

Линейная алгебра. Матрицы

Содержание:

Что такое матрицы? Зачем они нужны?

Матричные операции

Специальные матрицы

Линейные отображения

Обратная матрица

Матрицы и системы линейных уравнений

Определитель матрицы

Что такое матрицы? Зачем они нужны?

$$M = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \end{bmatrix}$$

Матрица — математический объект, записываемый в виде прямоугольной таблицы.

Диагональная матрица — квадратная матрица, все элементы которой, кроме диагональных, равны 0.

$$M = \begin{bmatrix} A_{1,1} & 0 & 0 \\ 0 & A_{2,2} & 0 \\ 0 & 0 & A_{3,3} \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Единичная матрица — квадратная матрица, все диагональные элементы которой равны 1, остальные элементы равны 0.

Так же как вектор является строкой или столбцом чисел, матрица это прямоугольная таблица чисел размера $m \times n$. Они бывают абсолютно любых размеров - 2×2 , 2×3 , или даже 3×1 , которая на самом деле то же самое, что вектор, столбец.

Примеры матриц

Матрица - прямоугольная таблица чисел:

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix}$$

Матрица размера $m \times n$

Матрица размера 2×2

$$\begin{pmatrix} 0 & 1 \\ -1 & 2 \end{pmatrix}$$

Матрица размера 2×3

$$\begin{pmatrix} 0 & 1 & 5 \\ -1 & 2 & -2 \end{pmatrix}$$

Матрица размера 3×1

$$\begin{pmatrix} 3 \\ -1 \\ 8 \end{pmatrix}$$

Обычно матрицы используются для решения следующих задач:

1. Набор векторов, когда у нас есть несколько векторов, и мы их записываем по столбцам, получается прямоугольная таблица из отдельных векторов. Суммарно это прямоугольник чисел.
2. Удобный, компактный способ — записывать набор векторов, потом эта матрица может использоваться для записи преобразований векторов и систем координат в векторном пространстве, так называемых линейных отображений.
3. Запись соотношений между наборами векторов (так называемых билинейных форм или попарных скалярных произведений между системами векторов)

Пример того, как работать с матрицами на языке Python.

Понадобится библиотека `numpy`. Мы импортируем ее в первой строке и создаем такую матрицу из чисел 1, 2, 3, 4, 5, 6. Здесь обращаем внимание, что в отличие от вектора мы используем не один список чисел, а как бы список списков, то есть у нас первый список это 1, 2, 3, а второй список — это 4, 5, 6, все это упаковано в еще одни квадратные скобки. Когда мы

вызываем команду `np.array`, то на выходе у нас получается не одномерный массив, как было с векторами, а такой как бы двумерный массив чисел, то есть матрица.

```
[16]: import numpy as np

[17]: A = np.array([[1, 2, 3], [4, 5, 6]])
      A
```

← Создаём матрицу 2 x 3

```
[17]: array([[1, 2, 3],
            [4, 5, 6]])
```

Среди полезных команд, которые используются для создания матриц, можно упомянуть полезную команду `zeros`, которая делает матрицу, состоящую из одних нулей нужного нам размера. В данном случае мы указываем нужный нам размер матрицы в скобках. Обращаем внимание, что скобок на самом деле две пары, здесь у нас в качестве аргумента используется питоновский кортеж. Если так не сделать, мы получим ошибку.

```
[18]: B = np.zeros((2, 2))
      B
```

← Создаём нулевую матрицу 2 x 2

```
[18]: array([[0., 0.],
            [0., 0.]])
```

Пример. Как с помощью команды `eye` создать матрицу размером 3 x 3 с единицами по диагонали и остальными нулями?

Это очень полезная матрица, так называемая единичная матрица, она нам встретится дальше. Она часто нужна, поэтому ее удобно уметь делать автоматически.

```
[19]: C = np.eye(3)
      C
```

← Создаём матрицу 3 x 3 с единицами на диагонали

```
[19]: array([[1., 0., 0.],
            [0., 1., 0.],
            [0., 0., 1.]])
```

Матричные операции

Сложение и вычитание матриц.

Сложение и вычитание матриц

Как и векторы, две матрицы одного и того же размера $m \times n$ можно складывать и вычитать поэлементно:

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix} \quad B = \begin{pmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,n} \end{pmatrix}$$

$$A + B = \begin{pmatrix} a_{1,1} + b_{1,1} & \cdots & a_{1,n} + b_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} + b_{m,1} & \cdots & a_{m,n} + b_{m,n} \end{pmatrix}$$

А также поэлементно умножать на скаляры, как и векторы.

Как и векторы, матрицы одного и того же размера, то есть с одинаковым количеством строк и столбцов, можно складывать и вычитать поэлементно, то есть так же, как вы складываете в столбик и то, что будет в сумме на месте с индексом (1, 1) — это сумма $a_{1,1}$ и $b_{1,1}$ и так далее. Очень важно, чтобы размеры совпадали, потому что иначе ничего не получится, у нас не будет подходящих чисел для сложения на нужных местах.

И точно так же, как и векторы, матрицы можно поэлементно умножать на числа, на скаляры, то есть просто каждый элемент матрицы умножается на одно и то же число, как мы уже видели раньше. Ничего более сложного пока что не происходит.

Умножение матриц

В отличие от векторов, которые умножать между собой скорее бессмысленно, умножение матриц — это очень важная операция, которая встречается повсеместно. И хотя она выглядит очень сложно, нам будет очень важно разобраться, как это устроено. Итак, вот у нас есть две матрицы — одна размера $m \times n$, вторая размера $n \times k$. Тут очень важно, что вот эта n , которая называется «внутренняя размерность», одно и то же число, иначе умножить матрицу не получится. Итак у нас есть матрица A и матрица B , тогда их можно умножить что называется «строка на столбец». Как это работает. Чтобы получить

первое число в произведении матриц, то есть то, что мы запишем в ячейку с номером (1,1), мы берем первую строку первой матрицы и первый столбец второй матрицы. Эти векторы мы скалярно умножаем друг на друга и записываем в ячейку (1,1) в матрице $A \times B$ и так далее. Чтобы например понять что нам записать в ячейку с номером (m,1), мы берем m-ую строку из первой матрицы и первый столбец из второй матрицы. А чтобы понять, что будет в ячейке с номером (m,k), мы берем m-ую строку из первой матрицы и k-ый столбец из второй и скалярно умножаем.

Умножение матриц

Две матрицы, одна размера $m \times n$, другая $n \times k$, можно перемножить по правилу "строка на столбец":

$$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix} \quad B = \begin{pmatrix} b_{1,1} & \dots & b_{1,k} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \dots & b_{n,k} \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} (a_1, b_1) & \dots & (a_1, b_k) \\ \vdots & \ddots & \vdots \\ (a_m, b_1) & \dots & (a_m, b_k) \end{pmatrix}$$

Здесь (a_i, b_j) - скалярные произведения векторов
 $a_i = (a_{i,1}, \dots, a_{i,n})$
 $b_j = (b_{1,j}, \dots, b_{n,j})$

Кажется, что операция очень сложная, в действительности нужно сделать очень много сложений и умножений чисел, чтобы это руками посчитать. К счастью все умножения матрицы давно реализованы на Python, и это работает более-менее автоматически, поэтому нам важно было узнать как чисто математически эта операция устроена.

Умножения матриц обладают некоторыми хорошими свойствами, которые являются некоторым обобщением обычных операций с числами. Например как и с умножением обычных вещественным чисел, мы можем раскрывать скобки, когда умножаем A на $(B+C)$, мы можем расписать это как $A \times B + A \times C$. Тоже самое работает, когда у нас скобка слева, а не справа, то есть $(A+B) \times C$, точно так же можно скобки раскрыть. И тоже все очень здорово с умножением на скаляры, очень похоже на скалярное умножение векторов, то есть скаляр мы можем выносить за скобку. Все работает интуитивно, как бы мы от этой операции и хотели.

Но есть одно очень важное но, очень важный нюанс, что операция умножения матриц не перестановочная, то есть как правило $A \times B$ не то же самое, что $B \times A$, даже когда у матриц подходящая размерность. То есть например когда у нас A и B — квадратные матрицы одного и того же размера, тогда мы могли бы их попробовать переставить, но как правило результат операций будет отличаться. А если A и B — это какие-то произвольные матрицы, с неравными внешними размерностями, то их и переставит просто не получится, операция не будет иметь смысла, она не будет определена.

Свойства умножения матриц

Некоторые свойства умножения чисел всё ещё работают:

$$A \cdot (B + C) = A \cdot B + A \cdot C$$

С умножением на скаляры тоже всё хорошо (как у векторов).

! Но, как правило, $A \cdot B \neq B \cdot A$ (даже когда размерности подходящие!)

Пример. Умножение на Python.

Нам опять понадобится библиотека numpy. Создаем две матрицы, обе размером 2x2 и умножаем их с помощью такой операции, которая записывается символом «@». В numpy этот символ используется для матричного умножения. Умножаем A на B и видим результат. Тоже получается квадратная матрица. А потом умножаем B на A и видим, что результат тоже квадратная матрица, но числа уже совсем другие, не равные первому результату.

Умножение матриц на python

```
[29]: A = np.array([[1, 2], [4, 5]])
A
[29]: array([[1, 2],
           [4, 5]])

[30]: B = np.array([[ -2,  3], [ 0,  1]])
B
[30]: array([[ -2,  3],
           [  0,  1]])

[31]: A @ B
[31]: array([[ -2,  5],
           [-8, 17]])

[32]: B @ A
[32]: array([[10, 11],
           [ 4,  5]])
```

Операция умножения

$AB \neq BA$

Второй способ записывать умножение матриц — с помощью функции `dot`, которую мы уже видели, когда обсуждали скалярное произведение векторов. На самом деле умножение матриц можно рассматривать как некоторое обобщение скалярного произведения векторов, а именно когда у нас есть один вектор размера m и второй вектор размера n , то мы можем один вектор расположить как вектор-строку, а второй как вектор-столбец и тогда, если мы их умножим по правилу умножения матриц, то это будет ровно то же самое, что и скалярное умножение этих векторов. Поэтому и неудивительно, что та же самая функция используется и для умножения матриц. Мы видим, что результат точно такой же. Можно пользоваться и тем, и тем. Иногда удобнее так, иногда так.

Умножение матриц на python

```
[29]: A = np.array([[1, 2], [4, 5]])
      A
[29]: array([[1, 2],
            [4, 5]])

[30]: B = np.array([-2, 3], [0, 1])
      B
[30]: array([-2,  3],
            [ 0,  1])

[31]: A @ B
[31]: array([-2,  5],
            [-8, 17])

[32]: B @ A
[32]: array([[10, 11],
            [ 4,  5]])
```

Два варианта записи

```
[34]: np.dot(B, A)
[34]: array([[10, 11],
            [ 4,  5]])
```

Перемножение матриц и векторов

Пример. Перемножение матриц

$$A \cdot B = \begin{pmatrix} 1 & 2 & 2 \\ 3 & 1 & 1 \end{pmatrix} \begin{pmatrix} 4 & 2 \\ 3 & 1 \\ 1 & 5 \end{pmatrix} = \begin{pmatrix} 1 \cdot 4 + 2 \cdot 3 + 2 \cdot 1 & 1 \cdot 2 + 2 \cdot 1 + 2 \cdot 5 \\ 3 \cdot 4 + 1 \cdot 3 + 1 \cdot 1 & 3 \cdot 2 + 1 \cdot 1 + 1 \cdot 5 \end{pmatrix} = \begin{pmatrix} 12 & 14 \\ 16 & 12 \end{pmatrix}$$

$$B \cdot A = \begin{pmatrix} 4 & 2 \\ 3 & 1 \\ 1 & 5 \end{pmatrix} \begin{pmatrix} 1 & 2 & 2 \\ 3 & 1 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} 4 \cdot 1 + 2 \cdot 3 & 4 \cdot 2 + 2 \cdot 1 & 4 \cdot 2 + 2 \cdot 1 \\ 3 \cdot 1 + 1 \cdot 3 & 3 \cdot 2 + 1 \cdot 1 & 3 \cdot 2 + 1 \cdot 1 \\ 1 \cdot 1 + 5 \cdot 3 & 1 \cdot 2 + 5 \cdot 1 & 1 \cdot 2 + 5 \cdot 1 \end{pmatrix} = \begin{pmatrix} 10 & 10 & 10 \\ 6 & 7 & 7 \\ 16 & 7 & 7 \end{pmatrix}$$

Даны 2 матрицы: A и B. Умножение матрицы A на B можно выполнить, если количество столбцов матрицы A равно количеству строк матрицы B

Когда мы хотим умножить матрицу на вектор, а это такая операция, которая тоже очень часто используется в самых разных приложениях. На самом деле можно сказать, что одно из основных использований матриц — это компактно записывать операции с векторами в виде умножения матриц на векторы. Мы можем вектор-строку длины n рассматривать как матрицу размера $1 \times n$. А вектор-столбец длины n можно рассматривать как матрицу размера $n \times 1$. И можем умножать их на большую матрицу по тому же самому правилу умножения матриц.

Перемножение матриц и векторов

Вектор-строку длины n можно рассматривать как матрицу размера $1 \times n$.
 Вектор-столбец длины n можно рассматривать как матрицу размера $n \times 1$.
 Умножаем по правилу умножения матриц:

$$(u_1, \dots, u_m) \cdot \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix} = (v_1, \dots, v_n)$$

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix}$$

Это значит, что если у нас есть вектор-строка, то мы можем его умножить на прямоугольную матрицу подходящего размера слева, то есть когда m (длина нашего вектора) равна m в высоте нашей матрицы, то можно сделать операцию умножения и на выходе мы получаем тоже строку, но уже другой длины n , где n — это количество столбцов в матрице, на которую мы умножили наш вектор. И умножается здесь все по такому же правилу умножения матриц, а именно: мы берем наш вектор u и умножаем сначала на вектор-столбец, который первый столбец нашей матрицы, скалярно умножаем и это будет первое число в результате (v_1). И так далее.

Когда у нас есть вектор-столбец, мы можем его умножить на матрицу справа. Ровно так же, и на выходе мы получаем вектор-столбец уже другой длины, где m — длина результата — это в точности количество строк в нашей матрице.

Специальные матрицы

Разберем одну очень важную матрицу, а именно единичную матрицу. На самом деле их много, по одной на каждую размерность. Но как правило говорят просто единичная матрица, как бы подразумевая, что из контекста будет ясно какого она должна быть размера. Задается она вот так.

Единичная матрица

Специальные матрицы

Единичная матрица I_n размера $n \times n$, где n любое:

$$I_n = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

Обладает свойством: для любой матрицы A размера $n \times k$

$$A \cdot I_k = A, \quad I_n \cdot A = A$$

Это матрица $n \times n$, такая квадратная матрица, на ее диагонали стоят единицы, а все, кроме диагонали занято нулями. Небольшое отступление — когда мы говорим о диагонали матрицы, то имеем в виду именно те числа, которые идут из левого верхнего угла в правый нижний. Числа, которые расположены по другой диагонали, они обычно диагональю не называются или называются не главной диагональю. То есть когда мы говорим о диагонали матрицы, то имеется в виду именно диагональ с левого верхнего угла в правый нижний.

Такая матрица обладает следующим уникальным свойством. Будучи умножена на любую другую матрицу, мы получаем ровно ту же матрицу, на которую умножили, то есть $A \cdot I_k = A$, для любой матрицы размера $n \times k$. И $I_n \cdot A = A$. Это в точности обобщение того, как работает обычная единица в вещественных числах. Мы знаем, что если мы единицу умножаем на любое другое число, получаем то самое число. Точно так же работает единичная матрица. То есть в каком-то смысле умножение на единичную матрицу — это операция ничего не делать.

Линейные отображения

Линейное отображение — такое отображение пространства векторов, то есть как бы функция из пространства векторов, со следующими свойствами, которые собственно называются свойствами линейности. А именно — это когда мы можем, как мы видели уже раньше на других примерах, как бы раскрыть скобки относительно сложения и умножения на числа, то есть мы берем нашу функцию F , наше отображение F от вектора $\alpha_1 u + \alpha_2 v$. Свойство линейности означает, что это в точности то же самое, что число α_1 , умноженное на функцию F , примененную к u + число α_2 , умноженное на функцию F , примененную к вектору v .

$$F(\alpha_1 u + \alpha_2 v) = \alpha_1 F(u) + \alpha_2 F(v)$$

Это верно для любых векторов u и v и любых чисел α_1 и α_2 .

Оказывается, что любое линейное отображение — это на самом деле умножение на некоторую матрицу A . То есть как только у нас есть отображение пространства векторов, которое обладает свойством линейности, это автоматически означает, что оно на самом деле есть умножение на некоторую матрицу.

Например, такое отображение, тождественное отображение, которое к каждому вектору v ставит в соответствии ровно его же самого.

$$F(v) = A \cdot v$$

Легко заметить, что оно удовлетворяет свойству линейности, потому что $\alpha_1 u + \alpha_2 v$ — это должны быть в точности тот же самый вектор.

Легко заметить, что поскольку $F(u) = u$, а $F(v) = v$, то левая часть равна правой, то есть $F(\alpha_1 u + \alpha_2 v) = \alpha_1 F(u) + \alpha_2 F(v)$.

Тождественное отображение — это линейное отображение, и оно задается умножением на единичную матрицу, которую мы уже видели раньше.

Единичная матрица, умноженная на любой вектор, дает нам ровно тот же самый вектор.

$$F(v) = I \cdot v = v$$

Когда векторов много.

Часто мы на практике хотим применить наше отображение не к одному вектору, а сразу к большому количеству. Ну например у нас есть какая-то задача машинного обучения и таблица с точками данных, где каждая отдельная точка — это на самом деле вектор в пространстве признаков. И мы делаем некоторое линейное преобразование с нашими векторами признаков, например понижаем размерность или еще как-то их преобразуем, скажем масштабируем. Тогда мы хотим обычно сделать одно и то же преобразование сразу со всеми векторами. На python это можно было бы реализовать циклом, но можно лучше и компактнее.

1. Поместим все векторы в одну большую матрицу $n \times K$, K — количество векторов:

$$V = \begin{pmatrix} v_{1,1} & \cdots & v_{1,K} \\ \vdots & \ddots & \vdots \\ v_{n,1} & \cdots & v_{n,K} \end{pmatrix}$$

$v_1 \qquad v_K$

2. Умножим матрицу отображения A на матрицу V :

$$R = AV$$

3. Вуаля! Столбцы полученной матрицы R есть образы наших векторов!

Мы сначала поместим все векторы в одну большую матрицу размера $n \times K$, где K — это количество векторов. Получается такая матрица, где по столбцам записаны наши исходные векторы, а K — это просто их количество. Это большая матрица V . И всю эту матрицу мы одновременно умножаем на матрицу того самого отображения, которое мы хотим применить, на A . А умножить на V , получается некоторая матрица R . Столбцы полученной матрицы R — это в точности есть образы всех наших векторов. Их будет ровно K штук, и каждый столбик новой матрицы — это исходные вектора, к которым мы применили отображение A , то есть умножили на матрицу A . Такой очень удобный способ, он еще вычислительно гораздо более быстрый, потому что как правило умножение матриц в разных численных библиотеках, как например в numpy, сильно оптимизировано и иногда даже задействует графические ускорители на вашем компьютере, то есть это будет очень быстрая операция, а цикл будет скорее всего гораздо медленнее.

Композиция линейных отображений

Когда у нас есть два линейных отображения, мы можем сначала применить одно, а потом второе. Такая штука называется композицией двух отображений. Говоря формально, композиция двух отображений F и G — это новое линейное отображение, можно доказать, что композиция двух линейных отображений всегда тоже линейное отображение. И оно получается как последовательное применение сначала от одного, то есть мы сначала применяем G к нашему вектору, а потом применяем F и все это обозначается как $F \cdot G$:

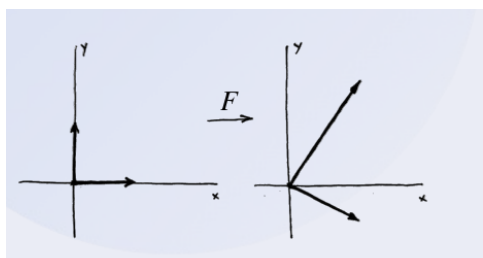
$$(F \cdot G)(v) = F(G(v))$$

Порядок имеет значение: композиция двух отображений F и G и композиция отображения G и F — вообще говоря, разные отображения, даже если это преобразование одного и того же пространства. И уже можно догадаться, что матрица композиции отображений есть произведение матриц самих отображений, то есть когда мы вычисляем произведение матриц, это то же самое, что мы находим композицию отображений, заданных нашими матрицами.

Разберем примеры какие вообще бывают линейные отображения. Линейные отображения — это на самом деле какие-то преобразования пространства.

Линейные отображения. Примеры

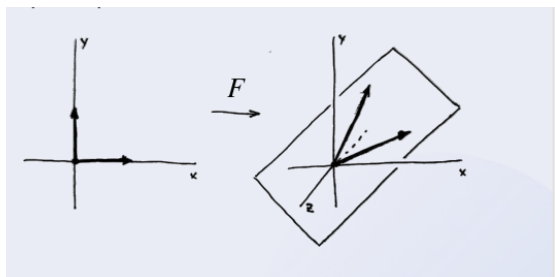
Например когда у нас есть квадратная матрица $n \times n$, то она задает нам преобразование n -мерного пространства. Это может выглядеть вот так, что у нас было двумерное пространство, мы применяем к нему некоторое отображение F , которое горизонтальный вектор отправляет в один вектор, а вертикальный в другой, все остальные векторы как бы поворачиваются и растягиваются, как будто они были наклеены на эту плоскость, и растяжение задается нам в точности как то, что случилось с базисными векторами.



Оказывается на самом деле, чтобы задать линейное отображение достаточно задать образы векторов базиса, и это полностью определяет судьбу всех остальных векторов при этом отображении.

Когда у нас матрица размера $n \times k$, где $k > n$, это можно представлять себе как отображение в пространстве большей размерности, когда мы нашу исходную плоскость как бы вкладываем в трехмерное, например, пространство, как показано на рисунке ниже.

Пример. Вектор (1,0) стал вектором в трехмерном пространстве. Второй базисный вектор стал тоже другим вектором в трехмерном пространстве. Все остальные векторы в нашей плоскости отобразились на некоторую плоскость, которая как бы натянута на образы базисных векторов в трехмерном пространстве.

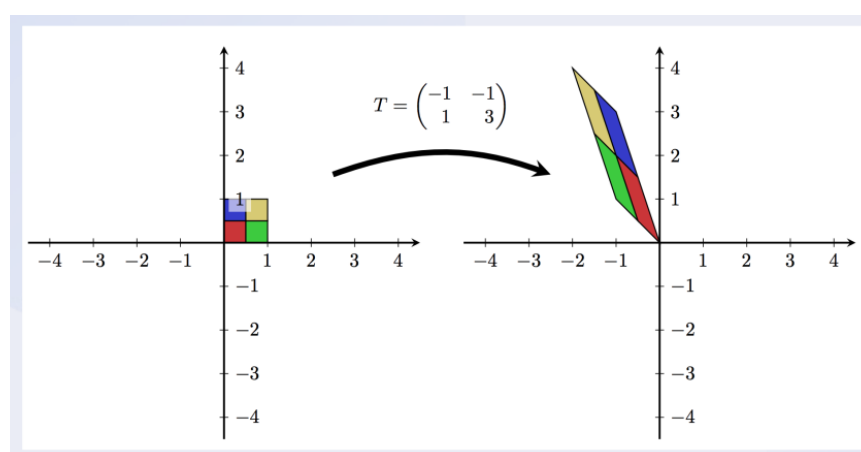


И наоборот, когда у нас прямоугольная матрица размера $n \times k$, но $k < n$, это как будто мы отображаем пространство большей размерности в пространство меньшей размерности. Мы как бы схлопываем некоторые координаты и получаем пространство размерности меньшей, чем была изначально.

Посмотрим еще на некоторые примеры. Уже преобразование одного и того же двумерного пространства. Например дана вот такая матрица

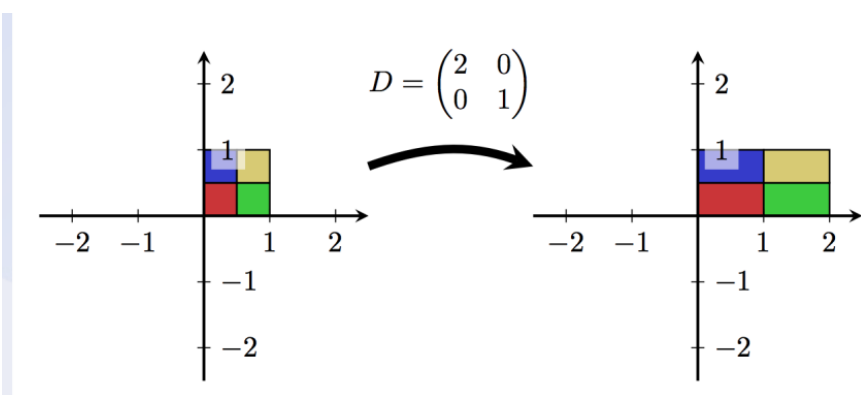
$$T = \begin{pmatrix} -1 & -1 \\ 1 & 3 \end{pmatrix}$$

Чтобы понять как она работает, как действует отображение, заданное этой матрицей, или как действует умножение на эту матрицу, что то же самое, как мы уже знаем, мы можем посмотреть что будет с разноцветным квадратиком, когда мы подействуем матрицей. Мы можем вычислить, что 0 остается на месте, как всегда, когда мы действуем линейными отображениями, а например желтый угол отправляется в левый верхний угол. Из картинке видно как все остальные точки отобразились. Можно представить что будет с остальными точками этой плоскости, если мы продолжим действия по линейности, то есть нам надо получившееся отображение сильно растягивать в нужном нам направлении и сжимая в перпендикулярном направлении.



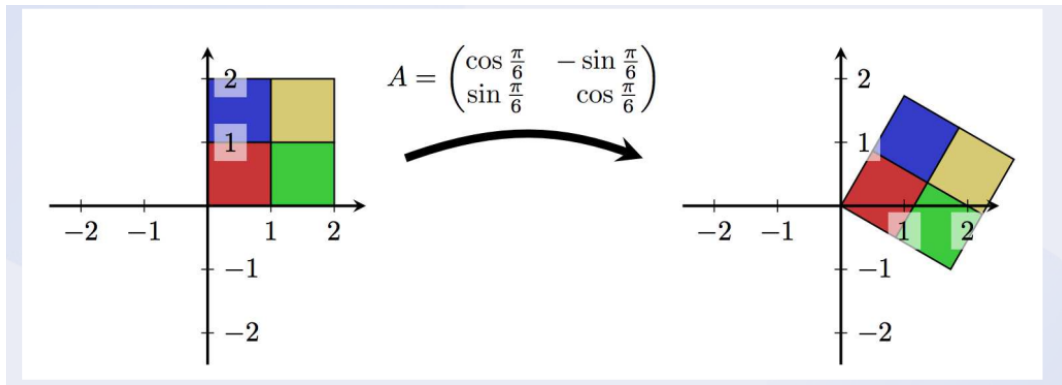
Другой пример попроще. Матрица:

Это диагональная матрица. Когда у нас есть диагональная матрица, то умножение на такую матрицу действует так, что мы как бы по отдельности отображаем, по отдельности растягиваем разные векторы базиса. В данном случае умножение на такую матрицу растягивает в два раза первый вектор базиса, то есть вектор (1,0) или направляющий вектор по оси x. Он растянется в два раза, а поскольку на втором месте по диагонали стоит 1, значит, что вторая координата у умножится на единицу, то есть с ней ничего не произойдет. Это ровно то, что мы видим на рисунке ниже.



Разноцветный квадратик становится более длинным по оси x, а по оси y с ним ничего не случилось. Такая матрица растяжения по одной оси.

Наконец матрица поворота может выглядеть вот так:



Тут у нас записан поворотный $\frac{\pi}{6}$ угол
То есть на 30 градусов.

И видим, что такая матрица из косинусов и синусов на самом деле это просто числа какие-то тоже. И наш исходный квадратик повернулся на 30 градусов как мы видим на рисунке выше. Все остальные точки на этой плоскости тоже как бы поворачиваются вслед за квадратиком.

Обратная матрица

Матрица, обратная к данной, — такая матрица, что при их перемножении получается единичная матрица. Это очень похоже на обратные вещественные числа, когда у нас скажем было изначально число 2, то обратное число это $\frac{1}{2}$. К 3 обратное число — это $\frac{1}{3}$. Когда мы умножаем 2 на $\frac{1}{2}$, то мы получаем 1. Ровно так же когда мы умножаем матрицу на обратную матрицу, то получаем единичную матрицу.

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

Обратная матрица к данной — это матрица при перемножении которой с текущей матрицей получается единичная матрица.

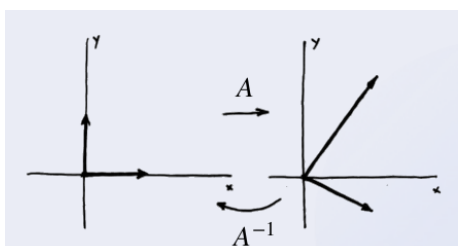
$$AA^{-1} = I$$

Например:

$$B = \begin{bmatrix} 2 & 5 & 7 \\ 6 & 3 & 4 \\ 5 & -2 & -3 \end{bmatrix} \quad B^{-1} = \begin{bmatrix} 1 & -1 & 1 \\ -38 & 41 & -34 \\ 27 & -29 & 24 \end{bmatrix}$$

$$B \cdot B^{-1} = \begin{bmatrix} 2 & 5 & 7 \\ 6 & 3 & 4 \\ 5 & -2 & -3 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & 1 \\ -38 & 41 & -34 \\ 27 & -29 & 24 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$$

Но в отличие от вещественных чисел, обратные матрицы существуют не всегда. Их еще надо поискать. Их вычисление не такая простая операция. Они существуют только для квадратных матриц. Если подумать какой размерности должна быть обратная матрица, то чтобы можно было ее переставить с исходной матрицей, мы догадаемся что когда у нас матрица A $n \times k$, и n не равно k , то не может быть такой матрицы, что можно и так, и так ее переставлять. Но даже и для квадратных матриц обратная матрица не всегда определена. Однако если она существует, то она в точности единственная. То есть уж если мы ее нашли, то она ровно одна. И ее геометрическая интерпретация - это то, что обратная матрица задает нам обратное преобразование пространства. То есть если матрица A действовала вот так как на рисунке ниже, то есть она делала из вот такого базиса растянутый и повернутый, то обратная матрица нам все вернет обратно как было до преобразования.



Как вычислить обратную матрицу на питоне. К счастью в библиотеке numpy есть встроенная функция, которая нам обратную матрицу вычисляет. Мы возьмем квадратную матрицу 2x2 вот такую:

```
[1]: import numpy as np
[2]: A = np.array([[1, 2], [4, 5]])
      A
[2]: array([[1, 2],
            [4, 5]])
```


и вызовем вот такую функцию:

```
[3]: B = np.linalg.inv(A)
B
[3]: array([[ -1.66666667,  0.66666667],
          [ 1.33333333, -0.33333333]])
```

inv от английского inverse, а linalg — это такая подбиблиотека для работы с линейной алгеброй, там довольно много полезных операций. Мы получаем матрицу B, обратную к A. Проверим это утверждение. A умножаем на B, получаем единичную матрицу, умножаем B на A, снова та же единичная матрица.

С другой стороны возьмем вот такую матрицу C и попробуем тоже вычислить ее обратную матрицу. Но у нас возникает ошибка, numpy говорит нам, что эта матрица вырождена, по английски singular. Как раз у таких матриц обратных нет.

```
[6]: C = np.array([[1, 2], [2, 4]])
C
[6]: array([[1, 2],
          [2, 4]])
[7]: D = np.linalg.inv(C)
[4]: A @ B
[4]: array([[1., 0.],
          [0., 1.]])
[5]: B @ A
[5]: array([[1., 0.],
          [0., 1.]])
```

Системы линейных уравнений

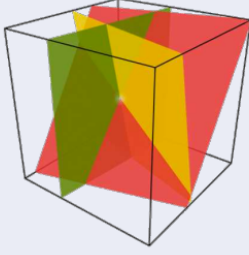
Мы их видели уже в школе, может быть какие-то маленькие системы, не из такого количества уравнений как здесь.

Системы линейных уравнений

$$\begin{cases} a_{1,1}x_1 + \dots + a_{1,n}x_n = y_1 \\ \dots \\ a_{m,1}x_1 + \dots + a_{m,n}x_n = y_m \end{cases}$$

Возникают повсеместно, в том числе в

- приближении функций методом наименьших квадратов
- задачах оптимизации
- линейной и логистической регрессии
- интерполяции пропущенных значений



Линейные уравнения — это когда наши неизвестные переменные x_1 и так далее x_n входят в само уравнение просто с какими-то числовыми коэффициентами без каких-то степеней, символов, синусов, косинусов, как бы без всего вот этого, просто каждое отдельное уравнение — это сумма неизвестных переменных x с какими-то числами, просто определенными числами. И в правой части y — это тоже какое-то число. Система линейных уравнений понятно — это просто система вот таких вот уравнений. И мы должны найти x_1 и так далее x_n , которые одновременно удовлетворяют всем нашим уравнениям.

Геометрически это устроено так, что каждое отдельно уравнение — оно на самом деле задает нам некоторую гиперплоскость в n -мерном пространстве, то есть множество решений одного уравнения — это может быть к примеру вот такая зеленая гиперплоскость как на рисунке выше. А решение второго уравнения — это красная гиперплоскость, то есть все точки из этой плоскости удовлетворяют какому-то одному уравнению, но когда у нас есть система уравнений, это значит, что мы ищем точки, которые принадлежат одновременно всем этим гиперплоскостям. То есть геометрически мы смотрим на пересечение всех этих гиперплоскостей. Когда мы пересекаем две плоскости в трехмерном пространстве, то мы получаем на их пересечении какую-то прямую, то есть пространство меньшей размерности.

Ровно то же самое работает и в n -мерном пространстве, то есть когда мы пересекаем две гиперплоскости, то мы получаем в пересечении пространство на единичку меньше, потом пересекаем еще с одной, получаем еще на единичку меньше и так далее.

В конце, когда наша система линейных уравнений достаточно хорошая и имеет единственное решение, то может остаться ровно одна точка, которая лежит на пересечении всего вот этого. Но если у нас уравнений меньше, чем n неизвестных, то соответственно решение будет целая какая-то прямая или плоскость, то есть вот это наше пересечение меньшего числа геометрических плоскостей.

Для чего нужны линейные уравнения даже сложно перечислить, потому что возникают они просто повсеместно. Когда мы приближаем функции методом наименьших квадратов и решаем какие-то сопутствующие задачи оптимизации, то есть находим такие значения x -ов, которые минимизируют некоторый функционал, находим его наименьшую точку. Или когда в

классическом машинном обучении мы обучаем линейную или логистическую регрессию. Там мы тоже ищем коэффициенты нашей регрессии как на самом деле решение определенной системы линейных уравнений. Нужно сначала составить нужную систему уравнений и, решив ее, мы находим наши коэффициенты. Или когда мы решаем задачу интерполяции, когда у нас есть данные, но для некоторых точек данных мы не знаем каких-то значений, и мы хотим их заполнить осмысленно, так чтобы если у нас есть одно измерение, второе измерение, и мы хотим сгладить дырку между ними. Это называется «интерполяция». Там тоже возникают системы линейных уравнений, когда мы это делаем. Это далеко не полный список, просто для понимания интуиции для чего это может быть нужно.

Системы линейных уравнений

Системы линейных уравнений можно записать в виде матриц, а именно, мы возьмем и из коэффициентов уравнения, то есть из чисел a_1, a_{m1}, \dots, a_n , в-общем из всех этих a образуем следующую матрицу A , как на рисунке ниже.

$$\begin{cases} a_{1,1}x_1 + \dots + a_{1,n}x_n = y_1 \\ \dots \\ a_{m,1}x_1 + \dots + a_{m,n}x_n = y_m \end{cases}$$

Матричная форма записи:

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \quad \text{или просто} \quad Ax = y$$

$A \qquad \qquad x \qquad \qquad y$

Просто запишем их в строки и столбцы. И наши x -ы мы тоже будем представлять себе как будто это один неизвестный вектор x , а x_1, \dots, x_n — это просто его компоненты. И тогда левая часть нашего уравнения — это умножение вектор x_1, \dots, x_n на матрицу A . А правая часть — это тоже вектор-столбец с компонентами y_1, \dots, y_m , то есть все это мы компактно переписываем в виде $Ax = y$.

Казалось бы мы не сделали ничего такого, просто переписали в другом виде, но стала уже заметно короче запись и на самом деле заметно проще и работать с этим тоже, потому что если матрица A оказалась обратима, то система линейных уравнений очень легко решить. Мы можем найти ее обратную матрицу и умножить обе части нашего уравнения слева на обратную матрицу.

Если матрица A обратима, то умножим обе части слева на A^{-1} :

$$Ax = y \implies A^{-1}Ax = A^{-1}y \iff x = A^{-1}y$$

Тогда у нас слева будет такое выражение $A^{-1}Ax$, а $A^{-1}y$ справа

A^{-1} умножить на A дают нам единичную матрицу, а единичная матрица, умноженная на x , это в точности x , то есть A^{-1} и A они как бы сократились, и у нас слева остался просто неизвестный x , а справа A^{-1} умноженный на y . И все, мы уже как бы систему уравнений решили. Мы решение получили в явном виде, как $A^{-1}y$

Возникает немедленно вопрос. А когда же матрица A обратима? Она ведь обратима не всегда. И тут уже вопрос дошел до того, чтобы понять когда же все-таки? Для начала посмотрим как бы косвенно, что на самом деле обратная матрица существует всегда, когда информации необходимо и достаточно, чтобы нашу систему уравнений однозначно решить. То есть как только у системы уравнений на самом деле есть единственное решение, то матрица обязательно обратима и наоборот. Это эквивалентные условия.

Систему уравнений можно однозначно решить, когда во-первых неизвестных ровно столько же, сколько и уравнений, то есть $m=n$, это значит, что матрица левой части A квадратная. Второе условие — то, что наши уравнения нельзя сократить, то есть нельзя их число уменьшить, складывая их и умножая на числа, то есть это значит, что каждое уравнение на самом деле нам важно и как бы его нельзя отбросить или геометрически это означает, что никакая гиперплоскость не совпадает с какой-то комбинацией других. Или на самом деле это то же самое, что строки матрицы линейно независимы. Вспоминаем линейную независимость векторов, которая у нас была на первой лекции. Это в точности эквивалентные условия. Если строки матрицы линейно независимы, то выполнено второе условие — число уравнений сократить нельзя.

Определитель матрицы

Как понять, когда строки матрицы линейно независимы или есть какая-то зависимость?

Мы какую-то зависимость можем угадать, когда мы видим как одну строку можно выразить через линейную комбинацию других. Вообще говоря как будто похоже, что вообще каждый раз нужно отдельно думать как же их там складывать,

чтобы в сумме получился ноль. Но есть очень прямолинейный алгоритм, как можно вычислительно понять верно ли, что строки матрицы линейно независимы?

Для этого нам понадобится так называемый **определитель или детерминант матрицы**, которая обозначается как $\det A$ (слово детерминант) и строки квадратной матрицы линейно независимы тогда и только тогда, когда ее определитель не равен нулю.

$\det A \neq 0$.

Обратная матрица должна быть квадратной и невырожденной (определитель не равен нулю).

Детерминант (определитель) матрицы $A = (\alpha_{ij})$ -

$$\Delta A = \sum_{\alpha_1, \alpha_2, \dots, \alpha_n=1}^{n \times n} (-1)^{N(\alpha_1, \alpha_2, \dots, \alpha_n)} \cdot \alpha_{1i_1} \alpha_{2i_2} \dots \alpha_{ni_n},$$

$n \times n$ размер матрицы
 $\alpha_1, \alpha_2, \dots, \alpha_n$ все перестановки матрицы
 $N(\alpha_1, \alpha_2, \dots, \alpha_n)$ число инверсий в перестановке

<https://ru.wikipedia.org/wiki/Определитель>

Размерность 1, то есть матрица — это просто число. Тогда ее определитель — это и есть это число. Ничего необычного.

$$\det(a_{1,1}) = a_{1,1}$$

Когда у нас размерность становится равной двум, то есть у нас есть квадратная матрица 2x2, то ее определитель записывается вот так:

$$\det \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} = a_{1,1}a_{2,2} - a_{2,1}a_{1,2}$$

Когда у нас есть матрица размера 3x3, то ее определитель — это уже более хитрое выражение:

$$\det \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{2,1}a_{1,3}a_{3,2} - a_{3,1}a_{2,2}a_{1,3} - a_{1,2}a_{2,1}a_{3,3} - a_{1,1}a_{2,3}a_{3,2}$$

Где у нас шесть слагаемых, в каждом по три сомножителя. И для большей размерности еще более сложная формула. Как это можно попробовать запомнить.

Для размерности два правило такое, что мы берем сначала числа на главной диагонали, умножаем их и вычитаем числа, которые на второстепенной диагонали.

$$\det \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} = \underline{a_{1,1}a_{2,2}} - \underline{a_{2,1}a_{1,2}}$$

А для трех это выглядит вот так:

$$\det \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = \underline{a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{2,1}a_{1,3}a_{3,2}} - \underline{a_{3,1}a_{2,2}a_{1,3} - a_{1,2}a_{2,1}a_{3,3} - a_{1,1}a_{2,3}a_{3,2}}$$

Мы сначала берем числа на главной диагонали и числа, который стоят в двух треугольниках со сторонами как бы

параллельными главной диагонали. То есть сторона $a_{1,2} a_{2,3}$, она как бы можно считать, что параллельна диагонали и другой синий треугольник симметричен ему относительно диагонали. Все это отрезки и треугольники берутся со знаком +, то есть мы берем $a_{1,1} a_{2,2} a_{3,3}$, то есть это первое наше слагаемое и так далее. А то, что относится к второстепенной диагонали мы берем со знаком -. То есть мы умножаем $a_{3,1}$ на $a_{2,2}$ на $a_{1,3}$ со знаком минус подставляем в формулу и так далее.

$$\det \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = \underline{a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{2,1}a_{1,3}a_{3,2}} - \underline{a_{3,1}a_{2,2}a_{1,3} - a_{1,2}a_{2,1}a_{3,3} - a_{1,1}a_{2,3}a_{3,2}}$$

Получается шесть суммарно слагаемых. И вот так можно запомнить как вычислять определители 2×2 и 3×3 .
Иногда определитель матрицы еще обозначается как модуль матрицы, в вертикальных палочках. $|A|$ Но тут важно помнить, что определитель бывает и отрицательным, то есть это не такой модуль, как мы привыкли видеть у чисел.