Capacité numérique n° 1 — Échantillonnage d'un signal

Le fichier samplig.py propose trois procédures.

Procédure sampling

Permet d'échantillonner un signal.

```
1  def sampling(s,F_ech,T_tot):
2      N = int(T_tot * F_ech)              # nombre points échantillon
3      te = np.linspace(0., T_tot, N+1)    # liste des instants d'échantillonage
4      tp = np.linspace(0,T_tot,1000)      # abscisse du graphe du signal initial
5      plt.close()
6      plt.plot(te, s(te), 'or', markersize = 4, label = u"Signal echantillonne")
7      plt.plot(te, s(te), 'r--',linewidth=2)
8      plt.plot(tp, s(tp), 'b', markersize = 4, label = u"Signal reel")
9      plt.grid()
10     plt.xlabel("$t$")
11     plt.ylabel("$s(t),s_n$")
12     plt.legend(loc="upper right")
13     plt.title("$F_\mathrm{éch}=$"+str(F_ech))
14     plt.show()
15     return
```

Les arguments sont :

s *fonction* définissant le signal.

F_ech fréquence d'échantillonnage.

T_tot durée totale de l'acquisition.

La procédure retourne le graphe du signal initial, et celui du signal échantillonné (les points de l'échantillon sont reliés par des traits en tirets pour plus de lisibilité).

➤ La légende est positionnée à la ligne 12. On pourra modifier le placement en choisissant `'upper left'`, `'lower left'` ou `'lower right'`

Procédure EB

Cette procédure construit le signal obtenu à l'aide d'un échantillonneur bloqueur : à chaque échantillonnage, on maintient la valeur du signal constante jusqu'à l'échantillonnage suivant.

```
1  def EB(s,F_ech,T_tot):
2      N = int(T_tot * F_ech)
3      T_ech = 1/F_ech
4      time_ech = [0]
5      Ueb = [s(0)]
6      for i in range(N):
7          time_ech.append(i*T_ech)
8          time_ech.append(i*T_ech)
9          Ueb.append(Ueb[-1])
10         Ueb.append(s(i*T_ech))
11     time_ech.append(N*T_ech)
12     Ueb.append(s((N-1)*T_ech))
13     tp = np.linspace(0,T_tot,1000)
14     plt.close()
15     plt.plot(time_ech,Ueb, label = u"Signal echantillonne")
16     plt.plot(tp,s(tp), label = u"Signal reel")
17     plt.xlabel("$t$")
18     plt.legend(loc='upper right')
19     plt.title("$F_\mathrm{éch}=$"+str(F_ech))
20     plt.show()
21     return
```

Les arguments sont :

s *fonction* définissant le signal.

F_ech fréquence d'échantillonnage.

T_tot durée totale de l'acquisition.

Procédure EBQ

Cette procédure construit le signal obtenu à l'aide d'un échantillonneur bloqueur, en effectuant une quantification du signal.

```
def EBQ(s,F_ech,T_tot,Cal,q):
    N = int(T_tot * F_ech)              # nombre de points échantillon
    T_ech = 1/F_ech
    tp = np.linspace(0,T_tot,1000)
    time_ech = [0]                      # abscisses du signal bloqué
    n = 2**q                            # nombre valeurs quantifiées
    pas = 2*Cal/(n-1)                   # pas quantification
    valQ = [-Cal]
    for i in range(1,n):
        valQ.append(-Cal+i*pas)         # liste des valeurs accessibles par quantification
    time_ech = [0]
    Ueb = [s(0)]
    for i in range(N+1):
        time_ech.append(i*T_ech)
        time_ech.append(i*T_ech)
        Ueb.append(Ueb[-1])
        Ueb.append(s(i*T_ech))
    UebQ = []
    for x in Ueb:
        indice = int((x+Cal)/pas)
        if x < -Cal:
            UebQ.append(-Cal)
        elif x > Cal:
            UebQ.append(Cal)
        elif np.abs(x-valQ[indice]) <= pas/2:
            UebQ.append(valQ[indice])
        else:
            UebQ.append(valQ[indice+1])
    plt.close()
    fig, ax = plt.subplots()
    ax.grid(axis='y')
    while len(valQ)>40:             # pour éviter d'avoir plus de 40 valeurs de graduation en y
        valQ = valQ[::2]
    major_locator = FixedLocator(valQ)
    ax.yaxis.set_major_locator(major_locator)
    plt.ylim(-Cal,Cal)
    ax.plot(time_ech,UebQ, label = u"Signal echantillonne")
    ax.plot(tp,s(tp), label = u"Signal reel")
    plt.xlabel("$t$")
    plt.legend()
    plt.title("$F_\mathrm{éch}=$"+str(F_ech)+" et échantillonnage sur "+str(q)+" bits")
    plt.show()
    return
```

Les arguments sont :

s *fonction* définissant le signal.

F_ech fréquence d'échantillonnage.

T_tot durée totale de l'acquisition.

Cal calibre sur lequel se fait la quantification

q nombre de bits de la quantification

➤ L'intervalle [−Cal,+Cal] est découpé en 2^q intervalles pour une quantification sur q bits.

Signaux prédéfinis

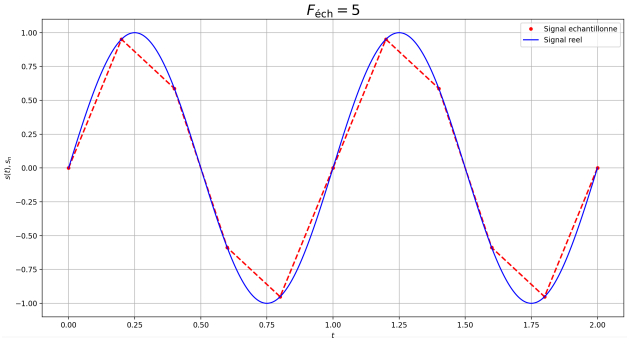
Quelques signaux usuels ont été prédéfinis :

```
f = 1       # fréquence du signal

def sinus(t):
    return np.sin(2. * np.pi * f * t)

def triangle(t):
    return signal.sawtooth(2. * np.pi * f * t,0.5)

def dent_scie(t):
    return signal.sawtooth(2. * np.pi * f * t,1)

def carre(t):
    return signal.square(2. * np.pi * f * t,.5)
```

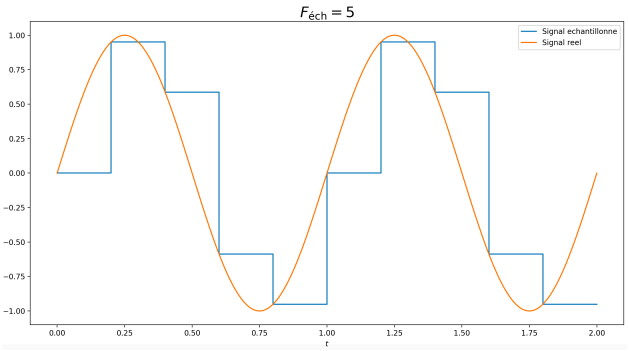
➤ On pourra modifier la valeur .5 de la fonction carre pour obtenir un signal créneau de rapport cyclique différent.

Exemples d'utilisation

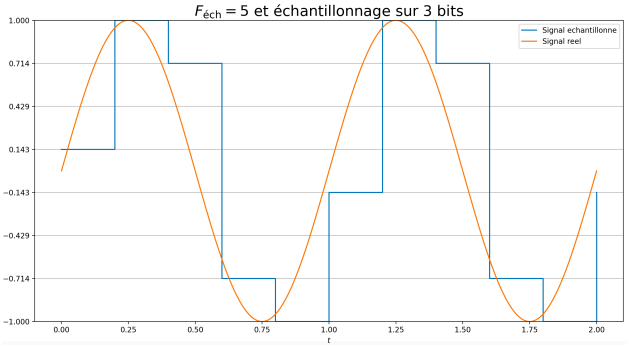
`sampling(sinus,5,2)`



`EB(sinus,5,2)`

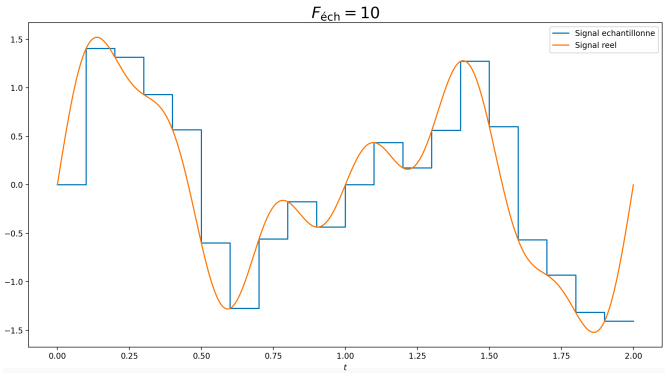


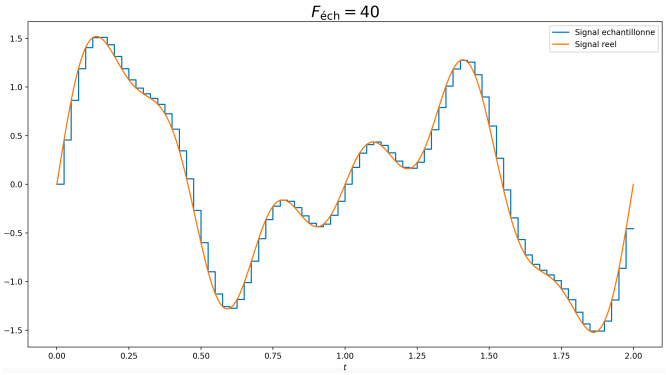
`EBQ(sinus,5,2,1,3)`



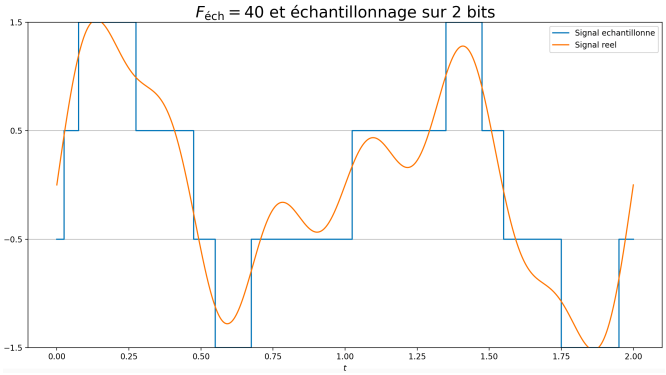
```
def x(t):
    return np.sin(2*np.pi*t)+.6*np.sin(2*np.pi*1.5*t)+.35*np.sin(2*np.pi*3*t)
```

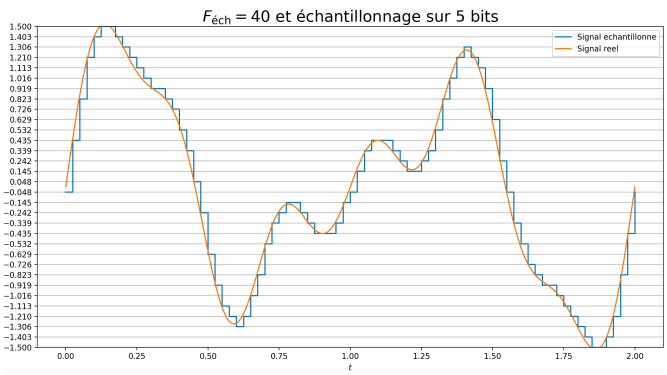
Influence de la fréquence d'échantillonnage



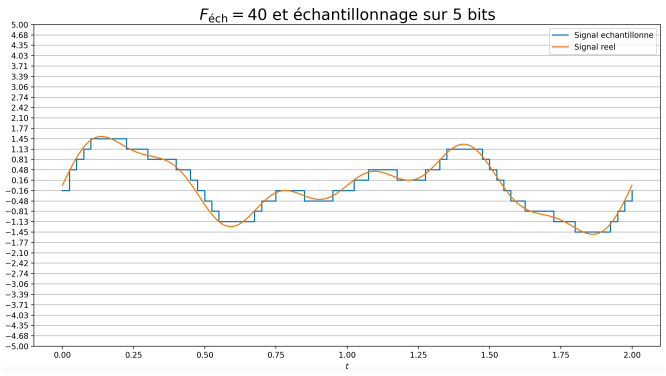


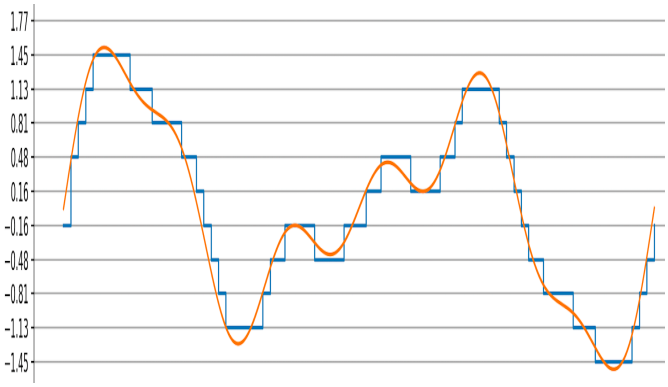
Influence du pas de quantification





Influence du calibre





➤ On cherchera à prendre un calibre le plus proche des valeurs extrêmes du signal afin de moins perdre en précision de quantification.