МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО

ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

«Национальный исследовательский ядерный университет
«МИФИ» (НИЯУ МИФИ)

Институт Интеллектуальных Кибернетических Систем
Кафедра Кибернетики

Лабораторная работа №2:

По курсу «Численные методы»

Работу выполнил: студент группы Б22-511: Рябов Н.А.

Проверил: Саманчук В.Н.

Москва 2024

Постановка задачи

Решить систему нелинейных уравнений методом Ньютона:

$$\begin{cases} x^5 - 2{,}1 \cdot z^2 - 3 \cdot x^2 \cdot y^4 = 17{,}9 \\ 0{,}6 \cdot y \cdot z^3 + 1{,}7 \cdot x^2 \cdot y^3 - 20{,}9 = -14{,}7 \\ 5{,}2 \cdot y^5 - 2{,}5 \cdot z^4 \cdot x^2 = -4{,}8 \end{cases}$$

Методика решения

Для решения поставленной задачи была написана программа на языке программирования Python, в которой реализован метод Ньютона для решения системы нелинейных уравнений.

Теоретическая справка

Рассмотрим систему нелинейных уравнений

$$F(x) = 0,\ F(x),\ x \in \mathbb{R}^n, \qquad (1)$$

и предположим, что существует вектор $\bar{x} \in D \subset \mathbb{R}^n$, являющийся решением системы (1). Будем считать, что $F(x) = (f_1(x),\ f_2(x),\ \ldots,\ f_n(x))^T$, причём $f_i(\cdot) \in C^1(D)\ \ \forall i$.

Разложим $F(x)$ в окрестности точки $\bar{x}:\ F(x) = F(x^0) + F'(x^0)(x - x^0) + o(\|x - x^0\|)$. Здесь

$$F'(x) = \frac{\partial F(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1}, & \frac{\partial f_1(x)}{\partial x_2}, & \ldots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1}, & \frac{\partial f_2(x)}{\partial x_2}, & \ldots & \frac{\partial f_2(x)}{\partial x_n} \\ \ldots & \ldots & \ldots & \ldots \\ \frac{\partial f_n(x)}{\partial x_1}, & \frac{\partial f_n(x)}{\partial x_2}, & \ldots & \frac{\partial f_n(x)}{\partial x_n} \end{bmatrix}$$

называется матрицей Якоби, а её определитель – якобианом системы (1). Исходное уравнение заменим следующим: $F(x^0) + F'(x^0)(x - x^0) = 0$. Считая матрицу Якоби $F'(x^0)$ неособой, разрешим это уравнение относительно $x:\ \hat{x} = x^0 - [F'(x)]^{-1} F(x^0)$. И вообще положим

$$x^{k+1} = x^k - \left[F'\left(x^k\right)\right]^{-1} F\left(x^k\right). \tag{2}$$

При сделанных относительно $F(\cdot)$ предположениях имеет место сходимость последовательности $\{x^k\}$ к решению системы со скоростью геометрической прогрессии при условии, что начальное приближение x^0 выбрано из достаточно малой окрестности решения $\bar{x}$.

При дополнительном предположении $F(\cdot) \in C^2[a, b]$ имеет место квадратичная сходимость метода, т.е. $\left|\left|x^{k+1} - \bar{x}\right|\right| \leq \omega \left|\left|x^k - \bar{x}\right|\right|^2$.

Решение задачи

```
import sympy as sp
import numpy as np

# Определение переменных и функций
x, y, z = sp.symbols('x y z')

f1_expr = x ** 5 - 2.1 * z ** 2 - 3 * x ** 2 * y ** 4 - 17.9
f2_expr = 0.6 * y * z ** 3 + 1.7 * x ** 2 * y ** 3 - 20.9 + 14.7
f3_expr = 5.2 * y ** 5 - 2.5 * z ** 4 * x ** 2 + 4.8

# Производные по x, y, z для всех функций
f1_dx = sp.diff(f1_expr, *symbols: x)
f1_dy = sp.diff(f1_expr, *symbols: y)
f1_dz = sp.diff(f1_expr, *symbols: z)

f2_dx = sp.diff(f2_expr, *symbols: x)
f2_dy = sp.diff(f2_expr, *symbols: y)
f2_dz = sp.diff(f2_expr, *symbols: z)

f3_dx = sp.diff(f3_expr, *symbols: x)
f3_dy = sp.diff(f3_expr, *symbols: y)
f3_dz = sp.diff(f3_expr, *symbols: z)
```

```
# Функция для вычисления значений функций
def f1(x0, y0, z0):  2 usages
    return f1_expr.subs([(x, x0), (y, y0), (z, z0)])


def f2(x0, y0, z0):  2 usages
    return f2_expr.subs([(x, x0), (y, y0), (z, z0)])


def f3(x0, y0, z0):  2 usages
    return f3_expr.subs([(x, x0), (y, y0), (z, z0)])


# Функция для вычисления Якобиана
def jacobian(x0, y0, z0):  1 usage
    return np.array( object: [
        [f1_dx.subs([(x, x0), (y, y0), (z, z0)]), f1_dy.subs([(x, x0), (y, y0), (z, z0)]),
         f1_dz.subs([(x, x0), (y, y0), (z, z0)])],
        [f2_dx.subs([(x, x0), (y, y0), (z, z0)]), f2_dy.subs([(x, x0), (y, y0), (z, z0)]),
         f2_dz.subs([(x, x0), (y, y0), (z, z0)])],
        [f3_dx.subs([(x, x0), (y, y0), (z, z0)]), f3_dy.subs([(x, x0), (y, y0), (z, z0)]),
         f3_dz.subs([(x, x0), (y, y0), (z, z0)])]
    ], dtype='float')
```

```python
# Реализация метода Ньютона
def newton_method(x0, y0, z0, eps):
    xi, yi, zi = x0 + 2 * eps, y0 + 2 * eps, z0 + 2 * eps
    i = 0
    while abs(xi - x0) > eps or abs(yi - y0) > eps or abs(zi - z0) > eps:
        i += 1
        J = jacobian(x0, y0, z0)  # Якобиан (матрица производных)
        F = np.array([-f1(x0, y0, z0), -f2(x0, y0, z0), -f3(x0, y0, z0)], dtype='float')  # Значения функций

        # Решаем линейную систему для нахождения поправок
        try:
            dx, dy, dz = np.linalg.solve(J, F)
        except np.linalg.LinAlgError:
            print("Ошибка: Якобиан вырожден. Метод не сходится.")
            return

        xi, yi, zi = x0, y0, z0
        x0 += dx
        y0 += dy
        z0 += dz
        print(f'Шаг {i}: x = {x0}, y = {y0}, z = {z0}')

    print('Значения функций в корне:')
    print(f'f1(x, y, z) = {f1(x0, y0, z0)}')
    print(f'f2(x, y, z) = {f2(x0, y0, z0)}')
    print(f'f3(x, y, z) = {f3(x0, y0, z0)}')
    print(f'Решение: x = {x0}, y = {y0}, z = {z0}')


# Пример использования
newton_method(2, 1, 1, 0.000001)
```

Результат работы

```
Шаг 1: x = 1.9671468676840582, y = 0.9552826968969883, z = 0.9791470360620278
Шаг 2: x = 1.9666978056461069, y = 0.9515779981460518, z = 0.9782801988752263
Шаг 3: x = 1.966701477639883, y = 0.9515604103213283, z = 0.9782844827650904
Шаг 4: x = 1.9667014777186822, y = 0.9515604100361266, z = 0.9782844829153441
Значения функций в корне:
f1(x, y, z) = 0
f2(x, y, z) = -2.220446049250313E-16
f3(x, y, z) = -3.552713678800501E-15
Решение: x = 1.9667014777186822, y = 0.9515604100361266, z = 0.9782844829153441
```

Заключение

В работе требовалось решить систему нелинейных уравнений методом Ньютона. Для решения задачи была написана программа на языке программирования Python.

Ответ: x = 1.9667014777186822, y = 0.9515604100361266, z = 0.9782844829153441