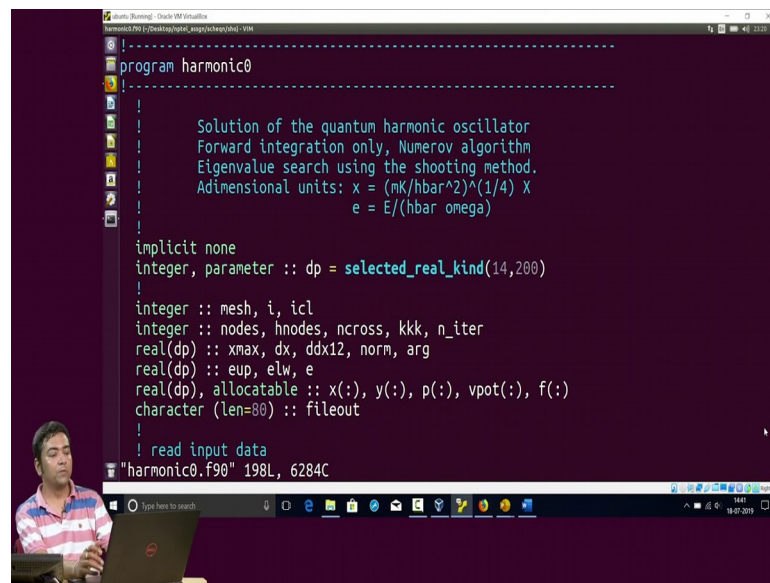


Computational Physics
Dr. Apratim Chatterji
Dr. Prasenjit Ghosh
Department of Physics
Indian Institute of Science Education and Research, Pune

Lecture – 40
Differential Equation for Quantum Mechanical Problems:
Numerov Algorithm Part 05

(Refer Slide Time: 00:17)

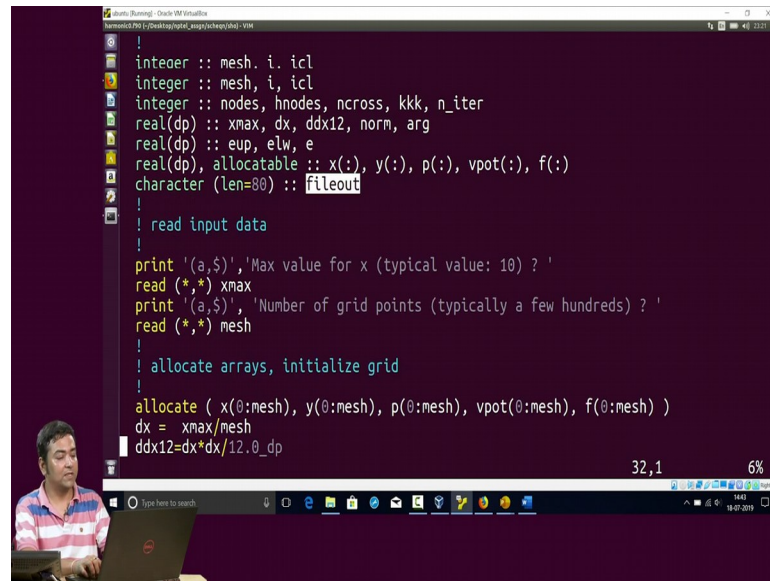


```
-----  
program harmonic0  
-----  
!  
! Solution of the quantum harmonic oscillator  
! Forward integration only, Numerov algorithm  
! Eigenvalue search using the shooting method.  
! Adimensional units:  $x = (mK/\hbar^2)^{1/4} X$   
!  $e = E/(\hbar\omega)$   
!  
implicit none  
integer, parameter :: dp = selected_real_kind(14,200)  
!  
integer :: mesh, i, icl  
integer :: nodes, hnodes, ncross, kkk, n_iter  
real(dp) :: xmax, dx, ddx12, norm, arg  
real(dp) :: eup, elw, e  
real(dp), allocatable :: x(:), y(:), p(:), vpot(:), f(:)  
character (len=80) :: fileout  
!  
! read input data  
"harmonic0.f90" 198L, 6284C
```

So, the code which I am going to show you now this particular code is using the numerical algorithm to solve the one dimensional quantum mechanical simple harmonic oscillator. So, this code we will be using the bisection method to compute the energy, but we have purposefully not implemented or not made sure that the code gives us the correct asymptotic limits.

This is done to just show you the effect that if you do not take care of the fact that the asymptotic limits of your solutions are correctly recognized by the code, then what might happen. So, this code is written here it is not written by me, but its written by Professor Paul G Anansi. So, I will just take you through the code.

(Refer Slide Time: 01:11)

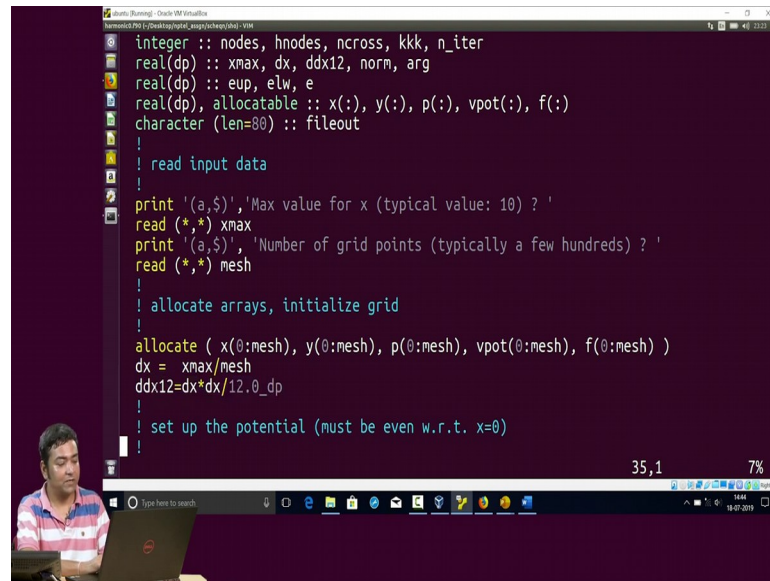


```
!
integer :: mesh, i, icl
integer :: mesh, i, icl
integer :: nodes, hnodes, ncross, kkk, n_iter
real(dp) :: xmax, dx, ddx12, norm, arg
real(dp) :: eup, elw, e
real(dp), allocatable :: x(:), y(:), p(:), vpot(:), f(:)
character (len=80) :: fileout
!
! read input data
!
print '(a,$)', 'Max value for x (typical value: 10) ? '
read (*,*) xmax
print '(a,$)', 'Number of grid points (typically a few hundreds) ? '
read (*,*) mesh
!
! allocate arrays, initialize grid
!
allocate ( x(0:mesh), y(0:mesh), p(0:mesh), vpot(0:mesh), f(0:mesh) )
dx = xmax/mesh
ddx12=dx*dx/12.0_dp
```

So, initially as like all other Fortran codes we have here the declaration of variables. So, basically this part is the declaration of variables. So, mesh is gives me the mesh size, icl is the index of my classical inversion points, then nodes gives the number of times the number of nodes I want the in my solution, n cross is the number of times my solution crosses or changes sign.

And then this is these are some temporary arrays this is the array where I am controlling the value of the potential at different values of x that is half k x square then this contains the name of the output file.

(Refer Slide Time: 02:01)

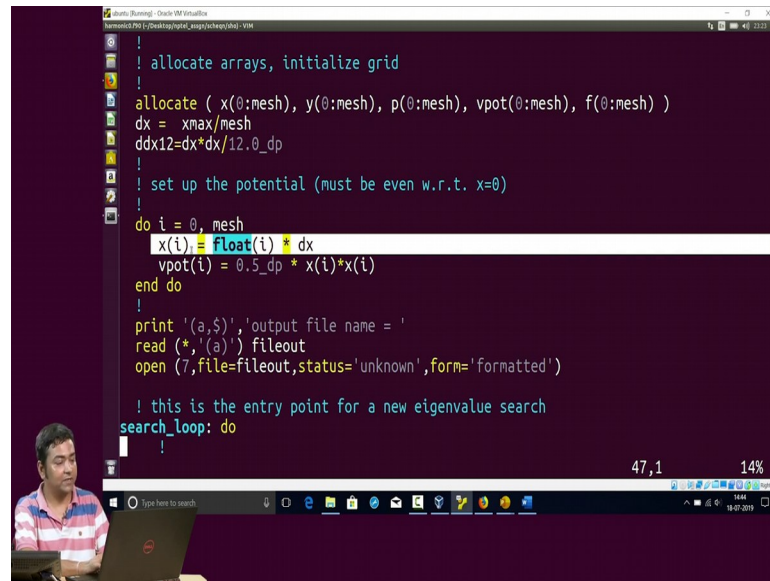


```
integer :: nodes, hnodes, ncross, kkk, n_iter
real(dp) :: xmax, dx, ddx12, norm, arg
real(dp) :: eup, elw, e
real(dp), allocatable :: x(:), y(:), p(:), vpot(:), f(:)
character (len=80) :: fileout
!
! read input data
!
print '(a,$)', 'Max value for x (typical value: 10) ? '
read (*,*) xmax
print '(a,$)', 'Number of grid points (typically a few hundreds) ? '
read (*,*) mesh
!
! allocate arrays, initialize grid
!
allocate ( x(0:mesh), y(0:mesh), p(0:mesh), vpot(0:mesh), f(0:mesh) )
dx = xmax/mesh
ddx12=dx*dx/12.0_dp
!
! set up the potential (must be even w.r.t. x=0)
!
```

So, what the code first does is it asks the value of x max that is up till what value of x or the maximum value of x till which we are going to do the integrate integration and the integration is done from minus x to x max to plus x max. But, in reality we do only from 0 to xmax and then take either depending on the whether it is an even whether you want an even number of nodes or odd number of nodes we make it symmetric or anti symmetric; mesh is the number of grid points you want to do the integration on and then using that we determine the grid size here.

So, x_{max} by mesh gives me the difference of the value of x between two consecutive grid points, we also allocate this local variables based on the mesh size. So, they this quantity is my Δx^2 by 12 which will be using several times. So, I have just calculated.... I mean this has been calculated in the very beginning of the code here.

(Refer Slide Time: 03:13)

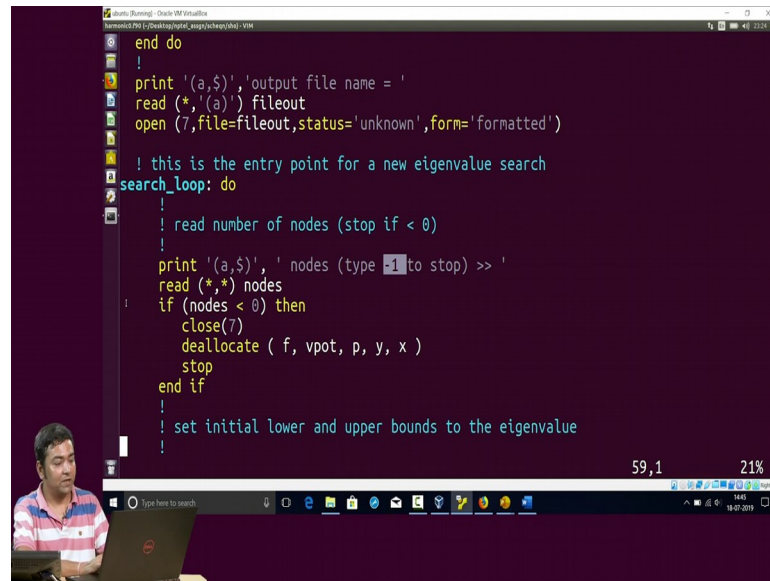


```
! allocate arrays, initialize grid
!
allocate ( x(0:mesh), y(0:mesh), p(0:mesh), vpot(0:mesh), f(0:mesh) )
dx = xmax/mesh
ddx12=dx*dx/12.0_dp
!
! set up the potential (must be even w.r.t. x=0)
!
do i = 0, mesh
  x(i) = float(i) * dx
  vpot(i) = 0.5_dp * x(i)*x(i)
end do
!
print '(a,$)', 'output file name = '
read (*, '(a)') fileout
open (7,file=fileout,status='unknown',form='formatted')
! this is the entry point for a new eigenvalue search
search_loop: do
  !
```

Now, what this part of the code is doing is it is setting up the potential. So, by potential you know is half kx square here since we have used reduced units my k is one. So, it is just half x square.

So, so what I do is I do the iterations from 0 to the mesh that is the maximum points, then I evaluate the value of x which is i and float means I am converting it to floating point here i is the index here into the dx that will give me the value of x at i th index. And once I know the value of x the i th index I get the value of the potential i and that is given by half kx square. Then I ask the user then this part of the code asks the user to support to give the name of the input file where or the output file where the data will be stored.

(Refer Slide Time: 04:11)

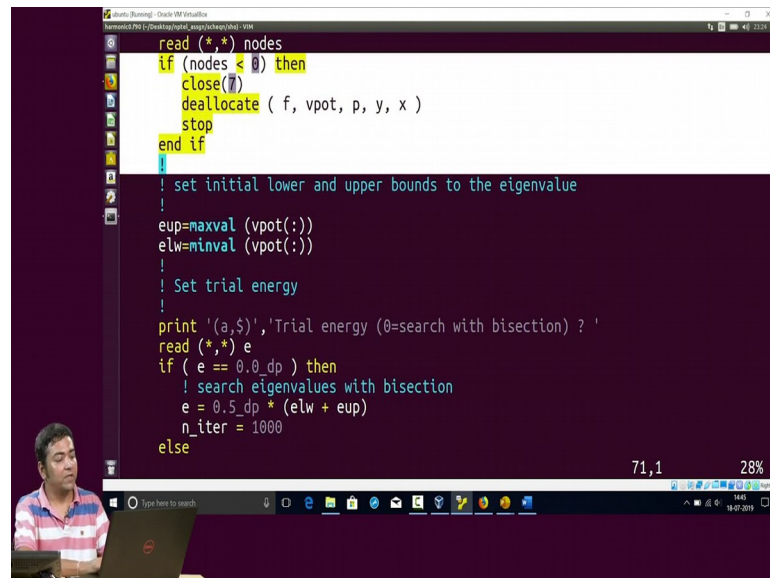


```
end do
!
! print '(a,$)', 'output file name = '
read (*, '(a)') fileout
open (7, file=fileout, status='unknown', form='formatted')

! this is the entry point for a new eigenvalue search
search_loop: do
! read number of nodes (stop if < 0)
!
! print '(a,$)', ' nodes (type -1 to stop) >> '
read (*, *) nodes
if (nodes < 0) then
close(7)
deallocate ( f, vpot, p, y, x )
stop
end if
!
! set initial lower and upper bounds to the eigenvalue
!
```

Then this part it is a its sort of the entry point for the new eigenvalue search. So, if you give a negative number of nodes, so the code will keep on doing it until and unless took. So, to come out of the code you need to give a negative value of the code and if it is negative value then it de allocates and then comes out of the code ok.

(Refer Slide Time: 04:39)



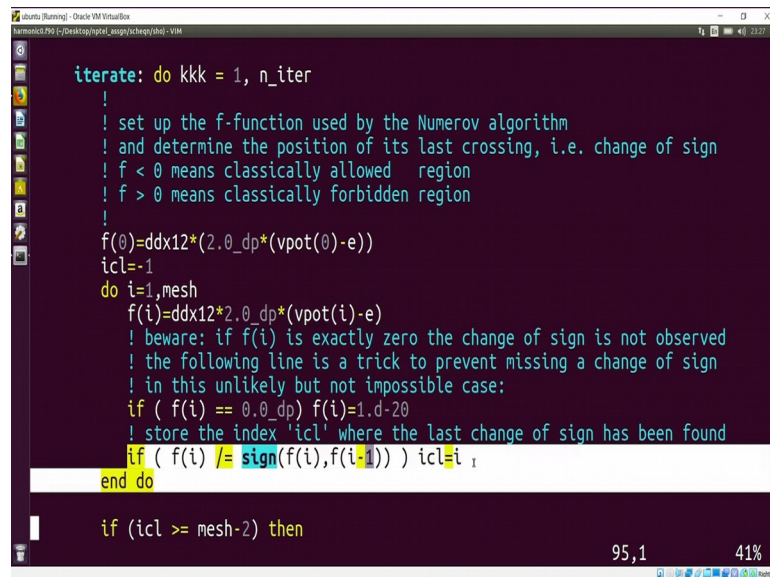
```
read (*, *) nodes
if (nodes < 0) then
close(7)
deallocate ( f, vpot, p, y, x )
stop
end if
!
! set initial lower and upper bounds to the eigenvalue
!
eup=maxval (vpot(:))
elw=minval (vpot(:))
!
! Set trial energy
!
print '(a,$)', 'Trial energy (0=search with bisection) ? '
read (*, *) e
if ( e == 0.0_dp ) then
! search eigenvalues with bisection
e = 0.5_dp * (elw + eup)
n_iter = 1000
else
```

So, then what we are doing is we are setting up our e max and e min. So, this is a eup is my upper limit or the maximum limit of the nth guess energy and elw is the lower limit

and this is given by the maximum value of the potential that is half $k m x_{\max}^2$ and the minimum value of the potential which in this case is 0.

So, once it does then again here there is a if sort of a detour here in this part of the code. So, the code automatically if you do not give any initial guess of energy what the code is will do. So, that if your trial energy if you set to 0 the code will start with the initial guess which is the midpoint of e_{up} and e_{lw} or else and it will do keep on doing the bisection method till it finds the energy or else if you just want to test for a single value of energy you will you will do you can give the input value of energy and the code will give the wave function.

(Refer Slide Time: 05:55)



```
iterate: do kkk = 1, n_iter
!
! set up the f-function used by the Numerov algorithm
! and determine the position of its last crossing, i.e. change of sign
! f < 0 means classically allowed region
! f > 0 means classically forbidden region
!
f(0)=ddx12*(2.0_dp*(vpot(0)-e))
icl=-1
do i=1,mesh
f(i)=ddx12*2.0_dp*(vpot(i)-e)
! beware: if f(i) is exactly zero the change of sign is not observed
! the following line is a trick to prevent missing a change of sign
! in this unlikely but not impossible case:
if ( f(i) == 0.0_dp) f(i)=1.d-20
! store the index 'icl' where the last change of sign has been found
if ( f(i) /= sign(f(i),f(i-1)) ) icl=i
end do

if (icl >= mesh-2) then
```

So, the this is my f at 0 that is at the beginning this is this dx^2 square delta x^2 by 12 into this potential which we are using several times, then what we do is we try to find out what is the classical point of inversion. So, in the beginning we set it to minus one and then we start the integration start going over the whole mesh from i equals to 1 to the x_{\max} from and at each point we calculate this quantity f_i .

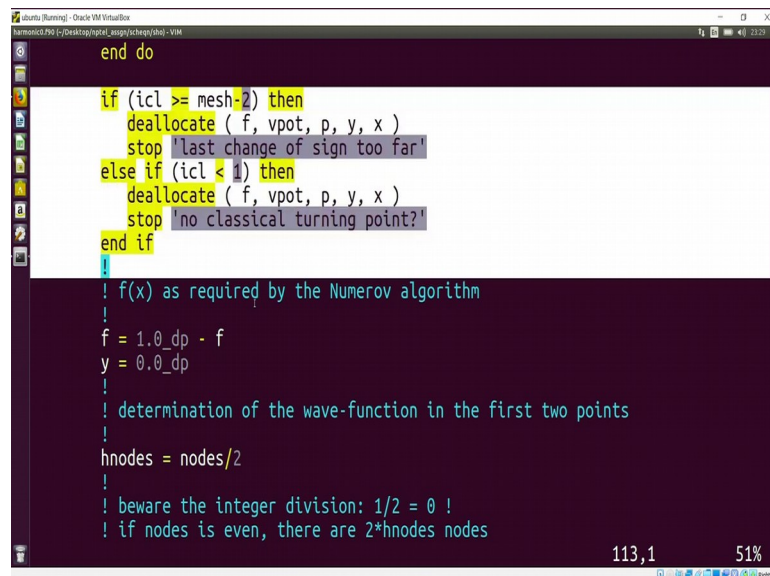
Now, beyond the classical point of inversion what will happen is this term will be. So, the f value will change sign. So, as long as your potential energy is less than potential energy is less than the energy of for which you are finding out the solution this term will be negative, the moment the potential energy is more than the energy for which you are finding the solution this term will be positive. So, the product of these two will give will

this will change sign and the point at where this change sign the value of i at where this change sign, then this will give you the point of classical inversion.

So, here he has also taken another care that is the following that sometimes it might happen that luckily you have this f_i becomes exactly 0 ok, then you would not be able to observe the sign. So, instead of because the way the sign is taken care of is it takes the product of these two functions and then it gives the sign. So, if it is 0 then you will sign you will not get this will not return any value and so we will not be able to find out. So, what is done is if it is exactly 0 this f_i function.

So, what you do is you set it to a very small very tiny and nonzero value, so that you can do it. So, this loop basically what it does is it determines the value of i at which your inversion.

(Refer Slide Time: 08:13)



```
end do
if (ic1 >= mesh-2) then
  deallocate ( f, vpot, p, y, x )
  stop 'last change of sign too far'
else if (ic1 < 1) then
  deallocate ( f, vpot, p, y, x )
  stop 'no classical turning point?'
end if
!
! f(x) as required by the Numerov algorithm
!
f = 1.0_dp - f
y = 0.0_dp
!
! determination of the wave-function in the first two points
!
hnodes = nodes/2
!
! beware the integer division: 1/2 = 0 !
! if nodes is even, there are 2*hnodes nodes
```

You go from the from the classically allowed region to the classically disallowed region. So, once you do you also check that the. So, that where at which point the how far the point at which this classical inversion is happening. So, how far a is that from your maximum mesh size.

So, if it is too close to the $x_{x \max}$, then the last change of sign is too far and then you do not do; that means, something is wrong with that. So, you come out of the of it, but if

that condition is not satisfied that if things are fine you move into the numerical part of the algorithm that is a this part here.

(Refer Slide Time: 09:07)

```
hnodes = nodes/2
!
! beware the integer division: 1/2 = 0 !
! if nodes is even, there are 2*hnodes nodes
! if nodes is odd, there are 2*hnodes+1 nodes (one is in x=0)
! hnodes is thus the number of nodes in the x>0 semi-axis (x=0 excepted)
!
if (2*hnodes == nodes) then
! even number of nodes: wavefunction is even
y(0) = 1.0_dp
! assume f(-1) = f(1)
y(1) = 0.5_dp*(12.0_dp-10.0_dp*f(0))*y(0)/f(1)
else
! odd number of nodes: wavefunction is odd
y(0) = 0.0_dp
y(1) = dx
end if
!
! outward integration and count number of crossings
!
ncross=0
```

So, first what you do is you check for even and then odd number of nodes here and based on that you set the initial conditions if you have even number of nodes then you set this one as the initial conditions as we discussed. If you have odd number of nodes, then the solution is y is it at 0 is 0 and y 0 you take as x, then you start the outward integration.

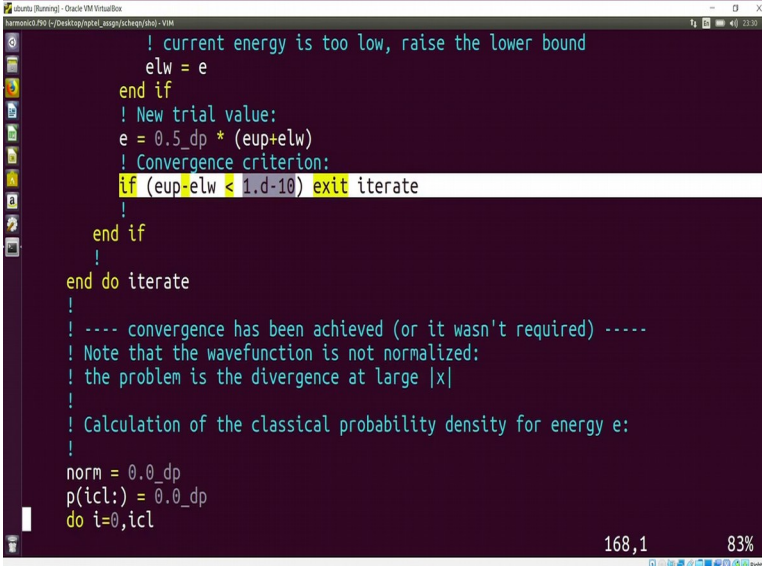
(Refer Slide Time: 09:31)

```
ncross=0
do i =1,mesh-1
y(i+1)=((12.0_dp-10.0_dp*f(i))*y(i)-f(i-1)*y(i-1))/f(i+1)
if ( y(i) /= sign(y(i),y(i+1)) ) ncross=ncross+1
end do
!
print *, kkk, e, ncross, hnodes
!
! if iterating on energy: check number of crossings
!
if (n_iter > 1) then
!
if (ncross > hnodes) then
! Too many crossings: current energy is too high
! lower the upper bound
eup = e
else
! Too few (or correct) number of crossings:
! current energy is too low, raise the lower bound
elw = e
```


So, this is where the outward integration is done and then you also in this part particularly in this region you check whether your wave function changes sign or not.

So, once that is decided then you check whether the sign cross is the number of times the wave function is changing sign whether it is more or equal to or less than the number of nodes you desire if it is greater, then you replace your eup with the new with your guess value of e the upper boundary of your bracketing region. So, you reduce the bracketing region you go to the lower half or if it is less than if the number of crossings is less than h nodes you go to the upper half, then you set up the new trial value of e.

(Refer Slide Time: 10:25)



```
! current energy is too low, raise the lower bound
elw = e
end if
! New trial value:
e = 0.5_dp * (eup+elw)
! Convergence criterion:
if (eup-elw < 1.d-10) exit iterate
!
end if
!
end do iterate
! ---- convergence has been achieved (or it wasn't required) ----
! Note that the wavefunction is not normalized:
! the problem is the divergence at large |x|
!
! Calculation of the classical probability density for energy e:
!
norm = 0.0_dp
p(icl:) = 0.0_dp
do i=0,icl
168,1 83%
```

So, you continue doing this till your difference between eup and elw is greater than equal to is less than equal to minus 10.

(Refer Slide Time: 10:39)

```
end if
!
end do iterate
!
! ---- convergence has been achieved (or it wasn't required) ----
! Note that the wavefunction is not normalized:
! the problem is the divergence at large |x|
!
! Calculation of the classical probability density for energy e:
!
norm = 0.0_dp
p(icl:) = 0.0_dp
do i=0, icl
  arg = (e - x(i)**2/2.0_dp)
  if (arg > 0.0_dp) then
    p(i) = 1.0_dp/sqrt(arg)
  else
    p(i) = 0.0_dp
  end if
  norm = norm + 2.0_dp*dx*p(i)
enddo
```

So, once it done, so, then you try to compute the norm of the function. So, this part of the code computes the norm of your wave function and then also then your wave function is written in this file.

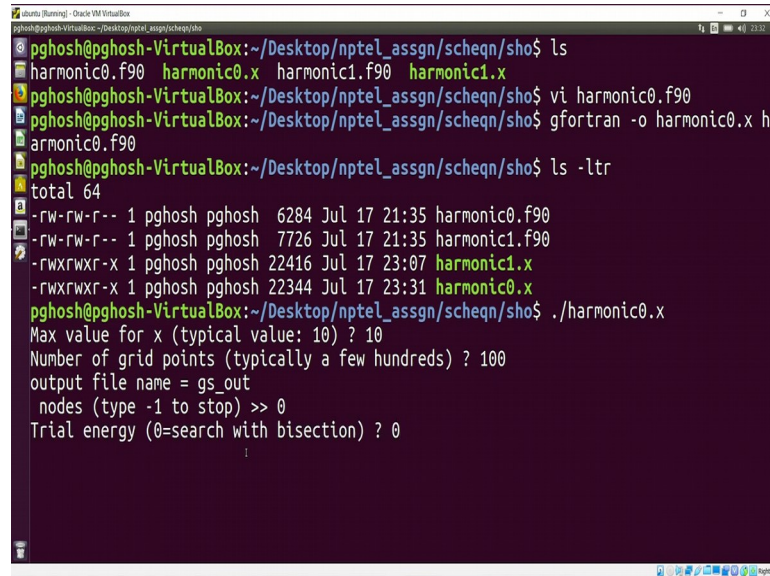
(Refer Slide Time: 10:55)

```
norm = norm - dx*p(0)
! Normalize p(x) so that Int p(x)dx = 1
p(:icl-1) = p(:icl-1)/norm
! lines starting with # ignored by gnuplot
write (7, '# x y(x) y(x)^2 classical p(x) V")')
! x<0 region:
do i=mesh,1,-1
  write (7, '(f7.3,3e16.8,f12.6)') &
    -x(i), (-1)**nodes*y(i), y(i)*y(i), p(i), vpot(i)
enddo
! x>0 region:
do i=0, mesh
  write (7, '(f7.3,3e16.8,f12.6)') &
    x(i), y(i), y(i)*y(i), p(i), vpot(i)
enddo
! two blank lines separating blocks of data, useful for gnuplot plotting
write (7, '/')
end do search_loop
end program harmonic0
```

So, basically what it right is the x value the minus x value and the negative part of the wave function and then. So, this is for the negative part of the wave function when its less than..... for negative values of x not the negative part of the wave function I am

sorry. So, this part is printing out the wave function for negative values of x and this part for the positive values of x and then it comes out.

(Refer Slide Time: 11:23)



```
ubuntu [Reming] - Oracle VM VirtualBox
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho$ ls
harmonic0.f90  harmonic0.x  harmonic1.f90  harmonic1.x
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho$ vi harmonic0.f90
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho$ gfortran -o harmonic0.x h
armonic0.f90
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho$ ls -ltr
total 64
-rw-rw-r-- 1 pghosh pghosh 6284 Jul 17 21:35 harmonic0.f90
-rw-rw-r-- 1 pghosh pghosh 7726 Jul 17 21:35 harmonic1.f90
-rwxrwxr-x 1 pghosh pghosh 22416 Jul 17 23:07 harmonic1.x
-rwxrwxr-x 1 pghosh pghosh 22344 Jul 17 23:31 harmonic0.x
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho$ ./harmonic0.x
Max value for x (typical value: 10) ? 10
Number of grid points (typically a few hundreds) ? 100
output file name = gs_out
nodes (type -1 to stop) >> 0
Trial energy (0=search with bisection) ? 0
```

So, this is the program. So, let us run and see what we get. So, what we do is we compile it gfortran minus sorry minus harmonic 0 dot x. So, if we do ls minus ltr get it. So, let us try to run it. So, this is the; so what I do is I type say I want from 0 to x or minus minus 10 to plus 10, I want 100 grid points for my wave function I give the name of the output file. So, I am interested in finding out the ground state. So, I give the name of the output file as ground state out.

So, in the simple harmonic oscillator in the ground state we know that it has 0 nodes. So, I put 0 I want the code to decide the energy and all, so this is what it is done.

(Refer Slide Time: 12:29)

```
ubuntu [Running] - Oracle VM VirtualBox
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/cho$
19 0.50001144409179688      1      0
20 0.49996376037597656      0      0
21 0.49998760223388672      0      0
22 0.49999952316284180      0      0
23 0.50000548362731934      1      0
24 0.50000250339508057      1      0
25 0.50000101327896118      1      0
26 0.50000026822090149      1      0
27 0.49999989569187164      1      0
28 0.49999970942735672      1      0
29 0.49999961629509926      1      0
30 0.49999956972897053      0      0
31 0.49999959301203489      0      0
32 0.49999960465356708      0      0
33 0.49999961047433317      1      0
34 0.49999960756395012      0      0
35 0.49999960901914164      1      0
36 0.49999960829154588      0      0
37 0.49999960865534376      0      0
38 0.49999960883724270      1      0
39 0.49999960874629323      0      0
nodes (type -1 to stop) >> -1
```

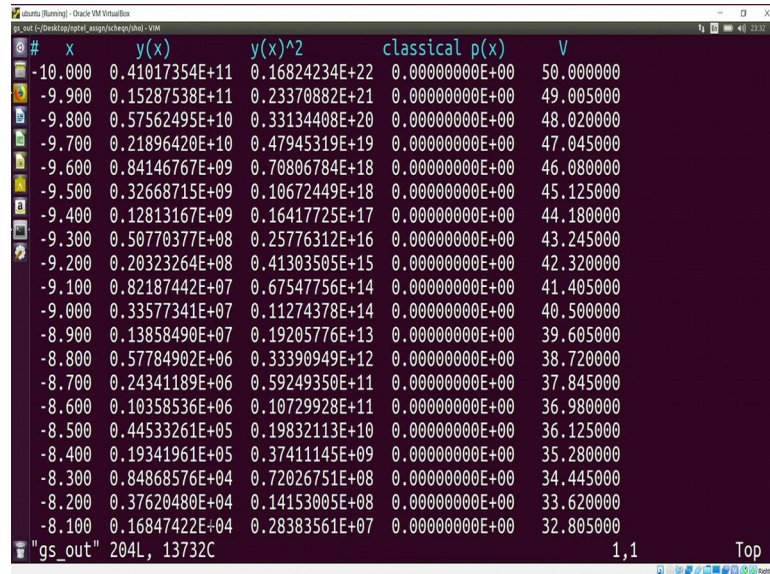
So, you see it converges after 39 iterations after 39 bisection operations it converges and the ground state energy is half we know and it is very close to that. So, let us now plot the wave function and see what we get.

(Refer Slide Time: 12:43)

```
ubuntu [Running] - Oracle VM VirtualBox
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/cho$
38 0.49999960883724270      1      0
39 0.49999960874629323      0      0
nodes (type -1 to stop) >> -1
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/cho$ ls -ltr
total 80
-rw-rw-r-- 1 pghosh pghosh 6284 Jul 17 21:35 harmonic0.f90
-rw-rw-r-- 1 pghosh pghosh 7726 Jul 17 21:35 harmonic1.f90
-rwxrwxr-x 1 pghosh pghosh 22416 Jul 17 23:07 harmonic1.x
-rwxrwxr-x 1 pghosh pghosh 22344 Jul 17 23:31 harmonic0.x
-rw-rw-r-- 1 pghosh pghosh 13732 Jul 17 23:32 gs_out
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/cho$ vi gs_out
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/cho$ xmgrace &
[1] 4886
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/cho$ xmgrace &
[2] 4902
[1] Done          xmgrace
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/cho$ ./harmonic1.x
Max value for x (typical value: 10) > 10
Number of grid points (typically a few hundreds) > 100
output file name > a
nodes (type -1 to stop) > 0
Trial energy (0=search with bisection) >
```

So, if I do ls minus ltr, so this is the file where my wave function is plotted, the data for the wave function is written.

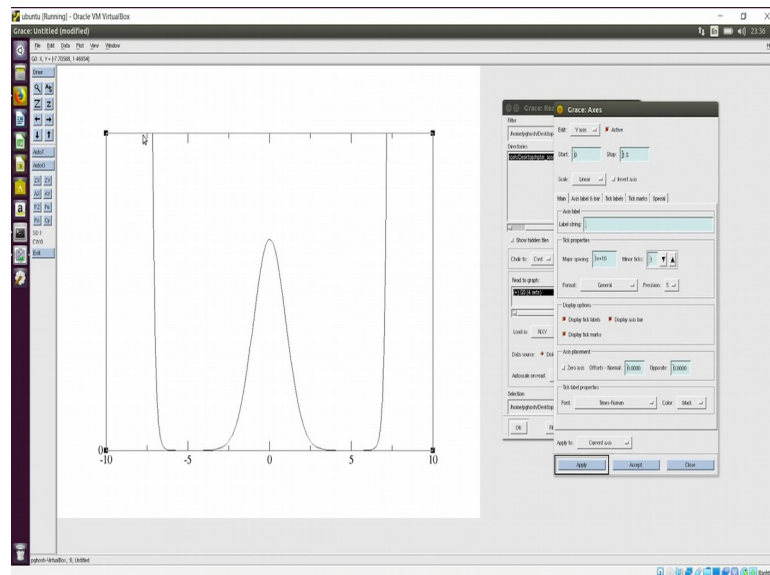
(Refer Slide Time: 12:51)



#	x	y(x)	y(x)^2	classical p(x)	V
-10.000	0.41017354E+11	0.16824234E+22	0.00000000E+00	50.000000	
-9.900	0.15287538E+11	0.23370882E+21	0.00000000E+00	49.005000	
-9.800	0.57562495E+10	0.33134408E+20	0.00000000E+00	48.020000	
-9.700	0.21896420E+10	0.47945319E+19	0.00000000E+00	47.045000	
-9.600	0.84146767E+09	0.70806784E+18	0.00000000E+00	46.080000	
-9.500	0.32668715E+09	0.10672449E+18	0.00000000E+00	45.125000	
-9.400	0.12813167E+09	0.16417725E+17	0.00000000E+00	44.180000	
-9.300	0.50770377E+08	0.25776312E+16	0.00000000E+00	43.245000	
-9.200	0.20323264E+08	0.41303505E+15	0.00000000E+00	42.320000	
-9.100	0.82187442E+07	0.67547756E+14	0.00000000E+00	41.405000	
-9.000	0.33577341E+07	0.11274378E+14	0.00000000E+00	40.500000	
-8.900	0.13858490E+07	0.19205776E+13	0.00000000E+00	39.605000	
-8.800	0.57784902E+06	0.33390949E+12	0.00000000E+00	38.720000	
-8.700	0.24341189E+06	0.59249350E+11	0.00000000E+00	37.845000	
-8.600	0.10358536E+06	0.10729928E+11	0.00000000E+00	36.980000	
-8.500	0.44533261E+05	0.19832113E+10	0.00000000E+00	36.125000	
-8.400	0.19341961E+05	0.37411145E+09	0.00000000E+00	35.280000	
-8.300	0.84868576E+04	0.72026751E+08	0.00000000E+00	34.445000	
-8.200	0.37620480E+04	0.14153005E+08	0.00000000E+00	33.620000	
-8.100	0.16847422E+04	0.28383561E+07	0.00000000E+00	32.805000	

So, it goes from minus 10 to plus 10 and so the first column is my x the second column is my wave function which is y x and then we are not interested as of now for the other three columns. So, we do not worry. So, we use let us see how the wave functions look like we use xm grace.

(Refer Slide Time: 13:13)



So, we go to as the plotting software, so we go. So, basically this is the file which contains the information and xy and this is what we have. So, if you look at it then one second we just need the first graphs not others let me just. So, if you see that at x equals

to minus 10 and plus 10 the wave function completely diverges you would have expected it valued close to 0 while in other case it is gone to a very large value. Now, let us try to zoom in and see how much what it looks like in the classically within the classically allowed region.

So, here we put x and so the change the upper value of y we put it to 1 and we apply just maybe 1.5 yeah. So, you see for the ground state what we expected is a symmetric gaussian type wave function which is centered at x equals to 0 and we get that. So, in the classically allowed regime we see that the wave function we get behaviour of the wave function while in the classically forbidden regime the wave function blows up. So, that highlights the necessity of the asymptotic wave function.

So, if I take into account and do the corrections for the asymptotic term which I am not going to show the code for you I leave it for you to do then what and if I run it for the same wave function if I use the same values say 100 output file name a I give and number of nodes I give 0. Then I do a bisection search see it converges again at 39 after 39 iterations to a value of 0.5 and now let me plot it again so, xmgrace minus nx y a.

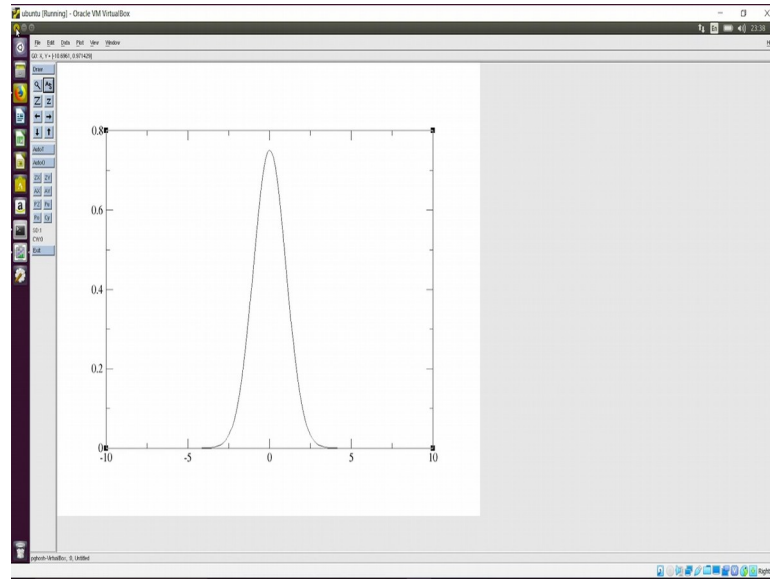
(Refer Slide Time: 15:25)

```

ubuntu [Running] - Oracle VM VirtualBox
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho
24 0.500038266181946 0 -0.00000075
25 0.500039756298065 0 0.00000289
26 0.500039011240005 0 0.00000107
27 0.500038638710976 0 0.00000016
28 0.500038452446461 0 -0.00000029
29 0.500038545578718 0 -0.00000007
30 0.500038592144847 0 0.00000005
31 0.500038568861783 0 -0.00000001
32 0.500038580503315 0 0.00000002
33 0.500038574682549 0 0.00000000
34 0.500038571772166 0 -0.00000000
35 0.500038573227357 0 0.00000000
36 0.500038572499761 0 -0.00000000
37 0.500038572863559 0 0.00000000
38 0.500038572681660 0 -0.00000000
39 0.500038572772610 0 0.00000000
nodes (type -1 to stop) > -1
[2]+ Done xmgrace
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho$ ls
a gs_out harmonic0.f90 harmonic0.x harmonic1.f90 harmonic1.x
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho$
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/sho$ xmgrace -nxy a &

```

(Refer Slide Time: 15:45)



If I plot it, so I have several things here. So, let me just keep the wave function and move the others. So, what you will see that this diverging behaviour which we are getting earlier that has disappeared now. So, that; so through these two pieces of code what we saw is how to numerically find the solutions using numerical algorithm the solutions of the one dimensional Schrödinger equation. And then we also saw how some of the difficulties extra difficulties I would say associated with when you are trying to solve a quantum mechanical problem numerically.