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

#define N 4

/* 行列の入力 */
void input_matrix( double **a, char c, FILE *fin, FILE *fout);
/* ベクトルの入力 */
void input_vector( double *b, 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);
/* 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 );
/* 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);
/* べき乗法 */
void power_method( double **a, double *x, FILE *fout);


int main(void)
{
  FILE *fin, *fout;
  double **a, *x;

  a = dmatrix( 1, N, 1, N );  /* 行列領域の確保 */
  x = dvector( 1, N );        /* ベクトル領域の確保 */

  /* 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 */
  input_vector( x, 'x', fin, fout);    /* ベクトル x の入出力 */
  power_method( a, x, fout );          /* べき乗法 */

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

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

  return 0;
}

/* べき乗法 */
void power_method( double **a, double *x,  FILE *fout)
{
  int i,k=0;                    /* k は反復回数 */
  double eps=pow(10.0,-8.0);    /* eps=10^(-8)とする */
  double v2, v2s, *v, lambda;

  v = dvector( 1, N );          /* ベクトル領域の確保 */

  do
  {
    matrix_vector_product( a, x, v );
    lambda = inner_product( 1, N, v, x );
    v2 = inner_product( 1, N, v, v );
    v2s = sqrt(v2);
    for( i = 1; i <= N ; i++) x[i]=v[i]/v2s;
    ++k;
   }while( fabs(v2-lambda*lambda) >= eps );

  fprintf(fout, "収束回数は %d\n",k);
  fprintf(fout, "絶対値最大固有値 lambda は %f\n",lambda);
  fprintf(fout, "これに対応する固有ベクトルは次のとおりです\n");

  for( i = 1; i <= N; i++ )
  {
    fprintf( fout, "v[%d]=%f\n", i, x[i] );
  }

  free_dvector( v, 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 ;
}
```