Final Exam Project Foundations of Data Science: Programming and Linear Algebra

(CDSCO1001U.LA_E20)

- REPORT -

Handed in by: Luca Ludwig (149890), Konrad Heinrich Schulte (149872), David Trotzky (149871) & Chiara Fiorio (149874)
Supervisors:
Number of Characters: 11871
Number of Pages: 6

Table of Contents

Overview

During our previous exercises and assignments, we wished for an efficient verification tool for matrix calculations. We, therefore, decided to build one ourselves: We developed a user-friendly programme that enables students to check their calculations.

Our program can perform the following calculations for any matrices:

- multiplication of two matrices
- addition of two matrices
- subtraction of two matrices
- calculation of a matrix inverse
- calculation of a matrix determinant and
- calculation of a matrix eigenvalue and eigenvector.

Maybe our tool will help next year's MSc Data Science students with their assignments and enable them to gain a solid understanding of matrix calculations? ☺

Assumptions

We only hold one assumption for our programme: We assume that the only case, in which we have one eigenvalue and two eigenvectors, is the case in which the matrix inserted by the user is an identity matrix.

Design Choices

We made the following design choices to ensure clarity and minor complexity for the user:

- We used a Graphical User Interface (GUI) to make our tool more convenient. The first version of our project was a python programme that required user input. We then decided that a GUI would be the better choice, to provide our users with a clear interface.
- We made use of NumPy to create arrays in our calculation code because it made the coding process significantly easier and less complex. The alternative would have been to use lists, which requires a lot of extra coding (e.g., looping through the lists to find specific values).
- In order to reduce the complexity of our tool, we chose to provide the eigenvector and eigenvalue calculations only for 2X2 matrices.

- To ensure the clarity of the user interface, we limited the input option for all other calculations to 10x10 matrices.
- Even though we had to compromise on the aesthetics of our user interface, we chose to not round the numbers to get the highest level of mathematical correctness and certainty.
- Moreover, we chose not to provide examples in this report as they cannot be copy-pasted into the GUI at once and it should be easy to quickly come up with test and/or use cases for the user.

Structure

Our programme can be subdivided into 6 different parts:

1. Main window
2. Multiplication window
3. Addition and subtraction window
4. Determinant window
5. Inverse window
6. Eigenvalue and eigenvector window

The functionality of these subparts of our programme is explained in the next section.

Functionality

Functionality of "Main window"

The main window is the center of the application and the first displayed window after running the code. We define the layout of this window by adding the title "Matrix Calculations", the information "Please choose a Calculation Type", and the buttons for calculating addition and subtraction, multiplication, determinant, inverse, and Eigenvalue and Eigenvector. The main window runs until the application is closed and listens to events in order to open the windows corresponding to the chosen calculation.

Functionality of "Multiplication window"

We first define the GUI windows for our input matrices M1 and M2 as well as for our output matrix M3. Moreover, we define the layout of the whole "Multiplication window", which includes the 3 matrices as well as a "Compute Multiplication" button and multiple text fields (e.g. for error messages).

The number of rows and columns of our input matrices as well as the respective values within M1 and M2 get defined by user input. The maximum size of the matrix is 10x10 which leads to the maximum number of values being 100 per matrix.

We make sure that our result matrix M3, as well as the "Compute Multiplication" button, are hidden when the user misses inserting single values of the input matrices or changes the number of rows, the number of columns or the single values.

We add error messages (shown in the GUI) for the following cases:

- If the columns of M1 and rows of M2 are not matching (necessity for multiplication)
- If not all values for M1 and M2 have been inserted by the user
- If not all values are floats or integers

We use try / except to show the "Compute Multiplication" button once the user followed all the instructions.

In order to compute the actual multiplication, we first transfer all the values from our two input matrices to a list of lists (each row is one list). We then define the dimensions of our output matrix (Rows of M1 x Columns of M2). To calculate the values of M3 we multiply the indices of a specific M1 row and the respective M2 column and store the result in a temporary variable. These single values are then stored in a list for the specific row in M3. Finally, we store all rows in the list for M3 and show the calculated matrix in the GUI.

With if / break we make sure that our program ends once the user closes the window.

Functionality of "Addition and subtraction window"

We first define the GUI windows for our input matrices M1 and M2 as well as for our output matrix M3. Moreover, we define the layout of the whole "Addition / Subtraction window", which includes the 3 matrices as well as a "Compute Addition" or "Compute Subtraction" button and multiple text fields (e.g. for error messages).

The number of rows and columns of our input matrices as well as the respective values within M1 and M2 get defined by user input. The maximum size of the matrix is 10x10 which leads to the maximum number of values being 100 per matrix.

We make sure that our result matrix M3, as well as the "Compute Addition" and "Compute Subtraction" button, are hidden when the user misses inserting single

values of the input matrices or changes the number of rows, the number of columns or the single values.

We add error messages (shown in the GUI) for the following cases:

- If M1 and M2 have a different number of rows and/or columns (necessity for addition/subtraction)
- If not all values for M1 and M2 have been inserted by the user
- If not all values are floats or integers

We use try / except to show the "Compute Addition" or "Compute Subtraction" button once the user followed all the instructions.

In order to compute the actual addition or subtraction, we first transfer all the values from our two input matrices to a list of lists (each row is one list). We then define the dimensions of our output matrix (Rows of M1 x Columns of M2 – this could be even the other way around as the number of rows and columns of both matrices must be equal).

To calculate the values of M3 we add or subtract the values of M1 with the respective value of M2 and store the result in a temporary variable. These single values are then stored in a list for the specific row in M3. Finally, we store all rows in the list for M3 and show the calculated matrix in the GUI.

With if / break we make sure that our program ends once the user closes the window.

Functionality of "Determinant window"

We define the sub window for determinant calculation which contains the input matrix M1, the button "Compute Determinant", and text fields for information, error handling, and the result. The input matrix is limited to 10x10 entries, resulting in a maximum of 100 values.

The number of columns and rows can be changed by adjusting the sliders at the top of the window. In case of a displayed result, the result gets hidden if the number of rows and/or columns is changed after calculating the result.

We add error messages (shown in the GUI) for the following cases:

- If M1 has a different number of rows and/or columns (square matrix necessary for calculating determinant)
- If not all values for M1 have been inserted by the user
- If not all values are floats or integers

We use try / except to show the “Compute Determinant” button once the user followed all the instructions.

In order to compute the determinant, we first transfer all the values from the input matrix to a list of lists (each row is one list). We then use a while loop to transform the matrix into Echelon form which runs until Echelon form is reached. With the matrix being in echelon form, we can calculate the determinant. After finishing the calculation, the result is updated and made visible to the user.

With if / break we make sure that our program ends once the user closes the window.

Functionality of “Inverse window”

We define the sub window for inverse calculation which contains the input matrix M1, the button “Compute Inverse”, the output matrix M3, and text fields for information and error handling. The input matrix is limited to 10x10 entries, resulting in a maximum of 100 values.

The number of columns and rows can be changed by adjusting the sliders at the top of the window. In case of a displayed result, the result gets hidden if the number of rows and/or columns is changed after calculating the result.

We add error messages (shown in the GUI) for the following cases:

- If M1 has a different number of rows and/or columns (square matrix necessary for calculating inverse)
- If not all values for M1 have been inserted by the user
- If not all values are floats or integers
- If M1 is singular and thus not invertible

We use try / except to show the “Compute Inverse” button once the user followed all the instructions.

In order to compute the inverse, we first transfer all the values from the input matrix to a list of lists (each row is one list). We then use a while loop to transform the matrix into Echelon form which runs until Echelon form is reached. Next, we calculate the determinant and append the identity matrix to the matrix in echelon form. We then use the algorithm to transform the original matrix to the identity matrix, which we learned in class. The resulting matrix M3 is the altered identity

matrix on the right side. After finishing the calculation, the result is updated and made visible to the user.

With if / break we make sure that our program ends once the user closes the window.

Functionality of "Eigenvalue and Eigenvector window"

We define the sub window for Eigenvalue and Eigenvector which contains the input matrix M1, the button "Compute Eigenvalue(s) and Eigenvector(s)", and text fields for information, error handling, and the result. The input matrix is limited to 2x2 entries, resulting in a maximum of 4 values. This limitation is done due to the increased complexity of calculating Eigenvalue and Eigenvector for bigger matrices.

We add layouts for different cases. In the case of one unique Eigenvalue (multiplicity of 2), we display only one Eigenvalue with its corresponding Eigenvector. In the case of two Eigenvalues, we display two Eigenvalues with their corresponding Eigenvectors. In the case of an identity matrix (one Eigenvalue with multiplicity of 2 but two Eigenvectors), we display only one Eigenvalue with its two corresponding Eigenvectors.

We add error messages (shown in the GUI) for the following cases:

- If not all values for have been inserted by the user
- If not all values are floats or integers
- If no real solution exists
- If rows are multiples of each other

We use try / except to show the "Compute Eigenvalue(s) and Eigenvector(s)" button once the user followed all the instructions.

In order to compute the Eigenvalue and Eigenvector, we first transfer all the values from the input matrix to a list of lists (each row is one list). We then calculate the Eigenvalues and check for negative values inside the root. If this is the case, there is no real Eigenvalue. We cover the edge case of the matrix being a multiple of the identity matrix which results in only one Eigenvalue with multiplicity two but two different Eigenvectors. In the case of different Eigenvalues, we calculate two Eigenvectors. If the Eigenvalues are identical, we only calculate one Eigenvector. After finishing the calculation, the result is updated and made visible to the user.

With if / break we make sure that our program ends once the user closes the window.