Computational Physics Exercise 1

Paul Hayes
Level 6 Computational Physics, School of Physics, University of Bristol.
(Dated: February 13, 2018)

INTRODUCTION

A set of simultaneous equations can be solved by representing them as a matrix in the form,

$$A\mathbf{x} = \mathbf{c}, \tag{1}$$

where A is a matrix written as,

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & ... & A_{1n} \\ A_{21} & A_{22} & A_{23} & ... & A_{2n} \\ A_{31} & A_{32} & A_{33} & ... & A_{3n} \\ ... & ... & ... & ... & ... \\ A_{n1} & A_{n2} & A_{n3} & ... & A_{nn} \end{pmatrix}$$

x is a vector of unknowns and c is a vector of constants, then inverting the matrix A to find the vector of unknowns x,

$$\mathbf{x} = A^{-1}\mathbf{c}. \tag{2}$$

TASK 1

Here, a matrix inversion algorithm was written to invert the matrix A analytically. Matrix inversion follows the general formula,

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

where det(A) is the determinant of A and C^T is the transpose of the matrix of cofactors.

Method

The algoritm is made up of three functions, one finds the determinant of an nxn matrix, one finds a matrix of minors and the third computes the inverse of an arbitrary nxn matrix.

The matrix minor function will delete row i and column j and output a matrix of size (n-1,n-1). This is done by the np.delete function. The determinant function computes the determinant by following the general determinant formula of a 2x2 matrix if the size is equal to 2, if not it will sum the deteriminants along a given row using a for loop.

The inverse function will output the inverse of a 1x1 by simply writing (1/number). If the matrix is 2x2 it will immedietly find the determinant of a 2x2 using the determinant function, and return the inverse by dividing the input matrix by this value. If the matrix is greater than 2x2, the function will find a matrix of cofactors using a nested loop to compute the determinants of the minor matrices. This matrix of cofactors is then divided by the determinant of the original input nxn array.

Analysis

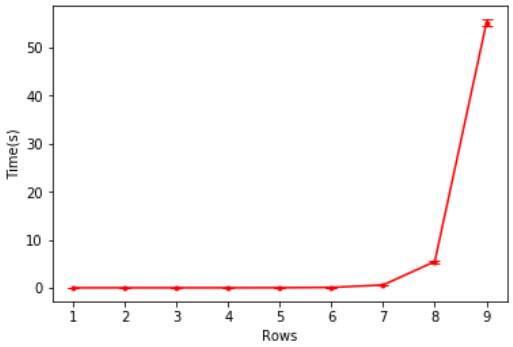

FIG. 1. Computational time of algorithm computing an inverse against number of rows in matrix, error bars are the standard deviation in time at a given row.

The inverse of a given nxn matrix showed values that were in agreement with that of known values. This shows the algorithm was successful in carrying out its function.

The time at which the algorithm computes the inverse shows an n! relationship with the size of the matrix, n. This is seen in FIG.1. For computing larger matrices, such as beyond 10, a different method should be used as it is not convenient to have to wait over an hour for such inverses, and indeed it will increase dramatically for even higher n, as described by the n! relationship.

TASK 2

This section compares the methods of solving simultaneous of Task 1's algorithm to LU decomposition (LU) and Singular Value Decomposition (SVD).

Method

In LU, the matrix A is rewritten as a sum of two matrices[1],

$$A = L.U \tag{4}$$

where L is the lower triangular matrix, where it only has values on and below the diagonal, and U is the upper triangular matrix, where it only has values on and above the diagonal. Using EQ.4, EQ.1 can be solved by writing,

$$A.x = (LU).x = b \tag{5}$$

then solving for x,

$$x = U^{-1}L^{-1}b. \tag{6}$$

In SVD, A is written as,

$$A = U\Sigma V^T, \tag{7}$$

where U and V are orthonormal matrices and D is diagonal matrix of singular values. This can then be used to calculate x by,

$$\bar{x} = V\Sigma^{+}U^T c, \tag{8}$$

where Σ^{+} is found by taking the inverse of each value that is non zero in Σ and then transposing the result, and when A is non-singular $\bar{x} = x$.

Analysis

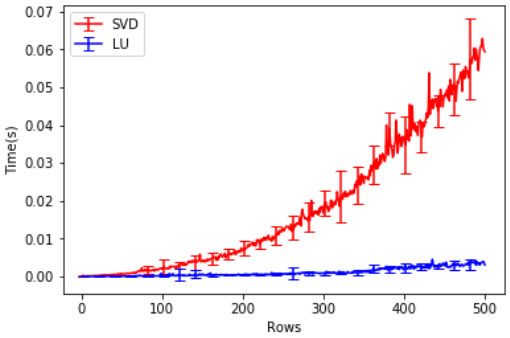

FIG. 2. Time for solving against matrix size for maximum matrix size 500, error bars are the standard deviation in time for a given row, seen at every other 20 rows for clarity.

From FIG.2, it is clear that SVD is slower at solving a set of arbitrary simultaneous equations than LU. Indeed, an exponential relationship is seen in both, yet SVD clearly has a steeper gradient. If speed is important to the user, or larger simultaneous equations, LU should be used.

SVD is useful for when sets of equations or matrices are singular or close to singular, where LU decomposition fails to work.

TASK 3

The methods previously investigated for solving simultaneous equations are now applied to a physical problem. A camera is suspended over a football pitch by 3 wires each individually attached to a pole. The tension in the wires is plotted as a function of the cameras position (x,y,z) for a constant z=-7m.

Method

To solve this problem, a simultaneous equation is produced as a function of the cameras position (x,y,z). The simultaneous equation will have the form

$$Ax = C, \tag{9}$$

where x is the vector (T_1, T_2, T_3) where T_1, T_2, T_3 are the tensions from poles 1, 2 and 3 respectively, C is the vector of resolved forces in x, y and z which has values (0,0,mg) and A is the 3x3 matrix:

$$\begin{pmatrix} \frac{x_1rel}{\sqrt{x_1rel^2+y_1rel^2}^2} & \frac{x_2rel}{\sqrt{x_2rel^2+y_2rel^2}^2} & \frac{x_3rel}{\sqrt{x_3rel^2+y_3rel^2}^2} \\ \frac{y_1rel}{\sqrt{x_1rel^2+y_1rel^2}^2} & \frac{y_2rel}{\sqrt{x_2rel^2+y_2rel^2}^2} & \frac{y_3rel}{\sqrt{x_3rel^2+y_3rel^2}^2} \\ \frac{7}{\sqrt{x_1rel^2+y_1rel^2+49}} & \frac{7}{\sqrt{x_2rel^2+y_2rel^2+49}} & \frac{7}{\sqrt{x_3rel^2+y_3rel^2+49}} \end{pmatrix}$$

Where $(x_1rel, x_2rel, ...)$ are $(x-x1, x-x2, ...)$ where x_1, x_2 are the positions of poles 1 and 2 and x is the position of the camera in the x axis.

By inverting matrix A using previous methods in Task 1 and 2 it is possible to find the tensions in vector x.

First a matrix resolver function was made (resolvematrix(x,y,z)), this generates the 3x3 array of values at a value of the camera's position (x,y). Next a simultaneous equation solver function (solvesimtask3(A), where A is the matrix) was used to solve for T1,T2,T3 using the C matrix previously stated. This used LU, SVD and the analytical methods. np.dot, np.transpose and np.diag functions were used. These Tensions are then used to plot surface and 3D plots of the tensions as the camera moves around (x,y). A meshgrid is used to generate the axes. Arctan(y/x) ¡ 60 degrees was used to bound the plot to values allowed within the equilateral triangle, seen in FIG.1.

Analysis

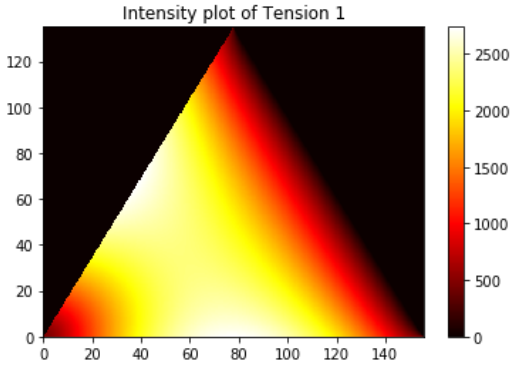

FIG. 3. Tensions in wire one holding camera as a function of (x,y).

The highest tension in wire 1 is 2739N at x= 116.3, y=67.4 and x=39.7, y=67.4 from code that finds these values..

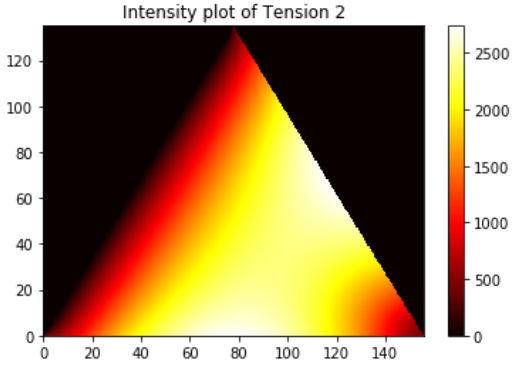

FIG. 4. Tensions in wire two holding camera as a function of (x,y).

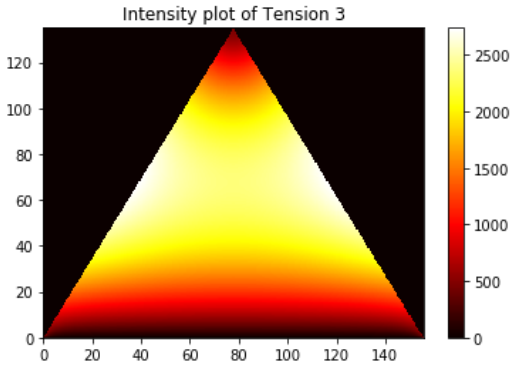

FIG. 5. Tensions in wire three holding camera as a function of (x,y).

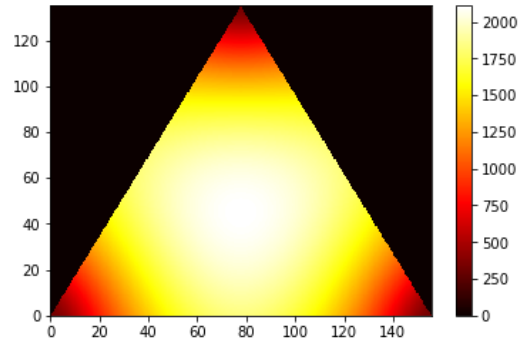

FIG. 6. Average tensions in all 3 wires holding camera as a function of (x,y).

If FIG.3, FIG.4, FIG.5 and FIG.6 are correct, an equilateral triangle should be seen. This is indeed the case, confirming the validity of the results seen.

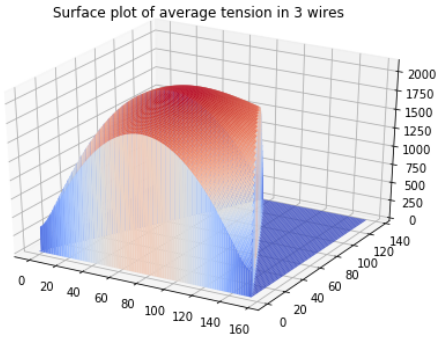

FIG. 7. 3D plot of the average tensions in all 3 wires as a function of camera's position (x,y).

CONCLUSIONS

In conclusion, the analytical matrix inverse function was far slower to calculate a larger matrix (n¿5) than LU or SVD. SVD was also slower than LU, and as such LU should be used for larger or if the user requires fast processing. The expected graph was obtained for task 3, an equilateral triangle of tensions in each wire.

REFERENCES

[1] William H. Press, Saul A.Teukolsky, William T. Vetterling, Brian P. Flannery, Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, Third Edition (2007)..