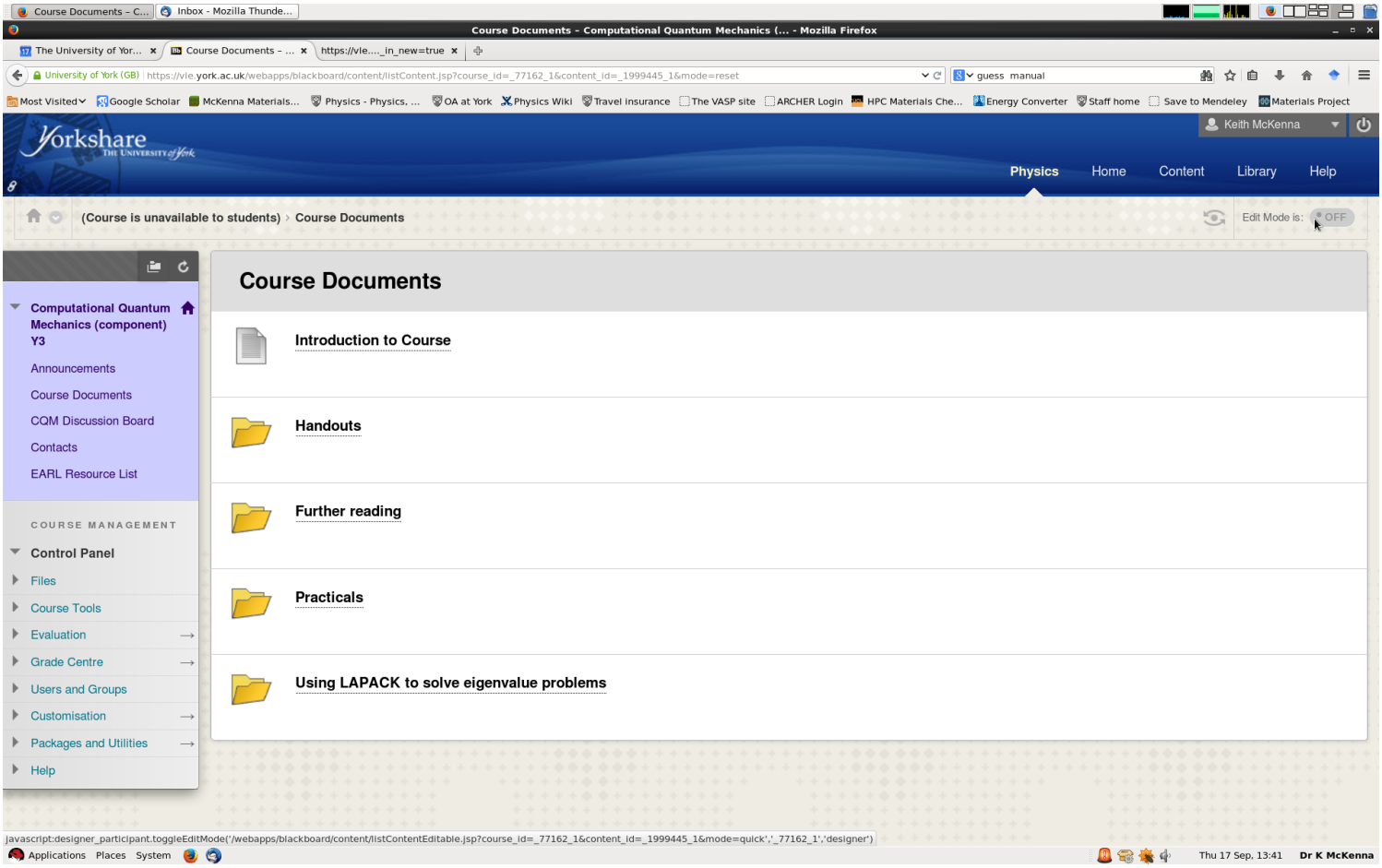

Course Documents - Computational Quantum Mechanics (... - Mozilla Firefox
Yorkshare
THE UNIVERSITY of York
Keith McKenna
Physics
Home
Content
Library
Help
(Course is unavailable to students) › Course Documents
Edit Mode is: OFF
Computational Quantum Mechanics (component) Y3
Announcements
Course Documents
CQM Discussion Board
Contacts
EARL Resource List
COURSE MANAGEMENT
Control Panel
Files
Course Tools
Evaluation
Grade Centre
Users and Groups
Customisation
Packages and Utilities
Help
Course Documents
Introduction to Course
Handouts
Further reading
Practicals
Using LAPACK to solve eigenvalue problems
Applications Places System
Thu 17 Sep, 13:41
Dr K McKenna

1 Real example of one-dimensional square well

K. Sagisaka *et al*, Appl. Phys. Lett. **88** 203118 (2006)

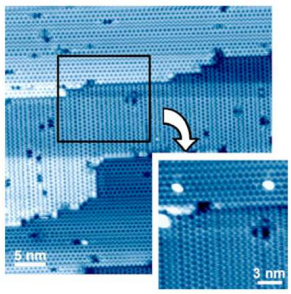


FIG. 2. (Color online) STM images of Si(100) surface before and after tungsten deposition. Depositions were done on single dimer row of upper terrace in boxed region. V_s=+0.6 V and I_t=0.5 nA.

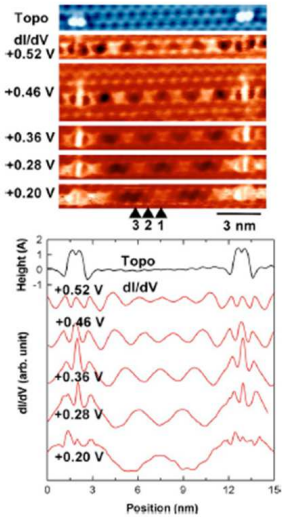


FIG. 3. (Color online) (Top) Topography and selected dI/dV maps for different bias voltages of section of dimer row situated between two tungsten nanodots shown in Fig. 2. dI/dV for +0.46 V displays vertically wider area to verify that standing waves were confined in quasi-1D. (Bottom) Cross sectional profiles of the same quantum well as top images.

2 Eigenfunctions of the hydrogen atom

The Schrödinger equation for the H atom in spherical coordinates is:

$$\left[-\frac{1}{2}\nabla^2 - \frac{1}{r}\right]\psi_{nlm}\left(r,\theta,\phi\right) = E\psi_{nlm}\left(r,\theta,\phi\right)$$

This differential equation can be separated into a radial and angular part giving the general solution:

$$\psi_{nlm}\left(r,\theta,\phi\right) = R_{nl}\left(r\right)Y_{lm}\left(\theta,\phi\right)$$

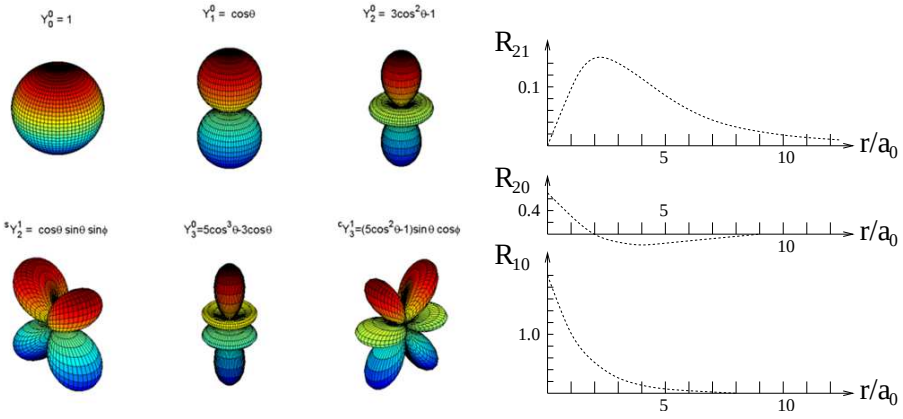


Figure 1: (left) Graphical depictions of spherical harmonic functions ($Y_{lm}\left(\theta,\phi\right)$) which describe the angular variation of the hydrogen eigenfunctions. (right) Sketch of first three radial eigenfunctions ($R_{nl}\left(r\right)$) for the hydrogen atom.

3 Numerical methods for integration of differential equations

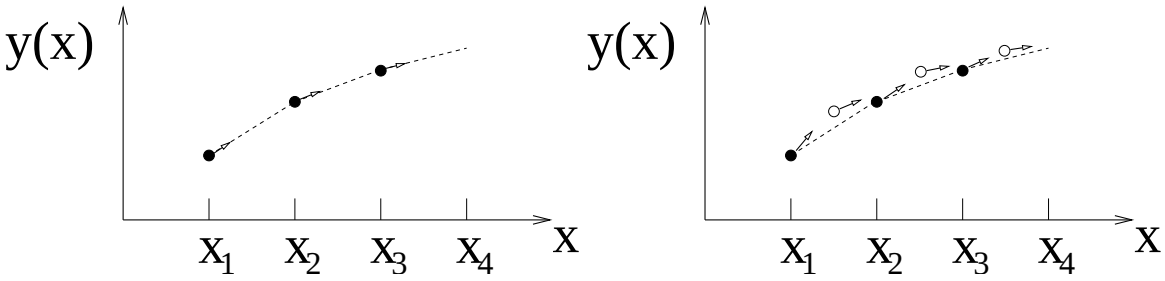


Figure 2: Graphical illustration of the Euler (left) and second order Runge-Kutta (right) methods for numerical integration of differential equations.

4 Further reading and problems

- Continuum Physics notes
- Lecture 1 Further Reading on VLE

Lecture 1: Further reading

Revision of the Schrödinger equation

The Schrödinger equation is a *time-dependent partial differential equation.* In this introductory lecture we do not consider the time dependence (that will come in lecture 3) so we will instead consider the simpler case of the *time independent* version. We can sometimes restrict things even more by considering just one spatial dimension, in which case, we only have a reasonably straightforward ordinary differential equation to solve, subject to some set of boundary conditions. This is exactly the type of problem that you have solved analytically in the past.

We start by rewriting the time independent Schrödinger equation

$$-\frac{\hbar^2}{2m_e}\nabla^2\psi\left(\mathbf{r}\right)+V\left(\mathbf{r}\right)\psi\left(\mathbf{r}\right)=E\psi\left(\mathbf{r}\right) \tag{1}$$

in Hartree atomic units, that is $\hbar = m_e = e = 4\pi\varepsilon_0 = 1$ [so from now on we shall always work in energy units of Hartrees ($27.2\,eV$) and in length units of Bohr radii ($a_0 = 0.529$ Å)] and so :

$$-\frac{1}{2}\nabla^2\psi+V\psi=E\psi \tag{2}$$

or in terms of the Hamiltonian operator $\widehat{H}$,

$$\widehat{H}\psi=E\psi \tag{3}$$

We know that this is an example of an *eigenvalue equation*, and that for a given potential $V\left(\mathbf{r}\right)$ and boundary conditions, there will in general be a number of different possible solutions, corresponding to different *eigenenergies* E_i with corresponding *eigenfunctions* ψ_i .

The first term in equation 1 corresponds to the kinetic energy of the state, and so we see that a smoothly varying wavefunction (small $\nabla^2\psi$) will generally be a lower energy state than a rapidly varying one. This can also be related to the number of nodes (zero crossings) in the wavefunction - a smooth wavefunction will have less nodes than a rapidly varying one.

Example - the hydrogen atom

In order to solve the Schrödinger equation for the hydrogen atom, we need to deal with a 3D equation in spherical polar coordinates. However, because of the spherical symmetry of the Coulomb potential, we can separate the solution into a radial and an angular part:

$$\psi(\mathbf{r}) = \psi(r,\theta,\phi) = R_{nl}(r)\,Y_{lm}(\theta,\phi) \tag{4}$$

where $\{n,l,m\}$ are *quantum numbers* (integers) which characterise the solution. If we substitute equation 4 into equation 2 and apply the *method of separation of variables*, we get two equations - a 2D angular equation, whose solution is given in terms of *spherical harmonics* and a 1D *radial equation:*

$$-\frac{1}{2}\frac{d^2\chi_{nl}(r)}{dr^2} + \left[V(r) + \frac{l(l+1)}{2r^2}\right]\chi_{nl}(r) = E\chi_{nl}(r) \tag{5}$$

where the substitution

$$\chi_{nl}(r) = rR_{nl}(r) \tag{6}$$

is used both to simplify the equation and ease its interpretation. Remember that the probability of finding the particle in a small volume d^3r is

$$\begin{aligned} P(\mathbf{r}\to\mathbf{r}+d^3r) &= |\psi(\mathbf{r})|^2\,d^3r \\ &= |\psi(\mathbf{r})|^2\,r^2dr\cdot\sin(\theta)\,d\theta d\phi \\ &= |\chi(r)|^2\,dr\cdot Y_{lm}^2(\theta,\phi)\sin(\theta)\,d\theta d\phi \end{aligned} \tag{7}$$

and so we see that the normalisation of χ_{nl} and Y_{lm} can be chosen such that

$$\int_0^\infty |\chi(r)|^2\,dr = 1 \tag{8}$$

i.e. $|\chi(r)|^2$ is the *radial probability density.*

Note that the second term in equation 5 looks very like a potential energy term in the standard time independent Schrödinger equation, and so is known as the *effective potential:*

$$V_{eff}(r) = V(r) + \frac{l(l+1)}{2r^2} \tag{9}$$

and the $\frac{l(l+1)}{2r^2}$ term is known as the *centrifugal barrier.*

What can we deduce about the general form of $\chi(r)$? Well, we know the form of the Coulomb potential and so as long as $V(r)\to-\infty$ as $r\to 0$ no more quickly than r^{-1} then the centrifugal barrier part of the effective potential dominates and we get

$$\frac{d^2\chi}{dr^2} = \frac{l(l+1)}{r^2}\chi \tag{10}$$

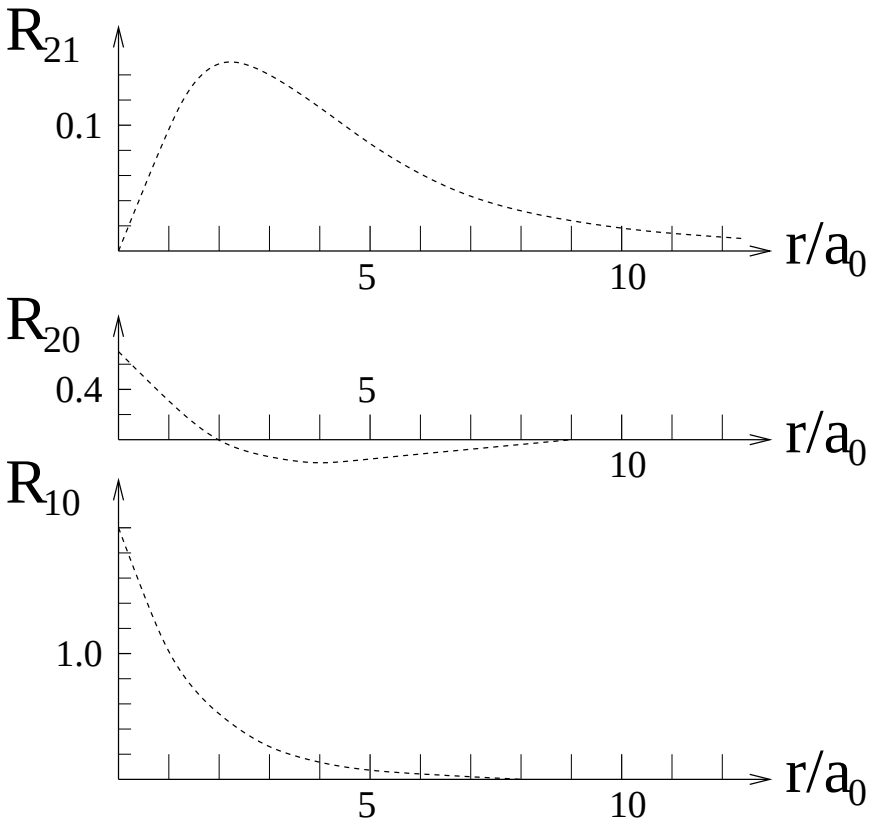


Figure 1: Sketch of first three radial eigenfunctions for the hydrogen atom.

which gives a non-divergent solution as

$$\begin{aligned} \chi(r \to 0) &\sim r^{l+1} \\ \Rightarrow R_l(r \to 0) &\sim r^l \end{aligned} \tag{11}$$

Similarly, as $r \to \infty$ we get

$$\frac{d^2\chi}{dr^2} = -2E\chi(r) \tag{12}$$

which gives two classes of solution:

$$\begin{aligned} \chi(r) &\sim e^{-\sqrt{2|E|}r} \quad E < 0 \quad (bound\ states) \\ \chi(r) &\sim e^{\pm i\sqrt{2E}r} \quad E > 0 \quad (continuum) \end{aligned} \tag{13}$$

For intermediate values of r we can either use more elaborate mathematics (which is possible for hydrogen but not for more complex atoms) or turn to numerical solutions. We can therefore use this known asymptotic behaviour to test our numerical solutions - every bound state solution, corresponding to an eigenenergy E_n, should tend to zero as $r \to \infty$, and for

$r \to 0$ should either tend to zero (for non-zero l) or tend to a cusp (for $l = 0$). See figure 1 for a sketch of the three lowest eigenenergy solutions which demonstrates this behaviour. We now turn to the problem of how to solve differential equations numerically.

Revision of solving ordinary differential equations numerically

Unfortunately, the number of cases where we can solve the Schrödinger equation exactly are rather small. More usually, we are faced with the problem of trying to find numerical solutions. As the Schrödinger equation is a linear second-order differential equation, this should not pose too much difficulty. We therefore begin by revising what we already know about solving ordinary differential equations, before going on to explore particular techniques that are suitable for the Schrödinger equation.

Most methods are designed for first-order differential equations of the general form

$$\frac{dy}{dx} = f\left(y\left(x\right), x\right) \tag{14}$$

Second-order equations of the general form

$$\frac{d^2y}{dx^2} = f\left(y\left(x\right), x\right) \tag{15}$$

can be transformed into a pair of coupled first-order equations using

$$\begin{aligned} \frac{dy}{dx} &= z\left(x\right) \\ \frac{dz}{dx} &= f\left(y\left(x\right), x\right) \end{aligned} \tag{16}$$

and then solved using the same techniques.

Most methods then proceed to use an approximation to the derivative based on a truncated Taylor series expansion, such as the *forward difference*:

$$\frac{dy}{dx} \approx \frac{y\left(x + \delta x\right) - y\left(x\right)}{\delta x} \tag{17}$$

the *backwards difference*:

$$\frac{dy}{dx} \approx \frac{y\left(x\right) - y\left(x - \delta x\right)}{\delta x} \tag{18}$$

or the *centred difference*:

$$\frac{dy}{dx} \approx \frac{y\left(x + \delta x\right) - y\left(x - \delta x\right)}{2\delta x} \tag{19}$$

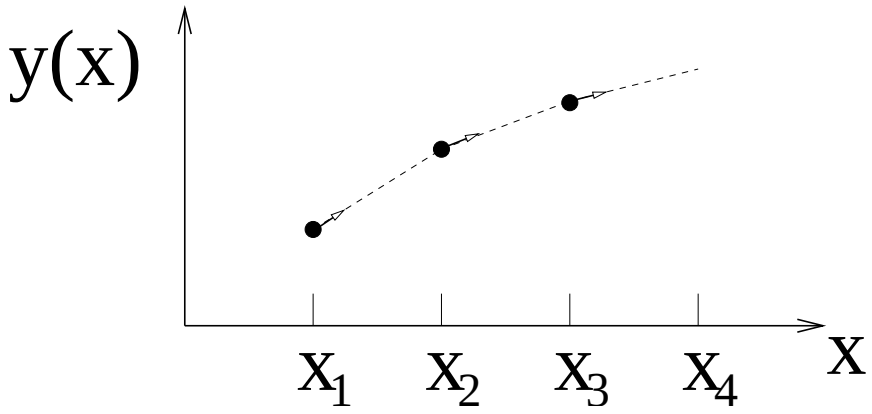


Figure 2: Illustration of Euler's method - the derivative at each point x_n is extrapolated over the whole step length h to form the next point.

A straightforwards implementation of this approach results in the *Euler method*:

$$y_{n+1} \approx y_n + f(y_n, x_n)\, h \tag{20}$$

where the solution advances from x_n to $x_{n+1} = x_n + h = x_n + \delta x$ and h is the step size. See figure 2 for an illustration of the method. Comparison to the Taylor series shows that this method has an error $O(h^2)$ and hence is called a *first-order method* (an n^{th}-order method has an error term $O(h^{n+1})$). Whilst the Euler method is the theoretical basis of most schemes, and as such is important, it should not be used as it stands as it suffers from large errors and is unstable. However, it can be improved upon to form the schemes we shall discuss below.

The Runge-Kutta method

The Runge-Kutta method is a common class of methods used for solving many differential equations. The simplest version improves upon the Euler method by doing it twice: we evaluate y_n using a single Euler step of length h and also using a midpoint step as follows:

$$\begin{aligned} k_1 &= f(x_n, y_n)\, h \\ k_2 &= f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) h \\ y_{n+1} &= y_n + k_2 + O\left(h^3\right) \end{aligned} \tag{21}$$

which is known as the *second-order Runge-Kutta* or *midpoint method*. It has the advantage of being more stable than the Euler method, and so can be

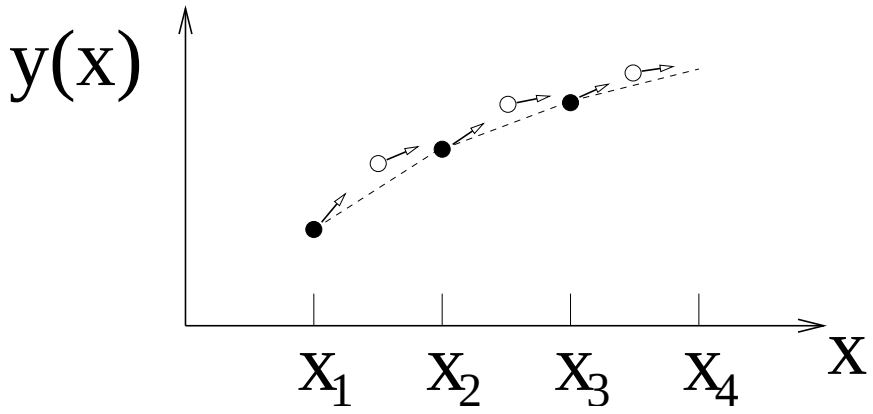


Figure 3: Second-order Runga-Kutta method - the derivative at the start of the step is used to find a point half-way across the interval h and this midpoint value is then used to improve the calculation of the final point at the end of the step. The filled circles represent final values which are of interest, whilst open circles are intermediate values k_i which are discarded at the end of each step.

used with a larger step size, and so requires less steps to span a given interval. Unfortunately, the cost for this higher order method is that it requires two function evaluations of $f(x)$ per step. Often, this is the most time consuming part of the calculation. See figure 3 for an illustration of this method.

This method can be extended to arbitrarily high orders. In practice, it is often found that the fourth-order method is the most efficient, in that higher order methods require many more function evaluations without bringing significant improvements in stability or step size. Note that a higher-order method does not always have higher accuracy! The *fourth-order Runge-Kutta method* is:

$$\begin{aligned} k_1 &= f(x_n, y_n)\, h \\ k_2 &= f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) h \\ k_3 &= f\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) h \\ k_4 &= f(x_n + h, y_n + k_3)\, h \\ y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O\left(h^5\right) \end{aligned} \tag{22}$$

A significant advantage of the Runge-Kutta method is that it can be used with a variable step size: at each step, an estimate of the error in the solution can be made, and the step size reduced if the error is larger than some prescribed tolerance, or the step size may be increased if a larger

error can be accepted. In this way, more attention is paid to those regions where the solution is rapidly varying, whilst not paying an unnecessarily high price of using an unnecessarily small step size where the function is slowly changing. This variable step size is achieved by evaluating y_{n+1} twice, once using a step size of h and then again using a step size of $\frac{h}{2}$ (i.e. using two steps to get to the same point). If the difference in y_{n+1} for the two different step sizes is Δ, then it can be shown that for the fourth order method, the required step size $h^{'}$ to keep the prescribed error to within δ is given by

$$h^{'} = \frac{15}{16} h \left| \frac{\delta}{\Delta} \right|^{\frac{1}{5}} \tag{23}$$

as the error term is $O\left(h^5\right)$. That is, if $h^{'} < h$ we must repeat the current step with the smaller step size, whilst if $h^{'} \geq h$ we may use $h^{'}$ as the new step size for the next step. There is, of course, some freedom in how to choose δ depending on the type of problem being solved - this may either be an absolute error, a fractional error, or an accumulated fractional error since the start of the calculation.

The fourth-order Runge-Kutta method, with variable step size, is perhaps the most widely used and trusted general purpose method for integrating ordinary differential equations. It has the advantage of being a self-starting method, that is, does not require knowledge of previous values to complete each step. However, it does not exploit any particular properties of the Schrödinger equation, and so in this instance there may be more specialised integration methods that can outperform it. We shall now consider one such method.

The Numerov method

The Numerov method is a specialised method for solving a restricted set of differential equations - those that can be written in the form:

$$\frac{d^2 y}{dx^2} = f\left(x\right) y\left(x\right) \tag{24}$$

which includes the time independent Schrödinger equation. The method exploits the special structure of this equation to produce a method that has fifth-order accuracy but only requires two function evaluations per step!

We can derive the method by expanding both $y\left(x \pm h\right)$ and $\left.\frac{d^2 y}{dx^2}\right|_{x \pm h}$ in Taylor series up to powers of h^4 and combining terms. There is a perfect cancellation of all odd-powers of h due to the symmetry of equation 24 and so the resulting method is accurate to order h^6. The result is

$$z\left(x+h\right) = 2z\left(x\right) - z\left(x-h\right) + h^2 f\left(x\right) y\left(x\right) + O\left(h^6\right) \tag{25}$$

where

$$z(x) = \left[1 - \frac{h^2}{12} f(x)\right] y(x) \tag{26}$$

The advantage of this method is that it has an error $O(h^6)$ which is an order of magnitude better than 4th-order Runge-Kutta and only requires two (rather than four) function evaluations per step. Hence it is the preferred method for solving the Schrödinger equation. However, it does have a drawback in that it is not self-starting due to the $z(x-h)$ term - that is, it requires another method to generate the first step, whereupon the Numerov method can be used. This feature of the method will also give problems if there are any discontinuities in the potential whereas the single-step Runga-Kutta method will be OK.

Singularities

There is a problem as to what to do at singularities. For example, in the hydrogen atom both the Coulomb potential and the centrifugal barrier diverge at the origin. However, for the hydrogen atom we have a known analytic result for $\chi(r=0)$ and so this can be substituted instead. But what about the general case of a singularity with non-zero l? Well, as long as the divergence is not too fast, then χ will still be well behaved, and we can do a Taylor series expansion for $\chi(r)$ around the singularity:

$$\chi(r) = \sum_{s=0}^{\infty} a_s r^{s+l+1} \tag{27}$$

from which it can be shown that

$$a_s = \frac{2}{s(s+2l+1)} \left(r V_{eff} a_{s-1} + \left(\frac{d(r V_{eff})}{dr} - E \right) a_{s-2} \right) \tag{28}$$

with $a_0 = 1$ (unless $l = 0$) and $a_{-1} = 0$. Therefore we can generate all the terms for χ, with an arbitrary normalisation that can be fixed later.

Other methods of solution

There are other methods that can be used to solve differential equations. For example, we might choose to discretize space and solve the equation on a grid, replacing differentials by appropriate sums over neighbouring grid points, and successively iterating to self-consistency. This is known as the finite-difference approach. It can be incorporated in various ways, such as with a single fixed grid, or with a hierarchy of grid sizes, or with a grid that adapts to the solution (e.g. putting more points in the regions where the function is rapidly changing). Such methods can be very successful and an example will be considered in a later lecture.

There are also indirect methods that can be used to produce information about the solution without directly solving the equation. Such methods exploit special properties of the equation, for example, that it has particular symmetries, or is an eigenvalue equation, or can be expressed in terms of a variational principle, etc. This will form a very important theme to later lectures in this course.

Errors

In general, when solving any differential equation numerically, there will be two dominant sources of error:

- truncation error
  - this is caused by the truncation of the Taylor series and represents the fundamental limit to the accuracy of a given algorithm. In general the error in a solution caused by truncation error can be reduced by using a smaller step size h.
- rounding error
  - this is caused by the finite representation of numbers within a digital computer. For example, when using single precision arithmetic in a typical modern computer, all *floating point numbers* will be accurate to about 8 significant figures, and when using double precision arithmetic they will be accurate to around 16 significant figures. In general, the error in the final solution caused by rounding error can be reduced by using fewer integration steps so that there is less accumulation of error, corresponding to a larger step size h.

It can be seen then that there is a competition between these two sources of error. Usually truncation error dominates, especially when using double precision arithmetic (which is essential for any kind of scientific computing). So when computing any solution, it is always a good idea to repeat the calculation with a smaller step size and see if there is any significant change in the solution. If there is, then the calculation must be repeated again with an even smaller step size, etc. until there is no significant change. However, if the step size is reduced too much, then rounding error will start to become signifcant and accuracy will be lost again.

Application to the Schrödinger equation

How then shall we use either the Runge-Kutta or Numerov method to solve the Schrödinger equation for any particular problem? If we are interested in

finding the bound states, i.e. the eigenenergies and eigenfunctions, then we must:

- choose a particular angular momentum (value of l)
- choose a trial energy (value of E_{trial})
- starting from $r = 0$ integrate the radial equation up to some appropriate large value of $r = r_{max}$
- examine the behaviour of the solution - in particular $\chi(r \to \infty)$
- if $\chi(r) \to \pm\infty$ then E_{trial} does not correspond to a bound state so adjust E_{trial} and repeat

This process of tuning E_{trial} can of course be automated, resulting in a variant of the "shooting method".

Note that the ground state solution will have no nodes. A general solution will have integer values for $\{n, l, m\}$ - but we will get no information on m from the radial equation and we have chosen l at the outset - so by counting the number of nodes we can deduce the value of n:

$$n = (number\ of\ nodes) + l + 1$$

Final comments

A few final points to highlight:

- The Schrödinger equation can often be written as an ordinary differential equation and solved using standard numerical techniques.
- Bound states should have a vanishing wavefunction in the long-range limit.
- The Numerov method is much more efficient than fourth-order Runga-Kutta for solving the Schrödinger equation.
- Different algorithms have different characteristic features, such as the ability to handle discontinuities or variable step sizes.
- A higher-order algorithm in general has a smaller truncation error than a lower-order one (but it depends on the unknown prefactor!)
- Rounding error will dominate if the step size is made too small.

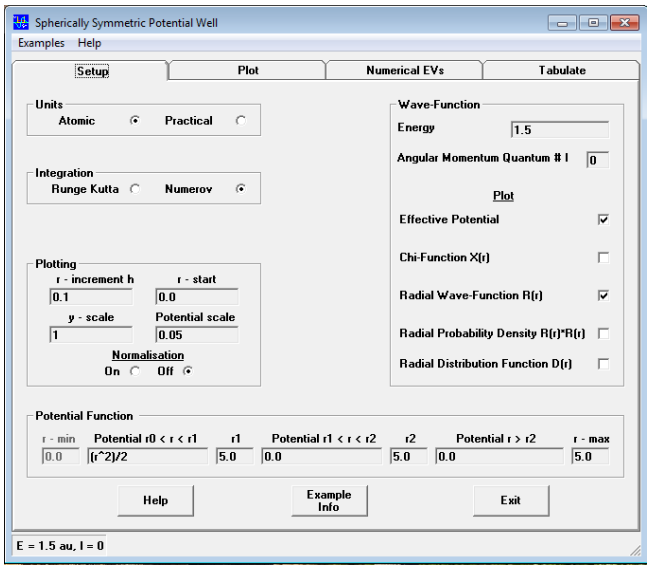
Computational Quantum Mechanics - Practical 1

In this practical session we will use the Quantum Mechanics program which is installed on all university managed windows computers.

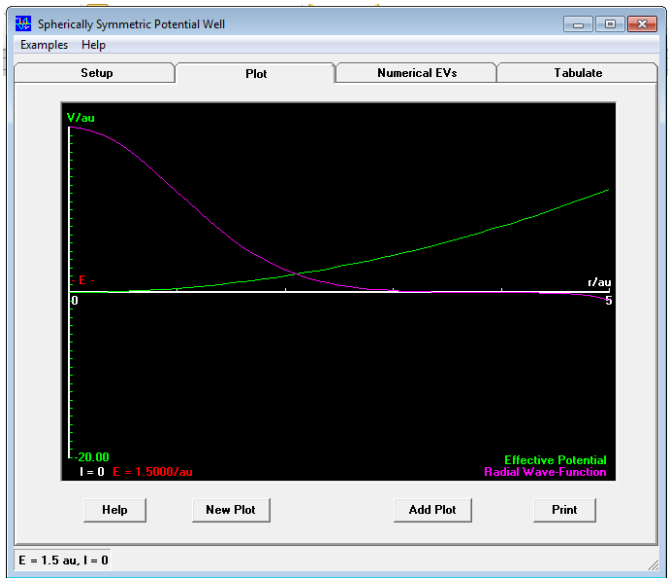
Load the spherically symmetric potentials program. This code solves the radial Schrodinger equation for a spherically symmetric potential $V(r)$. The help gives more details on how the code works and what it can do.

Part 1 - 3D Isotropic Harmonic Oscillator

On the setup tab set up the potential for a three-dimensional harmonic oscillator as shown below:



Double-clicking on the Plot tab shows the numerically integrated radial wavefunction:



Note that the radial wavefunction shown does not have the correct asymptotic form for a bound eigenstate. R(r) should tend towards zero as $r \rightarrow \infty$.

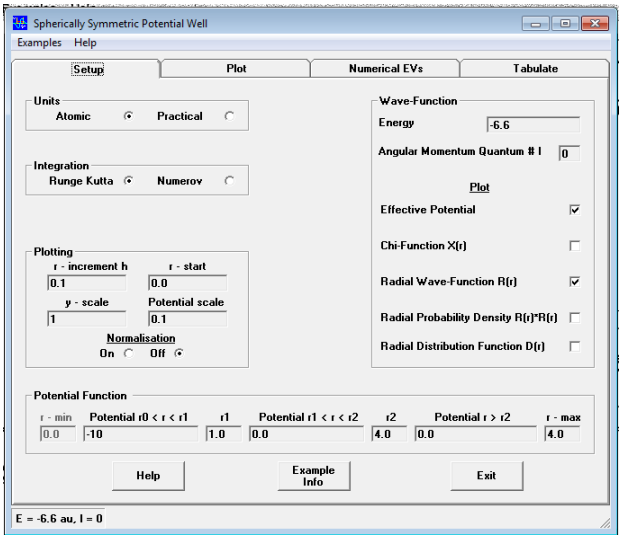
1. By manually adjusting the energy in the setup tab and replotting find the ground state eigenenergy to 6 decimal places.

2. You can also use the Numerical EVs tab to find eigenvalues using the shooting method automatically (see the help section for details). Find the ground state eigenvalue for both the Numerov and Runge-Kutta integration methods with step sizes h=0.1, 0.05, 0.01, 0.005, 0.001. Comment on the factors affecting the accuracy of the results.

3. Find the eigenenergy of the first excited state using the Numerov method and step sizes h=0.1, 0.05, 0.01, 0.005, 0.001.

Part 2 – Step potential well

Set up the potential as shown below:



1. Find the converged ground state eigenenergy (hint: in range -6.6 → -6.8 eV).

2. Find the converged first excited state eigenenergy (hint: in range -3.0 → -3.6 eV).

Part 3 – Hydrogen atom

1. Find the best converged value for the ground state eigenenergy of the H atom using both the Numerov and Runge-Kutta method (consider both the effect of the step size and the range of the potential).

How to use LAPACK to solve eigenvalue problems

What is LAPACK?

LAPACK is a collection of efficient routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. These routines can be called from within your codes to perform these tasks. The LAPACK website (www.netlib.org/lapack/) describes the purpose of the various routines and the algorithms which are employed.

Which LAPACK routine should I use?

The driver routines section in the LAPACK user guide (http://www.netlib.org/lapack/lug/node25.html) contains information on routines for solving many standard linear algebra problems. We can choose whichever routine meets our needs. For example, in the general linear (matrix) method we expand eigenfunctions in terms of some set of basis functions, e.g. in one-dimension:

$$\psi(x) = \sum_i c_i \chi_i(x).$$

Using the variational principle it can be shown that the coefficients which give the best estimate of energy satisfy the following eigenvalue equation:

$$\boldsymbol{Hc} = E\boldsymbol{Sc},$$

where ***H*** is the Hamiltonian matrix with matrix elements:

$$H_{ij} = \int \chi_i^* \, \hat{H} \chi_j^* dx,$$

S is the overlap matrix with matrix elements:

$$S_{ij} = \int \chi_i^* \, \chi_i dx,$$

E is energy and ***C*** is the vector containing the coefficients $\boldsymbol{c_i}$. Looking at the descriptions in the LAPACK user guide we can see that this equation is an example of a “Generalized Eigenvalue Problem” for which there are separate routines available for the symmetric and nonsymmetric cases.

If the basis functions used to expand the eigenfunctions are orthogonal the problem is simplified as the overlap matrix reduces to the identity matrix. In this case the equation we need to solve is:

$$\boldsymbol{Hc} = E\boldsymbol{c}.$$

Again, looking at the descriptions in the LAPACK user guide we can see that this equation is an example of a “Standard Eigenvalue Problem”. Again, there are separate routines available for the symmetric and nonsymmetric cases.

To find the particular routine you need look at the summary tables for the general eigenvalue problem (http://www.netlib.org/lapack/lug/node36.html#tabdrivegeig) or the standard eigenvalue problem (http://www.netlib.org/lapack/lug/node32.html#tabdriveseig). There are simple and expert versions of the routines. In most cases the simple version is sufficient. There are also separate routines depending on whether the matrix has real or complex values, and whether numbers are represented using single or double precision.

An example using LAPACK

Write a program to calculate the eigenvalues of the following equation,

$$\boldsymbol{Ac} = \lambda \boldsymbol{c}.$$

where A is a matrix whose elements are given by,

$$A_{ij} = |j - i|$$

You should allow the user to choose the size of the matrix.

This is a "standard eigenvalue problem" since there is no overlap matrix on the right hand side and the matrix ***A*** is symmetric. If we would like to use double precision the appropriate routine is DSYEV (see http://www.netlib.org/lapack/lug/node32.html#tabdriveseig). The definition of the routine and its required arguments can be found online (http://www.netlib.org/lapack/explore-html/).

The DSYEV subroutine requires quite a number of arguments. To simplify the fortran code we will write we will define a subroutine which takes as input only the matrix and its size and returns the eigenvalues in an array.

```
!-----------------------------------------------------------!
!Calls the LAPACK diagonalization subroutine DSYEV          !
!input:  a(n,n) = real symmetric matrix to be diagonalized!
!            n  = size of a                                 !
!output: a(n,n) = orthonormal eigenvectors of a             !
!        eig(n) = eigenvalues of a in ascending order       !
!-----------------------------------------------------------!
  subroutine diasym(a,eig,n)
    implicit none
    integer n,l,inf
    real*8  a(n,n),eig(n),work(n*(3+n/2))
    l=n*(3+n/2)
    call dsyev('V','U',n,a,n,eig,work,l,inf)
  end subroutine diasym
```

In our code we can now simply write

```
call diagsym(a,eig,n)
```

where a is the matrix ***A***, n is the dimension of the matrix, and eig is a 1D array that will contain the eigenvalues in ascending order after the LAPACK routine has been called.

An example full code is given below:

```
program eigensolve
  ! Declare variables
  implicit none
  integer :: i, j
  integer :: matrix_size
  real*8, allocatable :: A(:,:), eig(:)

  ! Read in matrix size
  write(*,'(A)') 'Please enter required matrix dimension:'
  read(*,*) matrix_size

  ! Generate matrix
  allocate (A(matrix_size,matrix_size))
  allocate (eig(matrix_size))
  do i=1,matrix_size
    do j=1,matrix_size
      A(i,j)= abs(j-i)
    enddo
  enddo

  ! Print out matrix
  write(*,'(/A)') 'Matrix A:'
  do i=1,matrix_size
    write(*,'(100F8.2)') (A(i,j), j=1,matrix_size)
  enddo

  ! Call diagonalisation routine
  call diasym(A,eig,matrix_size)

  ! Print out eigenvalues
  write(*,'(/A)') 'Eigenvalues of A is ascending order:'
  do i=1,matrix_size
    write(*,'(F8.2)') eig(i)
  enddo

  deallocate (A,eig)

  contains
```

```
!-----------------------------------------------------------!
!Calls the LAPACK diagonalization subroutine DSYEV          !
!input:  a(n,n) = real symmetric matrix to be diagonalized!
!            n  = size of a                                 !
!output: a(n,n) = orthonormal eigenvectors of a             !
!        eig(n) = eigenvalues of a in ascending order       !
!-----------------------------------------------------------!
  subroutine diasym(a,eig,n)
    implicit none
    integer n,l,inf
    real*8  a(n,n),eig(n),work(n*(3+n/2))

    l=n*(3+n/2)
    call dsyev('V','U',n,a,n,eig,work,l,inf)
  end subroutine diasym

end program eigensolve
```

Compiling the code

If you compile the above code in the normal way, e.g.

```
gfortran eigensolver.f90
```

you will get an error message saying something like (depending on the compiler)

```
undefined reference to dsyev_
```

This is the compiler telling us we are calling a subroutine that is not defined in our source file. We need to tell the compiler that it should use the LAPACK library routines. We do this in the following way

```
gfortran eigensolver.f90 -l lapack
```

This tells the compiler to look for a library file called liblapack.a and use it in compilation of the program. If the LAPACK library is installed on your computer it will compile successfully. If you still get an error message, the library is not installed. For linux operating systems it is simple to download the LAPACK libraries as a RPM and install them in the usual way. LAPACK should be already installed on the machines in the comp lab.

Summary

The above example demonstrates how library routines such as LAPACK can be used to solve common linear algebra problems such as solving eigenvalue equations. In the linear (matrix) method we expand the eigenfunctions of a Hamiltonian in terms of a set of basis functions and find that determining the eigenenergies and eigenfunctions reduces to a eigenvalue problem. For N basis functions we have a NxN Hamiltonian matrix. Importantly, library routines can be very efficient allowing large matrices (i.e. large basis sets) to be employed resulting in very accurate energies.