Desigur! Mai jos este documentația completă pentru scriptul MATLAB actualizat (`test1.m`) care conține animațiile brațului robotic cu 3 traiectorii: normală, optimizată prin Least Squares și optimizată cu metoda DFP (Quasi-Newton).

📄 Documentație și Explicații pentru Scriptul `test1.m`

🎯 Scopul proiectului

Acest script MATLAB simulează un braț robotic plan cu două segmente și creează **3 animații** diferite ale mișcării efectorului final pe traiectorii:

1. **Traiectorie normală** – generată manual, fără analiză matematică.
2. **Traiectorie optimizată (Least Squares)** – netezită matematic pe baza unor date discrete.
3. **Traiectorie optimizată (DFP)** – obținută prin metoda Davidon-Fletcher-Powell (Quasi-Newton), minimizând variațiile de unghi (mișcare realistă și fluidă).

⚙️ Componentele scriptului

1. Parametrii inițiali

```
L1 = 5; % Lungimea primului segment
L2 = 3; % Lungimea celui de-al doilea segment
```

Definim lungimile celor două segmente ale brațului robotic.

---

2. Traiectorie normală

```
x_norm = linspace(2, 6, 100);
y_norm = linspace(3, 5, 100);
```

Aceasta este o traiectorie simplă, interpolată liniar, fără optimizare.

---

3. Traiectorie optimizată cu Least Squares

x_data = [2, 3, 4, 5, 6]';
y_data = [3, 3.5, 4.1, 4.6, 5]';

- Se aplică metoda Least Squares pentru a obține o relație liniară `y = a*x + b` care aproximează cât mai bine punctele discrete.
- Această traiectorie este **mai netedă**, dar nu ține cont de unghiurile articulațiilor.

---

4. Traiectorie optimizată cu DFP (Quasi-Newton)

Această parte este cheia optimizării cinematice:

fun = @(v) cost_variatie_unghiuri(v, L1, L2);
sol = fminunc(fun, x0, options);

🔍 **Funcția de cost:**
J = sum(diff(theta).^2);

- Obiectivul este **minimizarea variațiilor unghiului θ_1** între pași succesivi.
- Aceasta reduce mișcările bruște ale articulației — **traiectorie mai realistă**.

🧠 **Metoda DFP:**

- Este o metodă **Quasi-Newton**, care optimizează o funcție neliniară fără a calcula direct Hessiana.
- Se construiește o aproximare a inversei Hessienei, actualizată iterativ:
  Hk+1=Hk+corecțiiH_{k+1} = H_k + \text{corecții}
- Este eficientă și stabilă pentru funcții neliniare, cum este cazul variației unghiurilor într-un braț robotic.

---

5. Funcția de animație

Funcția `animeaza_brat()` construiește în timp real pozițiile celor două segmente ale brațului pentru fiecare punct al traiectoriei, folosind **cinematică inversă**:

theta2 = atan2(s2, c2);
theta1 = atan2(ye, xe) - atan2(k2, k1);

---

6. Rularea animațiilor

```
animeaza_brat(x_norm, y_norm, L1, L2, 'Traiectorie Cinematica Inversa (normală)');
animeaza_brat(x_opt1, y_opt1, L1, L2, 'Traiectorie Optimizată (Least Squares)');
animeaza_brat(x_opt2, y_opt2, L1, L2, 'Traiectorie Optimizată (DFP - Quasi-Newton)');
```

Fiecare animație este rulată separat cu titlu explicativ.

📊 Comparație între traiectorii

Traiectorie	Generare	Avantaje	Limitări
Normală	Liniar cu `linspace`	Simplu de implementat	Poate fi nerealistă, sacadată
Least Squares	Regresie liniară	Netedă, rapidă, bună pentru date	Nu ține cont de unghiuri
DFP (Quasi-Newton)	Optimizare variație unghiuri	Realistă, fluidă, adaptabilă	Mai complexă computațional

---

🧪 Posibile extinderi

- Adăugare de obstacole și optimizare cu penalizare
- Integrarea cu metode BFGS sau PSO
- Generare de traiectorii în 3D
- Export animații ca fișiere video
