**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС**

**«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»**

**НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

**КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ**

**Лабораторна робота №6**

**з курсу «Чисельні методи»**

**тема: «ПОШУК ВЛАСНИХ ЧИСЕЛ ТА ВЕКТОРІВ**

**СИМЕТРИЧНИХ МАТРИЦЬ»**

**Виконав: студент 3 курсу**

**групи КА-23**

**Деундяк О.В.**

**Прийняла: Кузнєцова Н. В.**

**Київ – 2014р.**

**Умова**



**Текст програми:**

size\_t Matrix::eigenValuesJacobi(vector<double> &eigenValues, vector< vector<double> > &eigenVector, size\_t itLimit /\*= 1000\*/) const

{

GlobalLog << "####JACOBI METHOD####" << std::endl << std::endl;

Matrix Ai(\*this), B(n, true);

size\_t it;

for (it = 0; it < itLimit; it++)

{

GlobalLog << "Iteration #" << it << std::endl;

GlobalLog << "Matrix:" << std::endl << Ai << std::endl;

size\_t i, j;

Ai.findMax(i, j);

double fi = (abs(Ai[i][i] - Ai[j][j]) > eps) ? 0.5 \* atan(2 \* Ai[i][j] / (Ai[i][i] - Ai[j][j])) : M\_PI / 4;

GlobalLog << "Maximal element is: " << Ai[i][j] << " which is in position (" << i << "," << j << ")" << std::endl << std::endl;

GlobalLog << "alpha=a[i][i]=" << Ai[i][i] << std::endl

<< "beta=a[j][j]=" << Ai[j][j] << std::endl

<< "gamma=a[i][j]=" << Ai[i][j] << std::endl << std::endl;

GlobalLog << "fi=" << fi << std::endl

<< "cos(fi)=" << cos(fi) << std::endl

<< "sin(fi)=" << sin(fi) << std::endl << std::endl;

GlobalLog << "sigma=" << Ai.sigma() << std::endl

<< "omega=" << Ai.omega() << std::endl

<< "sum=" << Ai.sigma() + 2 \* Ai.omega() << std::endl << std::endl;

if (pow(Ai.omega(), 0.5) < eps)

break;

Matrix U(n, 0.0);

for (size\_t \_i = 0; \_i < n; \_i++)

U[\_i][\_i] = (\_i != i && \_i != j) ? 1 : 0;

U[i][i] = cos(fi);

U[j][j] = cos(fi);

U[i][j] = -sin(fi);

U[j][i] = sin(fi);

// iteration

Ai = U.Trans() \* Ai \* U;

B = B \* U;

}

eigenValues.clear();

for (size\_t i = 0; i < n; i++)

{

eigenValues.push\_back(Ai[i][i]);

eigenVector.push\_back(B.GetColumn(i));

}

GlobalLog << "---RESULTS---" << std::endl

<< "Finished in " << it << " iterations" << std::endl << std::endl;

for (size\_t i = 0; i < eigenValues.size(); i++)

{

GlobalLog << "Eigen value #" << i + 1 << ": Value=" << eigenValues[i] << "\tVector: " << eigenVector[i] << std::endl

<< "Error=" << \*this \* eigenVector[i] - eigenValues[i] \* eigenVector[i] << std::endl << std::endl;

}

return it;

}

size\_t Matrix::eigenValuesScalar(double eigenValue, std::vector<double> &eigenVector, size\_t itLimit) const

{

GlobalLog << "####SCALAR METHOD####" << std::endl << std::endl;

vector<double> &y = eigenVector;

y.clear();

y = vector<double>(n, 1);

eigenValue = 0;

double old = 0;

size\_t it;

for (it = 0; it < itLimit; it++)

{

GlobalLog << "Iteration #" << it << std::endl;

GlobalLog << "lambda=" << eigenValue << " y=" << y << std::endl << std::endl;

if (abs(old - eigenValue) < eps && it > 1)

break;

old = eigenValue;

eigenValue = (((\*this \* y) \* y) / (y \* y));

y = \*this \* y;

y = y \* pow(y \* y, -0.5);

}

GlobalLog << "---RESULTS---" << std::endl

<< "Finished in " << it << " iterations" << std::endl << std::endl;

GlobalLog << "Eigen value=" << eigenValue << "\tVector: " << eigenVector << std::endl

<< "Error=" << \*this \* eigenVector - eigenValue \* eigenVector << std::endl << std::endl;

return it;

}

void Matrix::CCInvers(Matrix& B, Matrix& C) const

{

for (size\_t i = 0; i < n; i++)

for (size\_t j = 0; j < n; j++)

{

if (j == n - 1)

B[i][j] = A[i][j];

else if (i == j + 1)

B[i][j] = 1;

else

B[i][j] = 0;

if (j == 0)

C[i][j] = ((i < n - 1) ? -A[i + 1][n - 1] : 1) / A[0][n - 1];

else if (i == j - 1)

C[i][j] = 1;

else

C[i][j] = 0;

}

}

void Matrix::eigenValuesDanilevski(std::vector<double> &eigenValuesvector, std::vector< std::vector<double> > &eigenVector) const

{

GlobalLog << "####DANILEVSKI METHOD####" << std::endl << std::endl;

Matrix B(\*this);

Matrix C1(n), C2(n), S1(n, true), S2(n, true);

for (size\_t i = 0; i < n - 1; i++)

{

B.CCInvers(C1, C2);

GlobalLog << "ITERATION #" << i << std::endl;

GlobalLog << "C\n" << C1 << std::endl << std::endl;

GlobalLog << "C-1\n" << C2 << std::endl << std::endl;

S2 = C2 \* S2;

S1 = S1 \* C1;

B = C2 \* B \* C1;

GlobalLog << "B\n" << B << std::endl << std::endl;

}

std::vector<double> pol = B.GetColumn(n - 1);

pol.push\_back(1);

if (n % 2 == 0)

pol = pol \* (-1);

pol[pol.size() - 1] = (n % 2 == 0) ? 1 : 0;

Polynomial p(pol);

GlobalLog << "EQUATION: " << std::endl

<< p << std::endl;

eigenValuesvector.push\_back(p.GetRootBin(9, 10, eps));

eigenValuesvector.push\_back(p.GetRootBin(5, 6, eps));

eigenValuesvector.push\_back(p.GetRootBin(4, 5, eps));

eigenValuesvector.push\_back(p.GetRootBin(2, 3, eps));

GlobalLog << "EIGEN VALUES: " << eigenValuesvector << std::endl << std::endl;

for (size\_t i = 0; i < eigenValuesvector.size(); i++)

{

vector<double> v(n);

v[v.size() - 1] = 1;

for (size\_t j = v.size() - 1; j != 0; j--)

v[j - 1] = eigenValuesvector[i] \* v[j] - B[j][n-1] \* v[v.size() - 1];

v = S1 \* v;

eigenVector.push\_back(v);

GlobalLog << "EIGEN VECTOR #" << i + 1 << ' ' << v << std::endl;

}

}

**Результати роботи програми**

####JACOBI METHOD####

Iteration #0

Matrix:

6.92 1.16 0.91 1.15

1.16 3.5 1.3 0.16

0.91 1.3 6.1 2.1

1.15 0.16 2.1 5.44

Maximal element is: 2.1 which is in position (2,3)

alpha=a[i][i]=6.1

beta=a[j][j]=5.44

gamma=a[i][j]=2.1

fi=0.707464

cos(fi)=0.760012

sin(fi)=0.649909

sigma=126.94

omega=9.6218

sum=146.184

Iteration #1

Matrix:

6.92 1.16 1.43901 0.282598

1.16 3.5 1.092 -0.723279

1.43901 1.092 7.89577 0

0.282598 -0.723279 0 3.64423

Maximal element is: 1.43901 which is in position (0,2)

alpha=a[i][i]=6.92

beta=a[j][j]=7.89577

gamma=a[i][j]=1.43901

fi=-0.621958

cos(fi)=0.812739

sin(fi)=-0.582628

sigma=135.76

omega=5.2118

sum=146.184

Iteration #2

Matrix:

5.88842 0.306547 -4.44089e-016 0.229678

0.306547 3.5 1.56336 -0.723279

-8.88178e-016 1.56336 8.92735 0.164649

0.229678 -0.723279 0.164649 3.64423

Maximal element is: 1.56336 which is in position (1,2)

alpha=a[i][i]=3.5

beta=a[j][j]=8.92735

gamma=a[i][j]=1.56336

fi=-0.261332

cos(fi)=0.966047

sin(fi)=-0.258368

sigma=139.901

omega=3.14106

sum=146.184

Iteration #3

Matrix:

5.88842 0.296139 0.0792019 0.229678

0.296139 3.08188 1.11022e-016 -0.741261

0.0792019 4.44089e-016 9.34547 -0.0278131

0.229678 -0.741261 -0.0278131 3.64423

Maximal element is: -0.741261 which is in position (1,3)

alpha=a[i][i]=3.08188

beta=a[j][j]=3.64423

gamma=a[i][j]=-0.741261

fi=0.604123

cos(fi)=0.823001

sin(fi)=0.56804

sigma=144.79

omega=0.696965

sum=146.184

Iteration #4

Matrix:

5.88842 0.374189 0.0792019 0.0208063

0.374189 2.57026 -0.015799 0

0.0792019 -0.015799 9.34547 -0.0228902

0.0208063 -2.22045e-016 -0.0228902 4.15585

Maximal element is: 0.374189 which is in position (0,1)

alpha=a[i][i]=5.88842

beta=a[j][j]=2.57026

gamma=a[i][j]=0.374189

fi=0.110914

cos(fi)=0.993855

sin(fi)=0.110687

sigma=145.889

omega=0.147497

sum=146.184

Iteration #5

Matrix:

5.9301 0 0.0769665 0.0206785

5.55112e-017 2.52858 -0.0244685 -0.00230299

0.0769665 -0.0244685 9.34547 -0.0228902

0.0206785 -0.00230299 -0.0228902 4.15585

Maximal element is: 0.0769665 which is in position (0,2)

alpha=a[i][i]=5.9301

beta=a[j][j]=9.34547

gamma=a[i][j]=0.0769665

fi=-0.0225201

cos(fi)=0.999746

sin(fi)=-0.0225182

sigma=146.169

omega=0.00747942

sum=146.184

Iteration #6

Matrix:

5.92836 0.000550986 0 0.0211887

0.000550986 2.52858 -0.0244623 -0.00230299

-4.16334e-016 -0.0244623 9.3472 -0.0224188

0.0211887 -0.00230299 -0.0224188 4.15585

Maximal element is: -0.0244623 which is in position (1,2)

alpha=a[i][i]=2.52858

beta=a[j][j]=9.3472

gamma=a[i][j]=-0.0244623

fi=0.00358752

cos(fi)=0.999994

sin(fi)=0.00358751

sigma=146.18

omega=0.00155557

sum=146.184

Iteration #7

Matrix:

5.92836 0.000550983 -1.97667e-006 0.0211887

0.000550983 2.5285 -1.73472e-018 -0.0023834

-1.97667e-006 3.1225e-016 9.34729 -0.0224104

0.0211887 -0.0023834 -0.0224104 4.15585

Maximal element is: -0.0224104 which is in position (2,3)

alpha=a[i][i]=9.34729

beta=a[j][j]=4.15585

gamma=a[i][j]=-0.0224104

fi=-0.00431669

cos(fi)=0.999991

sin(fi)=-0.00431668

sigma=146.182

omega=0.00095717

sum=146.184

Iteration #8

Matrix:

5.92836 0.000550983 -9.34414e-005 0.0211885

0.000550983 2.5285 1.02884e-005 -0.00238338

-9.34414e-005 1.02884e-005 9.34739 0

0.0211885 -0.00238338 1.9082e-016 4.15576

Maximal element is: 0.0211885 which is in position (0,3)

alpha=a[i][i]=5.92836

beta=a[j][j]=4.15576

gamma=a[i][j]=0.0211885

fi=0.011951

cos(fi)=0.999929

sin(fi)=0.0119507

sigma=146.183

omega=0.000454945

sum=146.184

Iteration #9

Matrix:

5.92861 0.00052246 -9.34347e-005 0

0.00052246 2.5285 1.02884e-005 -0.00238979

-9.34347e-005 1.02884e-005 9.34739 1.11669e-006

-1.11022e-016 -0.00238979 1.11669e-006 4.1555

Maximal element is: -0.00238979 which is in position (1,3)

alpha=a[i][i]=2.5285

beta=a[j][j]=4.1555

gamma=a[i][j]=-0.00238979

fi=0.00146883

cos(fi)=0.999999

sin(fi)=0.00146882

sigma=146.184

omega=5.99292e-006

sum=146.184

Iteration #10

Matrix:

5.92861 0.00052246 -9.34347e-005 -7.67403e-007

0.00052246 2.52849 1.029e-005 8.67362e-019

-9.34347e-005 1.029e-005 9.34739 1.10158e-006

-7.67403e-007 -2.09034e-016 1.10158e-006 4.15551

Maximal element is: 0.00052246 which is in position (0,1)

alpha=a[i][i]=5.92861

beta=a[j][j]=2.52849

gamma=a[i][j]=0.00052246

fi=0.000153659

cos(fi)=1

sin(fi)=0.000153659

sigma=146.184

omega=2.81802e-007

sum=146.184

Iteration #11

Matrix:

5.92861 0 -9.34331e-005 -7.67403e-007

6.348e-017 2.52849 1.03044e-005 1.17918e-010

-9.34331e-005 1.03044e-005 9.34739 1.10158e-006

-7.67403e-007 1.17918e-010 1.10158e-006 4.15551

Maximal element is: -9.34331e-005 which is in position (0,2)

alpha=a[i][i]=5.92861

beta=a[j][j]=9.34739

gamma=a[i][j]=-9.34331e-005

fi=2.73295e-005

cos(fi)=1

sin(fi)=2.73295e-005

sigma=146.184

omega=8.83773e-009

sum=146.184

Iteration #12

Matrix:

5.92861 2.81613e-010 0 -7.67373e-007

2.81613e-010 2.52849 1.03044e-005 1.17918e-010

-4.18285e-016 1.03044e-005 9.34739 1.1016e-006

-7.67373e-007 1.17918e-010 1.1016e-006 4.15551

Maximal element is: 1.03044e-005 which is in position (1,2)

alpha=a[i][i]=2.52849

beta=a[j][j]=9.34739

gamma=a[i][j]=1.03044e-005

fi=-1.51115e-006

cos(fi)=1

sin(fi)=-1.51115e-006

sigma=146.184

omega=1.07982e-010

sum=146.184

Iteration #13

Matrix:

5.92861 2.81613e-010 4.25559e-016 -7.67373e-007

2.81613e-010 2.52849 -8.47033e-022 1.16254e-010

7.27363e-018 3.14685e-016 9.34739 1.1016e-006

-7.67373e-007 1.16254e-010 1.1016e-006 4.15551

Maximal element is: 1.1016e-006 which is in position (2,3)

alpha=a[i][i]=9.34739

beta=a[j][j]=4.15551

gamma=a[i][j]=1.1016e-006

fi=2.12178e-007

cos(fi)=1

sin(fi)=2.12178e-007

sigma=146.184

omega=1.80239e-012

sum=146.184

---RESULTS---

Finished in 13 iterations

Eigen value #1: Value=5.92861 Vector: 0.782136 0.0757401 -0.51409 -0.343857

Error=-6.06704e-008 4.22079e-007 3.33616e-007 -5.43811e-007

Eigen value #2: Value=2.52849 Vector: -0.211007 0.790282 -0.446888 0.362245

Error=2.29442e-010 -4.25808e-011 -1.95334e-010 -1.44345e-011

Eigen value #3: Value=9.34739 Vector: 0.580947 0.259813 0.588941 0.498143

Error=8.70101e-008 -6.05596e-007 -4.79103e-007 7.80814e-007

Eigen value #4: Value=4.15551 Vector: 0.0789851 -0.549741 -0.434915 0.708798

Error=3.97579e-008 2.28181e-007 1.04323e-006 8.12663e-007

####SCALAR METHOD####

Iteration #0

lambda=0 y=1 1 1 1

Iteration #1

lambda=8.88 y=0.560774 0.338455 0.575705 0.489433

Iteration #2

lambda=9.30414 y=0.575162 0.285635 0.589113 0.490457

Iteration #3

lambda=9.34287 y=0.579129 0.268744 0.590547 0.4936

Iteration #4

lambda=9.34679 y=0.580333 0.263088 0.59006 0.495811

Iteration #5

lambda=9.34729 y=0.580725 0.261081 0.589547 0.497021

Iteration #6

lambda=9.34737 y=0.580862 0.260325 0.58924 0.497621

Iteration #7

lambda=9.34738 y=0.580913 0.260027 0.589082 0.497905

Iteration #8

lambda=9.34738 y=0.580933 0.259904 0.589006 0.498035

---RESULTS---

Finished in 8 iterations

Eigen value=9.34738 Vector: 0.580933 0.259904 0.589006 0.498035

Error=7.73909e-005 -0.000483921 -0.000330693 0.0005544

####DANILEVSKI METHOD####

ITERATION #0

C

0 0 0 1.15

1 0 0 0.16

0 1 0 2.1

0 0 1 5.44

C-1

-0.13913 1 0 0

-1.82609 0 1 0

-4.73043 0 0 1

0.869565 0 0 0

B

3.33861 1.17339 0 3.2251

-0.818261 4.43826 0 -4.29607

-5.3273 -2.2047 -8.88178e-016 -41.8045

1.0087 0.791304 1 14.1831

ITERATION #1

C

0 0 0 3.2251

1 0 0 -4.29607

0 1 0 -41.8045

0 0 1 14.1831

C-1

1.33207 1 0 0

12.9622 0 1 0

-4.39773 0 0 1

0.310068 0 0 0

B

6.00131 0 0 -14.0781

13.0051 -8.88178e-016 0 66.517

-4.36896 1 0 -67.1339

0.363831 0 1 15.9587

ITERATION #2

C

0 0 0 -14.0781

1 0 0 66.517

0 1 0 -67.1339

0 0 1 15.9587

C-1

4.72485 1 0 0

-4.76867 0 1 0

1.13358 0 0 1

-0.0710322 0 0 0

B

-8.88178e-016 0 0 -582.276

1 0 0 530.915

0 1 0 -168.029

0 0 1 21.96

EQUATION:

+1.0000000000x^4 -21.9600000000x^3 +168.0290000000x^2 -530.9150700000x +582.2763609600

EIGEN VALUES: 9.347385 5.928615 4.155506 2.528493

EIGEN VECTOR #1 35.026669 15.664713 35.508623 30.034184

EIGEN VECTOR #2 5.543190 0.536789 -3.643482 -2.436995

EIGEN VECTOR #3 0.838532 -5.836222 -4.617191 7.524821

EIGEN VECTOR #4 2.883363 -10.799002 6.106614 -4.949986

**Висновки:**

Під час виконання лабораторної роботи, ми ознайомилися з деякими ітераційними методами пошуку власних чисел та векторів симетричних матриць.

Скалярним методом було розв’язано часткову проблему власних значень. Цей метод простий у реалізації, проте не дозволяє розв’язати повну проблему пошуку власних значень. Для цього було використано метод Данилевського. За допомогою подібних перетворень матриця приводиться до спеціального вигляду ( матриці Фробеніуса ), що дозволяє легко знайти власні значення матриці. Недоліком цього методу є те, що він потребує окремого розв’язання характеристичного рівняння для пошуку власних чисел.