Лабораторная работа №3

Метод Ньютона и метод простых итераций

группа Б01-818
Слынко Денис

МФТИ, 2020

Содержание

1. Постановка задачи
2. Результаты решения и данные, характеризующие задачу
3. Код программы

Постановка задачи

Методом простой итерации найти ширину функции *f(x)* на полувысоте с точностью 10^{-3}, где $f(x) = \sqrt{x} \cdot e^{-x}, x \geq 0$.

Результаты решения и данные, характеризующие задачу

Далее левый и правый корни обозначают корни $x_{left} < x_{max}$ и $x_{right} > x_{max}$ соответственно, где $x_{max} = 0.5$ – точка максимум функции, а n_{left} и n_{right} – число итераций до схождения соответственно.

Для метода Ньютона:

$x_{left} = 0.05090301229490115$

$x_{right} = 1.8434903510626373$

$\Delta x = 1.792587338767736$

$n_{left} = 3$

$n_{right} = 3$

Для метода простых итераций:

$x_{left} = 0.05141814451652499$

$x_{right} = 1.8408970890810212$

$\Delta x = 1.7894789445644963$

$n_{left} = 34$

$n_{right} = 30$

Код программы

// Main.py

```
import math
import function as function


class Main:
    f = function.Function()
    e = 0.001

    def __init__(self, epsilon=0.001):
        e = epsilon

    def newtons_method(self, start_approximation, func):
        while start_approximation > 0\
              and ((self.f.function(0)) * (self.f.function(start_approximation))
>= 0):
            start_approximation -= 0.005

        n = 0
        while abs(func.function(start_approximation)) > self.e:
            n += 1
            start_approximation -
= (func.function(start_approximation)) / func.derivative(start_approximation)

        return start_approximation, n

    def fixed_point_iteration_method(self, start_approximation, func):
        while start_approximation > 0\
              and ((self.f.function(0)) * (self.f.function(start_approximation))
>= 0):
            start_approximation -= 0.005

        n = 0
        while abs(func.function(start_approximation)) > self.e:
            n += 1
            start_approximation += func.function(start_approximation)

        return start_approximation, n

    def fixed_point_iteration_method_reverse(self, start_approximation, func):
        while start_approximation > 0\
              and ((self.f.function(0)) * (self.f.function(start_approximation))
>= 0):
            start_approximation -= 0.005

        n = 0
        while abs(func.function(start_approximation)) > self.e:
            n += 1
            start_approximation -= func.function(start_approximation)
```

```python
        return start_approximation, n

    def find_fwhm(self, method_left, method_right):
        max_x, max_value = function.Function.get_max()

        x_l, n_l = method_left(max_x - 0.4, self.f)
        print(f"Left roote is {x_l}")
        x_r, n_r = method_right(max_x + 0.2, self.f)
        print(f"Right roote is {x_r}")

        return x_r - x_l, n_l, n_r

    def show_results(self):
        print("Solving with Newton's method")
        r_n, n_l_n, n_r_n = self.find_fwhm(self.newtons_method, self.newtons_meth
od)
        print("\nSolving with fixed-point iteration method")
        r_fpi, n_l_fpi, n_r_fpi = self.find_fwhm(self.fixed_point_iteration_metho
d_reverse, self.fixed_point_iteration_method)
        print()

        print(f"For the fixed-
point iteration method results are:\nx_r - x_l = {r_fpi}")
        print(f"Number of iterations for finding the left limit: {n_l_fpi}")
        print(f"Number of iterations for finding the right limit: {n_r_fpi}\n")

        print(f"For the Newton's method results are:\nx_r - x_l = {r_n}")
        print(f"Number of iterations for finding the left limit: {n_l_n}")
        print(f"Number of iterations for finding the right limit: {n_r_n}")


if __name__ == '__main__':
    main = Main()
    main.show_results()
```

// function.py

```python
import math


class Function:
    name = "sqrt(x) * exp(-x)"

    @staticmethod
    def get_max():
        return 0.5, math.sqrt(0.5) * math.exp(-0.5)

    @staticmethod
    def function(x):
```

```
        if x < 0:
            raise ValueError(f"{x} < 0")
        return math.sqrt(x) * math.exp(-x) - Function.get_max()[1] * 0.5

    @staticmethod
    def derivative(x):
        if x <= 0:
            raise ValueError(f"{x} < 0")
        return -1.0 * math.sqrt(x) * math.exp(-x) + 0.5 * math.exp(-
x) / math.sqrt(x)

    @staticmethod
    def second_derivative(x):
        if x <= 0:
            raise ValueError(f"{x} < 0")
        return -1.0 * math.exp(-x) / math.sqrt(x) + math.sqrt(x) * math.exp(-
x) - 0.25 * math.exp(-x) / math.sqrt(x) / x
```