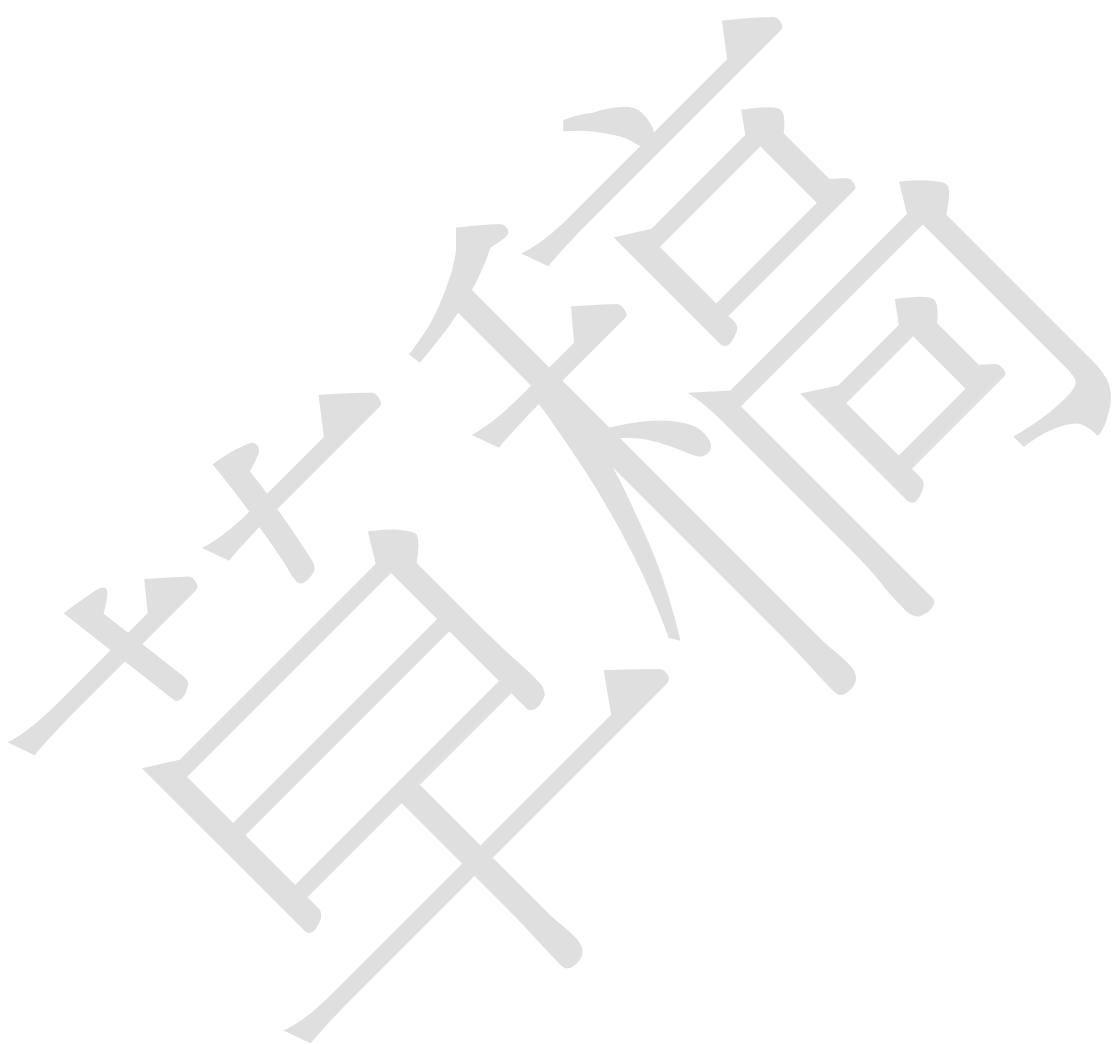


第一章:常用數學觀念介紹



1 前言

本

書嘗試用簡單之數學觀念：如向量、空間(3D)平面方程式，矩陣(matrix)，張量(tensor)及三元一次聯立方程式解法等最基本之數學工具，將複雜三度空間構造地質學問題，藉簡單之數學運算及公式推導，期能為讀者提供一套完整清晰之正確觀念。本篇所介紹者，僅先就向量空間及平面方程式為主，矩陣(matrix)及張量則將另闢專章介紹。內容雖然簡單易懂，但如能善用在構造地質學或岩坡工程上，好處甚多，很多艱澀深奧難懂之問題，多可迎刃而解，對於電腦程式之製作應該也會是助益良多。本書中比較常用之數學公式運算，儘量以 Python 或 Visual Basic 6 編製成電腦套裝軟體程式，或(和)Excel 試算表，或(和)Scratch 等電腦程式代勞，以減少徒手計算之人為謬誤，以符需要。另外也採用 Apsg(A Python package of structural geologists)或 Mplsteronet 程式來相互印證。

The book attempts to use simple mathematical concepts, such as vectors, space (3D) plane equations, matrix, tensor and system of linear equations in three unknown , and other basic mathematical tools to provide readers with a complete and clear set of correct concepts for complex three-dimensional space structural geology problems through simple mathematical operations and formulas. In this article, we will focus on vector spaces and plane equations, and we will introduce matrices and tensors in a separate chapter. Although the content is simple and easy to understand, if it can be used in structural geology or rock slope engineering, there are many benefits, and many difficult and esoteric problems can be easily solved, which should also be of great help to the production of computer programs. The more commonly used mathematical formula operations in this book should be compiled into computer software packages in Python or Visual Basic 6 as much as possible, or/or Excel spreadsheets, and/or computer programs such as Scratch to reduce the human fallacy of freehand calculations and meet the needs. In addition, Apsg (A Python package of structural geologists) or Mplsteronet programs are used to corroborate each other.

1.1 NED 座標系統

如(圖 1-1) 3 度空間問題有關之分析計算，可假定 x 為水平面(或赤道面)上指北(N)之座標，y 為水平面上指東(E)座標，z 則為垂直 x、y 平面之朝下(D)座標，且 x、y、z 互相垂直並成右手螺旋定則，此即為所謂之 NED 座標系統(圖 1-2)。傾角方向 (Dip direction)或稱傾向，即傾斜平面上物體沿斜坡自由滑動之方向，與走向 (Strike)垂直。 α 則以順時針方向由北起算為正，傾角 β 則為 x、y 水平面與任意直線或不連續面傾向向量(Dip vector)之夾角，夾角由水平面向下量測依順時針方向為正。則通過半徑長 1 單位之參考球體(Reference sphere, $x^2+y^2+z^2=1.0$)球心 0(0, 0, 0)之任意直線或不連續面的朝下單位位置向量在 x, y, z 三軸之分量，如以球體座標(α , β , α)表示(圖 1-3)為： $x=\cos \alpha \cos \beta$, $y=\sin \alpha \cos \beta$, $z=\sin \beta$

其單位位置向量(即平面之單位傾向向量，以下同)r，

$$\underline{r}=[x, y, z]=[\cos \alpha \cos \beta, \sin \alpha \cos \beta, \sin \beta]$$

而垂直於該不連續面或直線之單位向量 n 應為

$$\underline{n} = \pm [a, b, c] = \pm [\cos \alpha \sin \beta, \sin \alpha \sin \beta, -\cos(\beta)]$$

因 $\underline{n} \cdot \underline{r}=0$ (向量內積為方編書寫有時會以 $\text{dot}(n, r)$ 表示)，故 $ax+by+cz=0$ ，此表示參考球體球心之不連續面平面方程式。n 朝上取正號(即 c 為負值)；朝下取負號(即 c 為正值)。

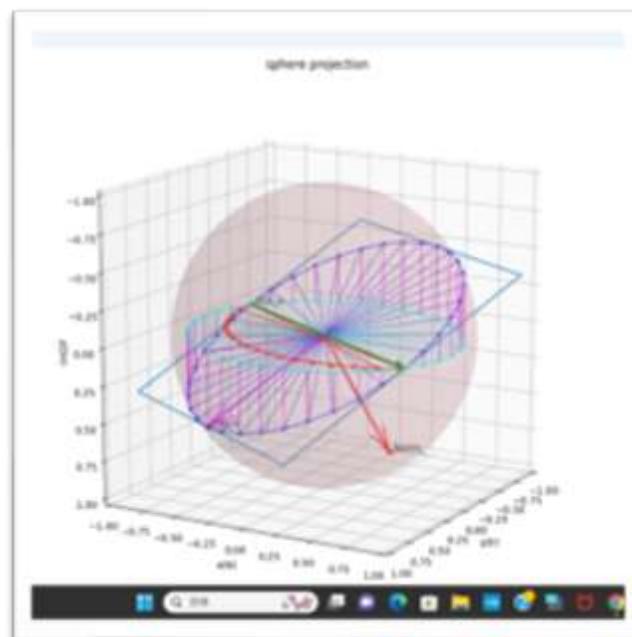


圖 1-1 球體與平面關係圖

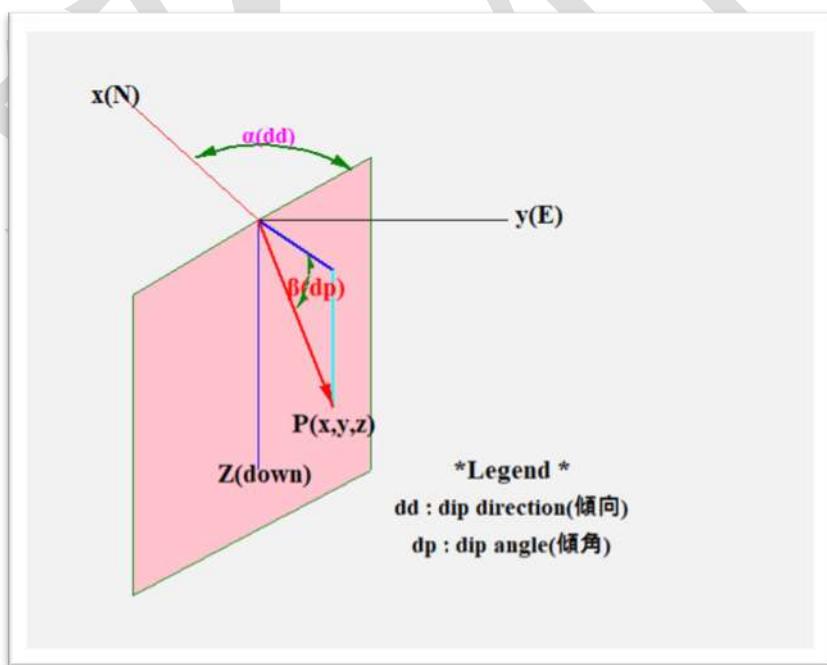


圖 1-2 NED xyz 座標系統

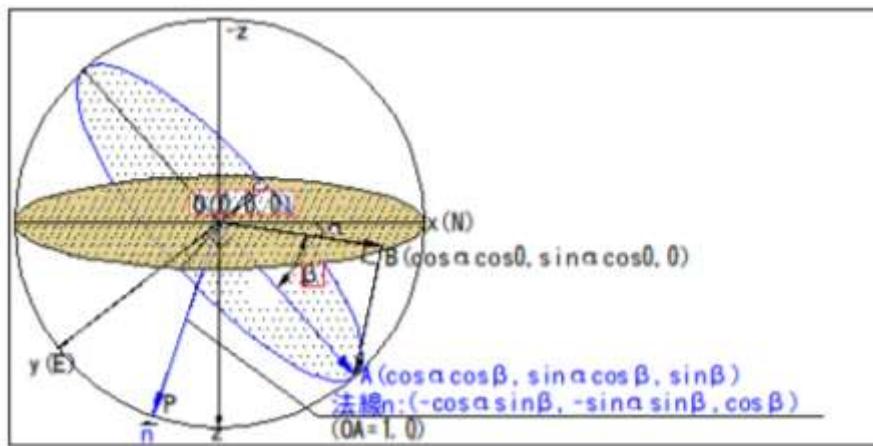
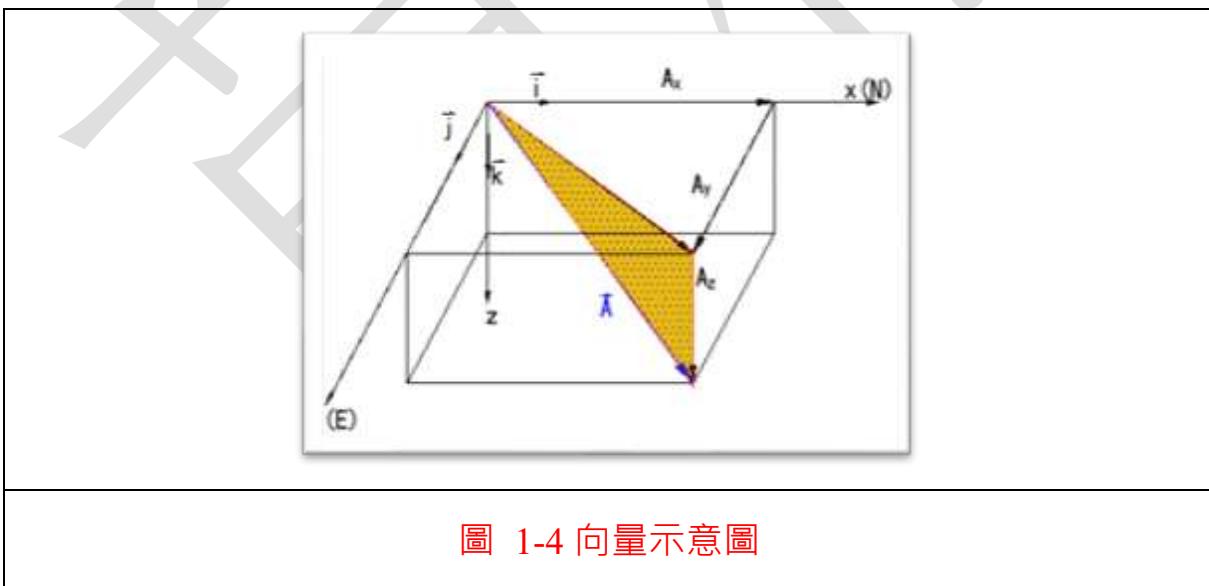


圖 1-3 直角座標系統示意圖

1.2 向量

向量(Vector)或稱矢量，顧名思義就是有方向之量如速度、加速度、力等，其是兼具有『大小』及『方向』之有向線段。它的起點是沒有位置限制，是自由向量，可以在空間自由平行移動。如將向量起點平移至座標原點，則起點至終點之向量即為位置向量。如向量 \underline{A} (以下均採用粗體英文字底加直線者)，在三度空間 x、y 及 z 三軸互相垂直之基底單位向量 \underline{i} 、 \underline{j} 及 \underline{k} ，對應之分量大小分別為 A_x 、 A_y 及 A_z (圖 1-4)， \underline{A} 之大小 A (以下多採用粗體斜英文字或粗體英文字 A或 $|A|$)定義為 $A=\sqrt{A_x^2+A_y^2+A_z^2}=$ $(A_x^2+A_y^2+A_z^2)^{1/2}$ ，即 $\underline{A}=[A_x, A_y, A_z]^*=[A_x/A_y/A_z]^*=A_x\underline{i}+A_y\underline{j}+A_z\underline{k}$ 。

(註:本書向量，原則上多採用如 $[A_x, A_y, A_z]$ 之形式，如中括弧內數字或符號多，不易區別時，則改用 $[A_x, A_y, A_z]$ 之形式)。 \underline{A} 除於其本身長度 A (i.e \underline{A}/A)稱之單位向量 \underline{a} ， \underline{a} 大小為 1.0($\underline{a}=\underline{A}/A=1.0$)。如空間之兩點 A(x_1, y_1, z_1)及 B(x_2, y_2, z_2)，則向量 \underline{AB} 為 $[x_2-x_1, y_2-y_1, z_2-z_1]$ ，大小為 A、B 兩點之直線距離，方向為由 A 指向 B。



1.2.1 向量點積

向量 A 及 B 點積 $A \cdot B$ (Dot product，圖 1-5)，定義為向量 A 在 B 方向之投影長度

$(A \cos \theta)$ 與 B 本身長度(B)之乘積大小，或向量 B 在 A 方向之投影長度($B \cos \theta$)與 A 本身長度(A)之乘積大小。

$$A \cdot B = (A_x, A_y, A_z) \cdot (B_x, B_y, B_z) = A_x B_x (i, i) + A_x B_y (i, j) + A_x B_z (i, k) +$$

$$A_y B_x (i, i) + A_y B_y (i, j) + A_y B_z (i, k) + A_z B_x (i, i) + A_z B_y (i, j) + A_z B_z (i, k)$$

因為 $(i, i) = (j, j) = (k, k) = \cos(0) = 1$, $(i, j) = (j, i) = (i, k) = (k, i) = \cos(90) = 0$

$$A \cdot B = A_x B_x + A_y B_y + A_z B_z$$

$A \cdot B = B \cdot A = A_x B_x + A_y B_y + A_z B_z = AB \cos \theta$	
--	--

如 $a=1$, $b=1$ ，則

$a \cdot b = \cos \theta$	
---------------------------	--

上兩式中 θ 為 A 及 B 之夾角，如 $\theta < 90^\circ$ ， $AB \cos \theta$ 為正值； $\theta > 90^\circ$ ， $AB \cos \theta$ 為負值。

利用向量點積可以求一已知向量在另一向量之投影大小，亦可以求兩向量間之夾角，其常用在大圓、小圓及向量迴轉等球體投影作業工作。向量點積 $A \cdot B$ 具交換性，即 $(A \cdot B) = (B \cdot A)$ 。

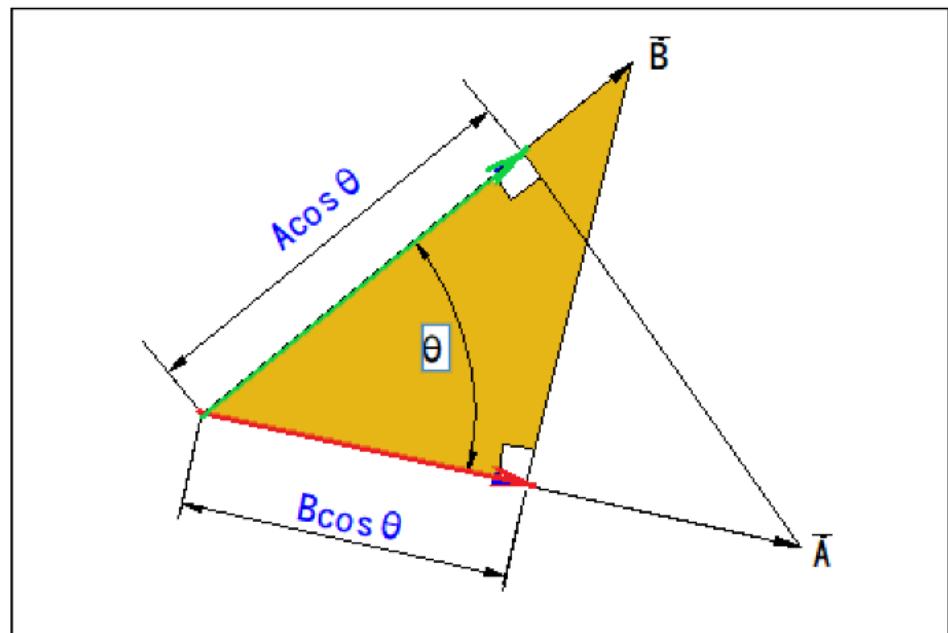


圖 1-5 向量點積示意圖

積所計算之結果及圖示。





圖 1-7 向量點積計算結果及圖示(等角度_Vb6)

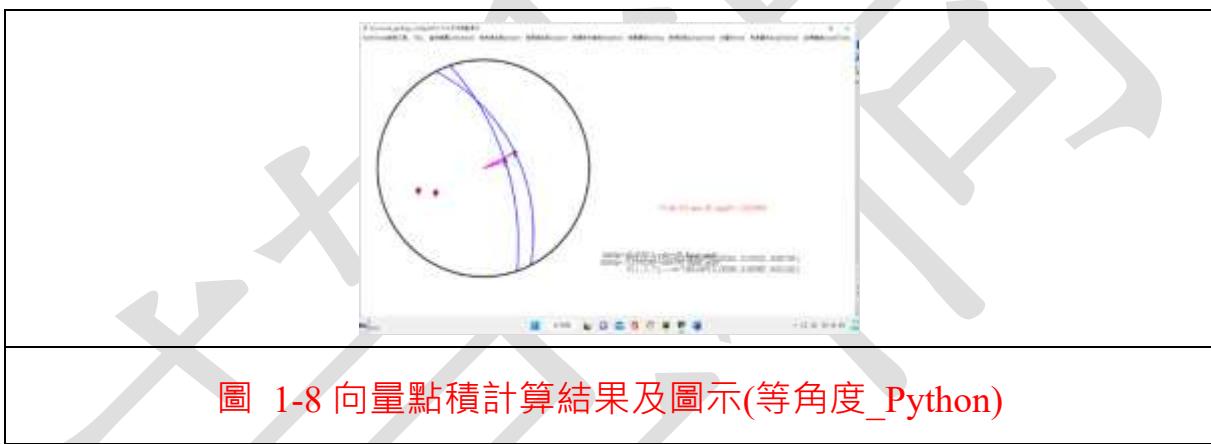


圖 1-8 向量點積計算結果及圖示(等角度_Python)

1.2.2 向量叉積

兩向量叉積 $\underline{A} \times \underline{B}$ (Cross product, 圖 1-9) 仍是向量，大小為 $|\underline{A} \times \underline{B}| = AB \sin \theta$ (θ 為 \underline{A} 及 \underline{B} 之夾角)，其方向分別垂直於向量 \underline{A} 及 \underline{B} ，且向量 $\underline{A} \times \underline{B}$ 、 \underline{A} 及 \underline{B} 三者依右手螺旋定則方式，以 $\underline{A} \times \underline{B}$ 為軸，由 \underline{A} 旋轉一個 θ 角 ($\theta \leq 180^\circ$) 至 \underline{B} 方向， $\underline{A} \times \underline{B}$ (cross(A, B)) 定義為：

$$\underline{A} \times \underline{B} = (A_x, A_y, A_z) \times (B_x, B_y, B_z)$$

$$\underline{A} \times \underline{B} = i \begin{vmatrix} A_y & A_z \\ B_y & B_z \end{vmatrix} - j \begin{vmatrix} A_x & A_z \\ B_x & B_z \end{vmatrix} + k \begin{vmatrix} A_x & A_y \\ B_x & B_y \end{vmatrix}$$

$$\underline{\mathbf{N}} = \underline{\mathbf{A}} \times \underline{\mathbf{B}} = \begin{vmatrix} i & j & k \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix} = i \begin{vmatrix} A_y & A_z \\ B_y & B_z \end{vmatrix} - j \begin{vmatrix} A_x & A_z \\ B_x & B_z \end{vmatrix} + k \begin{vmatrix} A_x & A_y \\ B_x & B_y \end{vmatrix}$$

eq_ 1-1

$$= [(A_y B_z - A_z B_y), (A_z B_x - A_x B_z), (A_x B_y - A_y B_x)]$$

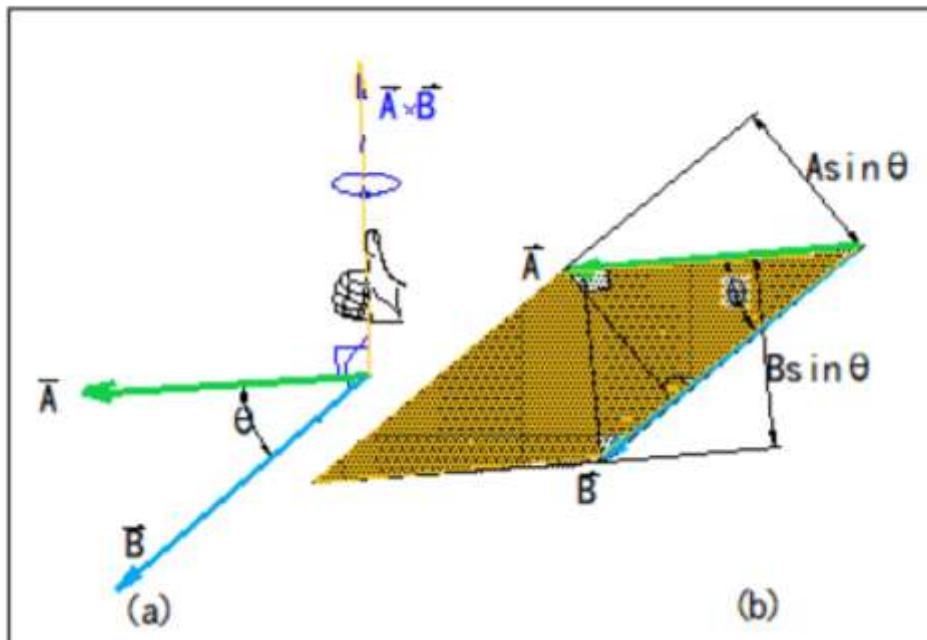


圖 1-9 向量叉積示意圖



圖 1-10 向量叉積結果 (位置向量 · 等角度_Vb6)

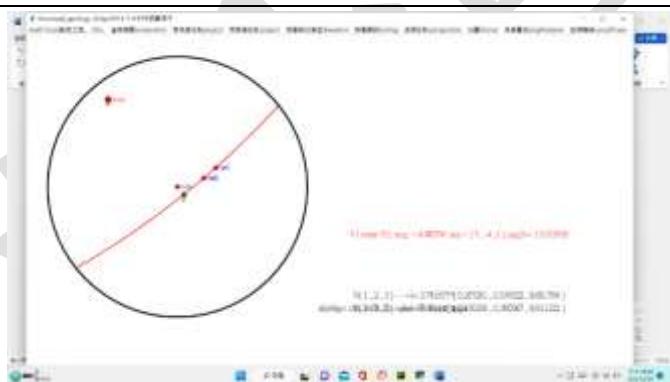


圖 1-11 向量叉積結果(法線向量 · 等角度_Python3)



圖 1-12 向量叉積結果(法線向量 · 等角度_Vb)

1.3 空間平面

圖(圖 1-13)為 120/30 之透視圖。今考慮如(圖 1-14)所示之平面 PL-E 上一定點 $A(x_a, y_a, z_a)$ ，及另一動點 $P(x, y, z)$ ，如座標原點在 $O(0, 0, 0)$ ，則向量 $\underline{A}=\underline{OA}=[x_a - 0, y_a - 0, z_a - 0]$ ，

$\underline{OP}=[x-0, y-0, z-0]$ ，PL-E 上之任一位置向量為 \underline{R} ， i.e

$\underline{R}=\underline{PA}=[x-x_a, y-y_a, z-z_a]$ ，今有一垂直於 PL-E 之法線向量

$\underline{N}=[a, b, c]$ ，則因 \underline{R} 在 PL-E 上，且 \underline{N} 與 \underline{R} 互相垂直，

即 $\underline{N} \cdot \underline{R} = \cos 90^\circ = 0$ ，

i.e $a(x-x_a) + b(y-y_a) + c(z-z_a) = 0$ ，即 $ax+by+cz = ax_a+by_a+cz_a$ ，如令

$d=ax_a+by_a+cz_a$ ，則

$$ax+by+cz=d$$

eq_ 1-2

上式代表 N 與 R 所定義之空間平面方程式，如 PL-E 通過座標原點 $O(0, 0, 0)$ ，即空間平面通過地球球心時，則 $d=0$ ，故 $ax+by+cz=0$

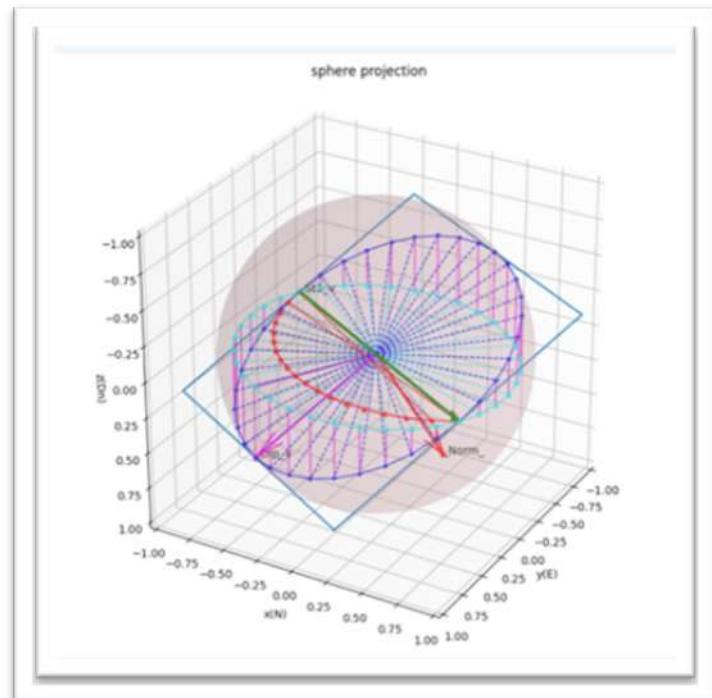


圖 1-13 120/30 3D 圖示

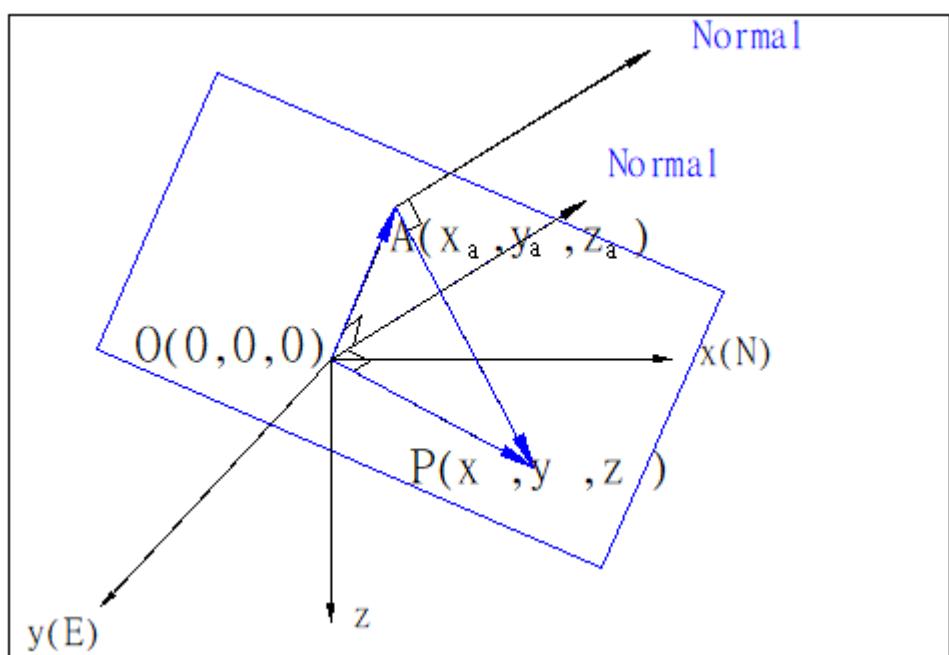


圖 1-14 空間平面示意圖

平面方程式為

$$\begin{vmatrix} x - x_c & y - y_c & z - z_c \\ x_a - x_c & y_a - y_c & z_a - z_c \\ x_b - x_c & y_b - y_c & z_b - z_c \end{vmatrix}$$

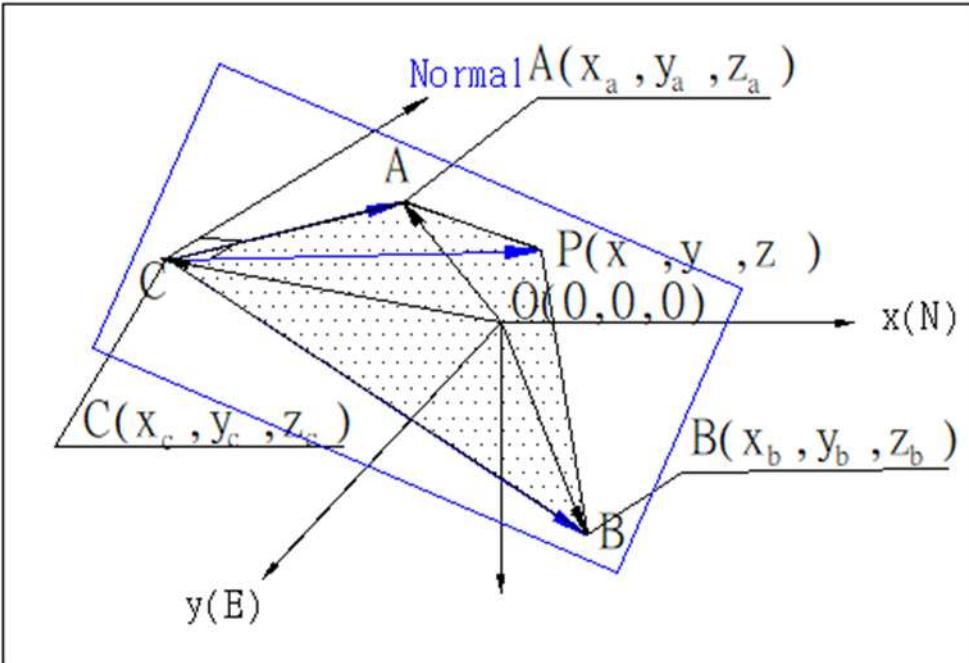


圖 1-15 過空間三定點平面示意圖



圖 1-16 通過(1,2,7)、(3,5,8)及(2,4,7)三點平面圖示(等角度_Vb6)

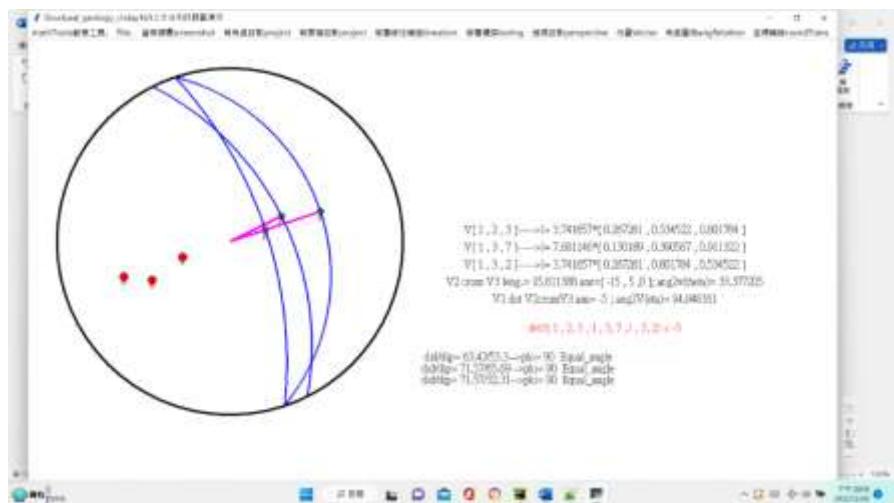


圖 1-17 通過(1,2,7)、(3,5,8)及(2,4,7)三點平面圖示(等角度 Python3)

(圖 1-18)為任意 3D 平面 $P'Q'P''Q''$ ($ax+by+cz=0$)與單位球體 ($\rho=1$)所切割之關係示意圖， OP 為平面之走向， OB 為垂直於走向 OP 之傾向， OA 為代表平面 $ax+by+cz=0$ 之傾向向量。

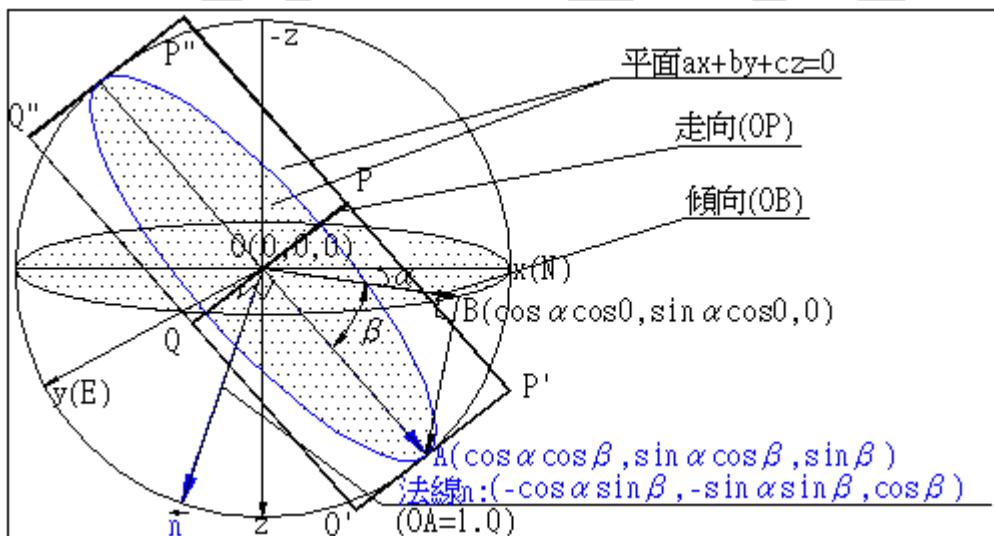


圖 1-18 平面與單位球體關係示意圖

1.4 三元一次聯立方程式

1.4.1 聯立方程式解法

三元一次 3 個聯立方程式的解法如不利用電腦程式，一般以 Excel 求解最為方便。如定義 2*2 行列式(Determinant) $\det2(a, b, c, d)$ 為：

$$\det2(a, b, c, d)=a*d-b*c, \text{ 則}$$

$$\det3(a, b, c, p, q, r, x, y, z)=a*\det2(q, r, y, z)-b*\det2(p, r, x, z)+c*\det2(p, q, x, y)$$

$$a_{11}*x+a_{12}*y+a_{13}*z=a_{14}$$

$$a_{21}*x+a_{22}*y+a_{23}*z=a_{24}$$

$$a_{31}*x+a_{32}*y+a_{33}*z=a_{34}$$

$$\det3_T=\det3(a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33})$$

$$\det3_x=\det3(a_{14}, a_{12}, a_{13}, a_{24}, a_{22}, a_{23}, a_{34}, a_{32}, a_{33})$$

$$\det3_y=\det3(a_{11}, a_{14}, a_{13}, a_{21}, a_{24}, a_{23}, a_{31}, a_{34}, a_{33})$$

$$\det3_z=\det3(a_{11}, a_{12}, a_{14}, a_{21}, a_{22}, a_{24}, a_{31}, a_{32}, a_{34})$$

$$x=\det3_x/\det3_T$$

$$y=\det3_y/\det3_T$$

$$z=\det3_z/\det3_T$$

上述解答亦可以下面之方式求得：

式(1.11)左端常數項如改為 $1*a_{14}, 1*a_{24}, 1*a_{34}$ ，即

$$\det3_T=\det3(a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33})$$

$$\det3_x1=\det3(1, a_{12}, a_{13}, 0, a_{22}, a_{23}, 0, a_{32}, a_{33})$$

$$\det3_y1=\det3(a_{11}, 1, a_{13}, a_{21}, 0, a_{23}, a_{31}, 0, a_{33})$$

$$\det3_z1=\det3(a_{11}, a_{12}, 1, a_{21}, a_{22}, 0, a_{31}, a_{32}, 0)$$

The screenshot shows an Excel spreadsheet with a 3x3 matrix in cells A1 to C3. The matrix is:

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & -1 & 1 \\ 3 & 2 & -1 \end{bmatrix}$$

The determinant calculation for the entire matrix is shown in cell M6: $\text{Det.}_T = 2$.

Below the matrix, individual elements are labeled: $\text{Det.}_x = 2$, $\text{Det.}_y = -2$, and $\text{Det.}_z = 4$. To the right of these are check values: $\text{chk1} = 0$, $\text{chk2} = 0$, and $\text{chk3} = 0$. Cells A6 through G8 are highlighted in orange.

201.0x+0.0y+0.0z+0.0
0.0x+201.0y+0.0z+0.0
0.0x+0.0y+201.0z+0.0
0.0x+0.0y+0.0z+201.0
Det_x = 201.0, Det_y = 0.0, Det_z = 0.0
det = 201.0

圖 1-19 利用 Python 之計算結果(同表 1.1)

The screenshot shows a Scratch script titled "矩阵乘法". It consists of the following blocks:

- A yellow control block "when green flag clicked" with two branches:
 - One branch containing a yellow control block "repeat [3] [clear pen]" which then contains a yellow control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]". This block is connected to a blue control block "repeat [3] [clear pen]" which then contains a blue control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]". Both branches of this loop are connected to a blue control block "repeat [3] [clear pen]" which then contains a blue control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]".
 - The other branch contains a pink control block "repeat [3] [clear pen]" which then contains a pink control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]".
- A yellow control block "repeat [3] [clear pen]" which then contains a yellow control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]".
- A blue control block "repeat [3] [clear pen]" which then contains a blue control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]".
- A pink control block "repeat [3] [clear pen]" which then contains a pink control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]".
- A blue control block "repeat [3] [clear pen]" which then contains a blue control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]".
- A yellow control block "repeat [3] [clear pen]" which then contains a yellow control block "repeat [3] [say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec] say [100 x %] over [100 y %] over [100 z %] wait [0.1 sec]]".

圖 1-20 利用 Scratch 程式之計算結果((同表 1.1)

The screenshot shows an Excel spreadsheet with data in columns A through O. The first two rows contain numerical values. Row 3 contains the equation $x^3 + y^3 + z^3 = 3xyz$. Row 4 contains the equation $2x^3y - xy^3 + 2y^3z - zy^3 + 2z^3x - zx^3 = 6xyz(x+y+z)$. Rows 5 and 6 show the solutions for each equation, resulting in the same three pairs of values: (1.947725, 2.184656, -0.658518), (-0.125503, -0.406878, 2.969629), and (-0.571429, -0.714286, 1.571429).

圖 1-21 二聯立方程式及球體解(x,y,z)

```

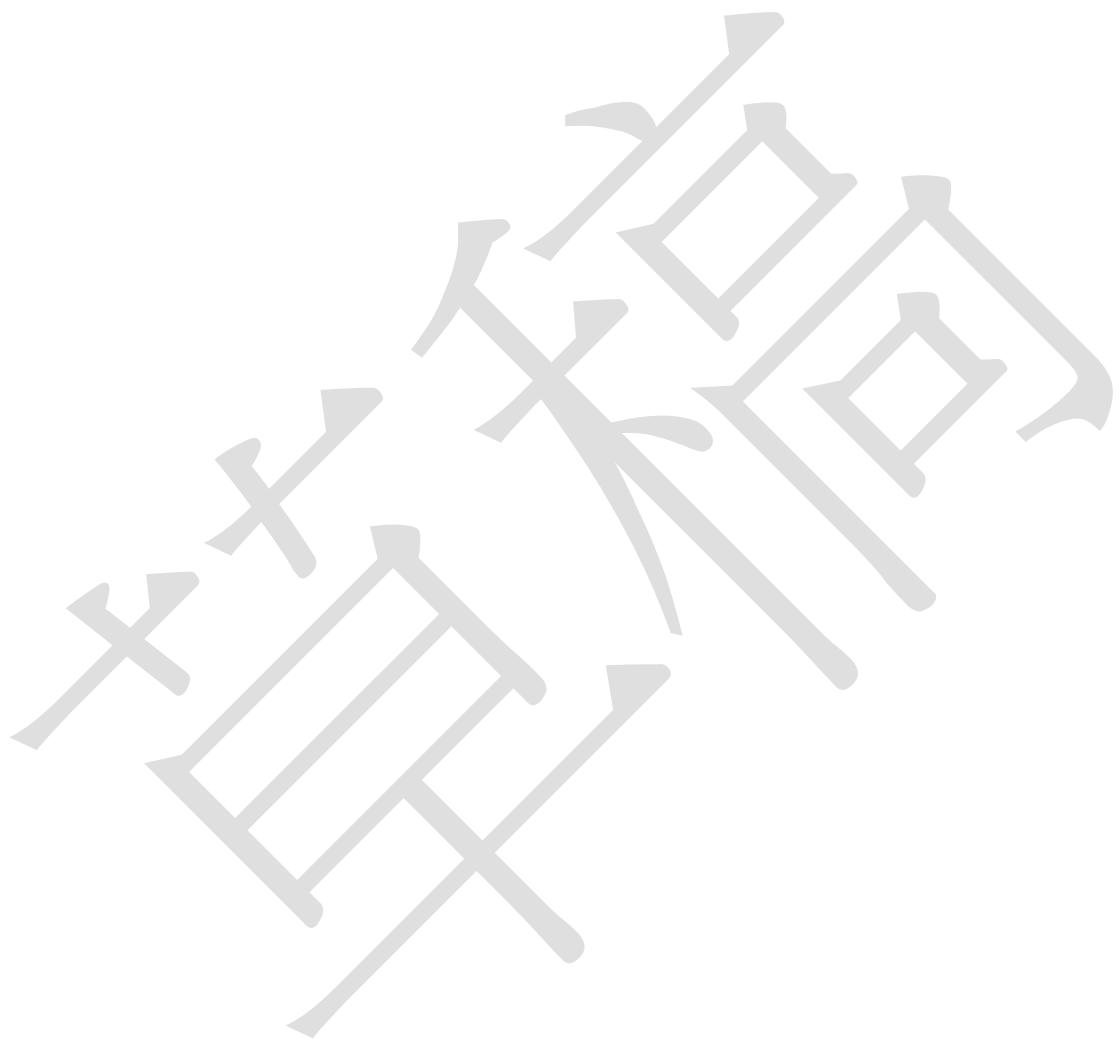
000.0+x1,y1=0.0,0.0
004.0x1y1y0+0.01y0=0.0
0000x1+0.0333y0+0.001y0<
x3+y3+z3=3xyz(3)
<0> 1.947725 , y1= 2.184656 , z1= -0.658518
<0> -0.125503 , y1= -0.406878 , z1= 2.969629
abs(x1)+r(y1)+c(z1)-cd*ad<0.0>_min(3*x1+y2*z1+a2*d-cd*ad)<0.0>

```

圖 1-22 利用 Python 計算結果(同表 1.2)



圖 1-23 利用 Scratch 程式之計算結果(同表 1.2)



1.5 兩平面交線

1.5.1 聯立方程式解法

兩平面如傾角方向及傾角已知(錯誤！找不到參照來源。)，其單位法線向量 $\underline{n}_1 = [a_1, b_1, c_1]$ 及 $\underline{n}_2 = [a_2, b_2, c_2]$ ，則解代表兩平面之聯立方程式為，

$$a_1x + b_1y + c_1z = d_1 \text{ 及}$$

$$a_2x + b_2y + c_2z = d_2$$

可得兩平面交線向量之方向比為

兩平面交線向量之方向比為

$$\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} : - \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} : \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = (b_1c_2 - b_2c_1) : (c_1a_2 - c_2a_1) : (a_1b_2 - a_2b_1)$$

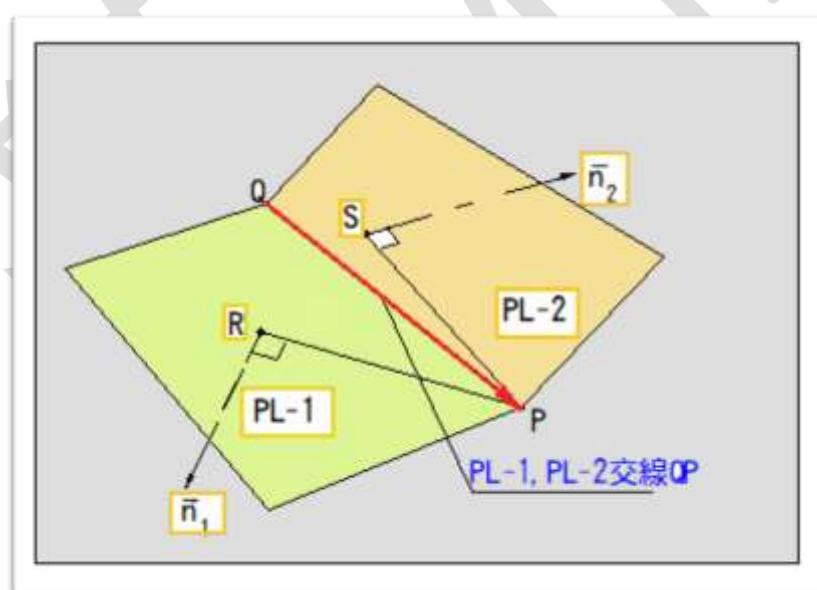


圖 1-24 兩平面交線示意圖



圖 1-25 平面交線計算結果及圖示(等面積_Vb6)



圖 1-26 向量平面計算結果及圖示(等面積_Vb6)

1.5.2 向量解法

空間之兩平面如其法線向量已知時，依向量叉積之定義，可計算出其交線，假定兩平面之任意單位位置向量為，

$\underline{i}_A = [x_A, y_A, z_A] = [\cos p_A \cos q_A, \sin p_A \cos q_A, \sin q_A]$	
$\underline{i}_B = [x_B, y_B, z_B] = [\cos p_B \cos q_B, \sin p_B \cos q_B, \sin q_B]$	

朝下單位法線向量分別為，

$\underline{n}_A = [a_A, b_A, c_A] = [-\cos p_A \sin q_A, -\sin p_A \sin q_A, \cos q_A]$	
$\underline{n}_B = [a_B, b_B, c_B] = [-\cos p_B \sin q_B, -\sin p_B \sin q_B, \cos q_B]$	

式(錯誤！找不到參照來源。)~(錯誤！找不到參照來源。)中之 p_A 、 q_A 及 p_B 、 q_B 為平面上任意向量之線向及傾沒角。

則平面 PL-A 與平面 PL-B 相交之直線向量 \underline{I}_{AB} ，可以垂直於該兩平面之單位法線向量 (\underline{n}_A 與 \underline{n}_B) 叉積(Cross product)表示之，即

$\underline{I}_{AB} = [x_i, y_i, z_i] = \underline{n}_A \times \underline{n}_B = \pm \begin{vmatrix} i & j & k \\ -\cos p_A \sin q_A & -\sin p_A \sin q_A & \cos q_A \\ -\cos p_B \sin q_B & -\sin p_B \sin q_B & \cos q_B \end{vmatrix}$	
---	--

，如令 $\ell_i = I_{AB} = (x_i^2 + y_i^2 + z_i^2)^{1/2}$ ， \underline{i}_{AB} 為 \underline{I}_{AB} 之單位向量，

即

$\underline{i}_{AB} = \underline{I}_{AB} / \ell_i = [x_i, y_i, z_i] / \ell_i = [\cos \alpha_i \cos \beta_i, \sin \alpha_i \cos \beta_i, \sin \beta_i]$	
--	--

$\alpha_i = \tan^{-1}(y_i / x_i)$	
$\beta_i = \sin^{-1}(z_i / \ell_i)$	

式(錯誤! 找不到參照來源。)~(錯誤! 找不到參照來源。)中 α_i , β_i 分別為 \underline{I}_{AB} (或 \underline{i}_{AB})線向(Trend)之方位角及傾沒角(Plunge)。(注意因交線為位置向量，故 \underline{i}_{AB} 應假設如式(1.17)之形式，而非為單位法線之形式。)

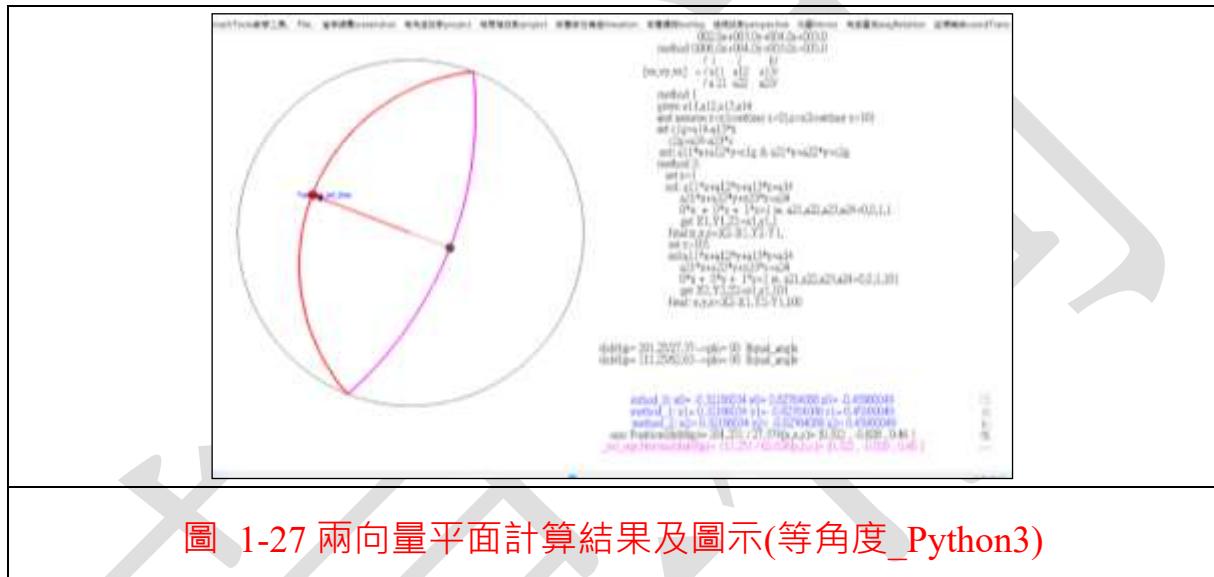


圖 1-27 兩向量平面計算結果及圖示(等角度_Python3)

1.6 岩楔交點相對位置

有關岩楔之稜線長度、面積、體積及各頂點之相對位置或煤層隱藏量估算等之求解，一般教科書或報告，多藉球體投影或正交投影方式，以圖解法或利用數學向量法求解。作圖法過程繁複費時，不適合電腦程式設計；向量解法及聯立方程式解法，比較適合以電腦計算，其可提供計算者岩楔之相對位置關係。聯立方程式解法比向量解法更簡單明瞭，觀念更為直接，對初學者而言，應該比較容易接受及理解，因此本節除介紹向量解法外，作者另外推介聯立方程式解法，提供讀者另類選擇。

1.6.1 向量解法

如(錯誤！找不到參照來源。)之岩楔，各不連續面傾向傾角已知，底部平面PL-1，假定在觀測者左邊；PL-2在底部右側；岩坡上部之平面為PL-3；下部平面則為PL-4。PL-1, 2, 3之交點假定為點P₁；PL-1, 2, 4之交點為P₂；PL-1, 3, 4之交點為P₃；PL-2, 3, 4之交點為P₄。點P₁P₂間之直線為l₁；點P₁P₃間之直線為l₂；點P₁P₄間之直線為l₃；點P₂P₃間之直線為l₄；點P₂P₄間之直線為l₅。並假設l₁~l₄之單位向量分別為*i₁*、*i₂*、*i₃*及*i₄*，方向均朝向點P₂(即指向觀測者)。由l₁*i₁*=l₂*i₂*+l₄*i₄*，及l₁*i₁*=l₃*i₃*+l₅*i₅*之恆等關係，可分別得相關之x、y及z三個方向之條件方程式。若岩楔任一稜線長度或任一x、y或z方向之尺寸已知時，可解任兩恆等式，即可求出同一平面鄰邊之長度，依序計算出其他稜線長度後，相關各平面之面積及岩楔體積，可依向量之叉積及三重積求得。

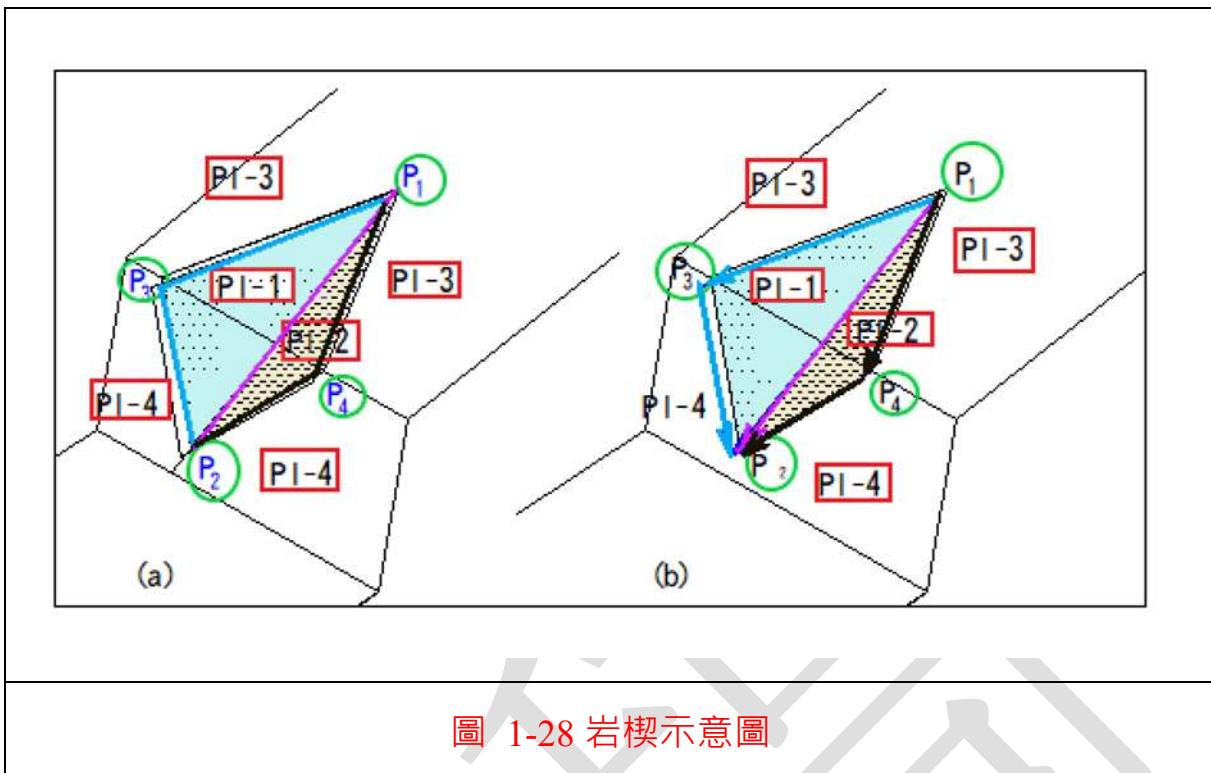


圖 1-28 岩楔示意圖

$$x_2 = 0 + 45.199 \times (-0.8438) = -38.1397$$

$$y_2 = 0 + 45.199 \times (0.2599) = 11.7493$$

$$z_2 = 0 + 45.199 \times (0.4695) = 21.2203, \text{ i.e.}$$

$P_2 [-38.1397, 11.7493, 21.2203]$, 依同樣程序得

$P_3 [-32.6810, 2.8592, 11.2203]$,

$P_4 [-33.3725, 16.1720, 10.2093]$

1.6.2 聯立方程式解法

如上圖(錯誤!找不到參照來源。)之岩楔，因各不連續面之傾向傾角已知，各別單位法線可以求得，故其對應之平面方程式，可以下列方式定義。

$a_1 x + b_1 y + c_1 z = d_1$	
-------------------------------	--

$a_2 x + b_2 y + c_2 z = d_2$	
$a_3 x + b_3 y + c_3 z = d_3$	
$a_4 x + b_4 y + c_4 z = d_4$	

$a_1 x + b_1 y + c_1 z = 0$	
$a_2 x + b_2 y + c_2 z = 0$	
$a_3 x + b_3 y + c_3 z = 0$	

解聯立方程式(錯誤！找不到參照來源。)、(錯誤！找不到參照來源。)及(錯誤！找不到參照來源。)，可得 P_2 座標

$x_2 = \begin{vmatrix} 0 & b_1 & c_1 \\ 0 & b_2 & c_2 \\ d_4 & b_4 & c_4 \end{vmatrix} / \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_4 & b_4 & c_4 \end{vmatrix}$	
$y_2 = \begin{vmatrix} a_1 & 0 & c_1 \\ a_2 & 0 & c_2 \\ a_4 & d_4 & c_4 \end{vmatrix} / \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_4 & b_4 & c_4 \end{vmatrix}$	
$z_2 = \begin{vmatrix} a_1 & b_1 & 0 \\ a_2 & b_2 & 0 \\ a_4 & b_4 & d_4 \end{vmatrix} / \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_4 & b_4 & c_4 \end{vmatrix}$	

同理解相關方程式可得點 P_3 座標及 P_4 座標。若岩楔任一稜線長度或 x 、 y 或 z 任一方
向之長度已知時，可求出 d_4 真正大小，因而求出 $P_2 \sim P_4$ 座標值及相關各平面之面積及
岩楔體積

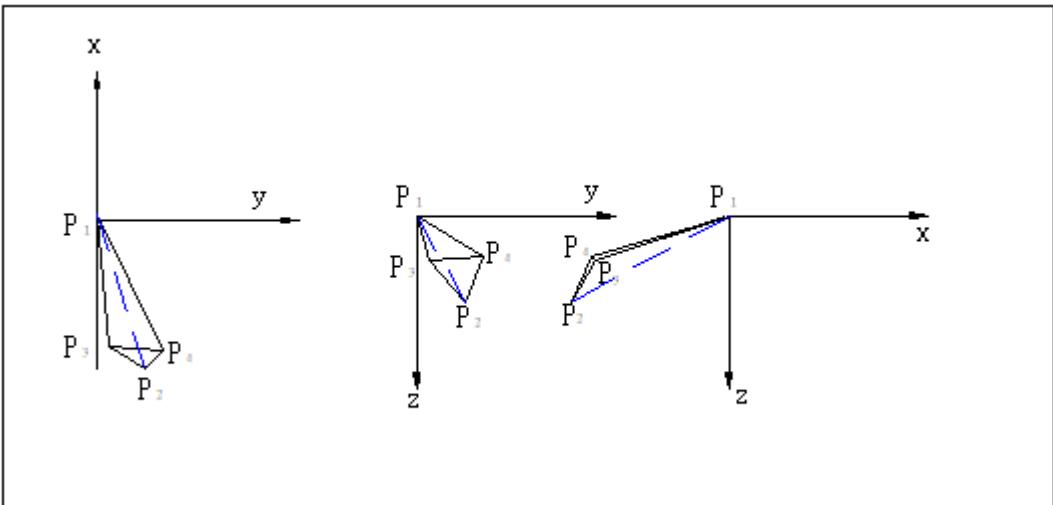


圖 1-29x-y,y-z,x-z 投影示意圖

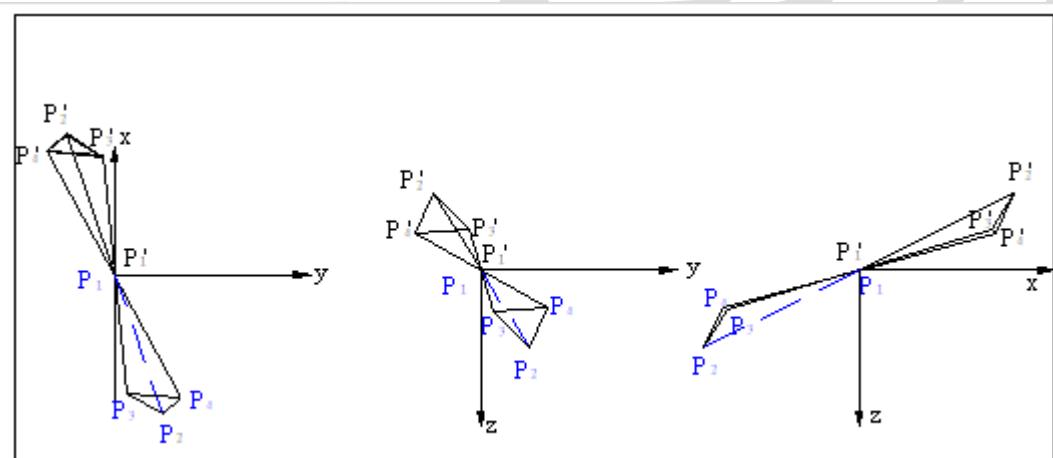


圖 1-30x-y,y-z,x-z 外伸與非外伸投影示意圖

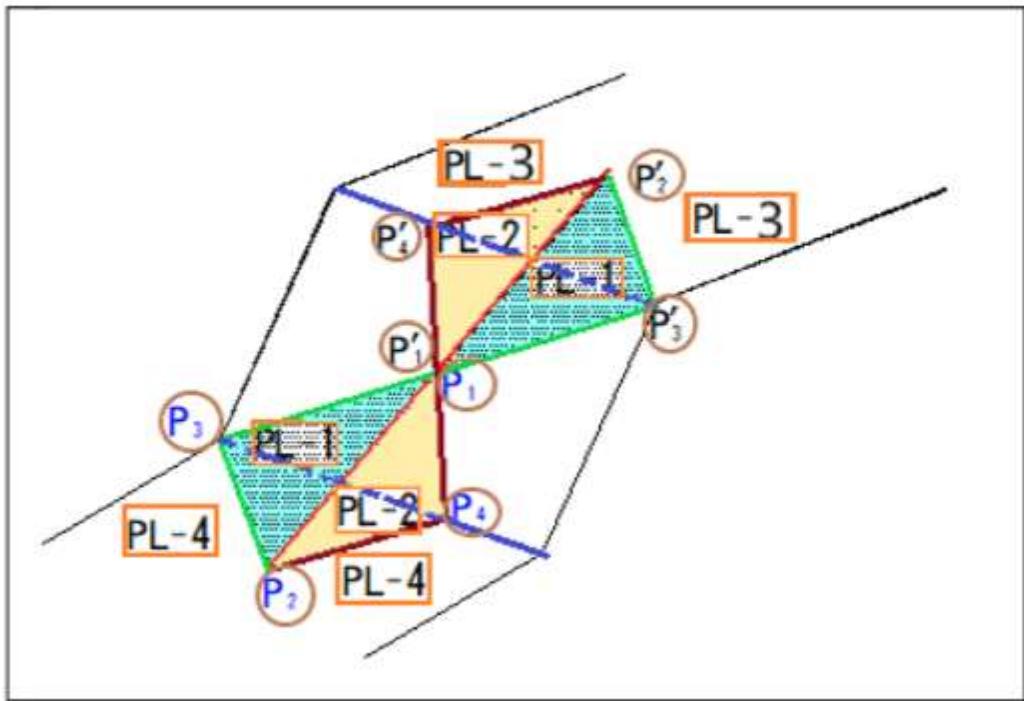


圖 1-31 外伸與非外伸岩坡示意圖

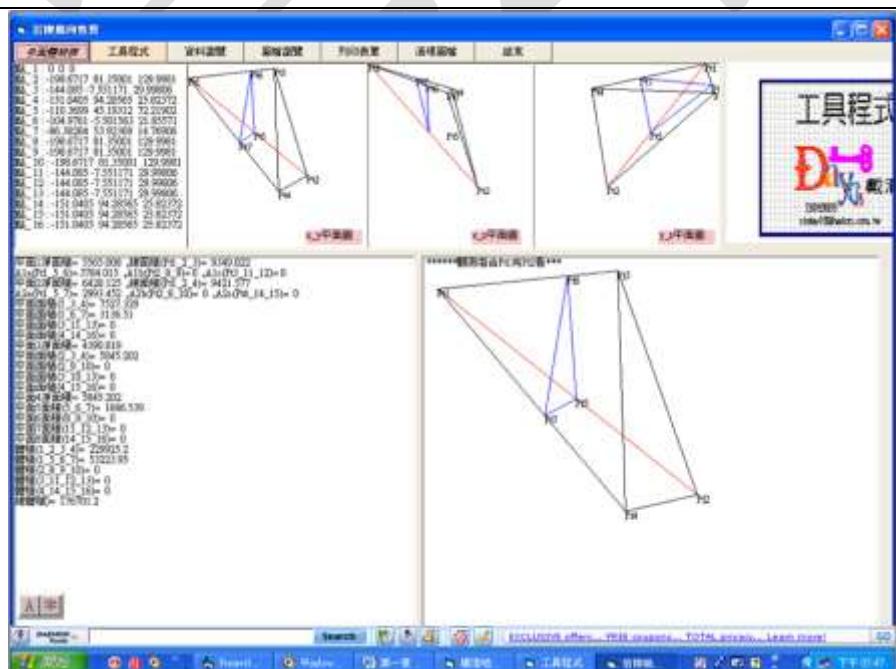


圖 1-32 多面體岩楔之計算結果及圖示(Vb)

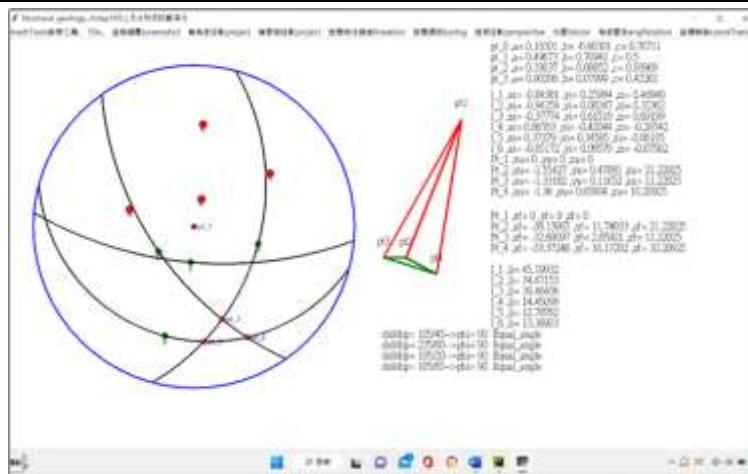
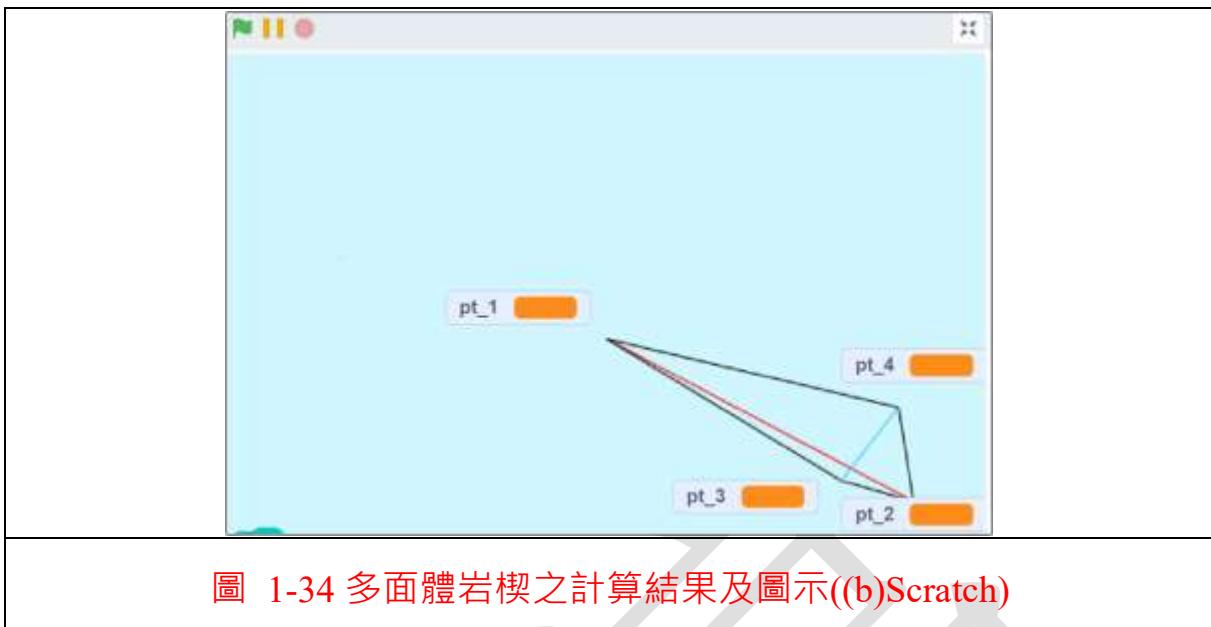


圖 1-33 多面體岩楔之計算結果及圖示(Python)

TurtleWin Desktop				
norm_x_list	norm_y_list	norm_z_list	list_xpt	
1 0.1830...	1 -0.68301...	1 0.707106...	1 0...	
2 0.4967...	2 0.709406...	2 0.5...	2 -1.554272...	
3 0.3303...	3 0.0888321...	3 0.939692...	3 -1.331820...	
4 0.9028...	4 0.078989...	4 0.422618...	4 -1.359999...	
+ length 4	= + length 4	= + length 4	= + length 4	=
list_ypt	list_zpt	list_xf	list_yf	list_zf
1 0	1 0	1 0	1 0	1 0
2 0.478810...	2 0.864770...	2 -36.13964...	2 11.74932...	2 21.2202...
3 0.116519...	3 0.457249...	3 -32.88097...	3 2.859214...	3 11.22024...
4 0.659044...	4 0.416049...	4 -33.37245...	4 18.17201...	4 10.2092...
+ length 4	+ length 4	+ length 4	+ length 4	+ length 4 =

圖 1-35 多面體岩楔之計算結果及圖示((a)Scratch)



1.7 三度空間向量繞任意軸旋轉

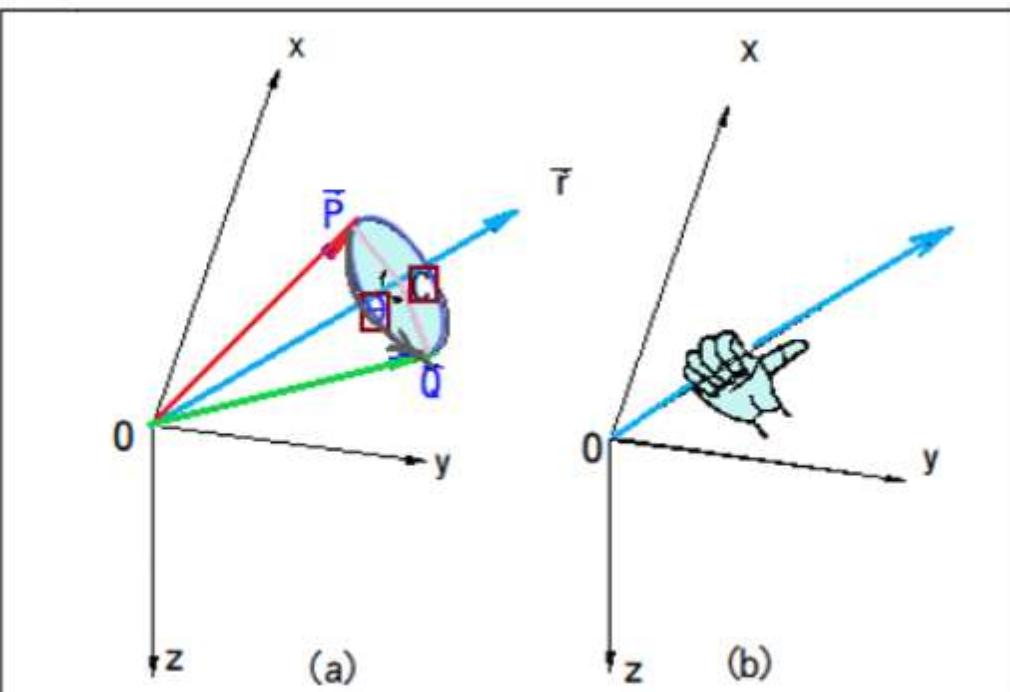


圖 1-35 向量繞傾斜軸旋轉示意圖

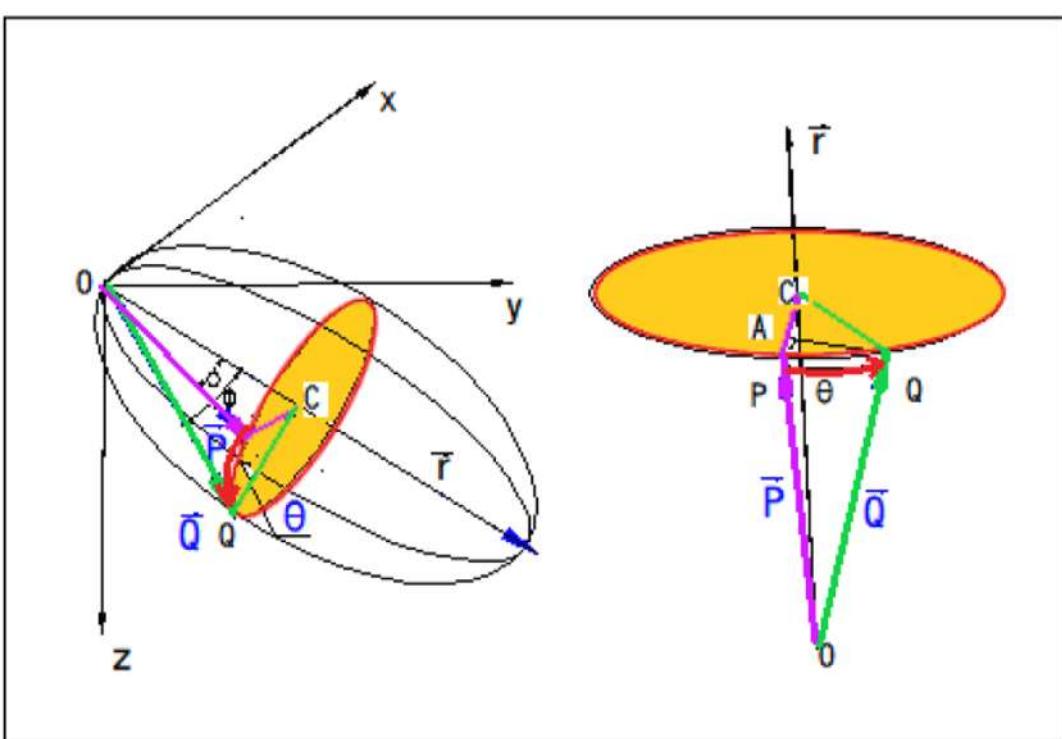


圖 1-36 向量繞任意軸旋轉示意圖

表 3 繞任意軸旋轉

以單位法線代表平面									
旋轉角度	85.0								
項目	傾向	傾角	x	y	z				
旋轉軸	0	80.0	30.0	0.1504	0.8529	0.500	R vect.	case	
面單位法線	00	210.	45.0	0.6124	0.3536	0.707	P vect.	v posit.	xprj yprj
									0.3587 0.2071
			y z	叉積 [R]*[P]_x	x z	叉積 [[R]*[P]]_y	x y	叉積 [[R]*[P]]_z	點積 [R].[P] 點積案 (1-COS())
旋轉軸	29	0.85	0.50	0.4263	0.1504	0.500	0.15	0.4691	0.7472 0.6821
面單位法線	36	0.35	0.70		0.6124	0.707	0.61		
		xq	yq	chk. 1e	did-rot	dip-rot	didf	xprj(no rm)	yprj(no rm)
		0.58	0.81	-	1.0000	54.42	86.29	54.4	86.29 -0.5453 -0.7623

	06	16	0.0647			2			
--	----	----	--------	--	--	---	--	--	--

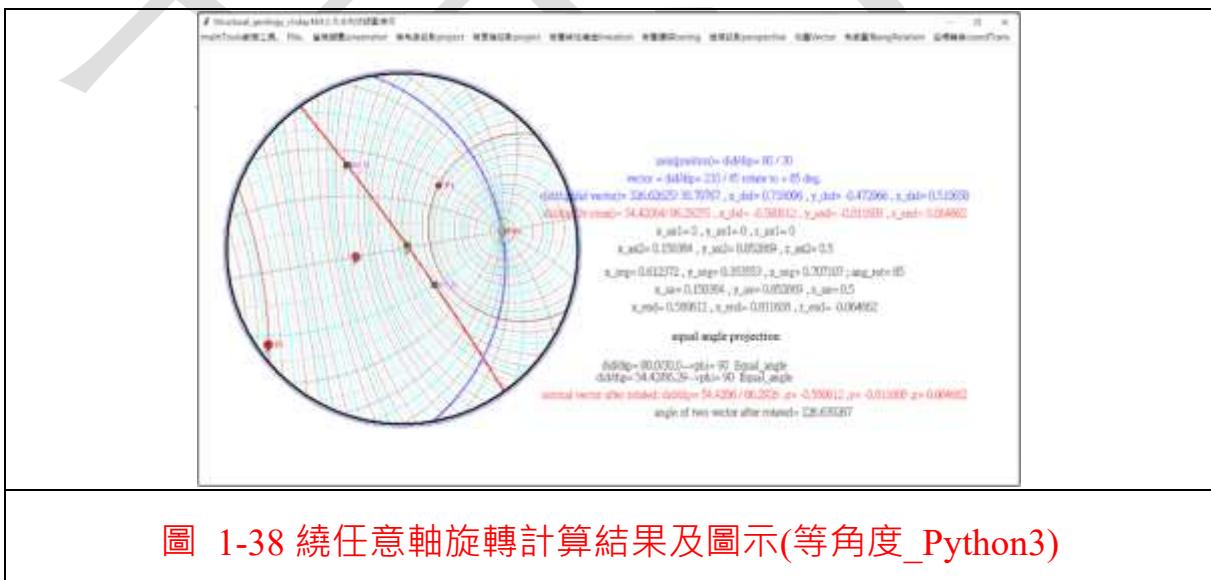
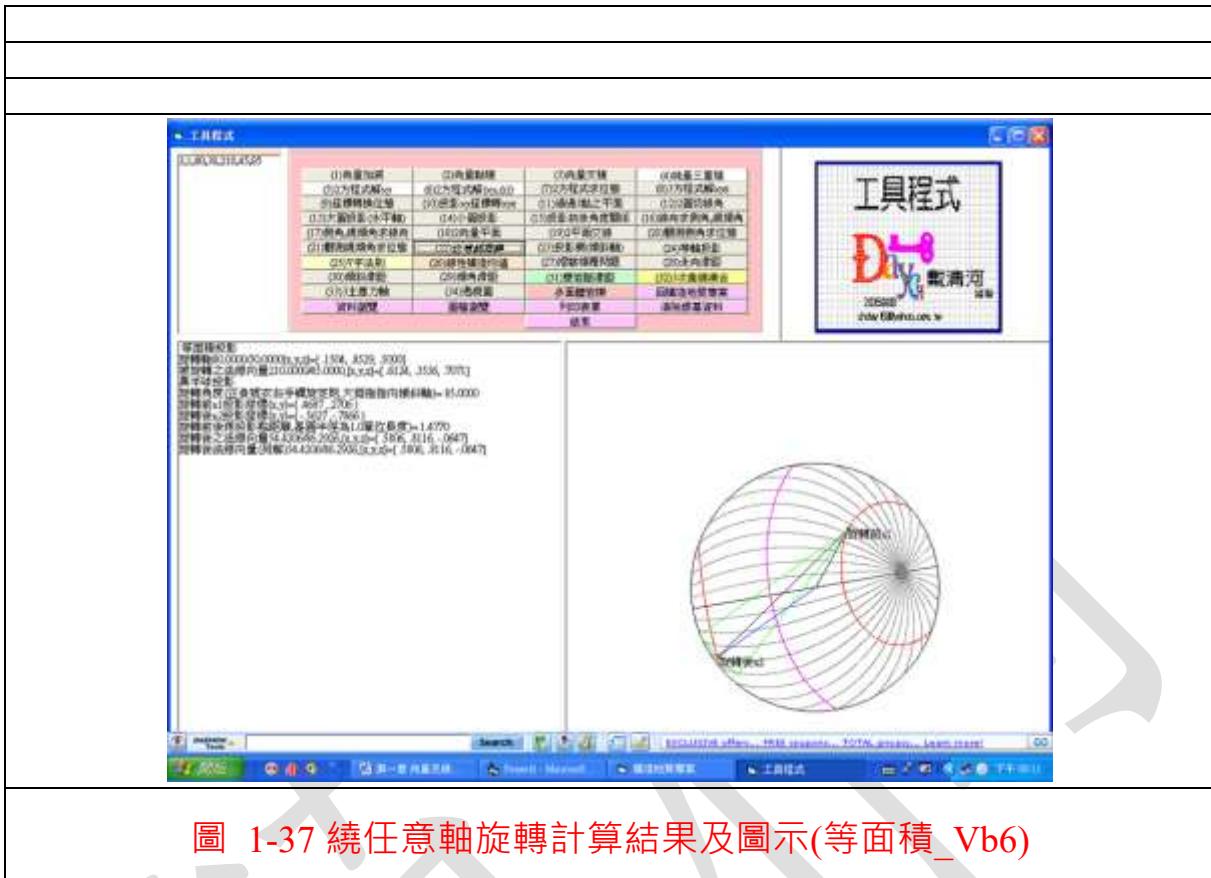




圖 1-39 繞任意軸旋轉計算結果及圖示(等角度_Scratch)

1.8 常用單位單位向量、單位法線向量與向量旋轉矩陣

Apsg 有 line()，pole() 與 u. rotate(v)，這些我們會另闢專章說明，有鑑於其重要性，特將相關之 Python 程式碼在此分享大家。

```
def line(dd=None, dp=None, xin=None, yin=None, zin=None, *kw, **kwg):
    if dd!=None and dp!=None:
        if abs(dp)<0.00001:
            dp=0.00001
        x, y, z=math.cos(dd*Ra)*math.cos(dp*Ra), math.sin(dd*=ra)*math.cos(dp*Ra), math.sin(dp* Ra)
        if z<=0.:
            x, y, z=-x, -y, -z
        #print(' dd, dp, x, y, z' , dd, dp, x, y, z)
        if 'return_x_y_z' in kw and kwg['return_x_y_z']:
            return x, y, z
        return dd, dp, x, y, z
    if xin!=None and yin!=None and zin!=None:
        l=math.sqrt(xin**2+yin**2+zin**2)
        x, y, z=xin/l, yin/l, zin/l
        if z<=0.:
            x, y, z=-x, -y, -z
        dd, dp=math.atan2(y, x)*Da, math.asin(z)*Da
        if dd<=0.:
            dd=dd+360
        if 'return_dd_dp' in kw and kwg['return_dd_dp']:
            return dd, dp
        if 'return_x_y_z' in kw and kwg['return_x_y_z']:
            return x, y, z
        #print(' dd, dp, x, y, z' , dd, dp, x, y, z)
    return dd, dp, x, y, z
```

did, dip, x, y, z=line(dd=120, d0=30):	120	30	-0.4330127018922192	0.7500000000000001
0.4999999999999994				
did, dip=line(xin=3, yin=4, zin=5, return_dd_dp=True):	53.13010235415598	44.99999999999999		
did, dip, x, y, z=line(xin=3, yin=4, zin=5):	53.13010235415598	44.99999999999999		
0.4242640687119285 0.565685424949238 0.7071067811865475				
x, y, z=line(dd=120, dp=30, return_x_y_z=True):	-0.4330127018922192	0.7500000000000001		
0.4999999999999994				
result=line(xin=3, yin=4, zin=5):	(53.13010235415598,	44.99999999999999,		
0.4242640687119285, 0.565685424949238, 0.7071067811865475)				

圖 1-40line()函數程式及執行成果

上面之函數 line(dd=None, dp=None, xin=None, yin=None, zin=None, *kw, **kwg) 相當好用有彈性，如只輸入 dd=***, dp=** 無關鍵字，可回傳 did, dip, x, y, z 或元組 tuple(); 如只輸入 xin=***, yin=***, zin=*** 無關鍵字，可回傳 did, dip, x, y, z 或元組

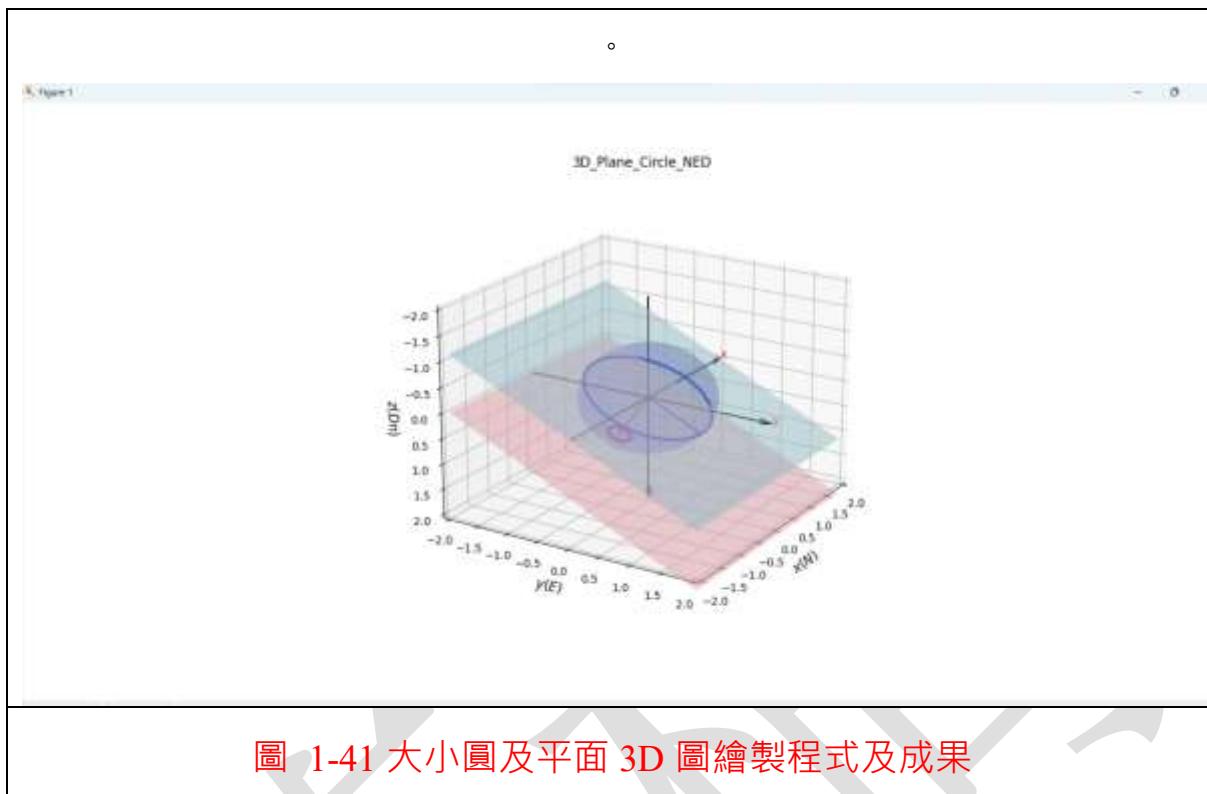
tuple()；使用關鍵字

return_dd_dp=True 只回傳 did, dip；使用關鍵字 return_x_y_z=True 只回傳 x, y, z。

◦



1.9 3D 圖示



1.10 利用 AI 自我學習構造地質學：

Ai 是目前很夯也很實用的工具，在自我學習方面非常方便有效，尤其是在需要資料收集及文案處理方面特別優異，更重要的是免費(目前)。有一點必須了解的是，Ai 並非萬能?Ai 可以幫忙寫程式，但編寫之程式(Ai 多只會收集資料庫中現有程式，彙整判斷後回應?)，不保證正確合用，必須多方偵錯、測試再測試、比對驗證(用已知答案)。當然使用者亦要俱備有編寫程式基本知識及能力。您所發問及要求之內容，愈詳實正確，Ai 回應也會愈能符合使用者的要求。如果程式需要使用到特別之公式，使用者必須提供及說明參數代表之意義，Ai 所回應之程式多比較簡潔，使用者已編寫可執行之程式可請Ai 修飾至更精簡。至於發問內容，可用中文或英文，或中英文夾雜。下面就以幾個實例示範用法，因為英文書輸入較快，輸入就多以本人破英文為主。

(a) Pls explain what is ‘Vector’ ?(請解釋向量?)

AI 回應(一刀未剪):

向量是數學中的基本概念，在物理、計算機科學和工程等領域中都有廣泛應用。讓我為您解釋一下：

定義：向量是一個具有大小（長度）和方向的量。它表示空間中一點到另一點的位移。

分量：向量可以表示為不同坐標軸上的分量的有序集合。例如，在二維空間（如平面）中，一個向量 v 可以表示為 $v = (v_x, v_y)$ ，其中 v_x 和 v_y 是分別沿 x 軸和 y 軸的分量。

幾何解釋：

幾何上，向量可以想像成一個帶有一定長度和方向的箭頭。

向量的大小（表示為 $|v|$ ）是箭頭的長度。

向量的方向是它與參考軸（通常是正 x 軸）之間的角度。

示例：

位移向量：在物理學中，位移向量描述了物體從一個位置移動到另一個位置。

速度向量：速度是一個結合了速度（大小）和方向的向量。

力向量：作用在物體上的力（如重力或張力）用向量表示。

位置向量：這些指示一個點相對於參考點（通常是原點）的位置。

運算：

相加：要相加兩個向量，只需將它們對應的分量相加。

純量乘法：您可以將向量乘以純量（一個數字）以改變其大小。

點積：兩個向量的點積給出一個純量值（不是向量），在物理學和工程中使用。

叉積：兩個向量的叉積給出一個新的向量，該向量垂直於兩個輸入向量。

表示法：

向量通常用粗體字母表示（例如， v 、 u 、 a ）。

或者，在字母上方加上箭頭（例如， $\rightarrow v$ ）表示它是一個向量。

請記住，向量是描述物理量、建模運動並在各種科學和工程領域中解決問題的基本工具！

(b)Plot 3d vectors

(1)Please give a python program which include(a) a input box to input n 3d vectors, the input box contain prompt, initial values. When user press 'Esc' key it mean that input data finished.

(2)Suppose we get 2 vectorA= [1, 2, 3] and B=[4, 5, 6] plot this 2 vectors in 3d diagram. the coord system is NED(x(N), y(E), z(down)) and in plane N(right), E(point to user). z(downward)

(3)Calculate C=A +B. . plot A, B C with quivers & print A, B, C.

(4)The plot provide grids, x, y, z axes, and text with coords on vectors' position

請求給出一個 python 程式，其中包括 (a) 一個輸入框來輸入 n 個 3d 向量，輸入框包含提示、初始值。當使用者按「Esc 鍵」時，輸入數據完成。假設我們得到 2 個向量 A= [1, 2, 3] 和 B=[4, 5, 6] 在 3d 圖中繪製這 2 個向量。坐標系為 NED (x (N), y (E), z (向下))，在平面 N (右)，E (指向使用者). z (向下) Calculate C=A +B. .

圖 A，B C 與箭頭和輸出 A，B，C。 該圖需提供網格、x、y、z 軸和文本，並在向量位置上提供座標

AI 第一次回應

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def input_vectors():
    vectors = []
    while True:
        try:
            vector_str = input("Enter a 3D vector (e.g., 1, 2, 3) or 'esc' to finish:")
        except ValueError:
            print("Invalid input. Please enter numeric values separated by commas.")
        if vector_str.lower() == "esc":
            break
        vector = tuple(map(float, vector_str.split(",")))
        if len(vector) != 3:
            print("Invalid input. Please enter a 3D vector.")
        else:
            vectors.append(vector)
    return vectors

# Input n 3D vectors
user_vectors = input_vectors()
print("User input vectors:", user_vectors)

# Plotting vectors A and B
A = np.array([1, 2, 3])
B = np.array([4, 5, 6])

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.quiver(0, 0, 0, A[0], A[1], A[2], color="r", label="A")
ax.quiver(0, 0, 0, B[0], B[1], B[2], color="g", label="B")

# Set labels and title
ax.set_xlabel("N (North)")
ax.set_ylabel("E (East)")
ax.set_zlabel("Z (Down)")
ax.set_title("Vectors A and B in 3D")

# Show grid
ax.grid(True)

# Show axes
ax.quiver(0, 0, 0, 1, 0, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 1, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 0, 1, color="k", alpha=0.5)
```

```

# Add text with coordinates
ax.text(A[0], A[1], A[2], "A", color="r", fontsize=10)
ax.text(B[0], B[1], B[2], "B", color="g", fontsize=10)

plt.legend()
plt.show()

```

```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import tkinter as tk
from tkinter.simpledialog import askstring

def input_vectors():
    vectors = []
    while True:
        try:
            vector_str = askstring("Input 3D Vector", "Enter a 3D vector (e.g., 1, 2, 3) or 'esc' to finish:")
            if vector_str is None or vector_str.lower() == "esc":
                break
            vector = tuple(map(float, vector_str.split(",")))
            if len(vector) != 3:
                print("Invalid input. Please enter a 3D vector.")
            else:
                vectors.append(vector)
        except ValueError:
            print("Invalid input. Please enter numeric values separated by commas.")
    return vectors

# Input n 3D vectors
user_vectors = input_vectors()
print("User input vectors:", user_vectors)

# Convert user input vectors to numpy arrays
A = np.array(user_vectors[0])
B = np.array(user_vectors[1])

# Calculate the maximum magnitude of vectors A and B
max_magnitude = max(np.linalg.norm(A), np.linalg.norm(B))

# Plotting vectors A and B
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.quiver(0, 0, 0, A[0], A[1], A[2], color="r", label="A")
ax.quiver(0, 0, 0, B[0], B[1], B[2], color="g", label="B")

```

```

# Set labels and title
ax.set_xlabel("N (North)")
ax.set_ylabel("E (East)")
ax.set_zlabel("Z (Down)", rotation=90)
ax.set_title("User Input Vectors A and B in 3D")

# Set axis limits based on maximum magnitude
ax.set_xlim([-max_magnitude, max_magnitude])
ax.set_ylim([-max_magnitude, max_magnitude])
ax.set_zlim([-max_magnitude, max_magnitude])

# Show grid
ax.grid(True)

# Show axes
ax.quiver(0, 0, 0, 1, 0, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 1, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 0, 1, color="k", alpha=0.5)

# Add text with coordinates
ax.text(A[0], A[1], A[2], "A", color="r", fontsize=10)
ax.text(B[0], B[1], B[2], "B", color="g", fontsize=10)

plt.legend()
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import tkinter as tk
from tkinter.simpledialog import askstring

def input_vectors():
    vectors = []
    while True:
        try:
            vector_str = askstring("Input 3D Vector", "Enter a 3D vector (e.g., 1, 2, 3) or 'esc' to finish:")
            if vector_str is None or vector_str.lower() == "esc":
                break
            vector = tuple(map(float, vector_str.split(",")))
            if len(vector) != 3:
                print("Invalid input. Please enter a 3D vector.")
            else:
                vectors.append(vector)
        except ValueError:
            print("Invalid input. Please enter numeric values separated by commas.")
    return vectors

# Input n 3D vectors
user_vectors = input_vectors()

```

```

print("User input vectors:", user_vectors)

# Convert user input vectors to numpy arrays
A = np.array(user_vectors[0])
B = np.array(user_vectors[1])

# Calculate the maximum magnitude of vectors A and B
max_magnitude = max(np.linalg.norm(A), np.linalg.norm(B))

# Plotting vectors A and B
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.quiver(0, 0, 0, A[0], A[1], -A[2], color="r", label="A") # Reverse z-axis
ax.quiver(0, 0, 0, B[0], B[1], -B[2], color="g", label="B") # Reverse z-axis

# Set labels and title
ax.set_xlabel("E (East)") # Rotate x-axis label
ax.set_ylabel("N (North)") # Rotate y-axis label
ax.set_zlabel("Z (Up)", rotation=90) # Reverse z-axis label
ax.set_title("User Input Vectors A and B in 3D (NED Coordinate System)")

# Set axis limits based on maximum magnitude
ax.set_xlim([-max_magnitude, max_magnitude])
ax.set_ylim([-max_magnitude, max_magnitude])
ax.set_zlim([-max_magnitude, max_magnitude])
ax.invert_zaxis()

# Show grid
ax.grid(True)

# Show axes
ax.quiver(0, 0, 0, 1, 0, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 1, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 0, 1, color="k", alpha=0.5)

# Add text with coordinates
ax.text(A[0], A[1], -A[2], "A", color="r", fontsize=10)
ax.text(B[0], B[1], -B[2], "B", color="g", fontsize=10)

plt.legend()
plt.show()

```

第五次回應

```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import tkinter as tk
from tkinter.simpledialog import askstring

def input_vectors(input_type):

```

```

vectors = []
if input_type == 0:
    A = np.array([4, 5, 6])
    B = np.array([7, -5, -9])
    vectors.append(A)
    vectors.append(B)
else:
    while True:
        try:
            vector_str = askstring("Input 3D Vector", "Enter a 3D vector (e.g., 1, 2, 3) or 'esc' to finish:", initialvalue="4, 5, 6")
            if vector_str is None or vector_str.lower() == "esc":
                break
            vector = tuple(map(float, vector_str.split(",")))
            if len(vector) != 3:
                print("Invalid input. Please enter a 3D vector.")
            else:
                vectors.append(vector)
        except ValueError:
            print("Invalid input. Please enter numeric values separated by commas.")
return vectors

# Choose input type (0 for predefined vectors, 1 for user input)
input_type = int(askstring("Input input_type", "Enter input_type 0 for default (e.g., 0, 1, 2, 3) or 'esc' to finish:", initialvalue="0"))
user_vectors = input_vectors(input_type)
print("User input vectors:", user_vectors)

# Convert user input vectors to numpy arrays
A = np.array(user_vectors[0])
B = np.array(user_vectors[1])

# Calculate the maximum magnitude of vectors A and B
max_magnitude = max(np.linalg.norm(A), np.linalg.norm(B))

# Plotting vectors A and B
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.quiver(0, 0, 0, A[0], A[1], -A[2], color="r", label="A") # Reverse z-axis
ax.quiver(0, 0, 0, B[0], B[1], -B[2], color="g", label="B") # Reverse z-axis

# Set labels and title
ax.set_xlabel("y (East)") # Rotate x-axis label
ax.set_ylabel("x (North)") # Rotate y-axis label
ax.set_zlabel("z (Down)", rotation=90) # Reverse z-axis label
ax.set_title("User Input Vectors A and B in 3D (NED Coordinate System)")

# Set axis limits based on maximum magnitude
ax.set_xlim([-max_magnitude, max_magnitude])
ax.set_ylim([-max_magnitude, max_magnitude])
ax.set_zlim([-max_magnitude, max_magnitude])

```

```

ax.set_aspect('equal')
plt.tight_layout()
ax.invert_zaxis()
ax.set_xlabel('y(E)')
ax.set_ylabel('x(N)')
ax.set_zlabel('z(Dn)')
# Show grid
ax.grid(True)

# Show axes
ax.quiver(0, 0, 0, 1, 0, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 1, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 0, 1, color="k", alpha=0.5)

# Add text with coordinates
ax.text(A[0], A[1], A[2], f"A = {A}", color="r", fontsize=10)
ax.text(B[0], B[1], B[2], f"B = {B}", color="g", fontsize=10)

plt.legend()
plt.show()

```

第七次回應

經多次討論及修正後，最後總算完成。

```

def vector_add_NED():
    import numpy as np
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
    import tkinter as tk
    from tkinter.simpledialog import askstring

    def input_vectors(input_type):
        vectors = []
        if input_type == 0:
            A = np.array([1, 3, 5])
            B = np.array([3, 8, -9])
            vectors.append(A)
            vectors.append(B)
        else:
            while True:
                try:
                    vector_str = askstring("Input 3D Vector", "Enter a 3D vector (e.g., 1, 2, 3) or 'esc' to finish:", initialvalue="4, 5, 6")
                    if vector_str is None or vector_str.lower() == "esc":
                        break
                    vector = tuple(map(float, vector_str.split(", ")))
                    if len(vector) != 3:
                        print("Invalid input. Please enter a 3D vector.")
                    else:
                        vectors.append(vector)
                except ValueError:
                    print("Invalid input. Please enter numeric values separated by commas.")


    vectors = input_vectors(0)
    A = vectors[0]
    B = vectors[1]
    C = A + B
    D = -B
    E = A - B
    F = A + C
    G = B + C
    H = C + D
    I = C + E
    J = D + E
    K = D + F
    L = E + F
    M = E + G
    N = F + G
    O = F + H
    P = G + H
    Q = G + I
    R = H + I
    S = H + J
    T = I + J
    U = I + K
    V = J + K
    W = J + L
    X = K + L
    Y = K + M
    Z = L + M
    AA = M + N
    BB = N + O
    CC = O + P
    DD = P + Q
    EE = Q + R
    FF = R + S
    GG = S + T
    HH = T + U
    II = U + V
    JJ = V + W
    KK = W + X
    LL = X + Y
    MM = Y + AA
    NN = AA + BB
    OO = BB + CC
    PP = CC + DD
    QQ = DD + EE
    RR = EE + FF
    SS = FF + GG
    TT = GG + HH
    UU = HH + II
    VV = II + JJ
    WW = JJ + KK
    XX = KK + LL
    YY = LL + MM
    ZZ = MM + NN

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.set_aspect('equal')
    plt.tight_layout()
    ax.invert_zaxis()
    ax.set_xlabel('y(E)')
    ax.set_ylabel('x(N)')
    ax.set_zlabel('z(Dn)')
    # Show grid
    ax.grid(True)

    # Show axes
    ax.quiver(0, 0, 0, 1, 0, 0, color="k", alpha=0.5)
    ax.quiver(0, 0, 0, 0, 1, 0, color="k", alpha=0.5)
    ax.quiver(0, 0, 0, 0, 0, 1, color="k", alpha=0.5)

    # Add text with coordinates
    ax.text(A[0], A[1], A[2], f"A = {A}", color="r", fontsize=10)
    ax.text(B[0], B[1], B[2], f"B = {B}", color="g", fontsize=10)
    ax.text(C[0], C[1], C[2], f"C = {C}", color="b", fontsize=10)
    ax.text(D[0], D[1], D[2], f"D = {D}", color="m", fontsize=10)
    ax.text(E[0], E[1], E[2], f"E = {E}", color="c", fontsize=10)
    ax.text(F[0], F[1], F[2], f"F = {F}", color="y", fontsize=10)
    ax.text(G[0], G[1], G[2], f"G = {G}", color="p", fontsize=10)
    ax.text(H[0], H[1], H[2], f"H = {H}", color="v", fontsize=10)
    ax.text(I[0], I[1], I[2], f"I = {I}", color="n", fontsize=10)
    ax.text(J[0], J[1], J[2], f"J = {J}", color="w", fontsize=10)
    ax.text(K[0], K[1], K[2], f"K = {K}", color="o", fontsize=10)
    ax.text(L[0], L[1], L[2], f"K = {L}", color="k", fontsize=10)
    ax.text(M[0], M[1], M[2], f"K = {M}", color="k", fontsize=10)
    ax.text(N[0], N[1], N[2], f"K = {N}", color="k", fontsize=10)
    ax.text(O[0], O[1], O[2], f"K = {O}", color="k", fontsize=10)
    ax.text(P[0], P[1], P[2], f"K = {P}", color="k", fontsize=10)
    ax.text(Q[0], Q[1], Q[2], f"K = {Q}", color="k", fontsize=10)
    ax.text(R[0], R[1], R[2], f"K = {R}", color="k", fontsize=10)
    ax.text(S[0], S[1], S[2], f"K = {S}", color="k", fontsize=10)
    ax.text(T[0], T[1], T[2], f"K = {T}", color="k", fontsize=10)
    ax.text(U[0], U[1], U[2], f"K = {U}", color="k", fontsize=10)
    ax.text(V[0], V[1], V[2], f"K = {V}", color="k", fontsize=10)
    ax.text(W[0], W[1], W[2], f"K = {W}", color="k", fontsize=10)
    ax.text(X[0], X[1], X[2], f"K = {X}", color="k", fontsize=10)
    ax.text(Y[0], Y[1], Y[2], f"K = {Y}", color="k", fontsize=10)
    ax.text(Z[0], Z[1], Z[2], f"K = {Z}", color="k", fontsize=10)
    ax.text(AA[0], AA[1], AA[2], f"AA = {AA}", color="k", fontsize=10)
    ax.text(BB[0], BB[1], BB[2], f"BB = {BB}", color="k", fontsize=10)
    ax.text(CC[0], CC[1], CC[2], f"CC = {CC}", color="k", fontsize=10)
    ax.text(DD[0], DD[1], DD[2], f"DD = {DD}", color="k", fontsize=10)
    ax.text(EE[0], EE[1], EE[2], f"EE = {EE}", color="k", fontsize=10)
    ax.text(FF[0], FF[1], FF[2], f"FF = {FF}", color="k", fontsize=10)
    ax.text(GG[0], GG[1], GG[2], f"GG = {GG}", color="k", fontsize=10)
    ax.text(HH[0], HH[1], HH[2], f"HH = {HH}", color="k", fontsize=10)
    ax.text(II[0], II[1], II[2], f"II = {II}", color="k", fontsize=10)
    ax.text(JJ[0], JJ[1], JJ[2], f"JJ = {JJ}", color="k", fontsize=10)
    ax.text(KK[0], KK[1], KK[2], f"KK = {KK}", color="k", fontsize=10)
    ax.text(LL[0], LL[1], LL[2], f"LL = {LL}", color="k", fontsize=10)
    ax.text(MM[0], MM[1], MM[2], f"MM = {MM}", color="k", fontsize=10)
    ax.text(NN[0], NN[1], NN[2], f"NN = {NN}", color="k", fontsize=10)
    ax.text(PP[0], PP[1], PP[2], f"PP = {PP}", color="k", fontsize=10)
    ax.text(QQ[0], QQ[1], QQ[2], f"QQ = {QQ}", color="k", fontsize=10)
    ax.text(RR[0], RR[1], RR[2], f"RR = {RR}", color="k", fontsize=10)
    ax.text(SS[0], SS[1], SS[2], f"SS = {SS}", color="k", fontsize=10)
    ax.text(TT[0], TT[1], TT[2], f"TT = {TT}", color="k", fontsize=10)
    ax.text(UU[0], UU[1], UU[2], f"UU = {UU}", color="k", fontsize=10)
    ax.text(VV[0], VV[1], VV[2], f"VV = {VV}", color="k", fontsize=10)
    ax.text(WW[0], WW[1], WW[2], f"WW = {WW}", color="k", fontsize=10)
    ax.text(XX[0], XX[1], XX[2], f"XX = {XX}", color="k", fontsize=10)
    ax.text(YY[0], YY[1], YY[2], f"YY = {YY}", color="k", fontsize=10)
    ax.text(ZZ[0], ZZ[1], ZZ[2], f"ZZ = {ZZ}", color="k", fontsize=10)

    plt.legend()
    plt.show()

```

```

commas. ")
    return vectors

# Choose input type (0 for predefined vectors, 1 for user input)
input_type = int(askstring("Input input_type", "Enter input_type 0 for default
(e.g., 0,1,2,3) or 'esc' to finish:", initialvalue="0"))
user_vectors = input_vectors(input_type)
print("User input vectors:", user_vectors)

# Convert user input vectors to numpy arrays
A = np.array(user_vectors[0])
B = np.array(user_vectors[1])

# Calculate the maximum magnitude of vectors A and B
max_magnitude = max(np.linalg.norm(A), np.linalg.norm(B))
# Plotting vectors A and B
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.quiver(0, 0, 0, A[1], A[0], A[2], color="r", label="A") # Reverse x,y-axis z(Dn)
ax.quiver(0, 0, 0, B[1], B[0], B[2], color="g", label="B") # Reverse x,y-axis
z(Dn)
ax.scatter(0, 0, 0, marker='+', s=120, c='red')
# Set labels and title
ax.set_xlabel("y (East)") # Rotate x-axis label
ax.set_ylabel("x (North)") # Rotate y-axis label
ax.set_zlabel("z (Down)", rotation=90) # Reverse z-axis label
ax.set_title("NED Coordinate System")

# Set axis limits based on maximum magnitude
ax.set_xlim([-max_magnitude, max_magnitude])
ax.set_ylim([-max_magnitude, max_magnitude])
ax.set_zlim([-max_magnitude, max_magnitude])

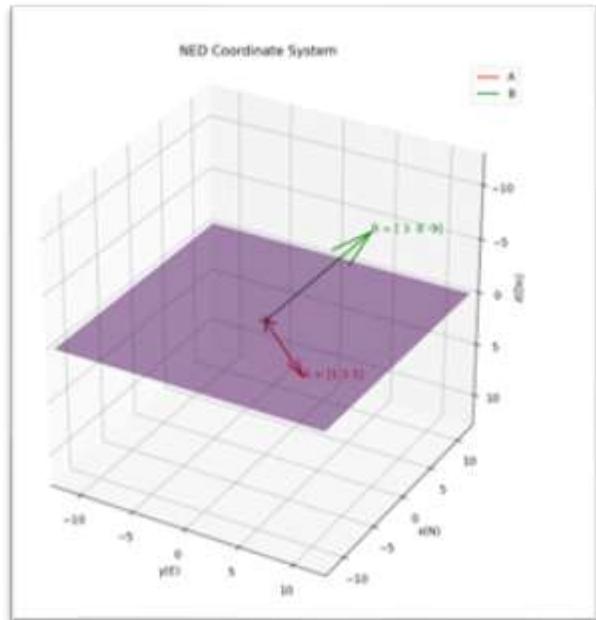
ax.set_aspect('equal')
plt.tight_layout()
ax.invert_zaxis()
ax.set_xlabel('y(E)')
ax.set_ylabel('x(N)')
ax.set_zlabel('z(Dn)')
# Show grid
ax.grid(True)

# Show axes
ax.quiver(0, 0, 0, 1, 0, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 1, 0, color="k", alpha=0.5)
ax.quiver(0, 0, 0, 0, 0, 1, color="k", alpha=0.5)

# Add text with coordinates
ax.text(A[1], A[0], A[2], f"A = {A}", color="r", fontsize=10)
ax.text(B[1], B[0], B[2], f"B = {B}", color="g", fontsize=10)
# Create a meshgrid for plotting the plane
x = np.linspace(-max_magnitude, max_magnitude, 100)
y = np.linspace(-max_magnitude, max_magnitude, 100)

```

```
X, Y = np.meshgrid(x, y)
Z = np.zeros_like(X) # Plane passes through (0, 0, 0)
ax.plot_surface(X, Y, Z, alpha=0.5, cmap="viridis")
plt.legend()
plt.show()
```



第九次修正

