Lecture 2
Linear Algebra (II)

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

范数
Norm

- 有时我们需要计算向量的大小，特别是在机器学习中。向量的大小称为范数，形式上，向量的p范数L^p由下式给出，其中$p \in \mathbb{R}$且$p \geq 1$：

  Sometimes we need to measure the size or magnitude of a vector, especially in machine learning. The magnitude of vectors are called norms. Formally, for $p \in \mathbb{R}$, $p \geq 1$, the L^p norm is given by:

$$\|\boldsymbol{x}\|_p = \left(\sum_i |x_i|^p\right)^{\frac{1}{p}}$$

- 范数是将向量映射到非负值的函数，直观的说，向量$\boldsymbol{x}$的范数衡量的是从原点到点$\boldsymbol{x}$的距离。更严格地说，范数是满足以下属性的任何函数f：

  Norms are functions mapping vectors to non-negative values. On an intuitive level, the norm of a vector $\boldsymbol{x}$ measures the distance from the origin to the point $\boldsymbol{x}$. More rigorously, a norm is any function f that satisfies the following properties:

  - $f(\boldsymbol{x}) = 0 \Rightarrow \boldsymbol{x} = 0$
  - $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$
  - $\forall \alpha \in \mathbb{R}, f(\alpha \boldsymbol{x}) = |\alpha| f(\boldsymbol{x})$

范数
Norm

- $p = 2$的L^2范数又被称为欧几里得范数，它是从原点到由$\boldsymbol{x}$标识的点的欧几里得距离。L^2范数在机器学习中的使用如此频繁，以至于下标2通常被省略，简单地表示为$\|\boldsymbol{x}\|$。使用L^2范数的平方计算向量的大小也很常见，因为它可以简单地表示为$\boldsymbol{x}^T\boldsymbol{x}$。

  The L^2 norm, with $p = 2$, is known as the Euclidean norm, which is simply the Euclidean distance from the origin to the point identified by $\boldsymbol{x}$. The L^2 norm is used so frequently in machine learning that it is often denoted simply as $\|\boldsymbol{x}\|$, with the subscript 2 omitted. It is also common to measure the size of a vector using the squared L^2 norm, which can be calculated simply as $\boldsymbol{x}^T\boldsymbol{x}$.

- 与L^2范数本身相比，平方L^2范数在数学和计算上更加便于使用。例如，平方L^2范数对$\boldsymbol{x}$的每个元素的导数仅依赖于$\boldsymbol{x}$的对应元素，而L^2范数的所有导数都依赖于整个向量。

  The squared L^2 norm is more convenient to work with mathematically and computationally than the L^2 norm itself. For example, each derivative of the squared L^2 norm with respect to each element of $\boldsymbol{x}$ depends only on the corresponding element of $\boldsymbol{x}$, while all the derivatives of the L^2 norm depend on the entire vector..

范数
Norm

- 在许多情况下，平方L^2范数可能是不太好用，因为它在原点附近增加得非常缓慢。在一些机器学习应用程序中，区分恰好为零的元素和小但非零的元素很重要。在这种情况下，我们引入另一个函数，该函数在所有位置都以相同的速率增长，但保留了数学上的简单性：L^1范数。L^1范数可以简化为：$\|\boldsymbol{x}\|_1 = \sum_i |x_i|$

  In many contexts, the squared L^2 norm may be undesirable because it increases very slowly near the origin. In several machine learning applications, it is important to discriminate between elements that are exactly zero and elements that are small but nonzero. In these cases, we turn to a function that grows at the same rate in all locations, but that retains mathematical simplicity: the L^1 norm. The L^1 norm may be simplified to $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$

- L^1范数常用于机器学习，特别是在零元素和非零元素之间的差异非常重要的场景。每当$\boldsymbol{x}$的一个元素从 0 移开ϵ时，L^1范数就会增加ϵ。

  The L^1 norm is commonly used in machine learning when the difference between zero and nonzero elements is very important. Every time an element of $\boldsymbol{x}$ moves away from 0 by ϵ, the L^1 norm increases by ϵ.

范数
Norm

- 我们有时会通过计算非零元素的个数来计算向量的大小。一些作者将此函数称为“L^0范数”，但这是不正确的，向量中非零元素的个数不是范数，因为按α缩放向量不会改变非零元素的个数。L^1范数通常用作非零元素个数的替代。

  We sometimes measure the size of the vector by counting its number of nonzero elements. Some authors refer to this function as the “L^0 norm”, but this is incorrect terminology. The number of nonzero entries in a vector is not a norm, because scaling the vector by α does not change the number of nonzero entries. The L^1 norm is often used as a substitute for the number of nonzero entries.

- 机器学习中另一种常见的范数是L^∞范数，也称为最大范数。该范数简化为向量中具有最大绝对值的元素，即$\|\boldsymbol{x}\|_\infty = \max_i |x_i|$。

  One other norm that commonly arises in machine learning is the L^∞ norm, also known as the max norm. This norm simplifies to the absolute value of the element with the largest magnitude in the vector, that is $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$.

范数
Norm

- 有时我们也可能希望计算矩阵的大小。在深度学习的背景下，最常见的方式是使用类似于向量的L^2范数的稍微晦涩一点儿的Frobenius范数：$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$

  Sometimes we may also wish to measure the size of a matrix. In the context of deep learning, the most common way to do this is with the otherwise obscure Frobenius norm which is analogous to the L^2 norm of a vector: $\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$

- 两个向量的点积可以用范数重写。具体来说，$\boldsymbol{x}^T\boldsymbol{y} = \|\boldsymbol{x}\|_2\|\boldsymbol{y}\|_2 \cos\theta$，其中$\theta$是$\boldsymbol{x}$和$\boldsymbol{y}$之间的夹角。但由于实际很难知道在高维空间中向量的夹角，因此读者知道其几何意义即可。

  The dot product of two vectors can be rewritten in terms of norms. Specifically, $\boldsymbol{x}^T\boldsymbol{y} = \|\boldsymbol{x}\|_2\|\boldsymbol{y}\|_2 \cos\theta$, where θ is the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$. But in practice it is hard to know the angle between two vectors especially in high dimensional space, the reader only needs to know its geometric meaning here.

矩阵
Matrices

- 下面代码展示了JAX对向量与矩阵范数的支持:

  The following code demonstrates the support of JAX for norms of vectors and matrices:

```
a = jax.numpy.array([1, 2, 3], dtype=jax.numpy.float32)
a_l1 = jax.numpy.linalg.norm(a, 1) # DeviceArray(6., dtype=float32)
a_l2 = jax.numpy.linalg.norm(a, 2) # DeviceArray(3.7416575, dtype=float32)
a_linf = jax.numpy.linalg.norm(a, jax.numpy.inf) # DeviceArray(3., dtype=float32)
a_l0 = jax.numpy.linalg.norm(a, 0) # DeviceArray(3., dtype=float32)
b = jax.numpy.array([1, 0, 0], dtype=jax.numpy.float32)
b_l0 = jax.numpy.linalg.norm(b, 0) # DeviceArray(1., dtype=float32)
A = jax.numpy.array([[1, 2, 3], [4, 5, 6]], dtype=jax.numpy.float32)
A_l2 = jax.numpy.linalg.norm(A, 2) # DeviceArray(9.508031, dtype=float32)
A_lfro = jax.numpy.linalg.norm(A, 'fro') # DeviceArray(9.539392, dtype=float32)
A_l2_r = jax.numpy.linalg.norm(A, 2, axis=0) # DeviceArray([4.1231055, 5.3851647, 6.708204 ], dtype=float32)
A_l2_c = jax.numpy.linalg.norm(A, 2, axis=1) # DeviceArray([3.7416575, 8.774964 ], dtype=float32)
```

特殊矩阵
Special Matrices

- 对角矩阵是沿主对角线元素非零、其它位置元素均为零的矩阵。形式上，一个矩阵$\boldsymbol{D}$是对角矩阵当且仅当$D_{i,j}=0$对所有$i\neq j$成立。对角元素都是1的单位矩阵是对角矩阵。我们用$\text{diag}(\boldsymbol{v})$来表示一个方对角矩阵，其对角元素由向量$\boldsymbol{v}$的元素给出。对角矩阵在计算上比较高效，要计算$\text{diag}(\boldsymbol{v})\boldsymbol{x}$，我们只需要将每个元素$x_i$缩放$v_i$倍，也就是说，$\text{diag}(\boldsymbol{v})\boldsymbol{x}=\boldsymbol{v}\odot\boldsymbol{x}$。求方对角矩阵的逆也很简单，但只有当每个对角线元素都非零时，逆矩阵才存在，此时，$\text{diag}(\boldsymbol{v})^{-1}=\text{diag}([1/v_1,1/v_2,\cdots,1/v_n]^T)$。一般地，我们可以根据任意矩阵推导出一些通用的机器学习算法，但通过将某些矩阵限制为对角矩阵时描述和推导都更为简单。

  Diagonal matrices consist of nonzero entries only along the main diagonal and zeros elsewhere. Formally, a matrix $\boldsymbol{D}$ is diagonal if and only if $D_{i,j}=0$ for all $i\neq j$. The identity matrix with all the diagonal entries 1 is a diagonal matrix. We write $\text{diag}(\boldsymbol{v})$ to denote a square diagonal matrix whose diagonal entries are given by the entries of the vector $\boldsymbol{v}$. Multiplying by a diagonal matrix is computationally efficient. To compute $\text{diag}(\boldsymbol{v})\boldsymbol{x}$, we only need to scale each element x_i by v_i. In other words, $\text{diag}(\boldsymbol{v})\boldsymbol{x}=\boldsymbol{v}\odot\boldsymbol{x}$. Inverting a square diagonal matrix is also efficient. The inverse exists only if every diagonal entry is non-zero, and in that case,$\text{diag}(\boldsymbol{v})^{-1}=\text{diag}([1/v_1,1/v_2,\cdots,1/v_n]^T)$. In general, we may derive some general machine learning algorithm in terms of arbitrary matrices but obtain a less expensive and less descriptive algorithm by restricting some matrices to be diagonal.

特殊矩阵
Special Matrices

- 并非所有对角矩阵都需要是方阵，但非方对角矩阵没有逆矩阵。对于非方对角矩阵$\boldsymbol{D}$，乘积$\boldsymbol{Dx}$将涉及对$\boldsymbol{x}$的每个元素进行缩放，如果$\boldsymbol{D}$的高度大于它的宽度，则将一些零连接到结果，或者如果$\boldsymbol{D}$的宽度大于它的高度，则丢弃向量的最后一些元素。

  Not all diagonal matrices need be square, however, non-square diagonal matrices do not have inverses. For a non-square diagonal matrix $\boldsymbol{D}$, the product $\boldsymbol{Dx}$ will involve scaling each element of $\boldsymbol{x}$ and either concatenating some zeros to the result, if $\boldsymbol{D}$ is taller than it is wide, or discarding some of the last elements of the vector, if $\boldsymbol{D}$ is wider than it is tall.

- 对称矩阵是转置等于其自身的矩阵，即$\boldsymbol{A} = \boldsymbol{A}^T$。当元素由不依赖于参数顺序的具有两个参数的某些函数生成时，通常会出现对称矩阵。例如，如果$\boldsymbol{A}$是距离测量矩阵，其中$A_{i,j}$给出点i到点j的距离，则$A_{i,j} = A_{j,j}$，因为距离函数是对称的。

  A symmetric matrix is any matrix that is equal to its own transpose, aka., $\boldsymbol{A} = \boldsymbol{A}^T$. Symmetric matrices often arise when the entries are generated by some function of two arguments that does not depend on the order of the arguments. For example, if $\boldsymbol{A}$ is a matrix of distance measurements, with $A_{i,j}$ giving the distance from point i to point j, then $A_{i,j} = A_{j,i}$, because distance functions are symmetric.

特殊矩阵
Special Matrices

- 单位向量是欧几里得范数为1或具有单位范数的向量，即$\|\boldsymbol{x}\|^2 = 1$。如果$\boldsymbol{x}^T\boldsymbol{y} = 0$，则称向量$\boldsymbol{x}$和向量$\boldsymbol{y}$正交。如果两个向量都具有非零范数，这意味着它们彼此成90度角。在$\mathbb{R}^n$中，至多n个向量可能是非零范数相互正交的。如果多个向量不仅正交而且欧几里得范数均为1，我们称它们是规范正交的或标准正交的。

  A unit vector is a vector with Euclidean norm equal to 1, or with unit norm, namely, $\|\boldsymbol{x}\|^1 = 1$. A vector $\boldsymbol{x}$ and a vector $\boldsymbol{y}$ are orthogonal to each other if $\boldsymbol{x}^T\boldsymbol{y} = 1$. If both vectors have nonzero norm, this means that they are at a 90 degree angle to each other. In $\mathbb{R}^n$, at most n vectors may be mutually orthogonal with nonzero norm. If the vectors not only are orthogonal but also have unit norm, we call them orthonormal.

- 正交矩阵是行相互正交且列相互正交的方阵，即$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I}$，这意味着$\boldsymbol{A}^{-1} = \boldsymbol{A}^T$，因此计算正交矩阵的逆矩阵计算成本非常低。请注意正交矩阵的定义，它们的行或列之间不仅是正交的，而且是标准正交的。行或列正交但非标准正交的矩阵没有特殊名称定义 。

  An orthogonal matrix is a square matrix whose rows are mutually orthonormal and whose columns are mutually orthonormal as well, that is $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I}$. This implies that $\boldsymbol{A}^{-1} = \boldsymbol{A}^T$, means their inverse are very cheap to compute. Pay careful attention to the definition of orthogonal matrices, their rows or columns are not merely orthogonal but fully orthonormal. There is no special term for a matrix whose rows or columns are orthogonal but not orthonormal.

本征分解
Eigendecomposition

- 许多数学对象可以通过将它们分解成基本的组成部分，或者找到它们的一些不依赖所选择的表示方式的一些普适性质，从而达到更好地理解。例如，整数可以分解为质因数。数字12的呈现方式会当我们选择基数是2以二进制形式表示时而改变，但12=2×2×3始终为真。从这个表示中我们可以得出有用的属性，例如，12不能被5整除，并且12的任何整数倍都可以被3整除。就像我们可以通过将整数分解为质因数来发现一些关于整数的本质一样，我们还可以分解矩阵，以呈现有关其性质的信息，这些信息在矩阵表示为元素的数组时并不明显。

  Many mathematical objects can be understood better by breaking them into constituent parts, or finding some properties of them that are universal, not caused by the way we choose to represent them. For example, integers can be decomposed into prime factors. The way we represent the number 12 will change depending on whether we write it in base tenor in binary, but it will always be true that 12 = 2×2×3. From this representation we can conclude useful properties, for example, that 12 is not divisible by 5, and that any integer multiple of 12 will be divisible by 3. Much as we can discover something about the true nature of an integer by decomposing it into prime factors, we can also decompose matrices in ways that show us information about their functional properties that is not obvious from the representation of the matrix as an array of elements.

本征分解
Eigendecomposition

- 一种广为使用的矩阵分解称为特征分解，即将矩阵分解为一组特征向量和特征值。

  One of the most widely used kinds of matrix decomposition is called eigen-decomposition, in which we decompose a matrix into a set of eigenvectors and eigenvalues.

- 方阵$\boldsymbol{A}$的特征向量是一个非零向量$\boldsymbol{v}$，乘以$\boldsymbol{A}$只会改变$\boldsymbol{v}$的尺度：$\boldsymbol{Av} = \lambda\boldsymbol{v}$。标量$\lambda$被称为对应于该特征向量的特征值。顺道说一下，也可以找到一个左特征向量使得$\boldsymbol{v}^T\boldsymbol{A} = \lambda\boldsymbol{v}^T$，但我们通常关心的是右特征向量。如果$\boldsymbol{v}$是$\boldsymbol{A}$的特征向量，那么对于$s \in \mathbb{R}$且$s \neq 0$，任何重新缩放的向量$s\boldsymbol{v}$也是特征向量。此外，$s\boldsymbol{v}$的特征值只需进行相反的缩放：$\lambda/s$。出于这个原因，我们通常只求解单位特征向量。

  An eigenvector of a square matrix A is a nonzero vector $\boldsymbol{v}$ such that multi-plication by $\boldsymbol{A}$ alters only the scale of $\boldsymbol{v}$: $\boldsymbol{Av} = \lambda\boldsymbol{v}$. The scalar λ is known as the eigenvalue corresponding to this eigenvector. BTW, One can also find a left eigenvector such that $\boldsymbol{v}^T\boldsymbol{A} = \lambda\boldsymbol{v}^T$, but we are usually concerned with right eigenvectors. If $\boldsymbol{v}$ is an eigenvector of $\boldsymbol{A}$, then so is any rescaled vectors $\boldsymbol{v}$ for $s \in \mathbb{R}$, $s \neq 0$. Moreover, $s\boldsymbol{v}$ has the reciprocal rescaled eigenvalue λ/s. For this reason, we usually look only for unit eigenvectors.

本征分解
Eigendecomposition

- 假设一个矩阵$\boldsymbol{A}$有n个线性独立的特征向量$\{\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \cdots, \boldsymbol{v}^{(n)}\}$，具有相应的特征值$\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$。我们可以将所有特征向量连接起来形成一个矩阵$\boldsymbol{V}$，$\boldsymbol{V} = [\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \cdots, \boldsymbol{v}^{(n)}]$。同样，我们可以连接特征值以形成向量$\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \cdots, \lambda_n]^T$。$\boldsymbol{A}$的特征分解为$\boldsymbol{A} = \boldsymbol{V}\,\mathrm{diag}(\boldsymbol{\lambda})\,\boldsymbol{V}^{-1}$。

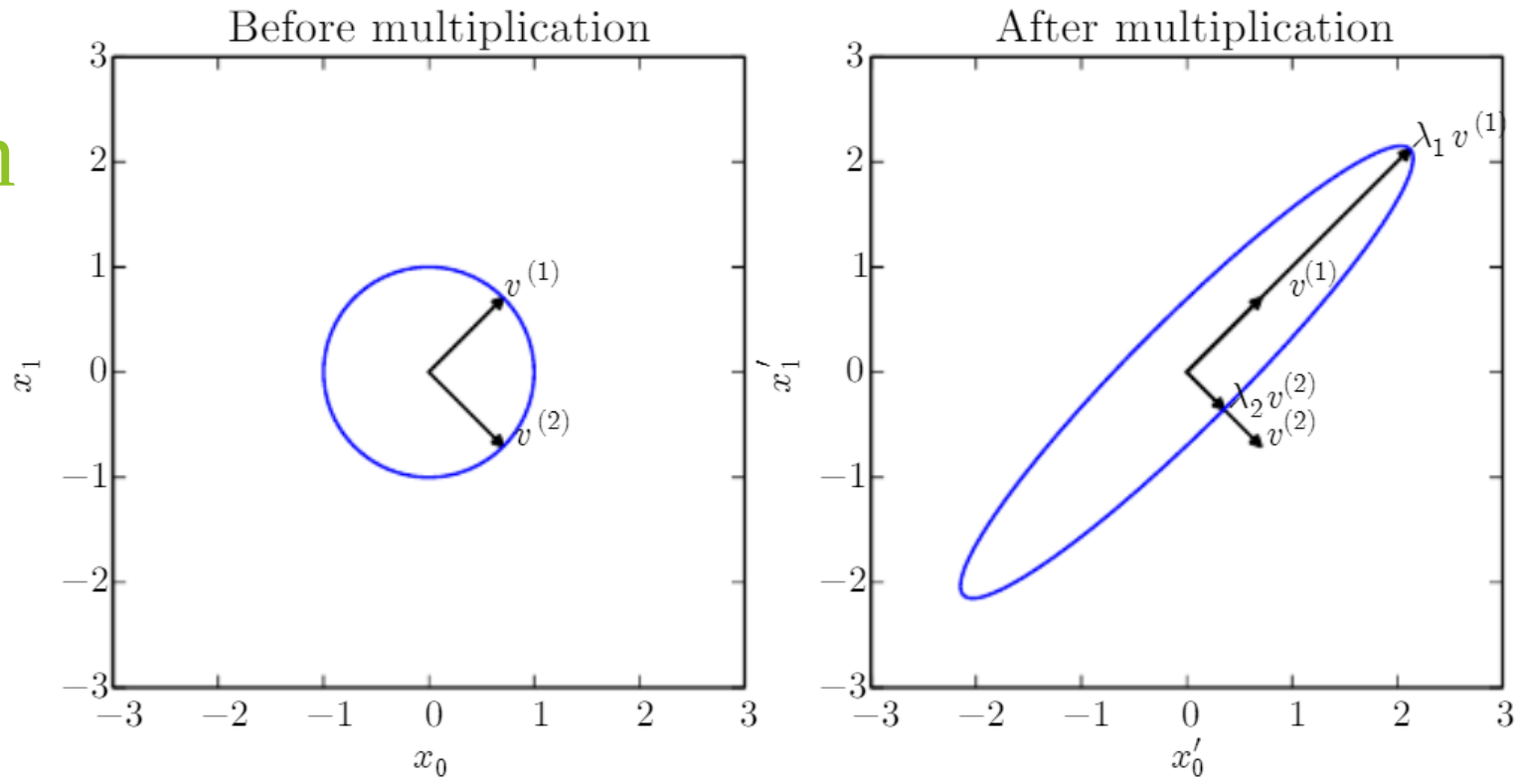
  Suppose that a matrix $\boldsymbol{A}$ has n linearly independent eigenvectors $\{\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \cdots, \boldsymbol{v}^{(n)}\}$ with corresponding eigenvalues $\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$. We may concatenate all the eigenvectors to form a matrix $\boldsymbol{V}$ with one eigenvector per column: $\boldsymbol{V} = [\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \cdots, \boldsymbol{v}^{(n)}]$. Likewise, we can concatenate the eigenvalues to form a vector $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \cdots, \lambda_n]^T$. The eigendecomposition of A is then given by $\boldsymbol{A} = \boldsymbol{V}\,\mathrm{diag}(\boldsymbol{\lambda})\,\boldsymbol{V}^{-1}$.

- 我们已经看到构造具有特定特征值和特征向量的矩阵使我们能够在期望的方向上拉伸空间。然而，我们经常想将矩阵分解为它们的特征值和特征向量。这样做可以帮助我们分析矩阵的某些属性，就像将一个整数分解成它的质因数可以帮助我们理解该整数的行为一样。

  We have seen that constructing matrices with specific eigenvalues and eigenvectors enables us to stretch space in desired directions. Yet we often want to decompose matrices into their eigenvalues and eigenvectors. Doing so can help us analyze certain properties of the matrix, much as decomposing an integer into its prime factors can help us understand the behavior of that integer.

本征分解 Eigendecomposition

- 一个关于特征向量和特征值的影响示例如右上。这里，我们有一个具有两个正交特征向量$\boldsymbol{v}^{(1)}$和$\boldsymbol{v}^{(2)}$的矩阵$\boldsymbol{A}$。$\boldsymbol{v}^{(1)}$对应于特征值λ_1，$\boldsymbol{v}^{(2)}$对应于特征值λ_2。（左）我们将所有单位向量$\boldsymbol{u} \in \mathbb{R}^2$的集合绘制为一个单位圆。（右）我们绘制了所有点$\boldsymbol{Au}$的集合。通过观察$\boldsymbol{A}$扭曲单位圆的方式，我们可以看到它在方向$\boldsymbol{v}^{(i)}$上按$\lambda_i$缩放空间。



- An example of the effect of eigenvectors and eigenvalues is as top-right. Here, we have a matrix $\boldsymbol{A}$ with two orthonormal eigenvectors, $\boldsymbol{v}^{(1)}$ with eigenvalue λ_1 and $\boldsymbol{v}^{(2)}$ with eigenvalue λ_2. (Left) We plot the set of all unit vectors $\boldsymbol{u} \in \mathbb{R}^2$ as a unit circle. (Right) We plot the set of all points $\boldsymbol{Au}$. By observing the way that $\boldsymbol{A}$ distorts the unit circle, we can see that it scales space in direction $\boldsymbol{v}^{(i)}$ by λ_i.

本征分解
Eigendecomposition

- 并非每个矩阵都可以分解为特征值和特征向量。在某些情况下，分解存在但涉及复数而不是实数。在本书中，我们只分解具有简单分解形式的实对称矩阵，它们都可以分解为仅使用实值特征向量和特征值的表达式：$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$，其中$\boldsymbol{Q}$是由$\boldsymbol{A}$的特征向量组成的正交矩阵，$\boldsymbol{\Lambda}$是对角矩阵。特征值$\Lambda_{i,i}$与$\boldsymbol{Q}$的第$i$列的特征向量相关联，记为$\boldsymbol{Q}_{:,i}$。因为$\boldsymbol{Q}$是一个正交矩阵，我们可以认为$\boldsymbol{A}$在方向$\boldsymbol{v}_i$上以$\lambda_i$倍缩放空间 。注意，虽然任何实对称矩阵都具有特征分解，但特征分解可能不是唯一的。如果任何两个或多个特征向量具有相同的特征值，则位于其张成的空间的任何一组正交向量，也是具有该特征值的特征向量。

  Not every matrix can be decomposed into eigenvalues and eigenvectors. In some cases, the decomposition exists but involves complex rather than real numbers. In this book, we usually just decompose real symmetric matrix which has a simple decomposition, aka., they can be decomposed into an expression using only real-valued eigenvectors and eigenvalues: $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$, where $\boldsymbol{Q}$ is an orthogonal matrix composed of eigenvectors of $\boldsymbol{A}$, and $\boldsymbol{\Lambda}$ is a diagonal matrix. The eigenvalue $\Lambda_{i,i}$ is associated with the eigenvector in column i of Q, denoted as $\boldsymbol{Q}_{:,i}$. Because $\boldsymbol{Q}$ is an orthogonal matrix, we can think of $\boldsymbol{A}$ as scaling space by λ_i in direction $\boldsymbol{v}_i$. Notably, While any real symmetric matrix is guaranteed to have an eigendecomposition, the eigendecomposition may not be unique. If any two or more eigenvectors share the same eigenvalue, then any set of orthogonal vectors lying in their span are also eigenvectors with that eigenvalue.

本征分解
Eigendecomposition

- 按照惯例，我们通常按降序对Λ的条目进行排序。此时，若所有的特征值都是唯一的，那么特征分解唯一。

  By convention, we usually sort the entries of $\boldsymbol{\Lambda}$ in descending order. Under this convention, the eigendecomposition is unique only if all the eigenvalues are unique.

- 矩阵的特征分解告诉我们关于矩阵的许多有用的事实。例如，当且仅当有特征值为零时，矩阵是奇异的。特征值均为正的矩阵称为正定矩阵，特征值全部为正值或零值的矩阵称为半正定矩阵。同样，如果所有特征值都是负数，则矩阵是负定的，如果所有特征值都是负数或零值，则矩阵是半负定的。半正定矩阵很有趣，因为它们保证$\forall \boldsymbol{x}$, $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{0}$。若此时$\boldsymbol{x}$为单位向量，令$f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$，则$f_{max} = \max_i\{\lambda_i\}$。

  The eigendecomposition of a matrix tells us many useful facts about the matrix. The matrix is singular if and only if any of the eigenvalues are zero. A matrix whose eigenvalues are all positive is called positive definite. A matrix whose eigenvalues are all positive or zero valued is called positive semi-definite. Likewise, if all eigenvalues are negative, the matrix is negative definite, and if all eigenvalues are negative or zero valued, it is negative semidefinite. Positive semidefinite matrices are interesting because they guarantee that $\forall \boldsymbol{x}$, $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{0}$. If $\boldsymbol{x}$ are unit norm vectors, let $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$, then $f_{max} = \max_i\{\lambda_i\}$.

奇异值分解
Singular Value Decomposition

- 奇异值分解(SVD)提供了另一种将矩阵因式分解为奇异向量和奇异值的方法。SVD使我们能够发现一些与特征分解所揭示的相同类型的信息。每个实矩阵未必有特征值分解，但都有奇异值分解，因此SVD更具有普遍的适用性。例如，如果矩阵不是方阵，则特征分解没有定义，我们必须使用奇异值分解来代替。

  The singular value decomposition (SVD) provides another way to factorize a matrix, into singular vectors and singular values. The SVD enables us to discover some of the same kind of information as the eigendecomposition reveals. Every real matrix might not have has an eigenvalue decomposition, but it always has a singular value decomposition, which means the SVD is more generally applicable. For example, if a matrix is not square, the eigendecomposition is not defined, and we must use a singular value decomposition instead.

- 回想一下，特征分解涉及分析矩阵$\boldsymbol{A}$以求出特征向量的矩阵$\boldsymbol{V}$和特征值向量$\boldsymbol{\lambda}$，这样我们就可以将$\boldsymbol{A}$重写为$\boldsymbol{A} = \boldsymbol{V}\,\mathrm{diag}(\boldsymbol{\lambda})\,\boldsymbol{V}^{-1}$。奇异值分解类似，只是这次我们将A写成三个矩阵的乘积：$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{T}$ 。

  Recall that the eigendecomposition involves analyzing a matrix $\boldsymbol{A}$ to discover a matrix $\boldsymbol{V}$ of eigenvectors and a vector $\boldsymbol{\lambda}$ of eigenvalues such that we can rewrite $\boldsymbol{A}$ as $\boldsymbol{A} = \boldsymbol{V}\,\mathrm{diag}(\boldsymbol{\lambda})\,\boldsymbol{V}^{-1}$. The singular value decomposition is similar, except this time we will write $\boldsymbol{A}$ as a product of three matrices: $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{T}$.

奇异值分解
Singular Value Decomposition

- 在矩阵$\boldsymbol{A}$的奇异值分解$\boldsymbol{A}=\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$的表示中，假设$\boldsymbol{A}$是一个$m\times n$矩阵，则$\boldsymbol{U}$被定义为一个$m\times m$矩阵，$\boldsymbol{D}$为一个$m\times n$矩阵，$\boldsymbol{V}$为一个$n\times n$矩阵。这些矩阵中的每一个都被定义为具有特殊结构的矩阵。矩阵$\boldsymbol{U}$和$\boldsymbol{V}$都被定义为正交矩阵，矩阵$\boldsymbol{D}$定义为对角矩阵，注意$\boldsymbol{D}$不一定是方阵。沿$\boldsymbol{D}$对角线的元素称为矩阵$\boldsymbol{A}$的奇异值。$\boldsymbol{U}$的列称为左奇异向量，$\boldsymbol{V}$的列被称为右奇异向量。我们实际上可以用关于$\boldsymbol{A}$的函数的特征分解来解释$\boldsymbol{A}$的奇异值分解。$\boldsymbol{A}$的左奇异向量是$\boldsymbol{A}\boldsymbol{A}^T$的特征向量。$\boldsymbol{A}$的右奇异向量是$\boldsymbol{A}^T\boldsymbol{A}$的特征向量。$\boldsymbol{A}$的非零奇异值是$\boldsymbol{A}^T\boldsymbol{A}$的特征值的平方根。SVD最有用的特征是我们可以用它来将矩阵求逆部分推广到非方矩阵。

  In the expression of The singular value decomposition of matrix $\boldsymbol{A}$, suppose that $\boldsymbol{A}$ is an $m\times n$ matrix, then $\boldsymbol{U}$ is defined to be an $m\times m$ matrix, $\boldsymbol{D}$ to be an $m\times n$ matrix, and $\boldsymbol{V}$ to be an $n\times n$ matrix. Each of these matrices is defined to have a special structure. The matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are both defined to be orthogonal matrices. The matrix $\boldsymbol{D}$ is defined to be a diagonal matrix. Note that $\boldsymbol{D}$ is not necessarily square. The elements along the diagonal of $\boldsymbol{D}$ are known as the singular values of the matrix $\boldsymbol{A}$. The columns of $\boldsymbol{U}$ are known as the left-singular vectors. The columns of $\boldsymbol{V}$ are known as the right-singular vectors. We can actually interpret the singular value decomposition of $\boldsymbol{A}$ in terms of the eigendecomposition of functions of $\boldsymbol{A}$. The left-singular vectors of $\boldsymbol{A}$ are the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$. The right-singular vectors of $\boldsymbol{A}$ are the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$. The nonzero singular values of $\boldsymbol{A}$ are the square roots of the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$. Perhaps the most useful feature of the SVD is that we can use it to partially generalize matrix inversion to non square matrices.

奇异值分解
Singular Value Decomposition

- 矩阵不是方阵时，逆矩阵没定义。但实际上，假设矩阵$\boldsymbol{A}$存在一个左逆$\boldsymbol{A_l^{-1}}$，我们就可以通过每边左乘$\boldsymbol{A_l^{-1}}$，求解线性方程$\boldsymbol{Ax = b}$，得到$\boldsymbol{x = A_l^{-1}b}$。当然，一般情况下，左逆并不唯一，并且，方程组可能无解，也可能有多组解。

  Matrix inversion is not defined for matrices that are not square. Suppose we want to make a left-inverse of a matrix $\boldsymbol{A}$, like $\boldsymbol{A_l^{-1}}$, then we can solve a linear equation $\boldsymbol{Ax = b}$ by left-multiplying each side to obtain $\boldsymbol{x = A_l^{-1}b}$. In general, the left inverse of an matrix is not necessary unique, meantime, it is possible for this equation to have no solution, or multiple possible solutions.

- 实际上，计算这个左逆$\boldsymbol{A_l^{-1}}$基于的正是奇异值分解：$\boldsymbol{A_l^{-1} = VD^{-}U^T}$，其中$\boldsymbol{U}$、$\boldsymbol{D}$和$\boldsymbol{V}$是对$\boldsymbol{A}$进行奇异值分解时的相应的矩阵，对角矩阵$\boldsymbol{D^{-}}$称为$\boldsymbol{D}$的伪逆，通过取$\boldsymbol{D}$非零元素的倒数，然后进行转置得到。验证如右：

  In fact, the computation of left inverse is based on SVD: $\boldsymbol{A_l^{-1} = VD^{-}U^T}$, where $\boldsymbol{U}$, $\boldsymbol{D}$ and $\boldsymbol{V}$ are the singular value decomposition of $\boldsymbol{A}$, and $\boldsymbol{D^{-}}$ is a diagonal matrix, called the pseudoinverse of $\boldsymbol{D}$, obtained by taking the reciprocal of nonzero elements of $\boldsymbol{D}$ then taking the transpose of the resulting matrix. To prove it, we have deduction on the right:

$$
\begin{aligned}
&\boldsymbol{A_l^{-1}A = VD^{-}U^TA} \\
\Longrightarrow\ &\boldsymbol{A_l^{-1}A = VD^{-}U^TUDV^T} \\
\Longrightarrow\ &\boldsymbol{A_l^{-1}A = VD^{-}I_mDV^T} \\
\Longrightarrow\ &\boldsymbol{A_l^{-1}A = VD^{-}DV^T} \\
\Longrightarrow\ &\boldsymbol{A_l^{-1}A = VE_nV^T} \\
\Longrightarrow\ &\boldsymbol{A_l^{-1}A = E_nVV^T} \\
\Longrightarrow\ &\boldsymbol{A_l^{-1}A = E_nI_n} \\
\Longrightarrow\ &\boldsymbol{A_l^{-1}A = E_n}
\end{aligned}
$$

注意，$\boldsymbol{E_n}$是对角矩阵，但对角线上元素可能为零。

迹、行列式
Trace, Determinant

- 矩阵的迹定义为矩阵所有对角线元素的总和：$\mathrm{Tr}(\boldsymbol{A}) = \sum_i A_{i,i}$，标量的迹是它本身：$a = \mathrm{Tr}(a)$。迹有许多有用的性质，如矩阵与其转置矩阵具有相同的迹：$\mathrm{Tr}(\boldsymbol{A}) = \mathrm{Tr}(\boldsymbol{A}^T)$。其满足交换律，即对$\boldsymbol{A} \in \mathbb{R}^{m\times n}$，$\boldsymbol{B} \in \mathbb{R}^{n\times m}$，$\mathrm{Tr}(\boldsymbol{AB}) = \mathrm{Tr}(\boldsymbol{BA})$。矩阵$\boldsymbol{A}$的Frobenius范数可写作$\|\boldsymbol{A}\|_F = \sqrt{\mathrm{Tr}(\boldsymbol{AA}^T)}$。

  The trace of a matrix the is the sum of all the diagonal entries: $\mathrm{Tr}(\boldsymbol{A}) = \sum_i A_{i,i}$. A scalar is its own trace: $a = \mathrm{Tr}(a)$. The trace operator is endowed with many useful properties, for example, it is invariant to the transpose: $\mathrm{Tr}(\boldsymbol{A}) = \mathrm{Tr}(\boldsymbol{A}^T)$. It is invariant to permutation, that is for $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times m}$, $\mathrm{Tr}(\boldsymbol{AB}) = \mathrm{Tr}(\boldsymbol{BA})$. An alternative way of writing the Frobenius norm of a matrix $\boldsymbol{A}$ is $\|\boldsymbol{A}\|_F = \sqrt{\mathrm{Tr}(\boldsymbol{AA}^T)}$.

- 方阵的行列式是将矩阵映射到实数的函数，表示为$\det(\boldsymbol{A})$。行列式等于矩阵所有特征值的乘积，其绝对值可以被认为是矩阵扩展或收缩空间的量度。如果行列式为0，则空间至少沿一个维度完全塌缩，导致它失去高维体积；如果行列式为1，则转换保留体积。

  The determinant of a square matrix, denoted $\det(\boldsymbol{A})$, is a function that maps matrices to real scalars. The determinant is equal to the product of all the eigenvalues of the matrix. The absolute value of the determinant can be thought of as a measure of how much multiplication by the matrix expands or contracts space. If the determinant is 0, then space is contracted completely along at least one dimension, causing it to lose all its volume. If the determinant is 1, then the transformation preserves volume.

例子：主成分分析
Example: Principal Components Analysis

- 一个可以仅使用基本线性代数知识推导出来的简单的机器学习算法是主成分分析 (PCA)。假设有$\mathbb{R}^n$中的m个点的集合$\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}\}$，我们想对这些点应用有损压缩，这意味着用较少内存存储这些点但以损失一些精度为代价。当然，我们希望损失的精度尽可能小。

  One simple machine learning algorithm, principal components analysis (PCA), can be derived using only knowledge of basic linear algebra. Suppose we have a collection of m points $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}\}$ in $\mathbb{R}^n$ and we want to apply lossy compression to these points. Lossy compression means storing the points in a way that requires less memory but may lose some precision. We want to lose as little precision as possible.

- 我们可以选择的一种方法是在低维空间对这些点进行编码。对于任一$\boldsymbol{x}^{(i)} \in \mathbb{R}^n$，我们将找到一个对应的编码向量$\boldsymbol{c}^{(i)} \in \mathbb{R}^l$。如果$l$比$n$小，则存储新编码值将比存储原始数据占用更少的内存。我们希望要找到一个编码函数来为输入生成编码：$\boldsymbol{c} = f(\boldsymbol{x})$，并找到一个解码函数来生成给定其代码的重构输入：$\boldsymbol{x} \approx g(\boldsymbol{c})$。PCA即由选择的解码函数定义。

  One way we can encode these points is to represent a lower-dimensional version of them. For each point $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ we will find a corresponding code vector $\boldsymbol{c}^{(i)} \in \mathbb{R}^l$. If l is smaller than n, storing the code points will take less memory than storing the original data. We will want to find some encoding function that produces the code for an input: $\boldsymbol{c} = f(\boldsymbol{x})$, and a decoding function that produces the reconstructed input given its code: $\boldsymbol{x} \approx g(\boldsymbol{c})$. PCA is thus defined by our choice of the decoding function.

例子：主成分分析
Example: Principal Components Analysis

- 为了使解码器简单，我们选择使用矩阵乘法将编码映射回$\mathbb{R}^n$。令$g(\boldsymbol{c}) = \boldsymbol{D}\boldsymbol{c}$，其中$\boldsymbol{D} \in \mathbb{R}^{n \times l}$是定义解码的矩阵。为了简化编码问题，PCA将$\boldsymbol{D}$的列约束为彼此正交。为了给问题一个唯一解，我们进一步将$\boldsymbol{D}$的所有列都限制为具有单位范数。

  To make the decoder very simple, we choose to use matrix multiplication to map the code back into $\mathbb{R}^n$. Let $g(\boldsymbol{c}) = \boldsymbol{D}\boldsymbol{c}$, where $\boldsymbol{D} \in \mathbb{R}^{n \times l}$ is the matrix defining the decoding. To keep the encoding problem easy, PCA constrains the columns of $\boldsymbol{D}$ to be orthogonal to each other. To give the problem a unique solution, we constrain all the columns of $\boldsymbol{D}$ to have unit norm..

- 我们需要做的第一件事就是确定每个输入点$\boldsymbol{x}$的最佳编码值$\boldsymbol{c}^*$。一种方法是最小化输入$\boldsymbol{x}$与其重构$g(\boldsymbol{c}^*)$之间的距离。在主成分算法中，我们使用L^2范数来衡量这个距离。由于平方L^2范数与L^2范数在最小化$\boldsymbol{c}$值上等价，我们实际上使用平方L^2范数：

  The first thing we need to do is figure out how to generate the optimal code point $\boldsymbol{c}^*$ for each input point $\boldsymbol{x}$. One way to do this is to minimize the distance between the input point $\boldsymbol{x}$ and its reconstruction, $g(\boldsymbol{c}^*)$. In the principal components algorithm, we use the L^2 norm. However, since the squared L^2 and the L^2 norm are equivalent in minimizing the same value of $\boldsymbol{c}$, actually we adopt the squared L^2 norm:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} \|\boldsymbol{x} - g(\boldsymbol{c})\|_2^2$$

例子：主成分分析
Example: Principal Components Analysis

- 根据平方L^2范数的定义，上述最小化的等式等价于：

  By the definition of the L^2 norm, The function being minimized simplifies to:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} \left(\boldsymbol{x} - g(\boldsymbol{c})\right)^T \left(\boldsymbol{x} - g(\boldsymbol{c})\right)$$
$$= \arg\min_{\boldsymbol{c}} \left(\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T g(\boldsymbol{c}) + g(\boldsymbol{c})^T g(\boldsymbol{c})\right)$$

- 由于第一项为定值，我们可以去掉；我们将$\boldsymbol{D}$代入继续化简，由于$\boldsymbol{D}$的列向量正交且范数为1，我们最终有：

  We can omit the first term since it is a fixed term. We can substitute $g(\boldsymbol{c})$ with $\boldsymbol{D}$ for further simplification. By the orthogonality and unit norm constraints on $\boldsymbol{D}$, we further have:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} \left(-2\boldsymbol{x}^T g(\boldsymbol{c}) + g(\boldsymbol{c})^T g(\boldsymbol{c})\right)$$
$$= \arg\min_{\boldsymbol{c}} (-2\boldsymbol{x}^T\boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^T\boldsymbol{D}^T\boldsymbol{D}\boldsymbol{c})$$
$$= \arg\min_{\boldsymbol{c}} (-2\boldsymbol{x}^T\boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^T\boldsymbol{I}_l\boldsymbol{c}) = \arg\min_{\boldsymbol{c}} (-2\boldsymbol{x}^T\boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^T\boldsymbol{c})$$

- 我们不再证明，$\boldsymbol{D}$的构造如下：
  - 构造矩阵$\boldsymbol{X}$，其中$\boldsymbol{X}_{i,:} = \boldsymbol{x}^{(1)}$
  - 求解$\boldsymbol{x}^T\boldsymbol{x}$的特征值$\{\lambda_i\}$与对应的特征向量$\{\boldsymbol{v}_i\}$，满足$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$
  - 构造矩阵$\boldsymbol{D}$，其中$\boldsymbol{D}_{:,j} = \boldsymbol{v}_j$，$j \leq l$。
  - 最后$\boldsymbol{c} = \boldsymbol{D}^T\boldsymbol{x}$，$\boldsymbol{x} = \boldsymbol{D}\boldsymbol{c}$

JAX例子
JAX Example

- 下面代码展示了JAX对上面所讲内容的支持:

  The following code demonstrates the support of JAX for content lectured above:

```
A = jax.numpy.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=jax.numpy.float32)
A_diag = jax.numpy.diag(A) # DeviceArray([1., 5., 9.], dtype=float32)
a = jax.numpy.array([1, 2, 3], dtype=jax.numpy.float32)
diag_from_a = jax.numpy.diag(a)  # DeviceArray([[1., 0., 0.], [0., 2., 0.], [0., 0., 3.]], dtype=float32)
a = jax.numpy.expand_dims(a, axis=0) # DeviceArray([[1., 2., 3.]], dtype=float32)
diag_from_a = jax.numpy.diag(a) # DeviceArray([1.], dtype=float32)
S = A + A.T # DeviceArray([[ 2.,  6., 10.], [ 6., 10., 14.], [10., 14., 18.]], dtype=float32)
w, v = jax.numpy.linalg.eig(S)
# w: [ 3.291648e+01+0.j, -2.916473e+00+0.j,  5.492732e-07+0.j]
# v: [[-0.35162514+0.j, -0.84243274+0.j,  0.40824804+0.j],
      [-0.55335623+0.j, -0.16471314+0.j, -0.8164966 +0.j],
      [-0.7550872 +0.j,  0.51300746+0.j,  0.4082485 +0.j]]
```

JAX例子
JAX Example

- 下面代码展示了JAX对上面所讲内容的支持：

  The following code demonstrates the support of JAX for content lectured above:

```
A = jax.random.normal(jax.random.PRNGKey(0), (3, 4))
U, S, V_h = jax.numpy.linalg.svd(A)
# U: [[ 0.51770115, -0.8475058 ,  0.11712989],
        [-0.8079263 , -0.4392296 ,  0.39285186],
        [ 0.28149736,  0.2980122 ,  0.9121117 ]]
# S: [2.6625412, 1.3281084, 1.0067053]
# V_h: [[ 0.32766604, -0.5286418 , -0.12515056,  0.7729877 ],
        [-0.67797697,  0.2505759 , -0.5876598 ,  0.36361343],
        [-0.20869376, -0.7544863 , -0.38393393, -0.4896854 ],
        [ 0.62404245,  0.2974891 , -0.70113325, -0.17459458]]
t = jax.numpy.trace(A) # DeviceArray(1.2038532, dtype=float32)
r = jax.numpy.trace(jax.numpy.matmul(A, A.T)) == jax.numpy.trace(jax.numpy.matmul(A, A.T)) # DeviceArray(True, dtype=bool)
```

JAX例子
JAX Example

- 下面代码展示了JAX对上面所讲内容的支持:

  The following code demonstrates the support of JAX for content lectured above:

```
import jax.random as random
import jax.numpy as jnp
def pca(X, l):
  n, m = X.shape # Data matrix X, assumes 0-centered
  C = jnp.matmul(X.T, X) / (n-1) # Compute covariance matrix
  eigen_vals, eigen_vecs = jnp.linalg.eig(C) # Eigen decomposition
  eigen_vecs = jnp.real(eigen_vecs[:, :l])
  X_pca = jnp.dot(X, eigen_vecs) # Project X onto PC space
  return X_pca
A = random.normal(random.PRNGKey(0), (3, 4))
A = A-A.mean(axis=0) # Normalize the data
A_pca = pca(A, 2) # [[-1.3866825   0.7750936 ], [ 2.1432798   0.16841218], [-0.7565974  -0.9435059 ]]
```