INSTITUT POLYTECHNIQUE DE PARIS
ENSTA PARIS

CSC_5RO16_TA, Planification et Contrôle

TP4, Model Predictive Control

by
Guilherme NUNES TROFINO

supervised by
Julien ALEXANDRE DIR SANDRETTO
David FILLIAT

Confidentiality Notice
Non-confidential and publishable report

ROBOTIQUE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET COMMUNICATION

Paris, FR
20 janvier 2025

Table des matières

1 Question 1 **2**
- 1.1 Algorithme . . . 2
- 1.2 Analyse . . . 3
  - 1.2.1 Méthode Réactive vs Méthode Anticipative . . . 3
  - 1.2.2 Variation `window_size` . . . 3

2 Question 2 **5**
- 2.1 Théorie . . . 5
  - 2.1.1 Système Description . . . 5
  - 2.1.2 Zone de Stabilité, Non Linéaire . . . 5
- 2.2 Algorithme . . . 6
- 2.3 Analyse . . . 7
  - 2.3.1 Zone de Stabilité . . . 7

3 Question 3 **8**
- 3.1 Théorie . . . 8
  - 3.1.1 Action Prédictive . . . 8
- 3.2 Algorithme . . . 9
- 3.3 Analyse . . . 10
  - 3.3.1 Commande Anticipative vs Commande Prédictive . . . 10

1. Question 1

1.1. Algorithme

Suite à la description donnée dans le Travaux Pratique et aux informations présentées pendant le cours, l'algorithme suivant a été implémenté pour contrôler de façon anticipative un vélo suivant une trajectoire :

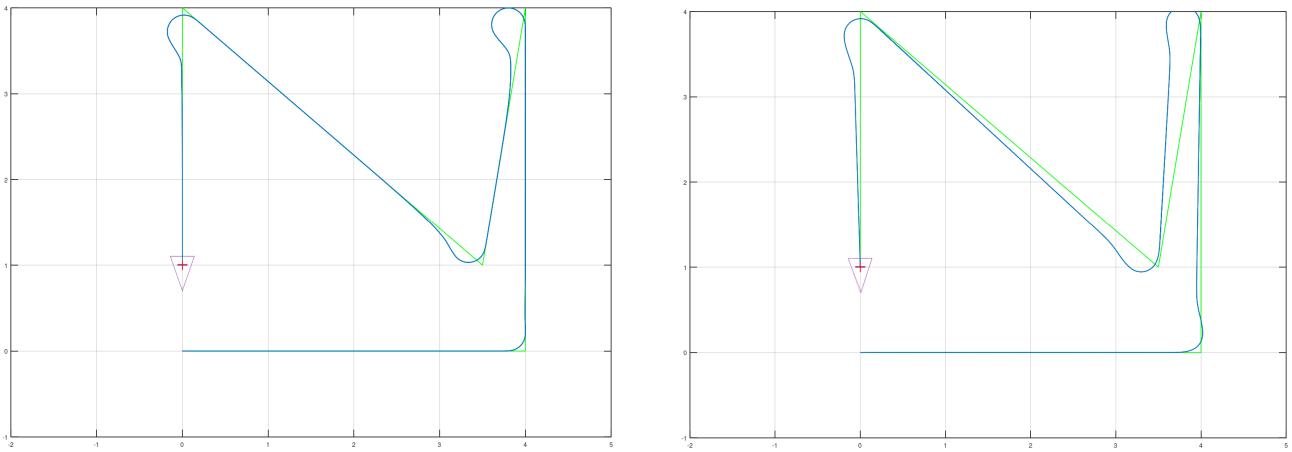
```
function [ u ] = BicycleToPathControl2( xTrue, Path, window_size )
    %Computes a control to follow a path for bicycle
    %   xTrue is the robot current pose : [ x y theta ]'
    %   Path is set of points defining the path : [ x1 x2 ...
    %                                              y1 y2 ...]
    %   u is the control : [v phi]'
    %   window_size is the antecipation window:
    %       1 : anticipe,
    %       5 : anticipe bien, controle plus souple,
    %      20 : coupe un peu, controle très souple,
    %     100 : coupe un peu,
    %    1000 : triche !

    persistent goalWaypointId xGoal;


    if xTrue == [0;0;0]
        goalWaypointId = 1;
        xGoal = Path(:,1);
    end

    rho = 0.3;
    dt = 0.01;


    vmax = 2.0;
    dmax = vmax * dt;

    list_points = [];
    xtemp = xTrue;

    while size(list_points, 2) < window_size
        if norm((Path(:, goalWaypointId) - xtemp)(1:2)) < rho
            xtemp = Path(:, goalWaypointId);
            list_points = [list_points, xtemp];

            goalWaypointId = goalWaypointId + 1;
            goalWaypointId = min(goalWaypointId, size(Path, 2));
        else
            direction = Path(:, goalWaypointId) - xtemp;
            direction = direction / norm(direction);
            xtemp = xtemp + dmax * direction;
            list_points = [list_points, xtemp];
        endif
    end


    anticipation = window_size;

    Krho = 10;
    Kalpha = 5;

    error = list_points(:, anticipation) - xTrue;
    goalDist = norm(error(1:2));
    AngleToGoal = AngleWrap(atan2(error(2), error(1)) - xTrue(3));

    u(1) = Krho * goalDist/(window_size * 10);
    u(2) = Kalpha * AngleToGoal;
end
```

Listing 1 : Algorithme `BicycleToPathControl2.m`

1.2. Analyse

1.2.1. Méthode Réactive vs Méthode Anticipative

Ci-dessous sont présentés le chemin obtenu par la méthode réactive proposée lors du dernier Travaux Pratique et le chemin obtenu par la méthode anticipative proposée dans ce Travaux Pratique.



(a) Méthode Réactive (b) Méthode Anticipative avec `window_size = 5`

FIGURE 1.1 : Comparaison entre Méthodes Réactives et Anticipative

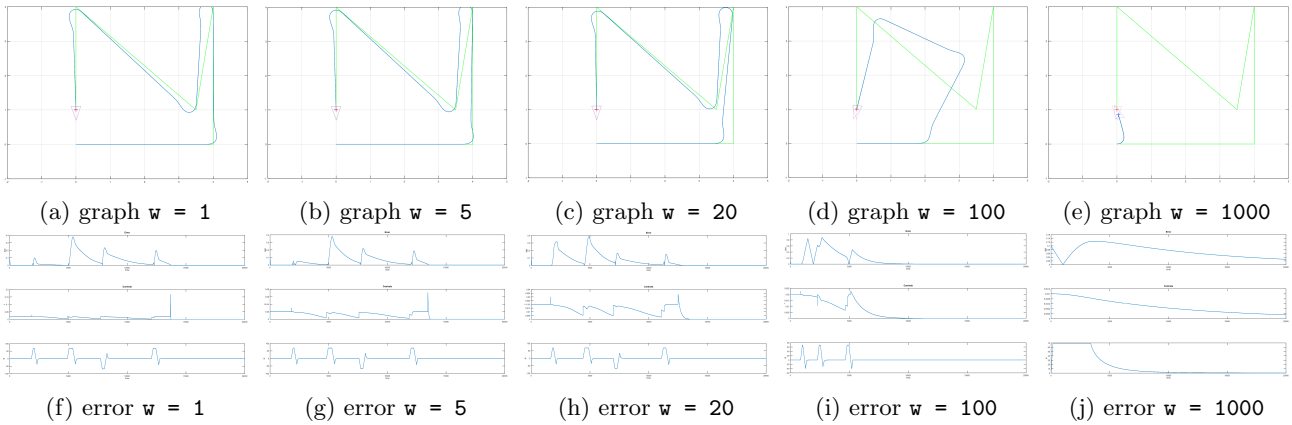
Il est remarquable que la méthode réactive est mieux adaptée à la trajectoire proposée, étant donné que le chemin suivi par la bicyclette et la trajectoire cible sont assez proches, avec une erreur réduite de moitié par rapport à celle de la méthode anticipative.

Ce phénomène est attendu, car la méthode réactive cherche constamment à rester autour de la trajectoire en utilisant des points intermédiaires pour s'ajuster. En revanche, la méthode anticipative utilise des points plus éloignés du chemin pour calculer ses commandes, ce qui peut augmenter l'erreur.

> **Remarque.** Dans ce cas particulier, la méthode réactive est plus performante. Cependant, il convient de noter que la méthode anticipative peut être plus efficace pour éviter des obstacles inattendus.

1.2.2. Variation `window_size`

Afin d'évaluer la performance de la méthode anticipative, différents `window_size` ont été testés. Les résultats de ces essais sont présentés ci-dessous :



(a) graph `w = 1` (b) graph `w = 5` (c) graph `w = 20` (d) graph `w = 100` (e) graph `w = 1000`

(f) error `w = 1` (g) error `w = 5` (h) error `w = 20` (i) error `w = 100` (j) error `w = 1000`

FIGURE 1.2 : Méthode Anticipative Comparaison par `window_size`

Remarque. Ici `window_size` était considéré comme `w`.

Les erreurs et les temps d'exécution sont présentés dans le tableau suivant :

window_size	error	time
1	867.6238	18.3457
5	865.9341	19.3566
20	1014.9327	23.2893
100	2405.0702	36.6221
1000	1410.1080	449.2239

TABLE 1.1 : Résultats d'exécution des Algorithmes

Il est notable que, pour des valeurs relativement petites de `window_size`, la méthode maintient une performance assez stable, avec des erreurs et des temps d'exécution proches. Cependant, à mesure que la taille de `window_size` augmente, les résultats se dégradent progressivement, jusqu'à atteindre un point où la trajectoire n'est plus suivie du tout

2. Question 2

2.1. Théorie

2.1.1. Système Description

Dans ce cas, le système considéré est décrit par les équations suivantes :

$$\boxed{\begin{aligned}\dot{x}_1 &= x_2 + \nu(\mu + 1(1-\mu)\, x_1)\\ \dot{x}_2 &= x_1 + \nu(\mu - 4(1-\mu)\, x_2)\end{aligned}} \tag{2.1}$$

Ce système peut être représenté sous forme matricielle, comme illustré ci-dessous :

$$\dot{\mathbf{x}}(k) = f(\mathbf{x}(k),\ \mathbf{u}(k)) \qquad \text{avec :} \quad \mathbf{x}(0) = \mathbf{x}_0, \quad \mathbf{x}(k) = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \text{et} \quad \mathbf{u}(k) = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \tag{2.2}$$

Où :

1. $\mathbf{x}(k)$ est le vecteur d'états du système ;
2. $\mathbf{u}(k)$ est le vecteur des commandes du système ;

> **Remarque.** Étant donné que le temps est discret, k représente les itérations de l'algorithme au fil du temps.

2.1.2. Zone de Stabilité, Non Linéaire

À partir des équations du système, il est nécessaire d'obtenir les **matrices Jacobiennes** pour linéariser le problème. Ces matrices permettront ensuite de vérifier la stabilité du système en utilisant les **équations de Lyapunov**, conformément aux principes abordés en classe et dans l'article de Chen & Allgower (1998), de la manière suivante :

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \dot{x}_1}{\partial x_1} & \cdots & \frac{\partial \dot{x}_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \dot{x}_n}{\partial x_1} & \cdots & \frac{\partial \dot{x}_n}{\partial x_n} \end{bmatrix} \quad \text{avec } A_{ij} = \left[\frac{\partial \dot{x}_i}{\partial x_j}\right] \tag{2.3}$$

$$\boxed{\mathbf{A} = \begin{bmatrix} \nu(1-\mu) & 1 \\ 1 & -4\nu(1-\mu) \end{bmatrix}}$$

$$\mathbf{B} = \begin{bmatrix} \frac{\partial \dot{x}_1}{\partial u_1} & \cdots & \frac{\partial \dot{x}_1}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial \dot{x}_n}{\partial u_1} & \cdots & \frac{\partial \dot{x}_n}{\partial u_m} \end{bmatrix} \quad \text{avec } B_{ij} = \left[\frac{\partial \dot{x}_i}{\partial u_j}\right] \tag{2.4}$$

$$\boxed{\mathbf{B} = \begin{bmatrix} \mu + 1(1-\mu)\, x_{1_0} \\ \mu - 4(1-\mu)\, x_{2_0} \end{bmatrix}}$$

Par la suite, il est indispensable le calculer un retour d'état linéaire **localement** stabilisant noté $\mathbf{K}$ qui résout l'équation $\mathbf{u} = \mathbf{K}\,\mathbf{x}$. Pour obtenir ce retour d'état, les **Équations de Riccati** doivent être résolues à l'aide de la fonction `[x, l, g] = care(A, B, Q, R)` dans Octave.

Ensuite, il est nécessaire de calculer la constante positive α, qui doit être inférieure à l'opposé de la valeur propre maximale de la matrice $\mathbf{A}_k = \mathbf{A} + \mathbf{B} \times \mathbf{K}$. Une fois α déterminée, l'équation de Lyapunov suivante doit être résolue :

$$(\mathbf{A}_k + \alpha\mathbf{I})^\top \mathbf{P} + \mathbf{P}\,(\mathbf{A}_k + \alpha\mathbf{I}) = -\left(\mathbf{Q} + \mathbf{K}^\top \mathbf{R}\mathbf{K}\right) \tag{2.5}$$

Finalement, les points stabilisants sont ceux qui correspondent aux conditions suivantes :

$$\boxed{\mathbf{x}^\top \mathbf{P}\mathbf{x} \leq \beta} \tag{2.6}$$

Où β est obtenu par la fonction `qp()` du Octave.

2.2. Algorithme

Après l'explication théorique donnée précédemment, l'algorithme suivant a été implémenté :

```
function ok = verify_stability(x_verif)
    try
    pkg load control
    end

    % Parameters
    mu = 0.5;
    u0 = 0;
    x10 = 0;
    x20 = 0;

    % Weight Matrices
    Q=[0.5, 0; 0, 0.5];
    R=[1];

    % Linearised Jacobians
    A = [u0 * (1 - mu), 1; 1, -u0 * 4 * (1 - mu)];
    B = [mu + (1 - mu) * x10; +mu - 4 * (1 - mu) * x20];

    % Riccati Equations
    [x, l, g] = care(A, B, Q, R);
    K = -g
    % disp(K)


    % Feedback Loop
    Ak = A + B * K;
    % disp(eig(Ak))

    M = [-1, 0; 0, -1] - (Ak);
    % disp(det(M));


    % Limited Lambda
    lambda = -max(eigs(Ak));
    alpha = 0.95 * lambda;

    % Lyapunov Equations
    Al = (Ak + [alpha, 0; 0, alpha])';
    Bl = (Q + K' * R * K);
    assert(all(real(eig(Al)) < 0), "Al is not stable.");
    assert(issymmetric(Bl), "Bl is not symmetric.");
    assert(all(eig(Bl) >= 0), "Bl is not positive semi-definite.");

    P = lyap(Al, Bl);
    assert(issymmetric(P), "P is not symmetric.");


    % Quadratic Problem
    [X, OBJ, INFO, LAMBDA] = qp([0.5; 0.5], -2*P, [], [], [], [-0.8; -0.8], [+0.8; +0.8], -2, K, 2);
    beta = -OBJ;


    % Controller Stable Zone
    test = x_verif' * P * x_verif;
    ok = (test < beta);


endfunction
```

Listing 2 : Algorithme `verify_stability.m`

2.3. Analyse

2.3.1. Zone de Stabilité

Suite à l'exécution de l'algorithme, la zone de stabilité suivante a été obtenue :

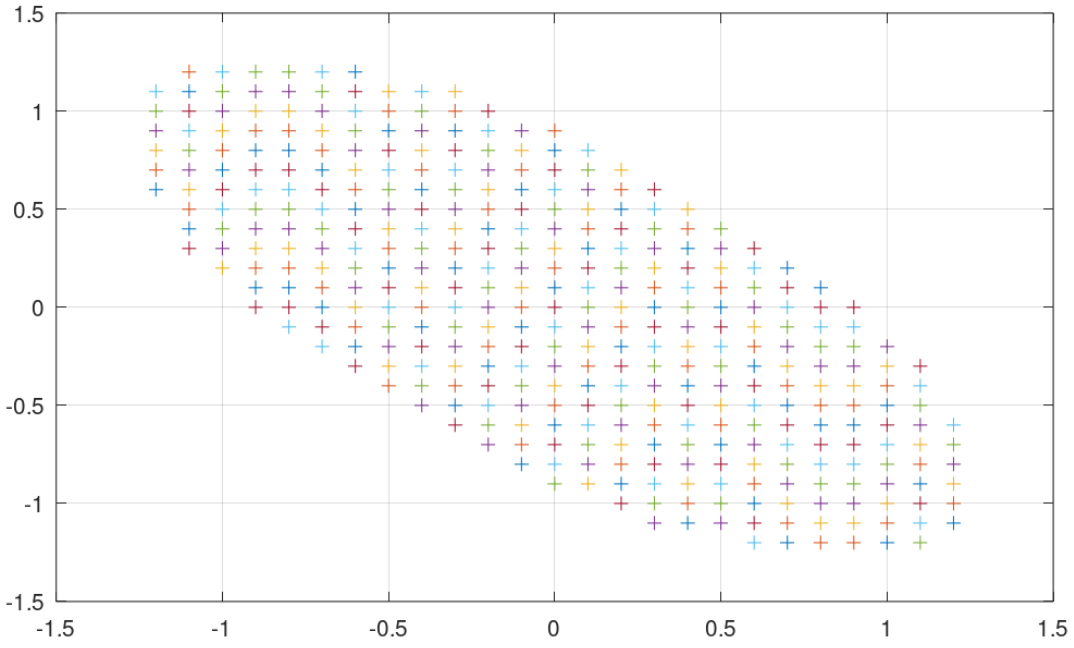


FIGURE 2.1 : Zone de Stabilité du Méthode Prédictive

Tous les points situés à l'intérieur de l'ellipse représentent des configurations où la commande du système est stable.

3. Question 3

3.1. Théorie

Le Modèle de Contrôle Predictive, ou **MPC** (Model Predictive Control) en anglais, est une méthode permettant de calculer les commandes optimales d'un système en résolvant un problème d'optimisation convexe.

3.1.1. Action Prédictive

Le MPC est applicable aux systèmes non linéaires à l'application d'une linéarisation, décrite comme suit :

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t),\ \mathbf{u}(t)) \qquad \text{avec} \quad \mathbf{x}(0) = \mathbf{x}_0 \\ \dot{\mathbf{x}}_r(t) &= f(\mathbf{x}_r(t),\ \mathbf{u}_r(t)) \end{aligned} \tag{3.1}$$

Dans cette approche, $\mathbf{x}_r$ et $\mathbf{u}_r$ représentent respectivement la trajectoire de la référence et la commande référence à suivre. L'équation du système est linéarisée à l'aide de développement en série de Taylor à l'ordre 1, évaluée en $(\mathbf{x}_r,\ \mathbf{u}_r)$:

$$\dot{\mathbf{x}} = f(\mathbf{x}_r,\ \mathbf{u}_r) + (\mathbf{x} - \mathbf{x}_r)\frac{\partial f(\mathbf{x},\ \mathbf{u})}{\partial \mathbf{x}}\bigg|_{(\mathbf{x}_r,\ \mathbf{u}_r)} + (\mathbf{u} - \mathbf{u}_r)\frac{\partial f(\mathbf{x},\ \mathbf{u})}{\partial \mathbf{u}}\bigg|_{(\mathbf{x}_r,\ \mathbf{u}_r)} \tag{3.2}$$

> **Remarque.** Ici, $\mathbf{x}$ correspond à $\mathbf{x}(t)$, mais pour simplifier la notation, la dépendance temporelle explicite n'est pas indiquée.

Après l'expansion de Taylor, les équations suivantes sont adoptées pour résoudre le problème de contrôle :

$$\begin{aligned} \dot{\widetilde{\mathbf{x}}} &= \mathbf{f}_{\mathbf{x},\,\mathbf{r}} + \mathbf{f}_{\mathbf{u},\,\mathbf{r}}\widetilde{\mathbf{u}} \qquad \text{avec} \qquad \begin{cases} \widetilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}_r \\ \widetilde{\mathbf{u}} = \mathbf{u} - \mathbf{u}_r \end{cases} \\ \boxed{\dot{\widetilde{\mathbf{x}}}(t) &= \mathbf{A_r} + \mathbf{B_r}\widetilde{\mathbf{u}}(t)} \end{aligned} \tag{3.3}$$

Ensuite, une discrétisation est effectuée. En appliquant le schéma d'Euler, les équations discrètes suivantes sont obtenues :

$$\begin{aligned} \widetilde{\mathbf{x}}(k+1) &= \widetilde{\mathbf{x}}(k) + (\mathbf{A_r}\widetilde{\mathbf{x}}(k) + \mathbf{B_r}\widetilde{\mathbf{u}}(k))\delta t \\ \widetilde{\mathbf{x}}(k+1) &= \mathbf{A}(k)\widetilde{\mathbf{x}}(k) + \mathbf{B}(k)\widetilde{\mathbf{u}}(k) \qquad \text{avec} \qquad \begin{cases} \mathbf{A}(k) = \mathbf{A}_r\delta t + \mathbf{I} \\ \mathbf{B}(k) = \mathbf{B}_r\delta t \end{cases} \end{aligned} \tag{3.4}$$

Pour utiliser un seul système linéaire dans la fonction d'optimisation `qp()` (Quadratic Programming) d'Octave, le système est vectorisé :

$$\mathbf{X}(k) = \underbrace{\begin{bmatrix} \mathbf{A}(k)^1 \\ \mathbf{A}(k)^2 \\ \vdots \\ \mathbf{A}(k)^N \end{bmatrix}}_{\hat{\mathbf{A}}} \widetilde{\mathbf{x}}(k) + \underbrace{\begin{bmatrix} \mathbf{B}(k) & 0 & \cdots & 0 \\ \mathbf{A}(k)^1\mathbf{B}(k) & \mathbf{B}(k) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}(k)^{N-1}\mathbf{B}(k) & \mathbf{A}(k)^{N-2}\mathbf{B}(k) & \cdots & \mathbf{B}(k) \end{bmatrix}}_{\hat{\mathbf{B}}} \mathbf{U}(k\ :\ k+N) \tag{3.5}$$

Finalement, la commande recherchée est celle qui minimise l'erreur de trajectoire. Une approche par moindres carrés est adoptée pour déterminer cette commande optimale :

$$\mathbf{U}(k\ :\ k+N) = -\hat{\mathbf{B}}^{\sharp}\ \hat{\mathbf{A}}\ \widetilde{\mathbf{x}}(k) \tag{3.6}$$

> **Remarque.** Ici $\hat{\mathbf{B}}^{\sharp}$ représente la pseudo-inverse de $\hat{\mathbf{B}}$.

3.2. Algorithme

Après l'explication théorique donnée précédemment, l'algorithme suivant a été implémenté :

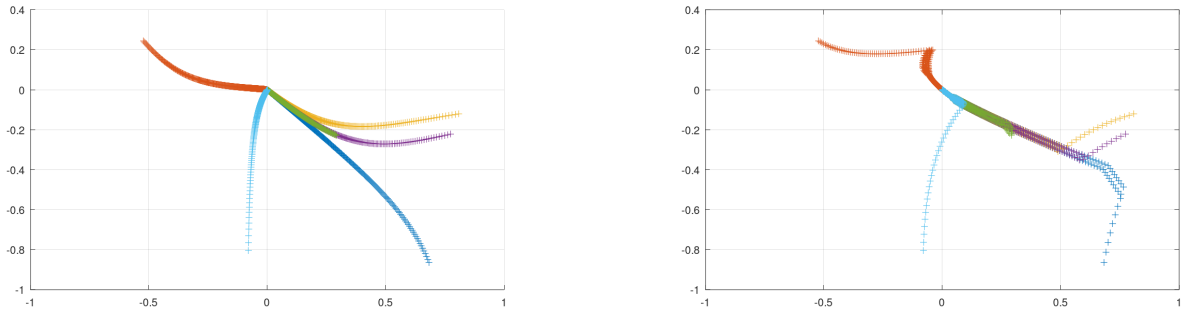
```
function simulateMPC(xinit,K,mu)

    dt = 0.01;
    x = xinit;
    xstore = NaN * zeros(2,10000);
    k = 1;
    u = 1;
    n = 4;

    while (norm(x) > 0.001) && (k<2000)
        xstore(:,k) = x;


        %TODO linearisation en x et t
        A = [  u * (1 - mu), 1; 1, - u * 4 * (1 - mu) ];
        B = [ mu + (1 - mu) * x(1); mu - 4 * (1 - mu) * x(2) ];

        A = A*dt + eye(2, 2);
        B = B*dt;

        %vecteur d'entrées
        U=[u,u,u,u]';
        H=eye(n,n)*2;

        %TODO écrire les matrices de la commande prédictive linéaire
        A_hat = [...
            A;...
            A^2;...
            A^3;...
            A^4];
        B_hat = [...
            B, zeros(2, n-1);...
            A^1 * B, B, zeros(2, n-2);...
            A^2 * B, A^1 * B, B, zeros(2, n-3);...
            A^3 * B, A^2 * B, A^1 * B, B];

        %TODO avec une pseudo inverse, calculer le vecteur d'entrées
        U = pinv(B_hat) * (-A_hat * x);


        if size(K) == 0
          u = U(1);
        else
          u = -K*x;
        endif

        if (u > 2)
          u = 2;
        elseif (u<-2)
          u = -2;
        endif

        %simu avec euler
        x1 = x(1);
        x2 = x(2);
        x(1) = x1 + dt*(x2 + u*(mu + 1*(1-mu)*x1));
        x(2) = x2 + dt*(x1 + u*(mu - 4*(1-mu)*x2));

        k++;
    endwhile

    if norm(x) < 0.01
        plot(xstore(1,:),xstore(2,:),'+');
    else
        disp("fail!")
    endif
endfunction
```

Listing 3 : Algorithme `simulateMPC.m`

3.3. Analyse

3.3.1. Commande Anticipative vs Commande Prédictive

Ci-dessous, il est possible de comparer les différentes méthodes de contrôle appliquées au même système :



(a) Méthode Anticipative

(b) Méthode Prédictive

FIGURE 3.1 : Méthode Prédictive Commande Comparaison

Il est notable que la méthode anticipative offre une convergence relativement fluide et douce. En revanche, la méthode prédictive présente un comportement moins stable, bien que le résultat final soit équivalent dans les deux cas.

L'instabilité observée dans la méthode prédictive est attribuée à l'utilisation de la pseudo-inverse, une technique moins précise par nature que les solveurs quadratiques employés dans la méthode anticipative.