МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра вычислительной математики

Отчёт по лабораторной работе №2, вариант 12

(«Методы вычислений»)

Жуковский Павел Сергеевич

Студента 2 курса 13 группы

Преподаватель

Бондарь Иван Васильевич

Минск 2020

Условие:

Задание 12. Метод наименьших квадратов (QR)

1. Написать программу, которая решает задачи наименьших квадратов вида

$$\|A_k x = b\| \to \min$$

методом QR-разложения с использованием преобразований отражения. Здесь A_k — матрица, составленная из первых k столбцов матрицы A, $k = 1, 2, \ldots, n$. При реализации на k-м шаге следует использовать результаты вычислений на предыдущих шагах.

$$A = \begin{pmatrix} 5 & 3 & 3 & -4 & 5 & -5 & -4 & -5 & 0 & 2 \\ 5 & 3 & -1 & 2 & 3 & 0 & -4 & 1 & -4 & -5 \\ 10 & 6 & 2 & -3 & -2 & -2 & -1 & 0 & -3 & -5 \\ 20 & 12 & 4 & -5 & -1 & -4 & 4 & -2 & -2 & -4 \\ 40 & 24 & 8 & -10 & 5 & -1 & 5 & 2 & 0 & -3 \\ 80 & 48 & 16 & -20 & 10 & -12 & -3 & -3 & 3 & 2 \\ 160 & 96 & 32 & -40 & 20 & -24 & -3 & -4 & 1 & 4 \\ 320 & 192 & 64 & -80 & 40 & -48 & -6 & -11 & 0 & -3 \\ 640 & 384 & 128 & -160 & 80 & -96 & -12 & -22 & -5 & 2 \\ -3 & -3 & 0 & 0 & 5 & 3 & -2 & 2 & 5 & -2 \end{pmatrix} \quad b = \begin{pmatrix} -18 \\ 2 \\ -8 \\ -6 \\ 24 \\ 30 \\ 64 \\ 100 \\ 216 \\ 0 \end{pmatrix}.$$

2. Построить график нормы невязки в зависимости от k.

Задание 1:

Для решения задачи наименьших квадратов вида $||A_k x = b|| \to \min$ я написал реализацию на языке Python (там же и нарисовал график). Моя программа действовала по следующему алгоритму:

```
"""
АЛГОРИТМ, ПО КОТОРОМУ ДЕЙСТВУЕТ ПРОГРАММА (объяснено для N = 10)
0) На этом шаге просто создаю два массива: 1) Первый будет хранить 10 значений
невязки, 2) Второй массив просто будет
хранить числа 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (это наши k), чтобы потом по этим
двум массива построить график невязки от k
1) Записываем все данные из матрицы A в матрицу ReflectionMatrix такой же
размерности (чтобы не испортить матрицу A)
2) Записываем вектор b в вектор ReflectionVectorB такой же размерности (чтобы не
испортить наш вектор b)
3) Далее, входим в цикл, где будет N итераций, где k = 0, 1, 2, ..., N - 3, N -
2, N - 1, а размерности матрицы
Ak будет соответственно {N}строк x {1, 2, 3, ..., N - 2, N - 1, N}столбцов
4) На каждом шаге мы берём в качестве вектора U нижний кусочек k-ого столбца
матрицы (на первой итерации вектор
U будет состоять из одного числа и девяти нулей, на второй итерации будет
состоять из двух чисел и восьми нулей, и т.д.)
5) Теперь нам нужно посчитать самый первый вектор U' текущей матрицы Ak, для
этого мы создаем нулевой вектор-столбец
такой же размерности, как обычный U, а потом вместо самого первого его числа
(самого верхнего в столбце) вписываем
норму от вектора U, и получится наш U'
6) Насчитанный нами U' мы вписываем в соответствующий кусочек соответствующего
столбца в матрице ReflectionMatrix
7) Далее, если итерация не последняя (т.е. k != N - 1), то нам также нужно
посчитать очередной вектор w (омега). Для
этого считаем его с помощью текущих векторов U и U'.
8) Также берём в качестве текущего вектора b соответствующий кусочек нашего
вектора ReflectionVectorB (размера кусочка
зависит от итерации: 1) на первой итерации мы берем вообще все N элементов
исходного вектора b, 2) на второй итерации
мы игнорируем самый верхний элемент столбца и берём 9 нижних (при N = 10). Потом
из вектора b и омеги считаем вектор b'.
```

```
9) Записываем насчитавшийся вектор b' в соответствующий кусочек столбца вектора
ReflectionVectorB.
10) Далее, нам нужно циклически посчитать остальные векторы Uj для текущей
матрицы Ak. На первой итерации это будут
векторы U2, U3, ..., U9, на второй итерации, нам нужно будет посчитать только
U2, U3, ..., U8, а на последней итерации
(где k будет равен N - 1) нам вообще не придется считать следующие векторы Uj.
Так вот, размер каждого из этих векторов
равен N - k, т.е. он зависит от текущей итерации (берем первые N - k элементов
снизу каждого столбца начиная с k-ого),
затем считаем их штрихи, и все насчитанные Uj' вставляем в соответствующие места
нашей матрицы ReflectionMatrix. Это
как если бы мы изначально решали задачу для k = 10, и в 10 этапов считали все
эти U1, U2, ..., U10, потом U1, U2, ...,
U9, потом U1, U2, ..., U8, а в конце вообще только один U1 посчитали и все, ведь
наша матричка все уменьшается и
уменьшается на одну строку сверху и один столбец слева.
11) И наконец из насчитавшейся матрицы (состоящей из кучи векторов Uj' и вектора
b') мы решаем СЛАУ и выводим вектор X,
а затем по формуле и невязку для этого X.
12) Рисуем график зависимости невязки от k (всего в графике будет 10 точек)
"""
```

Собственно, все шаги я также указал в комментариях прямо в коде для удобства. По вашей просьбе я исправил две ошибки, а также упростил и ускорил код в очень многих местах (поменял, где можно, range(…) на выражение вида X:X и даже поменял мой вектор-столбец b на обычный вектор и соответствующие действия с ним). Помимо того, я переименовал переменные таким образом, чтобы они все писались с маленькой буквы через прочерк, в то время как методы и функции я писал с заглавных букв и слитно. Вот такой стиль я выбрал для своего кода. А вот сам код программы (включая часть с рисованием графика):

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

#  Размерность
N = 10

#  Данная матрица A
A = np.array([[5., 3., 3., -4., 5., -5., -4., -5., 0., 2.],
              [5., 3., -1., 2., 3., 0., -4., 1., -4., -5.],
              [10., 6., 2., -3., -2., -2., -1., 0., -3., -5.],
              [20., 12., 4., -5., -1., -4., 4., -2., -2., -4.],
              [40., 24., 8., -10., 5., -1., 5., 2., 0., -3.],
              [80., 48., 16., -20., 10., -12., -3., -3., 3., 2.],
              [160., 96., 32., -40., 20., -24., -3., -4., 1., 4.],
              [320., 192., 64., -80., 40., -48., -6., -11., 0., -3.],
              [640., 384., 128., -160., 80., -96., -12., -22., -5., 2.],
              [-3., -3., 0., 0., 5., 3., -2., 2., 5., -2.]])

#  Данный вектор b
b = np.array([-18., 2., -8., -6., 24., 30., 64., 100., 216., 0.])

#  || Ak*X = b || --> min (Норма невязки)
```

```
#  Посчитать норму невязки для текущего k
def ResidualRate(k, x):
    Ak_x = np.dot(A[:, 0:k], x)
    Ak_x_b = Ak_x - b
    return np.linalg.norm(Ak_x_b)


# Посчитать вектор w (омега)
def GetOmega(u, u_shtrih):
    diff = u - u_shtrih
    return diff / np.linalg.norm(diff)


# Посчитать "штрих" для какого-то объекта: obj' = obj - 2(w, obj)w
def GetShtrih(obj, w):
    number = np.dot(w, obj)
    result = obj - 2*number*w
    return result


# Решение обыкновенного СЛАУ (на вход подается верхнетреугольная матрица, на
выходе вектор X)
def ResolveSLAU(curr_a, size, curr_b):
    x = np.zeros(size)
    for i in range(size - 1, -1, -1):
        for j in range(size - 1, i - 1, -1):
            curr_b[i] -= curr_a[i][j]*x[j]
        x[i] = curr_b[i] / curr_a[i][i]
    return x


# A = QR методом отражений
def QrDecompositionByReflectionMethod(A, N):

    # Шаг 0
    k_arr = np.arange(1, N + 1)  # Массив абсцисс (k) для графика
    nevyazka_arr = np.zeros(N)  # Массив ординат (значение невязки) для графика

    # Шаг 1
    reflection_matrix = np.zeros((N, N))  # Матрица с векторами отражений (мы их
позже будем считать)
    for i in range(N):
        reflection_matrix[:, i] = A[:, i]  # Обновляем текущую матрицу

    # Шаг 2
    reflection_vector_b = np.copy(b)  # Обновляем текущий вектор b

    # Шаг 3
    for k in range(N):  # Для каждой подматрицы (с первыми k столбцами) матрицы
A

        # Шаг 4
        u = np.copy(reflection_matrix[k:, k].transpose())  # За U берём нижний
кусочек k-ого столбца матрицы A

        # Шаг 5
        u_shtrih = np.zeros(N - k)  # Текущий вектор U' (первый U для текущей
матрицы Ak)
        u_shtrih[0] = np.linalg.norm(u)  # U' = (Norm(U1), 0, 0, ..., 0)^T
```

```
        # Шаг 6
        reflection_matrix[k:, k] = u_shtrih  # Записываем текущий U' в вектора
отражений

        # Шаг 7
        if k != N - 1:
            w = GetOmega(u, u_shtrih)  # Омега w

            # Шаг 8
            curr_b = np.copy(reflection_vector_b[k:])  # Кусочек вектора b
            b_shtrih = GetShtrih(curr_b, w)  # b' = b - 2(w, b)w

            # Шаг 9
            reflection_vector_b[k:] = b_shtrih  # Запоминаем вектор b отражения

            # Шаг 10
            for j in range(k + 1, N):  # U2, U3, U4, ..., U9
                u_next = np.copy(reflection_matrix[k:, j].transpose())  #
Считаем следующий Uj
                u_next_shtrih = GetShtrih(u_next, w)  # Считаем следующий U'
                reflection_matrix[k:, j] = u_next_shtrih.transpose()  #
Запоминаем его, он нам понадобится дальше

        # Шаг 11
        X = ResolveSLAU(reflection_matrix, k + 1, np.copy(reflection_vector_b))
# Вектор X из k элементов

        nevyazka_arr[k] = ResidualRate(k + 1, X)

        print("For k =", k + 1, ", X =", X, ", ||Ak*X - b|| = ",
nevyazka_arr[k])

    # Шаг 12
    plt.semilogy(k_arr, nevyazka_arr)
    plt.xlabel("Количество столбцов подматрицы Ak")
    plt.ylabel("Значение невязки")
    plt.show()

print("Program started work...")
QrDecompositionByReflectionMethod(A, N)
print("Program finished work...")
```

Я реализовал свою программу таким образом, чтобы помимо графика она также выводила вектор X для каждой подматрицы A_k. Таким образом в выводе моя программа показала десять векторов X для каждой итерации. По сути, последний вектор – это решение нашей системы (если оно конечно существует), а норма невязки тогда должна быть очень близко к нулю, что мы увидим на графике. После окончания работы программа выдала в консоль следующий вывод:

```
Program started work...
For k = 1 , X = [0.3360926] , ||Ak*X - b|| =  31.029691917696212
For k = 2 , X = [ 0.84024535 -0.84024535] , ||Ak*X - b|| =  31.01330584787212
For k = 3 , X = [ 3.34024535 -3.34024535 -5.        ] , ||Ak*X - b|| =  27.601180040238482
For k = 4 , X = [ 3.06018066 -3.06018066 -1.63891602  2.24072266] , ||Ak*X - b|| =  27.567052486845594
For k = 5 , X = [ 2.28461538 -1.3981685  -8.95164835 -2.45714286  0.53186813] , ||Ak*X - b|| =  27.483141685824002
For k = 6 , X = [-4.78943662  9.51083166  0.57243461  1.1138833   0.77424547  3.43098592] , ||Ak*X - b|| =  19.696285486655373
For k = 7 , X = [-7.59586614  8.48810461 99.73312711 59.58672666 -7.26906637  9.37480315
 -5.4488189 ] , ||Ak*X - b|| =  17.55800286384898
For k = 8 , X = [-11.90694444  10.72987213 142.42967372  83.98465608  -8.32821869
   2.94197531  -5.56790123   9.07407407] , ||Ak*X - b|| =  9.910712498212382
For k = 9 , X = [-13.38055556   7.17032628 229.45723104 134.30582011 -14.32416226
   6.09135802  -9.43209877  11.25925926  -1.33333333] , ||Ak*X - b|| =  9.33333333333246
For k = 10 , X = [ 1.39219227e-13  2.00000000e+00 -2.06498560e-12  2.00000000e+00
  1.15007208e-13  2.00000000e+00  8.50069398e-14  2.00000000e+00
  2.29150032e-14  2.00000000e+00] , ||Ak*X - b|| =  3.414006384903678e-13
Program finished work...
```

Мы можем наблюдать, что в каждой итерации вывелся некоторый вектор X и некоторое значение невязки, не равное нулю. Стоит отметить, что с каждой итерацией значение невязки становилось немножко лучше. Что касается последней итерации, где в качестве подматрицы A_k была взята вся матрица A, то там программа выдала уже вектор X, являющийся решением. Этот факт также подтверждается тем, что невязки в последней итерации была очень близко к нулю. По идее невязка при наличии решения должна быть равна нулю, однако её неравенство в нашем случае можно объяснить погрешностями, что совершает компьютер при вычислениях. Хотя, QR-разложение методом отражений обладает достаточно хорошим числом обусловленности, потому погрешности мы можем наблюдать не очень большие.

Мне стало интересно, и я даже проверил, правильно ли моя программа решила эту систему. Я нашёл точное решение своей системы (ну то есть для k = 10) на электронном ресурсе (я оставлю здесь ссылочку на решение системы с ответом): https://matrixcalc.org/slu.html#solve-using-Gaussian-elimination%28%7B%7B5,3,3,-4,5,-5,-4,-5,0,2,-18%7D,%7B5,3,-1,2,3,0,-4,1,-4,-5,2%7D,%7B10,6,2,-3,-2,-2,-1,0,-3,-5,-8%7D,%7B20,12,4,-5,-1,-4,4,-2,-2,-4,-6%7D,%7B40,24,8,-10,5,-1,5,2,0,-3,24%7D,%7B80,48,16,-20,10,-12,-3,-3,3,2,30%7D,%7B160,96,32,-40,20,-24,-3,-4,1,4,64%7D,%7B320,192,64,-80,40,-48,-6,-11,0,-3,100%7D,%7B640,384,128,-160,80,-96,-12,-22,-5,2,216%7D,%7B-3,-3,0,0,5,3,-2,2,5,-2,0%7D%7D%29

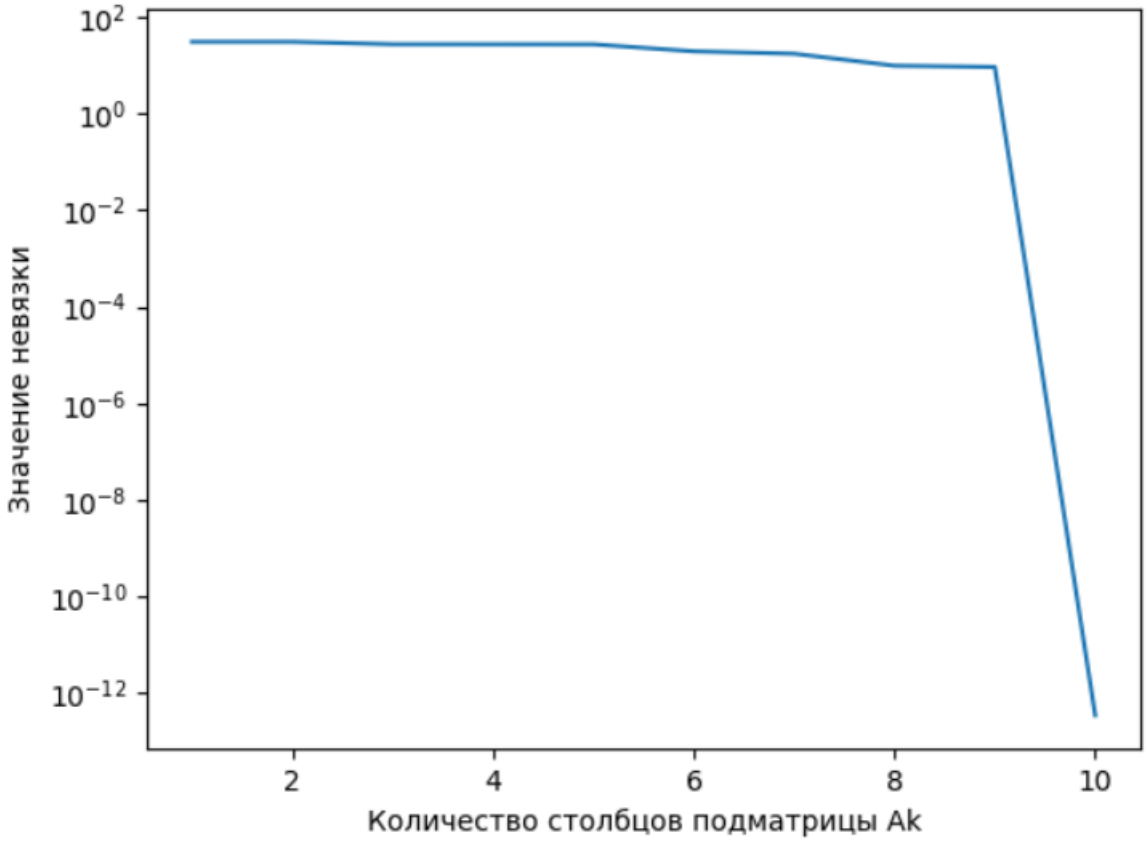
Там я узнал, что ответом к моей системе является вектор X = (0, 2, 0, 2, 0, 2, 0, 2, 0, 2). Если мы ещё разок внимательно посмотрим на вектор X, который насчитался у нас при k = 10, то мы увидим, что он почти такой же:

```
For k = 10 , X = [ 1.39219227e-13  2.00000000e+00 -2.06498560e-12  2.00000000e+00
  1.15007208e-13  2.00000000e+00  8.50069398e-14  2.00000000e+00
  2.29150032e-14  2.00000000e+00] , ||Ak*X - b|| =  3.414006384903678e-13
```

Есть, конечно, небольшие отклонения от ноликов и двоечек, но это можно объяснить погрешностями компьютера (в общем-то, потому и невязка у нас ненулевая).

Задание 2:

Вот так выглядит график, который изобразила моя программа:



Он показывает зависимость значения нашей невязки от k (то есть от количества столбцов в подматрице A_k). Мы можем наблюдать, что при k = 10, невязка стремится к нулю, что свидетельствует о том, что мы правильно реализовали решение задачи наименьших квадратов с помощью QR-разложения методом отражений.

Подытожив, можно сказать, что с помощью QR-разложения методом отражений мы можем решить не просто какую-то систему и получить ответ в виде вектора X, а еще и просто найти вектор X, который бы соответствовал решению задачи наименьших квадратов. Более того, в отличие от решения, скажем, методом нормальных уравнений, наш способ обладает хорошим числом обусловленности, то есть мы получили намного меньше погрешностей, чем могли. Этот метод достаточно сложно было реализовать, но нам это удалось, и, пожалуй, это того стоило.