Lecture 1
Linear Algebra

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

什么是线性代数
What is Linear Algebra?

▶ 线性代数是数学的一个分支，其使用矩阵处理线性方程及其在向量空间中的表示。换句话说，线性代数是对线性函数和向量的研究。线性代数是数学中最核心的话题之一，大多数现代几何概念都是基于线性代数的。

Linear algebra is a branch of mathematics that deals with linear equations and their representations in the vector space using matrices. In other words, linear algebra is the study of linear functions and vectors. It is one of the most central topics of mathematics. Most modern geometrical concepts are based on linear algebra.

▶ 线性代数有助于许多自然现象的建模，因此是工程和物理学的一个有机组成部分。线性方程、矩阵和向量空间是该主题最重要的组成部分。

Linear algebra facilitates the modeling of many natural phenomena and hence, is an integral part of engineering and physics. Linear equations, matrices, and vector spaces are the most important components of this subject.

标量
Scalar

▶ 标量是单个数字。例如，我们可以说"让$s \in \mathbb{R}$表示直线的斜率"，这样我们定义了一个取值于实数域的标量s；或者我们说"让$n \in \mathbb{N}$表示给定单位的数量"，这样我们定义了取值于自然数的一个标量n。

A scalar is just a single number. For example, we might say "Let $s \in \mathbb{R}$ be the slope of the line", while we define a real-valued scalar. Or we say "Let $n \in \mathbb{N}$ be the number of units," while defining a natural number scalar.

▶ 在TensorFlow中，定义标量时要首先明确类型，如自然数还是浮点数。同时要指明位宽，如32位还是64位。下面的代码定义了一些标量：

In TensorFlow, you need to specify the type when you define the scalar, for example, it is of an integer type or floating type. You also need to specify the number of bits or bit width as required by the range or accuracy, for example, 32 bits or 64 bits. The following code snippet defines some scalars:

import tensorflow as tf

a = tf.constant(1, dtype=tf.int32)

b = tf.constant(2.0, dtype=tf.float64)

向量
Vectors

- 向量是由标量组成的数组，我们默认其中的标量按顺序排列，可以通过索引按顺序识别每个单独的分量。通常，向量用粗体小写字母表示，例如$\boldsymbol{x}$。向量的元素用带有下标的斜体字母来标识，如x_1，x_2，依此类推。我们通常还需要说明向量的取值范围，如果每个元素是属于$\mathbb{R}$，并且向量中有n个元素，那么向量属于$\mathbb{R}$的n次笛卡尔积形成的集合，记作$\mathbb{R}^n$。当我们需要显式标识向量的元素时，我们将它们写成括在方括号中的列。

A vector is an array of scalars. The scalars are arranged in order. We can identify each individual number or component by its index in that ordering. Typically we give vectors lowercase names in bold typeface, such as $\boldsymbol{x}$. The elements of the vector are identified by writing its name in italic typeface, with a subscript, such as x_1, x_2, and so on. We also need to specify what kind of numbers are stored in the vector. If each element is in $\mathbb{R}$, and the vector has n elements, then the vector lies in the set formed by taking the Cartesian product of $\mathbb{R}$ n times, denoted as $\mathbb{R}^n$. When we need to explicitly identify the elements of a vector, we write them as a column enclosed in square brackets.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

向量
Vectors

- 在数学中，向量可以写成一行或一列，分别称为行向量或列向量。数学中不加说明的话，一般是指列向量。但在机器学习中，如果不加说明，则向量可能指行向量。向量中元素的个数称为向量的维度或维数，如$x = [x_1 \quad x_2 \quad x_3]^T$，则$x$为三维向量。由于数学抽象与工程实践的差异，维度这个词意义有所不同。在数学中，我们说x为三维向量。而在工程实践中，如Numpy中，x的大小，即所存储元素的个数，为3，但其维数仍为1。同时，由于工程中的实际存储必然涉及到如何遍历的问题，因此，我们也会谈x的形状，其与存储的方式相关。

In mathematics, vectors can be written in rows or in columns, so we have the row vectors or column vectors, respectively. In mathematics, if not specified, vectors generally means the column vectors. But in machine learning, if unspecified, a vector potentially indicate a row vector. The number of elements in a vector is called the dimension of the vector. For example, $x = [x_1 \quad x_2 \quad x_3]^T$, then x is a three-dimensional vector. Due to the difference between mathematical abstraction and engineering practice, the word dimension has different meanings. In mathematics, we simply say that x is a three-dimensional vector. In engineering, such as Numpy, the number of stored elements, we call the size of x, is 3, but its dimension is 1. At the same time, since the actual storage must answer the traverse of the array in order to store, we also talk about the shape of x, which relates the way of storage.

向量
Vectors

▶ 当向量默认写成列向量方式时，我们写成行向量时，要加转置，例如$x = [x_1 \quad x_2 \quad x_3]^T$。

If the vectors are column vectors by default, we need to apply transpose to the row vector in accordance with the convention. For example, $x = [x_1 \quad x_2 \quad x_3]^T$。

▶ 下面代码展示了TensorFlow对创建向量的支持：

The following code demonstrates the support of TensorFlow for creating vectors:

import numpy as np

import tensorflow as tf

a = tf.constant([1, 2, 3], dtype=tf.float32)

b = tf.constant(np.arange(1, 10, 2), dtype=tf.int32)

print(f"a => dimension: {tf.rank(a)}, #elements: {tf.size(a)}, shape: {tf.shape(a)}")

a => dimension: 1, #elements: 3, shape: [3]

print(f"b => dimension: {tf.rank(b)}, #elements: {tf.size(b)}, shape: {tf.shape(b)}")

b => dimension: 1, #elements: 5, shape: [5]

向量
Vectors

▶ 有时我们需要访问向量的一组元素。在这种情况下，可以定义一个包含索引的集合，并将该集合作为下标。例如，要访问x_1，x_3 与x_6，我们定义集合$S = \{1, 3, 6\}$，记为x_S。

Sometimes we need to index a set of elements of a vector. In this case, we define a set containing the indices and write the set as a subscript. For example, to access x_1, x_3 and x_6, we define the set $S = \{1, 3, 6\}$ and write x_S.

▶ 下面代码展示了Tensorflow对上述方式的支持：

The following code demonstrates the support of TensorFlow for accessing a set of components:

```
import numpy as np

import tensorflow as tf

indices = tf.constant([0, 2, 6], dtype=tf.int32)

data = tf.constant(np.arange(1, 10), dtype=tf.float32)

subset = tf.gather(data, indices)

print(subset) # tf.Tensor([1. 3. 7.], shape=(3,), dtype=float32)
```

矩阵
Matrices

▶ 矩阵是二维数字组成的数组，每个元素都由行与列两个索引标识。我们通常用大写字母加粗，比如$\boldsymbol{A}$，表示矩阵。如果一个实值矩阵的高为m，宽为n，即m行n列，则记$\boldsymbol{A} \in \mathbb{R}^{m \times n}$。我们通常使用不加粗体的斜体字体来标识矩阵的元素，并且索引用逗号分隔。例如，$A_{1,1}$是$\boldsymbol{A}$的左上角元素，$A_{m,n}$是右下角元素。我们用$A_{i,:}$表示$\boldsymbol{A}$的第i行的所有元素，同样，用$A_{:,j}$表示$\boldsymbol{A}$的第j列的所有元素。当我们需要显式标识矩阵的元素时，将它们写成一个括在方括号中的数组。

$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{bmatrix}$$
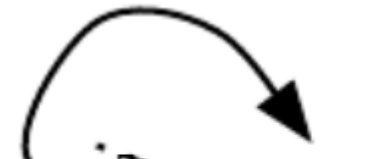
A matrix is a 2-D array of numbers, so each element is identified by two indices, namely, the row index and the column index. We usually give matrices upper-case variable names with bold typeface, such as $\boldsymbol{A}$. If a real-valued matrix $\boldsymbol{A}$ has a height of m and a width of n, then we say that $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. We usually identify the elements of a matrix using its name in italic but not bold font, and the indices are listed with separating commas. For example, $A_{1,1}$ is the upper left entry of $\boldsymbol{A}$ and $A_{m,n}$ is the bottom right entry. We can identify all the numbers with vertical coordinate i by writing a ":" for the horizontal coordinate, like $A_{i,:}$. This is known as the i-th row of $\boldsymbol{A}$. Likewise, $A_{:,j}$ is the j-th column of $\boldsymbol{A}$. When we need to explicitly identify the elements of a matrix, we write them as an array enclosed in square brackets.

矩阵
Matrices

- 矩阵上的一个常见的操作是转置。矩阵的转置是矩阵在从其左上角开始向下和向右延伸的对角线，称为主对角线上的镜像。有关此操作的图形描述，请参见下图。我们将矩阵$\boldsymbol{A}$的转置表示为$\boldsymbol{A}^T$，其定义为$(A^T)_{i,j} = A_{j,i}$。注意，由定义不难理解标量的转置是其本身，向量的转置是行列互换。

One common operation on matrices is the transpose. The transpose of a matrix is the mirror image of the matrix across the diagonal line, called the main diagonal, running down and to the right, starting from its upper left corner. See the following figure for a graphical depiction of this operation. We denote the transpose of a matrix $\boldsymbol{A}$ as $\boldsymbol{A}^T$, and it is defined such that $(A^T)_{i,j} = A_{j,i}$. From the definition we know that a scalar is its own transpose and the transpose of a vector is to switch from row to column representation or vice versa.

$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \boldsymbol{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

矩阵
Matrices

▶ 下面代码展示了TensorFlow对矩阵及其转置的支持：

The following code demonstrates the support of TensorFlow for creating matrices and perform the transpose:

A = tf.constant([[1, 2, 3], [4, 5, 6]], dtype=tf.float32)

print(f"A => dimension: {tf.rank(A)}, #elements: {tf.size(A)}, shape: {tf.shape(A)}")

A => dimension: 2, #elements: 6, shape: [2 3]

A_t = tf.transpose(A)

<tf.Tensor: shape=(3, 2), dtype=float32, numpy=

array([[1., 4.],   [2., 5.],   [3., 6.]], dtype=float32)

print(f"A_t => dimension: {tf.rank(A_t)}, #elements: {tf.size(A_t)}, shape: {tf.shape(A_t)}")

A_t => dimension: 2, #elements: 6, shape: [3 2]

s = tf.constant(1, dtype=tf.float32)

assert(s != tf.transpose(s)), "Oops!" # AssertionError: Oops!

矩阵
Matrices

▸ 我们可以将具有相同的形状的矩阵相加，规则是只需将它们相应位置的元素相加，如 $C = A + B$，其中 $C_{i,j} = A_{i,j} + B_{i,j}$。我们还可以将标量与矩阵相加或将标量与矩阵相乘，只需对矩阵的每个元素执行该操作，如 $D = a \cdot B + c$，这里 $D_{i,j} = a \cdot C_{i,j} + b$。

We can add matrices to each other, as long as they have the same shape, just by adding their corresponding elements: $C = A + B$ where $C_{i,j} = A_{i,j} + B_{i,j}$. We can also add a scalar to a matrix or multiply a matrix by a scalar, just by performing that operation on each element of a matrix: $D = a \cdot B + c$, where $D_{i,j} = a \cdot C_{i,j} + b$.

▸ 在深度学习的上下文中，我们也使用一些不太传统的符号。我们允许将一个矩阵和一个向量相加，产生另一个矩阵：$C = A + b$，其中 $C_{i,j} = A_{i,j} + b_j$。换句话说，将向量与矩阵的每一行相加。这种速记消除了在执行加法之前定义相容矩阵 B 并复制 b 到每一行的需要。这种对多个位置的隐式复制称为广播。

In the context of deep learning, we also use some less conventional notation. We allow the addition of a matrix and a vector, yielding another matrix: $C = A + b$, where $C_{i,j} = A_{i,j} + b_j$. In other words, the vector b is added to each row of the matrix. This shorthand eliminates the need to define a compatible matrix B with b copied into each row before doing the addition. This implicit copying of b to many locations is called broadcasting.

矩阵
Matrices

- 下面代码展示了TensorFlow对点加、点乘及广播的支持：

The following code demonstrates the support of TensorFlow for dot addition, dot multiplication and broadcasting:

A = tf.constant([[1, 2, 3], [4, 5, 6]], dtype=tf.float32)

b = tf.constant(2, dtype=tf.float32)

c = tf.constant(1, dtype=tf.float32)

D = b * A + c

<tf.Tensor: shape=(2, 3), dtype=float32, numpy=

array([[3., 5., 7.],   [9., 11., 13.]], dtype=float32)>

e = tf.ones((1, 3))

F = D + e

<tf.Tensor: shape=(2, 3), dtype=float32, numpy=

array([[4., 6., 8.],   [10., 12., 14.]], dtype=float32)>

矩阵
Matrices

▸ 矩阵的最重要的操作之一是矩阵的乘法。矩阵A与B的乘积是第三个矩阵C。为了定义此乘积，A必须具有与B的行相同的列数。如果A为$m \times n$维的，或A的形状为$m \times n$，B为$n \times p$维的，那么C的形状为$m \times p$。我们只需将两个或多个矩阵放在一起就可以表示矩阵的乘积，例如$C = AB$，乘积运算由$C_{i,j} = \sum_k A_{i,k} B_{k,j}$定义。

One of the most important operations involving matrices is multiplication of two matrices. The matrix product of matrices A and B is a third matrix C. In order for this product to be defined, A must have the same number of columns as B has rows. If A is of shape $m \times n$ and B is of shape $n \times p$, then C is of shape $m \times p$. We can write the matrix product just by placing two or more matrices together, for example, $C = AB$. The product operation is defined by $C_{i,j} = \sum_k A_{i,k} B_{k,j}$.

▸ 请注意，两个矩阵的标准乘积不是指包含各个元素乘积的矩阵。存在这样的运算，称为元素乘积或哈达玛积，记作$A \odot B$。

Note that the standard product of two matrices is not just a matrix containing the product of the individual elements. Such an operation exists and is called the element-wise product, or Hadamard product, and is denoted as $A \odot B$.

矩阵
Matrices

- 下面代码展示了TensorFlow对矩阵乘法及哈达玛积的支持：

The following code demonstrates the support of TensorFlow for matrix multiplication, and Hadamard multiplication:

A = tf.constant([[1, 2, 3], [4, 5, 6]], dtype=tf.float32)

B = tf.constant ([[1, 2], [3, 4], [5, 6]], dtype=tf.float32)

C = tf.linalg.matmul(A, B)

<tf.Tensor: shape=(2, 2), dtype=float32, numpy=

array([[22., 28.],   [49., 64.]], dtype=float32)>

D = tf.ones_like(A) * 2.0

E = A * D

<tf.Tensor: shape=(2, 3), dtype=float32, numpy=

array([[2., 4., 6.],   [8., 10., 12.]], dtype=float32)>

矩阵
Matrices

- 矩阵乘积运算具有许多有用的性质，使矩阵在数学上的分析更加方便。例如，矩阵乘法是满足分配率的：$A(B + C) = AB + AC$。它同时也满足结合律：$A(BC) = (AB)C$。但矩阵乘法不是可交换的，即条件 $AB = BA$ 并不总是成立，这与标量乘法不同。

Matrix product operations have many useful properties that make mathematical analysis of matrices more convenient. For example, matrix multiplication is distributive: $A(B + C) = AB + AC$. It is also associative: $A(BC) = (AB)C$. Matrix multiplication is not commutative (the condition $AB = BA$ does not always hold), unlike scalar multiplication.

- 相同维数的两个向量 x 与 y 之间的点积是矩阵乘积 $x^T y$。由于其是一个标量，两个向量之间的点积是满足交换律的。

The dot product between two vectors x and y of the same dimensionality is the matrix product $x^T y$. Since it is a scalar, the dot product between two vectors is commutative.

- 两个矩阵的乘积的转置具有如下形式：

The transpose of a matrix product has the following form:
$$(AB)^T = B^T A^T$$

矩阵
Matrices

- 我们可以认为矩阵乘积 $C = AB$ 中 $C_{i,j}$ 的计算是 A 的第 i 行与 B 的第 j 列 之间的点积。当 B 退化为一个列向量时，便引出下面线性系统的概念。

We can think of the matrix product $C = AB$ as computing $C_{i,j}$ as the dot product between row i of A and column j of B. When B is reduced to a column vector, it generates to the concept of linear system.

- 我们根据学到记法可将一般的线性方程组表示为 $Ax = b$ 的形式。其中 $A \in \mathbb{R}^{m \times n}$ 是系数矩阵， $b \in \mathbb{R}^m$ 是已知的列向量， $x \in \mathbb{R}^n$ 是我们想要求解的未知变量向量。x 的每个元素 x_i 都是这些未知变量之一。A 的每一行和 b 的每个元素都提供对 x 的一个约束。我们可以重写等式 $Ax = b$ 如右所示：

$$A_{1,:}x = b_1$$
$$A_{2,:}x = b_2$$
$$\vdots$$
$$A_{m,:}x = b_m$$

We now know enough linear algebra notation to write down a system of linear equations as $Ax = b$. Where $A \in \mathbb{R}^{m \times n}$ is the coefficient matrix, $b \in \mathbb{R}^m$ is the known column vector, and $x \in \mathbb{R}^n$ is a vector of unknown variables we would like to solve for. Each element x_i of x is one of these unknown variables. Each row of A and each element of b provide a constraint. We can rewrite equation $Ax = b$ as on the right:

矩阵
Matrices

▶ 实际上，我们对$Ax = b$的如下的最原始形式是颇为熟悉的，但矩阵向量积表示法为线性方程组提供了更紧凑的表示。

Actually, we now know primitive form of $Ax = b$ below pretty well, however, matrix-vector product notation provides a more compact representation for equations of this form.

$$A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n = b_1$$
$$A_{2,1}x_1 + A_{2,2}x_2 + \cdots + A_{2,n}x_n = b_2$$
$$\vdots$$
$$A_{m,1}x_1 + A_{m,2}x_2 + \cdots + A_{m,n}x_n = b_m$$

▶ 为求解方程组或线性系统$Ax = b$，我们需要引入矩阵的逆的概念。为此，我们首先需要定义单位矩阵的概念。单位矩阵是这样一个矩阵，当我们用该矩阵乘以任何向量时，该矩阵不会改变此向量，即对$I_n \in \mathbb{R}^{n \times n}$和$\forall x \in \mathbb{R}^n$，$I_n x = x$。

to analytically solve the linear equations $Ax = b$, we need a tool that linear algebra offers called matrix inversion. To describe matrix inversion, we first need to define the concept of an identity matrix. An identity matrix is a matrix that does not change any vector when we multiply that vector by that matrix. We denote the identity matrix that preserves n-dimensional vectors as I_n. Formally, $I_n \in \mathbb{R}^{n \times n}$, and $\forall x \in \mathbb{R}^n$, $I_n x = x$.

矩阵
Matrices

- 单位矩阵的结构很简单：沿主对角线的所有元素都是1，而所有其他元素都是零。矩阵 $A \in \mathbb{R}^{n \times n}$ 的逆矩阵表示为 A^{-1}，满足 $A^{-1}A = AA^{-1} = I_n$。

The structure of the identity matrix is simple: all the entries along the main diagonal are 1, while all the other entries are zero. The matrix inverse of $A \in \mathbb{R}^{n \times n}$ is denoted as A^{-1}, and it is defined as the matrix such that $A^{-1}A = AA^{-1} = I_n$.

- 我们可以通过如下步骤求解 $Ax = b$：

We can now solve equation $Ax = b$ using the following steps:
$$Ax = b \Rightarrow A^{-1} Ax = A^{-1}b \Rightarrow I_n x = A^{-1}b$$
$$x = I_n x \Rightarrow x = A^{-1}b$$

- 当然，上述过程取决于能否找到 A^{-1}，其存在与否需要一定的条件。当 A^{-1} 存在时，有多种算法可以以解析或封闭形式求出。

Of course, this process depends on it being possible to find A^{-1}, which exists or not depends on certain conditions. When A^{-1} exists, several different algorithms can find it in closed form.

矩阵
Matrices

- 下面代码展示了TensorFlow对矩阵求逆的支持：

$$1 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_3 = 5$$
$$2 \cdot x_1 + 0 \cdot x_2 - 1 \cdot x_3 = 4$$
$$0 \cdot x_1 + 3 \cdot x_2 + 1 \cdot x_3 = 2$$

A = tf.constant([[1, 1, 1], [2, 0, -1], [0, 3, 1]], dtype=tf.float32)

b = tf.constant ([5, 4, 2], dtype=tf.float32); A_inv = tf.linalg.inv(A)

<tf.Tensor: shape=(3, 3), dtype=float32, numpy=

array([[0.42857143, 0.28571427, -0.14285715],

[-0.2857143 , 0.14285715, 0.42857143],

[0.85714287, -0.42857143, -0.2857143]], dtype=float32)>

x = tf.linalg.matmul(A_inv, tf.expand_dims(b, axis=-1))

<tf.Tensor: shape=(3, 1), dtype=float32, numpy=

array([[2.9999998],   [0.   ],   [1.9999998]], dtype=float32)>

x_1 = tf.linalg.solve(A, tf.expand_dims(b, axis=-1))

assert(np.any((x - x_1) < np.finfo(np.float32).eps)), "Oops"

矩阵
Matrices

- 注意，在上面过程中，A^{-1} 主要用作理论工具，大多数软件不会应用其编制程序进行实际求解操作，这是由于 A^{-1} 在数字计算机上只能以有限的精度表示。实际上，利用 b 值的算法通常可以获得更准确的 x 的值。

Note in the above process, A^{-1} is primarily useful as a theoretical tool, in practice most software applications will not implement it to obtain the solution. This is because A^{-1} can be represented with only limited precision on a digital computer. Actually, algorithms that make use of the value of b can usually obtain more accurate estimates of x.

- 我们进一步指出，若矩阵 A 的逆存在，则必唯一。假设 M 与 N 均为 A 的逆，则有：

We point out further that if the inverse of A exists, it must be unique. Otherwise, we have:
$$M = IM = (NA)M = N(AM) = NI = N \Rightarrow M = N$$

- 上面结论指出，若 A^{-1} 存在，则 $Ax = b$ 必有唯一解。但我们知道，线性方程组可能无解（超定）或有无穷解（欠定），这说明矩阵 A 并不一定总是存在。

The above conclusion points out that if A^{-1} exists, then $Ax=b$ must have a unique solution. But we know that linear equations may have no solution (overdetermined) or infinite solutions (underdetermined), which shows that the matrix A does not always exist.

矩阵
Matrices

▶ 要分析方程有多少个解，可以将A的列作为方向向量，考虑从原点依次按指定的方向行进时，有多少种方法可以到达b。在这个观点中，x的每个元素x_i指定我们应该在每个方向上指定在A第i列$A_{:,i}$的方向上移动多远：$b = \sum_i x_i A_{:,i}$。一般来说，这种形式或运算称为线性组合。形式上，线性组合是将一组向量集$\{v^{(1)}, v^{(2)}, \cdots, v^{(n)}\}$的每个向量$v^{(i)}$通过乘以相应的标量系数再将结果相加得出：$\sum_i c_i v^{(i)}$。

To analyze how many solutions the equation has, think of the columns of A as specifying different directions we can travel in from the origin, then determine how many ways there are of reaching b. In this view, each element x_i of x specifies how far we should travel in the direction of i-th column $A_{:,i}$ of A: $b = \sum_i x_i A_{:,i}$. In general, this kind of operation is called a linear combination. Formally, a linear combination of some set of vectors $\{v^{(1)}, v^{(2)}, \cdots, v^{(n)}\}$ is given by multiplying each vector $v^{(i)}$ by a corresponding scalar coefficient and adding the results: $\sum_i c_i v^{(i)}$.

▶ 一组向量的张成的空间是由向量的所有可能的线性组合组成的点的集合。

The span of a set of vectors is the set of all points obtainable by linear combination of the original vectors.

矩阵
Matrices

▶ 确定 $Ax = b$ 是否有解决等价于测试 A 的列的张成的空间中是否存在 b。这个特定的空间称为 A 的列空间。因此，为了使 $Ax = b$ 对任意的 $b \in \mathbb{R}^m$ 有解，我们要求 A 的列空间为 $\mathbb{R}^m$。否则存在属于 A 的列空间但不属于 $\mathbb{R}^m$ 的点，该点可作为不存在解的 b 的值。

Determining whether $Ax = b$ has a solution thus amounts to testing whether b is in the span of the columns of A. This particular span is known as the column space, or the range, of A. In order for the system $Ax = b$ to have a solution for all values of $b \in \mathbb{R}^m$, we therefore require that the column space of A be all of $\mathbb{R}^m$. If any point in $\mathbb{R}^m$ is excluded from the column space, that point is a potential value of b that has no solution.

▶ 要求 A 的列空间张成 $\mathbb{R}^m$ 意味着 A 必须至少具有 n 列，且 $n \geq m$。否则，列空间的维数将小于 m。注意，$n \geq m$ 只是每个点都有解决方案的必要条件，这不是一个充分条件，因为某些列可能是冗余的。

The requirement that the column space of A be all of $\mathbb{R}^m$ implies immediately that A must have at least m columns, that is, $n \geq m$. Otherwise, the dimensionality of the column space would be less than m. Notably, having $n \geq m$ is only a necessary condition for every point to have a solution. It is not a sufficient condition, because it is possible for some of the columns to be redundant.

矩阵
Matrices

- 如果一组向量中有一个向量是其他向量的线性组合，则一组向量是线性相关的，否则称这组向量是线性无关的。如果我们向集合中添加一个向量，该向量是集合中其他向量的线性组合，则新向量不会对集合的张成的空间有贡献。这意味着要使矩阵的列空间包含$\mathbb{R}^m$，矩阵必须包含m个线性独立的列。此条件对于方程$\boldsymbol{Ax} = \boldsymbol{b}$的任一$\boldsymbol{b}$值都有一个解既是充分的又是必要的。请注意，要求是集合具有完全线性独立的列，而不是至少m，是因为没有一组m维向量可以有超过m个相互线性独立的列，但是一个有超过m个列的矩阵可能有不止一个这样的集合。

A set of vectors is linearly dependent if there exists a vector in the set which is a linear combination of the other vectors. Otherwise, it is linearly independent. If we add a vector to a set that is a linear combination of the other vectors in the set, the new vector does not add any points to the set's span. This means that for the column space of the matrix to encompass all of $\mathbb{R}^m$, the matrix must contain at least one set of m linearly independent columns. This condition is both necessary and sufficient for equation $\boldsymbol{Ax} = \boldsymbol{b}$ to have a solution for every value of $\boldsymbol{b}$. Note that the requirement is for a set to have exactly m linearly independent columns, not at least m, since no set of m-dimensional vectors can have more than m mutually linearly independent columns, but a matrix with more than m columns may have more than one such set.

矩阵
Matrices

- 我们知道当A存在逆矩阵时，方程$Ax = b$对于每个b值的解唯一。因此，这种情况下我们确定矩阵最多有m列，否则会有多个解。总而言之，这意味着矩阵必须是正方形的，也就是说，我们要求$m = n$并且所有列都是线性独立的。具有线性相关列的方阵称为奇异矩阵。如果A不是方阵或者是方阵但是奇异的，求解方程仍然是可能的，但我们不能使用矩阵求逆的方法来求解。

If the inverse of A exists, we know for every b there is an unique solution. So for the matrix to have an inverse, we need to make certain that the matrix has at most m columns. Otherwise there is more than one way of parametrizing each solution. Together, this means that the matrix must be square, that is, we require that $m = n$ and that all the columns be linearly independent. A square matrix with linearly dependent columns is known as singular. If A is not square or is square but singular, solving the equation is still possible, but we cannot use the method of matrix inversion to find the solution.

- 我们推到出了A可逆的条件。由于矩阵不满足交换律，因此存在矩阵M与N满足$MA = I$, $AN = I$，但$M \neq N$。此时不要求A为方阵，即行与列相同。

We deduced the condition that A has a inverse. Since the matrix does not meet the commutative law, there are matrices M and N satisfying $MA=I$, $AN=I$, but $M \neq N$. Now A is not required to be a square matrix, that is, the number of rows and columns are same.