Computing Exercise 1: Investigating the use of matrices in solving simultaneous equations

Saskia Jackson

July 20, 2020

Introduction

Simultaneous equations are a mathematical concept that occur in physics over and over again. Therefore, an efficient and accurate method for solving such equations is of great importance. This computational exercise is aimed at comparing various methods for solving simultaneous equations and testing them for accuracy and efficiency.

The majority of the methods for solving simultaneous equations involve using matrices to represent the equations. A set of simultaneous equations can be described using the following general matrix.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \tag{1}$$

The most basic method of solving this matrix is to rearrange the matrix equation and using the inverse of the matrix to find the unknown, in this case, x.

$$Ax = b \rightarrow x = A^{-1}b \tag{2}$$

The inverse of a matrix can be solved using Cramer's Rule.

$$A^{-1} = \frac{1}{det(C)} C^T \tag{3}$$

In this equation C represents the co-factor matrix is determined by reducing the matrix by the row and column corresponding to its position in the original matrix. An NxN matrix must be reduced N-2 times until a single value for the matrix entry can be determined from a 2x2 matrix.

There are conditions for which the inverse of a matrix cannot be determined and thus the matrix is said to be singular. This occurs when the determinant of the matrix is zero and, as the division of zero is undefined, the inverse cannot be found using Cramer's rule. The existence of such singular matrices means there are sets of equations that cannot be solved by this method. Additionally, using Cramer's rule to solve simultaneous equations gives rise to a large number of errors. The number of repetitions needed to reduce the matrix to single values introduces many rounding errors that, while individually insignificant, add up to have a significant effect. The repetitive nature of this method is also very time consuming.

It is easier to determine the inverse of a matrix if it can be reduced to a triangular matrix. This method used in LU decomposition where the matrix is split into an upper (U) and lower (L) triangular matrix. For an original matrix A, $PA = LU$ and therefore $A^{-1} = L^{-1}U^{-1}$, where P is the permutation matrix and originates from the how the original matrix A is altered to find the triangular matrices. As triangular matrices are easier to invert, this cancels out the error gained in multiplying the L and U matrices together.

Another alternative method is Single Value Decomposition, in which the matrix is split up into three matrices with different dimensions.

$$A = U\Sigma V^T \rightarrow A^{-1} = V\Sigma^+ U^T \tag{4}$$

To find A^{-1}, U and V can be reversed as they are orthogonal. Σ is a diagonal matrix made up of the singular values of the original matrix A and Σ^+ is determined by taking the inverse of the non zero elements of the diagonal. This reduces the number of matrices which need to be inverted and therefore improves the efficiency of the method.

Part One: Calculating the inverse of a matrix

This section of the exercise was simply to determine the inverse of an N x N matrix and then use Cramer's Rule to solve the simultaneous equations.

A function needed to be written to reduce the matrix down to a singular value so that both the determinant and matrix of co-factors could be determined. The NumPy delete function could be used in a for loop to eliminate entire rows and columns to reduce the matrix. This method finds the matrix of minors and has to be multiplied by the checkerboard matrix of positives and negatives to find the matrix of co-factors. This could be replicated in the code by using $(-1)^{(i+j)}$ where i and j represent the position in the matrix.

A random matrix could be generated using the random.randint() function which can be applied to an array to give a random matrix. Additionally the time the program takes to run can be measured using the time.time() function. Therefore the time taken to invert a random matrix can be plotted against the dimensions of the matrix.

The error accumulated when inverting the matrix can be determined by calculating $A^{-1}A - I$ (where I represents the identity matrix). The maximum value in the resulting matrix corresponds to the largest error in the method. This can then be plotted against the dimensions of the matrix.

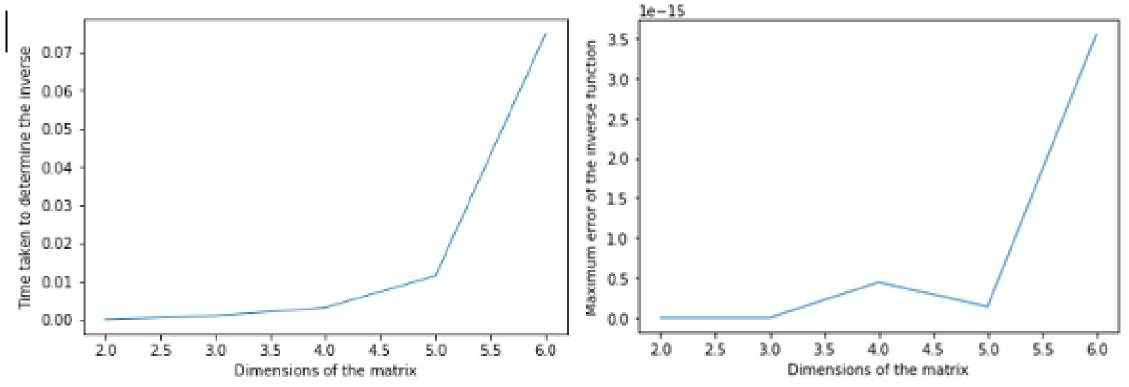

Figure 1: Left: Graph showing how the time to run the program varies with the matrix dimension. Right: Graph showing how the accumulated error varies with the matrix dimension.

As shown in the graphs, both the time taken and the error increase almost exponentially.

Part Two: Comparing different methods for solving simultaneous equations

The objective of this section of the exercise was to compare the three different methods; Cramer's rule, LU decomposition and SV decomposition.

The L, U and P matrices used in LUD can be found using the inbuilt function scipy.linalg.lu() similar to finding the U, Σ and V matrices for SVD using np.linalg.svd().

These can then be inverted, transposed and multiplied accordingly. Matrices can be multiplied using the np.matmul() function.

As was achieved in the first part of the exercise, a graph of the time taken to run the program can be plotted against the dimensions of the matrix to compare the efficiency of the three methods.

In a near singular matrix with a k value in it that is close to zero where that matrix becomes singular, the values of k can be investigated to see how the time changes near to a singular matrix.

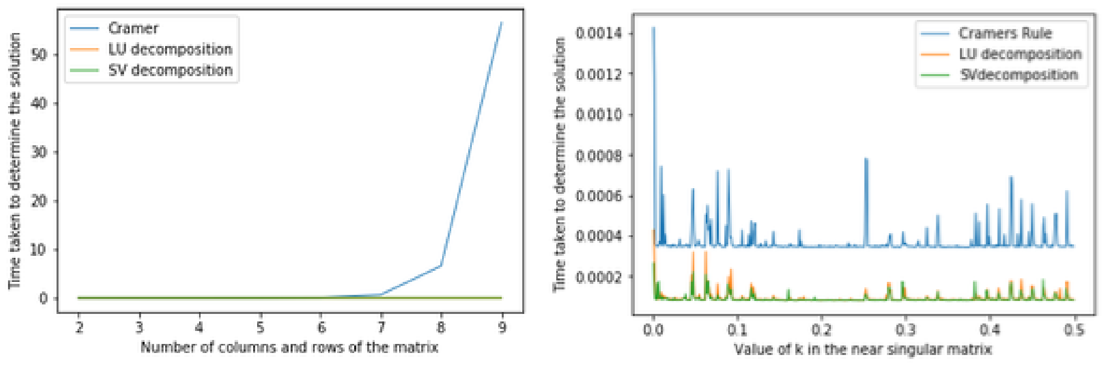

Figure 2: Left: Graph comparing the time taken to run the code compared to the dimensions of the matrix for the three methods. Right: Graph comparing the time taken to run the code for various values of k in the near singular matrix.

Part Three: Trapeze artist on three wires

The third part of the exercise is aimed at using the matrix inversion method applied to a physical problem.

The inbuilt python function np.linalg.inv() can be used to determine the inverse of a matrix to avoid rounding errors and large calculation times.

The physical problem consists of a trapeze artist held by three extendable wires attached to three drums placed around a stage; one in the front left corner, one in the front right corner and one at the back of the stage in the middle.

Initially, when ignoring the wire from the back of the stage, only movements in the y plane should be considered. The tensions in the wires can be represented using the matrix below.

$$\begin{pmatrix} cos\theta_1 & -cos\theta_2 \\ sin\theta_1 & sin\theta_2 \end{pmatrix} \cdot \begin{pmatrix} T_1 \\ T_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 70g \end{pmatrix}$$

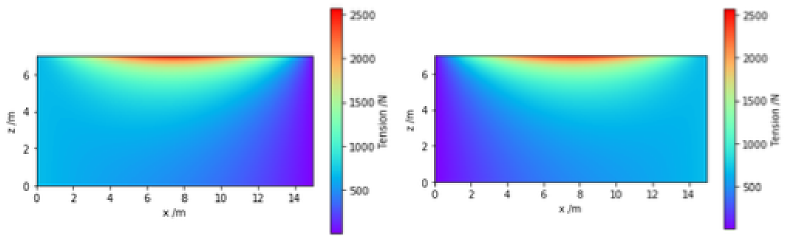

Figure 3: Left: Equation used to determine the tensions in the wires. Middle: Plot showing how the tension varies as the position of the trapeze artists changes on the stage for the wire from the top left drum. Right: Plot showing how the tension varies as the position of the trapeze artists changes on the stage for the wire from the top right drum.

Wire	Max Tension (kN)	Position (x,z) (m)
From the front left	2.57	(7.43, 6.99)
From the front right	2.57	(7.57, 6.99)

Table 1: The maximum tensions and their positions for the two wire problem.

Upon adding the third wire, three simultaneous equations are needed to describe the tensions in the wires. These equations can be represented using the following matrix:

$$\begin{pmatrix} -sin\theta_1 cos\alpha & sin\theta_2 cos\beta & \pm sin\phi cos\gamma \\ -sin\theta_1 sin\alpha & -sin\theta_1 sin\beta & sin\phi sin\gamma \\ sin\theta_1 & sin\theta_2 & sin\phi \end{pmatrix} \cdot \begin{pmatrix} T_1 \\ T_2 \\ T_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 70g \end{pmatrix} \tag{5}$$

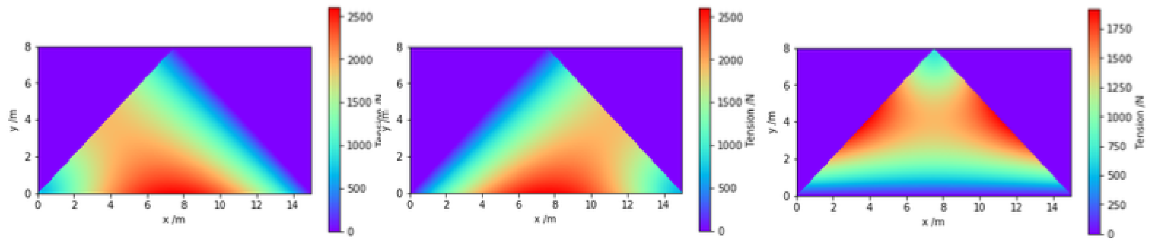

Figure 4: Plots showing how the tension in the wires change as the position of the trapeze artist changes. All for a constant height of 7m above the ground. Left: the wire from the top left drum, Middle: the wire from the top right drum. Right: the wire form the drum at the back of the stage.

Wire	Max Tension (kN)	Position (x,y) (m)
From the front left	2.57	(7.4, 0.1)
From the back of stage	1.90	(3.8, 4.0)

Table 2: The maximum tensions and their positions for the three wire problem.

Discussion

The code used to calculate the inverse of a matrix using Cramer's rule automatically uses the top row to determine the determinant of the matrix. This might not always be the most efficient method as if one of the rows has multiple zeros, it reduces the amount of work needed to determine the inverse. Therefore, with more time, some code could be added to scan the matrix for the largest amount of zeros in a row or column and choose this row or column accordingly.

This would also have an affect on the graphs of time taken and maximum error against matrix dimensions. This is because the matrix used is generated randomly, thus as some matrices will have larger errors and take a larger amount of time, this creates an error in the comparison.

In general the Cramer's rule takes much longer and has a much larger error compared to LUD and SVD. This is most likely due to the repetitive nature of the method while reducing the matrix smaller N-2 times. This also adds rounding errors to the final error.