代码内容审核建议 6（2.2.1）

1. 关于打印的建议

之前提到过打印格式的建议，此处着重于对输入数据的打印

首先将原始的输入数据打印出来，然后打印运算后的结果，这样，更加清晰和让读者理解

此处，可以打印各种运算结果之前，先打印原始矩阵，再打印处理后的矩阵，

```
因此可在 print("矩阵加法：")之前添加
print("A 矩阵=")
pprint(A)
print("_"*50)
print("B 矩阵=")
pprint(B)
print("_"*50)
```

且之后的 print 内容，可统一改为以下类似的样式

```
print("矩阵加法：A+B=")
print("矩阵数乘：2*A=")
```

```
import sympy
from sympy import Matrix,pprint

A=Matrix([[1,3,1],[1,0,0]])
B=Matrix([[0,0,5],[7,5,0]])
print("矩阵加法：")
pprint(A+B)
print("_"*50)

print("数乘：")
pprint(2*A)
print("_"*50)

C=Matrix([[1,0,2],[-1,3,1]])
D=Matrix([[3,1],[2,1],[1,0]])
print("乘法：")
pprint(C*D)
print("_"*50)

E=Matrix([[1,2],[3,4]])
print("幂：")
pprint(E**3)
print("_"*50)
```

```
矩阵加法：
⎡1  3  6⎤
⎢       ⎥
⎣8  5  0⎦
```

```
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  0  1⎦
判断方阵是否有逆矩阵：
-3123
-1/3123
```

2. 转置矩阵的记法有两个，可以添加一下（A^T和A'）

2）特殊矩阵

1. 零矩阵

所有元素均为0的矩阵，例如，$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

2. 转置矩阵（transpose）

是指将$m \times n$矩阵$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$，交换行列后得到的$n \times m$，$\begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{m,n} \end{bmatrix}$，可以用$\boxed{A^T}$表示，例如$3 \times 2$的矩阵$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$的转置矩阵为$2 \times 3$矩阵$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$。

3. 格式的统一

A．绿色长方形处，可添加换行，增加形式美感

B．蓝色框中的 A 和红色方框中的 det()应该为公式格式

如$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 5 & 2 & 0 & 0 \\ 6 & 8 & 3 & 0 \\ 7 & 9 & 10 & 4 \end{bmatrix}$,对角元素右上角的所有元素均为0的$n$阶矩阵。

6. 对角矩阵 (diagonal matrix)

如$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$，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：$\begin{bmatrix} a_{1,1} & 0 & 0 & 0 \\ 0 & a_{2,2} & 0 & 0 \\ 0 & 0 & a_{3,3} & 0 \\ 0 & 0 & 0 & a_{n,n} \end{bmatrix}^p = \begin{bmatrix} a_{1,1}^p & 0 & 0 & 0 \\ 0 & a_{2,2}^p & 0 & 0 \\ 0 & 0 & a_{3,3}^p & 0 \\ 0 & 0 & 0 & a_{n,n}^p \end{bmatrix}$，例如$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^2 = \begin{bmatrix} 2^2 & 0 \\ 0 & 3^2 \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 0 & 9 \end{bmatrix}$

7. 单位矩阵（identity matrix）

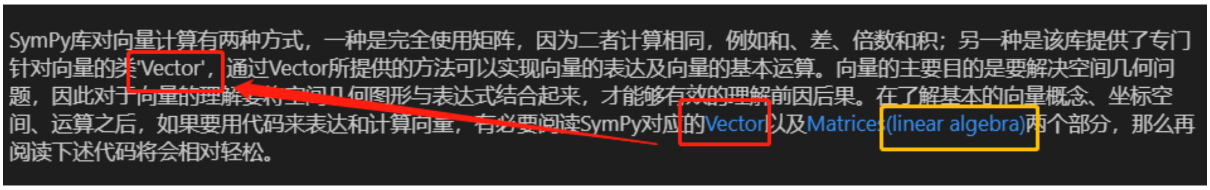
如$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$，对角元素均为1，对角元素以外的其它元素全部为0的n阶方阵，即$\boxed{diag(1,1,\ldots,1)}$。单位矩阵与任何矩阵相乘，都对这个矩阵没有任何影响。

8. 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$AB = BA = I_n$，其中I_n为n阶单位矩阵，则称A是可逆的，且B是A的逆矩阵，记作A^{-1}。只有方阵$n \times n$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。

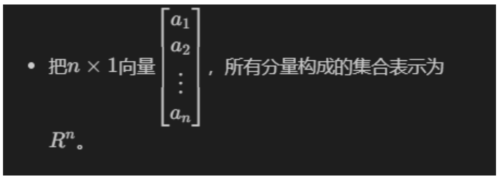
并不是所有的方阵都有逆矩阵，可以使用行列式指标determinant(det)，用sympy库的det()方法计算判断。如果$\boxed{det(A) \neq 0}$，则矩阵A可逆。

4. 对于超链接添加，最好对第一次出现的地方添加（Vector）。关于 Matrices 的超链接，可以隐去”(linear algebra)”

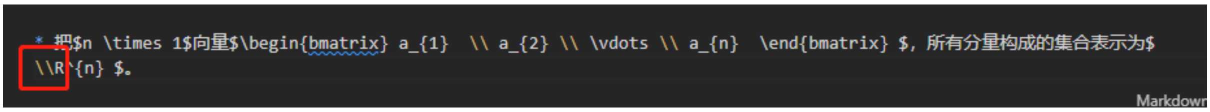


SymPy库对向量计算有两种方式，一种是完全使用矩阵，因为二者计算相同，例如和、差、倍数和积；另一种是该库提供了专门针对向量的类'Vector'，通过Vector所提供的方法可以实现向量的表达及向量的基本运算。向量的主要目的是要解决空间几何问题，因此对于向量的理解要将空间几何图形与表达式结合起来，才能够有效的理解前因后果。在了解基本的向量概念、坐标空间、运算之后，如果要用代码来表达和计算向量，有必要阅读SymPy对应的Vector以及Matrices(linear algebra)两个部分，那么再阅读下述代码将会相对轻松。

5. 出处 markdown 公式有误，应该删除”\\”，且应该统一为**上标格式【还有多处，不再一一列举】**



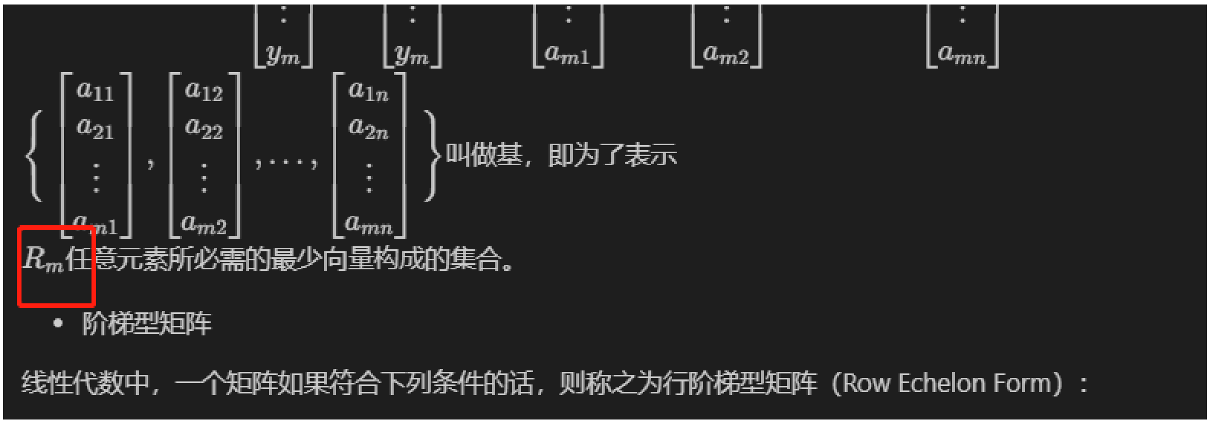
- 把$n \times 1$向量$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$，所有分量构成的集合表示为 R^n。



```
* 把$n \times 1$向量$\begin{bmatrix} a_{1}  \\ a_{2} \\ \vdots \\ a_{n}  \end{bmatrix} $，所有分量构成的集合表示为$
\\R^{n} $。
```

Markdown

相同的错误，在本章节靠下的部分也同样存在【*是否是 markdown 解释器的原因？*】

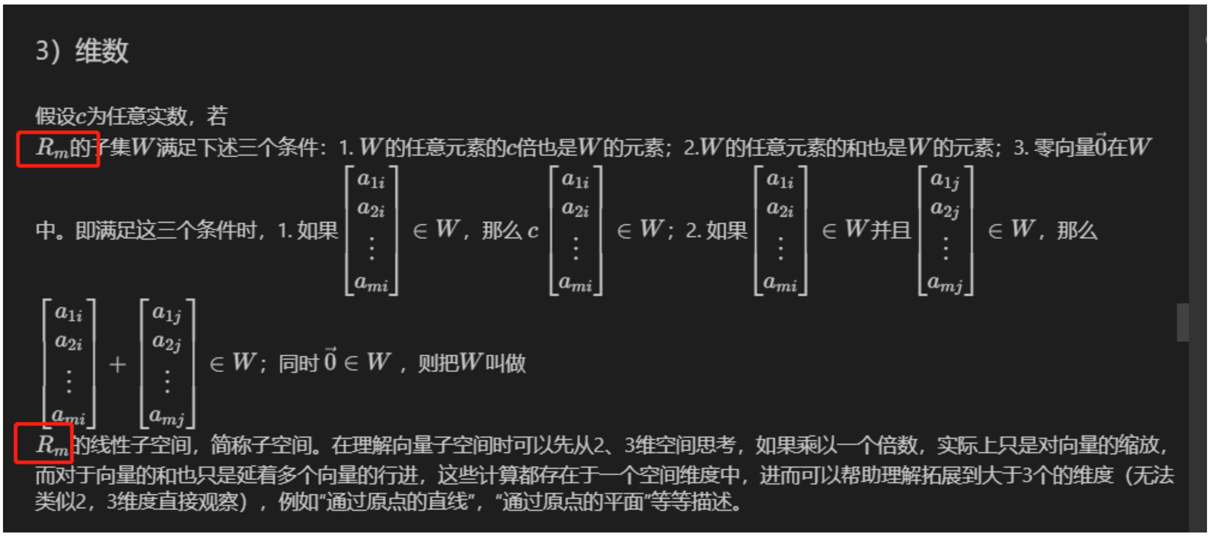


$\left\{ \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix}, \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix}, \ldots, \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} \right\}$叫做基，即为了表示

R_m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）：



3）维数

假设c为任意实数，若

R_m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$\vec{0}$在W中。即满足这三个条件时，1. 如果$\begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{mi} \end{bmatrix} \in W$，那么$c\begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{mi} \end{bmatrix} \in W$；2. 如果$\begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{mi} \end{bmatrix} \in W$并且$\begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix} \in W$，那么$\begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{mi} \end{bmatrix} + \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix} \in W$；同时$\vec{0} \in W$，则把$W$叫做

R_m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。

例如W是

R^3的子空间，向量vector_a=0C.i+3C.j+1C.k，即$\begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix}$，与向量vector_b=0C.i+1C.j+2C.k，即$\begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$是$W$的线性无关（通过

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 $\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix}$ 和 $\begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{bmatrix}$ 为
R^n的任意元素，f为从
R^n到
R^m的映射。当映射f满足以下两个条件时，则称映射f是从
R^n到
R^m线性映射。1. $f\left\{\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix}\right\}+f\left\{\begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{bmatrix}\right\}$与$f\left\{\begin{bmatrix} x_{1i}+x_{1j} \\ x_{2i}+x_{2j} \\ \vdots \\ x_{ni}+x_{nj} \end{bmatrix}\right\}$相等；2. $cf\left\{\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix}\right\}$与$f\left\{c\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix}\right\}$相等。从
R^n到
R^m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

6. 代码块顺序问题：调换上方红色框内容和蓝色框内对函数的定义位置

Import python 库后，优先对函数定义，随后再定义变量，然后书写逻辑和功能代码，ax 是和绘图有关的绘图句柄

```python
from sympy.vector import Vector

fig, ax=plt.subplots(figsize=(12,12))
ax=fig.add_subplot( projection='3d')

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

    Paras:
        ax_3d - matplotlib的3d格式子图
        C - /coordinate_system - SymPy下定义的坐标系
        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
        vector - 所要打印的向量
        color - 向量色彩
        label - 向量标签
        arrow_length_ratio - 向量箭头大小
    '''
    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)

#定义坐标系统，以及打印向量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)
```

具体代码修改见下页

包含了图片尺寸的修改（figsize=(12,12)→ figsize=(6, 6)），对基底位置的修改和投影的添加,以及图视角的修改

源代码

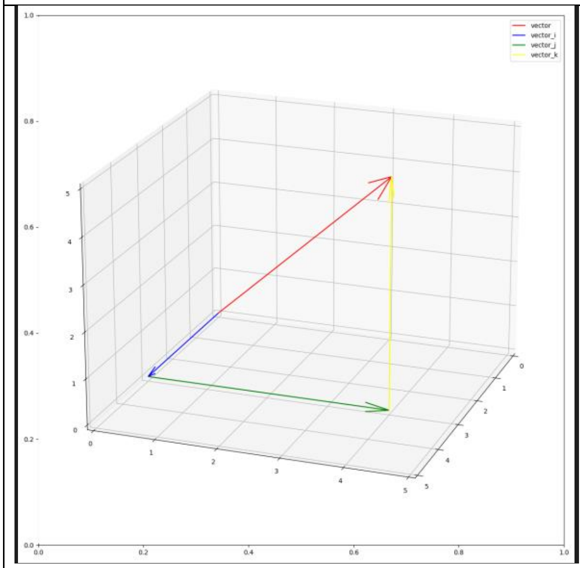
```python
import matplotlib.pyplot as plt
from sympy.vector.coordsysrect import CoordSys3D
from sympy.vector.vector import Vector, BaseVector
from sympy.vector import Vector

fig, ax=plt.subplots(figsize=(12,12))
ax=fig.add_subplot( projection='3d')

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

    Paras:
        ax_3d - matplotlib 的 3d 格式子图
        C - /coordinate_system - SymPy 下定义的坐标系
        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 - 所要打印的向量
        color - 向量色彩
        label - 向量标签
        arrow_length_ratio - 向量箭头大小
    '''
    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)

#定义坐标系统，以及打印向量 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)
```

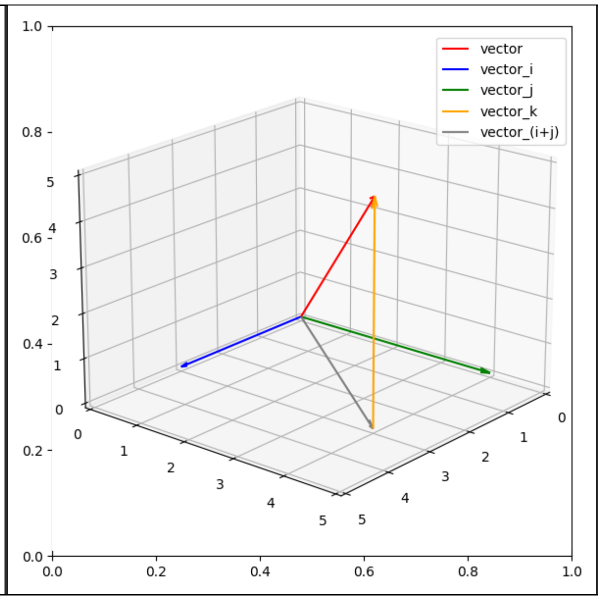
修改后

```python
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

    Paras:
        ax_3d - matplotlib 的 3d 格式子图
        C - /coordinate_system - SymPy 下定义的坐标系
        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 - 所要打印的向量
        color - 向量色彩
        label - 向量标签
        arrow_length_ratio - 向量箭头大小
    '''
    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=(6, 6))
ax = fig.add_subplot(projection='3d')
arrow_length_ratio = 0.05
# 定义坐标系统，以及打印向量 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=arrow_length_ratio)

# 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=arrow_length_ratio)

v1_j = v1.coeff(j) * j
vector_plot_3d(ax, C, v1_origin, v1_j, color='g', label='vector_j', arrow_length_ratio=arrow_length_ratio)

v1_k = v1.coeff(k) * k
vector_plot_3d(ax, C, v1_i+v1_j, v1_k, color='orange', label='vector_k', arrow_length_ratio=arrow_length_ratio)

vector_plot_3d(ax, C, v1_origin, v1_i+v1_j,color='grey', label='vector_(i+j)',
```

```
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()
```

```
arrow_length_ratio=arrow_length_ratio)

ax.set_xlim3d(0, 5)
ax.set_ylim3d(0, 5)
ax.set_zlim3d(0, 5)

ax.legend()
ax.view_init(20, 40)  # 可以旋转图形的角度，方便观察
plt.show()
```





7. 错别字，表达的修改

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 dependnet）（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$a_1, a_2, \ldots, a_n$，使得$a_1v_1 + a_2v_2 + \ldots + a_nv_n = 0$，或建立的表示为，$\sum_{i=1}^{n} a_iv_i = 0$，其中$v_1, v_2, \ldots, v_n$是$V$的向量，称它们为线性相关，其中右边的0，为$\vec{0}$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$v_1, v_2, \ldots, v_n$线性无关。对线性无关可以给出更直接的定义，向量$v_1, v_2, \ldots, v_n$线性无关，若且唯若它们满足以下条件：如果$a_1, a_2, \ldots, a_n$是K的元素，适合：$a_1v_1 + a_2v_2 + \ldots + a_nv_n = 0$，那么对所有$i = 1, 2, \ldots, n$都有$a_i = 0$。

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。

第一个红框，dependent 单词拼写错误
第二个黄框，对于类似的定义，“或建立的表示为”可以改成“也可以表示为”
第三个绿框，“若且唯若”建议换成“当且仅当”

8. 字符的表示方法，需要统一

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$\vec{i},\vec{j},\vec{k}$），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$a_1v_1+a_2v_2+\ldots+a_nv_n=0$，上述的a,b,c,d即为$a_1,a_2,\ldots,a_n$。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）。

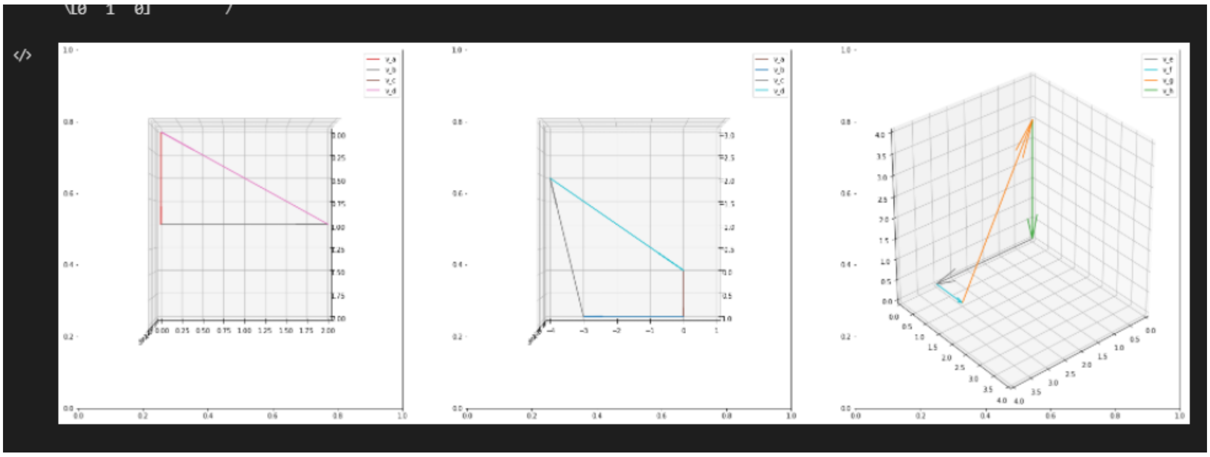
如果此处的举例对应代码中的话，那么也应该采用代码格式的表示

```
#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],)
```

9. 错别字：首相系数—>首项系数

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：每个首相系数是1，且是其所在列的唯一的非零元素，例如，$\begin{bmatrix}1 & 0 & a_1 & 0 & \mid & b_1\\ 0 & 1 & 0 & 0 & \mid & b_2\\ 0 & 0 & 0 & 1 & \mid & b_3\end{bmatrix}$，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。

10. 可视化图片角度选取不易理解，建议参考**修改条目 6** 修改



11. 排版格式的建议

“1.”和“2.” 之前可以添加换行

假设 $\begin{bmatrix} x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix}$ 和 $\begin{bmatrix} x_{2j} \\ \vdots \\ x_{nj} \end{bmatrix}$ 为

R^n的任意元素，f为从

R^n到

R^m的映射。当映射f满足以下两个条件时，则称映射f是从

R^n到

R^m线性映射。1. $f\left\{\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix}\right\}+f\left\{\begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{bmatrix}\right\}$与$f\left\{\begin{bmatrix} x_{1i}+x_{1j} \\ x_{2i}+x_{2j} \\ \vdots \\ x_{ni}+x_{nj} \end{bmatrix}\right\}$相等；2. $cf\left\{\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix}\right\}$与$f\left\{c\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix}\right\}$相等。从

R^n到

R^m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 $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ （输入）通过$m \times n$矩阵

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

12. 正文中涉及代码里的变量，应该采用代码格式的显示

。对像而言，即假设x_i是集合X的元素，把通过映射f与x_i对应的集合Y的元素叫做x_i通过映射f形成的像。

定义三维的C向量空间，单位向量为$\vec{i},\vec{j},\vec{k}$，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=a1i+a2j=-1i+2j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。

13. 建议修改描述方法

> 用sympy计算秩（rank）方法为matrix.rank()。

“用 sympy 计算秩（rank）方法为 matrix.rank()。”

可改为

Sympy 中计算秩（rank）的方法为 matrix.rank()。

14. 公式错误，应该是把英文的“\”写成了中文的“、”

R^m的线性映射f，形成的像是 $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ （输出），则把、$lambda$叫做方阵

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

15. Print 打印的变量，最好不要采用函数方法名作为输出

`print("eigenvals:")`和 `print("eigenvects:")`

可改为

`print("eigen values:")` 和 `print("eigen vectors:")`

同时，pprint(M)之前，可以添加未调整的矩阵和调整之后的提示信息

原代码	修改后
scale1,scale2,scale3=sympy.symbols(['scale1','scale2','scale3']) M=Matrix(3, 3, [scale1*1, 0, 0, 0, scale2*1, 0, 0, 0, scale3*1]) pprint(M) print("eigenvals:") pprint(M.eigenvals()) print("eigenvects:") pprint(M.eigenvects())	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())

打印内容展示如下

```
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⎦⎦⎠⎦
```

对比原来的打印信息如下

```
⎡scale₁    0       0   ⎤
⎢                      ⎥
⎢  0     scale₂    0   ⎥
⎢                      ⎥
⎣  0       0     scale₃⎦
eigenvals:
{scale₁: 1, scale₂: 1, scale₃: 1}
eigenvects:
⎡⎛           ⎡⎡1⎤⎤⎞  ⎛           ⎡⎡0⎤⎤⎞  ⎛           ⎡⎡0⎤⎤⎞⎤
⎢⎜           ⎢⎢ ⎥⎥⎟  ⎜           ⎢⎢ ⎥⎥⎟  ⎜           ⎢⎢ ⎥⎥⎟⎥
⎢⎜scale₁, 1, ⎢⎢0⎥⎥⎟, ⎜scale₂, 1, ⎢⎢1⎥⎥⎟, ⎜scale₃, 1, ⎢⎢0⎥⎥⎟⎥
⎢⎜           ⎢⎢ ⎥⎥⎟  ⎜           ⎢⎢ ⎥⎥⎟  ⎜           ⎢⎢ ⎥⎥⎟⎥
⎣⎝           ⎣⎣0⎦⎦⎠  ⎝           ⎣⎣0⎦⎦⎠  ⎝           ⎣⎣1⎦⎦⎠⎦
```