Assignment 5

Name: Mohamed Yusuf

Student Number: 22273364

Exercise 1

The code for this sections can be found in fortran/exercise 1

a

To solve this part i first wrote a module denoted matrix inverter found in matrix_inverter.f90. This module is made up of a calculate_minor_det and a mat_3_inverter function.

The mat_3_inverter function begins by calculating the determinant of the matrix, if that is 0 it prints an error due to the matrix not being invertible.

Other wise it loops through the initialized inverse matrix. I noted then that a minor matrix simply ignores a row and column which is related to the current loop. Utilizing the the calclate_minor_det function i calculated the corresponding minor matrices determinant , multiplied the output by an identifier and set that to the correct position in the inverse matrix.

the calclate_minor_det takes a row identifier , a column identifier and a matrix. It has a minor matrix which is of size 2 and fills in a pair of nested loops. In the nested loops it ignores the row and columns associated with our identifier and uses a succ_count to help fill up our minor matrix. Finally it returns the determinant of the matrix.

To test this out i created a test_matrix.f90 file and inputted the following singular matrix and a invertible matrix respectively

And got the following output

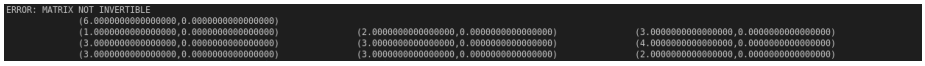


Figure 1: Output of singular matrix followed by invertible matrix

This first throws an error and then prints out the actual inverse of the matrix.

Next i developed the power iteration method found in the smallest_eigen_value.f90 file, this file has two functions the rayleigh_quotient function and the poweriterationmethod function. Initially the program inverses

an input matrix using our module from before. It then utilizes the power iteration function to calculate the dominant eigen value and eigen vector.

The powerIterationMethodSystem function simply starts with a 3 by 1 matrix and then performs then does a while loop. Said loop keeps running until the difference between the previous guess and current guess becomes less then 10^-8. The guesses themselves are updated in the loop using the rayleigh quotient function.

The rayleigh quotien function simply performs the following operation:

$$\lambda = \frac{b^T A b}{b^T b}$$

Where b is our multiplier and A is our matrix.

After running the algorithm we acquire:

Figure 2: Smallest Eigen Value

This was supported through calculating it again in wolfram alpha as well as double checking that it is an eigen value and eigen vector.

b

For this section we simply update our smallest_eigen_value.f90 file to write the current gues and count to a csv file names absolute_error.csv. We then plotted them with the exercise1.py/plot_absolute_error file.

We plotted both a log-log plot and an absolute error plot:

Hence we notice our system convergence is geometric in nature.

Exercise 2

The code for this sections can be found in fortran/exercise 1

a

To complete this question i developed a matrix_generators.f90 module that has 2 functions full_n_matrix and band_structure_generator.

The full_n_matrix function generates the whole N X N matrix through N times. It effectively uses the diagonal as an axis and with reference to that it updates the other diagonals which are parralel to it.

Meanwhile the band_structure_generator utilizes a similiar method btt ignore the bottom lower part of the matrix due to symmetricity. This will allow us to utilize llpack and lblas for quicker calculations of eigenvectors/values.

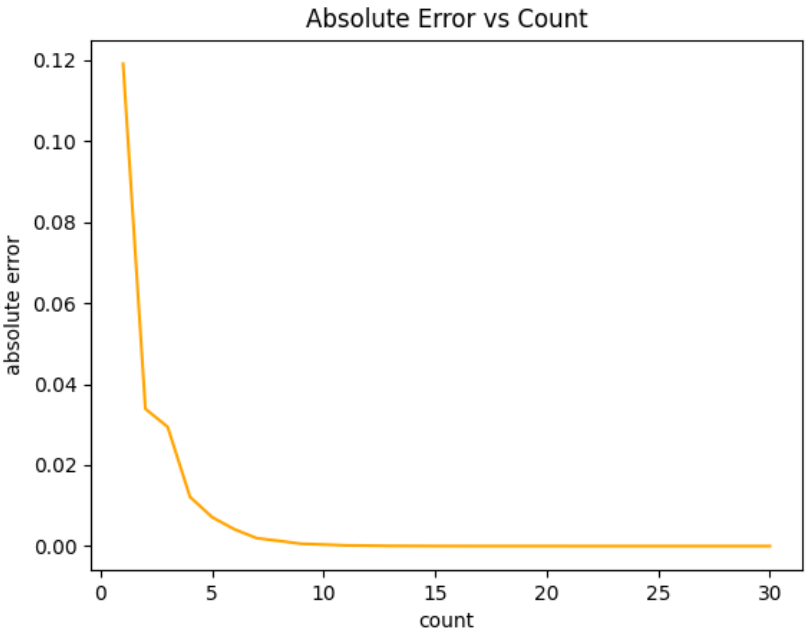


Figure 3: Absolute Error Plot

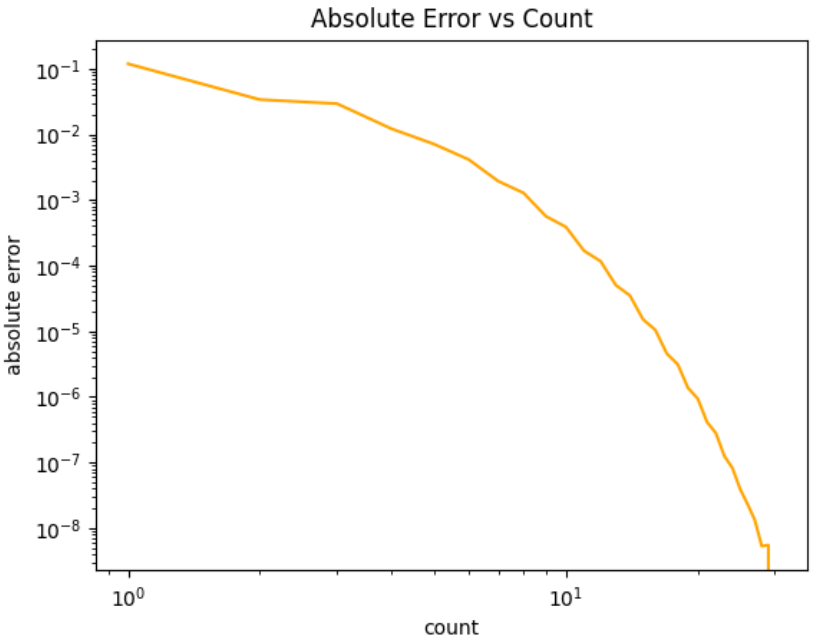


Figure 4: Absolute Error Log Log Plot

To test our system works we developed a test file denoted as test.f90. It outputs both for the test case

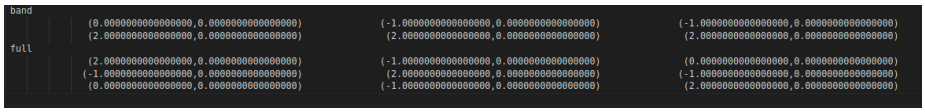


Figure 5: test file output

As we can see this follows our expected pattern.

b

For this portion we set up the lapack_set_up.f90 file , initially you are prompted for an input. This input will be used to allocate the relevant matrices needed for the LAPACK based ZHBEV file.

We then use the ZHBEV package to store it into a text file with eigen_vectors and corresponding eigen values. We get the following output for N = 3.

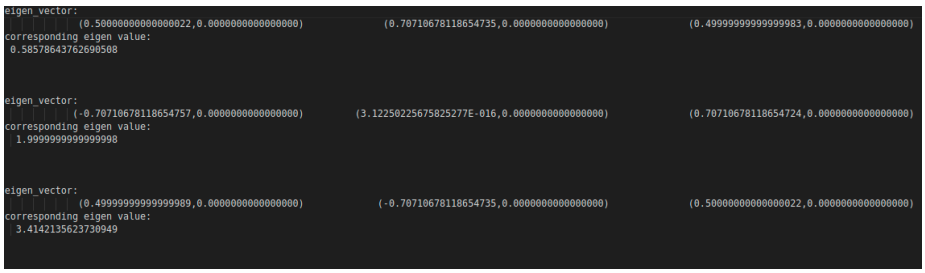


Figure 6: Lapack Output

This was then reaffirmed by manually checking it by hand and wolfram alpha.

c Blas stands for basic linear algebra subprograms, it provides fast and efficient ways of performing basic vector and matrix operations. It is made up of three levels. The first level handles scalar,vector and vector-vector operations. The second level handles matrix-vector operations and the 3rd level handles matrix-matrix operations.

d Due to efficiency of BLAS subprograms it was used when developing LAPACK. LAPACK routines are utilize these BLAS subprograms to speed up its calculations of eigen values and other matrix based numerical problems.

e One way to do this is found in the url (http://www.lahey.com/docs/blaseman.pdf). It begins on page 10 and finished on page 15, to compile it you must type `lf95 --openmp a.f -llapackmt -lblasmt`.

Exercise 3

The code for this sections can be found in fortran/exercise 3 and in the python exercise 3 file.

a

To solve this question we write our code in the hermitian_matrix.f90 file. I initially begin my running code setting all the constants needed for our hermitian matrix. Next i initialize and allocate the hermitian matrix with 49^2. Followed closely by allocating space to our lapack required variables.

I next discriteze my potential into a 2d potential_matrix through discretizing my coordinate space in a grid of 0.2 X 0.2. After that i attempt to set up my hermitian matrix. Due to its banded structure i only develop the upper triangle portion of the matrix and ignore the lower triangle. Finally i store that matrix in our output file.

Next i utilize the ZHEEV package to calculate the eigen_values and eigen_vectors. I store these into an eigen_vectors.csv file and eigen_values.csv file respectively.

b *i* Through python i plotted the percentage error and the actual difference.

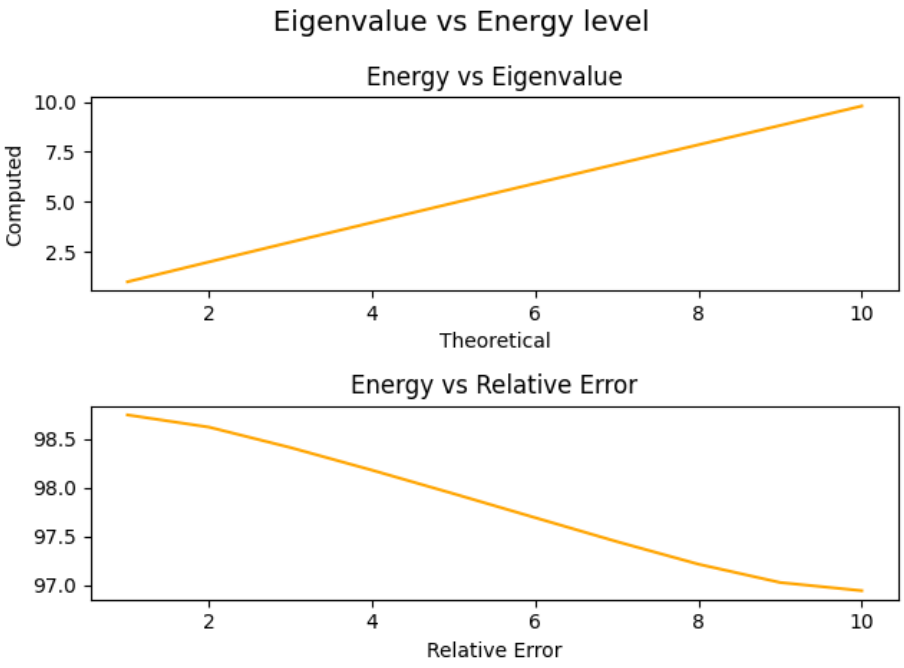
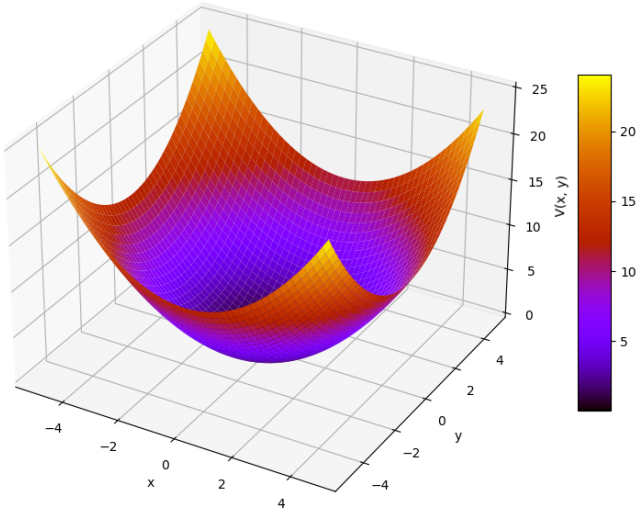


Figure 7: Difference

We notice its effectively the same value. But its accuracy is incorrect for higher energy levels

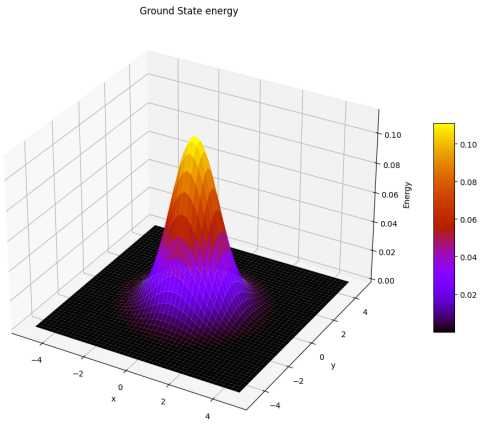
ii After plotting the potential with the function plot_potential in our python file i

Figure 3.03: Potential Funtion of QHO



acquired

iii After storing E0 in our fortran file i plotted it with the plot_EO function acquir-



ing the following

We notice its maximum at the ground state and decreases in a gaussian nature as expected.

iv

After plotting the difference between the actual ground state and theoretical ground state with the plot_E0_vs_theoretical function i acquired the following:

the maximum error we acquired was approximately -1.6 and it fails at the points.

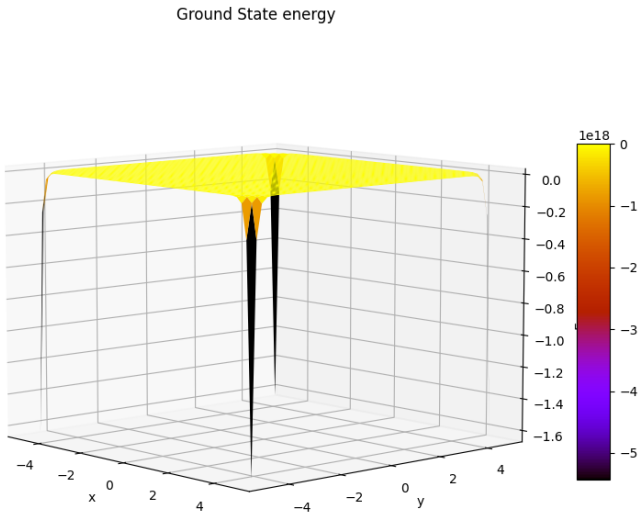


Figure 8: E0 vs Theoretical