НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет ПИиКТ

Отчёт по лабораторной работе на тему:

Решение нелинейных уравнений

Метод Ньютона

Выполнил студент

Агнистова Алина Юрьевна

Группа № P3225

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:
3. Построение матрицы якобиан:
4. Формирование СЛАУ, согласно формуле:
5. Подстановка значений начального приближения и решение СЛАУ:
6. Нахождение второго приближения:

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 .

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 системы равен рангу расширенной матрицы .

Ранг матрицы – число ненулевых строк ступенчатой матрицы.

Прямой ход в методе реализуется согласно формулам:

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

Обратный ход реализуется по формуле:

Таким образом на этом этапе будут вычислены все неизвестные.

**Блок-схема реализованного метода**

Изображение выглядит как снимок экрана, текст, Шрифт, Графика

Автоматически созданное описание

**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

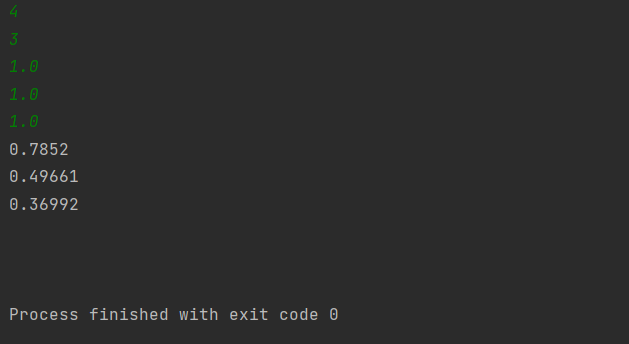
Выбранный случай: Сложная система нелинейных уравнений (3 неизвестных)

Входные данные:

|  |
| --- |
| 4 |
| 3 |
| 1 |
| 1 |
| 1 |

Выходные данные:

|  |
| --- |
| 0,7852 |
| 0,49661 |
| 0,36992 |



Пример 2

Выбранный случай: Простая система нелинейных уравнений

Входные данные:

|  |
| --- |
| 3 |
| 2 |
| 1 |
| 1 |

Выходные данные:

|  |
| --- |
| 0,92814 |
| 0,33519 |

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Пример 3

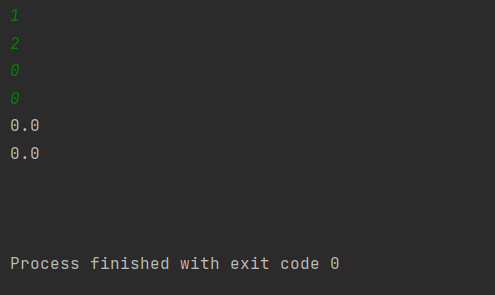
Выбранный случай: Нулевые начальные приближения

Входные данные:

|  |
| --- |
| 1 |
| 2 |
| 0 |
| 0 |

Выходные данные:

|  |
| --- |
| 0,0 |
| 0,0 |



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 |

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

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 каждом шаге (или Якобиана)) | Средняя (необходимо вычисление только функции на каждом шаге) |
| 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 нестабильность. 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 нестабильностью 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 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 элемента матрицы размера , что даёт нам алгоритмическую сложность . Нахождение матрицы Якобиана составляет . Таким образом, общая алгоритмическая сложность реализованного программного метода = .

Что касается численных ошибок численного метода, то основные ошибки вызваны высокой чувствительность 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 .

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