Computational Methods and Modelling

Antonio Attili & Edward McCarthy

antonio.attili@ed.ac.uk
ed.mccarthy@ed.ac.uk

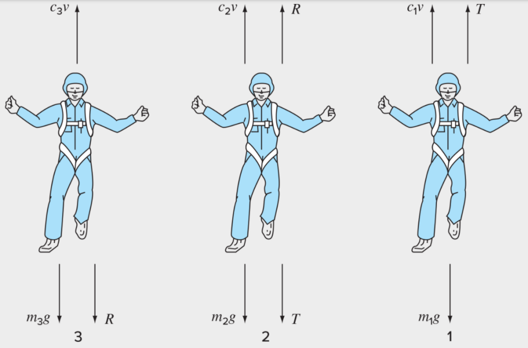
School of Engineering
University of Edinburgh
United Kingdom

Solution tutorial 6
Gauss Elimination

THE UNIVERSITY of EDINBURGH

Exercise 1: Calculate Cord Tensions in a Tandem Team of Parachutists

- We firstly write Newton's Second Law for each of the three parachutists:



- Here, the accelerations of the three parachutists are each expressed as a balance of gravitational force ($m_i g$), and the tension force on each parachutist is R or/and T.
- The drag force exerted by air against direction of fall is cv, where c_i are the drag coefficients and v the velocity.

Exercise 1: Calculate Cord Tensions in a Tandem Team of Parachutists

- These three equations are linear and together comprise a linear equation system that can be solved using Gauss:

$$
\begin{aligned}
m_1 g - T - c_1 v \qquad &= m_1 a \\
m_2 g + T - c_2 v - R &= m_2 a \\
m_3 g \qquad - c_3 v + R &= m_3 a
\end{aligned}
$$

- Filling in all the values we were provided with we get the following:

$$
\begin{bmatrix} 70 & 1 & 0 \\ 60 & -1 & 1 \\ 40 & 0 & -1 \end{bmatrix} \begin{Bmatrix} a \\ T \\ R \end{Bmatrix} = \begin{Bmatrix} 636.7 \\ 518.6 \\ 307.4 \end{Bmatrix}
$$

- This is equivalent to a system $A \cdot x = b$. We solve for x.

Exercise 1: Calculate Cord Tensions in a Tandem Team of Parachutists

- Python code to solve a linear system of equation

'GaussAF.py'

```
import numpy as np

def linearsolver(A,b):
    n = len(A)

    #Initialise solution vector as an empty array
    x = np.zeros(n)

    #Join A and use concatenate to form an
    #augmented coefficient matrix
    M = np.concatenate((A,b.T), axis=1)

    for k in range(n):
        for i in range(k,n):
            if abs(M[i][k]) > abs(M[k][k]):
                M[[k,i]] = M[[i,k]]
            else:
                pass
                for j in range(k+1,n):
                    q = M[j][k] / M[k][k]
                    for m in range(n+1):
                        M[j][m] +=  -q * M[k][m]

    #Python starts indexing with 0, so the last element is n-1
    x[n-1] =M[n-1][n]/M[n-1][n-1]

    #We need to start at n-2, because of Python indexing
    for i in range (n-2,-1,-1):
        z = M[i][n]
        for j in range(i+1,n):
            z = z  - M[i][j]*x[j]
        x[i] = z/M[i][i]

    return x

A=np.array([[70., 1., 0],[60., -1., 1.], [40, 0, -1]])
b=np.array([[636.7, 518.6, 307.4]])
print(linearsolver(A,b))
```