Time-dipendent Schrödinger Equation

Umberto Maria Tomasini
Quantum Information Course

27/11/2018

Abstract

Given the following quantum oscillator problem in one dimension, time dependent: $H = \frac{\hat{p}^2}{2} + \frac{1}{2}(\hat{q} - q_0(t))^2$, with $q_0(t) = t/T$, we consider the evolution in time of $|\psi_0\rangle = |n = 0\rangle$, for $t \in [0, T]$. The chosen method is the Split Operator Method.

Theory

The Quantum problem

We consider the following quantum oscillator problem in one dimension, time dependent (we consider natural units: $\hbar = 1$, $m = 1$ and $\omega = 1/\sqrt{2}$, referring to last week's exercise):

$$\hat{H}(t) = \frac{\hat{p}^2}{2} + \frac{1}{2}(\hat{q} - q_0(t))^2$$

$$q_0(t) = t/T$$

. The relative time dependent Schrödinger Equation is:

$$i\frac{\partial}{\partial t}|\psi(t)\rangle = \hat{H}(t)|\psi(t)\rangle$$

In general $|\psi(t)\rangle = \hat{U}|\psi(0)\rangle$, where $\hat{U}$ is the time evolution operator. We want to consider the evolution in time of $|\psi_0\rangle = |n = 0\rangle$, for $t \in [0, T]$.
In the end, for $t = T$, we expect to obtain the solution $\psi(T) = \psi_0(x - 1)$ (in position rapresentation), since the result of time evolution is only that the potential is translated of one unit in advance.

The Split Operator method

If $\hat{H}(t)$ had been time-indipendent, like in the last week's exercise, we would have written $|\psi(t)\rangle = e^{-iE_0 t}|\psi_0\rangle$, but that is not the case.
Nevertheless, if we consider Δt sufficently small, we can write $\psi(t + \Delta t) \approx e^{-i\hat{H}(t)\Delta t}\psi(t)$, where $\hat{H}(t) = \frac{\hat{p}^2}{2} + V(\hat{q}, t)$, including the above case. Hence we consider the hamiltonian costant in such Δt. The kinetic term and the potential term in general do not commute, but thanks to the smallness of the time interval is possible to show that the overall error in commuting them is of the $O(\Delta t)^3$, hence very small:

$$e^{-i\hat{H}(t)} = e^{-\frac{i}{2}\hat{V}(t)\Delta t}e^{-\frac{i}{2}\hat{p}^2(t)\Delta t}e^{-\frac{i}{2}\hat{V}(t)\Delta t} + O(\Delta t)^3$$

It is convenient to perform our computations of $\psi(t + \Delta t)$ in different rappresentations: the positions one for the potential operator, the momentum one for the kinetic operator. The Fourier Transform permits such a feat:

$$\psi(t + \Delta t) \approx e^{-\frac{i}{2}V(t)\Delta t}\mathcal{F}^{-1}e^{-\frac{i}{2}p^2(t)\Delta t}\mathcal{F}e^{-\frac{i}{2}V(t)\Delta t}\psi(t)$$

The space grid, for N points, is such that $x_j = x_{min} + (j - 1) \cdot \Delta x$, with $j \in \{1, ..., N\}$. Hence the momentum grid is such that $p_k = 2\pi\frac{k}{N\cdot\Delta x}$, with $k \in \{1, ..., N\}$, as reported in [1]. Diving our time interval in a sufficently dense grid, we should obtain $\psi(T)$ without much error, and it should be ψ_0 translated in $x = +1$.

Code Development

It was chosen to use the code of the previous week, where we compute the eigenvector and eigenvalues at $t = 0$. We use only the first eigen vector. The code written this week is appended at the end of the previous' week code, hence we report only the declaration of added variables and the added code, and only the related documentation.
The key is the computation of the Fourier Transform. We decided to follow two different approaches.

Fourier Transform

The added subroutines are two, one for the computation of the potential given the space position and the time istant, and the other one is the one dedicated to the Fourier Transform.

Listing 1: Fortran code: Fourier transform and Potential computation in a given time and space.

```
C================================================
C
C       FUNCTION pot(xmin,hh,ii,ti,hhtt,jj,tf)
C
C       .. Scalar Arguments ..
C       real*8              xmin
C       real*8              hh
C       integer             ii
C       real*8              ti
C       real*8              hhtt
C       integer             jj
C       real*8              tf
C       ..
C
C
C         Purpose
C         =======
C
C       \details \b Purpose:
C       \verbatim
C
C       Giving in input lower boundaries of space
   interval
C       and time interval, their steps of
   discretization
C       and the indexes of the considered point,
C       the potential in that point ai that time is
   given as output.
C       The potential used is:
C       0.5* ((xmin+ii*hh)-(ti+jj*hhtt)/(tf-ti))**2
```

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

        C        \param[in] xmin
        C        \verbatim
        C          xmin is real*8
        C                 Lower boundary space interval
        C        \endverbatim
        C
        C        \param[in] hh
        C        \verbatim
        C          hh is real*8
        C                 discretization step space grid
        C        \endverbatim
        C
        C        \param[in] ii
        C        \verbatim
        C          ii is integer
        C                 the ii point is computed (counting
           xmin as 0)
        C        \endverbatimm
        C
        C        \param[in] ti
        C        \verbatim
        C          ti is real*8
        C                 Lower boundary time interval
        C        \endverbatim
        C
        C        \param[in] hhtt
        C        \verbatim
        C          hhtt is real*8
        C                 discretization step time grid
        C        \endverbatim
        C
        C        \param[in] jj
        C        \verbatim
        C          jj is integer
        C                 the jj point is computed (counting
           ti as 0)
        C        \endverbatimm
        C
        C        \param[in] tf
```

```
C        \verbatim
C          tf is real*8
C                 Upper boundary time interval
C        \endverbatim
C
C   ==================================================C
    ==================================================
C
C       SUBROUTINE tfourier(nn, vec,signn, nnkk,
    tfvec,xmin,xmax,hh)
C
C       .. Scalar Arguments ..
C       INTEGER      nn
C       integer      signn
C       INTEGER      nnkk
C       real*8       xmin
C       real*8       xmax
C       real*8       hh
C       ..
C       .. Array Arguments ..
C       double complex          vec(nn)
C       double complex          tfvec(nn)
C       ..
C
C
C         Purpose
C         =======
C
C       \details \b Purpose:
C       \verbatim
C
C       This subroutine performs a fourier transform
    of vec if sign=-1,
C       an anti fourier transform if sign=1. The
    basis of Fourier space is
C       cexp(cmplx(0., signn*twopi*x/(xmax-xmin)*
    float(ii)))
C       where ii is integer and goes from -nnkk/2, (
    nnkk-2)/2
C       \endverbatim
C
C        Arguments:
C         ==========
C
```

```
C        \param[in] nn
C        \verbatim
C          nn is INTEGER
C                 The dimension of the vector to be
   transformed
C        \endverbatim
C
C        \param[in] signn
C        \verbatim
C          signnn is INTEGER
C                 -1 is forward FT, +1 is backwards
   FT
C        \endverbatim
C
C        \param[in] nnkk
C        \verbatim
C          nnkk is INTEGER
C                 The number of harmonics used
C        \endverbatim
C
C        \param[in] xmin
C        \verbatim
C          xmin is real*8
C                 Lower boundary space interval
C        \endverbatim
C
C        \param[in] xmax
C        \verbatim
C          xmin is real*8
C                 Upper boundary space interval
C        \endverbatim
C
C        \param[in] hh
C        \verbatim
C          hh is real*8
C                 discretization step space grid
C        \endverbatim
C
C        \param[in] vec
C        \verbatim
C         vec is double complex
C                 the vector which will be
   transformed
C        \endverbatim
C
```

```
C         \param[in] tfvec
C         \verbatim
C          tfvec is double complex
C                   the vector which is transformed
C         \endverbatim
C
C   ==================================================


function pot(xmin,hh,ii,ti,hhtt,jj,tf) result(vv)
!giving in imput lower boundaries of space interval;
!and time interval, their steps of discretization
!and the indexes of the considered point
!the potential in that point ai that time is given
integer :: ii,jj
real*8 :: xmin, ti, hh, hhtt, tf
real*8 :: vv

vv= 0.5* ((xmin+ii*hh)-(ti+jj*hhtt)/(tf-ti))**2

end function pot


subroutine tfourier(nn, vec,signn, nnkk, tfvec,xmin,
   xmax,hh)
!this subroutine performs a fourier transform
!of vec if sign=-1, an anti fourier transform
!if sign=1

double complex, allocatable:: vec(:)
double complex, allocatable:: tfvec(:)
integer :: signn
integer :: nn, nnkk
integer :: ii, jj
double precision :: twopi
double precision :: xmin,xmax, hh

twopi = 2*acos(-1.0)
do ii=-nnkk/2, nnkk/2
tfvec(ii+nnkk/2+1)=(0.,0.)
do jj=1, nn
tfvec(ii+nnkk/2+1)=tfvec(ii+nnkk/2+1) + vec(jj)*cexp(
   cmplx(0.,
$        signn*twopi*((xmin+float(jj-1)*hh)/(xmax-xmin
   ))*float(ii)))
```

```
        end do
        tfvec(ii+nnkk/2+1)=tfvec(ii+nnkk/2+1)
        $                /sqrt(float(nn))
        end do
        end subroutine tfourier
```

Listing 2: Fortran code: Eigenproblem of Harmonic oscillator, solved with Finite Difference method. Fourier transform.

```
integer :: nntt, nnttcheck
real*8:: ti, tf, hhtt

real*8 :: vv
double complex, dimension (:,:), allocatable :: psi
double complex, allocatable:: tfpsi(:)
double complex, allocatable:: psitemp(:)
integer :: signn
double precision :: p, two pi, xxx, nnn, sig
integer :: cc, bb
double complex :: temp

integer :: ist

twopi=2*acos(-1.0)


!TIME EVOLUTION of the first eigenvector,
! ACCORDING TO THE HAMILTONIAN:
!H(t)=0.5*p^2 + 0.5*(q-q(t))^2
!q(t)=t/(tf-ti)
!hence m=1, hbar=1 and omega=1/sqrt(2)

!FAST FOURIER TRANSFORM METHOD

!time interval[ti,tf]
ti=0.
tf=5

!HERE THE NUMBER OF ISTANTS IS TAKEN AS IMPUT FROM
!FILE "NumberIstants.txt"
!DISCRETIZATION OF TIME

open(unit = 50, file = "NumberIstants.txt",
$       status = "unknown",
$  iostat=my_stat, iomsg=my_msg)
```

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

read(50,*) nntt
nnttcheck=nntt

hhtt=(tf-ti)/float(nntt-1)
!discretization interval (nn-1 is spacing`s number)

!initialize a double complex matrix
!with each column equal to starting psi
!number of rows=nn
!number of col=nntt
allocate(psi(nn,nntt),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣psi␣with␣stat␣=␣'
$                , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if

do cc=1, nntt
do bb=1, nn
psi(bb,cc)=aa%elem(bb,1)
end do
end do




!initialize two double complex vectors
!useful for fourier
!length=# of spatial points
allocate(tfpsi(nn),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣psi␣with␣stat␣=␣'
$                , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if


!initialize a double complex vector useful for fourier
```

```
!length=# of temporal points
allocate(psitemp(nn),
$                         stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣psi␣with␣stat␣=␣'
$                  , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if

!cycle over the istants
do jj=1, nntt-1
!multiplication per factor exp(-0.5*img*V_ll*hhtt)
!with V_ll= V(xmin+hh*(ll-1))
do ll=1, nn
vv=pot(xmin,hh,ll-1,ti,hhtt,jj, tf)
psi(ll,jj+1)=psi(ll,jj)*cexp(cmplx(0., -0.5*vv*hhtt))
temp = psi(ll,jj+1)
psitemp(ll)= temp
end do

signn=-1!fourier transfom

call tfourier(nn, psitemp ,signn,
$                         nn, tfpsi, xmin, xmax, hh)

!multiplication per factor exp(-img*0.5*p*p*hhtt)
!p goes from -nn/2 to nn/2
do ii=-nn/2, -1
p=ii*twopi/(xmax-xmin)
tfpsi(ii+nn/2+1)=tfpsi((ii+nn/2+1))*cexp(cmplx(0.,
$                                 -0.5*p*p*hhtt))
end do

do ii= 1, nn/2
p=ii*twopi/(xmax-xmin)
tfpsi(ii+nn/2+1)=tfpsi((ii+nn/2+1))*cexp(cmplx(0.,
$                                 -0.5*p*p*hhtt))
end do

signn= 1!fourier anti transfom
call tfourier(nn, tfpsi ,signn,
$                         nn, psitemp, xmin, xmax, hh)

!multiplication per factor exp(-0.5*img*V_ll*hhtt)
!with V_ll= V(xmin+hh*(ll-1))
```

```
do ll=1, nn
vv=pot(xmin,hh,ll-1,ti,hhtt,jj-1, tf)
psitemp(ll)=psitemp(ll)*cexp(cmplx(0, -0.5*vv*hhtt))
temp=psitemp(ll)
psi(ll,jj+1)=temp
end do

end do

!PRINTING psi(t+hhtt*(jj-1)) on file
!jj goes from 1 to nntt
!nntt psi
!for each istant a different file is created

open(unit = 20, file = "psi_time",
$                        status = "unknown",
$                iostat=my_stat, iomsg=my_msg)

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

do ii=1, nn !do cycle in order to print in the proper order
write (20,*) xmin+(ii-1)*hh,
$                                (abs(psi(ii, ist)), ist = 1,
   nntt)
end do
close(20)


open(unit = 70, file = "averages",
$                        status = "unknown",
$                iostat=my_stat, iomsg=my_msg)

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

!COMPUTATION OF AVERAGE, MEAN SQUARE DISTANCE
!FROM AVERAGE AND NORM
!For each istant
!Printed on a file, called "averages"
```

```
do ist=1, nntt
xxx=0.0
nnn=0.0
sig=0.0
do ii=1,nn
!average
xxx=xxx+ (xmin+(ii-1)*hh)*hh*
$                                        abs(psi(ii, ist))**2
!norm
nnn=nnn+ hh*abs(psi(ii, ist))**2
end do
xxx=xxx/nnn
!mean square distance from average
do ii=1,nn
sig= sig + ((xmin+(ii-1)*hh-xxx)**2)*hh*
$                                        abs(psi(ii, ist))**2
end do
sig=sig/nnn
write(70,*) ti+(ist-1)*hhtt, xxx , nnn , sqrt(sig)

end do
print*, "f"
```

Fast Fourier Transform

We implemented the pack called $FFTPACK$. We report only the piece of code where we define and use the dedicated subroutines. The rest is equal to above.

Listing 3: Fortran code: Eigenproblem of Harmonic oscillator, solved with Finite Difference method. Fast Fourier transform (FFTPACK).

```
!initialization of a vector wwsave
!via the subroutine zffti which performs
!prime factorization of nntt together with
!a tabulation of the trigonometric functions,
!which are computed and stored in wwsave

lenwwsave= 2*nn + 16
!at least  4*nn + 15

allocate(wwsave(lenwwsave),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣wwsave␣with␣stat␣=␣'
$                 , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
```

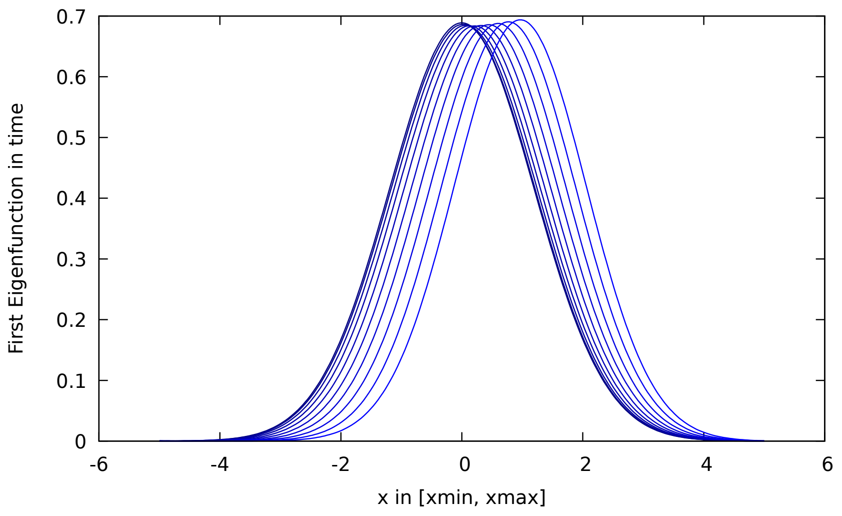
```
end if

call zffti(nn,wwsave)



!cycle over the istants
do jj=1, nntt-1
!multiplication per factor exp(-0.5*img*V_ll*hhtt)
!with V_ll= V(xmin+hh*(ll-1))
do ll=1, nn
vv=pot(xmin,hh,ll-1,ti,hhtt,jj-1, tf)
psi(ll,jj+1)=psi(ll,jj)*exp(cmplx(0., -0.5*vv*hhtt))
temp = psi(ll,jj+1)
psitemp(ll)= temp
end do

!subroutine which performs direct fourier transform
call zfftf(nn,psitemp,wwsave)

!multiplication per factor exp(-img*0.5*p*p*hhtt)
!p goes from -nn/2 to nn/2
do ii= 1, nn/2
p=ii*twopi/(xmax-xmin)
psitemp(ii)=psitemp(ii)*exp(cmplx(0.,
$                                -0.25*p*p*hhtt))
end do

do ii= (nn/2+1), nn
p=(ii-nn)*twopi/(xmax-xmin)
psitemp(ii)=psitemp(ii)*exp(cmplx(0.,
$                                -0.25*p*p*hhtt))
end do

!subroutine which performs inverse fourier transform
call zfftb(nn,psitemp,wwsave)

!multiplication per factor exp(-0.5*img*V_ll*hhtt)
!with V_ll= V(xmin+hh*(ll-1))
do ll=1, nn
vv=pot(xmin,hh,ll-1,ti,hhtt,jj-1, tf)
psitemp(ll)=psitemp(ll)*cexp(cmplx(0, -0.5*vv*hhtt))
end do

do ll=1, nn
```

```
psi(ll,jj+1)=psitemp(ll)/float(nn)
end do
end do
```

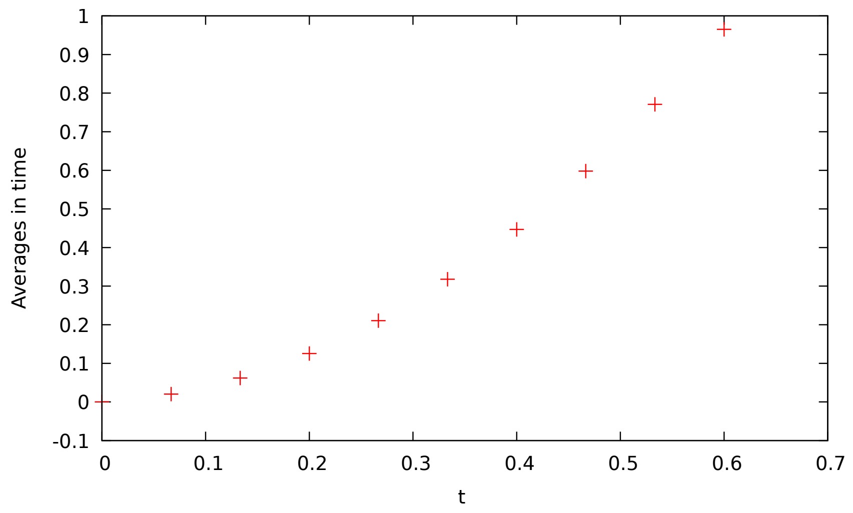
Results

Fourier Transform

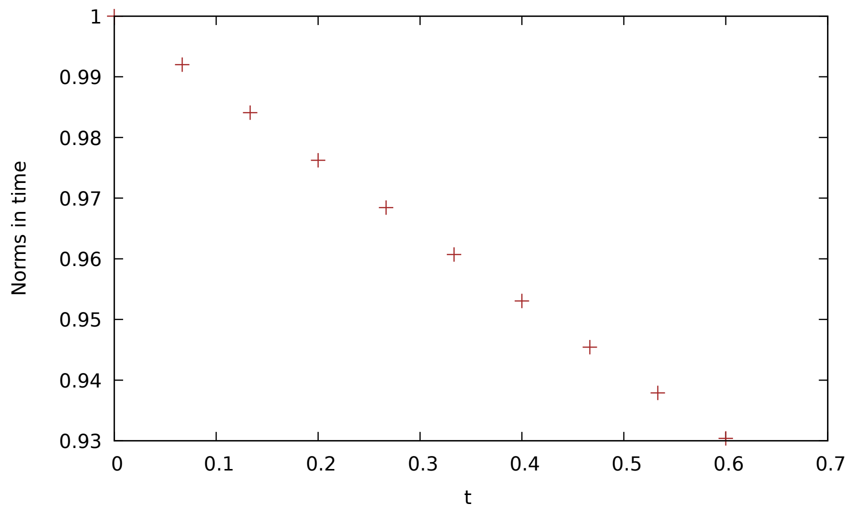
We decided to consider different T : $\{0.1,\ 0.6,\ 1.5,\ 5.0\}$, with 10 istants and 250 space points. The cause of this low-number choice is that the program gave **Segmentantion fault** as soon as more than 10 time istants were chosen. Instead, the error **Program received signal SIGABRT: Process abort signal** was given for 10 time istants, but actually worked. We noticed that the expected result (the translation of one unit in advance of the initial ψ) is obtained for $T = 0.6$.



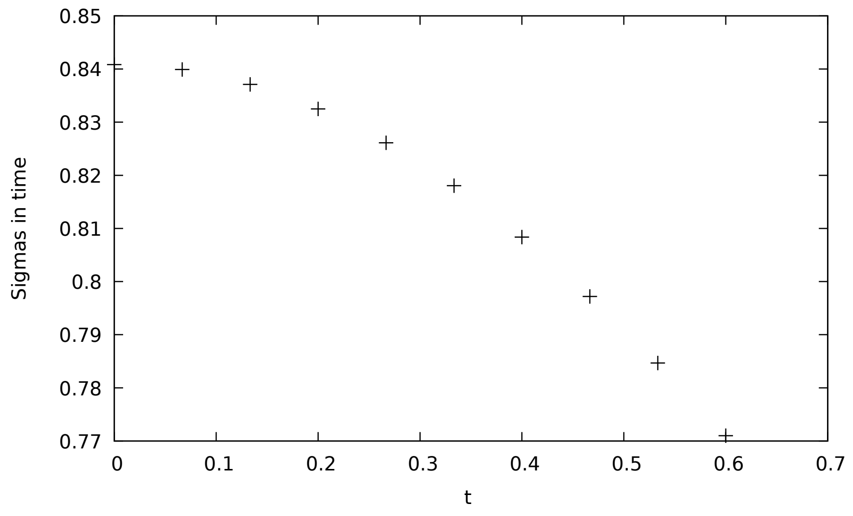
Time evolution of ψ_0 in $t \in [0, 0.6]$, with 10 istants. The lighter the blue, the more advanced in time is the wavefunction. Fourier Tranform.



Time evolution of averages of ψ_0 in $t \in [0, 0.6]$, with 10 istants. Fourier Tranform.

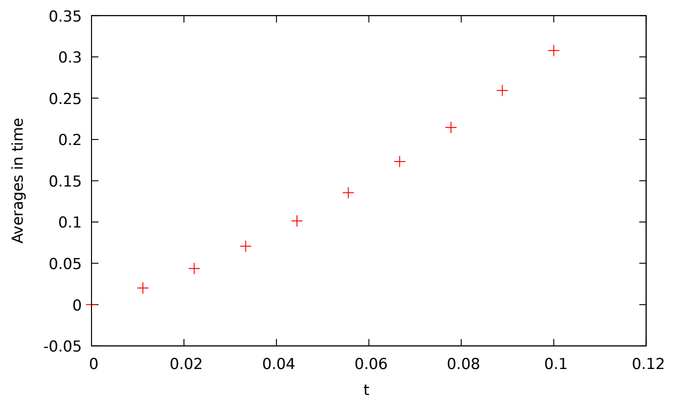


Time evolution of norms of ψ_0 in $t \in [0, 0.6]$, with 10 istants. Fourier Tranform.

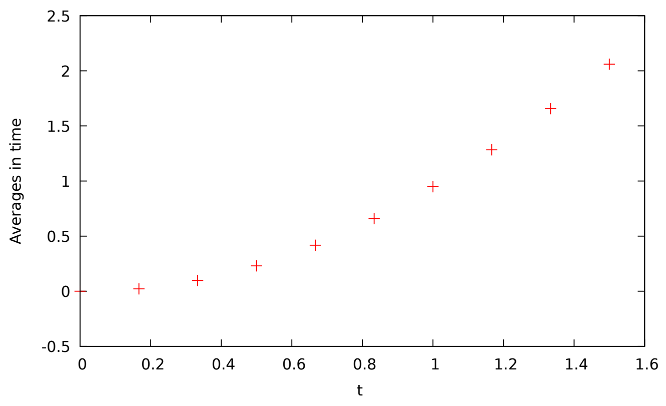


Time evolution of average of square distance from average of ψ_0 in $t \in [0, 0.6]$, with 10 istants. Fourier Tranform.

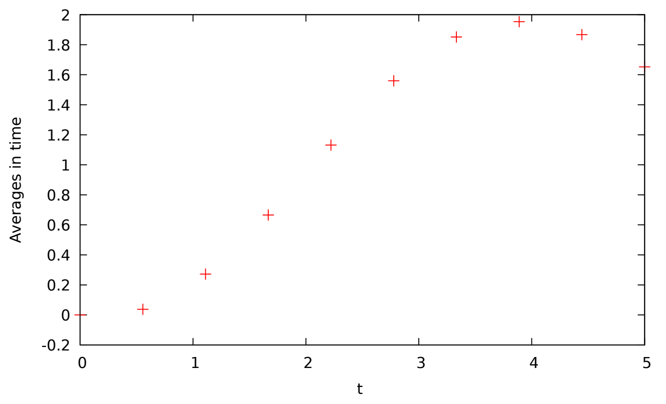
What we observe for the other $T = \{0.1,\ 1.5,\ 5.0\}$ is that the norms and the mean square distance from average go down with the same trend as in the $T = 0.6$ case, meaning that there is numerical instability intrinsic of the algorithm (the norms always go to 1.0 to 0.93). The averages instead, are different. For higher T, the more the wave function is translated. For $T = 5.0$ the wave function is bounced back. This is quite strange: we should have that the translation is indipendent on the time interval. The problem could be the fact that the time steps are not negligible for higher T (if the number of istants is fixed), but following this reasonment the $T = 0.1$ case should perform better than the $T = 0.6$ case.



Time evolution of averages of ψ_0 in $t \in [0, 0.1]$, with 10 istants. Fourier Tranform.



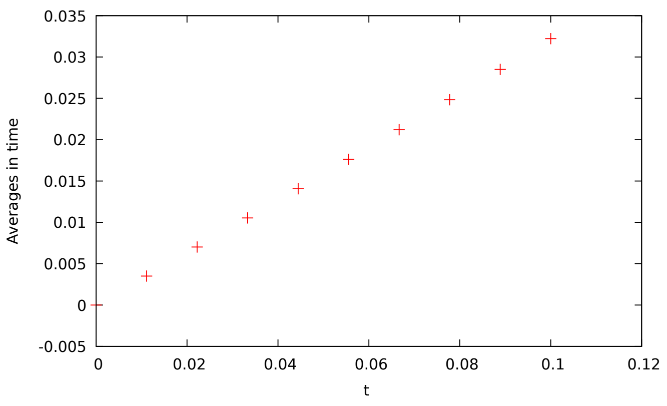
Time evolution of averages of ψ_0 in $t \in [0, 1.5]$, with 10 istants. Fourier Tranform.



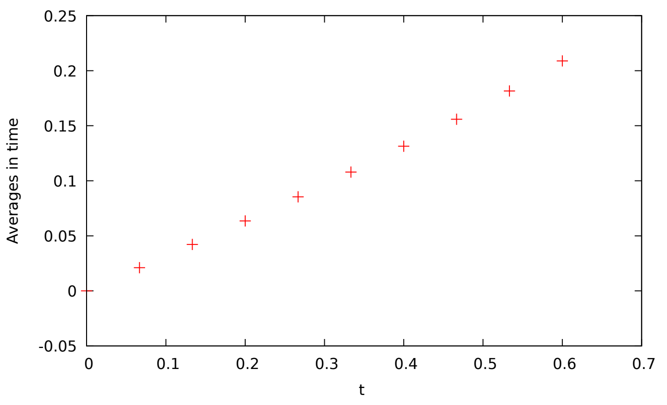
Time evolution of averages of ψ_0 in $t \in [0, 5.0]$, with 10 istants. Fourier Tranform.

Fast Fourier Transform

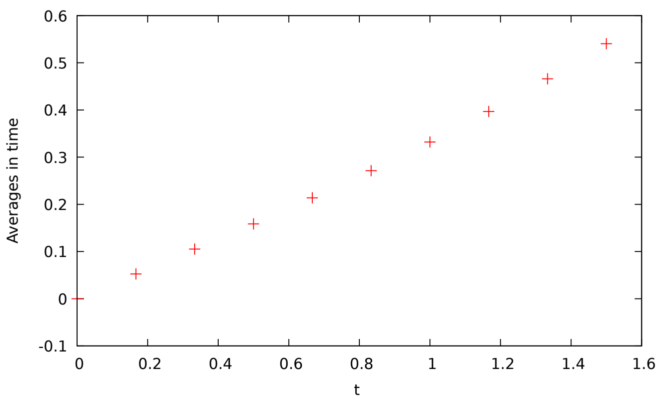
We decided to consider different $T : \{0.1,\ 0.6,\ 1.5,\ 5.0,\ 5.5\}$, with 10 istants and 250 space points. The cause of this low-number choice is that the program gave **Segmentantion fault** as soon as more than 10 time istants were chosen. Instead, the error **Program received signal SIGABRT: Process abort signal** was given for the 10 time istants case, but actually worked.
The following are the averages in time:



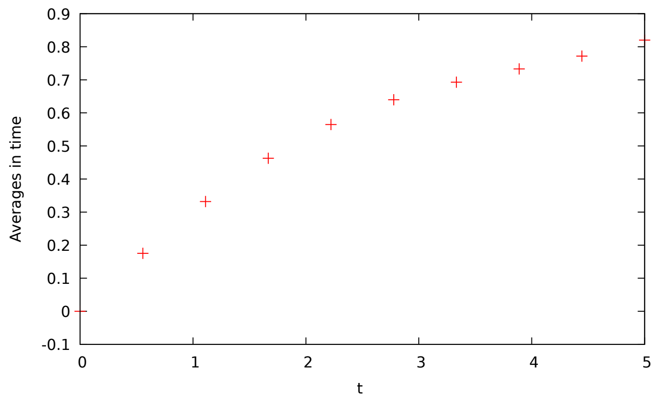
Time evolution of averages of ψ_0 in $t \in [0, 0.1]$, with 10 istants. Fast Fourier Transform.



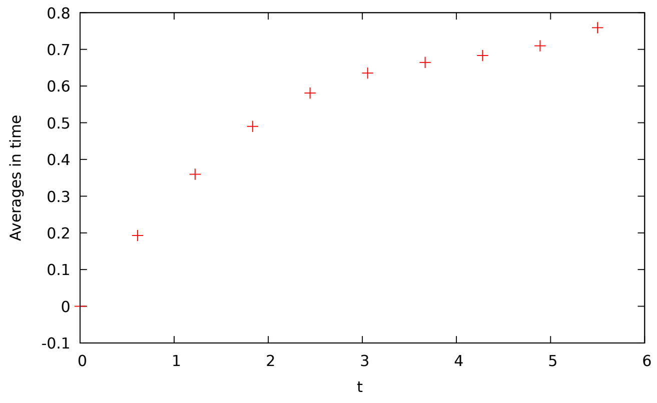
Time evolution of averages of ψ_0 in $t \in [0, 0.6]$, with 10 istants. Fast Fourier Transform.



Time evolution of averages of ψ_0 in $t \in [0, 1.5]$, with 10 istants. Fast Fourier Transform.

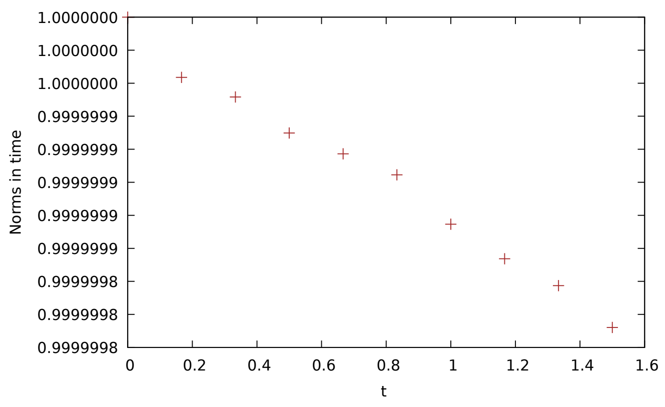


Time evolution of averages of ψ_0 in $t \in [0, 5.0]$, with 10 istants. Fast Fourier Transform.



Time evolution of averages of ψ_0 in $t \in [0, 5.5]$, with 10 istants. Fast Fourier Transform.

For higher T, the more the wave function is translated, until a certain point. Increasing T more than 5.0 the wave function translates less. This is quite strange: we should have that the translation is indipendent on the time interval. The problem could be the fact that the time steps are not negligible for higher T (if the number of istants is fixed), but following this reasonment the $T = 0.1$ case should perform better than the others, but that is not the case, beacause the wavefunction moves very little from the initial position. On the other hand, the norms are quite costant:



Time evolution of averages of ψ_0 in $t \in [0, 1.5]$, with 10 istants. Fast Fourier Transform.

Self-Evaluation

The main goals achieved:

- I have learnt how to write a Fourier Transform.
- I have learnt how to exploit a Fast Fourier Transform.
- I have learnt how to implement a Split Operator Method.

What can be done next:

- Understand why there is that Segmentation Fault for more than 10 istants, in both methods.
- Understand why there is the same loss of norm for each time interval $[0, T]$ chosen in the Fourier Transform method.
- Understand why the translation is dependent on the time interval $[0, T]$ chosen (with the number of points fixed).
- Understand why the algorithm does not work better for lower T (with the number of points fixed).

Bibliography

[1] C.M. Dion, A. Hashemloo and G. Rahali: *Program for quantum wave-packet dynamics with time-dependent potentials*. [arXiv:1306.3247v2], 12 Sep 2013.