Task 1: Inverting Matrices - Cramer's Rule

Although not a particularly effective method for numerical matrix inversion, the analytical formula known as Cramer's rule may be implemented in a program through recursion. The program task1.c uses this method to calculate the inverse of an N x N matrix.

Cramer's rule states that for a square matrix **A**:[1]

$$A^{-1} = \frac{1}{\det A} C^T \tag{1.1}$$

Where $\mathbf{C}^T$ is the transpose of the matrix of cofactors of **A**. The cofactor element ij is given by:

$$C_{ij} = (-1)^{i+j} \det M_{ij} \tag{1.2}$$

Where $\mathbf{M}_{ij}$ is the minor matrix corresponding to the ij element of **A**. Determinants have been calculated in the program via expansion of the top row so that:

$$\det A = \sum_{n=1}^{N} a_{1n} C_{1n} \tag{1.3}$$

My program contains a number of separate functions described below.
Creating (and freeing) matrices: uses the pointer to array of pointers to rows method (borrowed from nrutil.c[2]) which allows the elements of the produced matrices to be addressed using a[i][j] without knowing the size of the matrix at runtime[3]. The matrices are produced with indices [1 to n] rather than C's regular [0 to n-1] which is preferred as it reflects regular matrix indexing.
Producing minor matrices: produces the ij minor matrix for the matrix passed to the function. This is used prior to calculating cofactor elements in both the cofactor matrix and the determinant row expansions.
Calculating determinants: if the size of the matrix passed to this function is 2 it will calculate the determinant - otherwise it will expand the matrix recursively using (1.3), breaking it down until n=2.

This recursive calculation requires o(n!) steps (and therefore the time it takes scales with n!) since for example a 4x4 matrix will be expanded to 4*3x3 matrices which are then each expanded to 3*2x2 matrices before the determinant can be calculated.

The accuracy of this method is severely limited by the final calculation of the 2x2 determinants. This is because the calculation a[1][1]*a[2][2] – a[1][2]*a[2][1] becomes unstable and prone to round off errors whenever these two numbers are close to equal[4]. This then depends on the values of the specific matrix; an error message will be printed whenever this calculation produces a value of magnitude smaller than 10^{-6}, close enough to the calculation precision to induce errors.

This could be alleviated somewhat by searching for an optimum path through the matrix[5] – i.e. expanding by rows or columns which produce the least sensitive cofactors, however this would be difficult and time consuming to implement.

Task 2: SVD vs. LUD vs. Cramer's

Two methods that are more stable and take far less time than the above are SVD - singular value decomposition and LUD – lower upper decomposition. The programs SVD.c and LUD.c implement the GSL (GNU scientific library) functions[6] for these algorithms.

To compare the speed of the each the algorithms and to measure how computation time scales with the size of the matrix <time.h> was used with the programs running a loop – inverting successively larger matrices. The results are presented in fig 2.1.

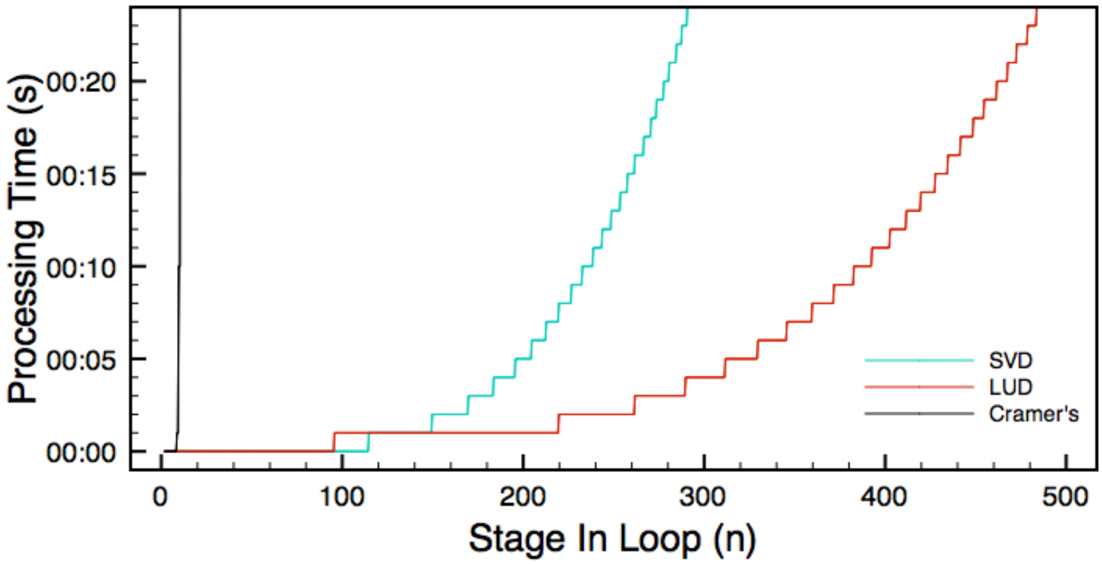

Fig 2.1 – The total time taken for each of the algorithms to invert successively larger matrices.

Cramer's algorithm took 10s to reach n=10 then 120s to reach n=11 showing the n! scaling. SVD and LUD were both far quicker, reaching 300 and 500 within 27s respectively. Using large n, the scaling's for these have been found to be approximately n^3 for SVD and $2n^3/3$ for LUD which is supported by [7].

The behaviour of the solutions of each has been tested as the matrix being inverted approaches becoming singular. This was done using matrix 2.1, which is singular when k=0.

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & -1 \\ 2 & 3 & k \end{pmatrix} \quad (2.1)$$

As k approaches zero (in increments of 10^{-6}) the elements of the inverse matrix go to infinity. There was found to be no difference in the behaviours of the algorithms, as shown in fig 2.2.

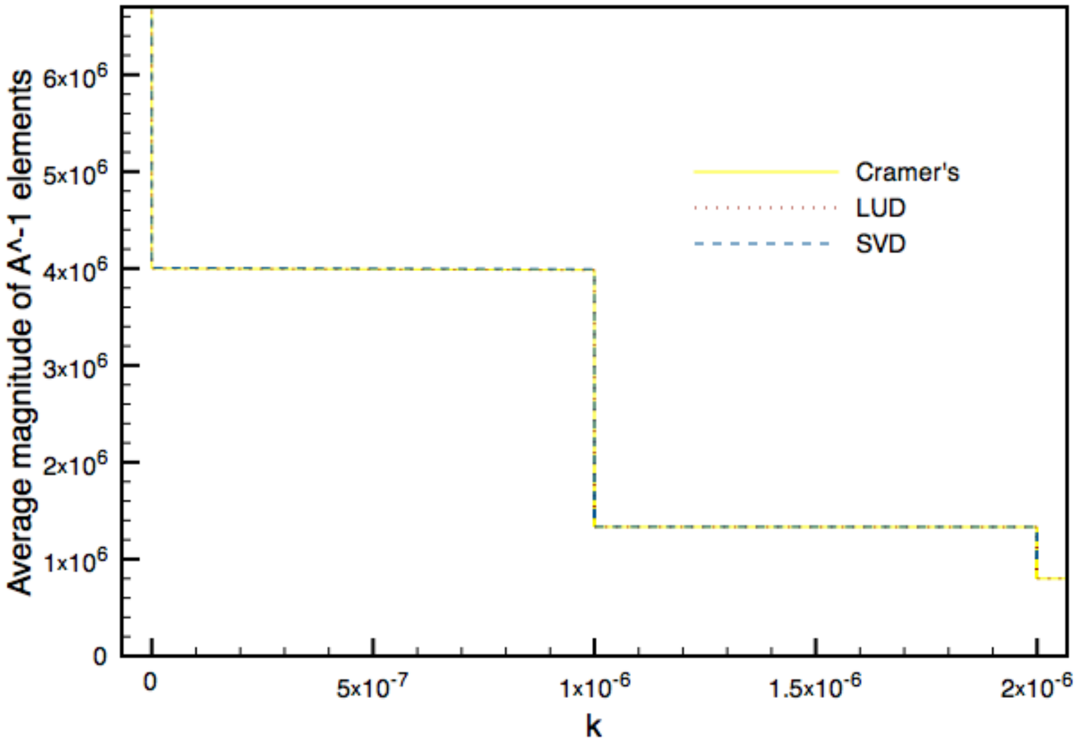

Fig 2.2 – The average magnitude of the elements of the inverse of eqn 2.1 as it becomes singular.

Task 3: Cables Problem

A 50kg camera hangs suspended from 3 cables (of negligible mass), 10m below the attachments as shown in fig 3.1. The lengths of the cables may be adjusted to allow the camera to move around in the x-y plane.

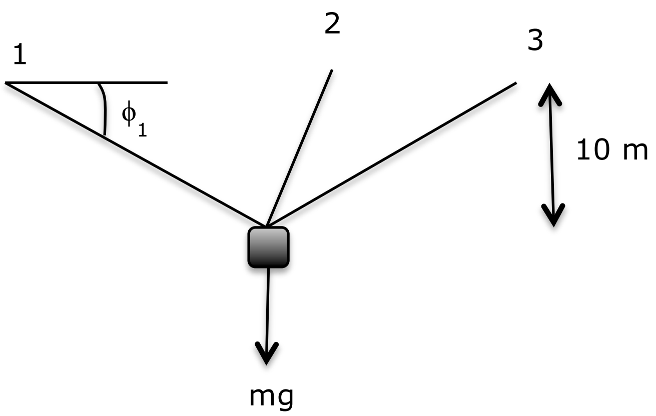

Fig 3.1 - The system viewed in the z plane.

We are told that the attachment points are all 70 m from the centre of the system, this specifies the dimensions of the possible x-y locations of the camera as indicated by the dotted lines in fig 3.2.

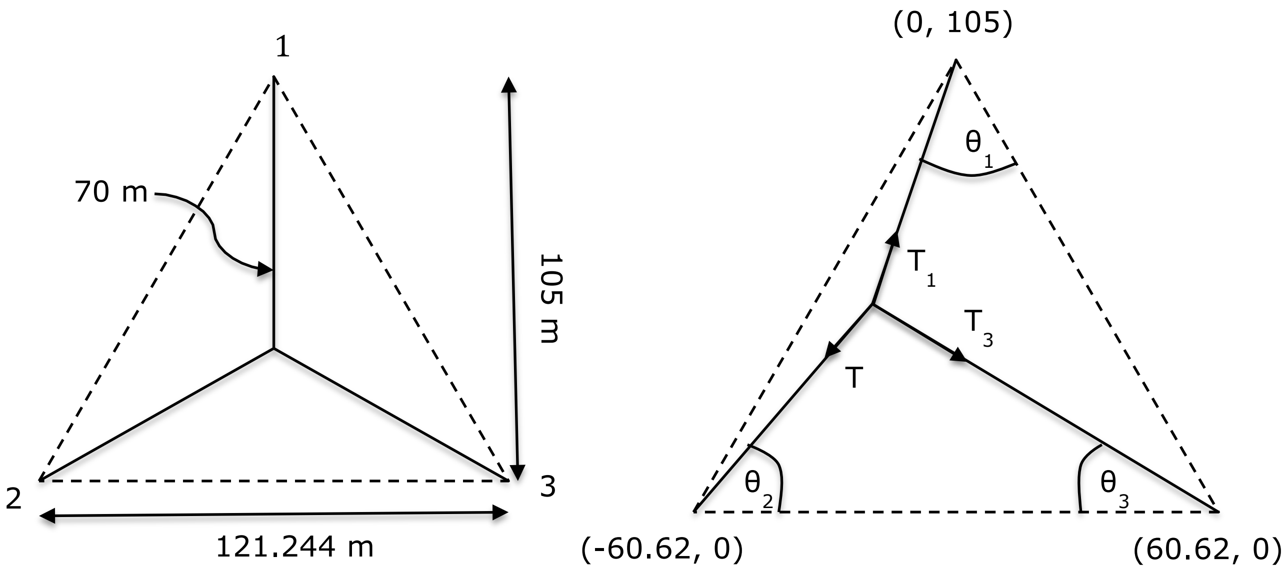

Fig 3.2 – Birds eye view of the system. The attachment points have been indexed (left) and their x, y coordinates in m are shown (right).

Using the angles defined in the above figures the total force on the camera may be resolved in each direction as follows:

$$\begin{aligned} x &: T_1 \sin(\theta_1 - 30°) - T_2 \cos\theta_2 + T_3 \cos\theta_3 = 0 \\ y &: T_1 \cos(\theta_1 - 30°) - T_2 \sin\theta_2 - T_3 \sin\theta_3 = 0 \\ z &: T_1 \sin\phi_1 + T_2 \sin\phi_2 + T_3 \sin\phi_3 = mg \end{aligned} \tag{3.1}$$

Which can be written in matrix form:

$$\begin{pmatrix} \sin(\theta_1 - 30°) & -\cos\theta_2 & \cos\theta_3 \\ \cos(\theta_1 - 30°) & -\sin\theta_2 & -\sin\theta_3 \\ \sin\phi_1 & \sin\phi_2 & \sin\phi_3 \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} \tag{3.2}$$

Using trigonometry these angles may be computed in terms of the camera's position (x_c, y_c) to be:

$$(\theta_1 - 30°) = \arctan\left(\frac{x_c}{105 - y_c}\right), \theta_2 = \arctan\left(\frac{y_c}{60.62 + x_c}\right),$$

$$\theta_3 = \arctan\left(\frac{y_c}{60.62 - x_c}\right), \phi_i = \arctan\left(\frac{-10}{\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2}}\right)$$

(3.3)

My program then loops over all possible values of (x_c, y_c):

$$0 \le y_c \le 105$$

$$\frac{(y_c - 105)}{\sqrt{3}} \le x_c \le \frac{(105 - y_c)}{\sqrt{3}} \tag{3.4}$$

Using a step size of 0.5 m. The matrix above is solved using LU decomposition, rather than the SVD, as this will produce a faster computation.

Figure 3.3 and 3.4 show plots of the tension in cable one as a function of the cameras position. The maximum tension was found to occur at (-30.02, 53.0). There is point of equal tension at (+30.02, 53.0).

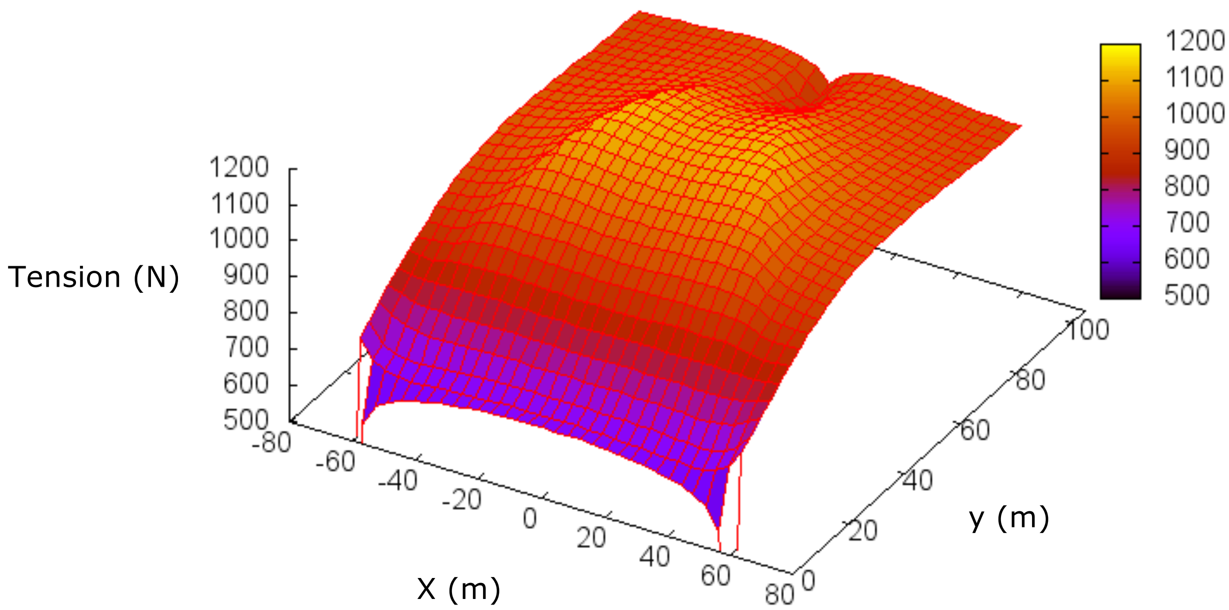

Fig – 3.3 A 3d colour plot of the tension in cable 1. The largest values occurs at the centre with the tension going to mg at (0, 105) and 0 at on the line y=0.

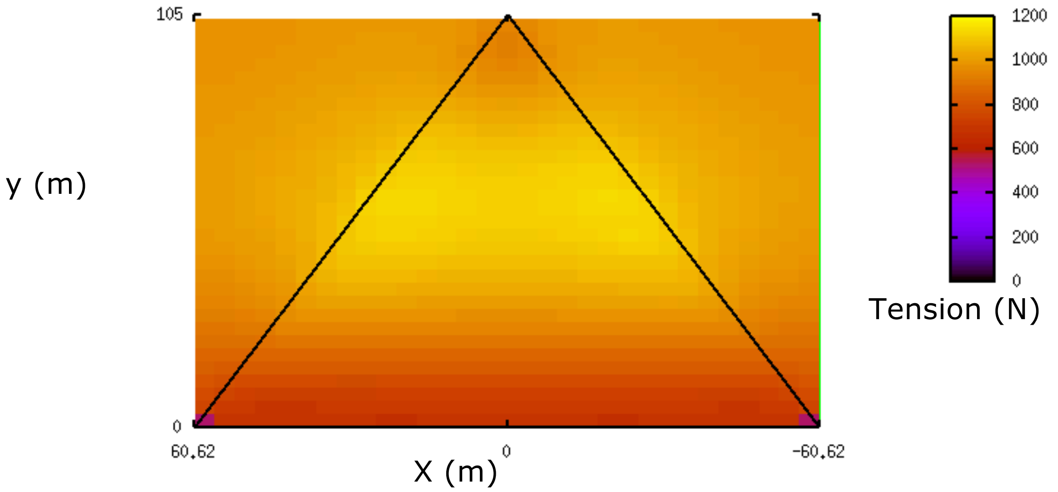

Fig 3.4 – A top down view with the region marked.

---

[1] Mathematical Techniques, D.W. Jordan, P. Smith, Oxford university press. 4th edition (2008) §7.4

[2] Numerical Recipes in C, W.H. Press et al., Second Edition
http://www.nr.com/pubdom/nrutil.c.txt

[3] Numerical Recipes §1.2

[4] Numerical Recipes §1.3

[5] Fundamental Numerical Methods and Data Analysis, G. W. Collins II, (2003)
http://ads.harvard.edu/books/1990fnmd.book/

[6] GNU Scientific Library Reference Manual, Edition 1.0, for GSL Version 1.0 (2001)

[7] Numerical Recipes §2.3 & §2.6