НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет Программной инженерии и компьютерных технологий

Направление: Нейротехнологии и программная инженерия

Дисциплина: Вычислительная математика

Лабораторная работа № 3

“Метод Ньютона”

Выполнил студент

Рязанов Демид Витальевич

Группа P3221

Преподаватель: Перл Ольга Вячеславовна

г. Санкт-Петербург

2024

Содержание

Описание метода

Метод Ньютона – итерационный алгоритм для поиска решений СНАУ. На каждой итерации метода получается более точное приближение решения, и при достижении определенной точности алгоритм завершает свою работу. Метод основан на локальной аппроксимации функций СНАУ и их Якобиане.

Алгоритм:

1) Сначала задаём начальное приближение $X^{(0)}$ и ϵ для определения точности.

2) Затем вычисляем матрицу Якоби для текущего приближения.

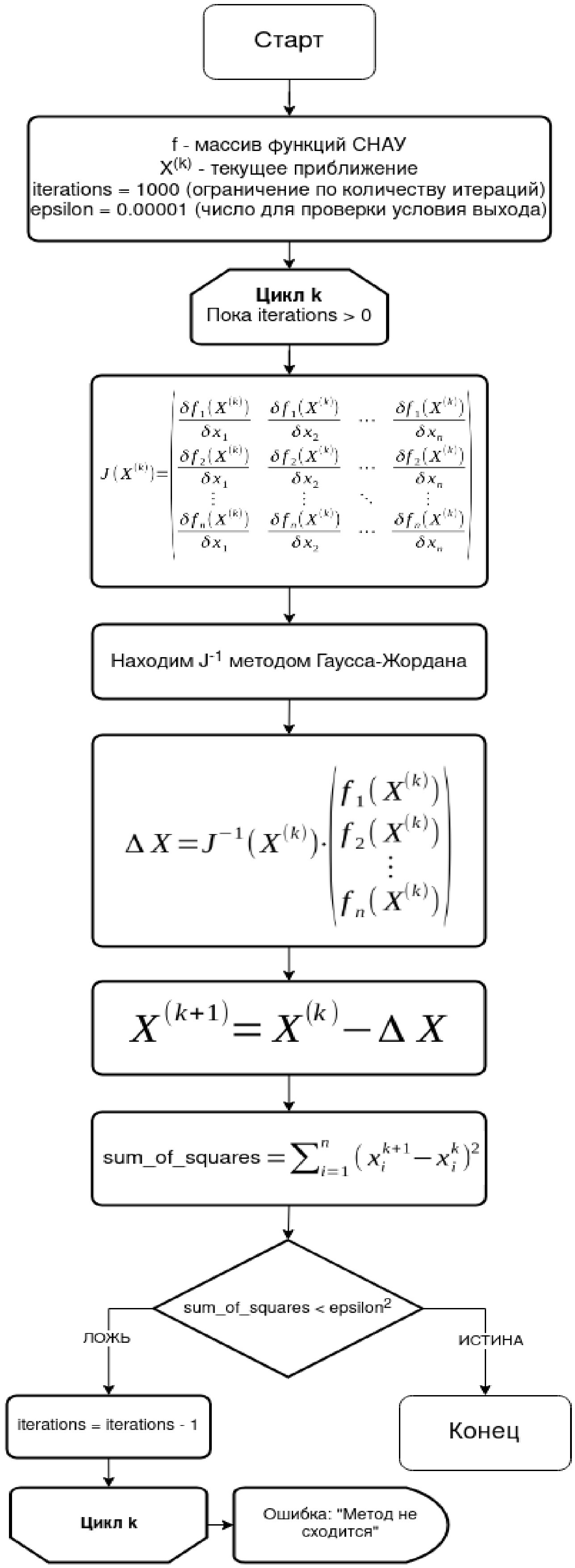
3) Находим J^{-1} обратную матрицу к матрице Якоби.

4) С помощью J^{-1} рассчитываем приращения аргументов $\Delta X = J^{-1}\left(X^{(k)}\right) \cdot F\left(X^{(k)}\right)$

5) С помощью рассчитанных приращений получаем новое приближение $X^{(k+1)} = X^{(k)} - \Delta X$

6) Повторяем пункты 2 – 5, пока не выполнится условие $\|F\left(x^{(k+1)}\right)\| < \epsilon$

Блок-схема



$$J(X^{(k)}) = \begin{vmatrix} \frac{\delta f_1(X^{(k)})}{\delta x_1} & \frac{\delta f_1(X^{(k)})}{\delta x_2} & \cdots & \frac{\delta f_1(X^{(k)})}{\delta x_n} \\ \frac{\delta f_2(X^{(k)})}{\delta x_1} & \frac{\delta f_2(X^{(k)})}{\delta x_2} & \cdots & \frac{\delta f_2(X^{(k)})}{\delta x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta f_n(X^{(k)})}{\delta x_1} & \frac{\delta f_n(X^{(k)})}{\delta x_2} & \cdots & \frac{\delta f_n(X^{(k)})}{\delta x_n} \end{vmatrix}$$

$$\Delta X = J^{-1}(X^{(k)}) \cdot \begin{vmatrix} f_1(X^{(k)}) \\ f_2(X^{(k)}) \\ \vdots \\ f_n(X^{(k)}) \end{vmatrix}$$

$$\text{sum\_of\_squares} = \sum_{i=1}^{n} (x_i^{k+1} - x_i^{k})^2$$

Исходный код

```
def solve_by_fixed_point_iterations(system_id, number_of_unknowns,
initial_approximations):
    functions = get_functions(system_id)

    def iteration(x):
        delta = multiply_matrix_and_vector(
            inverse_matrix(get_jacobian(x)),
            [fun(x) for fun in functions]
        )
        return [x[i] - delta[i] for i in range(number_of_unknowns)]

    def get_jacobian(x):
        h = 1e-5
        jacobian = []
        for i in range(number_of_unknowns):
            row = []
            for j in range(number_of_unknowns):
                row.append((functions[i]([x[k] + (h if k == j else 0) for k in
range(number_of_unknowns)]) - functions[
                    i](x)) / h)
            jacobian.append(row)
        return jacobian

    current_x = initial_approximations
    iterations = 1000
    epsilon = 1e-5
    while iterations > 0:
        new_x = iteration(current_x)
        sum_of_squares = sum((new_x[i] - current_x[i]) ** 2 for i in
range(number_of_unknowns))
        if math.sqrt(sum_of_squares) < epsilon:
            return new_x
        current_x = new_x
        iterations -= 1
    raise IOError("Method is not applicable")
```

```python
def multiply_matrix_and_vector(matrix, vector):
    return [sum(matrix[i][j] * vector[j] for j in range(number_of_unknowns)) for i
in range(number_of_unknowns)]


def inverse_matrix(matrix):
    n = len(matrix)
    identity_matrix = [[1 if i == j else 0 for j in range(n)] for i in range(n)]

    for i in range(n):
        if matrix[i][i] == 0:
            continue

        coefficient = matrix[i][i]
        for j in range(i, n):
            matrix[i][j] /= coefficient
        for j in range(n):
            identity_matrix[i][j] /= coefficient

        for k in range(i + 1, n):
            coefficient = matrix[k][i]
            for j in range(i, n):
                matrix[k][j] -= coefficient * matrix[i][j]
            for j in range(n):
                identity_matrix[k][j] -= coefficient * identity_matrix[i][j]

    for i in range(n - 1, 0, -1):
        for k in range(i - 1, -1, -1):
            coefficient = matrix[k][i]
            for j in range(n):
                identity_matrix[k][j] -= coefficient * identity_matrix[i][j]

    return identity_matrix
```

Примеры работы

Пример 1

Ввод	Вывод
1 2 0 0	0.0 0.0

Пример 2

Ввод	Вывод
3 3 2 3 3	-0.017935829439500884 -0.2496507452878699 0.23555734897888245

Пример 3

Ввод	Вывод
2 3 2 1 1	Method is not applicable

Пример 4

Ввод	Вывод
4 1 1	0.0

Пример 5

Ввод	Вывод
2 2 1 0.5	0.6583710532187106 -0.3845561836821646

Вывод

Метод Ньютона – итерационный метод для решения СНАУ. Метод показывает хорошую сходимость, если начальное приближение выбрано достаточно близко к корню. Но если начальное приближение выбрано достаточно далеко от корня, сходимость может быть медленной или вообще не достигнута. Метод Ньютона сходится быстрее метода простых итераций, но требует много вычислений (алгоритмическая сложность $O(n^3)$). Метод не применим в случаях, когда функция имеет разрывы.