**Triển khai Switching Linear Dynamical System: Kết hợp HMM và Kalman Filter để mô hình hóa hệ thống động học đa chế độ**

\*Github: <https://github.com/waanuu/TH2>

# Khám phá dữ liệu:

* + Slds\_data.csv chứa các cột như:
* Time: Chỉ số thời gian
* Mode: Chế độ ẩn
* X\_Position: Vị trí X
* Y\_Posittion: Vị trí Y
* VX: Vận tốc X
* VY: Vận tốc Y

# Mô hình HMM

**- Ý tưởng:**

Hệ thống chuyển động này **có nhiều chế độ hoạt động khác nhau** — ví dụ:

**Mode 0**: Đứng yên / chuyển động chậm / quay đầu

**Mode 1**: Chuyển động thẳng nhanh / đi đường chính

Nhưng trong thực tế, bạn **không quan sát được trực tiếp chế độ** đó. Thay vào đó, bạn chỉ đo được **vận tốc**, và từ đó **suy ra chế độ**.

**- Cách hoạt động :**

### ****+) Ẩn: Chế độ chuyển động****

Là "trạng thái ẩn" trong HMM: Mode = 0 hoặc Mode = 1

Bạn không biết mode thật tại từng thời điểm, phải suy luận

### ****+) Quan sát: Vận tốc****

Là dữ liệu quan sát (VX, VY)

HMM giả định rằng **mỗi mode sẽ tạo ra vận tốc theo phân phối Gaussian riêng**

### ****+ Học từ dữ liệu****

HMM sẽ học được:

**Phân phối vận tốc** đặc trưng cho mỗi mode

**Xác suất chuyển đổi** giữa các mode

**Xác suất ban đầu** của mỗi mode

n\_components = 2

hmm = GaussianHMM(n\_components=n\_components, covariance\_type="full", n\_iter=100, random\_state=42)

hmm.fit(velocity\_data)

predicted\_modes = hmm.predict(velocity\_data)

print("🔁 Ma trận chuyển trạng thái giữa các chế độ (A):")

print(hmm.transmat\_)

print("\n🎯 Xác suất ban đầu của các chế độ (π):")

print(hmm.startprob\_)

print("\n📈 Tham số phát xạ Gaussian cho mỗi chế độ:")

for i in range(hmm.n\_components):

    print(f"  ↪ Chế độ {i}:")

    print(f"     Trung bình: {hmm.means\_[i]}")

# 

# Mô hình **Kalman Filter**

# **- Ý tưởng:**

# Mục tiêu chính là để ước lượng các trạng thái liên tục (như vị trí) của hệ thống từ một chuỗi các quan sát bị nhiễu. Việc sử dụng **mô hình Kalman Filter** giúp loại bỏ phần nào sự nhiễu và dự đoán trạng thái của đối tượng trong tương lai.

#### ****- Các bước chính của mô hình:****

1. **Xử lý theo chế độ (Mode):** Đoạn mã chia dữ liệu thành n\_components chế độ khác nhau, và áp dụng Kalman Filter riêng biệt cho mỗi chế độ.

**2.Tạo mô hình Kalman Filter cho từng chế độ:**

+ Nếu chế độ có đủ quan sát (≥ 2 quan sát), một bộ lọc Kalman sẽ được khởi tạo.

+ Mỗi bộ lọc Kalman sử dụng **ma trận chuyển tiếp** và **ma trận quan sát** để dự đoán trạng thái của hệ thống.

**3.Ứng dụng Bộ Lọc Kalman:** Dự đoán trạng thái của hệ thống theo từng chế độ, đồng thời sử dụng các quan sát thực tế (true\_position) để tinh chỉnh các dự đoán.

**4.Fallback (Khi không thể áp dụng Kalman Filter):** Nếu không thể sử dụng bộ lọc Kalman (ví dụ: không đủ dữ liệu quan sát), thì kết quả dự đoán (filtered\_states) sẽ được gán bằng giá trị thực tế ban đầu (true\_position).

**- Cách hoạt động :**

+ K**hởi tạo mô hình Kalman cho từng chế độ:**

**Transition Matrices (**A**)**: Xác định cách các trạng thái thay đổi từ thời điểm này sang thời điểm tiếp theo (ở đây là một ma trận đơn giản với giá trị [1, 0], [0, 1] cho vị trí và vận tốc).

**Observation Matrices (**H**)**: Xác định mối quan hệ giữa trạng thái và quan sát (ở đây nó cũng được thiết lập là [1, 0], [0, 1]).

**Transition Covariance (**Q**)**: Đại diện cho mức độ nhiễu của quá trình hệ thống.

**Observation Covariance (**R**)**: Đại diện cho mức độ nhiễu của các quan sát.

**Initial State and Covariance**: Khởi tạo với giá trị đầu tiên trong quan sát (obs[0]), và giả sử phương sai là một ma trận đơn vị (np.eye(2)).

+ **Ứng dụng Kalman Filter cho mỗi chế độ:**

**Dự đoán trạng thái** từ bộ lọc Kalman cho mỗi chế độ riêng biệt.

**Tính toán giá trị dự đoán** (filtered states) bằng cách lọc qua các quan sát và dự đoán trạng thái mới cho từng phần tử trong true\_position.

+ **Fallback:**

Nếu bộ lọc Kalman không thể áp dụng (do thiếu quan sát hoặc lỗi trong dữ liệu), thì không áp dụng lọc và giữ nguyên giá trị ban đầu (true\_position).

+ **Kết quả cuối cùng:**

Mỗi chế độ có thể có một mô hình Kalman riêng biệt, và sau khi áp dụng lọc, các trạng thái được lọc sẽ được lưu vào filtered\_states. Nếu không có mô hình Kalman hợp lệ, filtered\_states sẽ được giữ nguyên từ true\_position.

kf\_models = []

for mode in range(n\_components):

    idx = predicted\_modes == mode

    obs = true\_position[idx]

    if len(obs) < 2:

        kf\_models.append(None)

        continue

    kf = KalmanFilter(

        transition\_matrices=[[1, 0], [0, 1]],

        observation\_matrices=[[1, 0], [0, 1]],

        transition\_covariance=1e-4 \* np.eye(2),

        observation\_covariance=np.cov(obs.T),

        initial\_state\_mean=obs[0],

        initial\_state\_covariance=np.eye(2)

    )

    kf = kf.em(obs, n\_iter=10)

    kf\_models.append(kf)

# Dự đoán trạng thái liên tục theo từng chế độ

filtered\_states = np.zeros\_like(true\_position)

for mode in range(n\_components):

    kf = kf\_models[mode]

    idx = predicted\_modes == mode

    if kf and np.sum(idx) > 1:

        obs = true\_position[idx]

        state\_means, \_ = kf.filter(obs)

        filtered\_states[idx] = state\_means

    else:

        filtered\_states[idx] = true\_position[idx]  # fallback

1. **Kết quả**



