МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет» (ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ОТЧЕТ по лабораторной работе № 4

по дисциплине

«Вычислительная математика»

Направление подготовки
02.03.01 «Математика и компьютерные науки»

Выполнил(а): студент гр. Б9122-02.03.01сцт
Бекболот Отгонцэцэг
Проверил: преподаватель
_____________ Ф.И.О.

Владивосток
2024

Цель работы :

Решить систему линейных алгебраических уравнений в виде $Ax = b$ с помощью QR-разложения.

Ход работы:

1. Находим матрицы Q и R по любому из трех методов ортогонализации. Матрицы Q и R будут иметь следующий вид:
   Q = [q1|q2|...|qn]
   R = ((a1, q1) (a2, q1) (a3, q1)
   (a2, q2) (a3, q2)
   0 0 (a3, q3))
2. Решить СЛАУ в два этапа:
   - Находим вектор значений "y" из системы $y = Q^T b$.
   - Вычислив массив "y" решаем СЛАУ вида $Rx = y$.
   - Полученный массив "x" будет являться решением исходной системы $Ax = b$.
3. Сравнить полученные результаты с точным решением x^* тестовых СЛАУ из таблицы 1:

Таблица 1. Тесты.

№	Матрица $\boldsymbol{A}$	Столбец $\boldsymbol{b}$	Точное решение $\boldsymbol{x}^*$
1	$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 6 & 7 \\ 8 & 9 & 0 \end{pmatrix}$	$b = \begin{pmatrix} 6 \\ 12 \\ 24 \end{pmatrix}$	$x^* = \begin{pmatrix} -11.538 \\ 12.923 \\ -2{,}769 \end{pmatrix}$
2	$A = \begin{pmatrix} 6.03 & 13 & -17 \\ 13 & 29.03 & -38 \\ -17 & -38 & 50.03 \end{pmatrix}$	$b = \begin{pmatrix} 2.0909 \\ 4.1509 \\ -5.1191 \end{pmatrix}$	$x^* = \begin{pmatrix} 1.03 \\ 1.03 \\ 1.03 \end{pmatrix}$

4. После отладки программы решить следующую систему:

$$\begin{cases} 2x_1 + x_3 = 3 \\ x_2 - x_3 = 0 \\ x_1 + x_2 + x_3 = 3 \end{cases}$$

5. Вывести полученное решение СЛАУ.

Код программы:

```
import numpy as np

def qr_decomposition(A: np.ndarray) -> tuple[np.ndarray, np.ndarray]:

    n = A.shape[0]
    Q = np.eye(n)
    R = A.copy()

    for k in range(n - 1):

        p = np.zeros(n)
        a_kk = R[k, k]

        if a_kk != 0:
            norm_a = np.sqrt(np.sum(R[k:, k] ** 2))
            p[k] = a_kk + (1 if a_kk >= 0 else -1) * norm_a
        else:
            p[k] = np.sqrt(2)

        p[k + 1:] = R[k + 1:, k]

        P = np.eye(n) - 2 * np.outer(p, p) / np.dot(p, p)

        Q = Q @ P
        R = P @ R

        print(f"Итерация {k + 1}:")
        print(f"p = \n{p}")
        print(f"P = \n{P}")
        print(f"R после обновления = \n{R}\n")

    return Q, R

def solve_system(A: np.ndarray, b: np.ndarray) -> np.ndarray:

    Q, R = qr_decomposition(A)
    y = Q.T @ b
    x = np.linalg.solve(R, y)
    return x

A1 = np.array([[1, 2, 3],
               [4, 6, 7],
               [8, 9, 0]])
b1 = np.array([6, 12, 24])
x1_exact = np.array([-11.538, 12.923, -2.769])
```


```
b1 = np.array([6, 12, 24])
x1_exact = np.array([-11.538, 12.923, -2.769])
```


```

Q1, R1 = qr_decomposition(A1)
print("A1 = \n", A1)
print("Q1 = \n", Q1)
print("R1 = \n", R1)
print("Проверка: Q1 @ R1 = \n", Q1 @ R1)

x1_r = solve_system(A1, b1)
print("Решение x1_r = \n", x1_r)
print("Разность x1_r - x1 = \n", x1_r - x1_exact)
print("-----------------------------------")

# Пример 2
A2 = np.array([[6.03, 13, -17],
               [13, 29.03, -38],
               [-17, -38, 50.03]])
b2 = np.array([2.0909, 4.1509, -5.1191])
x2_exact = np.array([1.03, 1.03, 1.03])

Q2, R2 = qr_decomposition(A2)
print("A2 = \n", A2)
print("Q2 = \n", Q2)
print("R2 = \n", R2)
print("Проверка: Q2 @ R2 = \n", Q2 @ R2)

x2_r = solve_system(A2, b2)
print("Решение x2_r = \n", x2_r)
print("Разность x2_r - x2 = \n", x2_r - x2_exact)
print("-----------------------------------")


A3 = np.array([[2, 0, 1],
               [0, 1, -1],
               [1, 1, 1]])
b3 = np.array([3, 0, 3])

x3_r = solve_system(A3, b3)
print("Решение x3_r = \n", x3_r)
```

Описание:

Эта программа реализует метод QR-разложения для решения систем линейных алгебраических уравнений (СЛАУ) вида $Ax = b$ с использованием ортогонализации Грама-Шмидта. Метод QR-разложения позволяет нам разложить матрицу A на произведение двух матриц: ортогональной матрицы Q и верхней треугольной матрицы R. Затем, зная матрицы Q и R, мы можем эффективно решать систему уравнений, используя свойства этих матриц.

Основные шаги программы:

1. **Функция QR-разложения**:
def qr_decomposition(A: np.ndarray) -> tuple[np.ndarray, np.ndarray]:

Эта функция принимает матрицу A и возвращает два значения: ортогональную матрицу Q и верхнюю треугольную матрицу R. Внутри функции:

- **Инициализация**: Создаются пустые массивы Q и копия матрицы R.
- **Ортогонализация**: Для каждого столбца матрицы A выполняется ортогонализация с использованием метода Грама-Шмидта.
- **Печать отладочной информации**: Показываются промежуточные результаты, такие как вектор p и матрица R после каждой итерации.

2. **Решение системы уравнений:**
def solve_system(A: np.ndarray, b: np.ndarray) -> np.ndarray:

Эта функция принимает матрицу коэффициентов A и вектор свободных членов b. Она использует QR-разложение для нахождения решения системы $Ax = b$:

- Вычисляет матрицы Q и R.
- Вычисляет вектор $y = Q^T b$.
- Решает верхнетреугольную систему $Rx = y$ с помощью функции np.linalg.solve.

3. **Тестовые примеры:**
- Пример 1:

A1 = np.array([[1, 2, 3], [4, 6, 7], [8, 9, 0]])   b1 = np.array([6, 12, 24]) x1_exact = np.array([-11.538, 12.923, -2.769])

Для первого примера решается система уравнений, показываются матрицы A, Q, R и разность между вычисленным и точным решением.

- Пример 2:

Аналогично первому пример, новый набор данных используется для проверки работы программы.

- Пример 3:

Третий пример демонстрирует, как программа может работать с другой системой.

4. **Функция** solve_system успешно решает системы линейных уравнений что подтверждается сравнением найденных решений с известными точными значениями. Разности между вычисленными и известными решениями близки к нулю, что свидетельствует о высокой точности реализованного метода QR-разложения.

5. **Вывод результатов**:

Для каждого примера выводятся результаты его выполнения, включая саму матрицу *A*, найденные матрицы Q и R, а также разность между найденным решением и точно известным.

```
A1 =
 [[1 2 3]
 [4 6 7]
 [8 9 0]]
Q1 =
 [[-0.11111111 -0.50664569 -0.8549646 ]
 [-0.44444444 -0.74413586  0.49872935]
 [-0.88888889  0.43539864 -0.1424941 ]]
R1 =
 [[-9.00000000e+00 -1.08888889e+01 -3.44444444e+00]
 [ 2.40464351e-16 -1.55951876e+00 -6.72888805e+00]
 [-3.73352544e-16 -3.16189972e-17  9.26211650e-01]]
Проверка: Q1 @ R1 =
 [[ 1.00000000e+00  2.00000000e+00  3.00000000e+00]
 [ 4.00000000e+00  6.00000000e+00  7.00000000e+00]
 [ 8.00000000e+00  9.00000000e+00 -6.52102286e-16]]
Итерация 1:
```

```
Решение x1_r =
 [-11.53846154  12.92307692  -2.76923077]
Разность x1_r - x1 =
 [-4.61538462e-04  7.69230769e-05 -2.30769231e-04]
------------------------------
```

```
A2 =
 [[  6.03  13.    -17.  ]
 [ 13.    29.03 -38.  ]
 [-17.   -38.    50.03]]
Q2 =
 [[-0.27120348  0.96131169 -0.04825465]
 [-0.58468412 -0.12471144  0.80161808]
 [ 0.76458692  0.24561534  0.59588585]]
R2 =
 [[-2.22342281e+01 -4.95533281e+01  6.50807391e+01]
 [ 2.25006454e-15 -4.56704032e-01  6.84871421e-01]
 [-1.36319944e-16 -4.73166675e-17  1.71011233e-01]]
Проверка: Q2 @ R2 =
 [[  6.03  13.    -17.  ]
 [ 13.    29.03 -38.  ]
 [-17.   -38.    50.03]]
```

```
Решение x2_r =
 [1.03 1.03 1.03]
Разность x2_r - x2 =
 [ 5.10702591e-15 -2.37587727e-14 -1.62092562e-14]
```

Система из задания лабораторной работы:

```
Решение x3_r =
 [1. 1. 1.]
```

Вывод:

Программа наглядно демонстрирует метод QR-разложения для решения систем линейных уравнений с использованием ортогонализации Грама-Шмидта. Она показывает, как легко и эффективно можно решать СЛАУ, минимизируя ошибки численных расчетов. Поскольку QR-разложение является одним из наиболее устойчивых методов, программа может успешно использоваться для решения сложных систем с различными характеристиками.

Код можно легко адаптировать для работы с другими матрицами и системами уравнений, что делает его универсальным инструментом для решения линейных задач в различных областях.

Тем не менее, в коде стоит рассмотреть возможность оптимизации вычислений, особенно в частях, где определяется вектор нормали и обновляются матрицы. Это будет особенно полезно при работе с большими матрицами, где время выполнения и эффективность алгоритмов становятся критическими.