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

ОТЧЁТ по лабораторной работе № 4

«QR-разложение Грама-Шмидта»

Вариант № 8

Выполнила: студент гр. Б9122-02.03.01сцт

Ф.И.О.

Ильяхова Алиса Алексеевна

Проверил: преподаватель

Ф.И.О.

Павленко Елизавета Робертовна

Владивосток

2024

Цели 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:
   $Q = [q_1|q_2|\dots|q_n]$
   $R = ((a_1, q_1)\ (a_2, q_1)\ (a_3, q_1)$
   $\quad 0\ (a_2, q_2)\ (a_3, q_2)$
   $\quad 0\ 0\ (a_3, q_3))$
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 *A*	Столбец *b*	Точное решение 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 и вывести его:

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

Основное:

1) Функции:

Функция **qr(A_orig: np.matrix) -> tuple[np.matrix, np.matrix]**:

- **Назначение**: реализует QR-разложение матрицы A с использованием метода отражений.
- **Параметры**: A_orig: исходная матрица, которую необходимо разложить на произведение матриц Q и R.
- **Возвращаемые значения**: кортеж (Q, R), где:
  Q - ортогональная матрица.
  R - верхняя треугольная матрица.
- **Алгоритм**:
  1. Инициализирует матрицу Q как единичную.

2. Для каждого столбца матрицы A вычисляет нормаль и обновляет матрицы Q и A с помощью отражения.
3. В конце вычисляет R как произведение транспонированной матрицы Q и исходной матрицы A.

Функция **calc_solve(A: np.matrix, b: np.matrix) -> np.matrix**:

- **Назначение**: решает систему линейных уравнений Ax = b с использованием QR-разложения.
- **Параметры**:
  A: Матрица коэффициентов.
  b: Вектор правых частей.
- **Возвращаемое значение**: вектор решений x.
- **Алгоритм**:
  1. Вызывает функцию qr для получения матриц Q и R.
  2. Вычисляет промежуточный вектор $y = Q^T * b$.
  3. Решает систему уравнений R * x = y с помощью функции np.linalg.solve.

2) Результаты:

Тестовые СЛАУ:

```
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]
 [ 0.00000000e+00 -1.55951876e+00 -6.72888805e+00]
 [-2.22044605e-16  2.49800181e-16  9.26211650e-01]]
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 7.85678349e-16]]
```

```
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]
 [-1.38777878e-16 -4.56704032e-01  6.84871421e-01]
 [ 1.88737914e-15  6.66133815e-16  1.71011233e-01]]
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 =
 [[ 7.54951657e-15]
 [-3.44169138e-14]
 [-2.37587727e-14]]
```

Оценка точности рассчитывается как евклидова норма разности между x и x*.

Для каждого примера код корректно вычисляет матрицы Q и R. Проверка **Q @ R** подтверждает, что произведение дает исходную матрицу A.

Система из задания лабораторной работы:

```
Решение x3_r =
 [[1.]
 [1.]
 [1.]]
```

3) Вывод:

Функция **calc_solve** успешно решает системы уравнений, что подтверждается выводом разности между найденным решением и известным решением. Разности близки к нулю, что указывает на высокую точность метода.

Код может быть легко адаптирован для работы с другими матрицами и системами уравнений, что делает его полезным инструментом для решения линейных задач.

В коде можно рассмотреть возможность оптимизации вычислений, особенно в части, где определяется нормаль и обновляются матрицы. Это может быть полезно для больших матриц.

Метод QR-разложения является эффективным инструментом для решения систем линейных уравнений, особенно когда матрица A обладает хорошими численными свойствами.

4) Листинг программы:

```python
import numpy as np
import warnings

warnings.simplefilter(action="ignore", category=DeprecationWarning)
debug_print = False

def qr(A_orig: np.matrix) -> tuple[np.matrix, np.matrix]:
    n = A_orig.shape[0]
    Q = np.eye(n)
    A = A_orig.copy()

    for k in range(n - 1):
        #вычисление нормали
        p = np.zeros((n, 1))
        a_kk = A[k, k]

        if a_kk != 0:
            a_kk_n = (1 if a_kk >= 0 else -1) * np.sqrt(sum(elem **
2 for elem in A[k:, k]))
        else:
            a_kk_n = np.sqrt(2)
```

```
        p[k] = a_kk + a_kk_n
        p[k + 1:] = A[k + 1:, k]

        P = np.eye(n) - 2 / float(sum(p[l] * p[l] for l in
range(n))) * p * p.T
        Q = Q @ P
        A = P @ A

        if debug_print:
            print(f"p_{k + 1} = ", p)
            print(f"P_{k + 1} = ", P)
            print(f"P_{k + 1} @ A_{k - 1 + 1} = ", A)
            print()

    R = Q.T @ A_orig
    return Q, R

def calc_solve(A: np.matrix, b: np.matrix) -> np.matrix:
    Q, R = qr(A)
    y = Q.T @ b
    x = np.linalg.solve(R, y)
    return x

A1 = np.matrix("1 2 3; 4 6 7; 8 9 0")
b1 = np.matrix("6; 12; 24")
x1 = np.matrix("-11.538; 12.923; -2.769")

Q1, R1 = qr(A1)
print("A1 = \n", A1)
print("Q1 = \n", Q1)
print("R1 = \n", R1)
print("Q1 @ R1 = \n", Q1 @ R1)

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

A2 = np.matrix("6.03 13 -17; 13 29.03 -38; -17 -38 50.03")
b2 = np.matrix("2.0909; 4.1509; -5.1191")
x2 = np.matrix("1.03; 1.03; 1.03")
```

```python
Q2, R2 = qr(A2)
print("A2 = \n", A2)
print("Q2 = \n", Q2)
print("R2 = \n", R2)
print("Q2 @ R2 = \n", Q2 @ R2)

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

A3 = np.matrix("2 0 1; 0 1 -1; 1 1 1")
b3 = np.matrix("3; 0; 3")

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