**Numerical methods used to solve linear equations**

**By khalil pierre**

The aims of this exercise were to investigate different methods of solving linear equations and apply a suitable method to calculate the tension in a system of wires used to suspend a trapeze artist above a stage.

**Background**

A set of linear equations can be represented in matrix form as:

where **A** is the matrix of coefficients, **x** is the set of unknowns and **b** is the set of constants that each equation is equal to. The linear equation 6 can be solved by finding the inverse of the matrix **A**:

The inverse of the square matrix **A** can be found using Cramer’s rule:

Det(**A**) is the determinate of **A** and **CT** isthetransposeof the matrix of cofactors.However, there are quite a few problems with this method; if **A** is singular (det(**A**)=0) then the inverse of **A** does not exist, this method is also very computationally expensive and becomes impractical on most computers for N>10 [1][3]. A faster and often more robust method of solving linear equations is to decompose the matrix **A** into a set of matrices that are easier to solve [1][3].

**Task 1) Cramer’s rule**

Task 1 was to create a program that used Cramer’s rule to calculate the inverse of a matrix. To do this the Determinate of **A** had to be calculated there are two main methods to calculating the determinate of A. The first method uses Gaussian elimination, row subtraction is used to set all the off-diagonal elements on the left-hand side of the matrix equal to zero [1]. The determinate is then equal to product of the diagonal elements [2]. However, this method cannot be used if any of the diagonal elements are equal to zero. The second method and the one used in the code uses Laplacian expansion where an NxN matrix can be written using as a series of 2x2 matrices. The determinate of a 2x2 matrix is trivial to calculate and so the determinate of the NxN matrix can be found form the 2x2 matrices.

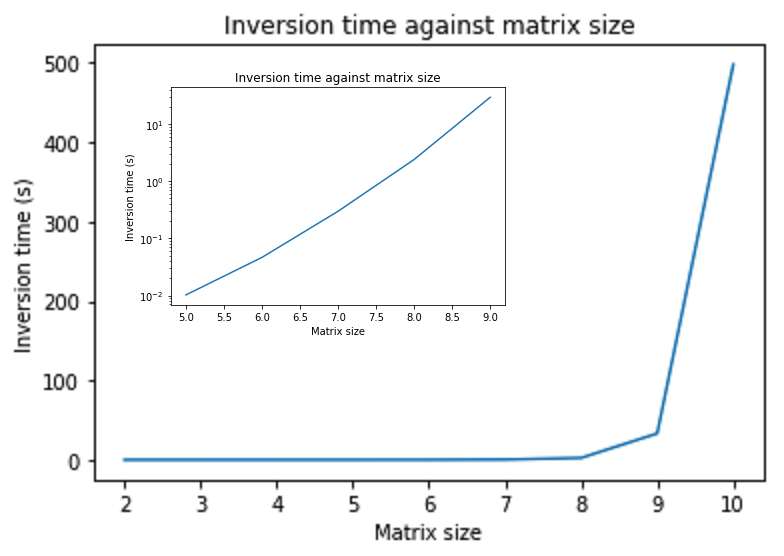
To be able to calculate the Laplacian expansion and find **CT** a function that calculated the minor matrix of each element in a matrix had to be written. Once this was done Cramer’s rule was used to calculate the inverse of a matrix. Task 1 asked to investigate how the inversion time scaled with the matrix size. Bellow is a plot of matrix size against inversion time:

Figure 1 Plot of the inversion time against matrix size for the Cramer function. This plot was generated using random matrices. The sub plot is the matrix size against the natural log of the inversion time for matrix sizes 5-9.

As you can see the time taken to invert a matrix using Cramer’s rule increases exponentially as the matrix size increases. The reason for this is that for an NxN matrix N!/2 2x2 matrices are needed to calculate the determinate therefor the number of calculation blows up as N increases.

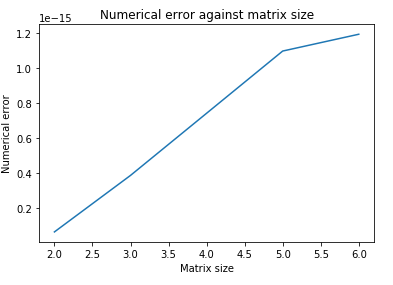
To assess the accuracy of my function I used the fact that by definition **A** multiplied by its inverse is equal to the identity matrix. I subtracted the identity matrix from the product of the matrix calculated by the Cramer function multiplied by the original matrix. The largest difference between the Cramer function and the identity matrix was taken as the error. Bellow is a plot of matrix size against error:

Figure 2 Plot of matrix size against numerical error. The plot was generated by averaging over approximately 500 random matrices for each matrix size.

Figure 2 seems to indicate that the general trend is for the size of the error to increase as the matrix size increases. However, to generate this plot the numerical error for each matrix size had to be averaged over 500 random matrices otherwise the results were unclear. This seem to indicate that the numerical error is dominated by the size of the elements within the matrix and not on the matrix size. This makes sense as the error in the Cramer method will be due to floating point error which will have a bigger impact for smaller elements. It also makes sense that the numerical error increases as matrix size increases as more calculations are required for larger matrices which allows more floating-point errors to be introduced. The reason approximately 500 matrices were used is because any matrix that was singular would cause the code to crash and so had to be removed from the sample before the inverse was calculated. The size of the errors suggest that the Cramer function used in the code was accurate.

The computational time of the Cramer function created in task 1 may be improved by using Gaussian elimination to calculate the determinate of a matrix. The reason that the Gaussian elimination was not used was because a single zero diagonal element would cause the code to not work. To make the Gaussian method more robust pivoting could be used to replace the rows/columns with diagonal zero elements with rows/columns with non-diagonal zero elements. Gaussian elimination however also fails to work with singular matrices.

**Task 2) Single value and LU decomposition**

As mentioned earlier to solve linear equations it is often easier to decompose a matrix into a set of matrices that are easier to solve [3]. LU decomposition resolves a matrix into two triangular matrices a so called upper and lower matrices [3]. Solving a linear equation with a triangular matrix does not require you to calculate the inverse of a matrix [1]. Consider the following equation:

x1 is given as b1/a0,0 the value of x1 can be substituted into the second row to calculate x2 this process can be repeated for an NxN matrix to calculate every xi value. Singular value decomposition (SVD) resolves a matrix into the following form [3]:

U and V are orthogonal matrices and a diagonal matrix of singular values [3]. The inverse of a diagonal matrix is trivial to calculate and the inverse of a orthogonal matrix is simply the transpose of said matrix, from equation 2 and 5 **x** is given as:

SVD decomposition has the bonus that it approximates a solution that best fits the set of linear equations if the matrix **A** is singular[1][3].

Task 2 asked us to investigate how the time taken to solve a linear equation scaled with the size of the linear equation. Bellow is a plot of run time for SVD and LU methods against matrix size:

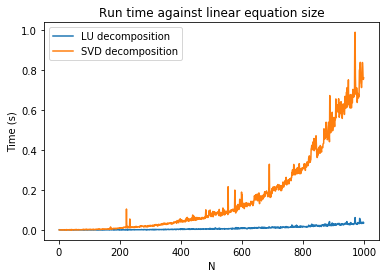


Figure 3 Run time against matrix size for SVD and LU decomposition. The pointed artefacts on the curves are a result of using random matrices, faster processing times would allow us to average each matrix size and would reduce these.

Both SVD and LU decomposition are a lot faster then the Cramer function however at N>100 the time taken for SVD to calculate the solution to a linear equation diverge from LU decomposition.

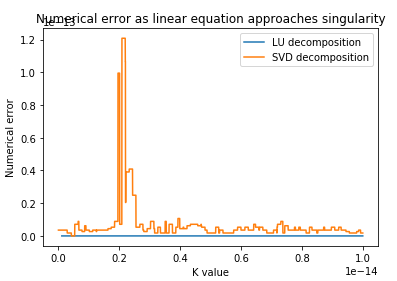
Task 2 also asked to investigate how the two decomposition methods behave as **A** approaches singularity. Bellow is a plot of the numerical error against the value of k where k is a value in a near singular matrix (if k=0 then the matrix is singular):

Figure 4 Numerical error against value of [2,2] element k of a near singular matrix. Numerical error was calculated using **b-Ax=** numerical error.

LU decomposition seems to have no numerical error as k approaches the floating-point limit but then stops after k drops below the floating-point limit at this point k is zero as far as python is concerned. LU decomposition does not work for singular matrices. SVD decomposition does have a larger error but approximates a solution passed the floating-point limit. There is a peak at k=2e-15 for the SVD method that needs further investigation. An extension of this may be to look at how the error varies as a function of the determinant.

**Task 3) physics problem**

Task 3 wanted the tension in three ropes used to suspend a trapeze artist above a stage at any position. Using newtons first law the following set of linear equations can be constructed:

Ti is the magnitude of the tension in each string, θ is the angle between the wire vector and the xi’-yi’ plane, φ is the angle between the projection of the tension vector in the xi’-yi’ plane and the xi’ axis. xi’ yi’ and zi’ are the components of the wire vector which is given as

Where **r** is the position vector **rDrum,i** and **ri’** For each wire the components of the wire vector can be used to find cosine and sine of φ and θ using simple trigonometry.

Task 3 wanted the tension at each position for the 3D and 2D case and the max tension in each wire and the position it occurred. By using the Cramer function, LU or SVD methods the set of linear equations given by 7,8 and 9 can be solved for the tension. For the 2D case the SVD method was used to solve the set of linear equations as the matrix of coefficients is singular at x=7.5. For the 3D case LU decomposition was used as the numerical error is the smallest. At N=3 the difference in computation time of each method is negligible. The position at which the max tension occur in the 2D case is when the artist is at the centre and at z=7. This is because the z component of the tension is given by Tz=Tsin(θ). For small angles sin(θ) will be small but from our equations the z component of the tension must be equal to mg so T must be large.

1. H, W., Teukoisky, S. A., Vetterling, W. T., & flannery, B. P. (1986). *Numerical recipes the art of Scientific computing third edition.* Cambridge university press.
2. hanna, S. (2016). *First year Essintial maths for Physics course.* University of bristol
3. brooke, J. (2019). *Computational physics lecture 1.* University of bristol.