Continuos time-indipendent S.E

Umberto Maria Tomasini
Quantum Information Course

20/11/2018

Abstract

In this report we solve the Eigenvalue problem for the quantum oscillator problem in one dimension, with Hamiltonian: $H = \hat{p}^2 + \omega^2 \hat{x}^2$. The method chosen is the Finite Difference Method.

Theory

The Quantum problem

We consider the quantum oscillator problem in one dimension, with Hamiltonian: $H = \hat{p}^2 + \omega^2\hat{x}^2$. It can be shown analitically that the spectrum of eigenvalues is discrete: $E_n = \hbar 2\omega(n + \frac{1}{2})$. The eigenvectors are (in the position rappresentation):

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

where $\{H_n\}$ are the Hermite polynomials. The eigenfunctions with even index are even respect to zero and the odd ones are odd respect to zero.
Later we compare the first three obtained eigenvector to the analytical results:

$$\psi_0(x) = \left(\frac{2m\omega}{\pi\hbar}\right)^{1/4} exp(-\frac{m\omega}{\hbar}x^2)$$

$$\psi_1(x) = \left(\frac{2m\omega}{\pi\hbar}\right)^{1/4} x\sqrt{\frac{4m\omega}{\hbar}} exp(-\frac{m\omega}{\hbar}x^2)$$

$$\psi_2(x) = \left(\frac{m\omega}{2\pi\hbar}\right)^{1/4} \left[\frac{4m\omega}{\hbar}x^2 - 1\right] exp(-\frac{m\omega}{\hbar}x^2)$$

Code Development

The finite difference method

It was decided to use the finite difference method to solve the following eigenproblem (in in the position rappresentation):

$$\left(\hat{p}^2 + \omega^2\hat{x}^2\right)\psi = E_n\psi$$

$$\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + \omega^2 x^2\right)\psi = E\psi$$

We decide to have natural units: $\hbar = 1$, $\omega = 1$ and $m = 1/2$. Discretizing a one-dimensional interval $[x_{min}, x_{max}]$ (which is taken even respect to zero in order to see the parity of the eigenfunction, so

$$x_{min} = x_{max}$$

) with N points, we have the step of the discretization $h = \frac{2x_{max}}{n}$. Then we exploit a discrete version of the above eigenproblem (with ψ_n the value of ψ at the $n - th$ point):

$$\frac{1}{h^2}\left[-\psi_{n+1} - \psi_{n-1} + (2 + h^2(x_{min} + (n-1)\cdot h)^2)\psi_n\right] = E\psi_n$$

Which can be rewritten as:

$$\frac{1}{h^2}\begin{pmatrix} 2 + h^2(x_{min})^2 & -1 & & & \\ -1 & 2 + h^2(x_{min} + h)^2 & -1 & & \\ & & ... & & -1 \\ & & -1 & & 2 + h^2(x_{min} + N\cdot h)^2 \end{pmatrix}\psi = E\psi$$

Solving this eigenproblem we find the eigenvalues and the discretized eigenfunctions of the quantum problem.

Fortran code

The eigenproblem is solved via the LAPACK subroutine *zheev*. The eigenvectors are stored in the imput matrix as columns, and they are orthonormal. The eigenvalues are sorted in ascending order.
It was decided to solve the eigenproblem without the prefactor $\frac{1}{h^2}$ in front of the matrix, dividing later the eigenvalues per that factor. The decision was taken in order to avoid dividing per very little numbers.

Listing 1: Fortran code: Eigenen problem of Harmonic oscillator, solved with Finite Difference method.

```
        C  =========== DOCUMENTATION ===========
C  Definition:
C ===========
C
C*************************************************
C        MODULE MATRICES
C*************************************************
C

C       TYPE dcm

C       Arguments:
C
C       INTEGER nr              number of rows
C       INTEGER nr              number of columns
C       double complex  elem    elements of matrix
C       double precision eval   eigenvalues hermitian matrix(
   real)
C       double complex m_trace  trace
C       double complex m_det    determinat


C         Purpose
C         =======
C
C       \details \b Purpose:
C       \verbatim
C       Can be used to contain double complex matrices and
   their features.
```

```
C       The eigenvalues are double precision.
C       \endverbatim

C
C      SUBROUTINE init_hermmat (aa, aacheck, numrow, numcol,
   debug,xmin, xmax, omega,hbar,m)
C
C       .. Scalar Arguments ..
C       INTEGER          numrow
C       INTEGER          numcol
C       LOGICAL          debug
C       REAL xmin
C       REAL xmax
C       REAL omega
C       REAL hbar
C       REAL m
C
C      .. Array Arguments ..
C       type(dcm)          aa
C       type(dcm)          aacheck
C
C
C       Purpose
C       =======
C
C       \details \b Purpose:
C       \verbatim
C
C       Purpose: solving Schrodinger equation for harmonic
   oscillator
C       in one-dimension: H=p^2+omega^2*x^2.
C       The Finite Difference method is exploited.
C       Step of discretization=hh=(xmax-xmin)/(nn-1), with nn
    the number of points
C       in the interval.
C       diagonal entries_(ii)=
C       (2*(hbar*hbar)/(2*m)+ omega*omega*x^2*hh^2)/(hh^2)
C       with x^2=(xmin+ ii*hh)**2
C       (ii,ii +/- 1) entries = ((hbar*hbar)/(2*m))*(-1)/(hh
   ^2)
C       (others)=0
C       The factor 1/h^2 is omitted in order to avoid
   dividing per zero (small hh).
C       Then is restored.
C       The dimension of matrix should be equal to numrow and
```

```
    numcol,
C       if different, a warning message is printed. This
   debug is done
C       if debug==.true.
C
C       \endverbatim
C
C       Arguments:
C        ==========
C

C       \param[in] aa
C       \verbatim
C          aa is type(dcm)
C               To be initialized
C       \endverbatim
C       \param[in] aacheck
C       \verbatim
C          aacheck is type(dcm)
C               A copy of the initialized matrix
C       \endverbatim
C
C       \param[in] numrow
C       \verbatim
C          numrow is INTEGER
C       \endverbatim
C
C       \param[in] numcol
C       \verbatim
C          numcol is INTEGER
C       \endverbatim
C
C
C       \param[in] debug
C       \verbatim
C          numrow is LOGICAL
C       \endverbatim

C       \param[in] xmin
C       \verbatim
C          xmin is REAL
C       \endverbatim

C       \param[in] xmax
C       \verbatim
```

```
C           xmax is REAL
C       \endverbatim

C       \param[in] omega
C       \verbatim
C           omega is REAL
C       \endverbatim

C       \param[in] hbar
C       \verbatim
C           hbar is REAL
C       \endverbatim
C
C       \param[in] m
C       \verbatim
C           m is REAL
C       \endverbatim
C
C================================================
C
C
C       SUBROUTINE print_matrices (AA, namefile)
C
C       .. Scalar Arguments ..
C       character(:)            namefile
C       ..
C      .. Array Arguments ..
C       type(dcm)           AA
C       ..
C
C       Purpose
C       =======
C
C       \details \b Purpose:
C       \verbatim
C
C       This subroutine prints a type(dcm) variable on a file
   , called
C       "namefile", at unit 40.
C       Number of rows, columns, matrix elements, determinant
   , trace and eigenvalues
C       are printed. The matrix elements are printed in
   proper order.
C       \endverbatim
C
```

```
C       Arguments:
C       ==========
C

C       \param[in] AA
C       \verbatim
C          AA is type(dcm)
C               The variable that will be printed
C       \endverbatim
C
C       \param[in] namefile
C       \verbatim
C               namefile is character(:)
C       \endverbatim
C
C =================================================
C
C
C       SUBROUTINE print_vector (vec, nn, namefile)
C
C       .. Scalar Arguments ..
C       character(:)            namefile
C       integer                 nn
C       ..
C      .. Array Arguments ..
C       double precision         vec(nn)
C       ..
C
C       Purpose
C       =======
C
C       \details \b Purpose:
C       \verbatim
C
C       This subroutine prints a double precision vector on a
    file, called
C       "namefile", at unit 80.
C       \endverbatim
C
C       Arguments:
C       ==========
C

C       \param[in] vec
C       \verbatim
```

```
C          vec is double precision, dimension(:)
C               The vector that will be printed
C       \endverbatim
C       \param[in] nn
C       \verbatim
C          nn is INTEGER
C       \endverbatim
C       \param[in] namefile
C       \verbatim
C               namefile is character(:)
C       \endverbatim
C  ============================================
C
C*********************************************
C        MODULE debugging
C*********************************************
C
C       SUBROUTINE CHECKDIM(nn,nncheck,debug)
C
C       .. Scalar Arguments ..
C       INTEGER*2             nn
C        INTEGER*2             nncheck
C        LOGICAL               debug
C       ..
C       .. Array Arguments ..
C       INTEGER*2           m(nn,nn)
C       INTEGER*2           mcheck(nn,nn)
C       ..
C       Purpose
C        =======
C
C       \details \b Purpose:
C       \verbatim
C
C       CHECKDIM checks if the dimension of the matrix are
   correct. Firstly checks
C       if the the dimension INTEGER*2 is above 10000. If it
   is so, it prints a WARNING
C       message because it takes too much time. Secondly, it
   checks if nn is inferior
C       to 1, printing a WARNING message. Lastly, it checks
   if nn is actually nncheck
C       as it should be. If it is not true, it prints a
   WARNING message, stating which
C       should be the dimension and the actual one. If
```

```
   everything goes well, it prints
C       "okay: right dimensions matrices",
C       and the actual dimension.

C       \endverbatim
C
C       Arguments:
C       ==========
C

C       \param[in] nn
C       \verbatim
C          nn is INTEGER*2
C          The dimension of the squared arrays
C       \endverbatim
C       \param[in] qq
C       \verbatim
C          qq is INTEGER*2
C               the number of the cycle, it should be nn=qq
   *100
C       \endverbatim
C
C       \param[inout] debug
C       \verbatim
C          debug is LOGICAL
C       \endverbatim
C
C=================================================
C
C       SUBROUTINE CHECKMAT(nn,m,mcheck,debug)
C
C       .. Scalar Arguments ..
C       INTEGER*2            nn
C        LOGICAL              debug
C       ..
C       .. Array Arguments ..
C       INTEGER*2          m(nn,nn)
C       INTEGER*2          mcheck(nn,nn)
C       ..
C
C
C         Purpose
C         =======
C
C       \details \b Purpose:
```

```
C       \verbatim
C
C        CHECKMAT checks if the two imput matrices m and
   mcheck are equal. This is done
C        via a do cycle which checks the equality between the
    two matrices element
C        by element. For each entry which is not equal, the
   internal variable INTEGER*4 accum
C        increseas by 1 (starting from 0). If accum in the
   end is bigger than 0,
C        a WARNING message states how much entries between
   the two matrices are
C        different. Otherwise, it is printed "Same matrices".
C       \endverbatim
C
C        Arguments:
C         ==========
C

C        \param[in] nn
C        \verbatim
C          nn is INTEGER*2
C                 The dimension of the squared arrays
C        \endverbatim
C
C        \param[in]
C        \verbatim
C                 m is INTEGER*2 ARRAY (nn,nn)
C        \endverbatim
C
C        \param[in]
C        \verbatim
C                 mcheck is INTEGER*2 ARRAY (nn,nn)
C        \endverbatim
C
C        \param[inout] debug
C        \verbatim
C                 debug is LOGICAL
C        \endverbatim
C
C  ================================================
C************************************************
C        PROGRAM eigenproblem
C************************************************
C================================================
```

```
C
C         Purpose
C         =======
C
C       \details \b Purpose:
C       \verbatim
C
C       Purpose: solving Schrodinger equation for harmonic
   oscillator
C       in one-dimension: H=p^2+omega^2*x^2.
C       The Finite Difference method is exploited.
C       Step of discretization=hh=(xmax-xmin)/(nn-1), with nn
    the number of points
C       in the interval.
C       Given the matrix dimension from file "MatDimension.
   txt", initialize a type(dcm),
C       with inside the matrix of the problem.
C       diagonal entries_(ii)=
C       (2*(hbar*hbar)/(2*m)+ omega*omega*x^2*hh^2)/(hh^2)
C       with x^2=(xmin+ ii*hh)**2
C       (ii,ii +/- 1) entries = ((hbar*hbar)/(2*m))*(-1)/(hh
   ^2)
C       (others)=0
C       The factor 1/h^2 is omitted in order to avoid
   dividing per zero (small hh).
C       Then is restored.
C       Via the lapack routine zheev, the (double precision)
   eigenvalues are stored inside
C       the type, in crescent order. Moreover, eigenvalues
   are placed as columns in the matrix.
C       The first k eigenvalues are printed on file "
   first_k_eigenval.txt".
C       The first k eigenvectors are printed on file "
   first_k_eigenvectors.txt".
C       If debug==.true., debugging is done: checking matrix
   dimension, if matrix is
C       still the same, printing type(dcm) variables on file
   (including eigenvalues
C       and eigenvectors).
C       Morever, via the output iinfo, we check if zheev
   worked.
C
C       \endverbatim
C         Authors:
C         ========
```

```
C
C         \author Univ. of Padua
C
C         \date 13 November 2018

C  ================================================
                module matrices
implicit none
type dcm !defining the type: doublecomplex matrix
integer :: nr !number of rows
integer :: nc !number of columns
!the actual matrix
double complex, dimension (: , :), allocatable :: elem
!eigenvalues
double precision, dimension (:), allocatable :: eval
!trace
double complex :: m_trace
!determinant
double complex :: m_det
end type dcm

contains

subroutine init_mat(aa, aacheck, numrow, numcol,debug, xmin,
$                          xmax, omega, hbar, mm)
!this subroutine initializes a type dcm,
!given as imput. The matrix becomes the one of
!harmonic oscillator problem
!Also the numbers of rows and columns
!are given as imput
!It gives a default value 0 to the determinant, the trace
!and to eigenvalues.

real*8 hh
integer ii,jj
type(dcm) :: aa
type(dcm) :: aacheck
integer numrow, numcol, nn
integer :: my_stat
character (256) :: my_msg
logical :: debug
real*8 :: omega, hbar, mm
real*8 :: xmax, xmin
```

```
aa%nr= numrow
aa%nc= numcol
aacheck%nr= numrow
aacheck%nc= numcol
nn= numrow

if(debug.eqv..TRUE.) then
if(mm <= 0) then
print*, "mass␣<=␣0"
print*, "The␣actual␣value␣is", mm
end if
end if

if(debug.eqv..TRUE.) then
if(hbar <= 0) then
print*, "hbar␣<=␣0"
print*, "The␣actual␣value␣is",hbar
end if
end if

if(debug.eqv..TRUE.) then
if(omega <= 0) then
print*, "omega␣<=␣0"
print*, "The␣actual␣value␣is", omega
end if
end if

!symmetric interval

if(debug.eqv..TRUE.) then
if(xmax <= 0) then
print*, "x_max␣<=␣0"
print*, "The␣actual␣value␣is", xmax
end if
if(xmin >= 0) then
print*, "x_min␣=>␣0"
print*, "The␣actual␣value␣is", xmin
end if
end if


hh= (xmax-xmin)/float(nn-1)
!discretization interval (nn-1 is spacing's number)

!ERROR HANDLING ALLOCATION VECTORS
```

```
allocate (aa%eval(aa%nr),
$                         stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aa%eval with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if
allocate (aacheck%eval(aa%nr),
$                         stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aacheck%eval with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

!ERROR HANDLING ALLOCATION MATRICES
allocate(aa%elem(aa%nr, aa%nc), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aa with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

allocate(aacheck%elem(aa%nr, aa%nc),
$                         stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aacheck with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if


!WARNING IF NOT SQUARED
if(debug .eqv. .true.) then
if(numrow==numcol) then !checking squareness
print*, " "
print*, "INITIALIZATION"
print*, "Okay, squared"
else
print*, "WARNING: NOT SQUARED"
print*, "num rows = ", numrow
print*, "num columns = ", numcol
end if
end if
```

```
!INITIALIZATION
!diagonal entries_(ii)=
!(2*(hbar*hbar)/(2*m)+ omega*omega*x^2*hh^2)/(hh^2)
!with x^2=(xmin+ ii*hh)**2
!(ii,ii +/- 1) entries = (hbar*hbar)/(2*m))*(-1)/(hh^2)
!(others)=0

do ii= 1, aa%nr
do jj= 1, aa%nc
if(ii==jj) then
aa%elem(ii,jj)=cmplx(2*((hbar*hbar)/(2*mm))
$                          +omega*omega*((xmin+ ii*hh)**2)
$                               *hh**4,0)
aacheck%elem(ii,jj)= aa%elem(ii,jj)
else if(jj==(ii+1)) then
aa%elem(ii,jj)=cmplx(-1* ((hbar*hbar)/(2*mm)),0)
aacheck%elem(ii,jj)= aa%elem(ii,jj)
end if

if(jj==(ii-1)) then
aa%elem(ii,jj)=cmplx(-1* ((hbar*hbar)/(2*mm)),0)
aacheck%elem(ii,jj)= aa%elem(ii,jj)
else
aacheck%elem(jj,ii)=(0d0,0d0)
aacheck%elem(jj,ii)= aa%elem(jj,ii)
end if
end do
end do


aa%m_trace=(0d0,0d0)
aa%m_det=(0d0,0d0)
aacheck%m_trace=(0d0,0d0)
aacheck%m_det=(0d0,0d0)

do ii= 1, numrow
aa%eval(ii)=(0d0,0d0)
aacheck%eval(ii)=(0d0,0d0)
end do

end subroutine init_mat


subroutine print_matrices (AA, namefile)
! This subroutine prints a type doublecomplex_matrix:
```

```
!the numbers of row and columns, the elements (in the proper
   order)
!, the trace and the determinant. This is all printed on a
   file

type(dcm) :: AA
character(:), allocatable :: namefile
integer :: ii,jj
integer :: my_stat
character (256) :: my_msg

open(unit = 40, file = namefile, status = "unknown",
$                       iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open␣failed␣with␣stat␣=␣",
$                      my_stat, "␣msg␣=␣"//trim(my_msg)
end if

write (40,*) "NUMBER␣OF␣ROWS:", AA%nr
write (40,*) "NUMBER␣OF␣COLUMNS:", AA%nc
write (40,*) "ELEMENTS:"
do ii=1, AA%nr !do cycle in order to print in the proper
   order
write (40,*) (AA%elem(ii, jj), jj = 1, AA%nc)
end do
write (40,*) "TRACE:", AA%m_trace
write (40,*) "DET:", AA%m_det
write (40,*) "EIGENVALUES:"
do ii=1, AA%nr !do cycle in order to print in the proper
   order
write (40,*) AA%eval(ii)
end do
close(40)

end subroutine print_matrices

subroutine print_vector (vec, nn, namefile)
! This subroutine prints a vector on file,
! given its length nn
character(:), allocatable :: namefile
integer :: ii,nn
integer :: my_stat
character (256) :: my_msg
double precision, dimension(:), allocatable :: vec
```

```
open(unit = 80, file = namefile, status = "unknown",
$                        iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open␣failed␣with␣stat␣=␣"
$                           , my_stat, "␣msg␣=␣"//trim(my_msg)
end if

do ii=1, nn !do cycle in order to print
write (80,*) vec(ii)
end do

end subroutine print_vector

end module matrices

c/////////////////////////////////////////////////

!MODULE DEBUGGING
module debugging

use matrices

interface check
module procedure checkdim,checkmat
end interface

contains
subroutine checkdim(nn,nncheck,debug)!checking dimensions nn
implicit none
integer :: nn, nncheck
logical :: debug

if(debug.eqv..TRUE.) then

if(nn>10000) then !is the dim too large?
print*, "WARNING:␣too␣large␣dimension:"
print*, nn
else if(nn<1) then !is the dim minor than 1?
print*, "WARNING:␣dimension␣minor␣than␣1"
else if((nn-nncheck)>0.5) then !is the dim wrong?
print*, "the␣dimension␣is␣wrong,␣it␣should␣be:"
$                                  , nncheck
print*, "but␣is", nn
```

```
print*, "␣"
else
print*, "okay:␣right␣dimensions␣matrices",nn
end if
end if
end subroutine checkdim

subroutine checkmat(nn,mm,mmcheck,debug)
implicit none
!checking if the input matrixes are equal
type(dcm) :: mm
type(dcm) :: mmcheck
integer :: tt,ss,nn
integer*4 accum
logical debug

if(debug.eqv..TRUE.) then
accum=0
do tt=1,nn
do ss=1,nn
if(abs(mm%elem(tt,ss)-mmcheck%elem(tt,ss))
$                        /abs(mmcheck%elem(tt,ss))> 10E-10)
   then
accum=accum+1
end if
end do
end do

if(accum>0) then
print*, "The␣two␣matrices␣have"
$                        , accum, "different␣entries"
else
print*, "Same␣matrices"
end if

end if
end subroutine checkmat

end module debugging

c/////////////////////////////////////////////

!PROGRAM
program harmosc
```

```
use debugging
use matrices


type(dcm) :: aa
type(dcm) :: aacheck
type(dcm) :: ddcheck
type(dcm) :: dd

integer  ii, jj, nn, nncheck
logical :: debug
character*1 :: choice
integer :: my_stat
character (256) :: my_msg
integer, allocatable :: ipiv(:)
integer :: iinfo

complex*16, allocatable:: work(:)
integer lwork
double precision, allocatable:: rwork(:)
integer iinfodiag
character(:), allocatable :: diagfile, matfile

real*8 :: omega, hh
real*8 :: xmin, xmax
integer :: kk
real*8 :: hbar, mm




choice="y" !Initialization of choice variable
!if it is "X", it asks the debug
!otherwise the programmer can choose between
!doing the debug or not writing "y" or "n"

!Do you want to debug?
if(choice=="X") then
print *, "Do␣you␣want␣to␣debug?"
print *, "y␣for␣yes,␣n␣for␣no"
read*, choice
end if

if(choice=="y") then
debug=.TRUE.
```

```
else if(choice=="n") then
debug=.FALSE.
else
print*, "Not␣understood"
stop
end if

!HERE THE MATRIX DIMENSION IS TAKEN AS IMPUT FROM
!FILE "MatDimension.txt"
!THE MATRIX DIMENSION IS THE NUMBER OF POINTS
!TAKEN INTO ACCOUNT IN THE DISCRETIZATION
!(the ones where psi is computed)


open(unit = 30, file = "MatDimension.txt",
$       status = "unknown",
$ iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, 'Open␣-MatDimension-␣failed␣with␣stat␣=␣'
$                          , my_stat, '␣msg␣=␣'//trim(my_msg)
end if

read(30,*) nn
nncheck=nn


!INITIALIZATION aa and aacheck:
!harm oscillator problem
!omega^2=0.5*usual omega^2 -> usual omega^2= 2*omega^2

mm=0.5
hbar=1
![natural units]

if(debug.eqv..TRUE.) then
if(mm <= 0) then
print*, "mass␣<=␣0"
print*, "The␣actual␣value␣is", mm
end if
end if

if(debug.eqv..TRUE.) then
if(hbar <= 0) then
print*, "hbar␣<=␣0"
```

```
print*, "The actual value is",hbar
end if
end if

xmax= 5.0
xmin= -xmax
!symmetric interval, order 10^0

if(debug.eqv..TRUE.) then
if(xmax <= 0) then
print*, "x_max <= 0"
print*, "The actual value is", xmax
end if
if(xmin >= 0) then
print*, "x_min => 0"
print*, "The actual value is", xmin
end if
end if


hh= (xmax-xmin)/float(nn-1)
!discretization interval (nn-1 is spacing's number)

omega= 1.0

if(debug.eqv..TRUE.) then
if(omega <= 0) then
print*, "omega <= 0"
print*, "The actual value is", omega
end if
end if

!diagonal entries_(ii)=
!(2*(hbar*hbar)/(2*m)+ omega*omega*x^2*hh^2)/(hh^2)
!with x^2=(xmin+ ii*hh)**2
!(ii,ii +/- 1) entries = ((hbar*hbar)/(2*m))*(-1)/(hh^2)
!(others)=0
!the factor 1/h^2 is omitted in order to avoid
!dividing per zero


call init_mat (aa, aacheck, nn, nn,debug, xmax, xmin, omega,
$                                                       hbar
   , mm)
```

```
if(debug.eqv..TRUE.) then!print matrix
matfile="Matrix.txt"
call print_matrices (aa, matfile)
end if

!DIAGONALIZATION


!calling the LAPACK subroutine for eigen values
!necessary to allocate work and rwork
!and initialize lwork
!for information read relative documentation
lwork= 2*nn

allocate(work(lwork),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate work with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

allocate(rwork(lwork),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate rwork with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if


call zheev( "V", "U", nn , aa%elem, nn , aa%eval , work ,
   lwork,
$                        rwork, iinfo1 )
!"V"->eigenvalues & eigenvectors
!"U"-> aa%elem contains the orthonormal eigenvectors of the
   matrix A.
!aa%eval will contain the eigen values of aa
!in ascending order(if INFO==0)

do ii=1, aa%nr !divide per the neglected factor /(hh*hh)
aa%eval(ii)=aa%eval(ii)/(hh*hh)
end do
```

```
do ii=1, aa%nr !divide per the neglected factor /(hh*hh)
do jj=1, aa%nr
aa%elem(ii,jj)=aa%elem(ii,jj)/sqrt(hh)
end do
end do

!DEBUG
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "DIAGONALIZATION"
print*, "␣"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if


if(debug.eqv..TRUE.) then!check diag
if (iinfo1==0) then
print*, "␣"
print*, "Successful␣DIAGONALIZATION"
else if (iinfo1 < 0) then
print*, "␣"
print*, "the", iinfo1, "-th␣argument
$␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣of␣ipiv␣had␣an␣illegal␣value"

else
print*, "␣"
print*,  "the␣algorithm␣failed␣to␣converge"
end if
end if

if(debug.eqv..TRUE.) then!print diag
diagfile="Eigen.txt"
call print_matrices (aa, diagfile)
end if

!PRINTING ON FILE FIRST kk EIGENVECTORS

kk=3

open(unit = 20, file = "first_k_eigenvec.txt",
$               status = "unknown",
$               iostat=my_stat, iomsg=my_msg)
```

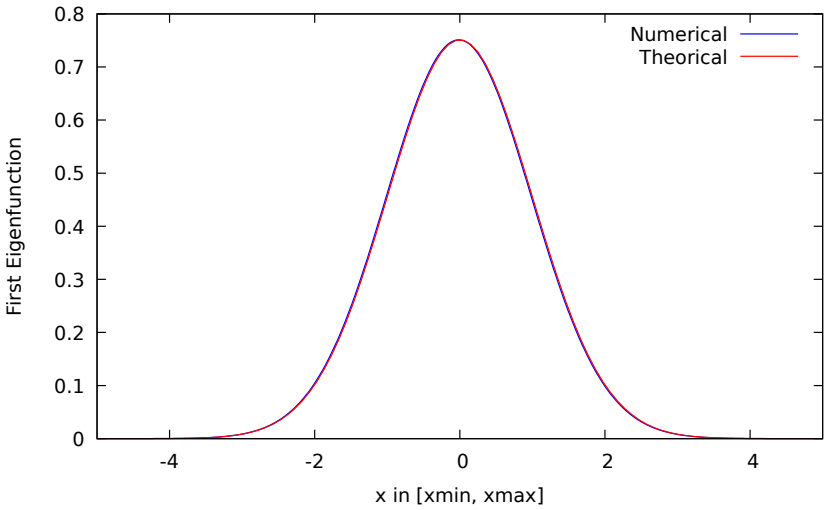
```
if(my_stat /= 0) then
print*, 'Open␣-first_k_eigenvec-␣failed␣with␣stat␣=␣'
$                          , my_stat, '␣msg␣=␣'//trim(my_msg)
end if

do ii=1, aa%nr !do cycle in order to print in the proper
   order
write (20,*) xmin+(ii-1)*hh,
$                         (real(aa%elem(ii, jj)), jj = 1, kk)
end do

!PRINTING ON FILE FIRST kk EIGENVALUES


open(unit = 10, file = "first_k_eigenval.txt",
$                status = "unknown",
$          iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, 'Open␣-first_k_eigenval-␣failed␣with␣stat␣=␣'
$                          , my_stat, '␣msg␣=␣'//trim(my_msg)
end if

do ii=1, kk !do cycle in order to print in the proper order
write (10,*) aa%eval(ii)
end do


stop
end program harmosc
```

Results

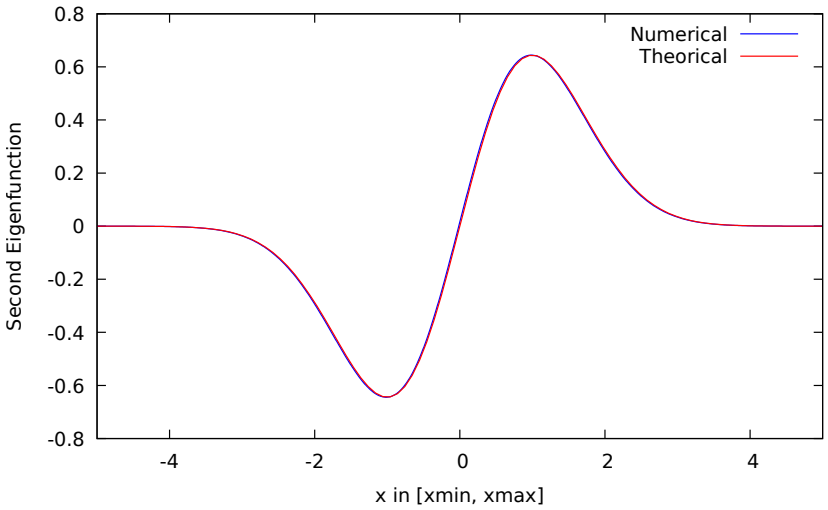
In order to decide which values of x_{min} and x_{max} use, the first computational eigenfunction was plotted several times, and judging by look it was decided to use $x_{min} = -5$ and $x_{max} = 5$ (of the order of 10^0, as expected by the fact we are using natural units). By rehersals and looking at the results, we decided to take 1000 points in that interval. It was chosen to store only the first $k = 3$ eigenvalues and eigenvectors.
The first three eigenvalues are (recalling that $\omega = 1$, $\hbar = 1$ and $m = 1/2$):

	Numerical	Theorical
E_1	0.99993	1
E_2	2.99991	3
E_3	4.99988	5

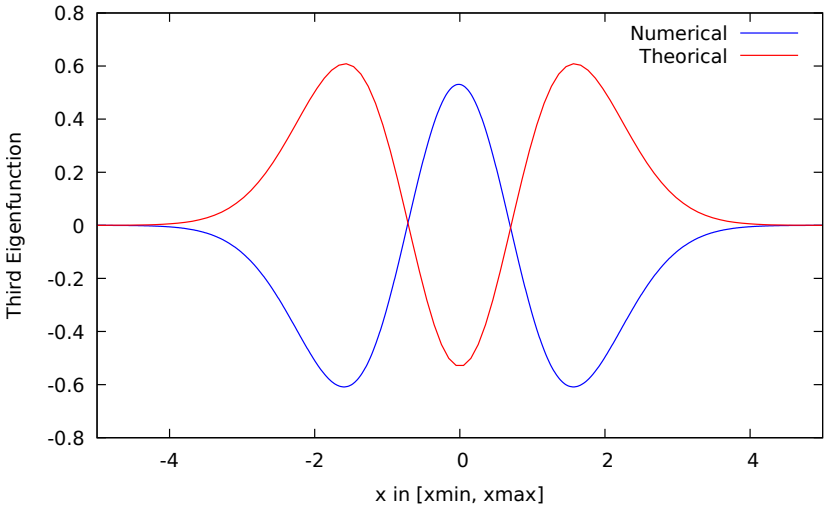
Follows the comparison with the first three theoretical eigenfunctions. Since the eigenvectors given by the LAPACK routine are normalized to 1, in order to compare with the eigenfunction (which is normalized to one over the space), we have multiplied each value of the computational eigenvector per $1/\sqrt{hh}$.



First Eigenfunction

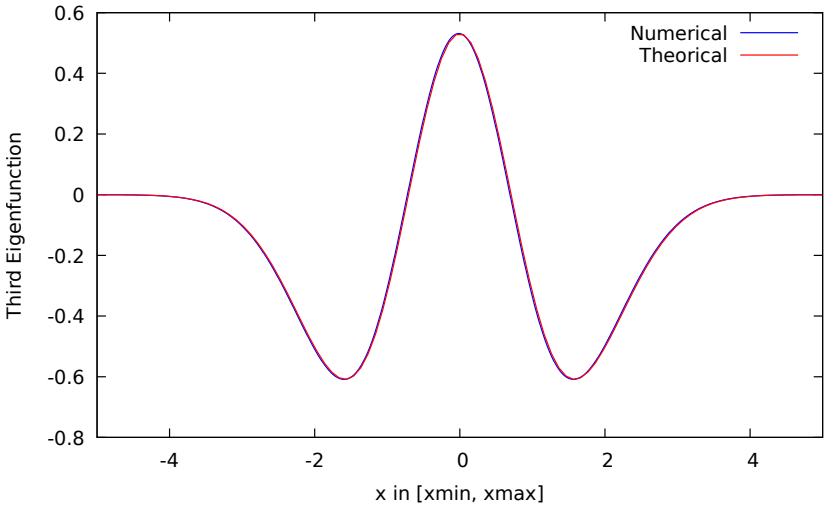


Second Eigenfunction



Third Eigenfunction

We notice that the third theorical eigenfunction and the computational one are the exact opposite of each other. This is reasonable: what is physical is the square module of the ψ. The sign of the eigenfunction is just a choice of convention. To be sure about that, we replot inverting the theorical eigenfunction.



Third Eigenfunction. The theoretical function is inverted

Self-Evaluation

The computational results (both eigenvalues and eigenvectors) have a very good compatibility with the theory. I have learnt how to exploit the Finite Difference method on an actual

problem and to compare the results with the theory.

Rating of the program

- **Correctness.** In the writing of the code, **incremental testing** was done. As an example, during the initialization process (with $debug = .true.$) the initialization of the matrix was checked printing the matrix on a file. After the call of the LAPACK function $zheev$, eigenvalues and eigenvectors were printed. It was done for different dimensions. Checks are implemented and variables were printed during the execution phase. Moreover, a matrix equal to the initial one is always present, in order to check even after the call of the LAPACK function the initialization. One useful addition could be add other variables-printing checkpoints, in order to know if the values are correct. By the way, **preconditions** on the costants and variables are present (checking the sign for ω, $\hbar$, m and for the boundaries of the interval x_{min} and x_{max}, in addition to check if the dimension of the matrix is always the same throughout the program and if the matrix are squared). Checks and comments in the program and in the dedicated subroutine are present. The output are checked via the printing on file of the matrix before and after the call of the function and of the eigenvalues. A good addition would be more **post conditions** on the matrices' entries and on the eigenvalues (as an example checking if the eigenvalues and eigenvectors are actually them). This is partly done via the output variable $info$ of the subroutine $zheev$. Another useful addition would be the use of an **automatic debugger**, in addition to the other automatic checks written by the programmer.

- **Numerical Stability.** In order to avoid dividing the matrix's entries for a very little real number (the square discretization interval hh^2), the LAPACK routine is applied on the initial matrix without the prefactor $\frac{1}{h^2}$ in front of the matrix, dividing later the eigenvalues per that factor. Another critical division could be the division of the eigenvectors per the factor $\sqrt{hh}$ (in order to have the same normalization of the theoretical eigenfunctions).

- **Accurate Discretization.** The chosen discretization (1000 points in an interval with boundaries of the order of 10^0) is sufficiently accurate to obtain successfully the eigenfunctions, as we can see from the figures. In order to decide which values of x_{min} and x_{max} use, the first computational eigenfunction was plotted several times, and judging by look it was decided to use $x_{min} = -5$ and $x_{max} = 5$ (of the order of 10^0, as expected by the fact we are using natural units). By rehersals and looking at the results, we decided to take 1000 points in that interval. An improvement could be re-iterate this process for different number of points within the interval (which stays always the same). Usually, in a code there is the oppurtuinity to choose between different execution methods, and the criteria are its speed and its accuracy. In this case there is no need to choose between several methods: only one is implemented (the LAPACK subroutine). An improvement could be to do the same operation via different method (like different LAPACK subroutines or other methods) and the check the results. After that, there would be the opportunity to choose between different methods.

- **Flexibility.** Speaking of the problem of the harmonic oscillator, the code is quite flexible. In fact the costants (ω, $\hbar$, m) could be tuned as pleasure and also the boundaries of the interval x_{min} and x_{max} and the number of points within this interval. According to these values, the step hh of the discretization is computed, then the rest. On the other hand, if a different Quantum problem should be implemented, it would be necessary to vary the number and the name of the imput parameters dedicated to the initialization (nevertheless, $\hbar$ and m could be useful anyway). The program works with both even and odd dimension's matrix.
- **Efficiency.**