«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет електроніки

Кафедра мікроелектроніки

Лабораторна робота №11

Варіант №21

Виконав: студент групи ДП-82
Мнацаканов Антон
Перевірив: Домбругов М.Р.

Київ-2020

Мета роботи: вивчення алгоритму і налаштування програми для знаходження власних векторів і власних чисел симетричних матриць методом обертань Якобі.

Що зробити: знайти власні числа і власні вектори симетричної матриці **A** шляхом її декомпозиції методом обертань Якобі в добуток виду $\mathbf{A} = \mathbf{QDQ}^T$, де **D** – діагональна матриця власних чисел, а **Q** – ортогональна матриця власних векторів. Впевнитись у правильності результату шляхом перевірки співвідношень $\mathbf{Q}^T\mathbf{Q} = \mathbf{E}$ та $\mathbf{QDQ}^T = \mathbf{A}$. Додатково – впевнитись в інваріантності сліду матриці **A** та її норми Фробеніуса при послідовних ітераціях. Оцінити порядок збіжності методу обертань Якобі.

Хід роботи

Варіанти 1, 6, 11, 16, 21: матриця 4 x 4, $a_{11} = V/2$, $a_{13} = a_{31} = 8.0$

Фрагмент коду на C:

```
#include<stdio.h>
#include <stdlib.h>
#include <math.h>
#define EPS 1e-8

int read_n(FILE* f){
  int nn;
  fscanf(f,"%d", &nn);
  return nn;
}

double** init_matrix(int n){
  double ** res;
  res = malloc(n*sizeof(double*));
  if(!res){
    return NULL;
  }
```

```
  for(int i=0; i<n; ++i){
    res[i] = malloc(n*sizeof(double));
    if(!res[i]){
      return NULL;
    }
  }
  return res;
}

double** init_one_matrix(int n){
  double** res = init_matrix(n);
  for(int i=0; i<n;++i)
  {
    for (int j = 0; j < n; j++)
    {
      if(i==j)
      {
       res[i][j]=1;
      }
       else {
         res[i][j]=0;
       }
     }
   }
  return res;
}
double** read_symm_matrix(FILE* f, int n){
  double ** res = init_matrix(n);
  for(int i=0; i<n; ++i)
  {
      for(int j=0; j<n; ++j)
      {
          fscanf(f,"%lf", &res[i][j]);
      }
  }
  return res;
}

double** transpon_matrix(int n, double** matrix){
  double** res = init_matrix(n);
  for(int i=0;i<n;++i){
    for(int j=0; j<n; ++j){
      res[j][i]=matrix[i][j];
    }
  }
  return res;
}

double** mul_matrix(int n, double** m1, double** m2){
  double** res = init_matrix(n);
```

```
  for(int i=0; i<n; ++i){
    for(int j=0; j<n; ++j){
      res[i][j]=0;
      for(int k=0;k<n; ++k){
        res[i][j]+=m1[i][k]*m2[k][j];
      }
    }
  }
  return res;
}

int print_matrix(int i, double** m);

int jacobi(int n, double** d, double** q, double eps)
{
  for(int i=0; i<n;++i)
  {
    for (int j = 0; j < n; j++)
    {
      if(i==j)
      {
       q[i][j]=1;
      }
       else {
         q[i][j]=0;
       }
    }
   }
   do {
     double dlm=d[1][0];
     int l=1;
     int m=0;
     for (int  i = 1; i < n; i++)
     {
       for (int j = 0; j < (i-1); j++)
       {
         if(fabs(d[i][j]) > fabs(dlm))
         {
           dlm=d[i][j];
           l=i;
           m=j;
         }
       }
     }

     if (fabs(dlm)<eps) break;

     double t=(d[l][l]-d[m][m])/(2.0f*d[l][m]);
     double u=t/sqrt(1.0f+t*t) ;
     double c=sqrt((1.0f+u)/2.0f);
```

```
    double s=sqrt((1.0f-u)/2.0f);

    for(int i=0; i<n; i++)
    {
     double qil = q[i][l]*c+q[i][m]*s;
     double qim = -q[i][l]*s+q[i][m]*c;
     q[i][l]=qil;
     q[i][m]=qim;
    }
    double dll=d[l][l]*c*c+2*d[l][m]*c*s+d[m][m]*s*s;
    double dmm=d[l][l]*s*s-2*d[l][m]*c*s+d[m][m]*c*c;
    d[l][l]=dll;
    d[m][m]=dmm;
    d[l][m]=0;
    d[m][l]=0;

    for (int i = 0; i < n; i++)
    {
     if(i!=l && i!=m)
     {
      double dil= d[i][l]*c+d[i][m]*s;
      double dim=-d[i][l]*s+d[i][m]*c;
      d[i][l]=dil;
      d[l][i]=d[i][l];
      d[i][m]=dim;
      d[m][i]=d[i][m];
     }
    }
   }while(1);
   return 0;
}

int print_matrix(int n, double** m)
{
  for(int i=0; i<n; ++i)
  {
     for(int j=0; j<n; ++j)
     {
        printf(" %e\t ", m[i][j]);
     }
     printf(" \n ");
  }
    printf(" \n ");
  return 0;
  }

int compare_matrix(int n, double** m1, double** m2){
  for(int i=0;i<n;++i){
    for(int j=0; j<n; ++j){
      if(fabs(m1[i][j]-m2[i][j])>EPS)// сравнение с точностью
```

```
      {
        printf("%d %d %e %e\n",i,j, m1[i][j], m2[i][j]);
        return 0;
      }
    }
  }
  return 1;
}

double** copy_matrix(int n, double** m){
  double** res=init_matrix(n);
  for(int i=0;i<n;++i){
    for(int j=0; j<n; ++j){
      res[i][j]=m[i][j];
    }
  }
  return res;
}

int main(int argc, char* argv[]){
  if(argc != 2){
    printf("Usage:\n\t %s <file with matrix>\n", argv[0]);
    return 1;
  }
  FILE *f = fopen(argv[1], "r");
  if(!f){
    printf("error: can't open file %s\n", argv[1]);
    return 2;
  }
  int n = read_n(f);
  double** a = read_symm_matrix(f, n);
  double** d = copy_matrix(n,a);
  double** q = init_matrix(n);
  print_matrix(n, d);
  jacobi(n, d, q, EPS);
  print_matrix(n, q);
  print_matrix(n, d);
  double** qt = transpon_matrix(n, q);
  print_matrix(n, qt);
  double** qqt = mul_matrix(n, q, qt);
  print_matrix(n, qqt);
  double** e=init_one_matrix(n);
  printf("Qt*Q %s E\n", compare_matrix(n, qqt, e)?"==":"!=");
  double** qd = mul_matrix(n, q, d);
  double** qdqt = mul_matrix(n, qd, qt);
  print_matrix(n, qdqt);
  printf("Q*D*Qt %s A\n", compare_matrix(n, qdqt, a)?"==":"!=");
  return 0;
}
```

Основний код був перероблений з цього на простий С.

```
DO                                      ' початок ітерацій

   Dlm=D[2,1] : l=2 : m=1               ' пошук найбільш.
   FOR i=2 TO n                         ' недіагоального
      FOR j=1 TO i-1                    ' елемента
         IF abs(D[i,j]) > abs(Dlm) THEN
             Dlm=D[i,j] : l=i : m=j
         END IF
      NEXT j
   NEXT i
                                        ' вихід, якщо до-
   IF abs(Dlm)<eps THEN EXIT LOOP       ' сягнута точність

   t=(D[l,l]-D[m,m])/(2*D[l,m])         ' cos та sin
   u=t/SQR(1+t*t)                       ' кута повороту
   c=SQR((1+u)/2)
   s=SQR((1-u)/2)

   FOR i=1 TO n                         ' перерахунок Q
      Qil= Q[i,l]*c+Q[i,m]*s
      Qim=-Q[i,l]*s+Q[i,m]*c
      Q[i,l]=Qil
      Q[i,m]=Qim
   NEXT i

   Dll=D[l,l]*c*c+2*D[l,m]*c*s+D[m,m]*s*s  ' перерахунок D:
   Dmm=D[l,l]*s*s-2*D[l,m]*c*s+D[m,m]*c*c  ' 4 елементи
   D[l,l]=Dll
   D[m,m]=Dmm
   D[l,m]=0 : D[m,l]=0

   FOR i=1 TO n                         ' решта елементів
      IF i<>l AND i<>m THEN
         Dil= D[i,l]*c+D[i,m]*s
         Dim=-D[i,l]*s+D[i,m]*c
         D[i,l]=Dil : D[l,i]=D[i,l]
         D[i,m]=Dim : D[m,i]=D[i,m]
      END IF
   NEXT i

LOOP                                    ' кінець циклу

END SUB
```

Результати

Ось це наша початкова матриця

```
antonmnacakanov@MacBook-Air exFAT % ./a.out lab11.txt
 1.050000e+01    3.000000e+00    8.000000e+00    2.000000e+00
  3.000000e+00    1.100000e+01    1.000000e+00    5.000000e+00
  8.000000e+00    1.000000e+00    1.200000e+01    4.000000e+00
  2.000000e+00    5.000000e+00    4.000000e+00    1.300000e+01
```

Ось це ми вивели ортогональну матрицю її власних векторів Q

```
  -4.084166e-01   -5.229095e-01   -3.369386e-01   6.680074e-01
  5.442371e-01    -3.758439e-01   -6.844582e-01   -3.066991e-01
  -4.728556e-01   -5.764517e-01   2.220955e-01    -6.283188e-01
  5.598298e-01    -5.030014e-01   6.071762e-01    2.547886e-01
```

А це діагональна матриця власних чисел D

```
1.302310e+01    3.306862e-09    8.065739e-12    8.138017e-16
3.306862e-09    2.339926e+01    -1.818963e-08   0.000000e+00
8.065739e-12    -1.818963e-08   7.716877e+00    1.315093e-08
8.138017e-16    0.000000e+00    1.315093e-08    2.360763e+00
```

Це транспонована матриця

```
-4.084166e-01   5.442371e-01   -4.728556e-01   5.598298e-01
-5.229095e-01   -3.758439e-01  -5.764517e-01   -5.030014e-01
-3.369386e-01   -6.844582e-01  2.220955e-01    6.071762e-01
6.680074e-01    -3.066991e-01  -6.283188e-01   2.547886e-01
```

```
Qt*Q == E
```

```
1.000000e+00    -1.387779e-16   0.000000e+00    -1.387779e-16
-1.387779e-16   1.000000e+00    -8.326673e-17   -9.714451e-17
0.000000e+00    -8.326673e-17   1.000000e+00    0.000000e+00
-1.387779e-16   -9.714451e-17   0.000000e+00    1.000000e+00
```

```
Q*D*Qt == A
```

```
1.050000e+01    3.000000e+00    8.000000e+00    2.000000e+00
 3.000000e+00   1.100000e+01    1.000000e+00    5.000000e+00
 8.000000e+00   1.000000e+00    1.200000e+01    4.000000e+00
 2.000000e+00   5.000000e+00    4.000000e+00    1.300000e+01
```

Висновок: в даній лабораторній роботі я написав программу знаходження власних чисел будь-якої симетричної матриці взявши за основу метод Якобі, також вивів проміжні результати в формі транспонованої матриці а також результати цих двох обчислень.

$$Q^TQ = E \text{ та } QDQ^T = A$$

Прекрасно.

Контрольні запитання до захисту:

Нехай є ортогональна матриця P розміром 2 × 2

$$\mathbf{P} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}$$

та симетрична матриця D

$$\mathbf{D} = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

Обрахуйте матрицю D' = P^TDP та доведіть, що D' та D мають однакові

1) **слід**

2) **норму Фробеніуса**

1) **слід**

a*cos^2(x)+c*sin^2(x) +2*b*sin(x)*cos(x)	sin(x)*cos(x)(c-a)-b
sin(x)*cos(x)(c-a)-b	a*sin^2(x) +c*cos^2(x)-2*b*sin(x)*cos(x)

Типо матрица:

a*cos^2(x)+c*sin^2(x)+~~2*b*sin(x)*cos(x)~~+a*sin^2(x)
+c*cos^2(x)-~~2*b*sin(x)*cos(x)~~ =a* (cos^2(x)+sin^2(x)) +c*(sin^2(x)+cos^2(x))
=>a+c - вроде так

2) **А норму Фробеніуса мне не удалось доказать там эти косинусы и синусы в слишком больших степенях выходят и ещё и под корнем (чот сильно сложно :D) и я запутался в конце**