

РОЗВ'ЯЗАННЯ СИСТЕМ ЛІНІЙНИХ
АЛГЕБРАЇЧНИХ РІВНЯНЬ. ІТЕРАЦІЙНІ
МЕТОДИ ЯКОБІ ТА ГАУСА – ЗЕЙДЕЛЯ

Алгоритми

Лабораторна робота №4

Варіант 7

Виконав: Конча Вадим

Завдання:

Вимоги до виконання роботи

1. Складіть програму для розв'язання СЛАР методом Якобі.
2. Доповніть програму лічильником числа ітерацій та проміжним друком k , $\mathbf{x}^{(k)}$ та загальної похибки наближення $\delta_k = \max_i \left\{ \left| x_i^{(k)} - x_i^{(k-1)} \right| \right\}$ після кожної ітерації (k - номер ітерації). Результати повинні мати вигляд охайної таблиці.
3. Іноді ітераційний процес може розбігатися. З метою гарантованого завершення програми навіть у випадку незбіжності до розв'язку, запровадьте в програмі обмеження на максимальну кількість ітерацій. Передбачте виведення відповідного повідомлення про незбіжність ітераційного процесу.
4. Зведіть систему $\mathbf{Ax} = \mathbf{b}$ вашого варіанту до вигляду, необхідного для ітерацій.
5. Отримайте розв'язок системи з вашого варіанту з точністю 0.0001, **попередньо оцінивши** число необхідних для цього кроків. Порівняйте кількість витрачених для отримання розв'язку ітерацій з її попередньою оцінкою.
6. Для перевірки отриманого результату обчисліть і надрукуйте вектор нев'язок $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$.
7. Дослідіть, як похибки поточного наближення до розв'язку залежать від номера ітерації. Побудуйте графік залежності $\lg \delta_k$ від k і на його основі з'ясуйте порядок збіжності методу.
8. Модифікуйте вашу програму для реалізації методу Гауса – Зейделя. Розв'яжіть задачу вашого варіанту та порівняйте розв'язок СЛАР і кількість здійснених ітерацій з отриманими раніше результатами методу Якобі.
9. З'ясуйте порядок збіжності методу Гауса – Зейделя.

$$7. \begin{cases} 3.2x_1 - 2.5x_2 + 3.7x_3 = 6.5; \\ 0.5x_1 + 0.34x_2 + 1.7x_3 = -0.24; \\ 1.6x_1 + 2.3x_2 - 1.5x_3 = 4.3. \end{cases}$$

Підготовка матриці:

$$\begin{pmatrix} 3.2 & -2.5 & 3.7 & 6.5 \\ 0.5 & 0.34 & 1.7 & -0.24 \\ 1.6 & 2.3 & -1.5 & 4.3 \end{pmatrix} \text{ -- start}$$

① міняєм стовбці з індексами (1) и (2)

$$\begin{pmatrix} 3.2 & 3.7 & -2.5 & 6.5 \\ 0.5 & 1.7 & 0.34 & -0.24 \\ 1.6 & -1.5 & 2.3 & 4.3 \end{pmatrix}$$

★ тепер 2е правило = (✓) $1.7 > 0.5 + 0.34$

②
$$\begin{array}{rrrr} + & 3.2 & 3.7 & -2.5 & 6.5 \\ & 1.6 & -1.5 & 2.3 & 4.3 \\ \hline & 4.8 & 2.2 & -0.2 & 10.8 \end{array}$$
 $2.2 + 0.2 < 4.8$
1) = (✓)

$$\begin{pmatrix} 4.8 & 2.2 & -0.2 & 10.8 \\ 0.5 & 1.7 & 0.34 & -0.24 \\ 1.6 & -1.5 & 2.3 & 4.3 \end{pmatrix} \begin{array}{l} \leftarrow \text{new} \\ \leftarrow \text{table} \end{array}$$

③
$$\begin{array}{rrrr} + & 0.5 & 1.7 & 0.34 & -0.24 \\ & 1.6 & -1.5 & 2.3 & 4.3 \\ \hline & 2.1 & 0.2 & 2.64 & 4.06 \end{array}$$
 $2.64 \leq 0.2 + 2.1$
3) = (✓)

$$\begin{pmatrix} 4.8 & 2.2 & -0.2 & 10.8 \\ 0.5 & 1.7 & 0.34 & -0.24 \\ 2.1 & 0.2 & 2.64 & 4.06 \end{pmatrix} \text{ -- final}$$

Код(до п.8):

```
#include <iostream>
#include <vector>
#include <stdlib.h> // max
```

```
using namespace std;
```

```

double E = 0.0001;
double x1, x2, x3;
double SAFEx1, SAFEx2, SAFEx3;
vector <double> x1TAB, x2TAB, x3TAB; //сюда будем писать историю
итерации, для вывода

//start --- vector <double> first{ 3.2, -2.5, 3.7, 6.5 }, second{ 0.5, 0.34, 1.7,
-0.24 }, third{ 1.6, 2.3, -1.5, 4.3 };
vector <double> first{ 4.8 , 2.2, -0.2, 10.8 }, second{ 0.5, 1.7, 0.34, -0.24 },
third{ 2.1, 0.2, 2.64, 4.06};
// ax1 + bx2 + cx3 = d; это коефы [a,b,c,d] ,
//             что {0,1,2,3}

void x_calc()
{
    SAFEx1 = x1; x1TAB.push_back(SAFEx1);
    SAFEx2 = x2; x2TAB.push_back(SAFEx2);
    SAFEx3 = x3; x3TAB.push_back(SAFEx3);

    x1 = (first[3] - first[1] * SAFEx2 - first[2] * SAFEx3) / first[0];
    //а это уравнение x1 = (d - bx2 - cx3) / a

    x2 = (second[3] - second[0] * SAFEx1 - second[2] * SAFEx3) /
second[1];
    //а это уравнение x2 = (d - ax1 - cx3) / b

    x3 = (third[3] - third[1] * SAFEx2 - third[0] * SAFEx1) / third[2];
    //а это уравнение x3 = (d - bx2 - ax1) / c
}
//даёт нам новые иксы и в сейве старые иксы
double justFUNCTION(vector <double> sho)
{
    return sho[0] * x1 + sho[1] * x2 + sho[2] * x3;
}
void printQ(vector <double> first)
{
    cout << first[0] << "x1 + " << first[1] << "x2 + " << first[2] << "x3 = " <<
first[3] << endl;
}
//банальный вывод для проверки уравнений

```

```

void uslovieALL()
{
    bool a, b, c;
    if ((abs(first[1]) + abs(first[2])) / abs(first[0]) > 1)
    {
        cout << "errMOMENT 1 " << (abs(first[1]) + abs(first[2])) /
abs(first[0]) << endl;
        a = true;
    }
    else a = false;
    //проверка первого уравнения

    if ((abs(second[0]) + abs(second[2])) / abs(second[1]) > 1)
    {
        cout << "errMOMENT 2 " << (abs(second[0]) + abs(second[2]))
<< endl;
        b = true;
    }
    else b = false;
    //проверка второго уравнения

    if ((abs(third[1]) + abs(third[0])) / abs(third[2]) > 1)
    {
        cout << "errMOMENT 3 " << (abs(third[1]) + abs(third[0])) <<
endl;
        c = true;
    }
    else c = false;
    //проверка третьего уравнения

    if (
        a == true
        ||
        b == true
        ||
        c == true
        )
    {
        cout << endl << "KOEFS are invalid." << endl;
    }
}

```

```

    }
}
//проверка всех уравнений

int main()
{
    uslovieALL(); //выводит сообщение, если коэфициенты
неподходящие, и показывает где конкретно трабблы
    //x(0) - нулевая итерация
    x1 = first[3];
    x2 = second[3];
    x3 = third[3];

    int limitation = 0;
    while
    (
        abs( x1 - SAFEx1 ) >= E
        &
        abs( x2 - SAFEx2 ) >= E
        &
        abs( x3 - SAFEx3 ) >= E
    ) //условием повтора - является проверка точности
    {
        x_calc(); //подсчёт иксов
        limitation += 1; //для пункта 3(бесполезная штука)

        if (limitation > 20)
        {
            cout << "\nmax limit of limitations ERROR" << endl;
            exit(0);
        }
    }

    x1TAB.push_back(SAFEx1);
    x2TAB.push_back(SAFEx2);
    x3TAB.push_back(SAFEx3);
    //запись последних иксов (последней итерации)

    //вывод

```

```

        cout << "\n" << "0" << " iteration\t" << "x1 = " << x1TAB[0] << "\tx2 = "
<< x2TAB[0] << "\tx3 = " << x3TAB[0]; //отдельно для нулевого
        for (int i = 1; i < x1TAB.size(); i++)
        {
            cout << "\n" << i << " iteration\t" << "x1 = " << x1TAB[i] << "\tx2 = "
<< x2TAB[i] << "\tx3 = " << x3TAB[i]
                << "\tFLUFF = " << max(max(abs(x1TAB[i] - x1TAB[i - 1]),
abs(x2TAB[i] - x2TAB[i - 1])), abs(x3TAB[i] - x3TAB[i - 1]))
                ;
        }

//невязка (пункт 6)
        cout << "\nfirst: " << justFUNCTION(first) << "\t real be === " << first[3]
<< "\t NEVAZKA: " << abs(justFUNCTION(first) - first[3])
                << "\nsecond: " << justFUNCTION(second) << "\t real be === "
<< second[3] << "\t NEVAZKA: " << abs(justFUNCTION(second) - second[3])
                << "\nthird: " << justFUNCTION(third) << "\t real be === " <<
third[3] << "\t NEVAZKA: " << abs(justFUNCTION(third) - third[3]);
    }

```

Код(після):

```

void x_calc()
{
    SAFEx1 = x1; x1TAB.push_back(x1);
    SAFEx2 = x2; x2TAB.push_back(x2);
    SAFEx3 = x3; x3TAB.push_back(x3);

    x1 = (first[3] - first[1] * x2 - first[2] * x3) / first[0];
    //а это уравнение x1 = (d - bx2 - cx3) / a

    x2 = (second[3] - second[0] * x1 - second[2] * x3) / second[1];
    //а это уравнение x2 = (d - ax1 - cx3) / b

    x3 = (third[3] - third[1] * x2 - third[0] * x1) / third[2];
    //а это уравнение x3 = (d - bx2 - ax1) / c
}
//даёт нам новые иксы и в сейве старые иксы

```

*Зміни тільки у функції “x_calc”

```

void x_calc()
{
    SAFEx1 = x1; x1TAB.push_back(x1);
    SAFEx2 = x2; x2TAB.push_back(x2);
    SAFEx3 = x3; x3TAB.push_back(x3);

    x1 = (first[3] - first[1] * x2 - first[2] * x3) / first[0];
    //а это уравнение x1 = (d - bx2 - cx3) / a

    x2 = (second[3] - second[0] * x1 - second[2] * x3) / second[1];
    //а это уравнение x2 = (d - ax1 - cx3) / b

    x3 = (third[3] - third[1] * x2 - third[0] * x1) / third[2];
    //а это уравнение x3 = (d - bx2 - ax1) / c
}

```

Вивід(до п.8):

Консоль отладки Microsoft Visual Studio

```

0 iteration      x1 = 10.8        x2 = -0.24       x3 = 4.06
1 iteration      x1 = 2.52917     x2 = -4.12965    x3 = -7.03485    FLUFF = 11.0948
2 iteration      x1 = 3.84964     x2 = 0.521921    x3 = -0.161106   FLUFF = 6.87374
3 iteration      x1 = 2.00407     x2 = -1.2412     x3 = -1.56387    FLUFF = 1.84556
4 iteration      x1 = 2.75372     x2 = -0.417836   x3 = 0.0377597   FLUFF = 1.60163
5 iteration      x1 = 2.44308     x2 = -0.958647   x3 = -0.620928   FLUFF = 0.658688
6 iteration      x1 = 2.66351     x2 = -0.735544   x3 = -0.332857   FLUFF = 0.288072
7 iteration      x1 = 2.57326     x2 = -0.85799    x3 = -0.525098   FLUFF = 0.192241
8 iteration      x1 = 2.62137     x2 = -0.792997   x3 = -0.44403    FLUFF = 0.0810679
9 iteration      x1 = 2.59496     x2 = -0.823361   x3 = -0.487223   FLUFF = 0.0431937
10 iteration     x1 = 2.60707     x2 = -0.806954   x3 = -0.463915   FLUFF = 0.0233088
11 iteration     x1 = 2.60052     x2 = -0.81518    x3 = -0.474796   FLUFF = 0.0108815
12 iteration     x1 = 2.60384     x2 = -0.811077   x3 = -0.468964   FLUFF = 0.00583217
13 iteration     x1 = 2.6022      x2 = -0.813219   x3 = -0.471913   FLUFF = 0.00294904
14 iteration     x1 = 2.60306     x2 = -0.812148   x3 = -0.470448   FLUFF = 0.0014646
15 iteration     x1 = 2.60263     x2 = -0.812693   x3 = -0.471213   FLUFF = 0.000764329
16 iteration     x1 = 2.60285     x2 = -0.812414   x3 = -0.470829   FLUFF = 0.000383378
17 iteration     x1 = 2.60274     x2 = -0.812555   x3 = -0.471024   FLUFF = 0.000194717
18 iteration     x1 = 2.60274     x2 = -0.812555   x3 = -0.471024   FLUFF = 0
first: 10.8001   real be === 10.8        NEVAZKA: 0.000138221
second: -0.239938   real be === -0.24        NEVAZKA: 6.21548e-05
third: 4.06013   real be === 4.06        NEVAZKA: 0.000132908

```


Після редагування коду:

```
Консоль отладки Microsoft Visual Studio

0 iteration      x1 = 10.8      x2 = -0.24      x3 = 4.06
1 iteration      x1 = 2.52917   x2 = -1.69705   x3 = -0.345394   FLUFF = 8.27083
2 iteration      x1 = 3.01342   x2 = -0.958398   x3 = -0.786556   FLUFF = 0.738651
3 iteration      x1 = 2.65649   x2 = -0.765187   x3 = -0.517272   FLUFF = 0.35693
4 iteration      x1 = 2.57916   x2 = -0.796298   x3 = -0.453398   FLUFF = 0.0773352
5 iteration      x1 = 2.59608   x2 = -0.814049   x3 = -0.465513   FLUFF = 0.0177514
6 iteration      x1 = 2.60371   x2 = -0.813871   x3 = -0.471597   FLUFF = 0.00763127
7 iteration      x1 = 2.60337   x2 = -0.812555   x3 = -0.47143    FLUFF = 0.00131539
8 iteration      x1 = 2.60278   x2 = -0.812414   x3 = -0.470967   FLUFF = 0.000595924
9 iteration      x1 = 2.60278   x2 = -0.812414   x3 = -0.470967   FLUFF = 0
first: 10.7998   real be === 10.8      NEVAZKA: 0.000182737
second: -0.239986   real be === -0.24      NEVAZKA: 1.44042e-05
third: 4.06       real be === 4.06      NEVAZKA: 0
```

*знадобилося у два рази менше ітерацій

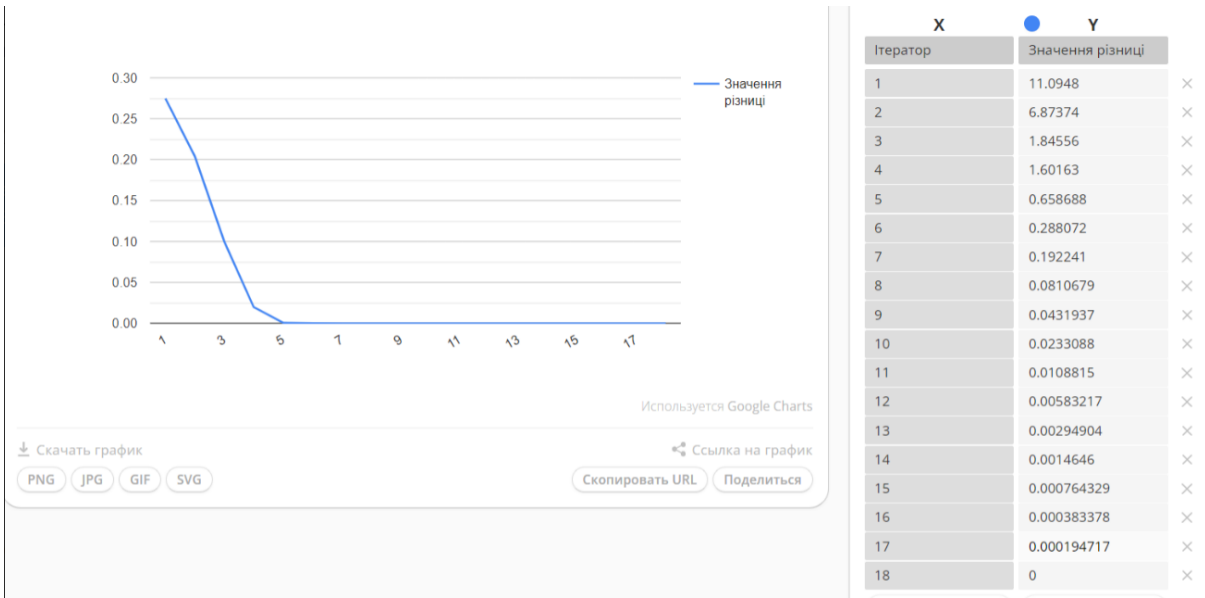
*розв'язки схожі

Процес розрахунку для графіку залежності:

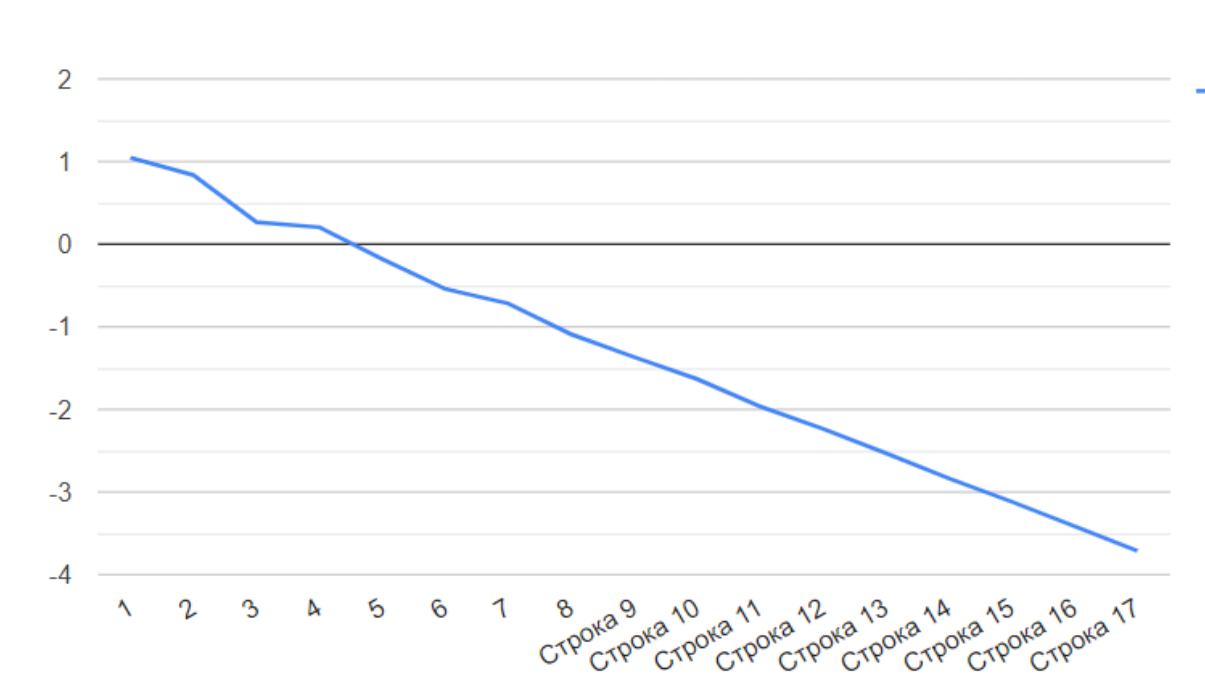
The image shows a web application interface for calculating logarithms. The main input field shows $\log_{10} 0.658688 = -0,181$. Below the input field are buttons labeled "РАССЧИТАТЬ" (Calculate) and "СБРОСИТЬ" (Reset). The application also displays a table of iteration results, which is shown in a separate window below the main interface.

Итератор	lg &(k)
1	1,045
2	0,837
3	0,266
4	0,205
5	-0,181
6	-2,117
7	-2,881
8	-3,225

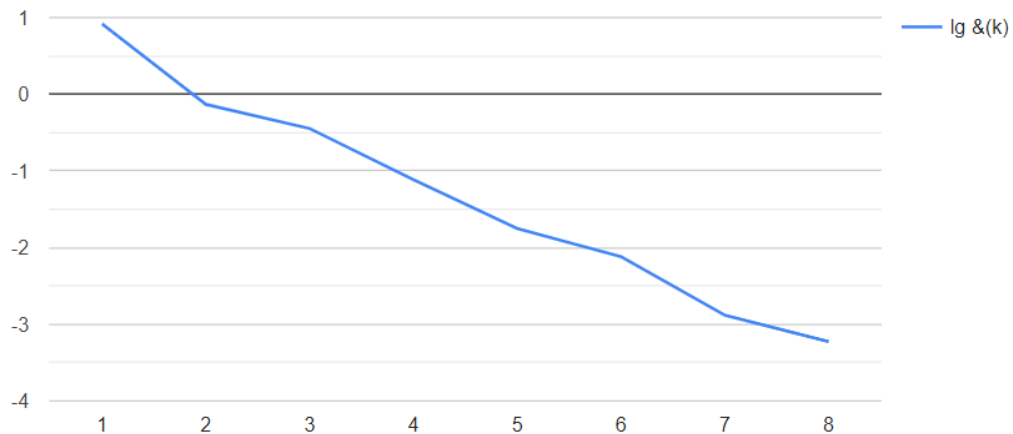
Графік(з даних Якобі - значення різниці ітератора):



Графік(з даних Якобі - залежності $\lg \delta_k$):



Графік(з даних методу Зейделя залежності $\lg \delta_k$):



Висновок: з кожною новою ітерацією похибка менше

Непосредственно скорость сходимости оценивают по тангенсу угла наклона логарифмического графика зависимости $\|x_n - x^*\|$ от $\|x_{n-1} - x^*\|$.

Виходячи з цього(дуже приблизно),
порядок збіжності по Якобі $\text{tg}(\alpha) \approx f'(x) \approx -1$
а порядок збіжності по методу Зейделя ≈ -0.3679

Джерела:

[Итерационные методы решения СЛАУ: метод Якоби, Зейделя, простой итерации \(zaoschnik.com\)](http://zaoschnik.com)

[1.2.1. Метод простой итерации \(метод Якоби\) \(matica.org.ua\)](http://matica.org.ua)

[1.2.3. Метод Зейделя \(метод Гаусса-Зейделя, метод последовательных замещений\) \(matica.org.ua\)](http://matica.org.ua)

[Скорость сходимости — Википедия \(wikipedia.org\)](http://wikipedia.org)

[Вычисление логарифма числа онлайн | umath.ru](http://umath.ru)

І посібники