Chắc chắn rồi! Dưới đây là một chương trình Python hoàn chỉnh sử dụng các thư viện pyserial, matplotlib, và numpy để vẽ đồ thị thời gian thực từ dữ liệu trên cổng COM theo đúng cấu trúc khung truyền bạn đã mô tả.

Chương trình được thiết kế để dễ dàng cấu hình và có cả **chế độ mô phỏng** để bạn có thể chạy thử ngay cả khi chưa có thiết bị phần cứng kết nối.

Bước 1: Cài đặt các thư viện cần thiết

Trước tiên, bạn cần cài đặt các thư viện nếu chưa có. Mở Terminal hoặc Command Prompt và chạy lệnh sau:

Bash

```
pip install pyserial matplotlib numpy
```

Bước 2: Mã nguồn chương trình Python

Hãy sao chép đoạn mã dưới đây và lưu vào một tệp tin, ví dụ: realtime_plotter.py.

Python

```
import serial
import struct
import time
import collections
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
```

```
# --- CẤU HÌNH ---
# Chế độ mô phỏng: True để dùng dữ liệu giả, False để đọc từ cổng COM thật
SIMULATION_MODE = True

SERIAL_PORT = 'COM3'  # Thay đổi thành cổng COM của bạn (ví dụ: 'COM3' trên Windows, '/dev/ttyUSB0' trên Linux)
BAUD_RATE = 115200

# Cấu trúc khung truyền
FRAME_START_BYTE = 0x03
FRAME_END_BYTE = 0xFC
FRAME_LENGTH = 22  # 1 start + 5 * 4 bytes data + 1 end
NUM_DATA_POINTS = 5

# Cấu hình đồ thị
PLOT_WINDOW_SIZE = 100  # Số điểm dữ liệu hiển thị trên đồ thị
PLOT_UPDATE_INTERVAL = 20  # Thời gian cập nhật đồ thị (ms)
# --- KẾT THÚC CẤU HÌNH ---

# Khởi tạo bộ đệm (deque) để lưu trữ dữ liệu cho 5 kênh
# deque là một cấu trúc dữ liệu hiệu quả để thêm/xóa phần tử ở cả hai đầu
data_channels = [collections.deque(np.zeros(PLOT_WINDOW_SIZE), maxlen=PLOT_WINDOW_SIZE) for _ in range(NUM_DATA_POINTS)]

# Khởi tạo đối tượng serial
ser = None
if not SIMULATION_MODE:
    try:
        ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
        print(f"Đã kết nối thành công đến cổng {SERIAL_PORT} với baudrate {BAUD_RATE}")
    except serial.SerialException as e:
        print(f"Lỗi: Không thể mở cổng {SERIAL_PORT}. {e}")
        print("Chương trình sẽ thoát.")
        exit()

def read_and_parse_data():
    """
    Hàm đọc và phân tích một khung dữ liệu từ cổng COM.
    """
    if not ser:
        return None

    # Vòng lặp để đồng bộ hóa, tìm byte bắt đầu
```

```
    while True:
        start_byte = ser.read(1)
        if not start_byte:  # Timeout
            print("Chờ dữ liệu...")
            return None
        if start_byte[0] == FRAME_START_BYTE:
            break

    # Đọc phần còn lại của khung (21 bytes)
    frame_data = ser.read(FRAME_LENGTH - 1)

    if len(frame_data) < (FRAME_LENGTH - 1):
        print("Khung dữ liệu không đủ độ dài.")
        return None

    # Kiểm tra byte kết thúc
    if frame_data[-1] != FRAME_END_BYTE:
        print("Lỗi byte kết thúc.")
        return None

    # Giải nén dữ liệu
    # '<'    : Little-endian (byte có trọng số thấp trước)
    # 'i'    : Integer 32-bit có dấu (signed)
    # '5i'   : 5 số integer liên tiếp
    # Dữ liệu nằm từ byte 0 đến byte 19 của frame_data (tổng 20 bytes)
    try:
        unpacked_data = struct.unpack('<5i', frame_data[0:20])
        return list(unpacked_data)
    except struct.error as e:
        print(f"Lỗi giải nén dữ liệu: {e}")
        return None

def generate_simulation_data(time_step):
    """
    Hàm tạo dữ liệu giả để mô phỏng.
    """
    # Tạo dữ liệu dạng sóng sin với các tần số và biên độ khác nhau
    d1 = 100 * np.sin(2 * np.pi * 0.1 * time_step) + np.random.randn() * 2
    d2 = 75 * np.cos(2 * np.pi * 0.25 * time_step) + 50
    d3 = 50 * np.sin(2 * np.pi * 0.5 * time_step) + np.random.randn() * 5
    d4 = 150 * (time_step % 50) / 50.0 # Sóng răng cưa
    d5 = np.random.randint(-100, 100) # Nhiễu ngẫu nhiên
    return [d1, d2, d3, d4, d5]
```

```
# Thiết lập đồ thị
fig, ax = plt.subplots(figsize=(12, 7))
lines = [ax.plot([], [], lw=2, label=f'Dữ liệu {i+1}')[0] for i in range(NUM_DATA_POINTS)]
ax.set_title('Đồ thị dữ liệu thời gian thực từ cổng COM')
ax.set_xlabel('Mẫu')
ax.set_ylabel('Giá trị')
ax.legend()
ax.grid(True)
ax.set_xlim(0, PLOT_WINDOW_SIZE)
# Đặt giới hạn y ban đầu, sẽ được tự động điều chỉnh
ax.set_ylim(-200, 200)

# Biến đếm thời gian cho chế độ mô phỏng
simulation_time_step = 0

def update_plot(frame):
    """
    Hàm cập nhật đồ thị, được gọi lặp đi lặp lại bởi FuncAnimation.
    """
    global simulation_time_step

    new_data = None
    if SIMULATION_MODE:
        new_data = generate_simulation_data(simulation_time_step)
        simulation_time_step += 0.1 # Tăng bước thời gian
    else:
        new_data = read_and_parse_data()

    if new_data and len(new_data) == NUM_DATA_POINTS:
        # Cập nhật dữ liệu cho từng kênh
        for i in range(NUM_DATA_POINTS):
            data_channels[i].append(new_data[i])
            lines[i].set_data(range(PLOT_WINDOW_SIZE), data_channels[i])

        # Tự động điều chỉnh giới hạn trục y để hiển thị rõ ràng
        ax.relim()
        ax.autoscale_view(scalex=False, scaley=True)

    return lines

# Tạo animation
# blit=True giúp tối ưu hóa việc vẽ lại, chỉ vẽ những phần thay đổi.
# cache_frame_data=False để tránh rò rỉ bộ nhớ.
```

```
ani = FuncAnimation(
    fig,
    update_plot,
    interval=PLOT_UPDATE_INTERVAL,
    blit=True,
    cache_frame_data=False
)

# Hiển thị đồ thị
plt.show()

# Đóng cổng COM khi cửa sổ đồ thị được đóng
if ser and ser.is_open:
    ser.close()
    print(f"Đã đóng cổng {SERIAL_PORT}.")
```

Bước 3: Hướng dẫn sử dụng và Giải thích mã

Cách sử dụng

1. **Chạy ở chế độ mô phỏng:**
   - Giữ nguyên SIMULATION_MODE = True.
   - Chạy tệp tin bằng lệnh: python realtime_plotter.py
   - Một cửa sổ đồ thị sẽ hiện ra và vẽ 5 đường dữ liệu được tạo giả. Đây là cách tuyệt vời để kiểm tra giao diện đồ thị mà không cần phần cứng.
2. **Chạy với thiết bị thật:**
   - Kết nối thiết bị của bạn vào máy tính.
   - Xác định đúng tên cổng COM.
     - **Trên Windows:** Mở *Device Manager*, tìm trong mục *Ports (COM & LPT)*. Tên thường là COM theo sau là một con số (ví dụ: COM3, COM4).
     - **Trên Linux/macOS:** Mở terminal và gõ ls /dev/tty.*. Tên thường là /dev/ttyUSB0 hoặc /dev/tty.usbmodem....
   - Sửa đổi dòng SERIAL_PORT = 'COM3' thành cổng COM chính xác của bạn.
   - Đảm bảo BAUD_RATE trong mã khớp với baudrate trên thiết bị của bạn.
   - Chuyển SIMULATION_MODE thành False: SIMULATION_MODE = False.

- Chạy lại tệp tin: python realtime_plotter.py.

Giải thích các phần quan trọng của mã

- **collections.deque(..., maxlen=PLOT_WINDOW_SIZE)**: Chúng ta dùng deque với độ dài tối đa (maxlen). Khi một phần tử mới được thêm vào (append), nếu deque đã đầy, phần tử cũ nhất ở đầu kia sẽ tự động bị xóa. Điều này tạo ra hiệu ứng "cửa sổ trượt" cho đồ thị một cách hiệu quả.
- **read_and_parse_data()**: Đây là hàm cốt lõi để xử lý dữ liệu từ cổng COM.
  1. **Đồng bộ hóa**: Nó đọc từng byte một cho đến khi tìm thấy FRAME_START_BYTE (0x03). Việc này đảm bảo rằng chương trình luôn bắt đầu đọc từ đầu một khung truyền hoàn chỉnh, ngay cả khi có dữ liệu rác hoặc bị mất gói tin.
  2. **Kiểm tra**: Sau khi đọc đủ 22 byte, nó kiểm tra xem byte cuối cùng có phải là FRAME_END_BYTE (0xFC) hay không. Nếu không, khung dữ liệu sẽ bị loại bỏ.
  3. **struct.unpack('<5i', ...)**: Đây là lệnh "thần kỳ" để giải mã.
     - '<': Chỉ định rằng dữ liệu được sắp xếp theo kiểu **Little-Endian** (byte có trọng số thấp nhất được truyền trước), đúng như mô tả của bạn (byte 1 là bit 7-0).
     - '5i': Chỉ định rằng chúng ta muốn giải nén **5** số nguyên **32-bit có dấu** (i).
     - Lệnh này lấy 20 byte dữ liệu (5 kênh x 4 byte) và chuyển đổi chúng thành một tuple chứa 5 số nguyên.
- **FuncAnimation**: Đây là công cụ của matplotlib để tạo hoạt ảnh. Nó sẽ liên tục gọi hàm update_plot sau mỗi PLOT_UPDATE_INTERVAL mili giây.
  - **update_plot()**: Hàm này lấy dữ liệu mới (thật hoặc mô phỏng), cập nhật các deque, sau đó cập nhật dữ liệu cho các đường kẻ trên đồ thị bằng lines[i].set_data().
  - **ax.relim() và ax.autoscale_view()**: Hai lệnh này phối hợp để tự động điều chỉnh lại giới hạn của trục Y, đảm bảo đồ thị không bị "tràn" ra ngoài khung nhìn khi giá trị dữ liệu thay đổi lớn.

Chúc bạn thành công với dự án của mình!