Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Фізико-технічний інститут

«Методи обчислень»

Лабораторна робота №4
Варіант 2

«Обчислення власних значень»

Виконала:
студентка групи ФБ-95
Гурджия Валерія Вахтангівна

Київ 2022

Завдання

Для методу Данилевського: привести матрицю до вигляду Фробеніуса, розв’язати отриману систему за допомогою методу із практикуму 2 або 3, отримати коефіцієнти характеристичного рівняння. Розв’язати характеристичне рівняння за допомогою одного з методів із практикуму 1 і отримати власні числа.

Для методу Крилова: побудувати систему (4.2), розв’язати її за допомогою методу із практикуму 2 або 3, отримати коефіцієнти характеристичного рівняння, яке розв’язати за допомогою одного з методів із практикуму 1 і отримати власні числа.

Для методу Якобі: привести вихідну матрицю за допомогою подібних обертань до майже діагонального вигляду (сума недіагональних елементів має дорівнювати нулю із точністю 10^{-5}), на головній діагоналі будуть міститись наближені значення власних чисел.

Для всіх варіантів: виконати перевірку отриманих результатів за допомогою математичного пакета (наприклад, можна використати функцію Matlab eig())

№ вар.	Матриця	Метод
2	6,26 1,10 0,98 1,25 1,10 4,16 1,00 0,16 0,98 1,00 5,44 2,12 1,25 0,16 2,12 6,00	Данілевського

Результат роботи програми

Матриці M_{n-1}, M_{n-2}, M_{n-3}, та M_{n-1}^{-1}, M_{n-2}^{-1}, M_{n-3}^{-1}

```
Matrix M(3):
       1        0        0        0
       0        1        0        0
-0.589623 -0.0754717 0.471698 -2.83019
       0        0        0        1

Matrix M(3)^-1:
       1        0        0        0
       0        1        0        0
    1.25     0.16     2.12        6
       0        0        0        1

Matrix A(3)
 5.68217  1.02604 0.462264 -1.52359
0.510377  4.08453 0.471698 -2.67019
 2.46197  3.18567  12.0933 -30.4773
       0        0        1        0
```

```
Matrix M(2):
       1        0        0        0
-0.772827 0.313906 -3.79615    9.567
       0        0        1        0
       0        0        0        1

Matrix M(2)^-1:
       1        0        0        0
 2.46197  3.18567  12.0933 -30.4773
       0        0        1        0
       0        0        0        1

Matrix A(2)
 4.88922 0.322079 -3.43273  8.29252
 3.60703  16.9708 -86.8214  136.395
       0        1        0        0
       0        0        1        0
```

```
Matrix M(1):
0.277237 -4.70492  24.0701 -37.8137
       0        1        0        0
       0        0        1        0
       0        0        0        1

Matrix M(1)^-1:
 3.60703  16.9708 -86.8214  136.395
       0        1        0        0
       0        0        1        0
       0        0        0        1

Matrix A(1)
   21.86 -168.633  548.502 -636.954
       1        0        0        0
       0        1        0        0
       0        0        1        0
```

Результуюча матриця у формі Фробеніуса

```
Frobenius Matrix:
   21.86 -168.633  548.502 -636.954
       1        0        0        0
       0        1        0        0
       0        0        1        0
```

Коефіцієнти характеристичного рівняння

```
Characteric equation coeficients:
1   -21.86   168.633   -548.502   636.954
```

Власні числа

```
roots of polinom (by besection method)
x1:   2.99634
x2:   4.26074
x3:   5.45264
x4:   9.15047
```

Перевірка у MatLab

```
A =

    6.2600    1.1000    0.9800    1.2500
    1.1000    4.1600    1.0000    0.1600
    0.9800    1.0000    5.4400    2.1200
    1.2500    0.1600    2.1200    6.0000

>> poly(A)

ans =

    1.0000  -21.8600  168.6335 -548.5019  636.9541

>> eig(A)

ans =

    2.9963
    4.2607
    5.4525
    9.1505

fx >>
```

Код програми

```
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

void ShowPolinom(vector<float> coef) {
    for (int i = coef.size() - 1; i >= 0; i--) {
        if ((coef[i] < 0) || (i == coef.size() - 1)) {
            if (coef[i] != 0 && coef[i] != 1) {
                if (i > 1) {
                    cout << coef[i] << "x^" << i;
                }
                else if (i == 1) {
                    cout << coef[i] << "x";
                }
                else if (i == 0) {
                    cout << coef[i] << endl;
                }
            }
            else if (coef[i] == 1) {
                if (i > 1) {
                    cout << "x^" << i;
                }
                else if (i == 1) {
                    cout << "x";
                }
                else if (i == 0) {
                    cout << endl;
                }
            }
        }
        else {
            if (coef[i] != 0 && coef[i] != 1) {
                if (i > 1) {
                    cout << "+" << coef[i] << "x^" << i;
                }
                else if (i == 1) {
                    cout << "+" << coef[i] << "x";
                }
                else if (i == 0) {
                    cout << "+" << coef[i] << endl;
                }
            }
            else if (coef[i] == 1) {
                if (i > 1) {
                    cout << "+" << "x^" << i;
                }
                else if (i == 1) {
                    cout << "+" << "x";
                }
                else if (i == 0) {
                    cout << endl;
                }
            }
        }
    }
    cout << endl;
}

float Polinom(vector<float> coef, float x) {
    float sum = 0;
```

```
    for (int i = 0; i < coef.size(); i++) {
        sum += pow(x, i) * coef[i];
    }
    return sum;
}

float BesectionMethod(vector<float> coef, float a, float b, float epsilon) {
    float c = (a + b) / 2;;
    float fc = Polinom(coef, c);;
    float fa = Polinom(coef, a);
    float fb = Polinom(coef, b);
    int i = 0;
    while (abs(fc) > epsilon) {
        if (fa * fb < 0) {
            // cout << "Iteration № " << ++i << endl;
            // cout << "a: " << a << "\nb: " << b << "\nc: " << c << endl;
            while (abs(fc) > epsilon) {
                //  cout << "false\n";
                //  cout << "f(a)*f(c) = " << fa * fc << endl;
                if (fa * fc < 0) {
                    b = c;
                    c = (a + b) / 2;
                    fc = Polinom(coef, c);
                    fb = Polinom(coef, b);
                }
                else if (fa * fc > 0) {
                    a = c;
                    c = (a + b) / 2;
                    fc = Polinom(coef, c);
                    fa = Polinom(coef, a);
                }
            }
            // cout << "true\n";
            //cout << "root x:   " << c << endl;
            return c;
        }
        else {
            cout << "cannot solve\n";
        }
    }
}

void ShowMatrix(vector<vector<float>> Matrix) {
    for (auto M : Matrix) {
        for (auto m : M) {
            cout.width(8);
            cout << m << " ";
        }
        cout << endl;
    }
}

vector<vector<float>> MMulty(vector<vector<float>> Matrix_A, vector<vector<float>> Matrix_B)
{
    vector<vector<float>> result;
    for (int i = 0; i < Matrix_A.size(); i++) {
        vector<float> raw;
        for (int j = 0; j < Matrix_A.size(); j++) {
            float a = 0;
            for (int k = 0; k < Matrix_A[i].size(); k++)
            {
                a += Matrix_A[i][k] * Matrix_B[k][j];
```

```
            }
            raw.push_back(a);
        }
        result.push_back(raw);
    }
    return result;
}

vector<vector<float>> toFrobenius(vector<vector<float>> Matrix_A) {
    vector<vector<float>> Matrix_P;
    vector<vector<float>> Matrix_MS;
    vector<vector<vector<float>>> Matrix_A_ = { Matrix_A };
    vector<vector<vector<float>>> Matrix_S;
    int n = Matrix_A.size();
    for (int i = 0; i < n - 1; i++) {
        vector<vector<float>> Matrix_M;
        vector<vector<float>> Matrix_M_1;
           // поиск M //
        for (int j = 0; j < n; j++) {
            vector<float> raw;
            for (int k = 0; k < n; k++) {
                if (j == n - 2 - i) {
                    float a;
                    if(k==j){
                        a = 1/ Matrix_A_[i][n - 1 - i][n - 2 - i];
                    }
                    else {
                        a = -Matrix_A_[i][n - 1 - i][k] / Matrix_A_[i][n - 1 - i][n - 2 - i];
                    }
                    raw.push_back(a);
                }
                else if (j == k) {
                    raw.push_back(1);
                }
                else {
                    raw.push_back(0);
                }
            }
            Matrix_M.push_back(raw);
        }
        Matrix_S.push_back(Matrix_M);
               // поиск M^-1 //
        for (int j = 0; j < n; j++) {
            vector<float> raw;
            for (int k = 0; k < n; k++) {
                if (j == n - 2 - i) {
                    float a = Matrix_A_[i][n - 1 - i][k];
                    raw.push_back(a);
                }
                else if (j == k) {
                    raw.push_back(1);
                }
                else {
                    raw.push_back(0);
                }
            }
            Matrix_M_1.push_back(raw);
        }
        vector<vector<float>> Matrix1 = MMulty(Matrix_A_[i], Matrix_M);
        vector<vector<float>> Matrix2 = MMulty(Matrix_M_1, Matrix1);
        Matrix_A_.push_back(Matrix2);
```

```
        cout << "\nMatrix M(" << n-i-1 << "):\n";
        ShowMatrix(Matrix_M);
        cout << "\nMatrix M(" << n-i-1 << ")^-1:\n";
        ShowMatrix(Matrix_M_1);
        cout << "\nMatrix A(" << n-i-1 << ")\n";
        ShowMatrix(Matrix_A_[i+1]);
    }

        // найдем произведение всех M //
    Matrix_MS = MMulty(Matrix_S[0], Matrix_S[1]);
    for (int i = 2; i < n - 1; i++) {
        Matrix_MS = MMulty(Matrix_MS, Matrix_S[i]);
    }

    Matrix_P = Matrix_A_[Matrix_A_.size()-1];

    cout << "\nMatrix S:\n";
    ShowMatrix(Matrix_MS);
    cout << "\nFrobenius Matrix:\n";
    ShowMatrix(Matrix_P);

    return Matrix_P;
}

vector<float> GetCoeficients(vector<vector<float>> Matrix_P) {
    vector<float> Coeficients_P;
    int n = Matrix_P.size();
    for (int i = n-1; i >= 0; i--) {
        if (n % 2 == 0) {
            Coeficients_P.push_back(-Matrix_P[0][i]);
        }
        else {
            Coeficients_P.push_back(Matrix_P[0][i]);
        }
    }
    if (n % 2 == 0) {
        Coeficients_P.push_back(1);
    }
    else {
        Coeficients_P.push_back(-1);
    }
    return Coeficients_P;
}

int main()
{
    setlocale(LC_ALL, "ru");
    vector<vector<float>> Matrix_A = {
        {6.26, 1.1, 0.98, 1.25},
        {1.1, 4.16, 1, 0.16},
        {0.98, 1, 5.44, 2.12},
        {1.25, 0.16, 2.12, 6}
    };

    vector<float> Coeficients = GetCoeficients(toFrobenius(Matrix_A));

    cout << "\nCharacteric equation coeficients:\n";
       for (int i = Coeficients.size() - 1; i >= 0; i--) {
        cout << Coeficients[i] << "  ";
    }
    cout << endl;
    float x1, x2, x3, x4;
```

```
float epsilon = 0.001;
cout << "\nroots of polinom (by besection method)\n";
x1 = BesectionMethod(Coeficients, 2, 4, epsilon);
x2 = BesectionMethod(Coeficients, 4, 5, epsilon);
x3 = BesectionMethod(Coeficients, 5, 6, epsilon);
x4 = BesectionMethod(Coeficients, 8, 10, epsilon);
cout << "x1:  " << x1 << "\nx2:  " << x2 << "\nx3:  " << x3 << "\nx4:  " << x4 << endl;

return 0;}
```