

MGM657 Outils Numériques pour l'Ingénieur

Traitement de signal

ludovic.charleux@univ-savoie.fr

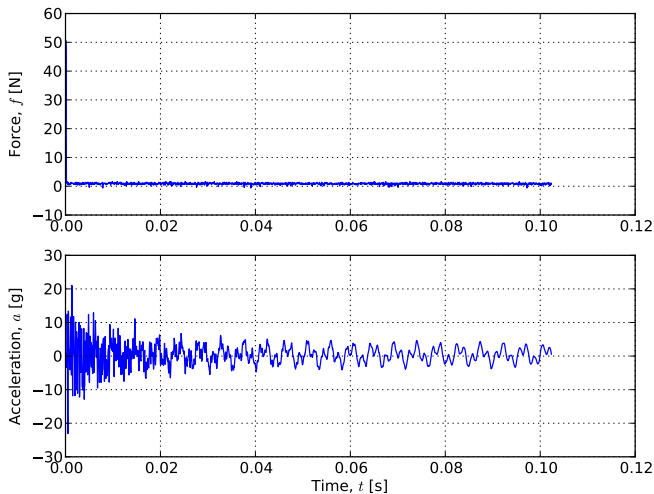
www.polytech.univ-savoie.fr

- 1 Introduction
- 2 Notations
- 3 Outils
- 4 Échantillonnage
- 5 Analyse fréquentielle

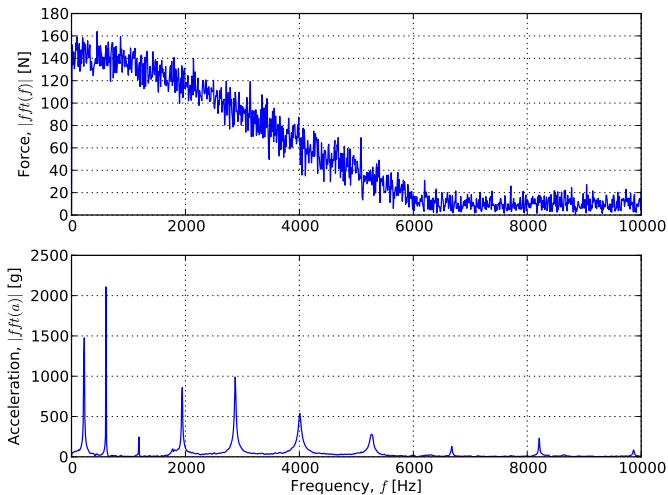
Plan

- 1 Introduction
- 2 Notations
- 3 Outils
- 4 Échantillonnage
- 5 Analyse fréquentielle

Vibration d'une poutre



Vibration d'une poutre : spectre



Plan

- 1 Introduction
- 2 Notations**
- 3 Outils
- 4 Échantillonnage
- 5 Analyse fréquentielle

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

- 1 Introduction
- 2 Notations
- 3 Outils**
- 4 Échantillonnage
- 5 Analyse fréquentielle

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 langage 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/](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) :

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

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

```
1 python test.py
```

Ou :

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

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

Plan

- 1 Introduction
- 2 Notations
- 3 Outils
- 4 Échantillonnage**
- 5 Analyse fréquentielle

Les bases

Principe

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

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

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

$$D = N/f_e$$

Le signal échantillonné est alors obtenu par :

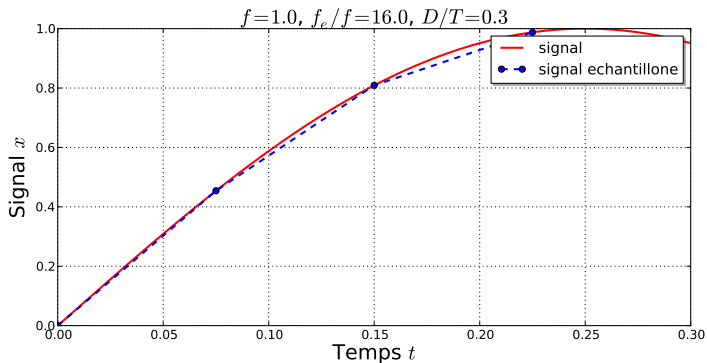
$$x_n = x(t_n)$$

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

Paramètres importants

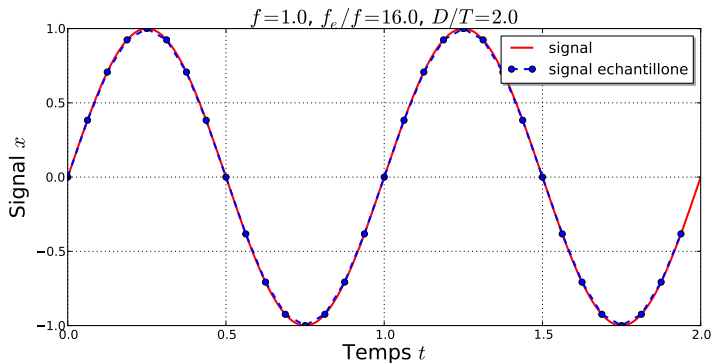
- $\frac{D}{T}$: ajustable en modifiant la durée d'observation D
- $\frac{f_e}{f}$: ajustable en modifiant la fréquence d'échantillonnage f_e .

Choix de la durée d'observation (1/2)



Temps d'observation D trop court !

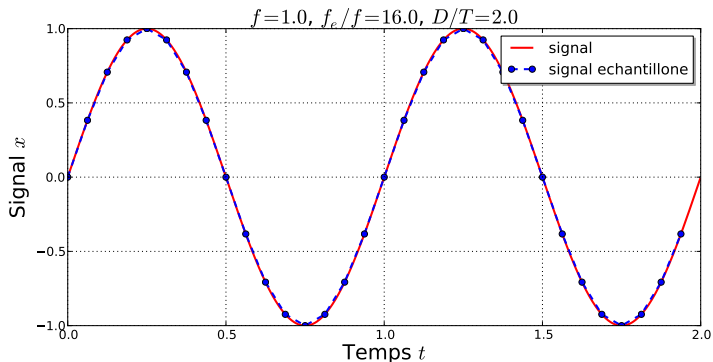
Choix de la durée d'observations (2/2)



Conclusion

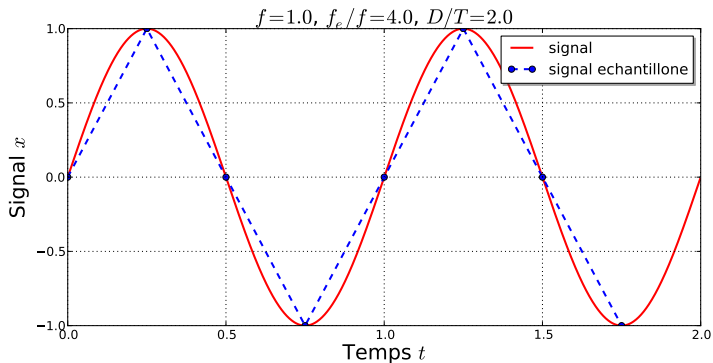
- Il faut que $D/T \geq 1$
- Idéalement, $D/T \in \mathbb{N}$

Une borne basse de la fréquence d'échantillonnage ?



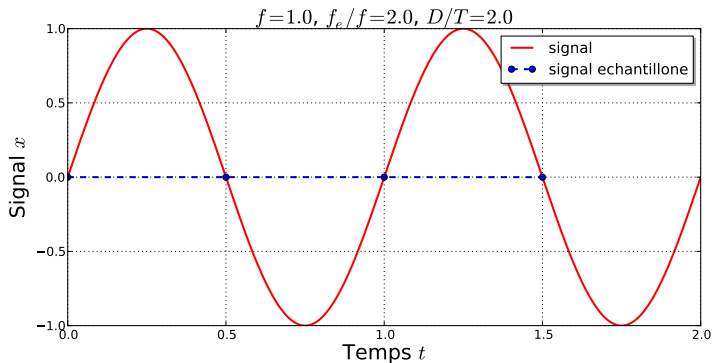
Échantillonnage correct.

Une borne basse de la fréquence d'échantillonnage ?



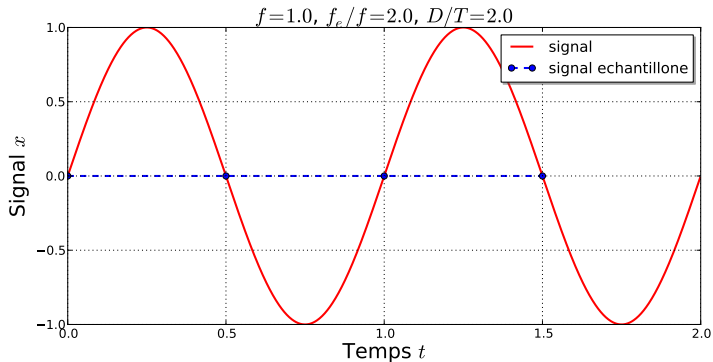
Échantillonnage correct.

Une borne basse de la fréquence d'échantillonnage ?



Fréquence d'échantillonnage trop basse.

Borne basse de la fréquence d'échantillonnage ?

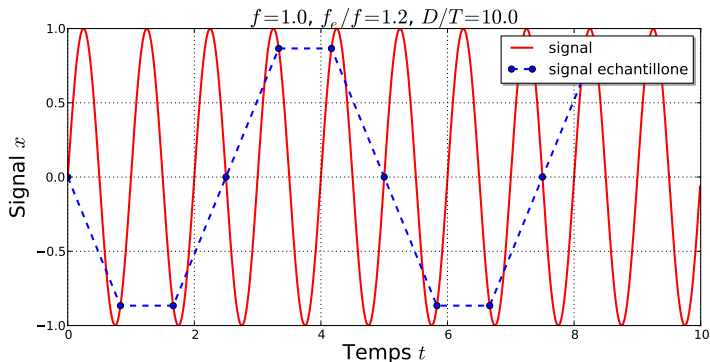


Fréquence d'échantillonnage f_e trop basse.

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'échantillonnage.

Hautes fréquences et aliasing



Hautes fréquences et aliasing

Les fréquences trop hautes vis-à-vis de la fréquence d'échantillonnage (*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 échantillonnage 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 :

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

Ce qu'il faut comprendre

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

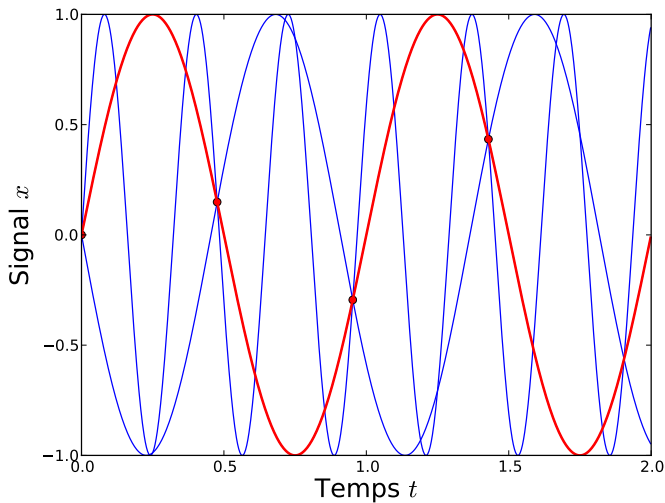
Mises en pratique (1/2)

```

1 # listings/exemple_aliasing.py
2 from matplotlib import pyplot as plt
3 from math import sin, pi
4 from signal import sinusoidal as signal
5 from numpy import arange, floor
6 beaucoup = 1000
7 f = 1. # Frequence du signal
8 D = 2./f # Duree d'observation
9 t_min = 0. # Debut du calcul du signal
10 t_max = t_min+D # Fin du calcul du signal
11 fe = 2.1*f # Frequence d'echantillonnage
12 N = int(floor(D*fe)) # Nombre de points d'evaluation
13 plt.figure(0)
14 plt.clf()
15 plt.xlabel('Temps $t$', fontsize=20)
16 plt.ylabel('Signal $x$', fontsize=20)
17 kmin,kmax = -1, 2
18 t = arange(beaucoup)/float(beaucoup)*(t_max-t_min)+t_min
19 for k in xrange(kmin,kmax):
20     f1 = f + k*fe
21     x = [signal(tt,T=1./f1) for tt in t]
22     plt.plot(t,x,'b-',linewidth=1.)
23 tn = arange(N)/(D*fe)*(t_max-t_min)+t_min
24 xn = [signal(tt,T=1./f) for tt in tn]
25 plt.plot(tn,xn,'or')
26 x = [signal(tt,T=1./f) for tt in t]
27 plt.plot(t,x,'r-',linewidth=2.)
28 plt.savefig('../figures/exemple_aliasing.pdf')

```

Mises en pratique (2/2)



Plan

- 1 Introduction
- 2 Notations
- 3 Outils
- 4 Échantillonnage
- 5 Analyse fréquentielle**

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

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

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) :

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

Les coefficients associés à $n < 0$ sont donc inutiles car conjugués de coefficients obtenus pour $n \geq 0$. On remarque aussi que les coefficients 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

```

1 # listings/serieF_carre.py
2 from cmath import exp
3 from numpy import arange, array
4 from math import pi
5 from matplotlib import pyplot as plt
6 from trace_complexes import *
7 N = 1 # Nombre de coefficients calculés
8 for N in [1,2,4,8,16,32,64,128,256,1024]: # Valeurs de N
9     # Indices n impairs
10    n = [2*k+1 for k in range(-N,0)]+[2*k+1 for k in range(N)]
11    c = [-2j/(pi*nn) for nn in n] # Coefficients c
12    T,beaucoup= 1., 2000
13    t,f = arange(beaucoup)/float(beaucoup)*T, array(n[N:2*N])/T
14    trace_complexes(f,c[N:2*N], '../figures/serieF_carre_c_{0}.pdf'.format(
15        N),title='Coefficients $c_n$ pour $N = {0}$'.format(N))
16    x = []
17    for tt in t:
18        x.append(0.)
19        for i in xrange(len(n)):
20            x[-1] = x[-1]+c[i]*exp(2j*pi*n[i]*tt/T)
21    x1 = [xx.real for xx in x]
22    plt.figure(0,figsize=(10,2))
23    plt.clf()
24    plt.plot(t,x1)
25    plt.title('$N={0}$'.format(N))
26    plt.savefig('../figures/serieF_carre_{0}.pdf'.format(N))

```

Exercice : signal carré

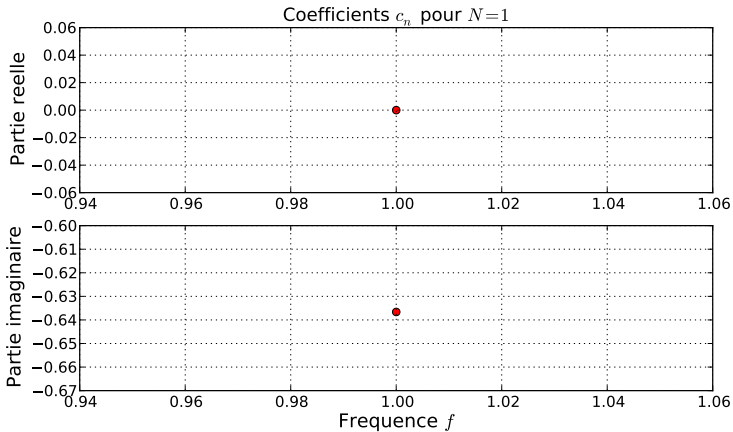
Programme de tracé de vecteurs complexes

```

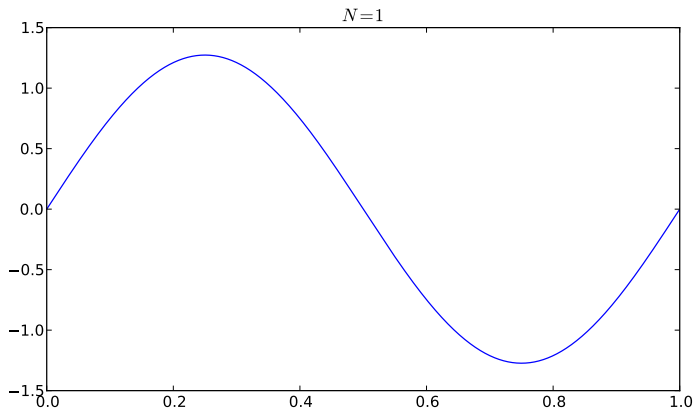
1 # trace_complexes.py
2 import matplotlib.pyplot as plt
3
4 def trace_complexes(x,y,fichier,xlabel='Frequence $f$', title='', style=
    'ro'):
5     plt.figure(0,figsize=(9,5))
6     plt.clf()
7     plt.grid(True)
8     p0 = plt.subplot(2,1,1)
9     p0.set_title(title)
10    p0.grid()
11    p0.plot(x,[yy.real for yy in y],style,linewidth=2.0)
12    p0.set_ylabel('Partie reelle', fontsize=15)
13    p1 = plt.subplot(2,1,2)
14    p1.grid()
15    p1.plot(x,[yy.imag for yy in y],style,linewidth=2.0)
16    p1.set_xlabel(xlabel, fontsize=15)
17    p1.set_ylabel('Partie imaginaire', fontsize=15)
18    plt.savefig(fichier)

```

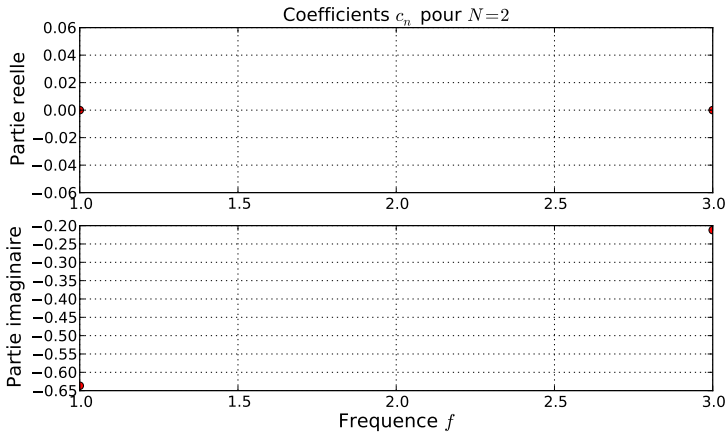
Exercice : signal carré



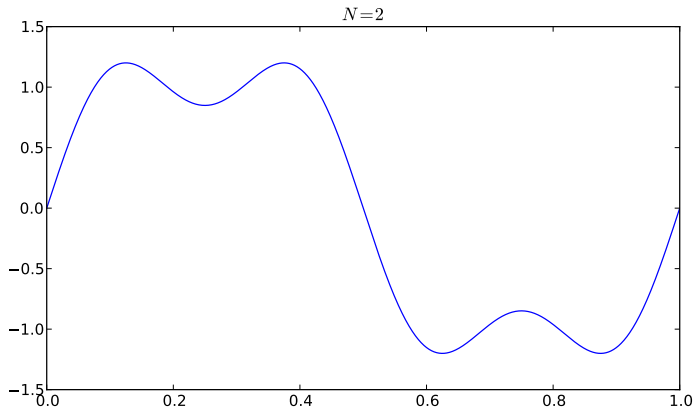
Exercice : signal carré



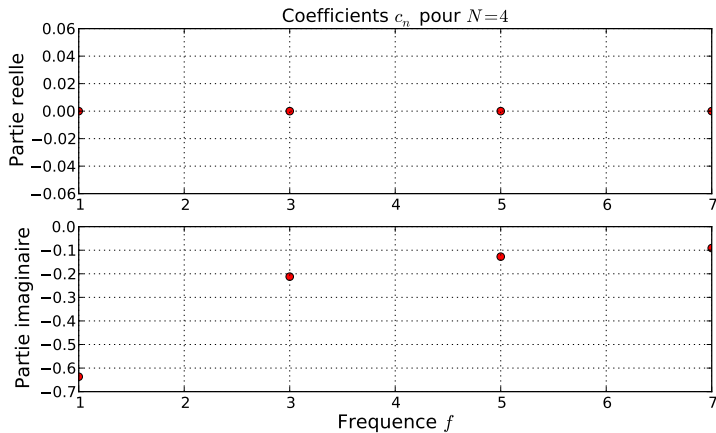
Exercice : signal carré



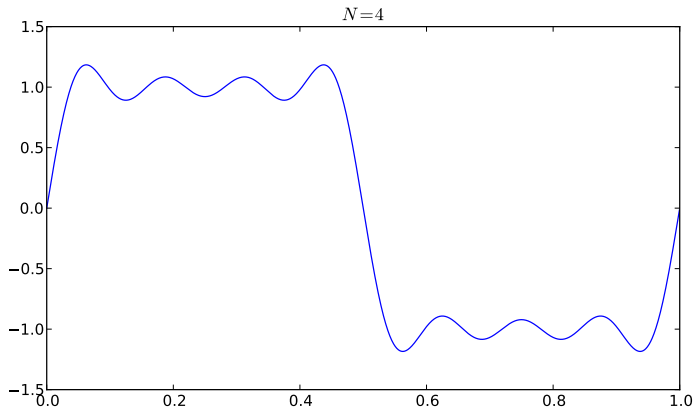
Exercice : signal carré



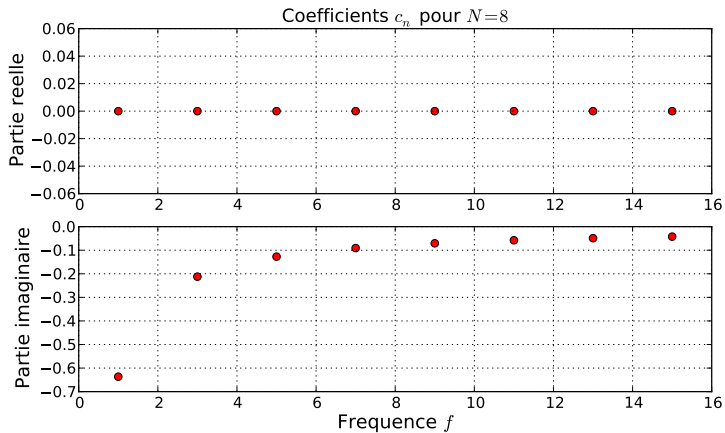
Exercice : signal carré



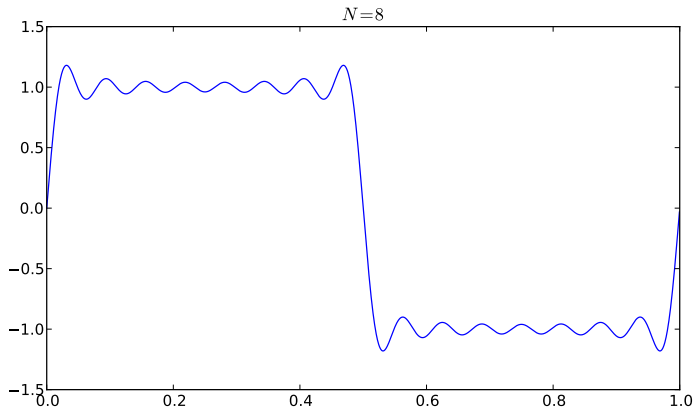
Exercice : signal carré



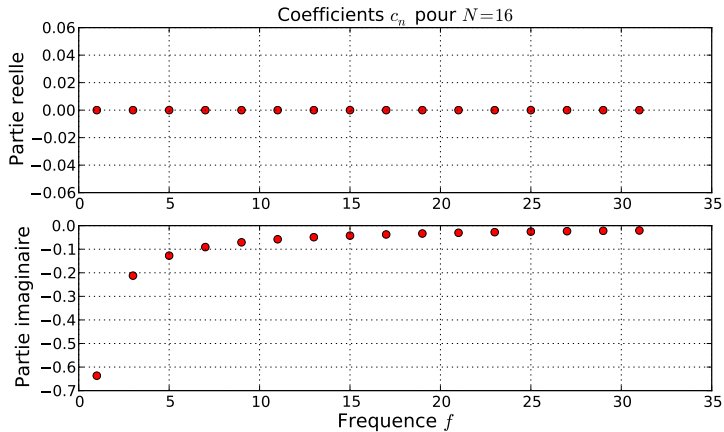
Exercice : signal carré



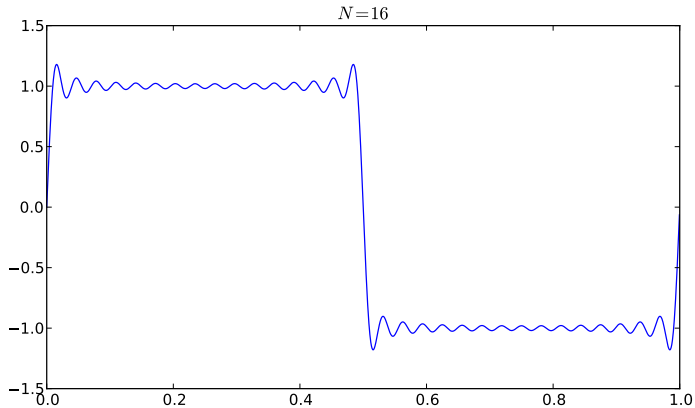
Exercice : signal carré



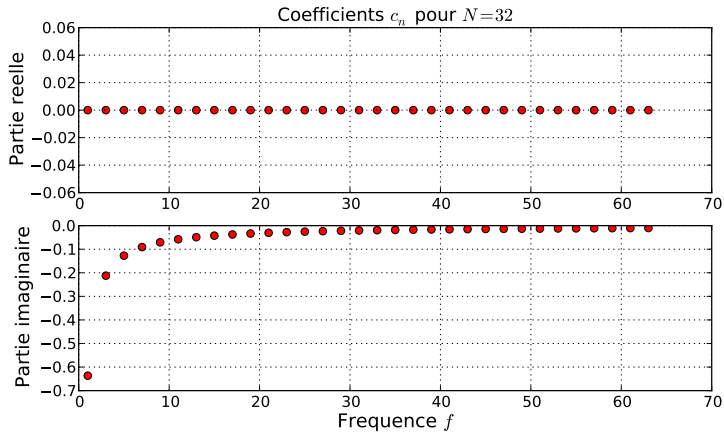
Exercice : signal carré



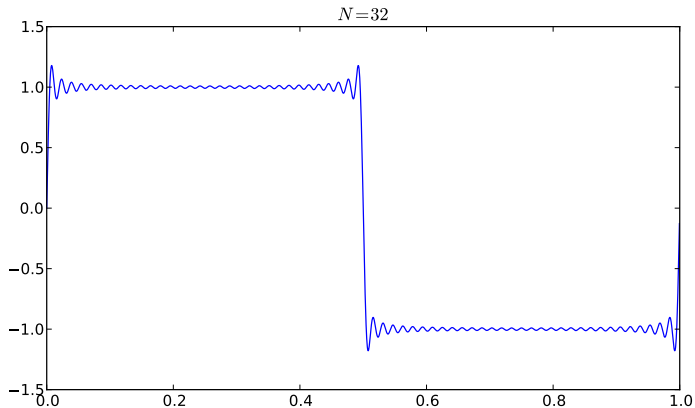
Exercice : signal carré



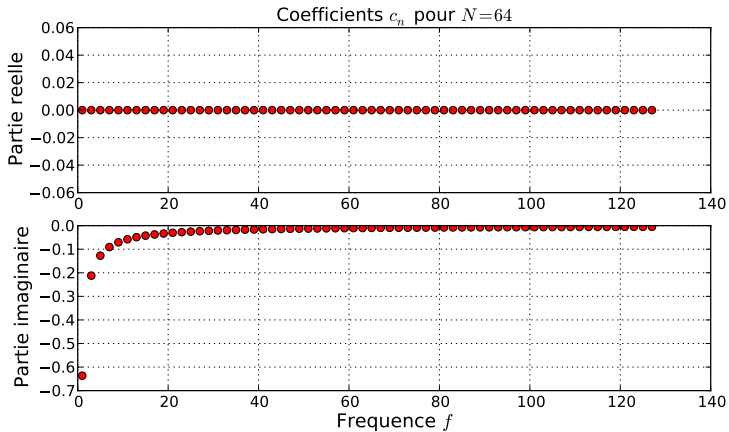
Exercice : signal carré



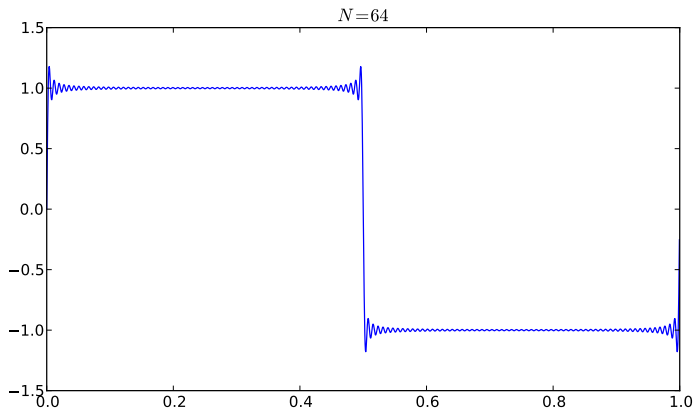
Exercice : signal carré



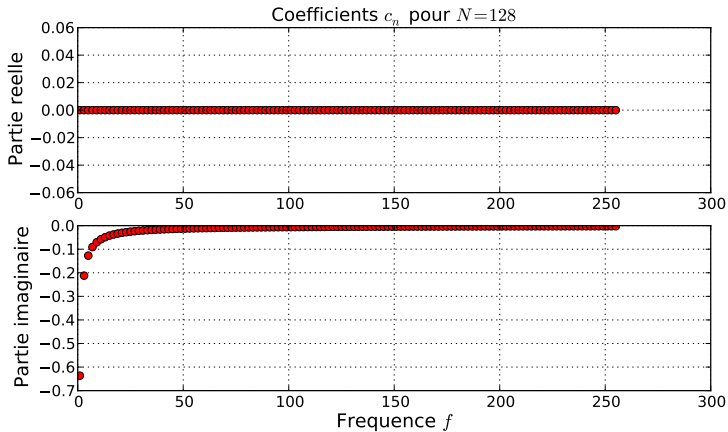
Exercice : signal carré



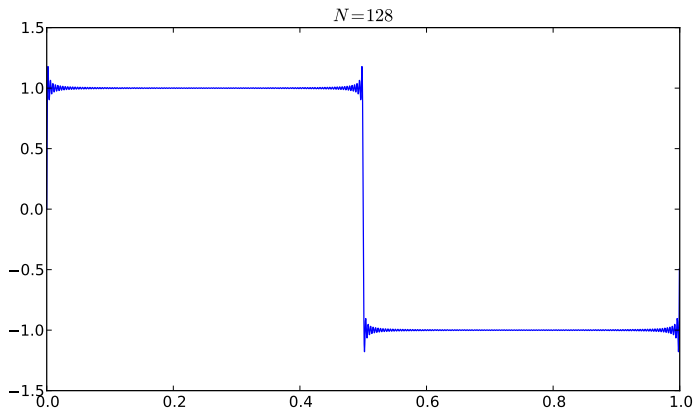
Exercice : signal carré



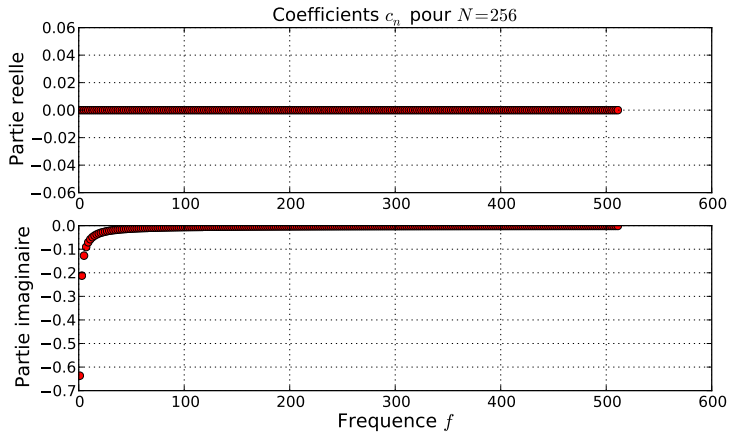
Exercice : signal carré



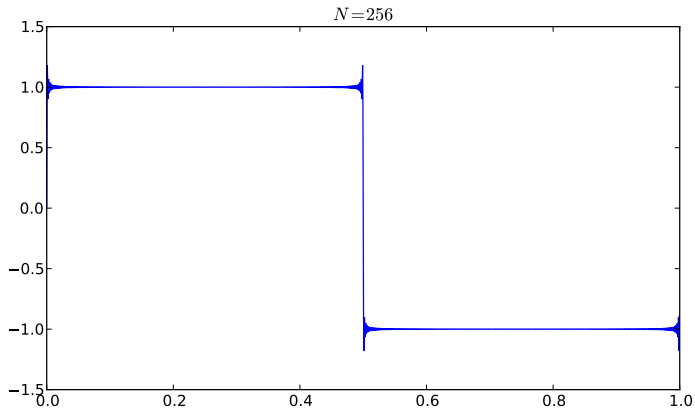
Exercice : signal carré



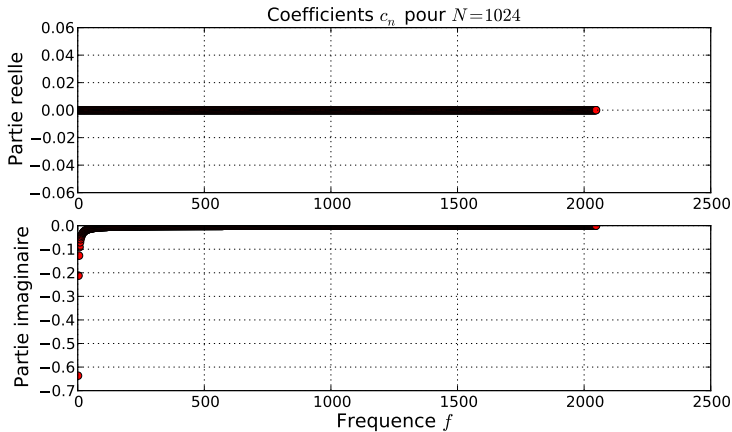
Exercice : signal carré



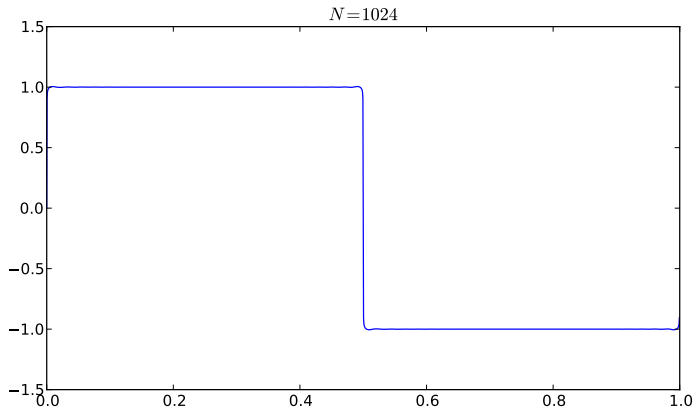
Exercice : signal carré



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 transformée 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 échantillonné $[x_n]$ comportant N échantillons. Sa transformée de Fourier discrète $[X_k]$ s'écrit :

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

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

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 où 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'interpréter 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

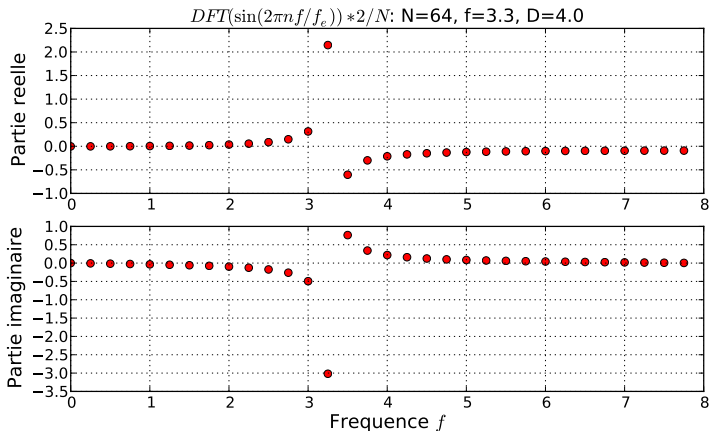
Programme de calcul de la DFT

```

1 # listings/exemple_DFT.py
2 from signal_sinusoidal import *
3 from signal_carre import *
4 from trace_complexes import *
5 from numpy import arange, floor
6 from cmath import exp
7 signal = signal_sinusoidal #signal = signal_carre
8 beaucoup, rien = 1000, 1.e-10
9 f = 3.3 # Frequence du signal
10 D = 4. # Duree d'observation
11 t_min = 0. # Debut du calcul du signal
12 t_max = t_min+D # Fin du calcul du signal
13 fe = 16. # Frequence d'echantillonnage
14 N = int(floor(D*fe)) # Nombre de points d'evaluation
15 tn = arange(N)/(D*fe)*(t_max-t_min)+t_min # Discretisation du temps
16 xn = [signal(tt,T=1./f, k=4.) for tt in tn] # Discretisation du signal
17 Xk = [] # DFT de xn
18 for k in range(N): # Boucle sur k
19     Xk.append(0.)
20     for n in range(N): # Boucle sur n
21         Xk[-1] = Xk[-1] + xn[n]*exp(-2j*pi*n*k/N) # Calcul de Xk
22         if abs(Xk[-1].real) < rien : Xk[-1] = Xk[-1] - Xk[-1].real
23         if abs(Xk[-1].imag) < rien : Xk[-1] = Xk[-1] - 1j*Xk[-1].imag
24     Xk[-1] = Xk[-1]*2/N
25 fk = arange(N)/D # Discretisation des frequences
26 tit='$DFT(\sin(2\pi nf/f_e))*2/N$: N={0}, f={1}, D={2}'.format(N,f,D)
27 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'exécution par rapport Python.

Comment?

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

```
1 # listings/exemple_FFT.py
2 from math import pi, sin
3 from scipy.fftpack import fft
4 N = 8
5 xn = [sin(2*pi*n/N) for n in xrange(N)]
6 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

```

1 # listings/exemple_FFT_frequence.py
2 from math import pi, sin, exp
3 from scipy.fftpack import fft
4 from random import gauss
5 from numpy import array, arange, floor
6 from matplotlib import pyplot as plt
7 import matplotlib.gridspec as gridspec
8 from signal import sinusoidal import *
9 beaucoup = 1000
10 fe = 64. # Frequence d'echantillonnage
11 N = 4096 # Nombre de points d'echantillonnage
12 D = N/fe # Duree d'observation
13 f = 8./D # Frequence du signal
14 t_min = 0. # Debut du calcul du signal
15 t_max = t_min+D # Fin du calcul du signal
16 stddev = 0. # Ecart type du bruit
17 nom = '../figures/FFT_frequence.pdf'
18 amort = 0.
19 val = [fe-2, 64./D, 16./D, 8./D] # Frequence
20 lab = '$f={0}$'
21 tn = arange(N)/(D*fe)*D+t_min
22 for i in xrange(N):
23     if tn[i]<=1./f: i_t = i
24 xn = [sin(2*pi*f*t) for t in tn]
25 fk = arange(N)/D # Discretisation des frequences
26 plt.figure(0, figsize=(12,8))
27 plt.clf()
28 gs = gridspec.GridSpec(4, 3) # Grille de zone de trace

```

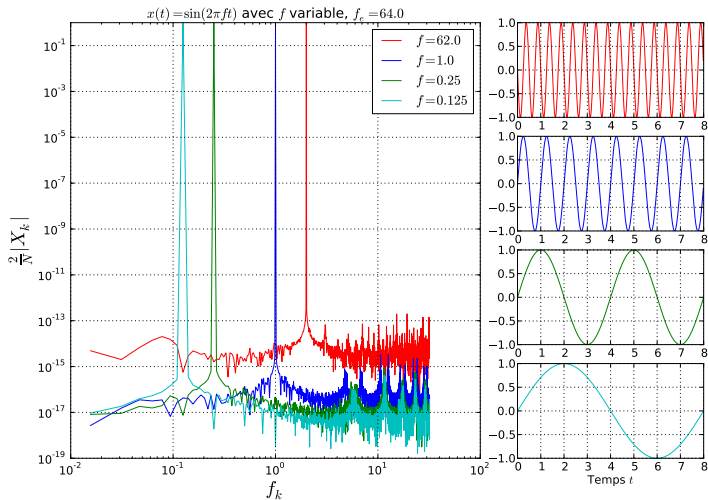
FFT : effet de la fréquence f

Programme

```

29 p0 = plt.subplot(gs[:,2])
30 p0.set_title(r'$x(t) = \sin(2 \pi f t)$ avec $f$ variable, $f_e={}$$.format(fe))
31 p0.grid()
32 p0.set_xlabel(r'$f_k$', fontsize=20)
33 p0.set_ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
34 p0.set_xscale('log')
35 p0.set_yscale('log')
36 for z in xrange(len(val)):
37     v = val[z]
38     #stddev = v
39     f = v
40     xn = [sin(2*pi*f*t) for t in tn]
41     v_amort = array([exp(-t*amort) for t in tn])
42     color = ['r','b','g','c'][z]
43     bruit = array([gauss(0,stddev) for i in xrange(N)])
44     xxn = (xn + bruit)*v_amort
45     Xk = abs(fft(xxn))*2./N
46     p0.plot(fk[0:N/2],Xk[0:N/2],'-'+color, label = lab.format(v))
47     p0.legend()
48     p1 = plt.subplot(gs[z,-1])
49     p1.grid()
50     p1.plot(tn[:i_t],xxn[:i_t],'-'+color)
51 p1.set_xlabel('Temps $t$')
52 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

```

1 # listings/exemple_FFT_D.py
2 from math import pi, sin, exp
3 from scipy.fftpack import fft
4 from random import gauss
5 from numpy import array, arange, floor, hanning
6 from matplotlib import pyplot as plt
7 import matplotlib.gridspec as gridspec
8 from signal import sinusoidal import *
9 beaucoup = 1000
10 fe = 128. # Frequence d'echantillonnage
11 f = 1. # Frequence du signal
12 t_min = 0. # Debut du calcul du signal
13 stddev = 0. # Ecart type du bruit
14 nom = '../figures/FFT_D.pdf'
15 amort = 0.
16 val = 8./f*array([.8, 1., 1.2, 1.6])
17 lab = '$D/T={0}$'
18
19
20 plt.figure(0, figsize=(12,8))
21 plt.clf()
22 gs = gridspec.GridSpec(4, 3) # Grille de zone de trace
23 p0 = plt.subplot(gs[:, :2])
24 p0.set_title(r'$x(t) = \sin(2 \pi f t)$ avec $D$ variable')
25 p0.grid()
26 p0.set_xlabel(r'$f_k$', fontsize=20)
27 p0.set_ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
28 p0.set_xscale('log')

```

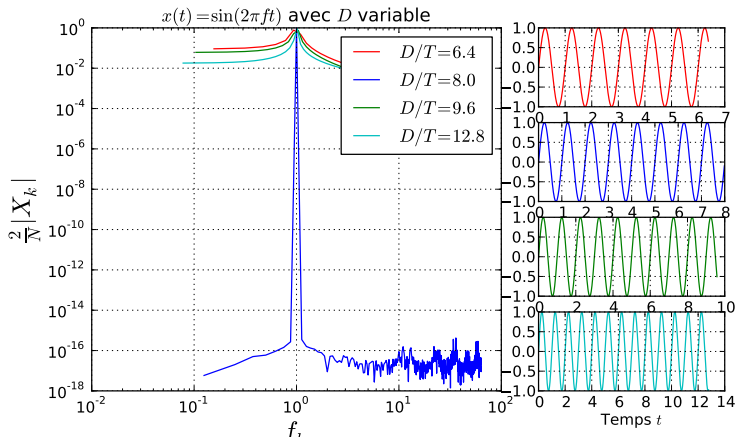
FFT : effet de la durée d'observation D

Programme

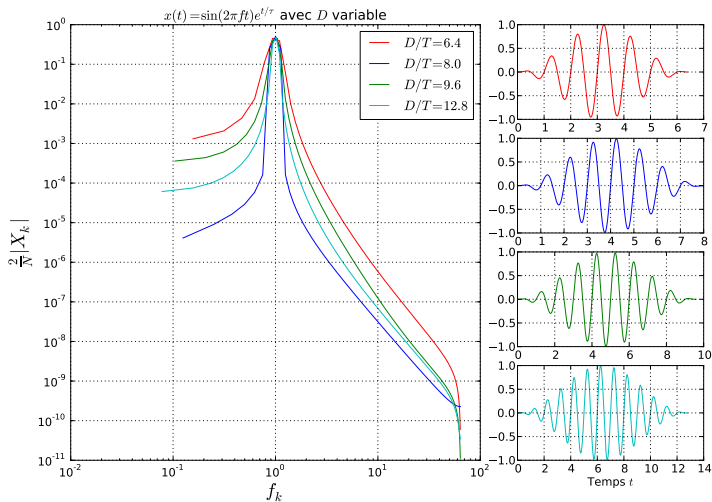
```

29 p0.set_yscale('log')
30 for z in xrange(len(val)):
31     v = val[z]
32     D = v
33     i_t=-1
34     N_ = int(D*fe)
35     fk = arange(N)/D # Discretisation des frequences
36     t_max = t_min+D # Fin du calcul du signal
37     tn = arange(N)/float(N)*D+t_min
38     xn = [sin(2*pi*f*t) for t in tn]
39     v_amort = array([exp(-t*amort) for t in tn])
40     color = ['r','b','g','c'][z]
41     bruit = array([gauss(0,stddev) for i in xrange(N)])
42     xxn = (xn + bruit)*v_amort #*hanning(N)
43     Xk = abs(fft(xxn))*2./N
44     p0.plot(fk[0:N/2],Xk[0:N/2],'-'+color, label = lab.format(v*f))
45     p1 = plt.subplot(gs[z,-1])
46     p1.grid()
47     p1.plot(tn[:i_t],xxn[:i_t],'-'+color)
48 p0.legend()
49 p1.set_xlabel('Temps $t$')
50 plt.savefig(nom)

```

FFT : effet de la durée d'observation D 

Lorsque 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

```

1 # listings/exemple_FFT_bruit.py
2 from math import pi, sin, exp
3 from scipy.fftpack import fft
4 from random import gauss
5 from numpy import array, arange, floor
6 from matplotlib import pyplot as plt
7 import matplotlib.gridspec as gridspec
8 from signal import sinusoidal import *
9 beaucoup = 1000
10 fe = 64. # Frequence d'echantillonnage
11 N = 4096 # Nombre de points d'echantillonnage
12 D = N/fe # Duree d'observation
13 f = 8./D # Frequence du signal
14 t_min = 0. # Debut du calcul du signal
15 t_max = t_min+D # Fin du calcul du signal
16 stddev = 0. # Ecart type du bruit
17 nom = '../figures/FFT_bruit.pdf'
18 amort = 0.
19 val = [1., 1.e-1, 1.e-2, 0.] # Bruit
20 lab = 'Ecart type bruit: {0}'
21 tn = arange(N)/(D*fe)*D+t_min
22 for i in xrange(N):
23     if tn[i]<=1./f: i_t = i
24 xn = [sin(2*pi*f*t) for t in tn]
25 fk = arange(N)/D # Discretisation des frequences
26 plt.figure(0, figsize=(12,8))
27 plt.clf()
28 gs = gridspec.GridSpec(4, 3) # Grille de zone de trace

```


FFT : effet du bruit

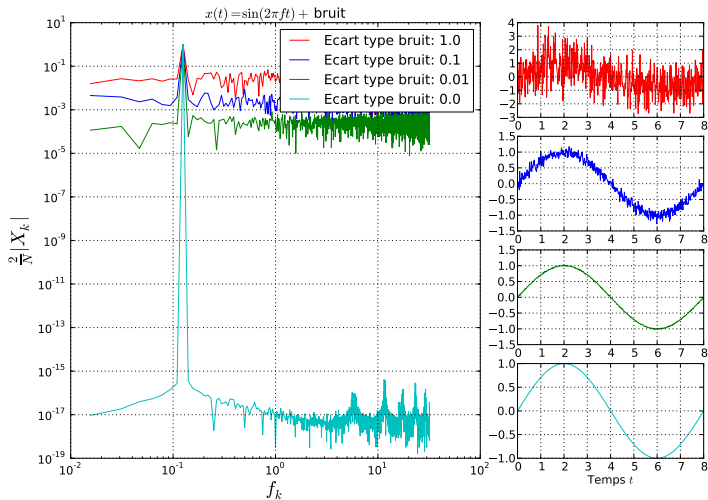
Programme

```

29 p0 = plt.subplot(gs[:,2])
30 p0.set_title(r'$x(t) = \sin(2 \pi f t) + $ bruit')
31 p0.grid()
32 p0.set_xlabel(r'$f_k$', fontsize=20)
33 p0.set_ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
34 p0.set_xscale('log')
35 p0.set_yscale('log')
36 for z in xrange(len(val)):
37     v = val[z]
38     stddev = v
39     v_amort = array([exp(-t*amort) for t in tn])
40     color = ['r','b','g','c'][z]
41     bruit = array([gauss(0,stddev) for i in xrange(N)])
42     xxn = (xn + bruit)*v_amort
43     Xk = abs(fft(xxn))*2./N
44     p0.plot(fk[0:N/2],Xk[0:N/2],'-'+color, label = lab.format(v))
45     p0.legend()
46     p1 = plt.subplot(gs[z,-1])
47     p1.grid()
48     p1.plot(tn[:i_t],xxn[:i_t],'-'+color)
49     p1.set_xlabel('Temps $t$')
50     plt.savefig(nom)

```

FFT : effet du bruit



Le bruit réduit la hauteur du pic.

FFT : effet de l'amortissement

Programme

```

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

```

FFT : effet de l'amortissement

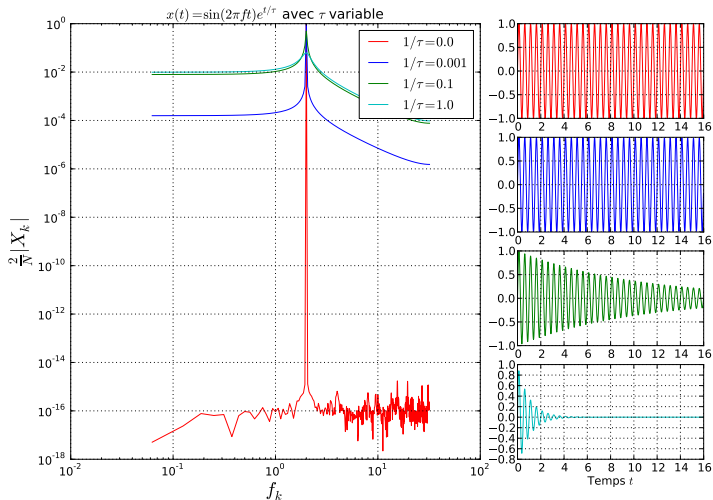
Programme

```

29 p0.grid()
30 p0.set_xlabel(r'$f_k$', fontsize=20)
31 p0.set_ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
32 p0.set_xscale('log')
33 p0.set_yscale('log')
34
35 for z in xrange(len(val)):
36     v = val[z]
37     amort = v
38     v_amort = array([exp(-t*amort) for t in tn])
39     color = ['r','b','g','c'][z]
40     bruit = array([gauss(0,stddev) for i in xrange(N)])
41     xxn = (xn + bruit)*v_amort*hanning(N)
42     Xk = abs(fft(xxn))*2./N
43     p0.plot(fk[0:N/2],Xk[0:N/2],'-'+color, label = lab.format(v))
44     p0.legend()
45     p1 = plt.subplot(gs[z,-1])
46     p1.grid()
47     p1.plot(tn[:],xxn[:],'-'+color)
48     p1.set_xlabel('Temps $t$')
49     plt.savefig(nom)

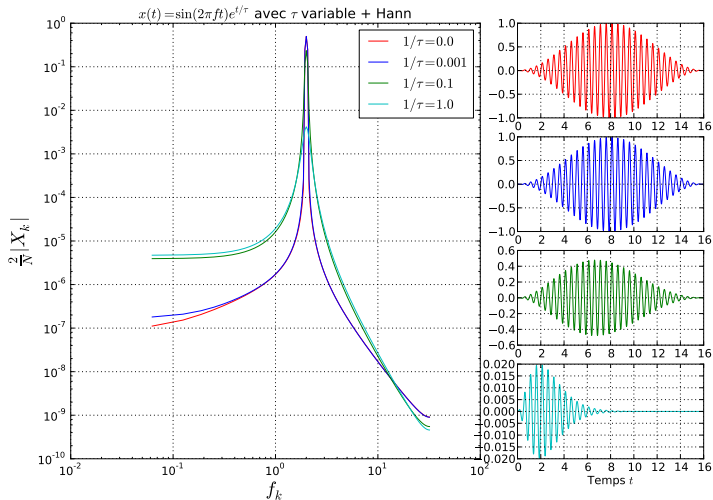
```

FFT : effet de l'amortissement



L'amortissement entraine une perte de hauteur du pic.

FFT : amortissement et fenêtrage de Hann



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

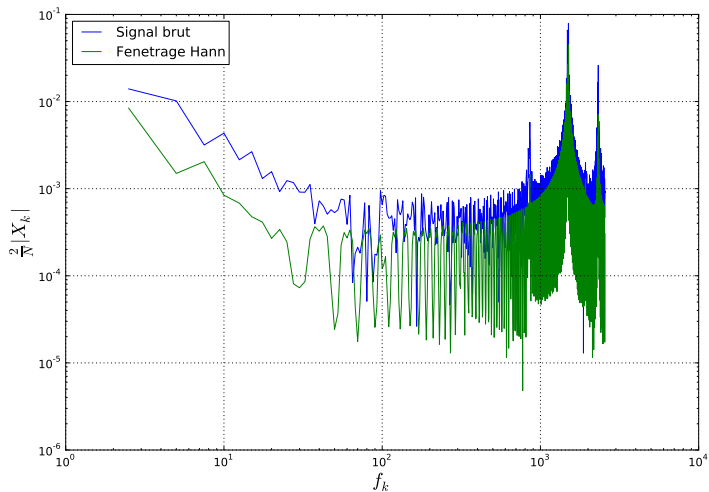
Application à la cloche

```

1 # listings/exemple_FFT_cloche.py
2 from scipy.fftpack import fft
3 from numpy import array, arange, hanning
4 from matplotlib import pyplot as plt
5 import pickle
6 nom = '../figures/FFT_cloche-log.pdf'
7 fichier = open('cloche.pckl','r') # Ouverture du fichier
8 cloche = pickle.load(fichier) # Chargement des donnees
9 fichier.close() # Fermeture du fichier
10 xn = cloche['x'][:32] # Redimensionnement des donnees
11 fe = float(cloche['fe']) # Definition de la frequence d'echantillonnage
12 tn = arange(len(xn))/float(fe)
13 N = len(tn)
14 D = N/fe
15 plt.figure(0, figsize=(12,8))
16 plt.clf()
17 fk = arange(N)/D # Discretisation des frequences
18 Xk = abs(fft(xn))*2./N
19 Xkh = abs(fft(xn*hanning(N)))*2./N
20 plt.plot(fk[0:N/2],Xk[0:N/2],'-', label='Signal brut')
21 plt.plot(fk[0:N/2],Xkh[0:N/2],'-', label='Fenetrage Hann')
22 plt.xlabel(r'$f_k$', fontsize=20)
23 plt.ylabel(r'$\frac{2}{N}|X_k|$', fontsize=20)
24 plt.xscale('log')
25 plt.yscale('log')
26 plt.grid(True)
27 plt.legend(loc='upper left')
28 plt.savefig(nom)

```

Application à la cloche



Application à la cloche

