Intégrateur numérique

1. Introduction

Pour un signal à temps continu $x(t)$, l'intégration est définie par :

$$y(t) = \frac{1}{\tau} \int_0^t x(t')\, dt' \tag{1}$$

où τ est une constante homogène à un temps lorsque $y(t)$ a les mêmes dimensions que $x(t)$. Pour réaliser une intégration vraie, on posera $\tau = 1$.

La fonction de transfert en régime sinusoïdal de l'intégrateur est :

$$H(\omega) = \frac{1}{j\tau\omega} \tag{2}$$

En régime sinusoïdal, un intégrateur est donc caractérisé par un déphasage de $-\frac{\pi}{2}$ et par un gain en décibel décroissant à -20 dB par décade.

Ce document montre comment réaliser l'intégration d'un signal échantillonné x_n. La période d'échantillonnage est notée T_e.

2. Intégrateur numérique parfait

2.a. Conception du filtre

Soit y_n le signal numérique obtenu par échantillonnage de $y(t)$. L'équation différentielle (1) peut aussi s'écrire :

$$\frac{dy(t)}{dt} = \frac{1}{\tau} x(t) \tag{3}$$

En remplaçant la dérivée par une différence finie, on obtient :

$$y_n = y_{n-1} + \frac{T_e}{\tau} x_n \tag{4}$$

La relation précédente est de la forme suivante :

$$a_0 y_n = b_0 x_n + b_1 x_{n-1} - a_1 y_{n-1} \tag{5}$$

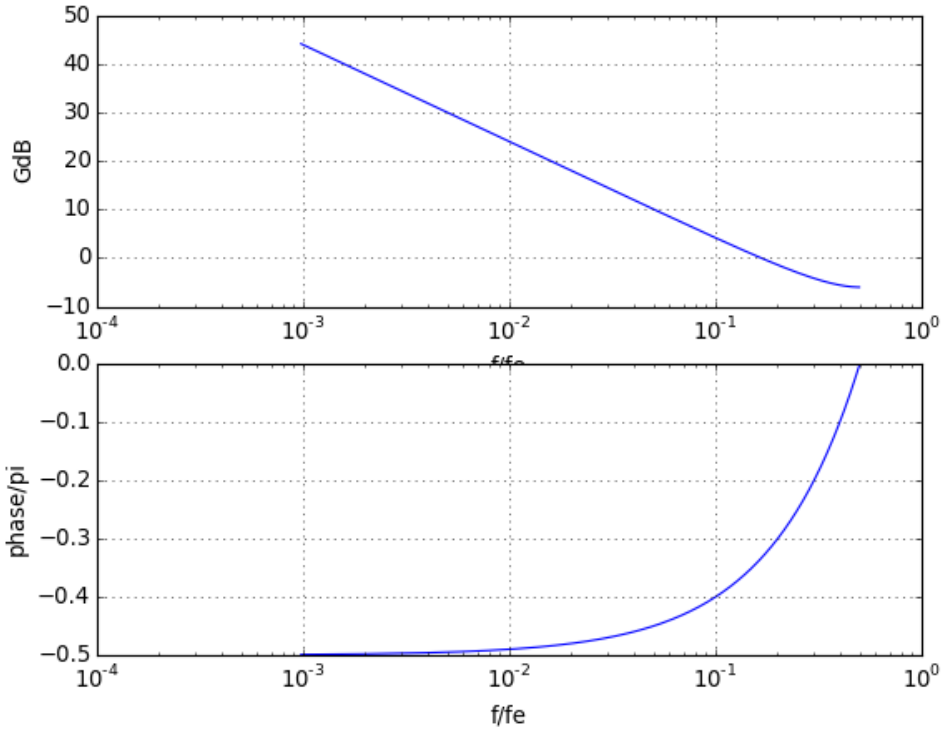
Cette relation de récurrence défini un filtre récursif (à réponse impulsionnelle infinie), dont la fonction de transfert en Z est :

$$H(z) = \frac{T_e}{\tau} \frac{1}{1 - z^{-1}} \tag{6}$$

Pour tracer sa réponse fréquentielle, on peut poser $T_e/\tau = 1$ puisque ce rapport n'a pas d'effet sur la forme du signal en sortie, seulement sur son amplitude :

```
import scipy.signal
import numpy
from matplotlib.pyplot import *
```

```
a=[1,-1]
b=[1]
w,h=scipy.signal.freqz(b,a)
figure()
subplot(211)
plot(w/(2*numpy.pi),20*numpy.log10(numpy.absolute(h)))
xlabel("f/fe")
ylabel("GdB")
xscale('log')
grid()
subplot(212)
plot(w/(2*numpy.pi),numpy.angle(h)/numpy.pi)
xlabel("f/fe")
ylabel("phase/pi")
xscale('log')
grid()
```

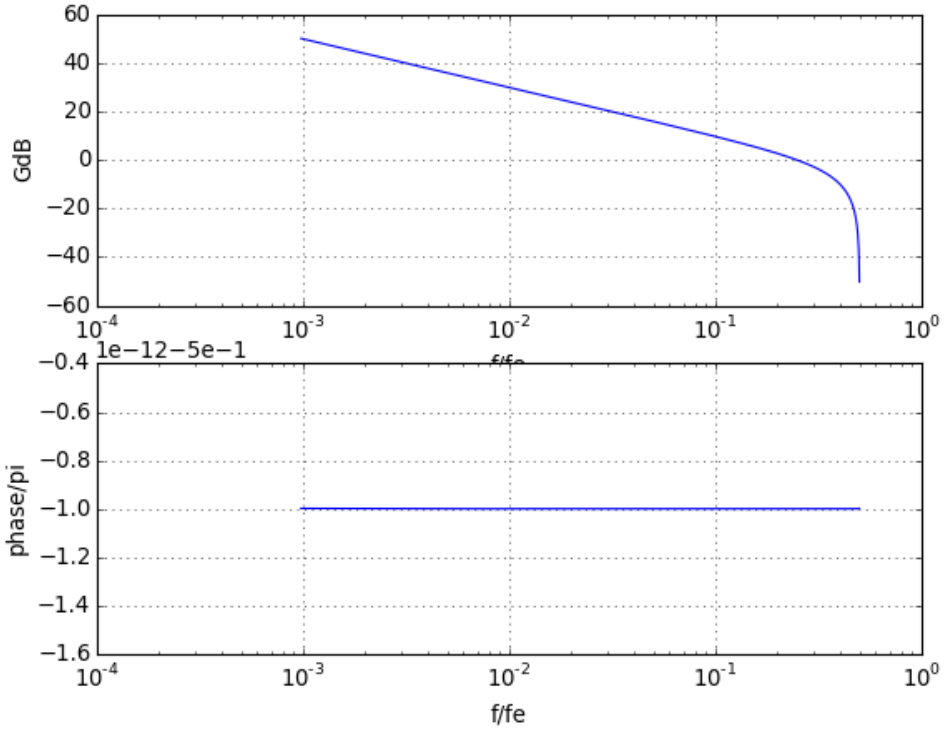


On n'obtient pas du tout le comportement intégrateur recherché, puisque le déphasage devrait être constamment égal à $-\pi/2$. La solution consiste à écrire un schéma numérique à deux pas (de type Adams-Moulton) :

$$y_n = y_{n-1} + \frac{T_e}{\tau}\left(x_n + x_{n-1}\right) \tag{7}$$

```
import scipy.signal
import numpy
from matplotlib.pyplot import *
```

```
a=[1,-1]
b=[1,1]
w,h=scipy.signal.freqz(b,a)
figure()
subplot(211)
plot(w/(2*numpy.pi),20*numpy.log10(numpy.absolute(h)))
xlabel("f/fe")
ylabel("GdB")
xscale('log')
grid()
subplot(212)
plot(w/(2*numpy.pi),numpy.angle(h)/numpy.pi)
xlabel("f/fe")
ylabel("phase/pi")
xscale('log')
grid()
```



On obtient ainsi un très bon intégrateur, sauf à proximité de la fréquence de Nyquist ($f_e/2$).

2.b. Réalisation du filtrage

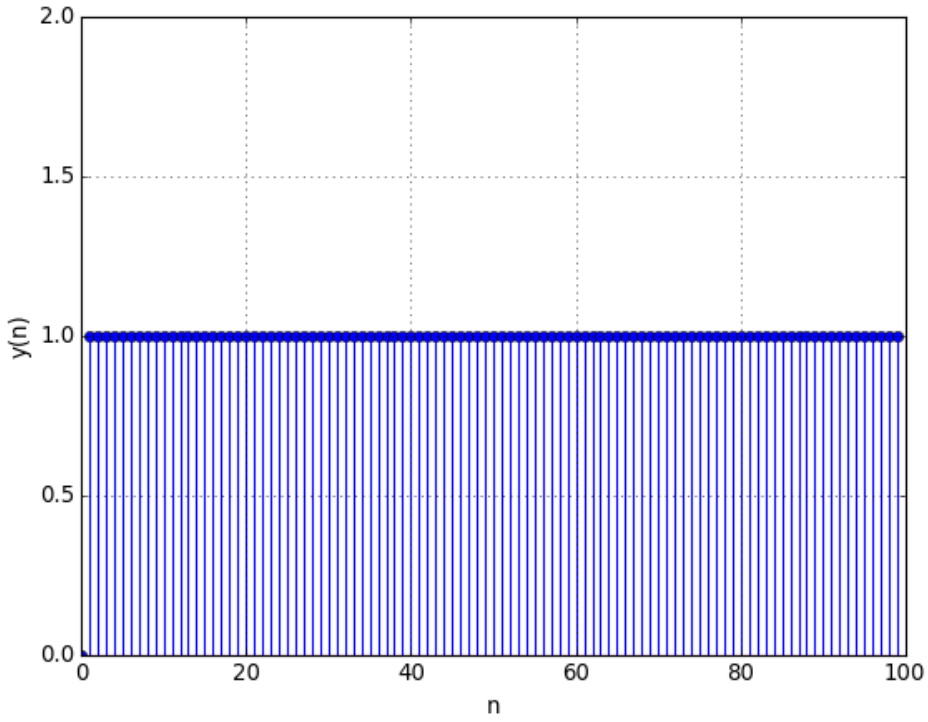
Pour réaliser le filtrage d'une liste d'échantillons, on remarque que le calcul de la sortie commence à y_1, et qu'il faut choisir une valeur pour y_0. On prendra $y_0 = 0$.

Voici une fonction réalisant le filtrage d'une liste d'échantillons stockés en mémoire, pour un filtre récursif avec un numérateur et un dénominateur de degré 1 :

```
def filtrage_recursif(x,a,b):
    N = len(x)
    y = numpy.zeros(N)
    for n in range(1,N):
        y[n] = (-a[1]*y[n-1]+b[0]*x[n]+b[1]*x[n-1])/a[0]
    return y
```

On applique le filtre précédent à une impulsion :

```
x = numpy.zeros(100)
x[0] = 1.0
y = filtrage_recursif(x,a,b)
figure()
stem(y)
xlabel("n")
ylabel("y(n)")
axis([0,100,0,2])
grid()
```



La réponse impulsionnelle d'un intégrateur est un échelon. Elle ne tend pas vers zéro lorsque n tend vers l'infini (elle est constante), ce qui montre que l'intégrateur est instable.

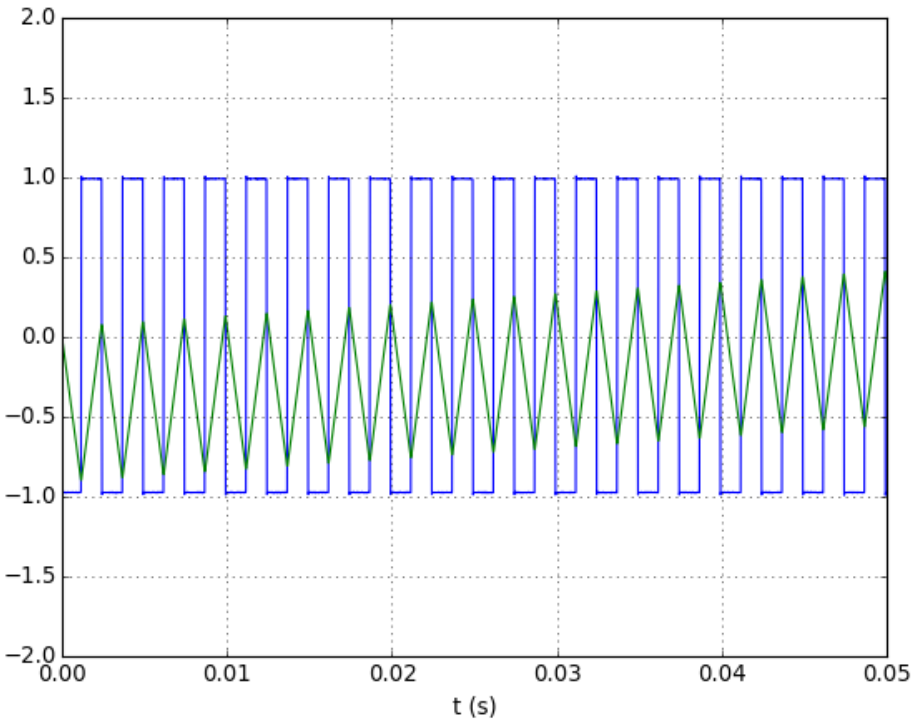
2.c. Test de l'intégrateur

Un bon moyen de tester un intégrateur est de lui fournir un signal créneau. Nous allons donc numériser un signal créneau délivré par un GBF avec la carte d'acquisition et le script suivant :

```
# -*- coding: utf-8 -*-
import pycan.main as pycan
import numpy
from matplotlib.pyplot import *
import scipy.signal
fe=40000.0
T=0.1
te=1.0/fe
N = int(T/te)
can = pycan.Sysam("SP5")
can.config_entrees([0],[10.0])
can.config_echantillon(te*10**6,N)
can.acquerir()
t0=can.temps()[0]
u0=can.entrees()[0]
can.fermer()
numpy.savetxt("signal-1.txt",[t0,u0])
```

On applique le filtre intégrateur. Le gain est réduit en choisissant une valeur plus faible de $b_0 = b_1$.

```
[t0,u0] = numpy.loadtxt("signal-1.txt")
a=[1,-1]
b=[0.01,0.01]
y = filtrage_recursif(u0,a,b)
figure()
plot(t0,u0)
plot(t0,y)
xlabel("t (s)")
axis([0,0.05,-2,2])
grid()
```



On observe une forte dérive, due à la présence d'une composante de fréquence nulle dans le signal x_n, bien que celle-ci soit très faible.

Lorsque le signal $x(t)$ comporte une composante de fréquence nulle que l'on souhaite intégrer, il faut utiliser cet intégrateur. Il faut cependant faire une compensation qui permette d'annuler tout décalage accidentel.

3. Filtre passe-bas du premier ordre

3.a. Filtre analogique

Pour stabiliser le filtre intégrateur afin de ne pas avoir de dérive en sortie, il faut ramener le gain à fréquence nulle à une valeur finie. Un moyen simple de le faire est d'utiliser un filtre passe-bas du premier ordre.

Considérons un filtre passe-bas dont la fonction de transfert est :

$$H(\omega) = \frac{g}{1 + j\tau\omega} \tag{8}$$

Ce filtre est quasiment intégrateur lorsque $\omega \gg 1/\tau$.

L'équation différentielle associée à ce filtre est :

$$y(t) + \tau \frac{dy(t)}{dt} = gx(t) \tag{9}$$

3.b. Filtre numérique

La discrétisation de l'équation différentielle conduit à la relation suivante :

$$y_n + \tau \frac{y_n - y_{n-1}}{T_e} = g \frac{x_n + x_{n-1}}{2} \tag{10}$$

On obtient ainsi un filtre récursif défini par :

$$y_n = \frac{1}{1+\frac{T_e}{\tau}} y_{n-1} + \frac{g}{2} \frac{\frac{T_e}{\tau}}{1+\frac{T_e}{\tau}} (x_n + x_{n-1}) \tag{11}$$

Sa fonction de transfert en Z est :

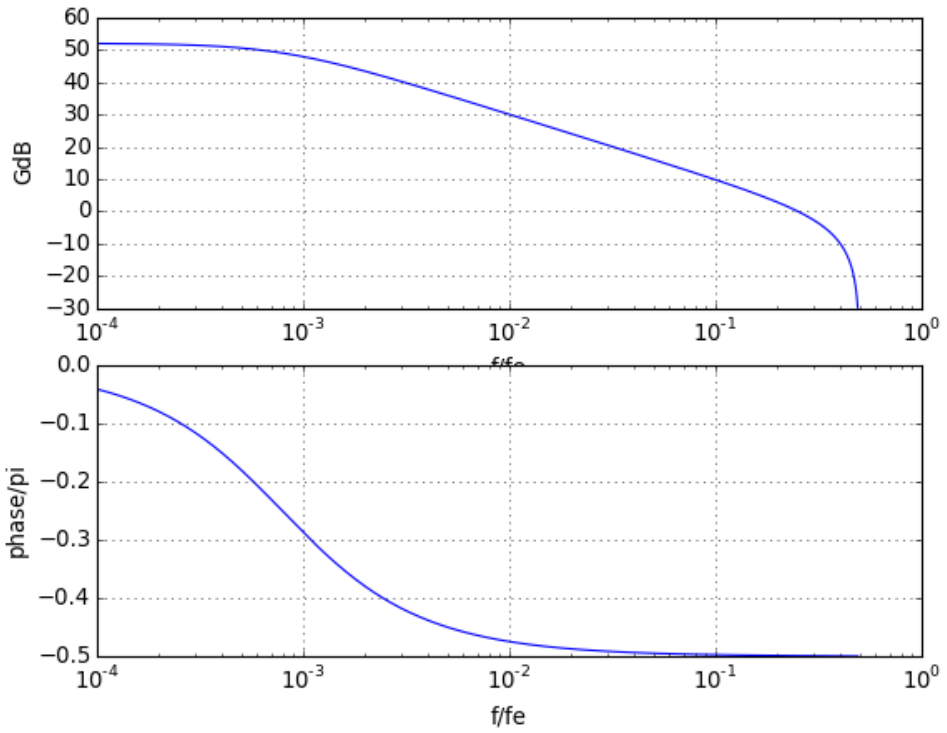
$$H(z) = \frac{g}{2} r \frac{T_e}{\tau} \frac{1+z^{-1}}{1-rz^{-1}} \tag{12}$$

avec :

$$r = \frac{1}{1+\frac{T_e}{\tau}} \tag{13}$$

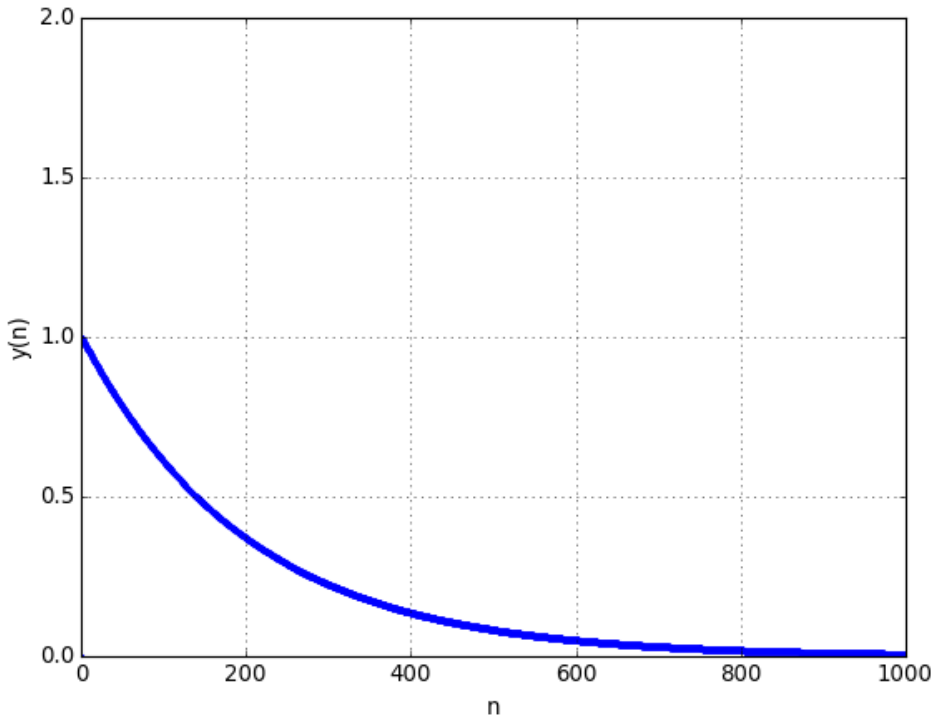
Le pôle est $z = r$ qui est strictement inférieur à 1, ce qui rend le filtre stable. Pour que le domaine de fréquence d'intégration soit large, il faut que le rapport T_e/τ soit faible, et donc que r soit proche de 1. Par exemple, si l'on veut un domaine d'intégration qui commence à environ un centième de la fréquence d'échantillonnage, on choisit $T_e/\tau = 1/100$ ou moins. Pour la forme de la réponse fréquentielle, on peut choisir librement le coefficient en facteur dans H, qui n'a d'effet que sur l'amplitude du signal en sortie, pas sur sa forme.

```
rt = 0.005
r=1.0/(1+rt)
a=[1,-r]
b=[1,1]
w,h=scipy.signal.freqz(b,a,worN=numpy.logspace(-4,-0.31,1000)*2*numpy.pi)
figure()
subplot(211)
plot(w/(2*numpy.pi),20*numpy.log10(numpy.absolute(h)))
xlabel("f/fe")
ylabel("GdB")
xscale('log')
grid()
subplot(212)
plot(w/(2*numpy.pi),numpy.angle(h)/numpy.pi)
xlabel("f/fe")
ylabel("phase/pi")
xscale('log')
grid()
```



Ce filtre passe-bas a une fréquence de coupure environ égale au millième de la fréquence d'échantillonnage, et un domaine intégrateur qui commence environ au centième de cette fréquence. Voyons la réponse impulsionnelle de ce filtre :

```
x = numpy.zeros(1000)
x[0] = 1.0
y = filtrage_recursif(x,a,b)
figure()
plot(y,".")
xlabel("n")
ylabel("y(n)")
axis([0,1000,0,2])
grid()
```

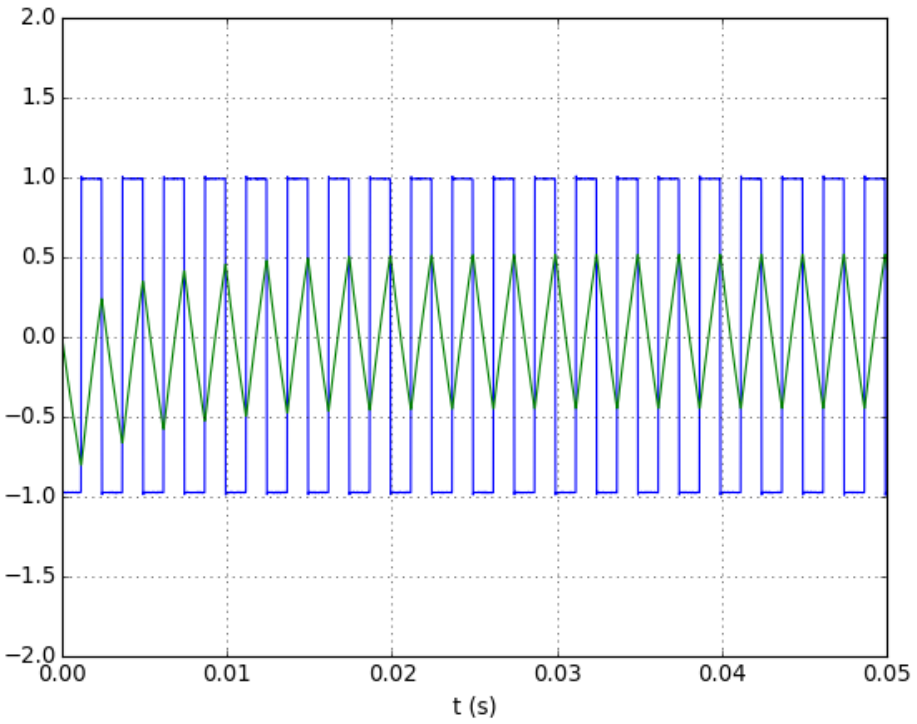


La réponse impulsionnelle tend vers zéro car le filtre est stable, mais la décroissance est lente, d'autant plus que r est proche de 1. Le temps de réponse est de l'ordre de τ. La contrepartie d'un filtre intégrateur à très large bande est donc un temps de réponse très long, qui peut être gênant lors des transitoires (changement de fréquence par exemple). Il faudra donc choisir τ au plus juste en fonction de la fréquence minimale des signaux à intégrer.

3.c. Test du filtre

Voici l'application du filtre au signal créneau. Les valeurs de $b_0 = b_1$ sont choisie pour abaisser le gain.

```
b=[0.01,0.01]
y = filtrage_recursif(u0,a,b)
figure()
plot(t0,u0)
plot(t0,y)
xlabel("t (s)")
axis([0,0.05,-2,2])
grid()
```

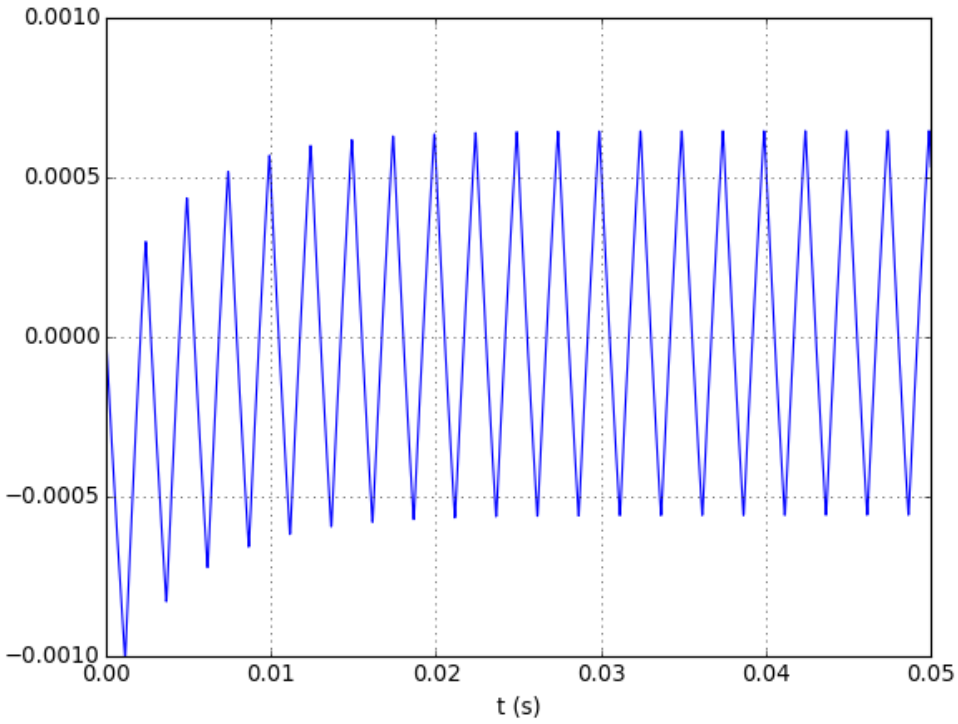


Après le régime transitoire de durée τ, on observe bien un état stationnaire avec un décalage en sortie relativement important mais constant.

Pour obtenir une intégration vraie, correspondant à $\tau = 1$ dans la relation (1), il faut utiliser :

$$y_n = r y_{n-1} + \frac{T_e}{2}\left(x_n + x_{n-1}\right) \tag{14}$$

```
te = t0[1]-t0[0]
b=[te/2,te/2]
y = filtrage_recursif(u0,a,b)
figure()
plot(t0,y)
xlabel("t (s)")
axis([0,0.05,-1e-3,1e-3])
grid()
```



4. Filtre intégrateur avec coupure de la fréquence nulle

Le filtre précédent peut être amélioré en annulant le gain à fréquence nulle. Il s'agit de faire un filtre passe-bande avec une fréquence de bande passante très faible. Un tel filtre peut être défini par la fonction de transfert en Z suivante :

$$H(z) = \frac{T_e}{\tau}\frac{g}{2}\frac{1-z^{-2}}{1-2rz^{-1}+r^2z^{-2}} \tag{15}$$

où r est un paramètre proche de 1 mais inférieur à 1. Son gain à fréquence nulle est bien nul à cause du zéro en $z = 1$. La relation de récurrence est :

$$y_n = 2ry_{n-1} - r^2y_{n-2} + \frac{T_e}{\tau}\frac{g}{2}(x_n - x_{n-2}) \tag{16}$$

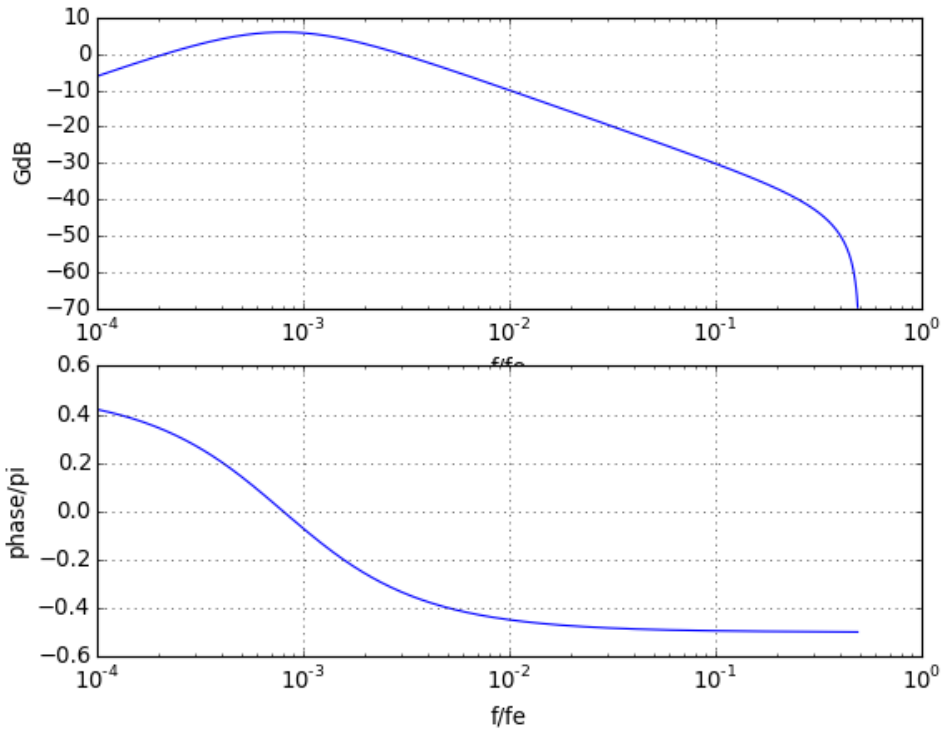
qui est un cas particulier de la forme générale suivante :

$$a_0y_n = b_0x_n + b_1x_{n-1} + b_2x_{n-2} - a_1y_{n-1} - a_2y_{n-2} \tag{17}$$

Voici un exemple :

```
r=0.995
a=[1,-2*r,r*r]
b=[0.01,0,-0.01]
w,h=scipy.signal.freqz(b,a,worN=numpy.logspace(-4,-0.31,1000)*2*numpy.pi)
figure()
subplot(211)
plot(w/(2*numpy.pi),20*numpy.log10(numpy.absolute(h)))
```

```
xlabel("f/fe")
ylabel("GdB")
xscale('log')
grid()
subplot(212)
plot(w/(2*numpy.pi),numpy.angle(h)/numpy.pi)
xlabel("f/fe")
ylabel("phase/pi")
xscale('log')
grid()
```



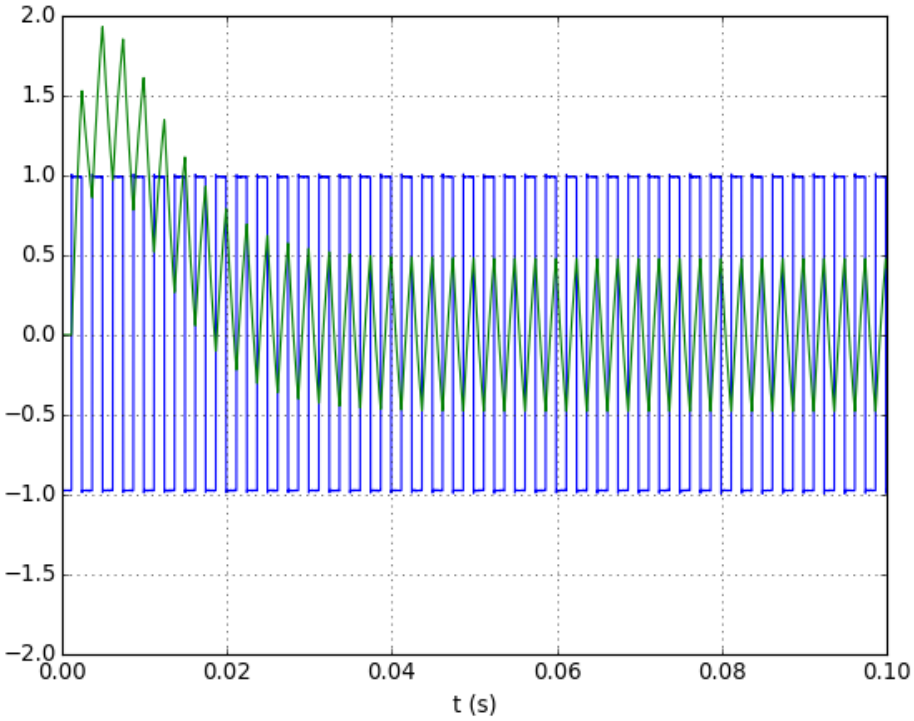
Voici la fonction qui réalise le filtrage les deux premières valeurs $y_0 = y_1 = 0$:

```
def filtrage_recursif(x,a,b):
    N = len(x)
    y = numpy.zeros(N)
    for n in range(2,N):
        y[n] = (-a[1]*y[n-1]-a[2]*y[n-2]+b[0]*x[n]+b[1]*x[n-1]+b[2]*x[n-2])/a[0]
    return y
```

et l'application au signal créneau :

```
y = filtrage_recursif(u0,a,b)
figure()
plot(t0,u0)
```

```
plot(t0,y)
xlabel("t (s)")
axis([0,0.1,-2,2])
grid()
```



On obtient un signal en sortie sans décalage. En contrepartie, la réponse transitoire est plus ample.

Pour obtenir une intégration vraie, il suffir de poser $g = \tau$.

Le script integrateurPermanent.py permet de tester ce filtre en temps réel. On peut ainsi observer le comportement transitoire lorsqu'on fait varier le signal en entrée.