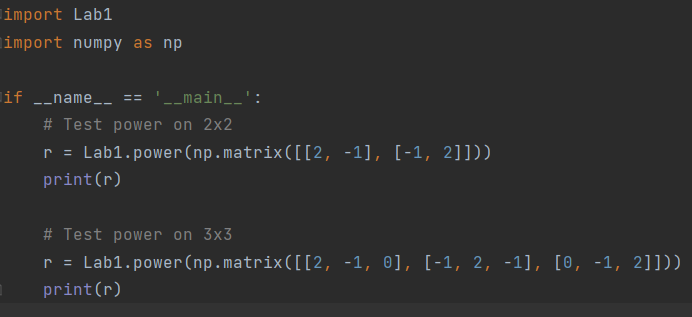
# **Lab 1 – EngSci 311 – Daniel Clark – 343733502**

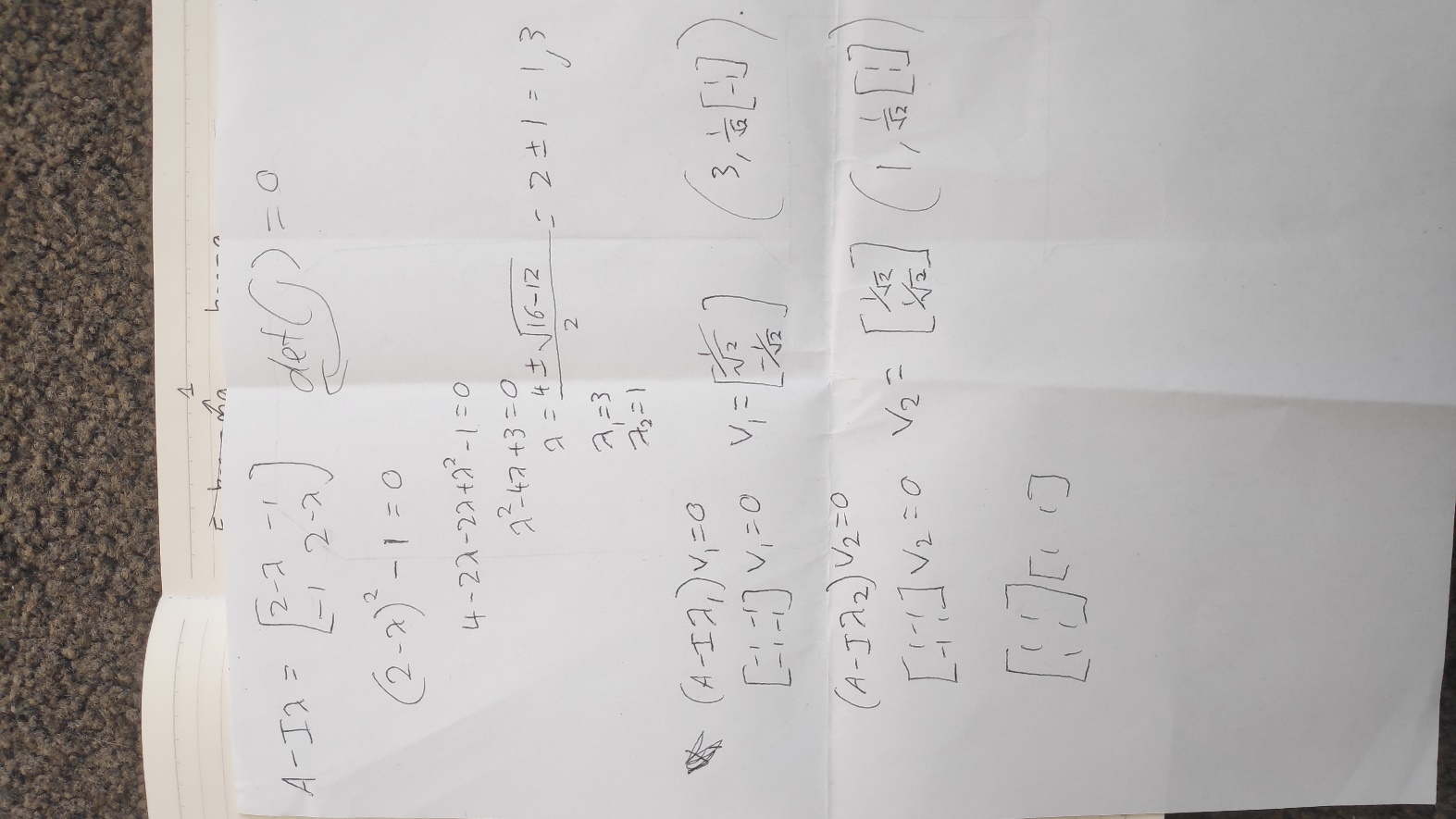
Output:

1. Verification of both the power and power\_w\_deflate scripts by on the matrix given in Question 1 and 3 ×3 symmetric positive definite (SPD) matrix with distinct eigenvalues. For the 3 ×3 use either analytic (pen and paper) methods or an independent eigenvalue/eigenvector solver such as numpy.linalg.eig. (I used wolfram alpha)

To Test my ‘power’ function, I created the following test module, to test it with the 2x2 matrix we were given, and a 3x3 SPD matrix which I found online.



To verify my results from the 2x2 matrix, I calculated the eigenvalues and eigenvectors by hand:



To verify my results from the 3x3 matrix, I used wolfram alpha:

A close-up of a logo

Description automatically generated

Which gave the following results:

A screenshot of a math equation

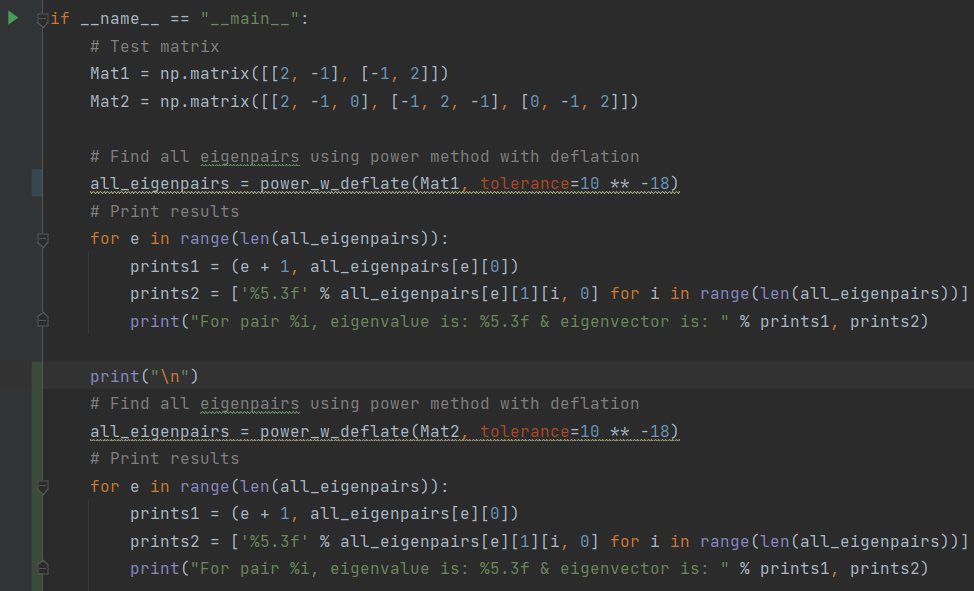
Description automatically generated

A computer screen shot of a code

Description automatically generated

The results for the 2x2 matrix are correct (aside from negligible numerical error) as the eigenvalue given is effectively equal to 3, and the vector is [-1/root(2),1/root(2)], which is simply -1 times the result I found by hand, so is also correct. The results for the 3x3 matrix are also correct, as 3.414=1+root(2), and[ 0.5, -1/root(2), 0.5] is the **normalised** version of eigenvector 1 given by wolfram alpha.

To test the power\_w\_deflate function, I used the following function call and output printing:



Which gave the following output:

A screen shot of a computer program

Description automatically generated

These results are all as expected from the earlier verification, so the functions are working correctly.

1. Display all the natural frequencies and eigenvectors for the case of N = 10 masses with all spring constants equal to 1 (i.e. K1 = K2 = ··· = K10, even though the code should handle arbitrary N and spring constants Ki).

A computer screen shot of a program code

Description automatically generated

Using the above code, I calculated all of the frequencies and eigenvectors with a small tolerance. I had set the default tolerance to be 10^-18. Here are all of the natural frequencies and eigenvectors for this case:

The 1th frequency is 0.314755

The 1th eigenvector is [[ 0.12864167 -0.24585298 0.34121918 -0.40626657 0.43521541 -0.42549345

0.37796452 -0.29685177 0.18936243 -0.06504739]]

The 2th frequency is 0.304168

The 2th eigenvector is [[ 0.24585307 -0.40626668 0.42549348 -0.29685176 0.0650474 0.18936236

-0.37796442 0.43521535 -0.34121918 0.12864168]]

The 3th frequency is 0.286787

The 3th eigenvector is [[-3.41219238e-01 4.25493409e-01 -1.89362341e-01 -1.89362440e-01

4.25493438e-01 -3.41219197e-01 -5.46998033e-08 3.41219266e-01

-4.25493427e-01 1.89362384e-01]]

The 4th frequency is 0.263000

The 4th eigenvector is [[ 0.4062666 -0.29685169 -0.18936242 0.43521541 -0.12864166 -0.34121926

0.37796446 0.06504741 -0.42549344 0.24585303]]

The 5th frequency is 0.233338

The 5th eigenvector is [[ 0.43521541 -0.06504736 -0.42549342 0.12864168 0.40626662 -0.18936237

-0.37796449 0.24585302 0.34121925 -0.29685172]]

The 6th frequency is 0.198463

The 6th eigenvector is [[ 4.25493420e-01 1.89362394e-01 -3.41219224e-01 -3.41219241e-01

1.89362375e-01 4.25493428e-01 1.26726927e-08 -4.25493425e-01

-1.89362395e-01 3.41219235e-01]]

The 7th frequency is 0.159155

The 7th eigenvector is [[-3.77964466e-01 -3.77964472e-01 -8.65019633e-09 3.77964466e-01

3.77964478e-01 1.07866189e-08 -3.77964471e-01 -3.77964481e-01

-4.80049714e-09 3.77964477e-01]]

The 8th frequency is 0.116292

The 8th eigenvector is [[ 0.29685172 0.43521541 0.34121923 0.06504738 -0.24585303 -0.42549342

-0.37796448 -0.12864171 0.18936239 0.40626662]]

The 9th frequency is 0.070831

The 9th eigenvector is [[ 1.89362390e-01 3.41219235e-01 4.25493426e-01 4.25493425e-01

3.41219232e-01 1.89362386e-01 -1.76073625e-09 -1.89362389e-01

-3.41219232e-01 -4.25493422e-01]]

The 10th frequency is 0.023787

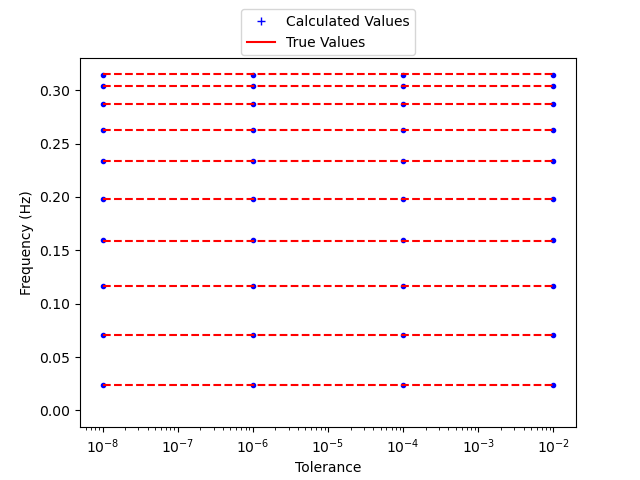
The 10th eigenvector is [[0.06504738 0.1286417 0.18936239 0.24585303 0.29685172 0.34121923

0.37796447 0.40626661 0.42549342 0.43521542]]

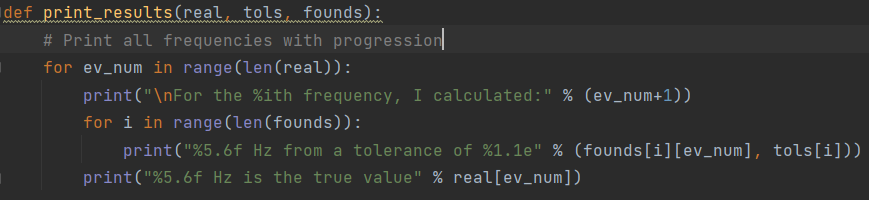
Interpretation:

1. Use a graph to compare the natural frequencies from Question 3 using an independent eigenvalue solver such as numpy.linalg.eig with the natural frequencies you obtained using the power method & deflation for ∆ ∈ {1e−2,1e−4,1e−6,1e−8}, Comment on any trends (Hint: Consider what ∆ represents and also the order in which modes/frequencies are computed).

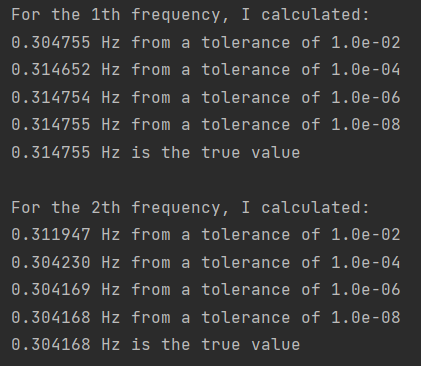
This is the graph that I plotted of the calculated frequencies based on the tolerance used, when compared to the real values of those frequencies. As you can tell, the graph is too zoomed out to show us any precise information on the accuracy of each point, so instead I decided to use print statements to determine whether the frequency values where converging for smaller values of delta (tolerance).



This is the print format that I decided on, which nicely shows how the frequencies are converging towards the correct value for smaller values of tolerance used:



Here are some of the results (from the first and second eigenvalues): (More in appendix)



We can see that it is fairly inaccurate for a tolerance of 10^-2, but becomes more and more accurate for each smaller tolerance, until, by a tolerance of 10^-8, they are virtually indistinguishable from the real values. The entire output is in the appendix, if you want to inspect further results.

The trend is that for each frequency value, using a smaller tolerance (delta) will give more accurate answers as to the true frequency value.

1. Write one paragraph explaining what the eigenvectors and eigenvalues found in Question 3 represent.

The eigenvalues found in question three represent the square of the natural frequencies (in rads-1) of the vibration of the mass-spring system, giving us the speed of the vibration. Meanwhile, the eigenvectors represent the mode shapes of the vibration of the mass-spring system, which give the shape of the vibration. Using these two features together, we can model both the speeds and shapes of the system, and model the whole system.

Appendices:

My Code:

Lab1.py:

import numpy as np  
  
  
def power(Mat, tolerance=10 \*\* -18):  
 # Initialise  
 n = len(Mat)  
 rel\_change = 1  
 # Generate Random Eigenvector  
 x = np.random.rand(n)  
 x = np.reshape(x / np.linalg.norm(x), [n, 1])  
 gamma = 0  
 # Loop until meets tolerance levels  
 while rel\_change > tolerance:  
 # Calc new vector  
 x = np.matmul(Mat, x) / np.linalg.norm(np.matmul(Mat, x))  
 # Calc gamma  
 gamma\_new = np.matmul(x.T, np.matmul(Mat, x))  
 # Calc relative change  
 rel\_change = (gamma\_new - gamma) / gamma\_new  
 gamma = gamma\_new  
  
 return gamma[0, 0], x  
  
  
def power\_w\_deflate(Mat, tolerance=10 \*\* -18):  
 n = len(Mat)  
 eigenpairs = []  
 for i in range(n):  
 eigenpair = power(Mat, tolerance)  
 eigenpairs.append(eigenpair)  
 # Find value of deflation and deflate the matrix  
 deflation = eigenpairs[i][0] \* np.matmul(eigenpairs[i][1], eigenpairs[i][1].T)  
 Mat = Mat - deflation  
  
 return (eigenpairs)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Test matrix  
 Mat1 = np.matrix([[2, -1], [-1, 2]])  
 Mat2 = np.matrix([[2, -1, 0], [-1, 2, -1], [0, -1, 2]])  
  
 # Find all eigenpairs using power method with deflation  
 all\_eigenpairs = power\_w\_deflate(Mat1, tolerance=10 \*\* -18)  
 # Print results  
 for e in range(len(all\_eigenpairs)):  
 prints1 = (e + 1, all\_eigenpairs[e][0])  
 prints2 = ['%5.3f' % all\_eigenpairs[e][1][i, 0] for i in range(len(all\_eigenpairs))]  
 print("For pair %i, eigenvalue is: %5.3f & eigenvector is: " % prints1, prints2)  
  
 print("\n")  
 # Find all eigenpairs using power method with deflation  
 all\_eigenpairs = power\_w\_deflate(Mat2, tolerance=10 \*\* -18)  
 # Print results  
 for e in range(len(all\_eigenpairs)):  
 prints1 = (e + 1, all\_eigenpairs[e][0])  
 prints2 = ['%5.3f' % all\_eigenpairs[e][1][i, 0] for i in range(len(all\_eigenpairs))]  
 print("For pair %i, eigenvalue is: %5.3f & eigenvector is: " % prints1, prints2)

Verification.py:

import Lab1  
import numpy as np  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Test power on 2x2  
 r = Lab1.power(np.matrix([[2, -1], [-1, 2]]))  
 print(r)  
  
 # Test power on 3x3  
 r = Lab1.power(np.matrix([[2, -1, 0], [-1, 2, -1], [0, -1, 2]]))  
 print(r)

Lab1Q3.py:

import numpy as np  
import Lab1  
import matplotlib.pyplot as plt  
  
def construct\_system(N, K):  
 # K should be a list  
 K.append(0) #K sub n+1 = 0  
 A = np.zeros([N,N])  
  
 for i in range(N):  
 for j in range(N):  
 if i==j:  
 A[i,j] = K[i]+K[i+1]  
 elif i==j+1:  
 A[i,j] = -1 \* K[i]  
 elif j==i+1:  
 A[i,j] = -1 \* K[j]  
 return(A)  
  
  
def plot\_results(real, tols, founds):  
  
 # Plot found values  
 for i, tol in enumerate(tols):  
 plt.plot([tol]\*10, founds[1], 'b.')  
 plt.plot(0, 0, 'b+', label="Calculated Values")  
  
 # Plot real values (with horizontal lines)  
 for ele in real:  
 plt.plot([1e-8, 1e-2], [ele]\*2, 'r--')  
 plt.plot(0, 0, 'r-', label="True Values")  
  
 plt.xlabel("Tolerance")  
 plt.ylabel("Frequency (Hz)")  
 plt.legend(bbox\_to\_anchor=(0.5, 1.15), loc='upper center')  
 plt.semilogx()  
 #plt.savefig("Freq\_by\_tolerance")  
 #plt.show()  
  
def print\_results(real, tols, founds):  
 # Print all frequencies with progression  
 for ev\_num in range(len(real)):  
 print("\nFor the %ith frequency, I calculated:" % (ev\_num+1))  
 for i in range(len(founds)):  
 print("%5.6f Hz from a tolerance of %1.1e" % (founds[i][ev\_num], tols[i]))  
 print("%5.6f Hz is the true value" % real[ev\_num])  
  
def solve\_freq\_modes(N,K):  
 A\_mat = construct\_system(N, K)  
 tols = [1e-2, 1e-4, 1e-6, 1e-8]  
 freqs\_from\_tols = []  
  
 for num, tol in enumerate(tols):  
 all\_eigenpairs = Lab1.power\_w\_deflate(A\_mat, tol)  
 found\_eig = []  
 for pair in range(N):  
 found\_eig.append(all\_eigenpairs[pair][0])  
 found\_freq = np.sqrt(found\_eig) / (2\*np.pi)  
 freqs\_from\_tols.append(found\_freq)  
  
 real\_eig = np.sort(np.linalg.eig(A\_mat)[0])[::-1]  
 real\_freq = np.sqrt(real\_eig) / (2\*np.pi)  
 plot\_results(real\_freq, tols, freqs\_from\_tols)  
 print\_results(real\_freq, tols, freqs\_from\_tols)  
  
def solve\_simple(N,K):  
 # Solve for default tolerance  
 A\_mat = construct\_system(N, K)  
 all\_eigenpairs = Lab1.power\_w\_deflate(A\_mat) # Use default tolerance  
 # Print results  
 for i,pair in enumerate(all\_eigenpairs):  
 print("The %ith frequency is %f" % (i+1, np.sqrt(pair[0]) / (2\*np.pi) ))  
 print("The %ith eigenvector is" % (i+1), pair[1].reshape([1, len(pair[1])]), "\n")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Test with spring constants of  
 #solve\_freq\_modes(10, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1])  
 solve\_simple(10, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

Full Frequency Print Outputs:

For the 1th frequency, I calculated:

0.304755 Hz from a tolerance of 1.0e-02

0.314652 Hz from a tolerance of 1.0e-04

0.314754 Hz from a tolerance of 1.0e-06

0.314755 Hz from a tolerance of 1.0e-08

0.314755 Hz is the true value

For the 2th frequency, I calculated:

0.311947 Hz from a tolerance of 1.0e-02

0.304230 Hz from a tolerance of 1.0e-04

0.304169 Hz from a tolerance of 1.0e-06

0.304168 Hz from a tolerance of 1.0e-08

0.304168 Hz is the true value

For the 3th frequency, I calculated:

0.286747 Hz from a tolerance of 1.0e-02

0.286818 Hz from a tolerance of 1.0e-04

0.286788 Hz from a tolerance of 1.0e-06

0.286787 Hz from a tolerance of 1.0e-08

0.286787 Hz is the true value

For the 4th frequency, I calculated:

0.265407 Hz from a tolerance of 1.0e-02

0.263027 Hz from a tolerance of 1.0e-04

0.263000 Hz from a tolerance of 1.0e-06

0.263000 Hz from a tolerance of 1.0e-08

0.263000 Hz is the true value

For the 5th frequency, I calculated:

0.233591 Hz from a tolerance of 1.0e-02

0.233346 Hz from a tolerance of 1.0e-04

0.233338 Hz from a tolerance of 1.0e-06

0.233338 Hz from a tolerance of 1.0e-08

0.233338 Hz is the true value

For the 6th frequency, I calculated:

0.159589 Hz from a tolerance of 1.0e-02

0.198477 Hz from a tolerance of 1.0e-04

0.198463 Hz from a tolerance of 1.0e-06

0.198463 Hz from a tolerance of 1.0e-08

0.198463 Hz is the true value

For the 7th frequency, I calculated:

0.201077 Hz from a tolerance of 1.0e-02

0.159164 Hz from a tolerance of 1.0e-04

0.159155 Hz from a tolerance of 1.0e-06

0.159155 Hz from a tolerance of 1.0e-08

0.159155 Hz is the true value

For the 8th frequency, I calculated:

0.116560 Hz from a tolerance of 1.0e-02

0.116297 Hz from a tolerance of 1.0e-04

0.116292 Hz from a tolerance of 1.0e-06

0.116292 Hz from a tolerance of 1.0e-08

0.116292 Hz is the true value

For the 9th frequency, I calculated:

0.071011 Hz from a tolerance of 1.0e-02

0.070832 Hz from a tolerance of 1.0e-04

0.070831 Hz from a tolerance of 1.0e-06

0.070831 Hz from a tolerance of 1.0e-08

0.070831 Hz is the true value

For the 10th frequency, I calculated:

0.023763 Hz from a tolerance of 1.0e-02

0.023788 Hz from a tolerance of 1.0e-04

0.023787 Hz from a tolerance of 1.0e-06

0.023787 Hz from a tolerance of 1.0e-08

0.023787 Hz is the true value

Process finished with exit code 0