Computational Physics and Modelling – Assessment 1

Programming / Debugging Exercise (Eigenvalues of a Matrix)

Elijah Gill

31 October 2025

1. Code Description

The functions created for this exercise can be found in a dedicated python file "Functions.py" which are then imported to the Jupyter notebook to be used. The functions are split into three groups: "General functions", "Functions for QU method" and "Functions for coupled oscillators".

"General Functions" contains the functions "det2( )" and "find_det( )". These functions are used to calculate the determinant of a square matrix of any size. Find_det( ) uses Laplace / Cofactor Expansion to find the determinant of any matrix larger than shape (2x2) by recursively calling itself until it is given a (2x2) matrix to find the determinant of, at which point it calls det2( ). It also contains the function "column( )", which will return a specified column of a given matrix as a row.

"Functions for QU Method" has all the functions that are specifically related to the QU factorisation and the Gram-Schmidt process. The function "find_fs( )" takes a matrix and returns a matrix with a new set of orthogonal columns using the equation,

$$\boldsymbol{f_k} = \boldsymbol{c_k} - \frac{\boldsymbol{c_k} \cdot \boldsymbol{f_1}}{|\boldsymbol{f_1}|^2}\boldsymbol{f_1} - \frac{\boldsymbol{c_k} \cdot \boldsymbol{f_2}}{|\boldsymbol{f_2}|^2}\boldsymbol{f_2} - \ldots - \frac{\boldsymbol{c_k} \cdot \boldsymbol{f_{k-1}}}{|\boldsymbol{f_{k-1}}|^2}\boldsymbol{f_{k-1}} \tag{1}$$

where $\boldsymbol{f_k}$ is a new orthogonal column with index $\boldsymbol{k}$ and $\boldsymbol{c_k}$ is the old column with index $\boldsymbol{k}$. The function "find_Q( )" takes this new matrix and changes it so that the columns take the form,

$$\boldsymbol{q_k} = \frac{\boldsymbol{f_k}}{|\boldsymbol{f_k}|} \tag{2}$$

where $\boldsymbol{q_k}$ is a new column with index $\boldsymbol{k}$. The function "find_U( )" then generates the upper triangular matrix, U, of the form,

$$\boldsymbol{U} = \begin{bmatrix} |\boldsymbol{f_1}| & \boldsymbol{c_2} \cdot \boldsymbol{q_1} & \boldsymbol{c_3} \cdot \boldsymbol{q_1} & \cdots & \boldsymbol{c_k} \cdot \boldsymbol{q_1} \\ 0 & |\boldsymbol{f_2}| & \boldsymbol{c_3} \cdot \boldsymbol{q_2} & \cdots & \boldsymbol{c_k} \cdot \boldsymbol{q_2} \\ 0 & 0 & |\boldsymbol{f_3}| & \cdots & \boldsymbol{c_k} \cdot \boldsymbol{q_3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & |\boldsymbol{f_k}| \end{bmatrix} \tag{3}$$

These individual functions come together in "find_eigenvals( )" to perform the QU algorithm. Firstly, the function checks the inputted matrix to see if it is invertible, as the process does not work otherwise. It does so by checking the matrix is square, this is also a requirement for the determinant functions but as they are called from this main function, they do not require the checks themselves. And the function checks that the determinant of the given matrix is non-zero. If either of these are flagged, the function stops and raises an error with a brief description of the problem. If the given matrix passes the checks the QU algorithm is ran until either the sum of the difference in eigenvalues between passes is less than **1×10^{-9}** or the maximum number of iterations is reached. If the maximum iteration is reached, the function will stop and flag an error stating the eigenvalues failed to converge. If the function successfully runs, it returns a matrix with the eigenvalues on its diagonal, as well as the number of iterations it took to converge.

The final section is "Functions for Coupled Oscillators" and contains the following functions: "find_freq( )" uses the eigenvalues calculated from other functions to calculate the oscillation frequencies for a given system as the eigenvalues are equal to the negative of frequency squared. The function form_mat( ) takes a spring constant $\boldsymbol{k}$ and masses $\boldsymbol{m_1}$ and $\boldsymbol{m_2}$ to generate a matrix that represents a coupled oscillator system with two masses and three springs attached to an immovable wall at either end. The matrix is defined as,

$$M = \begin{bmatrix} -\frac{2k}{m_1} & \frac{k}{m_1} \\ \frac{k}{m_2} & -\frac{2k}{m_2} \end{bmatrix} \tag{4}$$

The final function made for this assignment is the "hand_check( )" function, which solves for the eigenvalues, $\boldsymbol{\lambda}$, of a matrix M in eq(4), using the characteristic equation, and assuming that the masses are identical. Solving this characteristic equation gives the eigenvalues as,

$$\lambda = -\frac{2k}{m} \pm \frac{k}{m} \tag{5}$$

This function was used to easily test that the QU method was accurately finding the eigenvalues of the matrix correctly in the context of a coupled oscillator system.

The Jupyter Notebook intends to demonstrate the code working to find eigenvalues, debugging work and the code's validation. As well as showing how the functions were used to generate plots for different coupled oscillator systems.

2. Code Validation

The find_eigenvals( ) function was validated by calculating the eigenvalues for multiple matrices, with shapes ranging between (2x2) and (5x5), both by hand using the characteristic equation, and checking the result with what the function outputted.

Matrix	Characteristic Eigenvalues	Code Eigenvalues	Iterations
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	5.3723, -0.3723	5.372281323273059, -0.3722813232730577	9
$\begin{bmatrix} -1 & 0.5 \\ 4 & 5 \end{bmatrix}$	5.3166, -1.3166	5.316624790426178 -1.3166247904261765	17
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	14.0242, -7.9475, -1.0766	14.024176844478077, -7.947534534480645, -1.0766423099974347	38
$\begin{bmatrix} 3 & 0.4 & 2 \\ 1 & 9 & 1.5 \\ 2 & 8 & 6 \end{bmatrix}$	11.7109, 4.1137, 2.1754	11.710891773586603, 4.113721984103633, 2.175386242309771	34
$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix}$	6, -1.4142, 1.4142	N/A	1000

$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \\ 4 & 0 & 1 & 2 \\ 3 & 1 & 2 & 0 \end{bmatrix}$	8.0769, -3.0645, -2.0898, 0.0773	8.076882350030697, -3.0644585620067435, -2.0897571105929176, 0.07733332256897213	48
$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 2 & 4 & 5 \\ 4 & 0 & 1 & 2 & 5 \\ 3 & 1 & 2 & 0 & 0 \\ 5 & 4 & -3 & -1 & 9 \end{bmatrix}$	13.6971, -4.0860, 3.0219, -2.1000, 1.4669	13.69710725440928, -4.085992900592491, 3.021935446053917, -2.099982919587095, 1.4669331197163964	78

Table 1: Shows the eigenvalues for a particular matrix, calculated using both the characteristic equation (2nd column) and the find_eigenvals( ) function (3rd column). These matrices are provided in the Jupyter notebook.

The eigenvalues generated using code shown in table 1 are all identical to the precision that the characteristic equation eigenvalues were calculated to. This suggests that the find_eigenvals( ) function works as intended as long as the eigenvalues are able to converge. The number of iterations required to converge increases with matrix size, but successful convergence only requires a relatively small number of iterations for a precision of $\mathbf{1\times10^{-9}}$.

A more rigorous check was done using the hand_check( ) function with all the matrices having the form seen in eq(4). A batch of ten different spring constants, $\boldsymbol{k}$, and masses, $\boldsymbol{m}$, values were made, and these were used to create ten different (2x2) matrices to test with. This method allowed for quick generation of the eigenvalues using both methods and easier comparison between them. Eigenvalues generated using this method all agreed with each other to the specified precision of the find_eigenvals( ) function. The code for this, and resulting comparisons, can be found in the Jupyter notebook.

Debugging work was also implemented into the functions, these were typically sanity checks to ensure that the inputs given to a function would not lead to the code breaking due to an edge case it cannot handle. Examples of this include checking the size and determinant of a matrix to ensure it is invertible (seen in the find_eigenvals( ) function) and a check in the column( ) function to ensure that the index given correctly matches to an available column in the given matrix. As smaller functions like find_fs( ) are only called after relevant checks have been completed by the find_eigenvals( ) function, they do not need to perform the checks again for themselves, so nothing needed implementing for these functions. Examples that trigger these sanity checks are provided in the Jupyter notebook.

Some matrices do not work well with the QU method implemented; one such matrix can be seen in the fifth row in table 1. The find_eigenvals( ) function was unable to converge onto a set of eigenvalues, upon further investigation it was found that the QU algorithm ended up oscillating between two different matrices, one with the correct eigenvalues and the other with a random set of numbers.

3. Results

Using the functions to find the oscillation frequencies for a given system, plots of oscillation frequency against mass were created.

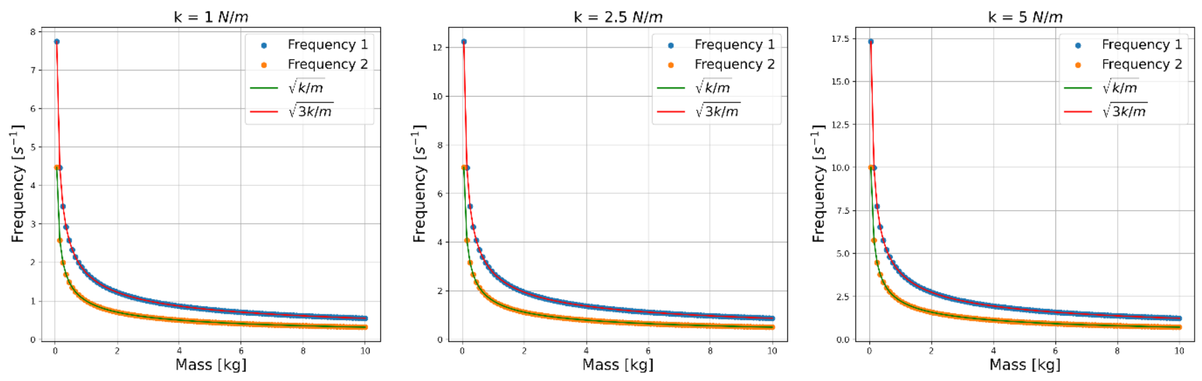

Figure 1: Plots showing oscillation frequencies against mass for three different spring constants. The two masses are identical for these plots.

From the plots in figure 1, the normal modes can be seen to be equal to $\sqrt{\frac{k}{m}}$ and $\sqrt{\frac{3k}{m}}$. Meaning that the oscillation frequency is inversely proportional to the square root of mass, for two identical masses connected to three springs.

Plotting for the case where one mass is constant, but the second mass varies,

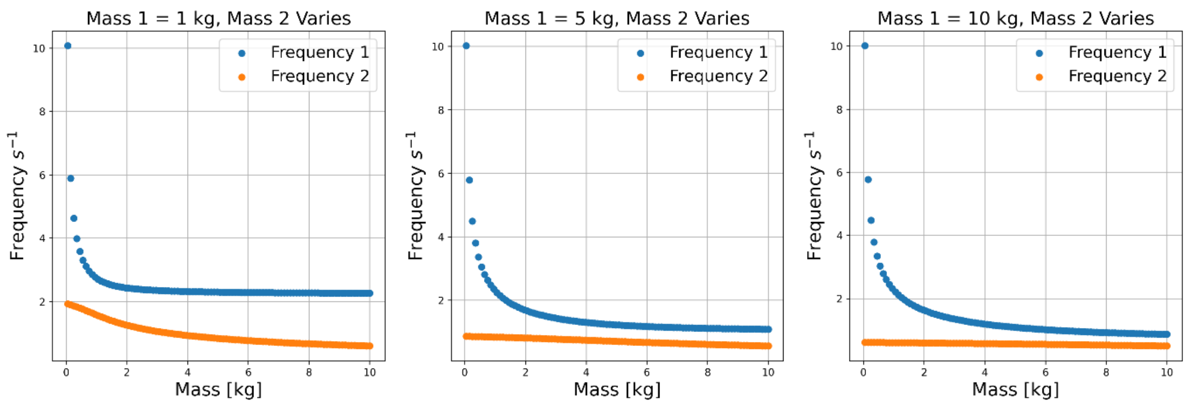

Figure 2: Plots showing oscillation frequencies against mass where one mass is constant. Plots arranged from lightest constant mass on the left to the heaviest on the right.

From the plots in figure 2, it can be seen that if one of the masses is much heavier than the other, both of the normal modes frequencies are decreased. The greater the mass discrepancy also causes one of the model frequencies of the system to tend to a constant value.

4. Extension

Due to the find_eigenvals( ) function working for any size matrix, the couple oscillator problem can be scaled up to contain three masses attached to four springs bound between two immovable walls. For this problem, the new matrix that needs to be calculated has the form,

$$M = \begin{bmatrix} -\frac{2k}{m_1} & \frac{k}{m_1} & 0 \\ \frac{k}{m_2} & -\frac{2k}{m_2} & \frac{k}{m_2} \\ 0 & \frac{k}{m_3} & -\frac{2k}{m_3} \end{bmatrix} \tag{6}$$

The function “form_mat3x3( )”, which can be found in the “Functions for Coupled Oscillators” section of “Functions.py”, will return a matrix in the form of matrix M from eq(6).

All the functions related to finding the eigenvalues of a matrix and converting eigenvalues to frequencies work with any square matrices inputted, so no changes need to be made to the functions to solve for this more complex case.

Plotting oscillation frequencies as a function of mass for the three masses case gives,

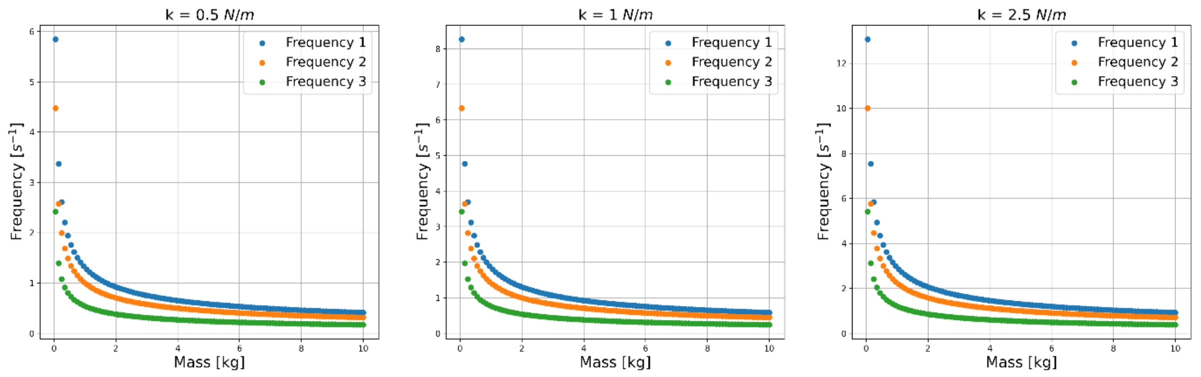

Figure 3: Plots showing oscillation frequencies against mass for three different spring constants. All three masses are identical for these plots and vary at the same rate.

The plots in figure 3 maintain the relationship that normal modes oscillation frequencies are inversely proportional to the square root of mass, even in a more complicated system.