БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №2

**Численное решение систем нелинейных уравнений**

**Выполнил:**

Соломевич Александр

2 курс 6 группа

**Преподаватель:**

Горбачева Ю.Н.

Минск, 2022

**Содержание**

Постановка задачи……………………………………………………………………………....3

Краткие теоретические сведения…………………………………………………………………………....4

Листинг программы……………………………………………….…………………….…..5

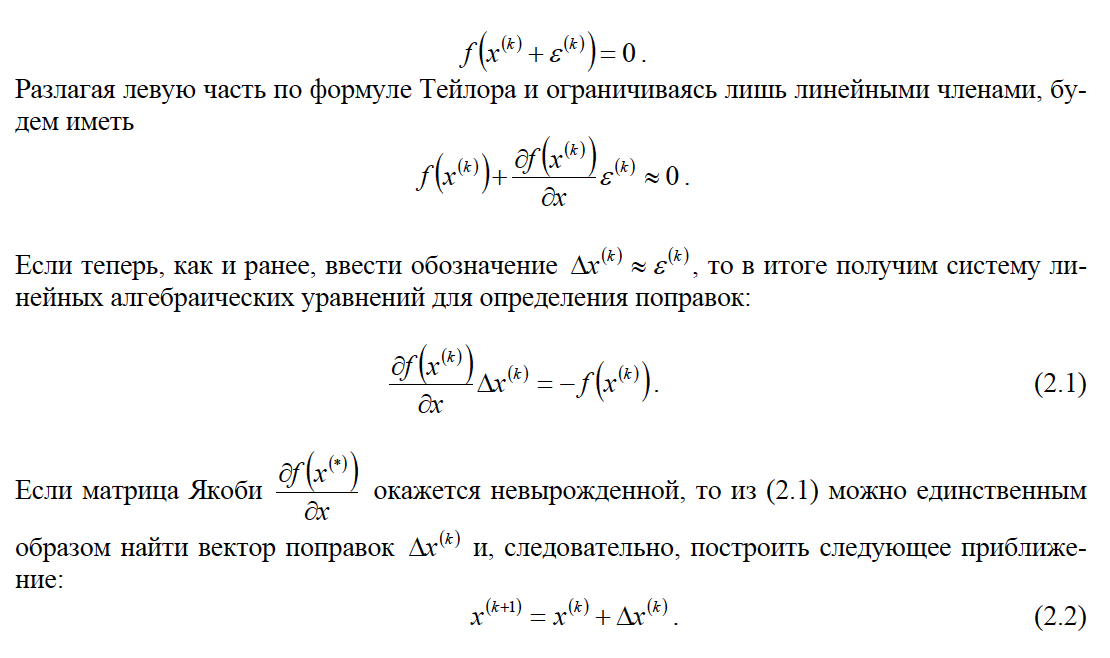
Результаты….……………………………………….….……………………….…8

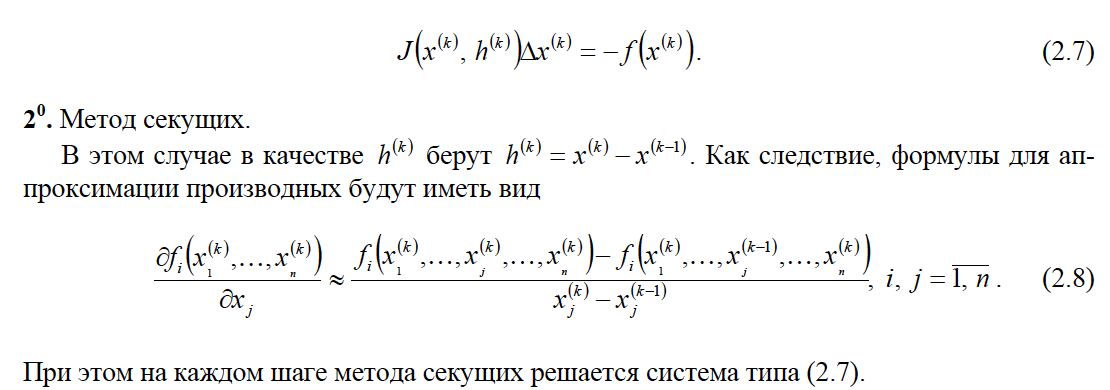
Выводы…..……………………………..………………………………………...10

**Постановка задачи**

Написать программу, которая решает систему нелинейных уравнений *f* (*x*) = 0 c точностью ε = 10-6 с помощью метода Ньютона, метода секущих, метода Гаусса-Зейделя. Начальное приближение выбрать графически. Провести сравнительный анализ полученных результатов.

**Краткие теоретические сведения**



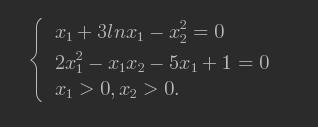


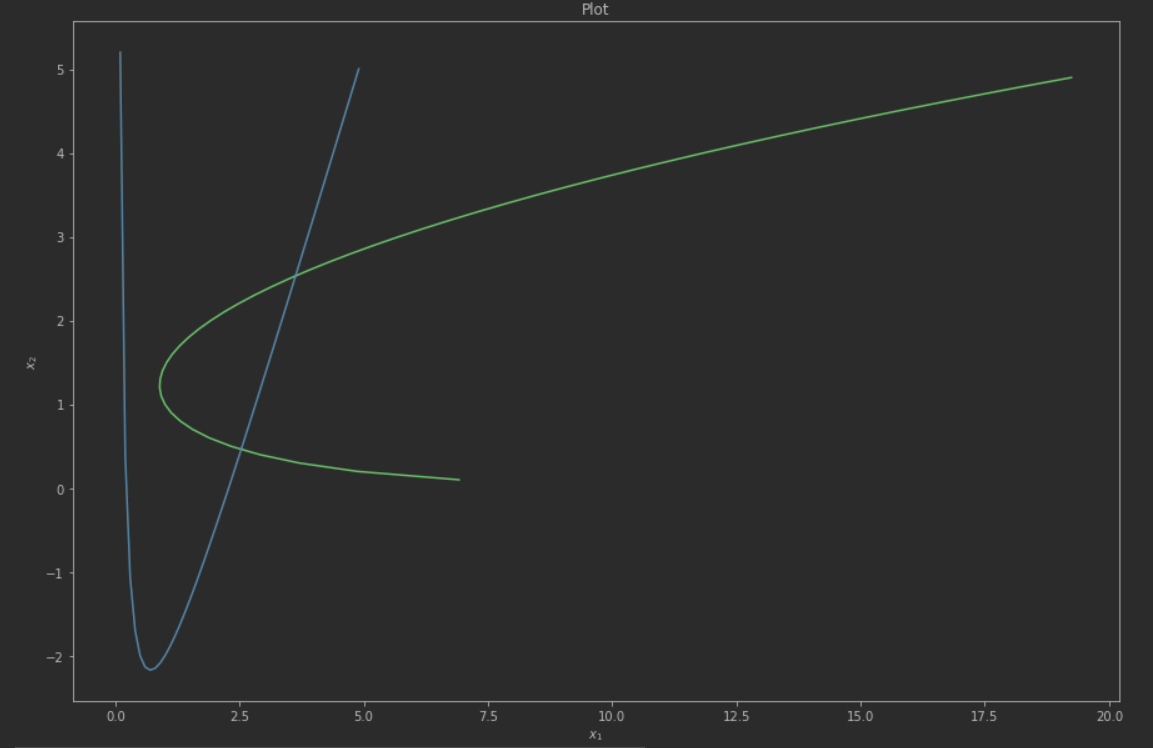
**Листинг программы**

**Lab2.ipybn**

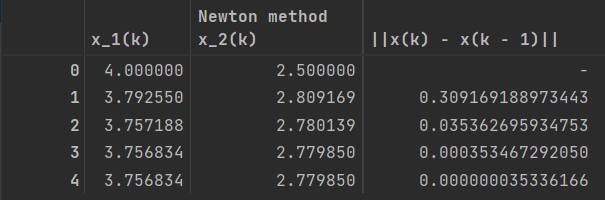
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
#%% md  
*### Initial system of equations:*  
\begin{equation\*}  
 \begin{cases}  
 x\_1 + 3lnx\_1 - x\_2^2 = 0  
 \\  
 2x\_1^2 - x\_1x\_2 - 5x\_1 + 1 = 0  
 \\  
 x\_1 > 0, x\_2 > 0.  
 \end{cases}  
\end{equation\*}  
#%%  
epsilon = 10 \*\* -6  
#%%  
def f(x):  
 return np.array(([x[0] + 3 \* np.log(x[0]) - x[1] \*\* 2, 2 \* (x[0] \*\* 2) - x[0] \* x[1] - 5 \* x[0] + 1]))  
  
def df(x):  
 return np.array([[1 + 3 / x[0], -2 \* x[1]], [4 \* x[0] - x[1] - 5, -x[0]]])  
#%%  
x = np.arange(0.1, 5, 0.1)  
#%%  
plt.figure(figsize=(15, 10))  
plt.plot(x \*\* 2 - 3 \* np.log(x), x, color='green')  
plt.plot(x, (2 \* (x \*\* 2) - 5 \* x + 1) / x)  
plt.xlabel('$x\_1$')  
plt.ylabel('$x\_2$')  
plt.title('Plot')  
plt.show()  
#%% md  
\* Taking plot into account I'll take $x^{(0)}$ = (4.0, 2.5) as initial approximation.  
#%% md  
*### Newton method:*#%%  
data = {'x\_1(k)': [4.0],  
 'x\_2(k)': [2.5],  
 '||x(k) - x(k - 1)||': ['-']}  
  
report\_data\_nt = pd.DataFrame(data)  
report\_data\_nt.columns = pd.MultiIndex.from\_tuples([('', 'x\_1(k)'), ('Newton method', 'x\_2(k)'), ('', '||x(k) - x(k - 1)||')])  
report\_data\_nt  
#%%  
x = np.array([4.0, 2.5])  
delta = x  
k = 0  
norm = np.linalg.norm(delta, ord=np.inf)  
while (norm >= epsilon) & (k < 1000):  
 delta = np.linalg.solve(df(x), -f(x))  
 norm = np.linalg.norm(delta, ord=np.inf)  
 x += delta  
 k += 1  
 report\_data\_nt.loc[len(report\_data\_nt)] = [x[0], x[1], format(norm, '.15f')]  
#%%  
np.linalg.norm(f(x), ord=np.inf)  
#%%  
report\_data\_nt  
#%% md  
*### Curves method:*#%%  
x\_1 = np.array([4.0, 2.5])  
x\_2 = x\_1 + np.linalg.solve(df(x\_1), -f(x\_1))  
#x\_2 = np.array([2.6, 1.2])  
norm = np.linalg.norm(x\_1 - x\_2, ord=np.inf)  
print(x\_1, x\_2)  
#%%  
data = {'x\_1(k)': [x\_1[0], x\_2[0]],  
 'x\_2(k)': [x\_1[1], x\_2[1]],  
 '||x(k) - x(k - 1)||': ['-', format(norm, '.15f')]}  
  
report\_data\_cur = pd.DataFrame(data)  
report\_data\_cur.columns = pd.MultiIndex.from\_tuples([('', 'x\_1(k)'), ('Curves method', 'x\_2(k)'), ('', '||x(k) - x(k - 1)||')])  
report\_data\_cur  
#%%  
  
def I(x1, x2):  
 return np.array([(f(x2) - f([x1[0], x2[1]])) / (x2[0] - x1[0]), (f(x2) - f([x2[0], x1[1]])) / (x2[1] - x1[1])]).T  
#%%  
k = 0  
while (norm >= epsilon) & (k < 1000):  
 matrixI = I(x\_1, x\_2)  
 delta = np.linalg.solve(matrixI, -f(x\_2))  
 norm = np.linalg.norm(delta, ord=np.inf)  
 k += 1  
 x\_1 = x\_2.copy()  
 x\_2 += delta  
 report\_data\_cur.loc[len(report\_data\_cur)] = [x\_2[0], x\_2[1], format(norm, '.15f')]  
#%%  
np.linalg.norm(f(x\_2), ord=np.inf)  
#%%  
report\_data\_cur  
#%% md  
*### Gauss-Zeidel method:*#%%  
def df1(x):  
 return 4 \* x[0] - x[1] - 5  
  
def df2(x):  
 return -2 \* x[1]  
#%%  
data = {'x\_1(k)': [4.0],  
 'x\_2(k)': [2.5],  
 '||x(k) - x(k - 1)||': ['-']}  
  
report\_data\_gz = pd.DataFrame(data)  
report\_data\_gz.columns = pd.MultiIndex.from\_tuples([('', 'x\_1(k)'), ('Gauss-Zeidel method', 'x\_2(k)'), ('', '||x(k) - x(k - 1)||')])  
report\_data\_gz  
#%%  
x = np.array([4.0, 2.5])  
k = 0  
norm = np.linalg.norm(f(x), ord=np.inf)  
while (norm >= 5 \* epsilon) & (k < 1000):  
 delta = f(x)[1] / df1(x)  
 while abs(delta) >= epsilon:  
 delta = f(x)[1] / df1(x)  
 x[0] -= delta  
 delta = f(x)[0] / df2(x)  
 while abs(delta) >= epsilon:  
 delta = f(x)[0] / df2(x)  
 x[1] -= delta  
 norm = np.linalg.norm(f(x), ord=np.inf)  
 k += 1  
 report\_data\_gz.loc[len(report\_data\_gz)] = [x[0], x[1], format(norm, '.15f')]  
#%%  
np.linalg.norm(f(x), ord=np.inf)  
#%%  
report\_data\_gz  
#%%  
report = pd.concat([report\_data\_nt, report\_data\_cur, report\_data\_gz], axis=1).fillna('-')  
report

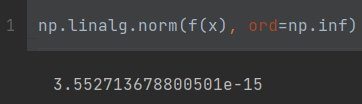
**Результаты:**

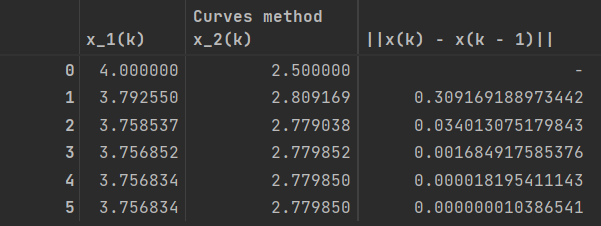
****

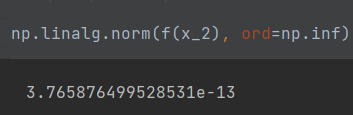
****

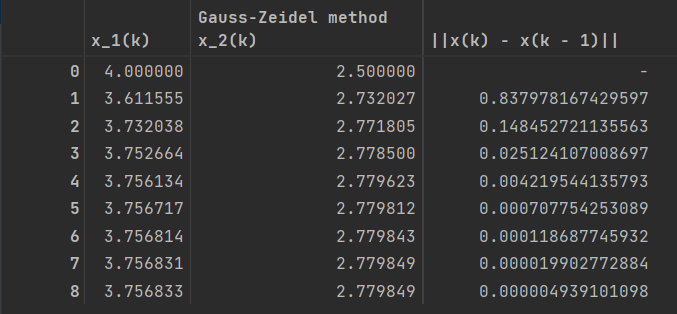
****

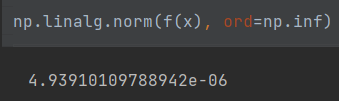
****

****

****

****

****

****

**Выводы:**

Наиболее быстрая сходимость наблюдалась при использовании метода Ньютона, который сошелся за 4 итерации при заданной точности. Методу секущих понадобилось 5 итераций для того, чтобы сойтись. Метод Гаусса-Зейделя оказался самым медленным и сошелся за 8 итераций.