ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

ПОЛИТЕХ
Санкт-Петербургский
политехнический университет
Петра Великого

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине «Элементы теории вероятности и линейной алгебры»

Вариант №11

Выполнил
Студент 3540201/10301 группы Ф.М. Титов

Руководитель
доцент, к.т.н. А.В. Востров

Санкт-Петербург
2021 г.

Оглавление

Постановка задачи 3
Теоретическая часть 4
Реализация 6
Результаты 8
Заключение 9

Постановка задачи

Методом простых итераций с точностью $\varepsilon = 10^{-7}$ решить систему линейных алгебраических уравнений, заданную в форме $A \cdot x = b$.

Значения для варианта 11:

№ варианта	Матрица A				Вектор правой части $\bar{b}$
11	0.58	0.32	-0.03	0.00	0.4400
	-0.11	1.26	0.36	0.00	1.4200
	-0.12	-0.08	1.14	0.24	-0.8300
	-0.15	0.35	0.18	1.00	-1.4200

Теоретическая часть

Метод простых итераций используется для решения разреженных систем большой размерности ($\sim 10^4 \div 10^6$), причем матрица такой системы помимо разреженности должна быть близкой к диагональной. Метод сходится тем быстрее, чем меньше норма матрицы коэффициентов B, при этом для сходимости метода необходимо $\|B\| < 1$.

В нашем случае будет использована Евклидова норма матрицы:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Основная формула метода простых итераций:

$$\overline{x}^{(k+1)} = B\overline{x}^{(k)} + \overline{c}, \quad k = 0,1,2\ldots$$

В нашем случае СЛАУ задана в традиционной форме:

$$A \cdot \overline{x} = \overline{b}$$

Для начала её необходимо привести к виду основной формулы метода простых итераций с помощью метода Якоби. Псевдокод метода Якоби представлен на рис. 1.

```
itera(A, n) := | for i ∈ 1..n
               |   | α ← 1 / A[i,i]
               |   | for j ∈ 1..n
               |   |   | B[i,j] ← -A[i,j]·α
               |   |   | B[i,j] ← 0 if i=j
               | B

Jakobi(A, b, ε) := | n ← rows(A)
                   | B ← itera(A, n)
                   | for i ∈ 1..n
                   |   | c[i] ← b[i] / A[i,i]
                   |   | x[i] ← 0
                   |   | x1[i] ← c[i]
                   | norm ← norm1(B)
                   | return norm if norm ≥ 1
                   | while norm > ε
                   |   | for i ∈ 1..n
                   |   |     x[i] ← c[i] + [ Σ_{j=1}^{n} (B[i,j]·x1[j]) ]
                   |   | norm ← √(x1·x1)
                   |   | x1 ← x - x1
                   |   | norm ← √(x1·x1) / norm
                   |   | x1 ← x
                   | x
```

Рис.1. Псевдокод метода Якоби.

Метод Зейделя является модификацией метода Якоби. Мы сравним результаты данного метода с результатами метода Якоби для того, чтобы убедиться в верности решения.

Основная идея модификации состоит в том, что новые значения $x_{i,j}$ используются здесь сразу же по мере получения, в то время как в методе Якоби они не используются до следующей итерации.

Расчетная формула метода Зейделя имеет вид:

$$\overline{x}^{(k+1)} = B_1 \overline{x}^{(k+1)} + B_2 \overline{x}^{(k)} + \overline{c}.$$

Реализация

Приведенный алгоритм на рис.1 был разбит на 3 вспомогательные функции:

1) Функция вычисления первой нормы матрицы get_norm. Она была реализована с использованием встроенного пакета Python numpy и функцией linalg.norm, взятой из этого пакета. Листинг функции представлен на рис. 2.
2) Функция вычисления матрицы *c* (рис. 3).
3) Функция вычисления матрицы *B* (рис. 4).

```
def get_norm(matrix):
    return np.linalg.norm(matrix)
```

Рис.2. Листинг функции нахождения нормы матрицы.

```
def get_c(A, b):
    return [b[i] / A[i][i] for i in range(len(b))]
```

Рис.3. Листинг функции матрицы с.

```
def get_B(A):
    rows = len(A)
    columns = len(A[0])
    B = np.zeros((rows, columns))
    for i in range(len(A)):
        for j in range(len(A[0])):
            if i != j:
                B[i][j] = - A[i][j] / A[i][i]
    return B
```

Рис.4. Листинг функции матрицы B.

Далее был реализован метод Якоби (рис. 5). На вход функции jacobi_method подаются предварительно полученные матрицы *B* и *c*, а также параметр eps, который задает точность работы метода. В результате возвращается вектор x – решение СЛАУ.

```python
def jacobi_method(B, c, eps):
    x1 = c.copy()
    x = np.zeros(len(c)).tolist()

    norm = get_norm(b)
    iterations = 0
    while norm > eps:
        iterations += 1

        x = [c[i] + np.dot(B[i], x1) for i in range(len(c))]

        norm = math.sqrt(np.dot(x1, x1))

        x1 = [x[k] - x1[k] for k in range(len(x1))]

        norm = math.sqrt(np.dot(x1, x1)) / norm

        x1 = x[:]

    print("Amount of iterations: {}".format(iterations))
    return x
```

Рис.5. Листинг функции метода Якоби.

Реализация метода Зейделя представлена на рис. 6. На вход функция принимает параметры *A*, *b*, *eps*.

```python
def seidel(A, b, eps):
    n = len(A)
    x = np.zeros(n)
    iterations = 0

    converge = False
    while not converge:
        iterations += 1

        x_new = np.copy(x)
        for i in range(n):
            s1 = sum([A[i][j] * x_new[j] for j in range(i)])
            s2 = sum([A[i][j] * x[j] for j in range(i + 1, n)])
            x_new[i] = (b[i] - s1 - s2) / A[i][i]

        converge = np.sqrt(sum([(x_new[i] - x[i]) ** 2 for i in range(n)])) <= eps
        x = x_new

    print("Amount of iterations: {}".format(iterations))

    return x
```

Рис.6. Листинг функции метода Зейделя.

Для поиска собственных чисел матрицы был использован метод np.linalg.eigh, взятый из библиотеки numpy.

Результаты

Исходные матрицы A и b были преобразованы (рис.7). В результате были получены матрицы B и c.

```
A:[[ 0.58  0.32 -0.03  0.  ]
 [-0.11  1.26  0.36  0.  ]
 [-0.12 -0.08  1.14  0.24]
 [-0.15  0.35  0.18  1.  ]]
b:[0.44, 1.42, -0.83, -1.42]
B:[[ 0.         -0.55172414  0.05172414 -0.        ]
 [ 0.08730159  0.         -0.28571429 -0.        ]
 [ 0.10526316  0.07017544  0.         -0.21052632]
 [ 0.15       -0.35       -0.18        0.        ]]
c:[0.7586206896551725, 1.126984126984127, -0.7280701754385965, -1.42]
```

Рис.7. Преобразование матриц A и b.

Далее они были поданы на вход функции jacobi_method. В результате за 14 итераций было получено решение – вектор x (рис.8).

```
Amount of iterations: 14
Jacobi: [0.07874157173548091, 1.2079647516695349, -0.2593722563107691, -1.7842894194264471]
```

Рис.8. Полученный результат для метода Якоби.

Далее был протестированы метод Зейделя и встроенная функция решения СЛАУ numpy.linalg.solve (рис. 9). Метод Зейделя нашел решение за 13 итераций.

```
Amount of iterations: 13
Seidel: [ 0.07874157  1.20796475 -0.25937225 -1.78428942]
np.linalg.solve: [ 0.07874157  1.20796475 -0.25937225 -1.78428942]
```

Рис.9. Тестирование метода Зейделя и встроенного метода.

Собственные числа матрицы А, которые были найдены в результате работы программы, представлены на рис. 10. Как можно заметить, исходная матрица положительно определена.

```
w:  [0.52394623 0.67667901 1.23264624 1.54672852]
```

Рис.10. Собственные числа матрицы А.

Заключение

В ходе работы были реализованы алгоритмы решения СЛАУ методами Якоби и Зейделя. Далее они были протестированы на заданном наборе значений A и b, и далее были сравнены с встроенным методом решения СЛАУ. В результате все значения, полученные с помощью всех трех методов, совпали до 7 знаков после запятой. Метод Якоби нашел решение за 14 итераций, метод Зейделя – 13. Метод Зейделя в общем случае сходится быстрее метода Якоби, когда исходная матрица симметрична и положительно определена. Дело в том, что эти методы используются для разных классов задач. Метод Якоби рассчитан на системы с матрицами, близкими к диагональным, а метод Зейделя - на системы с матрицами, близкими к нижним треугольным. В нашем случае более быстрая сходимость метода Зейделя обусловлена положительно определенной матрицей, что можно наблюдать из найденных собственных чисел.