Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
Московский государственный технический университет имени Н.Э. Баумана

Лабораторная работа
«Реализация метода отражений (Хаусхолдера) для QR-разложения матрицы A
по курсу:
«Вычислительные методы линейной алгебры»

Выполнил:
студент группы ИУ9-72
Иванов Георгий

Проверил:
Голубков А.Ю.

Москва, 2019

Содержание

1 Постановка задачи

Дано: A - произвольная матрица размера $n * m$.

Необходимо представить эту матрицу в виде $A = QR$, где Q - ортогональная или унитарная (в комплексном случае) матрица размера $n * n$ и R - верхняя треугольная матрица размера $n * m$. При помощи данного представления решить СЛАУ $Ax = b$ при осуществлении перехода к решению СЛАУ такого вида: $Rx = Q^* b$.

2 Теоретические сведения

2.1 Метод Хаусхолдера

Метод Хаусхолдера (метод отражений) - один из самых распространенных методов нахождения QR-разложения. Даннное отражение было предложено в 1958 году американским математиком Элстоном Скоттом Хаусхолдером. В основе данного метода лежит оператор отражения. Данный оператор отражает ненулевой вектор x из евклидова пространства E относительно гиперплоскости.

Рассмотрим вектор v - единичный вектор длины 1 (рисунок 1).

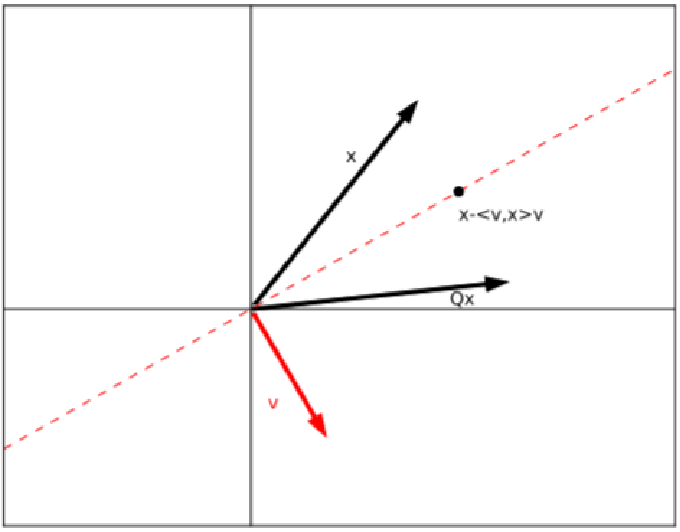

Рисунок 1. Отражение вектора

Так как мы имеем разложение n-мерного евклидова пространства на прямую сумму подпространства и его ортогонального дополнения $E =< v > \oplus < v >^{\perp}$, можно заметить, что вектор $x - \langle v, x\rangle v$ является проекцией вектора x на гиперплоскость $< v >^{\perp}$. Отраженный вектор x относительно гиперплоскости примет вид $Px = x - 2\langle v, x\rangle v$, то есть можно записать в виде $Px = x - 2v(v^{\perp}x)$, так как $\langle v, v\rangle = v^{\perp}v$. Поэтому матрица отражений P имеет представление $P = E - 2vv^{\perp}$, являющейся матрицей отражения относительно гиперплоскости $< v >^{\perp}$.

Заметим, что данная матрица симметрична. В самом деле, $P = P^{\perp}$, то есть $(vv^{\perp})^{\perp} = (v^{\perp})^{\perp}v^{\perp} = vv^{\perp}$. А $vv^{\perp}$ есть симметричная матрица. А также является ортогональной матрицей $P^{-1} = P^{\perp}$, принимая во внимание предыдущие формулы имеем $PP^{\perp} = PP = (E - 2vv^{\perp})(E - 2vv^{\perp}) = (E - 4vv^{\perp} + 4vv^{\perp}vv^{\perp}) = E$, что и означает справедливость. К тому же, данная матрица является инволютивной, так как ($P^2 = E$). Числа $\lambda = -1, 1$ являются собственными значениями матрицы P. Также можно заметить, что определитель данной матрицы -1, как произведение собственных значениях матрицы.

На i-м шаге метода с помощью метода отражения «убираются» ненулевые поддиагональные элементы в i-м столбце. Таким образом, после n-1 шагов преобразований получается матрица R из QR-разложения.

Определим вектор $u = \frac{v}{||v||}$, где v - единичный вектор длины 1. Перепишем оператор отражения относительно u.

$$P = E - 2vv^{\perp} = E - 2\frac{uu^{\perp}}{||u||^2}. \quad (1)$$

Разберем подробнее сам алгоритм для матрицы A.

Пусть x — вектор, составленный из первого столбца матрицы A. Выберем $u = x - ||x||e_1$, где e1 — единичный вектор $\begin{pmatrix} 1 & 0 & 0 & ... & 0 \end{pmatrix}^{\perp}$. Квадрат нормы этого вектора можно найти: $||u||^2 = u^{\perp}u = (x - ||x||e_1)^{\perp}(x - ||x||e_1) = x^{\perp}x - 2||x||x^{\perp}e_1 + ||x||^2 e_1^{\perp}e_1 = ||x||^2 - 2||x||x_1 + ||x||^2 = 2(||x||^2 - ||x||x_1)$
Применим оператор отражения к этому вектору.

$$Px = (E - 2\frac{uu^{\perp}}{||u||^2})x = (E - \frac{2u(x - ||x||e_1)^{\perp}}{||u||^2})x = x - \frac{2u||x||^2 - ||x||x_1}{2(||x||^2 - ||x||x_1)} = x - u = ||x||e_1, \quad (2)$$

то есть $Px = ||x|| \begin{pmatrix} 1 & 0 & ... & 0 \end{pmatrix}^{\perp}$ Это показывает, что матрица Хаусхолдера P действует на заданный вектор x, уничтожая все его элементы, кроме первого.

Рассмотрим некоторые нюансы. Если x почти коллинеарен вектору e_1, то вектор $u = x - sign(x)\sqrt{x^{\perp}x}e_1$ имеет малую норму. Поэтому возмонжо появление большой относительной ошибки при вычислении множителя $\frac{2}{u^{\perp}u}$. Эту трудность можно обойти, если взять с тем же знаком второй компоненты вектора, что и знак первой компоненты вектора x, т.е $u = x + sign(x)\sqrt{x^{\perp}x}e_1$. При таком выборе $||u||_{\infty} = |u_1|$. Полезно также придерживаться такой нормировки вектора u, что $u_1 = 1$.

Обозначим эту матрицу как P_1, то есть оператор отражения применен к первому столбцу матрицы A. Последовательно, применяя оператор отражений ко всем вектор, составленных из столбцов матрицы A, приводим матрицу A к треугольной форме. После k-итераций матрица $P_k P_{k-1} ... P_1 A$ имеет вид на рисунке 2. Алгоритм сохраняет собственные значения A, поскольку $A = Q^{-1}AQ$ имеет те же собственные значения, что и A (аналогичные матрицы имеют одинаковые собственные значения).

По завершении этого алгоритма матрица A содержит матрицу R факторизации QR и векторы $v_1, v_2, ... v_n$ - векторы отражения. Они будут использованы для вычисления произведений матрицы-вектора в форме Qx. Сама матрица Q не выводится. Её можно построить путем вычисления специальных матрично-векторных произведений.

2.2 Итерационное уточнение с переменной точностью

Для начала рассмотрим метод итерационного уточнения с постоянной точностью. В большинстве случаев метод Гаусса с выбором главного элемента позволяет найти приближенное решение с довольно высокой точностью. Однако иногда возникает необходимость найти решение с большей точностью. Полезно знать, что существует метод, позволяющий найти приближенное решение с относительной точностью, сравнимой с ем, если только число обусловленности не слишком велико. Этот метод, называемый итерационным уточнением, требует небольшого (примерно на 25%) увеличения машинного времени по сравнению с затратами на получение решения методом Гаусса. Пусть $x^{(0)}$-найденное на ЭВМ приближенное решение системы $Ax = b$.

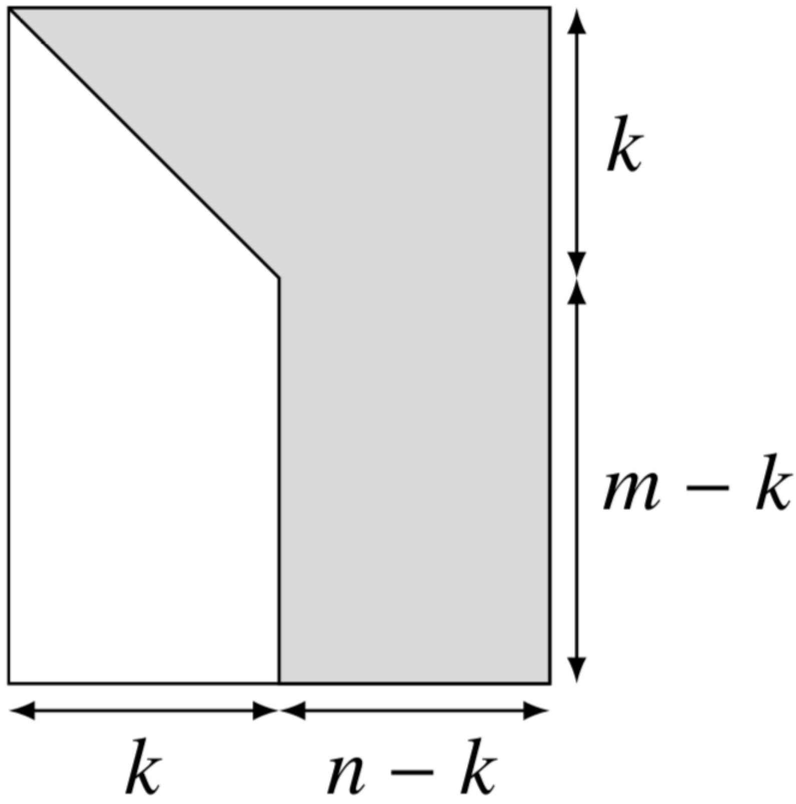

Рисунок 2. Вид матрицы $P_kP_{k-1}...P_1A$

Напомним, что невязка

$$r^{(0)} = b - Ax^{(0)} \tag{3}$$

и погрешность

$$e^{(0)} = x - x^{(0)} \tag{4}$$

связаны равенством

$$Ae^{(0)} = r^{(0)} \tag{5}$$

Если бы удалось найти $e^{(0)}$ как точное решение системы, то вектор $x^{(1)} = x^{(0)} + e^{(0)}$ дал бы точное решение системы. Однако в действительности вычисленное на ЭВМ значение $x^{(1)}$ неизбежно будет содержать ошибку. Тем не менее можно ожидать, что $x^{(1)}$ окажется лучшим приближением, чем $x^{(0)}$. Используя приближение $x^{(1)}$, аналогичным образом можно найти приближение $x^{(2)}$.

Опишем более подробно очередной $k+1$ шаг метода.

- Вычисляют $r^{(k)} \approx b - Ax^{(k)}$. Исключительно важно, чтобы вычисление производилось с повышенной точностью. Дело в том, что $b \approx Ax^{(k)}$ и поэтому при вычислении невязки неизбежно вычитание близких чисел, а следовательно, потеря большого числа значащих цифр. Одна из возможностей состоит в использовании для вычисления $Ax^{(k)}, b - Ax^{(k)}$ арифметики удвоенной точности.
- Вычисляют решение системы $r^{(k)} = Ae^{(k)}$ Так как матрица A не меняется, то получение очередного приближения с использованием однажды вычисленного разложения матрицы A требует сравнительно небольшого числа арифметических действий.

- Вычисляют $x^{(k+1)} = x^{(k)} + e^{(k)}$ Если число обусловленности не очень велико, то метод довольно быстро сходится. Сходимость характеризуется постепенным установлением значащих цифр в приближениях $x^{(k)}$. Если же процесс расходится, то в приближениях не устанавливаются даже старшие значащие цифры.

Естественно, что в точной арифметике мы не получилили бы в результате ничего нового, поскольку в этом случае $r = e^{(k)} = 0$. В арифметике с плавающей запятой такого же происходит, но тем не менее, найденный вектор $x^{(k+1)}$ может оказаться не сильно лучше вектора $x^{(k)}$, если тот уже был близок к решению, как это имеет место для методов Гаусса с выбором. Вместе с тем применительно к другим стратегиям выбора несколько шагов такого уточнения могут значительно улучшить качество решения (т.е количество разрядов). Значительно более эффективной версией процесса итерационного уточнения является версия этого алгоритма, в которой вектор невязки вычисляется с удвоенной точностью. Известны качественные оценки работы такого алгоритма итерационного уточнения с переменной точностью, оправдывающего его использование.

2.3 Практическая оценка ошибки

Чтобы вычислить практическую оценку ошибки, основанную на неравенстве:

$$||\delta x|| \leq ||A^{-1}|| * ||r|| \tag{6}$$

нужно уметь оценивать число $||A^{-1}||$. Этого достаточно, чтобы суметь оценить и число обусловленности $\kappa(A) = ||A^{-1}|| * ||A||$, поскольку $||A||$ находится легко. Одна из явных возможностей состоит в том, чтобы вычислить матрицу A^{-1} в явном виде, а затем определить её норму. Однако нет никакого выигрыша в решении системы путём вычисления данной матрицы. Поэтому вместо вычисления A^{-1} будем использовать оценщик обусловленности Хэйджера.

Алгоритм оценивает 1-норму $||B||_1$ матрицы B; предполагается, что мы можем вычислять произведения Bx и $B^T y$ для любых векторов x и y. Для начала выбирается произвольный вектор x с нормой $||x||_1 = 1$, например $x_i = \frac{1}{n}$. Вычисляется вектор $w = Bx, \zeta = sgn(w), z = B^T \zeta$. После же происходит сверка условия $||z||_\infty \leq z^T x$. Если выполняется, то возвращается $||w||_1$. Иначе $x = e_j$, где $|z_j| = ||z||_\infty$ и возвращаемся к вычислениям w, ζ, z.

На практике построенная оценка отличается от истинного значения нормы обратной матрицы приблизительно на 1 порядок.

Данный алгоритм можно применить к оцениванию относительного числа обусловленности или вычислению границы $||A^{-1}||*|r|$. Это сводится к задаче оценивания величины $||A^{-1}||*g|_\infty$, где g — вектор с неотрицательными компонентами. Пусть e- вектор с неотрицательными единичными компонентами. Далее

$$|||A^{-1}||| * |A|||_\infty = |||A^{-1}||| * |A|e||_\infty = |||A^{-1}||| * g||_\infty \tag{7}$$

Введём матрицу $G = diag(g_1, ...g_n)$. Тогда $g = Ge$.Выведем

$$|||A^{-1}||| * g||_\infty = ||A^{-1}G||_\infty \tag{8}$$

А сама практическая оценка ошибки вычисляется:

$$error = \frac{||x^{(k)} - x||_\infty}{||x^{(k)}||_\infty} \leq \frac{|||A^{-1}||r|||_\infty}{||x^{(k)||_\infty}} \tag{9}$$

Величину $|||A^{-1}||r|||_\infty$ оцениваем с помощью Хэйджера и алгоритма, описанного выше.

3 Практическая реализация

Программа написана на языке программирования Python3 с использованием библиотек *numpy*, *math* для работы с матрицами и *argparse* для работы с аргументами командной строки. Далее преполагается, что все матрицы осуществляют операции только с вещественными числами.

Листинг 1. Метод отражения

```
def householder_without_q(A):
    def sign(x):
        return -1 if x < 0 else 1

    m, n = np.shape(A)
    W = np.zeros((m,n))
    R = A.copy()

    for k in range(m-1):
        x = R[k:m,k]
        e = np.zeros(len(x))
        e[0] = 1
        alpha = sign(x[0]) * np.linalg.norm(x,2)
        u = x - alpha * e
        v = u / np.linalg.norm(u,2)
        R[k:m, k:n] = R[k:m, k:n] - 2 * np.outer(v,np.dot(v.transpose(),R[k:m, k:n]))
        W[k:m,k]=v
    return W, R
```

Листинг 2. Восстановление матрицы Q через вектора отражения.

```
def form_q(W):
    m, n = np.shape(W)
    Q = np.identity(m)
    for i in range(m):
        for k in range(n-1, -1, -1):
            Q[k:m,i] = Q[k:m,i]-2*np.dot(np.outer(W[k:m,k],W[k:m,k]),Q[k:m,i])
    return Q
```

Листинг 3. Итерационное уточнение.

```
def iterative_refinement(A, B, x1, eps, iterations=100):
    iterate = 0
    while True:
        delta_x = delta_error(A, B, x1, True)
        if np.linalg.norm(delta_x) < eps or iterate > iterations:
            break
        else:
            x1 = np.array([x1]).transpose() + delta_x
            x1 = x1.transpose().tolist()[0]
            iterate += 1

    return x1


def delta_error(A, B, x1, double_precision=False):
    m = A.shape[0]
    b1 = np.zeros(m)
    for i in range(m):
        b1[i] = np.dot(A[i, ], x1)

    chosen_type = np.longfloat if double_precision else A.dtype
    r = b1.astype(chosen_type) - B.astype(chosen_type)
    r = np.negative(r)

    delta_x = gauss(A, np.array([r]).transpose())

    return delta_x
```

Листинг 4. «Оценщик» матричной нормы.

```
def condition_hager(n, a):
    import numpy as np
    i1 = -1
    c1 = 0.0
    b = np.zeros(n)
    for i in range(0, n):
        b[i] = 1.0 / float(n)
    while (True):
        b2 = np.linalg.solve(a, b)
        for i in range(0, n):
            b[i] = b2[i]
        c2 = 0.0
        for i in range(0, n):
            c2 = c2 + abs(b[i])
        for i in range(0, n):
            b[i] = r8_sign(b[i])
        b2 = np.linalg.solve(np.transpose(a), b)
        for i in range(0, n):
            b[i] = b2[i]
        i2 = r8vec_max_abs_index(n, b)
```

```
        if (0 <= i1):
            if (i1 == i2 or c2 <= c1):
                break
        i1 = i2
        c1 = c2
        for i in range(0, n):
            b[i] = 0.0
        b[i1] = 1.0
    value = c2 * r8mat_norm_l1(n, n, a)
    return value

def r8vec_max_abs_index( n, a ):
  if ( n <= 0 ):
    max_abs_index = -1
  else:
    max_abs_index = 0
    for i in range ( 1, n ):
      if ( abs ( a[max_abs_index] ) < abs ( a[i] ) ):
        max_abs_index = i
  return max_abs_index

def r8mat_norm_l1 ( m, n, a ):
  value = 0.0
  for j in range ( 0, n ):
    row_sum = 0.0
    for i in range ( 0, m ):
      row_sum = row_sum + abs ( a[i,j] )
    value = max ( value, row_sum )
  return value

def r8_sign(x):
    if (x < 0.0):
        value = -1.0
    else:
        value = +1.0
    return value

def check_error(A, X1, B):
    m = A.shape[0]
    b1 = np.zeros(m)
    for i in range(m):
        b1[i] = np.dot(A[i,], X1)
    r = b1 - B
    R = np.eye(m)
    for i in range(m):
        R[i,i] = r[i]
    return norm(R.dot(condition_hager(m,np.linalg.inv(A)))) / norm(X1)
```

4 Тестирование

4.1 Матрица 3x3

В качестве первого теста были взяты следующие матрицы A и b:

$$A = \begin{pmatrix} 2 & 2 & 4 \\ 1 & 3 & -2 \\ 3 & 1 & -3 \end{pmatrix}, \quad b = \begin{pmatrix} 18 \\ 1 \\ 14 \end{pmatrix}.$$

Методом отражения былы найдены матрицы разложения:

$$Q = \begin{pmatrix} -0.53452248 & -0.21821789 & -0.81649658 \\ -0.26726124 & -0.87287156 & 0.40824829 \\ -0.80178373 & 0.43643578 & 0.40824829 \end{pmatrix},$$

$$R = \begin{pmatrix} -3.74165739 & -2.67261242 & -4.00891863 \\ 0. & -2.61861468 & 2.1821789 \\ 0. & 0. & -2.85773803 \end{pmatrix},$$

Соответственно был найден неизвестный вектор x:

$$x = \begin{pmatrix} 1.0000000000000027 \\ 1.9999999999999987 \\ 2.999999999999999 \end{pmatrix}$$

В результате итерационных уточнений:

```
Iteration [0]:[1.0000000000000027, 1.9999999999999987, 2.999999999999999]
```

Практическая оценка ошибка для матрицы A выдало:

$$\begin{pmatrix} 6.978544726215272e - 16 \end{pmatrix}$$

Оценим 1-норму матрицы A^{-1}. Для матрицы A^{-1} по Хэйджеру — $||A^{-1}||_1 = 6.0$, а вычисленная норма равна — $||A^{-1}||_1 = 1.0$

4.2 Матрица 6x6

Возьмем гильбертовую матрицу плохой обусловленности $A_{i,j} = \frac{1}{i+j-1}$

$$A = \begin{pmatrix} 1. & 0.5 & 0.33333333 & 0.25 & 0.2 & 0.16666667 \\ 0.5 & 0.33333333 & 0.25 & 0.2 & 0.16666667 & 0.14285714 \\ 0.33333333 & 0.25 & 0.2 & 0.16666667 & 0.14285714 & 0.125 \\ 0.25 & 0.2 & 0.16666667 & 0.14285714 & 0.125 & 0.11111111 \\ 0.2 & 0.16666667 & 0.14285714 & 0.125 & 0.11111111 & 0.1 \\ 0.16666667 & 0.14285714 & 0.125 & 0.11111111 & 0.1 & 0.09090909 \end{pmatrix}, \quad b = \begin{pmatrix} 0.352846 \\ 0.148772 \\ 0.38714 \\ 0.998776 \\ 0.27418 \\ 0.634250 \end{pmatrix}$$

Методом отражения были найдены матрицы разложения:

$$Q = \begin{pmatrix} -0.81885037 & 0.53968051 & -0.18926054 & -0.04823609 & 0.00902961 & -0.00110499 \\ -0.40942518 & -0.3319879 & 0.70241698 & 0.44892647 & -0.1616526 & 0.03314958 \\ -0.27295012 & -0.42193465 & 0.15293341 & -0.57232256 & 0.58539031 & -0.23204703 \\ -0.20471259 & -0.40672521 & -0.20151311 & -0.38664006 & -0.46868501 & 0.61879209 \\ -0.16377007 & -0.37352642 & -0.39626059 & 0.09151695 & -0.42854071 & -0.6961411 \\ -0.13647506 & -0.3399305 & -0.49977217 & 0.55742201 & 0.47727592 & 0.27845644 \end{pmatrix},$$

$$R = \begin{pmatrix} -1.22122434e+00 & -7.01871744e-01 & -5.04470316e-01 & -3.96969127e-01 & -3.28433734e \\ 0.00000000e+00 & -1.38466990e-01 & -1.51130170e-01 & -1.44364356e-01 & -1.34008232e \\ 0.00000000e+00 & 0.00000000e+00 & -9.56161345e-03 & -1.51932381e-02 & -1.81302905e \\ 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 & 4.80281542e-04 & 9.94238186e \\ 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 & 1.73389814e \\ 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 & 0.00000000e \end{pmatrix}$$

Соответственно был найден неизвестный вектор x:

$$x = \begin{pmatrix} -6016.3111399813393425 \\ 171527.09423790074548 \\ -1157523.7592948742912 \\ 3003185.0089350438998 \\ -3307413.1111909475205 \\ 1300684.439822783459 \end{pmatrix}$$

Практическая оценка ошибки для матрицы A выдала:

$$\begin{pmatrix} 3.802252320730109663e-11 \end{pmatrix}$$

В результате итерационных уточнений:

```
Iteration [0]:[-6016.311139982503, 171527.09423790494, -1157523.759294872, 3003185.00893504,
↪  -3307413.111190962, 1300684.4398227963]
Iteration [1]:[-6016.3111399825413708, 171527.09423790497988, -1157523.7592948721438,
↪  3003185.0089350396556, -3307413.111190961424, 1300684.439822796439]
Iteration [2]:[-6016.3111399824333345, 171527.09423790487698, -1157523.7592948722319,
↪  3003185.0089350396954, -3307413.111190961184, 1300684.4398227962126]
```

```
Iteration [3]:[-6016.3111399823541094, 171527.09423790475341, -1157523.7592948723334, 3003185.008935039726, -3307413.111190960951, 1300684.439822795985]
Iteration [4]:[-6016.3111399822908654, 171527.0942379046227, -1157523.7592948724372, 3003185.0089350397566, -3307413.1111909607175, 1300684.4398227958798]
Iteration [5]:[-6016.311139982236992, 171527.09423790449105, -1157523.7592948725381, 3003185.0089350397907, -3307413.1111909604804, 1300684.439822795818]
Iteration [6]:[-6016.3111399821892027, 171527.09423790436162, -1157523.7592948726343, 3003185.00893503983, -3307413.111190960238, 1300684.4398227957621]
Iteration [7]:[-6016.311139982145696, 171527.09423790423558, -1157523.7592948727247, 3003185.0089350398748, -3307413.1111909599908, 1300684.4398227956956]
Iteration [8]:[-6016.3111399821054563, 171527.09423790411356, -1157523.7592948728094, 3003185.0089350399253, -3307413.1111909597387, 1300684.4398227956116]
Iteration [9]:[-6016.3111399820678313, 171527.09423790399539, -1157523.7592948728889, 3003185.0089350399815, -3307413.1111909594817, 1300684.4398227955069]
Iteration [10]:[-6016.3111399820326866, 171527.09423790388111, -1157523.7592948729628, 3003185.0089350400424, -3307413.1111909592207, 1300684.4398227953818]
Iteration [11]:[-6016.3111399819995664, 171527.09423790377038, -1157523.7592948730319, 3003185.008935040108, -3307413.1111909589554, 1300684.4398227952365]
Iteration [12]:[-6016.3111399819682608, 171527.0942379036632, -1157523.7592948730967, 3003185.008935040178, -3307413.1111909586868, 1300684.4398227950732]
Iteration [13]:[-6016.311139981938634, 171527.09423790355909, -1157523.7592948731574, 3003185.0089350402516, -3307413.1111909584151, 1300684.4398227948926]
Iteration [14]:[-6016.311139981910556, 171527.09423790345791, -1157523.7592948732142, 3003185.0089350403284, -3307413.1111909581405, 1300684.4398227946963]
Iteration [15]:[-6016.3111399818838954, 171527.09423790335943, -1157523.7592948732677, 3003185.0089350404085, -3307413.111190957863, 1300684.4398227944855]
Iteration [16]:[-6016.311139981858488, 171527.09423790326339, -1157523.7592948733179, 3003185.0089350404912, -3307413.1111909575834, 1300684.439822794262]
Iteration [17]:[-6016.311139981834333, 171527.09423790316981, -1157523.7592948733652, 3003185.0089350405763, -3307413.1111909573017, 1300684.4398227940267]
Iteration [18]:[-6016.311139981811223, 171527.09423790307858, -1157523.7592948734102, 3003185.0089350406638, -3307413.1111909570182, 1300684.4398227937805]
Iteration [19]:[-6016.311139981789123, 171527.09423790298936, -1157523.7592948734527, 3003185.0089350407532, -3307413.1111909567333, 1300684.439822793525]
Iteration [20]:[-6016.3111399817679983, 171527.0942379029021, -1157523.759294873493, 3003185.008935040844, -3307413.1111909564472, 1300684.4398227932603]
Iteration [21]:[-6016.3111399817477456, 171527.09423790281674, -1157523.7592948735314, 3003185.0089350409364, -3307413.1111909561598, 1300684.4398227929883]
Iteration [22]:[-6016.311139981728243, 171527.094237902733, -1157523.759294873568, 3003185.0089350410303, -3307413.1111909558715, 1300684.4398227927092]
Iteration [23]:[-6016.311139981709477, 171527.09423790265086, -1157523.759294873603, 3003185.0089350411256, -3307413.1111909555825, 1300684.4398227924233]
Iteration [24]:[-6016.31113998169135, 171527.09423790256994, -1157523.7592948736362, 3003185.0089350412215, -3307413.1111909552928, 1300684.4398227921317]
Iteration [25]:[-6016.3111399816738714, 171527.09423790249069, -1157523.7592948736684, 3003185.0089350413186, -3307413.1111909550027, 1300684.4398227918356]
Iteration [26]:[-6016.3111399816569196, 171527.09423790241249, -1157523.7592948736991, 3003185.0089350414166, -3307413.1111909547121, 1300684.4398227915343]
Iteration [27]:[-6016.3111399816404914, 171527.0942379023356, -1157523.759294873729, 3003185.0089350415153, -3307413.1111909544213, 1300684.4398227912291]
Iteration [28]:[-6016.3111399816246267, 171527.09423790226016, -1157523.7592948737581, 3003185.0089350416144, -3307413.1111909541303, 1300684.4398227909203]
```

```
Iteration [29]:[-6016.3111399816091676, 171527.0942379021856, -1157523.7592948737858, 3003185.0089350417138, -3307413.1111909538392, 1300684.4398227906088]
Iteration [30]:[-6016.311139981594179, 171527.0942379021122, -1157523.759294873813, 3003185.0089350418136, -3307413.111190953548, 1300684.439822790294]
Iteration [31]:[-6016.3111399815795104, 171527.09423790203964, -1157523.7592948738394, 3003185.008935041914, -3307413.11119095325 7, 1300684.4398227899768]
Iteration [32]:[-6016.3111399815653217, 171527.09423790196819, -1157523.759294873865, 3003185.0089350420142, -3307413.11119095296 6, 1300684.4398227896575]
Iteration [33]:[-6016.3111399815513596, 171527.09423790189747, -1157523.75929487389, 3003185.0089350421147, -3307413.11119095267 5, 1300684.4398227893362]
Iteration [34]:[-6016.311139981537726, 171527.09423790182773, -1157523.7592948739147, 3003185.0089350422154, -3307413.1111909523847, 1300684.4398227890134]
Iteration [35]:[-6016.311139981524466, 171527.09423790175883, -1157523.7592948739389, 3003185.0089350423161, -3307413.1111909520944, 1300684.4398227886892]
Iteration [36]:[-6016.311139981511351, 171527.0942379016907, -1157523.7592948739626, 3003185.0089350424166, -3307413.1111909518045, 1300684.4398227883644]
Iteration [37]:[-6016.3111399814985534, 171527.09423790162339, -1157523.759294873986, 3003185.0089350425171, -3307413.1111909515153, 1300684.4398227880383]
Iteration [38]:[-6016.3111399814860363, 171527.09423790155665, -1157523.759294874009, 3003185.0089350426176, -3307413.1111909512263, 1300684.4398227877117]
Iteration [39]:[-6016.3111399814737883, 171527.09423790149083, -1157523.7592948740319, 3003185.0089350427181, -3307413.1111909509377, 1300684.4398227873846]
Iteration [40]:[-6016.311139981461661, 171527.09423790142542, -1157523.7592948740543, 3003185.0089350428182, -3307413.1111909506494, 1300684.4398227870565]
Iteration [41]:[-6016.311139981449742, 171527.09423790136077, -1157523.7592948740765, 3003185.008935042918, -3307413.111190950362, 1300684.4398227867288]
Iteration [42]:[-6016.311139981438091, 171527.09423790129682, -1157523.7592948740984, 3003185.0089350430176, -3307413.111190950075, 1300684.4398227864009]
Iteration [43]:[-6016.311139981426458, 171527.09423790123317, -1157523.7592948741201, 3003185.0089350431172, -3307413.1111909497888, 1300684.4398227860726]
Iteration [44]:[-6016.311139981415052, 171527.09423790117026, -1157523.7592948741417, 3003185.0089350432163, -3307413.1111909495032, 1300684.4398227857447]
Iteration [45]:[-6016.311139981403823, 171527.09423790110802, -1157523.7592948741633, 3003185.008935043315, -3307413.1111909492179, 1300684.4398227854168]
Iteration [46]:[-6016.31113998139275, 171527.09423790104633, -1157523.7592948741849, 3003185.0089350434134, -3307413.1111909489334, 1300684.4398227850895]
Iteration [47]:[-6016.3111399813818663, 171527.09423790098525, -1157523.7592948742064, 3003185.0089350435117, -3307413.1111909486497, 1300684.4398227847624]
Iteration [48]:[-6016.3111399813710296, 171527.09423790092448, -1157523.7592948742275, 3003185.0089350436092, -3307413.1111909483666, 1300684.439822784436]
Iteration [49]:[-6016.3111399813603475, 171527.09423790086433, -1157523.7592948742488, 3003185.0089350437065, -3307413.111190948084, 1300684.43982278411]
Iteration [50]:[-6016.31113998134974, 171527.09423790080454, -1157523.7592948742699, 3003185.0089350438034, -3307413.111190947802, 1300684.4398227837842]
Iteration [51]:[-6016.3111399813393425, 171527.09423790074548, -1157523.7592948742912, 3003185.0089350438998, -3307413.1111909475205, 1300684.439822783459]
```

Оценим норму матрицы A^{-1}. Для матрицы A^{-1} по Хэйджеру — $||A^{-1}||_1 = 29070279.00683587$, а вычисленная норма равна — $||A^{-1}||_1 = 11865420.001538314$

При увеличении размера матрицы Гильберта ошибка в решении будет только расти (это несложно проверить, задавая n = 7, 8, . . .). Причем, при n = 12 выведется сообщение о том, что матрица плохообусловлена и решение может оказаться неверным.

Так как становится проблематичным проверять матрицы большой размерности, то далее для проверки результатов будет использоваться библиотека *numpy*. В данном случае вычисления показали правильные результаты.

Далее будут сгенерированы несколько матриц разного размера.

4.3 Матрица 100x100

Оценим норму матрицы (случайная сгенерированная для 100x100) A^{-1}. Для матрицы A^{-1} по Хэйджеру — $||A^{-1}||_1 = 648.5457778131733$, а вычисленная норма равна — $||A^{-1}||_1 = 245.3424489834818$

Практическая оценка ошибки для матрицы A выдала:

$$\begin{pmatrix} 2.652568163522353e-09 \end{pmatrix}$$

4.4 Матрица 250x250

Оценим норму матрицы (случайная сгенерированная для 250x250) A^{-1}. Для матрицы A^{-1} по Хэйджеру — $||A^{-1}||_1 = 532.8694720046803$, а вычисленная норма равна — $||A^{-1}||_1 = 194.8191031341605$

Практическая оценка ошибки для матрицы A выдала:

$$\begin{pmatrix} 5.735121479435413e-05 \end{pmatrix}$$

4.5 Матрица 500x500

Оценим норму матрицы (случайная сгенерированная для 250x250) A^{-1}. Для матрицы A^{-1} по Хэйджеру — $||A^{-1}||_1 = 3025.6886832382224$, а вычисленная норма равна — $||A^{-1}||_1 = 1077.46022138407$

Практическая оценка ошибки для матрицы A выдала:

$$\begin{pmatrix} 1.804813588396503e-02 \end{pmatrix}$$

5 Вывод

В ходе выполнения лабораторной работы был изучен метод отражений, и было найдено решение системы $Ax = b$ с помощью данного разложения, для этого реализована программа на языке Python3. Использование преобразований Хаусхолдера является наиболее простым из численно устойчивых алгоритмов QR-разложения благодаря использованию отражений в качестве механизма для получения нулей в матрице R. Однако алгоритм отражения Хаусхолдера имеет большую пропускную способность и не распараллеливается, так как каждое отражение, создающее новый нулевой элемент, изменяет всю совокупность матриц Q и R.

Список литературы

[1] Gander, W. in: Nicolet et al. Informatik fur Ingenieure, Springer Verlag, 1979.

[2] Деммель Дж. Вычислительная линейная алгебра.Теория и приложения. — М.: Мир, 2001.

[3] Голуб Дж., Ван Лоун Ч. Матричные вычисления. — М.: Мир, 1999.