TP1 : Signaux Numériques

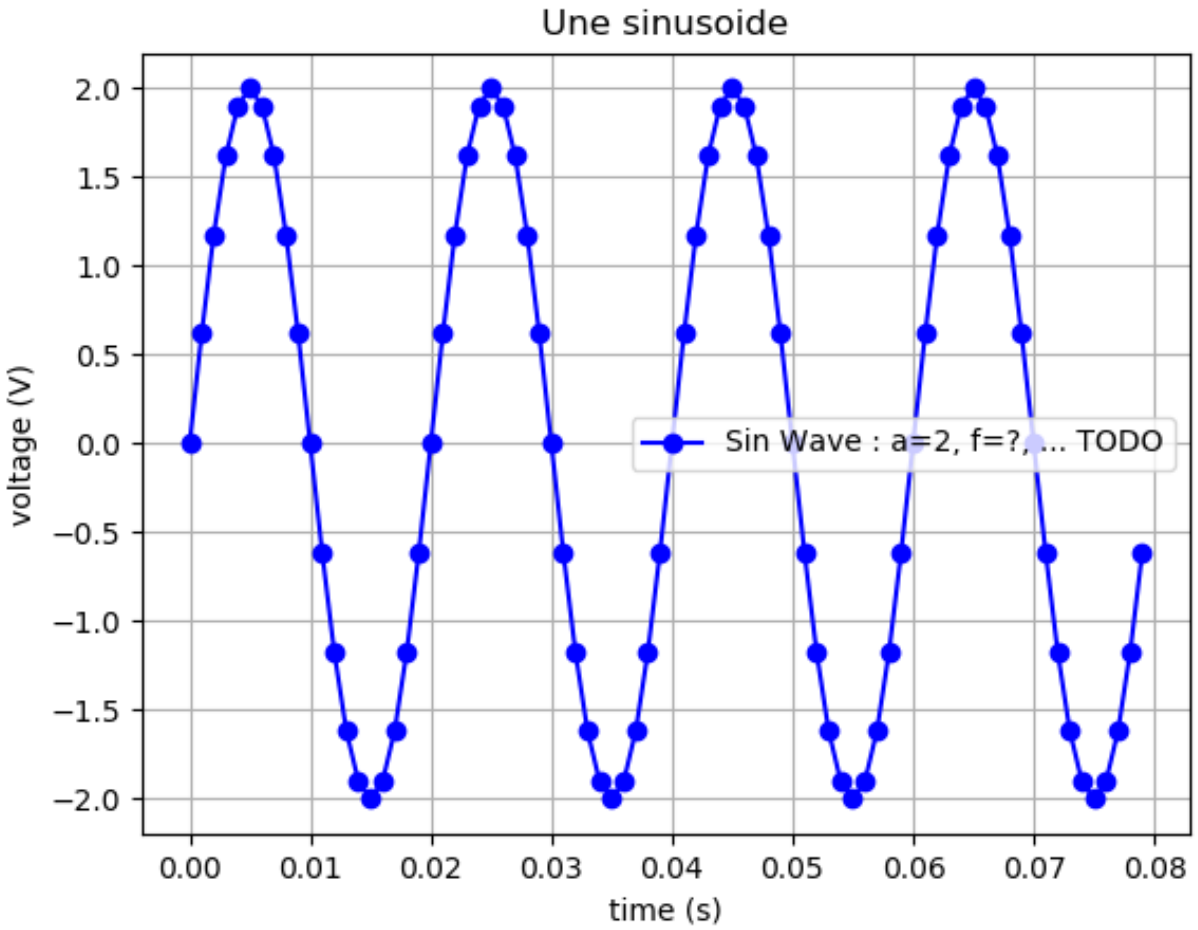
Gilles Menez - UCA - DS4H

---

1 Représentation temporelle d'un signal

L'objectif de cette première question est de générer un signal numérique et de montrer sa représentation temporelle en traçant ses échantillons.

- ➢ On débute avec une solution simpliste et incomplète !
- ➢ Au final, votre solution doit produire au final la même figure :



```python
'''
File : tp1_0_stud.py

Creer un signal numerique et l'afficher
RMQ : On utilise ici le B,A,BA de Python.

     Si vous savez faire mieux : objets ?, numpy ? ... SURTOUT lachez vous !!!!!
     car la solution "pythonique" n'est pas là :-(
'''

import math
import matplotlib.pyplot as plt

#----------------------------------------

def make_sin(a=1.0, f=440.0, fe=8000.0, ph=0, d=1):
    """
    Create a synthetic 'sine wave'
    """
    omega = 2*math.pi*f
    N = int(d*fe)
    te = 1.0/fe

    sig_t = []
    sig_s = []
    for i in range(N):
        t = te*i
        sig_t.append(t)
        sig_s.append(a*math.sin((omega*t)+ph))

    return sig_t, sig_s

#----------------------------------------

def plot_on_ax(ax, inx, iny, label, format='-bo'):
    ax.plot(inx,iny,format,label=label)
    ax.set_xlabel('time (s)')
    ax.set_ylabel('voltage (V)')

#----------------------------------------

def decorate_ax(ax, title):
    ax.set_title(title)
    ax.grid(True)
    ax.legend()

#========================================

if __name__ == '__main__':
    a=TODO
    f=TODO
    fe=TODO
    ph=TODO
    d=TODO

    # Generation du signal
    x,y=make_sin(a,f,fe,ph,d)

    # Representation graphique
    fig,ax = plt.subplots()
    plot_on_ax(ax,x,y,"Sin Wave : a={}, f={}, ... TODO".format(a,"?"))
    decorate_ax(ax,"Une sinusoide")

    plt.savefig("./basic_sin.png")
    plt.show()

#========================================
```

Remarque :

Je sais bien que vous êtes des "fans" du "copier-coller". **Mais pour une fois, prenez le temps de taper ce code et de comprendre ce qu'il signifie**. De plus, un "copier-coller" (à partir d'un document en pdf) va introduire des codes de caractères qui ne sont pas reconnus par l'interpréteur Python ... **vous allez perdre du temps** !

Dans l'état actuel, le code fourni est **incomplet** et ne permettra pas de produire la figure attendue !
Certes, le signal qui est dessiné sur le sujet est une sinusoïde, mais quels sont ses paramètres ?

- fréquence ?
- fréquence d'échantillonnage ?
- amplitude ?
- ...

Vous devez donc les déterminer à partir du graphique afin de pouvoir compléter le code Python et pouvoir faire exécuter.

N'hésitez pas à placer des commentaires dans le code décrivant le rôle des variables

➢ Par exemple, que représente la variable N ?

Matplotlib

Ce code utilise "Matplotlib" qui est une (la !?) bibliothèque Python pour réaliser des tracés 2D/3D.

On peut l'utiliser pour générer différentes représentations graphiques :

➡ des nuages de points,

➡ des courbes de points,

➡ des histogrammes,

➡ des camemberts, ...

`https://matplotlib.org/2.0.2/gallery.html`

Le problème de l'approche "bibliothèques" utilisée par l'informatique moderne est le temps nécessaire à "la montée en compétences".
Si on devait lire attentivement les 2000 pages de la documentation avant de commencer à l'utiliser. Il faut donc apprendre à extraire l'essentiel pour commencer l'utilisation :

`https://matplotlib.org/stable/tutorials/introductory/quick_start.html`

Sur cette page Web, la figure qui décrit l'anatomie d'une figure "Matplotlib" est **FONDAMENTALE** !

➢ On y voit ce qu'est une "figure", un "axe", un "plot", un "scatter plot", une "legend" ...

La plupart de ces éléments sont utilisés dans le code qui vous est fourni.

➢ Vous pouvez donc ainsi apprendre "par l'exemple" à utiliser "Matplotlib".

`https://matplotlib.org/2.0.2/api/pyplot_api.html`

Vous noterez aussi comment changer le "format" du tracé. Pour l'instant, le format utilisé par le tracé est spécifié dans la définition de la fonction `plot_on_ax` par la valeur par défaut du paramètre format : `'-bo'`

➢ Un tel format de tracé signifie : "en utilisant la couleur bleu représente les points par un 'o' et relie ces points par un trait continu."

Mais attention ! ... les formats d'affichages peuvent être trompeurs :

➢ Quel est l'inconvénient du format `':b'` ?

1.1 TODO

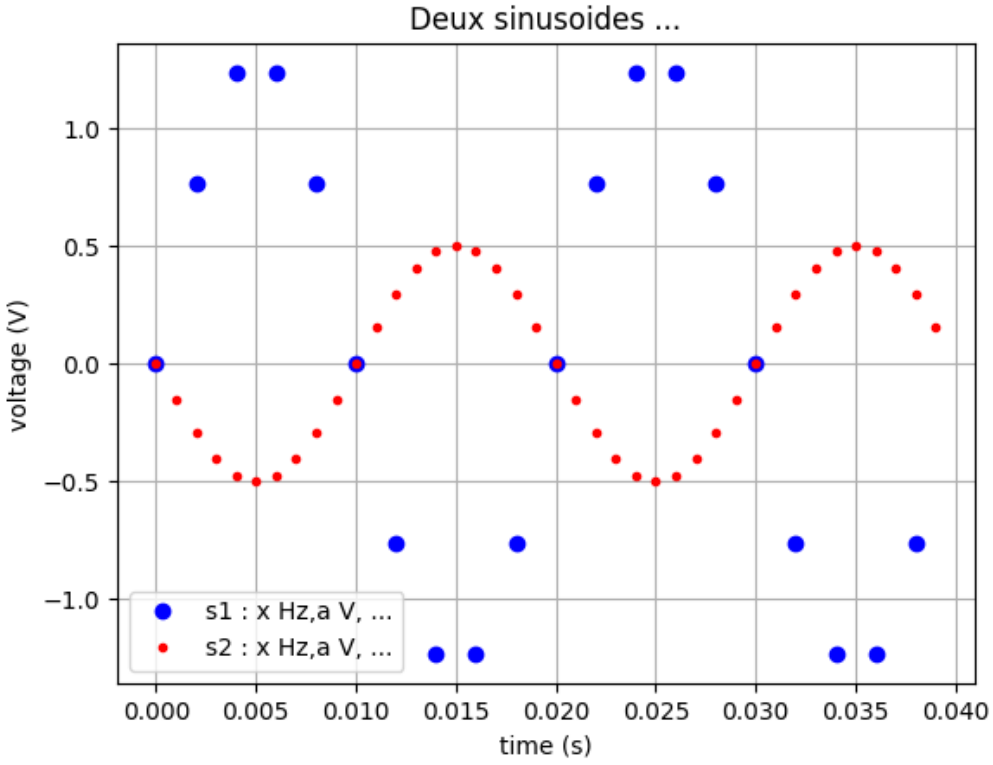
(a) Vous devez donc reproduire et complèter la courbe donnée en exemple.

(b) Finir la ligne de code permettant de construire la légende de l'axe. Vous évitez d'utiliser des littéraux. Vous pouvez soit nommer des constantes, soit utiliser des variables.

(c) **Au final, on veut que la courbe se dessine en vert avec des points d'échantillonnage représentés par des étoiles et reliés par un trait en pointillé.**

2 TODO : Deux signaux

Faire évoluer le code pour être capable d'afficher la "même" fenêtre que celle-ci (et avec la légende complétée) :



Même si il doit être conforme, **le dessin n'est pas la seule finalité de l'exercice !**

➢ Il s'agit avant tout de comprendre les caractéristiques des signaux.

(a) Quantifiez en quoi ils sont différents ?

- ➢ amplitudes (a_1, a_2) ?
- ➢ fréquences $(freq_1, freq_2)$?
- ➢ phases à l'origine $(ph1, ph2)$?
- ➢ fréquences d'échantillonage (fe_1, fe_2).

On ne procède pas par "tâtonnement" pour définir ces paramètres.

- ➢ On calcule !
- ➢ Et au passage . . . une amplitude n'est pas négative.

(b) Là encore, vous remarquez que la légende est incomplète. A vous de remplir !

2.1 Petite aide au niveau du graphisme :

L'affichage de deux courbes sur un même axe peut être fait grâce à l'appel unique :

```
1    ax.plot(t, s1 , "r*", t , s2, "bo")
```

ou décomposé en deux appels :

```
1    ax.plot(t, s1, "r*")
2    ax.plot(t, s2, "bo")
```

La **bonne méthode c'est la deuxième méthode** car généralisable/iterable si le nombre de signaux venait à augmenter.

2.2 Au niveau du code Python . . .

Vous avez remarqué que l'UE Réseaux/Télécommunication se situe dans la filière informatique et par conséquent, on ne peut pas se satisfaire de codes Python médiocres.

En l'occurrence, la modification d'une grandeur paramétrant un signal doit "automatiquement" se répercuter sur la légende !

Dans l'organisation à venir de vos codes, il serait souhaitable d'introduire intelligemment les "bons" modules . . . histoire de ne pas se retrouver dans six semaines avec des fichiers très/trop longs.

Dans la mesure du possible essayez de "REUTILISER" vos codes. Les TPs sont longs, mais ils seront infaisables si vous passez votre temps à tout re-écrire et donc tout re-tester . . .

Enfin, l'utilisation de variables de modules est prohibée et nous vous rappelons que la notion de fonction permet d'abstraire élégamment une problématique et de structurer le code.

Bref, pas de fonctions/méthodes . . . pas de chocolats ;-)

3 Génération de signaux

Pour faire référence au cours, à ce stade, vous avez développé un code réalisant la **numérisation** d'un signal continu dont l'équation est :

$$s(t) = asin(\omega\, t + \phi) \tag{1}$$

avec $\omega = 2 \times \pi \times f$ la pulsation en rad/s et f la fréquence du signal $s(t)$ en H_z.

Pour l'instant nous nous sommes essentiellement intéressé à la **discrétisation en temps** puisque nous avons calculé $s(t)$ pour $t = i \times \frac{1}{f_e}$.

Par contre toutes les amplitudes sont "autorisées"(/potentiellement présentes) et on n'a donc pas **discrétisé en amplitude** !

3.1 Signal carré

Sur la base des équations en temps continu fournie par l'url suivante :

https://fr.wikipedia.org/wiki/Signal_carré

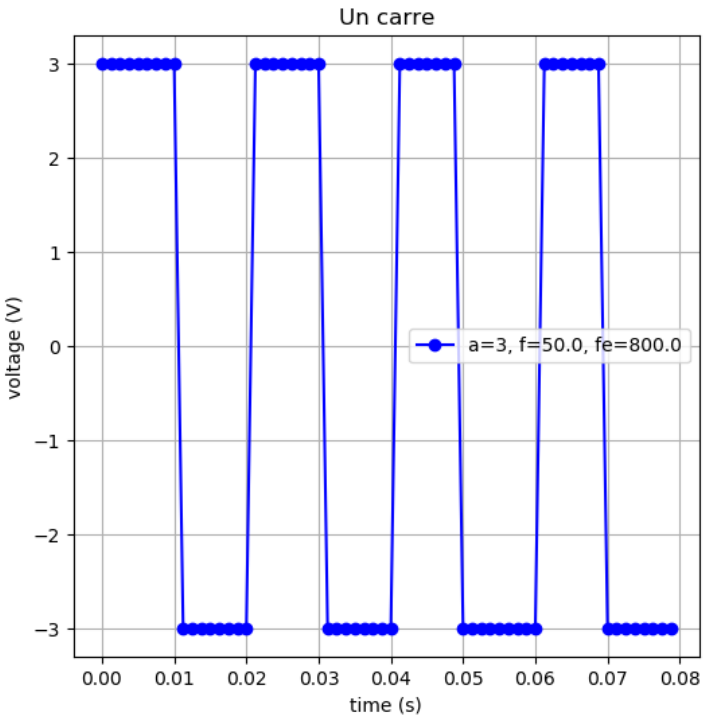
et notamment l'équation :

$$s(t) = a * sgn(sin(\omega\, t)) \tag{2}$$

avec $sgn(x)$, la fonction "signe" qui est égale à 1 quand x est positif ou nul, et -1 quand x est négatif.

(a) réaliser une fonction générant un signal carré.

Vous devriez obtenir, si $a = 3, f = 50.0, fe = 800.0, d = 0.080$:



Un algorithme si simple . . . mais FAUX !

Essayer votre méthode/fonction de génération de carré avec les paramètres suivants : $a = 3$, $f_e = 300$, $f = 50$

(a) Que constatez vous au niveau de la durée des "états" (-1 et +1) du signal ? Expliquez et corrigez ?

Pour vous aider à diagnostiquer le problème :

➢ Tracer sur la même figure les échantillons du sinus qui est utilisé pour fabriquer le carré.

(b) Trouver une alternative sur `https://en.wikipedia.org/wiki/Square_wave`.

Prenez la formule avec les opérateurs "floor".

Vous **éviterez d'utiliser** les séries de Fourier que l'on garde pour plus tard.

3.2 Signal Dent de scie

Sur la base de l'équation en temps continue : `http://en.wikipedia.org/wiki/Sawtooth_wave`

$$x(t) = 2 \times \left(\frac{t}{T} - \left\lfloor \frac{t}{T} - \frac{1}{2} \right\rfloor \right) \tag{3}$$

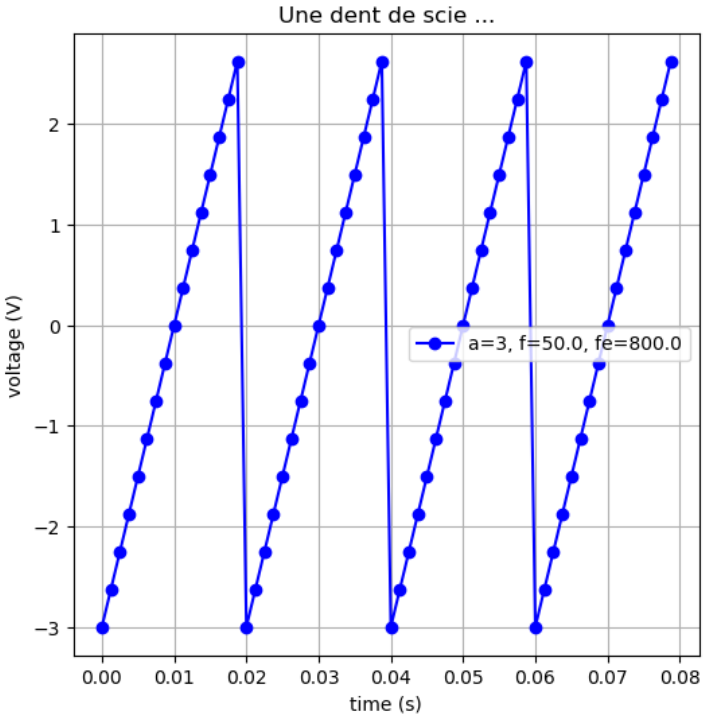
$$= 2 \times \left(\frac{t}{T} - \text{floor}\left(\frac{t}{T} - \frac{1}{2} \right) \right) \tag{4}$$

Mais à mon avis, l'équation fournie par wikipédia est fausse . . . je vous propose :

$$x(t) = 2 \times a \times \left(\frac{t}{T} - \left\lfloor \frac{t}{T} \right\rfloor - \frac{1}{2} \right) \tag{5}$$

(a) Proposer une fonction permettant d'obtenir un signal "dent de scie".

Vous devriez obtenir, si $a = 3, f = 50.0, fe = 800.0, d = 0.08$:



Attention !

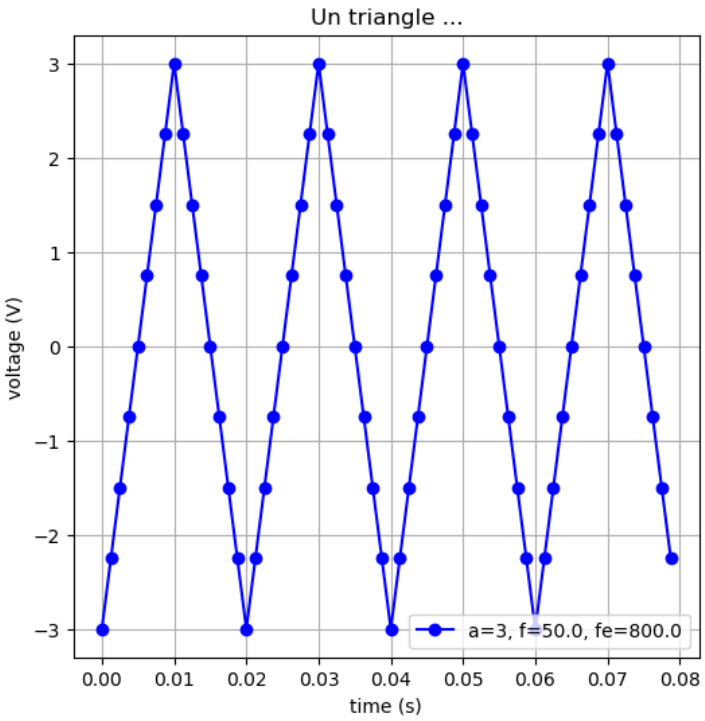
Cette figure est "imparfaite" car l'amplitude du signal n'est pas 2.0 (en 0.02 sec) ... pourquoi ? peut-on faire mieux ?

3.3 Signal Triangle

$$x(t) = a \times \left(4 \times \left(\left|\frac{t}{T} - \left\lfloor \frac{t}{T} + \frac{1}{2} \right\rfloor\right|\right) - 1.0\right) \tag{6}$$

(a) Proposer une fonction (dans le même module) permettant d'obtenir un signal "triangle".

Vous devriez obtenir, si $a = 3, f = 50.0, fe = 800.0, d = 0.08$:



4 Echantillonnage/Qualité/Débit

4.1 Petits Quizzs

Imaginons que l'on transmette un signal sinusoïdal (a,f, f_e,ph) sur un réseau :

(a) Quel est le paramètre du signal qui influe directement sur le débit (en bit/s) nécessaire sur le réseau ?

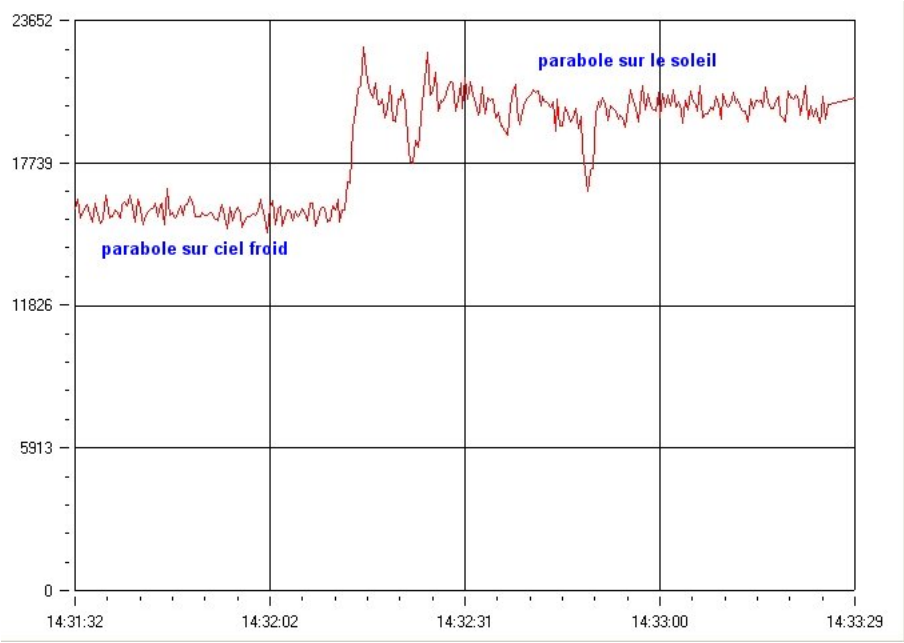
(b) Si le signal sinusoïdal tel qu'on le génère actuellement a une amplitude de 2 Volts, une fréquence de 440 Hz, une fréquence d'échantillonnage de 44100 Hz , une phase de 0 , **quel est le débit nécessaire au transport** de ce signal ?

(c) Si on était capable de concevoir un détecteur de sinusoïde (dont on fera l'hypothèse qu'elle a une amplitude de 1), quel est le minimum du nombre d'échantillons nécessaire à sa reconnaissance ?

5 Du bruit sur la ligne

Un câble et à fortiori l'atmosphère sont des milieux sensibles aux perturbations et aux bruits.

Par exemple le soleil est un source de bruit. Si vous prenez une parabole et que vous la pointez vers le ciel vous constatez que vous recevez un signal. Ce signal est encore plus fort si vous pointez vers le soleil.



http://www.f1afz.fr/AO_40/bruit_solaire.htm

Ce signal viendra "bruiter" les signaux transportés par l'atmosphère mais aussi par les câbles.

➢ Si le câble n'est pas "écranté", ce n'est pas le plastique enrobant le câble qui suffira à bloquer le bruit.

"En juillet 2012, une tempête solaire hors norme a manqué la Terre de peu. Elle aurait pu provoquer plus de mille milliards de dollars de dégâts. Et mettre hors service les systèmes de communication."

https://www.nouvelobs.com/sciences/20140320.OBS0677/comment-une-monstrueuse-eruption-solaire-a-failli-precipiter-la-terre-dans-le-chaos.html

5.1 Formalisation

Il y a plusieurs formes de bruits parmi lesquels les bruits additifs.

➢ Il s'agit de signaux (b_n) d'origine tout à fait extérieure et indépendants au signal véhiculé par le canal de transmission (x_n) et qui viennent se superposer à celui-ci (y_n).

$$y_n = x_n + b_n \tag{7}$$

Le signal de bruit b_n est par nature imprévisible et inconnu.

Dans cette question, on supposera deux formes de bruit :

① Le bruit statistiquement modélisé, par exemple un bruit gaussien.

Ce bruit est présent sur tous les échantillons et son amplitude est supposée issue d'une loi gaussienne (cf random.gauss(mu,sigma)).

② Le bruit impulsif qui vient modifié seulement quelques échantillons en modifiant souvent fortement leur amplitude.

5.2 TODO :

(a) Faire une fonction qui génère un signal de bruit gaussien.

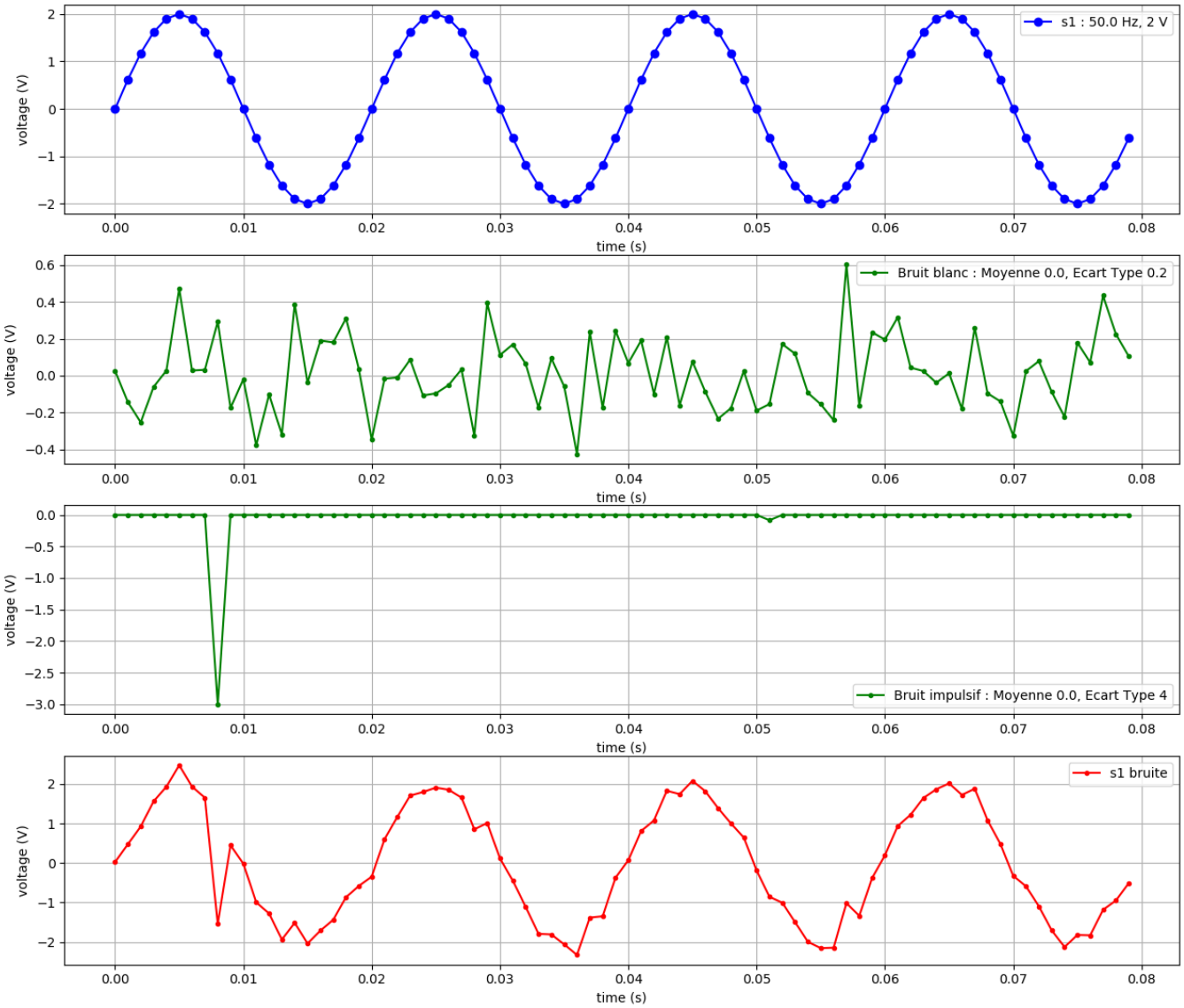
Attention dans la mesure, où la dernière question vise à additionner des signaux, ils faudrait créer ce signal de bruit à partir d'une base temporelle déjà existante. De façon à ce que les signaux aient les mêmes abscisses !

(b) Faire une fonction qui génère un signal de bruit impulsif.

(c) Faire une fonction permettant d'ajouter un signal à un autre.

(d) Utiliser ces fonctions pour bruiter par exemple le signal sinusoïdale de la première question.

Ca devrait produire cela :



On vous propose un ”main” (configuration simpliste) pour vous guider/aider :

```
# version simple MAIS "pas assez objet" :-(
import tp1_0

#A remplir ....

"""======================================="""

if __name__ == '__main__':

    a=2
    f=50.0
    fe=1000.0
    ph=0
    d=0.08
    x1,y1 = tp1_0.make_sin(a,f,fe,ph,d)

    m = 0.0 # mean
    e = 0.2 # ecart type
    x2,y2 = noise_white(x1, m, e)

    m1 = 0.0 # mean
    e1 = 2*a # ecart type
    nbi = 2 # nombre d'impulsions sur le signal
    di = 2  # duree d'une impulsion en sample
    x3,y3 = noise_impulse(nbi, di, x1, m1, e1)

    x4,y4 = signal_add(x1,y1,x2,y2)
    x4,y4 = signal_add(x4,y4,x3,y3)

    fig,ax = plt.subplots(4)
    tp1_0.plot_on_ax(ax[0],x1,y1,"s1 : {} Hz, {} V".format(f,a),'bo-')
    tp1_0.decorate_ax(ax[0],"")

    tp1_0.plot_on_ax(ax[1],x2,y2,"Bruit blanc : Moyenne {}, Ecart Type {}".format(m,e),'g.-')
    tp1_0.decorate_ax(ax[1],"")

    tp1_0.plot_on_ax(ax[2],x3,y3,"Bruit impulsif : Moyenne {}, Ecart Type {}".format(m1,e1),'g.-')
    tp1_0.decorate_ax(ax[2],"")

    tp1_0.plot_on_ax(ax[3],x4,y4,"s1 bruite",'r.-')
    tp1_0.decorate_ax(ax[3],"")

    plt.savefig("../../Figs/noises_on_sin.png")
    plt.show()
```

6 Ecouter les signaux

Pour l'instant vous visualisez vos sons, et si on les écoutait ?

➢ A partir des signaux que vous êtes désormais capable de générer, on va essayer de produire un fichier au format audio WAV . . . pour écouter.

Le format **WAV** est une extension de fichiers audio, il s'agit d'un **conteneur** capable de recevoir des formats aussi variés que le MP3, le WMA, l'ATRAC3, l'ADPCM, et le **PCM**.

✔ C'est ce dernier qui est cependant le plus courant, et c'est pour cela que l'extension `.wav` est souvent – et donc à tort – considérée comme correspondant à des fichiers "sans pertes" (i.e. lossless).

✔ Le format WAV est standardisé sous Windows ; son pendant sous la plate-forme Macintosh est l'AIFF/AIFC.

Le conteneur WAV est désormais ancien, et peu pratique :

`http://fr.wikipedia.org/wiki/WAVEform_audio_format`

. . . mais il reste incontournable sous Windows et beaucoup de baladeurs.

6.1 Structure des fichiers WAV

L'en-tête d'un fichier WAV,

✔ commence dès le premier octet (offset 0),

✔ a une taille de 44 octets,

✔ et est constitué des champs suivants (listés dans l'ordre) :

```
[Bloc de déclaration d'un fichier au format WAVE]

   FileTypeBlocID  (4 octets) : Constante "RIFF"  (0x52,0x49,0x46,0x46)
   FileSize        (4 octets) : Taille du fichier moins les 8 octets
                                (FileTypeBlocID + FileFormatID)
   FileFormatID    (4 octets) : Format = "WAVE"  (0x57,0x41,0x56,0x45)

[Bloc décrivant le format audio]

   FormatBlocID    (4 octets) : Constante "fmt"  (0x66,0x6D, 0x74,0x20)
   BlocSize        (4 octets) : Nombre d'octets du bloc moins les 8
                                octets (FormatBlocID+BlocSize)
   AudioFormat     (2 octets) : Format du stockage dans le fichier (1: PCM, ...)
   NbrCanaux       (2 octets) : Nombre de canaux (de 1 à 6, cf. ci-dessous)
   Frequence       (4 octets) : Fréquence d'échantillonnage (en Hertz)
                                [Valeurs standardisées : 11025, 22050, 44100 et éventuellement 48000 et 96000]
   BytePerSec      (4 octets) : Nombre d'octets à lire par seconde (i.e., Frequence * BytePerBloc).
   BytePerBloc     (2 octets) : Nombre d'octets par bloc d'échantillonnage
                                (i.e., tous canaux confondus : NbrCanaux * BitsPerSample/8).
   BitsPerSample   (2 octets) : Nombre de bits utilisées pour le codage de chaque
                                échantillon (8, 16, 24) sur chaque canal

[Bloc des données]

   DataBlocID      (4 octets) : Constante "data"  (0x64,0x61,0x74,0x61)
   DataSize        (4 octets) : Nombre d'octets des données
                  (i.e. "Data[]", i.e. taille_du_fichier - taille_de_l'entête  (qui fait 44 octets normalement).
   DATAS[] : [Octets du Sample 1 du Canal 1] [Octets du Sample 1 du Canal 2]
             [Octets du Sample 2 du Canal 1] [Octets du Sample 2 du Canal 2] ...
```

On peut se douter que dans le cadre d'une fonction de lecture/écriture (fournie par une bibliothèque), on sera amené à fournir certaines de ces informations.

Par exemple, pour le champs "Nombre de canaux" :

1 pour mono,

2 pour stéréo,

3 pour gauche, droit et centre

4 pour face gauche, face droit, arrière gauche, arrière droit

5 pour gauche, centre, droit, surround (ambiant)

6 pour centre gauche, gauche, centre, centre droit, droit, surround (ambiant)

Attention : Eternel problème de la représentation mémoire. Ces fichiers vont "naviguer" sur des machines différentes, il fallait bien choisir une représentation de l'information :

➡ Les octets des mots sont stockés sous la forme "little endian" (i.e. octet de poids fort à l'adresse la plus petite i.e à droite) . . . normal Microsoft/Intel !

6.2 Dans un fichier . . .

Il existe un module Python permettant la manipulation des fichiers/formats WAVE :

`http://docs.python.org/library/wave.html?highlight=wave#module-wave`

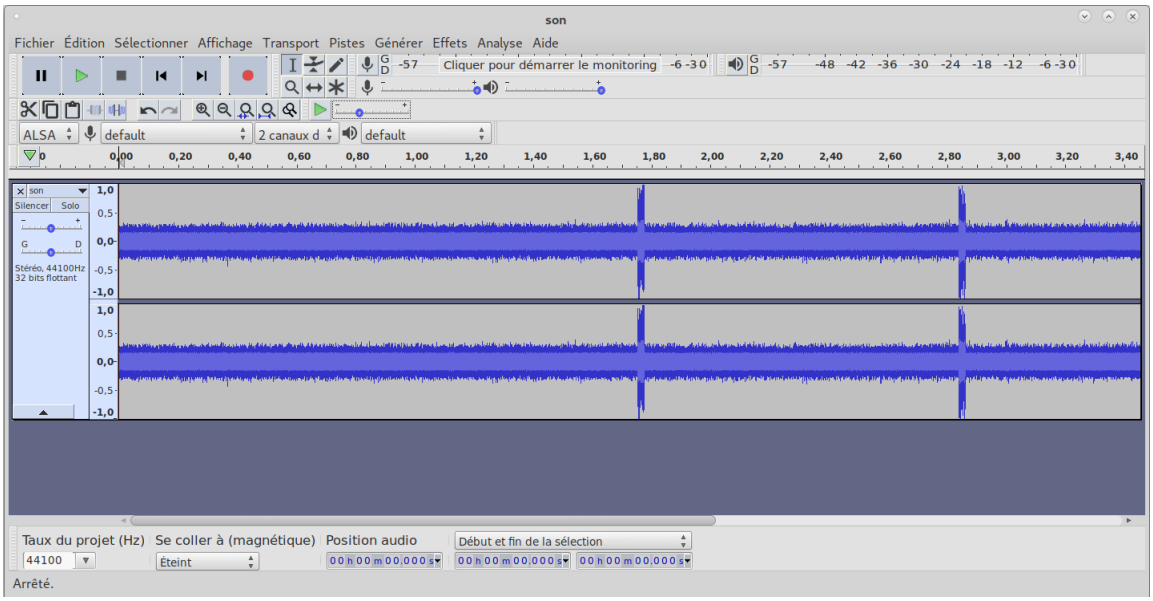
Le fichier suivant vous donne une fonction et son utilisation permettant de convertir/écrire votre signal en un son dans un fichier `"son.wav"` .

```
'''
File : write_a_wave.py
@author: menez
'''

import struct, wave, math
import tp1_0, noise

#--------------------------------------

def write_a_wave_file(x,y,fn="son.wav"):
    """
    Ecriture d'un signal audio (x,y) dans un fichier audio au format
    WAV (PCM 8 bits stereo 44100 Hz)

    IL Y A DES CONTRAINTES sur le signal !
    fe = 44100
    Amplitude  <=1
    """
    nbCanal = 2    # stereo
    nbOctet = 1    # taille d'un echantillon : 1 octet = 8 bits
    fe = 44100   # frequence d'echantillonnage
    nbEchantillon = len(x) # nombre d'echantillons

    wave_file = wave.open(fn,'w')
    parametres = (nbCanal,nbOctet,fe,nbEchantillon,'NONE','not compressed') # tuple
    wave_file.setparams(parametres)    # creation de l'en-tete (44 octets)

    # niveau max dans l'onde positive : +1 -> 255 (0xFF)
    # niveau max dans l'onde negative : -1 ->   0 (0x00)
    # niveau sonore nul :                 0 -> 127.5 (0x80 en valeur arrondi)

    print("La sinusoide devient un son ... ")
    for i in range(0,nbEchantillon):
        val = y[i]
```

```
36          if val > 1.0:
37              val = 1.0
38          elif val < -1.0:
39              val = -1.0
40
41          val = int(127.5 + 127.5 * val)
42          try:
43              fr  = struct.pack('BB', val,val) # unsigned int
44          except struct.error as err:
45              print(err)
46              print("Sample {}  = {}/{}".format(i,y[i],val))
47
48          wave_file.writeframes(fr) # ecriture de la frame
49
50      wave_file.close()
51
52  #--------------------------------------
53
54  def make_anoisysignal(a,f,fe,ph,d):
55
56      x1,y1 = tp1_0.make_sin(a,f,fe,ph,d)
57
58      m = 0.0 # mean
59      e = 0.05 # ecart type
60      x2,y2 = noise.noise_white(x1, m, e)
61
62      m1 = 0.0 # mean
63      e1 = 1.6*a # ecart type
64      x3,y3 = noise.noise_impulse(2, 1000, x1, m1, e1)
65
66      x4,y4 = noise.signal_add(x1,y1,x2,y2)
67      x4,y4 = noise.signal_add(x4,y4,x3,y3)
68
69      return x4,y4
70
71  #======================================
72
73  if __name__ == '__main__':
74
75
76      # Generation du signal
77      x,y = make_anoisysignal(a=0.2,f=440.0,fe=44100.0,ph=0,d=5)
78
79      write_a_wave_file(x,y)
```

6.3 Audacity

Essayer et lire/écouter le résultat avec l'excellent outil : `audacity` ?



7 Qualité

Il y aurait des choses intéressantes à faire sur le "ressenti" !

La question est : "Quel est le niveau d'un bruit audible ?"

Mais on a encore plein de choses à faire dans les TPs suivants ...