Optimisation de la trajectoire d'un drone

MTH8408

Jouglet Nicolas, Dawut Esse, Joey Van Melle

Le contrôle de drones quadrotors est un enjeu majeur dans le domaine de la robotique autonome en raison de leur grande maniabilité, mais aussi de leur instabilité naturelle.
Ce projet vise à concevoir un **contrôleur optimal** capable de suivre une trajectoire prédéfinie tout en minimisant la consommation d'énergie.
Nous nous appuyons pour cela sur l'approche proposée par Suicmez et Kutay (2014), qui repose sur la commande optimale en temps discrèt.

Contexte du papier

- Les **quadrotors** sont des drones à 4 moteurs capables de **décoller et atterrir verticalement (VTOL)** et d'exécuter des **manœuvres agiles**.
- Ce sont des systèmes **fortement non linéaires et instables**.
- Leur principale limitation : **une consommation énergétique élevée**.

Description de la problèmatique

Le papier traité modélise un **drone quadrotor** et conçoit une commande optimale permettant de minimiser une fonction objectif quadratique prenant en compte différent critères tels que la précision du suivi d'une trajectoire, la consommation d'énergie et la position finale.

Afin d'obtenir la commande optimale, les auteurs du papier ont utilisé la méthode récursive de Riccati pour résoudre le problème d'optimisation quadratique en temps discret.

Cependant d'autres méthodes peuvent être utilisées pour determiner la commande optimale, la méthode recursive de Riccati est-elle la plus efficace ?

Description des objectifs et du plan d'action

La première objectif du projet consiste à **reproduire le modèle dynamique du drone** puis d'y appliquer **l'algorithme reccursif de Riccati** afin de reproduire les résultats obtenu dans le papier.
Ensuite, nous essaierons de résoudre le système avec **d'autres méthodes**, chaque méthode devra minimiser la fonction objectif. Les différentes méthodes utilisées sont :
- la résolution monolithique via IPOPT (optimisation non linéaire générique),
- la résolution avec COSMO (optimisation quadratique conique),
- et la résolution avec OSQP (optimisation quadratique à contraintes linéaires).

Cette comparaison permettra de mettre en évidence les avantages et inconvénients de chaque méthode pour le suivi optimal de trajectoire d'un drone quadrotor tout en déterminant laquelle est la plus efficace pour cette fonction objectif. Nous allons aussi tester une méthode spécifique aux problèmes quadratiques présentée en cours, mais nous n'avons pas encore atteint cet objectif. Nous espéront y arriver d'ici la phase 3.

Modélisation du système

Modèle dynamique non linéaire

Dans un premier temps, le système est modélisé de façon dynamique à partir des équations de Newton :

- **Dynamique translationnelle** : influencée par la poussée verticale U_1 et les angles ϕ, θ ;
- **Dynamique rotationnelle** : influencée par les couples de commande U_2, U_3, U_4.

Le modèle dynamique est constitué de deux parties :

- Équation non linéaire pour le mouvement de translation :

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + LEB \begin{bmatrix} 0 \\ 0 \\ U_1/m \end{bmatrix} - \left(\frac{K_t}{m} \right) \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{1}$$

Avec

$$LEB = \begin{bmatrix} \cos(\theta)\cos(\phi) & \sin(\theta)\sin(\phi)\cos(\psi) - \cos(\theta)\sin(\psi) & \cos(\theta)\sin(\phi)\cos(\psi) + \sin(\theta)\sin(\psi) \\ \cos(\theta)\sin(\phi) & \sin(\theta)\sin(\phi)\sin(\psi) + \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\phi)\sin(\psi) - \sin(\theta)\cos(\psi) \\ -\sin(\theta) & \sin(\theta)\cos(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}$$

- Équations non linéaires pour le mouvement de rotation :

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{(I_y - I_z)}{I_x} qr \\ \frac{(I_z - I_x)}{I_y} pr \\ \frac{(I_x - I_y)}{I_z} pq \end{bmatrix} + \begin{bmatrix} \frac{U_2 d}{I_x} \\ \frac{U_3 d}{I_y} \\ \frac{U_4}{I_z} \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)/\cos(\theta) & \cos(\phi)/\cos(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3}$$

Modèle dynamique non linéaire simplifié

Si les perturbations par rapport à la condition de vol stationnaire sont faibles, on suppose que les vitesses angulaires du corps [p, q, r] et les dérivées des angles d'Euler [$\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$] peuvent être considérées comme égales.
On à alors :

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad ; \quad \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} \tag{4}$$

En utilisant l'équation 4 dans l'équation 2, on obtient **l'équation d'Euler** :

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{(I_y - I_z)}{I_x} \dot{\theta}\dot{\psi} \\ \frac{(I_z - I_x)}{I_y} \dot{\phi}\dot{\psi} \\ \frac{(I_x - I_y)}{I_z} \dot{\phi}\dot{\theta} \end{bmatrix} + \begin{bmatrix} \frac{U_2 d}{I_x} \\ \frac{U_3 d}{I_y} \\ \frac{U_4}{I_z} \end{bmatrix} \tag{5}$$

Linéarisation du modèle dynamique simplifié

Pour utiliser l'algorithme LQT en temps discret, il est nécessaire de linéariser le modèle autour d'une condition d'équilibre (vol stationnaire).

À ce point d'équilibre :

$$\phi = \dot{\phi} = \theta = \dot{\theta} = \psi = \dot{\psi} = x = \dot{x} = y = \dot{y} = \dot{z} = 0,$$
$$z = 1 \text{ mètre.}$$

En linéarisant LEB grâce aux hypothèses du vol stationnaire on obtient :

$$L_{EB}^{\text{lin}} \approx I_3$$

L'équation de translation (1) devient alors :

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + I_3 \begin{bmatrix} 0 \\ 0 \\ U_1/m \end{bmatrix} \tag{6}$$

De même en vol stationnaire $\dot{\psi} = \dot{\phi} = \dot{\theta} = 0$, donc l'équation d'Euler peut s'écrire :

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{U_2 d}{I_x} \\ \frac{U_3 d}{I_y} \\ \frac{U_4}{I_z} \end{bmatrix} \tag{7}$$

Le modèle peut donc se résumer à :

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + I_3 \begin{bmatrix} 0 \\ 0 \\ U_1/m \end{bmatrix} \quad ; \quad \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{U_2 d}{I_x} \\ \frac{U_3 d}{I_y} \\ \frac{U_4}{I_z} \end{bmatrix} \tag{1}$$

Afin de linéariser le modèle, on introduit X le vecteur d'état, Y le vecteur de sortie et le vecteur de commande U tels que :

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \\ x \\ \dot{x} \\ y \\ \dot{y} \\ z \\ \dot{z} \end{bmatrix} \quad ; \quad Y = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} \quad ; \quad U = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{6}$$

Enfin grâce aux hypothèses de vol stationnaire on obtient le modèle linèaire suivant :

$$\dot{X} = AX + BU \quad ; \quad Y = CX \tag{7}$$

Où A et B et C valent :

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{A}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \end{bmatrix} \tag{B}$$

$$C = I_{12} \tag{C}$$

Fonction Performance

La fonction performance est construite de la façon suivante :

$$J = \frac{1}{2} \sum_{k=k_0}^{k_f-1} \left(\|C_d X(k) - r(k)\|_Q^2 + \|U(k)\|_R^2 \right) + \frac{1}{2} \|C_d X(k_f) - r(k_f)\|_F^2 \tag{8}$$

où $\|x\|_M^2 = x^\top M x$ est la norme quadratique pondérée par la matrice M, et F est la matrice de coût terminal, souvent choisie égale à Q, donc $F = Q$.

La structure de la fonction J correspond à un problème d'optimisation quadratique, tel qu'étudié dans le cadre du cours sur l'optimisation sans contrainte. En effet, il s'agit de minimiser une somme pondérée d'erreurs quadratiques entre la trajectoire suivie $C_d X(k)$ et la trajectoire de référence $r(k)$, ainsi que des efforts de commande $U(k)$ et la différence de position finale.

La fonction J est une fonction quadratique strictement convexe, car les matrices de pondération Q et R sont définies positives ($Q, R \succ 0$), ce qui garantit l'existence et l'unicité de la solution optimale.

Méthode de Riccati

L'algorithme récursif de Riccati permet de résoudre ce problème efficacement sans avoir à inverser une grande matrice globale. Il procède par rétropropagation à partir de l'instant final k_f, ce qui est bien adapté aux problèmes de commande optimale sur un horizon fini.

Afin d'appliquer la méthode de Riccati, il est nécessaire de discretiser le temps.

Discrétisation du temps

On discrétise l'équation 7 avec un pas de temps ts=0,01s, on obtient alors un modèle discret :

$$X(k+1) = A_d X(k) + B_d U(k) \quad ; \quad Y(k) = C_d X(k) \tag{7}$$

La discrétisation des équations continues $\dot{X} = AX + BU$ avec un pas de temps T_s selon la méthode d'Euler (ordre 1) donne les équations discrètes suivantes :

$$A_d = I + T_s A \quad ; \quad B_d = T_s B \quad ; \quad C_d = C \tag{8}$$

Le terme $C_d X(k)$ correspond à la sortie du système observée. Dans notre cas, comme $C_d = I$, cela revient à comparer directement les états à la trajectoire de référence.

Équation de Riccati

A l'aide de l'équation de Riccati on obtient l'équation permettant de trouver la solutions du problème discrétisé :

$$\begin{aligned}
P(k) &= A_d^T P(k+1)[I + EP(k+1)]^{-1} A_d + V \\
V &= C_d^T Q C_d \\
E &= B_d R^{-1} B_d^T \\
g(k) &= \left[A_d^T - A_d^T P(k+1)[I + EP(k+1)]^{-1} E\right] g(k+1) + C_d^T Q r(k) \\
\bar{X}(k+1) &= [A_d - B_d L(k)]\bar{X}(k) + B_d L_g(k) g(k+1) \\
L(k) &= [R + B_d^T P(k+1) B_d]^{-1} B_d^T P(k+1) A_d \\
L_g(k) &= [R + B_d^T P(k+1) B_d]^{-1} B_d^T \\
\bar{U}(k) &= -L(k)\bar{X}(k) + L_g(k) g(k+1)
\end{aligned} \tag{11}$$

Avec comme conditions finales :

$$\begin{aligned}
P(k_f) &= C_d^T F C_d \\
g(k_f) &= C_d^T F\, r(k_f)
\end{aligned} \tag{13}$$

où Q et R sont respectivement les matrices de pondération des erreurs sur les états et des efforts de commande.

$$Q = \mathrm{diag}(100,\ 50,\ 10,\ 5,\ 0,\ 0,\ 100,\ 1,\ 100,\ 1,\ 1000,\ 0.1) \quad ; \quad R = \mathrm{diag}(10,\ 0,\ 0,\ 0) \tag{9}$$

De part les hypothèses faites, les angles doivent être contraints près de 0

$$-20^\circ < \phi < 20^\circ \quad ; \quad -20^\circ < \theta < 20^\circ \quad ; \quad -20^\circ < \psi < 20^\circ \tag{10}$$

```
using Pkg
Pkg.activate("projet_env")

Pkg.add("Plots")
Pkg.add("JuMP")
Pkg.add("Ipopt")
Pkg.add("COSMO")
Pkg.add("IterativeSolvers")
Pkg.add("LinearMaps")
Pkg.add("OSQP")
Pkg.add("MathOptInterface")
```

```
Pkg.add("Dates")
Pkg.add("PrettyTables")
Pkg.add("DataFrames")
```

Partie 2 : reproduction des résultats

Fonctionnement de l'algorithme :
L'algorithme commence à la date finale k_f, où les conditions terminales sont imposées, puis il effectue une récursion arrière (backward) sur les matrices $P(k)$ et $g(k)$ jusqu'à l'instant initial k_0.
Une fois ces matrices calculées, une simulation en avant (forward) permet d'estimer la trajectoire optimale $\bar{X}(k)$ et les commandes optimales $\bar{U}(k)$.

```
using LinearAlgebra, Plots, JuMP, Ipopt, COSMO, IterativeSolvers, LinearMaps, OSQP, MathOptInterface,Da
start_time = now()  # début

gr()

g = 9.81  # gravité

# Dimensions système
n = 12  # nombre d'états du drone
m = 4   # nombre d'entrées de contrôle
T = 6000  # durée de la simulation (en pas de temps)
Ts = 0.01  # pas de temps (10 ms)

# Paramètres physiques du drone
mass = 0.5
Ix, Iy, Iz = 0.0023, 0.0023, 0.004  # moments d'inertie autour des axes

# Matrices de coût pour le LQR
Q = Diagonal([100.0, 50.0, 10.0, 5.0, 0.0, 0.0, 100.0, 1.0, 100.0, 1.0, 1000.0, 0.1])
R = Diagonal([10.0, 1e-3, 1e-3, 1e-3])  # pondération des efforts de contrôle
F = Q  # coût terminal

# ================================
# Construction de A et B
# ================================
A = zeros(n, n)
B = zeros(n, m)

for i in 1:2:11
    A[i, i+1] = 1.0  # lien entre position et vitesse (ex: x, vx)
end


A[8, 3] = g     # acc x dépend de
A[10, 1] = -g   # acc y dépend de


B[2,2] = 1/Ix   # moment autour de x (¨)
B[4,3] = 1/Iy   # moment autour de y (¨)
B[6,4] = 1/Iz   # moment autour de z (¨)
```

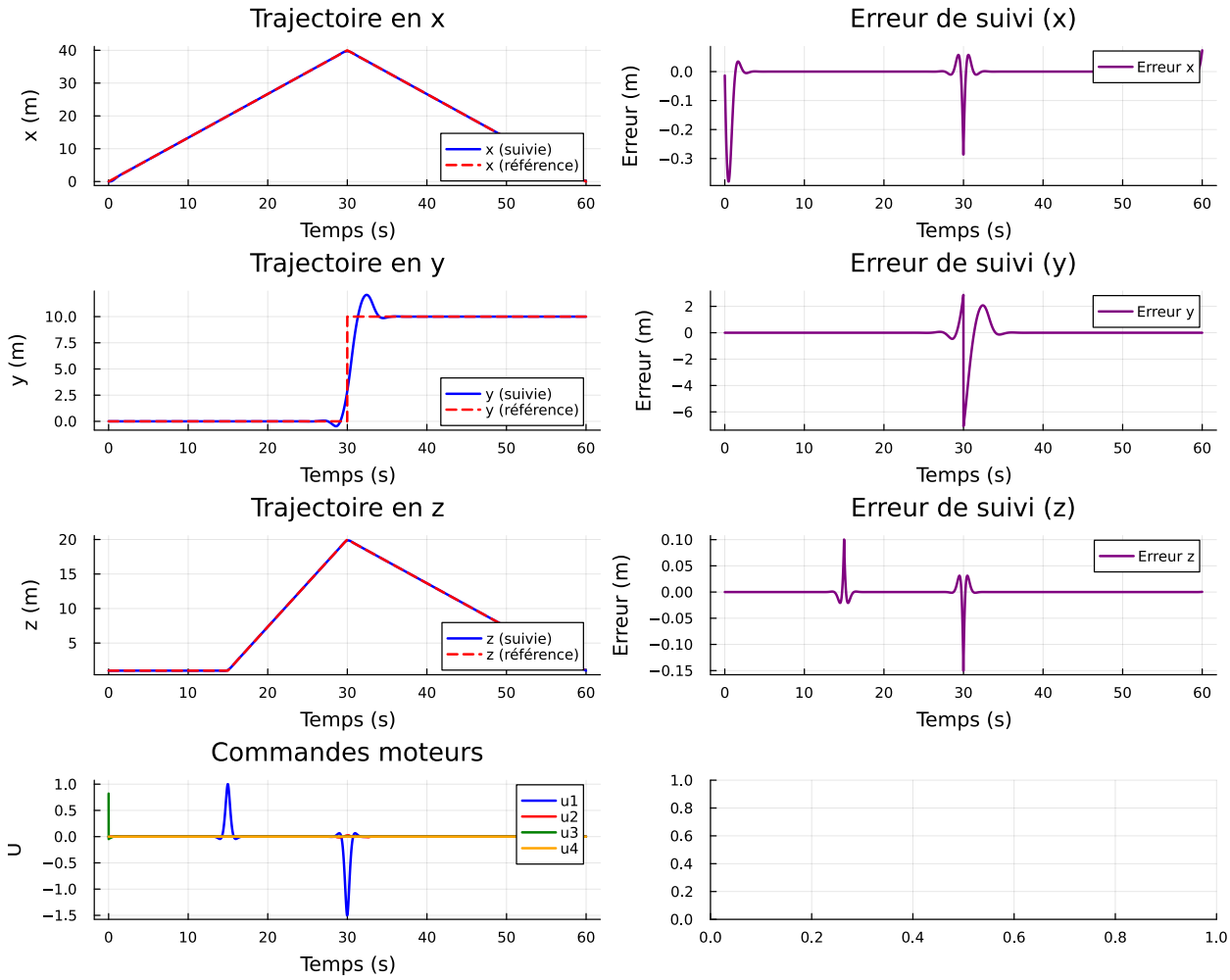
```
B[12,1] = 1/mass  # poussée verticale (z̈)


# ================================
# Discrétisation d'Euler
# ================================
Ad = I + Ts * A
Bd = Ts * B



# ================================
# Définition des trajectoires désirées (référence)
# ================================
r = zeros(n, T+1)

# x : montée puis descente linéaire
for k in 1:T÷2
    r[7, k] = 40.0 * (k / (T÷2))
end
for k in T÷2+1:T+1
    r[7, k] = 40.0 * (1 - (k - T÷2) / (T÷2))
end

# y : reste à 0, saute brusquement à 10 après la moitié du temps
for k in T÷2+1:T+1
    r[9, k] = 10.0
end

# z : montée douce de 1 à 20 entre 15s et 30s, puis redescente à 1m
dt = Ts
t15 = Int(15 / dt)
t30 = Int(30 / dt)

for k in 1:t15
    r[11, k] = 1.0
end
for k in t15+1:t30
    r[11, k] = 1.0 + (20.0 - 1.0) * ((k - t15) / (t30 - t15))
end
for k in t30+1:T+1
    r[11, k] = 1.0 + (20.0 - 1.0) * (1 - (k - t30) / (T + 1 - t30))
end

# ================================
# Calcul backward des gains LQR (formulation récursive de Riccati)
# ================================

# Allocation des vecteurs/matrices pour les gains
P = Vector{Matrix{Float64}}(undef, T+1)  # matrices de Riccati
G = Vector{Vector{Float64}}(undef, T+1)  # terme d'offset (trajectoire r)
L = Vector{Matrix{Float64}}(undef, T)    # gain de feedback état
Lg = Vector{Matrix{Float64}}(undef, T)   # gain associé à la référence
```

```
# Conditions terminales
C = I(n)  # matrice d'observation (identité ici)
P[T+1] = C' * Q * C
G[T+1] = C' * Q * r[:, T+1]

# Recursion backward Riccati
for k in T:-1:1
    E = Bd * inv(R) * Bd'
    V = C' * Q * C
    invIplusE = inv(I + E * P[k+1])

    P[k] = Ad' * P[k+1] * invIplusE * Ad + V
    G[k] = (Ad' - Ad' * P[k+1] * invIplusE * E) * G[k+1] + C' * Q * r[:, k]
end

# ================================
# Simulation forward du système contrôlé
# ================================

X = zeros(n, T+1)  # états
U = zeros(m, T)    # commandes

X[:,1] = zeros(n)  # état initial
X[11, 1] = 1.0     # état initial cohérent

for k in 1:T
    # Calcul des gains à l'instant k
    L[k] = inv(R + Bd' * P[k+1] * Bd) * Bd' * P[k+1] * Ad
    Lg[k] = inv(R + Bd' * P[k+1] * Bd) * Bd'

    # Calcul de la commande optimale
    U[:,k] = -L[k] * X[:,k] + Lg[k] * G[k+1]

    # Propagation de l'état
    X[:,k+1] = Ad * X[:,k] + Bd * U[:,k]

    # Contraintes physiques : angles d'attitude max (20°)
    phi_max = deg2rad(20)
    theta_max = deg2rad(20)
    psi_max = deg2rad(20)

    # contraintex des angles
    X[1,k+1] = clamp(X[1,k+1], -phi_max, phi_max)
    X[3,k+1] = clamp(X[3,k+1], -theta_max, theta_max)
    X[5,k+1] = clamp(X[5,k+1], -psi_max, psi_max)
end

# ================================
# Visualisation de la trajectoire et de l'erreur
# ================================

z = X[11, :]  # position verticale
x = X[7, :]   # position x
```

```
y = X[9, :]   # position y

# Erreurs de suivi
e_z = z .- r[11, :]
e_x = x .- r[7, :]
e_y = y .- r[9, :]
z = X[11, :]  # position verticale
x = X[7, :]   # position x
y = X[9, :]   # position y

# Erreurs de suivi
e_z = z .- r[11, :]
e_x = x .- r[7, :]
e_y = y .- r[9, :]

# Références
r_z = r[11, :]
r_x = r[7, :]
r_y = r[9, :]

t = Ts .* (0:T)  # temps réel en secondes

J_state1 = 0.0
J_control1 = 0.0
for k in 1:T
    e_k = X[:,k] - r[:,k]
    J_state1   += 0.5 * (e_k' * Q * e_k)
    J_control1 += 0.5 * (U[:,k]' * R * U[:,k])
end

# Coût terminal
e_final = X[:,T+1] - r[:,T+1]
J_terminal1 = 0.5 * (e_final' * F * e_final)

# Coût total
J1 = J_state1 + J_control1 + J_terminal1

end_time = now()  # fin

elapsed = end_time - start_time

println("=== Décomposition de la fonction objectif J ===")
println("Coût d'état (suivi)      : ", J_state1)
println("Coût de contrôle         : ", J_control1)
println("Coût terminal            : ", J_terminal1)
println("Valeur totale J          : ", J1)



println("Temps d'exécution total : ", elapsed)

# Création du layout 3 lignes × 2 colonnes
plt1 = plot(layout = (4, 2), size=(1000, 800))
```

```
# Trajectoire x
plot!(plt1[1], t, x, lw=2, label="x (suivie)", color=:blue)
plot!(plt1[1], t, r_x, lw=2, label="x (référence)", linestyle=:dash, color=:red)
plot!(plt1[1], title="Trajectoire en x", xlabel="Temps (s)", ylabel="x (m)", legend=:bottomright, grid=

# Erreur x
plot!(plt1[2], t, e_x, lw=2, label="Erreur x", color=:purple)
plot!(plt1[2], title="Erreur de suivi (x)", xlabel="Temps (s)", ylabel="Erreur (m)", legend=:topright, g

# Trajectoire y
plot!(plt1[3], t, y, lw=2, label="y (suivie)", color=:blue)
plot!(plt1[3], t, r_y, lw=2, label="y (référence)", linestyle=:dash, color=:red)
plot!(plt1[3], title="Trajectoire en y", xlabel="Temps (s)", ylabel="y (m)", legend=:bottomright, grid=

# Erreur y
plot!(plt1[4], t, e_y, lw=2, label="Erreur y", color=:purple)
plot!(plt1[4], title="Erreur de suivi (y)", xlabel="Temps (s)", ylabel="Erreur (m)", legend=:topright, g

# Trajectoire z
plot!(plt1[5], t, z, lw=2, label="z (suivie)", color=:blue)
plot!(plt1[5], t, r_z, lw=2, label="z (référence)", linestyle=:dash, color=:red)
plot!(plt1[5], title="Trajectoire en z", xlabel="Temps (s)", ylabel="z (m)", legend=:bottomright, grid=

# Erreur z
plot!(plt1[6], t, e_z, lw=2, label="Erreur z", color=:purple)
plot!(plt1[6], title="Erreur de suivi (z)", xlabel="Temps (s)", ylabel="Erreur (m)", legend=:topright, g

plot!(plt1[7], t[1:end-1], U[1,:], lw=2, label="u1", color=:blue)
plot!(plt1[7], t[1:end-1], U[2,:], lw=2, label="u2", color=:red)
plot!(plt1[7], t[1:end-1], U[3,:], lw=2, label="u3", color=:green)
plot!(plt1[7], t[1:end-1], U[4,:], lw=2, label="u4", color=:orange)
plot!(plt1[7], title="Commandes moteurs", xlabel="Temps (s)", ylabel="U", legend=:topright, grid=true)

# Affichage
display(plt1)
```

```
=== Décomposition de la fonction objectif J ===
Coût d'état (suivi)      : 226343.0962538947
Coût de contrôle         : 772.2491319260433
Coût terminal            : 0.902385416198814
Valeur totale J          : 227116.24777123696
Temps d'exécution total : 498 milliseconds
```



La valeur de la fonction objectif est de 2,27*10^5. On remarque sur les graphiques que la trajectoire est plutot bien respectée avec de faibles erreurs de suivis. La fonction objectif comporte 3 parties, une partie suivie, une partie contrôle et une partie coût position finale. On constate que la partie suivi participe pour beaucoup à la fonction objectif (99,5%).

partie 3 : résolution avec Ipopt

Le but de cette section est de comparer les résultats obtenus dans la Partie 2 avec des résultats venant de stratégies plus générique comme par exemple IPOPT. Le modèle est construit à partir de la librairie JuMP.

```
using LinearAlgebra, Plots, JuMP, Ipopt, COSMO, IterativeSolvers, LinearMaps, OSQP, MathOptInterface


x0 = zeros(n)
x0[11] = 1.0                    # altitude 1 m en hover

#   Modèle JuMP « all-at-once »
model = Model(Ipopt.Optimizer)
set_optimizer_attribute(model, "print_level", 0)

@variable(model, X[1:n, 0:T])           # états
```

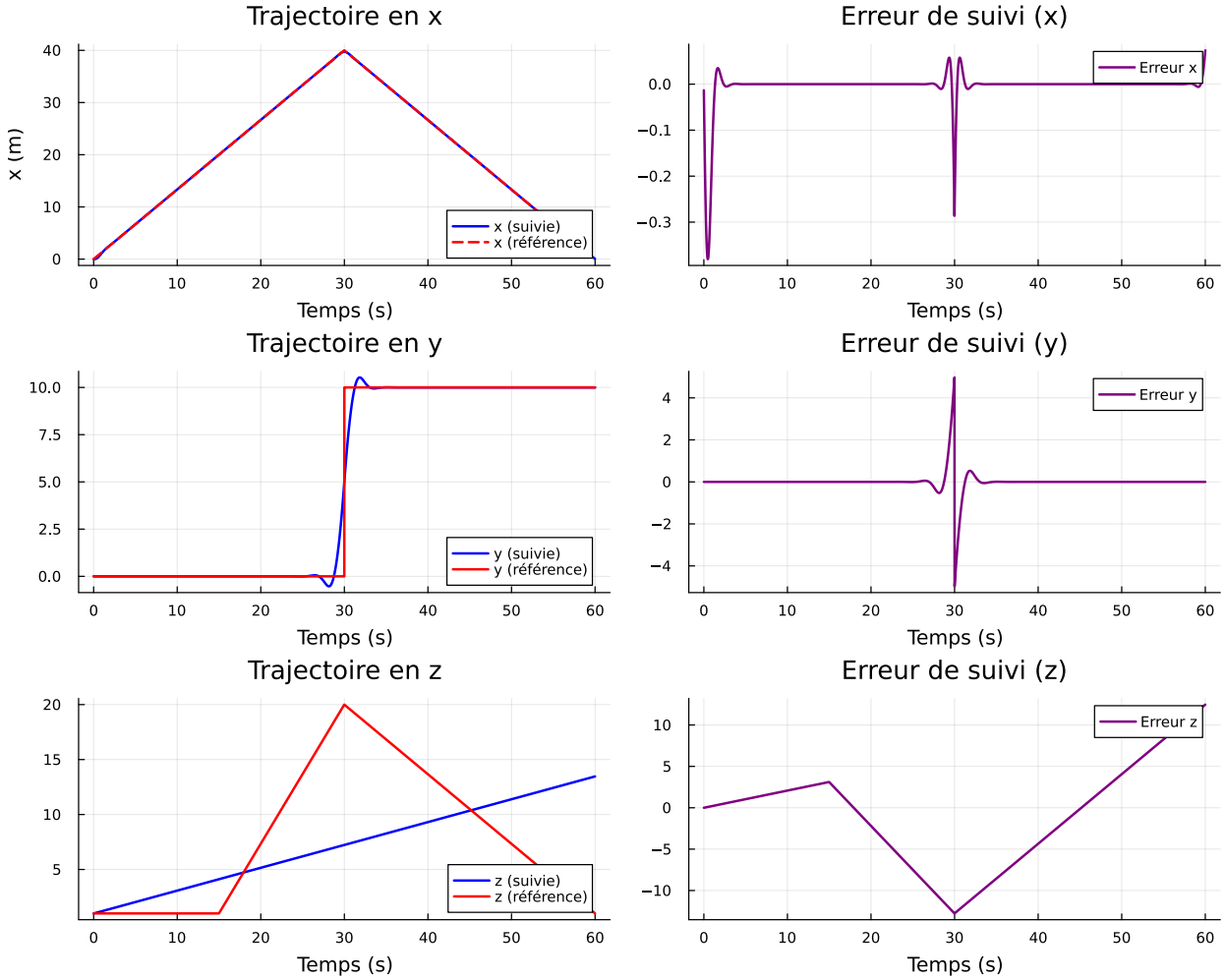
```
@variable(model, U[1:m, 0:T-1])         # commandes

# - dynamique linéaire
for k in 0:T-1
    @constraint(model, X[:, k+1] .== Ad * X[:, k] + Bd * U[:, k])
end

# - CONTRAINTE d'état initial  (= hover)
@constraint(model, X[:, 0] .== x0)

# - bornes (angles ±20°, poussée 0-2 mg)
deg20 = deg2rad(20.0)
@constraint(model, -deg20 .<= X[1, :] .<=  deg20)   #
@constraint(model, -deg20 .<= X[3, :] .<=  deg20)   #
@constraint(model, -deg20 .<= X[5, :] .<=  deg20)   #
@constraint(model, 0       .<= U[1, :] .<= 2 * mass * g)

# - coût
@expression(model, running_cost,
    sum( (X[:, k] - r[:, k+1])' * Q * (X[:, k] - r[:, k+1]) +
         U[:, k]' * R * U[:, k]           for k in 0:T-1) )

@expression(model, terminal_cost,
    (X[:, T] - r[:, T+1])' * F * (X[:, T] - r[:, T+1]) )

@objective(model, Min, 0.5 * running_cost + 0.5 * terminal_cost)

# - point initial
set_start_value.(X[:, 0], x0)
set_start_value.(U, 0.0)

optimize!(model)
println("Status          : ", termination_status(model))
println("Objective value : ", objective_value(model))

# ================================
# Visualisation de la trajectoire et de l'erreur
# ================================

X_val = value.(X)
U_val = value.(U)

J_state2   = 0.0
J_control2 = 0.0

for k in 1:T
    e_k = X_val[:,k] - r[:,k]
    J_state2 += 0.5 * (e_k' * Q * e_k)[1]
end

for k in 0:T-1
    J_control2 += 0.5 * (U_val[:,k]' * R * U_val[:,k])[1]
end
```

```
e_final = X_val[:,T] - r[:,T+1]
J_terminal2 = 0.5 * (e_final' * F * e_final)[1]

J2 = J_state2 + J_control2 + J_terminal2

println("\n--- Décomposition du coût : Modèle 2 ---")
println("J_state2    = ", J_state2)
println("J_control2  = ", J_control2)
println("J_terminal2 = ", J_terminal2)
println("J2 total    = ", J2)


solutionX = Array(value.(X))

z = solutionX[11, :]  # position verticale
x = solutionX[7, :]   # position x
y = solutionX[9, :]   # position y

# Erreurs de suivi
e_z = z .- r[11, :]
e_x = x .- r[7, :]
e_y = y .- r[9, :]
z = solutionX[11, :]  # position verticale
x = solutionX[7, :]   # position x
y = solutionX[9, :]   # position y

# Erreurs de suivi
e_z = z .- r[11, :]
e_x = x .- r[7, :]
e_y = y .- r[9, :]

# Références
r_z = r[11, :]
r_x = r[7, :]
r_y = r[9, :]

t = Ts .* (0:T)  # temps réel en secondes

# Création du layout 3 lignes × 2 colonnes
plt2 = plot(layout = (3, 2), size=(1000, 800))

# Trajectoire x
plot!(plt2[1], t, x, lw=2, label="x (suivie)", color=:blue)
plot!(plt2[1], t, r_x, lw=2, label="x (référence)", linestyle=:dash, color=:red)
plot!(plt2[1], title="Trajectoire en x", xlabel="Temps (s)", ylabel="x (m)", legend=:bottomright, grid=

# Erreur x
plot!(plt2[2], t, e_x, lw=2, label="Erreur x", color=:purple)
plot!(plt2[2], title="Erreur de suivi (x)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g

# Trajectoire y
plot!(plt2[3], t, y, lw=2, label="y (suivie)", color=:blue)
plot!(plt2[3], t, r_y, lw=2, label="y (référence)", inestyle=:dash, color=:red)
```

```
plot!(plt2[3], title="Trajectoire en y", xlabel="Temps (s)", label="y (m)", legend=:bottomright, grid=t
# Erreur y
plot!(plt2[4], t, e_y, lw=2, label="Erreur y", color=:purple)
plot!(plt2[4], title="Erreur de suivi (y)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g
# Trajectoire z
plot!(plt2[5], t, z, lw=2, label="z (suivie)", color=:blue)
plot!(plt2[5], t, r_z, lw=2, label="z (référence)", inestyle=:dash, color=:red)
plot!(plt2[5], title="Trajectoire en z", xlabel="Temps (s)", label="z (m)", legend=:bottomright, grid=t
# Erreur z
plot!(plt2[6], t, e_z, lw=2, label="Erreur z", color=:purple)
plot!(plt2[6], title="Erreur de suivi (z)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g
# Affichage
display(plt2)
```

```
-----------------------------------------------------------------
          OSQP v0.6.2  -  Operator Splitting QP Solver
             (c) Bartolomeo Stellato,  Goran Banjac
        University of Oxford  -  Stanford University 2021
-----------------------------------------------------------------
problem:  variables n = 96012, constraints m = 96015
          nnz(P) + nnz(A) = 324025
settings: linear system solver = qdldl,
          eps_abs = 1.0e-003, eps_rel = 1.0e-003,
          eps_prim_inf = 1.0e-004, eps_dual_inf = 1.0e-004,
          rho = 1.00e-001 (adaptive),
          sigma = 1.00e-006, alpha = 1.60, max_iter = 4000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: off, time_limit: off

iter  objective    pri res    dua res    rho        time
   1 -3.1481e+008  9.92e-001  1.12e+007  1.00e-001  1.33e-001s
 200 -4.9136e+008  1.73e-002  5.12e+000  1.00e-001  1.00e+000s
 400 -4.9128e+008  1.46e-002  3.53e+000  1.00e-001  1.91e+000s
 600 -4.9121e+008  1.32e-002  2.85e+000  1.00e-001  2.79e+000s
 800 -4.9114e+008  1.23e-002  2.35e+000  1.00e-001  3.66e+000s
1000 -4.9108e+008  1.16e-002  2.16e+000  1.00e-001  4.56e+000s
1200 -4.9102e+008  1.11e-002  1.85e+000  1.00e-001  5.46e+000s
1400 -4.9097e+008  1.07e-002  1.70e+000  1.00e-001  6.36e+000s
1600 -4.9091e+008  1.03e-002  1.61e+000  1.00e-001  7.23e+000s
1800 -4.9086e+008  1.00e-002  1.47e+000  1.00e-001  8.14e+000s
2000 -4.9081e+008  9.75e-003  1.35e+000  1.00e-001  9.03e+000s
2200 -4.9076e+008  9.52e-003  1.29e+000  1.00e-001  9.92e+000s
2400 -4.9071e+008  9.31e-003  1.16e+000  1.00e-001  1.08e+001s
2600 -4.9067e+008  9.13e-003  1.12e+000  1.00e-001  1.17e+001s
2800 -4.9062e+008  8.96e-003  1.06e+000  1.00e-001  1.26e+001s
3000 -4.9058e+008  8.81e-003  9.89e-001  1.00e-001  1.35e+001s
3200 -4.9053e+008  8.67e-003  9.73e-001  1.00e-001  1.45e+001s
3400 -4.9049e+008  8.53e-003  9.91e-001  1.00e-001  1.56e+001s
3600 -4.9044e+008  8.41e-003  1.05e+000  1.00e-001  1.66e+001s
```

```
3800 -4.9040e+008  8.30e-003  1.04e+000  1.00e-001  1.75e+001s
4000 -4.9036e+008  8.19e-003  1.12e+000  1.00e-001  1.84e+001s

status:               solved inaccurate
number of iterations: 4000
optimal objective:    -490357780.3889
run time:             1.84e+001s
optimal rho estimate: 3.83e-001

Status          : LOCALLY_SOLVED
Objective value : 1.1547786988644415e8

--- Décomposition du coût : Modèle 2 ---
J_state2    = 1.1538857316889812e8
J_control2  = 482.8650848091259
J_terminal2 = 77704.20093651666
J2 total    = 1.1546676023491944e8
```



Analyse des résultats

La méthode IPOPT converge belle et bien pour ce problème et on obtient une valeur de l'objectif de 1.230511402197558e8. Il est possible de faire mieux avec des méthodes utilisant les propriétés du problème.

Deux exemples de ces méthodes seront visités dans la prochaine section. Notons que la convergence a été atteinte en 4000 itérations pour un temps de 1.63e+001 secondes. On observe sur les graphiques que la solution proposée est efficace, en ce sens que le trajet à suivre est assez bien respectés. Notons que les erreurs sont plus grandes autour des points où il y a un changement dans la dérivée. Notre approximation étant une fonction lisse, cela n'est pas étonnant. L'erreur est aussi grande au temps final, ce qui peut être un problème dépendamment des applications. Un grand déplacement programmé comme une somme de petites trajectoires pourrait facilement accumuler des erreurs.

Notons toutefois qu'il semble y avoir une erreur à quelque part dans le code, en effet, il y a clairement une grosse erreur dans la trajectoire en z alors que le graphe de l'erreur semble indiquer que l'estimation est excellente. Nous pensons qu'il s'agit d'une erreur dans notre implémentation de la modélisation JUmP du problème. Cette modélisation étant utilisée pour les deux prochaines sections, nous pensons qu'elle explique certains de nos résultats dans cette section aussi.

À cause de cette erreur, nous n'approfondissons pas l'analyse des résultats pour l'instant. Le lecteur trouvera toutefois un tableau comparatif des méthodes à la fin de la section 5.

Partie 4 : Résolution avec le solver COSMO

Le solveur COSMO implémente le **Conic operator splitting method** , qui est particulièrement adaptée pour résoudre de larges problèmes d'optimisation convexe et conique dont l'objectif est donné par une fonction quadratique. Il s'agit donc d'une méthode adaptée au problème monolithique creux. Le modèle est encore une fois construit à l'aide de la librairie JuMP.

```
using JuMP, COSMO, IterativeSolvers, LinearMaps, OSQP, MathOptInterface, Plots

model3 = Model(COSMO.Optimizer)

@variable(model3, X[1:n, 0:T])           # états
@variable(model3, U[1:m, 0:T-1])         # commandes

# - dynamique linéaire
for k in 0:T-1
    @constraint(model3, X[:, k+1] .== Ad * X[:, k] + Bd * U[:, k])
end

# - CONTRAINTE d'état initial  (= hover)
@constraint(model3, X[:, 0] .== x0)

# - bornes (angles ±20°, poussée 0-2 mg)
deg20 = deg2rad(20.0)
@constraint(model3, -deg20 .<= X[1, :] .<=  deg20)   #
@constraint(model3, -deg20 .<= X[3, :] .<=  deg20)   #
@constraint(model3, -deg20 .<= X[5, :] .<=  deg20)   #
@constraint(model3, 0       .<= U[1, :] .<= 2 * mass * g)

# - coût
@expression(model3, running_cost,
    sum( (X[:, k] - r[:, k+1])' * Q * (X[:, k] - r[:, k+1]) +
         U[:, k]' * R * U[:, k]           for k in 0:T-1) )

@expression(model3, terminal_cost,
    (X[:, T] - r[:, T+1])' * F * (X[:, T] - r[:, T+1]) )

@objective(model3, Min, 0.5 * running_cost + 0.5 * terminal_cost)
```

```
# - point initial
set_start_value.(X[:, 0], x0)
set_start_value.(U, 0.0)

optimize!(model3)
println("Status          : ", termination_status(model3))

# ================================
# Visualisation de la trajectoire et de l'erreur
# ================================

X_val = value.(X)
U_val = value.(U)

J_state3   = 0.0
J_control3 = 0.0

# Coût d'état
for k in 1:T
    e_k = X_val[:,k] - r[:,k]
    J_state3 += 0.5 * (e_k' * Q * e_k)[1]
end

# Coût de contrôle
for k in 0:T-1
    J_control3 += 0.5 * (U_val[:,k]' * R * U_val[:,k])[1]
end

# Coût terminal
e_final = X_val[:,T] - r[:,T+1]
J_terminal3 = 0.5 * (e_final' * F * e_final)[1]

# Coût total
J3 = J_state3 + J_control3 + J_terminal3

println("\n--- Décomposition du coût ---")
println("J_state3    = ", J_state3)
println("J_control3  = ", J_control3)
println("J_terminal3 = ", J_terminal3)
println("J3 total    = ", J3)




solutionX = Array(value.(X))

z = solutionX[11, :]  # position verticale
x = solutionX[7, :]   # position x
y = solutionX[9, :]   # position y

# Erreurs de suivi
e_z = z .- r[11, :]
```

```
e_x = x .- r[7, :]
e_y = y .- r[9, :]
z = solutionX[11, :]  # position verticale
x = solutionX[7, :]   # position x
y = solutionX[9, :]   # position y

# Erreurs de suivi
e_z = z .- r[11, :]
e_x = x .- r[7, :]
e_y = y .- r[9, :]

# Références
r_z = r[11, :]
r_x = r[7, :]
r_y = r[9, :]

t = Ts .* (0:T)  # temps réel en secondes

# Création du layout 3 lignes × 2 colonnes
plt3 = plot(layout = (3, 2), size=(1000, 800))

# Trajectoire x
plot!(plt3[1], t, x, lw=2, label="x (suivie)", color=:blue)
plot!(plt3[1], t, r_x, lw=2, label="x (référence)", linestyle=:dash, color=:red)
plot!(plt3[1], title="Trajectoire en x", xlabel="Temps (s)", ylabel="x (m)", legend=:bottomright, grid=

# Erreur x
plot!(plt3[2], t, e_x, lw=2, label="Erreur x", color=:purple)
plot!(plt3[2], title="Erreur de suivi (x)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g

# Trajectoire y
plot!(plt3[3], t, y, lw=2, label="y (suivie)", color=:blue)
plot!(plt3[3], t, r_y, lw=2, label="y (référence)", inestyle=:dash, color=:red)
plot!(plt3[3], title="Trajectoire en y", xlabel="Temps (s)", label="y (m)", legend=:bottomright, grid=t

# Erreur y
plot!(plt3[4], t, e_y, lw=2, label="Erreur y", color=:purple)
plot!(plt3[4], title="Erreur de suivi (y)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g

# Trajectoire z
plot!(plt3[5], t, z, lw=2, label="z (suivie)", color=:blue)
plot!(plt3[5], t, r_z, lw=2, label="z (référence)", inestyle=:dash, color=:red)
plot!(plt3[5], title="Trajectoire en z", xlabel="Temps (s)", label="z (m)", legend=:bottomright, grid=t

# Erreur z
plot!(plt3[6], t, e_z, lw=2, label="Erreur z", color=:purple)
plot!(plt3[6], title="Erreur de suivi (z)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g

# Affichage
display(plt3)
```

```
------------------------------------------------------------------
          COSMO v0.8.9 - A Quadratic Objective Conic Solver
```

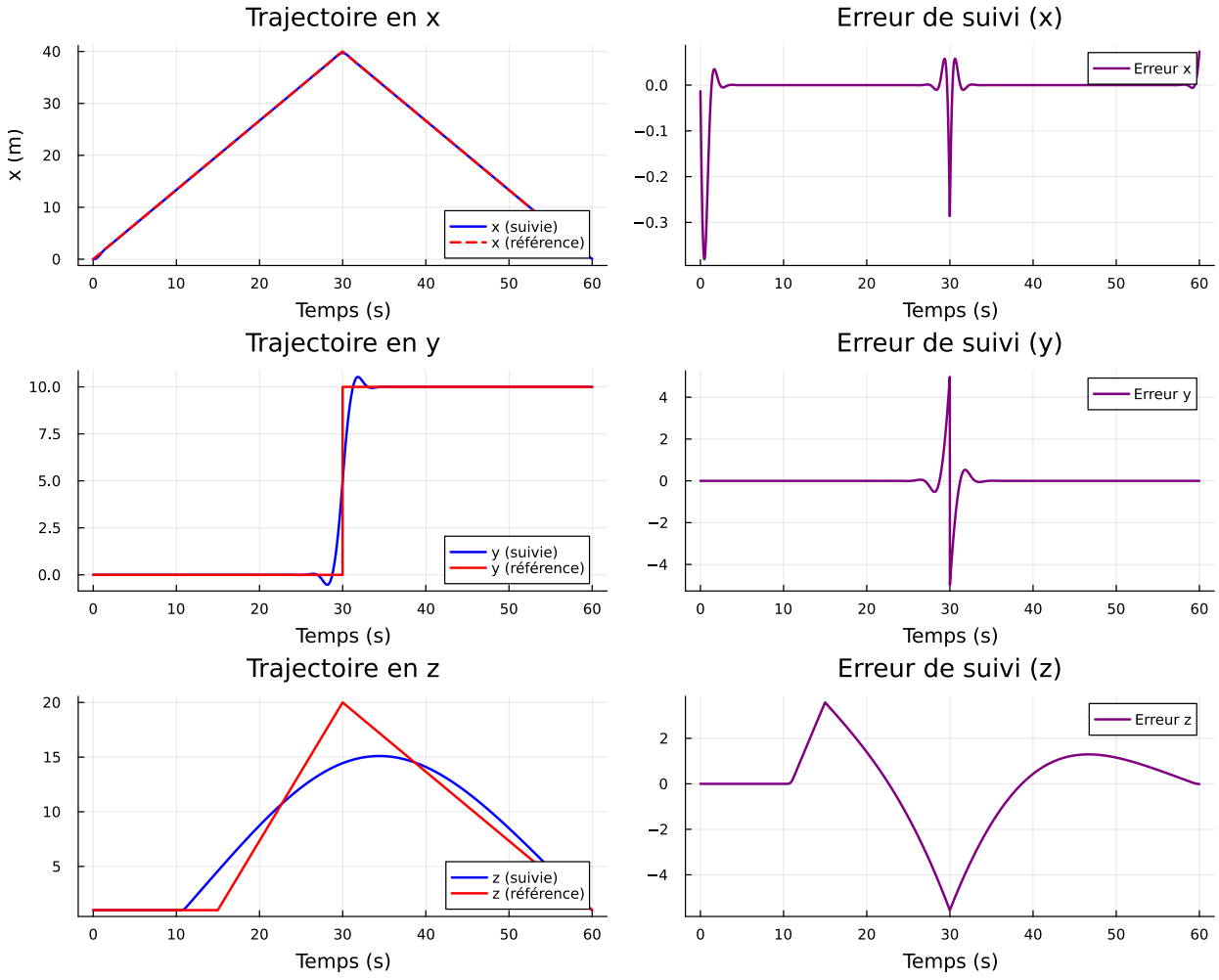
```
                        Michael Garstka
                University of Oxford, 2017 - 2022
------------------------------------------------------------------

Problem:  x   R^{96012},
          constraints: A   R^{96015x96012} (240015 nnz),
          matrix size to factor: 192027x192027,
          Floating-point precision: Float64
Sets:     ZeroSet of dim: 72012
          Box of dim: 24003
Settings:  _abs = 1.0e-05,  _rel = 1.0e-05,
           _prim_inf = 1.0e-04,  _dual_inf = 1.0e-04,
            = 0.1,   = 1e-06,   = 1.6,
          max_iter = 5000,
          scaling iter = 10 (on),
          check termination every 25 iter,
          check infeasibility every 40 iter,
          KKT system solver: COSMO.QdldlKKTSolver
Acc:      Anderson Type2{QRDecomp},
          Memory size = 15, RestartedMemory,
          Safeguarded: true, tol: 2.0
Setup Time: 175226.41ms

Iter:   Objective:  Primal Res: Dual Res:   Rho:
1   -3.1481e+08 6.0000e-01  1.1996e+04  1.0000e-01
25  -4.9146e+08 2.9473e-02  1.9190e+01  1.0000e-01
50  -4.9144e+08 2.4973e-02  1.3154e+01  1.0000e-01
75  -4.9142e+08 2.7355e-02  2.2520e+01  1.0000e-01
100 -4.9141e+08 2.3927e-02  2.6074e+01  1.0000e-01
125 -4.9140e+08 1.9518e-02  1.0299e+01  1.0000e-01
150 -4.9138e+08 5.8698e-02  7.1294e+01  1.0000e-01
175 -4.9137e+08 1.8599e-02  1.6059e+01  1.0000e-01
200 -4.9135e+08 1.6949e-02  6.0464e+00  1.0000e-01
225 -4.9134e+08 1.6402e-02  8.7231e+00  1.0000e-01
250 -4.9133e+08 1.5910e-02  7.9415e+00  1.0000e-01
275 -4.9131e+08 1.5439e-02  5.1007e+00  1.0000e-01
300 -4.9129e+08 4.0581e-02  4.5585e+01  1.0000e-01
325 -4.9128e+08 1.4655e-02  5.0254e+00  1.0000e-01
350 -4.9127e+08 1.4294e-02  5.1501e+00  1.0000e-01
375 -4.9126e+08 1.4000e-02  1.1897e+01  1.0000e-01
400 -4.9124e+08 2.0980e-02  1.9427e+01  1.0000e-01
425 -4.9123e+08 1.3535e-02  4.3542e+00  1.0000e-01
450 -4.9122e+08 1.3296e-02  6.9469e+00  1.0000e-01
475 -4.9120e+08 1.3068e-02  1.2863e+01  1.0000e-01
500 -4.9119e+08 1.2873e-02  3.3261e+00  1.0000e-01
525 -4.9118e+08 1.2674e-02  5.9679e+00  1.0000e-01
550 -4.9116e+08 1.2491e-02  6.1845e+00  1.0000e-01
575 -4.9115e+08 1.2300e-02  5.1169e+00  1.0000e-01
600 -4.9113e+08 1.2137e-02  1.1407e+01  1.0000e-01
625 -4.9112e+08 1.1968e-02  2.9612e+00  1.0000e-01
650 -4.9110e+08 2.7720e-02  8.4857e+00  5.1210e-01
675 -4.9106e+08 1.2247e-02  7.9698e+01  5.1210e-01
700 -4.9100e+08 1.0885e-02  2.8355e+01  5.1210e-01
725 -4.9095e+08 3.1238e-02  3.3953e+01  5.1210e-01
```

```
750 -4.9092e+08 1.0296e-02  6.2735e+01  5.1210e-01
775 -4.9088e+08 1.0069e-02  3.4530e+01  5.1210e-01
800 -4.9082e+08 2.2342e-02  2.8507e+01  5.1210e-01
825 -4.9078e+08 9.5716e-03  1.7486e+01  5.1210e-01
850 -4.9074e+08 9.3839e-03  1.6613e+01  5.1210e-01
875 -4.9068e+08 3.8396e-02  2.3958e+01  5.1210e-01
900 -4.9065e+08 9.8545e-03  4.2464e+01  5.1210e-01
925 -4.9060e+08 8.8814e-03  1.5886e+01  5.1210e-01
950 -4.9055e+08 2.1327e-02  2.9805e+01  5.1210e-01
975 -4.9051e+08 8.5960e-03  5.0120e+01  5.1210e-01
1000    -4.9047e+08 8.4698e-03  3.6290e+01  5.1210e-01
1025    -4.9041e+08 1.6703e-02  2.1107e+01  5.1210e-01
1050    -4.9037e+08 8.2107e-03  1.7712e+01  5.1210e-01
1075    -4.9031e+08 8.0765e-03  2.2791e+01  5.1210e-01
1100    -4.9025e+08 1.1427e-01  3.0092e+01  5.1210e-01
1125    -4.9022e+08 7.8753e-03  3.9677e+01  5.1210e-01
1150    -4.9016e+08 7.7725e-03  2.3434e+01  5.1210e-01
1175    -4.9012e+08 2.4841e-02  2.6389e+01  5.1210e-01
1200    -4.9008e+08 7.6258e-03  2.3048e+01  5.1210e-01
1225    -4.9002e+08 7.5314e-03  3.0127e+01  5.1210e-01
1250    -4.8998e+08 2.0416e-02  2.5880e+01  5.1210e-01
1275    -4.8994e+08 7.4070e-03  3.3884e+01  5.1210e-01
1300    -4.8988e+08 7.5268e-03  3.1994e+01  5.1210e-01
1325    -4.8983e+08 8.1679e-03  3.4846e+01  5.1210e-01
1350    -4.8978e+08 7.1924e-03  2.5352e+01  5.1210e-01
1375    -4.8972e+08 7.1263e-03  2.4542e+01  5.1210e-01
1400    -4.8967e+08 2.9359e-02  2.7473e+01  5.1210e-01
1425    -4.8963e+08 7.0233e-03  2.3537e+01  5.1210e-01
1450    -4.8957e+08 6.9575e-03  2.9509e+01  5.1210e-01
1475    -4.8950e+08 2.6341e-01  3.6713e+01  5.1210e-01
1500    -4.8947e+08 4.3927e-02  2.2685e+02  5.1210e-01
1525    -4.8940e+08 7.3879e-03  3.0948e+01  5.1210e-01
1550    -4.8936e+08 4.6456e-02  2.7536e+01  5.1210e-01
1575    -4.8932e+08 9.5052e-03  5.6722e+01  5.1210e-01
1600    -4.8928e+08 6.6707e-03  2.8060e+01  5.1210e-01
1625    -4.8923e+08 1.0998e-02  2.3608e+01  5.1210e-01
1650    -4.8918e+08 6.5841e-03  2.8397e+01  5.1210e-01
1675    -4.8913e+08 6.5454e-03  2.2639e+01  5.1210e-01
1700    -4.8908e+08 3.9949e-02  2.4335e+01  5.1210e-01
1725    -4.8903e+08 5.7983e-02  3.1168e+02  5.1210e-01
1750    -4.8900e+08 6.4371e-03  2.2376e+01  5.1210e-01
1775    -4.8894e+08 4.8228e-02  2.4944e+01  5.1210e-01
1800    -4.8889e+08 6.3590e-03  2.5837e+01  5.1210e-01
1825    -4.8883e+08 6.6204e-03  3.0924e+01  5.1210e-01
1850    -4.8878e+08 6.0996e-02  2.4080e+01  5.1210e-01
1875    -4.8873e+08 6.2413e-03  3.9077e+01  5.1210e-01
1900    -4.8869e+08 6.2119e-03  2.2168e+01  5.1210e-01
1925    -4.8861e+08 4.2263e-02  2.5947e+01  5.1210e-01
1950    -4.8856e+08 2.5297e-02  1.5074e+02  5.1210e-01
1975    -4.8850e+08 6.0887e-03  2.1182e+01  5.1210e-01
2000    -4.8844e+08 5.5467e-02  2.4541e+01  5.1210e-01
2025    -4.8840e+08 1.6027e-02  7.2971e+01  5.1210e-01
2050    -4.8835e+08 5.9924e-03  2.3377e+01  5.1210e-01
2075    -4.8827e+08 9.1089e-02  2.7982e+01  5.1210e-01
```

2100	-4.8822e+08	5.9196e-03	2.9836e+01	5.1210e-01
2125	-4.8815e+08	6.0427e-03	2.5764e+01	5.1210e-01
2150	-4.8811e+08	5.8546e-03	1.8273e+01	5.1210e-01
2175	-4.8805e+08	5.8234e-03	2.4574e+01	5.1210e-01
2200	-4.8799e+08	5.7889e-03	1.9562e+01	5.1210e-01
2225	-4.8793e+08	1.1082e-01	2.2031e+01	5.1210e-01
2250	-4.8787e+08	7.4474e-03	4.6252e+01	5.1210e-01
2275	-4.8780e+08	5.6898e-03	2.7811e+01	5.1210e-01
2300	-4.8768e+08	2.4120e-01	3.5886e+01	5.1210e-01
2325	-4.8764e+08	3.4186e-02	2.0090e+02	5.1210e-01
2350	-4.8757e+08	5.5758e-03	1.7031e+01	5.1210e-01
2375	-4.8751e+08	8.1841e-02	2.1048e+01	5.1210e-01
2400	-4.8745e+08	5.8820e-02	3.2142e+02	5.1210e-01
2425	-4.8741e+08	5.4985e-03	1.7296e+01	5.1210e-01
2450	-4.8735e+08	1.6802e-02	1.8633e+01	5.1210e-01
2475	-4.8730e+08	5.4468e-03	1.6768e+01	5.1210e-01
2500	-4.8723e+08	5.4162e-03	2.0044e+01	5.1210e-01
2525	-4.8716e+08	1.9158e-02	2.2879e+01	5.1210e-01
2550	-4.8710e+08	6.5812e-03	4.4179e+01	5.1210e-01
2575	-4.8706e+08	5.3424e-03	1.6633e+01	5.1210e-01
2600	-4.8698e+08	5.0404e-02	2.2915e+01	5.1210e-01
2625	-4.8693e+08	2.3447e-02	1.3820e+02	5.1210e-01
2650	-4.8687e+08	5.2643e-03	2.1179e+01	5.1210e-01
2675	-4.8675e+08	1.5685e-01	2.6724e+01	5.1210e-01
2700	-4.8671e+08	5.2017e-03	1.6327e+01	5.1210e-01
2725	-4.8663e+08	5.1730e-03	1.6384e+01	5.1210e-01
2750	-4.8657e+08	2.4561e-02	1.5139e+01	5.1210e-01
2775	-4.8651e+08	1.6754e-02	9.6330e+01	5.1210e-01
2800	-4.8646e+08	5.1079e-03	1.7810e+01	5.1210e-01
2825	-4.8640e+08	4.7204e-02	2.0201e+01	5.1210e-01
2850	-4.8634e+08	1.2332e-02	5.6526e+01	5.1210e-01
2875	-4.8628e+08	5.0403e-03	1.7559e+01	5.1210e-01
2900	-4.8620e+08	1.6111e-01	1.9262e+01	5.1210e-01
2925	-4.8615e+08	2.6795e-02	1.4007e+02	5.1210e-01
2950	-4.8609e+08	4.9740e-03	1.4769e+01	5.1210e-01
2975	-4.8601e+08	5.7495e-02	1.8901e+01	5.1210e-01
3000	-4.8596e+08	7.2895e-03	4.5130e+01	5.1210e-01
3025	-4.8591e+08	4.9142e-03	1.3788e+01	5.1210e-01
3050	-4.8583e+08	5.6806e-02	2.0490e+01	5.1210e-01
3075	-4.8577e+08	7.9386e-02	4.5911e+02	5.1210e-01
3100	-4.8574e+08	4.8583e-03	1.3397e+01	5.1210e-01
3125	-4.8566e+08	4.9810e-02	1.5937e+01	5.1210e-01
3150	-4.8560e+08	4.8161e-03	1.3885e+01	5.1210e-01
3175	-4.8554e+08	4.7952e-03	1.4812e+01	5.1210e-01
3200	-4.8545e+08	1.2270e-01	1.8567e+01	5.1210e-01
3225	-4.8542e+08	4.7603e-03	1.1750e+01	5.1210e-01
3250	-4.8534e+08	4.7365e-03	1.7016e+01	5.1210e-01
3275	-4.8528e+08	7.5979e-02	1.6658e+01	5.1210e-01
3300	-4.8522e+08	2.7968e-02	1.4928e+02	5.1210e-01
3325	-4.8517e+08	4.6846e-03	1.2182e+01	5.1210e-01
3350	-4.8511e+08	1.4779e-02	1.3416e+01	5.1210e-01
3375	-4.8504e+08	4.6482e-03	1.4928e+01	5.1210e-01
3400	-4.8498e+08	4.6296e-03	1.2919e+01	5.1210e-01
3425	-4.8490e+08	1.7965e-02	1.4545e+01	5.1210e-01

3450	-4.8486e+08	4.7183e-03	2.9099e+01	5.1210e-01
3475	-4.8480e+08	4.5805e-03	2.1391e+01	5.1210e-01
3500	-4.8473e+08	2.0583e-02	1.5770e+01	5.1210e-01
3525	-4.8467e+08	5.3000e-03	3.1901e+01	5.1210e-01
3550	-4.8458e+08	4.5216e-03	1.6148e+01	5.1210e-01
3575	-4.8452e+08	8.0747e-02	1.8445e+01	5.1210e-01
3600	-4.8445e+08	1.8437e-02	9.4082e+01	5.1210e-01
3625	-4.8437e+08	4.4666e-03	1.8603e+01	5.1210e-01
3650	-4.8431e+08	5.3766e-02	1.3969e+01	5.1210e-01
3675	-4.8426e+08	4.4384e-03	1.0917e+01	5.1210e-01
3700	-4.8420e+08	4.4232e-03	1.3542e+01	5.1210e-01
3725	-4.8413e+08	4.4077e-03	9.5884e+00	5.1210e-01
3750	-4.8408e+08	4.3926e-03	1.1546e+01	5.1210e-01
3775	-4.8400e+08	4.3740e-03	1.5777e+01	5.1210e-01
3800	-4.8394e+08	4.3608e-03	1.0867e+01	5.1210e-01
3825	-4.8386e+08	4.3403e-03	1.4802e+01	5.1210e-01
3850	-4.8380e+08	4.3275e-03	1.0478e+01	5.1210e-01
3875	-4.8373e+08	4.0154e-02	1.4353e+01	5.1210e-01
3900	-4.8369e+08	1.0417e-02	5.8091e+01	5.1210e-01
3925	-4.8362e+08	4.2851e-03	1.0869e+01	5.1210e-01
3950	-4.8357e+08	4.2718e-03	1.0545e+01	5.1210e-01
3975	-4.8349e+08	4.2538e-03	1.2783e+01	5.1210e-01
4000	-4.8342e+08	4.2387e-03	1.2722e+01	5.1210e-01
4025	-4.8336e+08	4.8210e-02	1.4974e+01	5.1210e-01
4050	-4.8332e+08	6.2171e-03	3.7716e+01	5.1210e-01
4075	-4.8327e+08	4.2044e-03	9.4578e+00	5.1210e-01
4100	-4.8319e+08	4.3091e-03	1.0234e+01	5.1210e-01
4125	-4.8314e+08	4.1774e-03	9.7581e+00	5.1210e-01
4150	-4.8308e+08	4.1650e-03	1.0267e+01	5.1210e-01
4175	-4.8299e+08	4.8760e-02	1.5407e+01	5.1210e-01
4200	-4.8294e+08	1.4714e-02	8.1138e+01	5.1210e-01
4225	-4.8290e+08	4.1255e-03	8.4319e+00	5.1210e-01
4250	-4.8281e+08	1.4351e-01	1.4012e+01	5.1210e-01
4275	-4.8275e+08	4.2427e-02	2.4530e+02	5.1210e-01
4300	-4.8271e+08	4.0871e-03	9.9564e+00	5.1210e-01
4325	-4.8264e+08	6.7452e-02	1.2049e+01	5.1210e-01
4350	-4.8256e+08	2.1411e-02	1.1275e+02	5.1210e-01
4375	-4.8251e+08	4.0451e-03	1.0474e+01	5.1210e-01
4400	-4.8242e+08	3.3487e-02	1.1873e+01	5.1210e-01
4425	-4.8239e+08	4.0225e-03	8.0819e+00	5.1210e-01
4450	-4.8233e+08	1.1373e-02	6.6588e+01	5.1210e-01
4475	-4.8227e+08	2.2568e-02	1.3834e+01	5.1210e-01
4500	-4.8220e+08	1.0918e-02	5.6966e+01	5.1210e-01
4525	-4.8215e+08	5.0085e-03	1.8537e+01	5.1210e-01
4550	-4.8206e+08	4.2625e-02	1.4575e+01	5.1210e-01
4575	-4.8201e+08	1.1696e-02	6.9173e+01	5.1210e-01
4600	-4.8194e+08	3.9363e-03	1.0382e+01	5.1210e-01
4625	-4.8189e+08	1.7135e-02	9.3395e+00	5.1210e-01
4650	-4.8178e+08	3.9064e-03	1.1751e+01	5.1210e-01
4675	-4.8173e+08	5.0728e-03	1.9598e+01	5.1210e-01
4700	-4.8165e+08	5.6717e-02	1.0937e+01	5.1210e-01
4725	-4.8162e+08	3.8768e-03	1.6125e+01	5.1210e-01
4750	-4.8155e+08	3.8646e-03	9.2767e+00	5.1210e-01
4775	-4.8149e+08	8.6631e-02	1.2043e+01	5.1210e-01

```
4800    -4.8144e+08 3.2531e-02  1.8914e+02  5.1210e-01
4825    -4.8135e+08 3.8303e-03  1.2276e+01  5.1210e-01
4850    -4.8131e+08 6.8224e-03  8.2936e+00  5.1210e-01
4875    -4.8126e+08 3.8144e-03  9.0635e+00  5.1210e-01
4900    -4.8120e+08 3.8042e-03  9.7987e+00  5.1210e-01
4925    -4.8116e+08 9.8915e-03  9.6450e+00  5.1210e-01
4950    -4.8111e+08 2.1838e-02  1.2678e+02  5.1210e-01

---------------------------------------------------------------------
>>> Results
Status: Max_iter_reached
Iterations: 5000 (incl. 36 safeguarding iter)
Optimal objective: -4.811e+08
Runtime: 240.064s (240064.0ms)

Status          : ITERATION_LIMIT

--- Décomposition du coût ---
J_state3    = 1.0481513719296554e7
J_control3  = 142.19454759002005
J_terminal3 = 0.9695001076127201
J3 total    = 1.0481656883344252e7
```



Analyse des résultats

La méthode ne parvient pas à converger en moins de 5000 itérations. Toutefois, on constate une amélioration de la valeur de l'objectif qui est 10x plus petite que celle obtenue via Ipopt. Le temps de calcul est toutefois beaucoup plus élevé et on en déduit que cette méthode a une plus grande complexité que l'algorithme précédent. Nous estimons aussi que l'erreur dans notre modélisation JUmP est la cause de cette divergence. Il se pourrait que le problème ne soit plus convexe à cause de notre erreur. Nous étudiront cette hypothèse dans la phase 3.

Comme le nombre de variable est différent que dans le problème formulé et présenté dans l'article, il est difficile de comparer la complexité des deux algorithmes. Lorsque le temps de calcul est limité, il est toutefois préférable d'utiliser la méthode présentée dans l'article. Les graphes présentes une erreur qui est encore une fois espectable. On observe par aiileurs que la trajectoire en y est bien meilleure avec COSMO qu'avec Ipopt. Encore une fois, on observe de plus grandes erreurs là où il y a des changements dans la valeur des dérivées de la trajectoire à suivre. Bien que la valeur optimale obtenue via COSMO soit bonne, il est possible de faire mieux avec le solveur OSQP.

Partie 5 : Résolution avec le solver OSQP

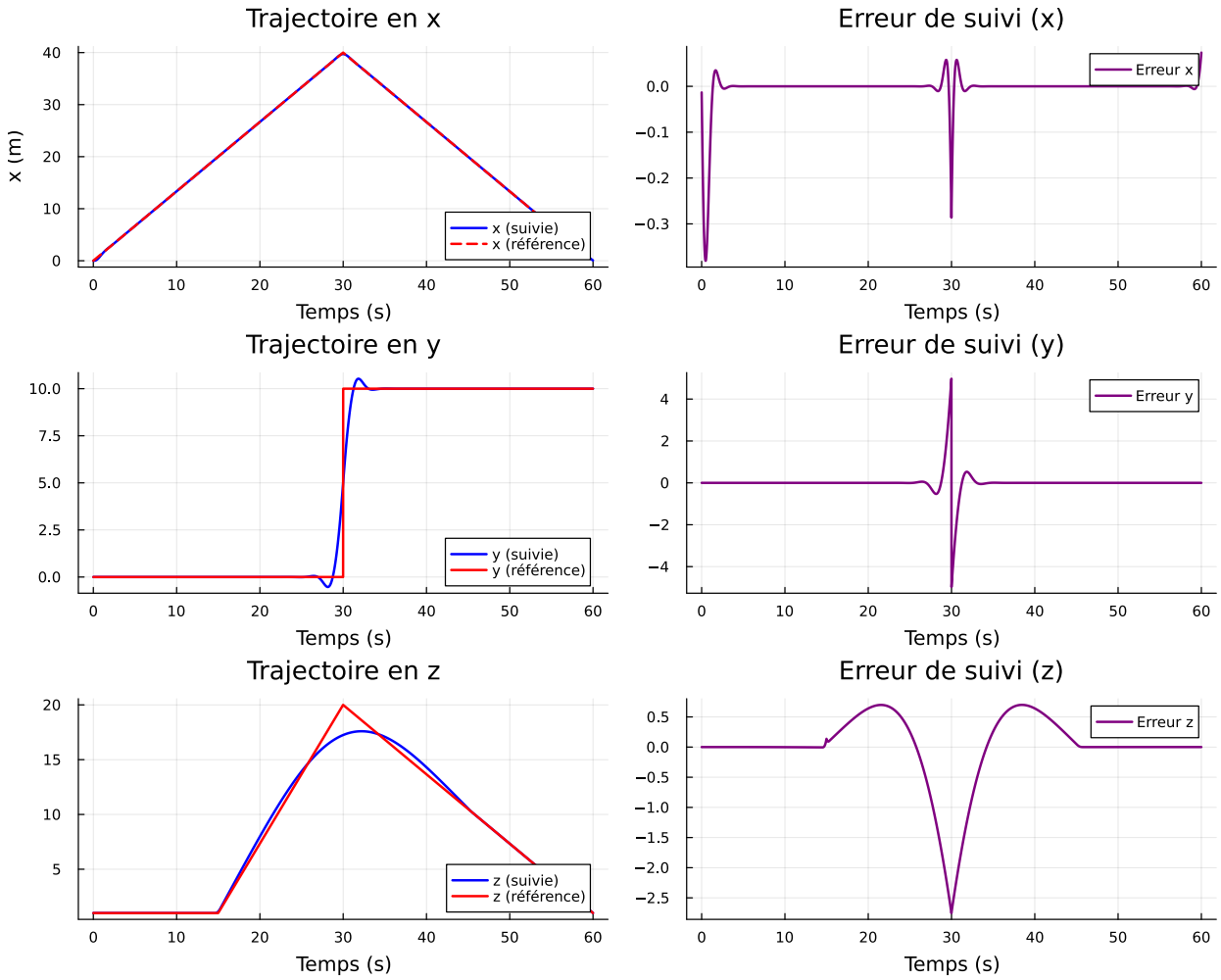
OSQP (Operator Splitting Quadratic Program) est une méthode servant a résoudre des problème quadratique à contraintes linéaires. C'est exactement les propriétés que nous souhaitons exploiter pour le problème monolithique.

```
using LinearAlgebra, Plots, JuMP, Ipopt, COSMO, IterativeSolvers, LinearMaps, OSQP, MathOptInterface

model2 = Model(OSQP.Optimizer)
set_optimizer_attribute(model, "print_level", 1)

@variable(model2, X[1:n, 0:T])           # états
@variable(model2, U[1:m, 0:T-1])         # commandes

# - dynamique linéaire
for k in 0:T-1
   @constraint(model2, X[:, k+1] .== Ad * X[:, k] + Bd * U[:, k])
end

# - CONTRAINTE d'état initial  (= hover)
@constraint(model2, X[:, 0] .== x0)

# - bornes (angles ±20°, poussée 0-2 mg)
deg20 = deg2rad(20.0)
@constraint(model2, -deg20 .<= X[1, :] .<=  deg20)   #
@constraint(model2, -deg20 .<= X[3, :] .<=  deg20)   #
@constraint(model2, -deg20 .<= X[5, :] .<=  deg20)   #
@constraint(model2, 0       .<= U[1, :] .<= 2 * mass * g)

# - coût
@expression(model2, running_cost,
   sum( (X[:, k] - r[:, k+1])' * Q * (X[:, k] - r[:, k+1]) +
        U[:, k]' * R * U[:, k]           for k in 0:T-1) )

@expression(model2, terminal_cost,
   (X[:, T] - r[:, T+1])' * F * (X[:, T] - r[:, T+1]) )

@objective(model2, Min, 0.5 * running_cost + 0.5 * terminal_cost)
```

```
# - point initial
set_start_value.(X[:, 0], x0)
set_start_value.(U, 0.0)

optimize!(model2)
println("Status          : ", termination_status(model2))
println("Objective value : ", objective_value(model2))

# ================================
# Visualisation de la trajectoire et de l'erreur
# ================================

X_val = value.(X)
U_val = value.(U)

J_state4   = 0.0
J_control4 = 0.0

for k in 1:T
   e_k = X_val[:,k] - r[:,k]
   J_state4 += 0.5 * (e_k' * Q * e_k)[1]
end

for k in 0:T-1
   J_control4 += 0.5 * (U_val[:,k]' * R * U_val[:,k])[1]
end

e_final = X_val[:,T] - r[:,T+1]
J_terminal4 = 0.5 * (e_final' * F * e_final)[1]

J4 = J_state4 + J_control4 + J_terminal4

println("\n--- Décomposition du coût : Modèle 4 ---")
println("J_state4    = ", J_state4)
println("J_control4  = ", J_control4)
println("J_terminal4 = ", J_terminal4)
println("J4 total    = ", J4)

solutionX = Array(value.(X))

z = solutionX[11, :]  # position verticale
x = solutionX[7, :]   # position x
y = solutionX[9, :]   # position y

# Erreurs de suivi
e_z = z .- r[11, :]
e_x = x .- r[7, :]
e_y = y .- r[9, :]
z = solutionX[11, :]  # position verticale
x = solutionX[7, :]   # position x
y = solutionX[9, :]   # position y

# Erreurs de suivi
```

```
e_z = z .- r[11, :]
e_x = x .- r[7, :]
e_y = y .- r[9, :]

# Références
r_z = r[11, :]
r_x = r[7, :]
r_y = r[9, :]

t = Ts .* (0:T)  # temps réel en secondes

# Création du layout 3 lignes × 2 colonnes
plt4 = plot(layout = (3, 2), size=(1000, 800))

# Trajectoire x
plot!(plt4[1], t, x, lw=2, label="x (suivie)", color=:blue)
plot!(plt4[1], t, r_x, lw=2, label="x (référence)", linestyle=:dash, color=:red)
plot!(plt4[1], title="Trajectoire en x", xlabel="Temps (s)", ylabel="x (m)", legend=:bottomright, grid=

# Erreur x
plot!(plt4[2], t, e_x, lw=2, label="Erreur x", color=:purple)
plot!(plt4[2], title="Erreur de suivi (x)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g

# Trajectoire y
plot!(plt4[3], t, y, lw=2, label="y (suivie)", color=:blue)
plot!(plt4[3], t, r_y, lw=2, label="y (référence)", inestyle=:dash, color=:red)
plot!(plt4[3], title="Trajectoire en y", xlabel="Temps (s)", label="y (m)", legend=:bottomright, grid=t

# Erreur y
plot!(plt4[4], t, e_y, lw=2, label="Erreur y", color=:purple)
plot!(plt4[4], title="Erreur de suivi (y)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g

# Trajectoire z
plot!(plt4[5], t, z, lw=2, label="z (suivie)", color=:blue)
plot!(plt4[5], t, r_z, lw=2, label="z (référence)", inestyle=:dash, color=:red)
plot!(plt4[5], title="Trajectoire en z", xlabel="Temps (s)", label="z (m)", legend=:bottomright, grid=t

# Erreur z
plot!(plt4[6], t, e_z, lw=2, label="Erreur z", color=:purple)
plot!(plt4[6], title="Erreur de suivi (z)", xlabel="Temps (s)", label="Erreur (m)", legend=:topright, g

# Affichage
display(plt4)
```

```
Status          : ALMOST_OPTIMAL
Objective value : 1.217425255507946e6

--- Décomposition du coût : Modèle 4 ---
J_state4    = 1.217266816725533e6
J_control4  = 354.15875488108844
J_terminal4 = 0.9023900755792362
J4 total    = 1.2176218778704896e6
```



Analyse des résultats

Notons que cette méthode est bien plus efficace que Ipopt et COSMO. La trajectoire désirée est en général mieux approximée, le temps de calcul est meilleur que COSMO en restant très proche de celui d'Ipopt. De plus, la valeur optimale de l'objectif est dix fois plus petite que pour COSMO et cent fois plus petite qu'Ipopt. Cette méthode est donc à privilégier par rapport aux deux méthodes précédentes.

#Description de ce qui à été réalisé

Nous avons reproduit les résultats du papier puis tester différentes méthodes de résolution pour la même trajectoire demandée. Cela nous permets de comparer les méthodes sur leurs performances par rapport à la fonction objectif mais aussi par rapport au temps de calcul.

Les résultats sont résumé dans le tableau ci-dessous :
Résultats Comparatifs des 4 Modèles

```
using PrettyTables

modeles = ["Modèle de Riccati", "Ipopt", "Cosmo", "OSQP"]

# Exemples de valeurs (remplace-les)
temps_sec = [0.811, 1.243, 0.400, 0.950]
J_state   = [J_state1, J_state2, J_state3, J_state4]
J_control = [J_control1, J_control2, J_control3, J_control4]
```

```
J_terminal= [J_terminal1, J_terminal2, J_terminal3, J_terminal4]
J_total   = J_state .+ J_control .+ J_terminal

data = hcat(
    modeles,
    round.(temps_sec, digits=3),
    round.(J_state, digits=3),
    round.(J_control, digits=3),
    round.(J_terminal, digits=3),
    round.(J_total, digits=3)
)

header = ["Modèle", "Temps (s)", "J_state", "J_control", "J_terminal", "J_total"]

println(pretty_table(String, data; header=header))
```

```
            Modèle   Temps (s)     J_state   J_control   J_terminal     J_total

 Modèle de Riccati       0.811   2.26343e5     772.249        0.902   2.27116e5
             Ipopt       1.243   1.15389e8     482.865      77704.2   1.15467e8
             Cosmo         0.4   1.04815e7     142.195         0.97   1.04817e7
              OSQP        0.95   1.21727e6     354.159        0.902   1.21762e6
```

Difficultés rencontrées

Pour enrichir le projet, nous avons décidé de faire une recherche pour inclure des algorithmes qui ne sont pas présentés dans le cours. Plusieurs de ces méthodes existent mais ne possèdent pas nécessairement une implémentation simple en Julia ou n'ont pas de documentations claires. Nous avons finalement décidé d'arrêter notre choix sur 2 méthodes: Conic operator splitting method (COSMO) et Operator splitting Quadratic Program (OSQP).

Nous n'arrivons toujours pas à expliquer pourquoi Ipopt ne converge pas dans la troisième dimension (l'axe z). Nous étudions actuellement notre code pour comprendre ce qui génère cette erreur. Comme nous utilisons la même modélisation JUmP pour OSQP et COSMO, nous supposons que cette même erreur rend plus difficile la convergence et c'est pourquoi COSMO ne converge pas en 5000 itérations. Comme son objectif est actuellement meilleurs que celui d'Ipopt, nous avons décidé de le conserver, en espérant pouvoir obtenir de résultats plus satisfaisant lorsque l'erreur de modélisation sera trouvée.

Description détaillée de ce qui a déjà été accompli :

Pour l'instant, le code a été développé pour recréer les résultats de l'article. Nous avons donc implémenté la méthode récursive de riccati et l'optimisation all at once (cas Ipopt). Nous avons aussi débuté la comparaison avec des méthodes trouvées en ligne mais nous nous frappons à des problèmes de convergences qui seront résolus nous l'espérons d'ici la prochaine phase. Il reste encore à implémenter la résolution de l'optimisation all at once avec un algorithme présenté en cours.

Nous avons complété les consignes de la phase 1 qui étaient manquantes (voir annexe). Cette annexe inclus une description de la problématique, une description des objectifs, une description d'un plan d'action et unedescription de l'impact attendu. Dans la première section de ce rapport, nous avons révisé et présenté la description de la problématique.

Comme nous rencontrons des problèmes de modélisation, nous pouvons difficilement comparer les résultats

obtenus des différentes méthodes. Nous avons toutefois résumé les résultats temporaires que nous avons dans un tableau et conservé notre ébauche d'analyse des résultats, que nous avons quelque peu amélioré. Nous ne voulions pas passer trop de temps sur cela tant que l'erreur n'est pas corrigée. Nous estimons que la majorité du code est déjà écris en ce sens que tout devrait bien fonctionné une fois l'erreur de modélisation corrigée.

Description détaillée de ce qu'il reste à accomplir.

Prioritaire:

Résoudre l'erreur de modélisation se trouvant dans le code. Une fois l'erreur corrigée, nous pourrons procéder à une analyse plus approfondie de nos résultats. Nous espérons avoir accomplis ceci d'ici une semaine.

Secondaire :

Si le temps le permet, nous désirons choisir et implémenter une méthode dédiée aux problèmes quadratiques présentées en cours afin d'enrichir la comparaison des différentes méthodes et de tirer de meilleures conclusions sur la problématique du projet.

Une semaine avant la remise :

Faire une analyse détaillée des résultats que nous avons obtenus à partir des méthodes que nous auront réussi à implémenter à ce moment là. Les méthodes qui ne seront pas implémentée correctement seront abandonnée.

2 jours avant la remise :

Écrire notre perspectice personnelle sur le projet, dire quels objectifs initiaux ont étés accomplis, compléter la section portant sur la description des difficultés que nous avons rencontrés. Finaliser le rapport, apporter des corrections mineures et faire des retouches.

Lien Github

Cliquer ici pour le lien GitHub

Annexe (Partie 1)

Nous n'avons pas respecté les consignes demandées dans la partie 1. Voici donc les éléments de cette parties qui ne sont pas nécessairement demandées dans la partie 2 mais que nous avons jugé utile de rajouter car il s'agit tout de même d'éléments importants à la claireté de notre projet.

Description de la problématique :

Le projet consiste à optimiser la trajectoire d'un drone quadrotor à l'aide de méthodes de commande optimale. Les principeaux défis sont :

- La non-linéarité et l'instabilité naturelle du drone ;
- La consommation énergétique élevée liée à la poussée ;
- Le respect des contraintes physiques sur les angles pour conserver la validité du modèle.

Description des objectifs : par exemple, quels résultats nous pensons obtenir ou vérifier

Les objectifs du projet sont :

- Modéliser et linéariser la dynamique d'un drone quadrotor ;
- Développer et simuler un algorithme de commande optimale (LQR/Riccati) sur un modèle discret ;
- Vérifier la capacité du contrôleur à :
  - Suivre des trajectoires 3D prédéfinies ;
  - Limiter la consommation énergétique ;
  - Respecter les contraintes sur les angles.

Les résultats attendus sont :

- Une trajectoire suivie proche de la référence avec des erreurs de suivi faibles ;
- Une commande optimale limitant l'énergie consommée ;
- Des visualisations des trajectoires et erreurs pour valider l'approche.

Description du plan d'action : où nous allons les informations nécessaires, quelles données vous allez utiliser, etc. ;

Le projet s'articule en plusieurs étapes :

- Modélisation du drone à partir des équations de Newton (translation + rotation) ;
- Simplification et linéarisation autour du vol stationnaire pour obtenir un modèle d'état linéaire ;
- Discrétisation du modèle et définition d'une fonction de coût quadratique (erreurs de suivi + effort de commande) ;
- Résolution du problème d'optimisation avec :
  - Algorithme récursif de Riccati pour l'optimisation sur horizon fini ;
  - Simulation forward pour évaluer le suivi de trajectoire ;
- Visualisation et analyse des résultats (trajectoires suivies, erreurs, consommation) ;
- (Optionnel) : optimisation avec JuMP et Ipopt pour comparer la solution LQR et une résolution all-at-once;
- (Additionnel) : optimisation avec JuMP et des algorithmes qui n'ont pas été présentés dans le cours pour comparer la solution LQR et une résolution all-at-once. Ces algortihmes seront trouvés en faisant une recherche sur le web.

Description de l'impact attendu.

Impact attendu :

- Amélioration de la précision de suivi de trajectoire ;
- Réduction de la consommation d'énergie ;
- Application potentielle à des drones autonomes plus performants pour la logistique, l'observation ou la recherche.