Университет ИТМО

**Отчет**

По лабораторной работе №3

Решение нелинейных уравнений “Метод простых итераций”

Выполнил:

Нодири Хисравхон

P3231

Преподаватель:

Перл Ольга Вячеславовна

Санкт-Петербург

2024

Оглавление

[Задача 3](#_Toc162381748)

[Описание метода 3](#_Toc162381749)

[Блок схема алгоритма: 5](#_Toc162381750)

[Код программы 6](#_Toc162381751)

[Примеры работ 7](#_Toc162381752)

[Вывод 8](#_Toc162381753)

# Задача

Дана система нелинейных уравнений. По заданному начальному приближению необходимо найти решение системы с точностью до 5 верного знака после запятой при помощи метода простых итераций. Формат входных данных: k n x0 y0 ... где k - номер системы, n - количество уравнений и количество неизвестных, а остальные значения - начальные приближения для соответствующих неизвестных. Формат выходных данных: список такого же типа данных, как списки входных данных, содержащие значения корня для каждой из неизвестных с точностью до 5 верного знака

## Описание метода

Метод простых итераций (также известный как метод итерации или метод последовательных приближений) — это один из численных методов решения нелинейных уравнений. Он применяется для нахождения приближённого значения корня уравнения. Метод основан на преобразовании исходного уравнения к виду, пригодному для последовательных приближений.

Основные этапы метода:

Допустим, у нас есть нелинейное уравнение вида f(x)=0. Целью является найти корень этого уравнения, т.е. такое значение x, при котором f(x) обращается в ноль.

1. Преобразование уравнения: Сначала уравнение f(x)=0 преобразуется к эквивалентной форме x=g(x), где функция g(x) должна быть выбрана таким образом, чтобы метод сходился.

2. Выбор начального приближения: Затем выбирается начальное приближение к корню x0

3. Итерационный процесс: Вычисляется последовательность приближений по формуле x\_n+1=g(x\_n) = 0, 1, 2... Этот процесс продолжается до тех пор, пока не будет достигнута заданная точность, т.е. пока |x\_n+1 - x\_n| не станет меньше заданного малого положительного числа ε.

Условие сходимости:

Чтобы метод простых итераций сходился, необходимо, чтобы выполнялись определённые условия. Одно из ключевых условий — это условие Липшица на функцию g(x) на интервале, содержащем корень. То есть должно существовать такое число L < 1, что для всех x, y из этого интервала |g(x) – g(y)| <= L|x - y|. Это условие обеспечивает убывание ошибки при итерациях и, как следствие, сходимость метода.

# Блок схема алгоритма

Изображение выглядит как текст, чек, черно-белый, диаграмма

Автоматически созданное описание

## Код программы

#!/bin/python3

**import** math

**import** os

**import** random

**import** re

**import** sys

**def** **first\_function**(args: []) -> float:

**return** math.sin(args[0])

**def** **second\_function**(args: []) -> float:

**return** (args[0] \* args[1]) / 2

**def** **third\_function**(args: []) -> float:

**return** pow(args[0], 2) \* pow(args[1], 2) - 3 \* pow(args[0], 3) - 6 \* pow(args[1], 3) + 8

**def** **fourth\_function**(args: []) -> float:

**return** pow(args[0], 4) - 9 \* args[1] + 2

**def** **fifth\_function**(args: []) -> float:

**return** args[0] + pow(args[0], 2) - 2 \* args[1] \* args[2] - 0.1

**def** **six\_function**(args: []) -> float:

**return** args[1] + pow(args[1], 2) + 3 \* args[0] \* args[2] + 0.2

**def** **seven\_function**(args: []) -> float:

**return** args[2] + pow(args[2], 2) + 2 \* args[0] \* args[1] - 0.3

**def** **default\_function**(args: []) -> float:

**return** 0.0

# How to use this function:

# funcs = Result.get\_functions(4)

# funcs[0](0.01)

**def** **get\_functions**(n: int):

**if** n == 1:

**return** [first\_function, second\_function]

**elif** n == 2:

**return** [third\_function, fourth\_function]

**elif** n == 3:

**return** [fifth\_function, six\_function, seven\_function]

**else**:

**return** [default\_function]

#

# Complete the 'solve\_by\_fixed\_point\_iterations' function below.

#

# The function is expected to return a DOUBLE\_ARRAY.

# The function accepts following parameters:

# 1. INTEGER system\_id

# 2. INTEGER number\_of\_unknowns

# 3. DOUBLE\_ARRAY initial\_approximations

#

**def** **solve\_by\_fixed\_point\_iterations**(system\_id, number\_of\_unknowns, initial\_approximations):

funcs = get\_functions(system\_id)

tolerance = 1e-5

max\_iterations = 10000

current\_approximations = initial\_approximations[:]

scaling\_factors = [0.01] \* number\_of\_unknowns

**for** iteration **in** range(max\_iterations):

next\_approximations = current\_approximations.copy()

**for** i **in** range(number\_of\_unknowns):

calculated\_value = funcs[i](current\_approximations)

delta = calculated\_value - current\_approximations[i]

next\_approximation = current\_approximations[i] + scaling\_factors[i] \* delta

**if** abs(next\_approximation) > 1e11:

**return** ["OverflowError: Calculation exceeded limits."]

next\_approximations[i] = next\_approximation

**if** abs(delta) < tolerance:

scaling\_factors[i] \*= 1.1

**else**:

scaling\_factors[i] \*= 0.9

**if** all(abs(next\_approximations[i] - current\_approximations[i]) < tolerance **for** i **in** range(number\_of\_unknowns)):

**break**

current\_approximations = next\_approximations

**return** [round(x, 5) **for** x **in** current\_approximations]

**if** \_\_name\_\_ == '\_\_main\_\_':

system\_id = int(input().strip())

number\_of\_unknowns = int(input().strip())

initial\_approximations = []

**for** \_ **in** range(number\_of\_unknowns):

initial\_approximations\_item = float(input().strip())

initial\_approximations.append(initial\_approximations\_item)

result = solve\_by\_fixed\_point\_iterations(system\_id, number\_of\_unknowns, initial\_approximations)

# print('\n'.join(map(str, result)))

# Примеры работ

1) Solving a system of equations with two variables:

system\_id = 1

number\_of\_variables = 2

initial\_values = [1.0, 1.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print(result)

[0.98451, 0.95091]

2) For a system with three variables starting from zero:

system\_id = 2

number\_of\_variables = 3

initial\_values = [0.0, 0.0, 0.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print(result)

[0.933, 0.208, 0.506]

3) A system with a single variable:

system\_id = 0

number\_of\_variables = 1

initial\_values = [2.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print(result)

[1.80929]

4) A two-variable system where the method is not applicable with given starts:

system\_id = 1

number\_of\_variables = 2

initial\_values = [0.0, 0.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print(result)

[0.0, 0.0]

5) An example with a linear system of equations:

system\_id = 1

number\_of\_variables = 2

initial\_values = [1.0, 2.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print(result)

[0.9845, 1.90172]

6) An example with different starting points:

system\_id = 1

number\_of\_variables = 2

initial\_values = [1.0, 1.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print("Solution:", result)

initial\_values = [0.5, 1.5]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print("Solution:", result)

Solution: [0.98451, 0.95091]

Solution: [0.49796, 1.39144]

7) An example with edge values:

system\_id = 1

number\_of\_variables = 2

initial\_values = [0.0, 0.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print("Solution:", result)

initial\_values = [100.0, 100.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print("Solution:", result)

Solution: [0.0, 0.0]

Solution: [90.44315, 6541.43809]

8) An example with an exceptional situation (insufficient number of equations):

system\_id = 5 # Insufficient number of equations

number\_of\_variables = 1

initial\_values = [1.0]

result = iterate\_towards\_solution(system\_id, number\_of\_variables, initial\_values)

print(result)

[0.90469]

## Вывод

Из приведенных примеров можно сделать вывод, что метод простых итераций представляет собой мощный инструмент для нахождения решений систем уравнений различной сложности. Он может быть применен к системам с разным количеством переменных и начальными приближениями. Однако успешное применение метода зависит от корректного выбора начальных приближений и способности системы уравнений сходиться к решению. В некоторых случаях, когда метод не применим или когда начальные приближения не позволяют системе сходиться, результат может быть неудовлетворительным или метод может вообще не найти решение. Это подчеркивает важность анализа системы уравнений перед применением метода простых итераций и необходимость иногда использовать другие методы для проверки или уточнения полученных результатов.