Assignment 1: Python Routines for Matrices and Linear Algebra

G. Flatters
Department of Physics, University of Bristol.
(Dated: October 20, 2020)

This assignment explores the advantages and disadvantages of different methods for solving systems of linear simultaneous equations. Evidence has been found to support the theoretical computational cost associated with each method. This was achieved by collecting data of method speed against matrix dimensions *n* and then performing a chi-square test of this data against that predicted by the theory. The LU Decomposition routine was then used to solve a physics problem.

MATRIX INVERSION FOR LINEAR ALGEBRA

A matrix equation can be written for a given set of linear simultaneous equations, with the number of unknown variables being equal to the dimensions of the square matrix. As shown in equation 1, the inverse of a matrix *A* can be used to find the vector of unknown variables $\boldsymbol{x}$ for a vector of known constants $\boldsymbol{b}$.

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow \mathbf{x} = A^{-1}\mathbf{b} \tag{1}$$

A routine has been written to find the inverse of a *nxn* matrix using Cramer's Rule (2), where C^T is the transpose of the matrix of cofactors of *A*. This function was initially tested against the *scipy.linalg.inv* using *np.allclose()* within a tolerance of 1×10^{-8} to take errors into account.

$$A^{-1} = \frac{1}{detA} C^T \tag{2}$$

The error of the analytic method was analysed for a set of linear equations that are close to singular (3). This was done by computing AA^{-1} for small steps in k and then finding the absolute difference between this matrix and the identity matrix. This absolute error was plotted in comparison with the absolute error produced by *scipy.linalg.inv* 1 (reflected in the x-axis). The symmetry of this graph in the x-axis suggests a successful matrix inversion routine.

The inverse is calculated for any k larger than 10^{-15}, below this an error is produced as the matrix is now considered to be singular as it no longer has an inverse. This is exactly as expected as 10^{-15} is the relative quantisation step of a floating point number (float64) [1]. There is a problem here in that the inverse of the matrix calculated becomes completely different despite the value of k being changed a small amount, this explains the spike at 10^{-15} on figure 1. At this point the solution of equation 3 changes from (0, 5, 0) to (0, 2, 0).

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

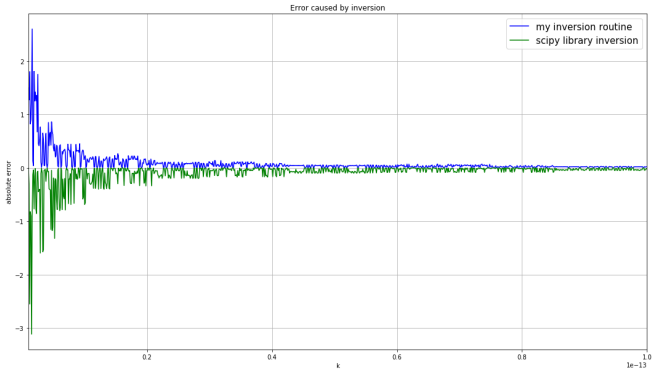

FIG. 1: Absolute error for k between 1×10^{-15} to 1×10^{-13}. Scipy data has been reflected in the x-axis so that data is easier to see.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, L = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix}, U = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

FIG. 2: LU factorisation

ALGORITHMS FOR LINEAR ALGEBRA

Matrix decompositions decompose a matrix into constituent components that make subsequent calculations simpler. To reduce computational 'cost' and problems with singular matrices, different algorithms are used in favour of the analytic method. The two algorithms discussed in this report are LU Decomposition and Singular Value Decomposition.

For LU decomposition a square matrix *A* is factorised such that $A = LU$, where *L* is a lower triangular matrix and *U* the upper triangular matrix (figure 2). The general solution to a set of linear equation is then reduced to the successive solution of two triangular equations. The *scipy.linalg.* routines for factorising and then solving were used to find solutions to equation 1.

Singular Value Decomposition has certain advantages over LU Decomposition; it can be used for matrices that are not square or matrices that have a determinant close to zero. This is at the expense of greater computational cost as can be seen in figure 3. Therefore, it is usually employed for matrices that have determinants close to the minimum floating point preci-

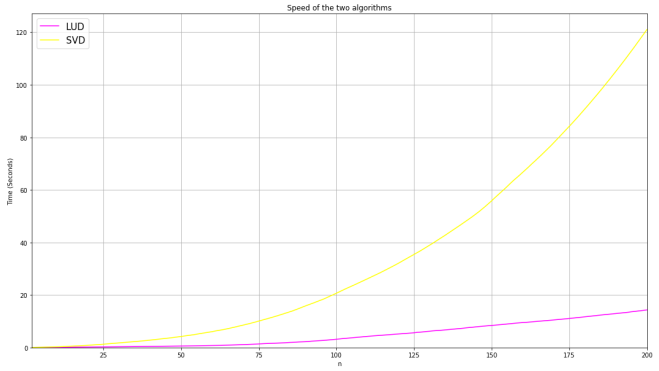

FIG. 3: Speed comparison of LU decomposition and singular value decomposition for varying n

sion where rounding errors would begin to become problematic for LU Decomposition. This makes SVD the more powerful method numerically as it is applicable to a wider range of problems and is more stable. Therefore, the answer to which algorithm is better depends on the task at hand.

For Singular Value Decomposition the matrix is written in the form $A = USV$, where *U* and *V* orthonormal matrices and *S* is a diagonal matrix containing the singular values. S, U and V were found using the *scipy.linalg.svd()* routine and solutions were found by applying these in three subsequent equations.

The solutions produced via these two algorithms gave the same results as the analytic method for a tolerance of eight significant figures. This was successful for random matrices with a maximum integer element value of 9 for matrix dimensions $2 \leq n \leq 10$. The speed difference between LUD and SVD were indistinguishable up to this value of *n* but the analytic method became too costly. For smaller matrices (*n = 2, 3*) the analytic method is actually slightly faster as the algorithms reproduce the analytic method with extra steps.

FLOATING POINT OPERATIONS

The reason for the discrepancies in cost between the methods is due to the number of floating point operations required for each method to be carried out. A floating point operation is defined as any mathematical operation or assignment that involves floating point numbers (as opposed to binary integer operations). In computing, the number of floating point operations per second is a measure of computer (central processing unit) performance [2]. The cost of each method is related to the dimensions *n* of the matrix as follows:

- Analytic method: $n \times n!$ (Cramer's Rule) [3]
- LU Decomposition: $2/3n^3$ [3]
- Singular Value Decomposition: n^3 [3]

A theoretical time taken to perform these algorithms for varying *n* can therefore be determined by dividing these costs by the number of floating point operations a given computer performs per second. This theoretical fit will be used to judge the confidence in the speed data for the different methods. An additional list was added to each timing function that plots the respective theoretical cost of each method in seconds for varying *n*. The time data for the analytic method was taken for 50 random matrices, the LU decomposition data was taken for 150 random matrices and the SVD data was taken for 100 random matrices. This was done in an attempt to improve the reproducibility of the results and was accounted for when calculating the theoretical time.

RESULTS

A chi-square goodness of fit test was used to measure how well the data corresponded to predictions of the FLOP model. The null hypothesis is that the model is correct and the alternative hypothesis is that it is not. The analytic method data has five degrees of freedom, making the critical chi-square statistic 0.412 for a confidence level of 99.5% [4]. The LU decomposition data has 148 degrees of freedom making the critical chi-square value 107.441 at the 99.5% confidence level. For SVD the data has 98 degrees of freedom making the critical chi-square value 65.694 at the 99.5% confidence level. The degrees of freedom is equal to the number of data points minus one. For all methods there was a significant variance in the chi-square result for successive measurements. This is most likely due to the low number of repeats that the measurement was performed over, which is necessary to reduce cost. To combat this the code was run 20 times and the average chi-square statistic found. These measurements were made by running the relevant snippets of code rather than the whole file as this helped reduce the variance.

For the analytic method the chi-square characteristic was found to be 0.235 and for the LU decomposition it was found to be 93.04. In both of these cases none of the values exceeded the respective critical chi-square statistic. This means that the null hypothesis can be accepted to the 99.5% confidence level in both cases. The SVD data was marginally less successful, with the average chi-square statistic being 70.118. This means the null hypothesis has to be rejected at the 99.5% confidence level. However, the chi-square statistic is below the critical value of 72.501 for a confidence level of 97.5%

The routine for plotting the fit and finding the chi-square statistic is limited due to it not being very robust; their is high variance in values depending on how many snippets of code are running. The biggest limitation is that more variance will be expected when this routine is running on computers with different processing power. It would be interesting to see the results of this code performed across multiple machines as currently there is a decent amount of uncertainty in the value of FLOPs. To further increase confidence in validity of the FLOP models, the routine needs to be adapted to give more reproducible results. Additionally, the model could be tested for larger values of n and a greater number of repeats.

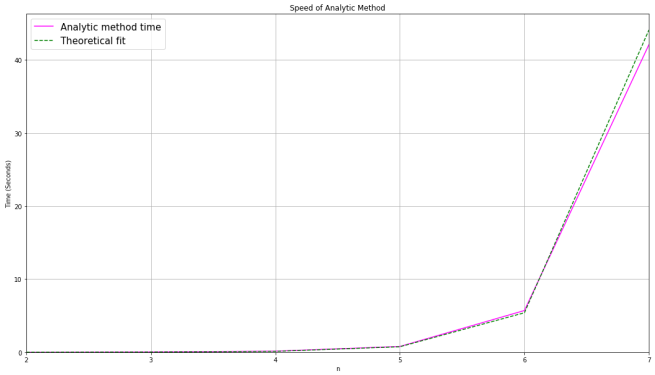

FIG. 4: Speed of analytic method vs n. Chi-square statistic = 0.143

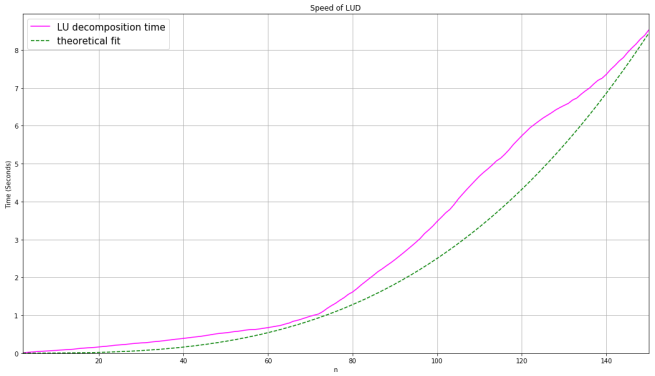

FIG. 5: Speed of LU decomposition vs n. Chi-square statistic = 84.331

A better method may be to use *scipy.optimize.curvefit* which would use parameters to fit a function to the data. For example values of a, b and c could be found for the LU decomposition data which has a theoretical fit of a/bn^c. The amount these parameters differ from those predicted by the theory ($a = 2$, $b = 3$ and $c = 3$) would then be analysed to test the accuracy of the data. This was the intended method to be used for this task but unfortunately it did not provide results of any value.

In terms of improvements for the code as a whole, the most important would perhaps be including a way of dealing with singular matrices. In task 1 if the random matrix generated happens to be singular an error is raised and the code does not run. An attempt was made at filtering out singular matrices using an *if determinant == 0* statement in the determinant function but regrettably this was not successful. If this were implemented it would also deal with the runtime warnings that are intermittently experienced in task 2.

It seems plausible that an interesting extension could be written such that the code optimises the fit of the data to that of the theory and outputs a FLOPs value, characteristic of the computer that the code is being run on.

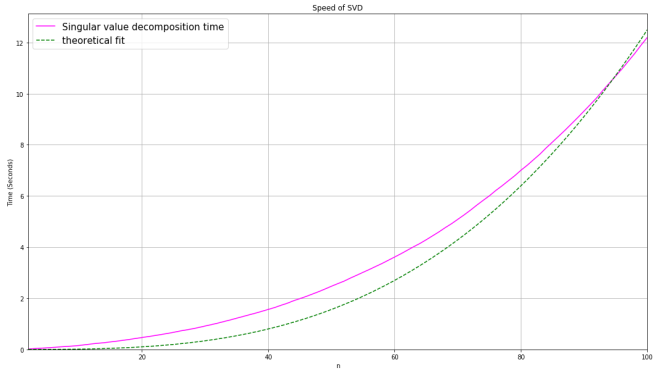

FIG. 6: Speed of SVD vs n. Chi-square statistic = 68.118

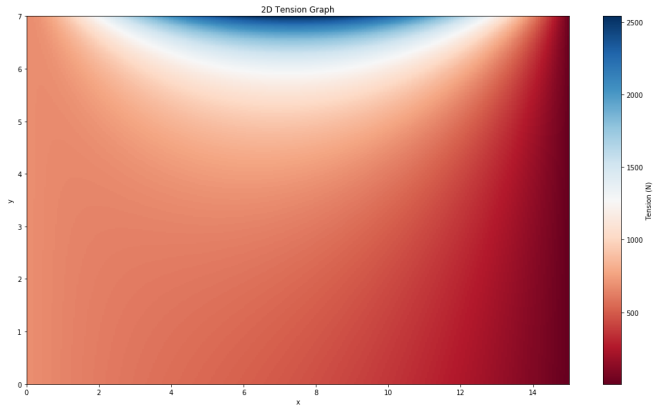

FIG. 7: Heatmap of tensions in the left string as a function of x and y

PHYSICS PROBLEM

For the two dimensional problem, LU decomposition was used due to its superior speeds that were observed in the previous task; the advantages of SVD discussed earlier are not relevant to this problem. The maximum tension was found to be *2598N*, this occurs at ($x = 7.46$, $y = 7.0$). The maximum tension was found simply by applying *np.amax* to the array of tension values. The index of this maximum tension was then used to find the corresponding coordinates. The graph of the tensions in the left string were plotted on a heatmap as a function of x and y (figure 7). This was done by writing a function which takes on values of x and y specified using *np.meshgrid()* and outputs a matrix containing the components of the length of each string for each value of x and y. This matrix, as well as the force vector, were then solved for in an LU decomposition function which returns an array of tensions. The array element corresponding to the tension in the left string was plotted for each value of x and y on the meshgrid.

For the three dimensional problem the method employed was much the same. The maximum tension in the left string was found to be *568N* at coordinates of (7.51, 0, 7.0). The validity of the physics problem results were tested against hand

written calculations, intuition about the symmetry of the problem and results produced by peers. It did not seem possible to write a test for the validity of the results as the accuracy of any code written would be at the discretion of the underlying mathematics.

CONCLUSIONS

In conclusion, the advantages and disadvantages of the analytic method, LU Decomposition and Singular Value Decomposition have been explored and discussed. Data has been collected which provides evidence to support the methods that have been used to derive the theoretical cost of each method. My analytic method routine supports the theory that this method requires $n \times n!$ floating point operations to a confidence level of 99.5%. My LU Decomposition routine supports the theory that this method requires $2/3n^3$ floating point operations to a confidence level of 99.5%. My Singular Value Decomposition routine supports the theory that this method requires n^3 floating point operations to a confidence level of 97.5%. Improvements and limitations of the method used to collect this data have been discussed. The LU Decomposition routine was then successfully applied to a physics problem.

[1] Christian Hill. *Learning scientific programming with Python*. Cambridge University Press, 2016.

[2] Raphael Hunger. *Floating point operations in matrix-vector calculus*. Munich University of Technology, Inst. for Circuit Theory and Signal , 2005.

[3] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[4] Edwin B Wilson and Margaret M Hilferty. The distribution of chi-square. *proceedings of the National Academy of Sciences of the United States of America*, 17(12):684, 1931.