ЛАБОРАТОРНАЯ РАБОТА №1

«МЕТОД НЬЮТОНА ДЛЯ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ»

Выполнил

Святослав Артюшкевич, 3 группа, 2 курс

Условие задачи

Имеется система нелинейных уравнений вида

$$A_0 x_0^i + \cdots + A_m x_m^i = g_i, i = 0, \dots, 2m+1$$

Здесь $\{A_k\}_k^{m=0}, \{x_k\}_{k=0}^m$ –неизвестные величины, $\{g_i\}_{i=0}^{2m+1}$ – числовые коэффициенты. $g_i = \int_{-1}^{1}(1-x)^2(1+x)x^i dx\,, m = 1$

Задание:

- Реализовать метод Ньютона для решения системы.
- Провести вычислительный эксперимент: взяв несколько начальных приближений, при которых итерационный процесс сходится, найти решение с точностью до 10^{-10}
- Построить логарифмические диаграммы сходимости.

Теоретические сведения

Общее описание алгоритма

Метод Ньютона решения систем нелинейных уравнений является обобщением метода Ньютона решения нелинейных уравнений, который основан на идее линеаризации. Пусть $F(x)\colon \mathbb{R}^1 \to \mathbb{R}^1$ - дифференцируемая функция и необходимо решить уравнение $F(x) = 0$

Взяв некоторое x_0 в качестве начального приближения решения, мы можем построить линейную аппроксимацию$F(x)$ в окрестности $x_0\colon F(x_0 + h) \approx F(x_0) + F'(x_0)h$ и решить получающееся линейное уравнение $F(x_0) + F'(x_0)h = 0$

Таким образом получаем итеративный метод : $x_{k+1} = x_k - F'(x_k)^{-1}F(x_k), k = 0, 1, \dots$

Данный метод был предложен Ньютоном в 1669 году. Более точно, Ньютон оперировал только с полиномами; в выражении для $F(x+h)$ он отбрасывал члены более высокого порядка по h, чем линейные. Ученик Ньютона Рафсон в 1690 г. предложил общую форму метода (т. е. не предполагалось что $F(x)$ обязательно полином и использовалось понятие производной), поэтому часто говорят о методе Ньютона—Рафсона. Дальнейшее развитие исследований связано с именами таких известных математиков, как Фурье, Коши и другие. Например, Фурье доказал в 1818 г., что метод сходится квадратично в окрестности корня, а Коши (1829, 1847) предложил многомерное обобщение метода и использовал метод для доказательства существования решения уравнения.

Математическое описание алгоритма

Пусть дана система из n нелинейных уравнений с n неизвестными.

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases}$$, где $f_i(x_1, x_2, \dots, x_n)\colon \mathbb{R}^n \to \mathbb{R}, i = 1, \dots, n$ - нелинейные функции, определенные и непрерывно дифференцируемые в некоторой области $G \subset \mathbb{R}^n$.

Запишем ее в векторном виде: $\overline{x} = (x_1, x_2, \dots, x_n)^T, F(x) = [f_1(x), f_2(x), \dots, f_n(x)]^T, F(x) = 0$

Требуется найти такой вектор $\overline{x^*} = (x_1^*, x_2^*, \dots, x_n^*)^T$, который, при подстановке в исходную систему, превращает каждое уравнение в верное числовое равенство.

При таком подходе формула для нахождения решения является естественным обобщением формулы одномерного итеративного метода: $x^{(k+1)} = x^{(k)} - W^{-1}\left(x^{(k)}\right) * F\left(x^{(k)}\right), k = 0, 1, 2, \dots$ где $W = \begin{pmatrix} \frac{\partial f_1(x_1)}{\partial x_1} & \dots & \frac{\partial f_1(x_n)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(x_1)}{\partial x_1} & \dots & \frac{\partial f_n(x_n)}{\partial x_n} \end{pmatrix}$ – матрица Якоби.

В рассмотренных предположениях относительно функции $F(\cdot)$ при выборе начального приближения $x^{(0)}$ из достаточно малой окрестности решения $\overline{x^*}$ имеет место сходимость последовательности $\{x^{(k)}\}$ При дополнительном предположении $F(\cdot) \in C^2$ имеет место квадратичная сходимость метода.

В качестве критерия окончания процесса итераций обычно берут условие $\| x^{(k+1)} - x^{(k)} \| < \varepsilon$ где ε - требуемая точность решения.

Основная сложность метода Ньютона заключается в обращении матрицы Якоби. Вводя обозначение $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ получаем СЛАУ для вычисления $\Delta x^{(k)}$: $\frac{\partial F(x^{(k)})}{\partial x} = -F\left(x^{(k)}\right)$

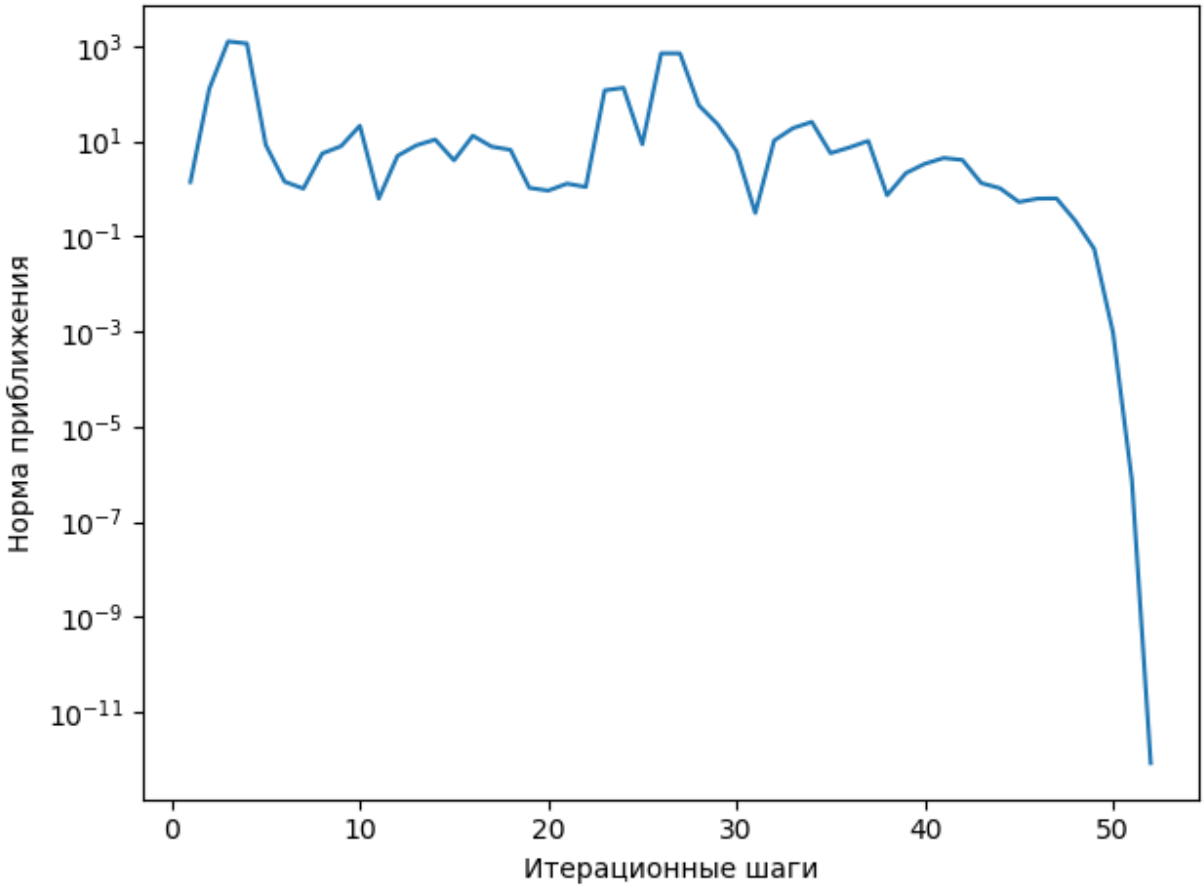
Тогда $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$

Эксперименты

Эксперимент 1

Начальное приближение: [-0.55984869 -0.29592335 0.85976977 0.57825904]

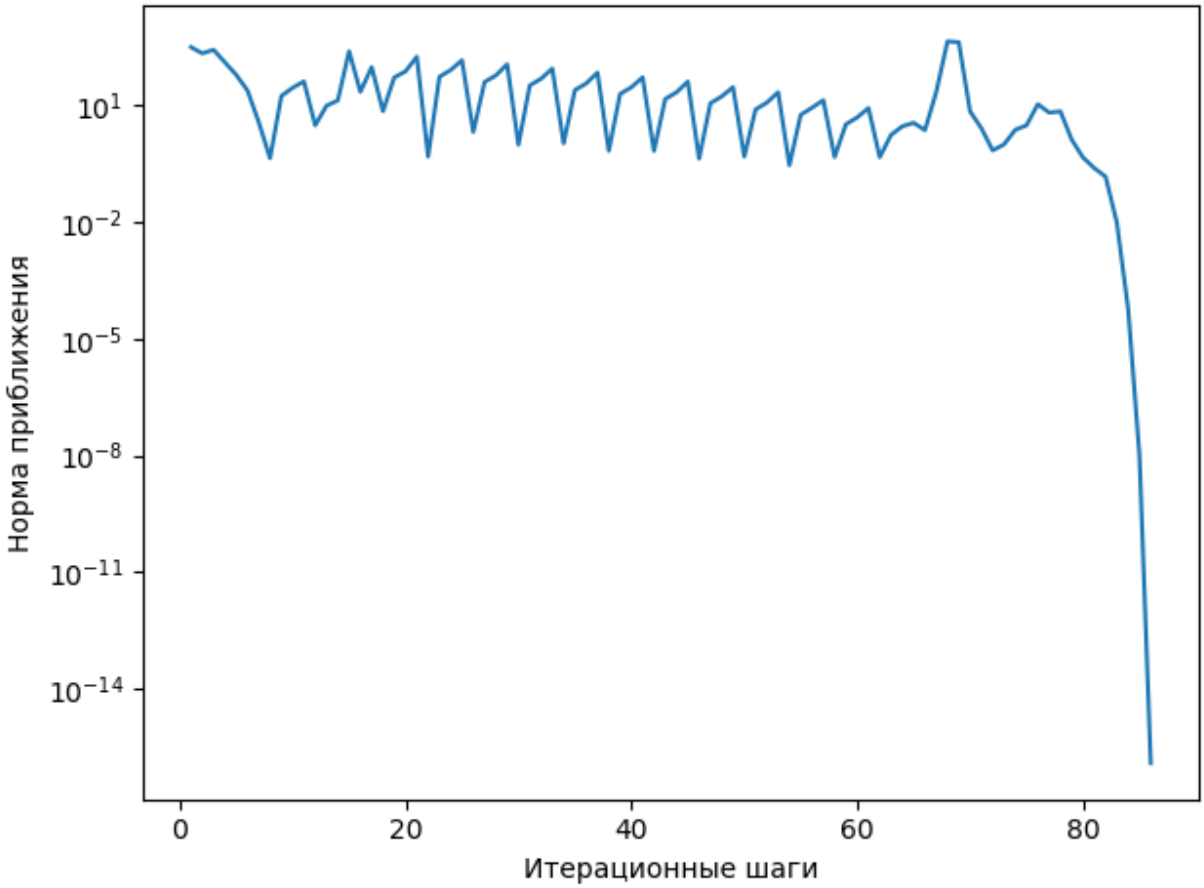
Ответ: [0.57238576 0.26120387 0.76094757 -0.54691816]



Эксперимент 2

Начальное приближение: [0.0717428 0.21955504 0.20665294 0.40031826]

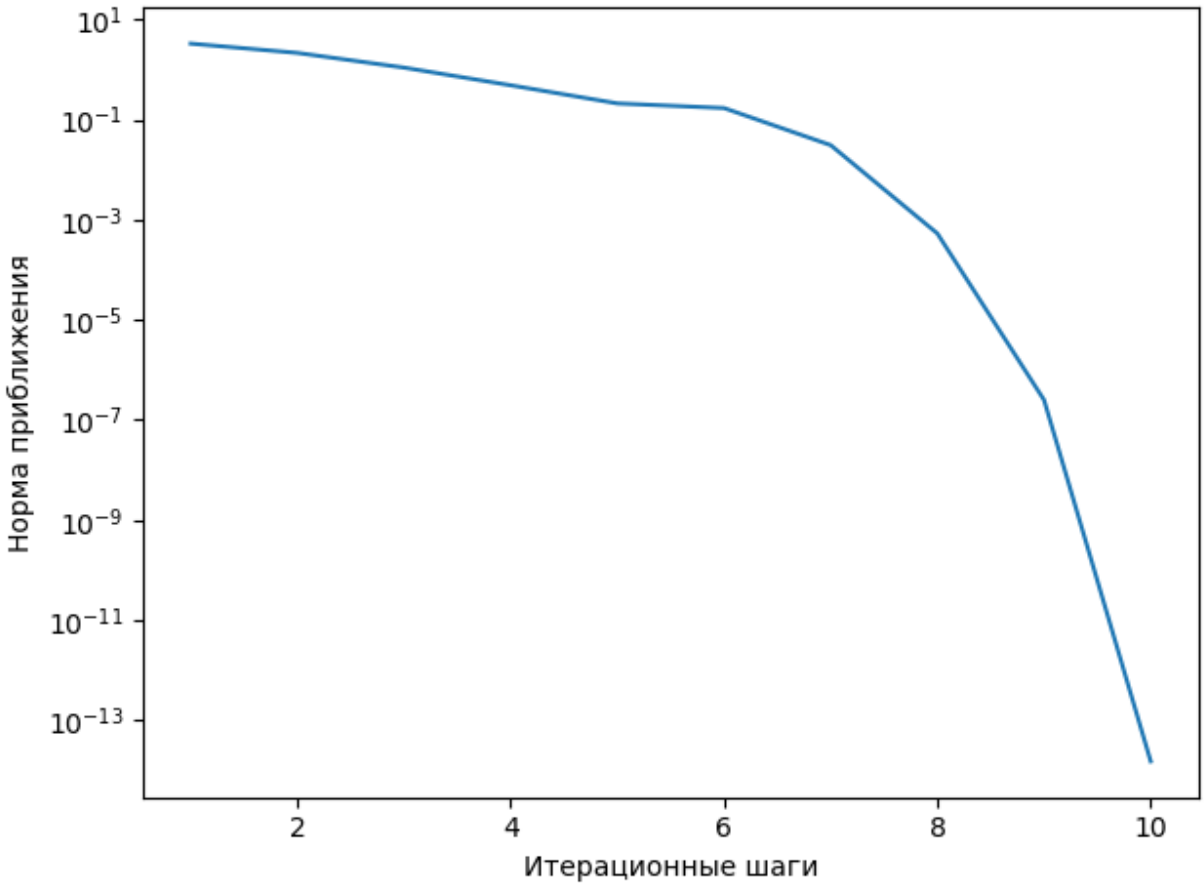
Ответ: [0.76094757 -0.54691816 0.57238576 0.26120387]



Эксперимент 3

Начальное приближение: [-0.7460998 0.29395764 -0.07553926 -0.36921346]

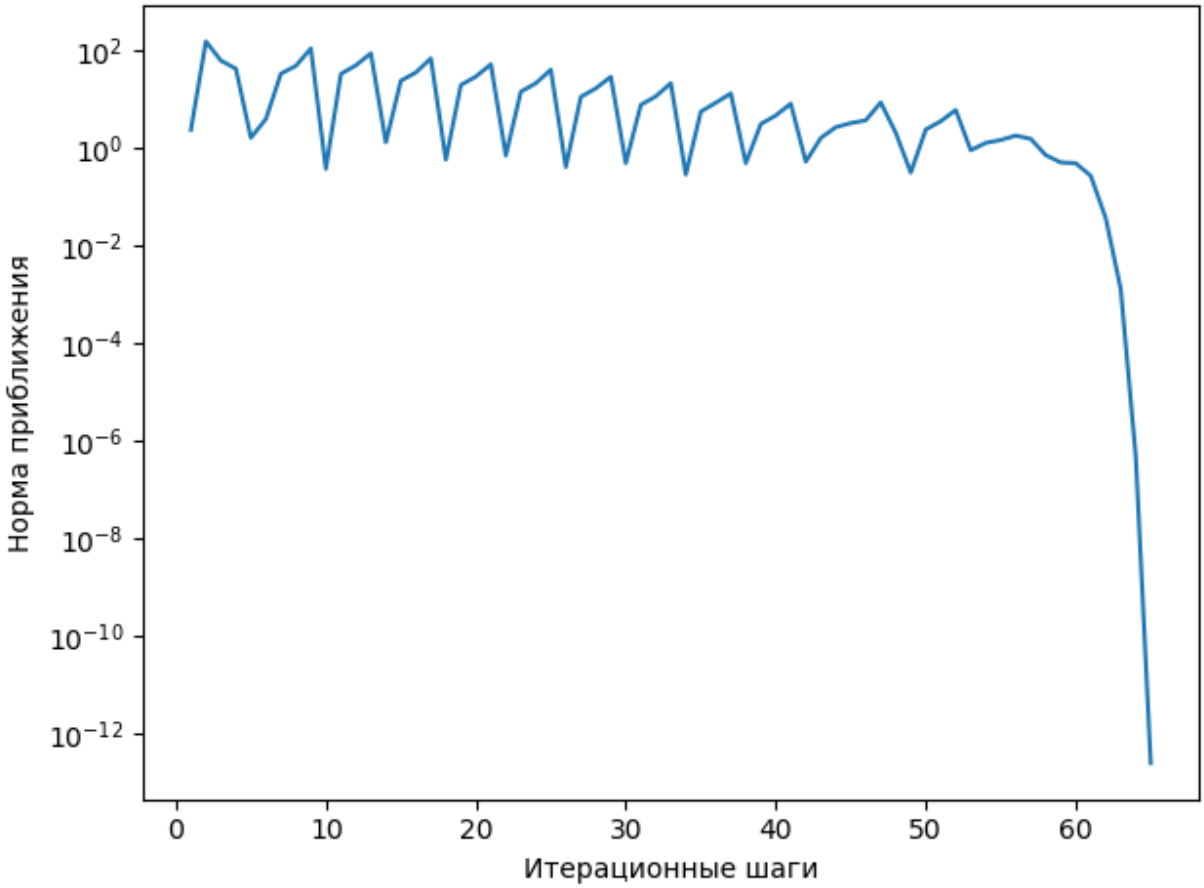
Ответ: [0.76094757 -0.54691816 0.57238576 0.26120387]



Эксперимент 4

Начальное приближение: [-0.47911843 0.84242057 -0.82632025 -0.03822266]

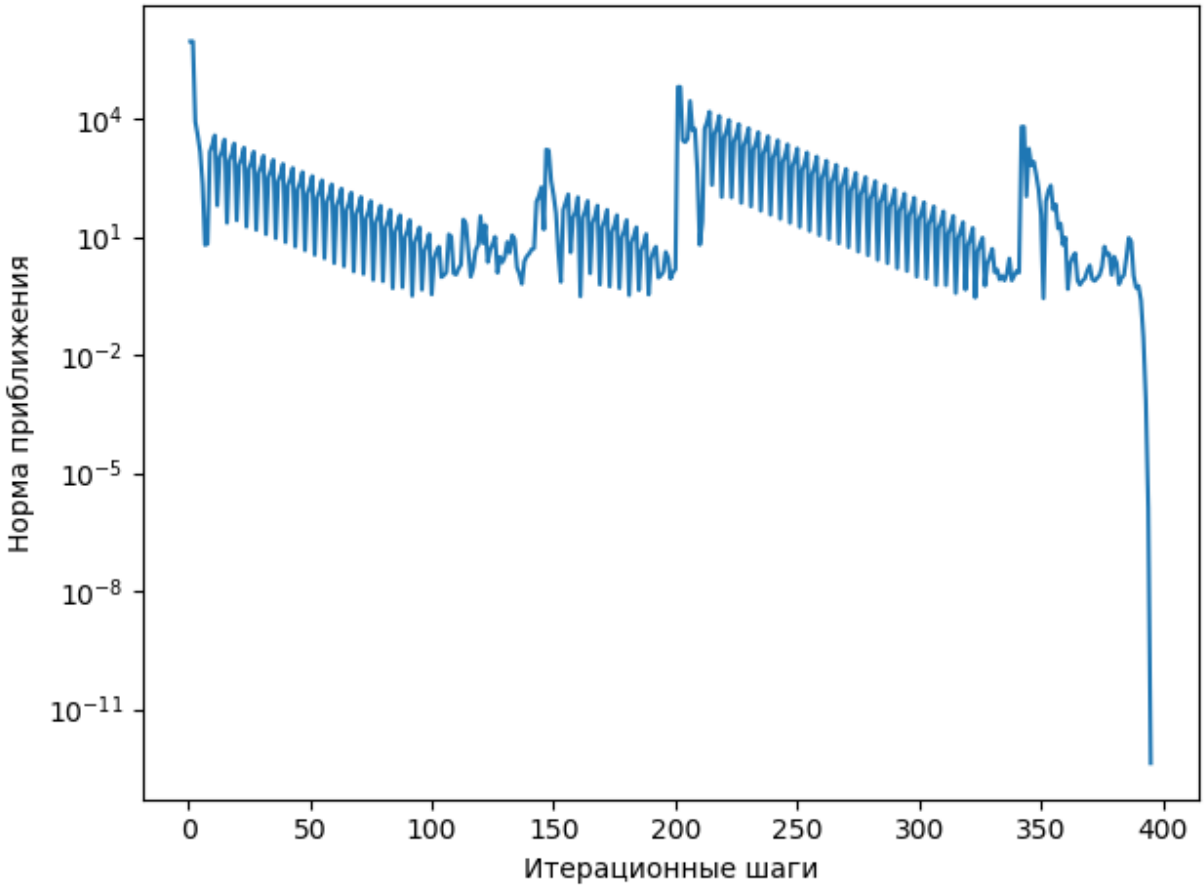
Ответ: [0.57238576 0.26120387 0.76094757 -0.54691816]



Эксперимент 5

Начальное приближение: [-0.61591991 0.77250762 0.39969757 0.75480238]

Ответ: [0.57238576 0.26120387 0.76094757 -0.54691816]



Код решения

```
import scipy.integrate
import random
import numpy as np
```

```
import matplotlib.pyplot as plt


def function(x, i):
    return ((1 - x) ** 2) * (1 + x) * (x ** i)


def g_counter(size):
    ans = []
    for i in range(2 * size + 2):
        ans.append(scipy.integrate.quad(function, -1, 1, i)[0])
    return ans


def jacobi_matrix(v, size):
    ans = np.array([[float(0) for i in range(2 * size + 2)] for i in range(2
* size + 2)])
    for i in range(0, 2 * size + 2, 2):
        ans[0][i] = 1
    for i in range(1, 2 * size + 2):
        for j in range(2 * size + 2):
            if j % 2 == 0:
                ans[i][j] = ans[i - 1][j] * v[j + 1]
            else:
                ans[i][j] = v[j - 1] * i * ans[i - 1][j - 1]
    return ans


def f(x, b, size):
    ans = np.array([float(0) for i in range(2 * size + 2)])
    for i in range(len(ans)):
        for j in range(0, len(x), 2):
            ans[i] += x[j] * (x[j + 1] ** i)
        ans[i] -= b[i]
    return ans


m = 1
g = np.array(g_counter(m))
Ax = np.array([random.uniform(-1, 1) for i in range(len(g))])
print(Ax)
S = np.inf
eps = 10 ** -10
k = 0
convergence = []
while S > eps:
    omega = jacobi_matrix(Ax, m)
    delta_x = np.linalg.solve(omega, f(Ax, g, m))
    S = np.linalg.norm(delta_x)
    Ax -= delta_x
    k += 1
    convergence.append(S)
print(Ax)
plt.plot([int(i) for i in range(1, k + 1)], convergence)
plt.xlabel("Итерационные шаги")
plt.ylabel("Норма приближения")
plt.yscale("log")
plt.show()
```