Звіт до лабораторної роботи №4

з навчальної дисципліни

«Чисельні методи»

на тему:

«РОЗВ’ЯЗАННЯ СИСТЕМ НЕЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ. ВЛАСНІ ЧИСЛА МАТРИЦІ. ІНТЕГРУВАННЯ»

студента II курсу групи К-24

Бондаря Дениса

варіант 2

Формула Сімпсона

Теоретичні відомості

Формулою Сімпсона називається інтеграл від інтерполяційного многочлена другого степеня на відрізку $[a\,,b]$

$$\int_a^b f(x)dx \approx \int_a^b p_2(x)dx = \frac{b-a}{6}(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\ \text{, де}$$

$f(a)\,, f\left(\frac{a+b}{2}\right)$ і $f(b)-$ значення функції у відповідних точках.

При умові, що функція $f(x)$ на відрізку $[a\,,b]$ має похідну четвертого порядку, похибка $E(f)$ дорівнює:

$$E(f) = -\frac{(b-a)^5}{2880}f^{(4)}(e), \qquad e\ \in [a\,,b]$$

Зважаючи, що значення e переважно не є відомим, для оцінки похибки використовується нерівність:

$$|E(f)| \leq \frac{(b-a)^5}{2880} max_{x\in[a\,,b]}|f^{(4)}(x)|$$

Постанова задачі

Порахувати інтеграл $\int_0^1 2x^4 + 2x^2$ методом Сімпсона

Роз’язок цієї задачі має такі етапи:

1) Розрахунок проміжку
2) Розрахунок «допоміжних» сум
3) Розрахунок основної суми
4) Збереження минулого інтегралу (потрібно для умови завершення процесу)
5) Обчислення інтегралу
6) Власне перевірка умови на закінчення процесу

```
double IntegrateFunction(const double A, const double B, const double EPSILON)
{
    double I = EPSILON + 1;
    double I1 = 0;

    for (int n = 2; (n <= 4) || (fabs(I1 - I) > EPSILON); n *= 2)   6)
    {
        double distance = (B - A) / (2.0 * n);                      1)
        double sum = 0;
        double sum2 = 0;
        double sum4 = 0;

        for (int i = 1; i <= 2 * n - 1; i += 2)                     2)
        {
            sum4 += f(A + distance * i);
            sum2 += f(A + distance * (i + 1.0));
        }

        sum = f(A) + 4 * sum4 + 2 * sum2 - f(B);                    3)

        I = I1;                                                     4)
        I1 = (distance / 3) * sum;                                  5)
    }

    return I1;
}
```

```
Консоль отладки Mic
Integral = 1.06673
```

Метод Ньютона

Теоретичні відомості

Лінеаризуючи рівняння

$$\bar{F}(\bar{x}) = \bar{0} \text{ , де}$$

$$\bar{F} = (f_1, \dots, f_n)^T, \qquad \vec{x} = (x_1, \dots, x_n)^T$$

в околі наближення до розв'язку $\vec{x}$, отримаємо систему лінійних рівнянь відосно нового наближення $\overset{\to k+1}{x}$:

$$\bar{F}(x^{\to k}) + F'(x^{\to k})(x^{\to k+1} - x^{\to k}) = \bar{0} \quad (5.4)$$

Можна запропонувати такий алгоритм розв'язання рівняння (5.4)

1) Задати початкове наближення $x^{\to 0}$;
2) Обчислимо матрицю Якобі $A_k = (\frac{\partial f_i}{\partial x_j}(x^{\to k}))_{i,j=1}^{n}$
3) Розв'язати СЛАР $A_k z^{\to k} = \bar{F}(x^{\to k})$
4) Обчислити нове наближення $x^{\to k+1} = x^{\to k} - z^{\to k}$
5) Перевірити умову $\left|\left|z^{\to k}\right|\right| < \varepsilon$. Якщо її виконано, припинити процес, а не то повторити обчислення, починаючи з 2

Як і для одного рівняння, метод Ньютона збігається, якщо початкове наближення $x^{\to 0}$ близьке до розв'язку $x^{\to}$. Для систем зберігається властивість квадратичної швидкості збіжності в околі розв'язку.

Оскільки для реалізації методу Ньютона потрібно розв'язувати СЛАР, то в разі довільної матриці A_k застосовують метод Гауса, для чого потрібно виконати $\frac{2}{3}n^3 + O(n^2)$ арифметичних операцій.

У разі симетричної матриці A_k доцільніше застосувати метод квадратних коренів, а в разі тридіагональної матриці – метод прогонки.

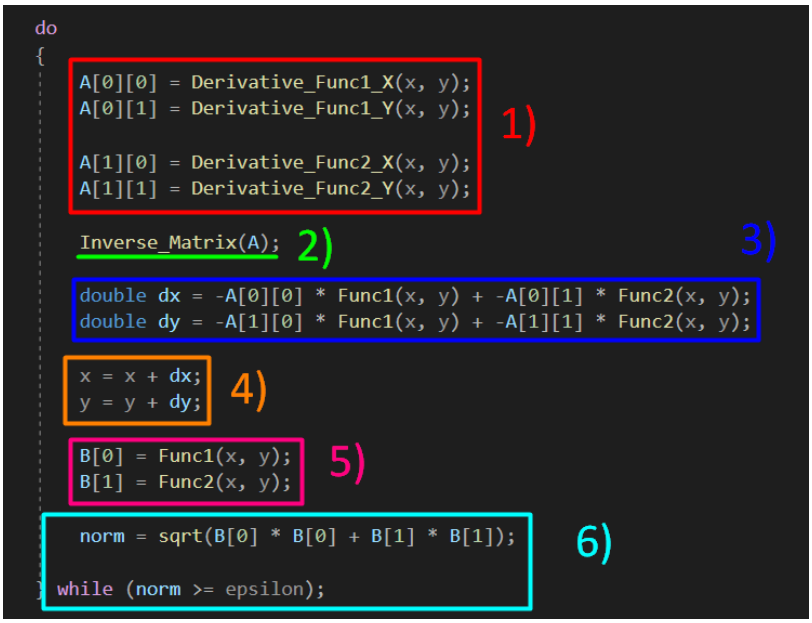
Постанова задачі

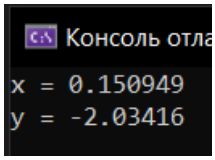
Розв’язати систему:

$$\begin{cases} \sin(x - 0.6) - y = 1.6 \\ 3x - \cos(y) = 0.9 \end{cases}$$

Роз’язок цієї задачі має такі етапи:

1) Обчислення матриці Якобі
2) Обернення матриці
3) Розрахунок приросту по x, y
4) Розрахунок роз’язку x , y
5) Розрахунок значень функцій (потрібно для умови завершення процесу)
6) Власне перевірка умови на закінчення процесу





Степеневий метод

Теоретичні відомості

На початку алгоритму генерується випадковий вектор x_0. Далі проводяться послідовні обчислення по ітеративній формулі:

$$x_{k+1} = \frac{A_{x_k}}{\left|\left|A_{x_k}\right|\right|}$$

Якщо початковий вектор не ортогональний власному підпростору з найбільшим по модулю власним значенням, то відстань від елементів даної послідовності до такого підпростору збігається до нуля.

Послідовність векторів не завжди збігається, оскільки вектор на кожному кроці може змінювати знак або в комплексному випадку обертатися, але це не заважає вибрати один з векторів в якості власного, коли отримано достатньо точне власне значення

Послідновність:

$$\mu_k = \frac{x_k^T A_{x_k}}{x_k^T x_k} = \frac{(x_k\ , A_{x_k})}{(x_k\ , x_k)}$$

при зазначеній вище умові збігається до максимального за модулем власним значенням. Але слід пам'ятати, що не у всіх дійсних матриць є дійсні власні значення.

Алгоритм простий і збігається зі швидкістю геометричної прогресії, якщо всі максимальні за модулем власні значення збігаються, в іншому випадку збіжності немає

В силу того, що алгоритм зводиться до послідовного множення заданої матриці на вектор, при правильній реалізації він добре працює для великих розріджених матриць.

Постанова задачі

Знайти максимальне власне значення матриці:

$$\begin{matrix} 3.1 & 1.0 & 2.1 \\ 1.0 & 3.6 & 2.1 \\ 2.1 & 2.1 & 4.1 \end{matrix}$$

Роз’язок цієї задачі має такі етапи:

1) Генерація випадкового початкового вектора x_0.
2) Розрахунок нового вектору
3) Зберігання минулого власного значення (потрібно для умови завершення процесу)
4) Розрахунок власного значення
5) Власне перевірка умови на закінчення процесу

```
double GetMaxEigenValue(const Matrix& matrix, const double epsilon)
{
    Vector Xn(matrix.size(), 1);
    Vector Xn1(matrix.size());

    double answer = 0;
    double answerPrev = 0;

    do
    {
        Xn1 = matrix * Xn;

        answerPrev = answer;

        answer = DotProduct(Xn, Xn1) / DotProduct(Xn, Xn);

        Xn = Xn1;

    } while ((fabs(answer - answerPrev)) > epsilon);

    return answer;
}
```

1) 2) 3) 4) 5)

```
Консоль отладки Microsoft Visu
Max eigenvalue = 9.62348
```