

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 algorithm 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. [27].

12.2 Eigenvalue problems

Let us consider the matrix \mathbf{A} of dimension n . The eigenvalues of \mathbf{A} are defined through the matrix equation

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

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

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

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

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

We can rewrite Eq. (12.1) as

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

with \mathbf{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 λ . The eigenvalues of a matrix $\mathbf{A} \in \mathbb{C}^{n \times n}$ are thus the n roots of its characteristic polynomial

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

or

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

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

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

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

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

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

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 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 $\lambda_1 \dots \lambda_n$ (distinct or not). Let \mathbf{D} be the diagonal matrix with the eigenvalues on the diagonal

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

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

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

and for $j = 1 : n$ we have $\mathbf{A} \mathbf{S}(:, j) = \lambda_j \mathbf{S}(:, j)$. See chapter 8 of Ref. [27] 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

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

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} .

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

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

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

which is the same as

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

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

The basic philosophy is to

- either apply subsequent similarity transformations so that

$$\mathbf{S}_N^T \dots \mathbf{S}_1^T \mathbf{A} \mathbf{S}_1 \dots \mathbf{S}_N = \mathbf{D}, \tag{12.5}$$

- 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 θ in the Euclidean n -dimensional space. It means that the 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

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

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

The algorithm is then quite simple. We perform a number of iterations until 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}

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

To demonstrate the algorithm, we consider the simple 2×2 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 2×2 case we have

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

which leads to

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

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

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

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

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

To implement the Jacobi algorithm we can proceed as follows

- Choose a tolerance ϵ , 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

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

This is however a very time-consuming test which can be replaced by the simpler test

$$\max(a_{ij}^2) > \epsilon.$$

- 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 $\mathbf{B} = \mathbf{S}(k, l, \theta)^T \mathbf{A} \mathbf{S}(k, l, \theta)$.
- Continue till

$$\max(a_{ij}^2) \leq \epsilon.$$

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.

The slow convergence is related to the fact that when a new rotation is performed, matrix elements which were previously zero, may change to non-zero values in the next rotation. To see this, consider the following simple example.

We specialize to a symmetric 3×3 matrix \mathbf{A} . We start the process as follows (assuming that $a_{23} = a_{32}$ is the largest non-diagonal matrix element) with $c = \cos \theta$ and $s = \sin \theta$

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{pmatrix}.$$

We will choose the angle θ in order to have $a_{23} = a_{32} = 0$. We get the new symmetric matrix

$$\mathbf{B} = \begin{pmatrix} a_{11} & a_{12}c - a_{13}s & a_{12}s + a_{13}c \\ a_{12}c - a_{13}s & a_{22}c^2 + a_{33}s^2 - 2a_{23}sc & (a_{22} - a_{33})sc + a_{23}(c^2 - s^2) \\ a_{12}s + a_{13}c & (a_{22} - a_{33})sc + a_{23}(c^2 - s^2) & a_{22}s^2 + a_{33}c^2 + 2a_{23}sc \end{pmatrix}.$$

Note that a_{11} is unchanged! As it should.

We have

$$\mathbf{B} = \begin{pmatrix} a_{11} & a_{12}c - a_{13}s & a_{12}s + a_{13}c \\ a_{12}c - a_{13}s & a_{22}c^2 + a_{33}s^2 - 2a_{23}sc & (a_{22} - a_{33})sc + a_{23}(c^2 - s^2) \\ a_{12}s + a_{13}c & (a_{22} - a_{33})sc + a_{23}(c^2 - s^2) & a_{22}s^2 + a_{33}c^2 + 2a_{23}sc \end{pmatrix}.$$

or

$$\begin{aligned} b_{11} &= a_{11} \\ b_{12} &= a_{12}\cos\theta - a_{13}\sin\theta, 1 \neq 2, 1 \neq 3 \\ b_{13} &= a_{13}\cos\theta + a_{12}\sin\theta, 1 \neq 2, 1 \neq 3 \\ b_{22} &= a_{22}\cos^2\theta - 2a_{23}\cos\theta\sin\theta + a_{33}\sin^2\theta \\ b_{33} &= a_{33}\cos^2\theta + 2a_{23}\cos\theta\sin\theta + a_{22}\sin^2\theta \\ b_{23} &= (a_{22} - a_{33})\cos\theta\sin\theta + a_{23}(\cos^2\theta - \sin^2\theta) \end{aligned}$$

We will fix the angle θ so that $b_{23} = 0$.

We get then a new matrix

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & 0 \\ b_{13} & 0 & b_{33} \end{pmatrix}.$$

We repeat assuming that b_{12} is the largest non-diagonal matrix element and get a new matrix

$$\mathbf{C} = \begin{pmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & 0 \\ b_{13} & 0 & b_{33} \end{pmatrix} \begin{pmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We continue this process till all non-diagonal matrix elements are zero. It is easy to convince oneself that when performing the above operations, the matrix element b_{23} which was previously set to zero may become different from zero. This is one of the problems which slows down the Jacobi procedure.

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 than 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,

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

where the primed quantities represent a matrix \mathbf{A}' of dimension $n - 1$ which will subsequently 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 submatrix \mathbf{A}' only

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

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

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

and off-diagonal matrix elements

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

The resulting matrix reads

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

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, $\mathbf{0}^T = \{0, 0, \dots\}$ 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 $\mathbf{u}^T \mathbf{u} = 1$, that is its 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 $\mathbf{v}^T = \{a_{21}, a_{31}, \dots, a_{n1}\}$ and \mathbf{P} must satisfy $(\mathbf{P}\mathbf{v})^T = \{k, 0, 0, \dots\}$. Then

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

with $\mathbf{e}^T = \{1, 0, 0, \dots, 0\}$. 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

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

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 via Francis' algorithm

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 diagonal¹.

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

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

¹This section is not complete it will be finished end of fall 2009.

which rewritten in matrix form reads

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

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 $\mathbf{A} \rightarrow$ tridiagonal matrix \rightarrow 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. [27], 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

$$\mathbf{S}_1^T \mathbf{A} \mathbf{S}_1 = \mathbf{A}' = \begin{pmatrix} d'_1 & e'_1 & 0 & 0 \\ e'_1 & d_2 & e_2 & 0 \\ 0 & e_2 & d_3 & e'_3 \\ 0 & 0 & e'_3 & d'_4 \end{pmatrix}$$

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 θ 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 The QR algorithm for finding eigenvalues

In preparation for 2010

12.7 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 α 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 ρ 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

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

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

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

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

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

If we now define an arbitrary value of x as

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

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 $u_k = u(x_k)$, $u_{k\pm 1} = u(x_k \pm h)$ 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 $N_{\text{step}} - 1$ values of k we can write the latter equation as a matrix eigenvalue problem

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

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

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

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

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

12.7.1 Numerical solution of the Schrödinger equation by diagonalization

The algorithm for solving Eq. (12.12) may take the following form

- Define values for N_{step} , R_{min} and R_{max} . These values define in turn the step size h . Typical values for R_{max} and R_{min} could be 10 and -10 respectively for the lowest-lying states. The number of mesh points N_{step} could be in the range 100 to some thousands. You can check the stability of the results as functions of $N_{\text{step}} - 1$ and R_{max} and R_{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_{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 $N_{\text{step}} - 1$, 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 $n = N_{\text{step}} - 1$ and a matrix z of dimension $N_{\text{step}} - 1 \times N_{\text{step}} - 1$ 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.
- 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

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

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

12.7.2 Program example and results for the one-dimensional harmonic oscillator

We present here a program example which encodes the above algorithm.

<http://www.fys.uio.no/compphys/cp/programs/FYS3150/chapter12/cpp/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 *outfile;
  // 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{
    outfile=argv [1];
  }
  ofile.open(outfile);
  // 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_1 * (orb_1 + 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 output 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 values 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 calculation of the wave function for the lowest eigenvalue is done in the function *plot*, while all output of the calculations is directed to the function *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.

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 $\approx N^3$.

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

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 called Richardson's deferred extrapolation scheme. Applied to this particular 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. (12.13) for different values of the step size h . We obtain then a series of eigenvalues $E(h/2^k)$ with $k = 0, 1, 2, \dots$. That will give us an array of 'x-values' $h, h/2, h/4, \dots$ and an array of 'y-values' $E(h), E(h/2), E(h/4), \dots$. 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, ..., 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 Fortran 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
  ENDF
```

Table 12.2: Result for numerically calculated eigenvalues of the one-dimensional harmonic oscillator after three iterations starting with a matrix of size 100×100 and ending with a matrix of dimension 800×800 . These four values are then used to extrapolate the 10 lowest-lying eigenvalues to $\hbar = 0$. 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

```

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)
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 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(1)
  inp=d(1)
  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(1),err1
  ELSE
    WRITE(6,'(D12.6,2X,D12.6,2X,D12.6)') d(1),d(1),err1
  ENDIF
ENDDO
DEALLOCATE (w,r,d,e)
ENDDO

```

12.8 Discussion of BLAS and LAPACK functionalities

In preparation, ready 2010.

12.9 Exercises and projects

Project 12.1: Schrödinger's equation for two electrons in a three-dimensional harmonic oscillator well

The aim of this project is to solve Schrödinger's equation for two electrons in a three-dimensional harmonic oscillator well with and without a repulsive Coulomb interaction. Your task is to solve this equation by reformulating it in a discretized form as an eigenvalue equation to be solved with Jacobi's method. To achieve this you will have to write your own code which implements Jacobi's method.

Electrons confined in small areas in semiconductors, so-called quantum dots, form a hot research area in modern solid-state physics, with applications spanning from such diverse fields as quantum nanomedicine to the construction of quantum gates.

Here we will assume that these electrons move in a three-dimensional harmonic oscillator potential (they are confined by for example quadrupole fields) and repel each other via the static Coulomb interaction. We assume spherical symmetry.

We are first interested in the solution of the radial part of Schrödinger's equation for one electron. This equation reads

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

In our case $V(r)$ is the harmonic oscillator potential $(1/2)kr^2$ with $k = m\omega^2$ and E is the energy of the harmonic oscillator in three dimensions. The oscillator frequency is ω and the energies are

$$E_{nl} = \hbar\omega \left(2n + l + \frac{3}{2} \right),$$

with $n = 0, 1, 2, \dots$ and $l = 0, 1, 2, \dots$.

Since we have made a transformation to spherical coordinates it means that $r \in [0, \infty)$. The quantum number l is the orbital momentum of the electron. Then we substitute $R(r) = (1/r)u(r)$ and obtain

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

The boundary conditions are $u(0) = 0$ and $u(\infty) = 0$.

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

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

We will set in this project $l = 0$. Inserting $V(\rho) = (1/2)k\alpha^2\rho^2$ we end up with

$$-\frac{\hbar^2}{2m\alpha^2} \frac{d^2}{d\rho^2} u(\rho) + \frac{k}{2} \alpha^2 \rho^2 u(\rho) = Eu(\rho).$$

We multiply thereafter with $2m\alpha^2/\hbar^2$ on both sides and obtain

$$-\frac{d^2}{d\rho^2} u(\rho) + \frac{mk}{\hbar^2} \alpha^4 \rho^2 u(\rho) = \frac{2m\alpha^2}{\hbar^2} Eu(\rho).$$

The constant α can now be fixed so that

$$\frac{mk}{\hbar^2} \alpha^4 = 1,$$

or

$$\alpha = \left(\frac{\hbar^2}{mk} \right)^{1/4}.$$

Defining

$$\lambda = \frac{2m\alpha^2}{\hbar^2} E,$$

we can rewrite Schrödinger's equation as

$$-\frac{d^2}{d\rho^2} u(\rho) + \rho^2 u(\rho) = \lambda u(\rho).$$

This is the first equation to solve numerically. In three dimensions the eigenvalues for $l = 0$ are $\lambda_0 = 3, \lambda_1 = 7, \lambda_2 = 11, \dots$

We use the by now standard expression for the second derivative of a function u

$$u'' = \frac{u(\rho+h) - 2u(\rho) + u(\rho-h)}{h^2} + O(h^2), \quad (12.11)$$

where h is our step. Next we define minimum and maximum values for the variable ρ , $\rho_{\min} = 0$ and ρ_{\max} , respectively. You need to check your results for the energies against different values ρ_{\max} , since we cannot set $\rho_{\max} = \infty$.

With a given number of steps, n_{step} , we then define the step h as

$$h = \frac{\rho_{\max} - \rho_{\min}}{n_{\text{step}}}.$$

Define an arbitrary value of ρ as

$$\rho_i = \rho_{\min} + ih \quad i = 0, 1, 2, \dots, n_{\text{step}}$$

we can rewrite the Schrödinger equation for ρ_i as

$$-\frac{u(\rho_i + h) - 2u(\rho_i) + u(\rho_i - h)}{h^2} + \rho_i^2 u(\rho_i) = \lambda u(\rho_i),$$

or in a more compact way

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \rho_i^2 u_i = -\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + V_i u_i = \lambda u_i,$$

where $V_i = \rho_i^2$ is the harmonic oscillator potential. Define first the diagonal matrix element

$$d_i = \frac{2}{h^2} + V_i,$$

and the non-diagonal matrix element

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

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

$$d_i u_i + e_{i-1} u_{i-1} + e_{i+1} u_{i+1} = \lambda u_i,$$

where u_i is unknown. We can write the latter equation as a matrix eigenvalue problem

$$\begin{pmatrix} d_1 & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_2 & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_3 & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & d_{n_{\text{step}}-2} & e_{n_{\text{step}}-1} \\ 0 & \dots & \dots & \dots & \dots & e_{n_{\text{step}}-1} & d_{n_{\text{step}}} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ \dots \\ \dots \\ u_{n_{\text{step}}-1} \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ \dots \\ \dots \\ u_{n_{\text{step}}-1} \end{pmatrix} \quad (12.12)$$

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

$$\begin{pmatrix} \frac{2}{h^2} + V_1 & -\frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2} + V_2 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2} + V_3 & -\frac{1}{h^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \frac{2}{h^2} + V_{n_{\text{step}}-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{n_{\text{step}}-1} \end{pmatrix} \quad (12.13)$$

Recall that the solutions are known via the boundary conditions at $i = n_{\text{step}}$ and at the other end point, that is for ρ_0 . The solution is zero in both cases.

- a) Your task here is to write a function which implements Jacobi's rotation algorithm in order to solve Eq. (12.12).

We Define the quantities $\tan \theta = t = s/c$, with $s = \sin \theta$ and $c = \cos \theta$ and

$$\cot 2\theta = \tau = \frac{a_{ll} - a_{kk}}{2a_{kl}}.$$

We can then define the angle θ so that the non-diagonal matrix elements of the transformed matrix a_{kl} become non-zero and we obtain the quadratic equation (using $\cot 2\theta = 1/2(\cot \theta - \tan \theta)$)

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

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 $s = tc$. Explain why we should choose t to be the smaller of the roots. Show that these choice ensures that $|\theta| \leq \pi/4$ and has the effect of minimizing the difference between the matrices \mathbf{B} and \mathbf{A} since

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

- b) How many points n_{step} do you need in order to get the lowest three eigenvalues with four leading digits? Remember to check the eigenvalues for the dependency on the choice of ρ_{max} .

How many similarity transformations are needed before you reach a result where all non-diagonal matrix elements are essentially zero? Try to estimate the number of transformations and extract a behavior as function of the dimensionality of the matrix.

You can check your results against the code based on Householder's algorithm, `tqli` in the file `lib.cpp`.

Comment your results (here you could for example compute the time needed for both algorithms for a given dimensionality of the matrix).

- c) We will now study two electrons in a harmonic oscillator well which also interact via a repulsive Coulomb interaction. Let us start with the single-electron equation written as

$$-\frac{\hbar^2}{2m} \frac{d^2}{dr^2} u(r) + \frac{1}{2} kr^2 u(r) = E^{(1)} u(r),$$

where $E^{(1)}$ stands for the energy with one electron only. For two electrons with no repulsive Coulomb interaction, we have the following Schrödinger equation

$$\left(-\frac{\hbar^2}{2m} \frac{d^2}{dr_1^2} - \frac{\hbar^2}{2m} \frac{d^2}{dr_2^2} + \frac{1}{2} kr_1^2 + \frac{1}{2} kr_2^2 \right) u(r_1, r_2) = E^{(2)} u(r_1, r_2).$$

Note that we deal with a two-electron wave function $u(r_1, r_2)$ and two-electron energy $E^{(2)}$.

With no interaction this can be written out as the product of two single-electron wave functions, that is we have a solution on closed form.

We introduce the relative coordinate $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$ and the center-of-mass coordinate $\mathbf{R} = 1/2(\mathbf{r}_1 + \mathbf{r}_2)$. With these new coordinates, the radial Schrödinger equation reads

$$\left(-\frac{\hbar^2}{m} \frac{d^2}{dr^2} - \frac{\hbar^2}{4m} \frac{d^2}{dR^2} + \frac{1}{4} kr^2 + kR^2 \right) u(r, R) = E^{(2)} u(r, R).$$

The equations for r and R can be separated via the ansatz for the wave function $u(r, R) = \psi(r)\phi(R)$ and the energy is given by the sum of the relative energy E_r and the center-of-mass energy E_R , that is

$$E^{(2)} = E_r + E_R.$$

We add then the repulsive Coulomb interaction between two electrons, namely a term

$$V(r_1, r_2) = \frac{\beta e^2}{|\mathbf{r}_1 - \mathbf{r}_2|} = \frac{\beta e^2}{r},$$

with $\beta e^2 = 1.44 \text{ eVnm}$.

Adding this term, the r -dependent Schrödinger equation becomes

$$\left(-\frac{\hbar^2}{m} \frac{d^2}{dr^2} + \frac{1}{4}kr^2 + \frac{\beta e^2}{r} \right) \psi(r) = E_r \psi(r).$$

This equation is similar to the one we had previously in (a) and we introduce again a dimensionless variable $\rho = r/\alpha$. Repeating the same steps as in (a), we arrive at

$$-\frac{d^2}{d\rho^2} \psi(\rho) + \frac{mk}{\hbar^2} \alpha^4 \rho^2 \psi(\rho) + \frac{m\alpha\beta e^2}{\rho\hbar^2} \psi(\rho) = \frac{m\alpha^2}{\hbar^2} E_r \psi(\rho).$$

We want to manipulate this equation further to make it as similar to that in (a) as possible. We define $k_r = 1/4k$. The constant α is then again fixed so that

$$\frac{mk_r}{\hbar^2} \alpha^4 = 1,$$

or

$$\alpha = \left(\frac{\hbar^2}{mk_r} \right)^{1/4}.$$

Defining

$$\lambda = \frac{m\alpha^2}{\hbar^2} E,$$

we can rewrite Schrödinger's equation as

$$-\frac{d^2}{d\rho^2} \psi(\rho) + \rho^2 \psi(\rho) + \frac{\gamma}{\rho} = \lambda \psi(\rho),$$

with

$$\gamma = \frac{m\alpha\beta e^2}{\hbar^2}.$$

We treat γ as a parameter which reflects the strength of the oscillator potential.

Here we will study the cases $\gamma = 0$, $\gamma = 0.5$, $\gamma = 1$, $\gamma = 2$ and $\gamma = 4$. for the ground state only, that is the lowest-lying state.

For $\gamma = 0$ you should get a result which corresponds to the relative energy of a non-interacting system. The way we have written the equations means you get the same as in (a) for $\gamma = 0$. Make sure your results are stable as functions of ρ_{\max} and the number of steps.

We are only interested in the ground state with $l = 0$. We omit the center-of-mass energy.

You can reuse the code you wrote for (a), but you need to change the potential from ρ^2 to $\rho^2 + \gamma/\rho$. Comment the results for the lowest state (ground state) as function of varying strengths of γ .

For specific oscillator frequencies, the above equation has analytic answers, see the article by M. Taut, Phys. Rev. A 48, 3561 - 3566 (1993). The article can be retrieved from the following web address http://prola.aps.org/abstract/PRA/v48/i5/p3561_1.

- d) In this exercise we want to plot the wave function for two electrons as functions of the relative coordinate r and different values of γ . For $\gamma = 0$ your wave function should correspond to that of a harmonic oscillator. Varying γ , the shape of the wave function will change.

We are only interested in the wave function for the ground state with $l = 0$ and omit again the center-of-mass motion.

You can choose between two approaches; the first is to use the existing *tqli* function. Here the eigenvectors are obtained from the matrix $z[i][j]$, where the index j refers to eigenvalue j . The index i points to the value of the wave function in position ρ_j . That is, $u^{(\lambda_j)}(\rho_i) = z[i][j]$.

The eigenvectors are normalized. Plot then the normalized wave functions for different values of γ and comment the results.

The other alternative is to add a piece to your Jacobi routine which also returns the eigenvectors. This is the more difficult part. You will need to normalize the eigenvectors.

