**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

**ПРИБЛИЖЕННЫЕ МЕТОДЫ РЕШЕНИЯ**

**НЕЛИНЕЙНЫХ УРАВНЕНИЙ**

**(Вариант 10)**

*Выполнил студент 3 курса МОиАИС*

*Соколов Арсений*

**Задание:**исследовать функцию  и решить уравнение .

1. Нахождение промежутка, содержащего наименьший положительный корень уравнения f(x) = 0 для которого выполняются достаточные условия сходимости одного из итерационных методов.
2. Получение приближенного всеми указанными методами (с точностью 10-7).

***Метод Ньютона (касательных)***

;

***Метод хорд***

;

***Метод секущих***

;

***Конечноразностный метод Ньютона***

 — малый параметр;

***Метод Стеффенсена***

;

***Метод простых итераций***



Если , то .

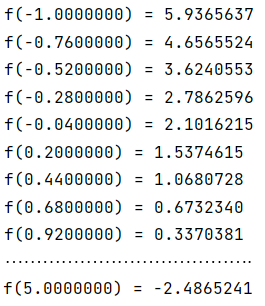
Для оценки погрешности приближенного решения, полученного любым методом, может использоваться неравенство

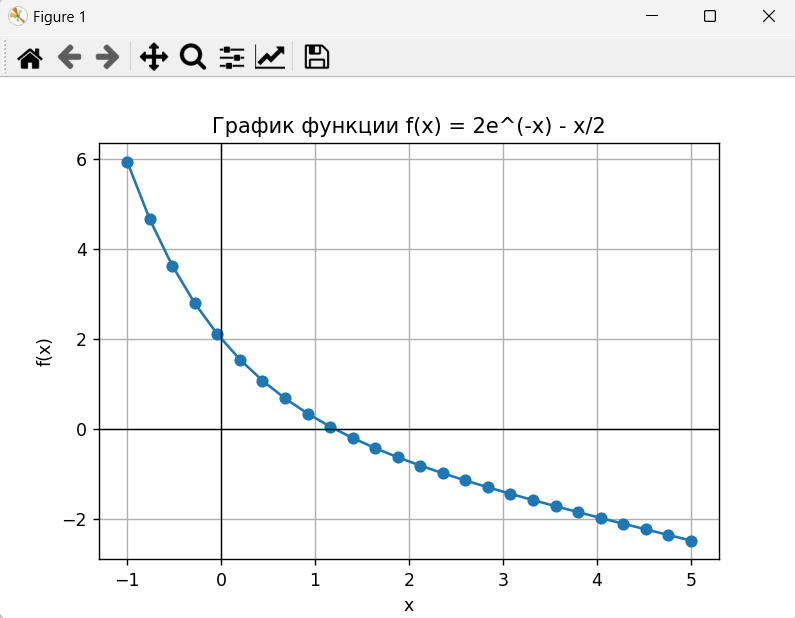
.

**Первый этап**

Имеем:

Полученные значения функции (см. программу в приложении 1):





Построив график функции, определяем, что уравнение имеет только один корень, который находится в интервале . Уточним значение корня с требуемой точностью 10-7, пользуясь предложенными методами.

**Второй этап**

***Метод Ньютона***

Определим знак второй производной функции *f(x)* на границах интервала [1, 2] (см. программу в приложении 2):

*f(1)= 0.2357589; f’’(1)= 0.7357589*

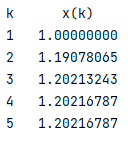
*f(2)= -0.7293294; f’’(2)= 0.2706706*

Видим, что знаки *f(1) и f’’(1)* совпадают, поэтому в качестве начального приближения можно взять левую границу интервала, т.е. .

Дальнейшие вычисления проводятся по формуле 

Итерации завершаются при выполнении условия 

Результаты вычислений (см. программу в приложении 3):

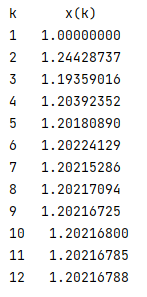


***Метод хорд***

Вычисления проводятся по формуле 

Итерации завершаются при выполнении условия 

Результаты вычислений (см. программу в приложении 4):

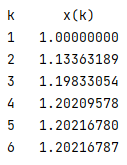


***Метод секущих***

Вычисления проводятся по формуле 

Итерации завершаются при выполнении условия 

Результаты вычислений (см. программу в приложении 5):



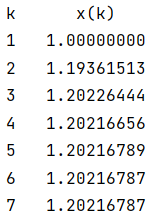
***Конечноразностный метод Ньютона***

Вычисления проводятся по формуле

Возьмем ;

Итерации завершаются при выполнении условия 

Результаты вычислений (см. программу в приложении 6):

******

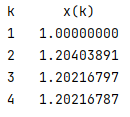
***Метод Стеффенсена***

Вычисления проводятся по формуле 

Возьмем

Итерации завершаются при выполнении условия 

Результаты вычислений (см. программу в приложении 7):



***Метод простых итераций***

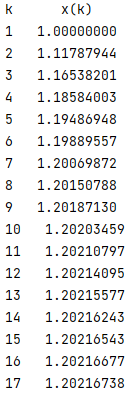
Вычисления проводятся по формуле .

= 1.2357589, = 0.7706706

2,59514246

Возьмем , удовлетворяющее условию ;

Результаты вычислений (см. программу в приложении 8):



**Итоговая таблица**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод решения | Выбранный интервал | Полученное решение | Количество итераций | Погрешность |
| 1. Метод Ньютона (метод касательных) |  | 1.20216787 | 4 | 10-8 |
| 2. Метод хорд |  | 1.20216788 | 12 | 10-8 |
| 3. Метод секущих |  | 1.20216787 | 6 | 10-8 |
| 4. Конечноразностный метод Ньютона |  | 1.20216787 | 6 | 10-8 |
| 5. Метод Стеффенсена |  | 1.20216787 | 4 | 10-8 |
| 6. Метод простых итераций |  | 1.20216738 | 17 | 10-8 |

**Вывод:** Метод Стеффенсена обладает одной из самых высоких скоростей [сходимости](https://en.wikipedia.org/wiki/Order_of_convergence) (квадратичная сходимость) и в то же время не требует использования [производных](https://en.wikipedia.org/wiki/Derivative), как для [метода Ньютона](https://en.wikipedia.org/wiki/Newton%27s_method). Однако ценой быстрой сходимости является выбор "адекватного" начального значения   . Если значение    недостаточно "близко" к фактическому решению, метод может дать сбой, и последовательность значений  ,,,,…  может либо беспорядочно переключаться между двумя крайними значениями, либо расходиться до бесконечности, либо и то, и другое.

Приближенным решением уравнения является

**ПРИЛОЖЕНИЕ 1**

***Программа вычисления значений функции и построения графика***

import numpy as np  
import matplotlib.pyplot as plt  
  
  
*# Функция*def func(x):  
 return 2 \* np.exp(-x) - x / 2  
  
  
*# Дано*y\_values = list()  
x\_values = list()  
a, b = -1, 5 *# границы отрезка*n = 25 *# количество точек*h = (b - a) / n *# шаг между точками на оси Х  
  
# Вычисление и вывод результатов*x = a - h  
for \_ in range(0, n + 1):  
 x += h  
 x\_values.append(x)  
 y\_values.append(func(x))  
 print(f"f({x\_values[-1]:.7f}) = {y\_values[-1]:.7f}")  
  
*# Построение графика*plt.plot(x\_values, y\_values, marker='o', label='f(x)')  
plt.axhline(0, color='black', linewidth=0.8)  
plt.axvline(0, color='black', linewidth=0.8)  
plt.title("График функции f(x) = 2e^(-x) - x/2")  
plt.xlabel("x")  
plt.ylabel("f(x)")  
plt.grid(True)  
plt.show()

**ПРИЛОЖЕНИЕ 2**

***Программа вычисления функции, первой и второй производных в точке***

import sympy as sp  
  
*# Функция*def func(x):  
 return 2 \* sp.exp(-x) - x / 2  
  
*# Определение переменной*x = sp.symbols('x')  
  
*# Определение функции*f = func(x)  
  
*# Первая и вторая производные*f\_prime = sp.diff(f, x)  
f\_double\_prime = sp.diff(f\_prime, x)  
  
*# Ввод пользователя*x\_value = float(input("Введите значение x: "))  
  
*# Вычисление производных в точке*first\_derivative = f\_prime.subs(x, x\_value)  
second\_derivative = f\_double\_prime.subs(x, x\_value)  
  
*# Вывод результатов*print(f"x = {x\_value}: значение функции = {func(x\_value):.7f}, первая производная = {first\_derivative:.7f}, вторая производная = {second\_derivative:.7f}")

**ПРИЛОЖЕНИЕ 3**

***Программа вычисления приближенных значений методом Ньютона***

import sympy as sp  
  
  
*# Функция*def func(x):  
 return 2 \* sp.exp(-x) - x / 2  
  
  
*# Дано*a, b = 1, 2 *# границы отрезка*k = 0  
X = [0] \* 6 *# список с приближенными значениями корня*X[k] = float(a) *# идем от левой границы  
  
# Вычисления*x = sp.symbols('x') *# определение переменной*f = func(x) *# определение функции*f\_prime = sp.diff(f, x) *# первая производная*print("k x(k)")  
while abs(X[k] - X[k-1]) > 0.0000001:  
 X[k+1] = X[k] - func(X[k]) / f\_prime.subs(x, X[k])  
 print(f"{k+1} {X[k]:.8f}")  
 k += 1  
print(f"{k+1} {X[k]:.8f}")

**ПРИЛОЖЕНИЕ 4**

***Программа вычисления приближенных значений методом хорд***

import math  
  
  
*# Функция*def func(x):  
 return 2 \* math.exp(-x) - x / 2  
  
  
*# Дано*a, b = 1, 2 *# границы отрезка*k = 0  
X = [0] \* 20 *# список с приближенными значениями корня*X[k] = a *# идем от левой границы  
  
# Вычисления*print("k x(k)")  
while abs(X[k] - X[k-1]) > 0.0000001:  
 X[k+1] = X[k] - (func(X[k]) / (func(b) - func(X[k]))) \* (b - X[k])  
 print(f"{k+1} {X[k]:.8f}")  
 k += 1  
  
print(f"{k+1} {X[k]:.8f}") *# выплевываем последнее (неподошедшее) значение, чтобы посмотреть на 8 знак после запятой*

**ПРИЛОЖЕНИЕ 5**

***Программа вычисления приближенных значений методом секущих***

import math  
  
  
*# Функция*def func(x):  
 return 2 \* math.exp(-x) - x / 2  
  
  
*# Дано*a, b = 1, 2 *# границы отрезка*k = 0  
X = [0] \* 20 *# список с приближенными значениями корня*X[k] = a *# идем от левой границы  
  
# Вычисления*print("k x(k)")  
while abs(X[k] - X[k-1]) > 0.0000001:  
 X[k+1] = X[k] - (func(X[k]) / (func(X[k]) - func(X[k-1]))) \* (X[k] - X[k-1])  
 print(f"{k+1} {X[k]:.8f}")  
 k += 1  
  
print(f"{k+1} {X[k]:.8f}") *# выплевываем последнее (неподошедшее) значение, чтобы посмотреть на 8 знак после запятой*

**ПРИЛОЖЕНИЕ 6**

***Программа вычисления приближенных значений конечноразностным методом Ньютона***

import math  
  
  
*# Функция*def func(x):  
 return 2 \* math.exp(-x) - x / 2  
  
  
*# Дано*a, b = 1, 2 *# границы отрезка*h = 0.05  
  
k = 0  
X = [0] \* 20 *# список с приближенными значениями корня*X[k] = a *# идем от левой границы  
  
# Вычисления*print("k x(k)")  
while abs(X[k] - X[k-1]) > 0.000000001:  
 X[k+1] = X[k] - h \* func(X[k]) / (func(X[k] + h) - func(X[k]))  
 print(f"{k+1} {X[k]:.8f}")  
 k += 1  
  
print(f"{k+1} {X[k]:.8f}") *# выплевываем последнее (неподошедшее) значение, чтобы посмотреть на 8 знак после запятой*

**ПРИЛОЖЕНИЕ 7**

***Программа вычисления приближенных значений методом Стеффенсена***

import math  
  
  
*# Функция*def func(x):  
 return 2 \* math.exp(-x) - x / 2  
  
  
*# Дано*a, b = 1, 2 *# границы отрезка*k = 0  
X = [0] \* 20 *# список с приближенными значениями корня*X[k] = a *# идем от левой границы  
  
# Вычисления*print("k x(k)")  
while abs(X[k] - X[k-1]) > 0.0000001:  
 X[k+1] = X[k] - func(X[k])\*\*2 / (func(X[k] + func(X[k])) - func(X[k]))  
 print(f"{k+1} {X[k]:.8f}")  
 k += 1  
  
print(f"{k+1} {X[k]:.8f}") *# выплевываем последнее (неподошедшее) значение, чтобы посмотреть на 8 знак после запятой*

**ПРИЛОЖЕНИЕ 8**

***Программа вычисления приближенных значений методом простых итераций***

import math  
  
  
*# Функция*def func(x):  
 return - (2 \* math.exp(-x) - x / 2)  
  
  
*# Дано*a, b = 1, 2 *# границы отрезка*t = 0.5  
  
k = 0  
X = [0] \* 100 *# список с приближенными значениями корня*X[k] = a *# идем от левой границы  
  
# Вычисления*print("k x(k)")  
while abs(X[k] - X[k-1]) >= 0.000001:  
 X[k+1] = X[k] - t \* func(X[k])  
 print(f"{k+1} {X[k]:.8f}")  
 k += 1  
  
print(f"{k+1} {X[k]:.8f}") *# выплевываем последнее (неподошедшее) значение, чтобы посмотреть на 8 знак после запятой*