

**Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»**

Образовательная программа
«Прикладная математика»

**ОТЧЕТ
По лабораторной работе № 3**

По предмету
«Численные методы»

По теме
**«Решение систем алгебраических уравнений
итерационными методами»**

Выполнил
студент группы БПМ211
Кудряшов Максим Дмитриевич

Проверил
Брандышев Петр Евгеньевич

Москва - 2024

Содержание

1	Метод Ньютона (№ 4.1.8)	2
1.1	Формулировка задания	2
1.2	Код на Python	2
1.2.1	Построение графика	2
1.2.2	Решение системы методом Ньютона	2
1.3	Результаты	5
2	Расстояния от поверхности до точки (№ 4.5.2)	5
2.1	Формулировка задания	5
2.2	Код на Python	6
3	Метод Зейделя (№ 5.1.8)	6
3.1	Формулировка задания	6
3.2	Код на Python	6
3.3	Результаты	7
4	Метод релаксации (№ 5.5.1)	8
4.1	Формулировка задания	8
4.2	Код на Python	8
4.3	Результаты	9

1 Метод Ньютона (№ 4.1.8)

1.1 Формулировка задания

Найти с точностью $\epsilon = 10^{-6}$ все корни системы нелинейных уравнений:

$$f_1(x_1, x_2) = 0$$

$$f_2(x_1, x_2) = 0$$

Использовать метод Ньютона для системы нелинейных уравнений, найти корни с помощью встроенного функционала решения уравнений.

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.
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, полученными в п. 3.

1.2 Код на Python

1.2.1 Построение графика

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  x1, x2 = np.meshgrid(np.arange(-10, 10, 0.05), np.arange(-10, 10, 0.05))
5  plt.figure(figsize=(6, 5))
6  plt.contour(x1, x2, np.cos(x1 + x2) + 2 * x2, [0], colors=['red'])
7  plt.contour(x1, x2, x1 + np.sin(x2) - 0.6, [0], colors=['blue'])
8  plt.savefig('nonlinear_newton.png', dpi=300)
9  plt.show()
```

1.2.2 Решение системы методом Ньютона

```

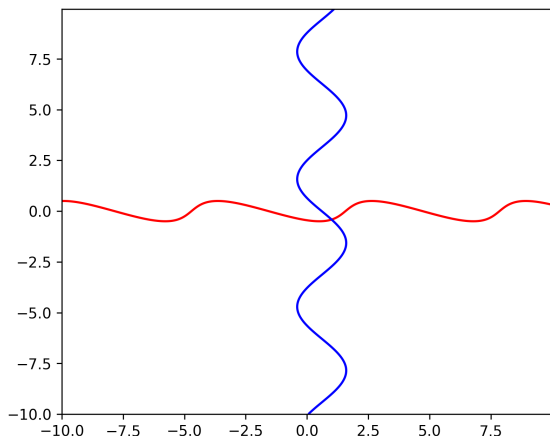
1  import numpy as np
2  import sympy as sp
3  import scipy
4  import matplotlib.pyplot as plt
5
6  x1 = sp.Symbol('x1')
7  x2 = sp.Symbol('x2')
8
9  f1 = sp.cos(x1 + x2) + 2 * x2
10 f2 = x1 + sp.sin(x2) - 0.6
11
```

```

12 F = sp.Matrix([f1, f2])
13
14
15 def eq(vars):
16     x1, x2 = vars
17     f1 = sp.cos(x1 + x2) + 2 * x2
18     f2 = x1 + sp.sin(x2) - 0.6
19     return [f1, f2]
20
21
22 def calculate_root(F, X_num: np.array, eps: float = 0.000001) -> np.array:
23     iter_count = 0
24     jacobian = F.jacobian([x1, x2]).inv() * F
25     while True:
26         jacobian_num = jacobian.subs([(x1, X_num[0]), (x2, X_num[1])])
27         X_num -= np.array(jacobian_num, dtype=float).flatten()
28         iter_count += 1
29         norm = np.linalg.norm(np.array(F.subs([(x1, X_num[0]), (x2, X_num[1])]), dtype=float).flatten())
30         if norm < eps:
31             break
32     return X_num, iter_count
33
34
35 X1 = np.array([2.5, 0.1])
36 X2 = np.array([-4.5, -0.1])
37
38 X1_my, count1 = calculate_root(F, X1, 0.000001)
39 X2_my, count2 = calculate_root(F, X2, 0.000001)
40
41
42 X1_scipy = scipy.optimize.fsolve(eq, X1)
43 X2_scipy = scipy.optimize.fsolve(eq, X2)
44
45 print(f'{X1_my = }')
46 print(f'Количество итераций: {count1}')
47 print()
48
49 print(f'{X1_my = }')
50 print(f'Количество итераций: {count1}')
51 print()
52
53 print(f'{X1_scipy = }')
54 print(f'{X2_scipy = }')
55
56 plt.plot(*X1_my, marker='o', color='green', ls='')
57 x1, x2 = np.meshgrid(np.arange(-10, 10, 0.05), np.arange(-10, 10, 0.05))
58 plt.contour(x1, x2, np.cos(x1 + x2) + 2 * x2, [0], colors=['red'])
59 plt.contour(x1, x2, x1 + np.sin(x2) - 0.6, [0], colors=['blue'])
60 plt.savefig('nonlinear_newton_with_point.png', dpi=300)
61 plt.show()

```


1.3 Результаты



Видно, что корень находится в пределах (0, 2.5) по оси x, и (-1, 1) по оси y.

Результат работы программы:

```
X1_my = array([ 1.00410184, -0.41599671])
```

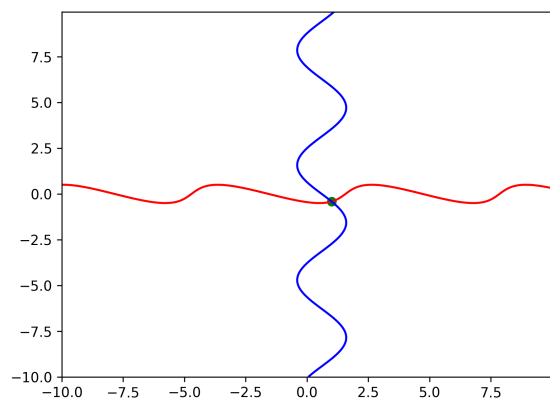
Количество итераций: 4

```
X1_my = array([ 1.00410184, -0.41599671])
```

Количество итераций: 4

```
X1_scipy = array([ 1.00410184, -0.41599671])
```

```
X2_scipy = array([ 1.00410184, -0.41599671])
```



2 Расстояния от поверхности до точки (№ 4.5.2)

2.1 Формулировка задания

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.

2.2 Код на Python

3 Метод Зейделя (№ 5.1.8)

3.1 Формулировка задания

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.

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. Проверить выполнение достаточного условия сходимости итерационных методов $\|B\|_{\infty} < \infty$
3. Написать программу-функцию `zeid`, решающую систему уравнений с помощью метода Зейделя, выполнить 10 итераций по методу Зейделя; взять любое начальное приближение. Принимая решение, полученное в п. 1 за точное, найти величину абсолютной погрешности итерационного решения.
4. Взять другое начальное приближение. Объяснить полученные результаты.

3.2 Код на Python

```

1  import numpy as np
2
3
4  def zeid(A, b, x0, n):
5      B = np.empty(A.shape, dtype=float)
6      for i in range(A.shape[0]):
7          for j in range(A.shape[1]):
8              B[i, j] = - A[i, j] / A[i, i] if i != j else 0
9
10     c = np.empty(b.shape, dtype=float)
11     for i in range(c.shape[0]):
12         c[i] = b[i] / A[i, i]
13
14     # B1 = np.zeros(B.shape)
15     # B2 = np.zeros(B.shape)
16     # for i in range(A.shape[0]):
17     #     for j in range(A.shape[1]):
18     #         if j < i:
19     #             B1[i, j] = B[i, j]
20     #         if j > i:
21     #             B2[i, j] = B[i, j]
22
23     x = x0
24     for _ in range(n):
25         x_new = np.zeros(x.shape, dtype=float)

```

```

26     for i in range(B.shape[0]): # x_new = B1 @ x_new + B2 @ x + c
27         x_new[i] = np.sum(B[i][:i] * x_new[:i]) + np.sum(B[i][i:] * x[i:]) + c[i]
28     x = x_new
29     return x, B
30
31
32
33
34
35 A = np.array([[118.8, -14, -5, -89.1],
36               [-59.4, 194, 5, 128.7],
37               [148.5, 12, -310, 148.5],
38               [0, 18.5, 90, -108.9]])
39
40 b = np.array([92.5, -340.1, -898, 184.1])
41
42 x_true = np.linalg.solve(A, b)
43
44 x1 = np.zeros(b.shape[0])
45 x2 = np.ones(b.shape[0])
46
47 x1_zaid, B = zeid(A, b, x1, 10)
48 x2_zaid, _ = zeid(A, b, x2, 10)
49
50 error1_abs = np.linalg.norm(x_true - x1_zaid, ord=np.inf)
51 error2_abs = np.linalg.norm(x_true - x2_zaid, ord=np.inf)
52
53 print(f'Решение методом Гаусса: {x_true}')
54 print()
55
56 print(f'Начальная точка: {x1}')
57 print(f'Решение методом Зейделя: {x1_zaid}')
58 print(f'Абсолютная погрешность: {error1_abs}')
59 print()
60
61 print(f'Начальная точка: {x2}')
62 print(f'Решение методом Зейделя: {x2_zaid}')
63 print(f'Абсолютная погрешность: {error2_abs}')
64
65 print(f'Норма матрицы B: {np.linalg.norm(B, ord=np.inf)}')
```

3.3 Результаты

Решение методом Гаусса: [1.90875981 -2.36310552 4.49843079 1.62572378]

Начальная точка: [0. 0. 0. 0.]

Решение методом Зейделя: [1.90048704 -2.35749698 4.48910538 1.61896961]

Абсолютная погрешность: 0.00932540991025288

Начальная точка: [1. 1. 1. 1.]

Решение методом Зейделя: [1.90673816 -2.36173495 4.49615191 1.62407324]

Абсолютная погрешность: 0.0022788803781175204

Норма матрицы B: 0.996774193548387

4 Метод релаксации (№ 5.5.1)

4.1 Формулировка задания

Дана система уравнений $Ax = b$, где A – симметричная положительно определенная матрица. Найти решение системы с точностью $\varepsilon = 10^{-5}$ с помощью метода релаксации (для этого модифицировать функцию `zeid` из задачи 5.2, реализующую метод Зейделя). Определить экспериментально параметр релаксации ω , при котором точность ε достигается при наименьшем числе итераций. Построить график зависимости числа итераций от параметра релаксации.

4.2 Код на Python

```

1  import numpy as np
2
3
4  def relax(A, b, x0, eps, w):
5      B = np.empty(A.shape, dtype=float)
6      for i in range(A.shape[0]):
7          for j in range(A.shape[1]):
8              B[i, j] = - A[i, j] / A[i, i] if i != j else 0
9
10     c = np.empty(b.shape, dtype=float)
11     for i in range(c.shape[0]):
12         c[i] = b[i] / A[i, i]
13
14     # B1 = np.zeros(B.shape)
15     # B2 = np.zeros(B.shape)
16     # for i in range(A.shape[0]):
17     #     for j in range(A.shape[1]):
18     #         if j < i:
19     #             B1[i, j] = B[i, j]
20     #         if j > i:
21     #             B2[i, j] = B[i, j]
22
23     x = x0
24     while True:
25         x_new = np.zeros(x.shape, dtype=float)
26         for i in range(B.shape[0]): # x_new = B1 @ x_new + B2 @ x + c
27             x_new[i] = np.sum(B[i][:i] * x_new[:i]) + np.sum(B[i][i:] * x[i:]) + c[i]
28             x_new[i] = w * x_new[i] + (1 - w) * x[i]
29         norma = np.linalg.norm(x_new - x)
30         x = x_new
31         if norma < eps:

```

```
32         break
33     return x
34
35
36 A = np.array([[118.8, -14, -5, -89.1],
37              [-59.4, 194, 5, 128.7],
38              [148.5, 12, -310, 148.5],
39              [0, 18.5, 90, -108.9]])
40
41 b = np.array([92.5, -340.1, -898, 184.1])
42
43 x_true = np.linalg.solve(A, b)
44 x_relax = relax(A, b, np.zeros(b.shape[0]), 0.00001, 1.25)
45
46 print(f'Решение методом Гаусса: {x_true}')
47 print(f'Решение системы методом релаксации: {x_relax}')
```

4.3 Результаты