CHAPTER IV

PROGRAM LISTING

```
/*-------------------------------------------------------   */
/* This is the main program.  In this program the
   Eigenvalues & Eigenvectors of the matrix is computed.     */
/*--------------------------------------------------------- */
#include <stdio.h>
#include <math.h>
#include "mhb2.c"
#include "jb.c"
#include "given.c"
#define MAXROWS 5
#define MAXCOLS 5
 extern void read_mat(int r1,int c1,double temp[5][5]);
 extern void print_mat(int r1,int c1,double temp[5][5]);
 extern void mat_mul(int r1,int c1,double x1[5][5],
          double y1[5][5],double z1[5][5]);
 extern void jb_mat(int r1,int c1,double a[5][5],
          double p[5][5],double s[5][5],double ev[5][5],double
 v[5][5],
          double b[5][5],double max );
 extern void gn_mat(int r1,int c1,double a[5][5],double s[5][5],
               double p[5][5]);
```

```
main()

{

int rows,cols;

double a[MAXROWS][MAXCOLS];

double s[MAXROWS][MAXCOLS];

double p[MAXROWS][MAXCOLS];

double ev[MAXROWS][MAXCOLS];

double b[MAXROWS][MAXCOLS];

double v[MAXROWS][MAXCOLS];

double max;

int Method;

printf("Give the numbers of rows\n");

scanf("%d",&rows);

printf("Give the numbers of cols\n");

scanf("%d",&cols);

printf("Enter the elements of the matrix\n");

read_mat (rows,cols,a);

printf("\n1 Jacobi method\n");

printf("\n2 power method\n");

printf("\n3 Given's method\n");

printf("\Choose the method\n");

scanf("%d",&Method);
```

```
 switch (Method) {

              case 1:

                jb_mat(rows,cols,a,s,p,ev,v,b,max );
                break;

              case 2:
                power_mat(rows,cols,a);
                break;

              case 3:
                gn_mat(rows,cols,a,s,p);
                break;

            default:
                break;

           }
 }

/* The end of the program.  */
/* This file contains functions */

 void read_mat(int r1,int c1,double temp[5][5])

/* This function is for reading the matrix */

{
  int i,j;
  for(i=0;i<r1;i++)
  {
   for (j=0;j<c1;j++)
    {
     fflush(stdin);
     scanf("%lf",&temp[i][j]);

     }
   }
     return;
}
```

```
void print_mat(int r1,int c1,double temp[5][5])
/* This function is to print the matrix */
{
 int i,j, *fp;

 for(i=0;i<r1;i++)
 {
  for (j=0;j<c1;j++)

   {
    printf("%10.5lf",temp[i][j]);

    }
      printf("\n");
  }
    return;
}
```

```
double mat_max(int r1,int c1,double temp[5][5])
/* For finding the maximum element of a matrix  */
{
 int i,j;
 double tmp1,tmp,max;
 tmp = temp[0][0];
 max = fabs(tmp);
 for(i=0;i<r1;i++)
  {
   for(j=0;j<c1;j++)
   {
     tmp1 = temp[i][j];
     if(fabs( tmp1) >  max)
      max = fabs(tmp1);
    }
  }
    return(max);
}
```

```
double vect_max(int r1,double temp[5])
/* For finding the maximum element of a vector */
 {
  int i;
  double tmp1,tmp,max;
  tmp = temp[0];
  max = fabs(tmp);
  for(i=0;i<r1;i++)

   {
      tmp1 = temp[i];
      if(fabs( tmp1) >  max)
       max = fabs(tmp1);
  }
     return(max);
}
```

```
 void power_mat(int r1,int c1,double temp[5][5])
 /* The function power_mat
    Purpose
    This function obtains the largest eigenvalue
    and eigenvector of a matrix by
    Power method Description of parameters
    r1 The number of rows
    c1 The number of columns
    temp The given matrix  */
{
 int i,j;
 int k;
 double v[5],u[5];
 double t,mx,e;
 int *fp;
 e = 0.0001;
 mx = 0.00;
 for(i=0;i<r1;i++)
 {
 v[i] = 1;
 }

do {
    for (k=0; k<r1;k++)
     {
    u[k] = 0;
    for(j=0;j<c1;j++)
     u[k] = u[k] + temp[k][j]*v[j];
    }
  t = mx;
  mx = vect_max(r1,u);
  for(i = 0; i<r1;i++)
   v[i] = u[i]/mx;
} while(fabs(t - mx )>e);
 fp = fopen("out.dat","a");
 printf("\nThe largest eigenvalue\n");
 printf("%lf",mx);
 printf("\nThe eigenvector\n");
 for(i=0;i<r1;i++)
 {
  printf("%17.10lf",v[i]);
 }
 return;
}
```

```
   void mat_tran(int r1,int c1,int temp[5][5])
 {
  int i,j;
  for(i=0;i<r1;i++)
  {
   for (j=0;j<c1;j++)
    {
     tran[i][j] = temp[j][i];
     }
     return;
   }
}
```

```
  void mat_mul(int r1,int c1,double x1[5][5],double
y1[5][5],double z1[5][5])
  /* Matrix multiplication  */
 {
  int i,j,k;
  for(i=0;i<r1;i++)
  {
   for (j=0;j<c1;j++)
    {
      z1[i][j] = 0;
      for(k = 0; k < r1; k++)
      {
          z1[i][j] = z1[i][j] + x1[i][k]*y1[k][j];
       }
     }
  }
  return;
}
double fab(double x)
{
   double y;
    if (x<0)
     y=-x;
      else y=x;
    return(y);
}
```

```
/* Function jb_mat
   Purpose
    This function obtains Eigenvalues & Eigenvectors  of a real
    symmetricmatrix by JACOBI method
    Description of parameters
    r1   : The number of rows
    c1   : The number of columns
    a    : The given real symmetric matrix
    s    : The transformation matrix
    p    : The reduced diagonal matrix, whose diagonal elements are
           the eigenvalues of the matrix A and
   ev   : The n x n matrix which is used for the storage of the
          eigenvectors of the given matrix
   v    : The eigenvector
   b    : The given matrix is stored  in this matrix  */

void jb_mat(int r1,int c1,double a[5][5],double s[5][5],double
p[5][5],
      double ev[5][5],double v[5][5],double b[5][5],double max )
 {
     double y,z;
     int i,j,k,l,q,r,rows,cols;
     rows=r1;
     cols=c1;
     for (i=0;i<r1;i++)  {
```

```
     for(j=0;j<c1;j++)  {

          b[i][j] = a[i][j];

     }

}

for (i=0;i<r1;i++)  {

  for (j=0;j<c1;j++)  {

     if (i==j)

     ev[i][j] = 1;

     else

     ev[i][j] = 0;

  }

}

do

    {

     max = fabs(a[0][1]);

     q = 0;

     r = 1;

     for(k=0;k<r1;k++)  {

        for(l=k+1;l<c1;l++)  {

           if(fabs(a[k][l]) > (float)max)

           {

           max =fabs( a[k][l]);

           r = l;
          q = k;
           }
        }
     }
     for(k=0;k<r1;k++)
    {
    for(l=0;l<c1;l++)    {
       if(k == l)
      s[k][l] = 1;
       else
      s[k][l] = 0;
    }
}
  if(a[q][q] == a[r][r]  )

  {
     if (a[q][r] < 0 )
     z = -1;
     else
     z = 1;
  }
  else
  {
       y = 2*a[q][r]/(a[q][q]-a[r][r]);

       z = (-1+sqrt(1+y*y))/y;
```

```
}
s[q][q] = 1/sqrt(1+z*z);

s[r][r] = s[q][q];
s[q][r] = -z*s[q][q];
s[r][q] = -1*s[q][r];
for(k=0;k<r1;k++)
{
   for(l=0;l<c1;l++)
   {
   p[k][l] = 0;
   for(q=0;q<r1;q++)
   {

     for(r=0;r<c1;r++)
     {
        p[k][l] = p[k][l] + s[q][k]*s[r][l]*a[q][r];
      }
  }
   }
}
for(q=0;q<r1;q++)
{
   for(r=0;r<c1;r++)
   {
  a[q][r] = p[q][r];
```

```
                }
            }
            mat_mul(rows,cols,ev,s,v);
            for (i=0;i<r1;i++)  {

              for(j=0;j<c1;j++)  {
                   ev[i][j] = v[i][j];
              }
            }
        }
        while (max > 0.0001);
        printf("The given matrix\n");
        print_mat( rows,cols,b);
        printf("\nThe eigenvalues\n\n");
        for(i=0;i<rows;i++)
        {
           printf("   %17.10lf\n",p[i][i]);
        }
        printf("\nThe eigenvector\n\n");
        print_mat(rows,cols,v);
    }
```

```
void gn_mat ( int r1,int c1,double a[5][5],double s[5][5],

        double p[5][5])
/* Tridiagonal matrix  by Given's method. */

/* The function gn-mat

   Purpose

     The matrix reduces to tridiagonal form

   Description of parameters

   r1 : number of rows.

   c1 : number of columns.

   a  : The given matrix.

   s  : The orthogonal matrix.

   p  : The tridiagonal matrix.                    */

 {

double z;

int rows,cols,i,j,k,l,q,r;

rows = r1;

cols = c1;

q = 0;

r = 2;

for(i=0;i<rows;i++)

 {

   for(j=i+2;j<cols;j++)

    {

    q = i+1;

    r = j;
```

```
/* The S  matrix */
    for(k=0;k<rows;k++)
     {
       for(l=0;l<cols;l++)
     {
       if(k == l)
       s[k][l] = 1;
      else
       s[k][l] = 0;
      }
       }
       z = a[i][j]/a[i][i+1];
       s[q][q] = 1/sqrt(1+z*z);
       s[r][r] = s[q][q];
       s[q][r] = -z*s[q][q];
       s[r][q] = -1*s[q][r];
     for(k=0;k<rows;k++)
      {
     for(l=0;l<cols;l++)
      {
        p[k][l] = 0;
        for(q=0;q<rows;q++)
         {
           for(r=0;r<cols;r++)
            {
             p[k][l] = p[k][l] + s[q][k]*s[r][l]*a[q][r];
            }
           }
          }

         }
           for(k=0;k<rows;k++)
        {
          for(l=0;l<cols;l++)
           {
            a[k][l] = p[k][l];
          }
       }

      }
    }
        printf("\n");
      printf(" The tridiagonal matrix\n");
         print_mat(rows,cols,p);
}
```

(4.2) OUTPUT OF THE PROGRAM

```
 Give the numbers of rows : 4

 Give the numbers of cols: 4

 Enter the elements of the matrix

*** MENU ***

1 Jacobi method

2 power method

3 Given's method

Choose the method

2

The largest eigenvalue

 98.521699

The eigenvector

 -0.6039723419 ,

  1.0000000000 ,

 -0.2511351318,

   0.1489534465
```

```
Give the numbers of rows : 4

Give the numbers of cols: 4

Enter the elements of the matrix

       ** MENU **

 1 Jacobi method

 2 power method

 3 Given's method

 Choose the method

  1

  The given matrix

7.00000    3.00000    2.00000    1.00000

3.00000    9.00000   -2.00000    4.00000

2.00000   -2.00000   -4.00000    2.00000

1.00000    4.00000    2.00000    3.00000

  The eigenvalues

     5.7830521572

    12.7198575387

    -5.6002432141

     2.0973335182

    The eigenvector

 0.85021    0.48325   -0.17742   -0.11013

-0.44325    0.79027    0.24328   -0.34614

 0.23051    0.00833    0.90424    0.35937

-0.16593    0.37665   -0.30280    0.85960
```

```
Give the numbers of rows : 5

Give the numbers of cols : 5

Enter the elements of the matrix

       ** MENU **

 1 Jacobi method

 2 power method

 3 Given's method

 Choose the method

  3

  The given matrix

10.0000     1.00000    2.00000    3.00000    4.00000

1.00000     9.00000   -1.00000    2.00000   -3.00000

2.00000    -1.00000    7.00000    3.00000   -5.00000

3.00000     2.00000    3.00000    12.0000    1.00000

4.00000    -3.00000   -5.00000   -1.00000    15.0000

   The Tridiagonal Matrix

 10.0000    5.47723    0.00000    0.00000    0.00000

 5.47723    10.8333    4.56321    0.00008    0.00000

 0.00000    4.56321    8.56682   -5.62814    0.00000

 0.00000    0.00000   -5.62814    14.13994   0.51925

 0.00000    0.00000    0.00008     0.51925   9.36990
```

(4.3) CONCLUSION

While computing the eigenvalues and eigenvectors of a Hermitian or real symmetric matrix, the Jacobi method can be used. For a more general eigenvalue problem of the type $Ax = \lambda Bx$, where A is real symmetric and positive definite, again the Jacobi method may be used.

If the given square matrix A is nonsymmetric and real, and a good symmetrizer can be computed, then we compute a symmetrizer of A and next solve this symmetric eigenvalue problem, using the Jacobi method.

When we want to compute only the largest magnitude eigenvalue and its corresponding eigenvector, then the Power method is used. It should be noted that no stable method is yet available for obtaining the eigenvalues of a nonsymmetric matrix.

In Jacobi method at each stage in the reduction of a real symmetric matrix to diagonal form, one must work with complete matrix. Here the elements reduced to zero in one rotation may become nonzero in other rotation, so the process may be infinte in case of larger matrix.

In Given's method an element reduced to zero in one rotation never becomes nonzero in latter rotations. After reducing the matrix to its tridiagonal form one can obtain its eigenvalues using Sturm sequence property or any other method for obtaining roots of a polynomial.

*****

REFERENCES

(1) Baker C.T.H.(1978), The Numerical Treatment of Integral Equation.

(2) Clint and Jenning.(1970), The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration. Computer Journal 13, 76-80.

(3) Collatz.(1966), Functional Analysis and Numerical Methods.

(4) Krishanmurthy E.V. and S. K. Sen. (1991), Numerical algorithms.

(5) Glourly A.R. and G. A. Watson. (1973), Computational Methods for Matrix Eigenproblems.

(6) Wilkinson J.H.(1971), Linear Algebra.

(7) Balagurusamy (1990), Programming in C.

(8) Dennis M.Ritchie. (1989), The C programming Language.

(9) A.Ramachandra Rao. and P.Bhimasankaram. (1992), Linear Algebra.