Outils Numériques pour l'Ingénieur

Traitement de signal

ludovic.charleux@univ-savoie.fr

www.polytech.univ-savoie.fr

Plan

Notations

Un signal ?

Dans ce cours on étudie le comportement d'un signal x issu de la mesure d'une grandeur physique (vitesse, température, ...). Le signal dépend d'une variable unique t qui peut représenter le temps, une position ...

Signal quelconque

D'un point de vue mathématique, le signal $x(t)$ défini par :

$$x : t \longmapsto x(t), \ \forall x \in [0, t_{max}]$$

Signal périodique

Si x est périodique, on note T sa période et f sa fréquence avec :

$$f = \frac{1}{T}$$

Plan

Python, c'est quoi ?

Pourquoi Python ?

- La lourdeur des calculs nécessite des outils numériques.
- Les signaux expérimentaux sont numérisés et donc aisément traités par ces outils.
- Python est un language simple, au spectre d'applications vaste.
- Python est libre et donc gratuit, vous pouvez donc l'installer rapidement sur toute machine. Il est présent sur la majorité des distributions de Linux.
- Pour l'installer et lire les documentations : http ://www.python.org/
- Modules utilisés :
  - Graphisme : Matplotlib
  - Calcul scientifique : Scipy
  - Calcul numérique : Numpy

Python : comment ça marche ?

Outils nécessaires

- Un éditeur de texte reconnaissant la syntaxe.
- Un terminal.

Exemple

On crée un fichier test.py dont le contenu est (voir dossier / listings) :

```
# listings/test.py
print "Hello world !"
```

On execute Python dans un terminal avec une des commandes suivantes :

```
python test.py
```

Ou :

```
python
execfile(test.py)
```

La seconde solution a l'intérêt de ne pas fermer l'interpréteur après l'exécution ce qui permet de débugger ou de modifier le code plus aisément.

Plan

Signal sinusoïdal (1/3)

Définition

```
# signal_sinusoidal.py
from math import sin, pi
# T: periode, k: amplitude, phi: dephasage
def signal_sinusoidal(t,T=1.,k=1.,phi=0.):
  return k*sin(2*pi*t/T+phi)
```

Fonction de tracé

On construit une fonction qui assure le tracé :

```
# trace_signal.py
import matplotlib.pyplot as plt

def trace_signal(t,x,fichier,xlabel='Temps $t$',
ylabel='Signal $x$',grid=True, style='r-'):
  plt.figure(0,figsize=(9,5))
  plt.clf()
  plt.plot(t,x,style,linewidth=2.0)
  plt.xlabel(xlabel, fontsize=15)
  plt.ylabel(ylabel, fontsize=15)
  plt.grid(grid)
  plt.savefig(fichier)
```

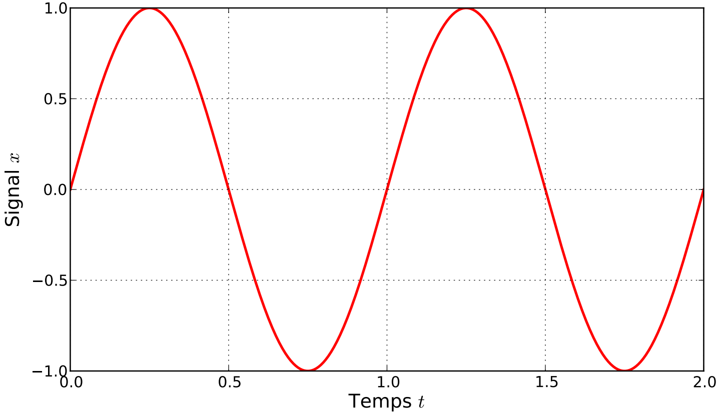
Signal sinusoïdal (2/3)

Tracé du signal

```
# listings/exemple1.py
# On charge la fonction signal_sinusoidal
from signal_sinusoidal import *
# On charge la fonction trace_signal
from trace_signal import *
from numpy import arange
signal = signal_sinusoidal # On definit le signal utilise
T = 1. # Periode du signal
t_min = 0. # Debut du calcul du signal
t_max = 2. # Fin du calcul du signal
np = 10000 # Nombre de points calcules
# On definit l'interval de temps a tracer
t = arange(np)/float(np)*(t_max-t_min)+t_min
# Fichier dans lequel tracer
fichier = '../figures/sinusoide.pdf'
x = [signal(tt,T=T) for tt in t]
trace_signal(t,x,fichier)
```

Signal sinusoïdal (3/3)

On obtient le fichier (vectoriel) sinusoide.pdf



Signal carré (1/3)

Construction de la fonction

```python
#listings/signal_carre.py
from numpy import floor
# T: periode, k: amplitude
def signal_carre(t,T=1.,k=1.):
   t = t - floor(t/T)
   x = -1.
   if t > T/2.: x = 1.
   return x
```

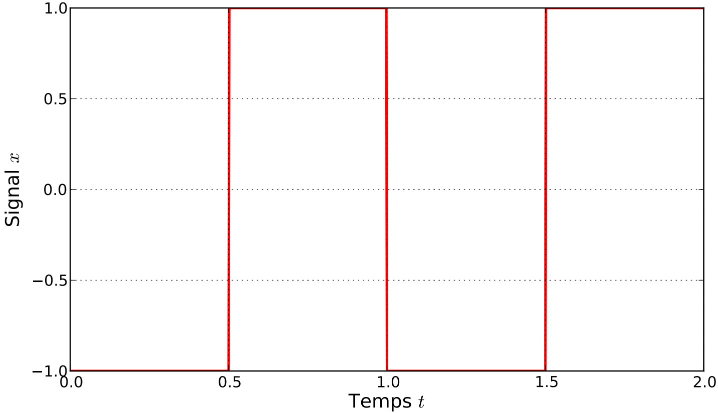
Signal carré (2/3)

Tracé du signal

```
# listings/exemple2.py
# On charge la fonction signal_carre
from signal_carre import *
# On charge la fonction trace_signal
from trace_signal import *
from numpy import arange
signal = signal_carre # On definit le signal utilise
T = 1. # Periode du signal
t_min = 0. # Debut du calcul du signal
t_max = 2. # Fin du calcul du signal
np = 1000 # Nombre de points calcules
# On definit l'interval de temps a tracer
t = arange(np)/float(np)*(t_max-t_min)+t_min
fichier = '../figures/carre.pdf'
x = [signal(tt,T=T) for tt in t]
trace_signal(t,x,fichier)
```

Signal carré (3/3)

On obtient le fichier (vectoriel) `carre.pdf`



Signal expérimental : mesure d'accélération sur une cloche

Intérêt

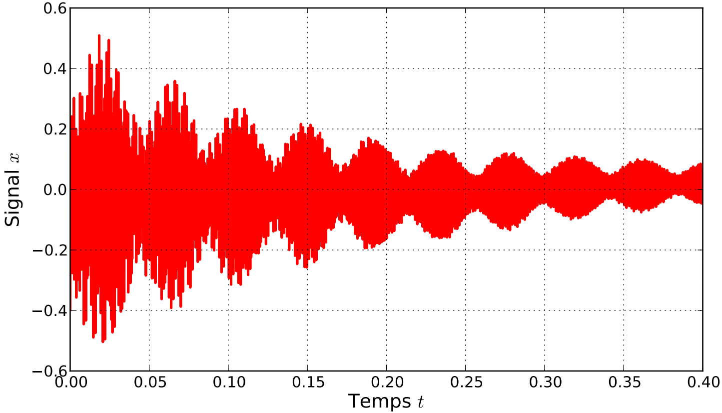
Ce signal pseudo périodique est obtenu expérimentalement au moyen d'un accélérometre fixé sur une cloche (visible en salle C114) et excitée par un battant. Il est fourni dans une fichier sérialisé par le module Python Pickle.

Tracé

```python
# listings/exemple4.py
import pickle
from numpy import arange
from trace_signal import *
fichier = open('cloche.pckl','r') # Ouverture du fichier
cloche = pickle.load(fichier) # Chargement des donnees
fichier.close() # Fermeture du fichier
x = cloche['x'][::32] # Redimensionnement des donnees
fe = cloche['fe'] # Definition de la frequence d'echantillonage
t = arange(len(x))/float(fe)
fichier = '../figures/cloche.pdf'
trace_signal(t,x,fichier)
```

Signal expérimental : mesure d'accélération sur une cloche

On obtient le fichier (vectoriel) cloche.pdf



Plan

Les bases

Principe

Échantilloner un signal x consiste à l'évaluer sur une grille comportant N points définie par :

$$t_n = t_{min} + \frac{n}{f_e}, \; n \in \; [0, N[$$

La fréquence f_e est la fréquence d'échantillonage. La durée d'observation du signal notée D est donc obtenue par :

$$D = N/f_e$$

Le signal échantilloné est alors obtenu par :

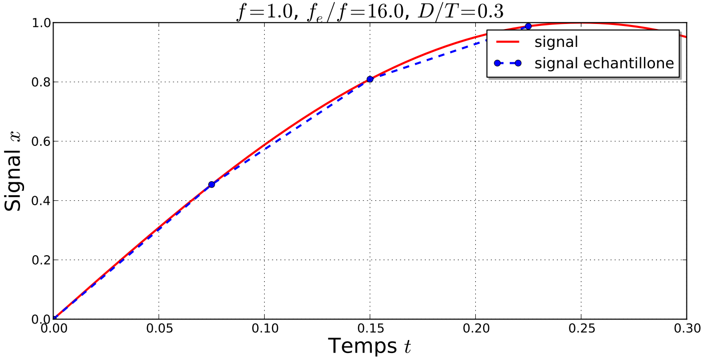
$$x_n = x(t_n)$$

On note $[t_n]$ et $[x_n]$ les vecteurs ainsi obtenus.

Paramètres importants

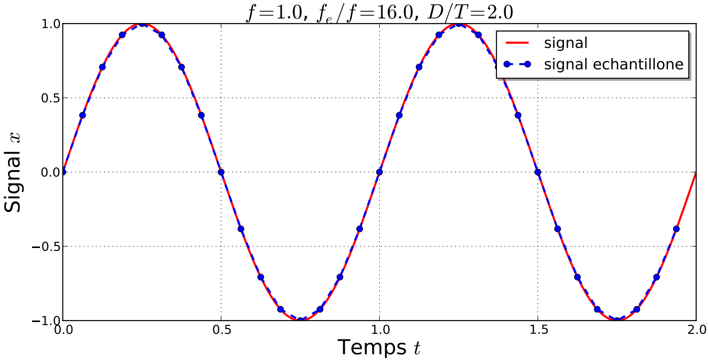
- $\frac{D}{T}$: ajustable en modifiant la durée d'observation D
- $\frac{f_e}{f}$: ajustable en modifiant la fréquence d'échantillonage f_e.

Choix de la durée d'observation (1/2)



Temps d'observation D trop court !

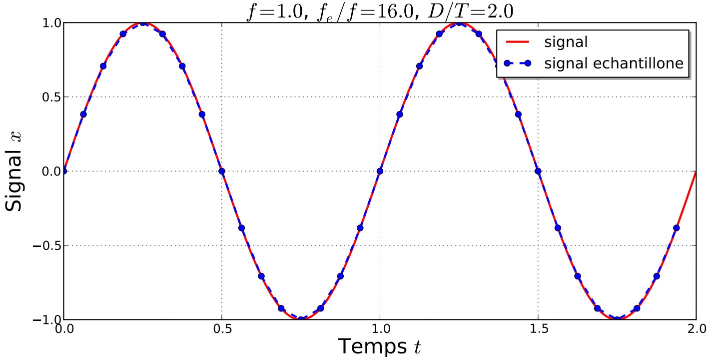
Choix de la durée d'observations (2/2)



Conclusion

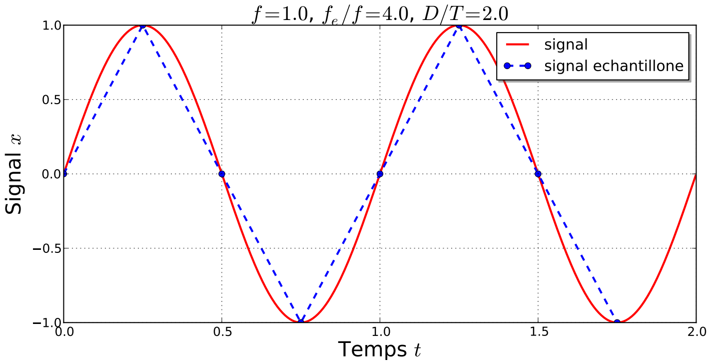
- Il faut que $D/T \geq 1$
- Idéalement, $D/T \in \mathbf{N}$

Une borne basse de la fréquence d'échantillonage ?



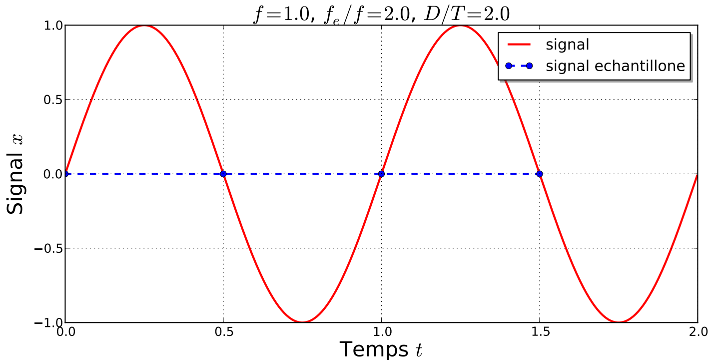
Échantillonage correct.

Une borne basse de la fréquence d'échantillonage ?



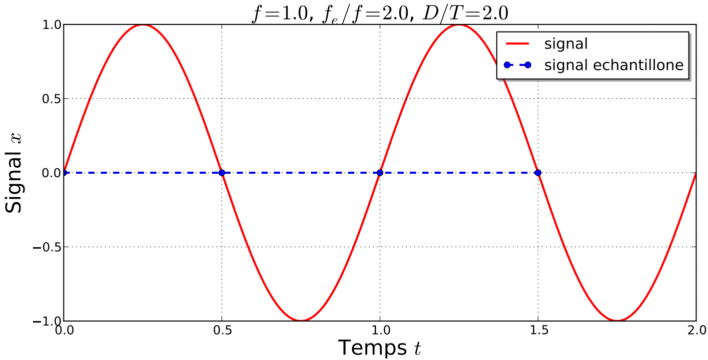
Échantillonage correct.

Une borne basse de la fréquence d'échantillonage ?



Fréquence d'échantillonage trop basse.

Borne basse de la fréquence d'échantillonage ?

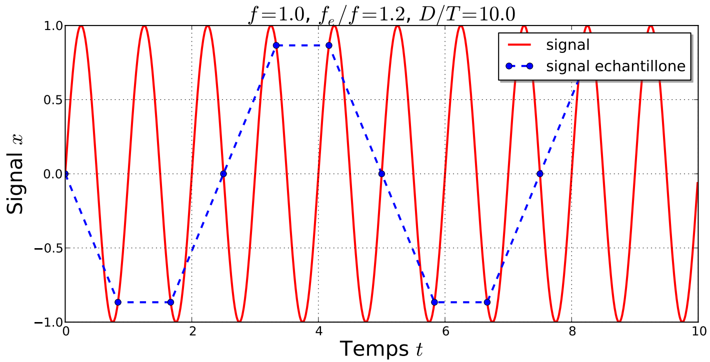


Fréquence d'échantillonage f_e trop basse.

Bilan : le théorème de Shannon-Nyquist

Toute composante du signal dont la fréquence est supérieure ou égale à $f_e/2$ sera perdue lors de l'échantillonage.

Hautes fréquences et aliasing



Hautes fréquences et aliasing

Les fréquences trop hautes vis-à-vis de la fréquence d'échantillonage (*i. e.* $f \geq f_e/2$) sont non seulement perdues mais peuvent produire des artefacts sous la forme de basses fréquences. Il est donc impératif de filtrer le signal préalablement à son échantillonage pour couper toutes les fréquences supérieures à $f_e/2$.

Hautes fréquences et aliasing

Explication mathématique

Intéressons nous aux signaux de fréquence $f^* = f + kf_e$ avec $k \in \mathbb{Z}$. On les échantillonne :

$$\begin{aligned} x_n^* &= \sin(2\pi \frac{f^*}{f_e} n) \\ &= \sin(2\pi \frac{f + kf_e}{f_e} n) \\ &= \sin(2\pi nk + 2\pi \frac{f}{f_e} n) \\ &= \sin(2\pi \frac{f}{f_e} n) \\ &= x_n \end{aligned}$$

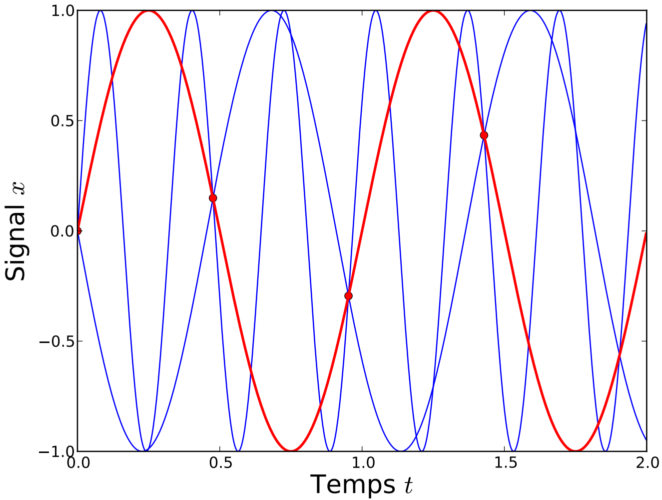
Ce qu'il faut comprendre

Les signaux de fréquence $f^* = f + kf_e$ avec $k \in \mathbb{Z}$ sont indiscernables par échantillonage. Dans le cadre d'une étude expérimentale, il faut donc s'assurer qu'une seule de ces fréquences est présente dans le signal échantilloné.

Mises en pratique (1/2)

```
# listings/exemple_aliasing.py
from matplotlib import pyplot as plt
from math import sin, pi
from signal_sinusoidal import signal_sinusoidal as signal
from numpy import arange, floor
beaucoup = 1000
f = 1. # Frequence du signal
D = 2./f # Duree d'observation
t_min = 0. # Debut du calcul du signal
t_max = t_min+D # Fin du calcul du signal
fe = 2.1*f # Frequence d'echantillonage
N = int(floor(D*fe)) # Nombre de points d'evaluation
plt.figure(0)
plt.clf()
plt.xlabel('Temps $t$', fontsize=20)
plt.ylabel('Signal $x$', fontsize=20)
kmin,kmax =-1, 2
t = arange(beaucoup)/float(beaucoup)*(t_max-t_min)+t_min
for k in xrange(kmin,kmax):
  f1 = f + k*fe
  x = [signal(tt,T=1./f1) for tt in t]
  plt.plot(t,x,'b-',linewidth=1.)
tn = arange(N)/(D*fe)*(t_max-t_min)+t_min
xn = [signal(tt,T=1./f) for tt in tn]
plt.plot(tn,xn,'or')
x = [signal(tt,T=1./f) for tt in t]
plt.plot(t,x,'r-',linewidth=2.)
plt.savefig('../figures/exemple_aliasing.pdf')
```

Mises en pratique (2/2)



Plan

Décomposition des signaux périodiques

Fonctions de base

On peut projeter les signaux de fréquence f sur une base de dimension infinie constituée de fonctions de la forme :

$$f_n(t) = sin(2\pi nft) \text{ et } g_n(t) = \cos(2\pi nft)$$

Décomposition sur la base

Un signal périodique $x(t)$ peut donc s'écrire sous la forme :

$$x(t) = \sum_{n=0}^{+\infty} a_n sin(2\pi nft) + b_n \cos(2\pi nft)$$

Points essentiels

- Connaître $[a_n]$ et $[b_n]$, c'est connaître $x(t)$ en tout point.
- Cette décomposition donne une interprétation fréquentielle de $x(t)$.
- La question est donc de savoir comment calculer analytiquement et numériquement $[a_n]$ et $[b_n]$.

Développement en séries de Fourier

Les grandes lignes

- Les séries de Fourier permettent d'effectuer la projection des signaux périodiques sur la base $[f_n(t), g_n(t)]$ de manière analytique.
- Quand N tend vers l'infini, la somme converge vers le signal $x(t)$.
- Pour les signaux présentant des singularités (triangle, carré), elle converge plus lentement.

Formulation

$$\sum_{n=-N}^{+N} c_n(x) e^{2j\pi nft} \underset{N\to\infty}{\longrightarrow} x(t)$$

Où les coefficients complexes $c_n(x) \in \mathbb{C}$ sont définis par :

$$c_n(x) = \frac{1}{T}\int_0^T x(t)e^{-2j\pi nft}dt \text{ avec : } T = \frac{1}{f}$$

Exercices

Remarques et notations

- $x(t)$, $y(t)$ et $z(t)$ sont des fonctions de période T du temps t.
- α est un nombre réel.
- On pourra introduire la pulsation $\omega = 2\pi f$ pour simplifier les calculs.
- On rappelle que :

$$\cos(a-b) = \cos(a)\cos(b)+\sin(a)\sin(b) \; ; \; \sin(a+b) = \sin(a)\cos(b)+\sin(b)\cos(a)$$

$$\cos(a) = \frac{e^{ja} + e^{-ja}}{2} \; ; \; \sin(a) = \frac{e^{ja} - e^{-ja}}{2j}$$

Pour chaque signal x, trouver l'ensemble des coefficients $c_n(x)$

1. $x(t) = \cos(2\pi ft)$
2. $x(t) = \cos^2(2\pi ft)$
3. $x(t) = \alpha y(t)$
4. $x(t) = y(t) + z(t)$

Exercice : signal sinusoïdal

Réécriture

$$\begin{aligned} x(t) &= sin(2\pi ft) \\ &= \frac{e^{2j\pi ft} - e^{-2j\pi ft}}{2j} \\ &= \frac{j}{2}e^{-2i\pi ft} - \frac{j}{2}e^{2j\pi ft} \end{aligned}$$

Coefficients

$$c_n(x) = \begin{cases} -\frac{j}{2} \text{ si : } n = 1 \\ \frac{j}{2} \text{ si : } n = -1 \\ 0 \; \forall \; n \in \mathbb{Z} - \{1, -1\} \end{cases}$$

Interprétation des coefficients c_n

Comment passer de $[c_n]$ à $[a_n, b_n]$

On note $\Re$ la fonction partie réelle et $\Im$ la fonction partie imaginaire. On remarque que dans le cas des signaux à valeurs réels (ce qui est majoritairement le cas en physique) :

$$c_{-n} = \Re(c_n) - j\Im(c_n) = \overline{c}_n$$

Les coefficients associés à $n < 0$ sont donc inutiles car conjugués de coefficients obtenus pour $n >= 0$. On remarque aussi que les coeffients a_n et b_n peuvent être calculés pour $n \geq 0$:

$$a_n = 2\Re(c_n)$$
$$b_n = -2\Im(c_n)$$

Exercice : signal carré

Calculs

$$x(t) = \begin{cases} -1 \text{ si } : t \in]0, T/2[\\ 1 \text{ si } : t \in]T/2, T[\end{cases}$$

Donc :

$$\begin{aligned} c_n(x) &= \frac{1}{T} \int_0^T x(t) e^{-2j\pi nft} dt \\ &= -\frac{1}{T} \int_0^{T/2} e^{-2j\pi nft} dt + \frac{1}{T} \int_{T/2}^{T} e^{-2j\pi nft} dt \\ &= \frac{1}{T} \left(\left[\frac{-j}{2\pi nf} e^{-2j\pi nft} \right]_0^{T/2} + \left[\frac{j}{2\pi nf} e^{-2j\pi nft} \right]_{T/2}^{T} \right) \end{aligned}$$

Coefficients

$$c_n(x) = -j \frac{2}{\pi n} \text{ avec : } n = 2k+1 \text{ et : } k \in \mathbb{Z}$$

Exercice : signal carré

Mise en pratique : voyons si ça marche

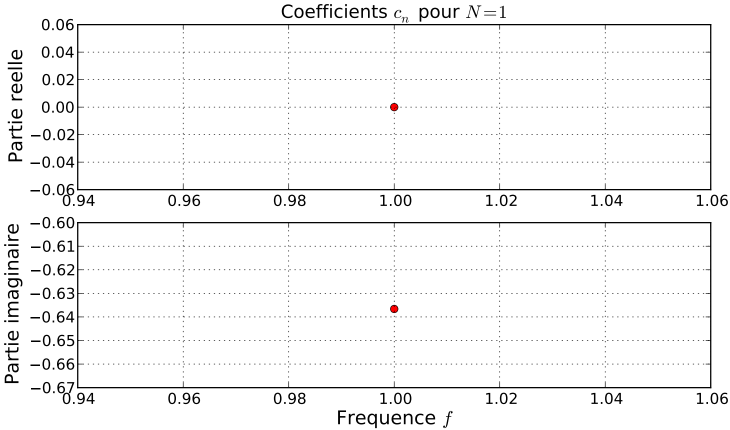
```
# listings/serieF_carre.py
from cmath import exp
from numpy import arange, array
from math import pi
from matplotlib import pyplot as plt
from trace_complexes import *
N = 1 # Nombre de coefficients calcules
for N in [1,2,4,8,16,32,64,128,256,1024]: # Valeurs de N
  # Indices n impairs
  n = [2*k+1 for k in range(-N,0)]+[2*k+1 for k in range(N)]
  c = [-2j/(pi*nn) for nn in n] # Coefficients c
  T,beaucoup= 1., 2000
  t,f = arange(beaucoup)/float(beaucoup)*T, array(n[N:2*N])/T
  trace_complexes(f,c[N:2*N],'../figures/serieF_carre_c_{0}.pdf'.format(
      N),title='Coefficients $c_n$ pour $N = {0}$'.format(N))
  x = []
  for tt in t:
    x.append(0.)
    for i in xrange(len(n)):
      x[-1] = x[-1]+c[i]*exp(2j*pi*n[i]*tt/T)
  x1 = [xx.real for xx in x]
  plt.figure(0,figsize=(10,2))
  plt.clf()
  plt.plot(t,x1)
  plt.title('$N={0}$'.format(N))
  plt.savefig('../figures/serieF_carre_{0}.pdf'.format(N))
```

Exercice : signal carré

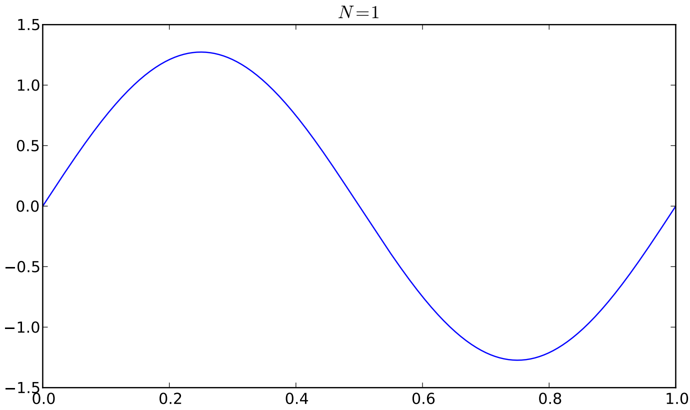
Programme de tracé de vecteurs complexes

```python
# trace_complexes.py
import matplotlib.pyplot as plt

def trace_complexes(x,y,fichier,xlabel='Frequence $f$', title='', style=
        'ro'):
    plt.figure(0,figsize=(9,5))
    plt.clf()
    plt.grid(True)
    p0 = plt.subplot(2,1,1)
    p0.set_title(title)
    p0.grid()
    p0.plot(x,[yy.real for yy in y],style,linewidth=2.0)
    p0.set_ylabel('Partie reelle', fontsize=15)
    p1 = plt.subplot(2,1,2)
    p1.grid()
    p1.plot(x,[yy.imag for yy in y],style,linewidth=2.0)
    p1.set_xlabel(xlabel, fontsize=15)
    p1.set_ylabel('Partie imaginaire', fontsize=15)
    plt.savefig(fichier)
```

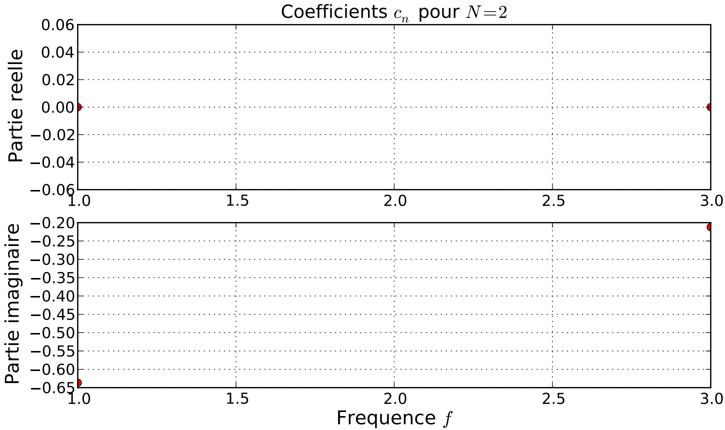
Exercice : signal carré



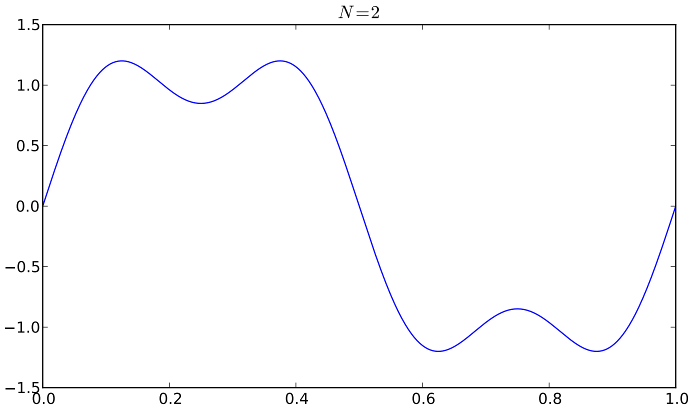
Exercice : signal carré



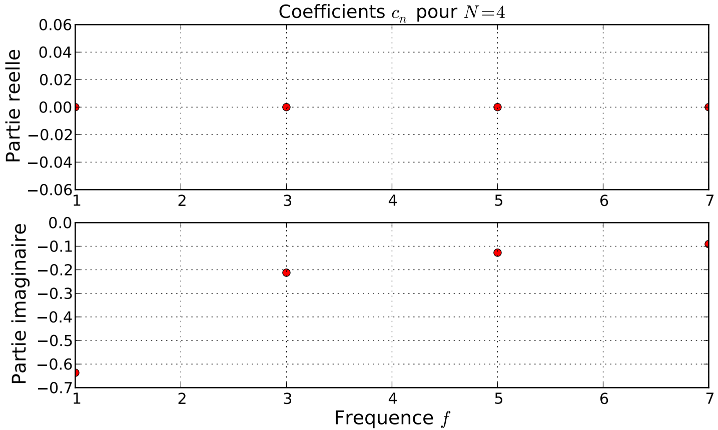
Exercice : signal carré



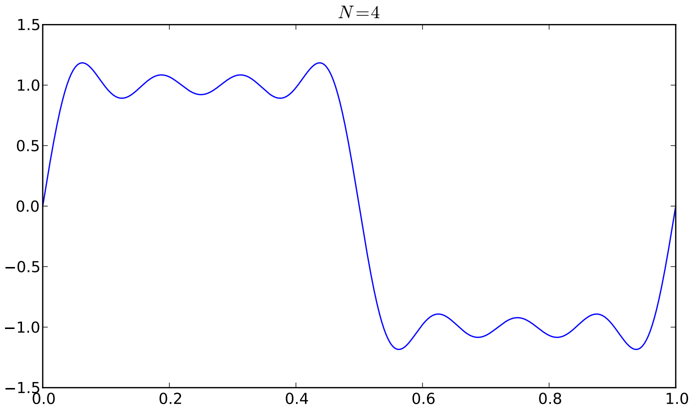
Exercice : signal carré



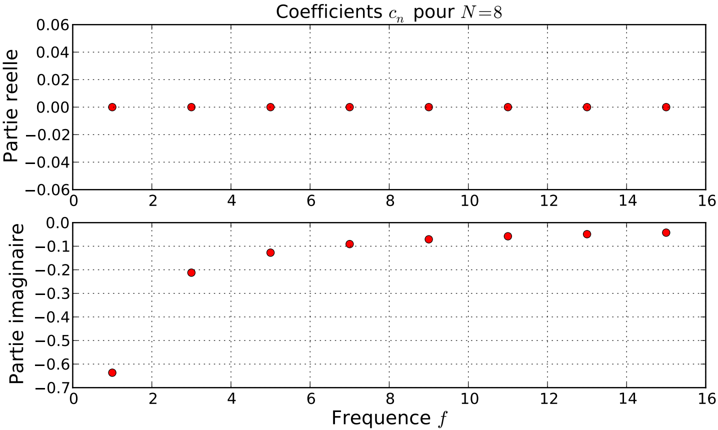
Exercice : signal carré



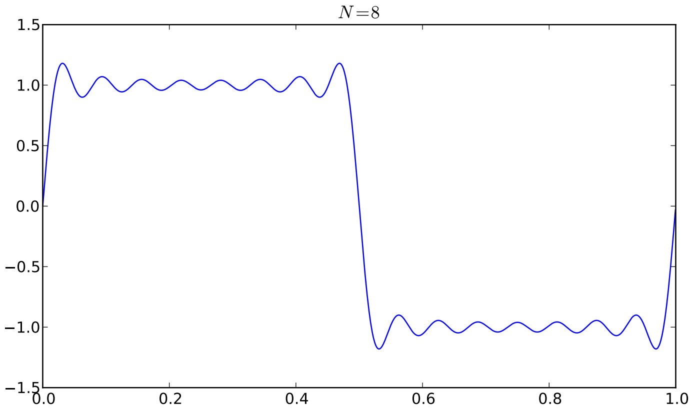
Exercice : signal carré



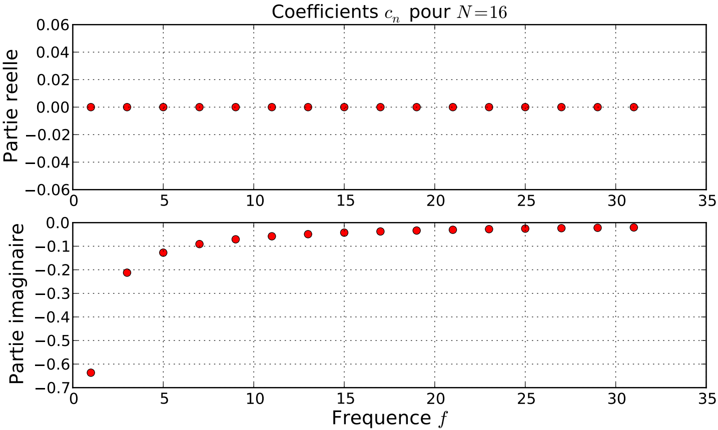
Exercice : signal carré



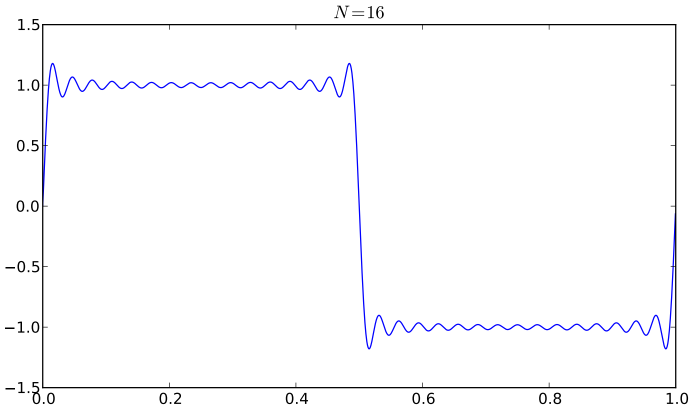
Exercice : signal carré



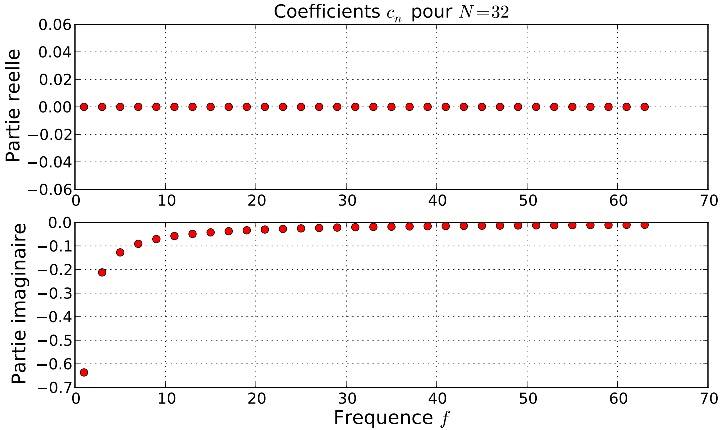
Exercice : signal carré



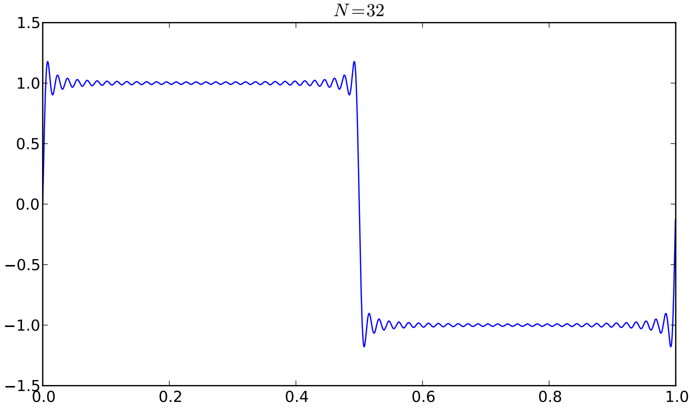
Exercice : signal carré



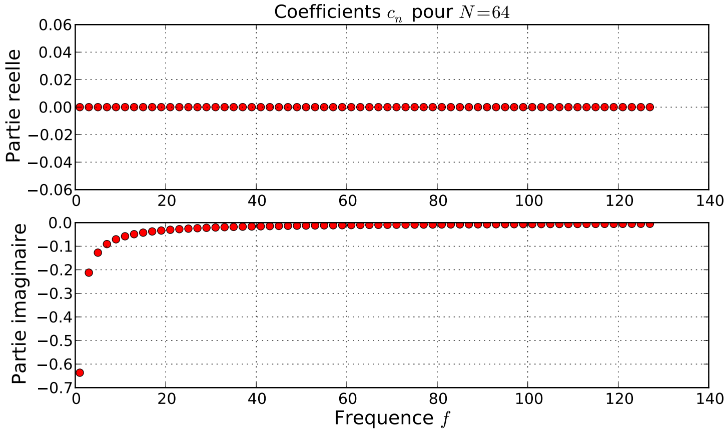
Exercice : signal carré



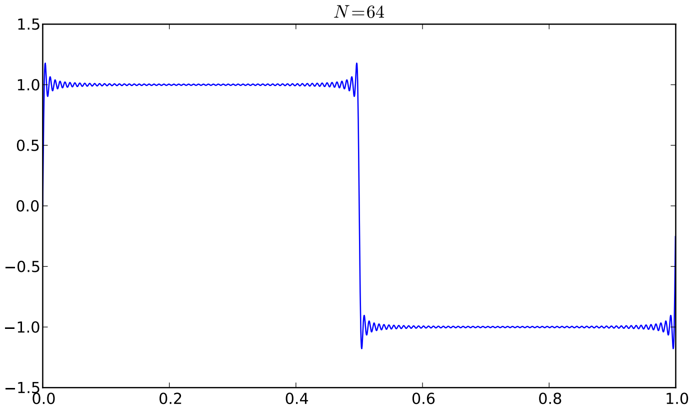
Exercice : signal carré



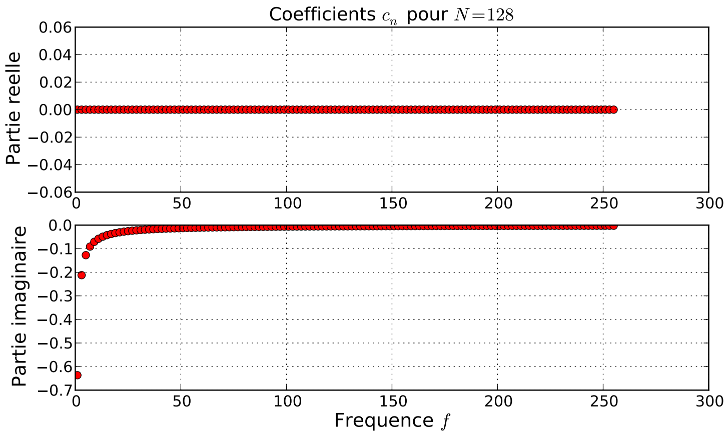
Exercice : signal carré



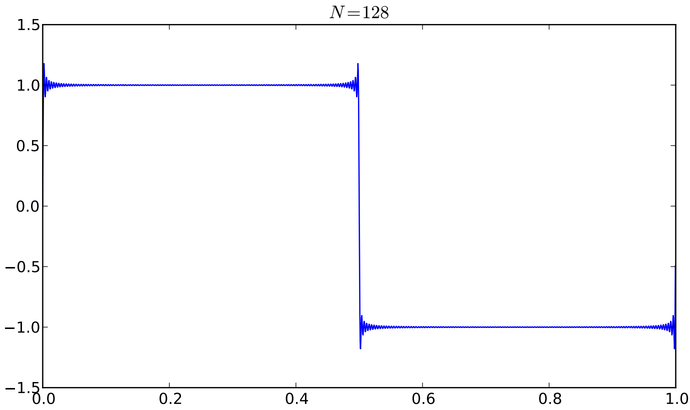
Exercice : signal carré



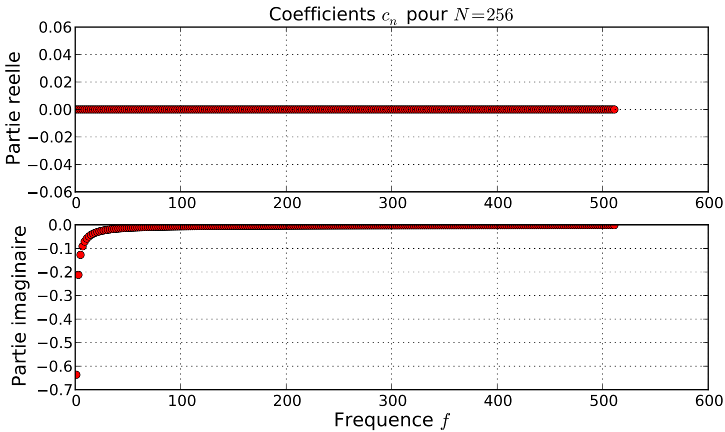
Exercice : signal carré



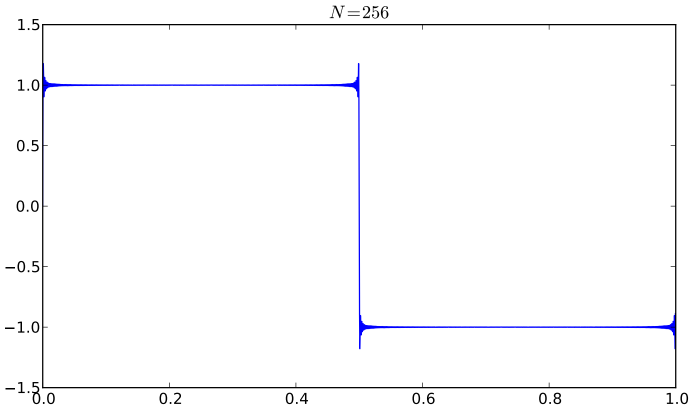
Exercice : signal carré



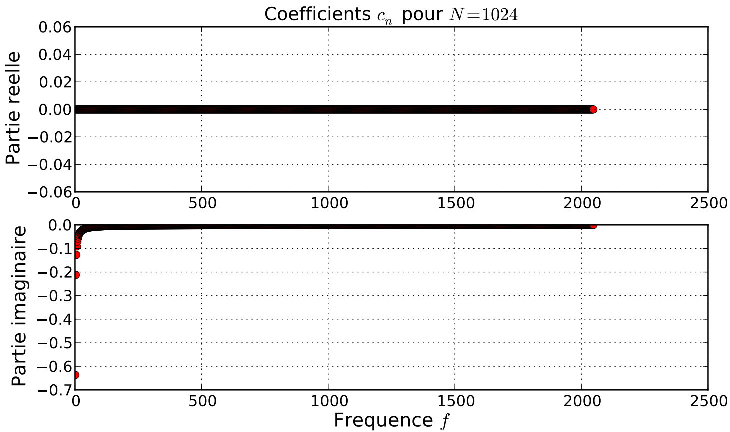
Exercice : signal carré



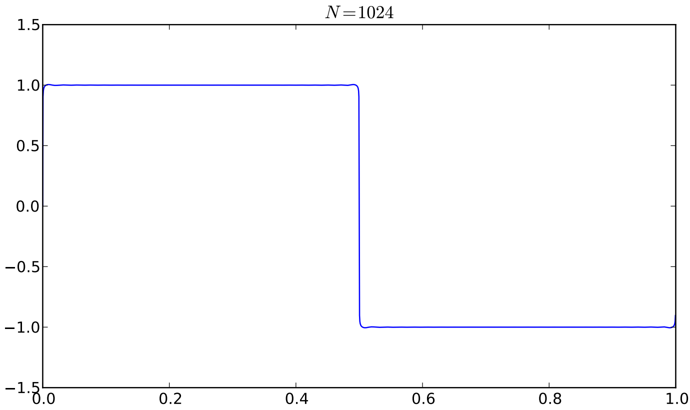
Exercice : signal carré



Exercice : signal carré



Exercice : signal carré



Transformées de Fourier

Formulation

$$x(t) \xrightarrow{\mathcal{F}} X(f) = \int_{-\infty}^{+\infty} x(t)e^{-2j\pi ft}dt$$

$$X(f) \xrightarrow{\mathcal{F}^{-1}} x(t) = \int_{-\infty}^{+\infty} X(f)e^{2j\pi ft}df$$

Points clés

- $\mathcal{F}$ est la tranformée de Fourier et $\mathcal{F}^{-1}$ la transformée de Fourier inverse.
- $\mathcal{F}$ s'applique à tous les signaux, même apériodiques.
- $X(f)$ est le spectre de $x(t)$.
- Un signal apériodique possède un spectre continu.
- Un signal périodique possède un spectre discret.

La Transformée de Fourier Discrète ou DFT

Formulation

On considère un signal échantilloné $[x_n]$ comportant N échantillons. Sa transformée de Fourier discrètre $[X_k]$ s'écrit :

$$[x_n] \overset{\mathcal{DFT}}{\longmapsto} [X_k] = \sum_{n=0}^{n=N-1} x_n e^{-2j\pi \frac{kn}{N}}$$

$$[X_k] \overset{\mathcal{DFT}^{-1}}{\longmapsto} [x_n] = \frac{1}{N} \sum_{k=0}^{k=N-1} X_k e^{2j\pi \frac{kn}{N}}$$

Interprétation de $[X_k]$

Le vecteur X_k représente le spectre discret de $[x_n]$. Chaque coefficient X_k est associé à une fréquence f_k obtenue par :

$$f_k = k/D = kf_e/N$$

La Transformée de Fourier Discrète ou DFT

Interprétation de $[X_k]$

Dans le cas ou le signal $x(t)$ est réel, les coefficients X_k pour $k > N/2$ sont les conjugués des coefficients d'indice $k < N/2$. On peut donc se contenter d'interprêter les $N/2$ premiers coefficients.

Liens entre $[X_k]$ et $[a_k, b_k]$

Les coefficients a_k et b_k peuvent être déterminés par :

$$a_k = \frac{2}{N}\Re(X_k)$$

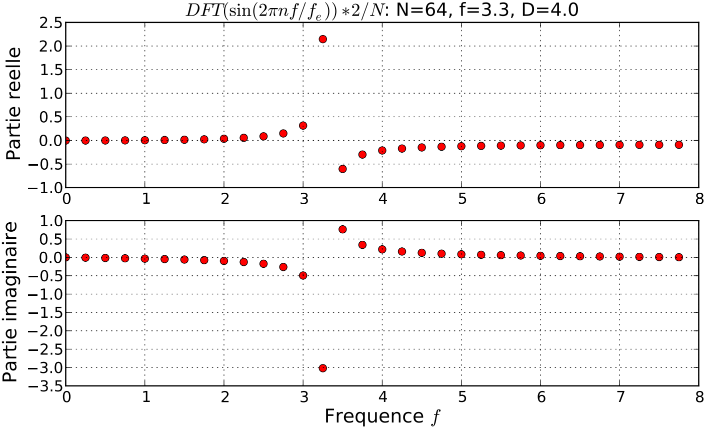
$$b_k = -\frac{2}{N}\Im(X_k)$$

Mise en pratique : calcul de la DFT

Programme de calcul de la DFT

```python
# listings/exemple_DFT.py
from signal_sinusoidal import *
from signal_carre import *
from trace_complexes import *
from numpy import arange, floor
from cmath import exp
signal = signal_sinusoidal #signal = signal_carre
beaucoup, rien = 1000, 1.e-10
f = 3.3 # Frequence du signal
D = 4. # Duree d'observation
t_min = 0. # Debut du calcul du signal
t_max = t_min+D # Fin du calcul du signal
fe = 16. # Frequence d'echantillonage
N = int(floor(D*fe)) # Nombre de points d'evaluation
tn = arange(N)/(D*fe)*(t_max-t_min)+t_min # Discretisation du temps
xn = [signal(tt,T=1./f, k=4.) for tt in tn] # Discretisation du signal
Xk = [] # DFT de xn
for k in range(N): # Boucle sur k
  Xk.append(0.)
  for n in range(N): # Boucle sur n
    Xk[-1] = Xk[-1] + xn[n]*exp(-2j*pi*n*k/N) # Calcul de Xk
  if abs(Xk[-1].real) < rien : Xk[-1] = Xk[-1] - Xk[-1].real
  if abs(Xk[-1].imag) < rien : Xk[-1] = Xk[-1] - 1j*Xk[-1].imag
  Xk[-1] = Xk[-1]*2/N
fk = arange(N)/D # Discretisation des frequences
tit='$DFT(\sin(2\pi nf/f_e))*2/N$: N={0}, f={1}, D={2}'.format(N,f,D)
trace_complexes(fk[:N/2],Xk[:N/2],'../figures/DFT_sinus.pdf',title=tit)
```

Vérification la DFT sur un signal sinusoïdal



On retrouve donc bien la série de Fourier avec le coefficient $2/N$.

La FFT, pourquoi ? Comment ?

Pourquoi ?

- Le calcul direct de la DFT demande de l'ordre de N^2 opérations alors que des algorithmes optimisés dits FFT demandent $N \times \log N$ opérations. Le gain de temps est très significatif quand N est grand.
- L'utilisation de langages rapides (C, Fortran) dans le module FFTPack disponible dans Scipy permet typiquement d'augmenter d'un facteur 400 la vitesse d'execution par rapport Python.

Comment ?

- Restrictions de la FFT : N doit être une puissance de 2.
- Utilisation :

```
# listings/exemple_FFT.py
from math import pi, sin
from scipy.fftpack import fft
N = 8
xn = [sin(2*pi*n/N) for n in xrange(N)]
Xk = fft(xn)
```

Renvoie $X_n = [0, -2j, 0, 2j]$ ce qui est identique au résultat obtenu par DFT. On utilisera donc préférentiellement la FFT pour des raisons de commodité et vitesse.

FFT : effet de la fréquence f

Programme

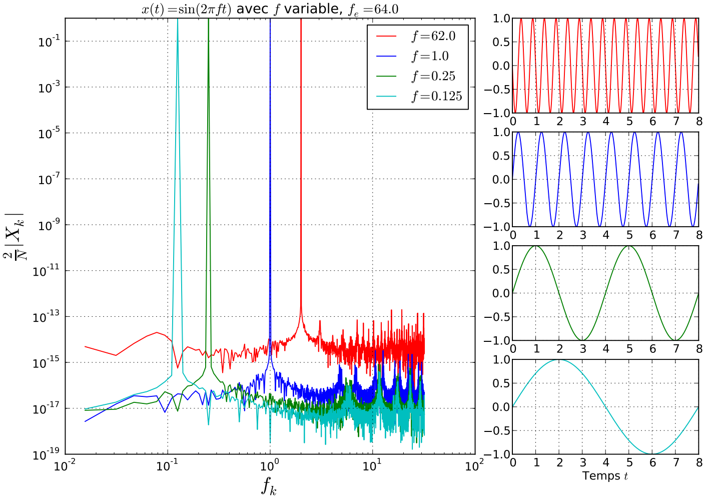
```
# listings/exemple_FFT_frequence.py
from math import pi,sin, exp
from scipy.fftpack import fft
from random import gauss
from numpy import array, arange, floor
from matplotlib import pyplot as plt
import matplotlib.gridspec as gridspec
from signal_sinusoidal import *
beaucoup = 1000
fe = 64. # Frequence d'echantillonage
N = 4096 # Nombre de points d'echantillonage
D = N/fe # Duree d'observation
f = 8./D # Frequence du signal
t_min = 0. # Debut du calcul du signal
t_max = t_min+D # Fin du calcul du signal
stddev = 0. # Ecart type du bruit
nom = '../figures/FFT_frequence.pdf'
amort = 0.
val = [fe-2, 64./D,16./D,8./D] # Frequence
lab = '$f={0}$'
tn = arange(N)/(D*fe)*D+t_min
for i in xrange(N):
  if tn[i]<=1./f: i_t = i
xn = [sin(2*pi*f*t) for t in tn]
fk = arange(N)/D # Discretisation des frequences
plt.figure(0,figsize=(12,8))
plt.clf()
gs = gridspec.GridSpec(4, 3) # Grille de zone de trace
```

FFT : effet de la fréquence f

Programme

```
p0 = plt.subplot(gs[:,:2])
p0.set_title(r'$x(t) = \sin(2 \pi f t)$ avec $f$ variable, $f_e={}$'.
    format(fe))
p0.grid()
p0.set_xlabel(r'$f_k$', fontsize=20)
p0.set_ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
p0.set_xscale('log')
p0.set_yscale('log')
for z in xrange(len(val)):
  v = val[z]
  #stddev = v
  f = v
  xn = [sin(2*pi*f*t) for t in tn]
  v_amort = array([exp(-t*amort) for t in tn])
  color = ['r','b','g','c'][z]
  bruit = array([gauss(0,stddev) for i in xrange(N)])
  xxn = (xn + bruit)*v_amort
  Xk = abs(fft(xxn))*2./N
  p0.plot(fk[0:N/2],Xk[0:N/2],'-'+color, label = lab.format(v))
  p0.legend()
  p1 = plt.subplot(gs[z,-1])
  p1.grid()
  p1.plot(tn[:i_t],xxn[:i_t],'-'+color)
p1.set_xlabel('Temps $t$')
plt.savefig(nom)
```

FFT : effet de la fréquence f



Changer la fréquence, c'est translater horizontalement le pic.

FFT : effet de la durée d'observation D

Programme

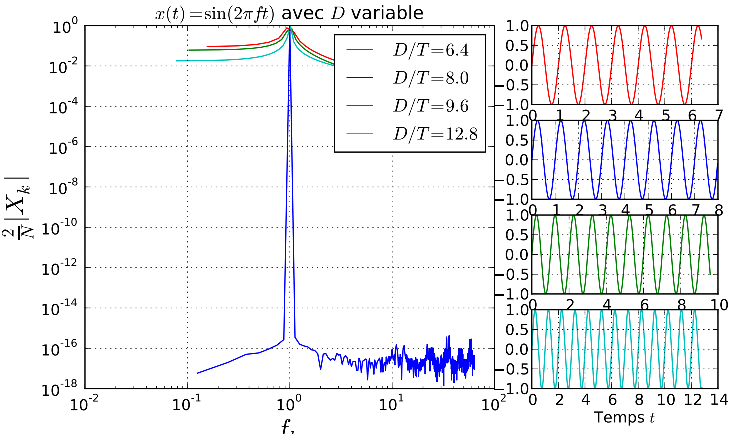
```
# listings/exemple_FFT_D.py
from math import pi,sin, exp
from scipy.fftpack import fft
from random import gauss
from numpy import array, arange, floor, hanning
from matplotlib import pyplot as plt
import matplotlib.gridspec as gridspec
from signal_sinusoidal import *
beaucoup = 1000
fe = 128. # Frequence d'echantillonage
f = 1. # Frequence du signal
t_min = 0. # Debut du calcul du signal
stddev = 0. # Ecart type du bruit
nom = '../figures/FFT_D.pdf'
amort = 0.
val = 8./f*array([.8,1.,1.2,1.6])
lab = '$D/T={0}$'


plt.figure(0,figsize=(12,8))
plt.clf()
gs = gridspec.GridSpec(4, 3) # Grille de zone de trace
p0 = plt.subplot(gs[:,:2])
p0.set_title(r'$x(t) = \sin(2 \pi f t)$ avec $D$ variable')
p0.grid()
p0.set_xlabel(r'$f_k$', fontsize=20)
p0.set_ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
p0.set_xscale('log')
```

FFT : effet de la durée d'observation D

Programme

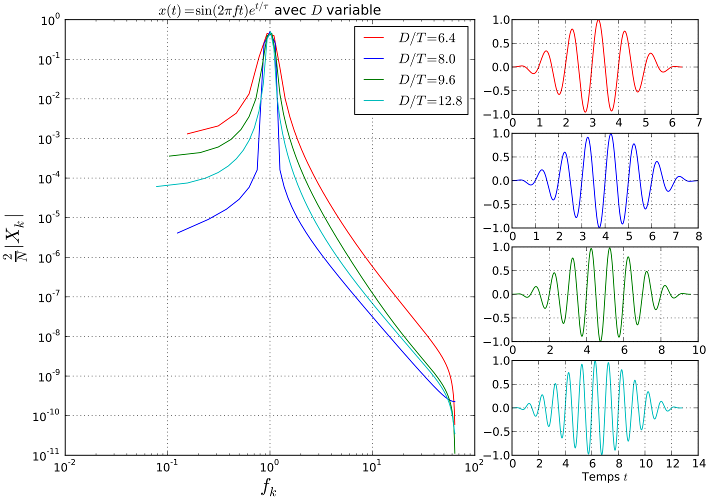
```
p0.set_yscale('log')
for z in xrange(len(val)):
  v = val[z]
  D = v
  i_t=-1
  N = int(D*fe)
  fk = arange(N)/D # Discretisation des frequences
  t_max = t_min+D # Fin du calcul du signal
  tn = arange(N)/float(N)*D+t_min
  xn = [sin(2*pi*f*t) for t in tn]
  v_amort = array([exp(-t*amort) for t in tn])
  color = ['r','b','g','c'][z]
  bruit = array([gauss(0,stddev) for i in xrange(N)])
  xxn = (xn + bruit)*v_amort #*hanning(N)
  Xk = abs(fft(xxn))*2./N
  p0.plot(fk[0:N/2],Xk[0:N/2],'-'+color, label = lab.format(v*f))
  p1 = plt.subplot(gs[z,-1])
  p1.grid()
  p1.plot(tn[:i_t],xxn[:i_t],'-'+color)
p0.legend()
p1.set_xlabel('Temps $t$')
plt.savefig(nom)
```

FFT : effet de la durée d'observation D



Lorque D n'est pas multiple de la période T, la hauteur du pic est réduite.

FFT : effet de la durée d'observation D



Le fenetrage temporel de Hann permet d'augmenter la hauteur du pic.

FFT : effet du bruit

Programme

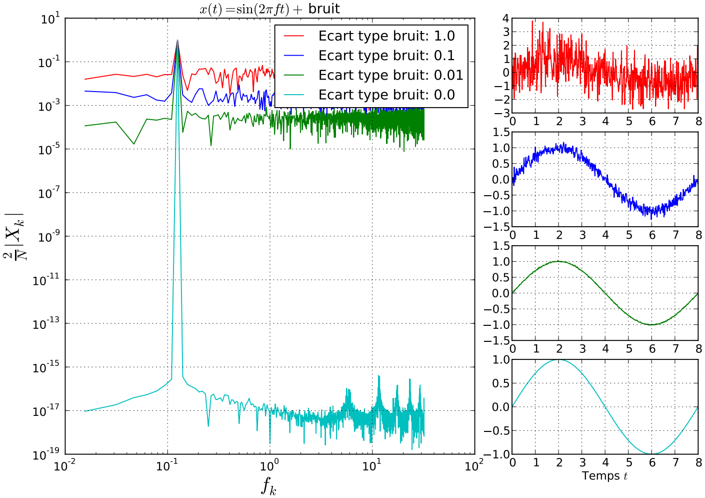
```
# listings/exemple_FFT_bruit.py
from math import pi,sin, exp
from scipy.fftpack import fft
from random import gauss
from numpy import array, arange, floor
from matplotlib import pyplot as plt
import matplotlib.gridspec as gridspec
from signal_sinusoidal import *
beaucoup = 1000
fe = 64. # Frequence d'echantillonage
N = 4096 # Nombre de points d'echantillonage
D = N/fe # Duree d'observation
f = 8./D # Frequence du signal
t_min = 0. # Debut du calcul du signal
t_max = t_min+D # Fin du calcul du signal
stddev = 0. # Ecart type du bruit
nom = '../figures/FFT_bruit.pdf'
amort = 0.
val = [1.,1.e-1,1.e-2,0.] # Bruit
lab = 'Ecart type bruit: {0}'
tn = arange(N)/(D*fe)*D+t_min
for i in xrange(N):
  if tn[i]<=1./f: i_t = i
xn = [sin(2*pi*f*t) for t in tn]
fk = arange(N)/D # Discretisation des frequences
plt.figure(0,figsize=(12,8))
plt.clf()
gs = gridspec.GridSpec(4, 3) # Grille de zone de trace
```

FFT : effet du bruit

Programme

```
p0 = plt.subplot(gs[:,:2])
p0.set_title(r'$x(t) = \sin(2 \pi f t)+$ bruit')
p0.grid()
p0.set_xlabel(r'$f_k$', fontsize=20)
p0.set_ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
p0.set_xscale('log')
p0.set_yscale('log')
for z in xrange(len(val)):
  v = val[z]
  stddev = v
  v_amort = array([exp(-t*amort) for t in tn])
  color = ['r','b','g','c'][z]
  bruit = array([gauss(0,stddev) for i in xrange(N)])
  xxn = (xn + bruit)*v_amort
  Xk = abs(fft(xxn))*2./N
  p0.plot(fk[0:N/2],Xk[0:N/2],'-'+color, label = lab.format(v))
  p0.legend()
  p1 = plt.subplot(gs[z,-1])
  p1.grid()
  p1.plot(tn[:i_t],xxn[:i_t],'-'+color)
p1.set_xlabel('Temps $t$')
plt.savefig(nom)
```

FFT : effet du bruit



Le bruit réduit la hauteur du pic.

FFT : effet de l'amortissement

Programme

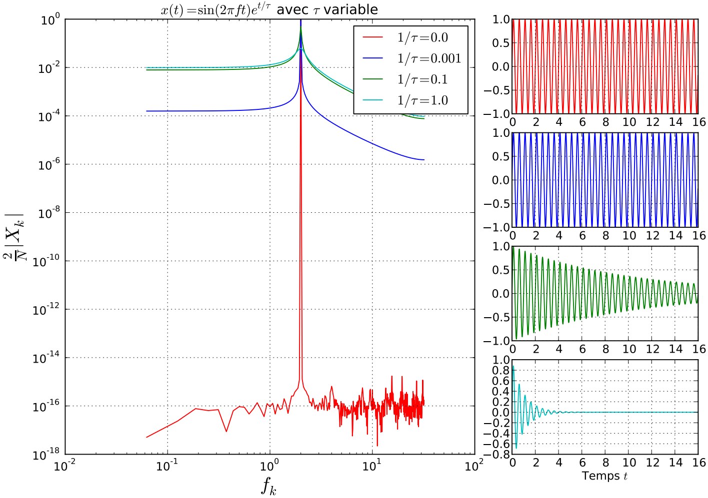
```
# listings/exemple_FFT_amortissement.py
from math import pi, sin, exp
from scipy.fftpack import fft
from random import gauss
from numpy import array, arange, floor, hanning
from matplotlib import pyplot as plt
import matplotlib.gridspec as gridspec
from signal_sinusoidal import *
beaucoup = 1000
fe = 64. # Frequence d'echantillonage
N = 1024 # Nombre de points d'echantillonage
D = N/fe # Duree d'observation
f = 32./D # Frequence du signal
t_min = 0. # Debut du calcul du signal
t_max = t_min+D # Fin du calcul du signal
stddev = 0. # Ecart type du bruit
nom = '../figures/FFT_amortissement-hann.pdf'
amort = 0.
val = [0.,.001,.1,1.] # Amortissement
lab = r'$1/\tau = {0}$'
tn = arange(N)/(D*fe)*D+t_min
xn = [sin(2*pi*f*t) for t in tn]
fk = arange(N)/D # Discretisation des frequences
plt.figure(0,figsize=(12,8))
plt.clf()
gs = gridspec.GridSpec(4, 3) # Grille de zone de trace
p0 = plt.subplot(gs[:,:2])
p0.set_title(r'$x(t) = \sin(2 \pi f t)e^{t/\tau}$ avec $\tau$ variable +
```

FFT : effet de l'amortissement

Programme

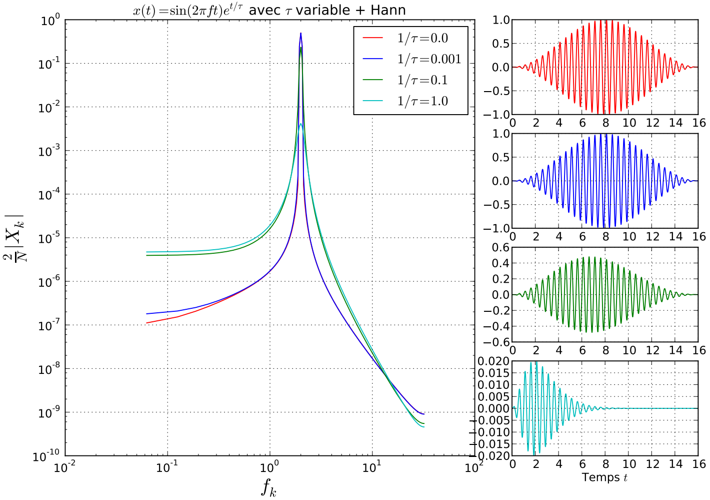
```
p0.grid()
p0.set_xlabel(r'$f_k$', fontsize=20)
p0.set_ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
p0.set_xscale('log')
p0.set_yscale('log')

for z in xrange(len(val)):
  v = val[z]
  amort = v
  v_amort = array([exp(-t*amort) for t in tn])
  color = ['r','b','g','c'][z]
  bruit = array([gauss(0,stddev) for i in xrange(N)])
  xxn = (xn + bruit)*v_amort *hanning(N)
  Xk = abs(fft(xxn))*2./N
  p0.plot(fk[0:N/2],Xk[0:N/2],'-'+color, label = lab.format(v))
  p0.legend()
  p1 = plt.subplot(gs[z,-1])
  p1.grid()
  p1.plot(tn[:],xxn[:],'-'+color)
p1.set_xlabel('Temps $t$')
plt.savefig(nom)
```

FFT : effet de l'amortissement



L'amortissement entraine une perte de hauteur du pic.

FFT : amortissement et fenetrage de Hann

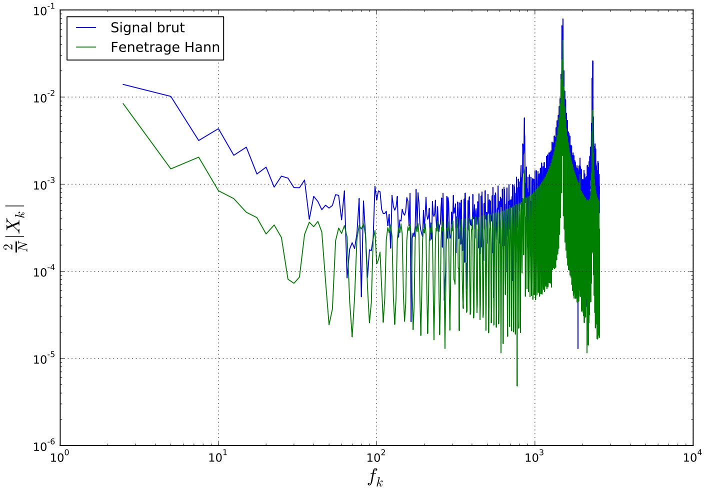


Le fenêtrage de Hann (méthode dite hanning) induit une hauteur de pic $\times 1000$! !

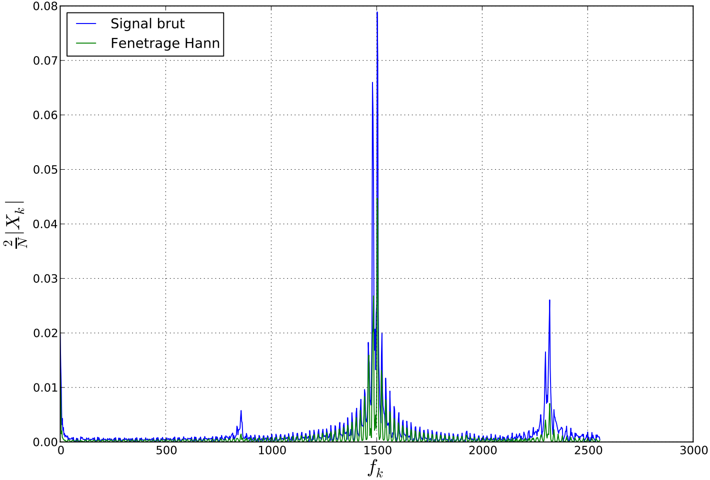
Application à la cloche

```python
# listings/exemple_FFT_cloche.py
from scipy.fftpack import fft
from numpy import array, arange, hanning
from matplotlib import pyplot as plt
import pickle
nom = '../figures/FFT_cloche-log.pdf'
fichier = open('cloche.pckl','r') # Ouverture du fichier
cloche = pickle.load(fichier) # Chargement des donnees
fichier.close() # Fermeture du fichier
xn = cloche['x'][::32] # Redimensionnement des donnees
fe = float(cloche['fe']) # Definition de la frequence d'echantillonage
tn = arange(len(xn))/float(fe)
N = len(tn)
D = N/fe
plt.figure(0,figsize=(12,8))
plt.clf()
fk = arange(N)/D # Discretisation des frequences
Xk = abs(fft(xn))*2./N
Xkh = abs(fft(xn*hanning(N)))*2./N
plt.plot(fk[0:N/2],Xk[0:N/2],'-', label='Signal brut')
plt.plot(fk[0:N/2],Xkh[0:N/2],'-', label='Fenetrage Hann')
plt.xlabel(r'$f_k$', fontsize=20)
plt.ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
plt.xscale('log')
plt.yscale('log')
plt.grid(True)
plt.legend(loc='upper left')
plt.savefig(nom)
```

Application à la cloche

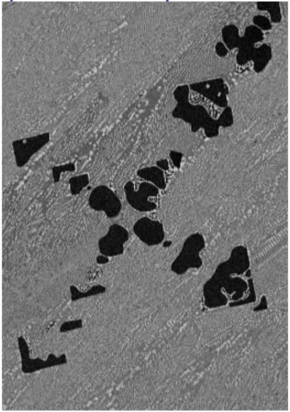


Application à la cloche



Plan

Séparation des phases dans un alliage SiNb



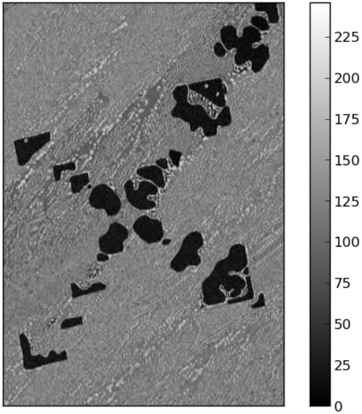
Image

- 260 x 372 pixels
- 8 bits (256 niveaux de gris).
- Nb_3Si en gris clair et Nb_3Si_5 en noir.
- Source : Onera

Objectif

- Isoler les deux phases.
- Calculer la proportion de chaque phase.
- Calculer la taille moyenne de chaque dendrite de Nb_3Si_5

Ouverture d'une image

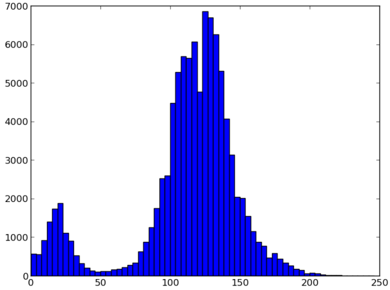



```python
# image.py
# Opening image file
from PIL import Image
from matplotlib import pyplot as plt
import numpy as np
from scipy import ndimage
import matplotlib.cm as cm
im = Image.open("SiNb.bmp")
z = np.array(im) # image to array
figdir = ''
plt.figure(0)
plt.clf()
grad = plt.imshow(z, cmap = cm.gray)
plt.colorbar(grad)
plt.xticks([])
plt.yticks([])
plt.savefig(figdir+'image_original.png')
```

Questions

Comment séparer les deux phases ?

L'histogramme

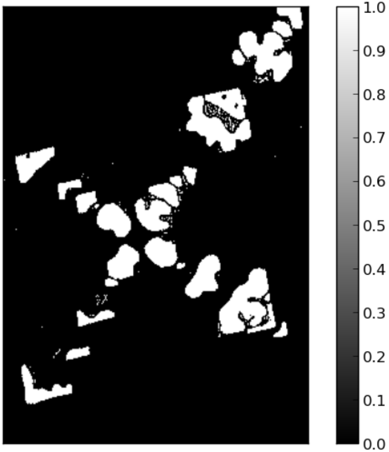



```python
# Trace de l'histogramme
n_classes = 64 # Nombre de classes
plt.figure(1)
plt.clf()
plt.hist(z.flatten(), bins=n_classes)
plt.savefig(figdir+'image_hist.png')
```

Questions

L'histogramme nous informe-t-il sur les niveaux correspondant aux différentes couleurs ?

Seuillage

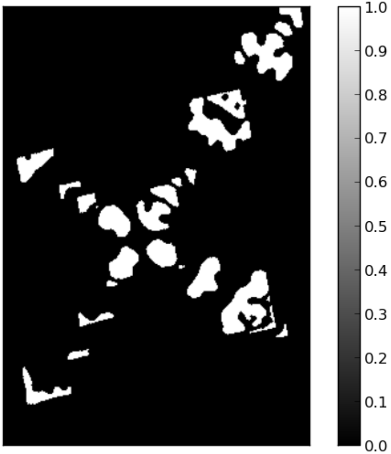



```
# Seuillage de l'image
zs = z<50
plt.figure(2)
plt.clf()
grad = plt.imshow(zs,cmap = cm.gray)
plt.colorbar(grad)
plt.xticks([])
plt.yticks([])
plt.savefig(figdir+'image_threshold.
    png')
```

Questions

L'image seuillée permet-elle de séparer proprement les deux phases ?

Érosion

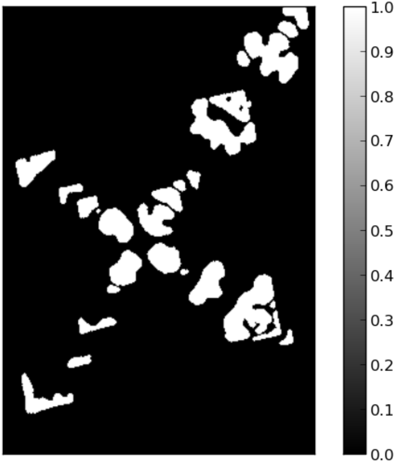



```
# Erosion
n_ero = 10 # Nombre de passes
ze = zs
for i in xrange(n_ero):
  ze = ndimage.binary_erosion(ze)
plt.figure(3)
plt.clf()
grad = plt.imshow(ze,cmap = cm.gray)
plt.colorbar(grad)
plt.xticks([])
plt.yticks([])
plt.savefig(figdir+'image_erosion.
    png')
```

Questions

L'érosion a-t-elle modifié la proportion des deux phases ? comment contrer cet effet ?

Dilatation

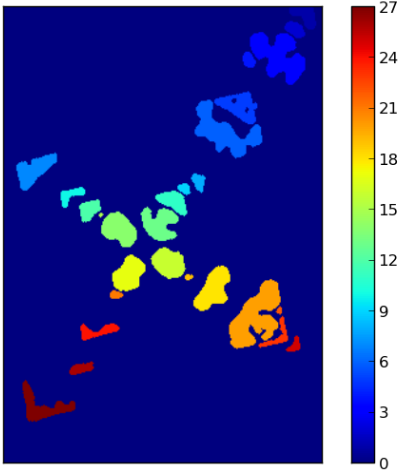



```python
# Dilatation
for i in xrange(n_ero):
  ze = ndimage.binary_dilation(ze)
plt.figure(4)
plt.clf()
grad = plt.imshow(ze,cmap = cm.gray)
plt.colorbar(grad)
plt.xticks([])
plt.yticks([])
plt.savefig(figdir+'image_dilation.png')
```

Questions

Comment séparer les dendrites ?

Labélisation




```
# Labeling
zl, nombre = ndimage.label(ze)
plt.figure()
plt.clf()
grad = plt.imshow(zl)
plt.colorbar(grad)
plt.xticks([])
plt.yticks([])
plt.savefig(figdir+'image_labels.png')
```

Questions

Comment compter les dendrites ?

Comptage

Résultats

- Proportion de dendrites : 9.6 %
- Surface moyenne : 358 $px \approx$ 128 μm^2

```
# Comptage
surface_dendrite = float(ze.sum())
surface_totale = float(np.size(ze))
taille_moyenne = surface_dendrite / nombre
proportion = surface_dendrite / surface_totale
```

Conclusion

- Outils de traitement quantitatif des images.
- Applications très vastes.

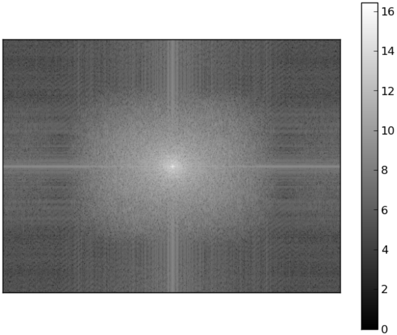
Analyse fréquentielle


```python
from scipy.fftpack import fft2, ifft2, fftshift, ifftshift
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.cm as cm

# Ouverture image
im = Image.open("houle.png")
z = np.array(im) # image to array

# Image originale
fig = plt.figure(0)
plt.clf()
plt.xticks([])
plt.yticks([])
grad = plt.imshow(z, cmap = cm.gray)
plt.colorbar(grad)
plt.savefig(figdir+'lapin_original.png')
```

Questions

Comment utiliser l'analyse fréquentielle pour modifier une image ?

FFT 2D

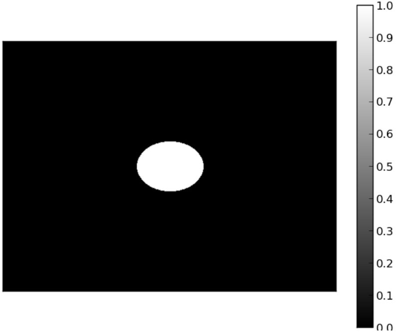



```
# FFT 2D
Z = fftshift(fft2(z))
fig = plt.figure(0)
plt.clf()
plt.xticks([])
plt.yticks([])
grad = plt.imshow(np.log(abs(Z)),
    cmap = cm.gray)
plt.colorbar(grad)
plt.savefig(figdir+'lapin_spectre.
    png')
```

Questions

Comment construire un filtre basique à appliquer au spectre du lapin.

Filtre passe bas (1/3)

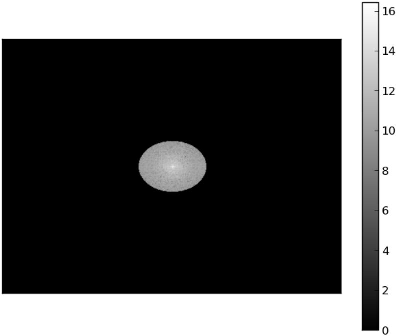



```
# Filtre passe bas
x = np.linspace(-1., 1., len(Z[0]))
y = np.linspace(-1., 1., len(Z))
X, Y = np.meshgrid(x, y)
R = (X**2 + Y**2)**.5
F = R < .1 # Voici notre filtre
#F = abs(X) <0.05
fig = plt.figure(0)
plt.clf()
plt.xticks([])
plt.yticks([])
grad = plt.imshow(F, cmap = cm.gray)
plt.colorbar(grad)
```

Questions

Comment appliquer ce filtre à notre spectre ?

Filtre passe bas (2/3)

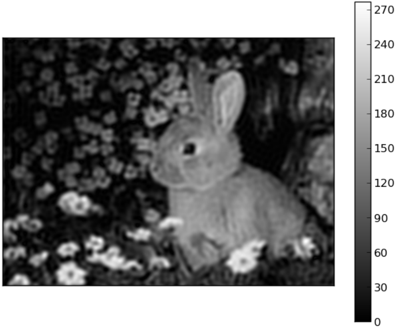



```python

# Application du filtre passe bas
Z = Z * F
fig = plt.figure(0)
plt.clf()
plt.xticks([])
plt.yticks([])
grad = plt.imshow(np.log(abs(Z+1)),
        cmap = cm.gray)
plt.colorbar(grad)
```

Questions

Quel est le résultat par FFT inverse ?

Filtre passe bas (3/3)

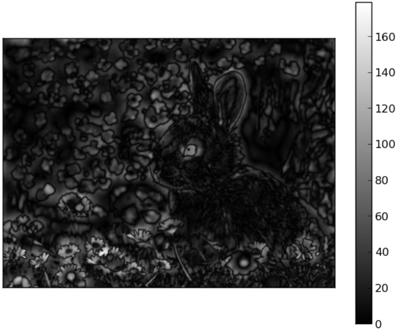



```python

# FFT inverse
z_lp = abs(ifft2(ifftshift(Z)))
fig = plt.figure(0)
plt.clf()
plt.xticks([])
plt.yticks([])
grad = plt.imshow(z_lp, cmap = cm.
    gray)
plt.colorbar(grad)
```

Questions

- Trouvez vous la qualité correcte ?
- En vous inspirant des exemples 1D, avez vous une idée pour améliorer les choses ?

Filtre passe haut




```python

# Filtre passe haut
x = np.linspace(-1., 1., len(Z[0]))
y = np.linspace(-1., 1., len(Z))
X, Y = np.meshgrid(x, y)
R = (X**2 + Y**2)**.5
F = R > .05 # Voici notre filtre
#F = abs(Y) >0.05
Z = fftshift(fft2(z))
z_hp = abs(ifft2(ifftshift(Z * F)))
fig = plt.figure(0)
plt.clf()
plt.xticks([])
plt.yticks([])
grad = plt.imshow(z_hp, cmap = cm.
    gray)
```

Questions

- Quelles applications ?