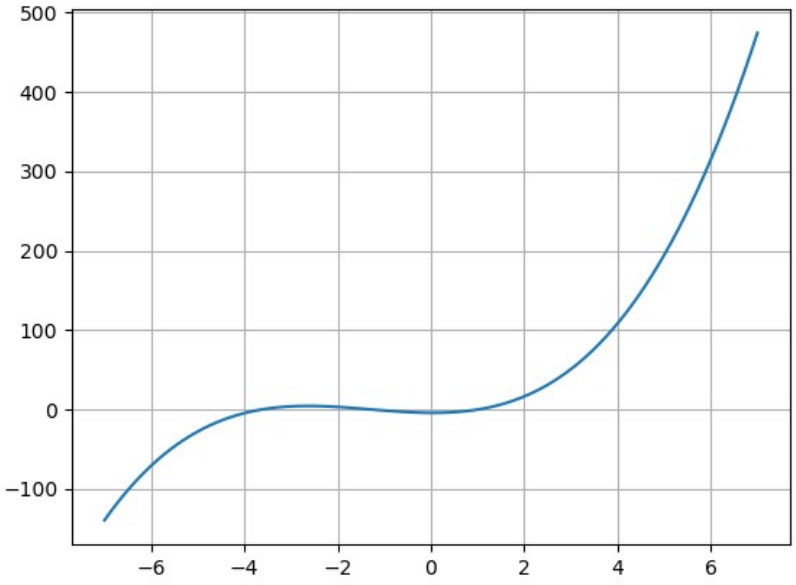
Чудов Богдан А-17-22, Вариант 16.
Лабораторная работа №2 «Решение нелинейных уравнений»

Задача 1. Локализуйте максимальный вещественный корень уравнения f(x) = 0 и найдите его с точностью ε , используя средства языка Python.
f(x) = $0.9x^3+3.5x^2-0.3x-4$
ε = 1e-6



По графику видно, что отрезок локализации максимального вещественного корня [0, 2]. Следовательно логично взять начачльное приближение x0 = 1.
Ниже приведен код для поиска корня методом секущих и методом ньютона при помощи ***scipy.optimize.newton***.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import newton

# заданная функция
def func(x):
   return 0.9 * x**3 + 3.5 * x**2 - 0.3 * x - 4

# производная
def derive(x):
   return 2.7 * x**2 + 7 * x - 0.3

x = np.linspace(-7, 7, 100)
y = func(x)

# построение графика
plt.plot(x,y)
plt.grid(True)

# решение двумя методами
x0 = 1
root1 = newton(func, x0, tol = 1e-6)
root2 = newton(func, x0, tol = 1e-6, fprime=derive)
print('Корень полученный методом секущих: ', root1)
print('Корень полученный методом ньютона: ', root2)

plt.show()
```

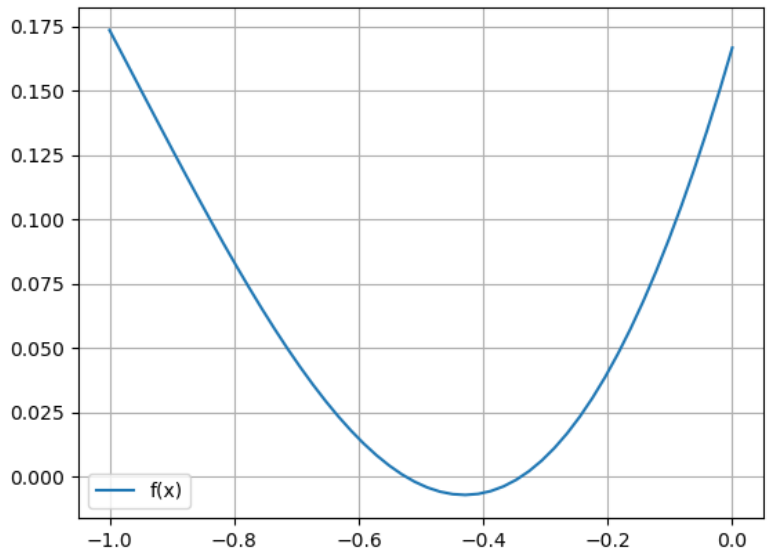
Корень полученный методом секущих: **0.989286**1090268065
Корень полученный методом ньютона: **0.989286**1089982815

Задача 2. Даны два уравнения f(x)=0 и g(x)=0. Найдите с точностью ε = 10^-10 все их корни, содержащиеся на отрезке [a, b]. Для решения задачи реализуйте программно метод бисекции.

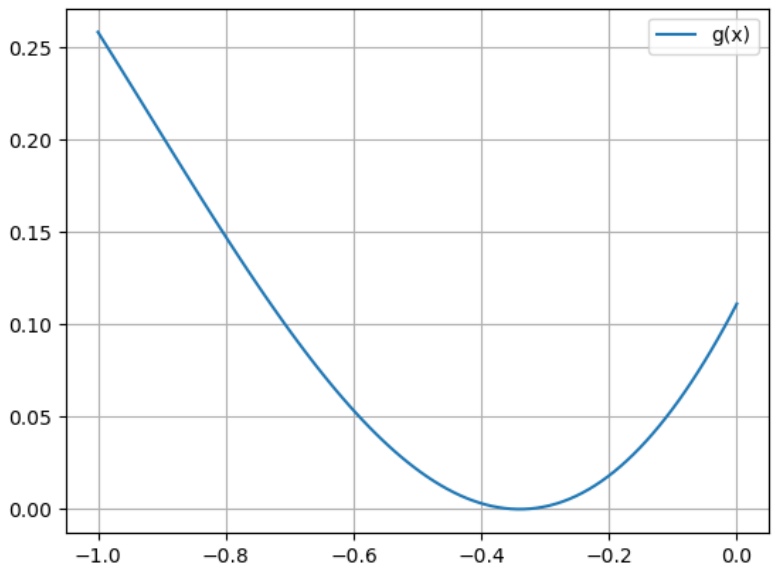
$f(x) = (sinx)^2 + \frac{5}{6} sinx + \frac{1}{6}$

$g(x) = (sinx)^2 + \frac{2}{3} sinx + \frac{1}{9}$

1. Аналитическое решение для **f(x) = 0:**
x1 = arcsin(-1/3) = **-0.3398369094**
x2 = arcsin(-1/2) = **-0.5235987756**
Были получены два отрезка локализации: [0.6, -0.5] и [-0.4, -0.3]
Ниже представлен график функции f(x).



Корни, полученные в ходе решения методом бисекции:
x1 = **-0.523598775**5935639
x2 = **-0.339836909**4356894
Корни, полученные в ходе решения при помощи ***scipy.optimize.root***:
x1 = **-0.523598775**5982989
x2 = **-0.339836909**45412176
2. Аналитическое решение для **g(x) = 0:**
x = arcsin(-1/3) = **-0.3398369094**
Был получен отрезок локализации [-0.4, -0.3]
Ниже представлен график функции g(x).



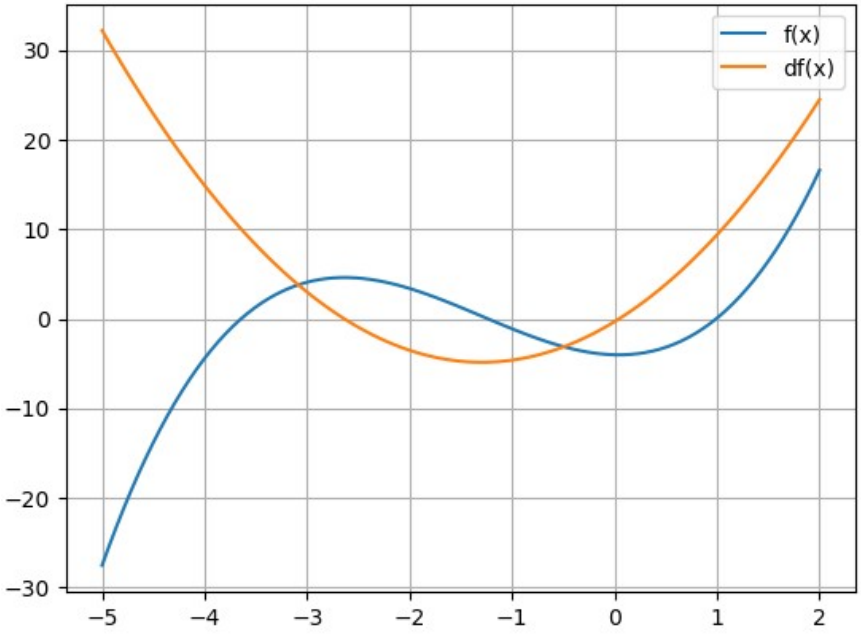
Корни, полученные в ходе решения методом бисекции:
x = **-0.300000000**1862645
Корни, полученные в ходе решения при помощи ***scipy.optimize.root***:
x = **-0.339836905**7031181

Вывод по задаче 2: Решение методом бисекции применимо для функции f(x), т.к функция имеет разные знаки на концах отрезка локализации, поэтому корни сошлись с аналитическим решением. Но в случае с функцией g(x) этот метод не применим, т.к во всех точках отрезка локализации функция положительна, либо равна нулю (непосредственно в корне), поэтому корни в результате расчета не совпали с аналитическим решением.

Ниже представлен код для задачи 2:

```
from scipy.optimize import root
from numpy import sin, arcsin, linspace
import matplotlib.pyplot as plt

def f(x):
   return sin(x)**2 + (5 * sin(x)) / 6 + 1 / 6

def g(x):
   return sin(x)**2 + (2 * sin(x)) / 3 + 1 / 9

def bisection(f, a, b, eps):
   c = 0
   while (b - a) / 2 > eps:
       c = (a + b) / 2
       if f(c) == 0:
           break
       elif f(c) * f(a) < 0:
           b = c
       else:
           a = c
   return c

# f(x)
print('\n------ Решение f(x) = 0 ------\n')
print('Аналитическое решение.\nx1 = arcsin(-1/3) = ', arcsin(-1/3), '\nx2 = arcsin(-1/2) = ', arcsin(-1/2))
print('\nДва отрезка локализации. [-0.6, -0.5] и [-0.4, -0.3]')

a1 = -0.6
b1 = -0.5
a2 = -0.4
b2 = -0.3
eps = 1e-10

print('Решение методом бисекции.\nx1 = ', bisection(f, a1, b2, eps), '\nx2 = ', bisection(f, a2, b2, eps))
print('\nРешение при помощи scipy.optimize.root')
print('x1 = ', root(f, -0.5, tol=eps).x[0])
print('x2 = ', root(f, -0.3, tol=eps).x[0])
print('\n')

# g(x)
print('------ Решение g(x) = 0 ------\n')
print('Аналитическое решение.\nx = arcsin(-1/3) = ', arcsin(-1/3))
print('\nОдин отрезок локализации. [-0.4, -0.3]')
print('Решение методом бисекции. \nx = ', bisection(g, -0.4, -0.3, eps))
print('\nРешение при помощи scipy.optimize.root')
print('x = ', root(g, -0.3, tol=eps).x[0])
x = linspace(-1, 0, 50)
plt.plot(x, f(x), label='f(x)')
plt.grid(True)
plt.legend()
plt.show()

x = linspace(-1, 0, 100)
plt.plot(x, g(x), label='g(x)')
plt.grid(True)
plt.legend()
plt.show()
```

Задача 3. Методом простой итерации найти все вещественные корни уравнения из задачи 1 точностью ε = 1e-13. Проследить за поведением погрешности.
f(x) = $0.9x^3+3.5x^2-0.3x-4$
eps = 1e-13



Ниже представлены графики функции f(x) и её первой производной.
На графике видны три корня f(x) = 0. Можно выделить три отрезка локализации: Отрезки локализации: [-4, -3], [-2, -1], [0.5, 1.5].
1) На отрезке локализации [-4, -3] производная монотонно убывает
m = $f'(-3)=3$
M = $f'(-4)=14.9$
2) На отрезке локализации [-2, -1] производная сначала убывает потом возрастает
m = $f'\left(-\frac{35}{27}\right)=-4.837$ (точка минимума)
M = $f'(-1)=-4.6$
3) На отрезке локализации [0.5, 1.5] производная монотонно возрастает
m = $f'(0.5)=3.875$
M = $f'(1.5)=16.275$

Формула для вычисления оптимального параметра: $q=\frac{2}{m+M}$

$q1=0.11173184357541899$
$q2=-0.2119309262166405$
$q3=0.09925558312655088$

Ниже представлена программа для нахождения корней методом простой итерации с оптимальным параметром:

```
import matplotlib.pyplot as plt
import numpy as np

def f(x):
   return 0.9 * x**3 + 3.5 * x**2 - 0.3 * x - 4

def df(x):
   return 2.7 * x**2 + 7 * x - 0.3

def mpi(f, df, a, b, mi, ma, tol):
   q = 2 / (mi + ma)
   print('q = ', q)
   i = 0
   xk = (a + b) / 2
   while True:
       x_prev = xk
       i += 1
       xk = x_prev - q * f(x_prev)
       print('Итерация: ', i, ' x = ', xk, ' Апостериорная оценка порешности: ', (q / (1 - q)) * abs(xk -
x_prev))
       if abs(xk - x_prev) < ((1 - abs(q)) / abs(q) * tol):
           return i, xk
       if i > 10000:
           print('Решение не найдено')
           exit()


# построение графиков
x = np.linspace(-5, 2, 100)
plt.plot(x, f(x), label='f(x)')
plt.plot(x, df(x), label='df(x)')
plt.grid(True)
plt.legend()
plt.show()
print('Отрезки локализации: [-4, -3], [-2, -1], [0.5, 1.5]')

print('На отрезке [-4, -3] производная убывает; на отрезке [-2, -1] убывает, потом возрастает; на отрезке
[0.5, 1.5]
возрастает.')

eps = 1e-13
a = -4
b = -3
x1 = mpi(f, df, a, b, df(b), df(a), eps)
a = -2
b = -1
# в точке -35/27 минимум производной
x2 = mpi(f, df, a, b, df(-35/27), df(b), eps)
a = 0.5
b = 1.5
x3 = mpi(f, df, a, b, df(a), df(b), eps)

print('x1 = ', x1[1], ' Итераций: ', x1[0])
print('x2 = ', x2[1], ' Итераций: ', x2[0])
print('x3 = ', x3[1], ' Итераций: ', x3[0])
```

В ходе выполнения программы были получены следующие данные:
1) q = 0.11173184357541899

Номер итерации	**Приближение $x^{(n)}$**	**Апостериорная оценка погрешности**
1	-3.649441340782123	0.018797652928568918
2	-3.6455295605631255	0.000492047826288982
3	-3.6460273009822153	6.260885774714304e-05
4	-3.645965172854633	7.814858815359267e-06
5	-3.6459729468913817	9.778662576810796e-07
6	-3.645971974432676	1.2232184975616445e-07
7	-3.645972096082781	1.5301899956775827e-08
8	-3.645972080864986	1.9141880533476618e-09
9	-3.6459720827686537	2.3945508050199453e-10
10	-3.645972082530514	2.995468303869524e-11
11	-3.645972082560305	3.747275026965e-12
12	-3.645972082556578	4.687794527121667e-13
13	-3.645972082557045	5.876501242292652e-14

2) q = -0.2119309262166405

Номер итерации	**Приближение $x^{(n)}$**	**Апостериорная оценка погрешности**
1	-1.2271389324960753	-0.04771534211532362
2	-1.2323172423135622	-0.0009055334525398083
3	-1.232200311956273	-2.044766610627435e-05
4	-1.2322029746012246	-4.6561796434018853e-07
5	-1.232202913980747	-1.06007311737001e-08
6	-1.2322029153609	-2.4134797787232716e-10
7	-1.2322029153294782	-5.4947367899206276e-12
8	-1.2322029153301934	-1.2506834945929532e-13
9	-1.2322029153301772	-2.8345201833370253e-15

3) q = 0.09925558312655088

Номер итерации	**Приближение $x^{(n)}$**	**Апостериорная оценка погрешности**
1	0.9900744416873448	0.0010937254339013975
2	0.9893489155931	7.994777898014443e-05
3	0.9892911407243735	6.366376719172461e-06
4	0.9892865122911383	5.100201912101364e-07
5	0.9892861413233481	4.087799340530132e-08
6	0.9892861115892346	3.2764863366070656e-09
7	0.9892861092059543	2.626204243359051e-10
8	0.9892861090149271	2.104982854689297e-11
9	0.9892861089996157	1.68720868314192e-12
10	0.9892861089983884	1.3523311640998877e-13
11	0.9892861089982901	1.0839202201850016e-14

3.1. Для достижения точности 1e-13 потребовалось: 13 итераций для первого корня; 9 итераций для второго корня; 11 итераций для третьего корня.
3.2-3. Количество итераций разное для каждого из корней, что объясняется разной скоростью убывания погрешности. Вычислим эту скорость для каждого корня.

$$\frac{x1^n}{x1^{n-1}} = 0.125$$

$$\frac{x2^n}{x2^{n-1}} = 0.0228$$

$$\frac{x3^n}{x3^{n-1}} = 0.08$$

3.4. Различную скорость сходимости при поиске различных корней можно объяснить большой либо меньшей разностью между максимумом и минимумом производной на отрезке локализации. В нашем случае для второго корня эта разность заметно меньше, чем для первого и третьего корня, поэтому скорость сходимости выше и требуется меньше итераций для вычисления корня с заданной точностью.