

**Time Dependent Quantum Chemistry**  
**Professor. Atanu Bhattacharya**  
**Department of Inorganic and Physical Chemistry**  
**Indian Institute of Science, Bengaluru**  
**Mod 05 Lecture 36**

**Wavepacket Dynamics under Zero Interaction Potential: Numerical**

Welcome back to Python tutorial 5 of the course Time Dependent Quantum Chemistry. We have gone over just how to normalize the initial wavefunction. And we will proceed further to check how to now time propagate numerically. That is a very simple procedure we will follow.

(Refer Slide Time: 00:41)

Python Tutorial 5: Gaussian Wavepacket Dynamics

Wavepacket Dynamics under Zero *free particle*  
Interaction Potential: Numerical

Step 2: Time-Evolution of the Gaussian Wavepacket  $t = 1$   
 $V = 0$

$$\psi(x, t) = \left[ \prod_N e^{-i \frac{\hat{p}^2 \Delta t}{2\hbar}} e^{-i \frac{\hat{p}^2 \Delta t}{2\hbar}} e^{-i \frac{\hat{p}^2 \Delta t}{2\hbar}} \right] \psi(x, 0)$$

*N-times product*

Repeated

Time dependent Quantum Chemistry

Python Tutorial 5: Gaussian Wavepacket Dynamics

Wavepacket Dynamics under Zero *free particle*  
Interaction Potential: Numerical

Step 2: Time-Evolution of the Gaussian Wavepacket  $t = 1$   
 $V = 0$

$$\psi(x, t) = \left[ \prod_N e^{-i \frac{\hat{p}^2 \Delta t}{2\hbar}} e^{-i \frac{\hat{p}^2 \Delta t}{2\hbar}} e^{-i \frac{\hat{p}^2 \Delta t}{2\hbar}} \right] \psi(x, 0)$$

*N-times product*

*initial wavefunction*

*for loop*

$$\psi(x, t) = \left[ \prod_N e^{-i \frac{\hat{p}^2 \Delta t}{2\hbar}} \right] \psi(x, 0)$$

$\hat{T} = \frac{1}{2} K^2 = 0.5 K^2$

$\psi(x, t) = \left[ \prod_N e^{-i \frac{\hat{p}^2 \Delta t}{2\hbar}} \right] \psi(x, 0)$

*new wavefunction at time (0 + dt)*

Repeated

Time dependent Quantum Chemistry

The time propagation part will be done through the split operator approach. So, we have already realized and we have also found the recipe for implementing the split operator method which propagates the wavefunction from,  $t$  equals 0 to some other time. So, this is my initial wavefunction which I have already discretized and normalized it and this is your final wavefunction after the time  $t$ . And the way it is done is we make up a short time propagator like  $\Delta t$  short time propagation and then we just keep propagating it in an iterative process.

$$\Psi(x, t) = \left[ \prod_n e^{-i\frac{2\pi T \Delta t}{2h}} e^{-i\frac{2\pi V \Delta t}{h}} e^{-i\frac{2\pi T \Delta t}{2h}} \right] \Psi(x, 0)$$

So, what does it mean? I start with this function and the recipe we have already discussed is that we start with this wavefunction, we do a full Fourier transform to convert it to the momentum domain. In the momentum domain, I employ the kinetic energy operator. And so this, and then we come and then we come back to the position domain and then we keep repeating.

So,  $\Delta t$  propagation will be repeated until I reach the final time. So, that is the, that is the basic idea we have already developed it. Here, this sign is we have never used this sign before. This is nothing but the N times product that we have seen already. We have to do this N time, this short time propagation N times. So, this is called N times product, that is all, it should not puzzle anybody.

Now recall that for each time in the product, the potential part of the propagator is carried out in the position space. So, we just remember that. We said that this is the potential part and these two are the kinetic energy part. So, this is the symmetrized product of the split operator approach. We have seen that initially, this part has to be acting on this. Then this part will be acting on the resultant part. Then this part will be acting on the resultant part.

But when I am employing this potential, the potential part has to be carried out in the position space. And the kinetic part of the propagator is carried out in the momentum space that we have already understood. If you need recapitulation, you please go ahead and check the split operator approach, the module which is covering the split operator approach.

So, with this idea, in atomic unit, we have already considered that  $\hbar/2\pi$  cut equals 1. So, this part, this  $\hbar/2\pi$ , I can eliminate if we consider atomic unit. And for the free particle, we have considered, because it is a free particle, we can consider  $V$ , this is not velocity, this is the potential, this is the potential.  $V$  is 0, because we are dealing with 0 interaction potential that is called free particle.

So, in that case, if we do that, then in that case, under atomic unit and free particle -- for the free particle, this entire expression becomes like this.

$$\Psi(x, t) = \prod_n e^{-i \frac{2\pi T \Delta t}{h}} \Psi(x, 0)$$

This is what we have. And that is exactly needs to be repeated here following this scheme.

So, explicit numerical steps are summarized here. We start with the initial wave packet, which is the X,  $\Psi(x,0)$ . Then because we have to carry out the kinetic energy part in the momentum domain, K domain or P domain, both are equivalent, the expressions are different, but they are equivalent, so we have to convert this one to momentum domain, and that conversion can be done through fast Fourier transform.

We are now familiar with it, we will see the implementation again, and then we have to carry out this kinetic energy part and get an another wavefunction, which is displaced to this time  $\Delta t$ . So, after  $\Delta t$  time, we have now though wavefunction in the momentum domain. Then we have to inverse full year transform to the position domain. This wavefunction is now new wavefunction for the feed to the next iteration. And this needs to be iterated until I reach the final time. That is the basic idea. And final time is going to be  $N \Delta t$  time is the final time. So, N times I have to repeat this one.

Here this  $\Psi(x,0)$ , this represents the initial wavefunction. And this one, it represents a new wavefunction, new wavefunction at time T plus  $\Delta t$ , sorry 0 plus  $\Delta t$ . 0 plus delta t. So, delta t time has been advanced for this wavefunction. To implement the, this, this product recipe, this iterative product recipe, the sequences repeated with the help of a for loop. So, you use a for loop to repeat this one, for every  $\Delta t$  with  $\Psi$ , this is  $\Psi'$  will be serving as the new initial wavefunction for the next iteration of the loop. So, we will use the for loop for implementing this one.

We must select and sufficiently small Delta t that we have also understood why we should select this very small delta t to achieve higher accuracy in this computation, because the split operator approaches and approximate approach and the error goes with like this. So, higher order errors will be neglected if the delta t is small. And the kinetic energy operator on the K grid is represented by. So, this, there is a kinetic energy operator here, T. So, this kinetic energy

operator will be represented by, as I have mentioned previously in the atomic unit, it is going to be

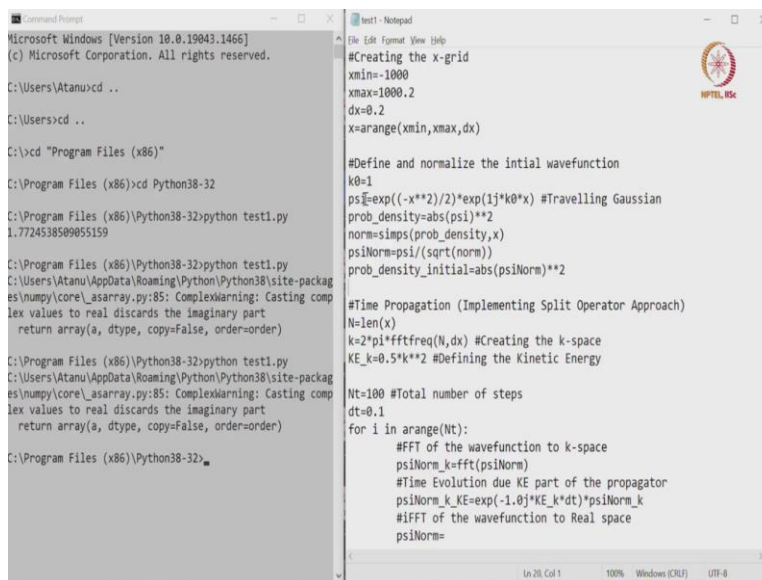
$$\hat{T} = \frac{1}{2} k^2$$

because it is for any K.

So, what is the kinetic energy? It is more like a K dependent function depending on what K I have. And so, in the K domain, I have the, there is a K domain I have created, discretized K domain. And for each K, I have a kinetic energy operator. So, what we can see that Kinetic energy in the K domain is actually quadratic, quadratic in nature. And sorry, quadratic in nature, which means like this. It spreads out between both 0, but both side of the 0. So, it is quadratic in nature in the momentum domain. And it should be half K square, because in the atomic unit. So, we will express it like 0.5 K, K square. That is kinetic energy of the kinetic energy operated in K space.

So, we will move, go ahead and check the function now. We will move to laptop and implement this.

(Refer Slide Time: 09:14)



```
Microsoft Windows [Version 10.0.19043.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Atanu>cd ..
C:\Users>cd ..
C:\>cd "Program Files (x86)"
C:\Program Files (x86)>cd Python38-32
C:\Program Files (x86)\Python38-32>python test1.py
1.7724538509055159
C:\Program Files (x86)\Python38-32>python test1.py
C:\Users\Atanu\AppData\Roaming\Python\Python38\site-packag
es\numpy\core\_asarray.py:85: ComplexWarning: Casting comp
lex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
C:\Program Files (x86)\Python38-32>python test1.py
C:\Users\Atanu\AppData\Roaming\Python\Python38\site-packag
es\numpy\core\_asarray.py:85: ComplexWarning: Casting comp
lex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
C:\Program Files (x86)\Python38-32>

test1 - Notepad
File Edit Format View Help
#Creating the x-grid
xmin=-1000
xmax=1000.2
dx=0.2
x=arange(xmin,xmax,dx)

#Define and normalize the intial wavefunction
k0=1
psi=exp((-x**2)/2)*exp(1j*k0*x) #Travelling Gaussian
prob_density=abs(psi)**2
norm=simps(prob_density,x)
psiNorm=psi/(sqrt(norm))
prob_density_initial=abs(psiNorm)**2

#Time Propagation (Implementing Split Operator Approach)
N=len(x)
k=2*pi*fftfreq(N,dx) #Creating the k-space
KE_k=0.5*k**2 #Defining the Kinetic Energy

Nt=100 #Total number of steps
dt=0.1
for i in arange(Nt):
    #FFT of the wavefunction to k-space
    psiNorm_k=fft(psiNorm)
    #Time Evolution due KE part of the propagator
    psiNorm_k_KE=exp(-1.0j*KE_k*dt)*psiNorm_k
    #IFFT of the wavefunction to Real space
    psiNorm=
```

```

Microsoft Windows [Version 10.0.19043.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Atanu>cd ..
C:\Users>cd ..
C:\>cd "Program Files (x86)"
C:\Program Files (x86)>cd Python38-32
C:\Program Files (x86)\Python38-32>python test1.py
1.772453850905159
C:\Program Files (x86)\Python38-32>python test1.py
C:\Users\Atanu\AppData\Roaming\Python\Python38\site-packag
es\numpy\core\_asarray.py:85: ComplexWarning: Casting comp
lex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
C:\Program Files (x86)\Python38-32>python test1.py
C:\Users\Atanu\AppData\Roaming\Python\Python38\site-packag
es\numpy\core\_asarray.py:85: ComplexWarning: Casting comp
lex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
C:\Program Files (x86)\Python38-32>

```

```

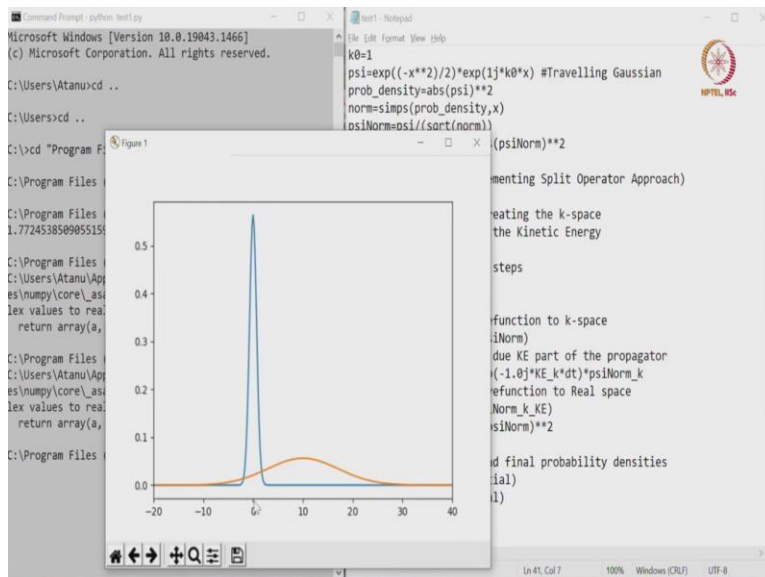
test1 - Notepad
File Edit Format View Help
#Define and normalize the intial wavefunction
k0=1
psi=exp((-x**2)/2)*exp(1j*k0*x) #Travelling Gaussian
prob_density=abs(psi)**2
norm=simps(prob_density,x)
psiNorm=psi/(sqrt(norm))
prob_density_initial=abs(psiNorm)**2

#Time Propagation (Implementing Split Operator Approach)
N=len(x)
k=2*pi*fftfreq(N,dx) #Creating the k-space
KE_k=0.5*k**2 #Defining the Kinetic Energy

Nt=100 #Total number of steps
dt=0.1
for i in arange(Nt):
    #FFT of the wavefunction to k-space
    psiNorm_k=fft(psiNorm)
    #Time Evolution due KE part of the propagator
    psiNorm_k_KE=exp(-1.0j*KE_k*dt)*psiNorm_k
    #IFFT of the wavefunction to Real space
    psiNorm=ifft(psiNorm_k_KE)
prob_density_final=abs(psiNorm)**2

#plotting the initial and final probability densities
plot(x,prob_density_initial)
plot(x,prob_density_final)

```



So the first few lines will be kept here from scipy. We have to import the same square root functionality, arange functionality, exponential functionality. In addition to that, I will also export pi, sorry, import pi. And there is a reason why we are importing pi, because for the preparation of the k-grid, I need this pi.

From integrate, I have to import again Simpson's method. And then I have to form, because we will make use of Fourier transform, we have to import some functionalities from scipy, fftpack sub-module import fft frequency, fft and ifft, these are the functionalities we are going to import. And then remaining part, matplotlib library will help us plot the function and creating the x-grid

remains to be the same. We have the x-grid, then we have to define the normal -- define and normalized the initial function, that we are keeping the same is a traveling Gaussian.

And then we have the normalized wavefunction. And once we have the normalized wave function, we can write down probability or  $\psi$ Norm, probability, density, initial. So, we will compare with the initial probability density. Probability density is nothing but absolute value of this  $\psi$ Norm square. So, we are writing this one because we would like to check the probability density, we would like to plot initial probability density and the final probability density after the iteration.

Now we will go back, go to the important step, the iteration, how we are going to start the iteration. But before we start the iteration, time propagation, we have to create the k space first, because we need to change to k space to carry out the potential part, sorry, kinetic energy part of the split operator component.

So, we will write down time, propagation, implementing, split, operator, approach N equals length of x, I need to know that. Because when you define K, I will define K by  $2\pi$  multiplied by the frequency components, fft frequency components. For that, I have two inputs in number of grid points I have in the x space as well as the separation of grid points. These are the two things. And this is the way we create the k space.

And then I am going to define the kinetic energy in the k space and that definition is very simple. It is going to be  $0.5$  multiplied by the K, each K multiple square. So, this is the definition of the kinetic energy defining the kinetic energy. Once the kinetic energy definition is over, then I can now define the Nt. Now Nt is total number of steps I am going to carry out. This is going to be total number of steps. And each step size, dt is taken to be  $0.1$ .

So, note that if I take dt to be  $0.1$ , what is the unit? It is in second. No, it is in all in atomic unit. So, dt is the timestep, it is taken to be atomic unit of time here, x is taken to be  $0.2$ , which is the separation  $0.2$  is in again atomic unit of length and that is the way we are moving. So, this is, these are presented in atomic unit.

Now I am going to implement the for loop for the, for the iteration part. So, the way we are going to implement the fall loop, it is going to be i in a range functionality. This is something which, we know, arange, for arange functionality, we use, we need, generally we need three

inputs. One is a starting point, end point and the division. But arange functionality, another format of arange, arrange functionality is there, it is going to be how it is the endpoint of this.

In that case, if we use this arange functionality with single input, it means that a single input is considered to be the end point. And definitely, endpoint will not be considered, so it will be considering only the point before the end point. But what it would be giving you is the -- it will be giving you the numbers separated by one. So, it will start with 0, then 1, 2, 3, 4, it will continue until it reaches 99.

So, how many iterations it will undergo? It will undergo a 100 iteration, but it will start, the index will start, the index is actually here  $i$ , index would be starting from 0, and we will end at 99. So, that is the basic idea. So, the first thing in this iteration we are going to do is there first we have Fourier transform the wavefunction, Fourier transform of the wavefunction. When you do the Fourier transform wavefunction to  $k$ -space. When you do the Fourier transform the wavefunction, we get that  $\psi_{\text{Norm}}$ .  $\psi_{\text{Norm}}$ ,  $\psi_{\text{Norm}}$ . We get  $\psi_{\text{Norm}}$  in the Fourier space, which is nothing but  $\text{fft}$  of  $\psi_{\text{Norm}}$ .

So,  $\psi_{\text{Norm}}$  underscore  $K$  is the representing the wavefunction in that Fourier domain. Then I have to use the Time Evolution. We have to carry out kinetic part of the split operator. Here, only we have kinetic part, so split operator is nothing but the kinetic part only. There is no potential part that I can carry out like this way. I name it  $\text{KE}$  equals, it is going to be exponential off minus  $1.0j$  multiplied by kinetic energy, multiplied by  $dt$ . That is the kinetic energy part of it. This part will be multiplied by the fourier transform part. So, this is a scalar multiplication. So, each point will be, each is element wise multiplication for this.

So, note that  $\text{KE}$ ,  $\text{KEK}$  underscore  $K$ , it is an array. And this array, when you are multiplying by  $dt$  and minus  $ij$  which is the complex number, it is giving you another array with this multiplication, element-wise multiplication. So, this, this, this entire thing is an array. This array is multiplied by this array, which means that I am element wise multiplying it and then getting the, another array. So, this is again an another array.

So, this is the way we have performed the time propagation  $\Delta t$ , and then what we need to do is that  $\text{iFFT}$  inverse fourier transform of the wavefunction to Real space. And for that, all we need to do is I write down  $\psi_{\text{Norm}}$ . Again, I am going to update this one. So, here I start with

this  $\psi_{\text{Norm}}$ , the iteration is started with  $\psi_{\text{Norm}}$ . When the iteration is started in  $\psi_{\text{Norm}}$ , you check this,  $\psi_{\text{Norm}}$  is actually a normalized wavefunction, this one. I start with it. But when, during the iteration, this  $\psi_{\text{Norm}}$  will be updated.

So, in the second iteration, after the first step, that second iteration, this  $\psi_{\text{Norm}}$  norm will be updated to this  $\psi_{\text{Norm}}$ . So, that is the way iteration continues. So, I am now updating the  $\psi_{\text{Norm}}$  value  $i\text{FFT}$ , which is nothing but  $i\text{FFT}$  of this function. So in the end, so this iteration will continue until  $Nt$  is satisfied, this -- until the endpoint of this array,  $NT$  array. And then once it is satisfied, finally, I get  $\psi_{\text{Norm}}$ , the updated value of the  $\psi_{\text{Norm}}$ . That is the, so I will find out the probability density, final which is going to be nothing but absolute value of the final updated  $\psi_{\text{Norm}}$ .

And now if I, plotting everything, it will be easier to visualize what I am doing, plotting the initial and final probability densities. Plot  $X$  versus initial probability density. Plot  $X$  versus final probability density. These are the two plots I will have. And because I have already done this problem, I know the limit should be set to be for better visualization. That is all. So, we can run the program and we see that the initial probability which was centered at 0 that is quite expected, this is a 0, which means the 0 length which means the in atomic unit.

So, it was centered at 0, it was normalized wavefunction, probability is normalized. So, the probability, so initial probability or the center of the probability was at  $X$  equals 0. But after time  $T$ , what is the time I am looking at?  $NT$  equals hundred. So, after a hundred atomic unit of time, what I get? The probability is centered almost near 10. So, probability is moving. And when the probability density is moving, it means that there is a quantum dynamics and that is the way we present them quantum dynamics.


So, we will move to the slides now. And we will see that this, this entire protocol has been implemented through this, through the, through this split operator approach.



(Refer Slide Time: 23:01)

Python Tutorial 5: Gaussian Wavepacket Dynamics

Wavepacket Dynamics under Zero Interaction Potential: Numerical



**Step 2: Time-Evolution of the Gaussian Wavepacket**

```
#Importing the Required Libraries
from scipy import sqrt, arange, exp, pi
from scipy.integrate import simpson
from scipy.fft import fftfreq, fft, ifft
from matplotlib.pyplot import plot, xlim, show

#Creating the x-Grid
xmin=-1000
xmax=1000.2
dx=0.2
x=arange(xmin,xmax,dx)

#Defining and Normalizing the Initial Wavefunction
k0=1
psi=exp(-(x**2)/2)*exp(1j*k0*x) #Travelling Wavepacket
prob_density=abs(psi)**2
norm=simpson(prob_density,x)
psiNorm=psi/sqrt(norm)
prob_density_initial=abs(psiNorm)**2
#Time Propagation (Implementing Split Operator Approach)
N=len(x)

k=2*pi*fftfreq(N,dx) #Creating the k-Grid
KE_k=(0.5*k**2) #Defining the Kinetic Energy
Nt=100 #Total Number of Time Steps
dt=0.1 # atomic unit
for i in arange(Nt):
    #FFT of the Wavefunction to k-Space
    psiNorm_k=fft(psiNorm)
    #Time Evolution due to KE Part of the Propagator
    psiNorm_k=exp(-1.0j*KE_k*dt)*psiNorm_k
    #iFFT of the Wavefunction to Real Space
    psiNorm=ifft(psiNorm_k)
prob_density_final=abs(psiNorm)**2
#Plotting the Initial and Final Probability Densities
plot(x,prob_density_final)
plot(x,prob_density_initial)
xlim(-20,40)
show()
```

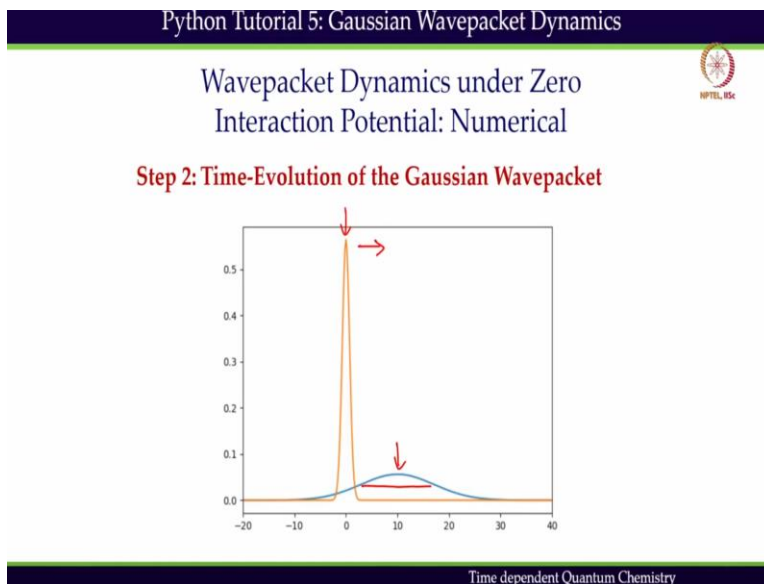
Time dependent Quantum Chemistry

Now, a couple of things can be interesting here to note again is that, these steps we are, these steps can be one more time clarified here. We are considering  $Nt$ ,  $Nt$  to be 100 atomic unit of time. After a 100 atomic unit of time, where the probability density is, and that is the final probability density we have found. So, this is something which you should remember that, and the  $\Delta T$  we have taken to be 0.1 atomic unit.

And one more thing we have to remember that the, in the beginning of the iteration, we have this  $\psi_{\text{Norm}}$  normalized wavefunction that is used here. But then so that is used here in the first iteration, but then this  $\psi_{\text{Norm}}$  will be updated and that is how you have kept the same name and it will be keep iterating until I get this 11, 100 step over. The moment 100 step is over, then I get the probability density as a final probability density.

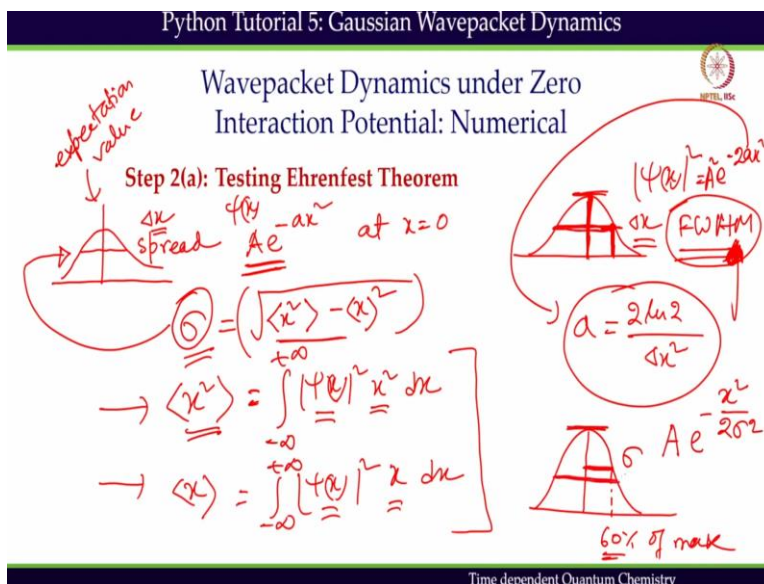
So, this is the major step, one can take a look at this part, this is the time propagation part, how we are doing this time propagation. As you can see that the kinetic part of the propagator has been implemented our carried out in the momentum space only.

(Refer Slide Time: 24:58)



So, with this we see that the center of the Gaussian is moving and as center of the Gaussian is moving, and not only that, it is spreading out also, and its amplitude is going down, it is spread out. And so this is the quite natural expectation for the particle, a free particle what we have already concluded from analytical approach previously.

(Refer Slide Time: 25:31)



Now we want to be a little more quantitative, quantitative in the sense that, we can test Ehrenfest theorem. Ehrenfest theorem suggests that the cent, the expectation value will follow the, it will follow the classical trajectory. So, a Gaussian function, if I have a Gaussian function like this, it

is characterized by its center position and its width, both are important characterization of the Gaussian function.

So, this suggests the spread of the Gaussian distribution and this shows the mean position or they expectation value of the Gaussian. So, this is the Gaussian distribution. And we, in general, express

$$Ae^{-ax^2}$$

that is the way we express it. So, this Gaussian distribution is centered at  $x$  equals 0, and its width is given by  $\Delta x$ .

Although  $\Delta x$  is not defined with respect to the wavefunction, it is defined, remember, it is defined with respect to probability density distribution. So, if this is the wavefunction, then probability density distribution looks like this.

$$|\Psi(x)|^2 = A^2 e^{-2ax^2}$$

In this distribution function, we define  $\Delta x$ , this is something which we have emphasized previously also.

And this  $\Delta x$  has many ways you can define that  $\Delta x$ . It can be full width half max or it can be, full width half max, what does it mean? It means that the width where the intensity has dropped down the half of its maximum intensity, that is called full width half max. So, that is the way in, the distribute, the spread of the, of a Gaussian function can be described.

And if we describe it as a full width half max, then we get immediately, this is also we have seen, this

$$a = \frac{2 \ln 2}{\Delta x^2}$$

that is we have -- we get it. So, full width half max,  $\Delta x$  is defined with respect to its distribution function not with respect to the wavefunction, this is something we should remember

that, that is the convenience, that is, this is the standard procedure we follow always. And then based on that only, we defined A equals this one.

Now in different context, when you deal with Gaussian distribution, it is the, the width is also expressed in terms of its variance. And that is defined

$$\sigma = \sqrt{\langle x^2 \rangle - \langle x \rangle^2}$$

like this. It is the variance. And if we look at that variance, so

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} |\Psi(x)|^2 x^2 dx$$

That is the value, that is the expression. And expectation value of x

$$\langle x \rangle = \int_{-\infty}^{\infty} |\Psi(x)|^2 x dx$$

that is the way, expectation value of x is given. So, graphically, if we think about it.

So, this is what we get from expression, that mathematical expression. And from this mathematical expression, we get this sigma. But sigma is the actually I told you, sigma is representing some kind of spread of the Gaussian distribution. And what is that spread is showing, in general, one can use a simple math and can show that this sigma is actually presenting the half width, sorry, this is, this is here. Sigma is a presenting, this is the sigma. So, sigma is representing half width, it is not the full width just like this, it is not the like full width, it is the half width at almost 60 percent of the maximum. So, this is actually 60 percent of maximum.

And one can prove it very easily the way we have calculated full width half max, similar way one can calculate it and can prove it. And in that case, the Gaussian distribution takes this following form.

$$A e^{-\frac{x^2}{2\sigma^2}}$$

Gaussian the wavefunction takes this form in terms of sigma.

So, sigma which can be calculated analytically, because as long as we know the wavefunction, we can find, we can multiply, this is multiplication, we can multiply by  $x$ , so both arrays has to be multiplied and we can get this two values, we can subtract and get the square root of that value. I can immediately get sigma. So, sigma can be very easily calculated and it can be implemented in the Python programming that way we have learned already.

And that is the reason why you are going to use sigma to describe the spread of Gaussian distribution. But the definition, graphical definition of sigma is following. I repeat one more time, in contrary to the full width half max, what we have learned, full width half max is the full width at the half maximum. Maximum is here, so at the half of the maximum, it is the width. But sigma, which is related to variance is also representing the spread of the Gaussian, but that corresponds to the half width, this is the half width, we are not taking full width here, this is the half width at about 60 percent off the maximum.

So, maximum is here, 60 percent of the maximum is here. So, full width half max somewhere here. So, this is  $\Delta X$ . So, this is the difference, but still sigma can represent the variation in this. So, we will go back to this testing, this entire thing in laptop right now, and we will see how it is going to, can be implemented in the Python programming.

(Refer Slide Time: 32:32)

## Python Tutorial 5: Gaussian Wavepacket Dynamics

### Wavepacket Dynamics under Zero Interaction Potential: Numerical



#### Step 2(a): Testing Ehrenfest Theorem

```
#Importing the Required Libraries
from scipy import sqrt,arange,exp,pi
from scipy.integrate import.simps
from scipy.fft import fftfreq,fft,ifft
from matplotlib.pyplot import plot,xlim,show

#Creating the x-Grid
xmin=-1000
xmax=1000.2
dx=0.2
x=arange(xmin,xmax,dx)

#Defining and Normalizing the Initial Wavefunction
k0=1
psi=exp((-x**2)/2)*exp(1j*k0*x) #Travelling Wavepacket
prob_density=abs(psi)**2
norm=simps(prob_density,x)
psiNorm=psi/sqrt(norm)
prob_density_initial=abs(psiNorm)**2
#Time Propagation (Implementing Split Operator Approach)
N=len(x)

k=2*pi*fftfreq(N,dx) #Creating the k-Grid
KE_k=(0.5*k**2) #Defining the Kinetic Energy
Nt=100 #Total Number of Time Steps
dt=0.1
for i in arange(Nt):
    #FFT of the Wavefunction to k-Space
    psiNorm_k=fft(psiNorm)
    #Time Evolution due to KE Part of the Propagator
    psiNorm_k_KE=exp(-1.0j*KE_k*dt)*psiNorm_k
    #iFFT of the Wavefunction to Real Space
    psiNorm=ifft(psiNorm_k_KE)
prob_density_final=abs(psiNorm)**2
avg1=simps(prob_density_final*x,x)
avg2=simps(prob_density_initial*x,x)
variance=sqrt(avg2-avg1**2)
print("width=%f"%variance)
print("<>=%f"%avg1)
```

Time dependent Quantum Chemistry

```
Microsoft Windows [Version 10.0.19043.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Atanu>cd ..
C:\Users>cd ..
C:\>cd "Program Files (x86)"
C:\Program Files (x86)>cd Python38-32
C:\Program Files (x86)\Python38-32>python test1.py
1.7724538509055159

C:\Program Files (x86)\Python38-32>python test1.py
C:\Users\Atanu\AppData\Roaming\Python\Python38\site-packag
es\numpy\core\_asarray.py:85: ComplexWarning: Casting comp
lex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)

C:\Program Files (x86)\Python38-32>python test1.py
C:\Users\Atanu\AppData\Roaming\Python\Python38\site-packag
es\numpy\core\_asarray.py:85: ComplexWarning: Casting comp
lex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)

C:\Program Files (x86)\Python38-32>
```

```
test1 - Notepad
File Edit Format View Help
k0=1
psi=exp((-x**2)/2)*exp(1j*k0*x) #Travelling Gaussian
prob_density=abs(psi)**2
norm=simps(prob_density,x)
psiNorm=psi/(sqrt(norm))
prob_density_initial=abs(psiNorm)**2

#Time Propagation (Implementing Split Operator Approach)
N=len(x)
k=2*pi*fftfreq(N,dx) #Creating the k-space
KE_k=0.5*k**2 #Defining the Kinetic Energy

Nt=100 #Total number of steps
dt=0.1
for i in arange(Nt):
    #FFT of the wavefunction to k-space
    psiNorm_k=fft(psiNorm)
    #Time Evolution due KE part of the propagator
    psiNorm_k_KE=exp(-1.0j*KE_k*dt)*psiNorm_k
    #iFFT of the wavefunction to Real space
    psiNorm=ifft(psiNorm_k_KE)
prob_density_final=abs(psiNorm)**2

#plotting the initial and final probability densities
plot(x,prob_density_initial)
plot(x,prob_density_final)
xlim(-20,40)
show()
```

```

C:\Program Files (x86)\Python38-32>python test1.py
C:\Users\Atanu\AppData\Roaming\Python\Python38\site-packages\numpy\core\_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)

C:\Program Files (x86)\Python38-32>python test1.py
C:\Users\Atanu\AppData\Roaming\Python\Python38\site-packages\numpy\core\_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)

C:\Program Files (x86)\Python38-32>python test1.py
width:7.106335
<xx>=10.000000
Traceback (most recent call last):
  File "test1.py", line 49, in <module>
    plot(x,psiNorm)
NameError: name 'plot' is not defined

C:\Program Files (x86)\Python38-32>python test1.py
width:7.106335
<xx>=10.000000

C:\Program Files (x86)\Python38-32>python test1.py
width:0.721110
<xx>=0.200000

C:\Program Files (x86)\Python38-32>python test1.py
width:1.581139
<xx>=2.000000

C:\Program Files (x86)\Python38-32>

```

```

norm=simps(prob_density,x)
psiNorm=psi/(sqrt(norm))
prob_density_initial=abs(psiNorm)**2

#Time Propagation (Implementing Split Operator Approach)
N=len(x)
k=2*pi*fftfreq(N,dx) #Creating the k-space
KE_k=0.5*k**2 #Defining the Kinetic Energy

Nt=20 #Total number of steps
dt=0.1
for i in arange(Nt):
    #FFT of the wavefunction to k-space
    psiNorm_k=fft(psiNorm)
    #Time Evolution due KE part of the propagator
    psiNorm_k_KE=exp(-1.0j*KE_k*dt)*psiNorm_k
    #IFFT of the wavefunction to Real space
    psiNorm=ifft(psiNorm_k_KE)
prob_density_final=abs(psiNorm)**2

avg1=simps(prob_density_final*x,x)
avg2=simps(prob_density_final*x**2,x)
variance=sqrt(avg2-avg1*avg1)
print("width=%f"%variance)
print("<xx>=%f"%avg1)

```

So, everything remains to be the same, I do not need to change anything. These are, should be there, but in the end, instead of plotting things, we do not need to plot it right now. We will do one thing, so we can remove this matplotlib library as well. Instead of plotting, what we will do, we will find out an average value, avg1 is defined as simpson method we will be using and we will find out the probability density final multiplied by x, x.

So, this is your, the first integration we discussed. The probability density has to be multiplied by x and then integrate over X. So, that is exactly what we have implemented here. Then another average we are defining avg2 as the integration. And in this integration, you have to multiply by x square.

So, x square will be multiplied, or we can say x multiply by x, that is also fine. So, x square will be multiplied by the probability density, and then we integrate over x, and variance would be defined by square root of this avg two minus avg one. These two values has to be subtracted, sorry. Avg2 minus avg1 multiplied by avg1. Remember, it is the difference between, square root of the difference between the expectation value of the x square, that is the value, and square of the expectation value of x, that is the way it is.

So, that is the way we have written it and then we can print it, print width equals percent F percent, this is the formatted print, percent I am going to width is actually the defined by the variance in our calculation, so I will write down percent variance. So, we have one place kept

within this quote, this, in this location, the variance value will be inserted. That is the way it will be printing.

Another thing will be printing here is that the center of the Gaussian which is the expectation value of the Gaussian and expectation value of the Gaussian is shown by this way. And now this slot, this percentage slot will be fueled by the expectation value of x which is the average one. This is the expectation value, and we can now run the program. So, if we run the program, we, we should, we should delete these things, because we are not defining anymore.

So, now we can run the program and we see that for this hundred after the hundred atomic unit of time, we get the width and the and average position. So, we can keep doing it, we can change the a 100 from 10, we can, we can use anything, we can use two and then run it. I get the width and the center position, I can make it, this was  $Nt$  to be 20, and I can see that with the end, Gaussian center position is changing. So, everything is changing. So, if we keep doing it, then in the end, we will go back to this slide right now.

(Refer Slide Time: 37:51)

Python Tutorial 5: Gaussian Wavepacket Dynamics

Wavepacket Dynamics under Zero  
Interaction Potential: Numerical

Step 2(a): Testing Ehrenfest Theorem

```
#Importing the Required Libraries
from matplotlib.pyplot import plot,xlim,show

x=[0,2,5,10,15,20,30,40,60,100]
width=[0.707,0.721,0.7905,1.0,1.27,1.58,2.23,2.91,4.3,7.1]
expectation_value=[0,0.2,0.5,1.0,1.5,2.0,3.0,4.0,6.0,10.0]

plot(x,width,color='red')
plot(x,expectation_value,color='blue')
show()
```

$x = at$   
 $\frac{dk}{dt} = v = a$   
 constant

Time dependent Quantum Chemistry



Wavepacket Dynamics under Zero  
Interaction Potential: Numerical

## Step 2(a): Testing Ehrenfest Theorem

```
#Importing the Required Libraries
from scipy import sqrt, arange, exp, pi
from scipy.integrate import.simps
from scipy.fftpack import fftfreq, fft, ifft
from matplotlib.pyplot import plot, xlim, show

#Creating the x-Grid
xmin=-1000
xmax=1000.2
dx=0.2
x=arange(xmin,xmax,dx)

#Defining and Normalizing the Initial Wavefunction
k0=1
psi=exp((-x**2)/2)*exp(1j*k0*x) #Travelling Wavepacket
prob_density=abs(psi)**2
norm=simps(prob_density,x)
psiNorm=psi/sqrt(norm)
prob_density_initial=abs(psiNorm)**2

#Time Propagation (Implementing Split Operator Approach)
N=len(x)

k=2*pi*fftfreq(N,dx) #Creating the k-Grid
KE_k=(0.5*k**2) #Defining the Kinetic Energy
Nt=100 #Total Number of Time Steps ←
dt=0.1
for i in arange(Nt):
    #FFT of the Wavefunction to k-Space
    psiNorm_k=fft(psiNorm)
    #Time Evolution due to KE Part of the Propagator
    psiNorm_k=KE_k*exp(-1.0j*KE_k*dt)*psiNorm_k
    #iFFT of the Wavefunction to Real Space
    psiNorm=ifft(psiNorm_k,KE)
    prob_density_final=abs(psiNorm)**2
    avg1=simps(prob_density_final*x,x)
    avg2=simps(prob_density_final**2,x,x)
    variance=sqrt(avg2-avg1**2) ←
    print("width=%f"%variance) ←
    print("<math>\langle x \rangle = %f</math>%avg1)
```

Time dependent Quantum Chemistry

And in the end, what we see is that we get some values. So, what we get is that if we use the -- this is time axis. And at 0 time, it is at the 0 position that is, we know, at 0 time and 0 position, and then a different time, it is actually, so this is X, this width versus, so this axis is representing the mean position as well as the width, and width is the sigma. So, both its representing. So, we can see that at starting, so at the starting point, so this part is that width and this is representing the sigma.

This is the width and this is the expectation value. What we see that the blue curve at the zero position, it is centered at 0. So, these are the points we get corresponding to X values. You can keep changing, this is actually time, but we are plotting it, this is the way one can plot further all these values. And if we plot it, we get this different time. So, this is time versus width, which is sigma, and this is the expectation value of X. And we can plot it. This is this something which we can do it.

So, after running this program for different Nt, different time, I want to check at different time, what is my final width and the position, expectation value of the position or center of Gaussian. If we do that, then we get these values and we just simply plot it, and if we plot it, we see that at the 0 time, when I start the dynamics, the center position is 0. That is expected because we started with center, 0, the wavefunction of centered at the 0 position, and it is linearly changing the position.

This is something we should note, because it is -- because it is linearly changing, it means that if X is linear with respect to time, let us say

$$x = at$$

it is linear with respect to time, then

$$\frac{dx}{dt} = v = a = \text{constant}$$

and that it should happen, because it is a free particle, free particle is not experiencing any potential, it is a 0 interaction potential. If no potential is there, which means there is no force acting on that. So, if there is no force on a particle, it should not change the momentum, it should not change the velocity, it will be forever traveling with that velocity only. So, that is something which is very interesting to note.

Second thing to note here is that, sigma the width spread is also almost linear with respect to time at a longer time. But then initial time, it is not a linear anymore, it is actually non-linear. So, the, the spread, how quickly it will be spreading, that is non-linear in the beginning, but later time, it is almost remained to be linear. Linearly, it is spreading out. So, these two facts have come off from the, from this study. We will we will stop here and we will look at the linear potential problem in the next session.