Chapitre 15

Régulation et contrôle de procédés

Dans ce chapitre, nous allons étudier de nombreuses méthodes de régulation et de contrôle de procédés.

I. Commande linéaire quadratique LQI

La commande LQI telle qu'elle est présentée ci-après, est basée sur la minimisation d'un critère quadratique. Elle est prédictive à un pas et possède une intégration afin de rejeter l'erreur en régime permanent.

Le critère J à minimiser permet d'assurer un compromis entre le carré de la variation de la commande de l'instant d'échantillonnage t et le carré de l'erreur de poursuite de l'instant future (t+1).

$$J=e(t+1)^2+R\Delta u(t)^2$$

La minimisation de ce critère consiste à calculer la variation de commande optimale qui satisfait à :

$$\frac{\partial J}{\partial(\Delta u)}=0$$

La présence de l'erreur future $e(t+1)$ et de l'incrément de commande $\Delta u(t)$ permet d'aboutir à une commande prédictive à un pas d'échantillonnage et d'inclure une intégration.

N. Martaj, M. Mokhtari, *MATLAB R2009, SIMULINK et STATEFLOW pour Ingénieurs, Chercheurs et Etudiants*, DOI 10.1007/978-3-642-11764-0_15, © Springer-Verlag Berlin Heidelberg 2010

Considérons un système du premier ordre de modèle discret :

$$\frac{B(z)}{A(z)}=\frac{z^{-1}(b_1+b_2 z^{-1})}{1-a_1 z^{-1}}$$

qui relie les entrées-sorties du processus par l'équation de récurrence suivante :

$$y(t)=a_1 y(t-1)+b_1 u(t-1)+b_2 u(t-2)$$

Pour faire apparaître l'incrément de la commande, on dérive les deux termes de cette expression pour obtenir le nouveau modèle prédicteur.

$$\hat{y}(t+1)=(1+a_1)y(t)-a_1 y(t-1)+b_1 \Delta u(t)+b_2 \Delta u(t-1)$$

En désignant par $r(t+1)$ la consigne future, l'erreur e(t+1) est estimée par :

$$e(t+1)=r(t+1)-\hat{y}(t+1)$$

La minimisation du critère permet d'aboutir à l'incrément de commande optimal suivant :

$$\Delta u(t)=\frac{b_1^2}{b_1^2+R}\left[r(t+1)-(1+a_1)y(t)+a_1 y(t-1)\right]$$

La commande à appliquer réellement au processus, à l'instant discret t, se calcule par la somme de la commande appliquée à l'instant (t-1) et de l'incrément Δu(t).

$$u(t)=u(t-1)+\Delta u(t)$$

A partir du modèle du processus et de la loi de commande, on obtient la fonction de transfert en boucle fermée suivante :

$$\frac{y(t)}{r(t)}=\frac{1-\lambda}{1-z^{-1}(1+a_1)\lambda+z^{-2}\,a_1\lambda}$$

avec $\lambda=\dfrac{R}{b_1^2+R}$

Le gain statique est égal à l'unité grâce à la présence de l'intégration, la dynamique est du second ordre de coefficient d'amortissement ξ et de pulsation non amortie ω_0 .

Ces 2 paramètres vérifient les relations suivantes :

$$(1+a_1)\lambda=2e^{-\xi\omega_0 T}\cos(\omega_0 T\sqrt{1-\xi^2})$$
$$a_1\lambda=e^{-2\xi\omega_0 T}$$

On s'impose une valeur du coefficient d'amortissement $\xi=\sqrt{2}/2$, et on déduit celle de la pulsation normalisée $\omega_0 T$ par le rapport des 2 relations précédentes.

$$\frac{1+a_1}{a_1} = 2\, e^{\xi\omega_0 T}\cos(\omega_0 T\sqrt{1-\xi^2})$$

C'est une relation non linéaire en $\omega_0 T$ que l'on peut résoudre facilement par MATLAB.
La deuxième relation permet le calcul du coefficient de réglage R.

$$R = \frac{b_1^2}{a_1 e^{2\xi\omega_0 T} - 1}$$

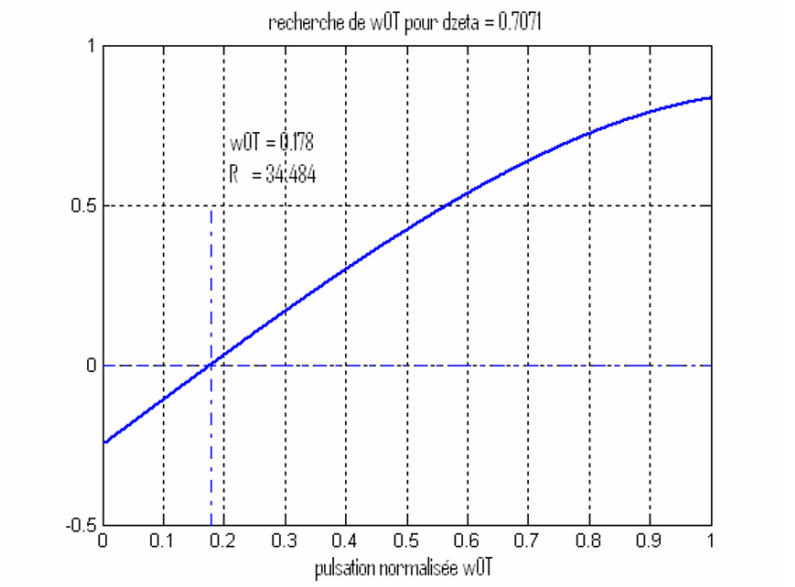
Le fichier `R_w0T.m` permet le calcul des paramètres $\omega_0 T$ et R si le modèle du processus est du premier ordre de pôle 0.8.

$$\frac{B(z)}{A(z)} = \frac{z^{-1}}{1-0.8z^{-1}}$$

fichier R_w0T

```
% Calcul des paramètres w0T et R de la commande LQI

z = sqrt(2)/2;  % coefficient d'amortissement

% paramètres du modèle du processus
a1 = 0.8;
b1 = 1;

x = 0:0.001:1;

y = 2*exp(z*x).*cos(x*sqrt(1-z*z))-(1+a1)/a1;
plot(x,y);
grid
hold on

%tracé de l'axe d'ordonnée nulle
plot(x,zeros(1,length(x)),'-.')
xlabel('pulsation normalisée w0T')
title('recherche de w0T pour dzeta = 0.7071')

% recherche de l'indice des tableaux x et y qui satisfait y=0
% avec la précision de 0,05%
j = find(abs(y)<0.0005);

% valeur de w0T recherchée
w0T = x(j);

% Tracé d'une droite verticale à w0T
```

```
droite(w0T,-0.5,w0T,0.5);

% calcul du coefficient de pondération R
R = (b1*b1)/(a1*exp(2*z*w0T)-1);

% affichage des valeurs des paramètres
text(0.21,0.7,['w0T = ' num2str(w0T)])
text(0.21,0.6,['R   = ' num2str(R)])

hold off
```



Dans le cas général, le modèle du processus est représenté par la fonction de transfert :

$$\frac{B(z)}{A(z)} = \frac{z^{-1}(b_1 + b_2 z^{-1} + \ldots + b_m z^{-m+1})}{1 - a_1 z^{-1} - a_2 z^{-2} - \ldots - a_n z^{-n}}$$

Par extension du cas particulier précédent, la loi de commande, dans ce cas général, s'écrit :

$$\Delta u(t) = \frac{b_1}{b_1^2 + R}\left[r(t+1) - (a_1 + 1)\, y(t) + [\sum_{i=1}^{n-1} (a_{i+1} - a_i)\, y(t-i)] - a_n\ y(t-n) - \sum_{i=2}^{m} b_i\ \Delta u(t-i+1) \right]$$

Pour programmer cette loi de commande, on utilisera les fonctionnalités polynomiales et matricielles de MATLAB. Les sommes sont programmées sous forme de produits scalaires et on utilisera le minimum de boucles `'for'`.

Le modèle du processus est défini par les 2 polynômes A et B comme suit :

$$A = [a_1\ a_2\ \ldots\ a_n]$$
$$B = [b_1\ b_2 \ldots\ b_m]$$

Le terme entre crochets peut être mis sous la forme du produit scalaire $\theta\varphi^T$ avec :

$$\theta = [a_1 + 1 \quad a_2 - a_1 \quad a_3 - a_2 \quad \ldots \quad a_n - a_{n-1} \quad -a_n]$$
$$\phi^T = [y(t) \quad y(t-1) \quad y(t-2) \quad \ldots \quad y(t-n+1) \quad y(t-n)]^T$$

Construction du vecteur de mesures θ :

Le vecteur θ est construit de la façon suivante :

$$\theta = [a_1\ a_2 - a_1\ a_3 - a_2 \ldots a_n - a_{n-1} - a_n] + [100\ldots00]$$

```
= diff([0 A 0]) + eye(1,n+1)
```

Construction du vecteur de mesures φ :

La sortie $y(t)$ de l'instant courant est donnée par le modèle (1) en fonction des sorties et des commandes précédentes.

$$y(t) = A * [y(t-1)\, y(t-2) \ldots y(t-n)]' + B * [u(t-1)\, u(t-2) \ldots u(t-m)]'$$

```
= A * y_t_1' + B * u_t_1'
```

Les composantes des vecteurs `y_t_1` et `u_t_1` sont les valeurs précédentes de la sortie et de la commande.

Les composantes du vecteur ϕ sont les valeurs précédentes de y(t).

La partie faisant intervenir les incréments de commande précédents est programmée comme suit :

$$\sum_{i=2}^{m} b_i\ \Delta u(t-i+1) = [b_2 \ldots b_m] * [\Delta u(t-1)\ \ldots\ \Delta u(t-m+1)]'$$

```
= B(1,2:m)*du_t_1'
```

avec `du_t_1`, le vecteur contenant les valeurs précédentes de l'incrément de commande.

Le fichier fonction `lqi.m` permet la programmation de cette commande.

```
[y,u] = lqi(a,b,r,N,R,u_min,u_max)
```

`a, b :`	polynômes A et B du modèle du processus,
`r, N :`	signal de consigne et nombre d'échantillons,
`R :`	coefficient de pondération de l'incrément de commande,
`u_min, u_max :`	valeurs limites de la commande.

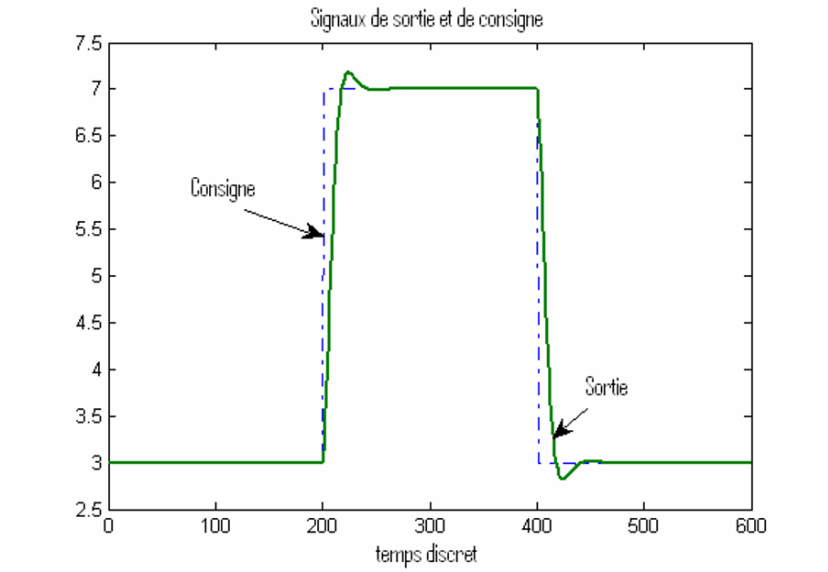
Les paramètres retournés sont la sortie du processus et le signal de commande.

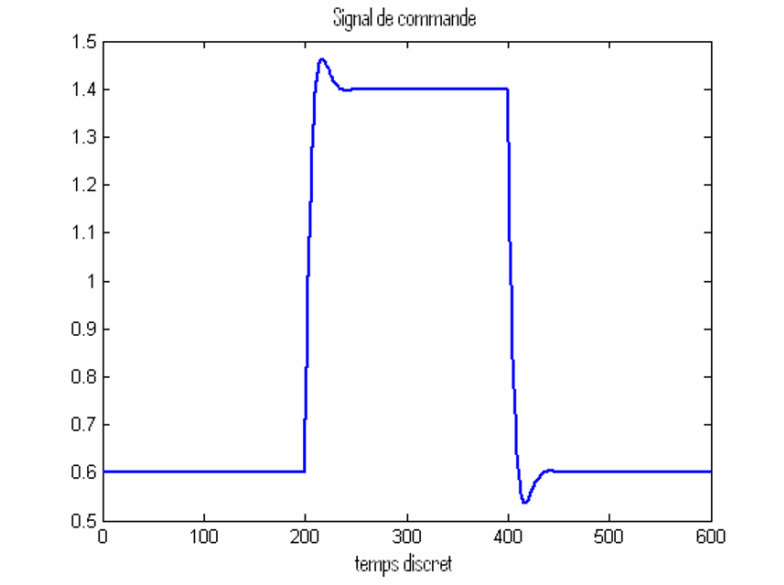
fichier lqi.m

```
function [y,u] = lqi(a,b,r,N,R,u_min,u_max)
% Commande Linéaire Quadratique avec Intégration (LQI)
% Commande prédictive à 1 pas
% Minimisation du critère quadratique :
%     J = R du(t)^2 + e(t+1)^2
% sans paramètres, les résultats correspondent à :
%   modèle du processus a1 = 0.8; b1 = 1;
%   nombre d'échantillons N = 600
%   coefficient de pondération R = 34.48
%   limitation de commande entre 0 et 10
%   consigne créneau entre 3 et 7

if nargin ~= 7
  home
  help lqi;
  disp('Appuyer sur une touche pour résultats d''un exemple')
  pause

  % consigne créneau
  N = 600;
  r = [3*ones(1,N/3) 7*ones(1,N/3) 3*ones(1,N/3)];

  % valeurs limites de la commande
  u_min = 0; u_max = 10;

  % modèle du processus
  aa =.8 ; bb = 1; R = 34.48;
  [y,u] = lqi(aa,bb,r,N,R,u_min,u_max);

  % affichage des résultats
  figure(1)
  plot(1:N,r,'-.',1:N,y);
  xlabel('temps discret')
  title('Signaux de sortie et consigne')
  figure(2)
  stairs(1:N,u);
  title('Signal de commande')
  xlabel('temps discret')
  return
end
```

```
% ordre du modèle du processus
na = length(aa);
nb = length(bb);

% initialisation des variables
% ----------------------------
du = zeros(1,N);
y = r;
u = r/(sum(bb)/(1-sum(aa)));

% algorithme de commande
% ----------------------
alpha = bb(1,1)/(bb(1,1)^2+R);

% polynôme AA
AA = diff([0 aa 0])+eye(1,na+1);

% polynôme BB
BB = bb(1,2:nb);
ok = isempty(BB);

if ok, BB = 0; end

for i = max(na,nb)+1:(N-1)
  %--------------------- boucle d'échantillonnage
  % valeurs précédentes de la sortie et de la commande
  y_t_1 = fliplr(y(i-na:i-1));
  u_t_1 = fliplr(u(i-nb:i-1));

 % valeurs précédentes des incréments de la commande
  du_t_1 = fliplr(du(i-nb+1:i-1));
  if ok, du_t_1 = 0; end

 % sortie courante du modèle du processus
 y(i) = y_t_1*aa'+u_t_1*bb';

 % calcul de l'incrément et de la commande à appliquer
 du(i) = alpha*(r(i+1)-fliplr(y(i-na:i))*AA'-du_t_1*BB');
 u(i) = du(i)+u(i-1);

 % saturation de la commande
 cde = u(i);

 cde = (cde >= u_max)*u_max+(cde <= u_min)*u_min+...
 ((cde < u_max)&(cde > u_min))*cde;

u(i) = cde;
end
```

L'appel de la fonction `lqi` sans paramètres, correspond au cas précédemment étudié.


Signaux de sortie et de consigne
Consigne
Sortie
temps discret


Signal de commande
temps discret

Si le modèle d'un processus n'est pas du type précédemment étudié, le calcul du coefficient R peut être déterminé a posteriori par simulation.

Un utilisateur pourrait choisir la valeur de R qui donne, soit le meilleur temps de réponse, soit des incréments de commande plus faibles, etc.

On va considérer le cas d'un modèle de processus du second ordre suivant :

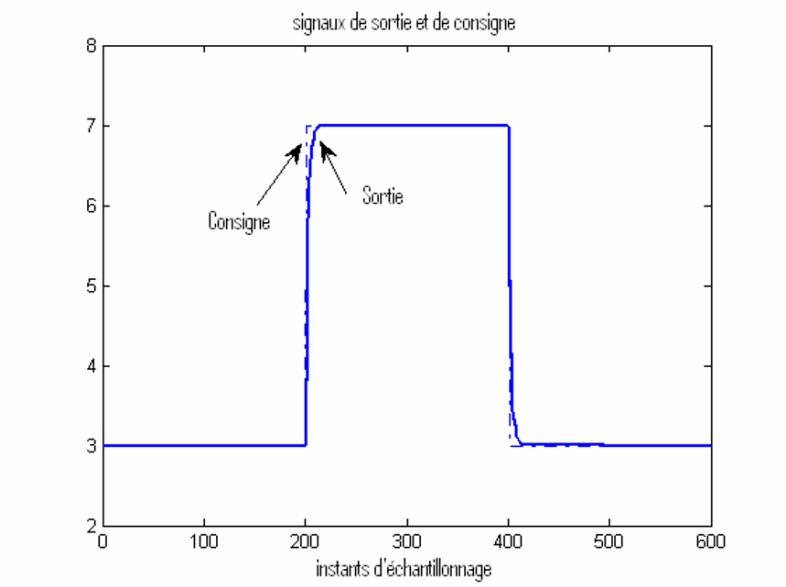
$$\frac{B(z)}{A(z)} = \frac{0.5\,z^{-1}}{1-0.76\,z^{-1}+0.3\,z^{-2}}$$

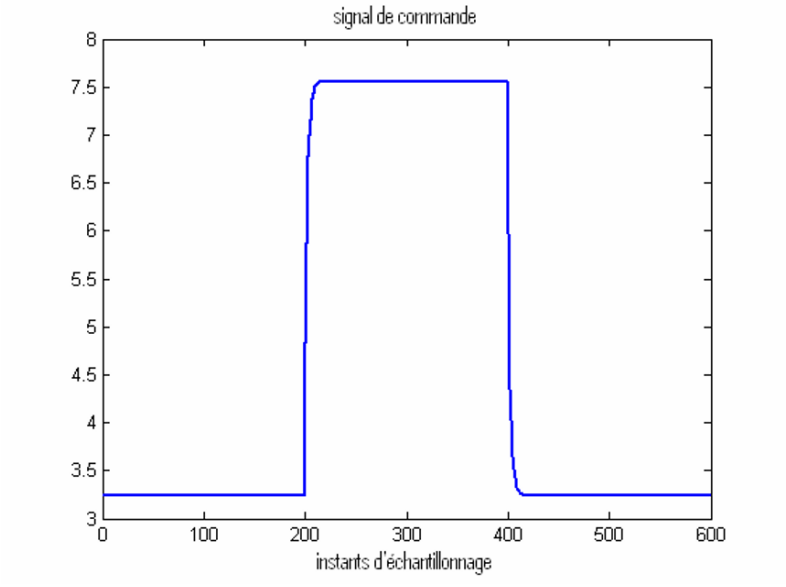
et on visualisera l'effet du coefficient de pondération R sur les performances de la loi de commande.

fichier cde_lqi.m

```
% Nombre d'échantillons
N = 600;

% signal de consigne
% génération de l'échelon
r = [3*ones(1,N/3) 7*ones(1,N/3) 3*ones(1,N/3)];

R = 1;
u_min = 0;
u_max = 10;
aa = [0.76 -0.3];
bb = 0.5;

[y,u] = lqi(aa,bb,r,N,R,u_min,u_max);

% Appel de la fonction lqi sans paramètres
lqi

figure(1)
plot(y)
hold on

% tracé du signal de consigne
plot(r,'-.')
hold off
title('signaux de sortie et de consigne')
xlabel('instants d''échantillonnage')
axis([0 N 2 8])

% affichage de la commande sous forme bloquée
figure(2)
stairs(u)
title('signal de commande')
xlabel('instants d''échantillonnage')
```

Avec R = 1, on obtient :





Le temps de réponse est très faible, en contre partie les variations de la commande sont très grandes lors des changements de consigne.

Pour obtenir un temps de réponse raisonnable tout en adoucissant la commande, on est amené à augmenter le coefficient de pondération R.

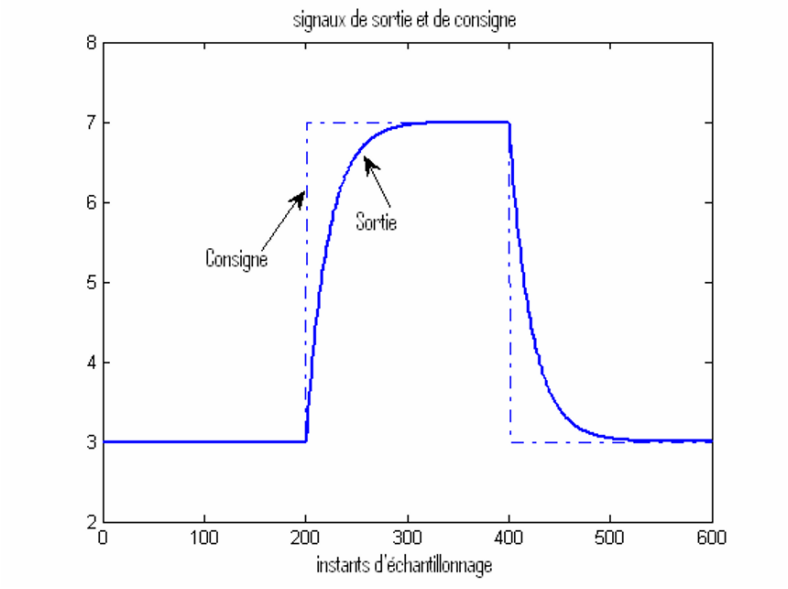
Pour R = 10, la variation de la commande est beaucoup moins pondérée par rapport à l'erreur de poursuite, la sortie du processus suit le signal de référence avec une dynamique beaucoup plus faible que précédemment.

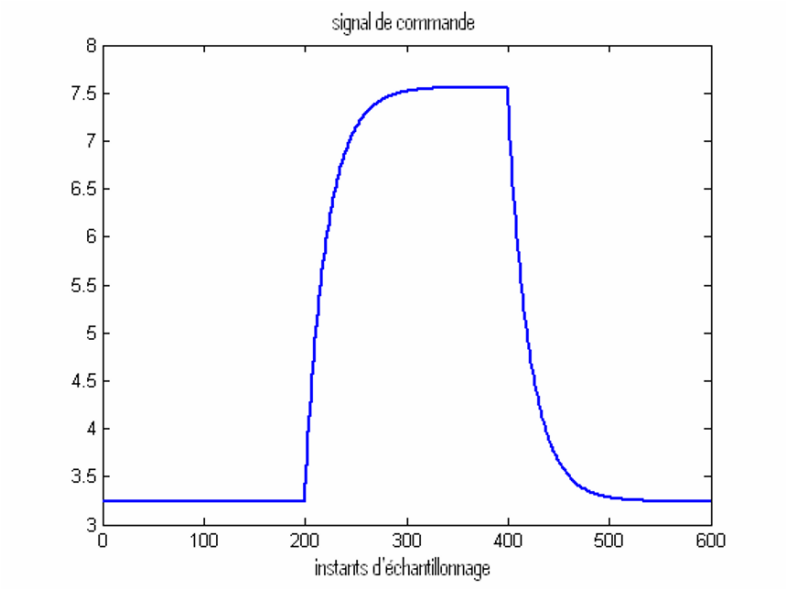
Dans un cas réel, on choisira une valeur du coefficient R qui satisfait à des critères de temps de réponse, de limitation de la commande ou de ses incréments.

Nous obtenons les résultats suivants qui peuvent être satisfaisants dans un cas réel.
Les valeurs statiques de la commande sont les mêmes quelque soit la valeur du coefficient de pondération R.

Il en est de même de l'erreur statique, nulle grâce à la présence de l'intégration dans la loi de commande.

La courbe suivante représente les signaux de consigne et de sortie pour une valeur du coefficient R = 10.





II. Commande RST

L'intérêt de ce type de commande -contrairement à beaucoup d'autres correcteurs numériques tel le classique PID- est de spécifier la loi de réjection des perturbations indépendamment de celle de la poursuite du signal de consigne.

Pour cette raison, ce régulateur est dit "poursuite et régulation à objectifs indépendants".

Le calcul de la loi de commande est basé sur un critère polynomial qui permet de spécifier la dynamique de réjection de l'erreur entre le signal de référence et le signal de sortie (perturbation).

$$P(z^{-1})\left[r(t+1)-y(t+1)\right]=0$$

Cette dynamique s'obtient par le choix du polynôme $P(z^{-1})$.

Si l'on choisit un polynôme de régulation du premier ordre

$$P(z^{-1})=1-0.5\,z^{-1}=1-p_1\,z^{-1}$$

la perturbation sera divisée par 2 à chaque instant d'échantillonnage.

Comme exemple d'application de cette commande, considérons un modèle de processus du premier ordre avec retard pur :

$$H(z^{-1}) = \frac{B(z^{-1})}{A(z^{-1})} = \frac{z^{-1}(b_1 + b_2 z^{-1})}{1 - a_1 z^{-1}}$$

Le modèle prédicteur est dans ce cas

$$y(t+1) = (1+a_1)\, y(t) - a_1\, y(t-1) + b_1\, \Delta u(t) + b_2\, \Delta u(t-1)$$

Avec le polynôme de régulation du premier ordre, la commande u(t) à appliquer au processus à l'instant d'échantillonnage t est :

$$u(t) = \frac{1}{b_1}\Big[r(t+1) - p_1\, r(t) - (1 + a_1 - p_1)\, y(t) + a_1\, y(t-1) - (b_2 + b_1)\, u(t-1) + b_2\, u(t-2) \Big]$$

Lorsqu'il s'agit de consignes telles l'échelon ou le signal carré, il est préférable de les filtrer pour donner le signal de référence $r(t)$.

On choisit généralement un filtre du premier ou du second ordre de gain statique unité.

Le filtre de référence du second ordre est de la forme :

$$H_r(z^{-1}) = \frac{1 - \alpha_1 - \alpha_2}{1 - \alpha_1 z^{-1} - \alpha_2 z^{-2}}$$

Si l'on choisit un filtre du premier ordre de pôle α_1, il suffit d'afficher $\alpha_2 = 0$.

Pour des consignes ne présentant pas de fronts raides (sinus, etc.), il n'est pas utile de les filtrer; dans ce cas, il faut afficher $\alpha_1 = \alpha_2 = 0$.

Les coefficients α_1 et α_2 déterminent la pulsation propre non amortie w_0 et le coefficient d'amortissement ξ par les relations suivantes :

$$\alpha_1 = 2 e^{-\xi \omega_0 T} \cos(\omega_0 T \sqrt{1 - \xi^2})$$
$$\alpha_2 = -e^{-2\xi \omega_0 T}$$

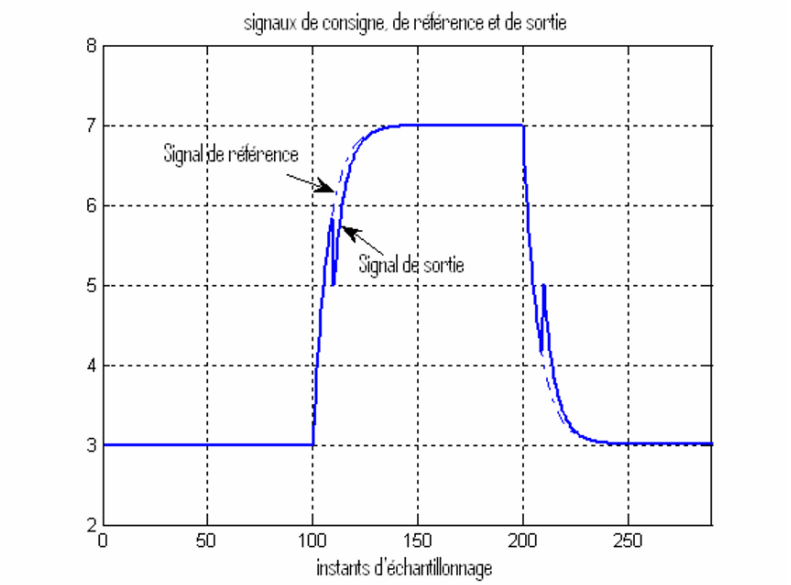
T est la période d'échantillonnage.

fichier Regul_RST.m

```
N = 300;

% génération du créneau de consigne
%c = [3*ones(1,N/2) 7*ones(1,N/3) 3*ones(1,N/3)];
c = [3*ones(1,N/3) 7*ones(1,N/3) 3*ones(1,N/3)];
 % filtrage par un filtre du 2nd ordre
% pulsation non amortie w0T = 0.2
% coefficient d'amortissement dzeta = 0.7071
```

```
%z=sqrt(2)/2;
z=3;
w0T = 0.8;

alpha1 = 2*exp(-z*w0T)*cos(w0T*sqrt(1-z*z));
alpha2 = -exp(-2*z*w0T);
r(1) = c(1);
r(2) = c(2);

% signal de référence
for i = 3:N
  r(i) = alpha1*r(i-1)+alpha2*r(i-2)+(1-alpha1-alpha2)*c(i);
end

% modèle du processus
num = [1 -.5];
den = [1 -0.8];

%polynôme de régulation du 1er ordre de pole 0.5
Preg = [1 -.8]; % p1 = 0.5

% valeurs limites de la commande
u_min = 0; u_max = 10;
```

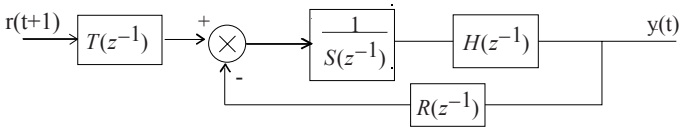
```
% initialisation des variables
u(2) = r(2)*sum(den)/sum(num); u(1) = u(2);
y(1) = r(1);
y(2) = r(2);
du1 = 0;
for i = 3:N-1
% calcul de la sortie du modèle du processus
y(i) = -den(2)*y(i-1)+num*[u(i-1) u(i-2)]';
du2 = ([r(i+1) r(i)]*Preg'-[1-den(2)+Preg(2) den(2)]*[y(i)
y(i-1)]'-num(2)*du1)/num(1);
u(i) = u(i-1)+du2;
du1 = du2;
% création de perturbations
y(i) = y(i)-(i == 110)+(i == 210);
%limitation de la commande entre 0 et 10
cde = u(i);
cde = (cde >= u_max)*u_max+(cde <= u_min)*u_min+((cde <
u_max)& (cde>u_min))*cde;
u(i) = cde;
end
plot(y), hold on, plot(r,'-.');
title('signaux de consigne, de référence et de sortie')
xlabel('instants d''échantillonnage')
axis([0 N-10 2 8])
hold off, grid
```



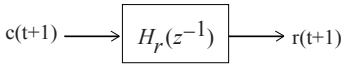
Aux instants d'échantillonnage 50, 110 et 250, nous avons crée des perturbations de valeurs respectives -0.5, -2 et 1. Elles sont rejetées avec une dynamique du premier ordre de pôle 0.5 (atténuation de 2 à chaque instant d'échantillonnage) alors que la sortie suit parfaitement le signal de référence obtenu par le filtrage de la consigne avec une dynamique du second ordre.

Les dynamiques de poursuite et de régulation sont bien indépendantes

La loi de commande dite à 3 branches R, S et T est décrite par le schéma suivant :



Le filtrage de la consigne c(t) donne le signal de référence r(t) qui définit la dynamique de poursuite.



La commande s'exprime en fonction des 3 polynômes R, S et T.

$$u(t)=\frac{1}{S(z^{-1})}\left[T(z^{-1})r(t+1)-R(z^{-1})y(t)\right]$$

Dans le cas du processus étudié, nous avons :

$$R(z^{-1}) = (1 + a_1 - p_1) - a_1 z^{-1}$$
$$S(z^{-1}) = b_1 + (b_2 - b_1) z^{-1} - b_2 z^{-2}$$
$$T(z^{-1}) = 1 - p_1 z^{-1}$$

Dans le cas général d'un processus d'ordre n, ayant m zéros et d'un choix de polynôme de régulation d'ordre l, les polynômes du régulateur ont pour expressions :

$$R(z^{-1}) = 1 + a_1 + \sum_{i=2}^{n} (a_i - a_{i-1}) z^{-i} - a_n z^{-n} - \sum_{i=2}^{l} p_i z^{-1}$$
$$S(z^{-1}) = b_1 + \sum_{i=2}^{m} (b_i - b_{i-1}) z^{-i} - b_m z^{-m}$$
$$T(z^{-1}) = P(z^{-1})$$

Programmation du régulateur

Le modèle du processus décrit par deux polynômes `A` et `B`, pour obtenir le polynôme `R`, il faut ajouter des zéros au plus petit des polynômes `A` et `P`.

```
if length(A)<length(P)
  A(length(A)+1:length(P)) = zeros(1,length(P)-length(A))
else
  P(length(P)+1:length(A)) = zeros(1,length(A)-length(P))
end
```

Ainsi, le polynôme `R` sous forme plus compacte,

$$R(z^{-1}) = (1 + a_1 - p_1) + \sum_{i=2}^{\max(n,l)} \left[(a_i - a_{i-1}) - p_i \right] z^{-i} - a_n z^{-n}$$

peut être décrit facilement à l'aide d'un seul tableau.

```
R = [1+A(1)-P(1)   diff(A)-P(2:length(P))-A(length(A))]
```

Les polynômes S et T sont donnés par :

```
S = [B(1)    diff(B)    -B(length(B)]
T = P
```

La commande, `u(t)`, à appliquer au processus, à l'instant discret `iT`, se calcule comme suit :

```
% ordres de A et P après augmentation du plus petit
q = length(A);
s = length(S);
u(i) = ([1 -P]*fliplr[r(i-q:r(i+1))]'-R*...
```

```
    fliplr[y(i-q:y(i))]'-S(2:s)*fliplr(u(i-s:i-1) )/B(1);
```

Le fichier fonction `rst.m` permet d'implémenter cette commande dans le cas le plus général de type de modèle de processus et de polynôme de régulation.

```
[y,u] = rst(aa,bb,r,N,Preg,den_Hr,u_min,u_max);
```

`aa, bb :`	polynômes A et B du modèle du processus,
`r, N :`	signal de référence et nombre d'échantillons,
`Preg :`	paramètres du polynôme de régulation,
`den_H :`	dénominateur du modèle de référence,
`u_min,u_max :`	valeurs limites de la commande,
`y, u :`	vecteurs des signaux de sortie et de commande.

fichier rst.m

```
function [y,u] = rst(aa,bb,r,N,Preg,den_Hr,u_min,u_max);

% Commande intégrale et prédictive à 1 pas
% sans paramètres, les résultats correspondent à :
% modèle du processus aa = [1 -0.74]; bb = [1 -0.5];
% nombre d'échantillons N = 600
% polynôme de régulation du 1er ordre Preg = [1 -0.5]
% limitation de commande entre u_min = 0 et u_max = 10
% consigne créneau 5 +/- 2
% dénominateur du modèle de référence den_Hr = [1 -.8];
if nargin ~= 8
help rst;
disp('Appuyer sur une touche pour voir résultats d''un
exemple')
  pause

  % consigne échelon
  N = 600;
  c = [3*ones(1,N/3) 7*ones(1,N/3) 3*ones(1,N/3)];

  % valeurs limites de la commande
  u_min = 0; u_max = 10;

  % polynôme de régulation
  Preg = 0.5;

  % modèle de référence
  den_Hr = [1 -0.9 0];

  % modèle du processus
  aa = 0.5; bb = [1 -0.5];

  % génération du signal de référence
  num_Hr = sum(den_Hr);
  r(1) = c(1);
```

```
  r(2) = c(2);
  for i = 3:N
    r(i) = -den_Hr(2:3)*r(i-2:i-1)'+num_Hr*c(i);
  end
 [y,u] = rst(aa,bb,r,N,Preg,den_Hr,u_min,u_max)
 return
end

% ordre du modèle du processus et des polynômes Hr et Preg
if length(aa)<length(Preg)
  aa(length(aa)+1:length(Preg)) = zeros(1,length(Preg)-...
  length(aa));
else
  Preg(length(Preg)+1:length(aa)) = zeros(1,length(aa)-...
  length(Preg));
end

nb = length(bb);

% ordres de aa et Preg après augmentation du + petit
na = length(aa);
% construction des polynômes R, S et T
R   =   [1+aa(1)-Preg(1)   diff(aa)-Preg(2:length(Preg))   -...
aa(length(aa))];
S = [bb(1) diff(bb) -bb(length(bb))];

% initialisation des signaux de commande et de sortie
y = r;
u = r/(sum(bb)/(1-sum(aa)));

for i = max(na,nb)+1:(N-1);
  y_t_1 = fliplr(y(i-na:i-1));
  u_t_1 = fliplr(u(i-nb:i-1));
  y(i) = y_t_1*aa'+u_t_1*bb';

u(i) = ([1 -Preg]*(fliplr(r(i-length(Preg)+1:i+1)))'-...
       R*(fliplr(y(i-length(R)+1:i)))'-S(2:length(S))*...
       (fliplr(u(i-length(S)+1:i-))')/S(1);

% saturation de la commande
cde = u(i);
cde = (cde >=u_max)*u_max+(cde<=u_min)*u_min+((cde<u_max)&...
((cde>u_min))*cde;
u(i) = cde;

end
```

Considérons le processus du second ordre suivant :

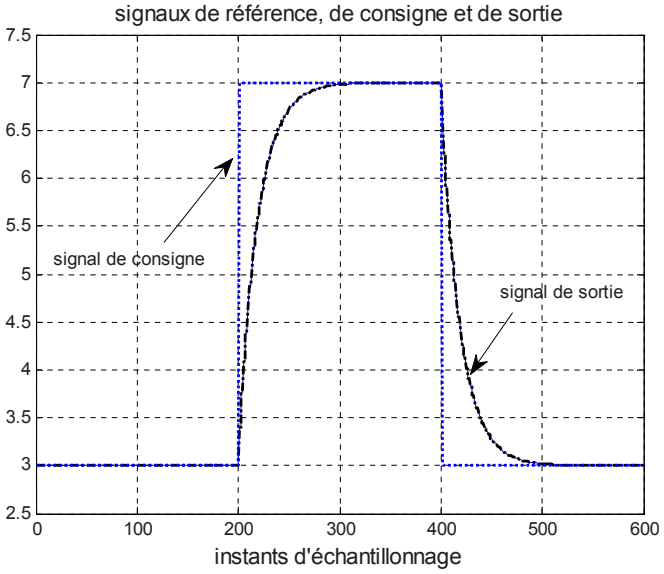
$$\frac{B(z)}{A(z)} = \frac{1}{1 - 0.76\ z^{-1} + 0.3\ z^{-2}}$$

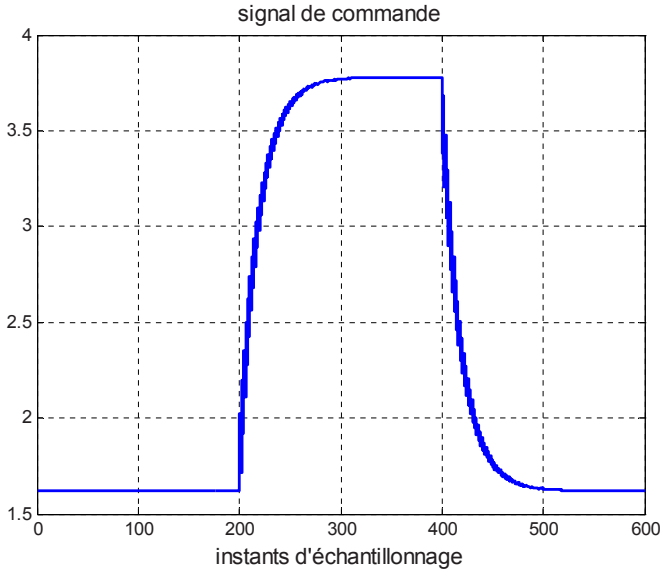
sur lequel on appliquera la commande RST. On spécifie un modèle de référence du premier ordre de pôle 0.9 ($\alpha_1 = 0.9, \alpha_2 = 0$) et une dynamique de régulation du premier ordre de pôle 0,5.

Pour appliquer cette commande, on spécifie tous les paramètres nécessaires dans le fichier script `cde_rst.m`.

fichier cde_rst

```
N = 600;
% consigne échelon
c = [3*ones(1,N/3) 7*ones(1,N/3) 3*ones(1,N/3)];
% valeurs limites de la commande
u_min = 0; u_max = 10;

% polynôme de régulation
Preg = 0.5;

% modèle de référence
den_Hr = [1 -0.9 0];

% modèle du processus
aa = [0.76 -0.3]; bb = 0.5;

% génération du signal de référence
num_Hr = sum(den_Hr); ²
r(1) = c(1); r(2) = c(2);
for i = 3:N
  r(i) = -den_Hr(2:3)*r(i-2:i-1)'+num_Hr*c(i);
end

[y,u] = rst(aa,bb,r,N,Preg,den_Hr,u_min,u_max);
figure(1)
plot(y)
hold on
plot(r,'g-.')
plot(c,':');
title('signaux de référence, de consigne et de sortie')
xlabel('instants d''échantillonnage'), grid
figure(2)
plot(u), grid
title('signal de commande')
xlabel('instants d''échantillonnage');
```


signaux de référence, de consigne et de sortie
signal de consigne
signal de sortie
instants d'échantillonnage


signal de commande
instants d'échantillonnage

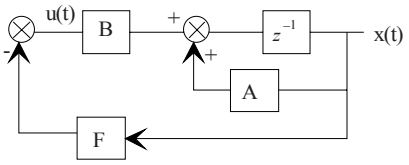
III. Commande asymptotique et commande optimale dans l'espace d'état

III.1. Commande asymptotique par placement de pôles

On suppose un système linéaire ou linéarisé décrit par une équation d'état discrète résultant de la discrétisation du processus continu.
Un régulateur a pour objet de maintenir la sortie à la valeur 0 en présence de perturbations.

Le signal de commande s'exprime par un retour de l'état, sous la forme : $u(t) = -F\,x(t)$
Si le processus possède `p` entrées et si mon modèle d'état est d'ordre n, la matrice `F` de coefficients de retour d'état est de dimensions (`p, n`).

La loi de commande par retour d'état est schématisée comme suit :



Le comportement du système bouclé par la matrice F est donné par l'équation matricielle suivante :

$$x(t+1) = (A - BF)\,x(t)$$

Cette équation permet de suivre l'évolution de l'état du système à partir de conditions initiales non nulles ou suite à l'application d'une perturbation.

Si le processus est commandable, on peut placer les pôles du système en boucle fermée (par retour d'état) n'importe où dans le plan z.

Ces pôles sont les solutions de l'équation caractéristique suivante :

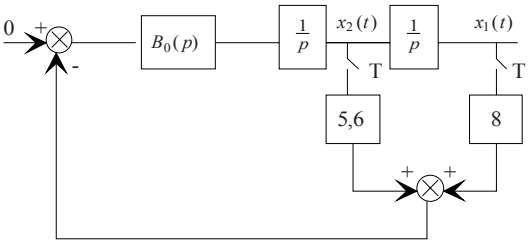
$$\det(zI - A + BF) = 0$$

Si l'on désire un retour à l'équilibre suivant une dynamique de second ordre de pôles réels $z_1 = 0.6$ et $z_2 = 0.8$, les deux coefficients de retour d'état sont solutions du système :

$$\begin{aligned} 2 - 0.005\,f_1 - 0.1\,f_2 &= 1.4 \\ 1 + 0.005\,f_1 - 0.1\,f_2 &= 0.48 \end{aligned}$$

```
>> F = inv([-0.005 -0.1;0.005 -0.1])*[-0.6; -0.52]
F =
    8.0000
    5.6000
```

La loi de commande appliquée à un double intégrateur (volant d'inertie) est schématisée comme suit :



Les matrices du modèle d'état discret du processus analogique précédé du bloqueur d'ordre 0 sont obtenues par la commande `c2dt` connaissant celles du système analogique et la période d'échantillonnage.

Matrices du modèle d'état analogique

```
>> A = [0 1; 0 0]; B = [0;1]; C = [1 0]; D = 0;
```

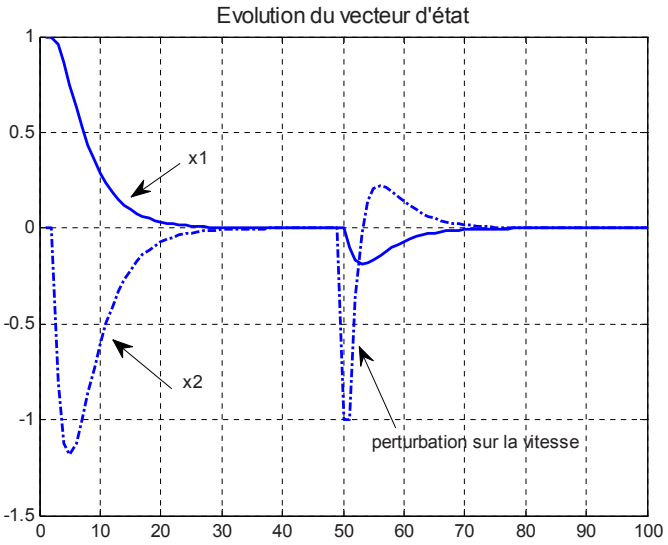
Matrices du modèle d'état du système discret

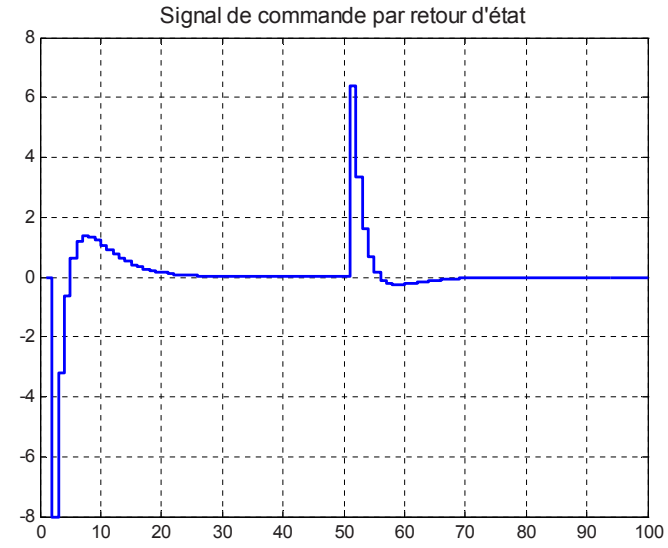
```
>>[A,B,C,D] = c2dt(A,B,C,0.1,0) % échantillonnage à 0.1 s
A =
    1.0000    0.1000
         0    1.0000
B =
    0.0050
    0.1000
C =
     1     0
D =
     0
```

fichier retetat.m

```
% matrices d'état du sysème discret
A = [1.0000 0.1000;0 1.0000]; B = [0.0050;0.1000]; C = [0 1];
N = 100; % horizon de commande
F = [8 5.6]; % matrice des coefficients du retour d'état
% algorithme de commande
u = []; u(1) = 0; x(:,1) = [1 0]'; % conditions
for i = 2:N
  x(:,i) = A*x(:,i-1)+B*u(i-1); u(i) = -F*x(:,i);
% perturbation sur la vitesse
  if i==N/2, x(2,i) = x(2,i)-1;   end
end
figure(1), plot(1:N,x(1,:)), hold on
plot(1:N,x(2,:),'-.'), hold off
title('Evolution du vecteur d''état');
```

```
gtext('x1'), gtext('x2')
gtext('perturbation sur la vitesse')
figure(2), stairs(1:N,u)
title('Signal de commande par retour d''état')
```

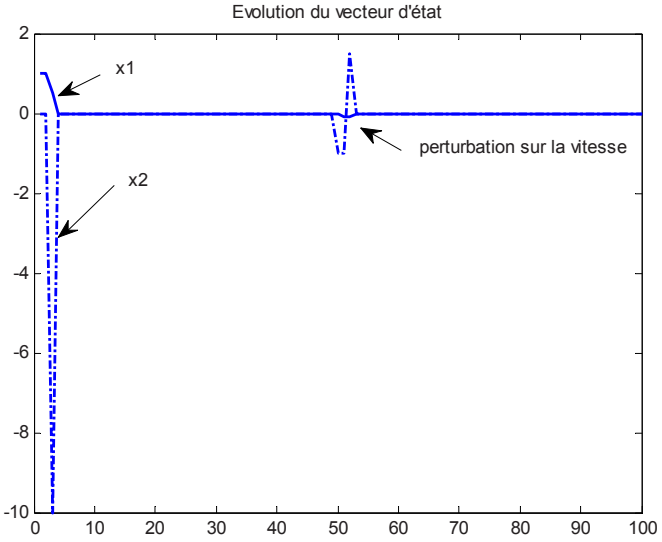


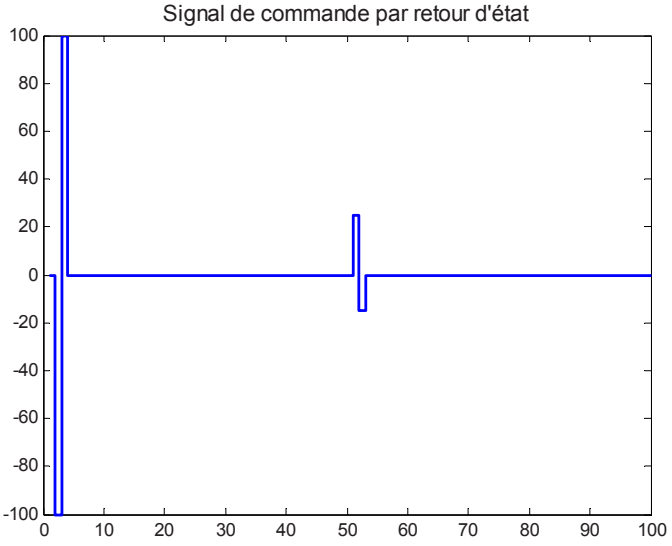


Si

l'on veut revenir à zéro le plus rapidement possible, il suffit de mettre tous les pôles à l'origine, ce qui correspond aux valeurs suivantes des coefficients du retour d'état.

$$f_1 = 100 \quad , \quad f_2 = 15$$





Quelques soient les conditions initiales, en deux périodes d'échantillonnage le système revient à sa position d'équilibre : c'est la réponse pile pour laquelle les actions sont très violentes.

III.2. Commande optimale dans l'espace d'état

Régulation autour d'une consigne nulle

La commande optimale dans l'espace d'état est basée sur la minimisation d'un critère quadratique, le plus général étant

$$J=\frac{1}{2}x(N)^T\,H\,x(N)+\frac{1}{2}\sum_{t=0}^{N-1}\left[x(t)^T\,Q\,x(t)+u(t)^T\,R\,u(t)\right]$$

N désigne l'instant auquel on atteint l'objectif qui est l'annulation de l'état dans le cas d'une régulation autour d'une consigne nulle.

Q et H sont des matrices de pondération, symétriques. La matrice H ou terme de pondération terminal permet de pénaliser plus ou moins l'écart final par rapport à la cible cherchée.

Q est une matrice de pondération de l'état intermédiaire. R est une matrice de pondération des signaux de commande.

Dans le cas d'un système à p entrées dont le modèle d'état est d'ordre n, les matrices H, Q et R sont carrées, symétriques et d'ordres n et p respectivement.
Le problème consiste à minimiser le critère J sous la contrainte suivante

$$x(t+1)=A\,x(t)+B\,u(t)$$

La minimisation du critère quadratique entre les étapes (N-1) et N sous la contrainte précédente permet d'obtenir.

$$u(N-1)=-\left[R+B^T\,P(N)\,B\right]^{-1}B^T\,P(N)\;A\;x(N-1)$$

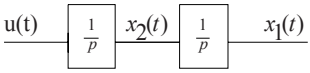
$$=-F(N-1)\;x(N-1)$$

$$P(N-1)=A^T\left[P(N)-P(N)\,B\left[R+B^T\,P(N)\,B\right]^{-1}B^T\,P(N)\right]A+Q$$

(Équation de RICCATI)

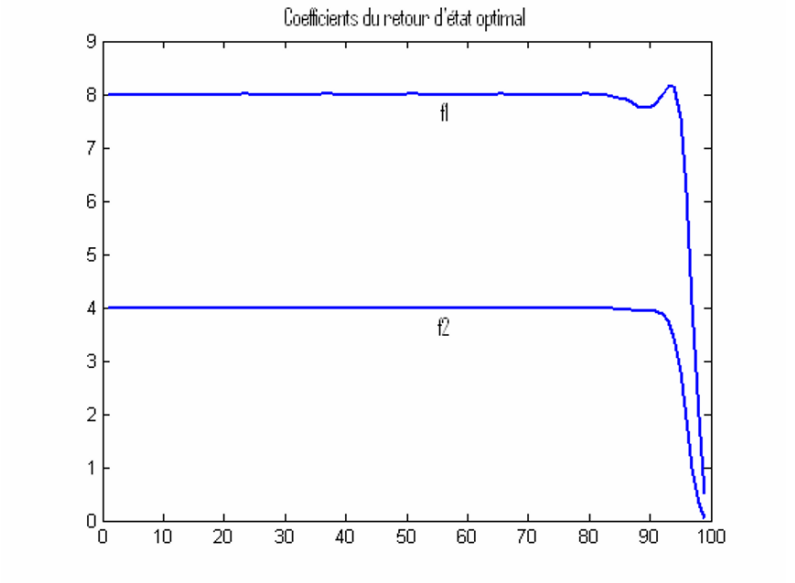
La détermination du retour d'état F(N-1) est basée sur le principe et optimalité dont l'énoncé est dû à BELLMAN (1962).

Application au double intégrateur échantillonné à la cadence T = 0.1 s.



fichier cde_opt1.m

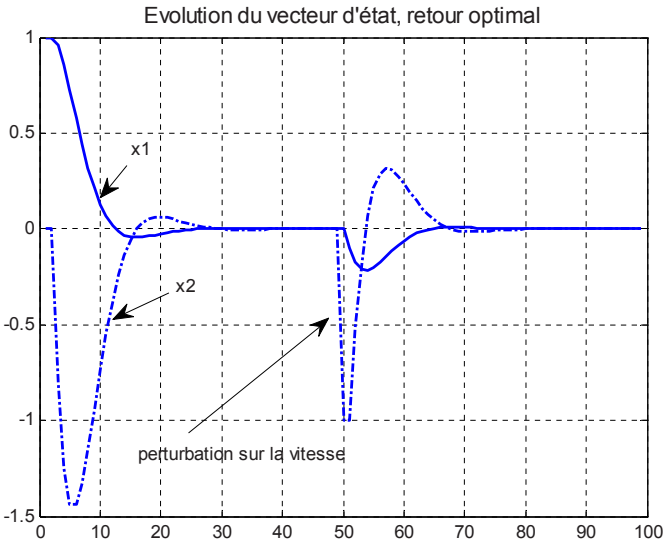
```
% commande optimale par retour d'état
% régulation autour d'une consigne non nulle
% matrices du modèle d'état analogique
A=[0 1; 0 0]; B=[0;1]; C=[0 1]; D=0;
% matrices du modèle d'état discret
[A,B,C,D]=c2dt(A,B,C,0.1,0);
N=100; % horizon de commande
% matrices de pondération
H=[1 0;0 0]; Q=H; R=0.01;
% calcul des coefficients du retour d'état optimal
P=H;
for i=N:-1:2
F(i-1,:)=inv(B'*P*B+R)*B'*P*A;
P=A'*(P-P*B*inv(B'*P*B+R)*B'*P)*A+Q;
end
% tracé des coefficients du retour d'état
figure(1), plot(1:N-1,F(:,1)), hold on
plot(1:N-1,F(:,2)),hold off
title('Coefficients du retour d''état optimal')
gtext('f1'), gtext('f2')
```



Le retour d'état est constant sauf au voisinage du point d'arrivée. Le régime permanent correspond à la solution permanente de l'équation de RICCATI. Le plus souvent on implante le régime permanent.

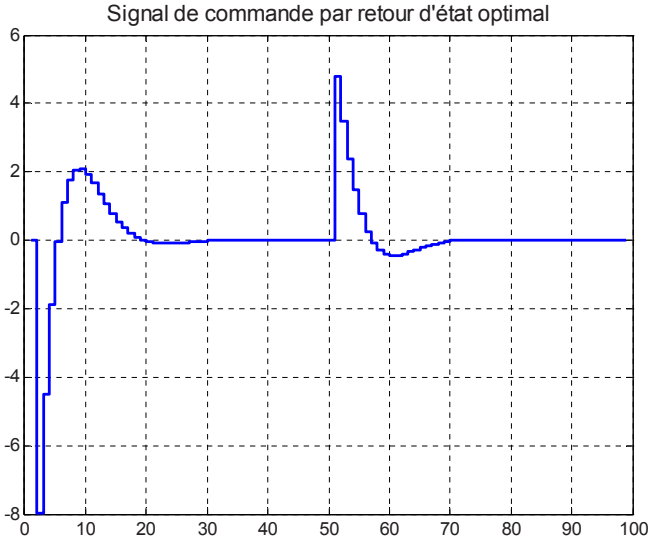
fichier cde_opt1.m (suite)

```
% algorithme de commande
u(1)=0; x(:,1)=[1 0]';
for i=2:N-1
 x(:,i)=A*x(:,i-1)+B*u(i-1);
 u(i)=-F(i,:)*x(:,i);
 if i==N/2
  x(2,i)=x(2,i)-1;
 end % perturbation sur la vitesse
end

figure(2)
plot(1:N-1,x(1,:))
hold on
plot(1:N-1,x(2,:),'-.')
grid
hold off
title('Evolution du vecteur d''état, retour optimal');
gtext('x1'),
gtext('x2'), gtext('perturbation sur la vitesse')
```



fichier cde_opt1.m (suite)

```
figure(3), stairs(1:N-1,u), grid
title('Signal de commande par retour d''état optimal')
```

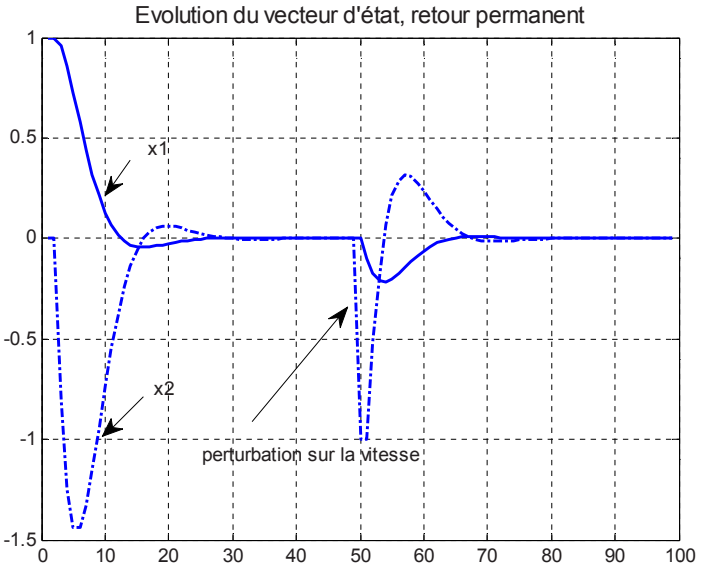


Les figures suivantes représentent les résultats de cette commande lorsqu'on implante la solution permanente de l'équation de RICCATI.

fichier cde_opt1.m (suite)

```
% retour d'état permanent
%------------------------

% coefficients du retour d'état permanent
F = [F(1,1) F(1,2)];

u(1)=0; x(:,1)=[1 0]';

for i=2:N-1
  x(:,i)=A*x(:,i-1)+B*u(i-1);
  u(i)=-F*x(:,i);

  if i==N/2
    x(2,i)=x(2,i)-1;
  end % perturbation sur la vitesse

end
```

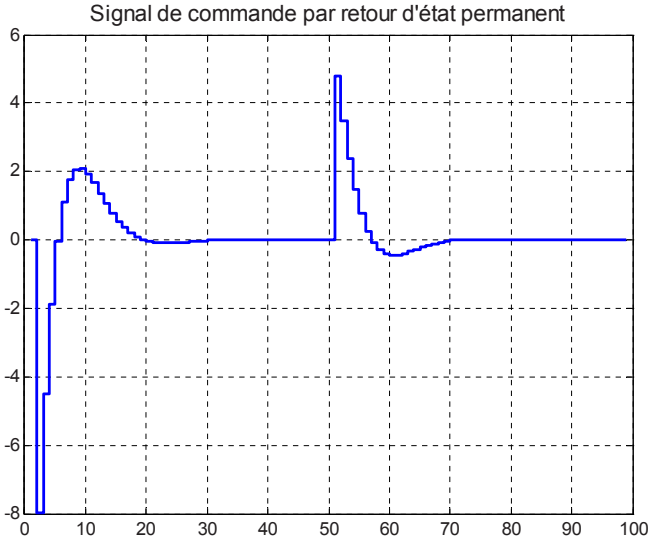
```
figure(4)
plot(1:N-1,x(1,:))
hold on

plot(1:N-1,x(2,:),'-.')
grid
hold off
title('Evolution du vecteur d''état, retour permanent');

gtext('x1')
gtext('x2')
gtext('perturbation sur la vitesse')
```



fichier cde_opt1.m (suite)

```
figure(5)
stairs(1:N-1,u);
grid
title('Signal de commande par retour d''état permanent')
```

Les résultats obtenus par un retour d'état permanents ne différent pas de beaucoup de ceux de la commande optimale avec un retour d'état variant dans le temps.



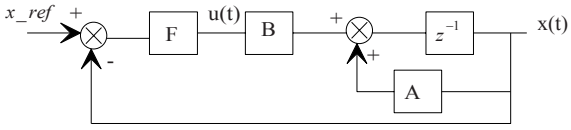
Régulation autour d'une consigne constante non nulle

Dans le programme suivant, on implante le retour d'état permanent pour une régulation autour d'une consigne constante de position. La deuxième composante du vecteur d'état est nulle.

L'état correspondant est représenté par le vecteur :

$$x\_ref = \begin{pmatrix} 5 \\ 0 \end{pmatrix}$$

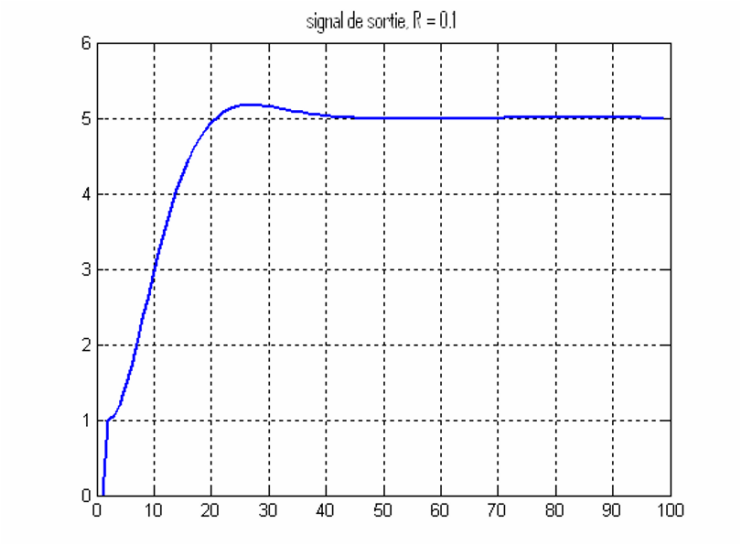
La loi de commande est schématisée comme suit :



fichier cde_opt2.m

```
% commande optimale par retour d'état
% régulation autour d'une consigne non nulle
clear all

% matrices du modèle d'état discret du processus
A =[1.0000 0.1000;0 1.0000];
B =[0.0050;0.1000];
```

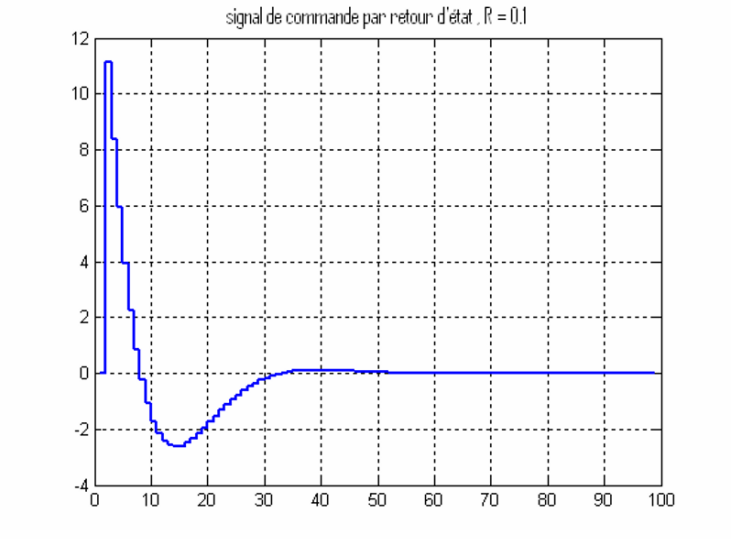
```
C =[1 0];

% horizon de commande
N=100;

% matrices de pondération
H=[1 0;0 0];
Q=H;
R=0.1;

% initialisation de la matrice P
P=H;

% calcul du retour d'état et de la matrice P
for i=N:-1:2
  F(i-1,:)=inv(B'*P*B+R)*B'*P*A;
  P = A'*(P-P*B*inv(B'*P*B+R)*B'*P)*A+Q;
end

% état de consigne
x_ref=[5 0]';

% algorithme de commande
u(1)=0;
x(:,1)=[1 0]';
y(1)=0;

for i=2:N-1

  % sortie position
  x(:,i)=A*x(:,i-1)+B*u(i-1);

% signal de commande
  u(i)=-[F(1,1) F(1,2)]*(x(:,i)-x_ref);

% signal de sortie
  y(i)=C*x(:,i);

end

figure(1)
plot(1:N-1,y);
grid
title(['signal de sortie, R = ' num2str(R)])
```



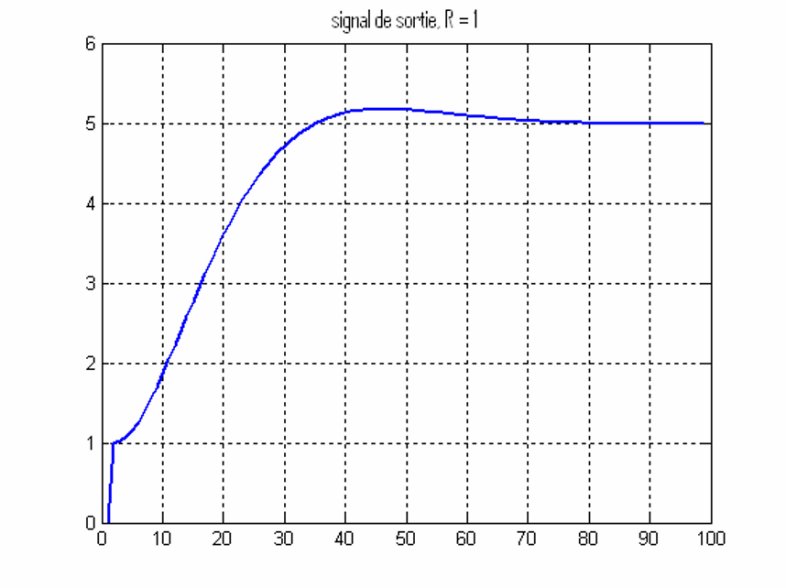
Le signal de sortie atteint la valeur de la consigne au bout d'une quarantaine de périodes d'échantillonnage.

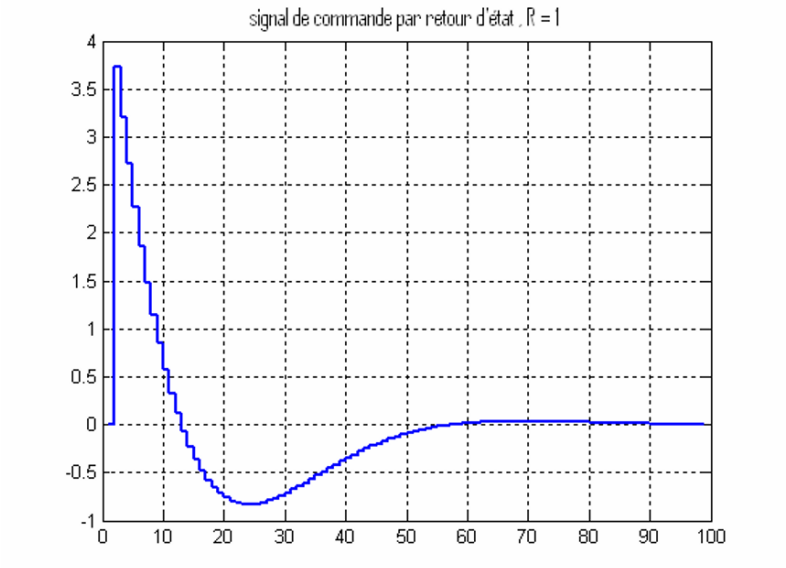
fichier cde_opt2.m (suite)

```
% tracé du signal de commande bloqué
figure(2), stairs(1:N-1,u); grid
txt = 'signal de commande par retour d''état , R = ';
title([txt, num2str(R)])
```



L'augmentation de la pondération `R`, a pour conséquence de rallonger le temps de réponse. Avec `R = 1`, nous obtenons une poursuite beaucoup plus lente et un signal de commande beaucoup plus faible à cause d'une pondération beaucoup plus forte.
Quelque soit la valeur de ce coefficient de pondération `R`, nous obtenons toujours une erreur statique nulle grâce à la présence de l'intégration mais au bout d'un temps de plus en long.





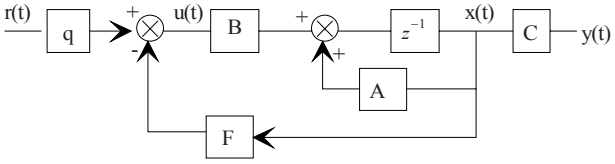
La dynamique est plus lente que précédemment à cause d'une plus forte pondération de la commande.

Régulation autour d'une consigne variant dans le temps

En général la loi de commande permettant de réguler autour d'une consigne r(t) non nulle,

$$u(t) = q\, r(t) - F\, x(t)$$

comporte un retour d'état et une commande anticipative.



L'équation d'état devient :

$$x(t+1) = A\,x(t) + B\,\left[q\, r(t) - F\, x(t)\right] \;= (A - B\,F)\, x(t) + B\, q\, r(t)$$

La sortie en régime permanent, s'exprime en fonction du signal de consigne par :

$$y = C(I - A + B\,F)^{-1} B\, q\; r$$

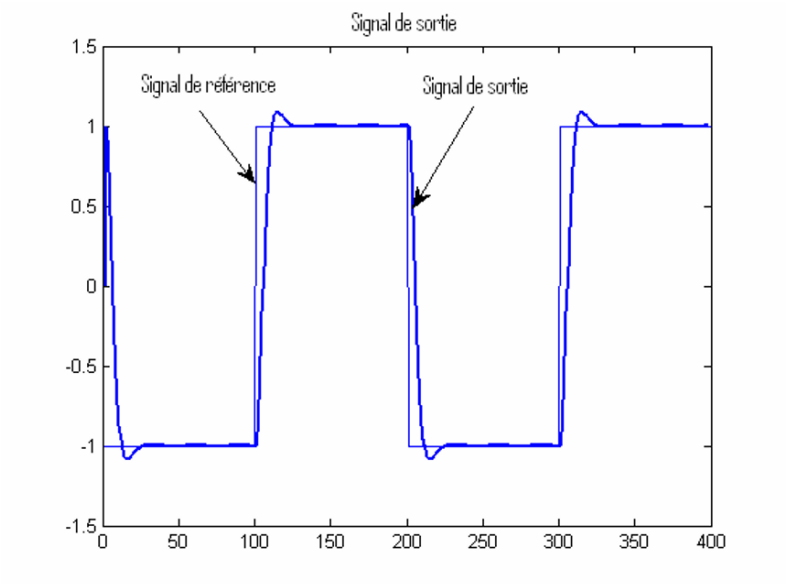
L'égalité entre la sortie et la consigne en régime permanent est obtenue avec la valeur suivante de l'anticipation :

$$q = \left[C(I - A + B\,F)^{-1} B \right]^{-1}$$

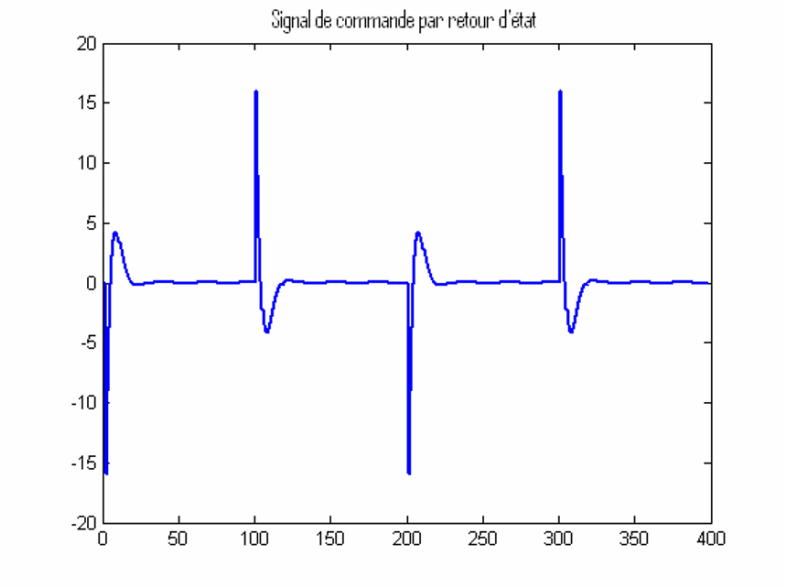
fichier cde_opt3.m

```
% commande optimale par retour d'état
% régulation autour d'une consigne non nulle

% matrices du modèle d'état discret du processus
A = [1.0000 0.1000;0 1.0000];
B = [0.0050;0.1000];
C = [1 0];

N = 400; % horizon de commande

% signal de référence carré
palier = ones(1,N/4); ref = [-palier palier -palier palier];

% matrices de pondération
H = [1 0;0 0]; Q = H; R = 0.01;
% initialisation de la matrice P
```

```
P = H;
for i = N:-1:2
  F(i-1,:) = inv(B'*P*B+R)*B'*P*A;
  P = A'*(P-P*B*inv(B'*P*B+R)*B'*P)*A+Q;
end
F = [F(1,1) F(1,2)];
% calcul de l'anticipation
q = inv(C*inv(eye(2)-A+B*F)*B);

% algorithme de commande
u(1) = 0;
x(:,1) = [1 0]';
y(1) = 0;

for i = 2:N-1
  x(:,i) = A*x(:,i-1)+B*u(i-1);
  u(i) = -F*x(:,i)+q*ref(i);
  y(i) = C*x(:,i); % sortie position
end

figure(1)
plot(1:N-1,y);
title('Signal de sortie')
hold on
plot(ref)
figure(2)
stairs(1:N-1,u)
grid
title('Signal de commande par retour d''état')
```



La figure suivante représente le signal de commande par retour d'état.



IV. La régulation PID

La régulation la plus répandue dans l'industrie est la commande P.I. (Proportionnelle et Intégrale) où la commande appliquée au processus est formée de 2 parties :

- une partie proportionnelle à l'erreur consigne-sortie Kp ,
- une partie intégrale de cette erreur $D(p)=\frac{Kp}{T_i\, p}$

L'expression analogique du régulateur PI est :

$$D(p)=Kp\,(1+\frac{1}{T_i\, p})$$

Une façon de discrétiser ce régulateur serait d'utiliser les équivalences analogique et discrète de la dérivée.

$$p \approx 1-z^{-1}$$

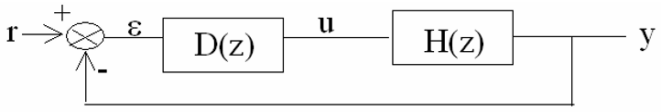
La fonction intégrale est conservée ($\frac{1}{p} \approx \frac{1}{1-z^{-1}}$) par la discrétisation. L'expression discrète du régulateur est la suivante :

$$D(z^{-1}) = \frac{Kp}{T_i} \quad \frac{1 + T_i - T_i z^{-1}}{1 - z^{-1}}$$

On désire réguler un système discret de fonction de transfert :

$$H(z) = \frac{b_1 z - 1}{1 - a_1 z - 1} \text{ Avec } \begin{cases} a_1 = 0{,}8 \\ b_1 = 1 \end{cases}$$

Le schéma du système régulé (boucle fermée) est :



où r, y, ε et u sont respectivement les signaux de consigne, de sortie, d'erreur et de commande issue du régulateur.

L'équation caractéristique (dénominateur) de la fonction de transfert en boucle fermée est : $A(z^{-1}) = 1 + [K_p b_1 (\frac{1 + T_i}{T_i}) - a_1 - 1] z^{-1} + (a_1 - K_p b_1) z^{-2}$ et le gain statique est égal à 1.

En notant l'équation caractéristique sous la forme :

$$A(z^{-1}) = 1 - \alpha_1 z^{-1} - \alpha_2 z^{-2}$$

Les paramètres α_1 et α_2 sont donnés en fonction du coefficient d'amortissement ς et de la pulsation propre non amortie normalisée $\Omega = \omega_0 T$ ($0 < \Omega < 1$) du système du 2nd ordre de la boucle fermée par :

$$\alpha_1 = 2 e^{-\varsigma \Omega} \cos(\Omega \sqrt{1 - \varsigma^2})$$
$$\alpha_2 = -e^{-2\varsigma \Omega}$$

Les expressions du gain K_p et de la constante de temps T_i du régulateur sont :

$$K_p = \frac{a_1 - e^{-2\varsigma \Omega}}{b_1}$$
$$\frac{1 + T_i}{T_i} = \frac{a_1 + 1 - 2 e^{-\varsigma \Omega} \cos(\Omega \sqrt{1 - \varsigma^2})}{K_p b_1}$$

On se propose de calculer ces valeurs pour $\Omega = 0.5$ et $\varsigma = \frac{\sqrt{2}}{2}$

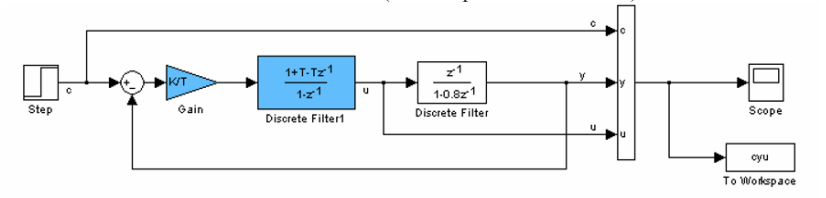
$$F(z^{-1}) = \frac{K_p}{T_i} \frac{b_1[1+T_i-T_i z^{-1}]\, z^{-1}}{1+[K_p b_1 (\frac{1+T_i}{T_i}) - a_1 - 1]z^{-1} + (a_1 - K_p b_1) z^{-2}}$$

fichier KT_pid.m

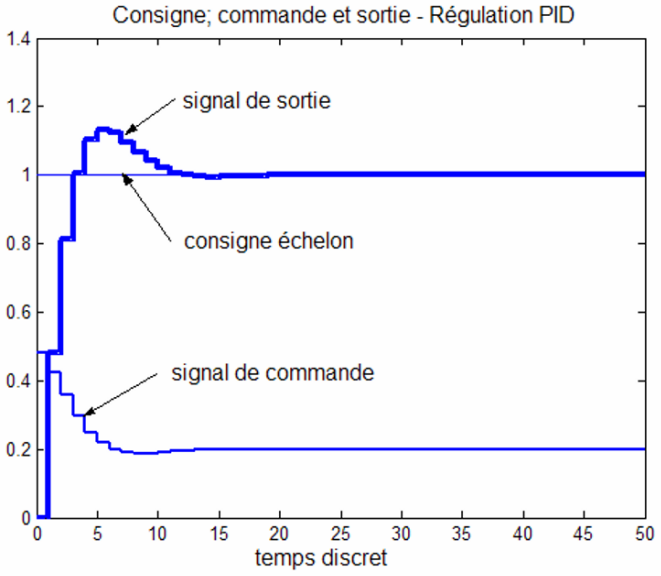
```
a=0.8; b=1; w0=0.5; z=sqrt(2)/2;
K= (a-exp(-2*z*w0))/b;
x=(a+1-2*exp(-z*w0)*cos(w0*sqrt(1-z*z)))/(K*b);
T=1/(x-1);
```

```
>> K
K =
    0.3069
>> T
T =
    1.7484
```

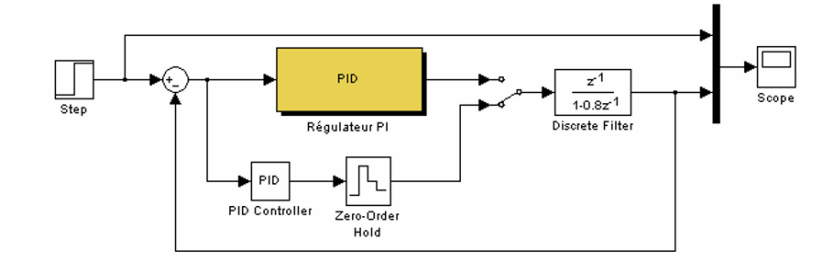
Le modèle `pid_1.mdl` propose cette régulation. Les paramètres `Kp` et `Ti` sont entrés dans le `Callback InitFcn` du modèle (voir chapitre `Callbacks`).



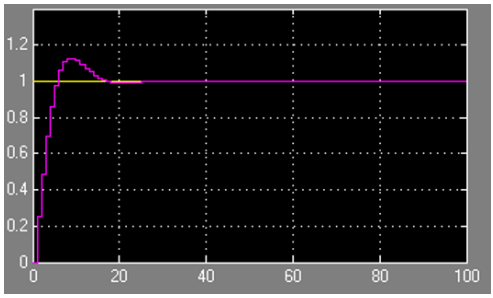
La figure suivante affiche les 3 signaux de consigne, du signal de commande du PID et de la sortie du processus.



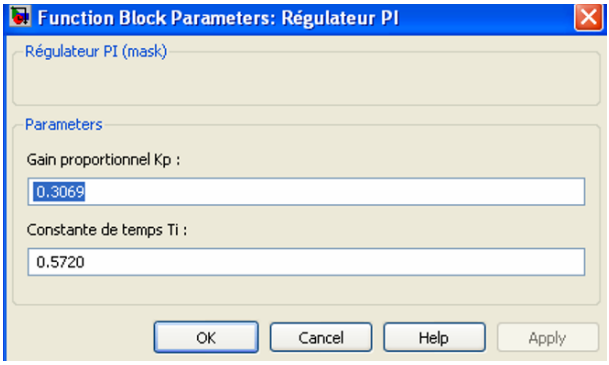
- ***Régulateur analogique suivi d'un bloqueur d'ordre 0***



Comme le PID est analogique, on le fait suivre par un bloqueur d'ordre 0.



Les paramètres du régulateur sont entrés dans la boite de dialogue du masque du PID (voir chapitres Masques et sous-systèmes).



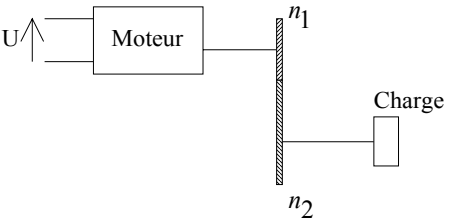
V. La boite à outils "Control System Toolbox"

V.1. Etude d'un système d'un moteur avec charge

MATLAB dispose de la boite à outils "`Control System Toolbox`" pour la modélisation de procédés et la mise en oeuvre de régulateurs.

Afin d'étudier quelques unes des fonctions prévues, nous allons considérer l'asservissement de la position angulaire d'une charge couplée à travers un réducteur à l'arbre d'un moteur à courant continu.

On utilisera la correction de la vitesse de l'arbre moteur par boucle tachymétrique et par un régulateur numérique.



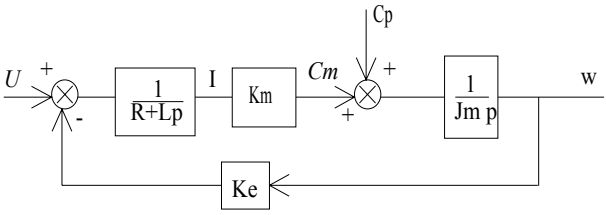
Les paramètres du système sont :

Ke = 6 10^{-2} V/(rad/s)	Coefficient de force contre-électromotrice
Kg = 5 10^{-2} V/(rad/s)	Coefficient de retour tachymétrique
Km = 5 10^{-1} N.m/A	Coefficient de couple moteur
R = 10Ω , L = 10 mH	Résistance et self d'induit
N = n_2/n_1 = 20	Rapport de réduction
Jm = 10^{-4} kg m^2	Moment d'inertie de l'arbre moteur
Jc = 0.15 kg m^2	Moment d'inertie de la charge

Etude du système moteur sans charge

On s'intéresse à l'étude du système en boucle ouverte ayant comme sortie la tension u(t) appliquée à l'induit et comme sortie la vitesse w de l'arbre moteur.

En désignant par I, Cm et Cp, respectivement le courant d'induit du moteur, le couple moteur et le couple perturbateur, on déduit, à partir des équations électromécaniques, le schéma bloc suivant :



Nous voulons calculer la fonction de transfert de ce système en l'absence de couple perturbateur, tracer sa réponse indicielle et sa réponse en fréquences (diagrammes de Bode et de Nyquist).et étudier sa réponse à une perturbation indicielle de couple perturbateur.

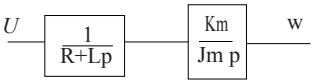
La boite à outils "`Control System Toolbox`" contient différentes fonctions pour le calcul de la fonction de transfert et du modèle d'état d'un système bouclé.

La fonction `series` permet de calculer la fonction de transfert ou le modèle d'état de deux systèmes mis en cascade.

Considérons la mise en cascade de 2 systèmes de fonctions de transfert respectives $H_1(p)$, $H_2(p)$. La fonction de transfert global est donnée par :

```
[numG, denG] =  series(numH1, denH1, numH2, denH2)
```

Dans le cas de la chaîne directe du moteur :



fichier moteur.m

```
% asservissement de la vitesse de la  charge

% paramètres électromécaniques
Km = 5e-1; Kg = 0.05; Ke = 6e-2;
R = 10; L = 1e-2;
Jm = 1e-4; Jc = 0.15;
N = 20;

% inertie ramenée sur l'arbre moteur
Jt = Jm + Jc/(N^2);

% fonction de transfert H1
% vitesse angulaire du moteur sans charge
% fonction de transfert Hcd de la chaîne directe
[num_cd, den_cd] = series(1, [L R], Km, [Jm 0]);
[num_cd, den_cd] = minreal(num_cd, den_cd);
disp('[transfert de la chaîne directe Hcd :')
printsys(num_cd,den_cd,'p')
```

```
0 pole-zero(s) cancelled
[transfert de la chaîne directe Hcd :
num/den =

      500000
   ------------
   p^2 + 1000 p
0 pole-zero(s) cancelled
```

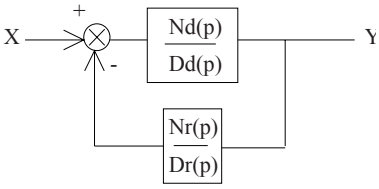
La fonction `minreal` utilisée précédemment permet de simplifier la fonction de transfert obtenue par la compensation de pôles.

On obtient ainsi une fonction de transfert normalisée; le coefficient de plus haut degré du dénominateur est égal à l'unité.

Dans notre cas, il n'y a pas eu de compensation de pôles.

La fonction de transfert du système bouclé peut être obtenue à l'aide de la commande `feedback`.

Considérons la fonction de transfert H(p) = Y(p)/X(p) du système suivant :



En notant `Nd`, `Dd`, `Nr`, et `Dr`, les polynômes des chaînes directe et de retour, le numérateur et le dénominateur de H(p) sont obtenus par cette commande en spécifiant le paramètre `'-1'` de la contre réaction.

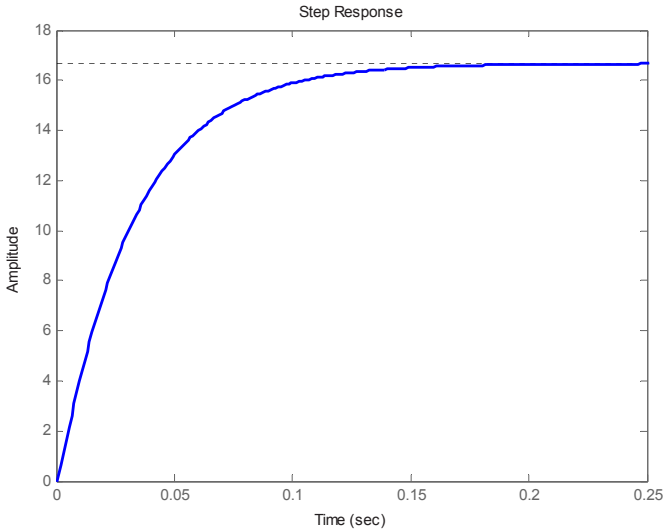
```
[numH,denH]= feedback(Nd,Dd,Nr,Dr,-1)
```

fichier moteur.m (suite)

```
% fonction de transfert du moteur
[numH1,denH1] = feedback(num_cd,den_cd,Ke,1,-1);
[numH1,denH1] = minreal(numH1,denH1);

disp('Fonction de transfert H1 du moteur :')

printsys(numH1,denH1)

num/den =
          5e+005
   ---------------------
   s^2 + 1000 s + 3e+004
```

```
>> step(numH1,denH1)
```

La courbe suivante donne la réponse indicielle du moteur.



On peut vérifier facilement que ce résultat correspond parfaitement à la fonction de transfert obtenue par le calcul.

$$H_m(p) = \frac{Km}{Ke\,Km + R\,Jm\,p + L\,Jm\,p^2}$$

fonction de transfert théorique

```
>> numTh = Km; denTh = [L*Jm R*Jm Ke*Km];
>> disp('Fonction de transfert théorique H1 du moteur :')
>> [numTh,denTh] = minreal(numTh,denTh)
>> printsys(numTh,denTh)

num/den =
            5e+005
   ---------------------
   s^2 + 1000 s + 3e+004
```

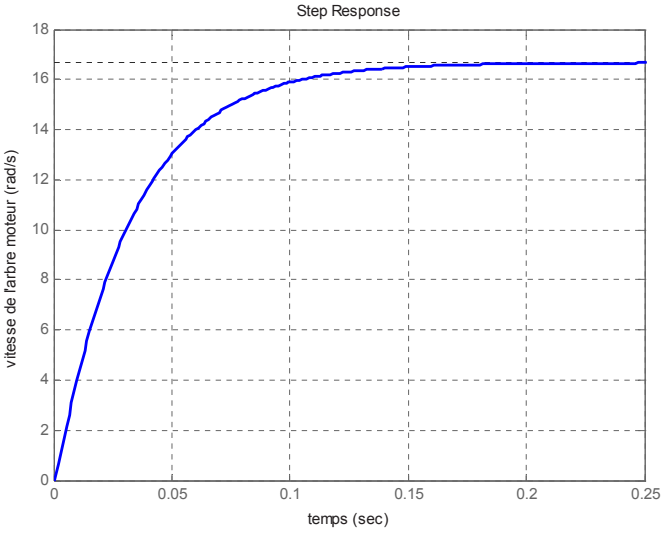
La réponse du système à un signal `x` peut être obtenue à l'aide de la commande `lsim`.

```
lsim(numH1,denH1,x)
```

Dans l'étude de procédés, on utilise souvent les réponses à l'échelon et à l'impulsion.
La boite à outils "`Control System Toolbox`" dispose des fonctions `step` et `impulse` pour obtenir les réponses indicielle et impulsionnelle.
Les paramètres d'appel peuvent être les polynômes de la fonction de transfert ou les matrices d'état du système.

fichier moteur.m (suite)

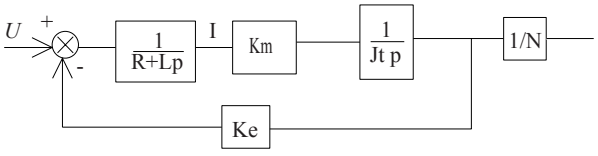
```
% réponse indicielle, vitesse moteur non chargé
figure(1)
step(numH1,denH1)

grid
ylabel('vitesse de l''arbre moteur (rad/s)')
xlabel('temps')
title ('réponse indicielle vitesse, moteur non charge);
k = dcgain(numH1,denH1);

disp('Gain statique en rad/s/V')
disp(num2str(k));
```




```
Gain statique : 16.67
```

Etude du système moteur avec charge

On s'intéresse au système ayant comme sortie la position angulaire θ de la charge sur l'arbre en fonction de la tension d'induit du moteur.



fichier moteur.m (suite)

```
% système du moteur chargé
[num_cd, den_cd] = series(1, [L R], Km, [Jt 0]);
[numH2,denH2] = feedback(num_cd,den_cd,Ke,1,-1);
denH2 = N*denH2;
[numH2,denH2] = minreal(numH2,denH2);
disp('Fonction de transfert, vitesse de la charge')

printsys(numH2,denH2)
0 pole-zeros cancelled
Fonction de transfert de vitesse de la charge
num/den =
           5263
   ------------------
   s^2 + 1000 s + 6316
```

Avant de corriger le système, on a besoin d'étudier ses caractéristiques en boucle ouverte, à travers ses réponses indicielle, impulsionnelle, ses réponses en fréquences afin de déterminer les marges de gain et de phase, etc.

De plus, un modèle de haut degré peut être simplifié. On s'intéresse dans ce qui suit, à l'étude de la réponse indicielle.

L'allure de cette réponse, donnée dans la figure suivante, est très proche de celle d'un système du premier ordre avec une intégration, de la forme :

$$\frac{G_S}{p(1+\tau p)}$$

G_S et τ sont le gain statique et la constante de temps du système de premier ordre ayant comme sortie la vitesse angulaire de la charge.

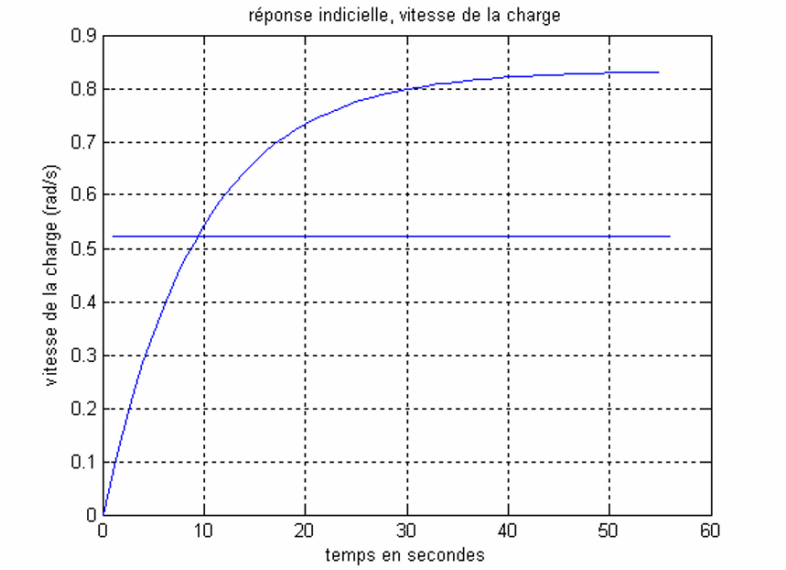
Le gain statique G_S est égal à celui du moteur sans chargé, divisé par le rapport de réduction N.

La constante de temps est le temps au bout duquel, la réponse impulsionnelle atteint 63% de la valeur statique. On utilisera la commande `zoom` pour mesurer la valeur de ce paramètre.

fichier moteur.m (suite)

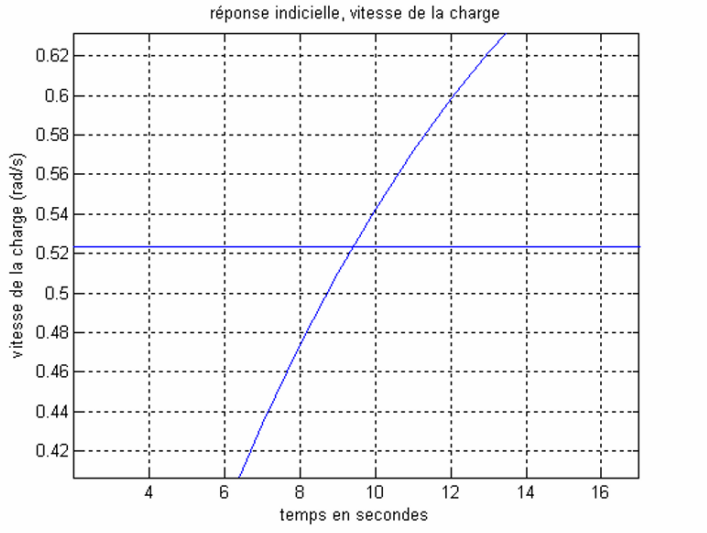
```
% réponse indicielle, vitesse charge
rep_ind = step(numH2,denH2);
t = 0:length(rep_ind)-1;
% tracé de la réponse indicielle
figure(2), plot(t,rep_ind), hold on
title('réponse indicielle, vitesse de la charge')
xlabel('temps en secondes')
ylabel('vitesse de la charge (rad/s)')
```

```
% calcul de la constante de temps
val_stat = rep_ind(length(rep_ind));
trait = 0.63*val_stat*ones(1,length(rep_ind));
plot(trait), grid, hold off
```



La lecture graphique de la constante de temps peut se faire en invoquant la commande `zoom`. Nous pouvons aussi modifier l'axe des temps par la commande `axis`.

```
>> zoom on
```

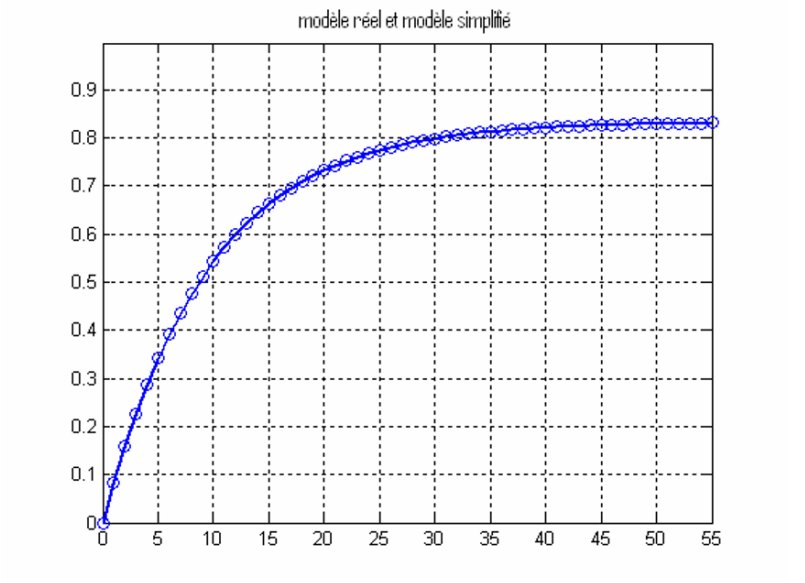


Après sélection à la souris, de rectangles qui entourent le point d'intersection de la courbe avec le trait représentant 63% de la valeur maximale, on obtient une constante de temps de 6 secondes environ.

Le modèle peut être considéré comme un premier ordre de constante de temps de 6 secondes.

fichier moteur.m (suite)

```
% modèle simplifie
num1 = k/N; den1 = [6 1];
ind1 = step(num1,den1);

t = 0:min(length(rep_ind), length(ind1))-1;

figure(3)

% tracé des réponses indicielles des modèles réél
% et simplifié

plot(t,ind1(t+1),'o')
hold on
plot(t,rep_ind(t+1))
grid
title('modèle réel et modèle simplifié')
axis([0 length(t)-1 0 val_stat*1.2])
```



Les réponses impulsionnelles des modèles réel et simplifié étant semblables, le système complet peut être alors décrit parla fonction de transfert

$$H(p)=\frac{0.833}{1+6p}$$

La possibilité de simplification du modèle initial peut être montrée en calculant les valeurs de ses pôles.

```
>> disp('Pôles du modèle réel :'),
>> roots(denH2)

Pôles du modèle réel :

ans =

 -993.6438
   -6.3562
```

Le mode correspondant au pôle `p = -993.6438` s'éteignant très rapidement; peut être négligé.

La fonction `residue` permet de décomposer cette réponse impulsionnelle en ses différents modes.

```
>> [r,p,k] = residue(numH2,denH2)

r =
   -5.3309
    5.3309

p =
 -993.6438
   -6.3562

k =
     []
```

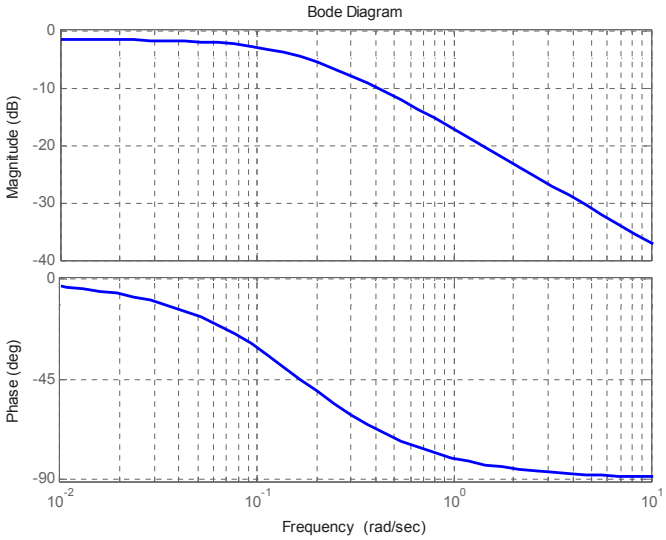
La réponse impulsionnelle du système peut être alors décomposée comme suit :

$$y(t)=5.3309\ (e^{-6.3562\ t}-e^{-993.6438\ t})$$

Les diagrammes de `Bode` peuvent être directement tracés en invoquant la commande `bode`.

Dans le cas du modèle simplifié du premier ordre :

```
>> bode(0.833,[6 1]);
```



On peut aussi calculer le module et la phase pour les tracer ensuite avec des unités semi - logarithmiques par `semilogx`.

`[mod,phase,w] = bode(num, den)` : retourne le module, la phase et l'étendue des pulsations.

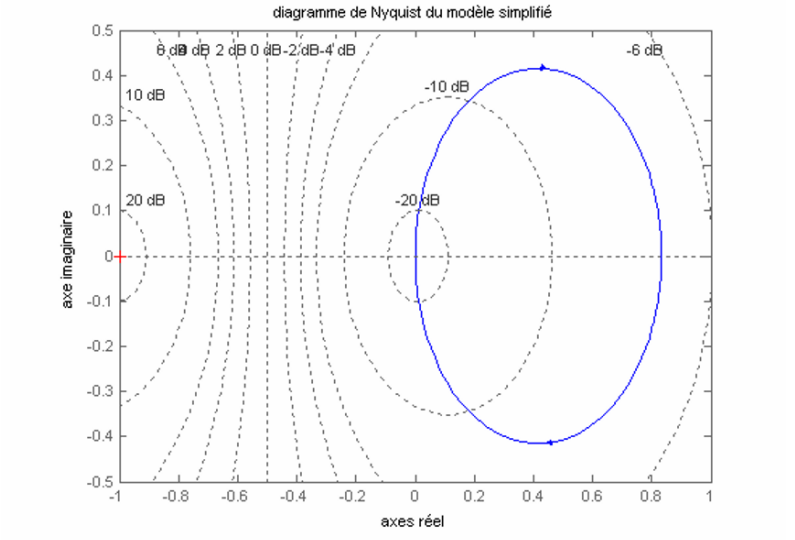
Les diagrammes de Bode servent essentiellement à mesurer les marges de gain et de phase. La commande `margin(mod,phase,w)` trace les diagrammes de `Bode` et affiche les marges de gain et de phase.

Les tracés des diagrammes de `Nyquist` et de `Nichols` sont obtenus respectivement par les fonctions `nyquist` et `nichols`.

`nyquist (num,den)` et `nichols(num,den)`

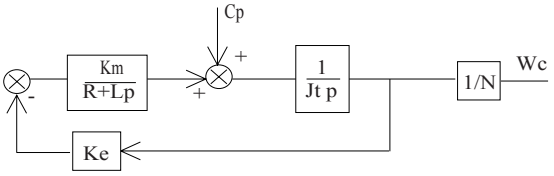
Nous traçons ci-après le digramme de Nyquist du modèle simplifié du premier ordre.

```
>> nyquist(0.833,[6 1])
>> grid
>> title('diagramme de Nyquist du modèle simplifié')
>> xlabel('axes réel')
>> ylabel('axe imaginaire')
```



Effet d'une perturbation de couple perturbateur

On se propose d'étudier l'effet d'un échelon de couple perturbateur sur la vitesse de la charge.



On peut obtenir la fonction de transfert Wc/Cp à l'aide de la commande `feedback`.

```
>> [num,den] = feedback(1,[Jt 0],Ke*Km,[L R],-1);
>> den  = den*N ; disp('Fonction de transfert Wc/Cp')
>> printsys(num,den)
Fonction de transfert Wc/Cp
num/den =
            0.01 s + 10
   ----------------------------
   0.000095 s^2 + 0.095 s + 0.6
```

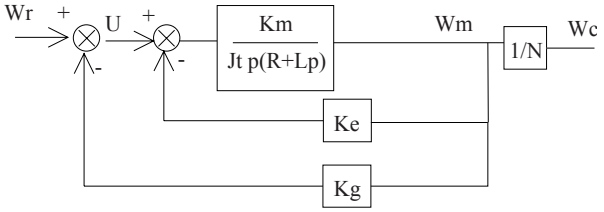
```
>> dcgain(num,den)
ans =
   16.6667
```

La réponse au couple perturbateur possède la même dynamique et le même gain statique que celle de la vitesse de l'arbre moteur vis-à-vis de la tension d'induit.

Correction tachymétrique de la vitesse du moteur

On désire asservir la vitesse angulaire de l'arbre moteur par l'intermédiaire d'un retour tachymétrique de coefficient Kg.

Le système bouclé par retour tachymétrique est :



A partir de la fonction de transfert Wm/U et le retour tachymétrique Kg, on obtient la fonction de transfert Wc/Wr.

fichier moteur.m (suite)

```
% retour tachymétrique
% fonction de transfert, vitesse moteur
[numH3, denH3] = feedback(Km,conv([Jt 0],[L R]),Ke,1,-1);
disp('Fonction de transfert Wm/U :')
printsys(numH3,denH3,'p')

[numH4,denH4] = feedback(numH3,denH3,Kg,1,-1);
disp('Fonction de transfert Wm/Wr :')
printsys(numH4,denH4,'p')
% fonction de transfert, vitesse charge
denH5 = denH4*N; numH5 = numH4;
disp('Fonction de transfert Wc/Wr :')
printsys(numH5,denH5,'p')
gain_stat = dcgain(numH5,denH5)

Fonction de transfert Wm/U :
num/den =
                  0.5
   --------------------------------
   4.75e-006 p^2 + 0.00475 p + 0.03

Fonction de transfert Wm/Wr :
num/den =
                   0.5
   ---------------------------------
   4.75e-006 p^2 + 0.00475 p + 0.055
```

```
Fonction de transfert Wc/Wr :
num/den =
                0.5
   ----------------------------
   0.000095 p^2 + 0.095 p + 1.1

gain_stat =
    0.4545
```

Le calcul théorique de la fonction de transfert Wm/Wr donne :

$$\frac{Wm}{Wr} = \frac{Km}{Km(Ke+Kg)+JtR\,p+JtL\,p^2}$$

```
>> disp('resultat theorique Wm/Wr')
>> num = Km;
>> den = [Jt*L Jt*R Km*(Kg+Ke)];
```

```
>> printsys(num,den);

résultat théorique Wm/Wr

num/den =
                  0.5
   ---------------------------------
   4.75e-006 s^2 + 0.00475 s + 0.055
```

Discrétisation du système analogique

Nous désirons corriger la vitesse de la charge à l'aide d'un régulateur numérique. Nous allons utiliser le modèle simplifié du premier ordre obtenu précédemment pour la synthèse du correcteur.

Pour obtenir le modèle discret à partir du modèle analogique échantillonné à une cadence T, on utilise la commande `c2dm`. Avec la syntaxe suivante :

```
[numd,dend] = c2dm(numa,dena,T,'zoh')
```

on discrétise le système analogique précédé du bloqueur d'ordre 0 (zero-order hold).

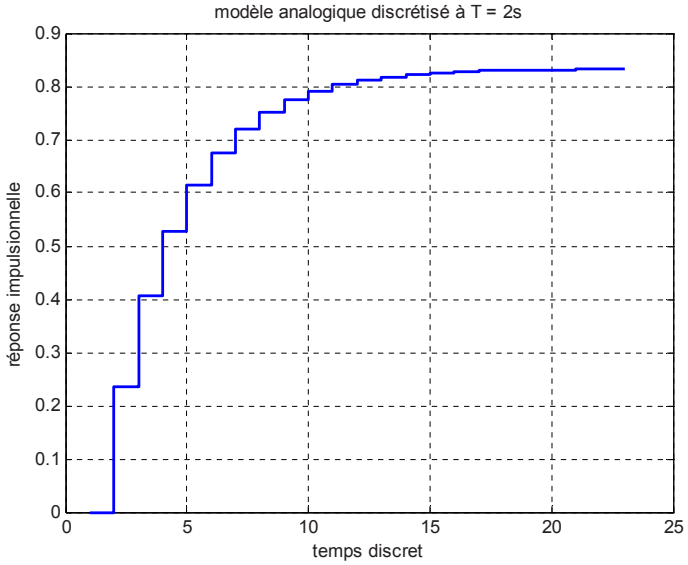
fichier reg_num.m

```
% syntèse de régulateur numérique
% modèle analogique
na = 0.833; da = [6 1];
% discrétisation du système
[nd,dd] = c2dm(na,da,2,'zoh');
printsys(nd,dd,'z'), figure(1)
indN = dstep(nd,dd);
stairs(indN), grid, xlabel('temps discret')
ylabel('réponse impulsionnelle')
```

```
title('modèle analogique discrétisé à T = 2s')

Fonction de transfert du modèle discret
num/den =

     0.23613
   -----------
   z - 0.71653
```

La discrétisation conserve l'ordre et le gain statique du modèle analogique.

```
>> ddcgain(nd,dd)
ans =
    0.8330
```



Les diagrammes de Nyquist et de Bode d'un modèle discret peuvent être obtenus et tracés en utilisant les fonctions `dnyquist` et `dbode`.

```
>> dbode(nd,dd,2)
```

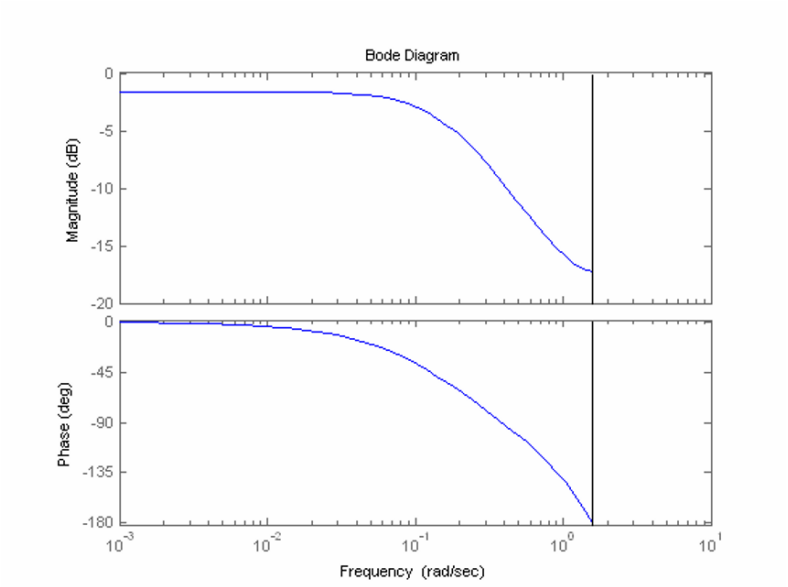
On spécifie la période d'échantillonnage `Ts`, qui est choisie égale à 2s dans ce cas.

Cette fonction accepte aussi un modèle d'état par la forme :

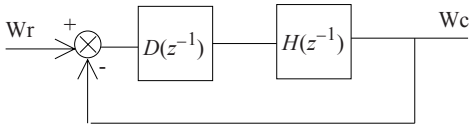
```
>> dbode(A,B,C,D,Ts,IU)
```

Elle retourne le module et la phase dans les vecteurs `Mag` et `Phase`.

[MAG,PHASE,W] = dbode (A,B,C,D,Ts,...)



On peut utiliser indifféremment la fonction de transfert du système discret ou son modèle d'état. En notant $H(z^{-1})$ et $D(z^{-1})$ les fonctions de transfert du modèle discret et du régulateur, le schéma de la loi de commande est, en général, le suivant :



avec Wr la consigne de vitesse.

Nous désirons que le système en boucle fermée se comporte comme un système du premier ordre de gain statique unité et de pôle $\alpha = 0.8$, soit une fonction de transfert en boucle fermée :

$$F(z^{-1}) = \frac{(1-a)\,z^{-1}}{1-a\,z^{-1}}$$

Le modèle du processus étant :

$$H(z^{-1}) = \frac{0.2361}{z - 0.7165} = \frac{b_1\, z^{-1}}{1 - a\, z^{-1}}$$

l'expression du correcteur peut être obtenue par l'expression suivante, obtenue après le calcul de la fonction de transfert en boucle fermée.

$$D(z^{-1}) = \frac{F(z^{-1})}{H(z^{-1})\,[1 - F(z^{-1})]}$$

Dans le cas des spécifications précédentes, ce correcteur a pour expression :

$$D(z^{-1}) = \frac{(1-\alpha)(1 - a\, z^{-1})}{b_1\,(1 - z^{-1})}$$

Ce correcteur compense les pôles et les zéros du processus et introduit une intégration.

Pour réaliser un retour unitaire, MATLAB dispose de la fonction `cloop`, de syntaxe :

```
cloop(num, den, signe)
```

`num, den` : numérateur et dénominateur de la chaîne directe,
`signe` : `-1` pour une contre-réaction et `+1` pour une `réaction`.

fichier reg_num.m (suite)

```
% expression du régulateur
alfa = 0.8; numD = (1-alfa)*dd; denD = conv([1 -1],nd);
disp('Expression du régulateur :')
printsys(numD,denD,'z')

Expression du régulateur :
num/den =
  0.2 z - 0.14331
   -------------------
   0.23613 z - 0.23613
```

Par l'utilisation de la commande `cloop`, on peut vérifier que la fonction de transfert en boucle fermée est identique à celle que nous avons spécifiée.

```
>> [numF, denF] = cloop(conv(numD,nd),conv(denD,dd),-1);
>> numF = suppz(numF);
>> denF = suppz(denF);
>> [numF, denF] = minreal(numF, denF);
>> disp('Fonction de transfert en boucle fermée :')
>> printsys(numF,denF,'z')

1 pole-zeros cancelled
Fonction de transfert en boucle fermée :

num/den =
```

```
    0.2
   -------
   z - 0.8
```

La fonction `suppz` permet de supprimer les premiers coefficients nuls de polynômes.

Mise en oeuvre du correcteur

Nous nous intéressons dans ce qui suit, à la mise en oeuvre du régulateur, lorsqu'on spécifie un signal de consigne carré.

Pour générer un signal carré, on dispose de la fonction `square` ayant les syntaxes :

`square(x)` : génère un signal carré de période 2π pour les éléments du vecteur x,

`square(x,rc)` : génère un signal carré de période 2π et de rapport cyclique `rc`.

Pour générer une période, nous pouvons utiliser les expressions logiques. Nous nous intéressons à l'écriture d'une fonction permettant de générer un signal carré, que l'on nommera `carre`.

Les paramètres d'appel sont la longueur de la période, les valeurs minimale et maximale du signal et le nombre de périodes.

fichier carre.m

```
function x = carre(xmin,xmax,per,N_per,flag)
% génération d'un signal carré
% x          : signal généré
% xmin, xmax    : valeurs min et max du signal
% per        : période du signal
% N_per      : nombre de périodes
% flag       : 0 pour que commence par sa valeur min
%            : 1 pour que x commence par sa valeur max

x =[];

% génération de la période
t = 0:per-1;
xP = (~(t<=floor(per/2)-1))+ flag*(~(t>floor(per/2)-1));

for k = 0:N_per-1
x = [x xP];
end

% mise à l'échelle des amplitudes
x = xmin*(x == min(x))+xmax*(x == max(x));
```

Pour générer un échelon, il suffit de spécifier un nombre de périodes égal à 1.
En notant $\varepsilon(t)$ l'erreur entre la consigne et la sortie du processus, à l'instant discret t, et après simplification de l'expression du correcteur :

$$D(z^{-1}) = \frac{\alpha_1 - \alpha_2\, z^{-1}}{1 - z^{-1}} = \frac{0.8471 - 0.6069\, z^{-1}}{1 - z^{-1}}$$

La variation de la commande $\Delta u(t)$ est donnée par :

$$\Delta u(t) = \alpha_1\, \varepsilon(t) - \alpha_2 \varepsilon(t-1$$

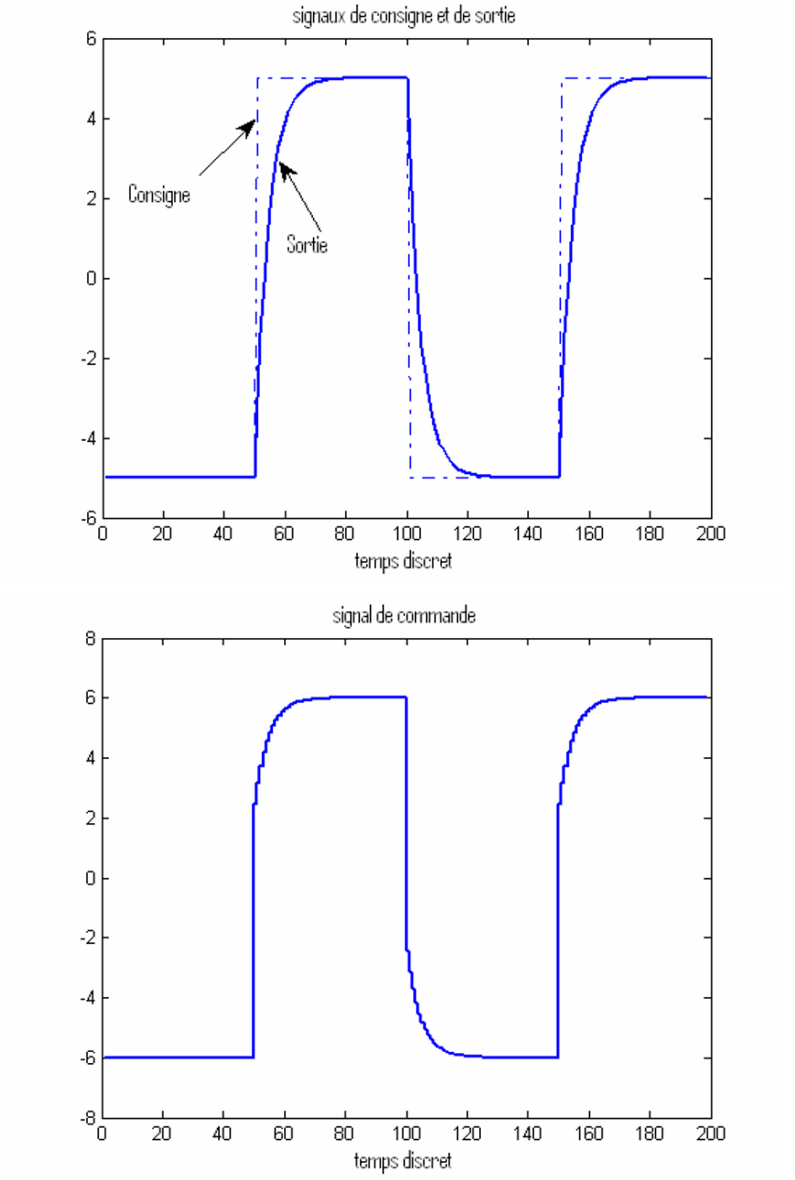
La sortie du processus est donnée par l'équation de récurrence :

$$y(t) = a_1 y(t-1) + b_1 u(t-1)$$

fichier commande.m

```
clc
% Implantation du régulateur

% Génération du signal de consigne carré
r =  carre(-5,5,100,2,0);
% coefficients du régulateur
alfa1 = 0.8471; alfa2 = 0.6069;
% coefficients du modèle du procédé
a1 = 0.7165; b1 = 0.2361;
% initialisations
err(1) = 0;
u(1) = r(1)/ddcgain(b1,[1 -a1]);
du(1) = 0;
y(1) = r(1);

for i = 2:length(r)-1
 % sortie du processus
 y(i) = a1*y(i-1) + b1*u(i-1);

 % erreur
 err(i) = r(i+1) - y(i);

 % calcul de la commande
 du(i) = alfa1*err(i) - alfa2*err(i-1);
 u(i) = u(i-1) + du(i);
end
% tracé des signaux de consigne et de sortie
figure(1), plot(r,'-.')
hold on
plot(y)
xlabel('temps discret')
title('signaux de consigne et de sortie')
axis([0 length(r) min(r)-1 max(r)+1])
figure(2) % tracé du signal de commande
stairs(u), xlabel('temps discret'),
title('signal de commande')
```

Les figures suivantes représentent les signaux de consigne et de sortie du processus. On remarque bien que la dynamique de poursuite correspond à un premier ordre de pôle 0.8. L'erreur statique est nulle grâce à la présence d'une intégration dans la loi de commande.



V.2. Le système linéaire et invariant dans le temps, LTI

Dans MATLAB, un système LTI peut être défini par sa fonction de transfert, ses équations d'état, ses zéros, gain et pôles ainsi que par sa réponse en fréquences.

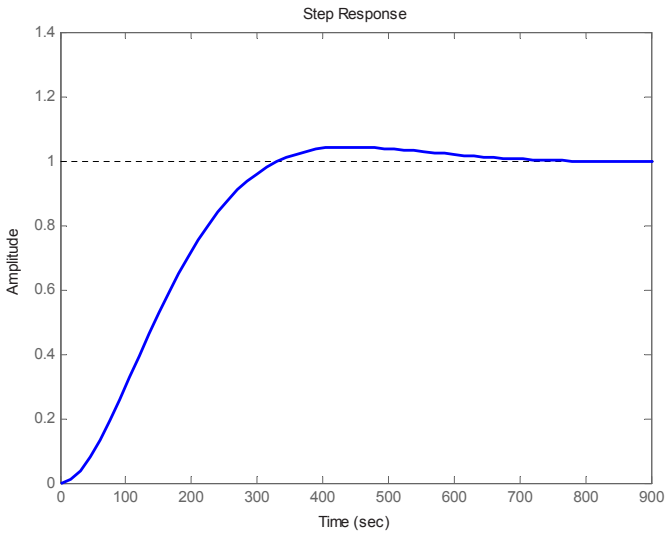
V.2.1. Fonction de transfert

On peut créer ue fonction de transfert en spécifiant son numérateur et son dénominateur en utilisant la fonction `tf`.

Si l'on veut spécifier un système du 2nd ordre de coefficient d'amortissement optimal $\zeta = \frac{\sqrt{2}}{2}$ et une pulsation propre ω_0, soit la fonction de transfert :

$$H(p) = \frac{1}{1 + 2\zeta \frac{p}{\omega_0} + \frac{p^2}{\omega_0^2}}$$

```
z=sqrt(2)/2 ;
w0=0.01;
num = [1];
den = [1/w0^2 2*z/w0 1];
sys = tf(num,den)
step(sys)
```



Notons que `step` donne la réponse indicielle quelque soit la définition de `sys` (équation d'état, ou autre).

Une autre façon plus simple d'écrire, en spécifiant le paramètre 's' de Laplace :

```
>> s=tf('s');
>> sys=1/(10000*s^2+141.4*s+1)
Transfer function:
           1
-----------------------
10000 s^2 + 141.4 s + 1
```

V.2.2. Zéro-Pôle-Gain

Pour spécifier la fonction de transfert suivante :

$$H(p)=\frac{1}{(p+2)(p+3)}$$

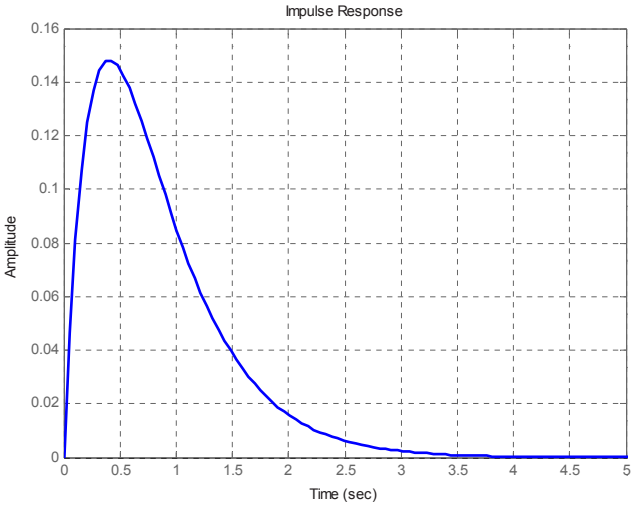
par la méthode ZPK, nous notons :

- 2 pôles de -2 et -3,
- Pas de zéro,
- Un gain de 1

```
>> sys = zpk([],[-2 -3],[1])
Zero/pole/gain:
     1
-----------
(s+2) (s+3)
```

La réponse impulsionnelle est obtenue par la commande `impulse`.

```
>> impulse(sys)
```



V.2.3. Espace d'état

La commande `ss` permet de construire le modèle d'état en spécifiant les matrices d'état `A`, `B`, `C` et `D`.

La commande `zp2ss` permet de passer de la forme Zéros-Pôles vers la forme Espace d'Etat. Considérons l'exemple précédent.

```
>> [A,B,C,D]=zp2ss([],[-2 -3],[1])
A =
   -5.0000   -2.4495
    2.4495         0
B =
     1
     0
C =
         0    0.4082
D =
     0
```

Nous définissons le système par la commande `ss` pour laquelle nous transmettons ces matrices d'état.

```
>> sys=ss(A,B,C,D)

a =
           x1      x2
   x1      -5  -2.449
   x2   2.449       0

b =
       u1
   x1   1
   x2   0

c =
           x1      x2
   y1       0  0.4082

d =
       u1
   y1   0

Continuous-time model.
```

Nous retrouvons les 2 pôles par ses valeurs propres:

```
>> eig(sys)
ans =
    -3
    -2
```

L'opération de passage du modèle d'état à la méthode Zéros-Pôles se fait par `ss2zp`.

Nous pouvons passer de la fonction de transfert au modèle d'état par `tf2ss` et `ss2tf` respectivement.

Pour passer au modèle d'état discret, échantillonné à la cadence `T=0.1s`, nous utilisons la commande `c2dt`.

```
>> [Ad,Bd,Cd,Dd] = c2dt(A,B,C,0.1,0)
Ad =
    0.5850   -0.1908
    0.1908    0.9746

Bd =
    0.0779
    0.0104

Cd =
         0    0.4082

Dd =
     0
```

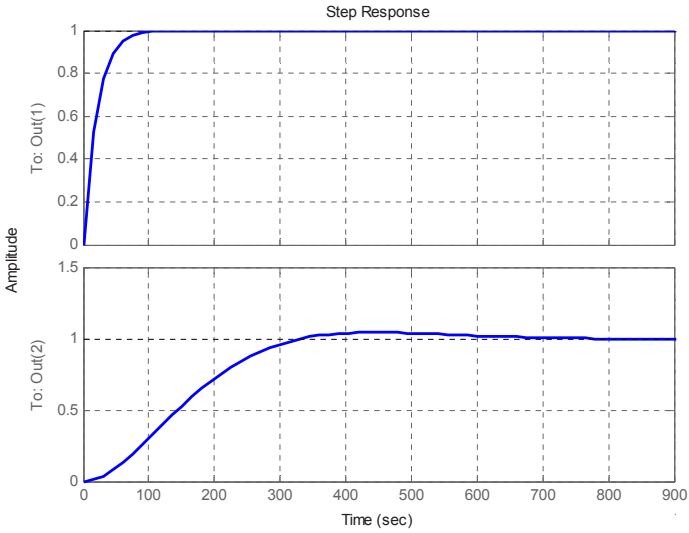
La commande `tf` peut aussi définir des systèmes multivariables. Pour un système à `m` sorties et `n` entrées, les polynômes `num` et `den` sont des tableaux de cellules de dimensions `m x n` où `num(i,j)` et `den(i,j)` spécifient la fonction de transfert liant l'entrée `j` à la sortie `i`.

Pour un système à 2 sorties et 1 entrée, un tel système peut être défini par sa fonction de transfert comme suit :

```
>> sys = tf( {1 ; 1 } , {[20 1] ; [10000 141.4 1]})

Transfer function from input to output...
          1
 #1:  --------
      20 s + 1

                  1
 #2:  ----------------------
      10000 s^2 + 141.4 s + 1
```

Nous trouvons les 2 réponses indicielles suivantes par la même commande `step`.

```
>> step(sys)
```



V.2.4. Les objets LTI et leurs propriétés

Un système LTI créé par `tf`, `zpk`, … est un objet qui possède des propriétés. L'implémentation des objets obéit à la programmation orienté Objets de MATLAB. Les objets sont des structures (Voir Chapitre Tableaux multidimensionnels) avec un flag qui indique leur classe (`tf`, `zpk`, `ss`, `frd`) et des champs appelés propriétés des objets.

Pour les objets LTI, ces propriétés comprennent :

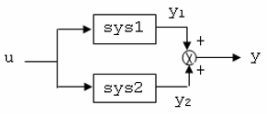
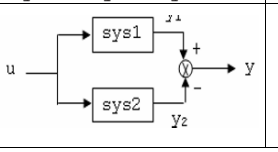
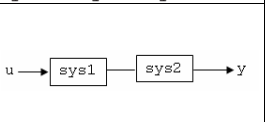
- les données du modèle,
- la cadence d'échantillonnage,
- les retards,
- les noms des entrées et des sorties

Les fonctions opérant sur un objet particulier sont appelées « `méthodes` ». Certaines opérations peuvent être effectuées sur des LTI, comme l'addition, multiplication ou concaténation. Ces opérations sont dites « `surchargées` » en ce sens qu'elles s'appliquent au LTI avec la même syntaxe que pour les matrices, mais adaptées aux objets LTI. Les objets LTI obtenus après ces opérations sont héritées des objets LTI utilisés.

Selon la classe des objets LTI, ces derniers possèdent une priorité, ou obéissent aux règles de précédence.
On peut ajouter deux objets LTI de classes différentes mais la classe de l'objet LTI résultant sera de la classe ayant plus de priorité.
Si `sys1` est obtenu par `tf`, `sys2` obtenu par `ss`, alors l'objet : `sys1+sys2` sera de la classe `ss`.

Pour éviter ces règles de précédence, on peut forcer la classe de l'objet résultant, comme par `sys = sys1 + tf(sys2)`, ou `sys = tf(sys1 + sys2)`.

Le tableau suivant résume les résultats des opérations arithmétiques réalisées sur les objets LTI.

`sys = sys1+sys2`	`sys = sys1-sys2`	`sys = sys1*sys2`
		

Les objets LTI possèdent tous, des propriétés génériques mais à chaque classe correspondent des propriétés spécifiques.

- **Propriétés génériques :**

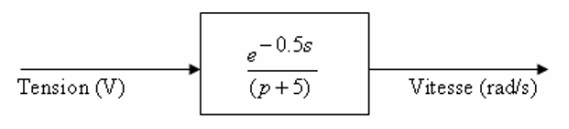
InputDelay	Retards sur les entrées	Vecteur
InputGroup	Groupes des entrées	Structure
InputName	Noms des entrées	Cellule de chaînes
Notes	Notes sur le modèle	Texte
OutputDelay	Retards sur les sorties	Vecteur
OutputGroup	Groupes des sorties	Structure
OutputName	Noms des sorties	Cellule de chaînes
Ts	Cadence d'échantillonnage	Scalaire
UserData	Données additionnelles	Type arbitraire

La période d'échantillonnage `Ts` maintient la valeur de la période des systèmes discrets. Par convention, `Ts` est nulle pour des systèmes continus. `Ts` vaut -1 pour des systèmes dont on ne spécifie pas la période d'échantillonnage.

`InputDelay`, `OutputDelay`, permettent de spécifier des retards purs sur les entrées ou les sorties.

Les propriétés `InputName` et `OutputName` permettent de donner un nom à chacune des entrées et sorties. Par défaut, si on ne spécifie pas de nom, ces propriétés sont des cellules vides.

Considérons l'exemple suivant :




```
>> sys = tf(1,[1 5],'Inputdelay',0.5);
>> set(sys,'inputname','Tension (V)','outputname', ...
'Vitesse (rad/s)','notes','modèle d''un moteur CC')
```

En invoquant la variable `sys`, nous retrouvons toutes les propriétés de l'objet.

```
>> sys
Transfer function from input "Tension (V)" to output "Vitesse
(rad/s)":
                1
exp(-0.5*s) * -----
              s + 5
```

Le type de ces propriétés sont rappelées grâce à la commande `set(sys)`.

```
>> set(sys)
num: Ny-by-Nu cell array of row vectors (Nu = no. of inputs)
den: Ny-by-Nu cell array of row vectors (Ny = no. of outputs)
       ioDelay: Ny-by-Nu array of delays for each I/O pair
      Variable: [ 's' | 'p' | 'z' | 'z^-1' | 'q' ]
            Ts: Scalar (sample time in seconds)
    InputDelay: Nu-by-1 vector
   OutputDelay: Ny-by-1 vector
     InputName: Nu-by-1 cell array of strings
    OutputName: Ny-by-1 cell array of strings
    InputGroup: structure with one field per channel group.
   OutputGroup: structure with one field per channel group.
          Name: String
         Notes: Text
      UserData: Arbitrary
```

Pour retrouver la valeur d'une propriété, on utilise la commande `get` (Voir Chapitre `handle Graphics`) en spécifiant la propriété que l'on recherche.

Ci-dessous, on recherche le nom du signal d'entrée et les notes sur le modèle.

```
>> get(sys,'InputName')
ans =
    'Tension (V)'
```

```
>> get(sys,'Notes')
ans =
    'modèle d'un moteur CC'
```

La commande `get(sys)` permet d'avoir toutes les valeurs spécifiées des propriétés de l'objet.

```
>> get(sys)
           num: {[0 1]}
           den: {[1 5]}
       ioDelay: 0
      Variable: 's'
            Ts: 0
```

```
    InputDelay: 0.5
   OutputDelay: 0
     InputName: {'Tension (V)'}
    OutputName: {'Vitesse (rad/s)'}
    InputGroup: [1x1 struct]
   OutputGroup: [1x1 struct]
          Name: ''
         Notes: {'modèle d'un moteur CC'}
      UserData: []
```

- **Propriétés spécifiques pour zpk :**

z	Zéros	Tableau de cellules en vecteurs colonnes.
p	Pôles	Tableau de cellules en vecteurs colonnes.
k	Gain	Matrice réelle
variable	Variable de la function de transfert :	Texte
ioDelay	Retards sur les entrées et les sorties	Matrice

Pour voir comment sont implémentées ces propriétés pour le même exemple précédent, nous forçons sa classe à celle du `zpk`. Le nouvel objet sera appelé `sys_zpk`.

```
>> sys_zpk=zpk(sys)

Zero/pole/gain from input "Tension (V)" to output "Vitesse
(rad/s)":
                1
exp(-0.5*s) * -----
              (s+5)
```

Pour avoir le pôle, par exemple, on utilise la même commande `get`.

```
>> get(sys_zpk,'p')
ans =

    [-5]
```

Les différentes propriétés sont définies comme suit :

```
>> get(sys_zpk)
               z: {[0x1 double]}
               p: {-5}
               k: 1
         ioDelay: 0
   DisplayFormat: 'roots'
        Variable: 's'
              Ts: 0
```

```
      InputDelay: 0.5
     OutputDelay: 0
       InputName: {'Tension (V)'}
      OutputName: {'Vitesse (rad/s)'}
      InputGroup: [1x1 struct]
     OutputGroup: [1x1 struct]
            Name: ''
           Notes: {'modèle d'un moteur CC'}
        UserData: []
```

- **Propriétés spécifiques pour ss :**

:

a	Matrice d'état A	Matrice 2D
b	Matrice de commande B	Matrice 2D
c	Matrice d'observation C	Matrice 2D
d	Matrice de passage direct D	Matrice 2D
e	Descripteur de la moyenne de l'état.	Matrice 2D
InternalDelay	Retards internes	Vecteur
StateName	Noms des états	Vecteur de cellules
Scaled	Etat mis à l'échelle pour une meilleure précision ?	0 pour faux, 1 pour vrai

On fait de même que précédemment en forçant la classe `ss`.

```
>> sys_ss=ss(sys)
a =
       x1
   x1  -5

b =
       Tension (V)
   x1            1

c =
                 x1
   Vitesse (rad   1

d =
                 Tension (V)
   Vitesse (rad            0

Input delays (listed by channel): 0.5

Continuous-time model.

>> get(sys_ss)

               a: -5
```

```
             b: 1
             c: 1
             d: 0
             e: []
        Scaled: 0
     StateName: {''}
  InternalDelay: [0x1 double]
            Ts: 0
    InputDelay: 0.5
   OutputDelay: 0
     InputName: {'Tension (V)'}
    OutputName: {'Vitesse (rad/s)'}
    InputGroup: [1x1 struct]
   OutputGroup: [1x1 struct]
          Name: ''
         Notes: {'modèle d'un moteur CC'}
      UserData: []
```

Dans le cas de ce modèle du 1er ordre, les matrices d'état sont des scalaires.

- **Propriétés spécifiques pour frd**

Frequency	Données en fréquences	Vecteur réel
ResponseData	Réponse en fréquences	Tableau multidimensionnel de valeurs complexes.
Units	Unités de fréquence	Chaîne : rad/s ou Hz

Le mode `frd` concerne la réponse en fréquences du système (uniquement des données en fréquences).

Considérons un simple système du 1er ordre de fonction de transfert :

$$H(p)=\frac{1}{p+5}$$

Nous cherchons la réponse en fréquences de ce système dans la bande de fréquences de 0 à 100 Hz.

```
>> sys=tf(1,[1 5]);
>> sys = frd(sys,0:100,'Units','Hz')
```

Les 10 premières réponses en fréquences sont:

```
  Frequency(Hz)          Response
  -------------          --------
         0          2.000e-001 + 0.0000i
         1          7.755e-002 - 0.0974i
         2          2.734e-002 - 0.0687i
         3          1.315e-002 - 0.0496i
         4          7.614e-003 - 0.0383i
         5          4.941e-003 - 0.0310i
         6          3.457e-003 - 0.0261i
         7          2.552e-003 - 0.0224i
```

```
         8          1.960e-003 - 0.0197i
         9          1.551e-003 - 0.0175i
        10          1.259e-003 - 0.0158i
```

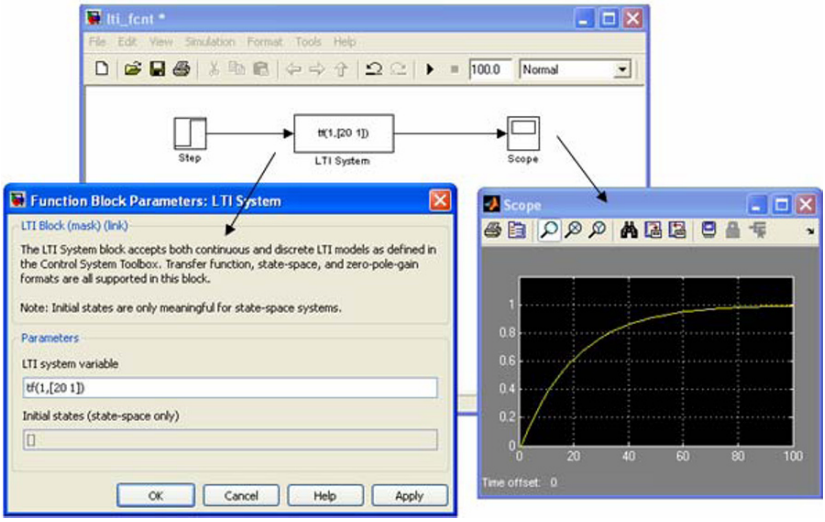
V.2.5. Les systèmes LTI dans SIMULINK

Dans la partie SIMULINK de la boite à outils «`Control System Toolbox`», nous pouvons utiliser le bloc `LTI System` pour définir un modèle LTI soit sous forme de fonction de transfert soit sous ses autres formes comme les équations d'état.

Dans le cas suivant, nous le définissons sous la forme de fonction de transfert dont nous spécifions le numérateur et le dénominateur.

Soit le modèle suivant du 1er ordre de constante de temps 20 s et de gain statique unité :

$$H(p)=\frac{1}{20p+1}$$



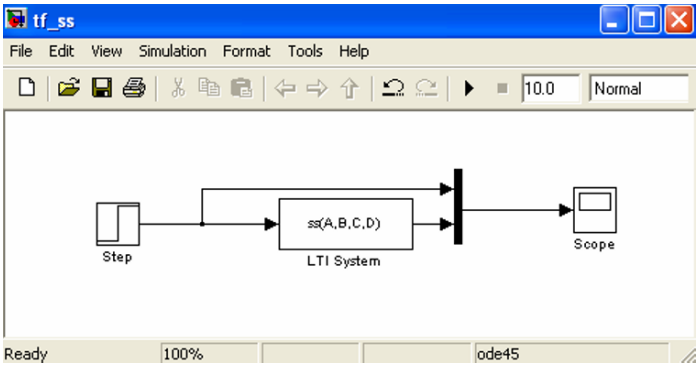
Dans le champ `LTI system variable`, nous pouvons utiliser les commandes vues précédemment.

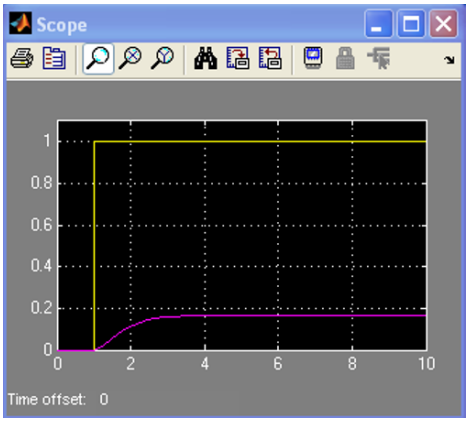
Nous définissons les matrices d'état d'un système défini par la méthode zéros-pôles-gain.

```
>> [A,B,C,D]=zp2ss([],[-2 -3],[1])
A =
   -5.0000   -2.4495
    2.4495         0
```

```
B =
     1
     0

C =
          0     0.4082

D =
     0
```

Nous utilisons la commande `ss(A,B,C,D)` dans le champ `LTI system variable` du bloc, comme on peut le voir en double-cliquant sur ce bloc.





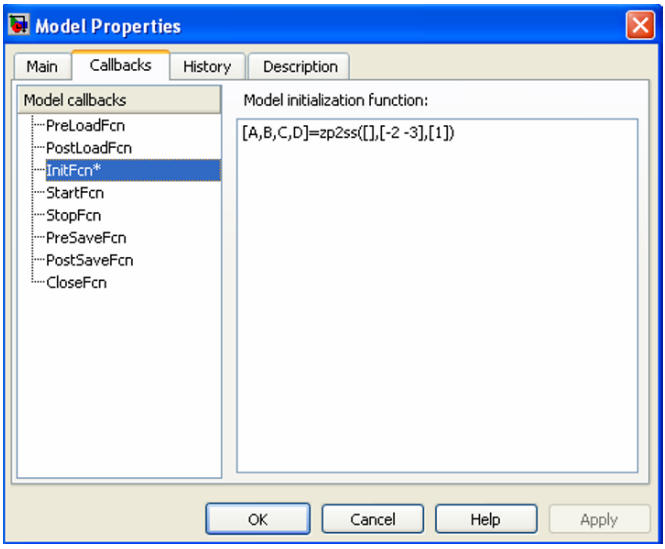
Les matrices `A`, `B`, C, `D` peuvent être entrées directement dans un callback de la fenêtre du modèle SIMULINK (Voir chapitre `Callbacks`).

On utilisera le callback `InitFcn` qui est appelé à chaque initialisation du modèle SIMULINK, `tf_ss.mdl`.

Les fonctions MATLAB, exécutées sont celles où l'on spécifie ces matrices.

On choisit l'option `Model Properties` du menu `File`.

La commande exécutée sera alors le calcul des matrices d'état par `zp2ss`.



Nous obtenons les mêmes courbes du signal d'entrée et de sortie sur l'oscilloscope.

V.2.6. LTI viewer

Le `LTI viewer` est un outil qui permet de spécifier les caractéristiques des objets LTI.

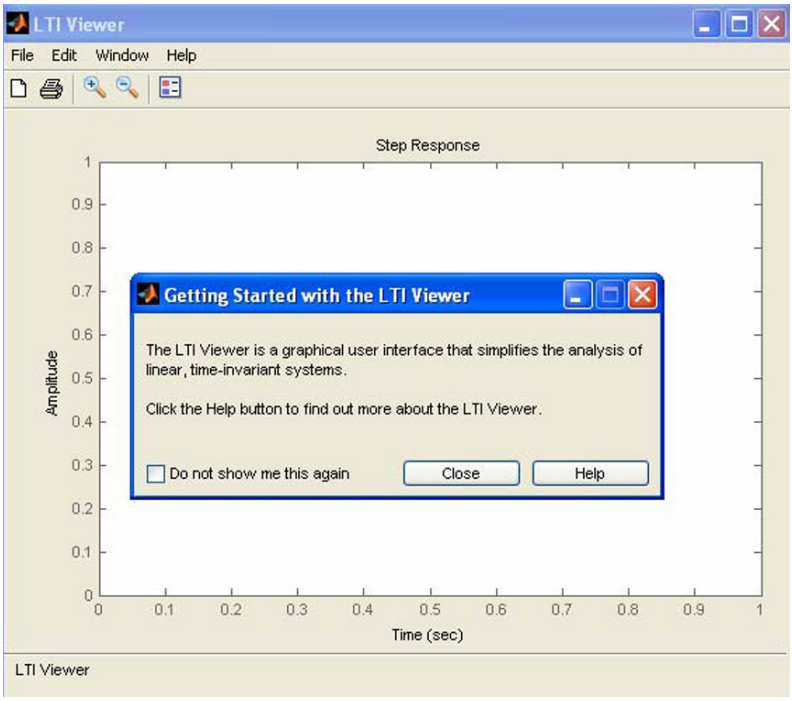
C'est un interface graphique (`GUI : graphical user interface`) qui simplifie l'analyse d'un système LTI.

Il permet de comparer plusieurs réponses en même temps, étudier l'effet de paramètres sur la réponse d'un système, étudier la stabilité, les marges de gain et de phase, le temps de réponse, etc.

Pour ouvrir cet éditeur, nous pouvons utiliser la commande suivante :

```
>> ltiview
```

Nous obtenons l'interface graphique suivant :



Pour ouvrir `LTI viewer` afin d'étudier un exemple de LTI, nous pouvons le spécifier entre parentheses.

MATLAB permet la sauvegarde de systèmes LTI, sous forme de fichiers binaires qu'on ouvre grâce à la commande `load`, comme par exemple `sys_dc`.

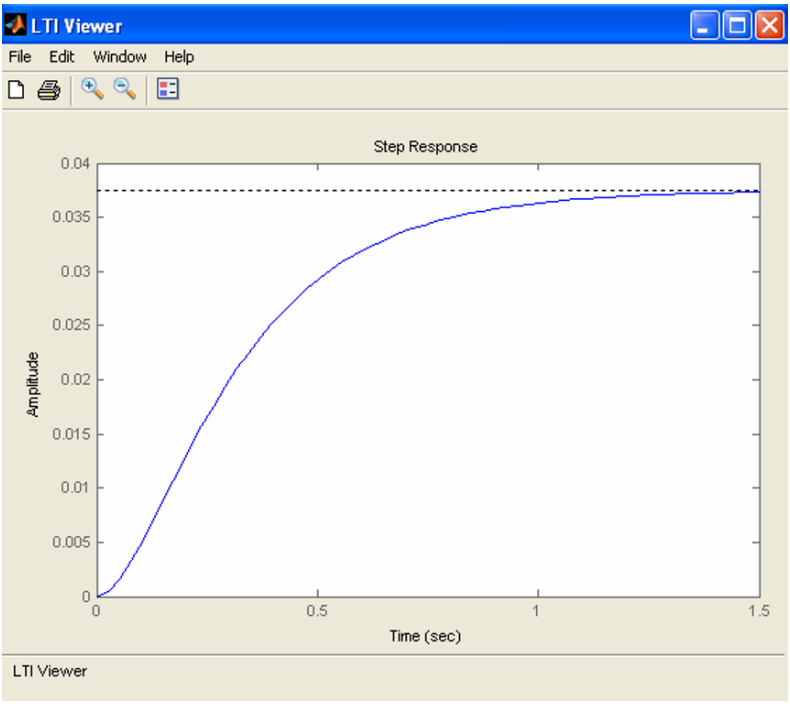
```
>> load ltiexamples
>> whos
  Name            Size              Bytes  Class     Attributes

  G               1x1                2526  tf
  Gcl1            1x1                2526  tf
  Gcl2            1x1                2526  tf
  Gcl3            1x1                2526  tf
  Gservo          1x1                2812  zpk
  clssF8          2x2                2993  ss
  diskdrive       1x1                2860  zpk
  frdF8           2x2                2806  frd
  frdG            1x1                2330  frd
  freq            5x1                  40  double
  gasf            4x6               11429  ss
```

```
  hplant          1x1              10589  ss
  m2d             4-D               6850  tf
  respF8          2x2x5              320  double    complex
  respG           5x1                 80  double    complex
  ssF8            2x2               2993  ss
  sys_dc          1x1               2385  ss
```

Nous avons différents systèmes LTI de différentes classes, `ss`, `tf`, `zpk` ou `frd`.

Pour étudier ce système dans `LTI viewer`, nous le spécifions entre parenthèses.

```
>> ltiview(sys_dc)
```

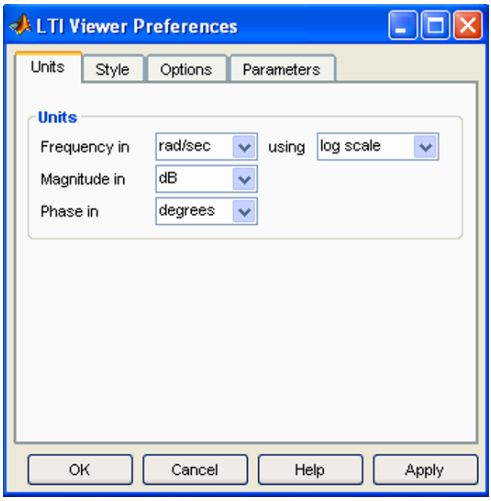


Le `Viewer Preferences` qui permet de spécifier les préférences d'affichage des courbes, textes, mode d'affichage des axes (linéaire ou logarithmique), etc. peut être ouvert par le menu `Edit`, en choisissant `Viewer Preferences`.

On peut spécifier les unités de fréquence, le style du texte, etc.

Le `Viewer Preferences` peut être aussi ouvert par la commande suivante :

```
>> ctrlpref
```



Par défaut, les fréquences sont spécifiées en rad/s, les amplitudes en dB et les phases en degrés.

Comme les amplitudes sont en dB, l'échelle est alors logarithmique.