Time independent Schrödinger equation:

One dimensional harmonic oscillator

Abstract

This exercise asks to solve numerically the eigenvalues problem of the quantum harmonic oscillator. The problem is considered in one dimension and has a, well known, analytical solution. In particular, the task to solve in this assignment is to write a program, in `FORTRAN90`, to find the first k eigenvalues and eigenvectors of this problem. Moreover the code has to take into account the characteristics for a good scientific software development, so *Correctness, Stability, Accurate discretization, Flexibility and Efficiency.*

Theory

The Hamiltonian of the quantum harmonic oscillator is

$$\hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2}m\,\omega^2\hat{x}^2 \tag{1}$$

where $\hat{p}$ is the momentum operator and $\hat{x}$ is the position operator. In the x-space representation, we will refer to, we have that

$$\hat{p} = -i\hbar\frac{\partial}{\partial x} \tag{2}$$

Numerically, to solve this eigenvalues problem, the *Finite Difference Method* is applied. Firstly we discretize our space in N points, so the hamiltonian becomes an NxN matrix. It has two contributions:

- The kinetic part, that contains a second derivative (a Laplacian in greater dimensions).
- The harmonic potential part

The discretized representation of the Laplacian is a tridiagonal matrix, with -2 on the diagonal and 1 at the sides. The potential, that depends only on x, can be represented as a diagonal matrix, in which the diagonal elements are simply the values that the potential so defined assumes in that point of the grid. The task is to diagonalize numerically the Hamiltonian defined above. In practise, the eigenvalues of H are calculated through the `LAPACK`'s `'ZHEEV' SUBROUTINE`. The informations on the latter can be found at the following link [1]. As stated before the one dimensional armonic oscillator problem has a analytical solution, to which the numerical one is compared. Precisely, the problem

$$\hat{H}|\psi\rangle = E|\psi\rangle \qquad with \qquad \hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2}m\,\omega^2\hat{x}^2 \tag{3}$$

has the eigenvectors

$$\psi_n(x) = \frac{1}{\sqrt{2^n n!}}\left(\frac{m\omega}{\pi\hbar}\right)^{1/4} exp\left(-\frac{m\omega x^2}{2\hbar}\right) H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right) \tag{4}$$

and eigenvalues

$$E_n = \hbar\omega\left(n+\frac{1}{2}\right) \tag{5}$$

Where $n = 0, 1, 2..$ and H_n are the Hermite polynomials of order n.

Code development

Three `MODULE`s are written to solve the task:

- `"DEBUGMOD"`, a module to debug a generic code, present also in previous exercises
- `"HERMITE"`, that contains tools to calculate the Hermite polynomials. This `MODULE` has been developed looking at this link [2]
- `"SOLVE_AO"`, that contains subroutines and functions to solve our specific problem

The constant $\hbar$ is fixed to 1.
I highlight that `"SOLVE_AO"` contains, above all a `SUBROUTINE`, shown in listing 1, that builds the matrix for the momentum operator, as described in the **Theory** section. It takes in input N, the size of the grid, and returns m, the matrix that we want.

```fortran
SUBROUTINE SEC_DERIVATIVE(N,m)
  ! WARNING: of course this is the LAPLACIAN
  ! in greater dimensions
  ! this subroutine computes the second derivative
  ! (present in the momentum operator) on a
  ! discretized 1dimensional grid of size N
  ! INPUT: N, the size of the grid
  ! OUTPUT: the matrix m
  DOUBLE COMPLEX, DIMENSION(:,:), ALLOCATABLE :: m  ! the matrix
  that we want
```

[1] http://www.netlib.org/lapack/explore-html/df/d9a/group__complex16_h_eeigen_gaf23fb5b3ae38072ef4890ba43d5cfea2.html

[2] https://sukhbinder.wordpress.com/hermite-polynomials-fortran-module/

```
    INTEGER :: N, ii

    ALLOCATE(m(N,N))  ! the matrix is tridiagonal
    DO ii = 1,N
       m(ii,ii) = -2
       m(ii,ii+1) = 1
       m(ii,ii-1) = 1
    END DO
    RETURN
  END SUBROUTINE SEC_DERIVATIVE
```

Listing 1: The SUBROUTINE 'SEC_DERIVATIVE'

Moreover the "SOLVE_AO" contains another SUBROUTINE, shown in listing 2, that builds the diagonal matrix that represent the discretized harmonic potential.

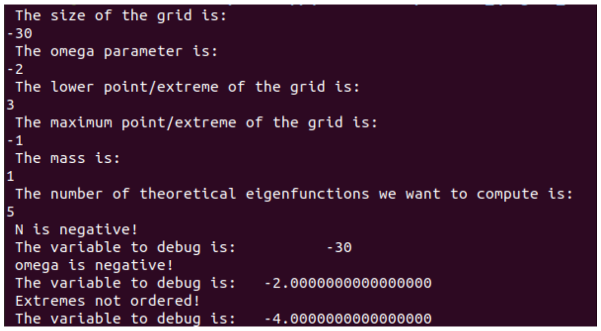
```
  SUBROUTINE POTENTIAL(N,m,OMEGA,x_small,x_high,mass)
    ! this subroutine computes the diagonal matrix
    ! that represent the discretized harmonic potential
    ! on 1dimensional grid of size N
    ! INPUTS:
    ! the size of the grid N
    ! omega
    ! mass
    ! x_small and x_high (extremes of the grid)
    ! OUTPUT: the matrix m
    DOUBLE COMPLEX, DIMENSION(:,:), ALLOCATABLE :: m  ! the matrix
    that we want
    INTEGER :: N, ii
    DOUBLE PRECISION :: OMEGA, x_small, x_high, step, mass

    step=(x_high-x_small)/N

    ALLOCATE(m(N,N))
    DO ii=1,N
       m(ii,ii)=mass/2*(omega*(x_small+step*(ii-1)))**2 ! diagonal
    elements
    END DO
  END SUBROUTINE POTENTIAL
```

Listing 2: The SUBROUTINE 'POTENTIAL'

This MODULE provides also a SUBROUTINE, named "EIGVAL" to calculate the eigenvalues of the hamiltonian through the 'ZHEEV' SUBROUTINE, linked above. The code is divided in two programs, Ex6-Quaglia-CODE.f90 and Ex6-Quaglia-CODEa.f90, that use the aforementioned MODULEs, to compute the eigenvalues and eigenvectors of H, respectively with the *Finite Difference Method* and analytically. Both the programs ask the user to insert the parameters N, *omega*, the extremes of the grid *x_small* and *x_high*, and the mass m. This guarantees a major *flexibility*, since at each run the user can decide the value he prefers. In the end the results are printed in ".dat" files, used by the GNUPLOT's script, 'plot.gnu'.

Results

Firstly, to avoid parameters out of their physical range, the `'DEBUGMOD' MODULE` can be used, setting, in both the programs, the `LOGICAL` flag `'debug_flag'` to true. If so, a warning message like the one in figure 1 is printed.



```
 The size of the grid is:
-30
 The omega parameter is:
-2
 The lower point/extreme of the grid is:
3
 The maximum point/extreme of the grid is:
-1
 The mass is:
1
 The number of theoretical eigenfunctions we want to compute is:
5
 N is negative!
 The variable to debug is:          -30
 omega is negative!
 The variable to debug is:   -2.0000000000000000
 Extremes not ordered!
 The variable to debug is:   -4.0000000000000000
```

Figure 1: Warning message example.

These following results are obtained with the parameters values: $N = 50$, $x\_small = -1$, $x\_high = 1$, $m = 1$ and $\omega = 10$. Figure 2 shows the first 50 eigenvalues computed by the `'ZHEEV' SUBROUTINE`. According to the equation 5 they should have a linear behaviour with n. The plot of the computed eigenvalues is, in fact, fitted with a function of the type

$$f(x) = a(x^b + c) \tag{6}$$

In the table the values that `GNUPLOT` gives with the corresponding errors are reported.

	a	$\sigma(a)$	b	$\sigma(b)$	c	$\sigma(c)$
	24	5	1.06	0.05	-2.9	0.6
Theoretical value	10	-	1	-	0.5	-

Then figure 3 shows the first 4 eigenvectors computed numerically and the comparison between them and the analytical results is shown in figure 4. One can see that the overlap between the numerical solution (coloured lines) and the analytical one (dashed lines) is very good, except for the ψ_3. However one can notice that the analytical solution is the up-side-down version of the numerical one, so it can be a matter of normalization.
Nevertheless, keeping the other parameters fixed, one can notice that, for lower values of ω, like $\omega = 1$ we have a discordance between the theoretical and the numerical eigenvectors. The same happens even, for any ω and range, after some eigenvalue k ($k \sim 20$). This incongruency may rise from the finitness of the interval we fix. For such reason some boundary conditions are needed, and in the method used here this correspond to take a second derivative defined by

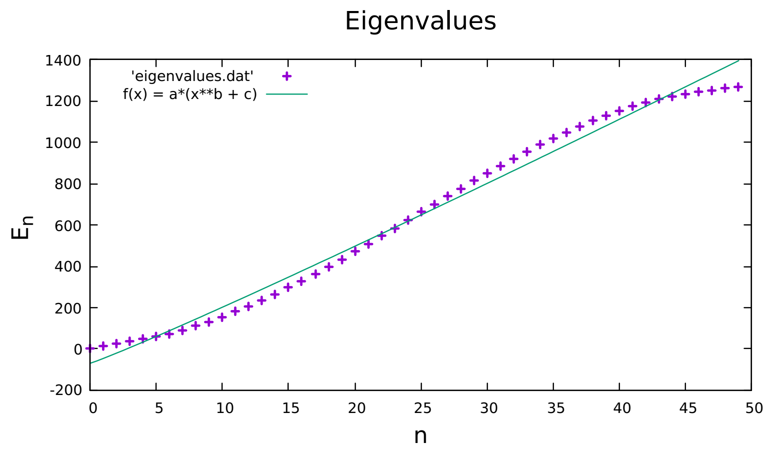


Figure 2: Eigenvalues of the hamiltonian computed numerically. The plot is fitted to verify the behaviour 5.

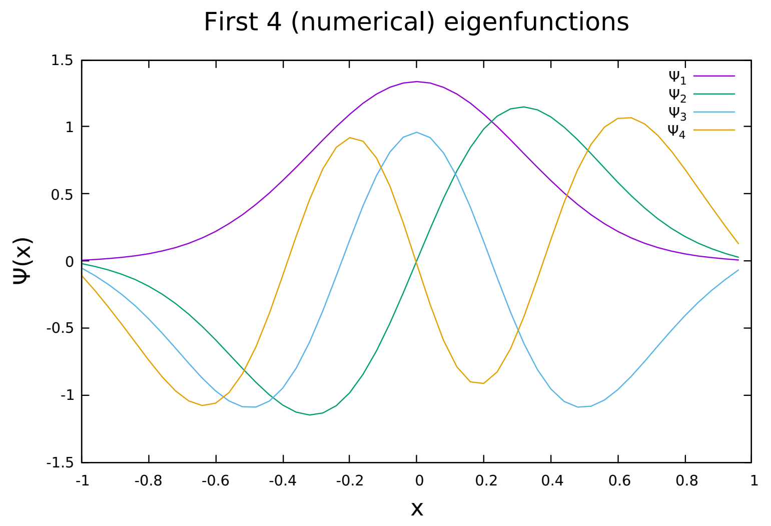


Figure 3: First 4 eigenvectors computed numerically.

a tridiagonal matrix (i.e. the eigenfunctions must be zero at the extremes). I think that the issues reported above could be solved by taking periodic boundary conditions, to reproduce the behaviour of 'infinite' grid.

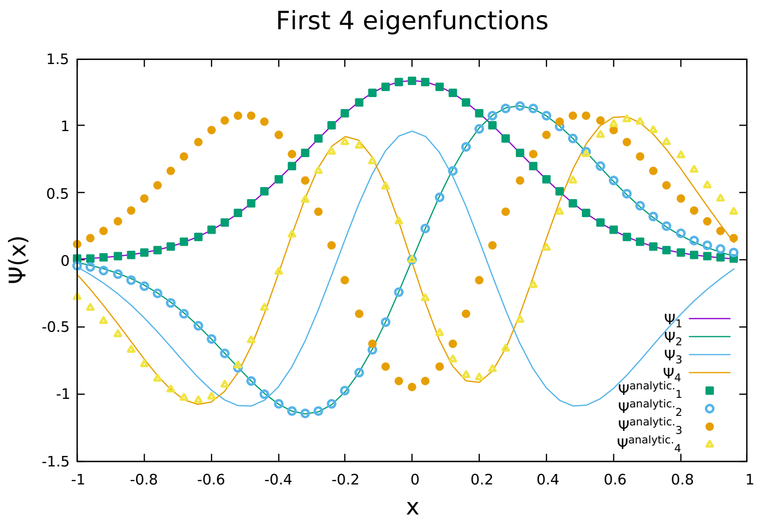


Figure 4: Comparison between analytical and numerical solution, fixed $N = 50$, $x\_small = -1$, $x\_high = 1$, $m = 1$ and $\omega = 10$.

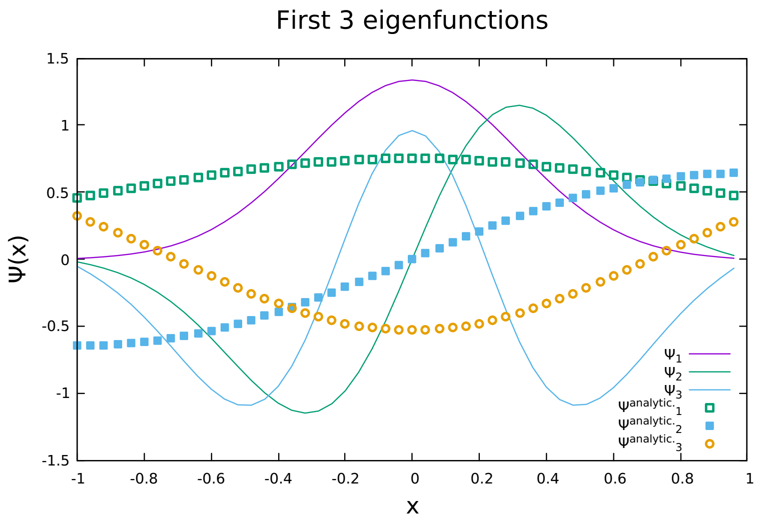


Figure 5: Comparison between analytical and numerical solution, fixed $N = 50$, $x\_small = -1$, $x\_high = 1$, $m = 1$ and $\omega = 1$.

Self-Evaluation

About the *Correctness*, the program does what we expect from the theory, even if there are some issues that can be better explored in the future.
Regarding the *Numerical stability*, it seems to be satisfied, since the program does not give strange or inconsistent results in multiple runs.
The *discretization* can be improved (i.e. increasing the value of N) to reach a more 'smooth' numerical approximation.
Since the program asks the user the values of the parameters, with the check if they are in a physical range, one can state that *Flexibility* is present.
To conclude, the *Efficiency* is taken into account only in changing the optimization flags of the compiler (so `'-o','-o1','-o2','-o3','-ofast'`), but the program is fast in all the cases, without depending in a relevant way from this flag.