# ЛАБОРАТОРНАЯ РАБОТА 2

**Решение нелинейных уравнений**

Вариант 20

Студент: Маркаров М.Г

Преподаватель: доц. Мамонтов

\

**Задача 2.1.** Найти с точностью  вещественные корни алгебраического уравнения  В программе предусмотреть подсчет количества итераций.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ**

1.Задать многочлен P(x). Используя функцию root из модуля numpy найти корни многочлена.

2. Используя средства библиотеки mathplotlib , локализовать корни уравнения *P(x)=0* графически.

3. Составить программу вычисления корня методом Ньютона, предусмотрев в ней подсчёт числа итераций. Найти

с заданной точностью  все корни уравнения. Сравнить полученные значения корней со значениями, найденными

в п.1.

Решение задачи 2.1:

P(x)= 

1.Воспользуемся функцией roots для полинома из модуля numpy из языка Python:

import numpy as np

coeff=[5.8,12.8,-6.4,-3]

print(np.roots(coeff))

Результат работы программы:

[-2.5591021 0.65893866 -0.30673311]

Имеем:

x1=-2.5591021

x2=-0.30673311

x3=0.65893866

2.Воспользуемся функцией plot для функции из модуля matplotlib из языка Python:

import numpy as np

import matplotlib.pyplot as plt

coeff=[5.8,12.8,-6.4,-3]

def f(x):

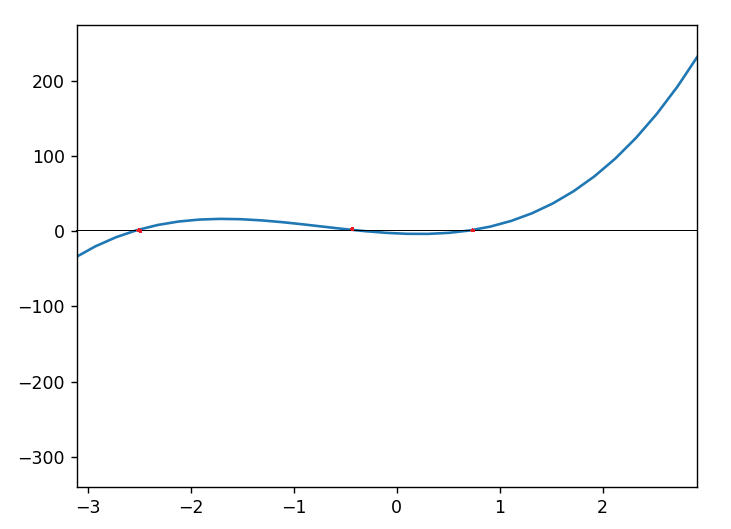
return coeff[0]\*x\*\*3+coeff[1]\*x\*\*2+coeff[2]\*x+coeff[3]

x = np.linspace(-10, 10, 100)

plt.plot(x,f(x))

plt.show()

Результат работы программы:



Очевидно, что функция имеет три корня.

Отрезок локализации х1 [-3,-2]

Отрезок локализации х2 [-1,0]

Отрезок локализации х3 [0.5,1]

3. Метод Ньютона

Применим язык Python:

import numpy as np

coeff=[5.8,12.8,-6.4,-3]

roots = np.roots(coeff) #root array

p = np.poly1d(coeff)

q = p.deriv() # // derrivative of polinomial

x1 =-3

while(abs(x1 - roots[0]) > 10\*\*-10):

x1 = x1 - p(x1) / q(x1)

print(x1)

x2 = -1

while(abs(x2 - roots[2]) > 10\*\*-10):

x2 = x2 - p(x2) / q(x2)

print(x2)

x3 = 0.5

while(abs(x3 - roots[1]) > 10\*\*-10):

x3 = x3 - p(x3) / q(x3)

print(x3)

print('%.16f %.16f %.16f' % (roots[1],x3, abs(roots[1] - x3))) # delta x3

print('%.16f %.16f %.16f' % (roots[2],x2, abs(roots[2] - x2)))

print('%.16f %.16f %.16f' % (roots[0],x1, abs(roots[0] - x1)))

Результат работы программы:

X1=-2.5591021043879295

X2=-0.30673310954947636

X3=0.658938662213267

Δx3=abs(0.6589386622132676-0.6589386622132670 )= 0.0000000000000006=6\*10^-16

Δx2 abs(-0.3067331095494759-(-0.3067331095494764) ) = 0.0000000000000005=5\*10^-16

Δx1=abs( -2.5591021043879295 -(-2.5591021043879296)) =0.0000000000000001=1\*10^16

**Задача 2.2.** Дано уравнение *f(x)=0.* Найти с точностью  все корни уравнений, содержащиеся на отрезке [с*, d*]. Для решения задачи использовать метод Ньютона и метод индивидуального варианта.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ.**

1.1 Найти корни уравнения аналитически.

1.2 Локализовать корни *f(x)=0* на указанном отрезке графически, используя средства библиотеки matplotlib.

2. Найти корни уравнения *f(x)=0* с точностью на указанном отрезке, используя программу нахождения корней методом Ньютона.

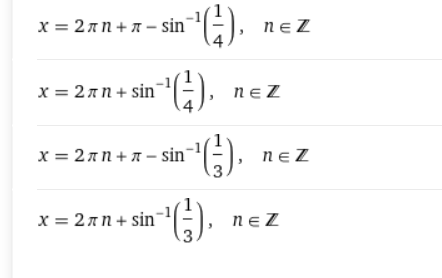
3. Составить программу вычисления корня методом(бисекции), указанным в индивидуальном варианте, предусмотрев в ней

подсчёт числа итераций. Найти с заданной точностью  все корни уравнения на отрезке [c,d].

4. Сравнить результаты проведенных расчётов п.2.2 и 2.3 и оформить результаты в виде таблицы.

f(x)= на отрезке [0,1]

* 1. f(x)=0



На отрезке:

x1= 0.2526803012717

x2= 0.3398368497238

1.2 Подключим пакет matplotlib языка Python:

import numpy as np

import matplotlib.pyplot as plt

def f(x):

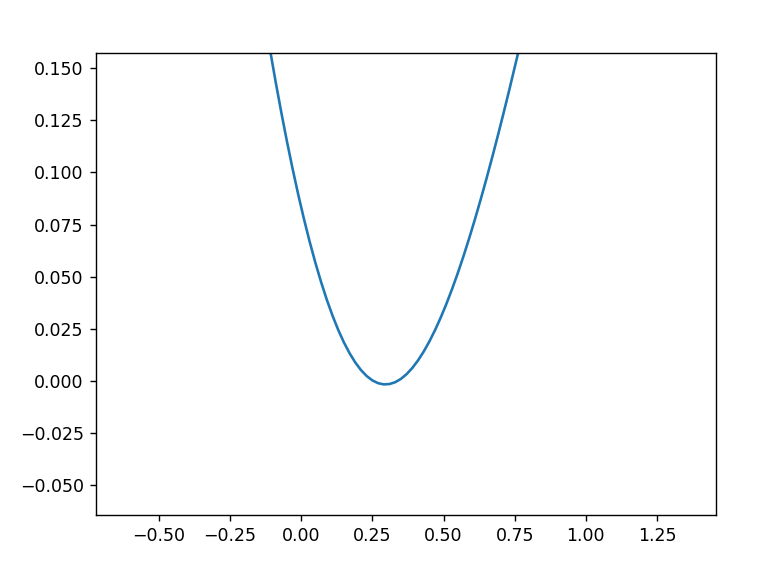
return np.sin(x)\*\*2 -7\*np.sin(x)/12 +1/12

x = np.linspace(-10, 10, 1000)

plt.plot(x,f(x))

plt.show()

Результат работы программы:



На отрезке от 0 до 1 функция имеет два корня.

3. Метод Ньютона

import numpy as np

def f(x):

return np.sin(x)\*\*2 -7\*np.sin(x)/12 +1/12

def derf(x):

return np.sin(2\*x)-7\*np.cos(x)/12

x1=0.27

while(abs(x1- 0.2526803012717 ) > 10\*\*-6):

x1 = x1 - f(x1) / derf(x1)

print(x1)

x2=0.35

while(abs(x2-0.3398368497238 ) > 10\*\*-6):

x2 = x2 - f(x2) / derf(x2)

print(x2)

Результат работы программы:

x1= 0.2526802551164367 и 4 итерации

x2=0.3398369104517468 и 3 итерации

4. Метод бисекции

Воспользуемся пакетом Python:

f = lambda x: np.sin(x)\*\*2 -7\*np.sin(x)/12 +1/12

c,d = 0.25,0.28

eps = 10\*\*-6

def half\_divide\_method(a, b, f):

x = (a + b) / 2

k=0

while np.abs(f(x)) >= eps:

x = (a + b) / 2

k=k+1

a, b = (a, x) if f(a) \* f(x) < 0 else (x, b)

return (a + b) / 2,k

print(half\_divide\_method(c,d,f))

c2,d2 = 0.3,0.35

print(half\_divide\_method(c2,d2,f))

Результат:

В формате (x,k) ,где k -число итераций

(0.25267333984375, 11)

(0.3394531249999999, 6)

5.Составим таблицу

Сравним работу методов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Значение корня, вычисленное с заданной точностью | Число итераций  метода Ньютона | Число итераций метода бисекции | Значение корня, найденное аналитически |
| 1 корень | 0.252673 | 4 | 11 | 0.252680 |
| 2 корень | 0.339453 | 3 | 6 | 0.339837 |

Аналитически вычисленное значение:

x1= 0.2526803012717

x2= 0.3398368497238

По методу Ньютона:

x1= 0.2526802551164367 Δx1=4.615526327800268e-08=4,62\*10^-8

x2=0.3398369104517468 Δx2=6,07\*10^-8

По методу бисекции:

х1=0.25267333984375 Δx1= 6.961427949980781e-06=6,96\*10^-6

х2=0.3394531249999999 Δx2=0.00038372472380010514=3,84\*10^-16