

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет ПИиКТ

Отчёт по лабораторной работе на тему:

Решение нелинейных уравнений

Метод Ньютона

Выполнил
студент

Агнистова Алина Юрьевна

Группа № Р3225

Преподаватель: Перл Ольга Вячеславовна

г. Санкт-Петербург

2024

Описание метода и расчётные формулы

Метод Ньютона – метод решения нелинейных уравнений, заключающийся в выделении из уравнений системы линейных частей, что позволяет свести задачу к решению СЛАУ (системы линейных алгебраических уравнений).

Решение нелинейного уравнения методом Ньютона производится в несколько этапов:

1. Определение уравнений
2. Нахождение частных производных по формулам:

$$\frac{\partial F_n(x, y)}{\partial x} \text{ и } \frac{\partial F_n(x, y)}{\partial y}, n = 1, 2$$

3. Построение матрицы якобиан:

$$J = \begin{bmatrix} \frac{\partial F_1(x, y)}{\partial x} & \frac{\partial F_1(x, y)}{\partial y} \\ \frac{\partial F_2(x, y)}{\partial x} & \frac{\partial F_2(x, y)}{\partial y} \end{bmatrix}$$

4. Формирование СЛАУ, согласно формуле:

$$J\Delta x = -F(x, y), \text{ где } J - \text{матрица якобиан, } \Delta x - \text{вектор дельта, } F(x, y) - \text{вектор значений функций в точке } (x, y)$$

5. Подстановка значений начального приближения и решение СЛАУ:

$$\begin{bmatrix} \frac{\partial F_1(x, y)}{\partial x} & \frac{\partial F_1(x, y)}{\partial y} \\ \frac{\partial F_2(x, y)}{\partial x} & \frac{\partial F_2(x, y)}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x_0 \\ \Delta y_0 \end{bmatrix} = \begin{bmatrix} F_1(x_0, y_0) \\ F_2(x_0, y_0) \end{bmatrix}$$

6. Нахождение второго приближения:

$$\begin{aligned} x_1 &= x_0 + \Delta x_0 \\ y_1 &= y_0 + \Delta y_0 \end{aligned}$$

Далее итерация завершается, и мы используем второе приближение для всех предыдущих шагов. На каждой итерации увеличивается порядок приближения (на следующей итерации используем третье приближение и так далее). Вычисления останавливаются, если достигнута необходимая точность или превышено количество итераций. После остановки алгоритмы мы будем иметь решение системы нелинейных уравнений в виде x_n, y_n , где n – количество итераций.

На этапе решения СЛАУ могут быть выбраны различные методы, в своём коде я использовала метод Гаусса, его описание и расчётные формулы:

Метод Гаусса – метод решения СЛАУ (систем линейных алгебраических уравнений), заключающийся в последовательном исключении неизвестных из уравнения с помощью элементарных преобразований строк (приведение матрицы к треугольному виду).

Прямой ход метода Гаусса – поочерёдное преобразование уравнений системы для избавления от переменных неизвестных. На этом этапе матрица системы будет приведена к треугольному виду.

Обратный ход метода Гаусса – последовательное вычисление искомых неизвестных от последнего уравнения к первому.

Метод Гаусса применим только в случае совместности системы.

СЛАУ может иметь:

- единственное решение;
- бесконечно много решений;
- 0 решений.

СЛАУ считается совместной, когда она имеет единственное решение. По теореме Кронекера-Капелли СЛАУ совместна тогда и только тогда, когда ранг матрицы коэффициентов A системы равен рангу расширенной матрицы \bar{A} .

$$r(A) = r(\bar{A}) = n$$

Ранг матрицы – число ненулевых строк ступенчатой матрицы.

Прямой ход в методе реализуется согласно формулам:

$d_{jk}^k = \frac{a_{jk}^k}{a_{kk}^k}, j = k + 1, \dots, n$, где a_{jk}^k – j -й элемент матрицы a в строке k , a_{kk}^k – k -й элемент матрицы a в строке k

$$a_{jj}^{k+1} = a_{jj}^k - d_{jk}^k * a_{kj}^k, j = k + 1, \dots, n; j > k$$

$$b^{k+1}_j = b_j^k - d_{jk}^k * b_k^k, \text{ где } b_j^k \text{ и } b_k^k \text{ – свободные члены системы}$$

Формулы применяются последовательно и пошагово преобразуют матрицу системы к треугольному виду.

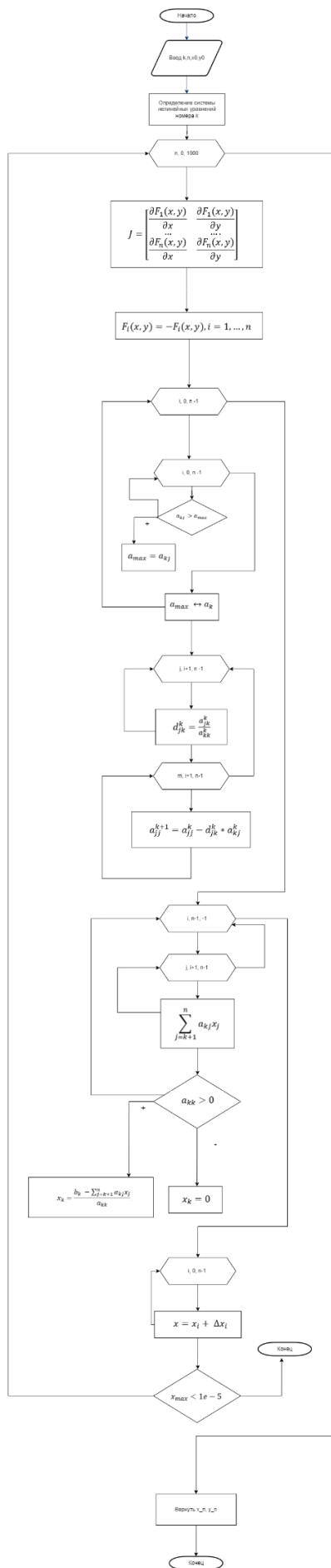
Обратный ход реализуется по формуле:

$$x_k = \frac{b_k - \sum_{j=k+1}^n a_{kj} x_j}{a_{kk}}, \text{ где } \sum_{j=k+1}^n a_{kj} x_j$$

– сумма произведений элементов строки матрицы
на уже найденные значения неизвестных

Таким образом на этом этапе будут вычислены все неизвестные.

Блок-схема реализованного метода



Код метода

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

    for _ in range(1000):
        d = solve_linear_system(calculate_jacobian(x, f_x,
number_of_unknowns), [-f(x) for f in f_x])
        x = [x[i] + d[i] for i in range(number_of_unknowns)]

        if max(abs(d_x) for d_x in d) < 1e-5:
            break

    return x

def calculate_jacobian(args, f, number_of_unknowns):
    h = 1e-5
    jacobian = [[(f[i]([args[j] + h if k == j else args[k] for k in
range(number_of_unknowns)])) -
        f[i](args)) / h for j in range(number_of_unknowns)] for i in
range(number_of_unknowns)]

    return jacobian

def solve_linear_system(coeff_matrix, const_vector):
    n = len(coeff_matrix)

    for i in range(n):
        max_row = max(range(i, n), key=lambda r: abs(coeff_matrix[r][i]))
        coeff_matrix[i], coeff_matrix[max_row] = coeff_matrix[max_row],
coeff_matrix[i]
        const_vector[i], const_vector[max_row] = const_vector[max_row],
const_vector[i]
        pivot = coeff_matrix[i][i]

        for j in range(i + 1, n):
            ratio = coeff_matrix[j][i] / pivot
            coeff_matrix[j][i] = 0

            for m in range(i + 1, n):
                coeff_matrix[j][m] -= ratio * coeff_matrix[i][m]
                const_vector[j] -= ratio * const_vector[i]

    solution = [0] * n
    for i in range(n - 1, -1, -1):
        solution[i] = const_vector[i]

        for j in range(i + 1, n):
            solution[i] -= coeff_matrix[i][j] * solution[j]
        if abs(coeff_matrix[i][i]) > 1e-10:
            solution[i] /= coeff_matrix[i][i]
        else:
            solution[i] = 0

    return solution
```

Примеры и результаты работы программы

Во всех примерах расчёт невязок не входит в сравнение результатов, так как в нашем случае невязки зависят от написанного программного кода.

Пример 1

Выбранный случай: Сложная система нелинейных уравнений (3 неизвестных)

Входные данные:

4
3
1
1
1

Выходные данные:

0,7852
0,49661
0,36992

```
4
3
1.0
1.0
1.0
0.7852
0.49661
0.36992

Process finished with exit code 0
```

Пример 2

Выбранный случай: Простая система нелинейных уравнений

Входные данные:

3
2
1
1

Выходные данные:

0,92814
0,33519

```
3
2
1
1
0.92814
0.33519

Process finished with exit code 0
```

Пример 3

Выбранный случай: Нулевые начальные приближения

Входные данные:

1
2
0
0

Выходные данные:

0,0
0,0

```
1
2
0
0
0.0
0.0

Process finished with exit code 0
```

Пример 4

Выбранный случай: Большие начальные приближения

Входные данные:

2
2
100
100
100

Выходные данные:

0,7852
0,49661
0,36992

Пример 5

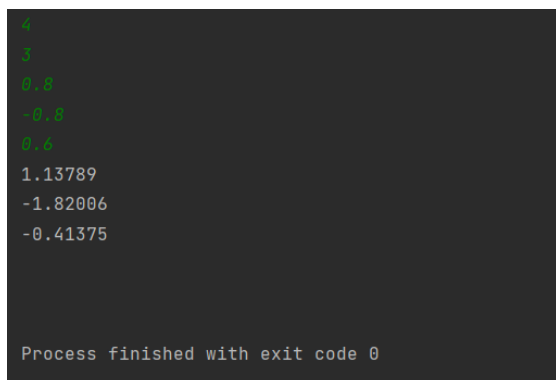
Выбранный случай: Сложная система уравнений с начальными приближениями разных знаков

Входные данные:

4
3
0,8
-0,8
0,6

Выходные данные:

1,13789
-1,82006
-0,41375



```
4
3
0.8
-0.8
0.6
1.13789
-1.82006
-0.41375

Process finished with exit code 0
```

Проанализировав результаты запуска реализованного метода, можно сделать вывод, что код справляется с основными крайними ситуациями, однако я не учла в коде вывод сообщений в случае, когда сформированная СЛАУ (система линейных алгебраических уравнений) не имеет решений.

Для оценки метода приведена таблица сравнения метода Ньютона с методом простых итераций:

	Метод Ньютона	Метод простых итераций
Основное преимущество	Быстрая сходимость при правильном начальном приближении.	Простота реализации.
Вычислительная сложность	Высокая (необходимо вычисление и функции, и производной на	Средняя (необходимо вычисление только функции на каждом шаге)

	каждом шаге (или Якобиана))	
Применимость	Эффективен для решения сложных систем.	Эффективен для решения систем, где трудно вычисляемые производные.
Численная стабильность	Высокая чувствительность к начальному приближению, метод может быть неустойчив в точках разрыва функции.	Неустойчивость метода при некоторых функциях, меньшая чувствительность к начальному приближению.

Таким образом, метод Ньютона будет оптимальным в случаях, когда важна точность и быстрая сходимость и допускается численная неустойчивость. В случаях, когда численная стабильность должна быть высокой лучше отдать предпочтение методу простых итераций.

Метод Ньютона – один из часто используемых методов для решения систем нелинейных уравнений. Метод сохраняет эффективность при решении сложных систем и обеспечивает быструю сходимость, однако обладает численной неустойчивостью к начальным приближениям, что может привести к неточному результату.

Временная алгоритмическая сложность зависит от выбора метода решения сформированной СЛАУ. В случае решения СЛАУ методом Гаусса временная алгоритмическая сложность составляет $O(n^3)$ из-за действий внутри метода, таких как: прямой и обратный ход метода Гаусса. При прямом ходе используется цикл, в который вложен вложенный цикл, внутри каждого цикла мы проходим по всем элементам матрицы размера n , что даёт нам алгоритмическую сложность $O(n^3)$. Нахождение матрицы Якобиана составляет $O(n^2)$. Таким образом, общая алгоритмическая сложность реализованного программного метода = $O(n^3)$.

Что касается численных ошибок численного метода, то основные ошибки вызваны высокой чувствительностью к начальным приближениям.

Вывод

В результате выполнения лабораторной работы была реализовано решение СНАУ методом Ньютона на языке программирования Python. Программа разделяется на несколько основных аспектов: нахождение частных производных и построение матрицы Якоби, формирование СЛАУ и итерационное нахождение приближений. Алгоритмическая сложность метода составила $O(n^3)$.

Метод Гаусса – метод решения СЛАУ, отличающийся высокой эффективностью и точностью для небольших и средних систем, однако возможна численная нестабильность из-за ошибок округления.