Лабораторная работа №2. Нормы векторов и матриц, решение переопределенной системы линейных уравнений

Представим, что перед нами поставили задачу составить траекторию облета квадрокоптером нескольких точек по наиболее выгодной траектории.

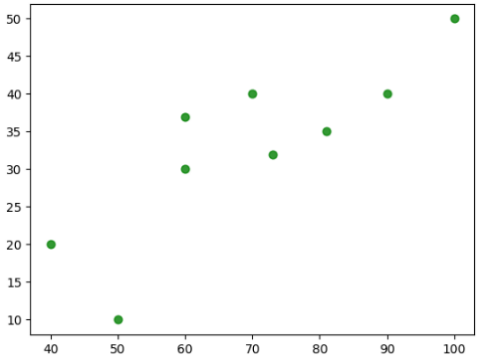

Рисунок 1. Карта точек

Облет всех точек по ломаной траектории не является выгодным по каким-либо причинам: например, критические затраты времени или энергии квадрокоптера.

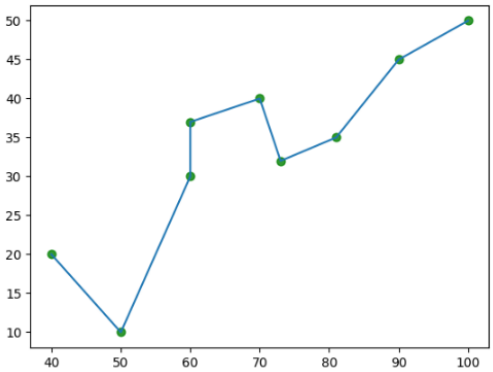

Рисунок 2. Ломаная траектория

Более выгодным является пролет квадрокоптера по прямой линии – он сможет максимально быстро пролететь над всеми обьектами, выполнив, например, аэрофотосъемку местности.

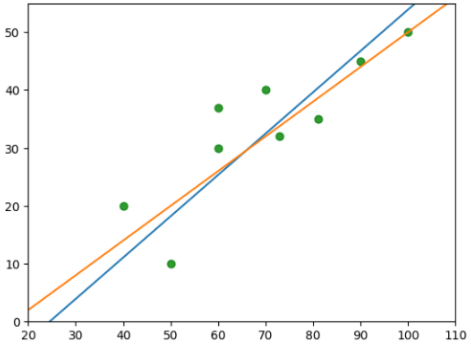

Рисунок 3. Прямолинейные траектории

На нашей карте показаны две возможные траектории, на самом деле их бесконечно много. Как оценить, какая траектория является наиболее выгодной? Скорее всего та, которая проходит

ближе к заданным точкам. А как с математической точки зрения оценить близость двух точек?

На самом деле мы столкнулись с известной задачей линейной регрессии. Давайте обсудим её математическое решение.

Пусть множество зеленых точек задано набором $\{(x_1, y_1),...,(x_n, y_n)\}$, линейная функция (траектория полета), которую мы хотим получить, уравнением $y = f(x)$. Подставив известные значения аргумента, получим набор значений функции: $\hat{y}_1 = f(x_1)$, $\dots \hat{y}_n = f(x_n)$. Итак, имеем два вектора, два набора значений $(y_1,...y_n)$ и $(\hat{y}_1,...\hat{y}_n)$.

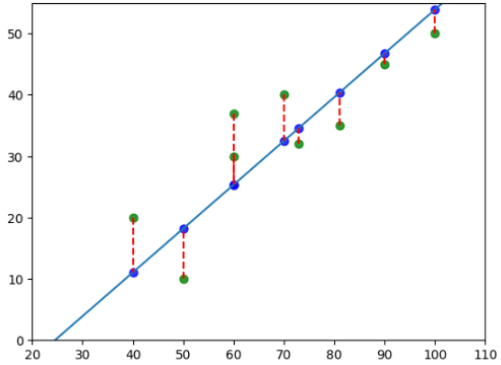

Рисунок 4. Разница значений

Разницу между этими наборами значений мы могли бы оценить по обычной формуле расстояния между двумя точками (в n-мерной пространстве): $\|y - \hat{y}\|_2 = \sqrt{(y_1 - \hat{y}_1)^2 + ... + (y_n - \hat{y}_n)^2}$, в этом случае мы оцениваем среднеквадратическую погрешность между двумя векторами. Можем использовать другие известные погрешности, например, оценить суммарную абсолютную

погрешность: $\|y-\hat{y}\|_1 = |y_1-\hat{y}_1| + ... + |y_n-\hat{y}_n|$ или максимальную из абсолютных погрешностей: $\|y-\hat{y}\|_\infty = \max_i |y_i-\hat{y}_i|$. С точки зрения поиска оптимальной траектории полета квадрокоптера, мы получили функцию, значение которой необходимо минимизировать. В задачах машинного обучения такую функцию называют целевой функцией. В курсе линейной алгебры эта функция называлась нормой вектора.

Данная лабораторная работа будет посвящена искусству нахождения норм векторов, норм матриц, а также поиску разложения матриц.

Первым делом познакомимся с различными программными реализациями умножения векторов и матриц. Напомню, что в NumPy матрица реализована как двумерный массив ndarray. Ndarray является многомерным однородным массивом с заранее заданным количеством элементов. Однородный — потому что практически все объекты в нем одного размера или типа. Количество размерностей и объектов массива определяются его размерностью (shape), кортежем N-положительных целых чисел. Они указывают размер каждой размерности. Размерности определяются как оси. Размер массивов NumPy фиксирован, а это значит, что после создания объекта его уже нельзя поменять. Это поведение отличается от такового у списков Python, которые могут увеличиваться и уменьшаться в размерах.

При работе с индексами массивов всегда используются квадратные скобки ([ ]). С помощью индексирования можно ссылаться на отдельные элементы, выделяя их или даже меняя значения. При создании нового массива шкала с индексами создается автоматически.

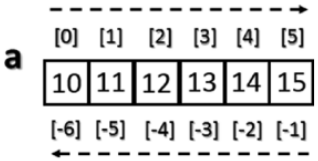

Рисунок 5. Шкала индексов в Python

Для получения доступа к одному элементу на него нужно сослаться через его индекс.

Можно получить не один элемент массива, а сразу его часть (срез). Для получения части массива, которую необходимо извлечь, нужно использовать синтаксис среза; это последовательность числовых значений, разделенная двоеточием (:) в квадратных скобках. Синтаксис среза start:stop:step.

Чтобы лучше понять синтаксис среза, необходимо рассматривать и случаи, когда явные числовые значения не используются. Если не ввести первое число, NumPy неявно интерпретирует его как 0 (то есть, первый элемент массива). Если пропустить второй — он будет заменен на максимальный индекс, а если последний — представлен как 1. То есть, все элементы будут перебираться без интервалов.

Таблица 1. Работа с индексами в массиве Python

команда	**результат**
import numpy as np	загрузили модуль NumPy
a = np.array([10, 11, 12, 13, 14, 15], int)	создание массива из целых чисел out: array([10, 11, 12, 13, 14, 15])

команда	результат
a[2]	получаем второй (третий) элемент массива (помним про нумерацию с нуля) out: 12
a[:]	срез из всех элементов out: array([10, 11, 12, 13, 14, 15])
a[:2]	срез до второго элемента out: array([10, 11])
a[2:]	срез от второго элемента и до конца out: array([12, 13, 14, 15])
a[2:4]	срез от второго до четвертого элемента, правую границу не включаем out: array([12, 13])
a[::2]	срез с шагом 2 out: array([10, 12, 14])
a[::-1]	срез с шагом -1 out: array([15, 14, 13, 12, 11, 10])
a[1:5:2]	срез от первого до пятого элемента с шагом 2 out: array([11, 13])
a[-1]	отрицательные индексы тоже в ходу, получим последний

команда	результат
	элемент out: 15
a[-3:-1]	out: array([13, 14])
a[5:1:-1]	out: array([15, 14, 13, 12])
b = np.array([[10, 11, 12, 13], [20, 21, 22, 23], [30, 31, 32, 33], [40, 41, 42, 43]], int)	такая конструкция позволяет получать матрицы out: array([[10, 11, 12, 13], [20, 21, 22, 23], [30, 31, 32, 33], [40, 41, 42, 43]])
b[2,1]	получаем элемент в третьей строке втором столбце (помним про нумерацию с нуля) out: 31
b[:, 1]	второй столбец out: array([11, 21, 31, 41])
b[2,:]	out: array([30, 31, 32, 33])
b[1:4:2,:]	out: array([[20, 21, 22, 23], [40, 41, 42, 43]])
b[1:4:2,::2]	out: array([[20, 22], [40, 42]])

Операция умножения матриц вам хорошо известна из курса линейной алгебры. Умножение матрицы A на матрицу B возможно только в случае, когда количество столбцов матрицы A совпадает

с количеством строк матрицы B. Элемент cij новой матрицы получается как умножение i-той строки матрицы A на j-тый столбец матрицы B.

Возможно несколько реализаций матричного умножения: в скалярном виде, в векторном и в матричном.

Скалярный вид: $c_{ij} = \sum_{t=1}^{k} a_{it} b_{tj},\ 1 \le i \le n, 1 \le j \le m$, где матрица A имеет размер nxk, матрица B имеет размер kxm.

Алгоритм умножения матриц в скалярном виде (поэлементный) представлен в приложенном файле.

```
Скалярный вид умножения матриц.
```

```
n = 3
k = 4
m = 5
A =  np.random.randint(0,11,(n, k))
B = np.random.randint(0,11,(k, m))
C = np.zeros((n,m), dtype = int)
```

```
for i in range(n):
    for j in range(m):
        for p in range(k):
            C[i,j] = C[i,j] + A[i,p]*B[p,j]
```

Рисунок 6. Поэлементное умножение матриц в Python

Однако такое решение является достаточно медленным.

Векторные операции существенно увеличивают скорость вычислений.

Алгоритм умножения матриц в векторном виде позволяет сразу находить произведение i-той строки матрицы A на j-тый столбец матрицы B.

Векторный вид: $c_{ij} = A(i,:) \cdot B(:, j),\ 1 \le i \le n, 1 \le j \le m$.

Напомню, что в этом случае операция умножения – это уже скалярное умножение из модуля NumPy (.dot)

Мы можем использовать также матричное умножение. В нашем случае будем «собирать» матрицу C по строкам (для нахождения i-той строки матрицы C умножим i-тую строку матрицы A на матрицу B)

Матричный вид: C(i, :) = A(i, :)·B .

Можно также «собрать» матрицу C по столбцам.

Вернемся к вопросу, поставленному в начале лабораторной работы. Необходимо оценить расстояние между двумя векторами, для этого будем использовать математическое понятие нормы вектора.

Напомню определение нормы в векторном пространстве X. Нормой называется такое отображение векторного пространства X во множество вещественных чисел R $\|\cdot\| : X \to R$, что выполняются следующие свойства для любых элементов (векторов или матриц) x, y векторного пространства X и скаляра λ: 1. $\|x\| \geq 0,\ \|x\| = 0 \Leftrightarrow x = 0$ (положительная определенность), 2. $\|\lambda \cdot x\| = |\lambda| \cdot \|x\|$ (однородность), 3. $\|x + y\| \leq \|x\| + \|y\|$ (неравенство треугольника).

Норма является естественным обобщением понятия длины вектора в евклидовом пространстве, таким образом, нормированные пространства — векторные пространства, оснащённые возможностью определения длины вектора.

Существует множество различных способов введения норм. В вычислительных методах наиболее употребительными являются следующие три нормы: $\|x\|_1 = \sum_{i=1}^{n} |x_i|$, $\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$, $\|x\|_\infty = \max_i |x_i|$.

Также используется р- норма: $\|x\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}$

Выбор той или иной конкретной нормы в практических задачах диктуется тем, какие требования предъявляются к точности решения. Выбор нормы $\|x\|_1$ фактически отвечает случаю, когда малой должна быть суммарная абсолютная погрешность в компонентах решения; выбор $\|x\|_2$ соответствует критерию малости среднеквадратичной погрешности, а принятие в качестве нормы $\|x\|_\infty$ означает, что малой должна быть максимальная из абсолютных погрешностей в компонентах решения.

Норма матрицы — норма в линейном пространстве матриц. Обычно, от матричной нормы требуют выполнение условия субмультипликативности: $\|A \cdot B\| \le \|A\| \cdot \|B\|$ для всех матриц A и B. Все нормы, которые рассматриваем в этой лабораторной работе, удовлетворяют условию сбмультипликативности.

$\|A\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$ - максимум из суммы модулей по всем столбцам.

$\|A\|_2 = \max_j \sqrt{\lambda_j (A^T \cdot A)}$ - максимум из собственных значений матрицы.

$\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$ - максимум из суммы модулей по всем строкам.

$\|A\|_F = \sqrt{\sum_{i,j=1}^{n} |a_{ij}|^2}$ - норма Фробениуса.

Для нахождения нормы вектора и матрицы в среде программирования Python можно использовать функцию np.linalg.norm(). Синтаксис: numpy.linalg.norm(x, ord=None, axis=None, keepdims=False). В зависимости от параметра ord данная функция может возвращать одну из восьми норм или одну из бесконечного числа векторных норм.

Приведу три из них:

- ord = None - для матриц возвращается норма Фробениуса, для векторов соответствует ord = 2 (то есть вычисляет корень квадратный из суммы квадратов векторов) (установлен по умолчанию);
- ord=np.inf - для матриц возвращается np.max(np.sum(np.abs(x), axis=1)), для векторов возвращается np.max(np.abs(x));
- ord = 1 - для матриц возвращается np.max(np.sum(np.abs(x), axis=0)), для векторов возвращается np.sum(np.abs(x)).

Рассмотрим еще один способ построения оптимальной траектории. Пусть даны четыре точки с координатами (40,20), (50,10), (60, 40). (70,30).

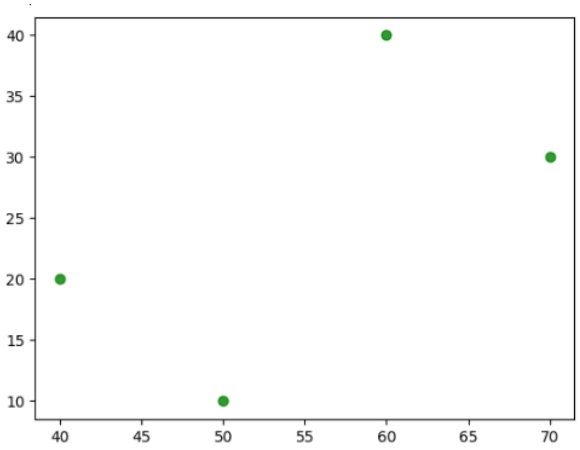

Рисунок 7. Множество точек

Мы по-прежнему хотим найти уравнение прямой $y = ax + b$ на оптимальном расстоянии ото всех точек. Попробуем составить систему уравнений, проходящих через все точки: $\begin{cases} 40a + b = 20 \\ 50a + b = 10 \\ 60a + b = 40 \\ 70a + b = 30 \end{cases}$.

Это переопределенная система линейных уравнений, она не имеет решения в классическом понимании. Но мы можем найти псевдорешение этой системы, воспользовавшись известным алгоритмом.

Итак, предположим, что количество уравнений больше количества неизвестных, линейная система имеет вид: $AX = B$, где A — матрица размера mxn, X — столбец nx1, B имеет размер

mx1. Обратной матрицы для матрицы A не существует, однако, если допустить, что столбцы A линейно независимы, то тогда существует обратная матрица к матрице $A^T \cdot A$. Умножим левую и правую часть уравнения на A^T: $A^T \cdot AX = A^T \cdot B$. Откуда следует, что $X = (A^T A)^{-1} \cdot A^T \cdot B$ псевдорешение исходной системы.

Матрица $(A^T A)^{-1} \cdot A^T$ называется псевдообратной матрицей к матрице A. Псевдорешение — это приближенное решение исходной системы, которое дает минимальную евклидову норму невязки $\|AX - B\|_2$.

В нашей задаче $A = \begin{pmatrix} 40 & 1 \\ 50 & 1 \\ 60 & 1 \\ 70 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 20 \\ 10 \\ 40 \\ 30 \end{pmatrix}$. Найденное решение по формуле $X = (A^T A)^{-1} \cdot A^T \cdot B$ представляет собой матрицу-строку $(0.6 \quad -8)$. Тогда искомая функция примет вид $y = 0.6x - 8$. График этой функции представлен на рисунке.

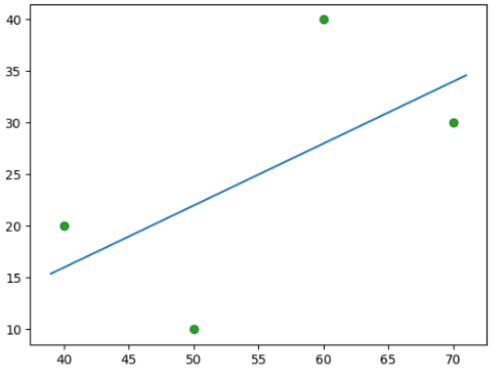

Рисунок 8. График оптимальной траектории

Решение систем линейных уравнений.

Выше был разобран один из вариантов поиска приближенного решения переопределенной системы линейных уравнений. Далее разбираем решения квадратной системы линейных уравнений, которая имеет решения.

Везде далее рассматриваем решение системы линейных уравнений вида: $\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_2 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{1n}x_1 + a_{2n}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$

или в матричном виде: $AX = B$, где на матрицу А будем накладывать определенные условия. Будем считать, что эта система имеет единственное решение.

Из курса линейной алгебры вам известны методы точного нахождения решения системы: метод Крамера, метод Гаусса, метод обратной матрицы.

В большинстве случаев решение линейной системы уравнений «руками» представляет определенные вычислительные сложности и становится труднореализуемо при большом количестве неизвестных. В данном курсе вы познакомитесь с численными методами решения линейных систем.

Решение треугольной системы линейных уравнений.

Наиболее просто линейная система решается, когда матрица A приведена к треугольному виду с единицами по главной диагонали (к унитреугольному виду). Напомню, что матрицу A всегда можно привести к подобному виду, если определитель матрицы A не равен 0.

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ a_{21} & 1 & \dots & 0 \\ & & \dots & \\ a_{n1} & a_{n2} & \dots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Решение системы прямой подстановкой:

$$\begin{cases} x_1 = b_1 \\ x_2 = b_2 - a_{21} x_1 \\ \dots \\ x_i = b_i - \sum_{k=1}^{i-1} a_{ik} x_k \end{cases}$$

Алгоритм решения в векторном виде приведен в приложенном файле:

```
A = np.array([[1, 0, 0], [3, 1, 0], [-4, 5, 1]], int) # Матрица (левая часть системы)
B = np.array([2, 4, 3], int) # Вектор (правая часть системы)
x = np.zeros((3,1), int)

x[0] = B[0]

for i in range(1,3):
    x[i] = B[i] - np.dot(A[i, :i], x[ :i])


print(x)
```

Рисунок 7. Решение треугольной системы

В цикле умножаем элементы i-той строки матрицы A до главной диагонали на часть вектора x.

Также возможно решение без введения переменной x, хранящей решение системы.

Аналогичным образом можно решить систему вида

$$\begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ & & \dots & \\ 0 & 0 & \dots & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Решение системы обратной подстановкой:

$$\begin{cases} x_n = b_n / u_{nn} \\ x_{n-1} = (b_{n-1} - u_{n-1n} x_n) / u_{n-1,n-1} \\ \dots \\ x_i = (b_i - \sum_{k=i+1}^{n} u_{ik} x_k) / u_{ii} \end{cases}$$

```
A  = np.array([[2,3,4], [0,4,5], [0,0,3]], float)
B = np.array([2,3,4], float)
n = np.size(B)
x = np.zeros_like(B)

x[-1] = B[-1] / A[-1, -1]
for i in range(n-2, -1, -1):
   x[i] =  (B[i] - np.sum(A[i, i+1:] * x[i+1:]))/A[i,i]
x
```

```
array([-0.2917, -0.9167,  1.3333])
```

```
np.linalg.solve(A,B)
```

```
array([-0.2917, -0.9167,  1.3333])
```

Рисунок 8. Реализация метода обратной подстановки

Решение системы линейных уравнений с помощью LU разложения.

LU разложение – это представление матрицы A в виде произведения двух матриц: L – нижняя унитреугольная, U – верхняя треугольная. LU-разложение используется для решения систем линейных уравнений, обращения матриц и вычисления определителя. LU-разложение существует только в том случае, когда матрица A обратима (невырождена), и все ведущие (угловые) главные миноры матрицы A невырождены.

Допустим, нам удалось найти матрицы L и U такие, что A = LU. Очевидно, это возможно только в случае невырожденной матрицы A. Тогда $a_{11} = l_{11} \cdot u_{11}$. Предположим, что $a_{11} = 0$, но тогда $l_{11} = 0$ или $u_{11} = 0$, что означает равенство нулю первой строки матрицы L или первого столбца матрицы U, из чего следует их вырожденность.

Итак, далее считаем, что $a_{11} \neq 0$. Представим матрицу A как блочную матрицу: $A = \begin{pmatrix} a_{11} & w \\ v & A' \end{pmatrix}$, где w – вектор-строка $w = (a_{12}, \ldots a_{1n})$ размера 1x(n-1), v – вектор-столбец $v = \begin{pmatrix} a_{21} \\ \vdots \\ a_{n1} \end{pmatrix}$ размера (n-1)x1, A' - матрица (минор) размера (n-1)x(n-1). Аналогичным образом представим матрицы L и U:

$L = \begin{pmatrix} 1 & 0 \\ v_L & L' \end{pmatrix}$, $U = \begin{pmatrix} a_{11} & w_U \\ 0 & U' \end{pmatrix}$. Тогда:

$$A = LU = \begin{pmatrix} 1 & 0 \\ v_L & L' \end{pmatrix} \cdot \begin{pmatrix} a_{11} & w_U \\ 0 & U' \end{pmatrix} = \begin{pmatrix} a_{11} & w_u \\ v_L \cdot a_{11} & v_L \cdot w_U + L' \cdot U' \end{pmatrix} = \begin{pmatrix} a_{11} & w \\ v & A' \end{pmatrix}$$

.

Заметим, что умножение $v_L \cdot w_U$ - это умножение столбца размера (n-1)x1 на строку размера 1x(n-1), что влечет получение матрицы размера (n-1)x(n-1). Получили формулы нахождения матриц L и U:

$$w_U = w$$
$$v_L = v / a_{11} \quad (\textit{деление вектора – столбца на число})$$
$$L' \cdot U' = A' - \frac{v \cdot w}{a_{11}}$$

Итак, нахождение LU разложения для матрицы A сведено к нахождению LU разложения для матрицы $L' \cdot U'$ размера (n-1)x(n-1). Выражение $A' - \frac{v \cdot w}{a_{11}}$ называется дополнением Шура элемента a_{11} в матрице A. Приведем алгоритм нахождения LU – разложения на языке Python.

```
#алгоритм с вики
U = np.zeros((n,n), float)
L = np.identity(n, float)

for i in range(n):
    for j in range(n):
        if i<=j:
            U[i,j] = A1[i,j] -  np.dot(L[i, :i], U[ :i, j])
        if i>j:
            L[i,j] = (A1[i,j] -  np.dot(L[i, :j], U[ :j, j]) )/ U[j,j]
```

Рисунок 9. LU разложение матрицы

Полученное LU-разложение матрицы A (матрица коэффициентов системы) может быть использовано для решения семейства систем линейных уравнений с различными векторами b в правой части: Ax=b

Если известно LU-разложение матрицы A=LU, исходная система может быть записана как: LUx=b.

Эта система может быть решена в два шага. На первом шаге решается система Ly=b. Поскольку L — нижняя треугольная матрица, эта система решается непосредственно прямой подстановкой.

На втором шаге решается система Ux=y. Поскольку U — верхняя треугольная матрица, эта система решается обратной подстановкой.

План решения системы AX = b с помощью LU разложения.

1. Записать систему в виде AX = b
2. Найти LU разложение матрицы A самостоятельно написанной функцией.
3. Записать систему в виде LUX = b.
4. Решить систему Ly = b. Решить эту систему самостоятельно написанной функцией прямой подстановки.
5. Решить систему UX = y самостоятельно написанным методом обратной подстановки.
6. Вывести решение системы X.
7. Проверить найденное решение с помощью функции np.linalg.solve().

Решение системы линейных уравнений с помощью QR разложения.

Невырожденная матрица A размера nxn с комплексными элементами может быть представлена в виде: A=QR, где Q —

унитарная матрица размера nxn, а R — верхнетреугольная матрица размера nxn.

В случае, когда матрица A состоит из вещественных чисел, её можно представить в виде A=QR, где Q является ортогональной матрицей размера nxn, а R — верхнетреугольная матрица размера nxn.

Напомню, что матрица A является ортогональной матрицей, тогда и только тогда, когда выполняется одно из следующих эквивалентных условий: 1) строки матрицы A образуют ортонормированный базис, 2) столбцы матрицы A образуют ортонормированный базис, 3) $A^{-1} = A^T$, 4) $A \cdot A^T = E$.

Есть несколько способов нахождения QR разложения матрицы: метод Грама-Шмидта, метод Гивенса (метод вращений) и метод Хаусхолдера (метод отражений). Программная реализация этих методов приведена на языке Python в файле к лабораторной работе.

```
def gram_schmidt(A):
    Q=np.zeros_like(A)
    cnt = 0
    for a in A.T:
        u = np.copy(a)
        for i in range(0, cnt):
            u -= np.dot(np.dot(Q[:, i].T, a), Q[:, i]) #
        e = u / np.linalg.norm(u)  # Нормализован
        Q[:, cnt] = e
        cnt += 1
    R = np.dot(Q.T, A)
    return (Q, R)
```

```
A  = np.array([[1,2,3], [2,3,-4], [9,5,6]], dtype=float)
```

```
gram_schmidt(A)
```

```
(array([[ 0.1078,  0.599 ,  0.7935],
        [ 0.2157,  0.765 , -0.6068],
        [ 0.9705, -0.2366,  0.0467]]),
 array([[ 9.2736,  5.7151,  5.2838],
        [-0.    ,  2.3102, -2.6827],
        [ 0.    ,  0.    ,  5.0877]]))
```

Рисунок 10. Реализация метода Грама-Шмидта

```
def givens_rotation(A):
    (r, c) = np.shape(A)
    Q = np.identity(r)
    R = np.copy(A)
    (rows, cols) = np.tril_indices(r, -1, c)
    for (row, col) in zip(rows, cols):
        if R[row, col] != 0:  # Q = 1, S = 0, R, Q без изменений
            r_ = np.hypot(R[col, col], R[row, col])  # d
            c = R[col, col]/r_
            s = -R[row, col]/r_
            G = np.identity(r)
            G[[col, row], [col, row]] = c
            G[row, col] = s
            G[col, row] = -s
            R = np.dot(G, R)  # R=G(n-1,n)*...*G(2n)*...*G(23,1n)*...*G(12)*A
            Q = np.dot(Q, G.T)  # Q=G(n-1,n).T*...*G(2n).T*...*G(23,1n).T*...*G(12).T
    return (Q, R)
```

```
givens_rotation(A)
```

```
(array([[ 0.1078,  0.599 , -0.7935],
        [ 0.2157,  0.765 ,  0.6068],
        [ 0.9705, -0.2366, -0.0467]]),
 array([[ 9.2736,  5.7151,  5.2838],
        [-0.    ,  2.3102, -2.6827],
        [-0.    ,  0.    , -5.0877]]))
```

Рисунок 11. Реализация метода Гивенса

```python
def householder_reflection(A):

    (r, c) = np.shape(A)
    Q = np.identity(r)
    R = np.copy(A)
    for cnt in range(r - 1):
        x = R[cnt:, cnt]
        e = np.zeros_like(x)
        e[0] = np.linalg.norm(x)
        u = x - e
        v = u / np.linalg.norm(u)
        Q_cnt = np.identity(r)
        Q_cnt[cnt:, cnt:] -= 2.0 * np.outer(v, v)
        R = np.dot(Q_cnt, R)  # R=H(n-1)*...*H(2)*H(1)*A
        Q = np.dot(Q, Q_cnt)  # Q = H (N-1) * ... * H (2) * H (1) H - матрица
    return (Q, R)

```

```python
householder_reflection(A)
```

```
(array([[ 0.1409396 ,  0.54013543, -0.82969256],
        [ 0.2147651 ,  0.80142062,  0.55821228],
        [ 0.96644295, -0.25686322, -0.00305034]]),
 array([[ 9.26845638e+00,  5.75838926e+00,  5.36241611e+00],
        [-1.68792321e-01,  2.20021661e+00, -3.12645551e+00],
        [ 2.59278926e-01, -6.85205592e-16, -4.74022884e+00]]))
```

Рисунок 12. Реализация метода Хаусхолдера

Решим исходную линейную систему с помощью QR разложения матрицы. Обратим внимание на то, что если QR разложение уже найдено, то найти обратную матрицу к Q не составит труда – это просто транспонированная матрица (или сопряженная, в комплексном случае). Итак, $AX = B$, используем разложение $QRX = B$, умножаем на транспонированную $Q^T QRX = Q^T B$, получаем верхнетреугольную систему $RX = Q^T B$, которую легко решить обратной подстановкой.

План решения системы AX = b с помощью QR разложения.

1. Записать систему в виде AX = b
2. Найти QR разложение матрицы A самостоятельно написанной функцией.

3. Проверить правильность найденного QR разлржения путем перемножения матриц и с помощью библиотечной функции.
4. Записать систему в виде QRX = b.
5. Решить систему Qy = b, домножением на матрицу Q^T.
6. Решить систему RX = y самостоятельно написанным методом обратной подстановки.
7. Вывести решение системы X.
8. Проверить найденное решение с помощью функции np.linalg.solve().

Решение системы линейных уравнений итерационным методом Якоби (метод простых итераций)

Напомним, что исходная система имеет вид: $\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$. Допустим, что все коэффициенты на главной диагонали не равны нулю (в противном случае можно переобозначить переменные). Выразим из первого уравнения x_1,

из второго уравнения x_2 и так далее. Получим систему

$$\begin{cases} x_1 = \dfrac{1}{a_{11}}(-a_{12}x_2 - \ldots - a_{1n}x_n + b_1) \\ x_2 = \dfrac{1}{a_{22}}(-a_{12}x_1 - \ldots - a_{2n}x_n + b_2) \\ \ldots \\ x_n = \dfrac{1}{a_{nn}}(-a_{n1}x_1 - \ldots - a_{nn-1}x_{n-1} + b_n) \end{cases}$$

Выберем начальное приближение $x^0 = (x_1^0, x_2^0, \ldots x_n^0)$, подставим в правую часть системы, получим новое решение $x^1 = (x_1^1, x_2^1, \ldots x_n^1)$. Таким образом, имеем итерационный процесс:

$$\begin{cases} {x_1}^{k+1} = \dfrac{1}{a_{11}}(-a_{12}{x_2}^k - \ldots - a_{1n}{x_n}^k + b_1) \\ {x_2}^{k+1} = \dfrac{1}{a_{22}}(-a_{12}{x_1}^k - \ldots - a_{2n}{x_n}^k + b_2) \\ \ldots \\ {x_n}^{k+1} = \dfrac{1}{a_{nn}}(-a_{n1}{x_1}^k - \ldots - a_{nn-1}{x_{n-1}}^k + b_n) \end{cases} .$$

Обычно записывают в матричном виде: $X^{k+1} = BX^k + C$, где B - квадратная матрица с нулями по главной диагонали, на остальных местах элементы вида $b_{ij} = \dfrac{-a_{ij}}{a_{ii}}$, C - вектор-столбец с элементами $c_i = \dfrac{b_i}{a_{ii}}$. Достаточным условием сходимости итерационного процесса является условие диагонального

преобладания матрицы A : $\sum_{j,i\neq j}\left|a_{ij}\right| < \left|a_{ii}\right|$ (для любой строки i) или 1также можно показать, что для сходимости метода достаточно, чтобы любая норма матрицы B была меньше 1.

Пример. Решим систему $\begin{cases} 100x_1 + 30x_2 - 70x_3 = 60 \\ 15x_1 - 50x_2 - 5x_3 = -40 \\ 6x_1 + 2x_2 + 20x_3 = 28 \end{cases}$ с точностью до $\varepsilon = 0.01$. Заметим, что матрица $A = \begin{pmatrix} 100 & 30 & -70 \\ 15 & -50 & -5 \\ 6 & 2 & 20 \end{pmatrix}$ не удовлетворяет условию диагонального преобладания, но в данной системе достаточно прибавить к первому уравнению второе:

$$\begin{cases} 115x_1 - 20x_2 - 75x_3 = 20 \\ 15x_1 - 50x_2 - 5x_3 = -40 \\ 6x_1 + 2x_2 + 20x_3 = 28 \end{cases} \Rightarrow \begin{cases} x_1 = 20/115 + 20/115x_2 + 75/115x_3 \\ x_2 = 40/50 + 15/50x_1 - 5/50x_3 \\ x_3 = 28/20 - 6/20x_1 - 2/20x_2 \end{cases}$$

Получили систему, удовлетворяющую достаточному условию сходимости метода Якоби (простых итераций). Результат выполнения итераций представлен на рисунке:

k	x_1	x_2	x_3	norma(x_k- x_k-1)
0	0.0	0.0	0.0	
1	0.17391304347826086	0.8	1.4	1.4
2	1.2260869565217392	0.7121739130434783	1.2678260869565217	1.0521739130434784
3	1.1246124763705103	1.0410434782608697	0.9609565217391303	0.3288695652173914
4	0.9816748582230623	1.0412880907372402	0.9585119092627599	0.142937618147448
5	0.9801230870387113	0.9986512665406427	1.0013687334593573	0.04285682419659742
6	1.0006580899153448	0.9939000527656776	1.0060979472343223	0.020535002876633457
7	1.0029160617207629	0.9995876322511712	1.0004125677488287	0.005687579485493588
8	1.0001973497929182	1.000833561741346	0.9991664182586539	0.002718711927844674

Рисунок 13. Пример работы метода Якоби

В левом столбце указан номер итерации. В последней строке набор координат (x_1, x_2, x_3) является приближенным решением исходной системы, найденный с указанной точностью. В последнем столбце выводится норма разности текущей итерации и предыдущей. Вывод таблицы оформлен в Python с помощью модуля PrettyTable. В лабораторной работе допускаются и другие варианты оформления вывода результатов.

Итерационный метод Зейделя.

Метод Зейделя является модификацией метода Якоби. В отличии от метода Якоби, в методе Зейделя при вычислении x_i^{k+1} используются уже найденные на $k+1$ итерации значения x_1^{k+1},…, x_{i-1}^{k+1}, то есть итерационный процесс выглядит следующим образом: $\begin{cases} x_1{}^{k+1} = \frac{1}{a_{11}}(-a_{12}x_2{}^{k} - \ldots - a_{1n}x_n{}^{k} + b_1) \\ x_2{}^{k+1} = \frac{1}{a_{22}}(-a_{12}x_1{}^{k+1} - \ldots - a_{2n}x_n{}^{k} + b_2) \\ \ldots \\ x_n{}^{k+1} = \frac{1}{a_{nn}}(-a_{n1}x_1{}^{k+1} - \ldots - a_{nn-1}x_{n-1}{}^{k+1} + b_n) \end{cases}$, или в матричном виде: $X^{k+1} = B_1 X^{k+1} + B_2 X^{k} + C$, где $B = B_1 + B_2$, B_1 нижнетреугольная матрица, B_2 верхнетреугольная матрица. Для сходимости метода Зейделя достаточно, чтобы $\max|b_{ij}| < 1$. Условием остановки итерационного процесса может служить условие $\left\|X^{k+1} - X^{k}\right\|_{\infty} < \varepsilon$, где ε заданная точность. Метод Зейделя сходится, как правило, быстрее метода Якоби.

Пример. Решим пример из предыдущего пункта методом Зейделя. $\begin{cases} x_1 = 20/115 + 20/115x_2 + 75/115x_3 \\ x_2 = 40/50 + 15/50x_1 - 5/50x_3 \\ x_3 = 28/20 - 6/20x_1 - 2/20x_2 \end{cases}$.

Подставляем уже пересчитанные значения вектора неизвестных x: $\begin{cases} x_1^{k+1} = 20/115 + 20/115x_2^{k} + 75/115x_3^{k} \\ x_2^{k+1} = 40/50 + 15/50x_1^{k+1} - 5/50x_3^{k} \\ x_3^{k+1} = 28/20 - 6/20x_1^{k+1} - 2/20x_2^{k+1} \end{cases}$.

В матричном виде:

$$X^{k+1} = \begin{pmatrix} 20/115 \\ 40/50 \\ 28/20 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 15/50 & 0 & 0 \\ -6/20 & -2/20 & 0 \end{pmatrix} X^{k+1} + \begin{pmatrix} 0 & 20/115 & 75/115 \\ 0 & 0 & -5/50 \\ 0 & 0 & 0 \end{pmatrix} X^{k}$$

Результат итераций представлен на рисунке:

k	x_1	x_2	x_3	norma(x_k- x_k-1)
0	0.0	0.0	0.0	
1	0.9996013269932693	1.00014256311201	0.9998574388879898	1.00014256311201
2	0.9999318189464299	0.9998946542091818	1.000105345590818 2	0.0003304919531605943

Рисунок 14. Метод Зейделя

Метод Зейделя для этого примера сошёлся гораздо быстрее метода Якоби.

Контрольные вопросы к лабораторной работе.

1. Определение нормы.
2. Часто используемые нормы векторов.
3. Часто используемые нормы матриц.

4. Особенности индексации в матрицах\массивах в Python
5. Что такое переопределенная система линейных уравнений
6. В каких задачах возникает переопределенная система линейных уравнений
7. Псевдообратная матрица, методы поиска псевдорешения линейной системы.
8. Определение линейной системы уравнений.
9. Методы решения систем: точные, численные, итерационные.
10. Верхнетреугольная, нижнетреугольная, унитреугольная матрицы, команды для создания таких матриц.
11. Определение LU разложения, достаточное условие существования LU разложения.
12. Определение QR разложения матрицы, достаточное условие существования QR разложения.
13. Идея решения системы линейных уравнений с помощью LU разложения, с помощью QR разложения.
14. Итерационные методы решения линейных систем. Достаточное условие сходимости метода Якоби, метода Зейделя.
15. Сходство и различие методов Якоби и Зейделя.

Вариант 1.

1. Создать квадратную матрицу из случайных целых чисел из [-3,3] размера 10. Найти и вычислить минор 4 порядка,

расположенный на пересечении 2,3,4,5 строк и 7,8,9, 10 столбцов. Использовать срезы матрицы.

2. Создать вектор-строку 1x10 из случайных целых чисел. Вычислить норму $\|x\|_1$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти спектральную норму матрицы с помощью самостоятельно написанного алгоритма (можно использовать функцию для нахождения собственных значений), проверить результат с помощью linalg.norm() Python.

4. Найти псевдорешение системы

$$\begin{cases} 4.4x_1 - 2.5x_2 + 19.2x_3 - 10.8x_4 = 4.3 \\ 5.5x_1 - 9.3x_2 - 14.2x_3 + 13.2x_4 = 6.8 \\ 7.1x_1 - 11.5x_2 + 5.3x_3 - 6.7x_4 = -1.8 \\ 14.2x_1 + 23.4x_2 - 8.8x_3 + 5.3x_4 = 7.2 \\ 8.2x_1 - 3.2x_2 + 14.2x_3 + 14.8x_4 = -8.4 \end{cases}$$

5. Даны пять точек (20; 15), (30; 40), (40; 21), (50, 65), (60, 54). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 4.4x_1 - 2.5x_2 + 19.2x_3 - 10.8x_4 = 4.3 \\ 5.5x_1 - 9.3x_2 - 14.2x_3 + 13.2x_4 = 6.8 \\ 7.1x_1 - 11.5x_2 + 5.3x_3 - 6.7x_4 = -1.8 \\ 14.2x_1 + 23.4x_2 - 8.8x_3 + 5.3x_4 = 7.2 \end{cases}$.

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Хаусхолдера. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом простых итераций с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 2.7x_1 + 3.3x_2 + 10.3x_3 = 2.1 \\ 6.5x_1 - 1.7x_2 + 2.8x_3 = 1.7 \\ 4.1x_1 + 7.8x_2 - 1.7x_3 = 0.8 \end{cases}$$

Вариант 2.

1. Создать квадратную матрицу из случайных вещественных чисел размера 7. Найти скалярное произведение 4 строки на 5 столбец. Использовать срезы матриц.

2. Создать вектор-строку 1x10 из случайных целых чисел. Вычислить норму $\|x\|_2$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти норму матрицы $\|A\|_\infty$ с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python.

4. Найти псевдорешение системы
$$\begin{cases} 3.1x_1 + 2.8x_2 + 1.9x_3 = 0.2 \\ 1.9x_1 + 3.1x_2 + 2.1x_3 = 2.1 \\ 7.5x_1 + 3.8x_2 + 4.8x_3 = 5.6 \\ 3.01x_1 - 0.33x_2 + 0.11x_3 = 0.13 \end{cases}$$

5. Даны пять точек (3; 7), (5; 2), (7; 10), (9, 1), (10, 8). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python

$$\begin{cases} 8.2x_1 - 3.2x_2 + 14.2x_3 + 14.8x_4 = -8.4 \\ 5.6x_1 - 12x_2 + 15x_3 - 6.4x_4 = 4.5 \\ 5.7x_1 + 3.6x_2 - 12.4x_3 - 2.3x_4 = 3.3 \\ 6.8x_1 + 13.2x_2 - 6.3x_3 - 8.7x_4 = 14.3 \end{cases}$$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Грама-Шмидта. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом простых итераций с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 2.1x_1 + 2.8x_2 + 8.9x_3 = 0.2 \\ 1.9x_1 + 5.1x_2 + 2.1x_3 = 2.1 \\ 7.5x_1 + 3.8x_2 + 4.8x_3 = 5.6 \end{cases}$$

Вариант 3.

1. Создать квадратную матрицу из случайных целых чисел из [-5,2] размера 11. Найти и вычислить минор 5 порядка, расположенный на пересечении 1,2,3,6,7 строк и 7,8,9, 10,11 столбцов. Использовать срезы матрицы.

2. Создать вектор-строку 1x8 из случайных целых чисел. Вычислить норму $\|x\|_\infty$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти фробениусову норму матрицы с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python.

4. Найти псевдорешение системы
$$\begin{cases} 5.7x_1 - 7.8x_2 - 5.6x_3 = 2.7 \\ 6.6x_1 + 13.1x_2 - 6.3x_3 = -5.5 \\ 14.7x_1 - 2.8x_2 + 5.6x_3 = 8.6 \\ 8.5x_1 + 12.7x_2 - 23.7x_3 = 14.7 \end{cases}$$

5. Даны пять точек (30; 17), (40; 22), (50; 40), (60, 37), (70, 29). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 5.7x_1 - 7.8x_2 - 5.6x_3 - 8.3x_4 = 2.7 \\ 6.6x_1 + 13.1x_2 - 6.3x_3 + 4.3x_4 = -5.5 \\ 14.7x_1 - 2.8x_2 + 5.6x_3 - 12.1x_4 = 8.6 \\ 8.5x_1 + 12.7x_2 - 23.7x_3 + 5.7x_4 = 14.7 \end{cases}$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Гивенса. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом Зейделя с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 0.21x_1 - 0.18x_2 + 0.75x_3 = 0.11 \\ 0.13x_1 + 0.9x_2 - 0.11x_3 = 2 \\ 3.01x_1 - 0.33x_2 + 0.11x_3 = 0.13 \end{cases}$$

Вариант 4.

1. Создать квадратную матрицу из случайных вещественных чисел размера 10 . Найти скалярное произведение 2 строки на 7 столбец. Использовать срезы матриц.

2. Создать вектор-строку 1x10 из случайных целых чисел. Вычислить норму $\|x\|_4$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти норму матрицы $\|A\|_\infty$ с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python.

4. Найти псевдорешение системы

$$\begin{cases} 3.3x_1 + 2.1x_2 + 2.8x_3 = 0.8 \\ 4.1x_1 + 3.7x_2 + 4.8x_3 = 5.7 \\ 2.7x_1 + 1.8x_2 + 1.1x_3 = 3.2 \\ 3.15x_1 - 1.72x_2 - 1.23x_3 = 2.15 \end{cases}$$

5. Даны пять точек (-4; 7), (-2; 1), (0; -3), (4, -8), (6, -9). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python.

$$\begin{cases} 3.8x_1 + 14.2x_2 + 6.3x_3 - 15.5x_4 = 2.8 \\ 8.3x_1 - 6.6x_2 + 5.8x_3 + 12.2x_4 = -4.7 \\ 6.4x_1 - 8.5x_2 - 4.3x_3 + 8.8x_4 = 7.7 \\ 17.1x_1 - 8.3x_2 + 14.4x_3 - 7.2x_4 = 13.5 \end{cases}$$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Хаусхолдера. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом простых итераций с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 7.3x_1 + 2.1x_2 + 2.8x_3 = 0.8 \\ 4.1x_1 + 8.7x_2 + 4.8x_3 = 5.7 \\ 2.7x_1 + 1.8x_2 + 9.1x_3 = 3.2 \end{cases}$$

Вариант 5.

1. Создать квадратную матрицу из случайных целых чисел из [0,8] размера 6. Создать две новые матрицы: первая – из двух последних строк исходной матрицы (должна получиться матрица размера 2x6), вторая – из двух первых столбцов матрицы (матрица размера 6x2).

2. Создать вектор-строку 1x10 из случайных целых чисел. Вычислить норму $\|x\|_3$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти норму матрицы Фробениуса с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python .

4. Найти псевдорешение системы
$$\begin{cases} 15.7x_1 + 6.6x_2 - 5.7x_3 = -2.4 \\ 8.8x_1 - 6.7x_2 + 5.5x_3 = 5.6 \\ 6.3x_1 - 5.7x_2 - 23.4x_3 = 7.7 \\ 14.3x_1 + 8.7x_2 - 15.7x_3 = 23.4 \end{cases}$$

5. Даны пять точек (20; 19), (25; 16), (30; 27), (35, 24), (40, 31). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python.

$$\begin{cases} 15.7x_1 + 6.6x_2 - 5.7x_3 + 11.5x_4 = -2.4 \\ 8.8x_1 - 6.7x_2 + 5.5x_3 - 4.5x_4 = 5.6 \\ 6.3x_1 - 5.7x_2 - 23.4x_3 + 6.6x_4 = 7.7 \\ 14.3x_1 + 8.7x_2 - 15.7x_3 - 5.8x_4 = 23.4 \end{cases}$$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Грама-Шмидта. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом Зейделя с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 5.15x_1 - 1.72x_2 - 1.23x_3 = 2.15 \\ 0.72x_1 + 0.67x_2 + 2.18x_3 = 1.43 \\ 2.57x_1 - 6.34x_2 - 0.68x_3 = 1.03 \end{cases}$$

Вариант 6.

1. Создать квадратную матрицу из случайных целых чисел из [-7,-2] размера 9. Найти и вычислить минор 4 порядка, расположенный на пересечении 1,2,7,8 строк и 2,4,6, 9 столбцов. Использовать срезы матрицы.

2. Создать вектор-строку 1x8 из случайных целых чисел. Вычислить норму $\|x\|_1$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python .

3. Создать матрицу из случайных целых чисел. Найти спектральную норму матрицы с помощью самостоятельно написанного алгоритма (можно использовать функцию для нахождения собственных значений), проверить результат с помощью linalg.norm() в Python.

4. Найти псевдорешение системы
$$\begin{cases} 14.4x_1 - 5.3x_2 + 14.3x_3 = 14.4 \\ 23.4x_1 - 14.2x_2 - 5.4x_3 = 6.6 \\ 6.3x_1 - 13.2x_2 - 6.5x_3 = 9.4 \\ 5.6x_1 + 8.8x_2 - 6.7x_3 = 7.3 \end{cases}.$$

5. Даны пять точек (20; 15), (30; 40), (40; 21), (50, 65), (60, 54). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 4.3x_1 - 12.1x_2 + 23.2x_3 - 14.1x_4 = 15.5 \\ 2.4x_1 - 4.4x_2 + 3.5x_3 + 5.5x_4 = 2.5 \\ 5.4x_1 + 8.3x_2 - 7.4x_3 - 12.7x_4 = 8.6 \\ 6.3x_1 - 7.6x_2 + 1.34x_3 + 3.7x_4 = 12.1 \end{cases}$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Гивенса. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом простых итераций с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 2.7x_1 + 3.3x_2 + 10.3x_3 = 2.1 \\ 6.5x_1 - 1.7x_2 + 2.8x_3 = 1.7 \\ 4.1x_1 + 7.8x_2 - 1.7x_3 = 0.8 \end{cases}$$

Вариант 7.

1. Создать квадратную матрицу из случайных вещественных чисел из (2,4) размера 8. Найти скалярное произведение 4 строки на 7 столбец. Использовать срезы матриц.

2. Создать вектор-строку 1x9 из случайных целых чисел. Вычислить норму $\|x\|_2$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти норму $\|A\|_1$ матрицы с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python.

4. Найти псевдорешение системы $\begin{cases} 3.6x_1 + 1.8x_2 - 4.7x_3 = 3.8 \\ 2.7x_1 - 3.6x_2 + 1.9x_3 = 0.4 \\ 1.5x_1 + 4.5x_2 + 3.3x_3 = -1.6 \\ 5.6x_1 + 8.8x_2 - 6.7x_3 = 7.3 \end{cases}$

5. Даны пять точек (3; 7), (5; 2), (7; 10), (9, 1), (10, 8). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 4.4x_1 - 2.5x_2 + 19.2x_3 - 10.8x_4 = 4.3 \\ 5.5x_1 - 9.3x_2 - 14.2x_3 + 13.2x_4 = 6.8 \\ 7.1x_1 - 11.5x_2 + 5.3x_3 - 6.7x_4 = -1.8 \\ 14.2x_1 + 23.4x_2 - 8.8x_3 + 5.3x_4 = 7.2 \end{cases}$.

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Хаусхолдера. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом Зейделя с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 5.15x_1 - 1.72x_2 - 1.23x_3 = 2.15 \\ 0.72x_1 + 0.67x_2 + 4.18x_3 = 1.43 \\ 3.57x_1 - 1.34x_2 - 0.68x_3 = 1.03 \end{cases}$$

Вариант 8.

1. Создать квадратную матрицу из случайных целых чисел из [-4,3] размера 10. Найти и вычислить минор 5 порядка,

расположенный на пересечении 1-5 строк и 2-4,9,10 столбцов. Использовать срезы матрицы.

2. Создать вектор-строку 1x5 из случайных целых чисел. Вычислить норму $\|x\|_5$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python и функцией norm() в Octave.

3. Создать матрицу из случайных целых чисел. Найти норму $\|A\|_\infty$ матрицы с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python и функцией norm() в Octave.

4. Найти псевдорешение системы
$$\begin{cases} 5.7x_1 - 7.8x_2 - 5.6x_3 = 2.7 \\ 6.6x_1 + 13.1x_2 - 6.3x_3 = -5.5 \\ 14.7x_1 - 2.8x_2 + 5.6x_3 = 8.6 \\ 8.5x_1 + 12.7x_2 - 23.7x_3 = 14.7 \end{cases}$$

5. Даны пять точек (30; 17), (40; 22), (50; 40), (60, 37), (70, 29). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 8.2x_1 - 3.2x_2 + 14.2x_3 + 14.8x_4 = -8.4 \\ 5.6x_1 - 12x_2 + 15x_3 - 6.4x_4 = 4.5 \\ 5.7x_1 + 3.6x_2 - 12.4x_3 - 2.3x_4 = 3.3 \\ 6.8x_1 + 13.2x_2 - 6.3x_3 - 8.7x_4 = 14.3 \end{cases}$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Грама-Шмидта. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом простых итераций с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 2.7x_1 + 3.3x_2 + 10.3x_3 = 2.1 \\ 6.5x_1 - 1.7x_2 + 2.8x_3 = 1.7 \\ 4.1x_1 + 7.8x_2 - 1.7x_3 = 0.8 \end{cases}$$

Вариант 9.

1. Создать квадратную матрицу из случайных вещественных чисел из интервала (-1,1) размера 8 . Найти скалярное произведение 1 строки на 8 столбец. Использовать срезы матриц.

2. Создать вектор-строку 1x6 из случайных целых чисел. Вычислить норму $\|x\|_2$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти норму матрицы Фробениуса с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python и функцией norm() в Octave.

4. Найти псевдорешение системы
$$\begin{cases} 3.3x_1 + 2.1x_2 + 2.8x_3 = 0.8 \\ 4.1x_1 + 3.7x_2 + 4.8x_3 = 5.7 \\ 2.7x_1 + 1.8x_2 + 1.1x_3 = 3.2 \\ 3.15x_1 - 1.72x_2 - 1.23x_3 = 2.15 \end{cases}$$

5. Даны пять точек (4; 7), (5; 1), (6; 10), (7, 8), (8, 12). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 5.7x_1 - 7.8x_2 - 5.6x_3 - 8.3x_4 = 2.7 \\ 6.6x_1 + 13.1x_2 - 6.3x_3 + 4.3x_4 = -5.5 \\ 14.7x_1 - 2.8x_2 + 5.6x_3 - 12.1x_4 = 8.6 \\ 8.5x_1 + 12.7x_2 - 23.7x_3 + 5.7x_4 = 14.7 \end{cases}$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Гивенса. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом Зейделя с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 3.1x_1 + 1.8x_2 + 1.9x_3 = 0.2 \\ 1.9x_1 + 4.1x_2 + 2.1x_3 = 2.1 \\ 1.5x_1 + 3.8x_2 + 6.8x_3 = 5.6 \end{cases}$$

Вариант 10.

1. Создать квадратную матрицу из случайных целых чисел из [-6, 6] размера 10. Создать две новые матрицы: первая – из

трех последних строк исходной матрицы (должна получиться матрица размера 3x10), вторая – из двух первых столбцов матрицы (матрица размера 10x2). Выполнить умножение этих матриц.

2. Создать вектор-строку 1x7 из случайных целых чисел. Вычислить норму $\|x\|_3$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти норму матрицы $\|A\|_\infty$ с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python .

4. Найти псевдорешение системы
$$\begin{cases} 15.7x_1 + 6.6x_2 - 5.7x_3 = -2.4 \\ 8.8x_1 - 6.7x_2 + 5.5x_3 = 5.6 \\ 6.3x_1 - 5.7x_2 - 23.4x_3 = 7.7 \\ 14.3x_1 + 8.7x_2 - 15.7x_3 = 23.4 \end{cases}.$$

5. Даны пять точек (-10; -5), (-4; -2), (-1; 0), (3, 10), (5, 7). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 3.8x_1 + 14.2x_2 + 6.3x_3 - 15.5x_4 = 2.8 \\ 8.3x_1 - 6.6x_2 + 5.8x_3 + 12.2x_4 = -4.7 \\ 6.4x_1 - 8.5x_2 - 4.3x_3 + 8.8x_4 = 7.7 \\ 17.1x_1 - 8.3x_2 + 14.4x_3 - 7.2x_4 = 13.5 \end{cases}$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Хаусхолдера. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом простых итераций с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 0.21x_1 - 0.18x_2 + 1.75x_3 = 0.11 \\ 0.13x_1 + 0.75x_2 - 0.11x_3 = 2 \\ 3.01x_1 - 0.33x_2 + 0.11x_3 = 0.13 \end{cases}$$

Вариант 11.

1. Создать квадратную матрицу из случайных целых чисел из [-3,3] размера 10. Найти и вычислить минор 4 порядка,

расположенный на пересечении 2,3,4,5 строк и 7,8,9, 10 столбцов. Использовать срезы матрицы.

2. Создать вектор-строку 1x10 из случайных целых чисел. Вычислить норму $\|x\|_1$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти спектральную норму матрицы с помощью самостоятельно написанного алгоритма (можно использовать функцию для нахождения собственных значений), проверить результат с помощью linalg.norm()Python.

4. Найти псевдорешение системы

$$\begin{cases} 4.4x_1 - 2.5x_2 + 19.2x_3 - 10.8x_4 = 4.3 \\ 5.5x_1 - 9.3x_2 - 14.2x_3 + 13.2x_4 = 6.8 \\ 7.1x_1 - 11.5x_2 + 5.3x_3 - 6.7x_4 = -1.8 \\ 14.2x_1 + 23.4x_2 - 8.8x_3 + 5.3x_4 = 7.2 \\ 8.2x_1 - 3.2x_2 + 14.2x_3 + 14.8x_4 = -8.4 \end{cases}$$

5. Даны пять точек (20; 15), (30; 40), (40; 21), (50, 65), (60, 54). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python.

$$\begin{cases} 15.7x_1 + 6.6x_2 - 5.7x_3 + 11.5x_4 = -2.4 \\ 8.8x_1 - 6.7x_2 + 5.5x_3 - 4.5x_4 = 5.6 \\ 6.3x_1 - 5.7x_2 - 23.4x_3 + 6.6x_4 = 7.7 \\ 14.3x_1 + 8.7x_2 - 15.7x_3 - 5.8x_4 = 23.4 \end{cases}$$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Грама-Шмидта. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом Зейделя с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 3.3x_1 + 7.1x_2 + 2.8x_3 = 0.8 \\ 4.1x_1 + 3.7x_2 + 8.8x_3 = 5.7 \\ 5.7x_1 + 1.8x_2 + 1.1x_3 = 3.2 \end{cases}$$

Вариант 12.

1. Создать квадратную матрицу из случайных вещественных чисел размера 10. Найти скалярное произведение 4 строки на 5 столбец. Использовать срезы матриц.

2. Создать вектор-строку 1x10 из случайных целых чисел. Вычислить норму $\|x\|_2$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python и функцией norm() в Octave.

3. Создать матрицу из случайных целых чисел. Найти норму матрицы $\|A\|_\infty$ с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python и функцией norm() в Octave.

4. Найти псевдорешение системы
$$\begin{cases} 3.1x_1 + 2.8x_2 + 1.9x_3 = 0.2 \\ 1.9x_1 + 3.1x_2 + 2.1x_3 = 2.1 \\ 7.5x_1 + 3.8x_2 + 4.8x_3 = 5.6 \\ 3.01x_1 - 0.33x_2 + 0.11x_3 = 0.13 \end{cases}$$

5. Даны пять точек (3; 7), (5; 2), (7; 10), (9, 1), (10, 8). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 4.3x_1 - 12.1x_2 + 23.2x_3 - 14.1x_4 = 15.5 \\ 2.4x_1 - 4.4x_2 + 3.5x_3 + 5.5x_4 = 2.5 \\ 5.4x_1 + 8.3x_2 - 7.4x_3 - 12.7x_4 = 8.6 \\ 6.3x_1 - 7.6x_2 + 1.34x_3 + 3.7x_4 = 12.1 \end{cases}$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Гивенса. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом простых итераций с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 3.45x_1 - 1.72x_2 - 1.23x_3 = 2.15 \\ 0.72x_1 + 0.67x_2 + 1.78x_3 = 1.43 \\ 2.57x_1 - 0.34x_2 - 0.68x_3 = 1.03 \end{cases}$$

Вариант 13.

1. Создать квадратную матрицу из случайных целых чисел из [-5,10] размера 7. Найти и вычислить минор 5 порядка, расположенный на пересечении 1,2,3,6,7 строк и 2,3,4,5,6 столбцов. Использовать срезы матрицы.

2. Создать вектор-строку 1x8 из случайных целых чисел. Вычислить норму $\|x\|_\infty$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python и функцией norm() в Octave.

3. Создать матрицу из случайных целых чисел. Найти фробениусову норму матрицы с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python.

4. Найти псевдорешение системы
$$\begin{cases} 5.7x_1 - 7.8x_2 - 5.6x_3 = 2.7 \\ 6.6x_1 + 13.1x_2 - 6.3x_3 = -5.5 \\ 14.7x_1 - 2.8x_2 + 5.6x_3 = 8.6 \\ 8.5x_1 + 12.7x_2 - 23.7x_3 = 14.7 \end{cases}$$

5. Даны пять точек (30; 17), (40; 22), (50; 40), (60, 37), (70, 29). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

   Решение системы должно быть проверено встроенной функцией Python.
$$\begin{cases} 8.2x_1 - 3.2x_2 + 14.2x_3 + 14.8x_4 = -8.4 \\ 5.6x_1 - 12x_2 + 15x_3 - 6.4x_4 = 4.5 \\ 5.7x_1 + 3.6x_2 - 12.4x_3 - 2.3x_4 = 3.3 \\ 6.8x_1 + 13.2x_2 - 6.3x_3 - 8.7x_4 = 14.3 \end{cases}$$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Хаусхолдера. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом Зейделя с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 0.7x_1 + 3.3x_2 + 1.3x_3 = 2.1 \\ 2.1x_1 - 1.7x_2 + 4.8x_3 = 1.7 \\ 4.1x_1 + 0.8x_2 - 1.7x_3 = 0.8 \end{cases}$$

Вариант 14.

1. Создать квадратную матрицу из случайных вещественных чисел размера 10 . Найти скалярное произведение 2 строки на 7 столбец. Использовать срезы матриц.

2. Создать вектор-строку 1x10 из случайных целых чисел. Вычислить норму $\|x\|_\infty$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти норму матрицы $\|A\|_\infty$ с помощью самостоятельно написанного

алгоритма, проверить результат с помощью linalg.norm() в Python.

4. Найти псевдорешение системы

$$\begin{cases} 3.3x_1 + 2.1x_2 + 2.8x_3 = 0.8 \\ 4.1x_1 + 3.7x_2 + 4.8x_3 = 5.7 \\ 2.7x_1 + 1.8x_2 + 1.1x_3 = 3.2 \\ 3.15x_1 - 1.72x_2 - 1.23x_3 = 2.15 \end{cases}$$

5. Даны пять точек (4; 7), (5; 1), (6; 10), (7, 8), (8, 12). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

   Решение системы должно быть проверено встроенной функцией Python.

$$\begin{cases} 5.7x_1 - 7.8x_2 - 5.6x_3 - 8.3x_4 = 2.7 \\ 6.6x_1 + 13.1x_2 - 6.3x_3 + 4.3x_4 = -5.5 \\ 14.7x_1 - 2.8x_2 + 5.6x_3 - 12.1x_4 = 8.6 \\ 8.5x_1 + 12.7x_2 - 23.7x_3 + 5.7x_4 = 14.7 \end{cases}$$

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Грама-Шмидта. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом простых итераций с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 3.1x_1 + 2.8x_2 + 8.9x_3 = 0.2 \\ 1.9x_1 - 7.1x_2 + 2.1x_3 = 2.1 \\ 7.5x_1 + 3.8x_2 - 4.8x_3 = 5.6 \end{cases}$$

Вариант 15.

1. Создать квадратную матрицу из случайных целых чисел из [0,8] размера 6. Создать две новые матрицы: первая – из двух последних строк исходной матрицы (должна получиться матрица размера 2x6), вторая – из двух первых столбцов матрицы (матрица размера 6x2). Выполнить умножение матриц.

2. Создать вектор-строку 1x10 из случайных целых чисел. Вычислить норму $\|x\|_3$ самостоятельно написанной функцией и проверить результат с помощью linalg.norm() в Python.

3. Создать матрицу из случайных целых чисел. Найти норму матрицы Фробениуса с помощью самостоятельно написанного алгоритма, проверить результат с помощью linalg.norm() в Python.

4. Найти псевдорешение системы

$$\begin{cases} 15.7x_1 + 6.6x_2 - 5.7x_3 = -2.4 \\ 8.8x_1 - 6.7x_2 + 5.5x_3 = 5.6 \\ 6.3x_1 - 5.7x_2 - 23.4x_3 = 7.7 \\ 14.3x_1 + 8.7x_2 - 15.7x_3 = 23.4 \end{cases}$$

5. Даны пять точек (20; 19), (25; 16), (30; 27), (35, 24), (40, 31). Найти уравнение наиболее выгодной траектории. Построить график.

6. Решить систему, используя LU разложение матрицы. LU разложение должно быть найдено самостоятельно написанной функцией, полученные матрицы выведены на экран. Придерживаться плана решения, приведенного в пособии!

   Решение системы должно быть проверено встроенной функцией Python. $\begin{cases} 4.4x_1 - 2.5x_2 + 19.2x_3 - 10.8x_4 = 4.3 \\ 5.5x_1 - 9.3x_2 - 14.2x_3 + 13.2x_4 = 6.8 \\ 7.1x_1 - 11.5x_2 + 5.3x_3 - 6.7x_4 = -1.8 \\ 14.2x_1 + 23.4x_2 - 8.8x_3 + 5.3x_4 = 7.2 \end{cases}$.

7. Решить систему из пункта 6 с помощью QR разложения матрицы A. QR разложение найти методом Гивенса. Придерживаться плана решения, приведенного в пособии! Проверить полученное решение непосредственной подстановкой в исходную систему, а также методом np.solve.

8. Решить систему методом Зейделя с точностью до 10^{-3}. Проверить выполнение достаточного условия сходимости. Если условие не выполняется, в программе выполнить эквивалентные преобразования системы, после этого привести к удобному для итераций виду. Оформить итерации в виде таблицы (можно пользоваться модулями PrettyTable или Pandas). Проверить полученное решение.

$$\begin{cases} 2.7x_1 + 3.3x_2 + 10.3x_3 = 2.1 \\ 6.5x_1 - 1.7x_2 + 2.8x_3 = 1.7 \\ 4.1x_1 + 7.8x_2 - 1.7x_3 = 0.8 \end{cases}$$