TP Euler

1 Exercice : *Asservissement d'un chariot*

Considérons le contrôle en vitesse du mouvement d'un chariot de technologie analogue à ceux situés sur les pistes d'athlétisme afin de suivre et filmer les coureurs.

Ce chariot est alimenté par une tension électrique. La grandeur de sortie est la vitesse de translation.

1.1 En boucle ouverte

Le comportement temporel du chariot soumis à une tension de commande est modélisé par une équation différentielle du premier ordre :

$$\tau_c \frac{dv(t)}{dt} + v(t) = K_c u_c(t)$$

L'objectif est d'obtenir la réponse temporelle du système à une entrée échelon (entrée constante) $u_c(t) = U_{mot}$.

1. Écrire une fonction `liste_temps(pas,tmax)` renvoyant une liste des abscisses en temps à partir du `pas` (intervalle entre deux abscisses) et de `tmax` (borne supérieure des temps).
2. Écrire une fonction `ordre1_euler(K,tau,u,temps)` renvoyant une liste d'ordonnées correspondant à la résolution par la méthode d'euler explicite de l'équation différentielle soumise à une entrée constante `u` et pour une liste d'abscisses `temps` fournie.

   La solution analytique de l'équation différentielle à une entrée constante s'écrit :

   $$v(t) = K_c U_{mot} \left(1 - exp\left(-t/\tau_c\right)\right)$$

3. Écrire une fonction `ordre1_th(K,tau,u,temps)` renvoyant une liste d'ordonnées correspondant à la solution analytique précédente pour une liste d'abscisses `temps` fournie.

   Nous allons tâcher de superposer le tracé de la réponse analytique avec les tracés approchés par la méthode d'Euler pour différentes valeurs de pas.

   Les valeurs numériques des paramètres sont : $K_c = 0.5 m.s^{-1}.V^{-1}$, $\tau_c = 1s$, $U_{mot} = 5V$ et $t_{max} = 5s$
4. Tester ces lignes afin de prendre en main les fonctionnalités de tracés

```
#Importation des modules
import matplotlib.pyplot as plt #module de gestion des traces
##########################################

Les fonctions etant definies

##########################################
#Les elements de base
##########################################
x=liste_temps(tmax,0.1)          #une liste d'abscisse
z=ordre1_th(K_c,tau_c,U_mot,x) #une liste d'ordonnee de meme taille que celle des abscisses
plt.plot(x,z,'-')  #Stocke en memoire le trace avec ses options
plt.show()  #pour afficher les traces en memoire

##############################
#Pour enrichir le trace
```

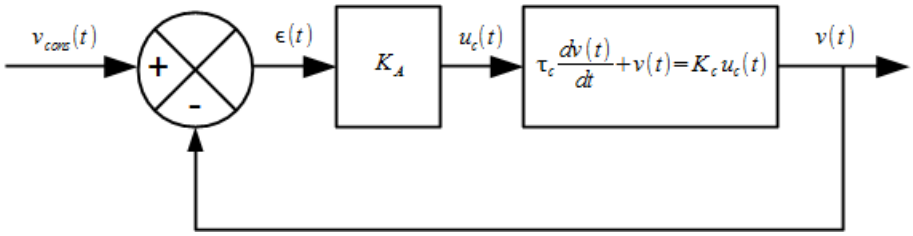
```
################################
marqueurs = ['^', '+', '.', 'x', '*', 'o'] #Les marqueurs
couleurs = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'] #Les couleurs
style = ['-', '--', '-.', ':'] #Les styles de trait

plt.plot(x,z,'--',marker='^',color='r',linewidth=2,label='analytique')
plt.title('Reponse temporelle') #titre du trace
plt.legend(loc='lower right') #positionne les \'etiquettes
plt.xlabel('temps') #legende de l'axe des abscisses
plt.ylabel('vitesse') #legende de l'axe des ordonnees
plt.show()  #pour afficher les traces en memoire
```

5. Tracer sur un même graphe la solution analytique ainsi que la solution approchée par la méthode d'Euler pour des pas de [0.3,0.5,0.7,1].

1.2 En boucle fermée

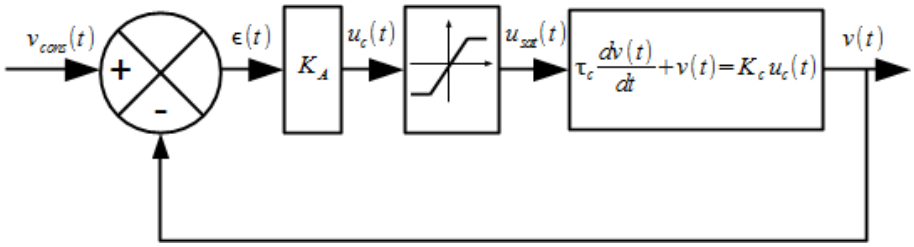
Afin de contrôler le chariot, le pilotage s'effectue en boucle fermée. Une consigne de vitesse $V_{cons}(t)$ est imposée. Cette consigne est comparée à la vitesse du chariot $v(t)$. L'écart générée est adaptée par un gain réglable K_A, soit $u_c(t) = K_A\epsilon(t)$.



1. Écrire une fonction `ordre1_BF(ka,k,tau,vc,temps)` renvoyant une liste d'ordonnées correspondant à la résolution par la méthode d'Euler explicite du système en boucle fermée
2. Tracer la réponse du système pour une vitesse de consigne constante $V_c = 6m.s^{-1}$ et un gain d'adaptation $K_A = 20V.s.m^{-1}$

1.3 Avec prise en compte de la saturation

Le choix d'un gain K_A élevé permet de gagner en précision et rapidité. Toutefois, la tension d'alimentation du chariot est écrêtée à +12V ou -12V. Il faut donc en tenir compte dans notre modélisation.



1. Écrire une fonction `ordre1_BFsat(ka,k,tau,vc,umax,temps)` correspondant à la résolution par la méthode d'Euler explicite du système en boucle fermée avec prise en compte de la saturation. La vitesse du chariot ainsi que la tension d'alimentation du chariot seront récupérées en sortie.
2. Tracer les réponses superposées avec et sans saturation, ainsi que la tension d'alimentation du chariot pour une vitesse de consigne constante $V_c = 6m.s^{-1}$ et un gain d'adaptation $K_A = 20V.s.m^{-1}$

2 Exercice : *Utilisation des méthodes Python*

Il est possible d'utiliser des modules Python pour intégrer des équations différentielles.
Voir : `http://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html`
On donne ici un exemple avec l'équation différentielle $y'(t) = 2y(t) - (t+1)\sqrt{y(t)}$ de solution générale $y(t) = (\frac{1}{2}t + Ce^{-t})^2$.
On commence par importer les modules nécessaires. On crée les abscisses dans le tableau x, dans le tableau y on met la solution donnée par `integrate.odeint` de paramètres la fonction f telle que $y' = f(y,t)$, la condition initiale y_0 et la liste des abscisses.
Pour comparer, on a mis la vraie solution dans la liste *yvraie* et on a tracé les deux graphes.

```
import scipy.integrate
import math
import matplotlib.pyplot as plt
def f(y,t):
    return(-2*y+(t+1)*math.sqrt(y))

n=50
a=0
b=5
x=[a+k*(b-a)/n for k in range(n)]
y=scipy.integrate.odeint(f,1,x)
yvraie=[(0.5*t+math.exp(-t))**2 for t in x]

plt.plot(x,y,'*',color='red')
plt.plot(x,yvraie)
plt.show()
```

Reprendre ce code pour résoudre $y'(t) + e^{t-y(t)} = 0$. La solution générale est $y(t) = \ln(C - e^t)$ pour tout t tel que $C - e^t > 0$. On va prendre $C = 4, a = 0, b = 1$ et on essaiera pour plusieurs valeurs de n.

3 Exercice : *Résolution d'une équation différentielle du second degré*

Considérons l'équation différentielle à coefficients constants du second ordre suivante:

$$\frac{1}{{\omega_0}^2}\frac{d^2x(t)}{dt^2} + \frac{2\xi}{\omega_0}\frac{dx(t)}{dt} + x(t) = Ku(t)$$

Ce type de modélisation est largement rencontré que ce soit en électrocinétique sur un circuit RLC par exemple où en mécanique à l'issue de l'application des lois de Newton.

L'objectif va être de tracer la réponse temporelle à une entrée constante $u(t) = U_0$ en utilisant la méthode d'Euler.
En fait, il s'agit de construire un système de deux équations couplées :

$$\begin{cases} \frac{dx(t)}{dt} = v(t) \\ \frac{1}{{\omega_0}^2}\frac{dv(t)}{dt} + \frac{2\xi}{\omega_0}v(t) + x(t) = Ku(t) \end{cases}$$

Soit :

$$\begin{cases} \frac{dx(t)}{dt} = f1(v,x,t) \\ \frac{dv(t)}{dt} = f2(v,x,t) \end{cases}$$

Les conditions initiales sont alors le couple vitesse, position (v_0, x_0).

1. Préciser les expressions des fonctions $f1(v,x,t)$ et $f2(v,x,t)$.
2. Écrire une fonction `ordre2_euler(K,w0,xi,u0,v0,x0,temps)` renvoyant une liste d'ordonnées pour la position et une autre pour la vitesse à partir d'une liste d'abscisses `temps` fournie.

   On pose $K = 10$, $U_0 = 5V$, $\omega_0 = 5rad/s$,
3. Proposer une suite d'instructions afin de tracer la réponse de cette équation différentielle avec des conditions initiales nulles et pour des valeurs de facteur d'amortissement ($\xi = \frac{1}{2Q}$ avec Q le facteur de qualité) $xi = [0.3, 0.69, 1, 2, 5]$

Nous allons essayer d'utiliser la méthode de résolution implantée dans Python et mis en évidence précédemment (`integrate.odeint`).

Cette méthode nécessite de présenter l'équation sous la forme : $\frac{dz}{dt} = f(z,t)$

Pour y arriver, nous allons considérer que Z est un tuple constitué des variables $\frac{dx(t)}{dt} = v(t)$ la vitesse et x la position. Soit $z = (\frac{dx(t)}{dt}, x(t)) = (v(t), x(t))$

Les conditions initiales sont alors $z_0 = (v_0, x_0)$

4. Tester les lignes suivantes correspondant à la résolution de l'équation différentielle d'ordre 2 pour un $\xi = 0.3$

```
####################################
###Resolution avec les methodes Python

import scipy.integrate
import math
import matplotlib.pyplot as plt
import numpy

def f(z,t):
    v,p=z
    return((K*u-p-2*xi*v/w_0)*((w_0)**2),v)


K=1
w_0=5
xi=0.3
u=5

x=liste_temps(10,0.01)       #Liste abscisses
z0 = [0,0]                    #conditions initiales
z = scipy.integrate.odeint (f, z0, x) #resolution python

y=z[:,1] #Pour avoir la deuxieme colonne (position, environnement numpy)


plt.plot(x,y,'--',color='r',marker='+')
plt.show()
```

```
##############################################
#####Asservissement d'un chariot
#Importation des modules
import matplotlib.pyplot as plt
import math


#Definition des parametres
K_c=0.5
tau_c=1
U_mot=5
pas = 0.5
tmax=6

#Definition d'une fonction abscisse des temps
def liste_temps(tmax,pas):
    '''Renvoie une liste des abscisses en temps
    a partir du pas et de la borne superieure des temps'''
    t=0
    temps=[0]
    while t<=(tmax-pas):
        t=t+pas
        temps=temps+[t]
    return (temps)

#Definition d'une fonction premier ordre
def ordre1_euler(k,tau,u,temps):
    ''' Renvoie une liste des ordonnees pour un premier ordre
    soumis a un echelon u de tension
    pour une liste abscisse des temps fournie'''
    s=0
    sortie=[0]
    for i in range(1,len(temps)):
        f=(k*u-s)/tau
        s=s+f*(temps[i]-temps[i-1])
        sortie=sortie + [s]
    return sortie

#reponse analytique d'un premier ordre
def ordre1_th(k,tau,u,temps):
    s=[0]*(len(temps))
    for i in range(len(temps)):
        s[i]=k*u*(1-math.exp(-temps[i]/tau))
    return s



#Les traces

#Trace reponse theorique
x=liste_temps(tmax,0.1)
z=ordre1_th(K_c,tau_c,U_mot,x)
plt.plot(x,z,'-')


#Trace superpose pour differents pas de temps
```

```
marqueurs = ['^', '+', '.', 'x', '*', 'o'] #Les marqueurs
couleurs = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'] #Les couleurs
style = ['-', '--', '-.', ':']
k=0
for i in (0.3,0.5,0.7,1):
    x=liste_temps(tmax,i)
    y=ordre1_euler(K_c,tau_c,U_mot,x)
    plt.plot(x,y,'--',color=couleurs[k],marker=marqueurs[k])
    k=k+1


#Affichage superpose des courbes
plt.show()



#Reponse en boucle fermee

#Premier ordre en boucle fermee sans prise en compte de la saturation
def ordre1_BF(ka,k,tau,vc,temps):
    ''' Renvoie une liste des ordonnees pour un premier ordre
    soumis a un echelon u de tension
    pour une liste abscisse des temps fournie'''
    s=0
    sortie=[0]
    for i in range(1,len(temps)):
        u=ka*(vc-s)
        f=(k*u-s)/tau
        s=s+f*(temps[i]-temps[i-1])
        sortie=sortie + [s]
    return sortie

#Premier ordre en boucle fermee avec prise en compte de la saturation
def ordre1_BFsat(ka,k,tau,vc,umax,temps):
    ''' Renvoie une liste des ordonnees pour un premier ordre
    soumis a un echelon u de tension
    pour une liste abscisse des temps fournie'''
    s=0
    sortie=[0]
    tension=[0]
    for i in range(1,len(temps)):
        u=ka*(vc-s)
        if u>umax:
            u=umax
        elif u<-umax:
            u=-umax
        f=(k*u-s)/tau
        s=s+f*(temps[i]-temps[i-1])
        sortie=sortie + [s]
        tension = tension +[u]
    return sortie, tension

#Les traces
#trace superpose pour differents pas de temps

x=liste_temps(5,0.001)
```

```
y=ordre1_BF(20,K_c,tau_c,6,x)
z , us =ordre1_BFsat(20,K_c,tau_c,6,12,x)
plt.plot(x,y,'--')
plt.plot(x,z,'--')
plt.plot(x,us,'--')
plt.show()



###################################
###Resolution avec les methodes Python

import scipy.integrate
import math
import matplotlib.pyplot as plt

def f(y,t):
    return(-math.exp(t-y))

n=50
a=0
b=1
C=4

x=[a+k*(b-a)/n for k in range(n)]

y=scipy.integrate.odeint(f,math.log(C-1),x)
yvraie=[math.log(C-math.exp(t)) for t in x]

plt.plot(x,y,'*',color='red')
plt.plot(x,yvraie)
plt.show()


################################################
#Rsolution d'une quation diffrentielle d'ordre 2

#Importation des modules
import matplotlib.pyplot as plt
import math


#Dfinition des paramtres
K=10
w_0=5
U_mot=5
pas = 0.5
tmax=5



#Dfinition d'une fonction deuxime ordre
#Equation du type
def ordre2_euler(w_0,xi,K,u,temps):
    ''' Renvoie une liste des ordonnes pour un premier ordre
    soumis  un chelon u de tension
```

```
    pour une liste abscisse des temps fournie'''
    p=0
    v=0
    position=[0]
    vitesse=[0]
    for i in range(1,len(temps)):
        f1=(K*u-p-2*xi*v/w_0)*((w_0)**2)
        f2=v
        v2=v+f1*(temps[i]-temps[i-1])
        p2=p+f2*(temps[i]-temps[i-1])
        position=position + [p2]
        vitesse=vitesse+[v2]
        v=v2
        p=p2
    return position, vitesse

#Les tracs

#Trac reponse
x=liste_temps(10,0.01)

#Trac superpos pour diffrents pas de temps
marqueurs = ['^', '+', '.', 'x', '*', 'o'] #Les marqueurs
couleurs = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'] #Les couleurs
style = ['-', '--', '-.', ':']
k=0
#Pour diffrentes valeurs de xi
for i in (0.3,0.69,1,2,5):
    y,z=ordre2_euler(w_0,i,K,U_mot,x)
    plt.plot(x,y,'--',color=couleurs[k],marker=marqueurs[k])
    k=k+1


#Affichage superpose des courbes
plt.show()


##############################################
###Resolution avec les methodes Python
#Utilisation de integrate.odeint

import scipy.integrate
import numpy

def f(z,t):
    v,p=z
    return((K*u-p-2*xi*v/w_0)*((w_0)**2),v)


xi=0.3

x=liste_temps(10,0.01)
z0 = [0,0]
z = scipy.integrate.odeint (f, z0, x)

y=z[:,1] #Pour avoir la deuxime colonne (position, environnement numpy)
```

```
plt.plot(x,y,'--',color='r',marker='+')
plt.show()
```