```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 4
#define EPS pow(10.0, -15.0)

/* 行列の入力 */
void input_matrix( double **a, char c, FILE *fin, FILE *fout);
/* 行列の領域確保 */
double **dmatrix(int nr1, int nr2, int nl1, int nl2);
/* 行列の領域解放 */
void free_dmatrix(double **a, int nr1, int nr2, int nl1, int nl2);
/* ベクトル領域の確保 */
double *dvector(int i, int j);
/* ベクトル領域の解放 */
void free_dvector(double *a, int i);
/* ハウスホルダー法 */
void householder( double **a, int n );
/* qr法 */
void qr( double **a, double eps, int n );

int main(void)
{
  FILE *fin, *fout;
  double **a, eps=pow(10.0,-8.0);
  int i, j;

  a = dmatrix( 1, N, 1, N );  /* 行列領域の確保 */

  /* ファイルのオープン */
  if ( (fin = fopen( "input_eigen.dat", "r")) == NULL )
  {
      printf("ファイルが見つかりません : input_eigen.dat \n");
      exit(1);
  }
  if( (fout = fopen( "result_eigen.dat", "w")) == NULL )
  {
    printf("ファイルが作成できません : result_eigen.dat \n");
    exit(1);
   }

  input_matrix( a, 'A', fin, fout);    /* 行列 A の入出力 */
  householder( a, N );                 /* ハウスホルダー法 */
  qr( a, eps, N );                     /* QR法 */

  /* 結果の出力 */
  printf( "QR法の結果は\n" );
  for( i = 1 ; i <= N ; i++)
  {
    for( j = 1 ; j <= N ; j++)
    {
      printf( "%10.7f\t", a[i][j] );
    }
    printf( "\n" );
  }
  printf( "固有値は\n" );
  for( i = 1; i <= N; i++ ) printf( "%10.7f\t", a[i][i] );
  printf( "\n" );

  /* 領域の解放 */
  free_dmatrix( a, 1, N, 1, N );

  /* ファイルのクローズ */
  fclose(fin); fclose(fout);

  return 0;
}

/* QR法 */
void qr( double **a, double eps, int n )
{
  int i, j, k, m;
  double **q, *work, r, s, sint, cost, tmp;

  /* 領域の確保 */
  q = dmatrix( 1, n, 1, n ); work = dvector( 1, n );

  m = n;
  while ( m > 1 )
  {
    /* 収束判定 */
    if ( fabs(a[m][m-1]) < eps )
    {
      m = m - 1; continue;
    }

    /* 原点移動 */
    s = a[n][n];
    if ( m == n ) s = 0.0;   /* m=n のときは原点移動なし */
    for ( i = 1; i <= m; i++) a[i][i] -= s ;

    /* QR 法 */

    for ( i = 1; i <= m; i++)
    {
      for( j = 1; j <= m; j++ )
      {
        q[i][j] = 0.0;         /* Q を m x m 単位行列で初期化 */
      }
      q[i][i] = 1.0;
    }

    /* R と Q の計算 */
    for ( i = 1; i <= m-1; i++)
    {
      r = sqrt( a[i][i]*a[i][i] + a[i+1][i]*a[i+1][i] );
      if ( r == 0.0 )
      {
        sint = 0.0; cost = 0.0;
      }
      else
      {
        sint = a[i+1][i]/r; cost = a[i][i]/r;
      }
      for ( j = i+1; j <= m; j++)
      {
        tmp = a[i][j]*cost + a[i+1][j]*sint;
        a[i+1][j] = -a[i][j]*sint + a[i+1][j]*cost;
        a[i][j] = tmp;
      }
      a[i+1][i] = 0.0;
      a[i][i] = r;
      for ( j = 1;  j <= m; j++ )
      {                          /* Q は P の転置 */
        tmp = q[j][i]*cost + q[j][i+1]*sint;
        q[j][i+1] = -q[j][i]*sint + q[j][i+1]*cost;
        q[j][i] = tmp;
      }
    }

    /* RQ の計算 */
    for ( i = 1; i <= m; i++)
    {
      for ( j = i; j <= m ; j++ ) work[j] = a[i][j];
      for ( j = 1; j <= m ; j++ )
      {
        tmp = 0.0;
        for( k = i; k <= m; k++ ) tmp += work[k]*q[k][j];
        a[i][j] = tmp;
      }
    }

    /* 原点移動後の処理 */
    for ( i = 1; i <= m; i++ ) a[i][i] = a[i][i] + s;
  }

  /* 領域の解放 */
  free_dmatrix( q, 1, n, 1, n ); free_dvector( work, 1 );
}

/* ハウスホルダー法 */
void householder( double **a, int n )
{
  int i, j, k;
  double *u, *f, *g, gamma, s, ss, uu;

  /* ベクトル領域の確保 */
  u = dvector( 1, n ); f = dvector( 1, n ); g = dvector( 1, n );

  for ( k = 1; k <= n-2; k++)
  {
    /* v の計算 */
    for ( i = 1; i <= k; i++) u[i] = 0.0;
    for ( i = k+1; i <= n; i++) u[i] = a[i][k];

    /* s の計算 */
    ss = 0.0;
    for ( i = k+2; i <= n; i++ ) ss += u[i]*u[i];
    if ( fabs(ss) < EPS ) continue;  /* 消去が必要ない場合の処理 */
    s = sqrt( ss + u[k+1]*u[k+1] );
    if ( u[k+1] > 0.0 ) s = -s;

    /* u の計算 */
    u[k+1] -= s;
    uu = sqrt( ss + u[k+1]*u[k+1] );
    for ( i = k+1; i <= n; i++) u[i] /= uu;

    /* f, g の計算 */
    for ( i = 1; i <= n; i++)
    {
      f[i] = 0.0; g[i] = 0.0;
      for ( j = k+1; j <= n; j++ )
      {
        f[i] += a[i][j]*u[j];
        g[i] += a[j][i]*u[j];
      }
    }

    /* gamma の計算 */
    gamma = 0.0;
    for ( j = 1; j <= n; j++) gamma += u[j]*g[j];

    /* f, g の計算 */
    for ( i = 1; i <= n; i++)
    {
      f[i] -=  gamma * u[i];
      g[i] -=  gamma * u[i];
    }

    /* A の計算 */
    for ( i = 1; i <= n; i++)
    {
      for ( j = 1; j <= n; j++ )
      {
        a[i][j] = a[i][j] - 2.0*u[i]*g[j] - 2.0*f[i]*u[j];
      }
    }

  }

  /* ベクトル領域の解放 */
  free_dvector( u, 1 ); free_dvector( f, 1 ); free_dvector( g, 1 );
}

/* a[1...N][1...N] の入力 */
void input_matrix(double **a, char c, FILE *fin, FILE *fout)
{
  int i, j;

  fprintf(fout, "行列%c は次の通りです\n", c);
  for (i = 1; i <= N; i++)
  {
    for (j = 1; j <= N; j++)
    {
      fscanf(fin, "%lf", &a[i][j]);
      fprintf(fout, "%5.2f\t", a[i][j]);
    }
    fprintf(fout, "\n");
  }
}

/* b[1...N]の入力 */
void input_vector(double *b, char c, FILE *fin, FILE *fout)
{
  int i;

  fprintf(fout, "ベクトル%c は次の通りです\n", c);
  for (i = 1; i <= N; i++)
  {
    fscanf(fin, "%lf", &b[i]);
    fprintf(fout, "%5.2f\t", b[i]);
    fprintf(fout, "\n");
  }
}

double **dmatrix(int nr1, int nr2, int nl1, int nl2)
{
  int i, nrow, ncol;
  double **a;

  nrow = nr2 - nr1 + 1; /* 行の数 */
  ncol = nl2 - nl1 + 1; /* 列の数 */

  /* 行の確保 */
  if ((a = malloc(nrow * sizeof(double *))) == NULL)
  {
    printf("メモリが確保できません (行列 a)\n");
    exit(1);
  }
  a = a - nr1; /* 行をずらす */

  /* 列の確保 */
  for (i = nr1; i <= nr2; i++)
    a[i] = malloc(ncol * sizeof(double));
  for (i = nr1; i <= nr2; i++)
    a[i] = a[i] - nl1; /* 列をずらす */

  return (a);
}

void free_dmatrix(double **a, int nr1, int nr2, int nl1, int nl2)
{
  int i;

  /* メモリの解放 */
  for (i = nr1; i <= nr2; i++)
    free((void *)(a[i] + nl1));
  free((void *)(a + nr1));
}

double *dvector(int i, int j)
{
  double *a;

  if ((a = malloc(((j - i + 1) * sizeof(double)))) == NULL)
  {
    printf("メモリが確保できません (from dvector) \n");
    exit(1);
  }

  return (a - i);
}

void free_dvector(double *a, int i)
{
  free((void *)(a + i)); /* (void *) 型へのキャストが必要 */
}

/* 行列 a[1...N][1...N] とベクトル b[1...N] との積 c <- Ab */
void matrix_vector_product(double **a, double *b, double *c)
{
  double wk;
  int i, j;

  for (i = 1; i <= N; i++)
  {
    wk = 0.0;
    for (j = 1; j <= N; j++)
    {
      wk += a[i][j] * b[j];
    }
    c[i] = wk;
  }
}

/* ベクトル a[m...n] と b[m...n] の内積を計算する */
double inner_product( int m, int n, double *a, double *b)
{
  int i;
  double s = 0.0;

  for( i = m; i <= n; i++) s += a[i]*b[i];

  return s ;
}
```