# 如何請 AI 協助撰寫 Python 程式

# How to Ask AI to Help Write Python Programs

chday169(戴清河)

## 1. 問題提出方式

**中文：** 當我們需要撰寫 Python 程式時，可以先清楚描述需求，例如：「我要一個程式能解三元一次聯立方程式，並用 matplotlib 畫出三個平面的交點。」這樣 AI 才能理解並生成符合需求的程式。

**English:** When writing Python programs, start by clearly describing your requirements. For example: *"I need a program that solves a system of three linear equations and plots the intersection of three planes using matplotlib."* This helps the AI understand and generate code that matches your needs.

## 2. 解法原理

**中文：** 三元一次聯立方程式的解法可以用 **Cramer's Rule（克拉默法則）**。

- 先建立係數矩陣 A 與常數向量 b。
- 計算行列式 D=det(A)。
- 若 D≠0，則有唯一解：

$$x=\det(A_x)/D, y=\det(A_y)/D, z=\det(A_z)/D$$

- 幾何上，每個方程式代表一個平面，三個平面交於一點即為解。

**English:** The solution principle is **Cramer's Rule**.

- Build the coefficient matrix A and constant vector b.

- Compute the determinant D=det(A).
- If D≠0, the system has a unique solution:

x=det(Ax)D,y=det(Ay)D,z=det(Az)D

- Geometrically, each equation represents a plane. The intersection point of the three planes is the solution.

## 3. 偵測程式錯誤的方法

**中文：** 程式可能出現語法錯誤或邏輯錯誤。常見偵測方式：

- **語法錯誤 (Syntax Error)：** Python 會直接顯示錯誤訊息，例如 `NameError` 或 `IndentationError`。
- **邏輯錯誤 (Logic Error)：** 程式能執行但結果不正確，需要檢查數學公式或程式流程。
- **偵測方式：**
  1. 使用 `print()` 或 `logging` 追蹤變數值。
  2. 逐步測試小功能，確認每一部分正確。
  3. 依照錯誤訊息檢查程式碼縮排、函式定義、變數名稱。

**English:** Programs may have syntax or logic errors. Common detection methods:

- **Syntax Errors:** Python shows error messages like `NameError` or `IndentationError`.
- **Logic Errors:** The program runs but produces incorrect results. Check formulas or program flow.
- **Detection methods:**
  1. Use `print()` or `logging` to trace variable values.
  2. Test small parts step by step to confirm correctness.
  3. Follow error messages to check indentation, function definitions, and variable names.

## 4. 修正程式錯誤的方法

**中文：**

- **根據錯誤訊息修正：** 例如 NameError: name 'LinearSystemApp' is not defined，代表程式呼叫了尚未定義的類別，需檢查縮排或定義位置。
- **逐步測試：** 先確保數學函式正確，再加入 GUI 或繪圖功能。
- **請 AI 協助：** 把錯誤訊息與程式片段提供給 AI，讓 AI 分析並建議修正方式。

English:

- **Fix based on error messages:** For example, NameError: name 'LinearSystemApp' is not defined means the class was called before being defined. Check indentation or definition order.
- **Step-by-step testing:** Ensure math functions work correctly before adding GUI or plotting features.
- **Ask AI for help:** Provide the error message and code snippet to AI, so it can analyze and suggest corrections.

## 5. 結論

**中文：** AI 能快速生成程式，但人類需要理解數學原理與程式結構，才能檢查並修正錯誤。這是一種「人機協作」：AI 提供初稿，人類負責驗證與改進。

**English:** AI can quickly generate code, but humans must understand the math principles and program structure to check and fix errors. This is a form of "human – AI collaboration": AI provides the draft, humans validate and improve it.

```
import tkinter as tk

from tkinter import messagebox

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

import numpy as np



# ---------- Pure math (Cramer's rule) ----------
```

```
def det3(a11, a12, a13, a21, a22, a23, a31, a32, a33):
    return (a11*(a22*a33 - a23*a32)
            - a12*(a21*a33 - a23*a31)
            + a13*(a21*a32 - a22*a31))


def solve_3x3_cramers(A, b):
    a11, a12, a13 = A[0]
    a21, a22, a23 = A[1]
    a31, a32, a33 = A[2]

    D = det3(a11, a12, a13, a21, a22, a23, a31, a32, a33)
    if D == 0:
        Dx = det3(b[0], a12, a13, b[1], a22, a23, b[2], a32, a33)
        Dy = det3(a11, b[0], a13, a21, b[1], a23, a31, b[2], a33)
        Dz = det3(a11, a12, b[0], a21, a22, b[1], a31, a32, b[2])
        if Dx == 0 and Dy == 0 and Dz == 0:
            raise ValueError("System has infinitely many solutions.")
        else:
            raise ValueError("System has no unique solution.")

    Dx = det3(b[0], a12, a13, b[1], a22, a23, b[2], a32, a33)
```

```
    Dy = det3(a11, b[0], a13, a21, b[1], a23, a31, b[2], a33)
    Dz = det3(a11, a12, b[0], a21, a22, b[1], a31, a32, b[2])

    x = Dx / D
    y = Dy / D
    z = Dz / D
    return x, y, z, D

# ---------- GUI ----------
class LinearSystemApp:
    def __init__(self, root):
        self.root = root
        self.root.title("3x3 Linear System Solver (Cramer's Rule) + 3D Plot")

        self.canvas = tk.Canvas(root, width=800, height=420, bg="#fafafa")
        self.canvas.pack(padx=10, pady=10)

        header = ("請輸入三元一次聯立方程式的係數與常數項：\n"
                  "格式：a1*x + b1*y + c1*z = d1 ； a2*x + b2*y + c2*z = d2 ； a3*x + b3*y + c3*z = d3\n"
                  "輸入完成後先按 Check 檢查，確認無誤再按 Run 執行並繪圖。")
        self.canvas.create_text(20, 20, anchor="nw", text=header, font=("Microsoft JhengHei", 11))
```

```
        # Labels
        self.canvas.create_text(20, 70, anchor="nw", text="第 1 式：a1*x + b1*y + c1*z = d1",
font=("Microsoft JhengHei", 10))
        self.canvas.create_text(20, 150, anchor="nw", text="第 2 式：a2*x + b2*y + c2*z = d2",
font=("Microsoft JhengHei", 10))
        self.canvas.create_text(20, 230, anchor="nw", text="第 3 式：a3*x + b3*y + c3*z = d3",
font=("Microsoft JhengHei", 10))


        # Entry widgets
        self.entries = {}
        for k in ['a1','b1','c1','d1','a2','b2','c2','d2','a3','b3','c3','d3']:
            self.entries[k] = tk.Entry(root, width=8)


        # Place entries row 1
        self.canvas.create_window(220, 100, window=self.entries['a1'])
        self.canvas.create_text(265, 100, text="x +")
        self.canvas.create_window(310, 100, window=self.entries['b1'])
        self.canvas.create_text(355, 100, text="y +")
        self.canvas.create_window(400, 100, window=self.entries['c1'])
        self.canvas.create_text(445, 100, text="z =")
        self.canvas.create_window(490, 100, window=self.entries['d1'])
```

```
        # Row 2
        self.canvas.create_window(220, 180, window=self.entries['a2'])
        self.canvas.create_text(265, 180, text="x +")
        self.canvas.create_window(310, 180, window=self.entries['b2'])
        self.canvas.create_text(355, 180, text="y +")
        self.canvas.create_window(400, 180, window=self.entries['c2'])
        self.canvas.create_text(445, 180, text="z =")
        self.canvas.create_window(490, 180, window=self.entries['d2'])

        # Row 3
        self.canvas.create_window(220, 260, window=self.entries['a3'])
        self.canvas.create_text(265, 260, text="x +")
        self.canvas.create_window(310, 260, window=self.entries['b3'])
        self.canvas.create_text(355, 260, text="y +")
        self.canvas.create_window(400, 260, window=self.entries['c3'])
        self.canvas.create_text(445, 260, text="z =")
        self.canvas.create_window(490, 260, window=self.entries['d3'])

        # Buttons
        self.btn_check = tk.Button(root, text="Check", command=self.check_inputs, bg="#ffd966")
        self.btn_run = tk.Button(root, text="Run", command=self.run_solver, bg="#93c47d")
```

```
        self.canvas.create_window(220, 330, window=self.btn_check)
        self.canvas.create_window(300, 330, window=self.btn_run)

        # Output label
        self.output_text = tk.StringVar(value="狀態：尚未檢查")
        self.output_label = tk.Label(root, textvariable=self.output_text, font=("Microsoft JhengHei",
10), bg="#fafafa")
        self.canvas.create_window(220, 370, window=self.output_label)

    def read_coeffs(self):
        vals = {}
        for k, e in self.entries.items():
            s = e.get().strip()
            if s == "":
                raise ValueError(f"欄位 {k} 未填寫。")
            try:
                vals[k] = float(s)
            except ValueError:
                raise ValueError(f"欄位 {k} 不是有效數字：{s}")
        A = [
            [vals['a1'], vals['b1'], vals['c1']],
            [vals['a2'], vals['b2'], vals['c2']],
```

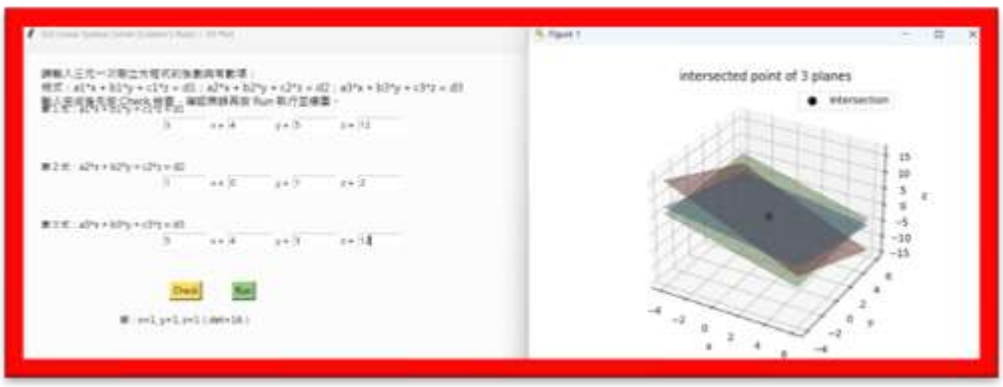
```
            [vals['a3'], vals['b3'], vals['c3']]
        ]
        b = [vals['d1'], vals['d2'], vals['d3']]
        return A, b

def check_inputs(self):
        try:
            A, b = self.read_coeffs()
            for i, row in enumerate(A, start=1):
                if all(v == 0 for v in row):
                    raise ValueError(f"第 {i} 式的係數全為 0。")
            self.output_text.set("狀態：檢查完成，可按 Run 執行。")
            messagebox.showinfo("Check", "輸入檢查完成，未發現錯誤。")
        except Exception as e:
            self.output_text.set(f"狀態：檢查失敗 - {e}")
            messagebox.showerror("Check Error", str(e))

def run_solver(self):
        try:
            A, b = self.read_coeffs()
            x, y, z, D = solve_3x3_cramers(A, b)
```

```
            self.output_text.set(f"解：x={x:.6g}, y={y:.6g}, z={z:.6g}（det={D:.6g}）")
            self.plot_planes_and_point(A, b, (x, y, z))
        except Exception as e:
            self.output_text.set(f"狀態：執行失敗 - {e}")
            messagebox.showerror("Run Error", str(e))

def plot_planes_and_point(self, A, b, sol):
    x, y, z = sol
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.set_title("intersected point of 3 planes")

    X = np.linspace(x-5, x+5, 20)
    Y = np.linspace(y-5, y+5, 20)

    X, Y = np.meshgrid(X, Y)

    colors = ['#6fa8dc', '#93c47d', '#e06666']
    labels = ['Plane 1', 'Plane 2', 'Plane 3']

    for i, (row, di) in enumerate(zip(A, b)):
```

```
        a, bb, c = row
        if c != 0:
            Z = (di - a * X - bb * Y) / c
            ax.plot_surface(X, Y, Z, alpha=0.5, color=colors[i])
        else:
            # Handle vertical planes (c=0)
            if a != 0:
                Xp = (di - bb * Y) / a
                Zp = np.linspace(z - 5, z + 5, 20)
                Zp = np.tile(Zp, (Y.shape[0], 1))
                ax.plot_surface(Xp, Y, Zp, alpha=0.5, color=colors[i])
            elif bb != 0:
                Yp = (di - a * X) / bb
                Zp = np.linspace(z - 5, z + 5, 20)
                Zp = np.tile(Zp, (X.shape[0], 1))
                ax.plot_surface(X, Yp, Zp, alpha=0.5, color=colors[i])

# Plot intersection point
ax.scatter([x], [y], [z], color='k', s=60, label='Intersection')

ax.set_xlabel('x')
```

```
            ax.set_ylabel('y')

            ax.set_zlabel('z')

            ax.legend()

            plt.show()



def main():

    root = tk.Tk()

    app = LinearSystemApp(root)

    root.mainloop()



if __name__ == "__main__":

    main()
```

Python codes




Results and figure