

School of Economics and Management of the University of Porto

Master in Modelling, Data Analysis and Decision Support Systems

Data Bases and Programming

Prof. Dr. Rui Leite

Gauss Elimination Algorithm

Group 9:

Marta Gonçalves

Maria Inês Tavares

Tim Luca Bernstiel

Contents

[Introduction 3](#_Toc120973491)

[Gauss Elimination Algorithm 3](#_Toc120973492)

[Forward Elimination 3](#_Toc120973493)

[Backwards substitution 4](#_Toc120973494)

[Implementation in Python 5](#_Toc120973495)

[Extras 6](#_Toc120973496)

[Graphical User Interface 6](#_Toc120973497)

[Conclusion 6](#_Toc120973498)

# Introduction

To solve a system of liner equations, the gaussian elimination is the most prestigious method.

# Gauss Elimination Algorithm

The gauss elimination algorithm is the transformation of the gaussian elimination into a computational procedure. The system of linear equations is displayed as a matrix, where is defined as the number of unknown variables present in the system (). Additionally, there is a right side which holds the result of each equation. In the end the gaussian elimination algorithms give the solution to the problem by calculating the correct values for all unknowns to solve the entirety of the linear equations.

The algorithm consists of two steps. Forward elimination is designed to bring the present matrix into a special form where there is a triangle of zeros in the bottom left of the matrix. This gives a set of equations which increasingly eliminate the number of unknowns by having a factor of zero. With the help of backwards substitution those newly formed linear equations getting solved by calculating the unknowns.

## Forward Elimination

As stated before, forward elimination transforms the matrix into a different state to make it possible to calculate the values for all present unknowns. Figure 1 gives an example of the required state of the matrix. The left matrix is the given system of linear equations, and the right matrix is the required state to solve the system. As highlighted in bold font style there is a triangle at the bottom left of the matrix only consisting of zeros. This allows the immediate calculation of , since the factors for and are zero, which results in them not affecting the third linear equation.

Figure : Transformed matrix state

Between those two states there are a couple of mathematical operations required. Figure 2 shows the operation diagram used for forward elimination. The input is defined as the matrix of variable coefficients , the number of unknowns present in the system and the right side containing the solutions of the equations .



Figure : Operation diagram forward elimination

After the execution of the operations flow shown in Figure 2, a transformed matrix in the same state as shown in Figure 1, as well as the updated right side results. The combination of those two are required to calculate the final values for the unknowns. This can be achieved by using backwards substitution presented in the following chapter.

## Backwards substitution

Backwards substitution gives the method to obtain the final values for the unknowns and therefor the solution to the system of linear equations. It requires the transformed matrix and right side acquired through forward elimination in the step before. Figure 3 presents the operation flow diagram for obtaining the solution vector containing values for each unknown.



Figure : Operation diagram backwards substitution

The solution vector contains the values for all the unknowns in the linear equation system.

# Implementation in Python

To implement the Gaussian Elimination an object-oriented approach has been chosen. The class GaussElimination contains all required methods. The class requires for initialization the matrix as well as the right side. Since python is not a typesafe programming language the type annotations bring a lot value to cooperating developers. While the matrix is a list containing lists of float values, the right side is a basic list vector containing float values as well. During initialization also the amount of unknows is being calculated by counting the columns of the first row of the matrix (which holds the first linear equation). The variables are being stored, so all methods of the class can access them.

class GaussElimination:

    def \_\_init\_\_(

        self,

        default\_matrix: List[List[float]] = None,

        right\_side: List[float] = None

    ) -> None:

        self.\_default\_matrix = default\_matrix

        self.\_right\_side = right\_side

        self.\_n = len(self.\_default\_matrix[0])

Figure : Initialization of class GaussElimination

To solve the saved system of linear equations the definition shown in Figure 5 initiates the process. The method takes an optional parameter called algorithm which basically could select the used algorithm in case multiple ones are being implemented. For now, only the combination of forward elimination and backwards substitution presented in Chapter 2 is available.

def solve(self, algorithm: str = "forward\_elimination"):

        if algorithm == "forward\_elimination":

            res = self.forward\_elimination\_back\_substitution(

                self.default\_matrix, self.right\_side)

            return res

        else:

            raise ValueError("Invalid algorithm!")

Figure : solve function

The solve function presented in Figure 5 calls the corresponding process method by passing the saved matrix and right side. Then the method shown in Figure 6 first converts the matrix and right side lists into numpy arrays. In class it was presented as the faster pendant to the normal python lists. Then the forward elimination method is being called. The numpy transformed matrix and right side being passed.

    def forward\_elimination\_back\_substitution(

        self,

        matrix: List[List[float]],

        right\_side: List[float]

    ):

        np\_matrix = self.convert\_matrix\_to\_numpy\_array(matrix)

        np\_right\_side = self.convert\_right\_side\_to\_numpy\_array(right\_side)

        np\_matrix, np\_right\_side = self.forward\_elimination(

            np\_matrix, np\_right\_side)

        solution = self.back\_substitution(

            np\_matrix, np\_right\_side)

        return solution

Figure : Process method for forward elimination and backwards substitution

Figure 7 shows the python equivalent to the operation flow diagram presented in Chapter 2.1. After all operations taken it returns the updated matrix and right side. The method accesses the in the initialization saved number of unknowns by calling “self.\_n”.

def forward\_elimination(

        self,

        matrix: np.ndarray,

        right\_side: np.ndarray

    ) -> Union[np.ndarray, np.ndarray]:

        for i in range(self.\_n - 1):

            for j in range(i + 1, self.\_n):

                m = matrix[j][i] / matrix[i][i]

                right\_side[j] -= m \* right\_side[i]

                for k in range(i + 1, self.\_n):

                    matrix[j][k] -= m \* matrix[i][k]

        return matrix, right\_side

Figure : Forward elimination method

After successful forward elimination the updated matrix and right side being passed to the backwards substitution shown in Figure 8. It begins by creating a vector of size containing dummy zeros. Then it follows the operations flow diagram presented in Chapter 2.2. It only returns the computed vector containing the solutions for each unknown.

def back\_substitution(

        self,

        matrix: np.ndarray,

        right\_side: np.ndarray

    ) -> np.ndarray:

        x = np.zeros(self.\_n)

        x[self.\_n - 1] = right\_side[self.\_n - 1] / \

            matrix[self.\_n - 1][self.\_n - 1]

        for i in range(self.\_n - 2, -1, -1):

            temp = right\_side[i]

            for j in range(i + 1, self.\_n):

                temp -= matrix[i][j] \* x[j]

            x[i] = temp / matrix[i][i]

        return x

Figure : Backwards substitution method

This solution could either be printed to the console or passed to any other interface.

# Extras

## Graphical User Interface

In order to present the user a better experience a graphical user interface (GUI) has been developed. There are a variety of open-source projects with the aim to give an accessible option to build quick interfaces. In this case the decision fell on PySimpleGUI which is a wrapper for a couple of other popular GUI libraries. Figure 9 shows a snapshot of the program. To interact with the GUI 180 lines of code have been developed.

The user can enter the wished number of unknowns in the system. Accordingly, to the user input the matrix changes its dimensions. The matrix is being saved after each change, so proceedings are not being lost when transforming the dimensions. After submitting the problem, the GUI methods access the GaussElimination class presented in Chapter 3 and gives the user the computed solution. There are also some error messages implemented to provide a better user experience.

Graphical user interface, application

Description automatically generated

Figure : Graphical User Interface for easier access to the Gauss Elimination class

# Conclusion