1 Atomic physics – hydrogen-like ions

- Here: focus on solving Schrodinger equation for single-electron wavefunction (e.g., hydrogen)
- Useful, even for more complex atoms, since we use single-electron wavefunctions to build multi-electron wavefunctions

Important to note: this is **not** a general overview to atomic physics. There are many extremely important topics I am not discussing at all, e.g., spin-orbit effect, theory of angular momentum addition and coupling etc., which are crucial for understanding quantum mechanics of atoms. However, those are essentially regular quantum mechanics, which you have seen in other courses, and do not require a computer. Here, I focus on the aspects of many-body atomic physics for which computational calculations are essential. Some excellent sources to pursue these other topics in more detail are books by Sakurai[1], Johnson[2], Sobelman[3], and Bethe and Salpeter[4], which are available in the library.

1.1 Hydrogenlike ions: quick review

A very quick review here; see above textbooks for more details. You have likely seen this before in your lectures (if not, don't worry, everything you need to know is here also).

1.1.1 Angular separation

In general, we have the Schrodinger equation:

$$\hat{H}\psi = \varepsilon\psi, \tag{1}$$

where

$$\hat{H}(\boldsymbol{r}) = \frac{\boldsymbol{p}^2}{2m} + V(\boldsymbol{r}) \tag{2}$$

with $\boldsymbol{p} = -i\hbar\boldsymbol{\nabla}$. For single-electron atoms, the potential is entirely spherically symmetric, meaning:

$$V(\boldsymbol{r}) = V(r) = -\frac{Ze^2}{4\pi\epsilon_0\, r}. \tag{3}$$

Therefore, it is simplest to work in spherical coordinates, in which

$$\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2}{\partial\phi^2}.$$

Motivated by the spherical symmetry, we use the separation of variables for ψ:

$$\psi(\boldsymbol{r}) = R(r)Y(\theta,\phi) = \frac{P(r)}{r}Y(\theta,\phi). \tag{4}$$

[1]J. J. Sakurai, Modern Quantum Mechanics (2011) [in particular Chapters 3, 5, 7]
[2]W. R. Johnson, Atomic Structure Theory (2007)
[3]I. I. Sobelman, Atomic Spectra and Radiative Transitions (1992)
[4]H. A. Bethe and E. E. Salpeter, Quantum Mechanics of One-and Two-Electron Atoms (1977)

Putting this back into the Hamiltonian equation, leads to two equations:

$$\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial Y}{\partial\theta}\right)+\frac{1}{\sin^2\theta}\frac{\partial^2 Y}{\partial\phi^2}+\lambda Y=0, \tag{5}$$

and

$$\frac{\partial^2 P}{\partial r^2}+\frac{2m}{\hbar^2}\left(\varepsilon-V(r)-\frac{\lambda\hbar^2}{2mr^2}\right)P=0, \tag{6}$$

where λ is a separation constant.

Notice that Eq. (5) is independent of the potential V and the energy ε; therefore the solutions are always the same. The solutions are the spherical harmonics

$$Y=Y_{lm} \tag{7}$$

for integer values of $\lambda\equiv l(l+1)$ and m. l takes values $0,1,2,3,...$, and m takes the values $-l,...,0,...,l$. This is not proved here, but is done in any quantum mechanics textbook.

The spherical harmonics are orbital momentum eigenstates according to:

$$L^2|lm\rangle=\hbar^2 l(l+1)|lm\rangle \tag{8}$$
$$L_z|lm\rangle=\hbar m|lm\rangle \tag{9}$$

where $\boldsymbol{L}=\boldsymbol{r}\times\boldsymbol{p}$ is the operator of (orbital) angular momentum, and using Dirac notation so that $|lm\rangle$ represents spherical harmonic Y_{lm}. Therefore, we recognise l as the total (orbital) angular momentum, and m as its projection onto the z quantisation axis. The spherical harmonics form a complete orthonormal set, and are normalised according to

$$\int Y^*_{l'm'}Y_{lm}\,\mathrm{d}\Omega=\delta_{ll'}\delta_{mm'}, \tag{10}$$

(I use the common short-hand for the angular integration measure: $\mathrm{d}\Omega=\sin\theta\mathrm{d}\phi\mathrm{d}\theta$).

Thus, the problem reduces to solving the radial equation (6), which depends on the potential V, which we can write as:

$$\underbrace{\left[\frac{-\hbar^2}{2m}\frac{\partial^2}{\partial r^2}+V(r)+\frac{l(l+1)\hbar^2}{2mr^2}\right]}_{H_r}P(r)=\varepsilon P(r). \tag{11}$$

We may therefore define the "radial Schrodinger equation", $H_rP=\varepsilon P$. For a given l, the equation has an infinite number of solutions – we label the bound-state solutions with n (principal quantum number) P_{nl}, $n=1,2,3,...$, with $n>l$. They are normalised as

$$\int P_{n'l}P_{nl}\,\mathrm{d}r=\delta_{nn'}. \tag{12}$$

For Hydrogenlike ions, $V(r)=-\frac{Ze^2}{4\pi\epsilon_0\,r}$, and the equation may be solved analytically. For example, the solutions for the energies are

$$\varepsilon_n=-\frac{Z^2R_y}{n^2}, \tag{13}$$

where $R_y=\frac{m_e e^4}{8(\pi\epsilon_0\hbar)^2}\approx 13.6\,\mathrm{eV}$. For more general potentials (e.g., for multi-electron atoms, or when considering finite-nuclear-size effects), the equation cannot be solved analytically, and must be solved numerically.

1.1.2 Bound-state solutions

Note that Eq. (1) has an infinite number of solutions, and has solutions for *any* given ε. However, we are most interested in a special set of solutions: the *bound state* solutions (states of definite energy, stationary states), which have $\psi \to 0$ as $r \to \infty$, and ψ normalisable (i.e. ψ everywhere finite).

- $\psi(\boldsymbol{r}) \to 0$ as $|r| \to \infty$
- $\psi(\boldsymbol{r})$ finite as $|r| \to 0$ – and continuous

The boundary conditions become simpler in terms of the radial function, P:

- $P(r) \to 0$ as $|r| \to \infty$
- $P(r) \to 0$ finite as $|r| \to 0$

  More explicitly

- $P_{nl}(r) \sim r^{l+1}$ as $|r| \to 0$
- $P_{nl}(r) \sim \exp(-\sqrt{2|\varepsilon|})$ finite as $|r| \to \infty$ (very rough)

Interestingly, these conditions hold for any atom, even multi-electron atoms. (The reason is that, for multi-electron atoms we still have $V(r) \sim -Z/r$ for small r, $V(r) \sim -1/r$ for large r.)

For bound states, we have $\varepsilon < 0$. There are also unbound (continuum) states, which have $\varepsilon > 0$, and $P(r) \sim \sin(\omega r)$ for large r – though we will not be directly concerned with these. The full set of solutions (including continuum) form a closed, complete, orthogonal set.

1.1.3 Matrix elements, expectation values

Typically, in any quantum mechanics problem, we are interested in calculating amplitudes (known as matrix elements):

$$\langle a|\hat{h}|b\rangle = \int \psi^*_{n_a l_a m_a} \hat{h}\, \psi_{n_b l_b m_b}\, \mathrm{d}V, \tag{14}$$

where for regular 3D spatial coordinates,

$$\mathrm{d}V \equiv r^2 \mathrm{d}r \mathrm{d}\Omega = r^2 \sin\theta\, \mathrm{d}r\, \mathrm{d}\theta\, \mathrm{d}\phi.$$

For now, we will consider only the simplest case: *radial* operators, which only depend on the radial coordinate: $\hat{h} = h(r)$ (sometimes called scalar operators). In this case, we have

$$\begin{aligned}\int \psi^*_{n_a l_a m_a} \hat{h}\, \psi_{n_b l_b m_b}\, \mathrm{d}V &= \left(\int P_{n_a l_a} h(r)\, P_{n_b l_b}\, \mathrm{d}r\right)\left(\int Y^*_{l_a m_a} Y_{l_b m_b}\, \mathrm{d}\Omega\right) &(15)\\ &= \left(\int P_{n_a l_a} h(r)\, P_{n_b l_b}\, \mathrm{d}r\right)\delta_{l_a l_b}\delta_{m_a m_b}. &(16)\end{aligned}$$

Therefore, matrix elements for such operators only depend on radial P functions; and are non-zero only for transitions in which the angular quantum numbers do not change. (Notice that the r^2 from the integration measure cancelled with the two factors of $1/r$ from the definition of P.)

1.1.4 Atomic units

- Measure mass in units of m_e
- Spin (/action) in units of $\hbar$
- Length in units of Bohr radius $a_0 = a_B = \frac{4\pi\epsilon_0\hbar^2}{m_e e^2}$
- Charge in units of e (with Gaussian choice of: $4\pi\epsilon_0 = 1$)
    - e is the *elementary* charge, so $e = |e| > 0$. Electron has charge $-e$. Beware: roughly half the textbooks choose the opposite definition. Usually, it's e^2 that arises in problems, so it rarely matters
- Energy in Hartree units, $2R_y = \frac{m_e e^4}{4(\pi\epsilon_0\hbar)^2} \approx 27.2\,\text{eV}$

In these units, important constants take the values: $m_e = e = |e| = 4\pi\epsilon_0 = \hbar = a_0 = 1$, and speed of light $c = 1/\alpha$, where $\alpha = \frac{e^2}{4\pi\epsilon_0\hbar c} \approx 1/137$ is the fine structure constant. In these units, the radial Schrodinger equation (11) takes the simple form:

$$\left[\frac{-1}{2}\frac{\partial^2}{\partial r^2} + V(r) + \frac{l(l+1)}{2r^2}\right]P(r) = \varepsilon P(r). \tag{17}$$

For Hydrogenlike ions, we again have:

$$V(r) = -\frac{Z}{r} \tag{18}$$

and

$$\varepsilon_n = -\frac{Z^2}{2n^2}. \tag{19}$$

1.2 Numerical solution to radial equation

$$\left[\frac{-1}{2}\frac{\partial^2}{\partial r^2} + V(r) + \frac{l(l+1)\hbar^2}{2r^2}\right]P(r) = \varepsilon P(r), \tag{20}$$

For hydrogenlike ions, this can be solved exactly. The analytic hydrogen-like solutions are given in many textbooks (and are on Wikipedia), so won't be repeated here. For more general potentials, however, it must be solved numerically. This will be the starting point for more complex systems – only $V(r)$ will change. The same techniques apply for general $V(r)$ potentials.

There are many methods available to solve the equation; here I will outline a few briefly.

1.2.1 Linear multi-step method

The Schrodinger equation is a second-order ODE. It is convenient to re-cast it in terms of a pair of coupled first-order ODEs, and represent this using a matrix notation.

Define

$$Q(r) \equiv \frac{\mathrm{d}P}{\mathrm{d}r}, \qquad y(r) \equiv \begin{pmatrix} P(r) \\ Q(r) \end{pmatrix}. \tag{21}$$

Then, the radial Schrodinger equation can now be expressed

$$\frac{\mathrm{d}y}{\mathrm{d}r} = \begin{pmatrix} Q(r) \\ -2\left(\varepsilon - V - \frac{l(l+1)}{2r^2}\right)P(r) \end{pmatrix} \tag{22}$$

$$= D(r)y(r), \tag{23}$$

where D is the 2x2 matrix (that depends on r)

$$D = \begin{pmatrix} 0 & 1 \\ -2\left(\varepsilon - V - \frac{l(l+1)}{2r^2}\right) & 0 \end{pmatrix} \tag{24}$$

This has the form of a first-order ODE for y, but is a matrix equation.

If you're not used to this notation, notice that this is simply a matrix form for the pair of equations:

$$P'(r) = Q(r) \tag{25}$$

$$Q'(r) = P''(r) = -2\left(\varepsilon - V - \frac{l(l+1)}{2r^2}\right)P(r), \tag{26}$$

i.e., we are casting the second-order ODE into a pair of coupled first-order ODEs, and then expressing this in matrix form.

Now; how do we solve the equation? First, assume we know $y(r)$ for some point r, and want to find $y(r+\Delta r)$. Since we know the derivative, we can project forwards:

$$y(r+\Delta r) \approx y(r) + \frac{\mathrm{d}y}{\mathrm{d}r}\Delta r. \tag{27}$$

So far, this is equivalent to the simple Euler's method, which we used in the first worksheet.

We can now be more accurate, and write instead:

$$y(r+\Delta r) = y(r) + \int_r^{r+\Delta r} D(r')y(r')\,\mathrm{d}r'. \tag{28}$$

(Notice that to lowest-order approximation of the integral for small Δr, these are equivalent.)

Using an $(N+1)$-step numerical integration method (more next lecture), with $\Delta r = N\delta r$, we have

$$y(r+\Delta r) \approx y(r+(N-1)\delta r) + \delta r\sum_{i=0}^{N} b_i D(r+i\delta r)y(r+i\delta r). \tag{29}$$

The b_i are numerical integration "weights" – will be discussed next lecture. We move the $i=N$ term to the left, and solve for $y(r+\Delta r)$:

$$y(r+\Delta r) \approx [1-\delta r b_N D(r+N\delta r)]^{-1}\left(y(r+(N-1)\delta r) + \delta r\sum_{i=0}^{N-1} b_i D(r+i\delta r)y(r+i\delta r).\right) \tag{30}$$

Therefore, we can approximate $y(r+\Delta r)$ so long as N previous values $y(r)$, $y(r+\delta r)$, $y(r+2\delta r)$,... are known. Note that it involves finding the inverse of a 2x2 matrix.

- Very accurate
- Need N initial points: e.g., Use low-r expansion
- Derivative operator depends on ε

Since the derivative operator depends on ε, we cannot solve for P and ε at the same time. Instead, we have to guess ε, and then solve the equation. Most likely, this guess will not be correct – this means the solution will not have the correct boundary condition. Keep making small adjustments to the guessed ε until the boundary conditions match – then we have successfully solved for the bound state.

1.2.2 Cast derivative operator to matrix (finite difference method)

- Not most accurate, but simplest

We can numerically approximate the first- and second-order derivative as:

$$\frac{\mathrm{d}f}{\mathrm{d}r} \approx \frac{f(x+\delta/2)-f(x-\delta/2)}{\delta} \tag{31}$$

$$\frac{\mathrm{d}^2f}{\mathrm{d}r^2} \approx \frac{f(x-\delta)-2f(x)+f(x+\delta)}{\delta^2}. \tag{32}$$

If we choose a finite spacing $\{..., x_1, x_2, x_3, ..., x_n, ...\}$, where each point is spaced δ apart: $x_{n+1} = x_n + \delta$, we may write the function f as a vector in this finite space:

$$f = \begin{pmatrix} \vdots \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_n) \\ \vdots \end{pmatrix}. \tag{33}$$

Then, we may cast the derivative as a matrix multiplication:

$$\frac{\mathrm{d}^2f}{\mathrm{d}r^2} \approx \frac{1}{\delta^2}\begin{pmatrix} & & \ddots & \vdots & & & \\ \dots & 0 & 1 & -2 & 1 & 0 & \dots \\ & & & \vdots & \ddots & & \end{pmatrix}\begin{pmatrix} \vdots \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \end{pmatrix} = \frac{1}{\delta^2}\begin{pmatrix} \vdots \\ f(x_0)-2f(x_1)+f(x_2) \\ f(x_1)-2f(x_2)+f(x_3) \\ f(x_2)-2f(x_3)+f(x_4) \\ \vdots \end{pmatrix}. \tag{34}$$

Take a moment to ensure this makes sense.

Of course, we cannot compute a matrix of infinite size. Therefore, we must choose a finite spacing δ. Further, we must truncate the vector somewhere. We can do this by assuming $f(x) = 0$ for $x < x_{\min}$ and $x > x_{\max}$. This is called the *hard boundary condition.*

In this case, we have a finite-dimensional matrix equation for the derivative:

$$\frac{\mathrm{d}^2f}{\mathrm{d}r^2} \approx \frac{1}{\delta^2}\begin{pmatrix} -2 & 1 & 0 & \dots & & & \\ 1 & -2 & 1 & 0 & \dots & & \\ 0 & 1 & -2 & 1 & 0 & \dots & \\ & & & & \ddots & & \\ & & \dots & 0 & 1 & -2 & \end{pmatrix}\begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_n) \end{pmatrix} = \frac{1}{\delta^2}\begin{pmatrix} -2f(x_1)+f(x_2) \\ f(x_1)-2f(x_2)+f(x_3) \\ f(x_2)-2f(x_3)+f(x_4) \\ \vdots \\ f(x_{n-1})-2f(x_n) \end{pmatrix}. \tag{35}$$

with $f(x_0) = f(x_{n+1}) = 0$. We use n equally spaced points from x_1 to x_n, so that δ,

$$\delta = \frac{x_n - x_1}{n-1}. \tag{36}$$

Note that ideally, we would use $r_{\min} = 0$, however, we cannot evaluate $V(r)$ at this point. That's fine, though, since we know $P(0) = 0$. Instead we 'drop' this point, and shift everything along by δr.

In practise, this is like choosing $r_{\rm min} = \delta r$. It's important that the radial grid "lines-up" correctly when using this method, since otherwise the derivative at the end-points would be ill-defined. (As an aside: in more accurate atomic calculations, we typically do not use a uniformly spaced radial grid, since the wavefunctions vary rapidly near the origin; in that case, the problem is circumvented differently.)

Using this, we may re-cast the radial Schroedinger equation into a matrix equation:

$$\left[\frac{-1}{2}D_2 + V(r) + \frac{l(l+1)}{2r^2}\right] P = \varepsilon P, \tag{37}$$

where D_2 in the $n \times n$ second-order derivative matrix. This is a simple eigenvalue problem, which can be solved using usual techniques. Notice that, in order for $V(r) * f(r)$ to work, $V(r)$ (and any other function of r) can be cast simply to a diagonal matrix:

$$\begin{pmatrix} V(x_1) & & & & 0 \\ & V(x_2) & & & \\ & & V(x_3) & & \\ & & & \ddots & \\ 0 & & & & V(x_n) \end{pmatrix}. \tag{38}$$

Since the finite-element Hamiltonian matrix

$$H = \left[\frac{-1}{2}D_2 + V(r) + \frac{l(l+1)}{2r^2}\right]$$

is real and symmetric, we can use the LAPACK routine DSYEV. The output will be a set of n eigenvalues, and n eigenvectors. The eigenvalues are the energies, and the eigenvectors are the wavefunctions (evaluated at the discrete x_i points).

Note that due the the "hard boundary condition", we have $P(r) = 0$ for $r > r_{\rm max}$. However, the correct boundary condition should be $P(r_n) \sim \exp(-\sqrt{2|\varepsilon|}r_n)$. This is not a problem, so long as all the wavefunctions we are directly interested in effectively go to zero well inside the maximum radius r_n. i.e., we must choose r_n to be much larger than the typical radius of the electron orbitals we are interested in. The expectation value for r scales as

$$\langle r\rangle \sim \frac{n^2}{Z}a_B.$$

Note that by solving the equation in this way, we have generated a full set of wavefunctions (eigenvectors). We are typically only directly interested in the first few. Of course, since we approximated the derivative using finite-elements, they are not exact, however, we have roughly a full set of orthonormal wavefunctions. This is important, since we often need to use a complete set of states to sum over in higher-orders of perturbation theory. Recall for a perturbative expansion $H \to H + \delta h$, $\psi \to \psi_0 + \delta\psi$, $\varepsilon \to \varepsilon + \delta\varepsilon$, we have:

$$|\delta\psi\rangle = \sum_n \frac{|n\rangle\langle n|\delta h|\psi_0\rangle}{\varepsilon_0 - \varepsilon_n}. \tag{39}$$

1.3 lapack/dsyev example: solve DE as matrix eigenvalue problem

LAPACK (Linear Algebra PACKage) is a large set of programs for solving general linear algebra problems. It in turn uses BLAS (Basic Linear Algebra Subprograms), which is a set of routines that provide standard building blocks for performing basic vector and matrix operations. LAPACK and

BLAS are the standard way to do computational linear algebra - essentially every other tool/programming language uses them 'under the hood'. LAPACK is written in FORTRAN90; we can make direct calls to these routines using c++.

LAPACK provides many functions for various circumstances. Since we have a real-values symmetric matrix, and we want to solve the eigenvalue problem, we shall use the `DSYEV` routine: D=double (as in double-precision float), SY=symmetric, EV=eigenvalue. We need to declare the function in our c++ file in order to use it; the function declaration should have the following form:

Example 1.1: `DSYEV` Parameters. Documentation: http://www.netlib.org/lapack/explore-html/index.html.

```
extern "C"
int dsyev_(
  char * jobz,   // 'V' = compute e. values and vectors. 'N' = values only
  char * uplo,   // 'U' = upper triangle of matrix is stored, 'L' = lower
  int * n,       // dimension of matrix a
  double * a,    // c-style array for matrix a (ptr to array, pointer to a[0])
                 // On output, a contains matrix of eigenvectors
  int * lda,     // For us, lda=n
  double * w,    // array of dimension n - will hold eigenvalues
  double * work,// 'workspace': array of dimension lwork
  int * lwork,   // dimension of workspace: ~ 6*n works well
  int * info     // error code: 0=worked.
);
```

- Since we are calling a function that lives outside of c++, we must mark it with "*extern "C"*" (this tells the linker to use low-level C-style linking). The 'symbol' (function name) that we call is `dsyev_`.
- Naming the parameters is not required in the function declaration, but I think it aids readability.
- Since we are interfacing with the Fortran routines, we must pass all the data into the function via pointers to variables/arrays.
- Output eigenvalues are sorted, and eigenvectors are normalised to 1 (via vector inner product)
- Note: FORTRAN (language LAPACK is written in) uses column-major ordering to access 2D arrays, wile c and c++ use row-major. This means m[i][j] in c++ is m[j][i] in FORTRAN.. so we often need to transpose the matrix before sending to LAPACK
  - Our matrix is symmetric, so this doesn't matter, except for 'uplo'
  - 'uplo': 'U' means upper triangle in FORTRAN is stored – so lower in c++ [we can just fill entire matrix though]
  - For other LAPACK functions, you can often just tell them the matrix is a transpose, so we don't need to waste time transposing it ourselves
- When compiling the code, we need to pass the '-llapack' linker flag. This tells the compiler to look in the lapack libraries for the function we are calling.
  - e.g., `g++ -O3 -o program program.cpp -llapack`
  - Some systems may also need '-lblas' : `g++ ...  -llapack -lblas`

We will go through an example for this in the workshop.

1.4 Basis of single-electron wavefunctions ("orbitals")

Here we briefly review another method for solving the radial Schrödinger equation that is particularly useful for multi-electron atoms. As we shall see below, the total wavefunction for a multi-electron atom is formed from combinations of single-particle wavefunctions (which we often refer to as "orbitals", though that term is used differently in many sources).

1.4.1 Algebraic solution to radial equation

Use some (finite) set of basis functions $\{b_i(r)\}$, and write:

$$P(r) = \sum_i c_i b_i(r). \tag{40}$$

In general, this is approximate, since the basis set is finite. As such, a good choice of basis is essential; this will be discussed below. Remember that here $P(r)$ is defined via the radial decomposition for the single-particle wavefunctions

$$\psi_{nlm}(\boldsymbol{r}) = \frac{P_{nl}(r)}{r} Y_{lm}(\theta, \phi), \tag{41}$$

and is a solution to the *radial* Schrödinger equation:

$$HP = \varepsilon P. \tag{42}$$

Solving this equation is cast to determining the expansion coefficients $\{c_i\}$. Note: in general, the basis states $|b_j\rangle$ are orthogonal or normalised, and they are not eigenstates of the Hamiltonian.

Using Dirac notation this gives:

$$\sum_i H|b_i\rangle c_i = \varepsilon \sum_j |b_j\rangle c_j \tag{43}$$

$$\sum_i \langle b_j|H|b_i\rangle c_i = \varepsilon \sum_i \langle b_j|b_i\rangle c_i, \tag{44}$$

where we multiplied on the left by $\langle b_j|$.The result is just a matrix equation:

$$\sum_i H_{ji} c_i = \varepsilon \sum_i B_{ji} c_i \tag{45}$$

$$\implies H\boldsymbol{c} = \varepsilon B\boldsymbol{c}. \tag{46}$$

This is a generalised eigenvalue matrix equation, which can be solved to yield a set of eigenvalues ε, and eigen vectors $\boldsymbol{c}$. Each eigenvector is the set of c_i expansion coefficients. Note that if the basis set is orthonormal, the matrix $B_{ij} = \langle b_i|b_j\rangle$ is just the identity; however, this is not true in general.

In theory, any basis set will do (e.g., polynomials). In practice, since we can only use a finite set, choosing a good basis set is important. Good choices are Gaussian basis states, B-splines, or hydrogen-like wavefunctions.

The FORTRAN routine, DSYGV, will solve the generalised eigenvalue problem for real, symmetric matrices. (DSYGV is similar to DSYEV, but takes in also second matrix, B.)

Here, H and B are $N_b \times N_b$ square matrices, with elements:

$$H_{ij} = \langle b_i|\hat{H}|b_j\rangle = \int b_i(r)\hat{H}b_j(r)\,\mathrm{d}r\,, \qquad B_{ij} = \langle b_i|b_j\rangle = \int b_i(r)b_j(r)\,\mathrm{d}r, \tag{47}$$

with N_b being the number of basis states (B-splines) used the the expansion (40) – not the number of radial grid-points used for integrals. If the basis was orthonormal, B would just be the identity matrix; in general it is not.

For the general case of $H = -\frac{1}{2}\frac{\partial^2}{\partial r^2} + V(r)$, we have

$$H_{ij} = -\frac{1}{2}\int b_i(r)b_j''(r)\,\mathrm{d}r + \int b_i(r)V(r)b_j(r)\,\mathrm{d}r \tag{48}$$

$$= +\frac{1}{2}\int b_i'(r)b_j'(r)\,\mathrm{d}r + \int b_i(r)V(r)b_j(r)\,\mathrm{d}r \tag{49}$$

(integration by parts). In theory, these integrals can be found with extremely high accuracy using Gaussian quadrature. For our purposes, any reasonable integration scheme will do just fine.

Example 1.2: `DSYGV` Parameters. Documentation: http://www.netlib.org/lapack/explore-html/index.html.

```
extern "C"
int dsygv_(
    int *ITYPE,    // =1 for problems of type Av=eBv
    char *JOBZ,    // ='V' means calculate eigenvectors
    char *UPLO,    // 'U': upper triangle of matrix is stored, 'L': lower
    int *N,        // dimension of matrix A
    double *A,     // c-style array for matrix A (ptr to array, pointer to a[0])
                   // On output, A contains matrix of eigenvectors
    int *LDA,      // For us, LDA=N
    double *B,     // c-style array for matrix B [Av=eBv]
    int *LDB,      // For us, LDB =N
    double *W,     // Array of dimension N - will hold eigenvalues
    double *WORK,  // 'workspace': array of dimension LWORK
    int *LWORK,    // dimension of workspace: ~ 6*N works well
    int *INFO      // error code: 0=worked.
);
```

- Note: FORTRAN (language LAPACK is written in) uses column-major ordering to access 2D arrays, while c and c++ use row-major. This means m[i][j] in c++ is m[j][i] in FORTRAN.. so we often need to transpose the matrix before sending to LAPACK
  - Our matrix is symmetric, so this doesn't matter, except for 'uplo'
  - 'uplo': 'U' means upper triangle in FORTRAN is stored – so lower in c++ [you may just fill entire matrix]
  - For other LAPACK functions, you can often just tell them the matrix is a transpose, so we don't need to waste time transposing it ourselves
- Don't forget to declare the 'dsygv_' function with 'extern "C"', and use the -llapack linker (compile) flag (you may also need the -lblas flag)

1.4.2 B-spline basis functions

There are many options for which set of basis functions to use in Eq. (40). One option that works very well for atomic physics is to use "B-splines".

B-splines are are a set of N piecewise polynomials of order k, defined over a sub-domain $r \in [0, R]$.

- Each spline function, b_i, is non-zero for only a small sub-region of the domain.

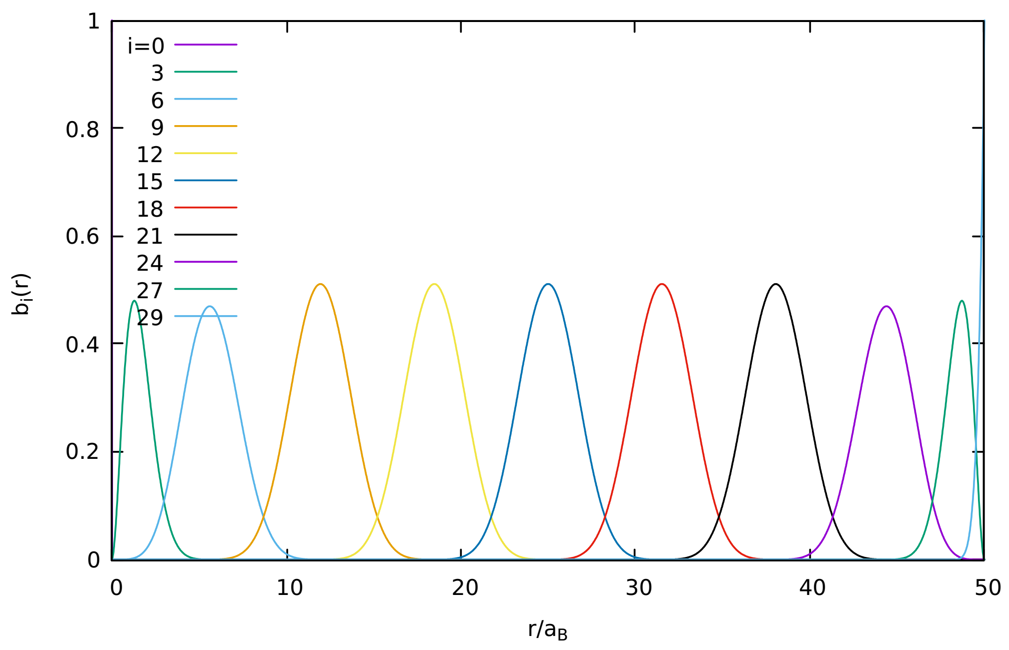

Figure 1: Every 3rd spline from a set of 30 B-splines of order $k = 7$ defined on $r \in [0, 50]$ a.u. The first non-zero knot was placed at $r_0 = 10^{-3}$ a.u., and the knots were distributed uniformly over $[r_0, R]$ (though it is typical to instead distribute them logarithmically).

- Technical details (you don't need to know):
- $b_i(r)$ is non-zero only for $t_i \leq r < t_{i+n}$, where $\{t_0, \ldots, t_N\}$ are a series of $N + 1$ "knots".
- First k knots are placed at $r = 0$; the final k knots are at $r = R$; the remaining knots are distributed between some non-zero r_0 and R.

- All splines go to zero at $r = 0$, except the 0th spline (index 0); the 0th spline (index 0) is only non-zero for $r < r_0$
- The kth spline and above [index k] are non-zero only for $r > r_0$
- All splines go to zero at $r = R$, except for the final one $(N - 1)$th spline
- For any $r \in [0, R]$, $\sum_i b_i(r) = 1$
- Form a complete basis for any polynomial of order k on interval $r \in [0, R]$
- Typically, we choose $r_0 \sim 10^{-4}$ a.u., $R \sim 30 - 75$ a.u., $N = 30 - 100$, and $k = 7 - 9$.

1.4.3 Enforce boundary conditions

There are several ways to enforce the boundary conditions, including adding extra fictitious infinite potentials to the Hamiltonian at $r = 0$ and $r = R$. We will take a simpler approach, and enforce the boundary conditions by discarding some of the splines.

- Discard b_0 – this is the only spline non-zero at $r = 0$, forcing wavefunctions to be $P(0) = 0$.
- Discard final spline b_{N-1} – this is the only spline non-zero at $r = R$, forcing wavefunctions to be $P(R) = 0$ ("hard" boundary condition).

- Discard b_1 – We can further take advantage of low-r behaviour of the wavefunctions; $P(r) \sim r^{l+1}$. Due to the form of the splines, we can force this behaviour by discarding b_1 for s and p states.
- So, with N total splines, we actually use only $N-3$ in the expansion Eq. (40).