Projet : Amélioration de l'asservissement de visée laser

Note d'application

Partie Contrôle : Identification SBPA

Élève :
Marouane HSAINI

Enseignant Référent :
Jacques Laffont
Client :
JTL Electronique

Table des matières

Table des figures

Introduction

Cette note d'application vise à détailler le processus d'identification SBPA (Séquence Binaire Pseudo-Aléatoire) et à expliquer ses différentes étapes de manière précise et structurée. Dans un premier temps, les objectifs de l'identification seront exposés, suivis d'une présentation des deux méthodes employées : la méthode de la Pseudo-Inverse et celle des Variables Instrumentales. Par la suite, nous décrirons l'algorithme d'Adam, son fonctionnement et sa pertinence dans le cadre de cette application. Enfin, nous afficherons les résultats de l'identification obtenus pour le système étudié.

I Objectif

L'objectif consiste à exciter le système avec une SBPA afin de redéterminer les coefficients du système selon un modèle donné.

II Simulation d'identifications SBPA selon différentes méthodes

Pour l'identification, nous avons prévu d'essayer 2 méthodes SBPA différentes (méthode de la Pseudo-inverse et méthode des variables instrumentales) afin de trouver laquelle nous donnerait les meilleurs résultats lorsque l'on fera l'identification réelle des galvanomètres.

1 Identification SBPA par la méthode de la pseudo-inverse

La méthode de la pseudo-inverse consiste à écrire l'équation récurrente d'un système d'un ordre donné à partir de sa transmittance (qui elle-même est obtenue à partir de la fonction de transfert en continu du système) :

$$G(z) = \frac{S(z)}{E(z)} = \frac{b_1 z^{-1} + ... + b_m z^{-m}}{1 + a_1 z^{-1} + ... + a_n z^{-n}}$$

Pour l'échantillon k :

$$S_k = -a_1 S_{k-1} + ... - a_n S_{k-n} + b_1 E_{k-1} + ... + b_m E_{k-m}$$

puis à considérer un ensemble d'échantillons pour ensuite mettre ces égalités sous forme matricielles :

$$\underbrace{\begin{bmatrix} S_k \\ S_{k-1} \\ S_{k-2} \\ \vdots \end{bmatrix}}_{V} = \underbrace{\begin{bmatrix} -S_{k-1} & \cdots & -S_{k-n} & E_{k-1} & \cdots & E_{k-m} \\ -S_{k-2} & \cdots & -S_{k-n-1} & E_{k-2} & \cdots & E_{k-m-1} \\ -S_{k-3} & \cdots & -S_{k-n-2} & E_{k-3} & \cdots & E_{k-m-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}}_{M} \times \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_n \\ b_1 \\ \vdots \\ b_m \end{bmatrix}}_{\Theta}$$

On observe que l'on obtient une équation $V = M\Theta$.

L'objectif d'une identification est de trouver Θ qui contient l'ensemble des coefficients de la transmittance de notre système.

On remarque alors que si M est une matrice carrée inversible, il suffit de calculer :

$$M^{-1}V = \Theta$$

Pour le cas où M ne serait pas une matrice carrée, on peut alors utiliser la pseudo-inverse de Moore-Penrose :

$$M^+ = \left(M^T \times M\right)^{-1} \times M^T$$
$$\rightarrow \Theta = M^+ V$$

La pseudo-inverse de Moore-Penrose ne peut être utilisée que si $\left(M^T \times M\right)^{-1}$ est inversible.

Pour créer notre vecteur V et notre matrice M, nous allons fournir en entrée de notre système un signal SBPA dans la plage de fonctionnement de nos galvanomètres dont nous prendrons soin de garder les valeurs et échantillonnerons également la sortie engendrée.

2 Identification SBPA par la méthode des variables instrumentales

Pour la méthode des variables instrumentales, la procédure est semblable à celle de la pseudo-inverse mais nous remplacerons la pseudo-inverse de M (M^+) par une matrice M' telle que :

$$M' = \left(M_{vi}^T \times M\right)^{-1} \times M_{vi}^T$$

La matrice M_{vi} est généralement construite en prenant la matrice M et en décalant les sorties d'un échantillon :

$$M = \begin{bmatrix} -S_{k-1} & \cdots & -S_{k-n} & E_{k-1} & \cdots & E_{k-m} \\ -S_{k-2} & \cdots & -S_{k-n-1} & E_{k-2} & \cdots & E_{k-m-1} \\ -S_{k-3} & \cdots & -S_{k-n-2} & E_{k-3} & \cdots & E_{k-m-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$\rightarrow M_{vi} = \begin{bmatrix} -S_{k-2} & \cdots & -S_{k-n-1} & E_{k-1} & \cdots & E_{k-m} \\ -S_{k-3} & \cdots & -S_{k-n-2} & E_{k-2} & \cdots & E_{k-m-1} \\ -S_{k-4} & \cdots & -S_{k-n-3} & E_{k-3} & \cdots & E_{k-m-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

L'avantage de cette méthode est qu'en présence de bruit, ce dernier n'est plus multiplié au carré comme il pouvait l'être pour certains termes avec la méthode de la pseudo-inverse, ce qui devrait diminuer son influence et nous donner donc de meilleurs résultats lors de l'identification du système réel.

3 Simulation des identifications

Nous avons réalisé les 2 méthodes d'identification pour un ordre 3 et avons simulé leurs fonctionnement en nous resservant des scripts pour simuler des systèmes d'ordres 1 à 3 en introduction à Scilab. Une fois que nous avions nos vecteurs de valeurs d'entrées et de valeurs de sorties, nous avons pu les utiliser comme argument pour nos fonctions et avons constaté le bon fonctionnement de ces dernières.

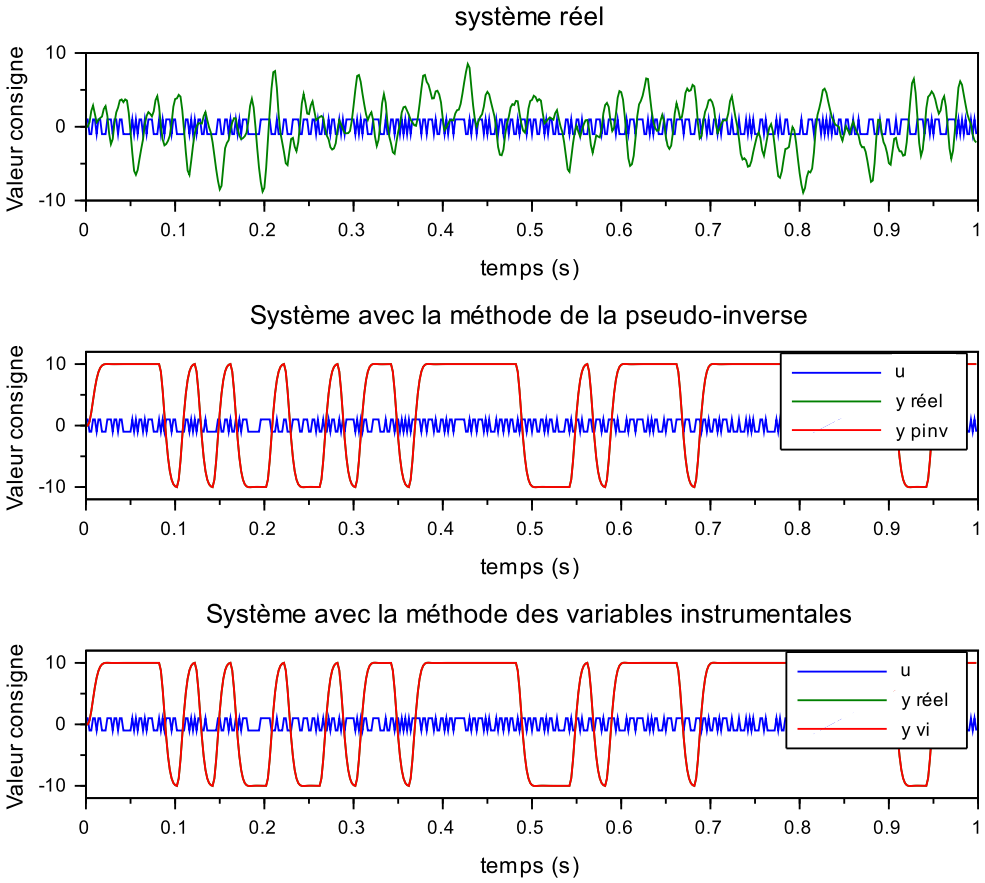


FIGURE II.1 – Simulation de l'identification d'un système d'ordre 3 avec les différentes méthodes sans bruit

```
"paramètres discrets réels :"
-1.7326914
 1.0816131
-0.2417401
 0.6180884
 0.4537274
"identification pinv"
-1.7326914
 1.0816131
-0.2417401
 0.6180884
 0.4537274
"identification variables instrumentales"
-1.7326914
 1.0816131
-0.2417401
 0.6180884
 0.4537274
"erreur Pseudo-inverse :"
 1.255D-27
"erreur Variable instrumentale :"
 1.141D-25
```

FIGURE II.2 – Comparaison des coefficients du système d'ordre 3 avec les différentes méthodes

La figure ci-dessus nous montre que les deux méthodes fonctionnent très bien avec une superposition des signaux par rapport au signal du sytème réel.

4 Simulation des identifications pour un signal soumis à un bruit blanc

Maintenant que nous avions ces méthodes d'identification fonctionnelles, nous avons pu nous intéresser à leur efficacité en présence de bruit.

En cherchant dans nos cours d'automatique, nous avons retrouvé une condition essentielle à l'identification des systèmes en présence de bruit qui est que la fréquence d'échantillonnage du système doit être choisie au cas par cas et doit être comprise entre 3 et 11 fois la fréquence de montée.

Nous avons donc refait la simulation en ajoutant cette fois-ci un bruit de 20% et en veillant à avoir $Te = t_m/5$ et nous avons obtenu les résultats suivants :

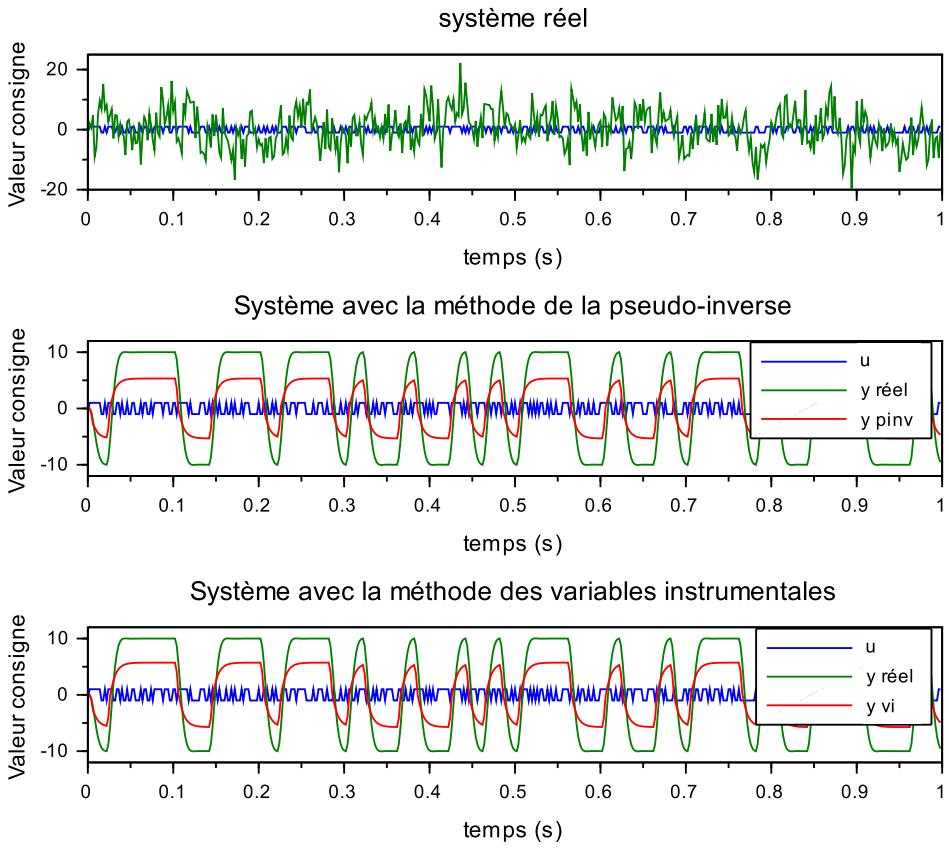


FIGURE II.3 – Simulation de l'identification d'un système d'ordre 3 soumis à un bruit blanc de 20 % échantillonné à $Te = t_m/5$

```
"paramètres discrets réels :"
-1.7326914
 1.0816131
-0.2417401
 0.6180884
 0.4537274
"identification pinv"
-0.2399613
-0.1584592
-0.1177041
 0.8317717
 1.7441483
"identification variables instrumentales"
-0.3501898
-0.1243331
-0.0968575
 0.7833037
 1.6686867
"erreur Pseudo-inverse :"
 19.740293
"erreur Variable instrumentale :"
 17.077338
```

FIGURE II.4 – Comparaison des coefficients du système d'ordre 3 avec les différentes méthodes

Les résultats présentés dans les figures II.3 et II.4 illustrent que, bien que les deux méthodes reproduisent globalement la dynamique du système simulé, une analyse plus détaillée de l'erreur (figure II.4) révèle que la méthode des variables instrumentales offre

de meilleures performances par rapport à celle de la pseudo-inverse. Pour confirmer ces observations, plusieurs essais ont été réalisés, montrant des résultats parfois différents, comme illustré dans les figures II.5 et II.6.

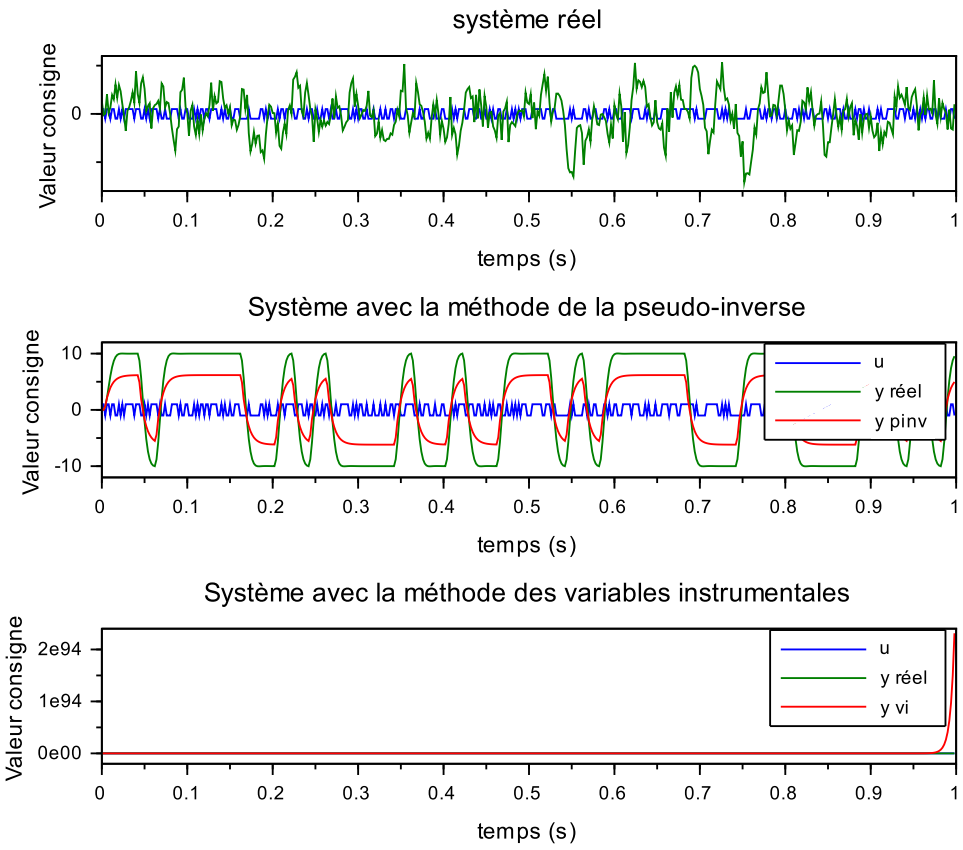


FIGURE II.5 – Simulation de l'identification d'un système d'ordre 3 soumis à un bruit blanc de 20% échantillonné à $Te = t_m/5$ avec les différentes méthodes

```
"paramètres discrets réels :"
-1.7326914
 1.0816131
-0.2417401
 0.6180884
 0.4537274
"identification pinv"
-0.5461464
-0.2555039
 0.0891060
 0.6376409
 1.1383583
"identification variables instrumentales"
-2.1457915
 0.7297927
 0.3082402
 0.7569387
 0.1146772
"erreur Pseudo-inverse :"
 14.222582
"erreur Variable instrumentale :"
 1.84D+186
```

FIGURE II.6 – Comparaison des coefficients du système d'ordre 3 avec les différentes méthodes

Cette fois-ci, après plusieurs essais, les identifications suivent globalement l'allure du système simulé. Cependant, on constate que la méthode des variables instrumentales peux diverger dans certains cas alors que la méthode de la pseudo-inverse ne diverge jamais. On a donc conclu que le choix qui serai le plus judicieux serai de prendre la méthode de la pseudo-inverse afin de garantir au mieux une fiabilité du système.

Maintenant que nous avons fait cela, nous pouvons passer à l'étape suivante qui est de convertir ces coefficients discrets en paramètres continus du système.

III Conversion des coefficients discrets en paramètres continus du système

Les coefficients discrets obtenus ne peuvent pas être utilisés tels quels pour calculer la correction car ils dépendent de la période d'échantillonnage, hors, les identifications ont été réalisées pour une période d'échantillonnage bien plus élevée que celle qui sera utilisé lors de l'asservissement. Il faut donc au préalable retrouver les coefficients continus correspondants aux coefficients discrets identifiés.

Afin de convertir nos coefficients discrets en coefficients continus, nous avons choisi d'utiliser un algorithme d'optimisation, l'algorithme d'Adam. Nous avons choisi cet algorithme plutôt qu'un autre, car nous avons vu en cours qu'il était présenté comme l'algorithme le plus performant actuellement, notamment en raison de sa capacité à converger rapidement vers un minimum local. Son objectif est donc de minimiser la fonction de coût, définie comme l'erreur quadratique moyenne entre sa réponse et celle du système discret identifié, afin de déterminer de manière optimale les paramètres K, τ, ξ et t_m. Le fichier *Conversion_identification_en_continu.sci* permet de faire cela.

Utilisation du fichier *Conversion_identification_en_continu.sci*

1 Paramétrage du signal SBPA

Listing 1 – Partie SBPA du code *Conversion_identification_en_continu.sci*

```
3 exec("toolbox_systemes.sci",-1);
4 Kb=500; N=2000; P=1;
5 u = SBPA(Kb, N, P);
```

Afin de créer notre signal SBPA désiré, on appel une fonction SBPA qui provient du fichier *toolbox_systemes.sci* et qui contient 3 paramètres à régler :

- Kb : permet de fixer l'amplitude de notre signal. Le signal varie entre $-Kb$ et $+Kb$.
- N : nombre de points que l'on veux.
- P : permet de multiplier la période de variation de la SBPA pour faciliter la stabilisation en faisant fonctionner la carte à une période plus faible que la periode

nécessaire à l'identification.

2 Définition des paramètres de notre fonction cible

Listing 2 – Définition des paramètres de notre fonction cible du code *Conversion_identification_en_continu.sci*

```
theta_cible=  [-2.0438249;
   1.1952930;
  -0.1432220;
  -0.0275236;
  -0.0172837];

a1_identif = theta_cible(1,1);
a2_identif = theta_cible(2,1);
a3_identif = theta_cible(3,1);
b2_identif = theta_cible(4,1);
b3_identif = theta_cible(5,1);


// simulation du systeme identif
z=poly(0,'z');
num_identif=b3_identif+b2_identif*z;
den_identif=z^3+a1_identif*z^2+a2_identif*z+a3_identif;
H_z=syslin('d',num_identif,den_identif);
v = tf2ss(H_z);
y=dsimul(v, u');
```

Il faut dans un premier temps, compléter θ_{cible} qui est un vecteur contenant les coefficients discrets du système cible. Ensuite, nous décomposons ces coefficients, nous créons le système identifié, et enfin, nous procédons à la conversion du système discret en un système continu.

3 Définition des paramètres de notre fonction cible

Listing 3 – Partie choix des paramètres continus du système cible dans le code *Conversion_identification_en_continu.sci*

```
// Paramtres continus initiaux du systme cible
K = -5;
tau = 0.002;
xi = 0.9;
tm = 0.002;

// Paramtre constant Te
Te = 120e-6;
```

Cette partie consiste à mettre les paramètres continus de notre système cible à atteindre en prenant en compte aussi le temps d'échantillonage Te.

4 Définition des paramètres de l'algorithme d'Adam

Listing 4 – Paramètres de l'algorithme d'Adam dans le code *Conversion_identification_en_continu.sci*

```
// Paramtres de l'algorithme Adam
alpha = 0.0001;  // Taux d'apprentissage
beta1 = 0.9;     // Paramtre de momentum pour le premier moment
beta2 = 0.999;   // Paramtre de momentum pour le second moment
e = 10^(-8);     // epsilon pour viter la division par zro
m0 = zeros(4,1); // Initialisation des moments
v0 = zeros(4,1);
t = 0;
cnt = 20000;     // Nombre d'itrations
X = zeros(4,cnt);
X(:,1) = [K; tau; xi; tm];  // Initialisation des paramtres
```

Cette partie définit les paramètres utilisés dans l'algorithme d'Adam :

— Le taux d'apprentissage α contrôle la vitesse de mise à jour des paramètres. Les coefficients β_1 et β_2 contrôlent les taux de décroissance des premier et deuxième instants, et elles doivent être proches de 1 pour donner plus de poids aux gradients récents. Le paramètre ϵ est ajouté afin d'éviter les divisions par zéro lors de la normalisation du gradient.

Nous avons pris les valeurs les plus courantes pour ces paramètres.

$m0$ est le premier moment du gradient et représente une estimation du gradient moyen au fil des itérations.
$v0$ est le second moment du gradient et représente une estimation de la variance du gradient.
X sera notre vecteur contenant les paramètres optimaux K, τ, ξ et t_m.

5 Définition de la fonction de coût

Listing 5 – Définition de la fonction de coût dans le code *Conversion_identification_en_continu.sci*

```
// Fonction de cout qui calcule l'erreur quadratique moyenne
function J = f(u, yidentif, X, Te)
    y = troisieme_ordre(X(1), X(2), X(3), X(4), u, Te);
    J = sum((y' - yidentif).^2) / length(yidentif);
endfunction
```

La fonction de coût J est définie comme l'erreur quadratique moyenne entre la sortie simulée y du modèle et la sortie du système cible y_{identif}. Elle permet de mesurer la différence entre le modèle identifié et le modèle réel, l'objectif étant de minimiser cette erreur.

6 Définition de la fonction de calcul du gradient

Listing 6 – Définition de la fonction de gradient dans le code *Conversion_identification_en_continu.sci*

```
// Fonction de calcul du gradient par diffrences finies
function g = gr(X, u, y, Te)
    g = zeros(4,1);
    delta = 1e-6;
    for i = 1:4
        X_plus = X;
        X_minus = X;
        X_plus(i) = X(i) + delta;
        X_minus(i) = X(i) - delta;
        g(i) = (f(u, y, X_plus, Te) - f(u, y, X_minus, Te)) / (2 * delta);
    end
endfunction
```

La fonction de gradient g est calculée par la méthode des différences finies. Cette méthode consiste à estimer la dérivée partielle de la fonction de coût par rapport à chaque paramètre du modèle en modifiant légèrement ces paramètres de $+\delta$ et $-\delta$. Le gradient est essentiel pour l'algorithme d'Adam, car il indique dans quelle direction ajuster les paramètres pour minimiser l'erreur.

7 Implémentation de l'algorithme d'Adam

Listing 7 – Implémentation de l'algorithme d'Adam dans le code *Conversion_identification_en_continu.sci*

```
// Implementation de l'algorithme Adam
xprev = X(:,1);
mtprev = m0;
vtprev = v0;
i = 1;

while (cnt > 0)
    t = t + 1;
    gt = gr(xprev, u, y, Te);  // Calcul du gradient
    mt = beta1 * mtprev + (1 - beta1) * gt;
    vt = beta2 * vtprev + (1 - beta2) * (gt .* gt);
    mthat = mt / (1 - beta1^t);
    vthat = vt / (1 - beta2^t);
    xnew = xprev - alpha * mthat ./ (sqrt(vthat) + e);
    i = i + 1;
    X(:,i) = xnew;
    xprev = xnew;
    mtprev = mt;
    vtprev = vt;
    cnt = cnt - 1;
end
```

Cette partie implémente l'algorithme d'Adam. À chaque itération, le gradient est calculé à l'aide de la fonction définie précédemment. Les moments du gradient sont ensuite mis à jour avec les coefficients β_1 et β_2. La mise à jour des paramètres est effectuée en normalisant le gradient à l'aide des moments corrigés et du taux d'apprentissage α.

8 Affichage des résultats et visualisation

Listing 8 – Affichage des résultats et visualisation dans le code *Conversion_identification_en_continu.sci*

```
// Affichage des resultats
disp(xprev, "Parametres optimaux X*: ");
[y_opti,coeffs_opti]=troisieme_ordre(xprev(1),xprev(2),xprev(3),xprev(4),u,Te)
plot(y,'b')
plot(y_opti,'r')
title("Comparaison de la sortie du systme identifi discret avec le systme optimis en continu")
legend("systeme reel", "systeme identif")
xlabel("Itrations")
ylabel("Valeur sortie du systme")
figure(2)
colormap(white)
plot(X(1,:))
xlabel("Itrations")
ylabel("Valeur de K")
title("Evolution de la variable K au fur et  mesure des itrations de l optimisation")
legend("K")
figure(3)
colormap(white)
plot(X(2,:))
xlabel("Itrations")
ylabel("Valeur de tau")
title("Evolution de la variable tau au fur et  mesure des itrations de l optimisation")
legend("$\tau$")
figure(4)
colormap(white)
plot(X(3,:))
xlabel("Itrations")
ylabel("Valeur de xi")
title("Evolution de la variable xi au fur et  mesure des itrations de l optimisation")
legend("$\xi$")
figure(5)
colormap(white)
plot(X(4,:))
xlabel("Itrations")
ylabel("Valeur de tm (s)")
```

```
128 title("Evolution de la variable tm au fur et  mesure des itrations de l
    optimisation")
129 legend("tm")
```

Cette dernière partie du code affiche les résultats des paramètres optimaux K, τ, ξ et t_m, et permet de comparer la sortie du système réel avec celle du modèle identifié. Les courbes générées permettent de visualiser la convergence de l'algorithme et d'évaluer l'efficacité de l'optimisation.

L'exécution de ce fichier donne les résultats suivant :

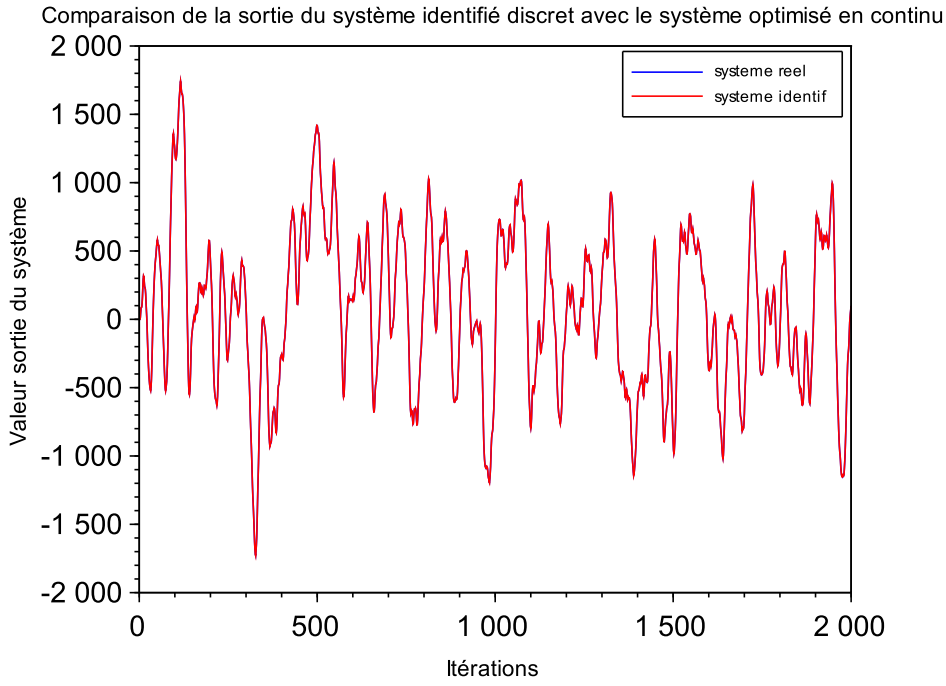
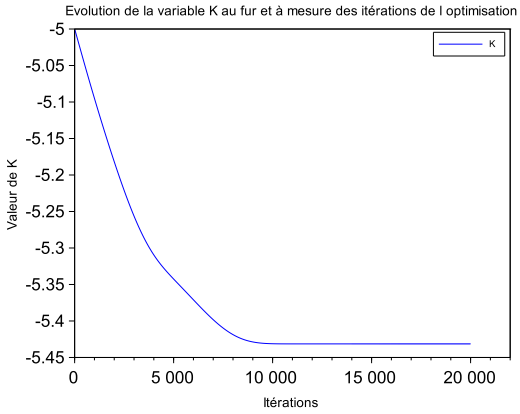
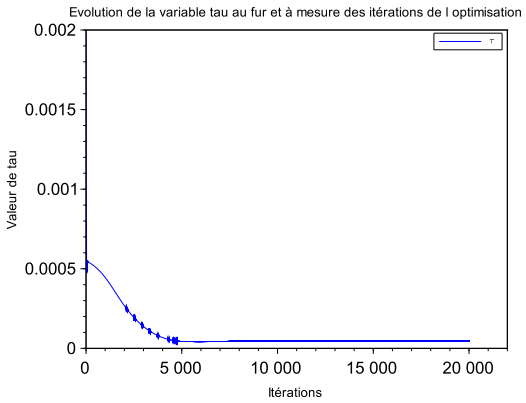


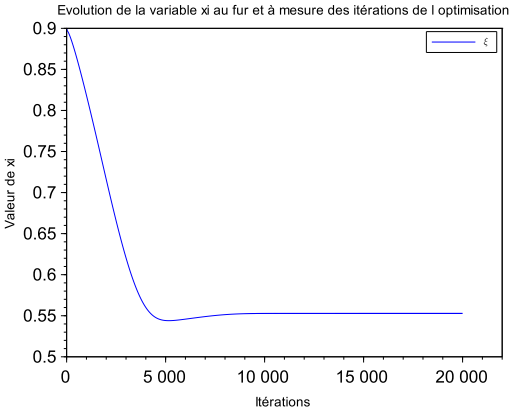
FIGURE III.1 – Comparaison de la sortie du sytème identifié discret avec le système optimisé en continu avec l'algorithme d'Adam



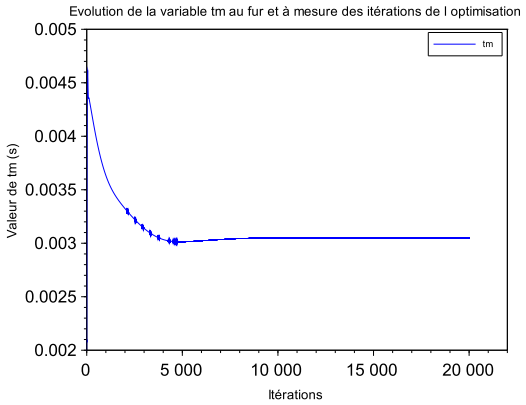
(a) Évolution de la variable K



(b) Évolution de la variable τ



(c) Évolution de la variable ξ



(d) Évolution de la variable t_m

FIGURE III.2 – Affichage des variables K, τ, ξ et t_m optimisées.