Exercise 6
Continuous Time-Independent Schrödinger Equation

Michele Guadagnini - ID 1230663

November 17, 2020

Abstract

The aim of this exercise is to solve the continuous time-independent Schrödinger equation for the *harmonic oscillator* potential in one dimension and to calculate the resulting *eigenvalues* and *eigenfunctions*. The computational method chosen is the Finite Difference Method.

1 Theory

The continuous time-independent Schrödinger equation for the one dimensional harmonic oscillator can be written in this form:

$$H\psi = E\psi \quad , \quad where: \quad H = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{1}{2}m\omega^2x^2 \tag{1}$$

The eigenvalues of this system are defined as: $E_n = \hbar\omega(n + \frac{1}{2})$. The eigenfunctions can be expressed by mean of the *Hermite Polynomials*.

We can simplify the problem by setting $\hbar$ and $2m$ to 1 and the hamiltonian H to:

$$H = -\frac{d^2}{dx^2} + \omega^2x^2 \tag{2}$$

In this way the expected eigenvalues are: $E_n = 2n + 1$.

To solve the problem numerically it is needed to discretize the space in a certain range and also the derivative computation. The resulting equation after discretization is:

$$-\frac{1}{h^2}(\psi_{n+1} + \psi_{n-1} - 2\psi_n) + \omega^2x_n^2\psi_n = E_n\psi_n \tag{3}$$

where h is the unit step in the discretized space. Finally, the eigenpairs can be computed by expressing in matrix form the hamiltonian above and by diagonalizing it.

2 Code Development

2.1 Design and Implementation

The code has been split in to files, one containing the module with all the needed subroutines definitions, *Ex6-Guadagnini-ContinuousTimeIndSE-CODE.f90*, and the other, *Ex6-Guadagnini-CODE.f90*, containing the main program. Also the *Debugger* module has been included in the main program.

The program receives as input 4 parameters: the frequency of the harmonic oscillator, ω; the system dimension in unit of $\frac{1}{\omega}$, *width*; the number of divisions to perform in the system space, $Ndiv$; the number of eigenvalues and eigenvectors to save on file, k.

The module file contains the following subroutines:

- *InitDefaults*: it sets the default values for the above parameters when command line arguments are not present or invalid.
- *Discretize*: it computes the system dimension L, the extremes of the system space in order to have them symmetric with respect to zero, the unit step h and the vector of discretized space points.
- *HarmonicOscillatorPot*: it computes the harmonic oscillator potential as defined in the previous section. Separating the computation of the potential allows to easily change it in future by creating a new subroutine for the new potential.
- *InitHamiltonian*: it creates the hamiltonian matrix as upper triangular. Indeed, the $DSYEV$ subroutine from $LAPACK$ does not require the full matrix because it is supposed to be symmetric, as in our case.
- *SymmetricEigenpairs*: it performs the diagonalization of the matrix using the subroutine *DSYEV*, returning both eigenvalues and eigenvectors.
- *PrintEigVals*: it simply prints in a file two columns with the expected and computed eigenvalues.
- *PrintEigVecs*: it normalizes the eigenvectors dividing them by $\sqrt{h}$ and prints them in a file beside the discretized space points.

Finally the main program has been implemented, that reads the arguments from command line, verifies them to be acceptable and then calls sequentially the subroutines defined above. It also prints some messages to the screen and calls the *debug* subroutines during execution.

2.2 Debug and Test

The program can be compiled and executed with the following commands:

```
gfortran *CODE.f90 -o ContTimeIndSE.x -lblas -llapack
./ContTimeIndSE.x 1 10 1001 20
```

Also some optimization flags has been tested without any problem, but without significant improvements in computation time, since the heavier calculations are done by the $DSYEV$ subroutine, which is already optimized in the code.

In order to obtain meaningful results, care must be used when setting the input parameters. Generally, it can be said that increasing the number of division $Ndiv$ improves the computation accuracy, while the *width* parameter must be not too small to allow the eigenfunctions to go to zero on the edges.

3 Results

After some tests, the command and parameters used to produce the results presented in this section are:

```
./ContTimeIndSE.x 1 25 2001 80
```

Expected	**Computed**	**Expected**	**Computed**	**Expected**	**Computed**
1	0.99999023	55	54.98522064	109	108.9419515
3	2.99995117	57	56.98412644	111	110.93980134
5	4.99987304	59	58.98299314	113	112.93761208
7	6.99975585	61	60.98182075	115	114.93538376
9	8.99959959	63	62.98060926	117	116.93311654
11	10.99940426	65	64.97935868	119	118.93081083
13	12.99916987	67	66.97806899	121	120.92846775
15	14.9988964	69	68.9767402	123	122.9260902
17	16.99858387	71	70.97537231	125	124.92368549
19	18.99823226	73	72.97396532	127	126.92127103
21	20.99784157	75	74.97251922	129	128.91888679
23	22.99741182	77	76.97103402	131	130.91662027
25	24.99694298	79	78.96950971	133	132.91465378
27	26.99643508	81	80.9679463	135	134.91334776
29	28.99588809	83	82.96634377	137	136.91337474
31	30.99530202	85	84.96470213	139	138.91591224
33	32.99467687	87	86.96302139	141	140.922881
35	34.99401265	89	88.96130153	143	142.93717682
37	36.99330934	91	90.95954255	145	144.96280141
39	38.99256694	93	92.95774447	147	147.00478354
41	40.99178546	95	94.95590726	149	149.06883168
43	42.9909649	97	96.95403094	151	151.16077089
45	44.99010524	99	98.9521155	153	153.2859264
47	46.9892065	101	100.95016095	155	155.44864445
49	48.98826867	103	102.94816727	157	157.65206631
51	50.98729175	105	104.94613447	159	159.8981537
53	52.98627574	107	106.94406254		

Table 1: First 80 eigenvalues obtained from the computation.

Table 1 reports the first 80 computed eigenvalues. It can be noticed that the first eigenvalue is very close to the expected value, while the greater eigenvalues tends to have a greater error. The same consideration can be done by looking at Figure 1 that shows the percentage relative error of the computed eigenvalues. The relative error seems to increase linearly until 137, where it starts to decrease rapidly, crossing the zero line for the eigenvalue 147 and worsening itself very fast.

Finally, Figure 2 shows as an example the first four approximated eigenfunctions.

4 Self-evaluation

This program is rated in the following way with respect to the five priorities introduced in class about writing good software:

- *Correctness*: the program seems overall correct and it also performs some checks on variables values, but probably it can be improved in exception handling and with pre- and post- conditions.
- *Numerical Stability*: in the program all the hard-coded real variables has been set

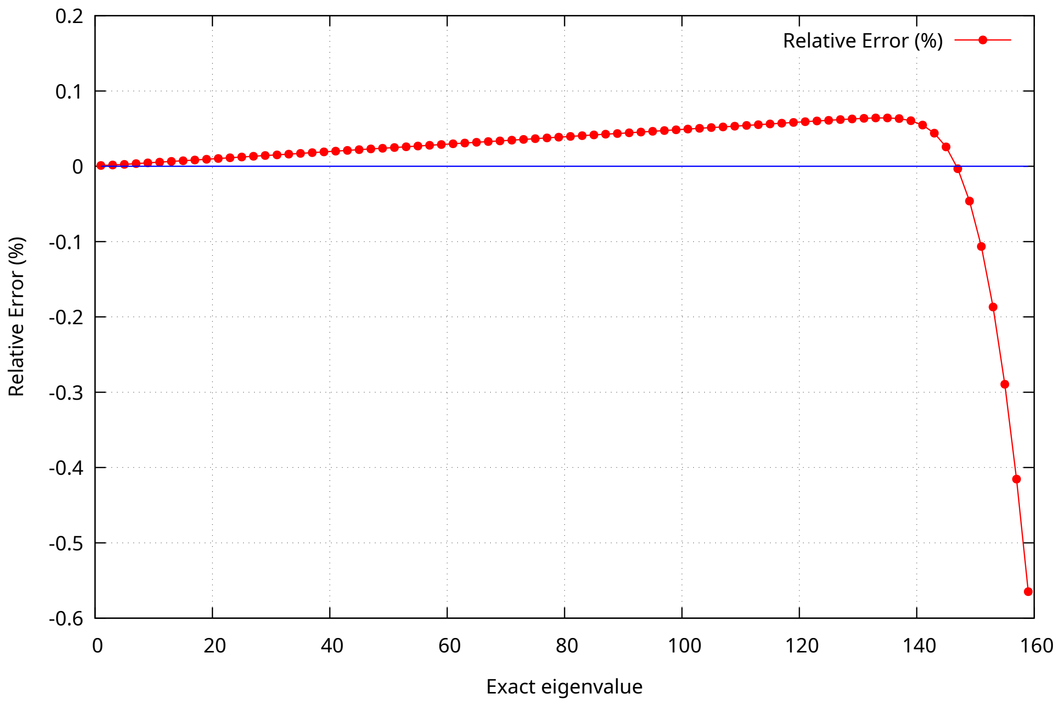

Figure 1: Percentage relative error of the first 80 computed eigenvalues.

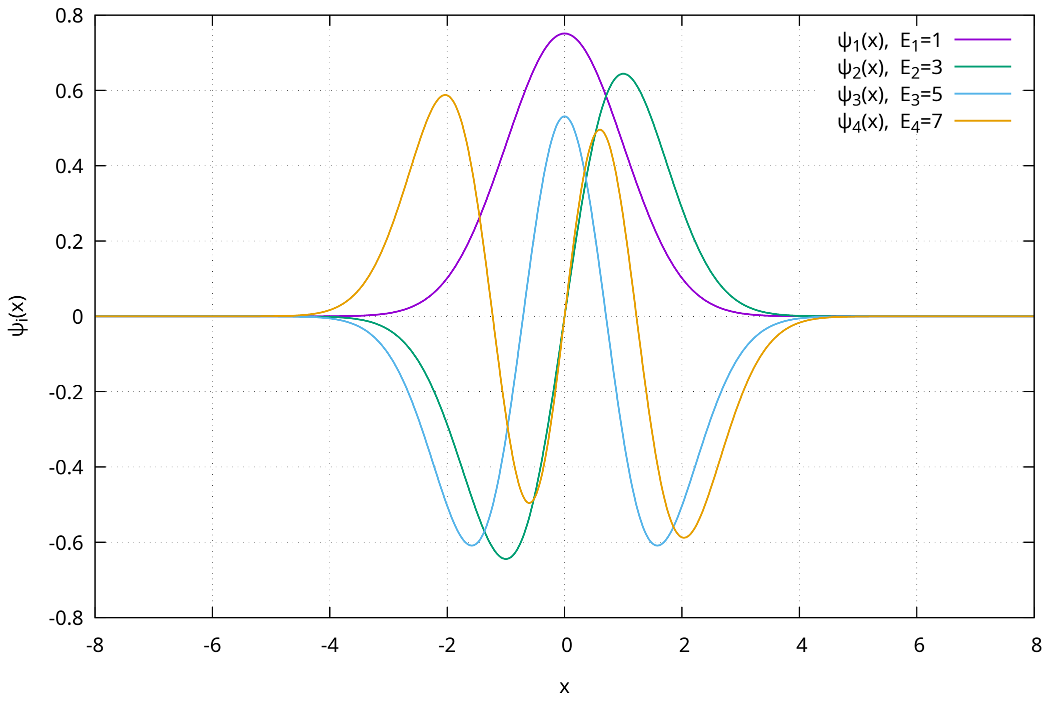

Figure 2: First four approximated eigenfunctions. In the legend is reported also the corresponding exact eigenvalue.

explicitly with double precision (for example by writing $1d0$ instead of 1). Also the conversions between types has been done explicitly with double precision function (for example using *DFLOAT()* instead of *FLOAT()*). However, Figure 1 suggests the presence of some error accumulation.

- *Accurate Discretization*: the space has been discretized symmetrically with respect to zero; this showed better accuracy in the results, probably because the harmonic potential is an even function.
- *Flexibility*: the program has been split in smaller parts that eventually allow to build a new program by reusing the same subroutines or by changing one of these. For example the potential computation has been separated from the hamiltonian matrix building in order to make easier to set a different potential. Also, by using command line arguments it allows to change some parameters without the need to recompile the program. One thing that could have been done is to make the *Discretize* subroutine receive in input also the extremes of the space to be discretized, allowing to use also potentials not centered at zero.
- *Efficiency*: the heaviest calculation are done by the *DSYEV* subroutine to diagonalize the matrix, which is part of the *LAPACK* library. Different optimization flags has been tested without useful improvements. When possible, vectorized operations has been preferred against hand-coded loops.