Федеральное государственное автономное образовательное учреждение высшего образования
Национальный исследовательский университет
"Высшая школа экономики"

Московский институт электроники и математики имени А. Н. Тихонова
Программа "Прикладная математика"

Лабораторная работа №3

РЕШЕНИЕ СИСТЕМ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ ИТЕРАЦИОННЫМИ МЕТОДАМИ

Группа БПМ213
Вариант 8
Номера выполняемых задач: 4.1.8, 4.5.2, 5.1.8, 5.5.1

Выполнила:
Варфоломеева Анастасия Андреевна
Преподаватель:
Токмачев Михаил Геннадьевич

Москва, 2024 г.

1 Метод Ньютона для системы нелинейных уравнений

1.1 Формулировка задачи

Найти с точностью $\varepsilon = 10^{-6}$ все корни системы нелинейных уравнений $f_1(x_1, x_2) = 0$ и $f_2(x_1, x_2) = 0$ используя метод Ньютона для системы нелинейных уравнений.

$$\cos(x_1 + x_2) + 2x_2 = 0$$

$$x_1 + \sin x_2 - 0.6 = 0$$

1.2 Порядок решения задачи

1. Используя встроенные функции, локализовать корни системы уравнений графически.
2. Написать программу-функцию, вычисляющую корень системы двух нелинейных уравнений по методу Ньютона с точностью ε. Предусмотреть подсчет количества итераций. Для решения соответствующей системы линейных алгебраических уравнений использовать встроенную функцию. В качестве критерия остановы использовать следующее:

$$\frac{\|x^{k+1} - x^k\|}{1 - \frac{\|x^{k+1} - x^k\|}{\|x^k - x^{k-1}\|}} < \varepsilon$$

А в качестве основных формул:

$$f'(x^n)\Delta x^{n+1} = -f(x^n)$$

$$x^{n+1} = x^n + \Delta x^{n+1}$$

3. Используя написанную программу, вычислить все корни заданной системы с точностью ε.
4. Используя встроенные функции, найти все корни системы с точностью ε. Сравнить с результатами, полученнными в п. 3.

1.3 Код на Python

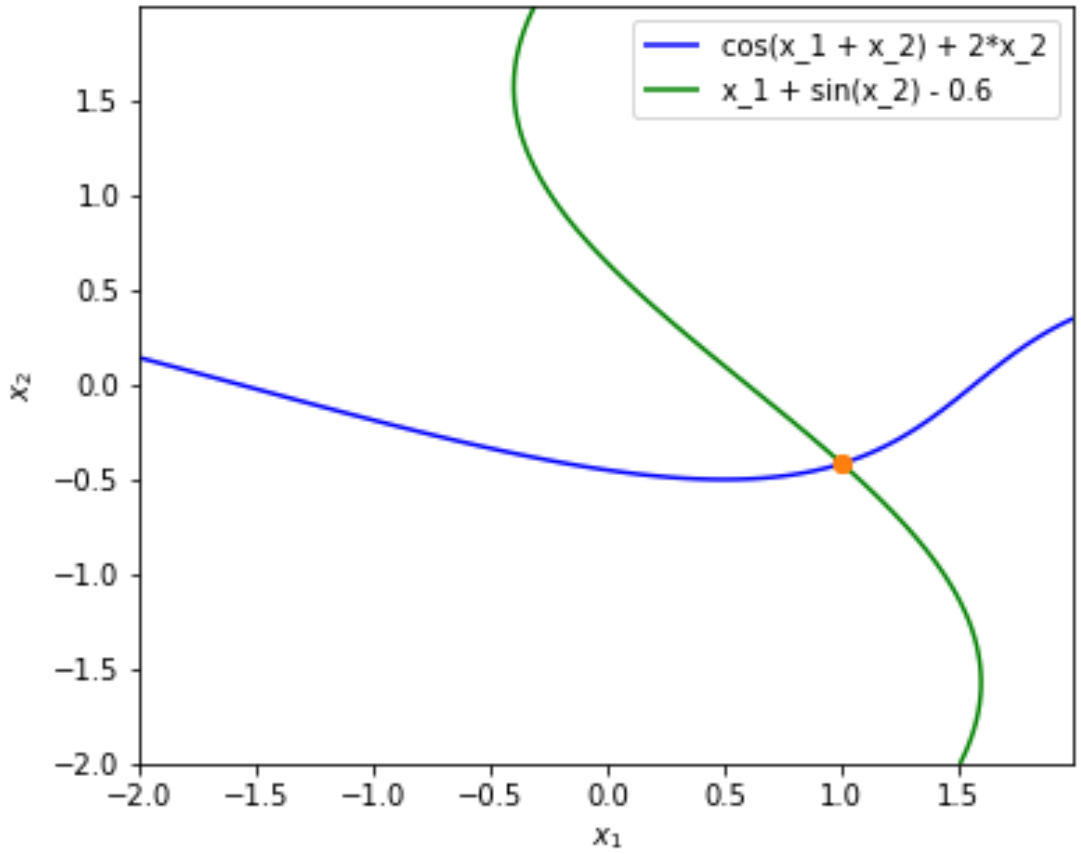
```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sym
import scipy.optimize

def find_point(arr):
    x_1 = arr[0]
    x_2 = arr[1]
    x1, x2 = sym.symbols("x1, x2")
    y = sym.cos(x1 + x2) + 2*x2
    z = x1 + sym.sin(x2) - 0.6
    df = np.array([[float(sym.diff(y, x1).subs([(x1,x_1), (x2,x_2)])), float(sym.diff
                    [float(sym.diff(z, x1).subs([(x1,x_1), (x2,x_2)])), float(sym.diff(z
```

```
    f = np.array([float(y.subs([(x1,x_1), (x2,x_2)])), float(z.subs([(x1,x_1), (x2,x_2
    return np.array([x_1 + np.linalg.solve(df, -f)[0], x_2 + np.linalg.solve(df, -f)[

def func_arr(x):
    return np.array([np.cos(x[0] + x[1]) + 2*x[1], x[0] + np.sin(x[1]) - 0.6])


x0 = [-3.48, 2.49]
eps = 10**(-6)
x_prev = np.array(x0)
x_real = find_point(x0)
x_new = np.array([])
count = 1

print(f'Start point: {x_prev}')
print('---------------')
while np.linalg.norm(np.linalg.norm(find_point(x_real) - x_real, 2)/(1 - np.linalg.nor
    x_prev = x_real
    x_real = find_point(x_real)
    print(f'{count} approximation: {x_prev}')
    print('---------------')
    count += 1

print(f'Result of presented function:  {x_prev}')
print(f'Result of build-in function:   {scipy.optimize.fsolve(func_arr, np.array(x0))

print(f'Difference in solutions: {np.linalg.norm((x_prev - scipy.optimize.fsolve(func_

x_1, x_2 = np.meshgrid(np.arange(-2, 2, 0.005), np.arange(-2, 2, 0.005))
fig, ax = plt.subplots(figsize=(6, 5))
contour1 = plt.contour(x_1, x_2, np.cos(x_1 + x_2) + 2*x_2, [0], colors=['blue'])
contour2 = plt.contour(x_1, x_2, x_1 + np.sin(x_2) - 0.6, [0], colors=['green'])
ax.scatter(*x_prev, c="#ff7f0e", alpha=1, zorder=2)
ax.scatter(*x_prev, c="#ff7f0e", alpha=1, zorder=2)
f1,_ = contour1.legend_elements()
f2,_ = contour2.legend_elements()
ax.legend([f1[0], f2[0]], ["cos(x_1 + x_2) + 2*x_2", "x_1 + sin(x_2) - 0.6"])
ax.set_xlabel("$x_1$")
ax.set_ylabel("$x_2$")
plt.show()
```

1.4 Результат работы программы

```
Start point: [-3.48  2.49]
---------------
1 approximation: [-1.92174048  0.08119439]
---------------
2 approximation: [ 1.63755262 -1.04116099]
---------------
3 approximation: [ 1.12358463 -0.36935936]
---------------
4 approximation: [ 1.00941475 -0.42125601]
---------------
5 approximation: [ 1.00410601 -0.41599511]
---------------
Result of presented function: [ 1.00410601 -0.41599511]
Result of build-in function:   [ 1.00410184 -0.41599671]
Difference in solutions: 4.468729912332492e-06
```

```
Start point: [1.5 0.3]
---------------
1 approximation: [ 1.0550031  -0.48561148]
---------------
2 approximation: [ 1.00498374 -0.4157718 ]
---------------
Result of presented function: [ 1.00498374 -0.4157718 ]
Result of build-in function:   [ 1.00410184 -0.41599671]
Difference in solutions: 0.0009101291070692241
```



1.5 Вывод

Для примера были взяты две точки: достаточно далекая от полученного графически решения и близкая. Написанная функция нашла ответ за 5 и 2 итерации соответственно. С помощью встроенной функции были найдены решения с разницей в $4.468729912332492e-06$ и 0.0009101291070692241 полученных своим решением. На графике также видно, что полученные ответы соответствуют графически представленному решению.

2 Определение расстояния точек в пространстве

2.1 Формулировка задачи

Даны координаты точек P_i, $i = 1, 2, 3$ и уравнение поверхности S в пространстве R^3. Определить ближайшую к поверхности точку и наиболее удаленную от поверхности точку. Построить на одном чертеже точечный график поверхности S и заданные точки P_i.

$$N = 2$$
$$a_1 = 8.5 - N * 0.25$$
$$a_2 = 2.3 + N * 0.3$$
$$a_3 = 4 + N * 0.1$$

Поверхность S задается уравнением:

$$(\frac{x_1}{a_1})^2 + (\frac{x_2}{a_2})^2 + (\frac{x_3}{a_3})^2 = 1$$

Координаты точки P_1: (16, 5.8, 11.879)
Координаты точки P_2: (8.485, 5.328, 8.91)
Координаты точки P_3: (15, 3.139, 5.25)
УКАЗАНИЯ. 1) Под расстоянием между точками $Q(x_1, x_2, x_3)$ и $P(x_1^0, x_2^0, x_3^0)$ в пространстве R^3 понимается величина $\rho(Q, P) = \sqrt{\sum_{j=1}^{3}(x_j - x_j^0)^2}$. Поэтому для решения задачи следует составить целевую функцию $H(x_1, x_2, x_3) = \sum_{j=1}^{3}(x_j - x_j^0)^2$ и минимизировать ее с помощью метода Ньютона при условии принадлежности точки поверхности S.
2) Условие принадлежности точки указанной поверхности S легко учесть, если ввести обобщенные координаты на этой поверхности.
3) При выборе начального приближения следует учесть, что все координаты заданных точек P_i, $i = 1, 2, 3$ положительны.
Обобщенные координаты:

$$x_1 = a_1 * \sin\phi * \cos\theta$$
$$x_2 = a_2 * \cos\phi * \sin\theta$$
$$x_3 = a_3 * \cos\phi$$

2.2 Код на Python

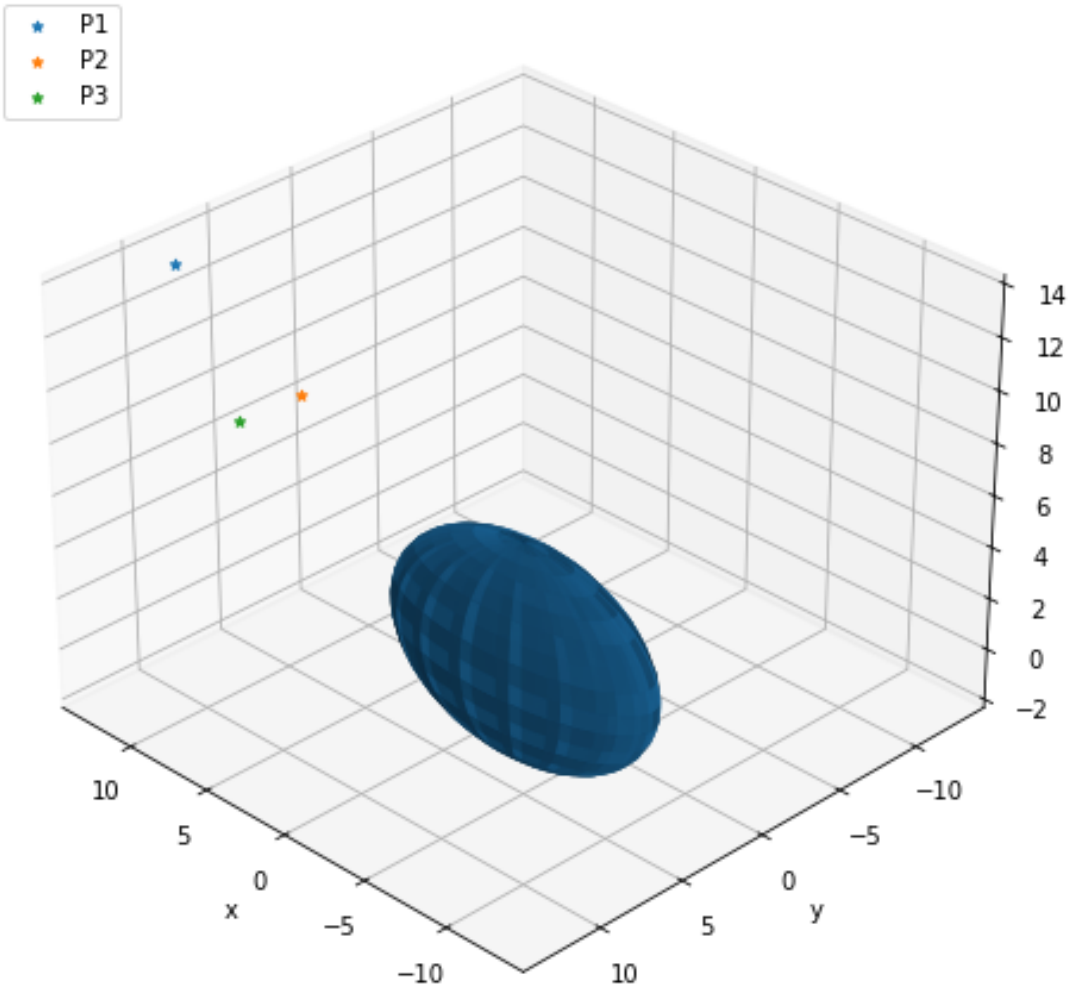
```
import numpy as np
import matplotlib.pyplot as plt

N = 2
a1 = 8.5 - N * 0.25
a2 = 2.3 + N * 0.3
a3 = 4 + N * 0.1
P1 = np.array([16, 5.8, 11.879])
P2 = np.array([8.485, 5.328, 8.91])
P3 = np.array([15, 3.139, 5.25])

def coordinates(q):
    phi = q[0]
    theta = q[1]
    return np.array([a1*np.sin(phi)*np.sin(theta), a2*np.cos(phi)*np.sin(theta), a3*np

def f(point):
    def func(q):
        return np.sum((coordinates(q) - point) ** 2)
    return func

def gradient(f, q, h = 1e-8):
    n = len(q)
    grad = np.empty_like(q)
    for i in range(n):
```

```
        delta = np.zeros_like(q)
        delta[i] = h
        grad[i] = (f(q + delta) - f(q-delta)) / 2 / h
    return grad

def hesse(f, q, h = 1e-8):
    n = len(q)
    hesse_matrix = np.empty((n, n), dtype=np.float64)
    for i in range(n):
        delta_i = np.zeros_like(q)
        delta_i[i] = h
        for j in range(n):
            delta_j = np.zeros_like(q)
            delta_j[j] = h
            hesse_matrix[i, j] = (f(q + delta_i + delta_j) - f(q + delta_i - delta_j)
            f(q - delta_i + delta_j) + f(q - delta_i - delta_j))/(4 * h * h)
    return hesse_matrix

def p(x2, x1, x0):
    norm_x2_x1 = np.max(np.abs(x2 - x1))
    norm_x1_x0 = np.max(np.abs(x1 - x0))
    return np.abs(norm_x2_x1 / (1 - norm_x2_x1/ norm_x1_x0))


def newton_method(f, x0,  eps = 1e-6):
    x1 = x0 + np.linalg.solve(hesse(f, x0), -gradient(f, x0))
    x2 = x1 + np.linalg.solve(hesse(f, x1), -gradient(f, x1))
    iter_cnt = 2
    while p(x2, x1, x0) > eps:
        x0 = x1
        x1 = x2
        try:
            x2 = x2 + np.linalg.solve(hesse(f, x2), -gradient(f, x2))
        except np.linalg.LinAlgError:
            return x2, iter_cnt
        iter_cnt += 1
    return x2, iter_cnt

def optimal(point, q0):
    func = f(point)
    q1,_ = newton_method(func, q0)
    dist = np.sqrt(func(q1))
    return dist

d1, d2, d3 = optimal(P1, [np.pi/6, np.pi/4]), optimal(P2, [np.pi/3, np.pi/4]), optimal

print(f"Point: {P1}")
print(f"Distance to P1 = {d1}")
print("-------------")
print(f"Point: {P2}")
```

```
print(f"Distance to P2 = {d2}")
print("-------------")
print(f"Point: {P3}")
print(f"Distance to P3 = {d3}")
print("-------------")
print(f"The closest point is P{np.argmin([d1, d2, d3]) + 1}")
print(f"The furthest point is P{np.argmax([d1, d2, d3]) + 1}")

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(projection='3d')

phi = np.linspace(0, 3 * np.pi, 100)
theta = np.linspace(0, 3 * np.pi, 100)
x = a1 * np.outer(np.cos(theta), np.sin(phi))
y = a2 * np.outer(np.sin(theta), np.sin(phi))
z = a3 * np.outer(np.ones_like(theta), np.cos(phi))

ax.plot_surface(x, y, z,  rstride=4, cstride=4, alpha = 0.5)
ax.scatter(*P1, marker='*', label = "P1")
ax.scatter(*P2, marker='*', label = "P2")
ax.scatter(*P3, marker='*', label = "P3")
ax.view_init(30, 135, 0)
ax.set_xlim(-14, 14)
ax.set_ylim(-14, 14)
ax.set_zlim(-2, 14)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
plt.legend(loc="upper left")
plt.show()
```

2.3 Результат работы программы

```
Point: [16.     5.8   11.879]
Distance to P1 = 23.16831416365651
-------------
Point: [8.485 5.328 8.91 ]
Distance to P2 = 7.2955225548433065
-------------
Point: [15.     3.139  5.25 ]
Distance to P3 = 8.6682388376331
-------------
The closest point is P2
The furthest point is P1
```



2.4 Вывод

В программе был реализован метод Ньютона для минимизации составленной целевой функции $H(x_1, x_2, x_3)$, описанной в формулировке задачи. В итоге были найдены расстояние от поверхности эллипса до указанных трех точек и построен график, отражающий положение точек и эллипса в трехмерном пространстве. В выводе программы указаны самая ближняя и дальняя точки, это P_2 и P_1 соответственно, что также можно увидеть на представленном графике.

3 Решение системы с помощью метода Гаусса

3.1 Формулировка задачи

Дана система уравнений $Ax = b$. Найти решение системы с помощью метода Гаусса. Выполнить 10 итераций по методу Зейделя. Принимая решение, полученное с помощью метода Гаусса за точное, найти величину абсолютной погрешности итерационного решения.

3.2 Порядок решения задачи

1. Задать матрицу системы A и вектор правой части b. Найти решение системы $Ax = b$ с помощью метода Гаусса.
2. Преобразовать систему $Ax = b$ к виду $x = Bx + c$, удобному для итераций, где

$B = B_1 + B_2$ (B_1 - нижнетреугольная матрица, а B_2 - верхнетреугольная). Проверить выполнение достаточного условия сходимости итерационных методов $\|B_2\|_\infty < 1$.
3. Написать программу-функцию *zeid*, решающую систему уравнений с помощью метода Зейделя, выполнить 10 итераций по методу Зейделя; взять любое начальное приближение. В качестве основной формулы использовать следующее:

$$x^{k+1} = B_1 x^{k+1} + B_2 x^k + c$$

Принимая решение, полученное в п. 1 за точное, найти величину абсолютной погрешности итерационного решения (использовать норму $\|\cdot\|_\infty$).
4. Взять другое начальное приближение. Объяснить полученные результаты.

$$A = \begin{pmatrix} 118.8 & -14 & -5 & -89.1 \\ -59.4 & 194 & 5 & 128.7 \\ 148.5 & 12 & -310 & 148.5 \\ 0 & 18.5 & 90 & -108.9 \end{pmatrix}$$

$$b = \begin{pmatrix} -92.5 \\ -340.1 \\ -898 \\ 184.1 \end{pmatrix}$$

3.3 Код на Python

```
import numpy as np

A = np.array([
    [118.8, -14, -5, -89.1],
    [-59.4, 194, 5, 128.7],
    [148.5, 12, -310, 148.5],
    [0, 18.5, 90, -108.9]
])
b = np.array([-92.5, -340.1, -898, 184.1])

def zeid(A, b, x):
    B = np.empty((len(A),len(A)))
    c = np.empty((len(A),1))
    for i in range(len(A)):
        B[i] = -A[i]/A[i][i]
        c[i] = b[i]/A[i][i]
    B1 = np.tril(B, -1)
    B2 = np.triu(B, 1)

    x_new = np.zeros(len(A))
    for i in range(len(x)):
        x_new[i] = np.dot(B1[i], x_new) + np.dot(B2[i], x) + c[i]
    return x_new

gauss = np.linalg.solve(A, b)

B = np.empty((len(A),len(A)))
```

```
c = np.empty((len(A),1))
for i in range(len(A)):
    B[i] = -A[i]/A[i][i]
    c[i] = b[i]/A[i][i]
B1 = np.tril(B, -1)
B2 = np.triu(B, 1)
print(f"Sufficient convergence condition is completed: {np.linalg.norm(B2, np.inf)} <
print(f"------------------")

eps = np.linalg.norm(((1 - np.linalg.norm(B, np.inf))/np.linalg.norm(B2, np.inf)))*10

num = 0
x = [0.6, 2.3, -2.8, -1.35]
print(f"Start approach: {x}")
while num <= 10:
    x = zeid(A, b, x)
    num += 1
print(f"Method Gauss solution:     {gauss}")
print(f"Solution of zeid function (10 iterations): {x}")
print(f"Difference in solution: {np.linalg.norm(gauss - x, np.inf)}")
print(f"------------------")

print(f"Start approach: {x}")
x_prev = np.array(x)
x_real = zeid(A, b, x)
count = 0
while np.linalg.norm(x_prev - x_real, np.inf) >= eps:
    count += 1
    x_prev = x_real
    x_real = zeid(A, b, x_real)
print(f"Method Gauss solution:                              {gauss}")
print(f"Solution of zeid function (epsilon with norm) : {x_real} was reached by {count
print(f"Difference in solution: {np.linalg.norm(gauss - x_real, np.inf)}")
print(f"------------------")

num = 0
x = [10, 10, 10, 10]
print(f"Start approach: {x}")
while num <= 10:
    x = zeid(A, b, x)
    num += 1
print(f"Method Gauss solution:     {gauss}")
print(f"Solution of zeid function: {x}")
print(f"Difference in solution: {np.linalg.norm(gauss - x, np.inf)}")
print(f"------------------")

x = [10, 10, 10, 10]
print(f"Start approach: {x}")
x_prev = np.array(x)
x_real = zeid(A, b, x)
```

```
count = 0
while np.linalg.norm(x_prev - x_real, np.inf) >= eps:
    count += 1
    x_prev = x_real
    x_real = zeid(A, b, x_real)
print(f"Method Gauss solution:                                  {gauss}")
print(f"Solution of zeid function (epsilon with norm) : {x_real} was reached by {count
print(f"Difference in solution: {np.linalg.norm(gauss - x_real, np.inf)}")
print(f"------------------")
```

3.4 Результат работы программы

```
Sufficient convergence condition is completed: 0.9099326599326599 < 1
------------------
Start approach: [0.6, 2.3, -2.8, -1.35]
Method Gauss solution:     [-1.08080808 -2.          2.2        -0.21212121]
Solution of zeid function (10 iterations): [-1.0834348  -1.99821921  2.19703906 -0.21426575]
Difference in solution: 0.0029609446929881322
------------------
Start approach: [-1.0834348  -1.99821921  2.19703906 -0.21426575]
Method Gauss solution:                          [-1.08080808 -2.          2.2        -0.21212121]
Solution of zeid function (epsilon with norm) : [-1.08080936 -1.99999913  2.19999856 -0.21212225] was reached by 13 iterations
Difference in solution: 1.439163140215527e-06
------------------
Start approach: [10, 10, 10, 10]
Method Gauss solution:     [-1.08080808 -2.          2.2        -0.21212121]
Solution of zeid function: [-1.04338179 -2.02537322  2.24218847 -0.18156512]
Difference in solution: 0.0421884748326713
------------------
Start approach: [10, 10, 10, 10]
Method Gauss solution:                          [-1.08080808 -2.          2.2        -0.21212121]
Solution of zeid function (epsilon with norm) : [-1.08080689 -2.00000081  2.20000134 -0.21212024] was reached by 29 iterations
Difference in solution: 1.3444507245274906e-06
------------------
```

3.5 Вывод

Для примера были взяты два начальных приближения: близкое к решению, полученному встроенным методом Гаусса, и достаточно далекое от него. Написанная функция нашла близкие к верному ответы для указанной точности (10 итераций) с соответствующими ошибками: 0.0029609446929881322 и 0.0421884748326713. Также был использован критерий остановки метода с точностью $\varepsilon = 10^{-6}$ чтобы понять, за сколько итераций с заданной точностью сойдется метод по следующей формуле:

$$\varepsilon = \frac{1 - \|B\|}{\|B_2\|} 10^{-6}$$

В итоге получились 13 и 29 итераций для указанных начальных приближений и соответствующие ошибки: $1.439163140215527e-06$ и $1.3444507245274906e-06$. Также в начале было проверено достаточное условие сходимости итерационных методов и, как видно из результатов, оно выполнилось, то есть метод сойдется и будет иметь разные ошибки в зависимости от выбора начального приближения и количества итераций.

4 Метод релаксации

4.1 Формулировка задачи

Дана система уравнений $Ax = b$, где A – симметричная положительно определенная матрица. Найти решение системы с точностью $\varepsilon = 10^{-5}$ с помощью метода релак-

сации (для этого модифицировать функцию *zeid* из задачи 5.2, реализующую метод Зейделя). Использовать формулу для итерационного процесса:

$$x^{k+1} = (1-\omega)x^k + \omega B_1 x^{k+1} + \omega B_2 x^k + \omega c$$

Определить экспериментально параметр релаксации ω, при котором точность ε достигается при наименьшем числе итераций. Построить график зависимости числа итераций от параметра релаксации.
УКАЗАНИЕ. Параметр релаксации ω следует задавать из условия сходимости метода: $\omega \in (0, 2)$. Например: ω =0.2, 0.4, ..., 1.8

$$A = \begin{pmatrix} 3.5 & -1 & 0.9 & 0.2 & 0.1 \\ -1 & 7.3 & 2 & 0.3 & 2 \\ 0.9 & 2 & 4.9 & -0.1 & 0.2 \\ 0.2 & 0.3 & -0.1 & 5 & 1.2 \\ 0.1 & 2 & 0.2 & 1.2 & 7 \end{pmatrix}$$

$$b = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$$

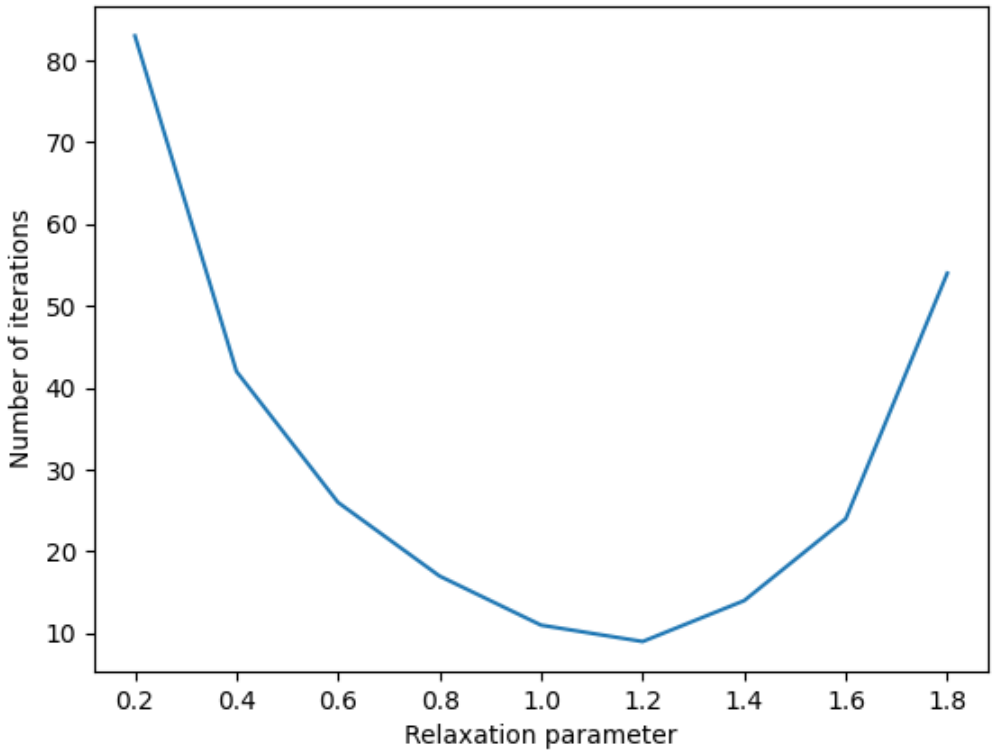
4.2 Код на Python

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

A = np.array([
    [3.5, -1, 0.9, 0.2, 0.1],
    [-1, 7.3, 2, 0.3, 2],
    [0.9, 2, 4.9, -0.1, 0.2],
    [0.2, 0.3, -0.1, 5, 1.2],
    [0.1, 2, 0.2, 1.2, 7]
])
b = np.array([1, 2, 3, 4, 5])

def zeid_mod(A, b, x, omega):
    B = np.empty((len(A), len(A)))
    c = np.empty((len(A),1))
    for i in range(len(A)):
        B[i] = -A[i]/A[i][i]
        c[i] = b[i]/A[i][i]
    B1 = np.tril(B, -1)
    B2 = np.triu(B, 1)

    x_new = np.zeros(len(A))
    for i in range(len(x)):
        x_h = np.zeros(len(A))
        x_new[i] = (1 - omega)*x[i] + omega*(np.dot(B1[i], np.hstack((x_new[:i], x_h[
    return x_new
```

```
print(f"Is matrix A symmetrical: {np.all(A==A.T)}")
print(f"Is matrix A positive: {np.all(np.linalg.eigvals(A) > 0)}")
print("-----------")

x0 = [0, 0, 0, 0, 0]
print(f"Start approach: {x0}")
print("-----------")
eps = 10**(-5)
num_of_it = []
omega = [om for om in np.arange(0.2, 2, 0.2)]
for om in omega:
    x_prev = np.array(x0)
    x_real = zeid_mod(A, b, x0, 0.2)
    count = 0
    while np.linalg.norm(x_prev - x_real, np.inf) >= eps:
        count += 1
        x_prev = x_real
        x_real = zeid_mod(A, b, x_real, om)
    num_of_it.append(count)

print(f"Method Gauss solution:         {np.linalg.solve(A, b)}")
print(f"Solution of relaxation method: {x_real}")
print(f"Difference in solutions: {np.linalg.norm(np.linalg.solve(A, b) - x_real, np.i
print("-----------")
print(f"Relaxation parameter for minimum iterations: {omega[num_of_it.index(min(num_o
print(f"Number of iterations for omega: {num_of_it}")

plt.plot(omega, num_of_it)
plt.xlabel("Relaxation parameter")
plt.ylabel("Number of iterations")
plt.show()
```

4.3 Результат работы программы

```
Is matrix A symmetrical: True
Is matrix A positive: True
-----------
Start approach: [0, 0, 0, 0, 0]
-----------
Method Gauss solution:         [ 0.04424026 -0.08535775  0.62794735  0.67068044  0.6051265 ]
Solution of relaxation method: [ 0.04423548 -0.08535637  0.6279469   0.67067998  0.6051239 ]
Difference in solutions: 4.7814813206384366e-06
-----------
Relaxation parameter for minimum iterations: 1.2
Number of iterations for omega: [83, 42, 26, 17, 11, 9, 14, 24, 54]
```



4.4 Вывод

В программе были проверены условия сходимости метода релаксации: матрица A является положилельно определенной симметричной матрицей, значит при любом значении ω ($\omega \in (0, 2)$) метод сойдется. Полученное решение сравнивалось с ответом встроенной функции метод Гаусса, которая подтвердила корректность написанного метода с разницей в $4.7814813206384366e-06$. Приведен график зависимости количества итераций от параметра релаксации. Из него и из вывода программы видно, что минимальное число итераций $= 9$ достигается при параметре релаксации $\omega = 1.2$.