**Национальный исследовательский университет ИТМО**

**Факультет ПиИКТ**

**Изображение выглядит как логотип

Автоматически созданное описание**

**Лабораторная работа № 2**

по «Вычислительной математике»

Численное решение нелинейных уравнений и систем

Работу выполнил: Велюс Арина

Группа: Р32151

Преподаватель: Машина Екатерина Александровка

**Санкт-Петербург**

**2023 г.**

**Цель работы:**

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения, выполнить программную реализацию методов.

**Задание лабораторной работы:**

1. Отделить корни заданного нелинейного уравнения графически

( **−1,38𝑥 3 − 5,42𝑥 2 + 2,57𝑥 +10,95)**

1. Определить интервалы изоляции корней.
2. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью ε=10-2
3. Используемые методы для уточнения каждого из 3-х корней многочлена

Уточнить *крайний правый корень* нелинейного уравнения методом половинного деления (**метод простых итераций**)

Уточнить *крайний левый корень* нелинейного уравнения (**методом хорд**)

Уточнить *центральный корень* нелинейного уравнения (**метод половинного деления**)

1. Вычисления оформить в виде таблиц (1-5), в зависимости от заданного метода. Для всех значений в таблице удержать 3 знака после запятой.

**Программная реализация задачи:**

Для нелинейных уравнений:

1. Все численные методы (**метод хорд, метод Ньютона, метод простых итераций)** должны быть реализованы в виде отдельных подпрограмм/методов/классов.

2. Пользователь выбирает уравнение, корень/корни которого требуется вычислить (3-5 функций, в том числе и трансцендентные), из тех, которые предлагает программа.

3. Предусмотреть ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя.

4. Выполнить верификацию исходных данных. Необходимо анализировать наличие корня на введенном интервале. Если на интервале несколько корней или они отсутствуют – выдавать соответствующее сообщение. Программа должна реагировать на некорректные введенные данные.

5. Для методов, требующих начальное приближение к корню, выбор начального приближения (а или b) вычислять в программе.

6. Для метода простой итерации проверять достаточное условие сходимости метода на введенном интервале.

7. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.

8. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).

Для систем нелинейных уравнений:

1. Пользователь выбирает предлагаемые программой системы двух нелинейных уравнений (2-3 системы).

2. Организовать вывод графика функций.

3. Начальные приближения ввести с клавиатуры.

4. Для метода простой итерации проверить достаточное условие сходимости.

5. Организовать вывод вектора неизвестных: 𝑥1, 𝑥2.

6. Организовать вывод количества итераций, за которое было найдено решение.

7. Организовать вывод вектора погрешностей: |𝑥𝑖 (𝑘) − 𝑥𝑖 (𝑘−1) |

8. Проверить правильность решения системы нелинейных уравнений

**Описание метода, расчетные формулы:**

1. Крайний левый корень: Метод хорд.

* Находим интервал изоляции корня [ a0 , b0 ]
* Вычисляем x0:
* Вычисляем
* В качестве нового интервала выбираем ту половину отрезка, на концах которого функция имеет разные знаки: [ a0 , x0 ], либо [ b0 , x0 ]
* Вычисляем и т.д.

Рабочая формула метода:

:

1. Центральный корень: Метод половинного деления

Рабочая формула метода:

Критерий окончания интеграционного процесса:

или

1. Крайний правый корень: Метод простой итерация.

Рабочая формула метода:

Критерий окончания интеграционного процесса:

**Вычислительная реализация:**

Исследуемое уравнение: **−1,38𝑥 3 − 5,42𝑥 2 + 2,57𝑥 +10,95**

**Изображение выглядит как седзи, белый, грязный

Автоматически созданное описание**

Крайний левый корень: Метод хорд.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | a | b | x | f(a) | f(b) | f(x) | |x(k) - x(k - 1)| |
| 1 | -4.0000 | -3.5000 | -3.8495 | 2.2700 | -5.2725 | -0.5390 | — |
| 2 | -4.0000 | -3.8495 | -3.8784 | 2.2700 | -0.5390 | -0.0375 | 0.0289 |
| 3 | -4.0000 | -3.8784 | -3.8804 | 2.2700 | -0.0375 | -0.0021 | 0.0020 |

Центральный корень: Метод половинного деления.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | a | b | x | f(a) | f(b) | f(x) | |a - b| |
| 1 | -2.0000 | -1.0000 | -1.5000 | -4.8300 | 4.3400 | -0.4425 | 1 |
| 2 | -1.5000 | -1.0000 | -1.2500 | -0.4425 | 4.3400 | 1.9641 | 0.5 |
| 3 | -1.5000 | -1.2500 | -1.3750 | -0.4425 | 1.9641 | 0.7565 | 0.25 |
| 4 | -1.5000 | -1.3750 | -1.4375 | -0.4425 | 0.7565 | 0.1549 | 0.125 |
| 5 | -1.5000 | -1.4375 | -1.4688 | -0.4425 | 0.1549 | -0.1449 | 0.0625 |
| 6 | -1.4688 | -1.4375 | -1.4532 | -0.1449 | 0.1549 | 0.0044 | 0.0313 |
| 7 | -1.4688 | -1.4532 | -1.4610 | -0.1449 | 0.0044 | -0.0703 | 0.0156 |
| 8 | -1.4610 | -1.4532 | -1.4571 | -0.0703 | 0.0044 | -0.0329 | 0.0039 |

Крайний правый корень: Метод простой итерации.

f’(x) = -4.14x^2 - 10.84x + 2.57

f’(1) = -12.4100

f’(1.5) = -23.005 => λ = 0.0435

φ(x) = -0.0601x^3 - 0.2358x^2 + 1.1118x - 0.4763,

x0 = 1.500

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | x(k) | x(k + 1) | φ(x(k + 1)) | f(x(k + 1)) | |x(k + 1) - x(k)| |
| 1 | 1.5000 | 1.4109 | 1.4070 | -0.0891 | 0.0891 |
| 2 | 1.4109 | 1.4070 | 1.4069 | -0.0075 | 0.0039 |

**Листинг программы:**

def chord\_method(self):  
 *"""  
 Решает уравнение методом хорд.  
  
 Returns:  
 float: Результат решения уравнения.  
 """* left, right = self.\_enter\_root\_isolation()  
  
 epsilon = 10 \*\* (-8)  
 x = left - self.calculate(left) \* (right - left) / (self.calculate(right) - self.calculate(left))  
 while abs(self.calculate(x)) > epsilon:  
 x = left - self.calculate(left) \* (right - left) / (self.calculate(right) - self.calculate(left))  
 if self.calculate(x) > 0:  
 right = x  
 else:  
 left = x  
 self.iterations += 1  
  
 return x  
  
def simple\_iteration\_method(self):  
 *"""  
 Решает уравнение методом простых итераций.  
  
 Returns:  
 float: Результат решения уравнения.  
 """* left, right = self.\_enter\_root\_isolation()  
 halflife = -0.01 / max(abs(self.derivative\_func(left)), abs(self.derivative\_func(right)))  
 phi = lambda x: x + halflife \* self.calculate(x)  
 print(f"Используемая лямбда={halflife}")  
  
 epsilon = 10 \*\* (-8)  
 x = InputCorrect.get\_float\_input("Введите начальное приближение: ")  
 while not (min(left, right) < x < max(left, right)):  
 x = InputCorrect.get\_float\_input("Введите начальное приближение внутри интервала изоляции корня: ")  
 try:  
 self.calculate(x)  
 except Exception:  
 print("Невозможно посчитать значение функции в заданной точке.")  
 return None  
  
 while abs(self.calculate(x)) > epsilon and self.iterations <= 1000000:  
 x = phi(x)  
 self.iterations += 1  
 if self.iterations == 1000000:  
 return None  
 return x  
  
def newtons\_method(self):  
 *"""  
 Решает уравнение методом Ньютона.  
  
 Returns:  
 float: Результат решения уравнения.  
 """* left, right = self.\_enter\_root\_isolation()  
 x = InputCorrect.get\_float\_input("Введите начальное приближение: ")  
 epsilon = 10 \*\* (-8)  
 while abs(self.calculate(x)) > epsilon:  
 x = x - self.calculate(x) / self.derivative\_func(x)  
 self.iterations += 1  
 return x

**Примеры и результаты работы программы:**

**Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описаниеИзображение выглядит как линия, График, диаграмма

Автоматически созданное описание**

**Вывод:**

В ходе этой лабораторной работы я познакомился с несколькими методами, позволяющими решать нелинейные уравнения и системы нелинейных уравнений. Все методы довольно легко программируются и дают высокую точность и быструю сходимость при удачном выборе начального приближения.