A picture containing logo

Description automatically generated

Лабораторная работа №2

по Вычислительной Математики

Вариант 13

Выполнил:

Пурэвсурэн Билгуун

Группа Р3213

Преподователь:

Малышева Татьяна Алексеевна

г. Санкт-Петербург 2022г

1. **Цель работы:**

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

1. **Задание лабораторной работы:**

Задание:

1. № варианта определяется как номер в списке группы согласно ИСУ. .
2. Отделить корни заданного нелинейного уравнения графически (см. табл. 5)
3. Определить интервалы изоляции корней.
4. **Вычислительная реализация задачи (в отчет):**
5. Уточнить *крайний правый корень* нелинейного уравнения методом половинного деления (или методом хорд, см. вариант задания) с точностью ε=10-2. Вычисления оформить в виде таблицы, удержать 3 знака после запятой (см. табл. 1).

Таблица 1

Уточнение корня уравнения методом половинного деления (хорд)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага | a | b | x | f(a) | f(b) | f(x) | |a-b| |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3…. |  |  |  |  |  |  |  |

* 1. Уточнить *центральный корень* нелинейного уравнения методом простой итерации с точностью ε=10-2. Вычисления оформить в виде таблицы, удержать 3 знака после запятой (см. табл. 4)

Таблица 4

Уточнение корня уравнения методом простой итерации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации | *xk* | *f*(*xk* ) | *xk*+1 |  | │*xk* − *xk*+1│ |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3… |  |  |  |  |  |

1. **Программная реализация задачи:**

**Для нелинейных уравнений:**

* 1. Все численные методы (см. табл. 6) должны быть реализованы в виде отдельных подпрограмм или 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. Для метода половинного деления (метода хорд) анализировать наличие корня на введенном интервале. Для метода Ньютона (метода секущих) – выбор начального приближения (а или b). Для метода простой итерации – достаточное условие сходимости метода. Программа должна реагировать на некорректные введенные данные.
  5. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.
  6. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).

**Для систем нелинейных уравнений:**

* 1. Рассмотреть систему двух уравнений.
  2. Организовать вывод графика функций.
  3. Для метода простой итерации проверить достаточное условие сходимости.
  4. Вывод вектора неизвестных:
  5. Вывод количества итераций, за которое было найдено решение.
  6. Вывод вектора погрешностей:

**3. Ход работы.**

Уточнение корня уравнения методом половинного деления

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага | a | b | x | f(a) | f(b) | f(x) | |a-b| |
| 1 | -0.5 | 3 | 1.25 | 2.972 | -41.513 | -16.974 | 3.5 |
| 2 | -0.5 | 1.25 | 0.375 | 2.972 | -16.974 | -4.417 | 1.750 |
| 3 | -0.5 | 0.375 | -0.062 | 2.972 | -4.417 | 0.175 | 0.875 |
| 4 | -0.062 | 0.375 | 0.156 | 0.175 | -4.417 | -1.928 | 0.438 |
| 5 | -0.062 | 0.156 | 0.047 | 0.175 | -1.928 | -0.825 | 0.219 |
| 6 | -0.062 | 0.047 | -0.008 | 0.175 | -0.825 | -0.311 | 0.109 |
| 7 | -0.062 | -0.008 | -0.035 | 0.175 | -0.311 | -0.065 | 0.055 |
| 8 | -0.062 | -0.035 | -0.049 | 0.175 | -0.065 | 0.056 | 0.027 |
| 9 | -0.049 | -0.035 | -0.042 | 0.56 | -0.065 | -0.004 | 0.014 |
| 10 | -0.049 | -0.042 | -0.045 | 0.056 | -0.004 | 0.026 | 0.007 |

Уточнение корня уравнения методом простой итерации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации | *xk* | *f*(*xk* ) | *xk*+1 |  | │*xk* − *xk*+1│ |
| 1 | 0 | -0.0383 | -0.042 | -0.042 | 0.042 |
| 2 | -0.042 | -0.008 | -0.042 | -0.042 | 0.001 |

Методы в варианте:

1. **Метод половинного деления** - В данном методе интервал делится ровно пополам.

Такой подход обеспечивает гарантированную сходимость метода независимо от сложности функции - и это весьма важное свойство. Недостатком метода является то же самое - метод никогда не сойдется быстрее, т.е. сходимость метода всегда равна сходимости в наихудшем случае.

Метод половинного деления:

* 1. Один из простых способов поиска *корней функции одного аргумента*.
  2. Применяется для нахождения *значений действительно-значной функции*, определяемому по какому-либо критерию (это может быть сравнение на *минимум*, *максимум* или конкретное число

Рабочая формула метода:

Критерий окончания итерационного процесса:

Программная реализация:

def d(n,x,f,h=0.00000001):

""" Найти значение произодной функции """

if n <= 0:

return None

elif n == 1:

return (f(x+h) - f(x)) / h

return (d(n-1, x+h, f) - d(n-1, x, f)) / h

def halfDivision\_method(a,b,f,e):

'''Метод половинного деления'''

if f(a) \* f(b) > 0:

print("Уравнение содержит 0 или несколько корней 1")

return None

itr = 0

table = [['№','a','b','x','F(a)','F(b)','F(x)','|a-b|']]

while True:

x = (a + b) / 2

table.append([itr,a,b,x,f(a),f(b),f(x),abs(a-b)])

if f(a) \* f(x) > 0:

a = x

else:

b = x

itr += 1

if abs(a-b) <= e or abs(f(x)) < e:

break

x = (a+b) / 2

table.append([itr,a,b,x,f(a),f(b),f(x),abs(a-b)])

return x, f(x), itr , table

1. **Метод простой итерации** – как и из прошлой лабораторной работы этот метод находит корней функции очередным приближением.

Достоинства:

Простота

Недостатки:

Недостатком этого метода является его сходимость в малой окрестности корня и вытекающая отсюда необходимость выбора начального приближения к корню из этой малой окрестности. В противном случае итерационный процесс расходится или сходится к другому корню этого уравнения. Если |𝜑(𝑥)| ≈ 1, то сходимость может быть очень медленной.

Рабочая формула метода:

Критерий окончания итерационного процесса:

(при 0 < q 0.5)

(при 0.5 < q < 1)

Программная реализация:

def iteration\_method(x0, f, e, maxitr=100):

""" Метод простой итерации """

log = [['x0', 'f(x0)', 'x', 'g(x0)', '|x - x0|']]

def g(g\_x):

return g\_x + (-1 / d(1, g\_x, f)) \* f(g\_x)

x = g(x0)

log.append([x0, f(x0), x, g(x0), abs(x - x0)])

itr = 0

while abs(x - x0) > e and itr < maxitr:

if d(1, x, g) >= 1:

return None

x0, x = x, g(x)

log.append([x0, f(x0), x, g(x0), abs(x - x0)])

itr += 1

return x, f(x), itr, log

1. **Метод Ньютона в систем нелинейных уравнений –** итерационный процесс решения систем нелинейных уравнений методом Ньютона состоит в определении приращений ∆x1, ∆x2, …, ∆xn к значениям неизвестных на каждой итерации.

В методе Ньютона:

* 1. Важен удачный выбор начального приближения для обеспечения хорошей сходимости.
  2. Сходимость ухудшается с увеличением числа уравнений системы

Критерий окончания итерационного процесса: *max*|∆xi |

Программная реализация:

def calculate(self):

while 1 and self.itr < 250000:

self.x = self.x0 - determinant(

self.get\_a(1, self.x0, self.y0)) / determinant(

self.jacobian(self.x0, self.y0))

self.y = self.y0 - determinant(

self.get\_a(2, self.x0, self.y0)) / determinant(

self.jacobian(self.x0, self.y0))

self.itr += 1

if abs(self.x - self.x0) <= self.e and abs(

self.y - self.y0) <= self.e:

break

self.x0 = self.x

self.y0 = self.y

print('Закройте график чтобы посмотреть ответ!')

plot(self.f, self.g)

return [self.x, self.y, self.itr, self.e, self.f(self.x, self.y), self.g(self.x, self.y)]

def x\_derivative(self, type\_eq, x, y, h=0.00001):

if type\_eq == 1:

return (self.f(x + h, y) - self.f(x, y)) / h

elif type\_eq == 2:

return (self.g(x + h, y) - self.g(x, y)) / h

def y\_derivative(self, type\_eq, x, y, h=0.00001):

if type\_eq == 1:

return ((self.f(x, y + h) - self.f(x, y)) / h)

elif type\_eq == 2:

return ((self.g(x, y + h) - self.g(x, y)) / h)

def jacobian(self, x, y):

return [[self.x\_derivative(1, x, y),

self.y\_derivative(1, x, y)],

[self.x\_derivative(2, x, y),

self.y\_derivative(2, x, y)]]

def get\_a(self, mode, x, y):

if mode == 1:

ans = [[self.f(x, y), self.y\_derivative(1, x, y)],

[self.g(x, y),self.y\_derivative(2, x, y)]]

return ans

elif mode == 2:

return [[self.x\_derivative(1, x, y), self.f(x, y)],[self.x\_derivative(2, x, y),self.g(x, y)]]

def determinant(matrix):

return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

1. A picture containing chart

   Description automatically generated**Результаты выполнения программы.**

Text

Description automatically generated

1. **Вывод**

В результате выполнения данной лабораторной работой я познакомился с численными методами решения нелинейных уравнений и реализовал метод Ньютона в системе нелинейныа уравнений, метод половинного деления и метод простой итерации на языке программирования Python.