# Chapter 12

# Eigensystems

## *12.1 Introduction*

Together with linear equations and least squares, the third major problem in matrix computations deals with the algebraic eigenvalue problem. Here we limit our attention to the symmetric case. We focus in particular on two similarity transformations, the Jacobi method, the famous QR algoritm with Householder's method for obtaining a triangular matrix and Francis' algorithm for the final eigenvalues. Our presentation follows closely that of Golub and Van Loan, see Ref. [25].

## *12.2 Eigenvalue problems*

Let us consider the matrix **A** of dimension n. The eigenvalues of **A** is defined through the matrix equation

$$\mathbf{A}\mathbf{x}^{(\nu)} = \lambda^{(\nu)}\mathbf{x}^{(\nu)}, \tag{12.1}$$

where $\lambda^{(\nu)}$ are the eigenvalues and $\mathbf{x}^{(\nu)}$ the corresponding eigenvectors. Unless otherwise stated, when we use the wording eigenvector we mean the right eigenvector. The left eigenvector is defined as

$$\mathbf{x}^{(\nu)}{}_L\mathbf{A} = \lambda^{(\nu)}\mathbf{x}^{(\nu)}{}_L$$

The above right eigenvector problem is equivalent to a set of $n$ equations with $n$ unknowns $x_i$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = &\quad \lambda x_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = &\quad \lambda x_2 \\ \cdots &\quad \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = &\quad \lambda x_n. \end{aligned}$$

We can rewrite Eq. (12.1) as

$$\left(\mathbf{A} - \lambda^{(\nu)}I\right)\mathbf{x}^{(\nu)} = 0,$$

with $I$ being the unity matrix. This equation provides a solution to the problem if and only if the determinant is zero, namely

$$\left|\mathbf{A} - \lambda^{(\nu)}\mathbf{I}\right| = 0,$$

which in turn means that the determinant is a polynomial of degree $n$ in $\lambda$ and in general we will have $n$ distinct zeros. The eigenvalues of a matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$ are thus the $n$ roots of its characteristic polynomial

$$P(\lambda) = det(\lambda\mathbf{I} - \mathbf{A}), \tag{12.2}$$

or

$$P(\lambda) = \prod_{i=1}^{n} (\lambda_i - \lambda)\,. \tag{12.3}$$

The set of these roots is called the spectrum and is denoted as $\lambda(\mathbf{A})$. If $\lambda(\mathbf{A}) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ then we have

$$det(\mathbf{A}) = \lambda_1\lambda_2 \ldots \lambda_n,$$

and if we define the trace of $\mathbf{A}$ as

$$Tr(\mathbf{A}) = \sum_{i=1}^{n} a_{ii}$$

then $Tr(\mathbf{A}) = \lambda_1 + \lambda_2 + \cdots + \lambda_n$.

Procedures based on these ideas can be used if only a small fraction of all eigenvalues and eigenvectors are required or if the matrix is on a tridiagonal form, but the standard approach to solve Eq. (12.1) is to perform a given number of similarity transformations so as to render the original matrix $\mathbf{A}$ in either a diagonal form or as a tridiagonal matrix which then can be be diagonalized by computational very effective procedures.

The first method leads us to Jacobi's method whereas the second one is given by Householder's algorithm for tridiagonal transformations. We will discuss both methods below.

## *12.3 Similarity transformations*

In the present discussion we assume that our matrix is real and symmetric, that is $\mathbf{A} \in \mathbb{R}^{n\times n}$. The matrix $\mathbf{A}$ has $n$ eigenvalues $\lambda_1 \ldots \lambda_n$ (distinct or not). Let $\mathbf{D}$ be the diagonal matrix with the eigenvalues on the diagonal

$$\mathbf{D} = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & \ldots & \ldots & \lambda_{n-1} & \\ 0 & \ldots & \ldots & \ldots & \ldots & 0 & \lambda_n \end{pmatrix}.$$

If $\mathbf{A}$ is real and symmetric then there exists a real orthogonal matrix $\mathbf{S}$ such that

$$\mathbf{S}^T\mathbf{A}\mathbf{S} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n),$$

and for $j = 1 : n$ we have $\mathbf{A}\mathbf{S}(:, j) = \lambda_j\mathbf{S}(:, j)$. See chapter 8 of Ref. [25] for proof.

To obtain the eigenvalues of $\mathbf{A} \in \mathbb{R}^{n\times n}$, the strategy is to perform a series of similarity transformations on the original matrix $\mathbf{A}$, in order to reduce it either into a diagonal form as above or into a tridiagonal form.

We say that a matrix $\mathbf{B}$ is a similarity transform of $\mathbf{A}$ if

$$\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}, \qquad \text{where} \qquad \mathbf{S}^T\mathbf{S} = \mathbf{S}^{-1}\mathbf{S} = \mathbf{I}.$$

The importance of a similarity transformation lies in the fact that the resulting matrix has the same eigenvalues, but the eigenvectors are in general different. To prove this we start with the eigenvalue problem and a similarity transformed matrix $\mathbf{B}$.

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \qquad \text{and} \qquad \mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}.$$

We multiply the first equation on the left by $\mathbf{S}^T$ and insert $\mathbf{S}^T\mathbf{S} = \mathbf{I}$ between $\mathbf{A}$ and $\mathbf{x}$. Then we get

$$(\mathbf{S^T A S})(\mathbf{S^T x}) = \lambda \mathbf{S^T x}, \tag{12.4}$$

which is the same as

$$\mathbf{B}\left(\mathbf{S^T x}\right) = \lambda\left(\mathbf{S^T x}\right).$$

The variable $\lambda$ is an eigenvalue of $\mathbf{B}$ as well, but with eigenvector $\mathbf{S^T x}$.

The basic philosophy is to

- either apply subsequent similarity transformations so that

$$\mathbf{S_N^T} \dots \mathbf{S_1^T A S_1} \dots \mathbf{S_N} = \mathbf{D}, \tag{12.5}$$

- or apply subsequent similarity transformations so that $\mathbf{A}$ becomes tridiagonal. Thereafter, techniques for obtaining eigenvalues from tridiagonal matrices can be used.

Let us look at the first method, better known as Jacobi's method or Given's rotations.

## *12.4 Jacobi's method*

Consider an $(n \times n)$ orthogonal transformation matrix

$$\mathbf{S} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & \dots \\ 0 & 0 & \dots & cos\theta & 0 & \dots & 0 & sin\theta \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & \dots \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & -sin\theta & \dots & \dots & 0 & cos\theta \end{pmatrix}$$

with property $\mathbf{S^T} = \mathbf{S^{-1}}$. It performs a plane rotation around an angle $\theta$ in the Euclidean $n-$dimensional space. It means that its matrix elements that differ from zero are given by

$$s_{kk} = s_{ll} = cos\theta, s_{kl} = -s_{lk} = -sin\theta, s_{ii} = -s_{ii} = 1 \quad i \neq k \quad i \neq l,$$

A similarity transformation

$$\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S},$$

results in

$$\begin{aligned} b_{ik} &= a_{ik}cos\theta - a_{il}sin\theta, i \neq k, i \neq l \\ b_{il} &= a_{il}cos\theta + a_{ik}sin\theta, i \neq k, i \neq l \\ b_{kk} &= a_{kk}cos^2\theta - 2a_{kl}cos\theta sin\theta + a_{ll}sin^2\theta \\ b_{ll} &= a_{ll}cos^2\theta + 2a_{kl}cos\theta sin\theta + a_{kk}sin^2\theta \\ b_{kl} &= (a_{kk} - a_{ll})cos\theta sin\theta + a_{kl}(cos^2\theta - sin^2\theta) \end{aligned}$$

The angle $\theta$ is arbitrary. The recipe is to choose $\theta$ so that all non-diagonal matrix elements $b_{kl}$ become zero.

The algorithm is then quite simple. We perform a number of iterations untill the sum over the squared non-diagonal matrix elements are less than a prefixed test (ideally equal zero). The algorithm is more or less foolproof for all real symmetric matrices, but becomes much slower than methods based on tridiagonalization for large matrices.

The main idea is thus to reduce systematically the norm of the off-diagonal matrix elements of a matrix $\mathbf{A}$

$$\mathrm{off}(\mathbf{A}) = \sqrt{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} a_{ij}^2}.$$

To demonstrate the algorithm, we consider the simple $2 \times 2$ similarity transformation of the full matrix. The matrix is symmetric, we single out $1 \leq k < l \leq n$ and use the abbreviations $c = \cos\theta$ and $s = \sin\theta$ to obtain

$$\begin{pmatrix} b_{kk} & 0 \\ 0 & b_{ll} \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}.$$

We require that the non-diagonal matrix elements $b_{kl} = b_{lk} = 0$, implying that

$$a_{kl}(c^2 - s^2) + (a_{kk} - a_{ll})cs = b_{kl} = 0.$$

If $a_{kl} = 0$ one sees immediately that $\cos\theta = 1$ and $\sin\theta = 0$.

The Frobenius norm of an orthogonal transformation is always preserved. The Frobenius norm is defined as

$$||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2}.$$

This means that for our $2 \times 2$ case we have

$$2a_{kl}^2 + a_{kk}^2 + a_{ll}^2 = b_{kk}^2 + b_{ll}^2,$$

which leads to

$$\mathrm{off}(\mathbf{B})^2 = ||\mathbf{B}||_F^2 - \sum_{i=1}^{n} b_{ii}^2 = \mathrm{off}(\mathbf{A})^2 - 2a_{kl}^2,$$

since

$$||\mathbf{B}||_F^2 - \sum_{i=1}^{n} b_{ii}^2 = ||\mathbf{A}||_F^2 - \sum_{i=1}^{n} a_{ii}^2 + (a_{kk}^2 + a_{ll}^2 - b_{kk}^2 - b_{ll}^2).$$

This results means that the matrix $\mathbf{A}$ moves closer to diagonal form for each transformation.

Defining the quantities $\tan\theta = t = s/c$ and

$$\tau = \frac{a_{kk} - a_{ll}}{2a_{kl}},$$

we obtain the quadratic equation

$$t^2 + 2\tau t - 1 = 0,$$

resulting in

$$t = -\tau \pm \sqrt{1 + \tau^2},$$

and $c$ and $s$ are easily obtained via

$$c = \frac{1}{\sqrt{1+t^2}},$$

and $s = tc$. Choosing $t$ to be the smaller of the roots ensures that $|\theta| \leq \pi/4$ and has the effect of minimizing the difference between the matrices $\mathbf{B}$ and $\mathbf{A}$ since

$$||\mathbf{B}-\mathbf{A}||_F^2 = 4(1-c)\sum_{i=1,i\neq k,l}^{n}(a_{ik}^2+a_{il}^2)+\frac{2a_{kk}^2}{c^2}.$$

To implement the Jacobi algorithm we can proceed as follows

- Choose a tolerance $\epsilon$, making it a small number, typically $10^{-8}$ or smaller.
- Setup a **while**-test where one compares the norm of the newly computed off-diagonal matrix elements
$$\mathrm{off}(\mathbf{A}) = \sqrt{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}a_{ij}^2} > \epsilon.$$
- Now choose the matrix elements $a_{kl}$ so that we have those with largest value, that is $|a_{kl}| = \max_{i\neq j}|a_{ij}|$.
- Compute thereafter $\tau = (a_{ll}-a_{kk})/2a_{kl}$, $\tan\theta$, $\cos\theta$ and $\sin\theta$.
- Compute thereafter the similarity transformation for this set of values $(k,l)$, obtaining the new matrix $\mathbf{B} = \mathbf{S}(k,l,\theta)^T\mathbf{A}\mathbf{S}(k,l,\theta)$.
- Compute the new norm of the off-diagonal matrix elements and continue till you have satisfied $\mathrm{off}(\mathbf{B}) \leq \epsilon$

The convergence rate of the Jacobi method is however poor, one needs typically $3n^2-5n^2$ rotations and each rotation requires $4n$ operations, resulting in a total of $12n^3-20n^3$ operations in order to zero out non-diagonal matrix elements. Although the classical Jacobi algorithm performs badly compared with methods based on tridiagonalization, it is easy to parallelize. We discuss how to parallelize this method in the next subsection.

### *12.4.1 Parallel Jacobi algorithm*

In preparation for Fall 2008.

## *12.5 Diagonalization through the Householder's method for tridiagonalization*

In this case the diagonalization is performed in two steps: First, the matrix is transformed into tridiagonal form by the Householder similarity transformation. Secondly, the tridiagonal matrix is then diagonalized. The reason for this two-step process is that diagonalising a tridiagonal matrix is computational much

faster then the corresponding diagonalization of a general symmetric matrix. Let us discuss the two steps in more detail.

### *12.5.1 The Householder's method for tridiagonalization*

The first step consists in finding an orthogonal matrix $\mathbf{S}$ which is the product of $(n-2)$ orthogonal matrices

$$\mathbf{S} = \mathbf{S}_1\mathbf{S}_2\dots\mathbf{S}_{n-2},$$

each of which successively transforms one row and one column of $\mathbf{A}$ into the required tridiagonal form. Only $n-2$ transformations are required, since the last two elements are already in tridiagonal form. In order to determine each $\mathbf{S_i}$ let us see what happens after the first multiplication, namely,

$$\mathbf{S}_1^T\mathbf{A}\mathbf{S}_1 = \begin{pmatrix} a_{11} & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & a'_{22} & a'_{23} & \dots & \dots & \dots & a'_{2n} \\ 0 & a'_{32} & a'_{33} & \dots & \dots & \dots & a'_{3n} \\ 0 & \dots & \dots & \dots & \dots & \dots & \\ 0 & a'_{n2} & a'_{n3} & \dots & \dots & \dots & a'_{nn} \end{pmatrix}$$

where the primed quantities represent a matrix $\mathbf{A}'$ of dimension $n-1$ which will subsequentely be transformed by $\mathbf{S_2}$. The factor $e_1$ is a possibly non-vanishing element. The next transformation produced by $\mathbf{S_2}$ has the same effect as $\mathbf{S_1}$ but now on the submatirx $\mathbf{A}^{'}$ only

$$\left(\mathbf{S}_1\mathbf{S}_2\right)^T\mathbf{A}\mathbf{S}_1\mathbf{S}_2 = \begin{pmatrix} a_{11} & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & a'_{22} & e_2 & 0 & \dots & \dots & 0 \\ 0 & e_2 & a''_{33} & \dots & \dots & \dots & a''_{3n} \\ 0 & \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & a''_{n3} & \dots & \dots & \dots & a''_{nn} \end{pmatrix}$$

Note that the effective size of the matrix on which we apply the transformation reduces for every new step. In the previous Jacobi method each similarity transformation is in principle performed on the full size of the original matrix.

After a series of such transformations, we end with a set of diagonal matrix elements

$$a_{11}, a'_{22}, a''_{33}\dots a^{n-1}_{nn},$$

and off-diagonal matrix elements

$$e_1, e_2, e_3, \dots, e_{n-1}.$$

The resulting matrix reads

$$\mathbf{S}^T\mathbf{A}\mathbf{S} = \begin{pmatrix} a_{11} & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & a'_{22} & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & a''_{33} & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & a^{(n-1)}_{n-2} & e_{n-1} \\ 0 & \dots & \dots & \dots & \dots & e_{n-1} & a^{(n-1)}_{n-1} \end{pmatrix}.$$

It remains to find a recipe for determining the transformation $\mathbf{S}_n$. We illustrate the method for $\mathbf{S}_1$ which we assume takes the form

$$\mathbf{S_1} = \begin{pmatrix} 1 & \mathbf{0^T} \\ \mathbf{0} & \mathbf{P} \end{pmatrix},$$

with $\mathbf{0^T}$ being a zero row vector, $\mathbf{0^T} = \{0, 0, \cdots\}$ of dimension $(n-1)$. The matrix $\mathbf{P}$ is symmetric with dimension $((n-1) \times (n-1))$ satisfying $\mathbf{P}^2 = \mathbf{I}$ and $\mathbf{P}^T = \mathbf{P}$. A possible choice which fullfils the latter two requirements is

$$\mathbf{P} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^T,$$

where $\mathbf{I}$ is the $(n-1)$ unity matrix and $\mathbf{u}$ is an $n-1$ column vector with norm $\mathbf{u}^T\mathbf{u}$(inner product. Note that $\mathbf{u}\mathbf{u}^T$ is an outer product giving a dimension $((n-1) \times (n-1))$. Each matrix element of $\mathbf{P}$ then reads

$$P_{ij} = \delta_{ij} - 2u_i u_j,$$

where $i$ and $j$ range from 1 to $n-1$. Applying the transformation $\mathbf{S}_1$ results in

$$\mathbf{S}_1^T\mathbf{A}\mathbf{S}_1 = \begin{pmatrix} a_{11} & (\mathbf{P}\mathbf{v})^T \\ \mathbf{P}\mathbf{v} & \mathbf{A}' \end{pmatrix},$$

where $\mathbf{v^T} = \{a_{21}, a_{31}, \cdots, a_{n1}\}$ and $\mathbf{P}$ must satisfy $(\mathbf{P}\mathbf{v})^T = \{k, 0, 0, \cdots\}$. Then

$$\mathbf{P}\mathbf{v} = \mathbf{v} - 2\mathbf{u}(\mathbf{u}^T\mathbf{v}) = k\mathbf{e}, \tag{12.6}$$

with $\mathbf{e^T} = \{1, 0, 0, \ldots 0\}$. Solving the latter equation gives us $\mathbf{u}$ and thus the needed transformation $\mathbf{P}$. We do first however need to compute the scalar $k$ by taking the scalar product of the last equation with its transpose and using the fact that $\mathbf{P}^2 = \mathbf{I}$. We get then

$$(\mathbf{P}\mathbf{v})^T\mathbf{P}\mathbf{v} = k^2 = \mathbf{v}^T\mathbf{v} = |v|^2 = \sum_{i=2}^{n} a_{i1}^2,$$

which determines the constant $k = \pm v$. Now we can rewrite Eq. (12.6) as

$$\mathbf{v} - k\mathbf{e} = 2\mathbf{u}(\mathbf{u}^T\mathbf{v}),$$

and taking the scalar product of this equation with itself and obtain

$$2(\mathbf{u}^T\mathbf{v})^2 = (v^2 \pm a_{21}v), \tag{12.7}$$

which finally determines

$$\mathbf{u} = \frac{\mathbf{v} - k\mathbf{e}}{2(\mathbf{u}^T\mathbf{v})}.$$

In solving Eq. (12.7) great care has to be exercised so as to choose those values which make the right-hand largest in order to avoid loss of numerical precision. The above steps are then repeated for every transformations till we have a tridiagonal matrix suitable for obtaining the eigenvalues.

### *12.5.2 Diagonalization of a tridiagonal matrix*

The matrix is now transformed into tridiagonal form and the last step is to transform it into a diagonal matrix giving the eigenvalues on the diagonal.

Before we discuss the algorithms, we note that the eigenvalues of a tridiagonal matrix can be obtained using the characteristic polynomial

$$P(\lambda) = det(\lambda\mathbf{I} - \mathbf{A}) = \prod_{i=1}^{n} (\lambda_i - \lambda),$$

which rewritten in matrix form reads

$$P(\lambda) = \begin{pmatrix} d_1-\lambda & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_2-\lambda & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_3-\lambda & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & d_{N_{\rm step}-2}-\lambda & e_{N_{\rm step}-1} \\ 0 & \dots & \dots & \dots & \dots & e_{N_{\rm step}-1} & d_{N_{\rm step}-1}-\lambda \end{pmatrix}$$

We can solve this equation in a recursive manner. We let $P_k(\lambda)$ be the value of $k$ subdeterminant of the above matrix of dimension $n \times n$. The polynomial $P_k(\lambda)$ is clearly a polynomial of degree $k$. Starting with $P_1(\lambda)$ we have $P_1(\lambda) = d_1 - \lambda$. The next polynomial reads $P_2(\lambda) = (d_2 - \lambda)P_1(\lambda) - e_1^2$. By expanding the determinant for $P_k(\lambda)$ in terms of the minors of the $n$th column we arrive at the recursion relation

$$P_k(\lambda) = (d_k - \lambda)P_{k-1}(\lambda) - e_{k-1}^2 P_{k-2}(\lambda).$$

Together with the starting values $P_1(\lambda)$ and $P_2(\lambda)$ and good root searching methods we arrive at an efficient computational scheme for finding the roots of $P_n(\lambda)$. However, for large matrices this algorithm is rather inefficient and time-consuming.

The programs which performs these transformations are matrix $\mathbf{A} \longrightarrow$ tridiagonal matrix $\longrightarrow$ diagonal matrix

| | |
|---|---|
| C: | void trd2(double $**$a, int n, double d[], double e[]) |
| | void tqli(double d[], double[], int n, double $**$z) |
| Fortran: | CALL tred2(a, n, d, e) |
| | CALL tqli(d, e, n, z) |

The last step through the function *tqli()* involves several technical details. Let us describe the basic idea in terms of a four-dimensional example. For more details, see Ref. [25], in particular chapters seven and eight.

The current tridiagonal matrix takes the form

$$\mathbf{A} = \begin{pmatrix} d_1 & e_1 & 0 & 0 \\ e_1 & d_2 & e_2 & 0 \\ 0 & e_2 & d_3 & e_3 \\ 0 & 0 & e_3 & d_4 \end{pmatrix}.$$

As a first observation, if any of the elements $e_i$ are zero the matrix can be separated into smaller pieces before diagonalization. Specifically, if $e_1 = 0$ then $d_1$ is an eigenvalue. Thus, let us introduce a transformation $\mathbf{S_1}$

$$\mathbf{S_1} = \begin{pmatrix} \cos\theta & 0 & 0 & \sin\theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\sin\theta & 0 & 0 & \cos\theta \end{pmatrix}$$

Then the similarity transformation

$$\mathbf{S_1^T A S_1} = \mathbf{A}' = \begin{pmatrix} d_1' & e_1' & 0 & 0 \\ e_1' & d_2 & e_2 & 0 \\ 0 & e_2 & d_3 & e'3 \\ 0 & 0 & e_3' & d_4' \end{pmatrix}$$

produces a matrix where the primed elements in $\mathbf{A}'$ have been changed by the transformation whereas the unprimed elements are unchanged. If we now choose $\theta$ to give the element $a_{21}^{'} = e^{'} = 0$ then we have the first eigenvalue $= a_{11}^{'} = d_1^{'}$.

This procedure can be continued on the remaining three-dimensional submatrix for the next eigenvalue. Thus after four transformations we have the wanted diagonal form.

## *12.6 Schrödinger's equation through diagonalization*

Instead of solving the Schrödinger equation as a differential equation, we will solve it through diagonalization of a large matrix. However, in both cases we need to deal with a problem with boundary conditions, viz., the wave function goes to zero at the endpoints.

To solve the Schrödinger equation as a matrix diagonalization problem, let us study the radial part of the Schrödinger equation. The radial part of the wave function, $R(r)$, is a solution to

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}-\frac{l(l+1)}{r^2}\right)R(r)+V(r)R(r)=ER(r).$$

Then we substitute $R(r) = (1/r)u(r)$ and obtain

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r)+\left(V(r)+\frac{l(l+1)}{r^2}\frac{\hbar^2}{2m}\right)u(r)=Eu(r).$$

We introduce a dimensionless variable $\rho = (1/\alpha)r$ where $\alpha$ is a constant with dimension length and get

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(r)+\left(V(\rho)+\frac{l(l+1)}{\rho^2}\frac{\hbar^2}{2m\alpha^2}\right)u(\rho)=Eu(\rho).$$

In the example below, we will replace the latter equation with that for the one-dimensional harmonic oscillator. Note however that the procedure which we give below applies equally well to the case of e.g., the hydrogen atom. We replace $\rho$ with $x$, take away the centrifugal barrier term and set the potential equal to

$$V(x)=\frac{1}{2}kx^2,$$

with $k$ being a constant. In our solution we will use units so that $k = \hbar = m = \alpha = 1$ and the Schrödinger equation for the one-dimensional harmonic oscillator becomes

$$-\frac{d^2}{dx^2}u(x)+x^2u(x)=2Eu(x).$$

Let us now see how we can rewrite this equation as a matrix eigenvalue problem. First we need to compute the second derivative. We use here the following expression for the second derivative of a function $f$

$$f''=\frac{f(x+h)-2f(x)+f(x-h)}{h^2}+O(h^2), \tag{12.8}$$

where $h$ is our step. Next we define minimum and maximum values for the variable $x$, $R_{\rm min}$ and $R_{\rm max}$, respectively. With a given number of steps, $N_{\rm step}$, we then define the step $h$ as

$$h=\frac{R_{\rm max}-R_{\rm min}}{N_{\rm step}}.$$

If we now define an arbitrary value of $x$ as

$$x_i = R_{\min} + ih \qquad i = 1, 2, \dots, N_{\rm step} - 1$$

we can rewrite the Schrödinger equation for $x_i$ as

$$-\frac{u(x_k+h) - 2u(x_k) + u(x_k-h)}{h^2} + x_k^2 u(x_k) = 2Eu(x_k),$$

or in a more compact way

$$-\frac{u_{k+1} - 2u_k + u_{k-1}}{h^2} + x_k^2 u_k = -\frac{u_{k+1} - 2u_k + u_{k-1}}{h^2} + V_k u_k = 2Eu_k,$$

where $u_k = u(x_k)$, $u_{k\pm1} = u(x_k \pm h)$ and $V_k = x_k^2$, the given potential. Let us see how this recipe may lead to a matrix reformulation of the Schrödinger equation. Define first the diagonal matrix element

$$d_k = \frac{2}{h^2} + V_k,$$

and the non-diagonal matrix element

$$e_k = -\frac{1}{h^2}.$$

In this case the non-diagonal matrix elements are given by a mere constant. *All non-diagonal matrix elements are equal*. With these definitions the Schrödinger equation takes the following form

$$d_k u_k + e_{k-1} u_{k-1} + e_{k+1} u_{k+1} = 2Eu_k,$$

where $u_k$ is unknown. Since we have $N_{\rm step} - 1$ values of $k$ we can write the latter equation as a matrix eigenvalue problem

$$\begin{pmatrix} d_1 & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_2 & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_3 & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & d_{N_{\rm step}-2} & e_{N_{\rm step}-1} \\ 0 & \dots & \dots & \dots & \dots & e_{N_{\rm step}-1} & d_{N_{\rm step}-1} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ \dots \\ \dots \\ u_{N_{\rm step}-1} \end{pmatrix} = 2E \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ \dots \\ \dots \\ u_{N_{\rm step}-1} \end{pmatrix} \tag{12.9}$$

or if we wish to be more detailed, we can write the tridiagonal matrix as

$$\begin{pmatrix} \frac{2}{h^2} + V_1 & -\frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2} + V_2 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2} + V_3 & -\frac{1}{h^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \frac{2}{h^2} + V_{N_{\rm step}-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{N_{\rm step}-1} \end{pmatrix} \tag{12.10}$$

This is a matrix problem with a tridiagonal matrix of dimension $N_{\rm step} - 1 \times N_{\rm step} - 1$ and will thus yield $N_{\rm step} - 1$ eigenvalues. It is important to notice that we do not set up a matrix of dimension $N_{\rm step} \times N_{\rm step}$ since we can fix the value of the wave function at $k = N_{\rm step}$. Similarly, we know the wave function at the other end point, that is for $x_0$.

The above equation represents an alternative to the numerical solution of the differential equation for the Schrödinger equation discussed in chapter 14.

The eigenvalues of the harmonic oscillator in one dimension are well known. In our case, with all constants set equal to 1, we have

$$E_n = n + \frac{1}{2},$$

with the ground state being $E_0 = 1/2$. Note however that we have rewritten the Schrödinger equation so that a constant 2 stands in front of the energy. Our program will then yield twice the value, that is we will obtain the eigenvalues $1, 3, 5, 7.. \ldots$ .

In the next subsection we will try to delineate how to solve the above equation. A program listing is also included.

### *12.6.1 Numerical solution of the Schrödinger equation by diagonalization*

The algorithm for solving Eq. (12.9) may take the following form

- Define values for $N_{\rm step}$, $R_{\rm min}$ and $R_{\rm max}$. These values define in turn the step size $h$. Typical values for $R_{\rm max}$ and $R_{\rm min}$ could be 10 and $-10$ respectively for the lowest-lying states. The number of mesh points $N_{\rm step}$ could be in the range 100 to some thousands. You can check the stability of the results as functions of $N_{\rm step} - 1$ and $R_{\rm max}$ and $R_{\rm min}$ against the exact solutions.

- Construct then two one-dimensional arrays which contain all values of $x_k$ and the potential $V_k$. For the latter it can be convenient to write a small function which sets up the potential as function of $x_k$. For the three-dimensional case you may also need to include the centrifugal potential. The dimension of these two arrays should go from 0 up to $N_{\rm step}$.

- Construct thereafter the one-dimensional vectors $d$ and $e$, where $d$ stands for the diagonal matrix elements and $e$ the non-diagonal ones. Note that the dimension of these two arrays runs from 1 up to $N_{\rm step} - 1$, since we know the wave function $u$ at both ends of the chosen grid.

- We are now ready to obtain the eigenvalues by calling the function *tqli* which can be found on the web page of the course. Calling *tqli*, you have to transfer the matrices $d$ and $e$, their dimension $n = N_{\rm step} - 1$ and a matrix $z$ of dimension $N_{\rm step} - 1 \times N_{\rm step} - 1$ which returns the eigenfunctions. On return, the array $d$ contains the eigenvalues. If $z$ is given as the unity matrix on input, it returns the eigenvectors. For a given eigenvalue $k$, the eigenvector is given by the column $k$ in $z$, that is z[][k] in C, or z(:,k) in Fortran 90.

- TQLI does however not return an ordered sequence of eigenvalues. You may then need to sort them as e.g., an ascending series of numbers. The program we provide includes a sorting function as well.

- Finally, you may perhaps need to plot the eigenfunctions as well, or calculate some other expectation values. Or, you would like to compare the eigenfunctions with the analytical answers for the harmonic oscillator or the hydrogen atom. We provide a function *plot* which has as input one eigenvalue chosen from the output of *tqli*. This function gives you a normalized wave function $u$ where the norm is calculated as

$$\int_{R_{\rm min}}^{R_{\rm max}} |u(x)|^2 \, dx \rightarrow h \sum_{i=0}^{N_{\rm step}} u_i^2 = 1,$$

and we have used the trapezoidal rule for integration discussed in chapter 7.

### *12.6.2 Program example and results for the one-dimensional harmonic oscillator*

We present here a program example which encodes the above algorithm. The corresponding Fortran 90/95 program is at programs/chapter12/program1.f90.

http://folk.uio.no/mhjensen/fys3150/2005/programs/chapter12/program1.cpp

```
/*
  Solves the one-particle Schrodinger equation
  for a potential specified in function
  potential(). This example is for the harmonic oscillator
*/
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "lib.h"
using namespace  std;
// output file as global variable
ofstream ofile;

// function declarations

void initialise(double&, double&, int&, int&) ;
double potential(double);
int comp(const double *, const double *);
void output(double, double, int, double *);

int main(int argc, char* argv[])
{
  int         i, j, max_step, orb_l;
  double      r_min, r_max, step, const_1, const_2, orb_factor,
              *e, *d, *w, *r, **z;
  char *outfilename;
  // Read in output file, abort if there are too few command-line arguments
  if( argc <= 1 ){
    cout << "Bad Usage: " << argv[0] <<
      " read also output file on same line" << endl;
    exit(1);
  }
  else{
    outfilename=argv[1];
  }
  ofile.open(outfilename);
  //    Read in data
  initialise(r_min, r_max, orb_l, max_step);
  // initialise constants
  step     = (r_max - r_min) / max_step;
  const_2 = -1.0 / (step * step);
  const_1 =  - 2.0 * const_2;
  orb_factor = orb_l * (orb_l + 1);

  // local memory for r and the potential w[r]
  r = new double[max_step + 1];
```

```
  w = new double[max_step + 1];
  for(i = 0; i <= max_step; i++) {
    r[i] = r_min + i * step;
    w[i] = potential(r[i]) + orb_factor / (r[i] * r[i]);
  }
  // local memory for the diagonalization process
  d = new double[max_step];     // diagonal elements
  e = new double[max_step];     // tridiagonal off-diagonal elements
  z = (double **) matrix(max_step, max_step, sizeof(double));
  for(i = 0; i < max_step; i++) {
    d[i]    = const_1 + w[i + 1];
    e[i]    = const_2;
    z[i][i] = 1.0;
    for(j = i + 1; j < max_step; j++)  {
      z[i][j] = 0.0;
    }
  }
  // diagonalize and obtain eigenvalues
  tqli(d, e, max_step - 1, z);
  // Sort eigenvalues as an ascending series
  qsort(d,(UL) max_step - 1,sizeof(double),
        (int(*)(const void *,const void *))comp);
  // send results to ouput file
  output(r_min , r_max, max_step, d);
  delete [] r; delete [] w; delete [] e; delete [] d;
  free_matrix((void **) z); // free memory
  ofile.close();  // close output file
  return 0;
} // End: function main()

/*
  The function potential()
  calculates and return the value of the
  potential for a given argument x.
  The potential here is for the 1-dim harmonic oscillator
*/

double potential(double x)
{
   return x*x;

} // End: function potential()

/*
  The function   int comp()
  is a utility function for the library function qsort()
  to sort double numbers after increasing values.
*/

int comp(const double *val_1, const double *val_2)
{
  if((*val_1) <= (*val_2))        return -1;
  else  if((*val_1) > (*val_2))   return +1;
```

```
  else                          return  0;
} // End: function comp()

// read in min  and max radius, number of mesh points and l
void initialise(double& r_min, double& r_max, int& orb_l, int& max_step)
{
  cout << "Min vakues of R = ";
  cin >> r_min;
  cout << "Max value of R = ";
  cin >> r_max;
  cout << "Orbital momentum = ";
  cin >> orb_l;
  cout << "Number of steps = ";
  cin >> max_step;
}  // end of function initialise
// output of results
void output(double r_min , double r_max, int max_step, double *d)
{
  int i;
  ofile << "RESULTS:" << endl;
  ofile << setiosflags(ios::showpoint | ios::uppercase);
  ofile <<"R_min = " << setw(15) << setprecision(8) << r_min << endl;
  ofile <<"R_max = " << setw(15) << setprecision(8) << r_max << endl;
  ofile <<"Number of steps = " << setw(15) << max_step << endl;
  ofile << "Five lowest eigenvalues:" << endl;
  for(i = 0; i < 5; i++) {
    ofile << setw(15) << setprecision(8) << d[i] << endl;
  }
}  // end of function output
```

There are several features to be noted in this program.

The main program calls the function *initialise*, which reads in the minimum and maximum values of $r$, the number of steps and the orbital angular momentum $l$. Thereafter we allocate place for the vectors containing $r$ and the potential, given by the variables $r[i]$ and $w[i]$, respectively. We also set up the vectors $d[i]$ and $e[i]$ containing the diagonal and non-diagonal matrix elements. Calling the function $tqli$ we obtain in turn the unsorted eigenvalues. The latter are sorted by the intrinsic C-function $qsort$.

The calculaton of the wave function for the lowest eigenvalue is done in the function $plot$, while all output of the calculations is directed to the fuction $output$.

The included table exhibits the precision achieved as function of the number of mesh points $N$. The exact values are $1, 3, 5, 7, 9$.

Table 12.1: Five lowest eigenvalues as functions of the number of mesh points $N$ with $r_{\min} = -10$ and $r_{\max} = 10$.

| $N$ | $E_0$ | $E_1$ | $E_2$ | $E_3$ | $E_4$ |
|---|---|---|---|---|---|
| 50 | 9.898985E-01 | 2.949052E+00 | 4.866223E+00 | 6.739916E+00 | 8.568442E+00 |
| 100 | 9.974893E-01 | 2.987442E+00 | 4.967277E+00 | 6.936913E+00 | 8.896282E+00 |
| 200 | 9.993715E-01 | 2.996864E+00 | 4.991877E+00 | 6.984335E+00 | 8.974301E+00 |
| 400 | 9.998464E-01 | 2.999219E+00 | 4.997976E+00 | 6.996094E+00 | 8.993599E+00 |
| 1000 | 1.000053E+00 | 2.999917E+00 | 4.999723E+00 | 6.999353E+00 | 8.999016E+00 |

The agreement with the exact solution improves with increasing numbers of mesh points. However, the agreement for the excited states is by no means impressive. Moreover, as the dimensionality increases, the time consumption increases dramatically. Matrix diagonalization scales typically as $\approx N^3$. In addition, there is a maximum size of a matrix which can be stored in RAM.

The obvious question which then arises is whether this scheme is nothing but a mere example of matrix diagonalization, with few practical applications of interest. In chapter 6, where we deal with interpolation and extrapolation, we discussed also a called Richardson's deferred extrapolation scheme. Applied to this particualr case, the philosophy of this scheme would be to diagonalize the above matrix for a set of values of $N$ and thereby the step length $h$. Thereafter, an extrapolation is made to $h \rightarrow 0$. The obtained eigenvalues agree then with a remarkable precision with the exact solution. The algorithm is then as follows

- Perform a series of diagonalizations of the matrix in Eq. (12.10 ) for different values of the step size $h$. We obtain then a series of eigenvalues $E(h/2^k)$ with $k = 0, 1, 2, \ldots$. That will give us an array of 'x-values' $h, h/2, h/4, \ldots$ and an array of 'y-values' $E(h), E(h/2), E(h/4), \ldots$. Note that you will have such a set for each eigenvalue.
- Use these values to perform an extrapolation calling e.g., the function POLINT with the point where we wish to extrapolate to given by $h = 0$.
- End the iteration over $k$ when the error returned by POLINT is smaller than a fixed test.

The results for the 10 lowest-lying eigenstates for the one-dimensional harmonic oscillator are listed below after just 3 iterations, i.e., the step size has been reduced to $h/8$ only. The exact results are $1, 3, 5, \ldots, 19$ and we see that the agreement is just excellent for the extrapolated results. The results after diagonalization differ already at the fourth-fifth digit.

Parts of a Fortran90 program which includes Richardson's extrapolation scheme is included here. It performs five diagonalizations and establishes results for various step lengths and interpolates using the function POLINT.

```
!  start loop over interpolations, here we set max interpolations to 5
      DO interpol=1, 5
         IF ( interpol == 1) THEN
            max_step=start_step
         ELSE
            max_step=(interpol-1)*2*start_step
         ENDIF
         n=max_step-1
         ALLOCATE ( e(n) , d(n) )
         ALLOCATE ( w(0:max_step), r(0:max_step))
         d=0. ; e =0.
!  define the step size
         step=(rmax-rmin)/FLOAT(max_step)
         hh(interpol)=step*step
!  define constants for the matrix to be diagonalized
         const1=2./(step*step)
```

Table 12.2: Result for numerically calculated eigenvalues of the one-dimensional harmonic oscillator after three iterations starting with a matrix of size $100 \times 100$ and ending with a matrix of dimension $800 \times 800$. These four values are then used to extrapolate the 10 lowest-lying eigenvalues to $h = 0$.. The values of $x$ span from $-10$ to 10, that means that the starting step was $h = 20/100 = 0.2$. We list here only the results after three iterations. The error test was set equal $10^{-6}$.

| Extrapolation | Diagonalization | Error |
|---|---|---|
| 0.100000D+01 | 0.999931D+00 | 0.206825D-10 |
| 0.300000D+01 | 0.299965D+01 | 0.312617D-09 |
| 0.500000D+01 | 0.499910D+01 | 0.174602D-08 |
| 0.700000D+01 | 0.699826D+01 | 0.605671D-08 |
| 0.900000D+01 | 0.899715D+01 | 0.159170D-07 |
| 0.110000D+02 | 0.109958D+02 | 0.349902D-07 |
| 0.130000D+02 | 0.129941D+02 | 0.679884D-07 |
| 0.150000D+02 | 0.149921D+02 | 0.120735D-06 |
| 0.170000D+02 | 0.169899D+02 | 0.200229D-06 |
| 0.190000D+02 | 0.189874D+02 | 0.314718D-06 |

```
          const2=-1./(step*step)
!      set up r, the distance from the nucleus and the function w for energy
   =0
!      w corresponds then to the potential
!      values at
          DO i=0, max_step
             r(i) = rmin+i*step
             w(i) = potential(r(i))
          ENDDO
!      setup the diagonal d and the non-diagonal part e  of
!      the  tridiagonal matrix matrix to be diagonalized
          d(1:n)=const1+w(1:n)   ;   e(1:n)=const2
!   allocate space for eigenvector info
          ALLOCATE ( z(n,n) )
!   obtain the eigenvalues
          CALL tqli(d,e,n,z)
!   sort eigenvalues as an ascending series
          CALL eigenvalue_sort(d,n)
          DEALLOCATE (z)
          err1=0.
!   the interpolation part starts here
          DO l=1,20
             err2=0.
             value(interpol,l)=d(l)
             inp=d(l)
             IF ( interpol > 1 ) THEN
                CALL polint(hh,value(:,l),interpol,0.d0 ,inp,err2)
                err1=MAX(err1,err2)
                WRITE(6,'(D12.6,2X,D12.6,2X,D12.6)') inp, d(l), err1
             ELSE
```

```
            WRITE(6,'(D12.6,2X,D12.6,2X,D12.6)') d(l), d(l), err1
         ENDIF
      ENDDO
      DEALLOCATE ( w, r, d, e)
   ENDDO
```

## *12.7 Discussion of BLAS and LAPACK functionalities*

In preparation for fall 2008.

## *12.8 Physics projects: Bound states in momentum space*

In this problem we will solve the Schrödinger equation in momentum space for the deuteron. The deuteron has only one bound state at an energy of $-2.223$ MeV. The ground state is given by the quantum numbers $l = 0$, $S = 1$ and $J = 1$, with $l$, $S$, and $J$ the relative orbital momentum, the total spin and the total angular momentum, respectively. These quantum numbers are the sum of the single-particle quantum numbers. The deuteron consists of a proton and neutron, with mass (average) of 938 MeV. The electron is not included in the solution of the Schrödinger equation since its mass is much smaller than those of the proton and the neutron. We can neglect it here. This means that e.g., the total spin $S$ is the sum of the spin of the neutron and the proton. The above three quantum numbers can be summarized in the spectroscopic notation $^{2S+1}l_J = ^3S_1$, where $S$ represents $l = 0$ here. It is a spin triplet state. The spin wave function is thus symmetric. This also applies to the spatial part, since $l = 0$. To obtain a totally anti-symmetric wave function we need to introduce another quantum number, namely isospin. The deuteron has isospin $T = 0$, which gives a final wave function which is anti-symmetric.

We are going to use a simplified model for the interaction between the neutron and the proton. We will assume that it goes like

$$V(r) = V_0 \frac{\exp(-\mu r)}{r}, \tag{12.11}$$

where $\mu$ has units m$^{-1}$ and serves to screen the potential for large values of $r$. The variable $r$ is the distance between the proton and the neutron. It is the relative coordinate, the centre of mass is not needed in this problem. The nucleon-nucleon interaction has a finite and small range, typically of some few fm[1]. We will in this exercise set $\mu = 0.7$ fm$^{-1}$. It is then proportional to the mass of the pion. The pion is the lightest meson, and sets therefore the range of the nucleon-nucleon interaction. For low-energy problems we can describe the nucleon-nucleon interaction through meson-exchange models, and the pion is the lightest known meson, with mass of approximately 138 MeV.

Since we are going to solve the Schrödinger equation in momentum, we need the Fourier transform of $V(r)$. In a partial wave basis for $l = 0$ it becomes

$$V(k',k) = \frac{V_0}{4k'k} ln\left(\frac{(k'+k)^2+\mu^2}{(k'-k)^2+\mu^2}\right), \tag{12.12}$$

where $k'$ and $k$ are the relative momenta for the proton and neutron system.

[1] 1 fm = $10^{-15}$ m.

For relative coordinates, the Schrödinger equation in momentum space becomes

$$\frac{k^2}{m}\psi(k) + \frac{2}{\pi}\int_0^\infty dpp^2V(k,p)\psi(p) = E\psi(k). \tag{12.13}$$

Here we have used units $\hbar = c = 1$. This means that $k$ has dimension energy. This is the equation we are going to solve, with eigenvalue $E$ and eigenfunction $\psi(k)$. The approach to solve this equations goes then as follows.

First we need to evaluate the integral over $p$ using e.g., gaussian quadrature. This means that we rewrite an integral like

$$\int_a^b f(x)dx \approx \sum_{i=1}^N \omega_i f(x_i),$$

where we have fixed $N$ lattice points through the corresponding weights $\omega_i$ and points $x_i$. The integral in Eq. (12.13) is rewritten as

$$\frac{2}{\pi}\int_0^\infty dpp^2V(k,p)\psi(p) \approx \frac{2}{\pi}\sum_{i=1}^N \omega_i p_i^2 V(k,p_i)\psi(p_i). \tag{12.14}$$

We can then rewrite the Schrödinger equation as

$$\frac{k^2}{m}\psi(k) + \frac{2}{\pi}\sum_{j=1}^N \omega_j p_j^2 V(k,p_j)\psi(p_j) = E\psi(k). \tag{12.15}$$

Using the same mesh points for $k$ as we did for $p$ in the integral evaluation, we get

$$\frac{p_i^2}{m}\psi(p_i) + \frac{2}{\pi}\sum_{j=1}^N \omega_j p_j^2 V(p_i,p_j)\psi(p_j) = E\psi(p_i), \tag{12.16}$$

with $i,j = 1,2,\ldots,N$. This is a matrix eigenvalue equation and if we define an $N \times N$ matrix $\mathbf{H}$ to be

$$H_{ij} = \frac{p_i^2}{m}\delta_{ij} + \frac{2}{\pi}\omega_j p_j^2 V(p_i,p_j), \tag{12.17}$$

where $\delta_{ij}$ is the Kronecker delta, and an $N \times 1$ vector

$$\Psi = \begin{pmatrix} \psi(p_1) \\ \psi(p_2) \\ \ldots \\ \ldots \\ \psi(p_N) \end{pmatrix}, \tag{12.18}$$

we have the eigenvalue problem

$$\mathbf{H}\Psi = E\Psi. \tag{12.19}$$

The algorithm for solving the last equation may take the following form

– Fix the number of mesh points $N$.

– Use the function $gauleg$ in the program library to set up the weights $\omega_i$ and the points $p_i$. Before you go on you need to recall that $gauleg$ uses the Legendre polynomials to fix the mesh points and weights. This means that the integral is for the interval [-1,1]. Your integral is for the interval [0,$\infty$]. You will need to map the weights from $gauleg$ to your interval. To do this, call first $gauleg$, with $a = -1$, $b = 1$. It returns the mesh points and weights. You then map these points over to the limits in your integral. You can then use the following mapping

$$p_i = \text{const} \times tan\left\{\frac{\pi}{4}(1+x_i)\right\},$$

and

$$\omega_i = \text{const}\frac{\pi}{4}\frac{w_i}{cos^2\left(\frac{\pi}{4}(1+x_i)\right)}.$$

$const$ is a constant which we discuss below.

– Construct thereafter the matrix $\mathbf{H}$ with

$$V(p_i,p_j) = \frac{V_0}{4p_ip_j}ln\left(\frac{(p_j+p_i)^2+\mu^2}{(p_j-p_i)^2+\mu^2}\right).$$

– We are now ready to obtain the eigenvalues. We need first to rewrite the matrix $\mathbf{H}$ in tridiagonal form. Do this by calling the library function *tred2*. This function returns the vector $d$ with the diagonal matrix elements of the tridiagonal matrix while e are the non-diagonal ones. To obtain the eigenvalues we call the function $tqli$. On return, the array $d$ contains the eigenvalues. If $z$ is given as the unity matrix on input, it returns the eigenvectors. For a given eigenvalue $k$, the eigenvector is given by the column $k$ in $z$, that is z[][k] in C, or z(:,k) in Fortran 90.

**The problem to solve**

1. Before you write the main program for the above algorithm make a dimensional analysis of Eq. (12.13)! You can choose units so that $p_i$ and $\omega_i$ are in fm$-1$. This is the standard unit for the wave vector. Recall then to insert $\hbar c$ in the appropriate places. For this case you can set the value of $const = 1$. You could also choose units so that the units of $p_i$ and $\omega_i$ are in MeV. (we have previously used so-called natural units $\hbar = c = 1$). You will then need to multiply $\mu$ with $\hbar c = 197$ MeVfm to obtain the same units in the expression for the potential. Why? Show that $V(p_i,p_j)$ must have units MeV$^{-2}$. What is the unit of $V_0$? If you choose these units you should also multiply the mesh points and the weights with $\hbar c = 197$. That means, set the constant $const = 197$.

2. Write your own program so that you can solve the Schrödinger equation in momentum space.

3. Adjust the value of $V_0$ so that you get close to the experimental value of the binding energy of the deuteron, $-2.223$ MeV. Which sign should $V_0$ have?

4. Try increasing the number of mesh points in steps of 8, for example 16, 24, etc and see how the energy changes. Your program returns equally many eigenvalues as mesh points $N$. Only the true ground state will be at negative energy.