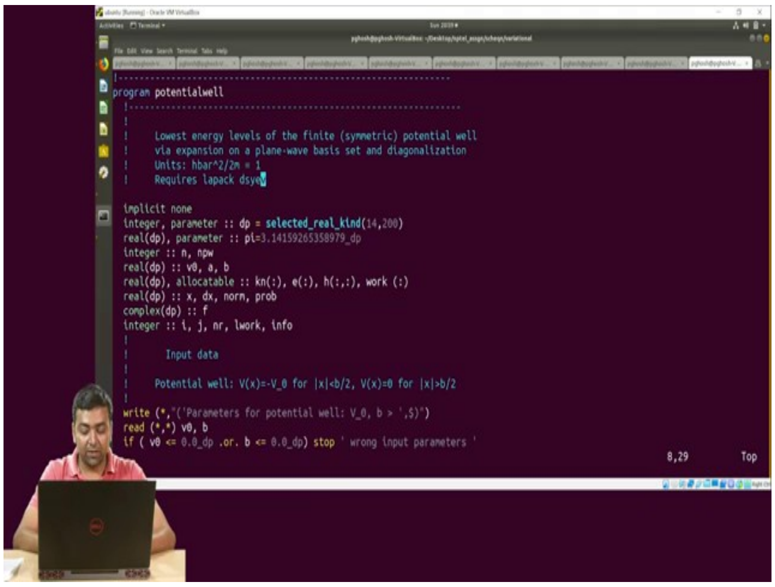
Computational Physics
Dr. Apratim Chatterji
Dr. Prasenjit Ghosh
Department of Physics
Indian Institute of Science Education and Research, Pune

Lecture – 44
Differential Equation for Quantum Mechanical Problems: Variational Principle Part 04

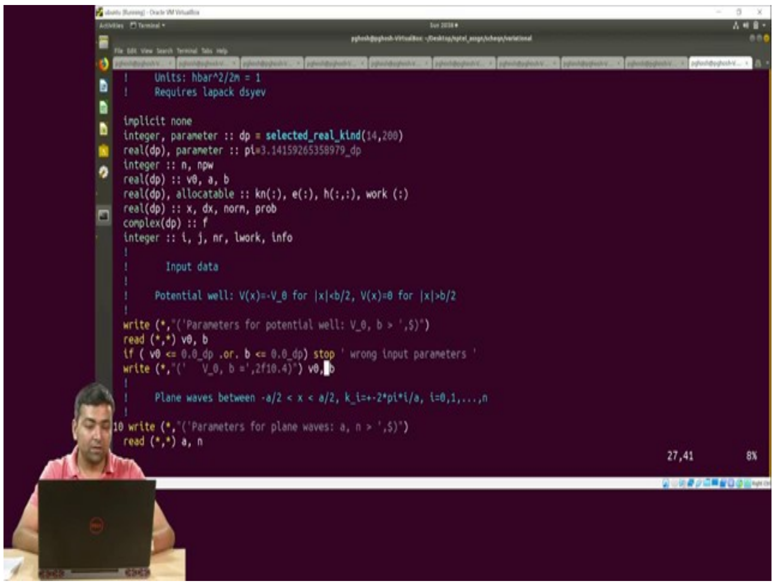
(Refer Slide Time: 00:16)



```
!-------------------------------------------------------------
program potentialwell
  !-------------------------------------------------------------
  !
  !    Lowest energy levels of the finite (symmetric) potential well
  !    via expansion on a plane-wave basis set and diagonalization
  !    Units: hbar^2/2m = 1
  !    Requires lapack dsyev

  implicit none
  integer, parameter :: dp = selected_real_kind(14,200)
  real(dp), parameter :: pi=3.14159265358979_dp
  integer :: n, npw
  real(dp) :: v0, a, b
  real(dp), allocatable :: kn(:), e(:), h(:,:), work (:)
  real(dp) :: x, dx, norm, prob
  complex(dp) :: f
  integer :: i, j, nr, lwork, info
  !
  !      Input data
  !
  !    Potential well: V(x)=-V_0 for |x|<b/2, V(x)=0 for |x|>b/2
  !
  write (*,"('Parameters for potential well: V_0, b > ',$)")
  read (*,*) v0, b
  if ( v0 <= 0.0_dp .or. b <= 0.0_dp) stop ' wrong input parameters '
                                                                 8,29          Top
```

So, this program shows us how to use the Variational Principle to compute or to solve the Schrodinger equation for a given potential well.

(Refer Slide Time: 00:34)



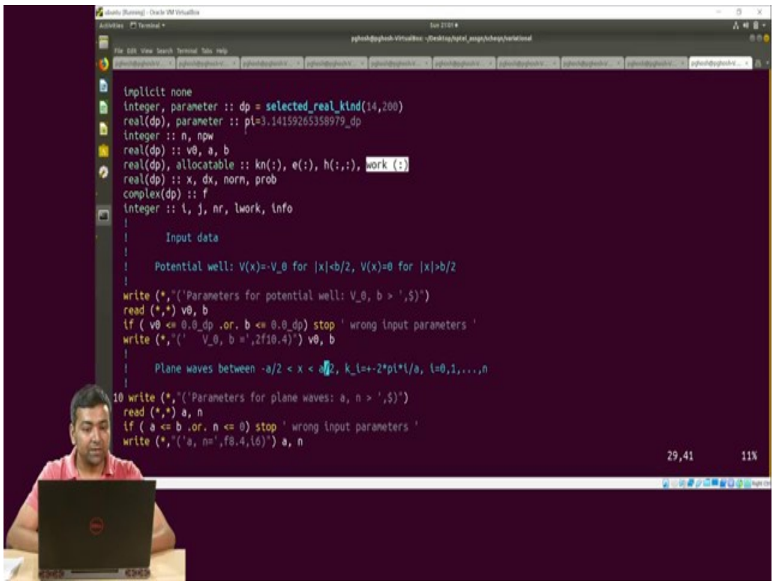
```
!     Units: hbar^2/2m = 1
!     Requires lapack dsyev

implicit none
integer, parameter :: dp = selected_real_kind(14,200)
real(dp), parameter :: pi=3.14159265358979_dp
integer :: n, npw
real(dp) :: v0, a, b
real(dp), allocatable :: kn(:), e(:), h(:,:), work (:)
real(dp) :: x, dx, norm, prob
complex(dp) :: f
integer :: i, j, nr, lwork, info
!
!       Input data
!
!     Potential well: V(x)=-V_0 for |x|<b/2, V(x)=0 for |x|>b/2
!
write (*,"('Parameters for potential well: V_0, b > ',$)")
read (*,*) v0, b
if ( v0 <= 0.0_dp .or. b <= 0.0_dp) stop ' wrong input parameters '
write (*,"('   V_0, b =',2f10.4)") v0, b
!
!     Plane waves between -a/2 < x < a/2, k_i=+-2*pi*i/a, i=0,1,...,n
!
10 write (*,"('Parameters for plane waves: a, n > ',$)")
read (*,*) a, n
                                                        27,41         8%
```

So, remember in the like when we talked about the methodology, so, towards the end we give I give a sort of an assignment to compute this potential well. The Eigen values and Eigen functions of the Schrodinger equation when your potential is given by an attractive square well. So, this highlighted part here. So, this is the definition of my potential. So, what it means is that the total width of the potential is given by b and it is a symmetric potential centered at the origin.

So, from minus b by 2, so, this part what this means is from minus b by 2 to plus b by 2 the potential as a value minus V 0 and for other parts of odd for other values of x the potential has a value 0. So, let us see how we can convert this to a numerical problem to solve the Schrodinger equation. So, what if you remember the methodology which we discussed? So, what we have we need to do is we need to expand our wave functions in our basis and as a basis we are using the plane waves and then we need to construct the Hamiltonian the h matrix and, then we need to diagnose it.

So, for the diagonalization we will be using this particular subroutine called dsyev. We will see what is there very shortly; this is a part of the lapack suit of course. And another thing is this code is written assuming that everything is in atomic units. What it means is that h bar square by twice m is equal to 1.

(Refer Slide Time: 02:19)



```
 implicit none
 integer, parameter :: dp = selected_real_kind(14,200)
 real(dp), parameter :: pi=3.14159265358979_dp
 integer :: n, npw
 real(dp) :: v0, a, b
 real(dp), allocatable :: kn(:), e(:), h(:,:), work (:)
 real(dp) :: x, dx, norm, prob
 complex(dp) :: f
 integer :: i, j, nr, lwork, info
 !
 !       Input data
 !
 !     Potential well: V(x)=-V_0 for |x|<b/2, V(x)=0 for |x|>b/2
 !
 write (*,"('Parameters for potential well: V_0, b > ',$)")
 read (*,*) v0, b
 if ( v0 <= 0.0_dp .or. b <= 0.0_dp) stop ' wrong input parameters '
 write (*,"('   V_0, b =',2f10.4)") v0, b
 !
 !     Plane waves between -a/2 < x < a/2, k_i=+-2*pi*i/a, i=0,1,...,n
 !
10 write (*,"('Parameters for plane waves: a, n > ',$)")
 read (*,*) a, n
 if ( a <= b .or. n <= 0) stop ' wrong input parameters '
 write (*,"('a, n=',f8.4,i6)") a, n
```

So, as we have seen for other fortran codes. So, we start with this implicit none and then this part in the first part we do the variable declarations.

So, now remember this when we talked about in the numerable method so, we talked about the importance of the accuracy of my calculations when doing a calculations for quantum mechanical problems. And as a result of which so, I need to do everything in double precision at least in double precision. So, therefore, I am declaring here through this parameter dp this integer variable that everything is in double precision. Then in many places I will be needing the value of pi. So, I am setting pi to 3.14 this value and then I am declaring it as a parameter and real number.

Then, I need integers. So, where npw is the number of plane waves that go into the calculations. So, remember the number of plane waves so, that depends on twice pi by a into n where n goes from 0 to 0, plus minus 1, plus minus 2 like that 2 plus minus n. So, this n is that number. So, basically say if you give n your input so, this will be my input and where if I give the value of n the npw will be 2 n plus 1 in our calculations. So, if I give say the value of n to be 20. So, the number of plane waves that goes into the expansion of my wave functions that will be 41.

Then, V 0 is the value of the potential this constant the depth of my potential well here, a is the periodicity of my plane wave which I am using and b is the width of the potential. So, this is this same b which is here. Then I have some allocatable arrays here. The first array this kn this towards the k vector corresponding to each plane wave, e towards the eigen value and h is my Hamiltonian matrix, work is array it is a temporary array which would be used by this subroutine which we will be using to norm diagonalize our matrix. So, this is this dsyev.

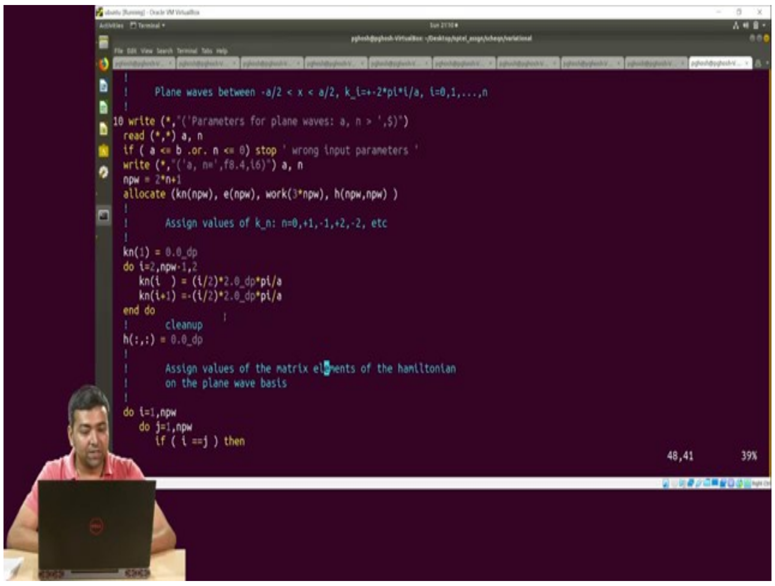
Then, we have some also some real numbers and so, x is just my position in space which corresponds to this value this x here in my potential. You remember when potential is a function of x. So, it is that x, dx is the grid I am using to the size of my grid I am using to discretize this x; norm stores the norm of my wave function because if you remember I mean all the wave functions should be normalized. So, that value of norm should be 1 if my wave function is properly normalized and, then prob this variables towards the probability. So, that means, the psi star psi part.

And, this complex function here if a gives me the wave functions and then I have some integers here so, where lwork and info these two are again some temporary integers which would be used by this subroutine dsyev, ok. So, now, what do we need as input data? So, for my input data we need a couple of things; we need to know the depth of my potential well, the user has to provide that. The width of my potential V 0, the periodicity of my plane wave which is given by a and the n that is which will determine the number of plane waves that is used. So, the same thing is read in this part of the code.

So, what this part says is first it provides a write statement on the standard display where it asks the user to provide the parameters for the potential well which is V 0 and b. So, once the user provides that information this is read in by the code, then you remember by is; so, we are talking here about an attractive potential. So, the way we are defining the potential in the problem so, that is minus V 0 that implies that the value of V 0 should always be positive.

So, here in this line what the score does is that it checks whether the value of v 0 is positive, also my potential has a finite width which is determined by b. So, it checks whether the value of b is greater than whether the value of b is less than 0 or whether it is negative. So, if these two condition any of these two conditions are satisfied the code will stop here printing the message that my input parameters are wrong. So, this is just to sort of give you I mean put in a checks, so that if the user by mistake gives some wrong input parameters the code can immediately identify that.

(Refer Slide Time: 07:31)



```
!
!     Plane waves between -a/2 < x < a/2, k_i=+-2*pi*i/a, i=0,1,...,n
!
10 write (*,"('Parameters for plane waves: a, n > ',$)")
 read (*,*) a, n
 if ( a <= b .or. n <= 0) stop ' wrong input parameters '
 write (*,"('a, n=',f8.4,i6)") a, n
 npw = 2*n+1
 allocate (kn(npw), e(npw), work(3*npw), h(npw,npw) )
 !
 !      Assign values of k_n: n=0,+1,-1,+2,-2, etc
 !
 kn(1) = 0.0_dp
 do i=2,npw-1,2
    kn(i  ) = (i/2)*2.0_dp*pi/a
    kn(i+1) =-(i/2)*2.0_dp*pi/a
 end do
 !      cleanup
 h(:,:) = 0.0_dp
 !
 !      Assign values of the matrix elements of the hamiltonian
 !      on the plane wave basis
 !
 do i=1,npw
    do j=1,npw
       if ( i ==j ) then
                                                              48,41        39%
```

And, once it has read in by the values of v 0 and b and then it has done the checking that everything is fine then for the reference of the user what it does is it prints out the values. So, once; so, this part goes reading in the parameters related to my potential then we also need to read in the parameters for by related to my plain waves. So, we need to; as I mentioned we need two things – one is the periodicity of the plane wave which is given by a and also we need to determine the number of plane waves that goes into my expansion which it depends on the value of n. So, that is what the code reads in here.

So, it reads in so, this line it asks again the user to. So, to give these values of a and n, then in this particular line it reads in that values then again in the like in the case of the parameters for the potential here it checks whether the parameters are wrong. So, one thing we need to know need to be careful about is since we are using plane waves and it has a periodicity of its own which is determined by a. So, the potential for which we are actually doing calculations, so those are a periodic potentials.

So, that is if I move from say a b; x equals to 0 to x equals to plus a or x equals to minus a then what we will have is we will have the same image or the replica of the potential, but the actual potential which we are solving is a it is a very localized and the just a single potential well.

So, what it means is that, my a value should be at least greater than b and in reality the value of a should be large enough. So, that the periodic images of my potential do not interact with each other also in the number of this value of n this should also be always positive and greater than 0 because the minimum number of plane waves you need is at least one plane wave to expand your wave function.

And, then once it is satisfied it has read in these values of a and n in it is satisfied that it is all the where the users have given the correct values it then prints out the values which it is reading for the user to check whether it has read in the values correctly. So, once that is done then it sets the number of plane waves. So, as I mentioned before the number of plane waves is twice n plus 1. So, once the number of plane waves is set then we can start allocating the arrays dimensions corresponding to this allocatable arrays which I have in the declaration.

So, the first array is this kn as I said that stores in the k vector corresponding to each plane wave and it is dimension should be their number of plane wave. Similarly, the so, the dimension of my matrix which is given by h here the Hamiltonian matrix that depends on the number of plane waves that goes into the calculations. So, my matrix will be npw clause cross npw matrix and that is why it has been assigned this value and it is a 2-dimensional matrix then the energy eigen values are stored in this array e and the number of energy eigen values will again be the same as the number of plane waves.

And, then work; in work we it is again a temporary array which is used by the subroutine for diagonalization where it creates this npw; it is 3 times the npw the dimension of my matrix which I need to diagonalize. So, once I have done I am done with allocating these areas so, what I do now is I start assigning the values of the array. So, to start off with I need to first define the k vectors corresponding to that each plane wave and once I have done that then I need to construct my h matrix.

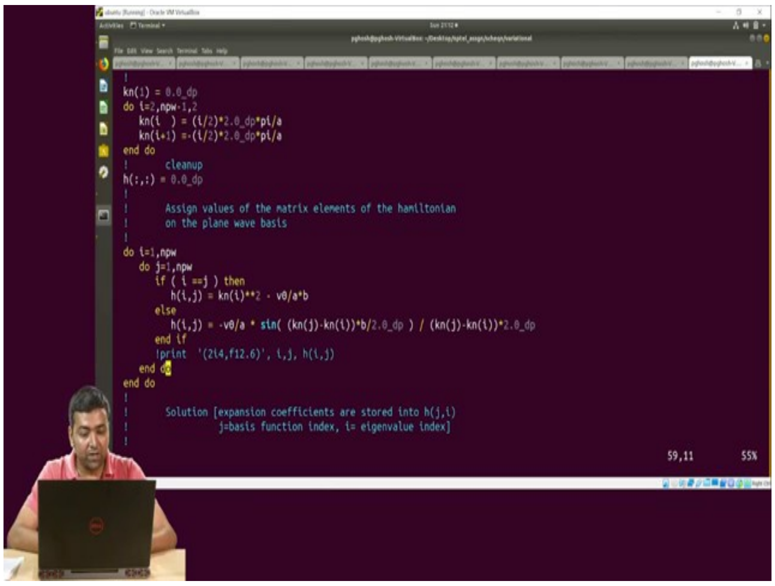
So, this part of the code what it does is it basically a assigns the values of kn. So, now remember we had discussed this that kn can take values of this form I mean the n values corresponds to 0, plus minus 1, plus minus 2 and so on and so forth. So, this is of the form of e to the power I k dot r. So, here the kn 1 here we have as the k vector we have assigned as 0 here, the first element of this array as 0 and then what we have done is in this using this do loop we are assigning the other elements.

So, by k vector is basically what I am doing is for the positive one so, my I goes from 2 to n number of plane waves minus 1 and I am changing it the do loop is in increased by 2 at each iteration. The reason I start from 2 is because kn 1 has already been assigned here. So, then what I do here is in the first one so, for example, if my i equals to 2 so, this will be k 2. So, this will be this value will be 1 I by 2 will be 1 and then twice pi by a. So, that will be the value of k.

And, then since in the next line I have i already i plus 1 so, and I am still at i equals to 2 so, this will be kn 3 and kn 3 corresponds to minus 1 and that is what I am doing. So, i equals to 2 will give me minus 1 and then twice pi by a. In this fashion so, when I go to the next value so, when I go to the next value I remember since the increment is by 2. So, the next value of I will be 4 and so, in that way I can build the all the other k vectors and build up this complete array.

Now, once this is done, so, then I start constructing my h matrix.

(Refer Slide Time: 13:39)



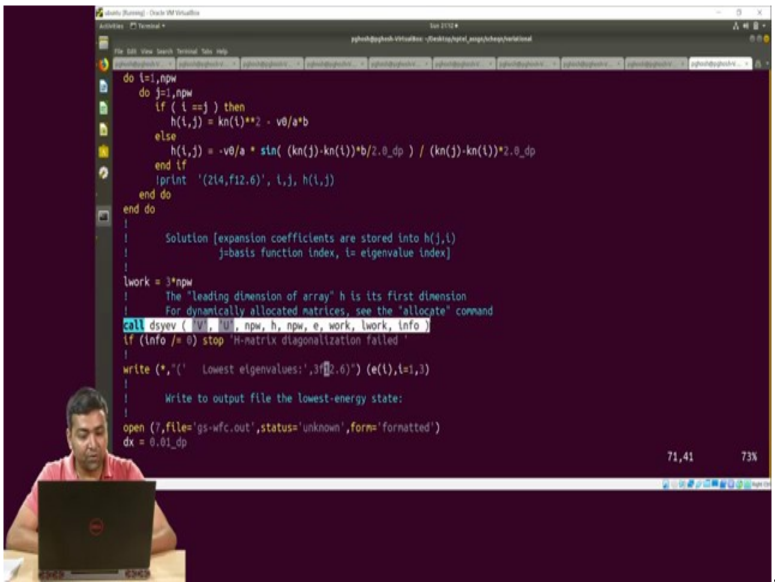
```
!
kn(1) = 0.0_dp
do i=2,npw-1,2
   kn(i  ) = (i/2)*2.0_dp*pi/a
   kn(i+1) =-(i/2)*2.0_dp*pi/a
end do
!       cleanup
h(:,:) = 0.0_dp
!
!       Assign values of the matrix elements of the hamiltonian
!       on the plane wave basis
!
do i=1,npw
   do j=1,npw
      if ( i ==j ) then
         h(i,j) = kn(i)**2 - v0/a*b
      else
         h(i,j) = -v0/a * sin( (kn(j)-kn(i))*b/2.0_dp ) / (kn(j)-kn(i))*2.0_dp
      end if
      !print  '(2i4,f12.6)', i,j, h(i,j)
   end do
end do
!
!       Solution [expansion coefficients are stored into h(j,i)
!                 j=basis function index, i= eigenvalue index]
!
                                                              59,11         55%
```

So, do before starting to construct the h matrix what I do is I first initialize my matrix to 0 and then I use this do loop, this do loop to construct the same thing. So, the do loop the size is the size of the matrix goes from is it is an npw cross npw matrix that is the number of plane waves; npw is the number of plane waves. So, the do loop goes from i equals to 1 to npw and the second do loop also goes from j equals to npw.

Now, if we remember, so, we had two parts in Hamiltonian – the kinetic energy part and the potential energy part. So, as we saw in the lectures that the kinetic energy part or the matrix corresponding to the kinetic energy is always a diagonal matrix and the matrix corresponding to the potential energy this can have both diagonal and off diagonal terms while the off diagonal terms are is independent of the value of k, the diagonal terms are dependent on the value of k. So, that is why the matrix is the h matrix is constructed in a fashion in the following fashion.

So, first I check whether I am doing the calculations for the diagonal elements. So, what that means, is if the I am doing the calculation for the diagonal element the value of i and j will be same. So, if the value of i and j will be same then I assign the those matrix elements with the values of the kinetic energy which is k square in my case and then also the diagonal component of my potential. And if it is off diagonal element I do not have the kinetic energy and I assign it only the values of the corresponding to the off diagonal element of my potential.

(Refer Slide Time: 15:34)

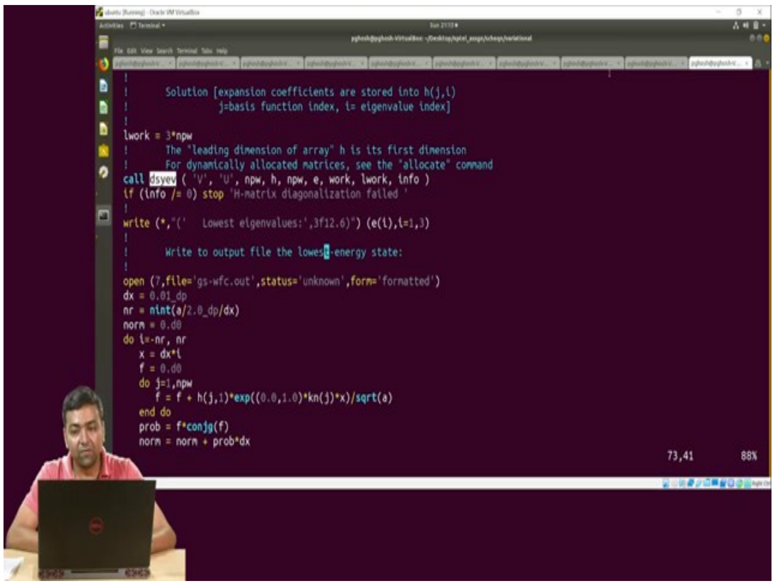


```
do i=1,npw
   do j=1,npw
      if ( i ==j ) then
         h(i,j) = kn(i)**2 - v0/a*b
      else
         h(i,j) = -v0/a * sin( (kn(j)-kn(i))*b/2.0_dp ) / (kn(j)-kn(i))*2.0_dp
      end if
      !print  '(2i4,f12.6)', i,j, h(i,j)
   end do
end do
!
!       Solution [expansion coefficients are stored into h(j,i)
!                 j=basis function index, i= eigenvalue index]
!
lwork = 3*npw
!       The "leading dimension of array" h is its first dimension
!       For dynamically allocated matrices, see the "allocate" command
call dsyev ( 'V', 'U', npw, h, npw, e, work, lwork, info )
if (info /= 0) stop 'H-matrix diagonalization failed '
!
write (*,"('   Lowest eigenvalues:',3f12.6)") (e(i),i=1,3)
!
!       Write to output file the lowest-energy state:
!
open (7,file='gs-wfc.out',status='unknown',form='formatted')
dx = 0.01_dp
                                                              71,41         73%
```

.

So, once it is done I am assigning with the matrix then what I do is I call this subroutine dsyev and provide this following informations. So, what I provide to the subroutine is the number of plane waves my a h matrix and this l work matrix; the subroutine what it returns to me is this energy eigen values which is stored in this e matrix.

(Refer Slide Time: 15:59)

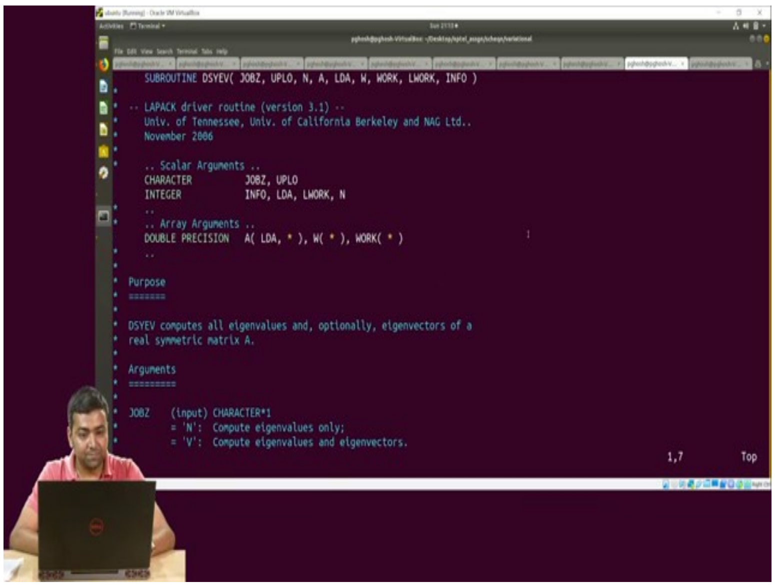


```
!
!       Solution [expansion coefficients are stored into h(j,i)
!                 j=basis function index, i= eigenvalue index]
!
lwork = 3*npw
!       The "leading dimension of array" h is its first dimension
!       For dynamically allocated matrices, see the "allocate" command
call dsyev ( 'V', 'U', npw, h, npw, e, work, lwork, info )
if (info /= 0) stop 'H-matrix diagonalization failed '
!
write (*,"('   Lowest eigenvalues:',3f12.6)") (e(i),i=1,3)
!
!       Write to output file the lowest-energy state:
!
open (7,file='gs-wfc.out',status='unknown',form='formatted')
dx = 0.01_dp
nr = nint(a/2.0_dp/dx)
norm = 0.d0
do i=-nr, nr
   x = dx*i
   f = 0.d0
   do j=1,npw
      f = f + h(j,1)*exp((0.0,1.0)*kn(j)*x)/sqrt(a)
   end do
   prob = f*conjg(f)
   norm = norm + prob*dx
                                                              73,41        88%
```

And, also the energy eigen vectors which it stores in the h matrix again. So, h is a type of an intent out type of variable.

So, we will very soon see what these means what this different the attributes what this different inputs one needs to provide means ok, but before I think before we go to the remaining part of the code let us just take a look at this particular subroutine.

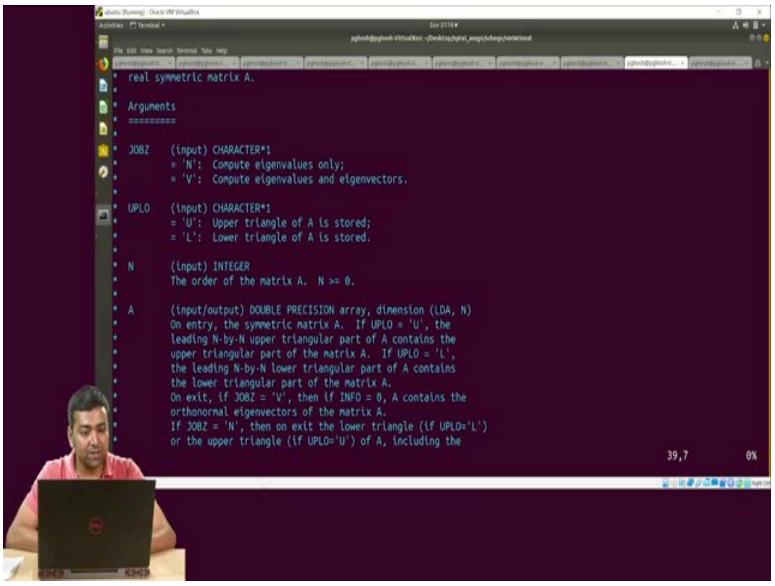
(Refer Slide Time: 16:27)



```
      SUBROUTINE DSYEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO )
*
*  -- LAPACK driver routine (version 3.1) --
*     Univ. of Tennessee, Univ. of California Berkeley and NAG Ltd..
*     November 2006
*
*     .. Scalar Arguments ..
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDA, LWORK, N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), W( * ), WORK( * )
*     ..
*
*  Purpose
*  =======
*
*  DSYEV computes all eigenvalues and, optionally, eigenvectors of a
*  real symmetric matrix A.
*
*  Arguments
*  =========
*
*  JOBZ    (input) CHARACTER*1
*          = 'N':  Compute eigenvalues only;
*          = 'V':  Compute eigenvalues and eigenvectors.
                                                              1,7          Top
```

So, if you open the subroutine you will find something like this. So, a it contains all the information and so, the name of the subroutine is this one and these are the arguments of the subroutine.

(Refer Slide Time: 16:44)



```
*  real symmetric matrix A.
*
*  Arguments
*  =========
*
*  JOBZ    (input) CHARACTER*1
*          = 'N':  Compute eigenvalues only;
*          = 'V':  Compute eigenvalues and eigenvectors.
*
*  UPLO    (input) CHARACTER*1
*          = 'U':  Upper triangle of A is stored;
*          = 'L':  Lower triangle of A is stored.
*
*  N       (input) INTEGER
*          The order of the matrix A.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA, N)
*          On entry, the symmetric matrix A.  If UPLO = 'U', the
*          leading N-by-N upper triangular part of A contains the
*          upper triangular part of the matrix A.  If UPLO = 'L',
*          the leading N-by-N lower triangular part of A contains
*          the lower triangular part of the matrix A.
*          On exit, if JOBZ = 'V', then if INFO = 0, A contains the
*          orthonormal eigenvectors of the matrix A.
*          If JOBZ = 'N', then on exit the lower triangle (if UPLO='L')
*          or the upper triangle (if UPLO='U') of A, including the
                                                          39,7          0%
```

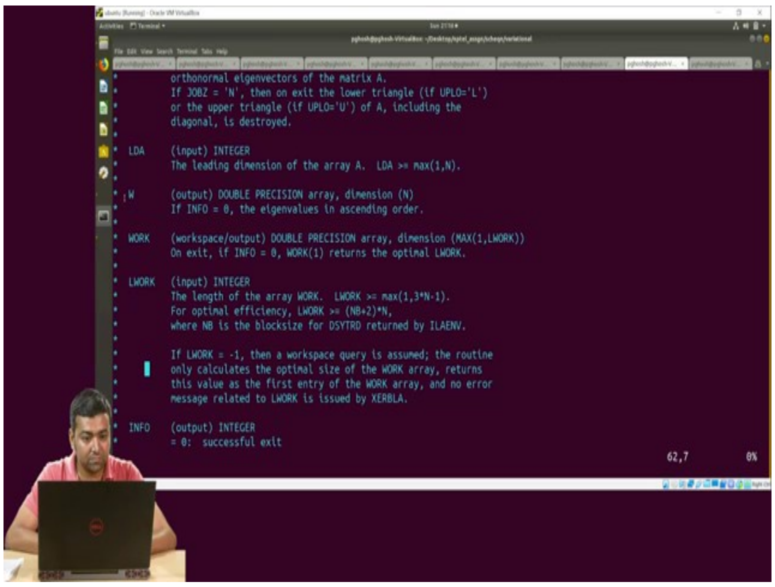
And, if you go down it also tells what is the purpose of the subroutine and then what the different arguments means. So, the first argument of this subroutine that is defined by JOBZ, which when we are calling it in the program here in my program this particular argument here, so, this is input or good type of argument or JOBZ is an input variable which has of character and it can character type, and it can take two values here. So, one it can I tell the subroutine to either compute the eigen values only or it can tell the subroutine to compute the are both the eigen values and the eigenvectors.

So, remember the eigenvectors will be; the information the eigen vectors contains, in our case when we are diagonalizing my hamiltonian matrix is the coefficients of which we have used for our expansion of the wave function in the plane wave basis. Since we are not only interested in the eigen values, but we are also interested in how the wave function looks like. So, therefore, in the main program we have chose the value to be V here because we want the code or the subroutine to not only provide us with diagonalize the matrix and not only provide us with the eigen values but also with the eigenvectors.

Then, this is an int again an input which we need to provide and that you can tells us which part of the matrix which we are diagonalizing here this is denoted by a is stored whether it is the upper triangle or the lower triangle is stored and the reason the this is provided to the code is because this matrix is a symmetry; it is a symmetric matrix and when doing the diagonalization the subroutine exploits the symmetry of the matrix.

Then N here is an again an integer which is of input type which contains the information about the order of the matrix and which is given by this variable npw here, which is same as the number of plane waves. Then A contains the matrix which we need to diagonalize which here in my main program here if you look at it is the h matrix.

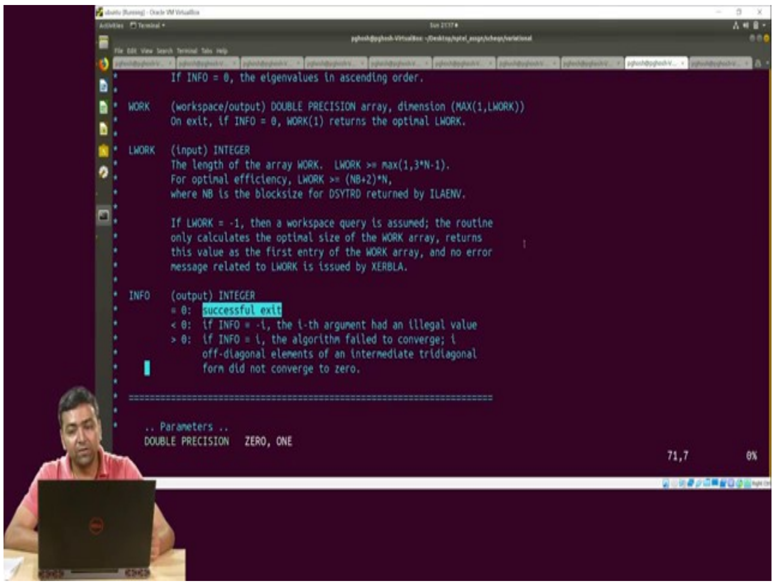
(Refer Slide Time: 19:07)



```
*          orthonormal eigenvectors of the matrix A.
*          If JOBZ = 'N', then on exit the lower triangle (if UPLO='L')
*          or the upper triangle (if UPLO='U') of A, including the
*          diagonal, is destroyed.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,N).
*
*  W       (output) DOUBLE PRECISION array, dimension (N)
*          If INFO = 0, the eigenvalues in ascending order.
*
*  WORK    (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
*          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
*
*  LWORK   (input) INTEGER
*          The length of the array WORK.  LWORK >= max(1,3*N-1).
*          For optimal efficiency, LWORK >= (NB+2)*N,
*          where NB is the blocksize for DSYTRD returned by ILAENV.
*
*          If LWORK = -1, then a workspace query is assumed; the routine
*          only calculates the optimal size of the WORK array, returns
*          this value as the first entry of the WORK array, and no error
*          message related to LWORK is issued by XERBLA.
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
                                                              62,7          0%
```

Then, there are some other internal parameters. So, basically it you need to provide the information about the leading dimension of the array which in our case is same as the order of the array which is say npw here. Then, the next variable here is this W which is an output type which corresponds to this array in my program the array e, where the eigen value values will be stored and the subroutine will return this to the main program that is why it is an output type.

(Refer Slide Time: 19:44)



```
*          If INFO = 0, the eigenvalues in ascending order.
*
*  WORK    (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
*          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
*
*  LWORK   (input) INTEGER
*          The length of the array WORK.  LWORK >= max(1,3*N-1).
*          For optimal efficiency, LWORK >= (NB+2)*N,
*          where NB is the blocksize for DSYTRD returned by ILAENV.
*
*          If LWORK = -1, then a workspace query is assumed; the routine
*          only calculates the optimal size of the WORK array, returns
*          this value as the first entry of the WORK array, and no error
*          message related to LWORK is issued by XERBLA.
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*          > 0:  if INFO = i, the algorithm failed to converge; i
*                off-diagonal elements of an intermediate tridiagonal
*                form did not converge to zero.
*
*  =====================================================================
*
     .. Parameters ..
     DOUBLE PRECISION   ZERO, ONE
                                                              71,7          0%
```

Then LWORK is some temporary array and then INFO is again an integer and of an output type. So, through this INFO the code conveys the following message. If the INFO the value of the INFO is equal to 0 then what it means is that it has successfully diagonalized the matrix, if it is less than 0 and that is if it returns a negative number. And so, what it means is that the say suppose it returns minus 5 so, what it mean is that the fifth argument had an illegal value.

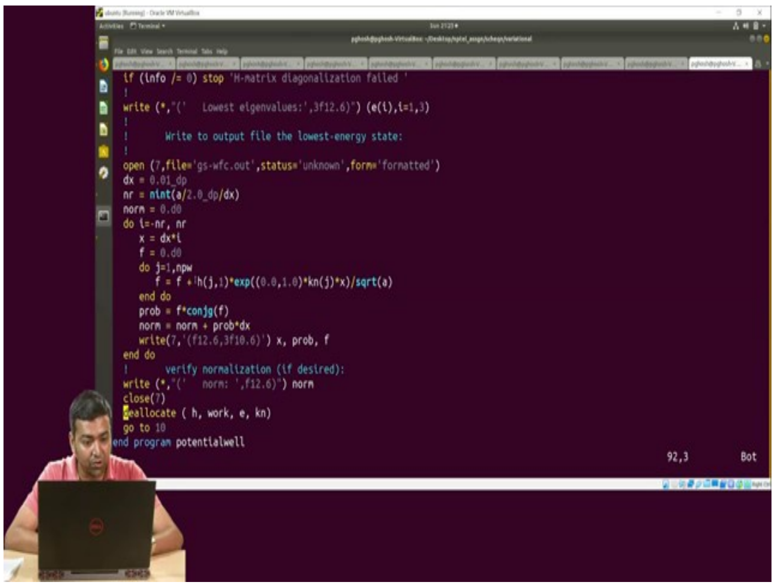
Similarly, if it is greater than 0, so, what it means is that the algorithm failed to converge the off diagonal element. So, remember in this we do not explicitly diagonalize the full matrix. So, the matrix diagonalization is done in a iterative way; though in this part of the course we will not go into the details of how this subroutine works and how the matrix diagonalization is done but this is just for your knowledge.

So, once in; so, now, back to my main program, once I have called this subroutine it has done its job then and it has returned the values that is the eigenvectors which is stored in h and the eigen values e and the status that is stored in info then I check then the code what the code does it is check first what the status is. So, if it is not info is not equal to 0, then the code stops here saying that there is some issue with the matrix diagonalization and it has failed.

So, if this condition is not satisfied the code goes to the next line and it prints out the three lowest eigen value. Here I printed out only the three lowest eigen value we can print out even more if you want and then what it does is it writes the output file writes to the output file the lowest energy eigen state. So, remember the matrix the array here, the array A which corresponds to a in my main code at this array variable h into in this subroutine this is an input output type of array.

So, what it means is that through A, I supply the Hamiltonian matrix which I or the matrix which I want to diagonalize and then once the diagonalization is done the code returns the eigenvectors through this matrix A. So, each column of this matrix A on returning to the main program gives me 1 eigenvector. So, for example, the first column corresponds to the eigenvector corresponding to the first eigen value.

(Refer Slide Time: 22:26)



```
if (info /= 0) stop 'H-matrix diagonalization failed '
!
write (*,"('   Lowest eigenvalues:',3f12.6)") (e(i),i=1,3)
!
!       Write to output file the lowest-energy state:
!
open (7,file='gs-wfc.out',status='unknown',form='formatted')
dx = 0.01_dp
nr = nint(a/2.0_dp/dx)
norm = 0.d0
do i=-nr, nr
   x = dx*i
   f = 0.d0
   do j=1,npw
      f = f + h(j,1)*exp((0.0,1.0)*kn(j)*x)/sqrt(a)
   end do
   prob = f*conjg(f)
   norm = norm + prob*dx
   write(7,'(f12.6,3f10.6)') x, prob, f
end do
!       verify normalization (if desired):
write (*,"('   norm: ',f12.6)") norm
close(7)
deallocate ( h, work, e, kn)
go to 10
end program potentialwell
                                                    92,3          Bot
```

And, since the eigen values have written in an increasing order of value so, what in my case in our case here when I come back to my main program, what it means is that; the first column of this particular array h which is returned by this subroutine contains the eigenvector corresponding to my ground state. And so, what I am doing is now what does it mean that it contains the eigenvectors of my ground state. So, what it means for my case or for our case here is that it contains the, if the coefficients of my plane wave expansion of my wave function. So, using this coefficients what I want to do is I want to build my wave function the ground state wave function here.

So, this is how it is done in this code. So, I first open a file with this particular file name ‘gs hyphen wfc dot out’ with this particular file name where this wave function is will be stored in this file and this I am giving a name unit name 7. Now, what I do is I so, my x here the value of x; so, my wave function is a function of x. So, I need to discretize this x here. So, the I take dx to be 0.01 and then what I do is so, from minus a by 2 to plus a by 2 remember a is the periodicity of my plane wave.

So, I find out how many grid points will be there if each size of each grid is determined by this value of dx and that is stored in the in this nr. So, this I do just for the half of the part. So, that is why it is a by 2 then by symmetry. So, this the other half will also have the same number of grids then norm is the variable where I stored the norm of the function and I compute and store the norm of the function. So, I initialize that first to 0 and then I start building my wave functions here so, using this do loop.

So, since I want to build from minus a to plus a. So, that is why it goes from the counter of the do loop goes from minus nr to plus nr. Once then the next step what it does is it computes the value of x which is the product of this dx into i the value of the counter of the do loop, then it sets the f where it stores the wave functions to 0 and then in this inner do loop which I have here, it builds up the wave functions.

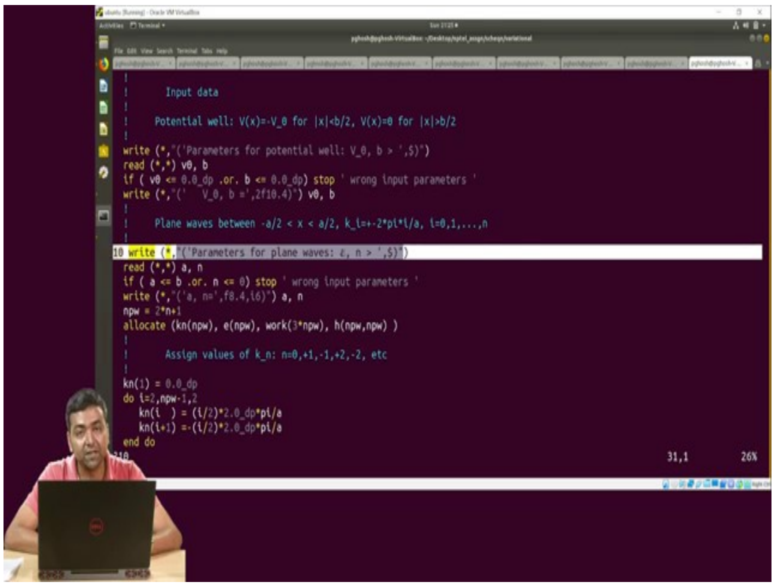
So, if you remember my wave function is like c 1 psi 1 plus c 2 psi 2 c 3 psi 3; where c 1 it is the coefficient and psi 1 is a given particular plane waves say it suppose it corresponds to k equals to 0, ok. Then, c 2 will be the another coefficient corresponding to 2 k corresponds to the value of n equals to plus 1 then c 3 will be 4 minus 1 and so on and so forth and so, that is what it is done. So, this a with the coefficients of this expansion are read from this matrix h whose from the first column of this matrix. So, that is why the j is the number of rows here and that is equal to the number of plane waves because we have same number of coefficients as the number of plane waves.

And, then so, this coefficient you need to multiply it with the plane wave here. So, the plane wave is my exponential e to the power i k dot r and in this case it is k x divided by routine. So, this is what is the plane wave is. So, this is for a particular plane wave and then to construct the full wave function what we need to do is we need to add up this for all the plane waves. So, that is why we have a do loop here. So, once this is done then for that particular value of x I know what is the value of my wave function. And once I know that then I can compute the probability at that particular value of x which corresponds to my real part of the wave function into the complex conjugate of the wave function.

Now, once I know the probability so, this probability then if I multiply by x and I integrate it that is I add up over all the values of x that is from minus that is for the complete do loop here then the value that I get at the end of the day and it is stored in norm is basically my norm of the wave function. And, then in this line what I do is I write down my value of x, the corresponding value of probability at that value of x and the corresponding value of the wave function.

And, then since I am interested to check whether it is properly normalized or not, I also write down the where I print out the value of the norm. So, once this is done I close this file and then I de-allocate the array and then I have a here our go to statement because I would like to play around with different values.

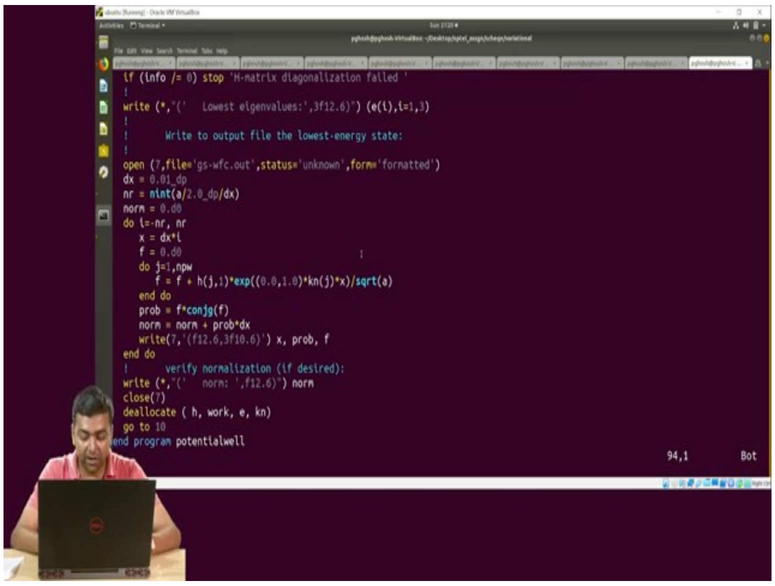
(Refer Slide Time: 27:52)



```
!
!       Input data
!
!     Potential well: V(x)=-V_0 for |x|<b/2, V(x)=0 for |x|>b/2
!
write (*,"('Parameters for potential well: V_0, b > ',$)")
read (*,*) v0, b
if ( v0 <= 0.0_dp .or. b <= 0.0_dp) stop ' wrong input parameters '
write (*,"('   V_0, b =',2f10.4)") v0, b
!
!     Plane waves between -a/2 < x < a/2, k_i=+-2*pi*i/a, i=0,1,...,n
!
10 write (*,"('Parameters for plane waves: a, n > ',$)")
read (*,*) a, n
if ( a <= b .or. n <= 0) stop ' wrong input parameters '
write (*,"('a, n=',f8.4,i6)") a, n
npw = 2*n+1
allocate (kn(npw), e(npw), work(3*npw), h(npw,npw) )
!
!       Assign values of k_n: n=0,+1,-1,+2,-2, etc
!
kn(1) = 0.0_dp
do i=2,npw-1,2
   kn(i  ) = (i/2)*2.0_dp*pi/a
   kn(i+1) =-(i/2)*2.0_dp*pi/a
end do
                                                          31,1          26%
```

So, go to here is basically brings you to this particular part of the code this one. So, you can rerun the code again by changing the parameters of related to the plane waves that is the periodicity of my plane waves and the number of plane waves.

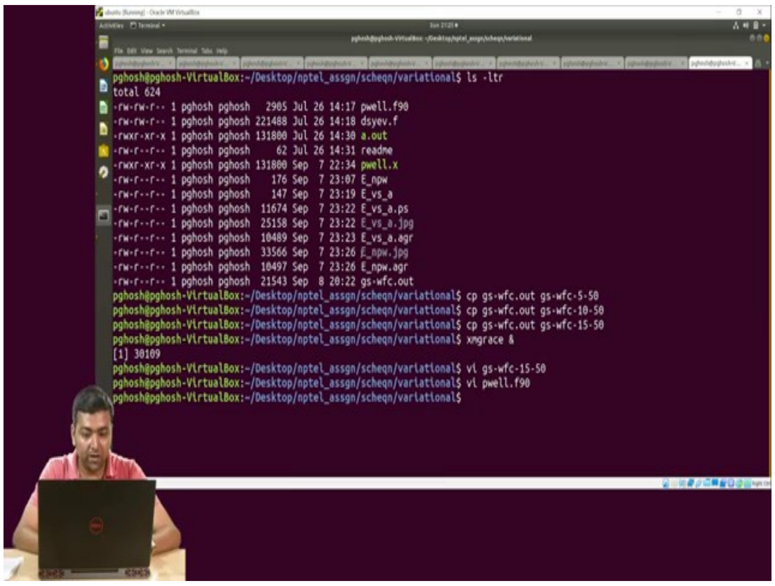
(Refer Slide Time: 28:08)



```
if (info /= 0) stop 'H-matrix diagonalization failed '
!
write (*,"('   Lowest eigenvalues:',3f12.6)") (e(i),i=1,3)
!
!      Write to output file the lowest-energy state:
!
open (7,file='gs-wfc.out',status='unknown',form='formatted')
dx = 0.01_dp
nr = nint(a/2.0_dp/dx)
norm = 0.d0
do i=-nr, nr
   x = dx*i
   f = 0.d0
   do j=1,npw
      f = f + h(j,1)*exp((0.0,1.0)*kn(j)*x)/sqrt(a)
   end do
   prob = f*conjg(f)
   norm = norm + prob*dx
   write(7,'(f12.6,3f10.6)') x, prob, f
end do
!      verify normalization (if desired):
write (*,"('   norm: ',f12.6)") norm
close(7)
deallocate ( h, work, e, kn)
go to 10
end program potentialwell
                                                          94,1          Bot
```

So, this is the code and then let us see how one can run it.

(Refer Slide Time: 28:12)



```
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/variational$ ls -ltr
total 624
-rw-rw-r-- 1 pghosh pghosh   2905 Jul 26 14:17 pwell.f90
-rw-rw-r-- 1 pghosh pghosh 221488 Jul 26 14:18 dsyev.f
-rwxr-xr-x 1 pghosh pghosh 131800 Jul 26 14:30 a.out
-rw-r--r-- 1 pghosh pghosh     62 Jul 26 14:31 readme
-rwxr-xr-x 1 pghosh pghosh 131800 Sep  7 22:34 pwell.x
-rw-r--r-- 1 pghosh pghosh    176 Sep  7 23:07 E_npw
-rw-r--r-- 1 pghosh pghosh    147 Sep  7 23:19 E_vs_a
-rw-r--r-- 1 pghosh pghosh  11674 Sep  7 23:22 E_vs_a.ps
-rw-r--r-- 1 pghosh pghosh  25158 Sep  7 23:22 E_vs_a.jpg
-rw-r--r-- 1 pghosh pghosh  10489 Sep  7 23:23 E_vs_a.agr
-rw-r--r-- 1 pghosh pghosh  33566 Sep  7 23:26 E_npw.jpg
-rw-r--r-- 1 pghosh pghosh  10497 Sep  7 23:26 E_npw.agr
-rw-r--r-- 1 pghosh pghosh  21543 Sep  8 20:22 gs-wfc.out
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/variational$ cp gs-wfc.out gs-wfc-5-50
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/variational$ cp gs-wfc.out gs-wfc-10-50
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/variational$ cp gs-wfc.out gs-wfc-15-50
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/variational$ xmgrace &
[1] 30109
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/variational$ vi gs-wfc-15-50
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/variational$ vi pwell.f90
pghosh@pghosh-VirtualBox:~/Desktop/nptel_assgn/scheqn/variational$
```