Chapter 11

The Eigenvalue Problem

11.1 Introduction

Applications of the eigenvalue problem can be found in practically every field of physics, from classical to quantum, mechanics. (*more specific examples: structure calculations in in atomic, molecular, nuclear and solid state physics*). Discretization of some types of ordinary and partial differential equation also leads to the eigenvalue problem in linear algebra.

Let's recollect some rudimentary knowledge from courses of linear algebra. If A is a square matrix of order n, then the eigenvalue problem is presented as

$$A\vec{x} = \lambda\vec{x}, \tag{11.1}$$

or as a system of linear equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= \lambda x_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= \lambda x_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= \lambda x_n. \end{aligned} \tag{11.2}$$

The system (11.2) looks like a regular linear system of equations. However, there is substantial difference between the eigenvalue problem and solving a linear system of equations. For the eigenvalue problem the scalars λ's are unknown, and solutions for the system (11.2) exist only for specific values of λ. These values are called eigenvalues. A vector $\vec{x}$ corresponding to an eigenvalue λ is an eigenvector.

Regrouping terms in the system (11.2) gives a system of homogeneous linear equations

$$\begin{pmatrix} a_{11}-\lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22}-\lambda & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn}-\lambda \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \end{pmatrix}, \tag{11.3}$$

Introducing a unit matrix I, which is

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{11.4}$$

the system of linear equations (11.3) may also be written as

$$(A - \lambda I)\vec{x} = 0. \tag{11.5}$$

Nontrivial solutions for the system (11.5) exists if and only if the determinant of the matrix $(A - \lambda I)$ to be zero, that is

$$\det |A - \lambda I| = 0. \tag{11.6}$$

If expanded, the determinant (11.6) is a characteristic polynomial of degree n in λ. It has n eigenvalues λ_i $(i = 1, 2, \dots, n)$, including multiple roots. Since a polynomial can have not only real but complex roots as well, the eigenvalues can be real and/or complex. It has been proved, that it is not possible to compute roots of a polynomial for $n > 4$ in a finite number of steps. Therefore, all numerical methods for finding eigenvalues are iterative. Than makes the eigenvalue problem in linear algebra different form other linear problems. All other linear problems can be solved in a finite number of calculations.

There are many methods for solving the eigenvalue problem. The direct solution of the characteristic equation derived from equation (11.6) would yield n roots (eigenvalues). Then, the eigenvectors $\vec{x}$, can be calculated by substituting the individual eigenvalues λ into the homogeneous system of equations (11.3). Looking straightforwardly, this approach is rarely used in practice, unless n the matrix is very small. If one only eigenvalue is needed (the largest or the smallest in absolute value), then the

iterative power method is a practical approach. The power method is based on the repetitive matrix multiplication of a trial eigenvector $\vec{y}$ by matrix A, which eventually yields the largest eigenvalue. Most methods for finding the eigenvalues and eigenvectors are based on the fact that the transformation

$$A' = R^{-1}AR \tag{11.7}$$

does not alter the eigenvalues of A. It is also called as the similarity transformation. This property can be easily demonstrated using determinant properties. Since $\det(AB) = \det(A)\det(B)$ and $\det(A^{-1}) = (\det(A))^{-1})$ then $\det(A') = \det(R^{-1}AR) = \det(R^{-1})\det(A)\det(R)$ and using the property for the inverse matrices $\det(A') = \det(A)\det(R^{-1})\det(R)$ $\det(A)$. With a proper similarity transformation it is possible to transform matrix A to the diagonal or triangular form. Then the problem is solved, since the eigenvalues can be read from the diagonal. The process is iterative and its convergence depends on the type of a matrix (*may add the definition for the determinant for these kind of matrices*). The Jacobi method for symmetric matrices transforms matrix A to its diagonal form by iterative rotation transformations, that is a subset of similarity transformations. The most efficient methods for solving the eigenvalue problem employ a two-step approach. As the first step, a similarity transformation is used to to reduce the original matrix to tridiagonal or Hessenberg form. It can be done in a *finite* number of steps. Then, using one of factorization methods (e.g. the QR method) all the eigenvalues are computed. The factorization methods converge faster for these specific matrices (tridiagonal or Hessenberg).

Since there are many forms of matrices, there is no single method that is universally suitable. The choice of a proper method for attacking the eigenvalue problem depends of the form of matrix A, the matrix dimension n, whether we need one or all eigenvalues. The most efficient methods are tailored to the form of a matrix. For example, many applications in physics deal with symmetric matrices with real coefficients, or Hermitian matrices.

In this chapter we will mostly work with this kind of matrices.

Comment on eigenvectors: Even so, this x is determined only up to a proportionality factor, since the homogeneity of our equations permits any solution to be multiplied by an arbitrary constant and still remain a solution. Geometrically, our solution vector x has a unique direction but indeterminant length.

11.2 The power method

In some applications we are interested in one or few of the eigenvalues. For example, in quantum mechanical structure calculations we often need only to evaluate the ground state energy (that is the largest negative eigenvalue).

In this section we consider a symmetric $n \times n$ matrix A with real coefficients, and having n eigenvalues. Most of equations in this section continue to be correct for asymmetric matrices, till the largest eigenvalue is a real number. In case of complex eigenvalues, the equations presented in this section are to be modified, or methods specifically designed for comples eigenvalues are to be used.

11.2.1 The basic power method

Without simplifying the consideration, we may assume, that the eigenvalues are aligned in decreasing order $|\lambda_1| > |\lambda_2| > \ldots |\lambda_n|$. The n eigenvectors of a nonsingular matrix span the n-dimensional space. (The eigenvectors of a symmetric matrix are mutually orthogonal). The power method is based on the fact that any arbitrary vector in the n-dimensional space may be expressed as a linear combination of the eigenvectors of the matrix A, as

$$\vec{y} = \sum_{i=1}^{n} c_i \vec{x}_i, \tag{11.8}$$

where $\vec{x}$ is a set of eigenvectors.

Multiplying both sides of (11.8) by A, AA, ..., $A^{(k)}$, and using that $A\vec{x} = \lambda\vec{x}$ we obtain

$$A\vec{y} = \sum_{i=1}^{n} c_i A\vec{x}_i = \sum_{i=1}^{n} c_i \lambda_i \vec{x}_i = \vec{y}_{(1)} \tag{11.9}$$

$$AA\vec{y} = A\vec{y}_{(1)} = \sum_{i=1}^{n} c_i \lambda_i A\vec{x}_i = \sum_{i=1}^{n} c_i \lambda_i^2 \vec{x}_i = \vec{y}_{(2)} \tag{11.10}$$

$$\ldots$$

$$A^{(k)}\vec{y} = A\vec{y}_{(k-1)} = \sum_{i=1}^{n} c_i \lambda_i^{k-1} A\vec{x}_i = \sum_{i=1}^{n} c_i \lambda_i^{k} \vec{x}_i = \vec{y}_{(k)}. \tag{11.11}$$

Thus, each multiplication by A changes the previous vector to a new one

$$\vec{y}_{(k+1)} = A\vec{y}_{(k)}. \tag{11.12}$$

Factoring out λ_1^k in (11.11) we may write

$$A^{(k)}\vec{y} = \lambda_1^k \sum_{i=1}^{n} c_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \vec{x}_i = \lambda_i^k \left[c_1\vec{x}_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k \vec{x}_2 + \ldots + c_n \left(\frac{\lambda_n}{\lambda_1}\right)^k \right] \quad (11.13)$$

Since λ_1 is the largest eigenvalue in absolute value, then all $(\lambda_i/\lambda_k)^k \to 0$ for $i = 2, 3, \ldots, n$ as $k \to \infty$. Thus, we may write

$$\vec{y}_{(k)} = \lambda_i^k c_1 \vec{x}_1. \quad (11.14)$$

The repeated pre-multiplication of an arbitrary vector $\vec{y}$ by matrix A would result in computing the largest eigenvalue, since for $k \to \infty$

$$\vec{y}_{(k+1)} = \lambda_1 \vec{y}_{(k)}. \quad (11.15)$$

For the algorithm to be practical we need to take into account that the repeated pre-multiplication results in unconstrained grows of the length of $\vec{y}_{(k)}$, while changing its direction rather slowly. The problem can be addressed by a normalization between iterations, that preserves the direction of $\vec{y}_{(k)}$ but rescale the length. In older textbooks, for in hand calculations, it was recommended to make the largest component of $\vec{y}_{(k)}$ equal to unity. For computer calculations we may normalize the length of the vector $\vec{y}_{(k)}$ to unity at each iteration, treating all components of the vector symmetrically.

The convergence of the iterative process is proportional to the ratio λ_2/λ_1, where λ_2 is the next largest in magnitude eigenvalue. It is clear that the power method fails if the ratio of the first two eigenvalues is ± 1. If the two first eigenvalues are very close (but not identically equal), the iterative process would be very slow to be practical. Therefore for the power method to be efficient, the largest eigenvalue must be distinct. Besides, the initial guess for the trial vector $\vec{y}$ must have some component of the eigenvector $\vec{x}$ corresponding to λ_1. It is common to choose all the components of the trial vector as equal to unity.

example - hand in calculations for 3x3 matrix The reader should try this iteration on the matrix ... starting with [1,0,0] and carrying out about five iterations,

Program 11.1. **The Power method for symmetric matrices**

```
  subroutine Power(a,y,lambda,eps,n,iter)
!=============================================================
! Evaluate the largest eigenvalue and corresponding eigenvector
! of a real matrix a(n,n): a*x = lambda*x
```

```
! method: the power method
! comment: the program works for real values only
! Alex G. (December 2009)
!-------------------------------------------------------------
! input ...
! a(n,n) - array of coefficients for matrix A
! x(n)   - initial vector
! n      - number of equations
! eps    - convergence tolerance
! output ...
! lambda - eigenvalue (the largest modulus)
! x(n)   - eigenvector corresponding to lambda
! iter   - number of iterations to achieve the tolerance
! comments ...
! kmax   - max number of allowed iterations
!=============================================================
implicit none
integer n, iter
double precision a(n,n),y(n),lambda, eps
double precision yp(n),lambda0, norm
integer k, i, j
integer, parameter::kmax=1000

lambda0=0.0

do k=1,kmax
! compute y'=A*y
  do i=1,n
    yp(i)=0.0
    do j=1,n
      yp(i) = yp(i)+a(i,j)*y(j)
    end do
  end do
! normalization coefficient
  norm = 0.0
  do i=1,n
    norm = norm + yp(i)*yp(i)
  end do
  norm = sqrt(norm)
! normalize vector y(n) to unity for the next iteration
  do i=1,n
```

```
    y(i)=yp(i)/norm
  end do
  lambda = norm
! check for convergence
  if (abs(lambda-lambda0) < eps) exit
! prepare for the next iteration
  lambda0 = lambda
end do

iter = k
if(k == kmax) write (*,*)'The eigenvlue failed to converge'

end subroutine Power
```

Example 11.1. Solution by the Power method

```
 The largest eigenvalues (Power method)
 Matrix A
    1.000000    2.000000    3.000000
    2.000000    2.000000   -2.000000
    3.000000   -2.000000    4.000000

 Initial vector
    1.000000    1.000000    1.000000

 The largest eigenvalue
    6.000000

 Eigenvector
    0.436487   -0.218143    0.872865

 iterations =    18
```

Have comments for arbitrary matrices - see comments at the end

11.2.2 The shifted power method

The eigenvalues λ of a matrix A may all be shifted by a scalar s by subtracting it from the main diagonal elements of A. Thus

$$(A - sI)\vec{x} = (\lambda - s)\vec{x}, \tag{11.16}$$

and

$$B\vec{x} = \beta\vec{x}, \tag{11.17}$$

where the new eigenvalue problem (11.17) has the same eigenvectors, and the old and new eigenvalues are connected in a simple way $\beta = \lambda - s$.

Shifting the eigenvalues of a matrix is very useful in finding the the opposite extreme eigenvalue, accelerating the convergence, and even to find intermediate eigenvalues.

Suppose a matrix A has five eigenvalues, for example 1, 4, 9, 16, 25. Using the direct power method we may find the largest eigenvalue, that is 25. Then, shifting the eigenvalues by this amount s=25 would result in a set of the following eigenvalues for the shifted matrix -24, -21, -16, -9, 0. Applying the basic power method to the shifted matrix B yields the largest (in absolute scale) eigenvalue $\beta = -24$, that corresponds to $\lambda = 1$. Thus, the shifting provides the power method with a tool to find both the largest and the smallest eigenvalues of a matrix.

The convergence rate of the power method for the original matrix A is the ratio of the largest to the second largest eigenvalue, i.e. $25/16 \simeq 1.56$. What would happen if we shift the above eigenvalues 1, 4, 9, 16, 25 by a smaller amount, for example s=5. Then the new set of β is -4, -1, 4, 11, 20, and the convergence rate accelerates $20/11 \simeq 1.82$. Thus if our iterative process seems to be slow, we may shift the eigenvalues by some amount, and continue iterations. If the convergence speeds up, we may try to shift more, if it gets worse, we may shift in the opposite direction. This approach was popular in the times of the in hand calculations, however, it is still useful in computer calculations.

11.2.3 The inverse power method

In many physics applications the eigenvalues are arranged in non-linear manner. For example, energy spectrum (eigenvalues) of most quantum systems (atoms, molecules, nuclei) have rather a distinct ground state (the most negative eigenvalue), with most excited states concentrated closer to the zero. Therefore, shifting in by the ground state energy will bring the smallest eigenvalue λ_n to the largest eigenvalue β_1, but the next eigenvalue β_2 could be so close to the first one, that the convergence rate may be impractical. For the example above, the shifting by s=25, yields a slower convergence $24/21 \simeq 1.14$ for finding the smallest eigenvalue.

The inverse power method, that is a variation of the basic power method, is more

powerful way to find the smallest distinct eigenvalue. In the original eigenvalue problem

$$A\vec{x} = \lambda\vec{x} \tag{11.18}$$

we multiple both sides by the inverse matrix A^{-1}

$$A^{-1}A\vec{x} = \vec{x} = \lambda A^{-1}\vec{x}, \tag{11.19}$$

hence

$$A^{-1}\vec{x} = \frac{1}{\lambda}\vec{x}. \tag{11.20}$$

The inverted matrix has the same eigenvectors as A, but inverted eigenvalues. Evidently, the power method applied to the inverse matrix yields the largest $1/\lambda$, that corresponds to the smallest (in absolute value) eigenvalue of matrix A. In practical calculations. As we recall from chapter 6, the LU Doolittle factorization is an efficient technique to compute the matrix A^{-1}.

Using together the inverse power method with the shifted power method makes possible to find other eigenvalues. We consider the same example with five eigenvalues 1, 4, 9, 16, 25 of A. Suppose we already calculated the largest and the smallest eigenvalues, i.e. 1 and 25. If we shift the eigenvalues by s=(25-1)/2=12, then the shifted set is -11, -8, -3, 4, 13. Applying the inverse power method yields the smallest eigenvalue in absolute value, that is -3, corresponding to the middle eigenvalue in the original set $\lambda = 9$. The methods sounds as feasible, however rarely use to find more that few eigenvalues. There are more efficient methods to find all eigenvalues of a matrix.

11.3 The Jacobi Method (Symmetric Matrices)

The power method with variations is a simple and fast method for computing the largest and the smallest eigenvalues, provided they are well distinct from the adjusted eigenvalues. We need a different approach if all the eigenvalues are to be computed. Most numerical method for calculating all the eigenvalues and eigenvectors are based on *similarity* transformations.

$$A \to Q^{-1}AQ \qquad \text{or} \qquad A \to QAQ^{-1}. \tag{11.21}$$

It is easy to demonstrate that similarity transformation preserves eigenvalues of A. We start with the eigenvalue equation

$$A\vec{x} = \lambda\vec{x} \tag{11.22}$$

and multiply both sides by inverse matrix Q^{-1}

$$Q^{-1}A\vec{x} = \lambda Q^{-1}\vec{x}. \tag{11.23}$$

Defining a new vector $\vec{y}$

$$Q^{-1}\vec{x} = \vec{y} \tag{11.24}$$

we get

$$\vec{x} = Q\vec{y}, \tag{11.25}$$

and then substituting it to (11.23)

$$Q^{-1}AQ\vec{y} = \lambda Q^{-1}Q\vec{y} = \vec{y}. \tag{11.26}$$

Thus the matrix $Q^{-1}AQ$ has the same eigenvalues, but different eigenvectors. Methods based on similarity transformation attempt to to find mutices Q such that matrix $Q^{-1}AQ$ has a form, that makes simple/easy to evaluate the eigenvalues.

There are various types (classes?) of similarity transformations. The orthogonal transformation is one of the most popular transformation in the eigenvalue problem. In this case the transpose matrix Q^T is equal to its inverse matrix Q^{-1}. The orthogonality transformation Q^-1AQ preserves both eigenvalues and symmetry of original A. One of the simples orthogonal transformation is the plane rotation. In 1846 Jacobi applied the plane rotation transformation to calculate all the eigenvalues and eigenvector of real symmetric and Hermitian matrices.

The Jacobi method iteratively uses orthogonal similarity transformations

$$A_{k+1} = R_k^{-1}A_kR_k \tag{11.27}$$

to transform the original matrix A to a diagonal form. Then the eigenvalues are the diagonal elements. The R matrices are the plane rotational matrices (also called Givens rotational matrices), where for $R_{i,j}$ the diagonal elements $r_{i,i} = r_{j,j} = c$, all the other diagonal elements are unity, and for off-diagonal elements $r_{i,j} = -r_{j,i} = s$, all other off-diagonal elements are zero. The coefficients c and s satisfy the following condition $c^2 + s^2 = 1$. For example 5×5 matrix $R_{2,4}$ has the following form

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & s & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -s & 0 & c & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{11.28}$$

The Givens rotational matrices have the following property $R_{i,j}^{-1} = R_{i,j}^T$ where $R_{i,j}^T$ is the transpose matrix. Thus $R_{i,j}$ is orthogonal since $R_{i,j}^T R_{i,j} = R_{i,j} R_{i,j}^T = I$.

We consider in detailes the transformation $R_{i,j}^T A R_{i,j}$. The pre-multiplication $R_{i,j}^T A$ has the effect of replacing rows i and j by linear combination of the original rows i and j, while $AR_{i,j}$ changes only columns i and j. In the transformed matrix A' we are most interested in the two diagonal elements $a'_{i,i}$, $a'_{j,j}$, and two off-diagonal elements $a'_{i,j}$ and $a'_{j,i}$. They follow the transformation

$$a'_{i,i} = c^2 a_{i,i} + s^2 a_{j,j} - 2sca_{i,j} \tag{11.29}$$

$$a'_{j,j} = s^2 a_{i,i} + c^2 a_{j,j} + 2sca_{i,j} \tag{11.30}$$

$$a'_{i,j} = (c^2 - s^2)a_{i,j} + sc(a_{i,i} - a_{j,j}) = a'_{j,i}. \tag{11.31}$$

The other affected elements are

$$a'_{k,i} = ca_{k,i} - sa_{k,j} \qquad (k \neq i, k \neq j) \tag{11.32}$$

$$a'_{k,j} = ca_{k,j} + sa_{k,i} \qquad (k \neq i, k \neq j) \tag{11.33}$$

but we are not interested in these.

We want to choose the coefficients c and s so that the off-diagonal elements $a'_{i,j} = a'_{j,i} = 0$. Then from equation (11.31) follows that

$$(c^2 - s^2)a_{i,j} + sc(a_{i,i} - a_{j,j}) = 0 \tag{11.34}$$

or

$$\frac{c^2 - s^2}{sc} = \frac{a_{j,j} - a_{i,i}}{a_{i,j}} = 2\beta \tag{11.35}$$

Since $c^2 + s^2 = 1$ we may eliminate s from (11.35) and after simple algebra

$$c^4 - c^2 + \frac{1}{4(1+\beta^2)} = 0 \tag{11.36}$$

Solving the quadratic equations for c^2 we get for the coefficients c and s

$$c = \left(\frac{1}{2} - \frac{\beta}{2(1+\beta^2)^{1/2}}\right)^{1/2} \tag{11.37}$$

$$s = \left(\frac{1}{2} + \frac{\beta}{2(1+\beta^2)^{1/2}}\right)^{1/2} \tag{11.38}$$

The good news - choosing the coefficients from (11.37,11.38) we may bring zero into any off-diagonal position i, j, while preserving the eigenvalues, and the symmetry of the matrix. The bad news - on the next transformation the zero elements will be transformed to non-zero. It looks like we do not gain much. However, there is a theorem stating that when the symmetric matrix A is transformed into $R_{i,j}^T A R_{i,j}$, with $R_{i,j}$ chosen so that $a'_{i,j} = 0$, the sum of the squares of the diagonal elements increases by $2a_{i,j}^2$, while the sum of squares of the off-diagonal elements decreases by the same amount. (*a reference?*). Thus, we make steady progress toward the diagonalization. (*more here?*)

There are a couple ways to practically implement the Jacobi method to transfer a real symmetric matrix to near diagonal form (within accepted tolerance). First, we may use a systematic way to treat $((a_{i,j}, j = i+1, \ldots, n), i = 1, 2, \ldots, n-1)$ till all off-diagonal elements are small. This way is definitely slow since we will zero in elements that are already small. Second, we search for the largest (in absolute value) off-diagonal element and transform it to zero (the original Jacobi method). It could be efficient for "in hand calculations" but not for real computing. Finding the largest element takes $O(n^2)$ operations, while the transformation (11.27) takes about $O(n)$ operations. The third, and the most efficient way would be to check all the off-diagonal elements in a systematic order, but zeroing only those whose squares $|a_{i,j}|^2$ is more that a half of the current average for all average for all off-diagonal. (*place a condition here*).

The convergence of the iterative process is at least linear, when far from the solution, and quadratic, when close to the solution.

Here goes the algorithm

Algorithm 7.1 *The Jacobi method for ...*
Step 1: ...

Program 11.2. **the Jacobi method for symmetric matrices**

```
     subroutine Jacobi(a,x,abserr,n)
!============================================================
! Evaluate eigenvalues and eigenvectors
! of a real symmetric matrix a(n,n): a*x = lambda*x
! method: Jacoby method for symmetric matrices
!------------------------------------------------------------
! input ...
! a(n,n) - array of coefficients for matrix A
! n      - number of equations
! abserr - abs tolerance [sum of (off-diagonal elements)^2]
```

```
! output ...
! a(i,i) - eigenvalues
! x(i,j) - eigenvectors
! comments ...
!=============================================================
implicit none
integer i, j, k, n
double precision a(n,n),x(n,n)
double precision abserr, b2, bar
double precision beta, coeff, c, s, cs, sc

! initialize x(i,j)=0, x(i,i)=1
! *** the array operation x=0.0 is specific for Fortran 90/95
x = 0.0
do i=1,n
  x(i,i) = 1.0
end do

! find the sum of all off-diagonal elements (squared)
b2 = 0.0
do i=1,n
  do j=1,n
    if (i.ne.j) b2 = b2 + a(i,j)**2
  end do
end do

if (b2 <= abserr) return

! average for off-diagonal elements /2
bar = 0.5*b2/float(n*n)

do while (b2.gt.abserr)
  do i=1,n-1
    do j=i+1,n
      if (a(j,i)**2 <= bar) cycle  ! do not touch small elements
      b2 = b2 - 2.0*a(j,i)**2
      bar = 0.5*b2/float(n*n)
! calculate coefficient c and s for Givens matrix
      beta = (a(j,j)-a(i,i))/(2.0*a(j,i))
      coeff = 0.5*beta/sqrt(1.0+beta**2)
      s = sqrt(max(0.5+coeff,0.0))
```

```
      c = sqrt(max(0.5-coeff,0.0))
! recalculate rows i and j
      do k=1,n
        cs =  c*a(i,k)+s*a(j,k)
        sc = -s*a(i,k)+c*a(j,k)
        a(i,k) = cs
        a(j,k) = sc
      end do
! new matrix a_{k+1} from a_{k}, and eigenvectors
      do k=1,n
        cs =  c*a(k,i)+s*a(k,j)
        sc = -s*a(k,i)+c*a(k,j)
        a(k,i) = cs
        a(k,j) = sc
        cs =  c*x(k,i)+s*x(k,j)
        sc = -s*x(k,i)+c*x(k,j)
        x(k,i) = cs
        x(k,j) = sc
      end do
    end do
  end do
end do
return
end
```

Example 11.2. Solution by the Jacobi method

```
 Eigenvalues and eigenvectors (Jacobi method)
 Matrix A
    1.000000    2.000000    3.000000
    2.000000    2.000000   -2.000000
    3.000000   -2.000000    4.000000

 Eigenvalues
   -2.541381    3.541381    6.000000

 Eigenvectors
   -0.703413   -0.561011    0.436436
    0.522158   -0.824459   -0.218218
    0.482246    0.074391    0.872872
```

The Jacobi method is about 10 times slower comparing to ... However, the Jacobi

method is invaluable when accuracy, reliability, and simplicity of calculations are more important than time.

11.4 The basic QR method

The QR method employs orthogonal transformations to transform matrix A into a triangular form. On the first step matrix A is factorized as

$$A = QR, \tag{11.39}$$

where columns matrix Q form a set of orthogonal (mutually orthogonal) vectors $\vec{q}$, and matrix R is the upper triangular matrix, whose elements are the vector products $r_{i,j} = \vec{Q}_i^T \vec{a}_j$. Here $\vec{a}_j$ is column j of matrix A. Once Q and R are evaluated, a new A' is calculated as

$$A' = RQ. \tag{11.40}$$

Matrices A and A' are connected by similarity transformation, and share the same eigenvalues. Multiplying from the left (11.39) by Q^{-1} we have

$$Q^{-1}A = Q^{-1}QR = R. \tag{11.41}$$

Multiplying (11.41) from the right by Q yields

$$Q^{-1}AQ = RQ = A'. \tag{11.42}$$

Vectors $\vec{q}$ of matrix Q are evaluated using Gram-Schmidt orthogonalization process. The first vector $\vec{q}_1$ is a normalized vector $\vec{a}_1$

$$\vec{q}_1 = \vec{a}_1 / \|\vec{a}_1\|, \tag{11.43}$$

where

$$\|\vec{a}_j\| = \left(a_{j1}^2 + a_{j2}^2 + \cdots + a_{jn}^2\right)^{1/2}. \tag{11.44}$$

The rest vectors $\vec{q}_j$ are evaluated as

$$\vec{a'}_j = \vec{a}_j - \sum_{m=1}^{j-1} (\vec{q}_m^T \vec{a}_j)\vec{q}_m \qquad (j = 2, \ldots, n), \tag{11.45}$$

and

$$\vec{q_j} = \vec{a'}_j / \|\vec{a'}_j\|. \tag{11.46}$$

The diagonal coefficients of the upper triangular matrix R are

$$r_{j,j} = \|\vec{a'}_j\| \qquad (j = 1, \ldots, n), \tag{11.47}$$

the off-diagonal coefficients are

$$r_{i,j} = \vec{q_i}^T \vec{a}_j \qquad (i = 1, \ldots, n, j = i+1, \ldots, n). \tag{11.48}$$

add more details + iterations

Program 11.3. **the QR method for symmetric matrices**

```
  subroutine QRbasic(a,e,eps,n,iter)
!=============================================================
! Compute all eigenvalues: real symmetric matrix a(n,n,)
! a*x = lambda*x
! method: the basic QR method
! Alex G. (January 2010)
!-------------------------------------------------------------
! input ...
! a(n,n) - array of coefficients for matrix A
! n      - dimension
! eps    - convergence tolerance
! output ...
! e(n)   - eigenvalues
! iter   - number of iterations to achieve the tolerance
! comments ...
! kmax   - max number of allowed iterations
!=============================================================
implicit none
integer n, iter
double precision a(n,n), e(n), eps
double precision q(n,n), r(n,n), w(n), an, Ajnorm, sum, e0,e1
integer k, i, j, m
integer, parameter::kmax=1000

! initialization
q = 0.0
```

```
r = 0.0
e0 = 0.0

do k=1,kmax              ! iterations

! step 1: compute Q(n,n) and R(n,n)
! column 1
  an = Ajnorm(a,n,1)
  r(1,1) = an
  do i=1,n
    q(i,1) = a(i,1)/an
  end do
! columns 2,...,n
  do j=2,n
    w = 0.0
    do m=1,j-1
! product q^T*a result = scalar
      sum = 0.0
      do i=1,n
        sum = sum + q(i,m)*a(i,j)
      end do
      r(m,j) = sum
! product (q^T*a)*q  result = vector w(n)
      do i=1,n
        w(i) = w(i) + sum*q(i,m)
      end do
    end do
! new a'(j)
    do i =1,n
      a(i,j) = a(i,j) - w(i)
    end do
! evaluate the norm for a'(j)
    an = Ajnorm(a,n,j)
    r(j,j) = an
! vector q(j)
    do i=1,n
      q(i,j) = a(i,j)/an
    end do
  end do

! step 2: compute A=R(n,n)*Q(n,n)
```

```
  a = matmul(r,q)
! egenvalues and the average eigenvale
  sum = 0.0
  do i=1,n
    e(i) = a(i,i)
    sum = sum+e(i)*e(i)
  end do
  e1 = sqrt(sum)

! check for convergence
  if (abs(e1-e0) < eps) exit
! prepare for the next iteration
  e0 = e1
end do

iter = k
if(k == kmax) write (*,*)'The eigenvlue failed to converge'

end subroutine QRbasic

  function Ajnorm(a,n,j)
implicit none
integer n, j, i
double precision a(n,n), Ajnorm
double precision sum

sum = 0.0
do i=1,n
  sum = sum + a(i,j)*a(i,j)
end do
Ajnorm = sqrt(sum)
end function Ajnorm
```

Example 11.3. Solution by the QR method

```
QR basic method - eigenvalues for A(n,n)
Matrix A
   1.000000    2.000000    3.000000
   2.000000    2.000000   -2.000000
   3.000000   -2.000000    4.000000

The eigenvalues
```

```
   6.000000    3.541381   -2.541381

iterations =    21
```

11.5 Practical methods

Three parameters are important for practical algorithms: (1) convergence, (2) stability, and (3) efficiency.

Given's method utilizes the same kind or rotational matrix transformations, however, the technique does not destroy any zeros which were created in the previous transformations. Unlike in Jacobi method, we use (c,s) rotation to zero in the (c-1,s) element with $c = 1, 2, \ldots, n-1$ and $s = c+2, c+3, \ldots, n$). The end result of the rotations is not a diagonal matrix, but a tridiagonal one. The eigenvalues of a tridiagonal matrix are then calculated using a Sturmanian recursive sequence of polynomials. The total number of multiplications to calculate all the eigenvalues in this approach is $4n^3/3$. However, there is even faster method. Householder's method also uses orthogonal transformations to reduce a symmetric matrix to tridiagonal form. Unlike Given's method, Householder method produces them a row at a time. The method is much more complicated computationally, but works faster than other methods. Both Given's and Householder methods are extremely stable.

It is worth to mention about other efficient methods. The LR method involves repeated factorization to bring the original matrix A into a product of left-triangular, and right-triangular matrices $A = LU$. It works the same way as Gaussian elimination. The LR decomposition works fast, but it is not very stable. Another decomposition, an exceedingly stable one, is QR algorithm.

Other methods: Faddeev-Leverrier, Lanczos

11.6 Problems

1. Modify the program "Power" in this chapter to carry out calculations with the shifted power method.

2. Using routines from this chapter and chapter "Systems of linear equations", write a program that calculates the smallers eigenvalue (the inverse power method)

3. Using the QRbasic routine together with one of programs from chapter "Systems

of linear equations", write a code that calculates all eigenvalues and eigenvectors of a symmetric matrix with real coefficients.

4. *plus some numerical calculations ...*