Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра Систем Штучного Інтелекту

Звіт

до лабораторної роботи № 3

з дисципліни

Чисельні методи

на тему:

“Ітераційні методи розв’язування систем лінійних алгебраїчних рівнянь.”

Варіант №24(Метод Якобі)

Виконав: студент КН-217

Ратушняк Денис

Прийняла: доцент каф. СШІ

Мочурад Л. І.

Львів – 2022

Мета роботи: набути навиків практичного використання ітераційних методів розв’язування СЛАР: методу Якобі, методу Зейделя, методу верхньої релаксації.

Завдання. Складіть програму, яка: **а) методом Якобі**, б) методом Зейделя, в) методом верхньої релаксації – знаходить наближений розв’язок системи лінійних рівнянь із заданою точністю ε . Представте детально документовану програму з результатами знаходження наближених розв’язків лінійних систем.

При відсутності діагональної переваги у системі необхідно добитися її шляхом еквівалентних перетворень. Примітка. Пункт **а) виконують студенти з номером у списку в журналі викладача, який представляється у вигляді 3∗ k**, пункт б) – з номером 3 * k + 1 , пункт в) – з номером 3 * k + 2 , де k – ціле число.

$$1)\begin{cases}15x_1 - 4x_2 - 3x_3 + 8x_4 = 2,\\ -4x_1 + 10x_2 - 4x_3 + 2x_4 = -12,\\ -3x_1 - 4x_2 + 10x_3 + 2x_4 = -4,\\ 8x_1 + 2x_2 + 2x_3 + 12x_4 = 6,\end{cases} \qquad 2)\begin{cases}3x_1 - 3x_2 + 4x_3 = 1,\\ -3x_1 - x_2 = 2,\\ 4x_1 - 4x_3 = 3.\end{cases}$$

Програмна реалізація на мові програмування С++

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef long long ll;
typedef vector< vector<ld > > matrix;
const ld eps = 1e-10;
ifstream tests("input.txt");
ofstream answers("output.txt");
void print(vector<ld> &A, ll cnt = 3)
{
    cout << fixed << setprecision(cnt);
    for(int i = 0; i < A.size(); ++i) cout << A[i] << " ";
    cout << endl;
    cout << fixed << setprecision(0);
}
void print(vector<ld> &A, ofstream &answers, ll cnt = 3)
{
    answers << fixed << setprecision(cnt);
    for(int i = 0; i < A.size(); ++i) answers << A[i] << " ";
    answers << "\n\n";
    answers << fixed << setprecision(0);
}
void print(matrix &A, ofstream &answers, ll cnt=3)
{
    ll n = A.size();
    answers << fixed << setprecision(cnt);
    for(int i = 0; i < n; ++i)
    {
        for(int j = 0; j < A[i].size(); ++j) answers << A[i][j] << " ";
        answers << "\n";
```

```
    }
    answers << fixed << setprecision(0);
    answers << "\n";
}
void print(matrix &A, ll cnt=3)
{
    ll n = A.size();
    cout << fixed << setprecision(cnt);
    for(int i = 0; i < n; ++i)
    {
        for(int j = 0; j < A[i].size(); ++j) cout << A[i][j] << " ";
        cout << endl;
    }
    cout << fixed << setprecision(0);
    cout << endl;
}
matrix operator * (const matrix &A, const matrix &B)
{
    ll n = A.size();
    matrix res(n, vector<ld>(n, 0.0));
    for(int k = 0; k < n; ++k)
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j)
                res[i][j] += A[i][k] * B[k][j];
    return res;
}
matrix get_random_matrix(ll size_, ll mxVal)
{
    ///randomize due to clock
    srand(clock());
    matrix res;
    res.resize(size_);
    for(int i = 0; i < size_; ++i)
    {
        res[i].resize(size_);
        for(int j = 0; j < size_; ++j) res[i][j] = (rand() - rand()/2)%mxVal;
    }
    return res;
}
ld linf_norm(matrix &A){
    ll n = A.size();
    ld norm = 0;
    for(int i = 0; i < n; ++ i){
        ld sum = 0;
        for(int j = 0; j < n; ++ j){
            sum += fabs(A[i][j]);
        }
        norm = max(norm, sum);
    }
    return norm;
}

ld l1_norm(matrix &A){
    ll n = A.size();
    ld norm = 0;
    for(int j = 0; j < n; ++ j){
        ld sum = 0;
        for(int i = 0; i < n; ++ i){
            sum += fabs(A[i][j]);
        }
        norm = max(norm, sum);
    }
```

```
    return norm;
}
ld eu_norm(matrix &A){
    ll n = A.size();
    ld sum = 0;
    for(int i = 0; i < n; ++ i){
        for(int j = 0; j < n; ++ j){
            sum += fabs(A[i][j]) * fabs(A[i][j]);
        }
    }
    return sqrt(sum);
}
void check_ans(matrix &A, vector<ld> &X, vector<ld> &B, ll cnt=0)
{
    answers << "\nHERE IS ANSWER(X): ";
    print(X, answers, cnt);
    answers << "HERE IS WHAT B SHOULD BE " << "\n";
    print(B, answers, cnt);
    answers << "HERE IS CALCULATED B" << "\n";
    ll n = A.size();
    vector<ld> ans(n,0);
    vector<ld> diff(n);
    ld euNorm = 0.0;
    for(int i = 0; i < n; ++i)
    {
        for(int j = 0; j < n; ++j)
        {
            ans[i] += A[i][j] * X[j];
        }
        diff[i] = B[i] - ans[i];
        euNorm += diff[i] * diff[i];
    }
    print(ans, answers, cnt);
    answers << "HERE is B - B(calculated)" << "\n";
    print(diff, answers, cnt);
    answers << "Norm of this vector = " << fixed << setprecision(cnt) << sqrt(euNorm) << "\n";
    answers << fixed << setprecision(0);
}
bool check_DD(matrix &A){
    ll n = A.size();
    bool if_one = 0;
    for(int i = 0; i < n; ++ i){
        ld sum = 0;
        for(int j = 0; j < n; ++ j) sum += fabs(A[i][j]);
        sum -= fabs(A[i][i]);
        if(fabs(A[i][i]) > sum) if_one = 1;
        if(fabs(A[i][i]) < sum) return false;
    }
    return if_one;
}
void solve(matrix &A, vector<ld> &B, ld allowed_error = 0.0000000001)
{
    answers << "Size = " << A.size() << "\n";
    answers << "Matrix A:\n";
    print(A, answers);
    answers << "B:\n";
    print(B, answers);
    ll n = A.size();
    matrix D(n, vector<ld>(n, 0));
    matrix LplusR(n, vector<ld>(n, 0));
    for(int i = 0; i < n; ++ i){
        for(int j = 0; j < n; ++ j){
```

```
                if(i == j) D[i][j] = -1.0/A[i][j];
                else LplusR[i][j] = A[i][j];
            }
        }
        matrix C(n, vector<ld>(n, 0));
        C = D * LplusR;
        ld n1 = l1_norm(C);
        ld n2 = linf_norm(C);
        ld n3 = eu_norm(C);
        answers << "Matrix C(-D^-1 * (L+R)):\n";
        print(C, answers, 5);
        answers << "\n Norms of C(-D^-1 * (L+R)):\n";
        answers << fixed << setprecision(15) << n1 << " " << n2 << " " << n3 << "\n";

        if(min({n1,n2,n3}) - 1 > eps){
            answers << "All norms of matrix C(-D^-1 * (L+R)) is > 1, so Jacobi method won`t converge\n";
            answers << fixed << setprecision(0) << "---------------------------------------------------------\n";
            return;
        }

        if(!check_DD(A)){
            answers << "This matrix is not diagonally dominant, so it is not garanteed for jacobi method to
converge\n";
            answers << fixed << setprecision(0) << "---------------------------------------------------------\n";
            return;
        }
        answers << fixed << setprecision(15) << "CURRENT EPS:" << eps << "\n";
        answers << fixed << setprecision(0);
        vector<ld> x(n, 0.0);
        vector<ld> tmp_x(n, 0.0);

        ld error;
        ll step = 1;
        do
        {
            error = 0;
            tmp_x = B;
            for(int i = 0; i < n; ++i)
                for(int j = 0; j < n; ++j)
                    if (i != j) tmp_x[i] -= A[i][j] * x[j];

            answers << fixed << setprecision(10) << "Step: " << step << " Current X: ";
            for(int i = 0; i < n; ++ i)
            {
                ld x_upd = tmp_x[i] / A[i][i];
                ld e = fabs(x[i] - x_upd);
                answers << x_upd << " ";
                x[i] = x_upd;
                error = max(error, e);
            }
            answers << "\n";
            step++;
        }
        while (error > allowed_error);
        check_ans(A, x, B, 13);
        answers << fixed << setprecision(0) << "---------------------------------------------------------\n";
    }
    int main()
    {
        ll n;
        while(tests >> n)
        {
```

```
        matrix A(n, vector<ld>(n, 0));
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j)
                tests >> A[i][j];
        vector<ld> B(n);
        for(int i = 0 ; i < n; ++i) tests >> B[i];
        solve(A, B);
    }
    ll sz = 300;
    matrix A(sz, vector<ld>(sz, 0));
    A = get_random_matrix(sz, 1000000);
    vector<ld> B(sz);
    for(int i = 0; i < sz; ++i) B[i] = rand() - rand() / 2;
    for(int i = 0; i < sz; ++i) A[i][i] = 1000000000 + 2*i;
    solve(A, B);
    return 0;
}
```

Вхідні дані

```
4
15 -4 -3 8
-4 10 -4 2
-3 -4 10 2
8 2 2 12

2 -12 -4 6

3
3 -3 4
-3 -1 0
4 0 -4

1 2 3

3
3 -3 4
0 -4 4
4 0 -4

1 3 3

3
7 -3 0
0 -4 4
4 0 -4

4 3 3

3
3 2 1
3 2 1
4 5 15

2 4 3
```

Вихідні дані

```
Size = 4
Matrix A:
15.000 -4.000 -3.000 8.000
-4.000 10.000 -4.000 2.000
-3.000 -4.000 10.000 2.000
8.000 2.000 2.000 12.000

B:
2.000 -12.000 -4.000 6.000

-0.06667 0.00000 0.00000 0.00000
0.00000 -0.10000 0.00000 0.00000
0.00000 0.00000 -0.10000 0.00000
0.00000 0.00000 0.00000 -0.08333

0.00000 -4.00000 -3.00000 8.00000
-4.00000 0.00000 -4.00000 2.00000
-3.00000 -4.00000 0.00000 2.00000
8.00000 2.00000 2.00000 0.00000

Matrix C(-D^-1 * (L+R)):
0.00000 0.26667 0.20000 -0.53333
0.40000 0.00000 0.40000 -0.20000
0.30000 0.40000 0.00000 -0.20000
-0.66667 -0.16667 -0.16667 0.00000


 Norms of C(-D^-1 * (L+R)):
1.366666666666667 1.000000000000000 1.243203746598101
CURRENT EPS:0.000000000100000
Step: 1 Current X: 0.1333333333 -1.2000000000 -0.4000000000 0.5000000000
Step: 2 Current X: -0.5333333333 -1.4066666667 -0.9400000000 0.6777777778
Step: 3 Current X: -0.7912592593 -1.9248888889 -1.2582222222 1.2466666667
Step: 4 Current X: -1.2965037037 -2.2691259259 -1.6566666667 1.5580246914
Step: 5 Current X: -1.6340467490 -2.6928730864 -2.0082064198 2.0186345679
```

```
Step: 1104 Current X: -21.3641025587 -22.0974358921 -19.9999999951 21.7589743536
Step: 1105 Current X: -21.3641025588 -22.0974358922 -19.9999999952 21.7589743537
Step: 1106 Current X: -21.3641025589 -22.0974358923 -19.9999999953 21.7589743538
Step: 1107 Current X: -21.3641025590 -22.0974358924 -19.9999999954 21.7589743539
Step: 1108 Current X: -21.3641025591 -22.0974358925 -19.9999999955 21.7589743540
Step: 1109 Current X: -21.3641025592 -22.0974358926 -19.9999999955 21.7589743541

HERE IS ANSWER(X): -21.3641025592258 -22.0974358926312 -19.9999999955486 21.7589743540835

HERE IS WHAT B SHOULD BE
2.0000000000000 -12.0000000000000 -4.0000000000000 6.0000000000000

HERE IS CALCULATED B
2.0000000014511 -11.9999999990469 -3.9999999991170 5.9999999988358

HERE is B - B(calculated)
-0.0000000014511 -0.0000000009531 -0.0000000008830 0.0000000011642

Norm of this vector = 0.0000000022692
---------------------------------------------------------
Size = 3
Matrix A:
3.000 -3.000 4.000
-3.000 -1.000 0.000
4.000 0.000 -4.000

B:
1.000 2.000 3.000

This matrix is not diagonally dominant, so it is not garanteed for jacobi method to converge
---------------------------------------------------------
```

```
Size = 3
Matrix A:
3.000 -3.000 4.000
0.000 -4.000 4.000
4.000 0.000 -4.000

B:
1.000 3.000 3.000

This matrix is not diagonally dominant, so it is not garanteed for jacobi method to converge
---------------------------------------------------------
Size = 3
Matrix A:
7.000 -3.000 0.000
0.000 -4.000 4.000
4.000 0.000 -4.000

B:
4.000 3.000 3.000

-0.14286 0.00000 0.00000
0.00000 0.25000 0.00000
0.00000 0.00000 0.25000

0.00000 -3.00000 0.00000
0.00000 0.00000 4.00000
4.00000 0.00000 0.00000

Matrix C(-D^-1 * (L+R)):
0.00000 0.42857 0.00000
0.00000 0.00000 1.00000
1.00000 0.00000 0.00000


 Norms of C(-D^-1 * (L+R)):
1.000000000000000 1.000000000000000 1.477725776112657
CURRENT EPS:0.000000000100000
Step: 1 Current X: 0.5714285714 -0.7500000000 -0.7500000000
Step: 2 Current X: 0.2500000000 -1.5000000000 -0.1785714286
Step: 3 Current X: -0.0714285714 -0.9285714286 -0.5000000000
```

```
Step: 76 Current X: -0.1249999996 -1.6249999994 -0.8749999999
Step: 77 Current X: -0.1249999998 -1.6249999999 -0.8749999996
Step: 78 Current X: -0.1250000000 -1.6249999996 -0.8749999998
Step: 79 Current X: -0.1249999998 -1.6249999998 -0.8750000000
Step: 80 Current X: -0.1249999999 -1.6250000000 -0.8749999998
Step: 81 Current X: -0.1250000000 -1.6249999998 -0.8749999999
Step: 82 Current X: -0.1249999999 -1.6249999999 -0.8750000000

HERE IS ANSWER(X): -0.1249999999192 -1.6249999998985 -0.8749999999855

HERE IS WHAT B SHOULD BE
4.0000000000000 3.0000000000000 3.0000000000000

HERE IS CALCULATED B
4.0000000002611 2.9999999996519 3.0000000002652

HERE is B - B(calculated)
-0.0000000002611 0.0000000003481 -0.0000000002652

Norm of this vector = 0.0000000005096
---------------------------------------------------------
Size = 3
Matrix A:
3.000 2.000 1.000
3.000 2.000 1.000
4.000 5.000 15.000

B:
2.000 4.000 3.000

This matrix is not diagonally dominant, so it is not garanteed for jacobi method to converge
---------------------------------------------------------
```

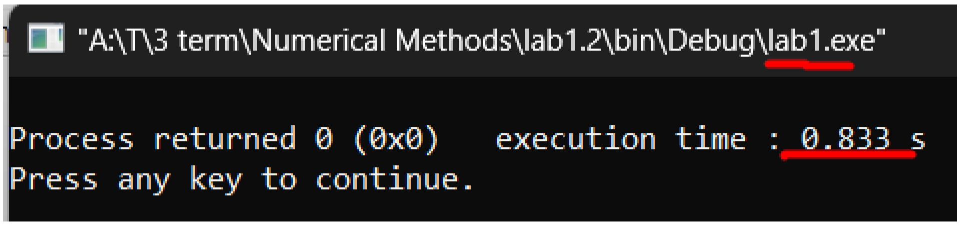
Аналіз чисельних експериментів

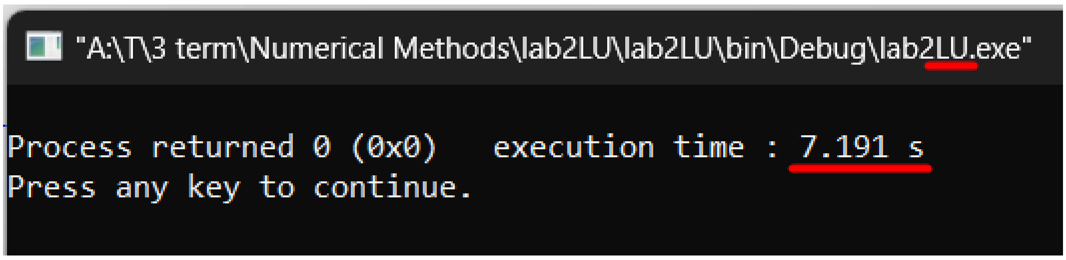
Метод Якобі на великих матрицях(з точністю 10^-10) працює швидше за метод LU розкладу та метод Гауса, проте при збільшені точності час його виконання буде збільшуватись.

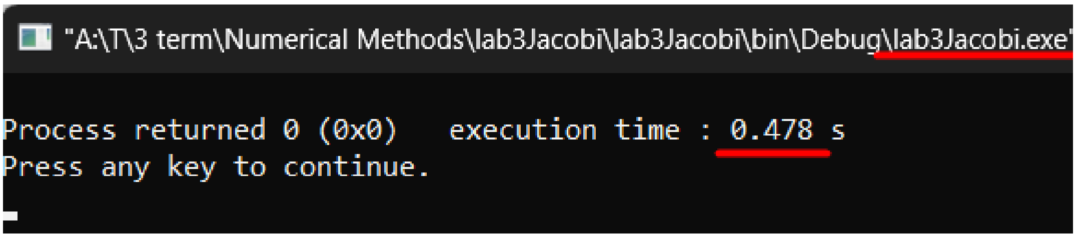
Метод Якобі спочатку генерує неточні результати, а потім уточнює свої результати на кожній ітерації, при цьому залишки збігаються з великою швидкістю. В багатьох випадках це може бути корисним.

Метод Якобі працює не на всіх матрицях, зазвичай на матрицях з діагональною перевагою, що робить його неуніверсальним.

Час роботи програми рандомної матриці 300 × 300.







Висновок

Я закріпив вміння та навички практичного використання ітераційних методу розв'язування СЛА Р- методу Якобі. Створив програму, яка по заданим матрицям перевіряє чи можна застосувати метод Якобі, якщо ні, виводить відповідне повідомлення. У разі успіху, програма застосовує метод Якобі до розв'язку СЛАР і знаходить результат з точністю (10^-10, яку можна змінити в самій програмі). Також програма виводить поточний результат вектору X і поточний номер ітерації. В кінці виводиться відповідь(вектор X), вектор B - ~B(A*X) та його норма.