## Национальный исследовательский университет «МЭИ» (Московский Энергетический Институт)

**Кафедра математического и компьютерного моделирования**

## Численные методы

## Лабораторная работа №2:

## «Решение нелинейных уравнений»

*Выполнил:* Солонин Е. В. А-14-19

*Преподаватель:* Амосова О. А.

Вариант 52

2021 Москва

**Постановка задачи:**

**Задача 2.1.** Методом простой итерации найти вещественные корни алгебраического уравнения  с точностью .

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ.

1. Задать функцию  и построить ее график . По графику определить отрезки локализации для каждого корня .
2. Задать производную от многочлена  и построить ее график. Проверить, что на отрезках локализации производная функции сохраняет постоянный знак. Если условие не выполнено, то следует уменьшить длину отрезка локализации корня.
3. Для каждого корня определить итерационный параметр  и параметр , используя формулы:

 , ,  , ,

здесь  - отрезок локализации корня. Минимумы и максимумы можно найти приближенно, используя график, построенный в п.2.

1. Составить программу для нахождения корня с заданной точностью по методу простой итерации. В качестве расчетной формулы использовать метод простой итерации с параметром:

.

1. Используя программу, найти все корни многочлена с указанной точностью .
2. Результаты свести в таблицу:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ФИО **Солонин Егор Витальевич** Группа **А-14-19** | | | | | | Номер варианта  **52** |
| Уравнение: | | | | | |  |
| Корни: |  |  |  |  |  | Корень с заданной точностью  Число итераций |
| 1-ый корень | (-1.0, -0.75) | 25.80 | 2.89 | 0.07 | 0.80 | -0.89479384 Итераций: 5 |
| 2-ой корень | (-0.5, -0.25) | -8.54 | -11.57 | -0.10 | 0.15 | -0.46653499 Итераций: 6 |
| 3-ый корень | (0.75, 1.0) | 24.2 | 15.08 | 0.05 | 0.23 | 0.93485048 Итераций: 7 |

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy  
import math  
import matplotlib.pyplot as plt  
  
# класс Корень с полями(характеристиками):

# a, b – границы отрезка локализации

# max, min – максимум и минимум производной на отрезке [a, b]

# alpha – итерационный параметр

#

class Root:  
 def \_\_init\_\_(self, a, b):  
 self.a = a  
 self.b = b  
 x\_data\_ = numpy.linspace(a, b, 100)  
 self.max = max(df(x\_data\_))  
 self.min = min(df(x\_data\_))  
 self.alpha = 2 / (self.max + self.min)  
 self.q = numpy.abs((self.max - self.min) / (self.max + self.min))  
 self.epsilon1 = (1 - self.q) / self.q \* epsilon  
  
#

def f(x):  
 return x\*\*5 - 5.1 \* x\*\*4 + 9.6 \* x\*\*3 + 9.8 \* x\*\*2 - 8.8 \* x - 5

#   
  
def df(x):  
 return 5 \* x\*\*4 - 5.1 \* 4 \* x\*\*3 + 9.6 \* 3 \* x\*\*2 + 9.8 \* 2 \* x - 8.8  
  
# функция для нахождения корня по методу простой итерации

def find\_root(x):  
 x\_curr = x.a  
 x\_next = x\_curr - x.alpha \* f(x\_curr)  
 i = 0  
 while (abs(x\_next - x\_curr) > x.epsilon1):  
 i += 1  
 x\_curr = x\_next  
 x\_next = x\_curr - x.alpha \* f(x\_curr)  
 return [x\_next, i]

# вспомогательная функция для переноса осей в центр на графике

def transfer\_axis(plt):  
 ax = plt.gca()  
 ax.spines[**'left'**].set\_position(**'center'**)  
 ax.spines[**'bottom'**].set\_position(**'center'**)  
 ax.spines[**'top'**].set\_visible(False)  
 ax.spines[**'right'**].set\_visible(False)  
  
  
epsilon = math.pow(10, -8)  
a0 = -1.  
b0 = 1.

# построение графика f(x)

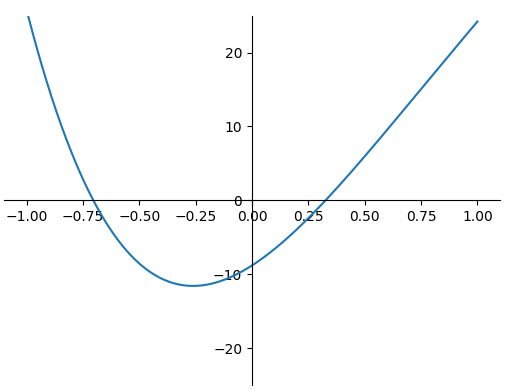
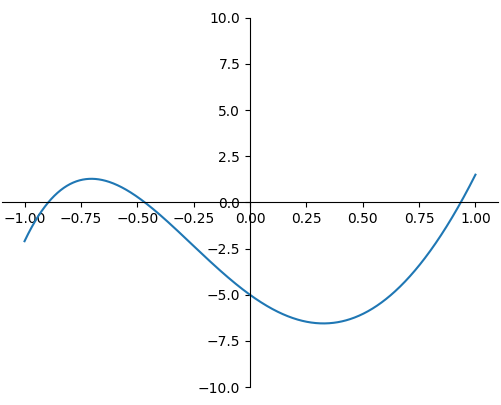
fig = plt.subplots()  
x\_data = numpy.linspace(a0, b0, 1000)  
plt.plot(x\_data, f(x\_data))  
plt.ylim([-10, 10])  
transfer\_axis(plt)  
plt.show()

# построение графика f’(x)

fig = plt.subplots()  
plt.plot(x\_data, df(x\_data))  
plt.ylim([-25, 25])  
transfer\_axis(plt)  
plt.show()

# отрезки локализации найдены по графику

x1 = Root(-1., -.75)  
x2 = Root(-.5, -.25)  
x3 = Root(.75, 1.)  
  
print(**'x1 = '** + round(find\_root(x1)[0], 8).\_\_str\_\_() + **' Iterations: '** + find\_root(x1)[1].\_\_str\_\_())  
print(**'x2 = '** + round(find\_root(x2)[0], 8).\_\_str\_\_() + **' Iterations: '** + find\_root(x2)[1].\_\_str\_\_())  
print(**'x3 = '** + round(find\_root(x3)[0], 8).\_\_str\_\_() + **' Iterations: '** + find\_root(x3)[1].\_\_str\_\_())

**Результаты выполнения:**

*График f(x) График f’(x)*

x1 = -0.89479384 Iterations: 5

x2 = -0.46653499 Iterations: 6

x3 = 0.93485048 Iterations: 7

**Анализ результатов:**

Многочлен имеет три корня. Метод простой итерации для указанной точности и для выбранных отрезков локализации сходится относительно быстро, потребовалось 5-7 итераций для определения корней с заданной точностью

**Постановка задачи:**

**Задача 2.2.** Дано уравнение  Найти все корни уравнения с заданной точностью  на указанном отрезке [a,b]. Для решения задачи использовать метод Ньютона и метод, указанный в индивидуальном варианте. Сравнить количество итераций, потребовавшихся для достижения заданной точности каждым методом.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ

1. Локализовать корни уравнения.

2. Составить программу вычисления корня методом Ньютона, предусмотрев в ней подсчёт числа итераций. Найти с заданной точностью корни уравнения на указанном в задании отрезке [a,b].

3. Составить программу вычисления корня методом, указанным в индивидуальном варианте, предусмотрев в ней подсчёт числа итераций. Найти с заданной точностью те же корни уравнения, что в п.2.

4. Сравнить результаты проведенных расчётов, сведя их в таблицу.

**5.** Модифицировать методы так, чтобы каждый метод делал заданное количество итераций и на каждом шаге сохранял значение модуля невязки . Методы должны возвращать массив, хранящий значения . Для каждого корня вызвать модифицированные методы так, чтобы они проделали 10 итераций. Построить графики зависимости  от n, , в логарифмической шкале. Каждому корню должно соответствовать одно изображение, на котором нарисованы зависимости для двух методов. Полученный результат объясните.

*Примечание: для построения графика в логарифмической шкале воспользуйтесь командой* plt.yscale(‘log’).

|  |  |  |
| --- | --- | --- |
| **Уравнение**  на [0,4]  Расчетная формула метода Ньютона:  Расчетная формула индивидуального метода : | | |
| **Задача 2.2** | | |
| *Корни уравнения* | Число итераций  метода Ньютона | Число итераций  метода |
| 0.0 | 0 | 0 |
| 1.495933004551 | 3 | 34 |
| 2.061035871927 | 4 | 9 |
| 2.51810456373 | 4 | 14 |
| 3.652002852693 | 4 | 107 |
| 3.881555642875 | 4 | 7 |
| 3.957882773604 | 4 | 15 |

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy  
import math  
import matplotlib.pyplot as plt

import matplotlib.ticker as ticker

# Аналогично задаче №1 объявляем класс Корень:

class Root:  
 def \_\_init\_\_(self, a, b):  
 self.a = a  
 self.b = b  
 self.x0 = (a + b) / 2  
 x\_data\_ = numpy.linspace(a, b, 100)  
 self.max = max(df(x\_data\_))  
 self.min = min(df(x\_data\_))  
 self.alpha = 2 / (self.max + self.min)  
 self.q = numpy.abs((self.max - self.min) / (self.max + self.min))  
 self.epsilon1 = (1 - self.q) / self.q \* epsilon  
  
#

def f(x):  
 return numpy.sqrt(x) \* numpy.cos(x\*\*2 / 3) - numpy.sin(x\*\*3 / 3)

#   
  
def df(x):  
 return (-4 \* x\*\*2 \* numpy.sin(x\*\*2 / 3) + 3 \* numpy.cos(x\*\*2 / 3) - 6 \* x\*\*2 \* numpy.sqrt(x) \* numpy.cos(x\*\*3 / 3)) / (6 \* numpy.sqrt(x))

# Шаг метода Ньютона:  
  
def newton\_step(x):  
 return x - f(x) / df(x)  
  
# Функция для поиска корня методом Ньютона,

# параметры – начальное приближение, ограничение на количество итераций:

# возвращаемое значение – корень, число итераций, вектор невязки

def newton(x, n):  
 j = 0  
 r = []  
 x\_curr = x.x0  
 x\_next = newton\_step(x\_curr)  
 while ((abs(x\_next - x\_curr) > epsilon) and (j < n)):  
 r.append(abs(x\_curr))  
 x\_curr = x\_next  
 x\_next = newton\_step(x\_curr)  
 j += 1  
 return [x\_next, j, r]  
  
# функция для нахождения корня по методу простой итерации

# параметры – начальное приближение, ограничение на количество итераций:

# возвращаемое значение – корень, число итераций, вектор невязки

def simple\_iteration(x, n):  
 j = 0  
 r = []  
 x\_curr = x.x0  
 x\_next = x\_curr - x.alpha \* f(x\_curr)  
 while ((abs(x\_next - x\_curr) > x.epsilon1) and (j < n)):  
 r.append(abs(x\_curr))  
 j += 1  
 x\_curr = x\_next  
 x\_next = x\_curr - x.alpha \* f(x\_curr)  
 return [x\_next, j, r]  
  
# вспомогательная функция для переноса осей в центр на графике

def transfer\_axis(plt):  
 ax = plt.gca()  
 ax.spines[**'bottom'**].set\_position(**'center'**)  
 ax.spines[**'top'**].set\_visible(False)  
 ax.spines[**'right'**].set\_visible(False)  
  
  
epsilon = math.pow(10, -12)  
a0 = 0.  
b0 = 4.

# построение графика f(x)

fig = plt.subplots()  
x\_data = numpy.linspace(a0, b0, 1000)  
plt.plot(x\_data, f(x\_data))  
plt.ylim([-3, 3])  
transfer\_axis(plt)  
plt.show()

# находим отрезки локализации для корней с помощью графика

# корни хранятся в массиве roots

roots = []  
*#roots.append(Root(-.1, .1))*roots.append(Root(1.25, 1.75))  
roots.append(Root(2., 2.25))  
roots.append(Root(2.4, 2.6))  
roots.append(Root(3.5, 3.75))  
roots.append(Root(3.85, 3.9))  
roots.append(Root(3.94, 4.))

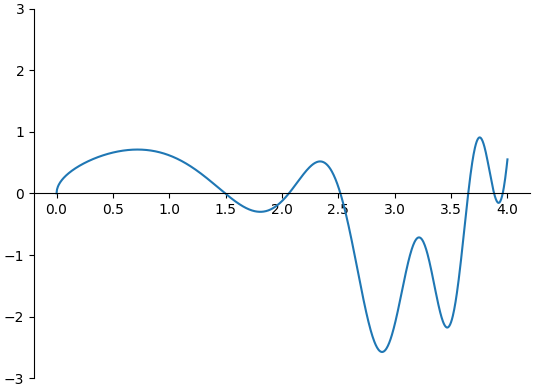
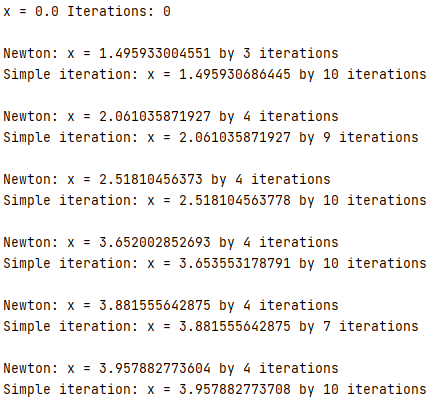
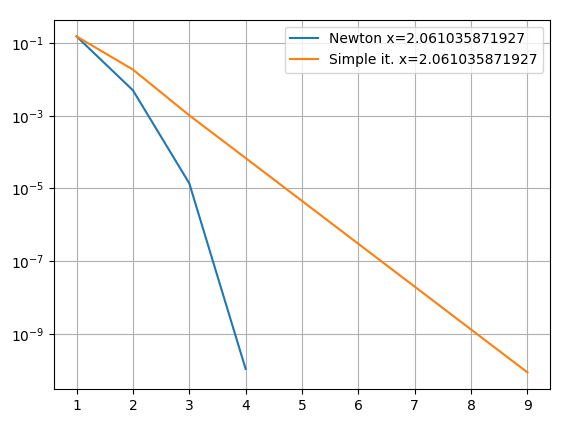
# корень 0 – очевиден и не требует апроксимации

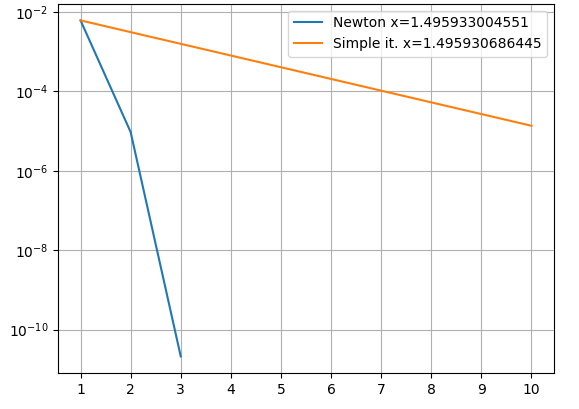
print(**'x = 0.0 Iterations: 0'**)  
print()

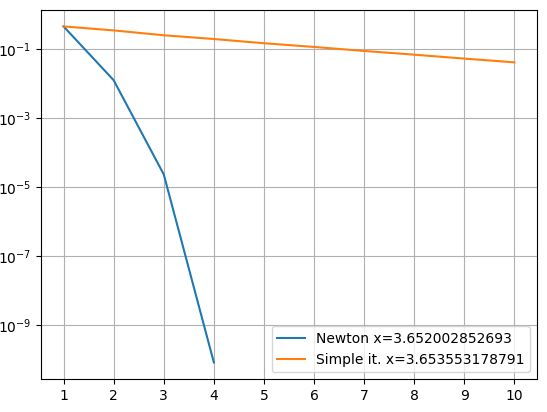
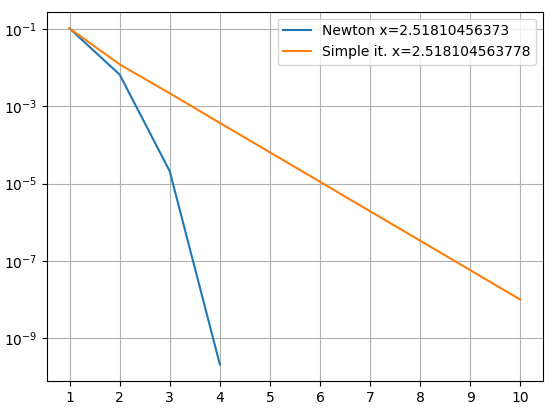
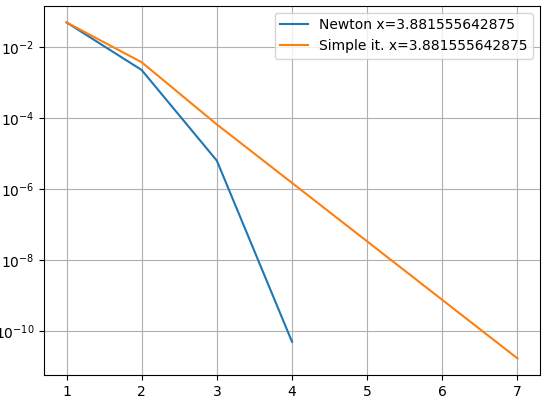
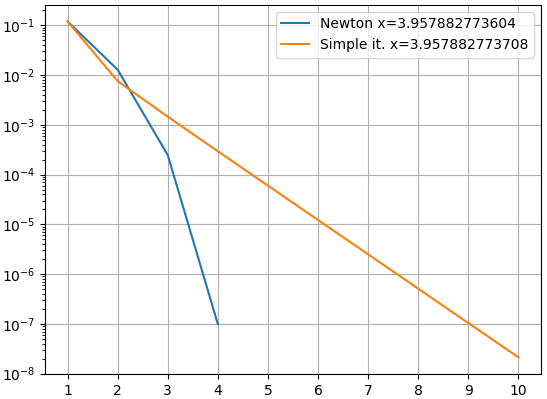
# для каждого корня вызываем метод Ньютона и метод простой итерации,

# строим график – вектора невязки в логарифмической шкале для каждого метода

for i in roots:  
 result\_1 = newton(i, 10)  
 result\_2 = simple\_iteration(i, 10)  
 a0 = 1  
 b0 = result\_1[2].\_\_len\_\_()  
 fig, ax = plt.subplots()  
 n\_data = numpy.linspace(a0, b0, b0)  
 plt.plot(n\_data, result\_1[2], label=**'Newton x='** + round(result\_1[0], 12).\_\_str\_\_())  
  
 b0 = result\_2[2].\_\_len\_\_()  
 n\_data = numpy.linspace(a0, b0, b0)  
 plt.plot(n\_data, result\_2[2], label=**'Simple it. x='** + round(result\_2[0], 12).\_\_str\_\_())  
 plt.yscale(**'log'**)  
 plt.grid()  
 plt.legend()  
 ax.xaxis.set\_major\_locator(ticker.MultipleLocator(1))  
 plt.show()  
  
 print(**'Newton: x = '** + round(result\_1[0], 12).\_\_str\_\_() + **' by '** + result\_1[1].\_\_str\_\_() + **' iterations'**)  
 print(**'Simple iteration: x = '** + round(result\_2[0], 12).\_\_str\_\_() + **' by '** + result\_2[1].\_\_str\_\_() + **' iterations'**)  
 print()

******Результаты выполнения:**

****

****

**Анализ результатов:**

У данного уравнения 7 корней на заданном отрезке. Корень x=0 вполне очевиден, он не требует апроксимации. Было установлено, что при попытке применить метод Ньютона к данному корню, обнаруживаются ошибки, так как знаменатель производной обращается в 0 при x=0.

Что касается других корней, то очевидно метод Ньютона сходится быстрее метода простой итерации. Из графиков невязок видно, что метод Ньютона имеет квадратичную скорость сходимости, то есть на каждой итерации точность увеливается примерно вдвое. Квадратичность сходимости также показывает, что все корни имеют кратность 1. Метод простой итерации сходится линейно.

Для некоторых корней достичь 10 итераций для заданной точности даже методом простой не удалось, поэтому вектор невязок будет иметь длину меньше 10. Попытка увеличения точности не приводит к желаемому результату. Увеличивать отрезки локализации не предоставляется возможным, так как корни расположены очень «кучно» и увеличив отрезок мы получим несколько корней на отрезке.

**Постановка задачи:**

**Задача 2.3 . Найти корни уравнения и определить их кратность.**

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy  
import math  
import matplotlib.pyplot as plt

# Аналогично задаче №1 объявляем класс Корень:  
  
class Root:  
 def \_\_init\_\_(self, a, b):  
 self.a = a  
 self.b = b  
 self.x0 = (a + b) / 2

#   
  
def f(x):  
 return 800 \* numpy.arctan((4 \* x - 11) / (5 \* x + 21)) - 168 \* x + 16 \* x\*\*2 + 341

#   
  
def df(x):  
 return 8 \* (164 \* x\*\*3 - 373 \* x\*\*2 - 314 \* x + 2098) / (41 \* x\*\*2 + 122 \* x + 562)

# Шаг метода Ньютона для кратных корней:

def newton\_step(x, m):  
 return x - m \* f(x) / df(x)  
  
# Функция для поиска корня методом Ньютона для кратных корней,

# параметры – начальное приближение, кратность корня, ограничение на

# количество итераций

# возвращаемое значение – корень, число итераций

def newton(x, m, n=50):  
 j = 0  
 x\_curr = x.x0  
 x\_next = newton\_step(x\_curr, m)  
 while ((abs(x\_next - x\_curr) > epsilon) and (j < n)):  
 x\_curr = x\_next  
 x\_next = newton\_step(x\_curr, m)  
 j += 1  
 return [x\_next, j]

# вспомогательная функция для переноса осей в центр на графике  
  
def transfer\_axis(plt):  
 ax = plt.gca()  
 ax.spines[**'left'**].set\_position(**'center'**)  
 ax.spines[**'bottom'**].set\_position(**'center'**)  
 ax.spines[**'top'**].set\_visible(False)  
 ax.spines[**'right'**].set\_visible(False)  
  
  
epsilon = math.pow(10, -12)  
a0 = -6.  
b0 = 6.

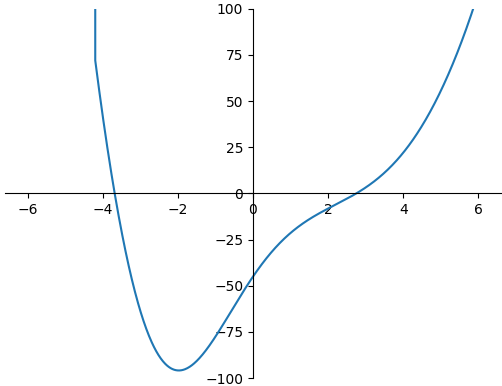
# построение графика f(x)  
  
fig = plt.subplots()  
x\_data = numpy.linspace(a0, b0, 1000)  
plt.plot(x\_data, f(x\_data))  
plt.ylim([-100, 100])  
transfer\_axis(plt)  
plt.show()

# находим отрезки локализации для корней с помощью графика

x1 = Root(-4., -3.)  
x2 = Root(2., 3.)

# вывод результатов для m=1  
  
result = newton(x1, 1)  
print(**'x1 = '** + round(result[0], 12).\_\_str\_\_() + **' by '** + result[1].\_\_str\_\_() + **' iterations. Multiplicity: 1'**)  
result = newton(x2, 1)  
print(**'x2 = '** + round(result[0], 12).\_\_str\_\_() + **' by '** + result[1].\_\_str\_\_() + **' iterations. Multiplicity: 1'**)

**Результаты выполнения:**

****

**Анализ результатов:**

Для решения задачи используем модификацию метод Ньютона для кратных корней. Для каждого корня будем подбирать значения параметра m = 1, 2, 3, 4… до тех пор пока метод не будет сходиться(количество итераций будет приемлемым). В данном конкретном примере процесс завершается при m=1 для обоих корней.