МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

КТС

Ракоть Валентин Викторович

Отчёт по лабораторной работе №1

(«Методы вычислений»)

студент 3 курса 12 группы

Преподаватель

Бондарь Иван Васильевич

Минск 2021

1.1.22

$f(x) = (1/16)^x - \log_{\frac{1}{16}}(x)$, метод секущих

Проанализируем функцию при помощи графического способа.

solve	$\left(\frac{1}{16}\right)^x - \frac{\log(x)}{\log\left(\frac{1}{16}\right)} = 0$	for	x

$\log(x)$ is the natural logarithm

Solution over the reals

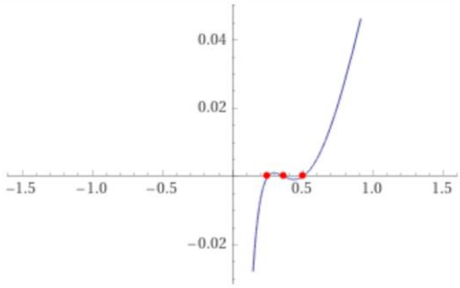
Exact forms More digits

$x \approx 0.25$

$x \approx 0.36425$

$x \approx 0.5$

Root plot



Функция определена на отрезке (0; +∞).

Так как первая производная от функции на концах отрезка (0.5; +∞) положительна, то график не убывает на этом промежутке, следовательно не имеет на нем корней.

Тогда для поиска корней при помощи методов бисекции и хорд, возьмем следующие отрезки:

(0.05; 0.3); (0.3; 0.4); (0,45; 0.6)

Для реализации данных методов нам понадобятся методы, которые возвращают значения функции и ее 2-ой производной.

```
def f1(x):                                        # функция
    return (1./16.)**x - math.log(x,1/16)

def f2(x):                                        # вторая производная
    return x*(x-1.)*(1./16.)**(x-2.) + (1./16.)**(x-1.) -((16.*x - math.log(1./16.))/(x*math.log(1./16.))**2)
```

Перейдем к реализации методов.

1) Метод бисекции.

```
def BisectionMethod(f, a, b, Eps, nevyazkaArr, needed_X):
    print("Корень функции, полученный с помощью метода бисекции на отрезке [", a, ";", b, "], с точностью ", Eps, ":")
    X = 0
    k = 0
    while abs(b - a) > 2.0*Eps:
        X = (a + b) / 2.0
        if f(X)*f(a) < 0:
            b = X
        else:
            a = X
        k += 1
        nevyazka = abs(needed_X - X)
        nevyazkaArr.append(nevyazka)
    print(X)
    return k
```

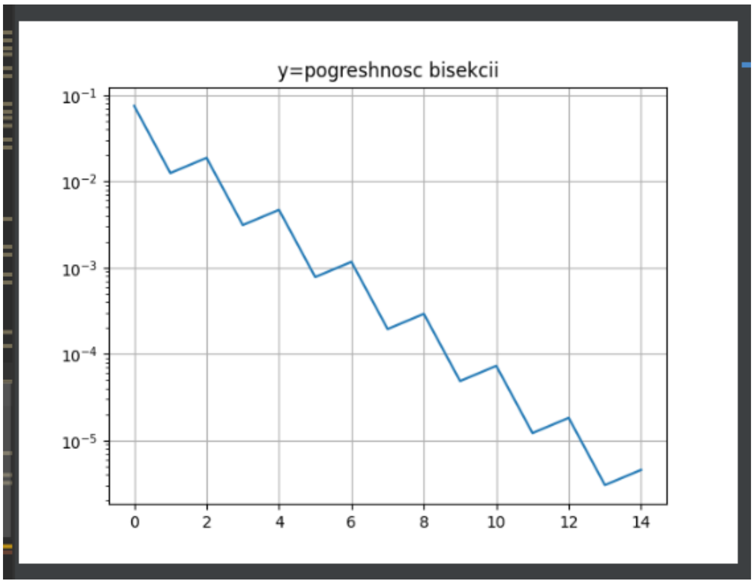
2) Метод хорд.

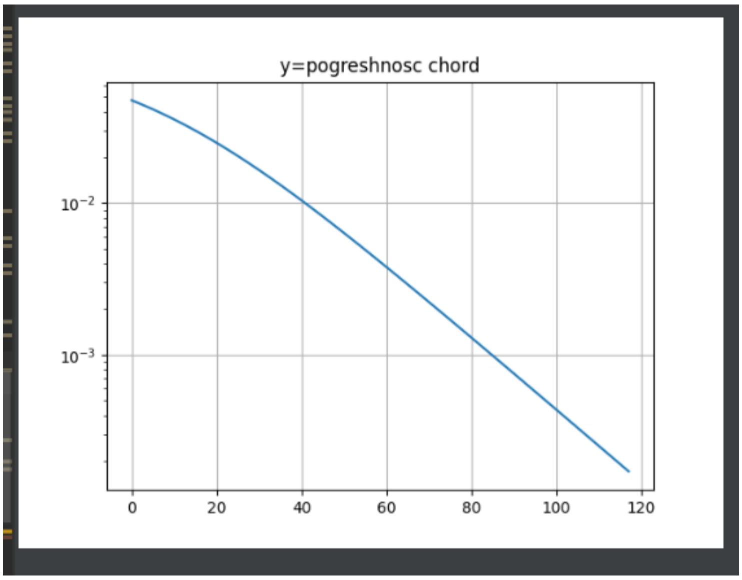
```
def ChordMethod(f, f2, a, b, Eps, nevyazkaArr, needed_X):
    if np.sign(f(a)) == np.sign(f2(a)):  # Если знак f(a) == f''(a), то берем X0 = a и X1 = b, иначе - берём X0 = b
        X0 = a
        X1 = b
    elif np.sign(f(b)) == np.sign(f2(b)):
        X0 = b
        X1 = a
    else:
        print("Невозможно выполнить метод хорд на данном отрезке.")
        return
    X = X1 - f(X1)*((X1 - X0)/(f(X1) - f(X0)))
    X_1 = 7.
    k = 0
    print("Корень функции, полученный с помощью метода хорд на отрезке [", a, ";", b, "], с точностью ", Eps, ":")
    while abs(f(X_1)) >= Eps:
        X_1 = X - f(X) * ((X - X0) / (f(X) - f(X0)))
        X = X_1
        k += 1
        nevyazka = abs(needed_X - X)
        nevyazkaArr.append(nevyazka)
    print(X)
    return k
```

Результаты:

1) на отрезке (0.05; 0.3)

```
Корень функции, полученный с помощью метода хорд на отрезке [ 0.05 ; 0.3 ], с точностью  1e-05 :
0.25017058755287047
Корень функции, полученный с помощью метода бисекции на отрезке [ 0.05 ; 0.3 ], с точностью  1e-05 :
0.25000457763671874
```

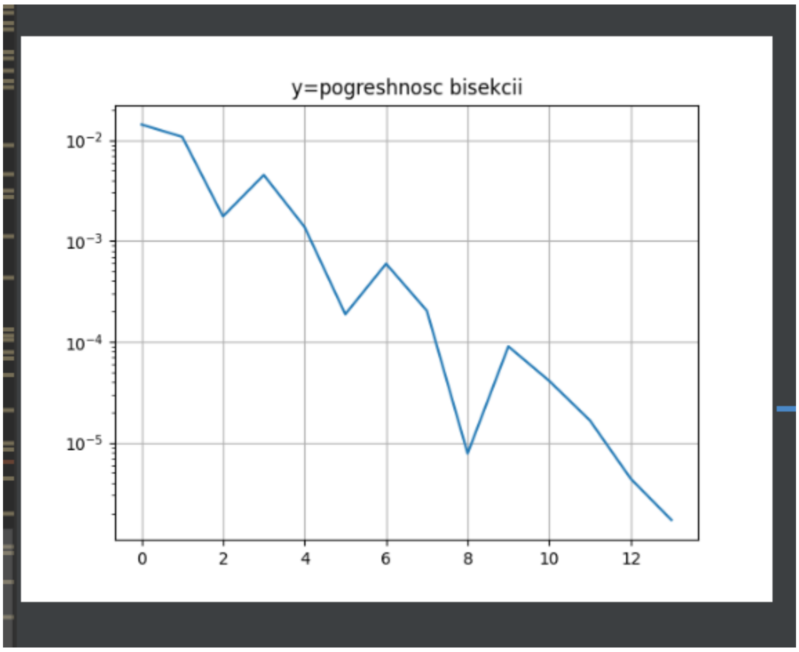




2) на отрезке (0.3; 0.4)

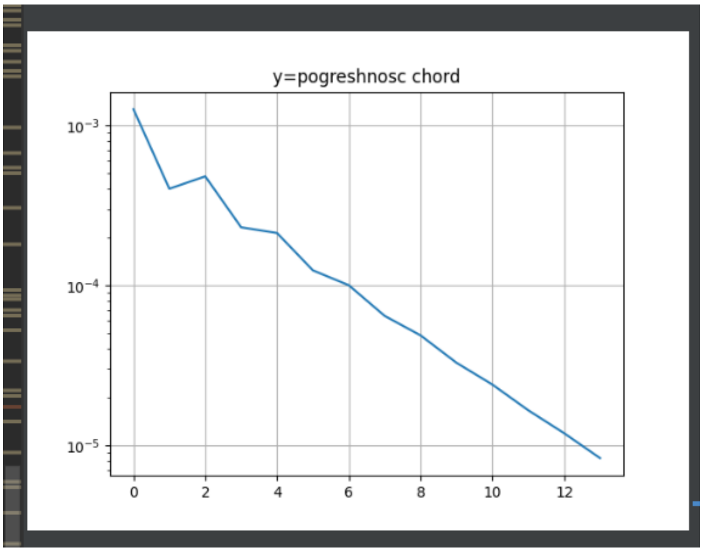
```
Корень функции, полученный с помощью метода хорд на отрезке [ 0.3 ; 0.4 ], с точностью  1e-05 :
0.3644395024526338
Корень функции, полученный с помощью метода бисекции на отрезке [ 0.3 ; 0.4 ], с точностью  1e-05 :
0.364251708984375
```

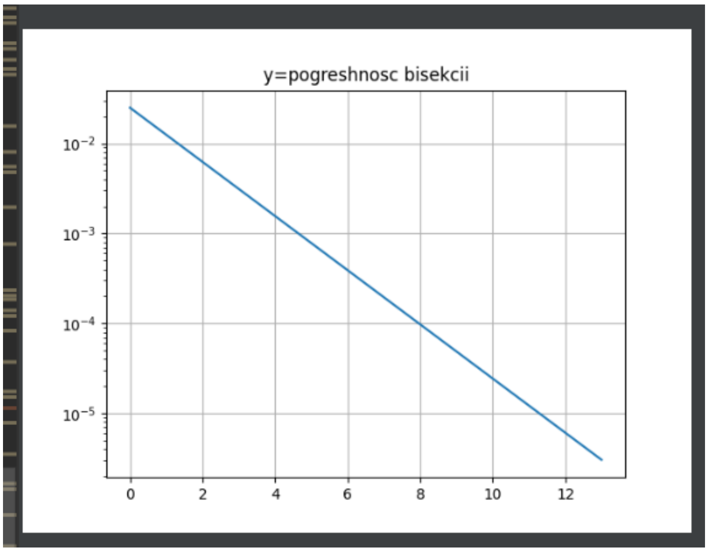
Так как метод хорд сходится за одну итерацию график создать не удалось.



3) на отрезке (0.45; 0.6)

```
Корень функции, полученный с помощью метода хорд на отрезке [ 0.45 ; 0.6 ], с точностью  1e-05 :
0.4997053095243997
Корень функции, полученный с помощью метода бисекции на отрезке [ 0.45 ; 0.6 ], с точностью  1e-05 :
0.4999969482421876
```

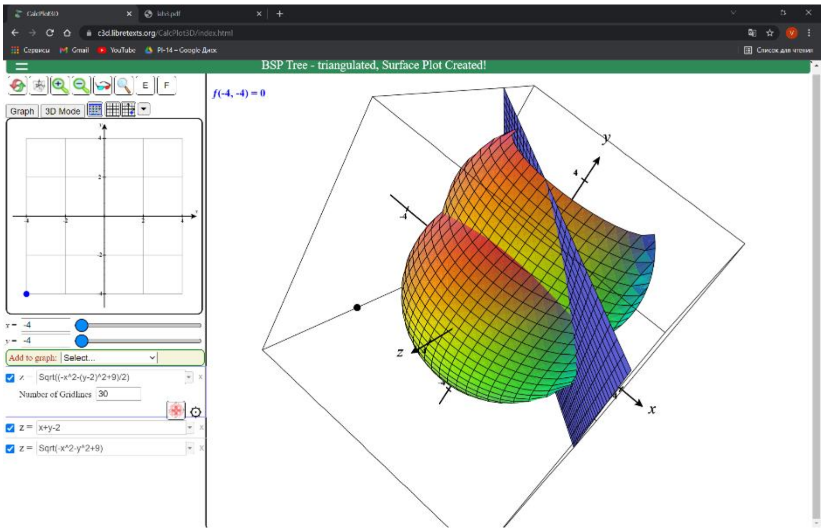




2.4

$$f(x_1, x_2) = \begin{pmatrix} x_1^2 + x_2^2 + x_3^2 - 9 \\ x_1 + x_2 - x_3 - 2 \\ x_1^2 + (x_2 - 2)^2 + 2x_3^2 - 9 \end{pmatrix} = 0.$$

Сначала нарисовал график и определил, что уравнение имеет один корень.

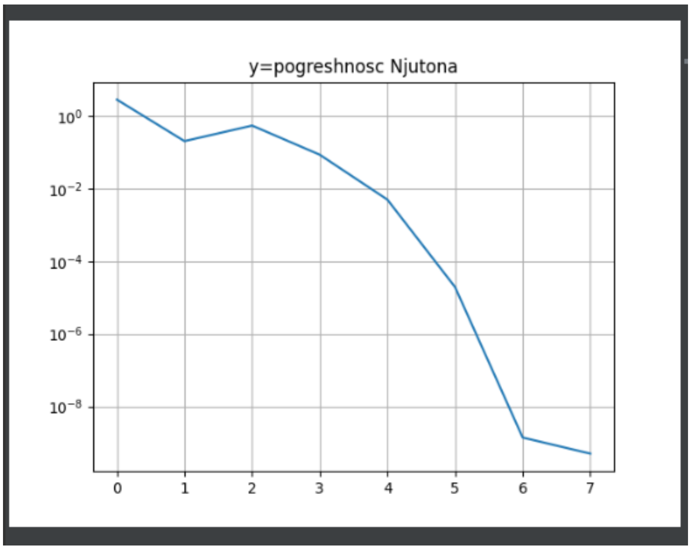


Реализация:

```
def NyutonMethod(X0, Eps, neviazkaArr, needed_X):
    x0 = X0
    k = 0
    while True:
        n1 = np.multiply(-1, f21(x0[0], x0[1], x0[2]))
        n2 = jacobi(x0[0], x0[1], x0[2])
        deltax = np.linalg.solve(n2, n1)
        x0 = np.add(x0,deltax)
        k += 1
        if k > 30 or evkl_norm(deltax) < Eps:
            break
    print("Корень системы по методу Ньютона, с точностью", Eps, ": \n", x0)
```

Ответ:

```
Корень системы по методу Ньютона, с точностью 1e-10 :
 [1.95718299 1.62897158 1.58615457]
```



Исходный код:

```
import numpy as np

import matplotlib.pyplot as plt

import math



def f1(x):                          # функция

    return (1./16.)**x - math.log(x,1/16)


def f2(x):                          # вторая производная

    return x*(x-1.)*(1./16.)**(x-2.) + (1./16.)**(x-1.) -((16.*x -
math.log(1./16.))/(x*math.log(1./16.))**2)


def BisectionMethod(f, a, b, Eps, nevyazkaArr, needed_X):

    print("Корень функции, полученный с помощью метода бисекции на отрезке [", a, ";", b,
"], с точностью ", Eps, ":")

    X = 0

    k = 0

    while abs(b - a) > Eps:

        X = (a + b) / 2.0

        if f(X)*f(a) < 0:

            b = X

        elif f(X)*f(a) > 0:

            a = X

        else:

            nevyazkaArr.append(0)

            break

        k += 1

        nevyazka = abs(needed_X - X)

        nevyazkaArr.append(nevyazka)

    print(X)

    return k
```

```
def ChordMethod(f, f2, a, b, Eps, nevyazkaArr, needed_X):
    if np.sign(f(a)) == np.sign(f2(a)):  # Если знак f(a) == f''(a), то берем X0 = a и X1 = b, иначе - берём X0 = b и X1 = a
        X0 = a
        X1 = b
    elif np.sign(f(b)) == np.sign(f2(b)):
        X0 = b
        X1 = a
    else:
        print("Невозможно выполнить метод хорд на данном отрезке.")
        return
    X = X1 - f(X1)*((X1 - X0)/(f(X1) - f(X0)))
    X_1 = 7.
    k = 0
    print("Корень функции, полученный с помощью метода хорд на отрезке [", a, ";", b, "], с точностью " , Eps , ":")
    while abs(f(X_1)) >= Eps:
        X_1 = X - f(X) * ((X - X0) / (f(X) - f(X0)))
        X = X_1
        k += 1
        nevyazka = abs(f(X_1))
        nevyazkaArr.append(nevyazka)
    print(X)
    return k



def f21(x,y,z):
```

```
    return np.array([x**2 + y**2 + z**2 - 9, x + y - z - 2, x**2 + (y - 2)**2 + 2*z**2 - 9])


def jacobi(x, y, z):
    n = np.array([[2.*x, 2.*y, 2.*z], [1., 1., -1.], [2.*x , 2*y-4., 4. *z]], dtype=float)
    return n

def evkl_norm(n):
    sum = 0
    for e in n:
        sum += e**2
    return sum**0.5



def NyutonMethod(X0, Eps, neviazkaArr, needed_X):
    x0 = X0
    k = 0
    while True:
        n1 = np.multiply(-1, f21(x0[0], x0[1], x0[2]))
        n2 = jacobi(x0[0], x0[1], x0[2])
        deltax = np.linalg.solve(n2, n1)
        x0 = np.add(x0,deltax)
        k += 1
        if k > 30 or evkl_norm(deltax) < Eps:
            break
        s = abs(np.max(np.add(needed_X,np.multiply(-1,x0))))
        neviazkaArr.append(s)
    print("Корень системы по методу Ньютона, с точностью", Eps, ": \n", x0)
    return k
```

```
def one():
    nevyazkaArr1 = []
    nevyazkaArr2 = []

    k = ChordMethod(f1,f2,0.05,0.3,0.00001,nevyazkaArr1,0.25)
    k2 =  BisectionMethod(f1,0.05,0.3,0.00001,nevyazkaArr2,0.25)



    t = np.arange(0, k, 1)
    plt.semilogy(nevyazkaArr1)
    plt.title('y=pogreshnosc chord')
    plt.grid(True)
    plt.show()

    t = np.arange(0, k2, 1)
    plt.semilogy(nevyazkaArr2)
    plt.title('y=pogreshnosc bisekcii')
    plt.grid(True)
    plt.show()

    nevyazkaArr1 = []
    nevyazkaArr2 = []

    k = ChordMethod(f1, f2, 0.3, 0.4, 0.00001, nevyazkaArr1, 0.36425)
    k2 = BisectionMethod(f1, 0.3, 0.4, 0.00001, nevyazkaArr2, 0.36425)

    t = np.arange(0, k, 1)
    plt.semilogy(nevyazkaArr1)
    plt.title('y=pogreshnosc chord')
    plt.grid(True)
```

```
    plt.show()

    t = np.arange(0, k2, 1)
    plt.semilogy(nevyazkaArr2)
    plt.title('y=pogreshnosc bisekcii')
    plt.grid(True)
    plt.show()

    nevyazkaArr1 = []
    nevyazkaArr2 = []

    k = ChordMethod(f1, f2, 0.45, 0.6, 0.00001, nevyazkaArr1, 0.5)
    k2 = BisectionMethod(f1, 0.45, 0.6, 0.00001, nevyazkaArr2, 0.5)

    t = np.arange(0, k, 1)
    plt.semilogy(nevyazkaArr1)
    plt.title('y=pogreshnosc chord')
    plt.grid(True)
    plt.show()

    t = np.arange(0, k2, 1)
    plt.semilogy(nevyazkaArr2)
    plt.title('y=pogreshnosc bisekcii')
    plt.grid(True)
    plt.show()


def two():
    nevyazkaArr = []
    x = np.array([10.,10.,10.])
    needed_x = np.array([1.95718299,1.62897158,1.58615457])
    k = NyutonMethod(x, 10**(-10), nevyazkaArr, needed_x)
```

```
    t = np.arange(0, k, 1)
    plt.semilogy(nevyazkaArr)
    plt.title('y=pogreshnosc Njutona')
    plt.grid(True)
    plt.show()


two()
```