Methods of solving linear systems using Python

Anna Griffith
University of Bristol Physics Department

(Dated: February 18, 2019)

Four methods for solving systems of linear equations are compared, analytical inversion using Cramer's method, LU decomposition (my own and a built in function), and SVD decomposition. Analytical inversion was found to be computationally inadvisable. LU decomposition was the fastest method and produced the smallest percentage errors, whereas SVD decomposition was the most versatile in term of the types of solutions offered. These were also put into practice in solving a physics problem.

I. INTRODUCTION

This programs aims to solve a system of linear equations, using several different techniques. The suitability of each will be compared, depending on the number and contents of the equations provided. Overall, four differently algorithms will be be compared; my own routine for analytical matrix inversion, my own routine for LU decomposition, and built in *scipy* functions for LU and SVD decomposition. All four algorithms will be solving the same set of equations, written in matrix form as

$$\mathbf{A}x = b \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{n,m}$ and $b \in \mathbb{R}^m$ are given, and $x \in \mathbb{R}^n$ will be computed. The computational speed of each algorithm will be compared, particularly how the speed changes as a function of n, the dimension of x. The errors produced in the calculation of x will also be analysed, by taking the percentage difference between $\mathbf{A}x$ and b.

The first method of analytical matrix inversion, using Cramer's Rule, requires certain conditions to be put on Equation 1. A square matrix input is needed for this algorithm, which computes the inverse matrix using the following formula,

$$\mathbf{A}^{-1} = \frac{1}{det(A)}\mathbf{C}^T \tag{2}$$

where $\mathbf{C}$ is the matrix of cofactors of $\mathbf{A}$ [5]. More specifically, this method requires a non zero determinant of $\mathbf{A}$, hence the rank of the matrix must equal n, ensuring no rows or columns are degenerate. Hence this algorithm is only applicable in a small number of cases, and in the vast majority of practical computational problems it is unnecessary and inadvisable to actually compute $\mathbf{A}^{-1}$ [3]. In this algorithm, the computation time quickly becomes impractical, as it increases with $\mathcal{O}(n!)$, furthermore it is widely seen as numerically unstable due to round errors introduced in computing the determinants [2].

The second algorithm, LU decomposition, puts the same restrictions on the system of equations, but the time complexity is $\mathcal{O}(n^3)$ [1], and the rounding error can be reduced with the application of a permutation matrix. It can be shown that for any non-singular, square matrix $\mathbf{A}$, there exists a permutation matrix $\mathbf{P}$ such that

$$\mathbf{PA} = \mathbf{LU} \tag{3}$$

where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices respectively [5]. Then, letting $y = \mathbf{U}x$, following equations can then simply be solved by forward and backward substitution in order to calculate x,

$$\mathbf{L}y = b, \mathbf{U}x = y \tag{4}$$

each requiring $\mathcal{O}(n^2)$ operations, giving a total of $\mathcal{O}(\frac{1}{3}n^3)$ [4]. $\mathbf{A}$ can only be factorised this way if it has non-zero diagonal elements, so a permutation matrix, which acts to swap the rows and columns is often required.

Finally, SVD decomposition is perhaps the most versatile method as it can be used to create solutions, or approximations to solutions of any matrix $\mathbf{A} \in \mathbb{R}^{n,m}$, which can be written in the form,

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T \tag{5}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal unitary matrices and Σ is a diagonal matrix containing the singular values of $\mathbf{A}$. SVD decomposition is also a computationally simple and accurate way to compute the psedo-inverse of a matrix,

$$\mathbf{A}^+ = \mathbf{V}\Sigma^+\mathbf{U}^T \tag{6}$$

which can be used to compute a least squares, 'best fit' solution to a system which is either over or under determined, [4]. It will equal the true inverse in the case that $\mathbf{A}$ is full rank. The number of operations required depends on the dimensions of the matrix, in the case that $m \geq n$ it is $\mathcal{O}(mn^2)$ [5].

The following techniques for solving Equation 1 will then be put to use in a physics problem, where a weight is suspended above a stage by three wires. The tension in one of the wires is plotted as a function of the position of the weight, and the maximum value found.

II. METHOD

First, an analytical inversion formula was written, involving three different functions. The first would create a matrix of minors; the original matrix with the ith and jth columns removed. The function to find the determinant is iterative, using the formula,

$$det(\mathbf{A}) = \sum_{i=1}^{n} (-1)^{i+j} a_{ij} det(\mathbf{A}_{ij}) \tag{7}$$

where $\mathbf{A}_{ij}$ is the matrix of minors described above [5]. The inverse can then easily be calculated using equation 2, as long as the determinant is non zero, and x determined since $x = \mathbf{A}^{-1}b$

To explicitly create an algorithm for LU decomposition, without using any *scipy* functions, three steps were required. The first function creates a permutation matrix through partial pivoting, which finds the maximum element in each row and then swaps rows so that it is in a diagonal position. This avoids having any zero-elements in the diagonal, in which case LU decomposition cannot be occur. To then factorise $\mathbf{PA}$ into lower and upper triangular matrices, Doolittle's algorithm is performed. This sets all diagonal $l_{ii} = 1$ and then uses the following formula to find $\mathbf{L}$ and $\mathbf{U}$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} u_{kj} l_{ik}, \tag{8}$$

$$l_{ij} = \frac{1}{u_{jj}} (a_{ij} - \sum_{k=1}^{j-1} u_{kj} l_{ik}) \tag{9}$$

[1] A third function was then created to perform forward and backward substitution to find x.

The built in *scipy* function uses a similar partial pivoting method to decompose $\mathbf{A}$, and x can be easily found using built in *solve* function. With all of the above methods an initial check is performed on $\mathbf{A}$ to ensure that it is both square and non singular before performing the calculation.

A built in *scipy* function *'svd'* returns the matrix decompositions of $\mathbf{A}$ as described in equation 5. To compute the pseudo-inverse the matrix Σ^{+} needs to be calculated. the diagonal elements are simply taken to be $1/\sigma_{ii}$ for σ_{ii} above a threshold value, set according to the machine epsilon. The quantity $\overline{x} = \mathbf{A}^{+}b$ can then be identified, which will equal x in the case that $\mathbf{A}$ is full rank, or a least squares solution is not. In the case where the system is under-determined, a space of solutions may be possible, in which case,

$$x = \bar{x} + \lambda v \tag{10}$$

where v is the null space of $\mathbf{A}$ and $\lambda \in \mathbb{R}$ [4].

In addition to creating a testing function to compare every output x, by calculating the percentage difference between $\mathbf{A}x$ and b, comprehensive additional testing was carried out for each algorithm. Random non-singular matrices of a given size were generated by importing *orthogroup* from *scipy.stats*, which were used to compare the calculation time of each method for a given size of matrix. Since the calculation time of the analytical inversion method increases with $\mathcal{O}(n!)$, the dimension of $\mathbf{A}$ was only taken up to $n = 10$, averaging over 100 matrices. The same was done with the other methods, but up to a value of $n = 100$. Additionally, the scaling of the percentage error introduced was also examined in terms of both the size and determinant of the matrix. To examine the error introduced as the matrix become closer to singular the following matrix was studied,

$$\mathbf{M} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & -1 \\ 2 & 3 & k \end{bmatrix}$$

where $det(\mathbf{M}) = k$. The percentage error introduced was plotted for values of $k \in [\,10^{-15}, 1\,]$.

The physics problem discussed before can be solved by inverting a matrix with columns composed of the normalised unit vectors between points of the wires and the position of the weight. This was first done in 2D, and then in 3D, comparing the maximum tension achieved in both cases. Any non physical solutions that return negative tension solutions are set to zero.

III. RESULTS

As seen in Figure 1, the calculation time increases rapidly with dimension for the analytical inverse. When plotted against $n!$ a linear relationship can be found, with equation $y = 8.2 \times 10^{-5}(n!) - 5.9 \times 10^{-2}$ and a residual value (of the least squares fit) of 0.1. An order 3 polynomial function was fitted to the other plots in figure 1, with my own LU decomposition function showing little deviation. The built in LU function, however, shows clear steps in the calculation time.

Figure 2(a) displays a similar relationship between the percentage error introduced and the size of the matrix for both the built in and my own LU decomposition function. The order 3 polynomial fitted shows slight differences between them, with my own function having lower error at $n < 20$ but higher at $n > 90$. The percentage error in the SVD method also scales with n^3, as expected since the number of operations for both goes as $\mathcal{O}(n^3)$, however it stays in the order of 10^{-8}, which is at the maximum of the range of LU decomposition. The same comparison was made with the analytical inverse, showing a general increase in percentage error with dimension, however as

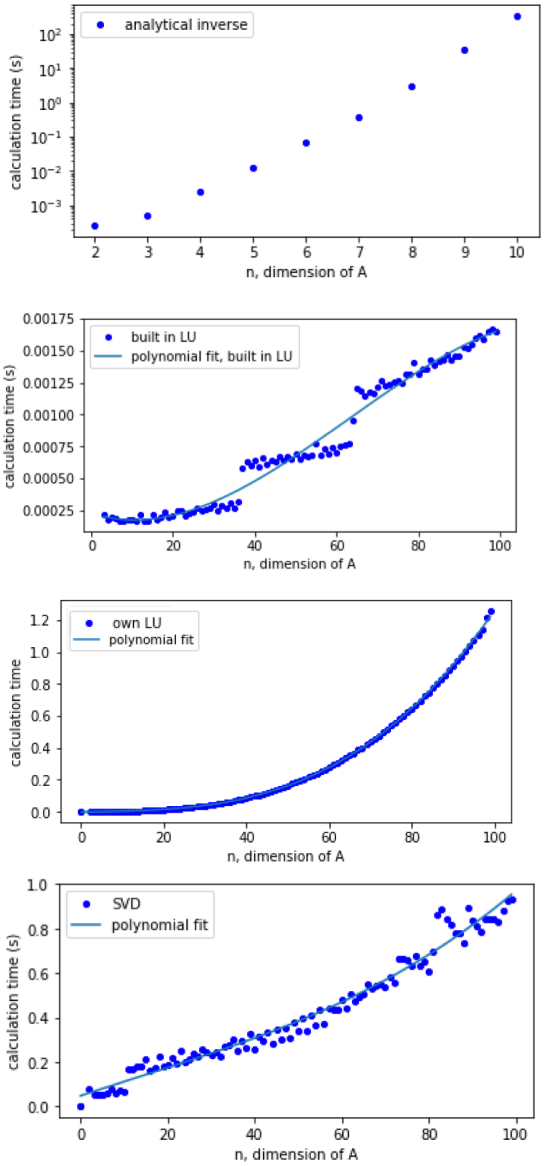

FIG. 1. (a) Plot of calculation time against the dimension of A for the analytical inversion method. (b) Same plot for the built in LU function, with an order 3 polynomial fitting. (c) Plot of calculation time against dimension of A for my own LU decomposition with an order 3 polynomial fitting. (d) Plot of calculation time against dimension of A for SVD decomposition, with an order 3 polynomial fitting.

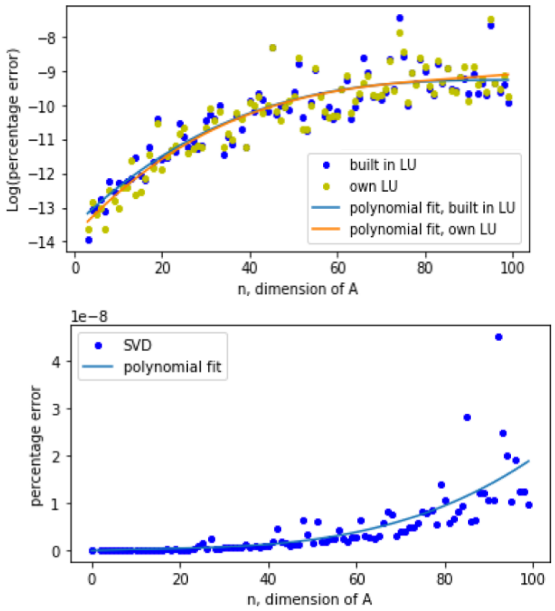

FIG. 2. (a) Plot of log(error) against the dimension of A, comparing my own and the built in LU. (b) Plot of percentage error against dimension of A for an SVD decomposition. All with an order 3 polynomial fitting.

the range of n was much smaller no clear relationship could be fitted.

Figure 3(a) shows two different linear relationships between the determinant of **A** and the percentage error introduced, at det(A) $\sim 10^{-15}$ there is a difference of $\sim$ 15 magnitudes between the percentage errors. The same plot for both LU decomposition functions showed very little relationship between the two, with the percentage error of the built in function less than 10^{-16}, and no percentage error on my own LU decomposition function, according to the floating point accuracy of the program. Figure 3(b) and (c) contrasts the percentage errors seen with and without a threshold value in place, acting as described in the above method section. As before in the analytical inversion method, there is a clear linear relationship between the percentage error and determinant, which is disrupted when a threshold value is introduced, stopping the percentage error from getting above $\sim 10^{-2}$.

Figure 4(a) shows all points in the x-z plane are allowed positions for the weight, with a maximum tension of 2598 N. In contrast, areas where the tension is set to zero in figure 4(b) are non physical solutions, indicating positions where the weight cannot be. In this particular z slice, the maximum tension is 743 N, the overall maximum in the 3D case is 2598N.

IV. DISCUSSION

As expected, the analytical inverse function is impractical both in terms of calculation time and the errors produced. All 3 decomposition functions significantly outperform it in both of these categories. Whilst my own LU decomposition function produces similar errors to the built in one, as seen in Figure 2(a), it is significantly slower. One way to improve the speed could be to create test to see whether a permutation matrix is necessary, by looking at the diagonal elements of **A**, rather than using one in every case.

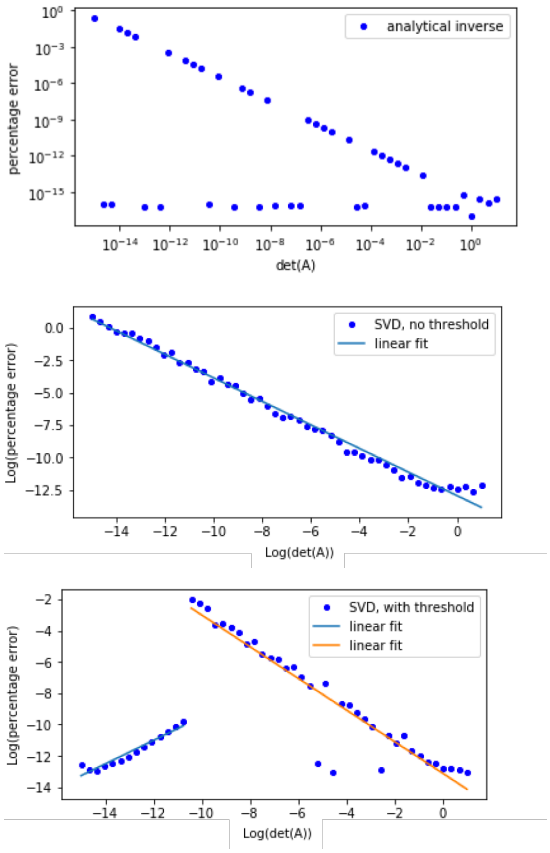

FIG. 3. (a) Plot of log(error) against the determinant of A for the analytical inverse (b) Same plot for SVD decomposition, without a threshold value included and a linear fitting. (c) Plot with a threshold value included and two separate linear fittings.

SVD decomposition produces significantly higher percentage errors than the LU functions, both when looking at how the errors scale with the size of the matrix and the determinant. The addition of a threshold value when computing Σ^+ is useful for reducing the maximum error introduced, and could potentially be increased. The calculation time is similar to that of my own LU function. However, it is by far the most versatile function as it puts no restrictions on the rank of **A**, and will provide least squares solution in both under and over determined cases.

For these reasons, LU decomposition is the most suitable method if you require exact solutions to equation 1, or know that **A** is full rank, whereas SVD decomposition is more appropriate when approximations to solutions, or a space of solutions is desired.

An interesting application of these methods is demonstrated in the physics problem. In the range of values used, the matrix to be inverted is non singular, allowing it to be inverted and the tension plotted. An interesting extension would be exploring different ways of displaying

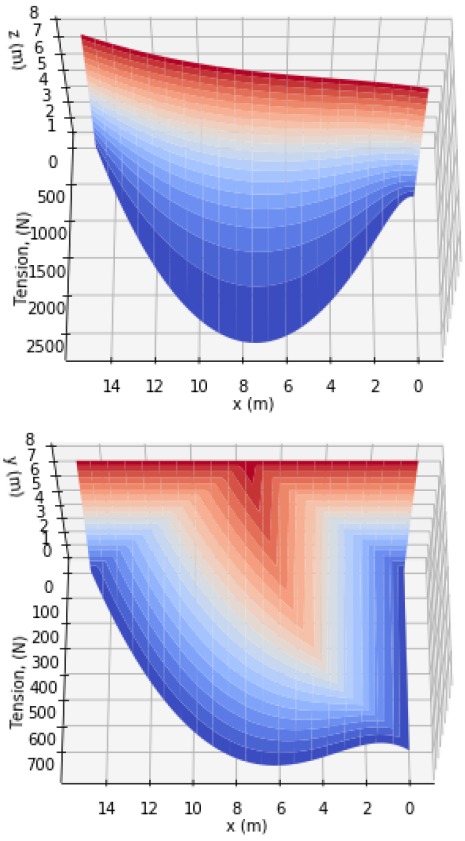

FIG. 4. (a) Plot of tension in the x-z plane for the 2D case. (b) Plot of tension in the x-y plane for the specific case of z = 4m.

the 3D case, without just plotting a slice for a particular value of z. This could be done using an animation looping though different values of z. Further, you could investigate how many wires would be needed to make every position on the stage accessible.

[1] R. L. Burden and J. D. Faires. *Numerical Analysis*. 1980.

[2] C. Moler and Cleve. Cramer's rule on 2-by-2 systems. *ACM SIGNUM Newsletter*, 9(4):13–14, 10 1974.

[3] C. B. Moler and Society for Industrial and Applied Mathematics. *Numerical computing with MATLAB*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2008.

[4] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, and H. Gould. *Numerical Recipes, The Art of Scientific Computing*, volume 55. 1987.

[5] L. N. Trefethen and D. Bau. Numerical linear algebra. In *Numerical Linear Algebra*, pages 105–137. 2012.