Exercise 1: Matrices and Linear Algebra

Oliver Sellers

University of Bristol

April 26, 2020

1 Introduction

This computational exercise concerns the manipulation of matrices for solving physical problems. In the first task, a matrix inversion routine was written for $n \times n$ matrices. The second task required the use of the matrix inversion routine to solve linear equations. Then to compare this to the lower-upper (LU) and singular value (SV) decomposition routines found in the scipy.linalg library. The final task involved using the most effective matrix algorithm to solve linear equations to describe the tension in three wires suspending a trapeze artist.

2 Analytical Inversion

The method used for writing a routine to find the inverse of an $n \times n$ matrix was analytical inversion. This is governed by equation 1 where A is the matrix to invert, and C is the matrix of cofactors [1].

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

This equation was implemented using a recursive function to find the determinant. Originally, the matrix was reduced to 2×2 but an decrease in computing time was found by using the determinant formula for a 3×3 matrix. The determinant and minor functions are then used with a transpose function to calculate the matrix of cofactors and then the inverse of the original matrix. Due to the nature of the method for calculating the determinant, it is expected to be computationally expensive. An increase of the matrix size to $n \times n$ leads to n times more determinants to be calculated compared to the $(n-1) \times (n-1)$ matrix. This will lead to an $n!$ increase in computing time per added row.

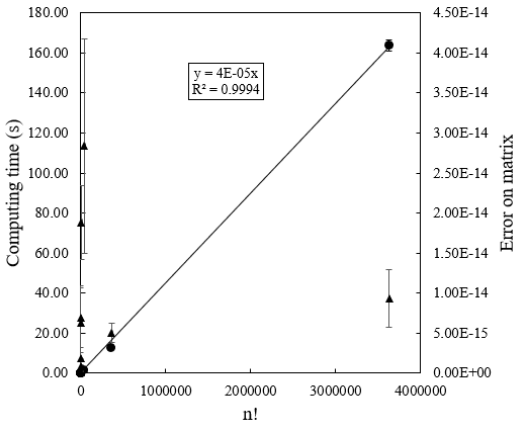

Figure 1: A graph showing the computing time and matrix inverse error for the analytical inversion routine. The circles represent the computing time and have been given a linear fit. The triangles represent the error on the inverse matrix by comparing to the identity matrix.

Figure 1 shows that the expected factorial behaviour for computing time is observed. This data was generated with five runs per matrix size and the matrices were randomly generated with the numpy.random function. The error bars were calculated using the average error on the mean. For the computing time, the data has been fitted with a linear relationship, the displayed R^2 value is close to unity and the error bars are small. In fact, they are only visible for the furthest right point, corresponding to a 10×10 matrix. Due to the factorial relationship being observed, an 11×11 matrix was not calculated as the computing time would be unpractical. The inverse matrix errors represented by the triangle points were calculated using the equation $A^{-1}A - I = 0$, where I represents the identity matrix. The inverse error was taken to be the largest value element of the resulting matrix. It can be seen that these are of the order 10^{-14} due to rounding. The higher error values have much larger error bars than the smaller val-

ues, showing this increase in inverse error may be caused by one or two calculations with unusually high calculation errors within the set of five runs. This could be due to the random matrix being close to singular, where the determinant would be close to zero and the rounding errors amplified.

3 Solving Linear Equations

In the second task, the analytical inversion method was compared to the LU and SV decompostion methods found in the scipy.linalg library. These functions were implemented to solve a set of linear equations in the form shown in equation 2, where a_n represents the coefficients of the variables x_n, and b_n are known constants.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots \\ a_{21} & a_{22} & a_{23} & \dots \\ \vdots & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \end{bmatrix} \tag{2}$$

To compare the speed of the LU and SV decomposition methods to the analytical inversion method, random matrices and constants were generated using the numpy.random function and solved. Similarly to the last section, the data shown in figure 2 was generated with five runs per matrix size.

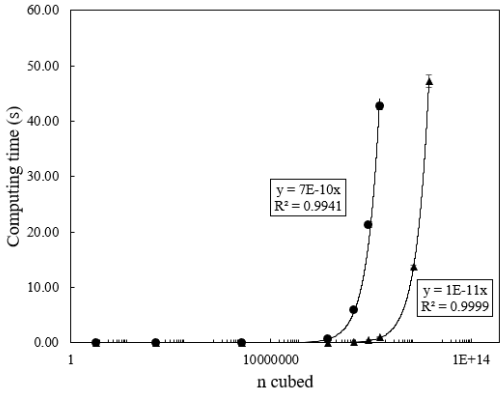

Figure 2: A graph showing the computing time plotted against the matrix size n cubed along a logarithmic axis. The circles represent the SV decomposition that has been given a linear fit. The triangles represent the LU decomposition, also given a linear fit.

Figure 2 shows the computing time of the SV and LU decomposition methods is far faster than the analytical inversion method. It can be seen that they both fit the cubic complexity relations expected [2]. For the SV decomposition, the R^2 value is 0.9941 which is close to unity suggesting a good fit. The error bars are barely visible showing that there is little spread in the data. For the LU decomposition method, the same can be said in terms of the R^2 value and the error bars. The error bars on the furthest right point are the largest but this is to be expected as the gradient of the fit is at its greatest. Although the SV and LU decomposition methods are both cubic in their complexity, the SV method shows a faster increase in the computational time. The LU decomposition method consists of three steps to solve a set of linear equations: forward elimination, back substitution, and forward substitution [3]. The sum of these three processes results in a computational complexity proportional to $\frac{2n^3}{3}$ [1]. The SV decomposition of a matrix is computed by a two step method. First, a square matrix is reduced to a bidiagonal matrix with a cost proportional to n^3. The second step involves computing the SV decomposition of the bidiagonal matrix leading to a cost proportional to n^2 [1]. Therefore, the SV decomposition has an additional contribution to the computational complexity and shows a greater increase in time for an increase in matrix dimensions.

To compare the accuracy of the analytical inversion, SV and LU decomposition methods, the routines were used to calculate the x, y, and z values of the following equation. As k is reduced, the matrix approaches singularity.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & -1 \\ 2 & 3 & k \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 10 \\ 15 \end{bmatrix} \tag{3}$$

The values of x, y, and z in equation 3 are expected to be $0, 5$, and 0 for all values of k not equal to zero. The cumulative error was calculated for the x, y, and z values combined.

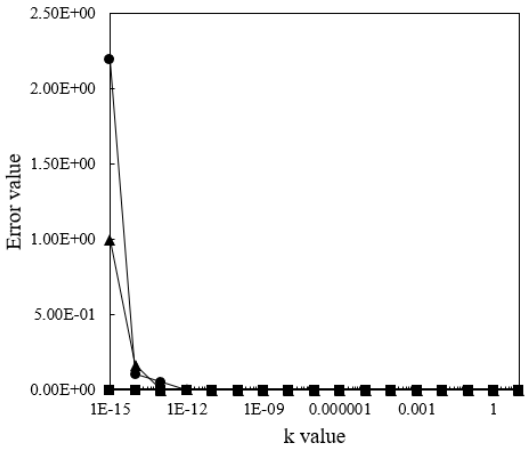

Figure 3: A plot showing the increase in the error of the matrix for a decrease in the value of k on a logarithmic axis. The triangles, circles, and squares represent the analytical inversion, SV, and LU decomposition methods respectively.

It can be seen in figure 3 that the SV decomposition method provides the greatest error as the matrix approaches singularity. This is because part of the SV decomposition involves computing a diagonal matrix with the singular values (w_j) of the original matrix. If the original matrix is close to singular, some of the singular values will become close to zero. To solve the linear equations, the inverse of this diagonal matrix is computed leading to the calculation of $1/w_j$. This then introduces a greater and greater rounding error as w_j approaches zero [2]. The error on the analytical inversion method is also large for small k. This can be explained by equation 1, as the determinant approaches zero as the matrix becomes singular. This introduces a rounding error that is increased as the determinant trends to zero. The error on the LU decomposition method is zero until $k = 10^{-16}$, at which point the LU and analytical inversion routines return that the matrix is singular. For each k value, the model was run a number of times but the error values did not change between each run, so error bars are not included. The result of this section has shown that the LU decomposition method gives the best accuracy and the fastest computing time. Therefore, this method will be used for the remainder of the exercise.

4 Physics Problem

In the final task of this exercise, the aim was to set up a system of linear equations to solve the tension in wires suspending a trapeze artist, first in 2-dimensions and then in 3-dimensions. The 2-dimensional case is shown below in figure 4. The two angles θ_1 and θ_2 were calculated using the x and y position. The x direction is taken to be the horizontal, and the y direction is taken to be the vertical.

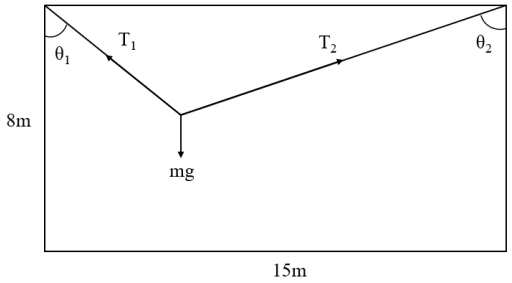

Figure 4: The 2-dimensional case of the wires suspending the trapeze artist.

To implement this in the code, the forces were resolved in the x and y directions using the angles θ_1 and θ_2. The 2-dimensional set of linear equations could then be solved with the LU decomposition method. This function was then used to produce a tension plot and find the maximum tension values.

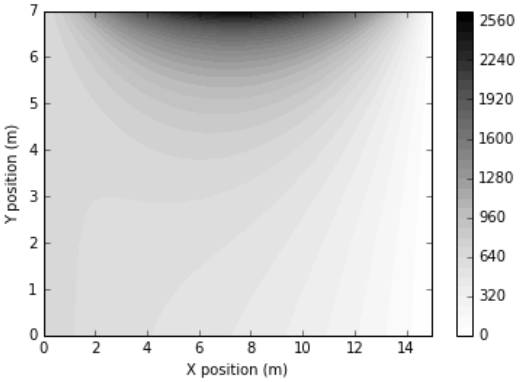

Figure 5: The tension of the first wire as a function of the x and y position. The colour bar on the right represents the tension in newtons.

As shown by the plot in figure 5, the tension in the first wire is at a maximum at the greatest y position of 7 m. In fact, the point of greatest tension was calculated to be at an x position of 7.43 m and a y position of 7 m for an array input with increments of 0.01 m. The point of maximum total tension is for an x position of 7.5 m and a y position of 7 m for the same array input corresponding to a combined total tension of 5196 N. This shows the system of the two wires is symmetrical with a line of symmetry at $x = 7.5$ m. It was expected that tension would be maximised at $y = 7$ m as this maximises potential energy. The problem is solved assuming a static system, so the kinetic energy is zero. Therefore, the maximum total energy, leading to the maximum tension, will be when the potential energy is greatest. An additional point to note is that the tension in the first wire goes to zero at an x position of 15 m. This is expected as the trapeze artist will be directly below the anchor point of the second wire, so the entire suspended mass is supported by the second wire.

Now, the problem is extended to 3-dimensions with a third wire anchored at an x position of 7.5 m, a y position of 8 m, and a z position of 8 m. This allows the trapeze artist to move forwards and backwards in the z direction. To implement this in the code, a new function was created to calculate the tension in the wires for the 3-dimensional system. As before, θ values were specified for each wire but now including the added z component. Additionally, a new angle (ϕ) was introduced to quantify the rotation of each wire in the x, z plane about the y axis. Similarly to the 2-dimensional case, the forces

were resolved in each direction which created a set of linear equations to be solved by the LU decomposition method. This new tension function was passed through a function to find the maximum tension values of the system at a position increment of 0.1 m. The maximum tensions in the first and second wires are the same as the 2-dimensional case and the maximum tension of the third wire is 1902 N at $x = 11.2$ m, $y = 7$ m, and $z = 4$ m. Again, the maximum tension is found when the potential energy is greatest. Therefore, the tension in the three wires was plotted as a function of x and z position at a y position of 7 m.

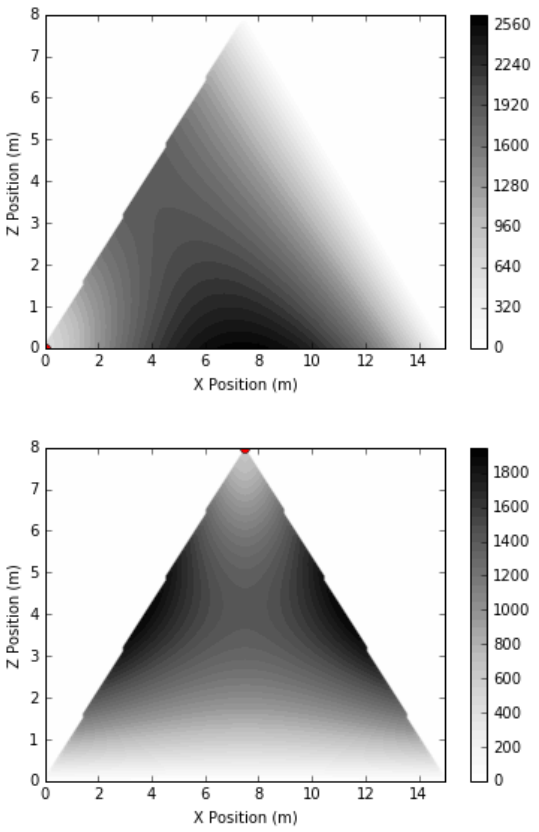

Figure 6: Top: a plot showing tension in first wire with a red dot signifying the anchor point at (0, 0). Bottom: a plot showing tension in third wire with a red dot showing the anchor point at (7.5, 8). The colour bar represents the tension in newtons.

The top plot in figure 6 shows the tension in the first wire as a function of x and z position. It can be seen that the maximum tension is when $z = 0$, so this agrees with the calculated maximum tension. A similarity with the 2-dimensional case is that the tension in the wire goes to zero when the x position is 15 m. However, it is also zero along the straight line from the second to the third anchor point, as the mass will be supported only by the second and third wires at these points. The tension of the second wire is a reflection of the first, so is not presented in this report. In the bottom plot, the tension of the third wire is shown as a function of x and z position. This appears to be a symmetrical plot, unlike the tension in the first and second wires. This is expected as the triangular area the mass moves in is isosceles in shape. The two equal length sides are those connected to the third anchor point, so its tension plot is symmetric with a line of symmetry at $x = 7.5$ m. A final point to note is the tension going to zero when $z = 0$ m, which simplifies the system back to the 2-dimensional case.

A model to determine the tension in the wires for a system such as this is a necessary step in the design process of a trapeze system. When designing these kinds of systems, the wires used will have a particular force at which they can be safely used. If this is exceeded then the wires can snap, leading to unwanted problems. Therefore, it is important to accurately quantify the forces that will be created in the system.

To conclude, various methods for solving a set of linear equations have been considered in this exercise. The LU decomposition has been found to be the most accurate and fastest way of performing this task. Therefore, this method has been used to create a model to determine the tension in a set of three wires suspending a trapeze artist. In the future, this model could be extended to include an additional wire to allow the trapeze artist to move over a greater volume of space. Or, it could be useful to allow more user inputs so it can be used as a design tool, rather the system described above.

References

[1] L.N. Trefethen, D. Bau, *Numerical Linear Algebra*, Philadelphia: Society for Industrial and Applied Mathematics, (1997).

[2] W. H. Press, *Numerical Recipes in C*, 2nd Edition, Cambridge University Press, Cambridge (1992).

[3] J. Kiusalaas, *Numerical Methods in Engineering with Python*, 1st Edition, Cambridge University Press, New York (2005).