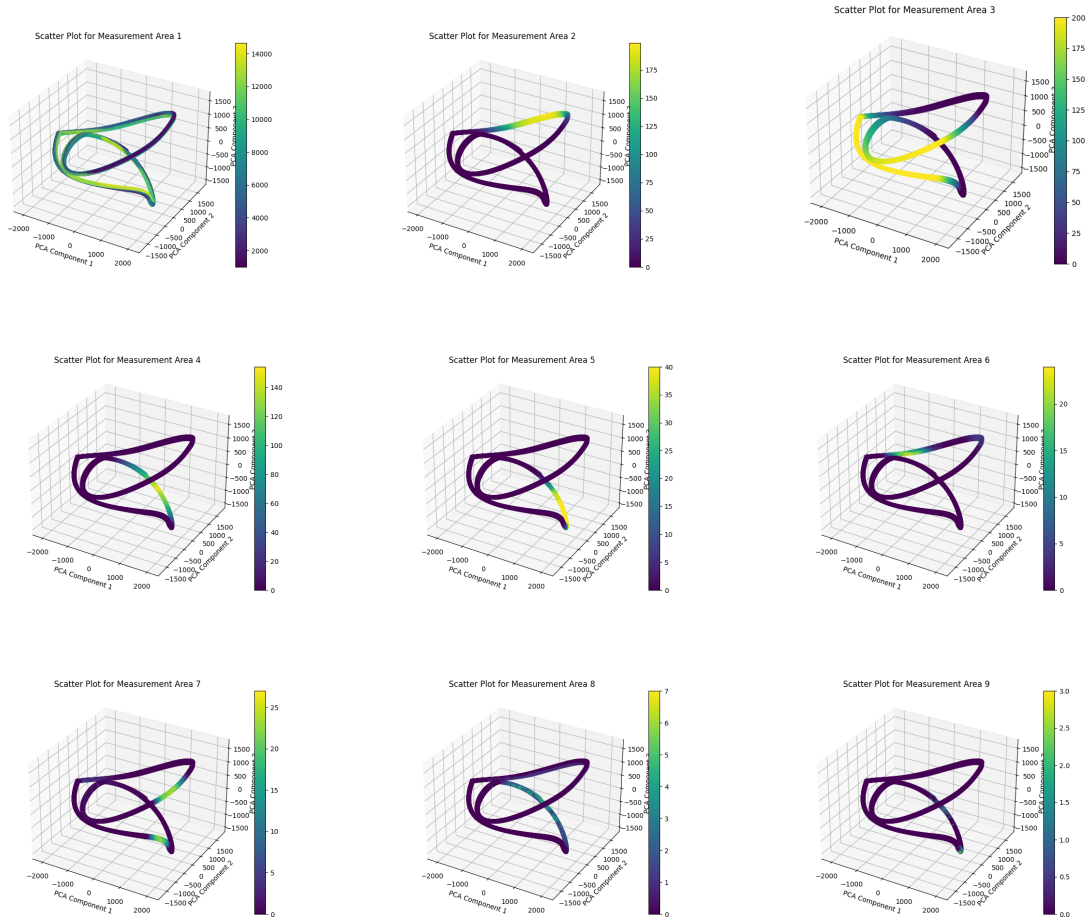
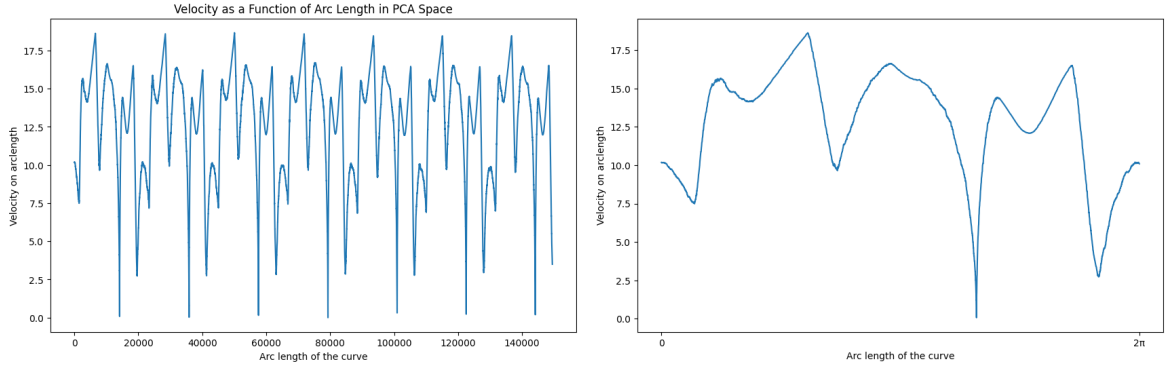



Figure 16

PCA is employed to capture the most significant patterns in the dataset by reducing its dimensionality. The three axes—PCA Component 1, PCA Component 2, and PCA Component 3—represent the directions with the highest variance in the data, which are crucial to understanding the dynamics of campus utilization. We can observe that all the points are in the same position only the color changes, The color scale in each plot corresponds to the first coordinate of the delay embedding, which is indicative of the sequential time at which the data was recorded. The gradation of colors provides insights into the temporal progression of the density count.

- **Part 3:**
  For this task, we have to learn the dynamics of the periodic curve that we already embedded in the principal components of the previous task. For this, we first extract the time steps present in the first column of the dataset. Then we calculate the distances between consecutive points (`deltas`) of the PCA space and use `np.linalg.norm` over the found deltas to compute the arclengths. We store the cumulative sum at each point using the `np.cumsum` method.
  Then, we calculate velocities or the rate of change of arclengths over time. We plot this velocity vs the cumulative arclength as seen in figure 17a. The periodic nature of the dataset becomes apparent in this plot. To gain a more detailed understanding, we can trim one complete period of the data and visualize it separately, as illustrated in figure 17b.




(a) Plot for the entire dataset

(b) Plot for one period

Figure 17

- **Part 4:**
  For this final task, we had to predict the utilization of the MI building for the next 14 days. To do this, we used the periodic feature of the dataset and extrapolated our calculated velocities to a period of 14 days. This extension enabled us to understand the system's dynamics over a more prolonged timeframe. We then need to calculate the arclength over this period of 14 days. To do this, we calculated the cumulative sum of the extrapolated velocities and extended time intervals embedded within the principal component. This allowed us to project the system's behaviours over a period of 14 days. Now, to model the relationship between arclength and the measurement parameter, we utilized the radial basis function (RBF) approximation implemented in task 1. The least square minimization technique was used to derive optimal coefficients for the RBF model. The final predicted vs original plot can be seen in figure 18.

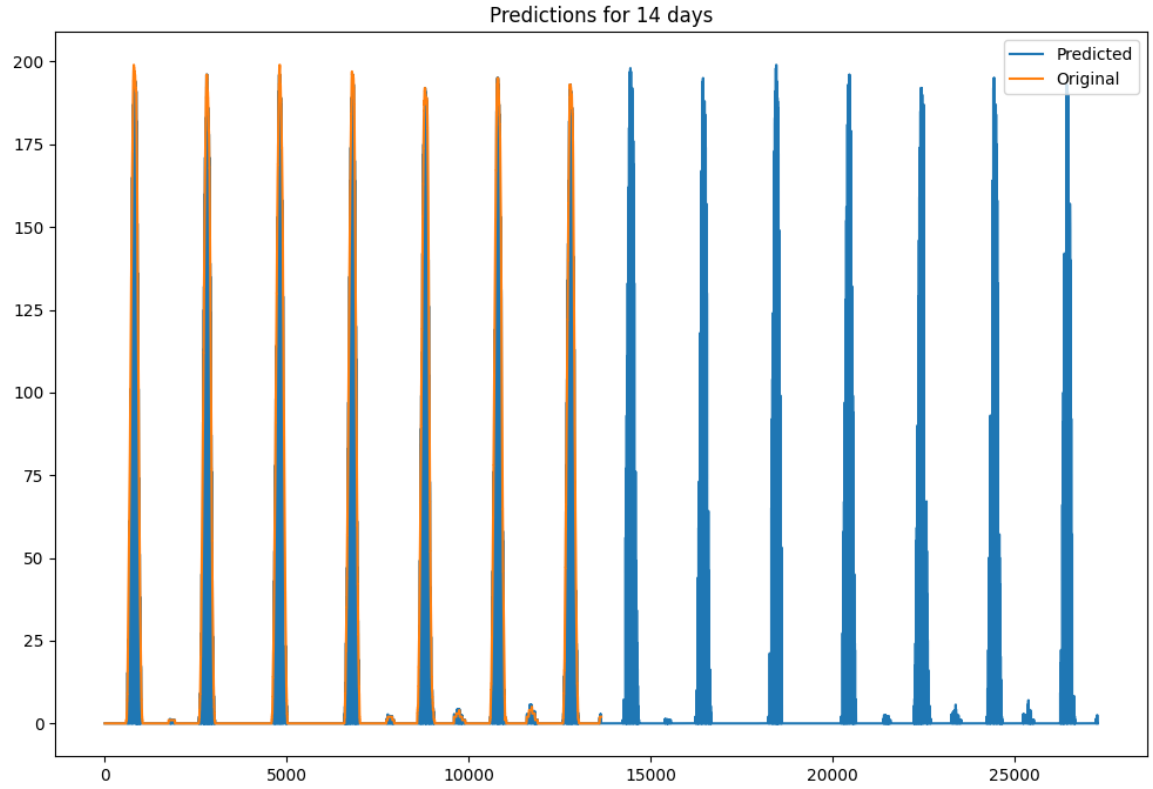



Figure 18: Predictions for 14 days

# References

[1] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.