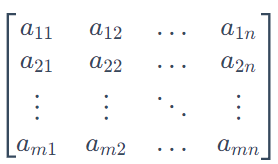
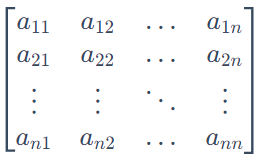
## 2.2.1 编程线性代数

如果想对某一解决问题的方法有比较清晰的理解，数学知识则很难规避，线性代数则是其中之一。回归部分方程式的矩阵求解，及特征向量与降维（例如主成成分分析，Principal component analysis，PCA），空间变换等内容都需要基础的线性代数知识。因此以《漫画线性代数》[1]结构为主要思路，结合*Gilbert Strang.Introduction to linear algbra*[2]及相关参考文献，借助Python编程语言，结合Python图表打印可视化，将主要的知识点串联起来，作为用到线性代数知识解决相关问题的基础。主要使用的库为[SymPy（Matrices-linear algebra）](https://docs.sympy.org/latest/modules/matrices/matrices.html)①。

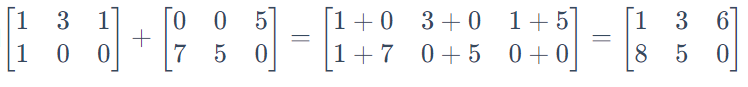
线性代数（linear algebra）是关于向量空间和线性映射的一个数学分支，包括对线、面和子空间的研究，同时也涉及到所有的向量空间的一般性质。笼统的说，是使用向量和矩阵的数学空间表达方式，将m维空间（世界）与n维空间（世界）联系起来的数学分支。在规划设计学科，实际上rhinoceros（grasshopper）等三维软件平台对大量几何对象空间变换的操作就是对线性代数的应用。

### 2.2.1.1 矩阵

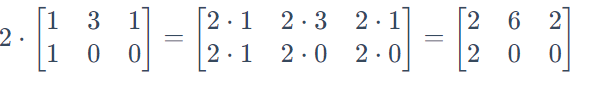
形如的数据集合即为m行n列， m×n矩阵，这同于NumPy的数组形式，也可以用Pandas的DataFrame格式表达。在统计学上就可以通过矩阵方式对样本特征表述，尤其具有多个特征的样本。而m可以称为行标， n则称为列标，组合起来就代表了矩阵中的每个元素。对于形如，即m=n的矩阵为n阶方阵，对角线上的元素称为对角元素。在相关性分析中，所计算两两相关性的数组结果，就是方阵。其对角元素皆为1，为自身相关性计算结果。

#### 1）矩阵的运算

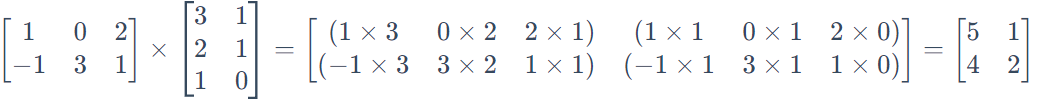
1. 加（减）法（addition，subtraction ）

m×n矩阵A和B的和（差）：A±B为一个m×n矩阵，其中每个元素是A和B相应元素的和（差）， (A±B)i,j​=Ai,j​±Bi,j​，例如

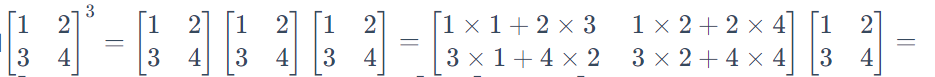
1. 数乘（倍数，multiple）

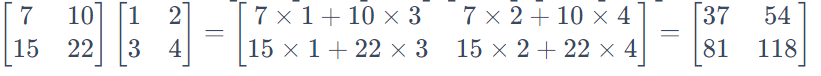
标量c与矩阵A的数乘，cA的每个元素是A的相应元素与c的乘积， (cA)i,j​=c⋅Ai,j​，例如

1. 乘法（Multiplication）

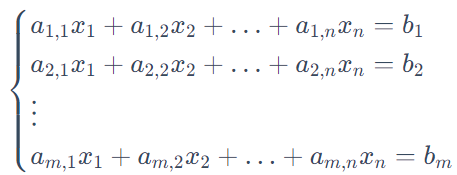
两个矩阵的乘法仅当第1个矩阵A的列数（column）和另一个矩阵B的行数（row）相等时才能定义。如果A是m×n矩阵， B是n×p，它们的乘积AB，是一个m×p矩阵， [AB]i,j​=Ai,1​B1,j​+Ai,2​B2,j​+…+Ai,n​Bn,j​=∑r=1n​Ai,r​Br,j​，例如

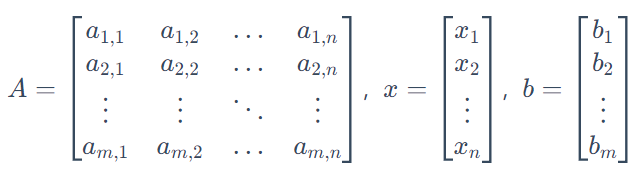
1. 幂（exponentiation）

相等于矩阵乘法，例如



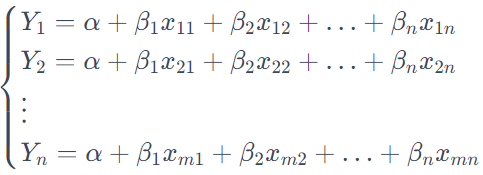
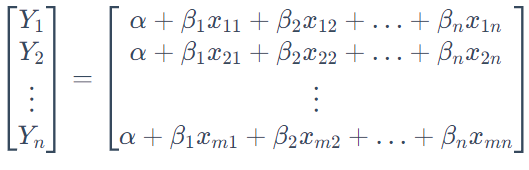
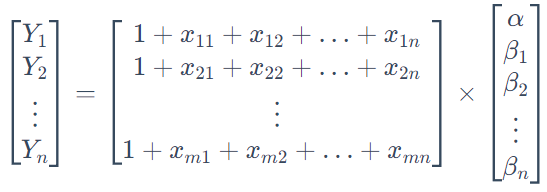
* 线性方程组

矩阵乘法的一个基本应用是在线性方程组上，线性方程组是数学方程组的一种，它符合以下的形式：，式中a1,1​，a1,2​，及b1​，b2​等等是已知的常数，而x1​，x2​等等则是要求的未知数。如果用线性代数中的概念来表达，则线性方程组可写为： Ax=b，其中A是m×n矩阵， x是含有n个元素列向量， b是含有m个元素列向量。



这是线性方程组的另一种记录方法，已知矩阵和向量的情况下求得未知向量是线性代数的基本问题。

在多元回归部分，阐述过使用矩阵的方式求解回归模型，对于样本包含多个特征（解释变量）即n列，多个样本（样本容量）即m行，常规表达为y=α+β1​x1​+β2​x2​+…+βn​xn​，其中x1​,x2​,…,xn​为n个特征，每个特征下实际上包含有多个样本实例（m行），即

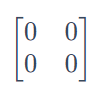
，表达为矩阵形式为，可以进一步简化为，，实际上就是应用矩阵的乘法。

import sympy  
from sympy import Matrix,pprint  
  
A=Matrix([[1,3,1],[1,0,0]])  
B=Matrix([[0,0,5],[7,5,0]])  
print("A 矩阵=")  
pprint(A)  
print("B 矩阵=")  
pprint(B)  
print("-"\*50)  
  
print("矩阵加法：A+B=")  
pprint(A+B)  
print("\_"\*50)  
  
print("数乘：2\*A=")  
pprint(2\*A)  
print("\_"\*50)  
  
C=Matrix([[1,0,2],[-1,3,1]])  
D=Matrix([[3,1],[2,1],[1,0]])  
print("C 矩阵=")  
pprint(C)  
print("D 矩阵=")  
pprint(D)  
print("-"\*50)  
  
print("乘法：C\*D=")  
pprint(C\*D)  
print("\_"\*50)  
  
E=Matrix([[1,2],[3,4]])  
print("E 矩阵=")  
pprint(E)  
print("-"\*50)  
  
print("幂：E\*\*3")  
pprint(E\*\*3)  
print("\_"\*50)

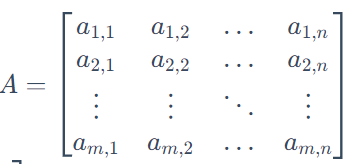
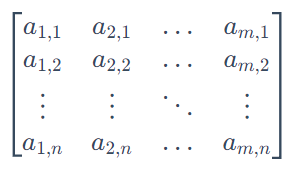
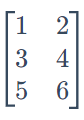
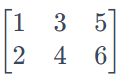
A 矩阵=  
⎡1 3 1⎤  
⎢ ⎥  
⎣1 0 0⎦  
B 矩阵=  
⎡0 0 5⎤  
⎢ ⎥  
⎣7 5 0⎦  
--------------------------------------------------  
矩阵加法：A+B=  
⎡1 3 6⎤  
⎢ ⎥  
⎣8 5 0⎦  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
数乘：2\*A=  
⎡2 6 2⎤  
⎢ ⎥  
⎣2 0 0⎦  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
C 矩阵=  
⎡1 0 2⎤  
⎢ ⎥  
⎣-1 3 1⎦  
D 矩阵=  
⎡3 1⎤  
⎢ ⎥  
⎢2 1⎥  
⎢ ⎥  
⎣1 0⎦  
--------------------------------------------------  
乘法：C\*D=  
⎡5 1⎤  
⎢ ⎥  
⎣4 2⎦  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
E 矩阵=  
⎡1 2⎤  
⎢ ⎥  
⎣3 4⎦  
--------------------------------------------------  
幂：E\*\*3  
⎡37 54 ⎤  
⎢ ⎥  
⎣81 118⎦  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#### 2）特殊矩阵

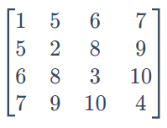
1. 零矩阵

所有元素均为0的矩阵，例如，

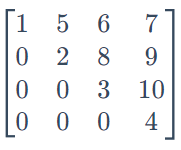
1. 转置矩阵（transpose）

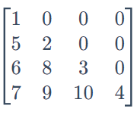
是指将m×n矩阵，交换行列后得到的n×m，，可以用和′表示，例如3×2的矩阵​​的转置矩阵为2×3矩阵。

1. 对称矩阵（symmetric matrix）

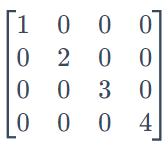
以对角元素为中心线对称的n阶方阵，例如​​，对称矩阵与其转置矩阵完全相同。

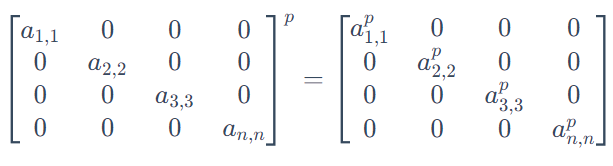
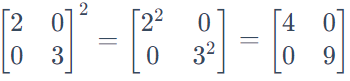
1. (5.)，上三角矩阵和下三角矩阵（triangular matrix）

如​​，对角元素左下角的所有元素均为0的n阶矩阵。

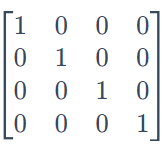
如,对角元素右上角的所有元素均为0的n阶矩阵。

1. 对角矩阵 (diagonal matrix)

如，对角元素以外的元素均为0的n阶矩阵，可表示为diag(1,2,3,4)。对角矩阵的p次幂，等于对角元素的p次幂，公式为：

，例如

1. 单位矩阵（identity matrix）

如，对角元素均为1，对角元素以外的其它元素全部为0的n阶方阵，即diag(1,1,…,1)。单位矩阵与任何矩阵相乘，都对这个矩阵没有任何影响。

1. 逆矩阵 （inverse matrix）

又称反矩阵，在线性代数中，给定一个n阶方阵A，若存在n阶方阵B，使得AB=BA=In​，其中In​为n阶单位矩阵，则称A是可逆的，且B是A的逆矩阵，记作。只有方阵n×n才可能有逆矩阵。若方阵A的逆矩阵存在，则称A为非奇异方阵或可逆矩阵。逆矩阵的求法有代数余子式法（不实用），消元法等，对解法感兴趣的可以参看《漫画线性代数》。在代码的世界里，直接使用SymPy库提供的方法。

并不是所有的方阵都有逆矩阵，可以使用行列式指标determinant(det)，用SymPy库的det()方法计算判断。如果，则矩阵A可逆。

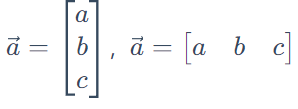
print("零矩阵：")  
pprint(sympy.zeros(2))  
  
print("转置矩阵：")  
F=Matrix([[1,2],[3,4],[5,6]])  
pprint(F.T)  
  
print("对称矩阵的转置矩阵：")  
G=Matrix([[1,5,6,7],[5,2,8,9],[6,8,3,10],[7,9,10,4]])  
pprint(G.T)  
  
print("对角矩阵的2次幂：")  
H=Matrix([[2,0],[0,3]])  
pprint(H\*\*2)  
  
print("单位矩阵：")  
pprint(sympy.eye(4))  
  
print("单位矩阵与任何矩阵相乘，都对这个矩阵没有任何影响:")  
pprint(sympy.eye(4)\*G)  
  
print("求解逆矩阵，并乘以自身：")  
print("使用-1次方计算逆矩阵:")  
G\_inverse=G\*\*-1  
pprint(G\*G\_inverse)  
print("使用.inv()计算逆矩阵:")  
G\_inverse\_=G.inv()  
pprint(G\*G\_inverse\_)  
  
print("判断方阵是否有逆矩阵：")  
print(G.det())  
print(G\_inverse.det())

零矩阵：  
⎡0 0⎤  
⎢ ⎥  
⎣0 0⎦  
转置矩阵：  
⎡1 3 5⎤  
⎢ ⎥  
⎣2 4 6⎦  
对称矩阵的转置矩阵：  
⎡1 5 6 7 ⎤  
⎢ ⎥  
⎢5 2 8 9 ⎥  
⎢ ⎥  
⎢6 8 3 10⎥  
⎢ ⎥  
⎣7 9 10 4 ⎦  
对角矩阵的2次幂：  
⎡4 0⎤  
⎢ ⎥  
⎣0 9⎦  
单位矩阵：  
⎡1 0 0 0⎤  
⎢ ⎥  
⎢0 1 0 0⎥  
⎢ ⎥  
⎢0 0 1 0⎥  
⎢ ⎥  
⎣0 0 0 1⎦  
单位矩阵与任何矩阵相乘，都对这个矩阵没有任何影响:  
⎡1 5 6 7 ⎤  
⎢ ⎥  
⎢5 2 8 9 ⎥  
⎢ ⎥  
⎢6 8 3 10⎥  
⎢ ⎥  
⎣7 9 10 4 ⎦  
求解逆矩阵，并乘以自身：  
使用-1次方计算逆矩阵:  
⎡1 0 0 0⎤  
⎢ ⎥  
⎢0 1 0 0⎥  
⎢ ⎥  
⎢0 0 1 0⎥  
⎢ ⎥  
⎣0 0 0 1⎦  
使用.inv()计算逆矩阵:  
⎡1 0 0 0⎤  
⎢ ⎥  
⎢0 1 0 0⎥  
⎢ ⎥  
⎢0 0 1 0⎥  
⎢ ⎥  
⎣0 0 0 1⎦  
判断方阵是否有逆矩阵：  
-3123  
-1/3123

### 2.2.1.2 向量 （euclidean vector）

#### 1）向量概念，表达及运算

一般同时满足具有大小和方向两个性质的几何对象即可认为是向量，其代数表示指，在指定了一个坐标系之后，用一个向量在该坐标系下的坐标来表示该向量，兼具符号的抽象性和几何形象性，具有最高的实用性，广泛应用于定量分析的情景。对于自由向量，将向量的起点平移到坐标原点后，向量就可以用一个坐标系下的一个点来表示，该点的坐标值即向量的终点坐标。

设一个向量，有坐标系S。在S中定义好若干个特殊的基本向量（称为基向量Base Vector，各个基向量共同组成该坐标系下的基底）​​之后，则向量在各个基向量下的投影即为对应的坐标系，各个投影值组成了该向量在该坐标系S下可唯一表示的有序数组（即坐标），且与向量的终点一一对应。换言之，其它的向量只需通过将这些基本向量拉伸后再按照平行四边形法则进行向量加法即可表示（通常被称为“用基底线性表出一个向量”，即该向量是基向量的某种线性组合），即：，其中​​是分别在​​下对应的投影。当基底已知，可直接省略各基向量的符号，类似于坐标系上的点， 直接用坐标表示为。在矩阵运算中，向量更多的被写成类似于矩阵的列向量或行向量。在线性代数中所指的向量，通常默认为列向量。如一个向量可以写成：，其中第一个为列向量写法，第二个为行向量写法。n维列向量可视为n×1矩阵， n维行向量可视为1×n矩阵。

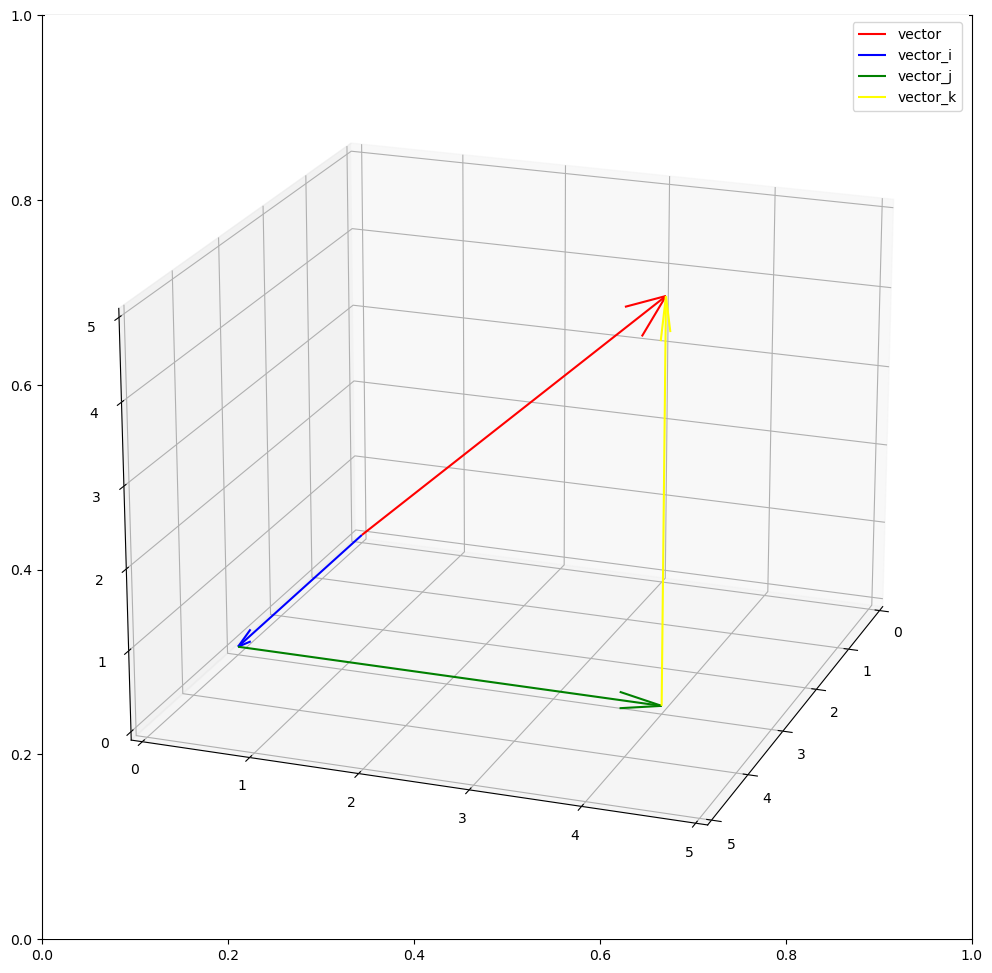
在常见的三维空间直角坐标系◯xyz（三维笛卡尔坐标系）里，基本向量就是行轴（◯x）、竖轴（◯y）、及纵轴（◯z）为方向的三个长度为1的单位向量，即基向量。这三个向量取好之后，其它的向量就可以透过三元数组来表示，因为它们可以表示成一定倍数的三个基本向量的总和。比如说一个标示为（2,1,3）的向量就是2个向量加上1个向量​加上3个向量得到的向量，即：。因此向量本质上讲即为矩阵，向量的计算同于矩阵的计算。

SymPy库对向量计算有两种方式，一种是完全使用矩阵，因为二者计算相同，例如和、差、倍数和积；另一种是该库提供了专门针对向量的类：“Vector”，通过Vector提供的方法可以实现向量的表达及向量的基本运算。向量的主要目的是要解决空间几何问题，因此对于向量的理解要将空间几何图形与表达式结合起来，才能够有效的理解前因后果。在了解基本的向量概念、坐标空间、运算之后，如果要用代码来表达和计算向量，有必要阅读SymPy对应的[Vector](https://docs.sympy.org/dev/modules/vector/index.html)②及[Matrices（linear algebra）](https://docs.sympy.org/latest/modules/matrices/matrices.html)③两个部分，那么再阅读下述代码将会相对轻松。

为了方便使用matplotlib打印三维空间下向量，定义了vector\_plot\_3d函数。首先定义了名称为’C’的三维坐标系统，获取其基向量’i,j,k’，用基向量构建向量’v1’，同时通过提取该向量在三个轴向上的系数建立v1向量在各轴上的分量（投影向量），各轴的投影向量之和即为v1向量，打印三维空间上的向量图形，观察向量间的关系。注意，向量空间起始位置的确定，vector\_k的起始点，以vector\_i和vector\_j之和的向量作为起点，而vector\_j以vector\_i作为起点。

* 把n×1向量​​​，所有分量构成的集合表示为Rn。

import matplotlib.pyplot as plt  
from sympy.vector.coordsysrect import CoordSys3D  
from sympy.vector.vector import Vector, BaseVector  
from sympy.vector import Vector  
  
def vector\_plot\_3d(ax\_3d,C,origin\_vector,vector,color='r',label='vector',arrow\_length\_ratio=0.1):  
 '''  
 funciton - 转换SymPy的vector及Matrix数据格式为matplotlib可以打印的数据格式  
   
 Params:  
 ax\_3d - matplotlib的3d格式子图；ax(matplotlib)  
 C - /coordinate\_system - SymPy下定义的坐标系；CoordSys3D()  
 origin\_vector - 如果是固定向量，给定向量的起点（使用向量，即表示从坐标原点所指向的位置），如果是自由向量，起点设置为坐标原点；vector(SymPy)  
 vector - 所要打印的向量；vector(SymPy)  
 color - 向量色彩，The default is 'r'；string  
 label - 向量标签，The default is 'vector'；string  
 arrow\_length\_ratio - 向量箭头大小，The default is 0.1；float   
   
 Returns:  
 None  
 '''  
 origin\_vector\_matrix=origin\_vector.to\_matrix(C)  
 x=origin\_vector\_matrix.row(0)[0]  
 y=origin\_vector\_matrix.row(1)[0]  
 z=origin\_vector\_matrix.row(2)[0]  
   
 vector\_matrix=vector.to\_matrix(C)  
 u=vector\_matrix.row(0)[0]  
 v=vector\_matrix.row(1)[0]  
 w=vector\_matrix.row(2)[0]  
 ax\_3d.quiver(x,y,z,u,v,w,color=color,label=label,arrow\_length\_ratio=arrow\_length\_ratio)  
  
fig, ax=plt.subplots(figsize=(12,12))  
ax=fig.add\_subplot( projection='3d')  
   
# 定义坐标系统，及打印向量v1=3\*i+4\*j+5\*k  
C=CoordSys3D('C')  
i, j, k = C.base\_vectors()  
v1=3\*i+4\*j+5\*k  
v1\_origin=Vector.zero  
vector\_plot\_3d(ax,C,v1\_origin,v1,color='r',label='vector',arrow\_length\_ratio=0.1)  
  
# 打印向量v1=3\*i+4\*j+5\*k在轴上的投影  
v1\_i=v1.coeff(i)\*i  
vector\_plot\_3d(ax,C,v1\_origin,v1\_i,color='b',label='vector\_i',arrow\_length\_ratio=0.1)  
  
v1\_j=v1.coeff(j)\*j  
vector\_plot\_3d(ax,C,v1\_i,v1\_j,color='g',label='vector\_j',arrow\_length\_ratio=0.1)  
  
v1\_k=v1.coeff(k)\*k  
vector\_plot\_3d(ax,C,v1\_i+v1\_j,v1\_k,color='yellow',label='vector\_k',arrow\_length\_ratio=0.1)  
  
ax.set\_xlim3d(0,5)  
ax.set\_ylim3d(0,5)  
ax.set\_zlim3d(0,5)  
  
ax.legend()  
ax.view\_init(20,20) # 可以旋转图形的角度，方便观察  
plt.show()

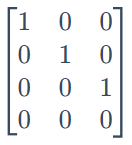
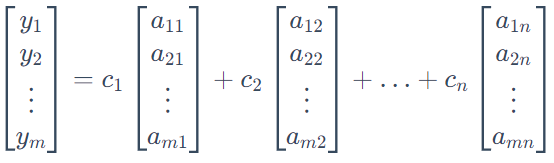
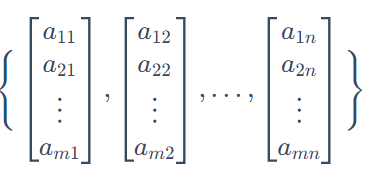


#### 2）线性无关

在线性代数里，向量空间的一组元素中，若没有向量可用有限个其它向量的线性组合所表示，则称为线性无关（linearly independent）或线性独立（例如，(1, 0, 0)，(0, 1, 0)和(0, 0, 1)），反之称为线性相关（linearly dependent）（例如(2, −1, 1)，(1, 0, 1)和(3, −1, 2)，因为第3个是前两个的和）。即假设V在域K上的向量空间，如果从域K中有非全零的元素a1​,a2​,…,an​，使得a1​v1​+a2​v2​+…+an​vn​=0，或表示为，，其中v1​,v2​,…,vn​是V的向量，称它们为线性相关，其中右边的0，为即0向量（vector），而不是0标量（scalar）；如果K中不存在这样的元素，那么v1​,v2​,…,vn​线性无关。对线性无关可以给出更直接的定义，向量v1​,v2​,…,vn​线性无关，当且仅当它们满足以下条件：如果a1​,a2​,…,an​是K的元素，适合： a1​v1​+a2​v2​+…+an​vn​=0，那么对所有i=1,2,…,n都有ai​=0。

在V中的一个无限集，如果它任何一个有限子集都是线性无关，那么原来的无限集也是线性无关。线性相关性是线性代数的重要概念，因为线性无关的一组向量可以生成一个向量空间，而这组向量则是这个向量空间的基。

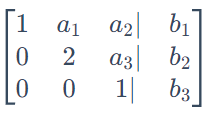
首先通过sympy.vector的CoordSys3D方法建立三维坐标系（可以直接提取单位向量），依托该坐标系建立（V）向量集合，包括v2、v3、v4、v5，均由单位向量的倍数建立，如果倍数等于0，例如v2=a\*1\*N.i+a\*0\*N.j，也保持了0的存在，以保持各个方向上的一致性，便于观察。可以通过打印各个单独向量，查看变量在SymPy中的表现形式。该向量集合是线性相关，可以给出除了a=b=c=d=0外，其它a、b、c、d的值，即有非全零的元素，例如a\_,b\_,c\_,d\_=1,2,0,-1，或者a\_,b\_,c\_,d\_=1,-3,-1,2，均也满足a1​v1​+a2​v2​+…+an​vn​=0，上述的a、b、c、d即为a1​,a2​,…,an​。因此可以打印图形，通过定义move\_alongVectors函数实现，可以看到第1，2个图形，形成闭合平面二维图形（回到起点）。同样在三维空间中，定义了向量集合v6、v7、v8、v9，因为线性相关，如第3个图形，形成空间闭合的折线（回到起点）。

判断一个向量集合V，是否线性无关，可以使用matrix.rref()将表示向量的矩阵转换为行阶梯形矩阵（Row Echelon Form），如果返回的简化行阶梯形矩阵末尾行存在全零值，则表明该向量数据集为线性相关。例如，使用matrix.to\_matrix(C)方法将v2、v3、v4、v5等向量转换为矩阵表达形式，并通过matrix.col\_insert()方法合并为矩阵，使用matrix.rref()计算，对于数据集v2、v3、v4、v5，其简化行阶梯形矩阵计算结果为​​，末尾包括全零行，因此为线性相关。同时返回了主元位置的列表为（0，1，2）。而v10、v11组成的向量集合则末尾不含全零行，因此为线性无关，无法通过各个向量构建闭合的空间折线，则可以生成向量空间，这组向量就是这个向量空间的基。对于Rm​的任意元素向量​​​，当的解c1​,c2​,…,cn​均为零时，则把集合叫做基，即为了表示Rm​任意元素所必需的最少向量构成的集合。

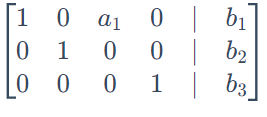
·阶梯型矩阵

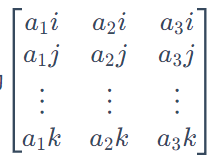
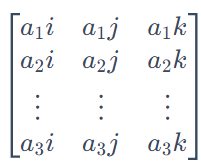
线性代数中，一个矩阵如果符合下列条件的话，则称之为行阶梯型矩阵（Row Echelon Form）：

1. 所有非零行（矩阵的行至少有一个非零元素）在所有全零行的上面。即全零行都在矩阵的底部；
2. 非零行的首项系数（leading coefficient），也称主元，即最左边的首个非零元素，严格的比上面行的首项系数更靠右（有些版本会要求非零行的首项系数必须是1）；
3. 首项系数所在列，在该首项系数下面的元素都是零（前两条的推论）。

例如：。

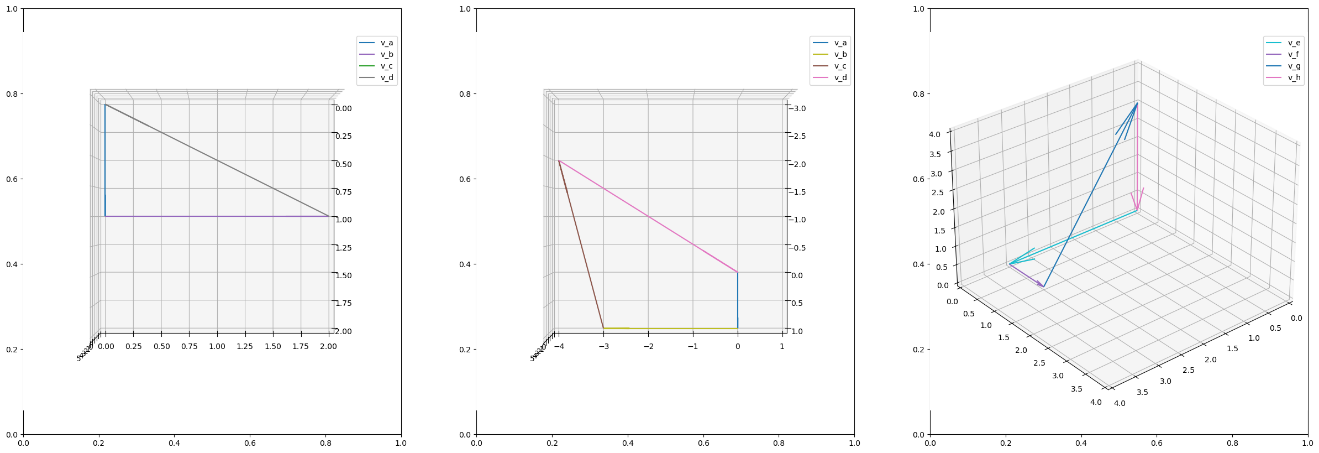
增广矩阵，又称广置矩阵，是在线性代数中系数矩阵的右边填上线性方程组等号右边的常数列得到的矩阵。例如，

化简后的行阶梯型矩阵（reduced row echelon form，或译为简约行梯形式），也称作行规范形矩阵（row canonical form），如果满足额外的条件：每个首项系数是1，且是其所在列的唯一的非零元素，例如，  
，注意化简后的行阶梯型矩阵的左部分（系数部分）不意味着总是单位阵。

通过有限步的行初等变换，任何矩阵可以变换到行阶梯形。由于行初等变换保持了矩阵的行空间，因此行阶梯型矩阵的行空间与变换前的原矩阵的行空间相同（这也是在使用sympy.rref()计算化简后的行阶梯型矩阵（reduced row echelon form），需要将形状为​的向量集合，转置矩阵变换为，保持矩阵行为。行阶梯形的结果并不是唯一的。例如，行阶梯型乘以一个标量系数仍然是行阶梯形。但是，可以证明一个矩阵的化简后的行阶梯形是唯一的。

from sympy.vector import CoordSys3D  
from sympy.abc import a, b, c,d,e,f,g,h,o,p # 相当于a,b=sympy.symbols(["a","b"])  
from sympy.vector import Vector  
from sympy import pprint,Eq,solve  
import matplotlib.pyplot as plt  
import matplotlib.colors as mcolors  
  
fig, axs=plt.subplots(1,3,figsize=(30,10))  
axs[0]=fig.add\_subplot(1,3,1, projection='3d')  
axs[1]=fig.add\_subplot(1,3,2, projection='3d')  
axs[2]=fig.add\_subplot(1,3,3, projection='3d')  
  
# A - 2维度，绘制v2+v3+v4+v5=0的解，线性相关，有多个解  
N=CoordSys3D('N')  
v2=a\*1\*N.i+a\*0\*N.j  
v3=b\*0\*N.i+b\*1\*N.j  
v4=c\*3\*N.i+c\*1\*N.j  
v5=d\*1\*N.i+d\*2\*N.j  
v0=Vector.zero  
  
def move\_alongVectors(vector\_list,coeffi\_list,C,ax,):  
 '''  
 function - 给定向量，及对应系数，延向量绘制  
   
 Params:  
 vector\_list - 向量列表，按移动顺序；list(vector)  
 coeffi\_list - 向量的系数，例如；list(tuple(symbol,float))  
 C - SymPy下定义的坐标系；CoordSys3D()  
 ax - 子图；ax(matplotlib)  
   
 Returns:  
 None  
 '''  
 import random  
 import sympy  
   
 colors=[color[0] for color in mcolors.TABLEAU\_COLORS.items()] # mcolors.BASE\_COLORS, mcolors.TABLEAU\_COLORS,mcolors.CSS4\_COLORS  
 colors\_\_random\_selection=random.sample(colors,len(vector\_list)-1)  
 v\_accumulation=[]  
 v\_accumulation.append(vector\_list[0])  
 # 每个向量绘制以之前所有向量之和为起点  
 for expr in vector\_list[1:]:  
 v\_accumulation.append(expr+v\_accumulation[-1])  
   
 v\_accumulation=v\_accumulation[:-1]   
 for i in range(1,len(vector\_list)):  
 vector\_plot\_3d(ax,C,v\_accumulation[i-1].subs(coeffi\_list),vector\_list[i].subs(coeffi\_list),color=colors\_\_random\_selection[i-1],label='v\_%s'%coeffi\_list[i-1][0],arrow\_length\_ratio=0.2)  
   
# v2+v3+v4+v5=0，向量之和为0的解,解-1  
vector\_list=[v0,v2,v3,v4,v5]  
a\_,b\_,c\_,d\_=1,2,0,-1   
coeffi\_list=[(a,a\_),(b,b\_),(c,c\_),(d,d\_)]  
move\_alongVectors(vector\_list,coeffi\_list,N,axs[0],)   
   
# v2+v3+v4+v5=0，向量之和为0的解,解-2  
vector\_list=[v0,v2,v3,v4,v5]  
a\_,b\_,c\_,d\_=1,-3,-1,2   
coeffi\_list=[(a,a\_),(b,b\_),(c,c\_),(d,d\_)]  
move\_alongVectors(vector\_list,coeffi\_list,N,axs[1],)   
  
# B - 3维度，绘制v6+v7+v8+v9=0的解，线性相关  
M=CoordSys3D('M')  
v6=e\*1\*M.i+e\*0\*M.j+e\*0\*M.k  
v7=f\*0\*M.i+f\*1\*M.j+f\*0\*M.k  
v8=g\*3\*M.i+g\*1\*M.j+g\*-3\*M.k  
v9=h\*0\*M.i+h\*0\*M.k+h\*3\*M.k  
v0=Vector.zero  
  
vector\_list=[v0,v6,v7,v8,v9]  
e\_,f\_,g\_,h\_=3,1,-1,-1  
coeffi\_list=[(e,e\_),(f,f\_),(g,g\_),(h,h\_)]  
move\_alongVectors(vector\_list,coeffi\_list,M,axs[2],)   
   
# C - 向量转换为矩阵，判断线性无关  
# C-1 - 对于向量集合v2,v3,v4,v5  
v\_2345\_matrix=v2.to\_matrix(N)  
for v in [v3,v4,v5]:  
 v\_temp=v.to\_matrix(N)  
 v\_2345\_matrix=v\_2345\_matrix.col\_insert(-1,v\_temp)  
print("v\_2345\_matrix:")  
pprint(v\_2345\_matrix)  
print("v\_2345\_matrix.T rref:")  
pprint(v\_2345\_matrix.T.rref())  
  
# C-2 - 对于向量集合v6,v7,v8,v9  
print("\_"\*50)  
v\_6789\_matrix=v6.to\_matrix(M)  
for v in [v7,v8,v9]:  
 v\_temp=v.to\_matrix(M)  
 v\_6789\_matrix=v\_6789\_matrix.col\_insert(-1,v\_temp)  
print("v\_6789\_matrix:")  
pprint(v\_6789\_matrix)  
print("v\_6789\_matrix.T rref:")  
pprint(v\_6789\_matrix.T.rref())  
  
# C-3 对于向量集合v10,v11  
print("\_"\*50)  
C=CoordSys3D('C')  
v10=o\*1\*C.i+o\*0\*C.j  
v11=p\*0\*C.i+p\*1\*C.j  
v\_10\_11\_matrix=v10.to\_matrix(C).col\_insert(-1,v11.to\_matrix(C))  
print("v\_10\_11\_matrix:")  
pprint(v\_10\_11\_matrix)  
print("v\_10\_11\_matrix.T rref:")  
pprint(v\_10\_11\_matrix.T.rref())  
  
axs[0].set\_xlim3d(0,2)  
axs[0].set\_ylim3d(0,2)  
axs[0].set\_zlim3d(0,5)  
  
axs[1].set\_xlim3d(-3,1)  
axs[1].set\_ylim3d(-4,1)  
axs[1].set\_zlim3d(0,5)  
  
axs[2].set\_xlim3d(0,4)  
axs[2].set\_ylim3d(0,4)  
axs[2].set\_zlim3d(0,4)  
  
axs[0].legend()  
axs[1].legend()  
axs[2].legend()  
axs[0].view\_init(90,0) # 可以旋转图形的角度，方便观察  
axs[1].view\_init(90,0)  
axs[2].view\_init(30,50)  
plt.show()

v\_2345\_matrix:  
⎡0 3⋅c d a⎤  
⎢ ⎥  
⎢b c 2⋅d 0⎥  
⎢ ⎥  
⎣0 0 0 0⎦  
v\_2345\_matrix.T rref:  
⎛⎡1 0 0⎤ ⎞  
⎜⎢ ⎥ ⎟  
⎜⎢0 1 0⎥ ⎟  
⎜⎢ ⎥, (0, 1)⎟  
⎜⎢0 0 0⎥ ⎟  
⎜⎢ ⎥ ⎟  
⎝⎣0 0 0⎦ ⎠  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
v\_6789\_matrix:  
⎡0 3⋅g 0 e⎤  
⎢ ⎥  
⎢f g 0 0⎥  
⎢ ⎥  
⎣0 -3⋅g 3⋅h 0⎦  
v\_6789\_matrix.T rref:  
⎛⎡1 0 0⎤ ⎞  
⎜⎢ ⎥ ⎟  
⎜⎢0 1 0⎥ ⎟  
⎜⎢ ⎥, (0, 1, 2)⎟  
⎜⎢0 0 1⎥ ⎟  
⎜⎢ ⎥ ⎟  
⎝⎣0 0 0⎦ ⎠  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
v\_10\_11\_matrix:  
⎡0 o⎤  
⎢ ⎥  
⎢p 0⎥  
⎢ ⎥  
⎣0 0⎦  
v\_10\_11\_matrix.T rref:  
⎛⎡1 0 0⎤ ⎞  
⎜⎢ ⎥, (0, 1)⎟  
⎝⎣0 1 0⎦ ⎠

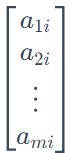
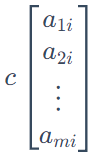
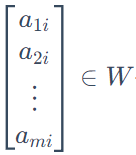
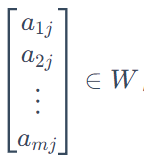
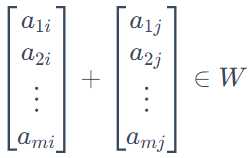


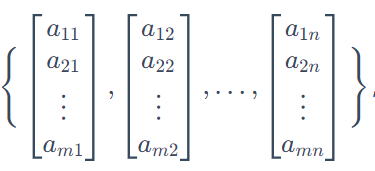
上述代码是以sympy.vector提供的向量方式建立向量数据集，能够比较直观的理解向量空间及向量在空间中的运算。下述代码则直接建立矩阵模式的向量系数矩阵并进行相关计算，包括求得行阶梯形矩阵，简化的行阶梯形矩阵，及求解系数，从而可以判断该向量集V\_C\_matrix，为线性相关，有非全零解，可以在空间中构建回到起点闭合的图形。

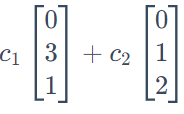
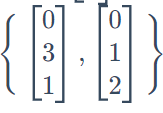
C=CoordSys3D('C')  
V\_C\_matrix=Matrix([[1,4,2,-3],[7,10,-4,-1],[-2,1,5,-4]])  
print("V\_C\_matrix=")  
pprint(V\_C\_matrix)  
print("-"\*50)  
  
lambda\_1,lambda\_2,lambda\_3=sympy.symbols(['lambda\_1','lambda\_2','lambda\_3'])  
coeffi\_expr=V\_C\_matrix.T\*Matrix([lambda\_1,lambda\_2,lambda\_3])  
print("向量集合(线性方程组)系数矩阵：")  
pprint(coeffi\_expr)  
print("echelon\_form:")  
pprint(coeffi\_expr.echelon\_form())  
print("reduced row echelon form:")  
pprint(coeffi\_expr.rref())  
  
# 解线性方程组  
print("解线性方程组（矩阵模式），sympy提供的几种方式：")  
pprint(solve(coeffi\_expr,(lambda\_1,lambda\_2,lambda\_3)))  
pprint(solve(coeffi\_expr,set=True))  
from sympy import Matrix, solve\_linear\_system  
pprint(solve\_linear\_system(V\_C\_matrix.T,lambda\_1,lambda\_2,))

V\_C\_matrix=  
⎡1 4 2 -3⎤  
⎢ ⎥  
⎢7 10 -4 -1⎥  
⎢ ⎥  
⎣-2 1 5 -4⎦  
--------------------------------------------------  
向量集合(线性方程组)系数矩阵：  
⎡ λ₁ + 7⋅λ₂ - 2⋅λ₃ ⎤  
⎢ ⎥  
⎢4⋅λ₁ + 10⋅λ₂ + λ₃ ⎥  
⎢ ⎥  
⎢2⋅λ₁ - 4⋅λ₂ + 5⋅λ₃⎥  
⎢ ⎥  
⎣-3⋅λ₁ - λ₂ - 4⋅λ₃ ⎦  
echelon\_form:  
⎡λ₁ + 7⋅λ₂ - 2⋅λ₃⎤  
⎢ ⎥  
⎢ 0 ⎥  
⎢ ⎥  
⎢ 0 ⎥  
⎢ ⎥  
⎣ 0 ⎦  
reduced row echelon form:  
⎛⎡1⎤ ⎞  
⎜⎢ ⎥ ⎟  
⎜⎢0⎥ ⎟  
⎜⎢ ⎥, (0,)⎟  
⎜⎢0⎥ ⎟  
⎜⎢ ⎥ ⎟  
⎝⎣0⎦ ⎠  
解线性方程组（矩阵模式），sympy提供的几种方式：  
⎧ -3⋅λ₃ λ₃⎫  
⎨λ₁: ──────, λ₂: ──⎬  
⎩ 2 2 ⎭  
⎛ ⎧⎛-3⋅λ₃ λ₃⎞⎫⎞  
⎜[λ₁, λ₂], ⎨⎜──────, ──⎟⎬⎟  
⎝ ⎩⎝ 2 2 ⎠⎭⎠  
{λ₁: 3/2, λ₂: -1/2}

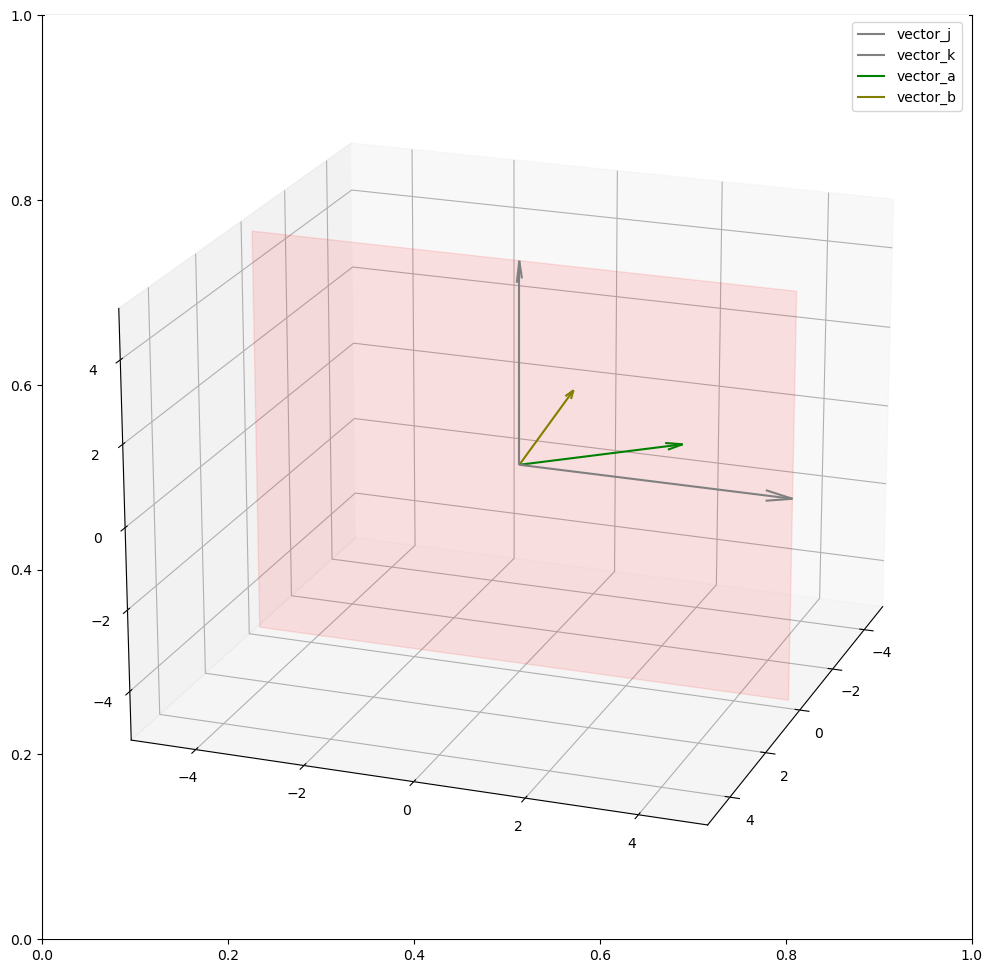
#### 3）维数

假设c为任意实数，若Rm​的子集W满足下述三个条件：1.  W的任意元素的c倍也是W的元素；2. W的任意元素的和也是W的元素；3. 零向量在W中。即满足这三个条件时，1. 如果∈W，那么​​​∈W；2. 如果并且，那么；同时，则把W叫做Rm​的线性子空间，简称子空间。在理解向量子空间时可以先从2、3维空间思考，如果乘以一个倍数，实际上只是对向量的缩放，而对于向量的和也只是延着多个向量的行进，这些计算都存在于一个空间维度中，进而可以帮助理解拓展到大于3个的维度（无法类似2，3维度直接观察），例如“通过原点的直线”，“通过原点的平面”等等描述。

W线性无关的元素，即基元素，，其中基元素的个数n叫做子空间W的维度，一般表示为dimW（dim为dimension缩写）。

例如W是R³的子空间，向量vector\_a=0*C.i+3*C.j+1*C.k，即，与向量vector\_b=0*C.i+1*C.j+2*C.k，即是W的线性无关（通过pprint(v\_a\_b.T.rref())查看，不含全零行）的元素。显然，{�1[031]+�2[012]�1,�2为任意实数}{​,c2​为任意实数}等式成立，因此集合是子空间W的基，子空间的维数为2。

from matplotlib.patches import PathPatch,Rectangle  
import mpl\_toolkits.mplot3d.art3d as art3d  
  
fig, ax=plt.subplots(figsize=(12,12))  
ax=fig.add\_subplot( projection='3d')  
p=Rectangle((-5, -5), 10,10,color='red', alpha=0.1)  
ax.add\_patch(p)  
art3d.pathpatch\_2d\_to\_3d(p, z=0, zdir="x",)  
  
C=CoordSys3D('C')  
vector\_0=Vector.zero  
vector\_1=5\*C.j  
vector\_plot\_3d(ax,C,vector\_0,vector\_1,color='gray',label='vector\_j',arrow\_length\_ratio=0.1)  
vector\_2=5\*C.k  
vector\_plot\_3d(ax,C,vector\_0,vector\_2,color='gray',label='vector\_k',arrow\_length\_ratio=0.1)  
  
vector\_a=0\*C.i+3\*C.j+1\*C.k  
vector\_b=0\*C.i+1\*C.j+2\*C.k  
vector\_plot\_3d(ax,C,vector\_0,vector\_a,color='green',label='vector\_a',arrow\_length\_ratio=0.1)  
vector\_plot\_3d(ax,C,vector\_0,vector\_b,color='olive',label='vector\_b',arrow\_length\_ratio=0.1)  
  
ax.set\_xlim3d(-5,5)  
ax.set\_ylim3d(-5,5)  
ax.set\_zlim3d(-5,5)  
  
ax.legend()  
ax.view\_init(20,20) # 可以旋转图形的角度，方便观察  
plt.show()  
  
v\_a\_b=vector\_a.to\_matrix(C).col\_insert(-1,vector\_b.to\_matrix(C))  
print("v\_a\_b.T rref:")  
pprint(v\_a\_b.T.rref())

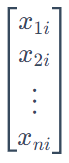
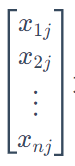


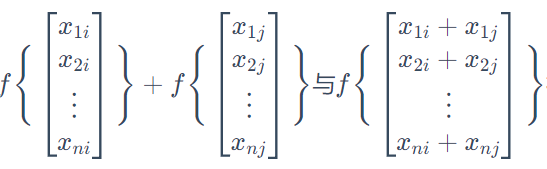
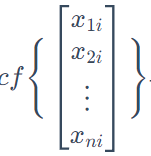
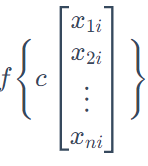
​

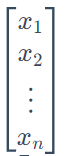
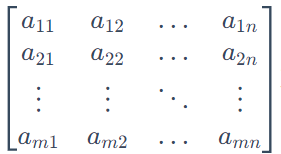
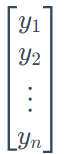
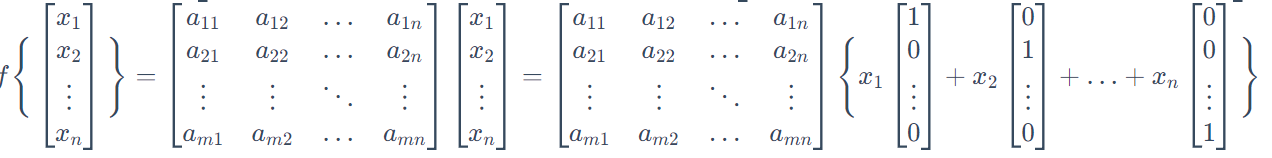
v\_a\_b.T rref:  
⎛⎡0 1 0⎤ ⎞  
⎜⎢ ⎥, (1, 2)⎟  
⎝⎣0 0 1⎦ ⎠

### 2.2.1.3 线性映射

#### 1）定义，线性映射的矩阵计算方法，像

假设和为Rn的任意元素， f为从到的映射。当映射f满足以下两个条件时，则称映射f是从到线性映射。

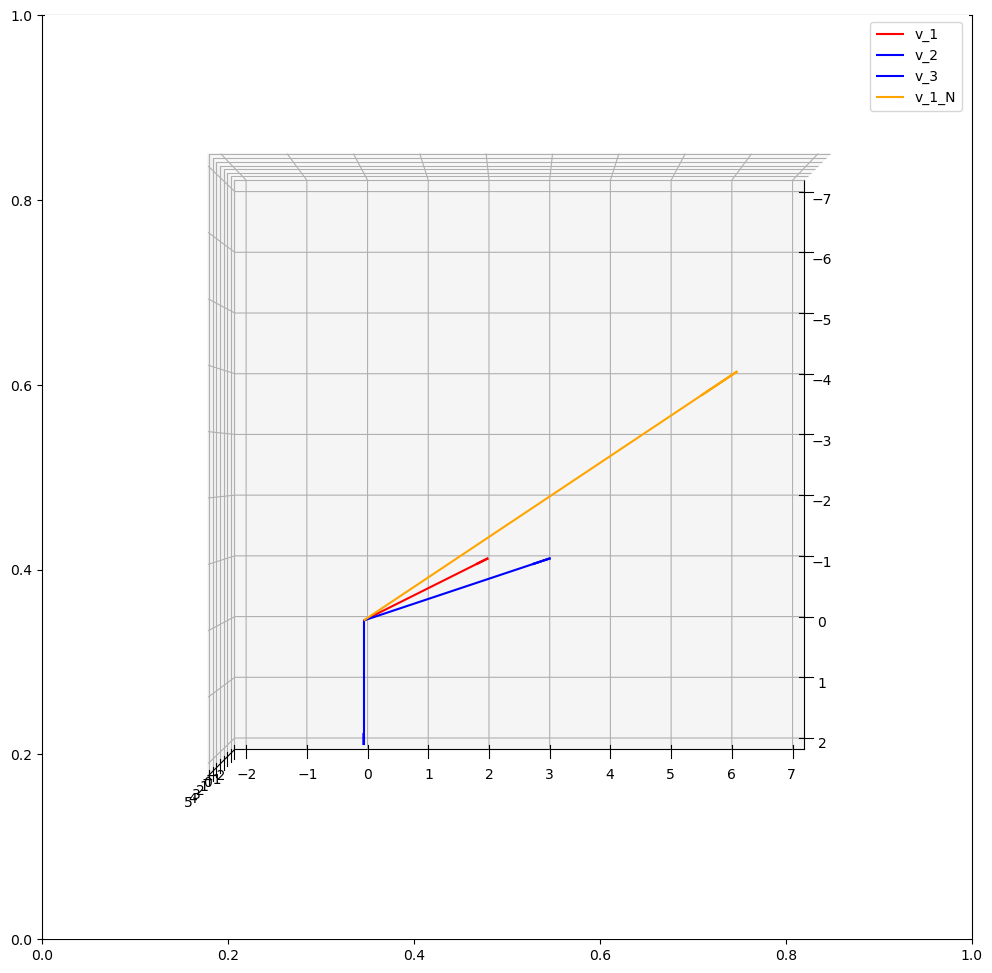
1. 相等；
2. 与相等。

从Rn到Rm线性映射，可以被称为线性变换或一次变换。为了方便理解线性变换，可以将映射f理解为一个函数（变换矩阵），输入一个向量，经过f作用后，而后输出一个向量的过程，即向量发生了运动。就是，当（输入）通过m×n矩阵对应的从到的线性映射，形成的像是（输出），其矩阵计算公式表示为：。对像而言，即假设xi​是集合X的元素，把通过映射f与xi​对应的集合Y的元素叫做xi​通过映射f形成的像。

定义三维的C向量空间，单位向量为，定义v\_1向量的系数为a1,a2,a3=-1,2,0，其中0可以省略，即为v\_1=a1\*i+a2\*j=-1\*i+2\*j向量。给定新的向量集合v\_2，v\_3，通过计算简化的行阶梯形矩阵，判断为向量无关，因此将该数据集作为基，可以建立新的向量空间，并由v\_2，v\_3的基构建变换矩阵，v\_1的系数矩阵乘以变换矩阵计算得新向量空间下的v\_1在相对于C（原）向量空间下的向量v\_1\_N。

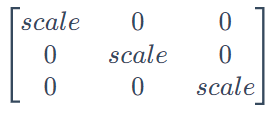
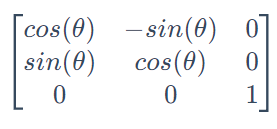
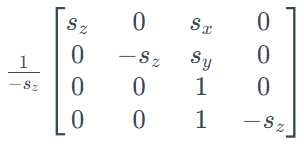
fig, ax=plt.subplots(figsize=(12,12))  
ax=fig.add\_subplot( projection='3d')  
  
a1,a2=sympy.symbols(["a1","a2",])  
a1=-1  
a2=2  
  
C=CoordSys3D('C')  
i, j, k = C.base\_vectors()  
v\_0=Vector.zero  
v\_1=a1\*i+a2\*j  
vector\_plot\_3d(ax,C,v\_0,v\_1,color='red',label='v\_1',arrow\_length\_ratio=0.1)  
  
v\_2=-1\*i+3\*j  
vector\_plot\_3d(ax,C,v\_0,v\_2,color='blue',label='v\_2',arrow\_length\_ratio=0.1)  
v\_3=2\*i+0\*j  
vector\_plot\_3d(ax,C,v\_0,v\_3,color='blue',label='v\_3',arrow\_length\_ratio=0.1)  
  
def vector2matrix\_rref(v\_list,C):  
 '''  
 function - 将向量集合转换为向量矩阵，并计算简化的行阶梯形矩阵  
   
 Params:  
 v\_list - 向量列表；list(vector)  
 C - sympy定义的坐标系统;CoordSys3D()  
  
 Returns:  
 v\_matrix.T - 转换后的向量矩阵,即线性变换矩阵   
 '''   
 v\_matrix=v\_list[0].to\_matrix(C)  
 for v in v\_list[1:]:  
 v\_temp=v.to\_matrix(C)  
 v\_matrix=v\_matrix.col\_insert(-1,v\_temp)  
   
 print("\_"\*50)  
 pprint(v\_matrix.T.rref())  
 return v\_matrix.T  
  
print("v\_2,v\_3简化的行阶梯形矩阵:")  
v\_2\_3=vector2matrix\_rref([v\_2,v\_3],C) # 通过rref()返回简化的行阶梯形矩阵，判断向量集合v\_2,v\_3为向量无关。返回线性变换矩阵  
  
# 由向量无关的向量集合v\_2,v\_3生成新的向量空间，相对于原向量空间下i,j,k单位向量，i,j,k新的位置  
v\_1\_N\_matrix=Matrix([a1,a2]).T\*v\_2\_3 # 根据变换矩阵，计算系数同为a1,a2向量的位置  
from sympy.vector import matrix\_to\_vector  
v\_1\_N=matrix\_to\_vector(v\_1\_N\_matrix,C) # 向量转矩阵  
vector\_plot\_3d(ax,C,v\_0,v\_1\_N,color='orange',label='v\_1\_N',arrow\_length\_ratio=0.1) # 绘制新位置向量  
  
ax.set\_xlim3d(-7,2)  
ax.set\_ylim3d(-2,7)  
ax.set\_zlim3d(-2,5)  
  
ax.legend()  
ax.view\_init(90,0) # 可以旋转图形的角度，方便观察  
plt.show()

v\_2,v\_3简化的行阶梯形矩阵:  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
⎛⎡1 0 0⎤ ⎞  
⎜⎢ ⎥, (0, 1)⎟  
⎝⎣0 1 0⎦ ⎠



#### 2）特殊的线性映射（线性变换）

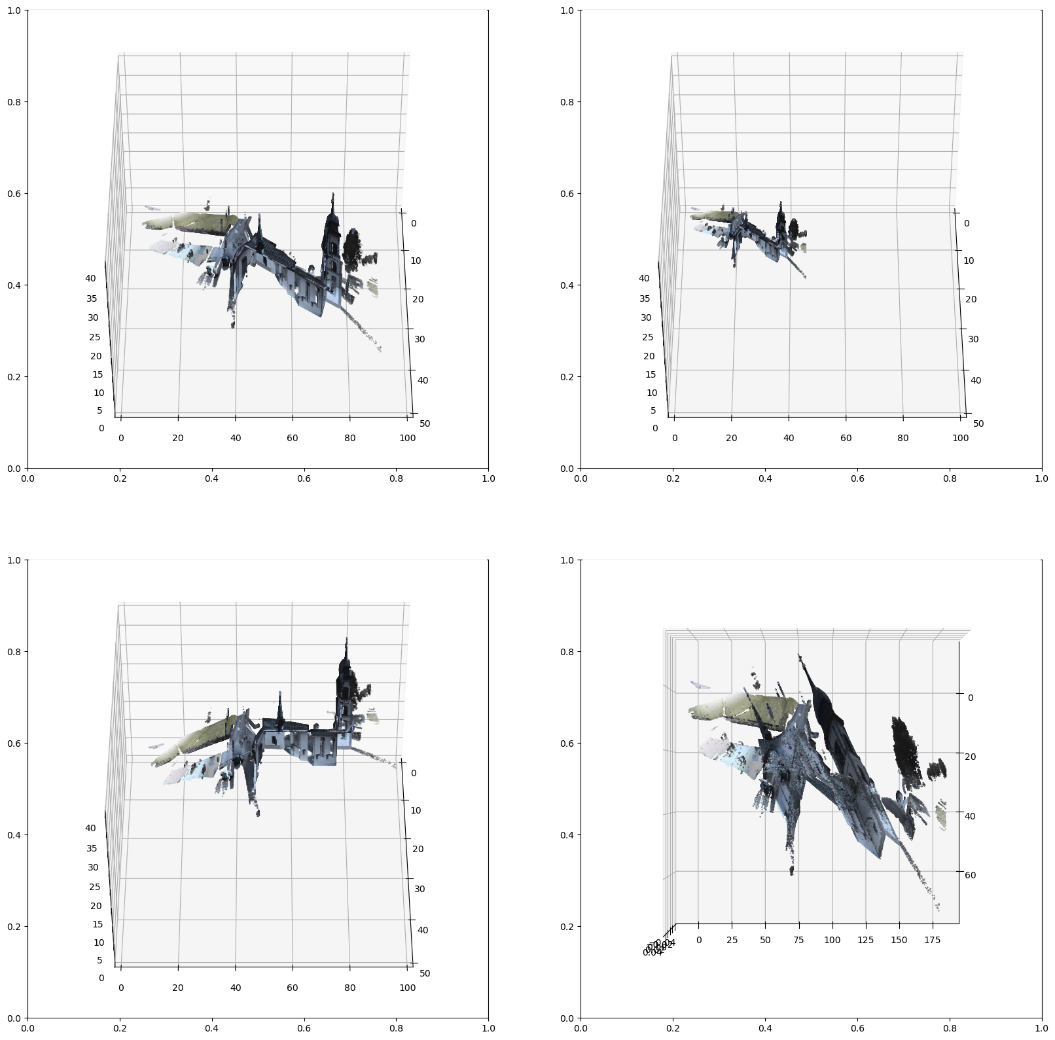
线性映射的矩阵计算方法就是将待变换的对象（向量集合），通过乘以变换矩阵，就可获得同一向量空间下新的对象。使用SymPy方便数据量比较小的矩阵运算，方便观察矩阵形式。但是如果数据量大，则使用NumPy库提供的矩阵计算方法。在下述案例中，从[semantic3d](http://semantic3d.net/)④下载城市点云数据“bildstein\_station1”，并对其进行了降采样，以加快计算速度。对于点云数据的处理可参考点云部分。

点云数据读取后，包括的信息有点坐标和点颜色。点坐标可以理解为在单位向量为的C向量空间下的向量，如果提供相应的变换矩阵，就可以对原点云数据加以空间上的变换，例如移动、旋转、缩放、透视、镜像、切变等。缩放对应的矩阵为：​；沿z轴旋转对应的矩阵为：​​；给定点S(x,y,z)的透视为：，透视变换中使用了齐次坐标（homogeneous coordinates）。

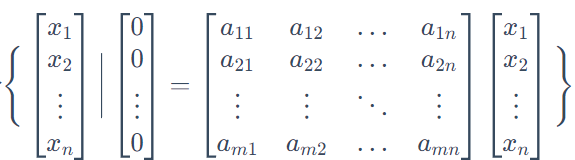
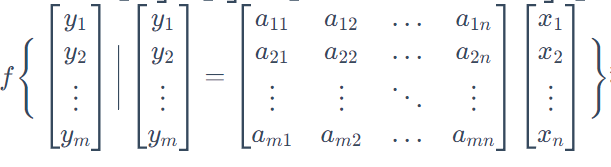
import open3d as o3d  
import numpy as np  
import pandas as pd  
import matplotlib  
  
cloud\_pts\_fp="./data/bildstein\_station1.txt"  
cloud\_pts=pd.read\_csv(cloud\_pts\_fp)  
#cloud\_pts['hex']=cloud\_pts.apply(lambda row:'%02x%02x%02x' % (int(row.r/255),int(row.g/255),int(row.b/255)),axis=1)  
print("data reading completed")  
cloud\_pts['hex']=cloud\_pts.apply(lambda row:matplotlib.colors.to\_hex([row.r/255,row.g/255,row.b/255]),axis=1) # 将色彩rgb格式转换为hex格式，用于matplotlib图形打印

data reading completed

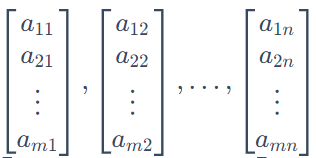
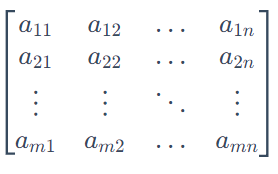
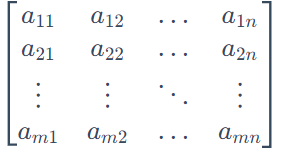
import matplotlib.pyplot as plt  
import matplotlib  
import sympy,math  
from sympy import Matrix,pprint  
import numpy as np  
  
pts=cloud\_pts[["x","y","z"]].to\_numpy()  
  
fig, axs=plt.subplots(ncols=2, nrows=2,figsize=(20,20))  
axs[0,0]=fig.add\_subplot(2,2,1, projection='3d')  
axs[0,1]=fig.add\_subplot(2,2,2, projection='3d')  
axs[1,0]=fig.add\_subplot(2,2,3, projection='3d')  
axs[1,1]=fig.add\_subplot(2,2,4, projection='3d')  
  
axs[0,0].scatter(cloud\_pts.x,cloud\_pts.y,cloud\_pts.z,c=cloud\_pts.hex,s=0.1)  
# A - 缩放  
scale=0.5  
f\_scale=np.array([[scale,0,0],[0,scale,0],[0,0,scale]])  
pts\_scale=np.matmul(pts,f\_scale)  
axs[0,1].scatter(pts\_scale[:,0],pts\_scale[:,1],pts\_scale[:,2],c=cloud\_pts.hex,s=0.1)  
  
# B - 旋转  
angle=math.radians(-20) # 度转换为弧度  
f\_rotate=np.array([[math.cos(angle),-math.sin(angle),0],[math.sin(angle),math.cos(angle),0],[0,0,1]])  
pts\_rotate=np.matmul(pts,f\_rotate)  
axs[1,0].scatter(pts\_rotate[:,0],pts\_rotate[:,1],pts\_rotate[:,2],c=cloud\_pts.hex,s=0.1)  
  
# C - 透视  
s1,s2,s3=2,2,2  
f\_persp=np.array([[-s3,0,0,0],[0,-s3,0,0],[s1,s2,0,1],[0,0,0,-s3]])\*1/(1-s3)  
pts\_homogeneousCoordinates=np.hstack((pts,np.ones(pts.shape[0]).reshape(-1,1)))  
pts\_persp=np.matmul(pts\_homogeneousCoordinates,f\_persp)  
axs[1,1].scatter(pts\_persp[:,0],pts\_persp[:,1],pts\_persp[:,2],c=cloud\_pts.hex,s=0.1)  
  
axs[0,0].set\_xlim3d(0,50)  
axs[0,0].set\_ylim3d(0,100)  
axs[0,0].set\_zlim3d(0,40)  
  
axs[0,1].set\_xlim3d(0,50)  
axs[0,1].set\_ylim3d(0,100)  
axs[0,1].set\_zlim3d(0,40)  
  
axs[1,0].set\_xlim3d(0,50)  
axs[1,0].set\_ylim3d(0,100)  
axs[1,0].set\_zlim3d(0,40)  
  
axs[0,0].view\_init(45,0) # 可以旋转图形的角度，方便观察  
axs[0,1].view\_init(45,0)  
axs[1,0].view\_init(45,0)  
axs[1,1].view\_init(90,0)  
plt.show()



#### 3）秩

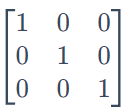
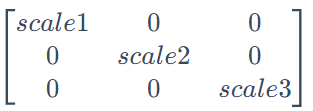
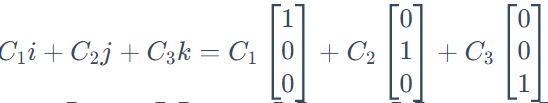
f为从到的映射，被映射到零元素的全体元素的集合，即集合称为映射f的核，一般表示为Ker​f。为了与“映射f的核”相呼应，则把“映射f的值域”，即集合称为“映射f的像空间”，一般表示为为Im​f。

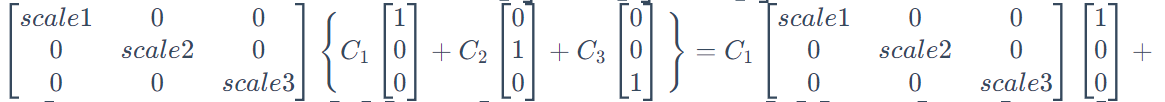
Ker​f是的子空间， Im​f是的子空间。在Ker​f与Im​f之间，有“维数公式”： 。

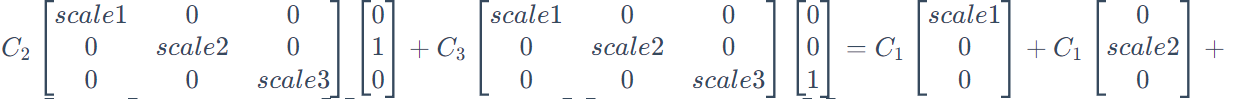
把向量中线性无关的向量的个数，即子空间Im​f的维数称为m×n矩阵​​​的秩，一般表示为，或r(A),rank(A),rk(A)。

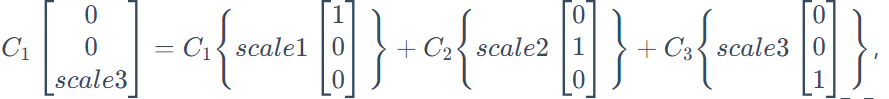
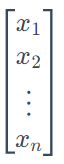
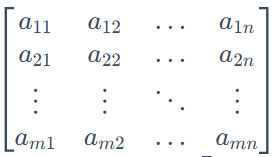
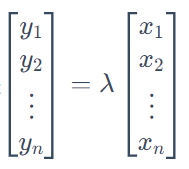
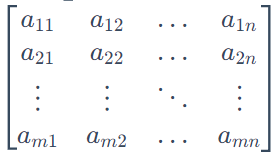
SymPy计算秩（rank）的方法为matrix.rank()。

### 2.2.1.4 特征值和特征向量

在特殊的线性映射部分，假设给定单位向量为，即​的C向量空间，通过变换矩阵实现了对原向量集（点云数据）的缩放。为方便解释，如果仅考虑点云数据中的一个点， c1​,c2​,c3​=x,y,z=39,73,22，则该点在C向量空间的向量为。进行线性映射和公式变换，





，变换后的公式保持了C向量空间下的单位向量不变，则scale1,scale2,scale3为变换矩阵的特征值，而scale1对应的，scale2对应的，scale3对应的，为其特征向量。当（输入）通过m×n矩阵对应的从到的线性映射f，形成的像是（输出），则把lambda叫做方阵的特征值（eigenvalue），把叫做与特征值λ对应的特征向量（eigenvector）。此外，零向量不能解释为特征向量。

SymPy库提供了matrix.eigenvals()，matrix.eigenvects()方法直接计算特征值和特征向量。

scale1,scale2,scale3=sympy.symbols(['scale1','scale2','scale3'])  
M0=Matrix(3, 3, [1, 0, 0, 0, 1, 0, 0, 0, 1])  
print("original matrix")  
pprint(M0)  
M=Matrix(3, 3, [scale1\*1, 0, 0, 0, scale2\*1, 0, 0, 0, scale3\*1])  
print("modified scaled matrix")  
pprint(M)  
print("eigen values:")  
pprint(M.eigenvals())  
print("eigen vectors:")  
pprint(M.eigenvects())

original matrix  
⎡1 0 0⎤  
⎢ ⎥  
⎢0 1 0⎥  
⎢ ⎥  
⎣0 0 1⎦  
modified scaled matrix  
⎡scale₁ 0 0 ⎤  
⎢ ⎥  
⎢ 0 scale₂ 0 ⎥  
⎢ ⎥  
⎣ 0 0 scale₃⎦  
eigen values:  
{scale₁: 1, scale₂: 1, scale₃: 1}  
eigen vectors:  
⎡⎛ ⎡⎡1⎤⎤⎞ ⎛ ⎡⎡0⎤⎤⎞ ⎛ ⎡⎡0⎤⎤⎞⎤  
⎢⎜ ⎢⎢ ⎥⎥⎟ ⎜ ⎢⎢ ⎥⎥⎟ ⎜ ⎢⎢ ⎥⎥⎟⎥  
⎢⎜scale₁, 1, ⎢⎢0⎥⎥⎟, ⎜scale₂, 1, ⎢⎢1⎥⎥⎟, ⎜scale₃, 1, ⎢⎢0⎥⎥⎟⎥  
⎢⎜ ⎢⎢ ⎥⎥⎟ ⎜ ⎢⎢ ⎥⎥⎟ ⎜ ⎢⎢ ⎥⎥⎟⎥  
⎣⎝ ⎣⎣0⎦⎦⎠ ⎝ ⎣⎣0⎦⎦⎠ ⎝ ⎣⎣1⎦⎦⎠⎦

注释（Notes）：

① SymPy，一个用于符号数学的Python库，目的是构建一个全功能的计算机代数系数（computer algebra system，CAS），同时保持代码尽可能的简单，便于理解和扩展（<https://www.sympy.org/>）。

② Vector，为SymPy库模块，该模块提供了与三维直角坐标系有关的基本矢量数学和微分计算的工具（<https://docs.sympy.org/dev/modules/vector/index.html>）。

③ Matrices（linear algebra），为线性代数模块，并使设计的代码结构尽可能简单（<https://docs.sympy.org/latest/modules/matrices/matrices.html>）。

④ semantic3d，为大规模点云分类基准数据（<http://semantic3d.net/>）。

参考文献（References）:

[1] [日]高桥 信著作,Inoue Iroha,株式会社 TREND-PRO漫画制作,腾永红译.漫画线性代数[M].科学出版社.北京.2009.08.

[2] Gilbert Strang. Introduction to linear algbra[M].Wellesley: Cambridge Press. June 10, 2016(Fifth Edition edition).