Resolution_de_Fourier

J.-F. Bercher

December 10, 2013

1 Résolution de Fourier

Pour le moment on n'examinera pas les spécificités de la Transformée de Fourier discrète. On ne s'intéressera qu'au problème de la limitation en temps. Les transformées de Fourier seront calculées sur un grand nombre de points (par exemple 1000), selon `fft(x,1000)`, afin de ne pas être embêtés par la discrétisation de l'axe fréquentiel.

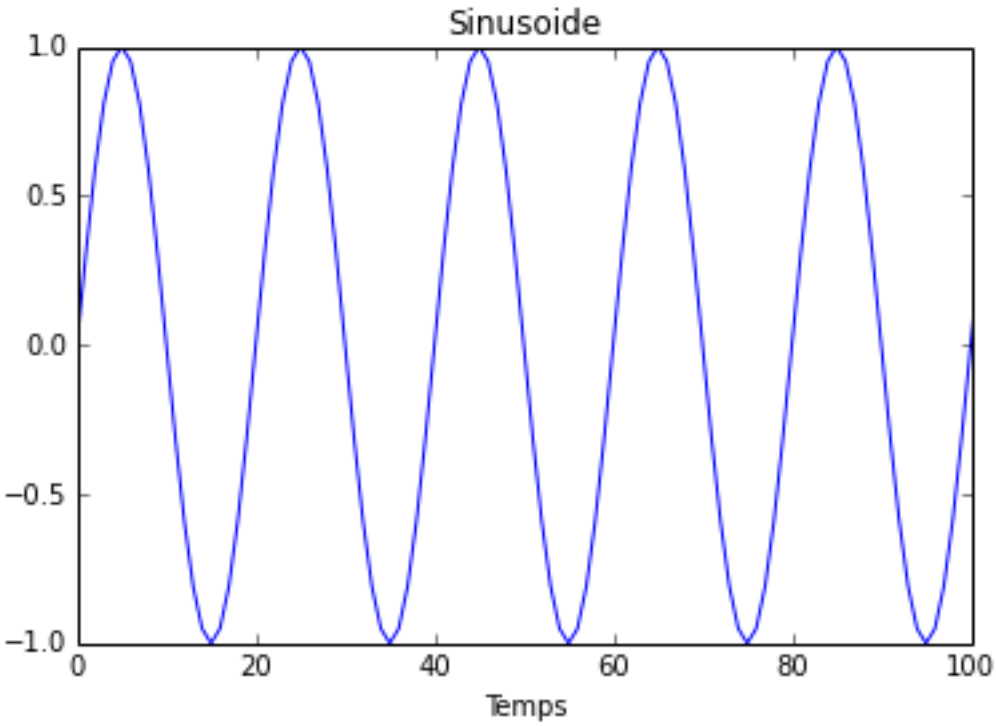
1.1 Analyse sur une durée limitée - résolution

On crée une sinusoïde sur 1000 points, avec une fréquence normalisée $f_0 = 0.05$. Remarquons que sur N=1000 points, on a exactement $Nf_0 = 50$ périodes, et qu'ici ce nombre est entier. On examinera plus tard ce qu'il advient lorsque le nombre de périodes n'est pas entier.

In [21]:
```
fo=0.05
N=1000
t=arange(N)
x=sin(2*pi*fo*t)
figure(1)
plot(t,x)
title("Sinusoide")
xlabel("Temps")
xlim([0, 100])
```

```
(0, 100)
```

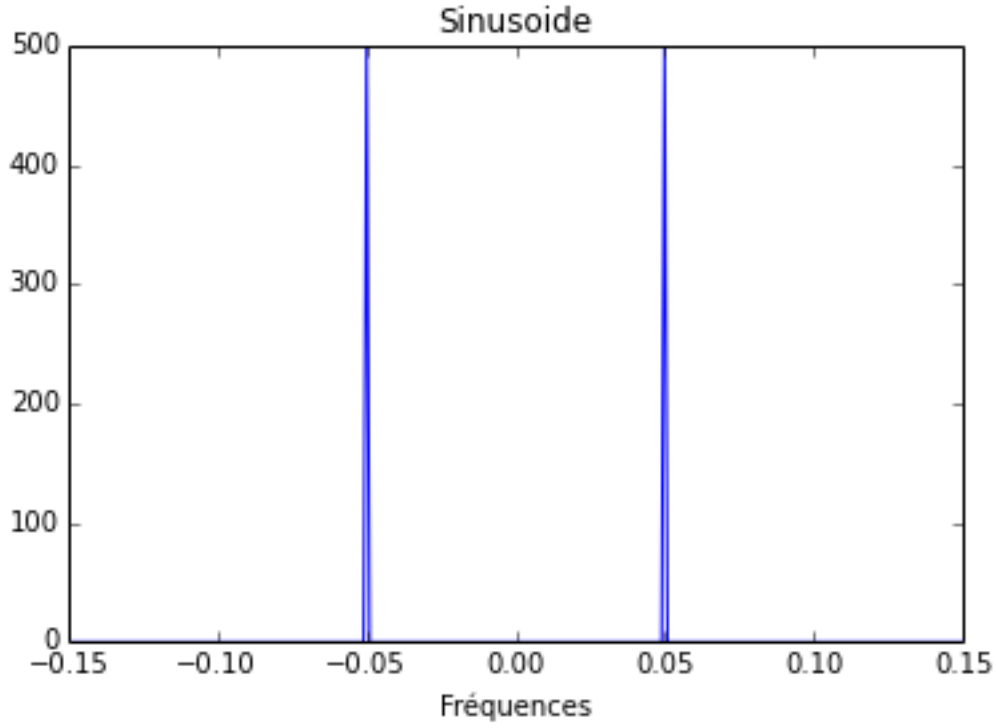
Out [21]:



In [22]:
```
from numpy.fft import fft, ifft
M=1000 # Nombre de points de calcul de la fft : 1000
X=fft(x,M)
f=arange(M)/M-1/2
figure(2)
plot(f,fftshift(abs(X)))
title("Sinusoide")
xlabel("Fréquences")
xlim([-0.15, 0.15])
```

```
(-0.15, 0.15)
```

Out [22]:



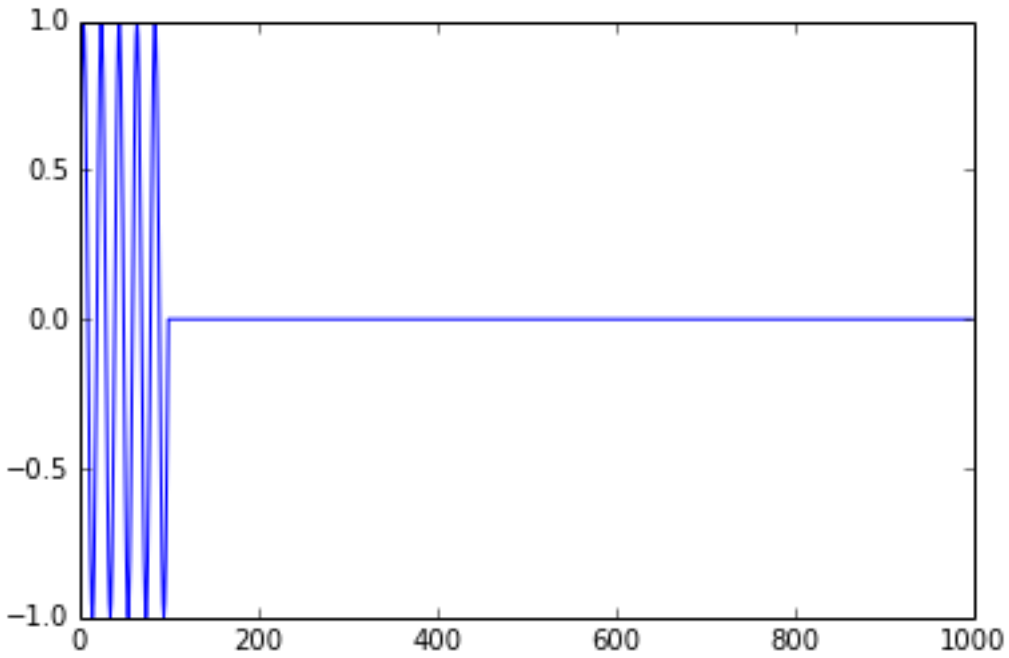
Supposons maintenant que la sinusoïde soit de durée limitée, par exemple sur $L = 100$ ou $L = 50$ points. Créez

une sinusoïde tronquée `x_tronq` sur L points (créez un vecteur de zéros puis remplissez les L premiers points par la sinusoïde). Examinez ensuite ce que vaut alors sa TF, et comparez ce résultat à la TF initiale (pour la visualisation, prendre en compte un facteur d'amplitude L/N reflétant le fait qu'il y a moins d'énergie dans le signal tronqué) :

In [17]:
```
L=100
x_tronq=zeros(1000)
t=arange(L)
x_tronq[t]=sin(2*pi*fo*t)
plot(x_tronq)
```

```
[<matplotlib.lines.Line2D at 0x7f66ec257810>]
```

Out [17]:

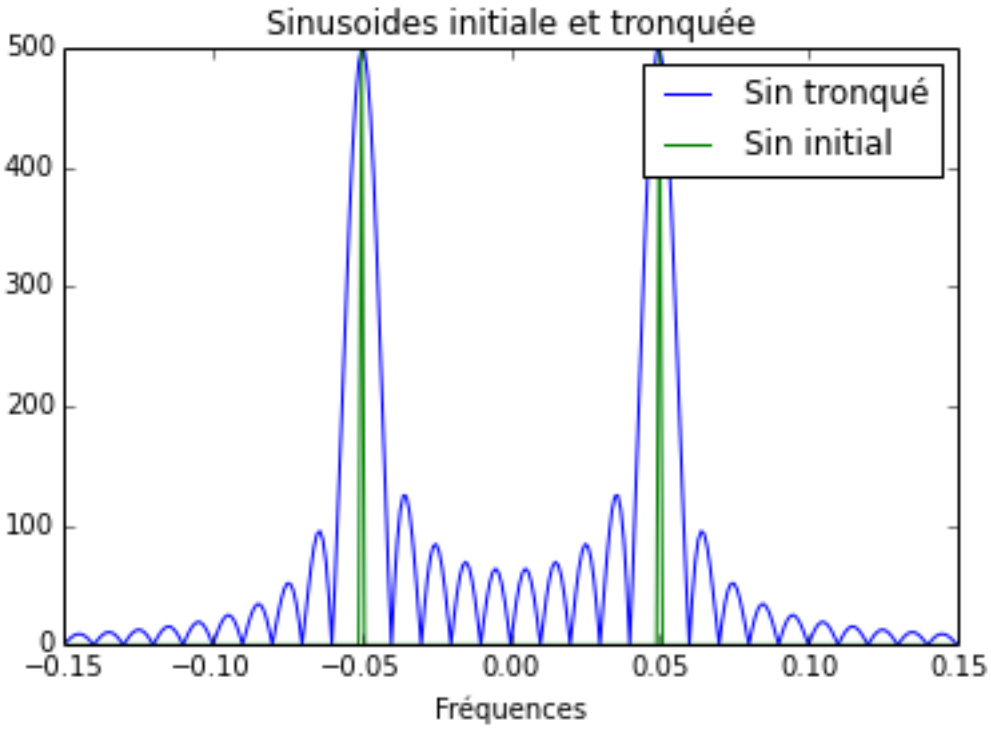


On recalcule et visualise la TF par les mêmes lignes que précédemment [on devrait en faire une fonction]

In [24]:
```
M=1000 # Nombre de points de calcul de la fft : 1000
X_tronq=fft(x_tronq,M)
f=arange(M)/M-1/2
figure(3)
plot(f,N/L*fftshift(abs(X_tronq)),label="Sin tronqué")
plot(f,fftshift(abs(X)),label="Sin initial")
title("Sinusoides initiale et tronquée")
xlabel("Fréquences")
xlim([-0.15, 0.15])
legend()
```

```
<matplotlib.legend.Legend at 0x7f66e37acf10>
```

Out [24]:

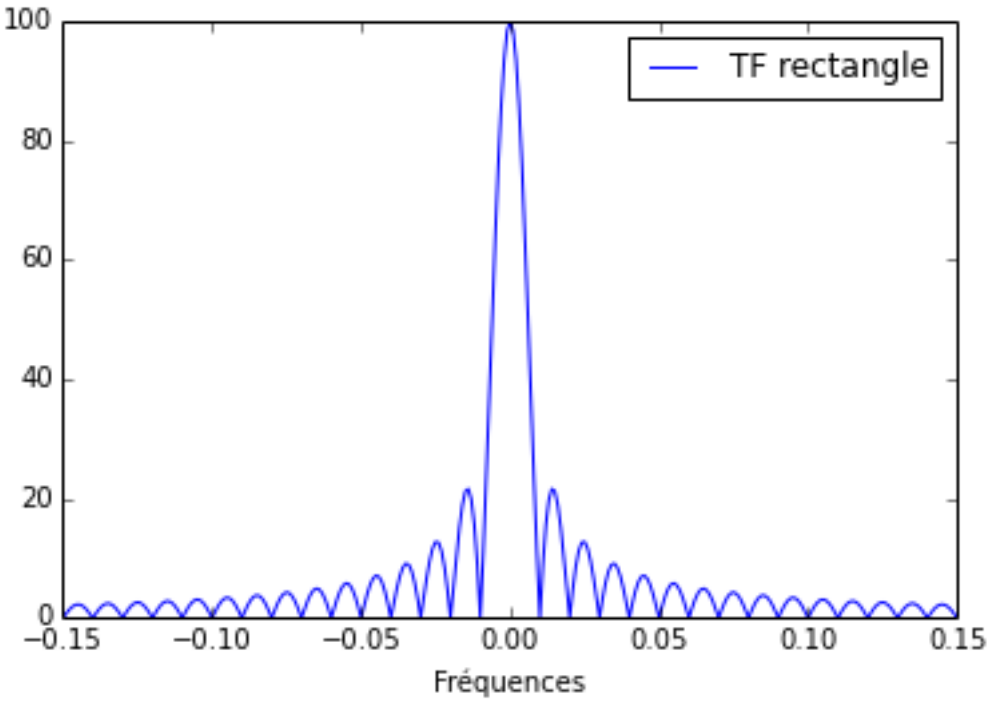


Il est donc apparent que les impulsions de départ ont été “élargies” par la limitation temporelle.En fait, le signal de départ peut-être vu comme ayant été multiplié par une fenêtre rectangulaire pour donner le signal tronqué. Il s’en suit une convolution dans le domaine fréquentiel. Vérifiez ceci en calculant la TF d’une fenêtre rectangulaire et en convoluant celle-ci (fonction `convolve` de numpy) avec la TF de la sinuoïde initiale.

— Convoluez les TF *centrées*, ie après un `fftshift`, sinon il peut y avoir des effets de bords désagréables
— Il y a un facteur N après la convolution - en tenir compte dans la comparaison

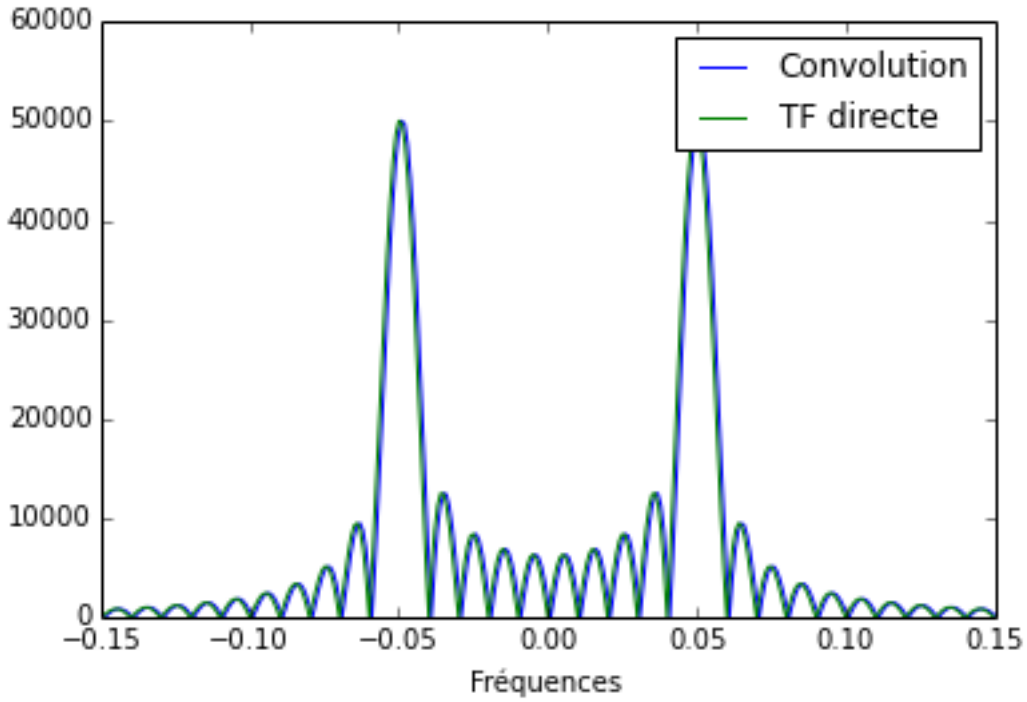
In [29]:
```
z=zeros(1000)
t=arange(L)
z[t]=1
Z=fft(z,N)
figure(4)
plot(f,fftshift(abs(Z)),label="TF rectangle")
xlabel("Fréquences")
xlim([-0.15, 0.15])
legend()
```

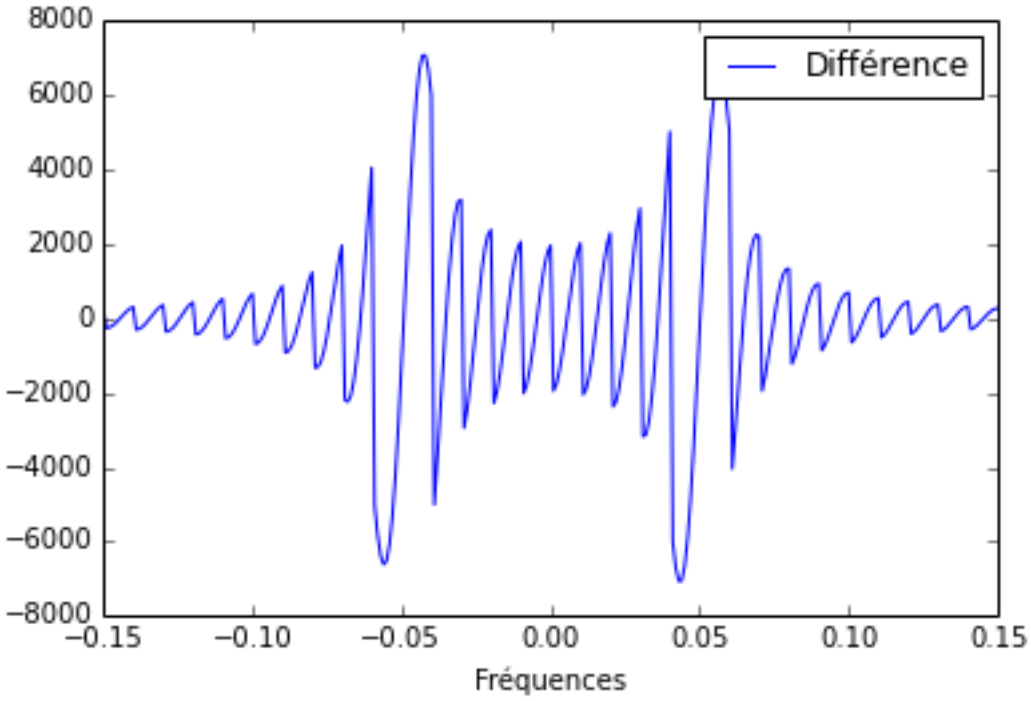
Out [29]:
```
<matplotlib.legend.Legend at 0x7f66ec26dfd0>
```



In [53]:
```
Z=squeeze(Z)
X=squeeze(X)
Res_conv=convolve(fftshift(Z),fftshift(X),'same')

figure(5)
plot(f,(abs(Res_conv)),label="Convolution")
plot(f,N*fftshift(abs(X_tronq)),label="TF directe")
xlabel("Fréquences")
xlim([-0.15, 0.15])
legend()

figure(6)
plot(f,(abs(Res_conv))-N*fftshift(abs(X_tronq)),label="Différence")
xlabel("Fréquences")
xlim([-0.15, 0.15])
legend()
```

Out [53]:
```
<matplotlib.legend.Legend at 0x7f66e3013bd0>
```

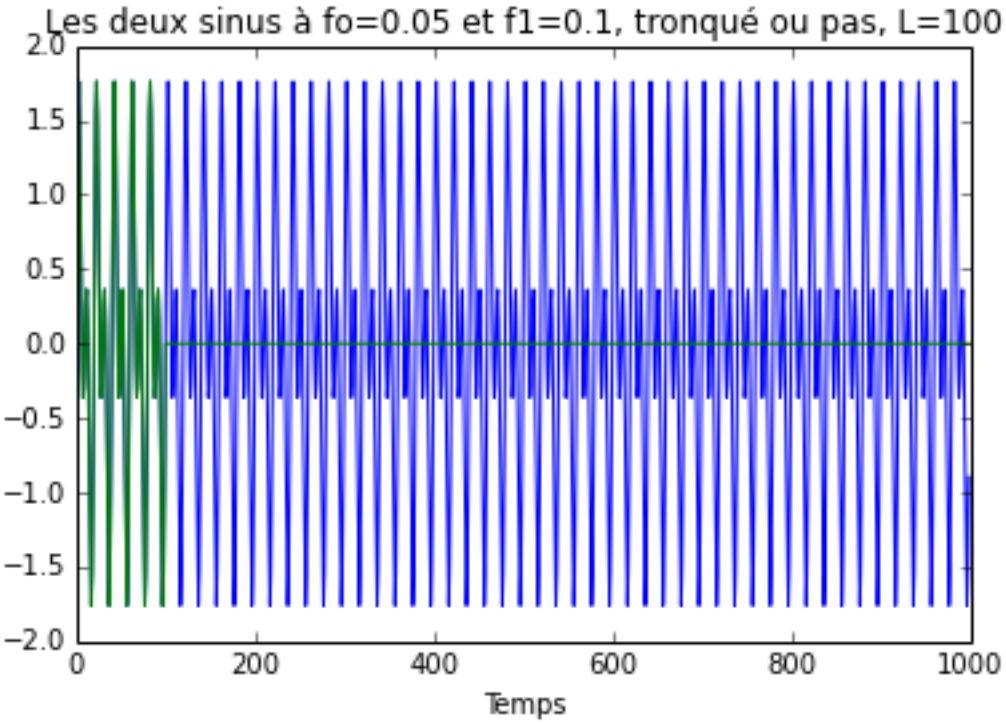


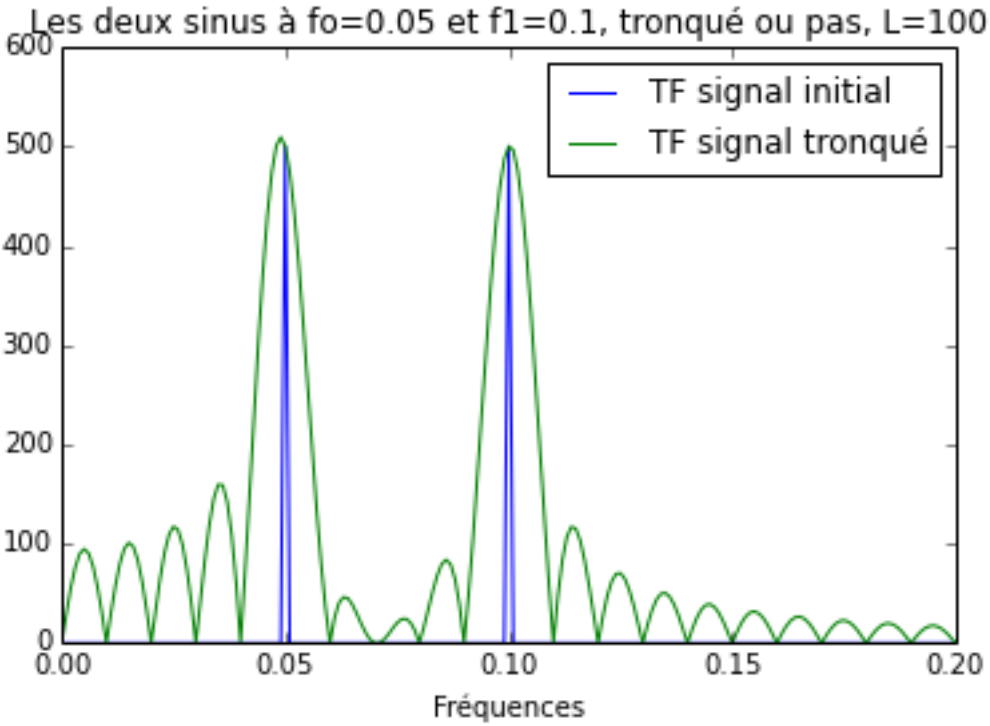


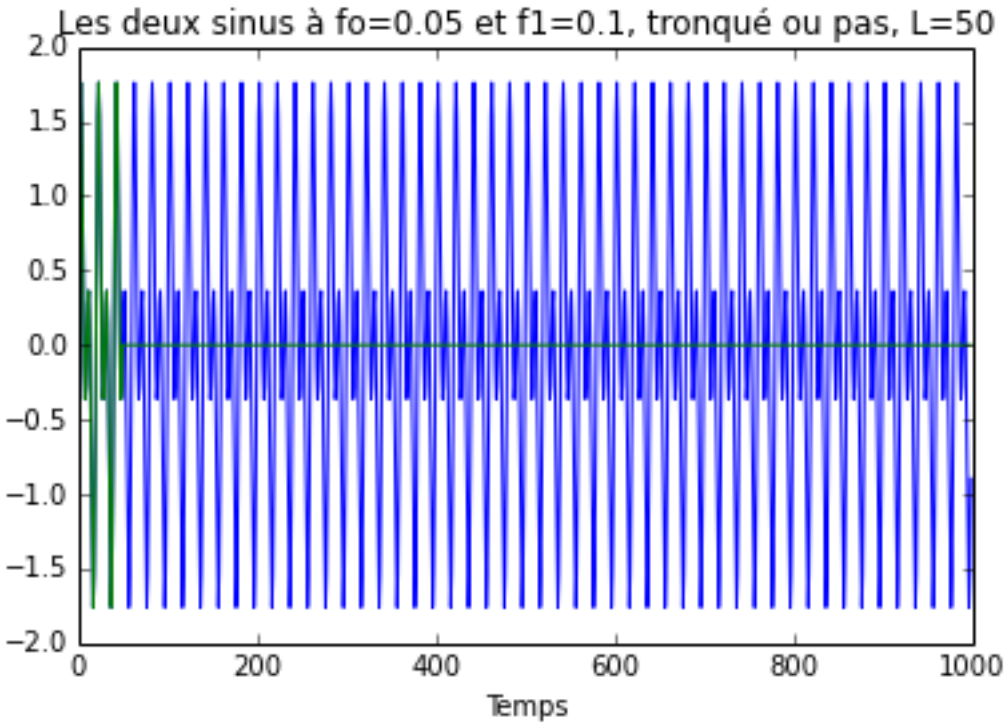
Prendre maintenant la somme de deux sinusoïde, de fréquences respectives 0.05 et 0.1. Ces deux sinusoïdes sont prises sur L points, sur un horizon total de N. Examiner les cas $L = 100$ puis $L = 50$

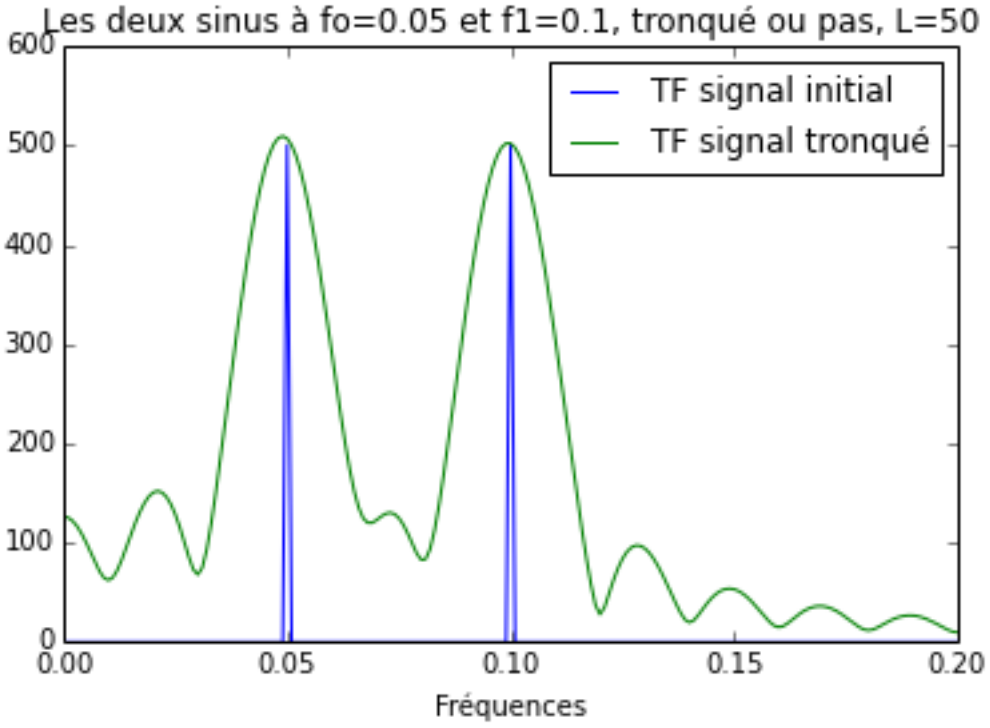
In [88]:
```
fo=0.05
f1=0.1

t=arange(N)
x[t]=sin(2*pi*fo*t)+sin(2*pi*f1*t)

for L in [100, 50]:
        x_tronq=zeros(N)
        x_tronq[arange(L)]=x[arange(L)]
        figure()
```

```
plot(t,x,t,x_tronq)
title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L
xlabel("Temps")
#
figure()
plot(f,abs(fftshift(fft(x,N))),label='TF signal initial')
plot(f,N/L*abs(fftshift(fft(x_tronq,N))),label='TF signal tronqué')
title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L
xlabel("Fréquences")
xlim([0, 0.2])
legend()
```



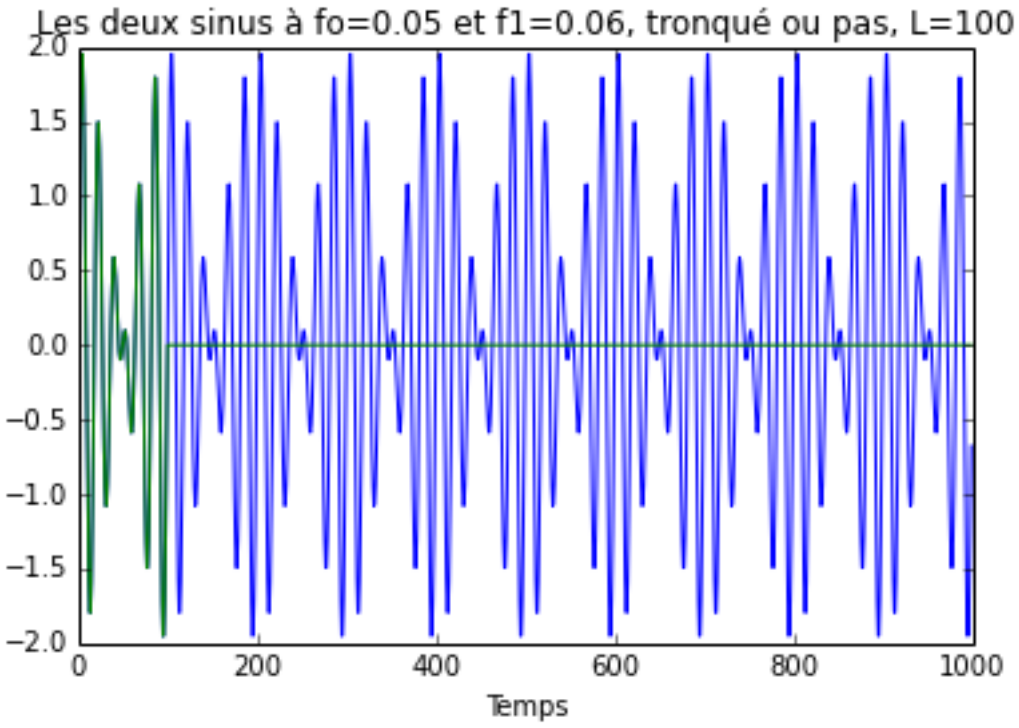


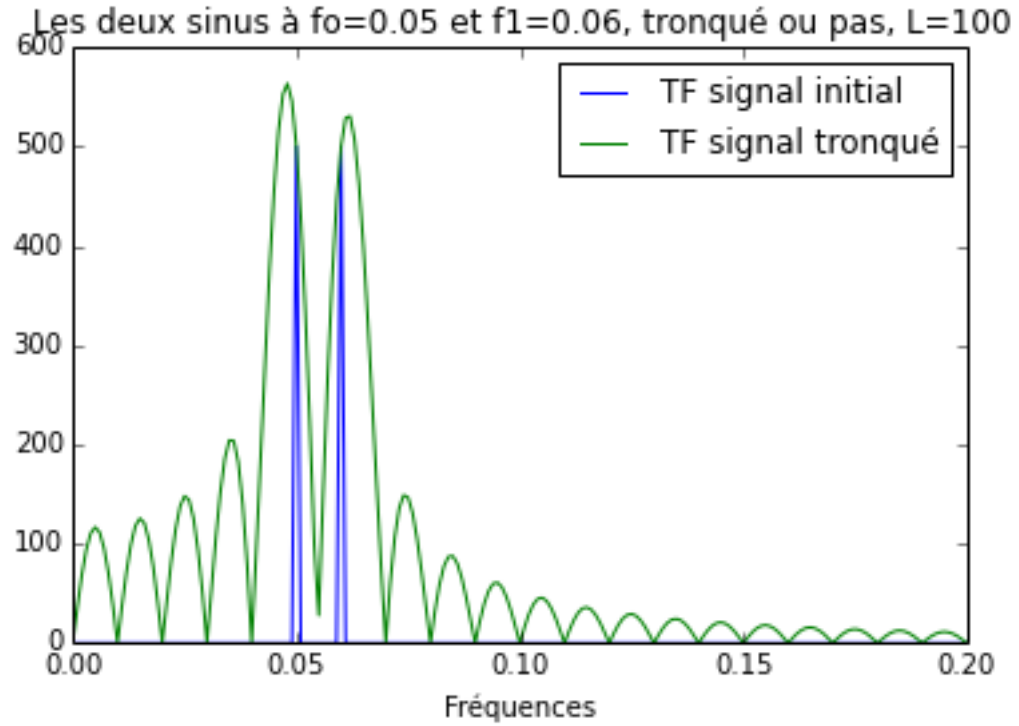


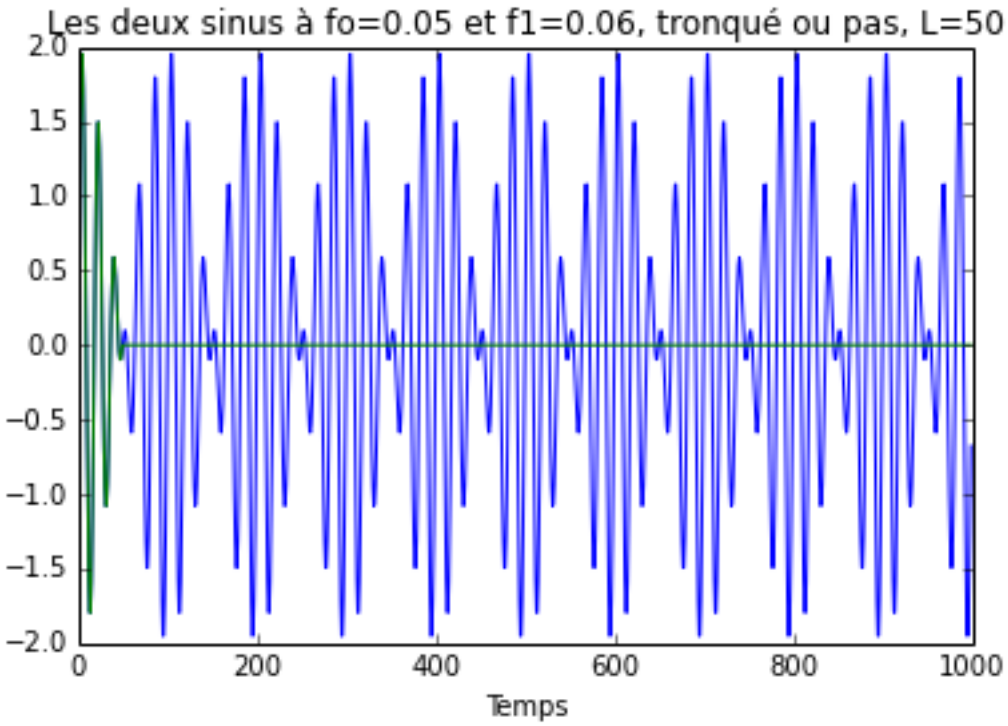


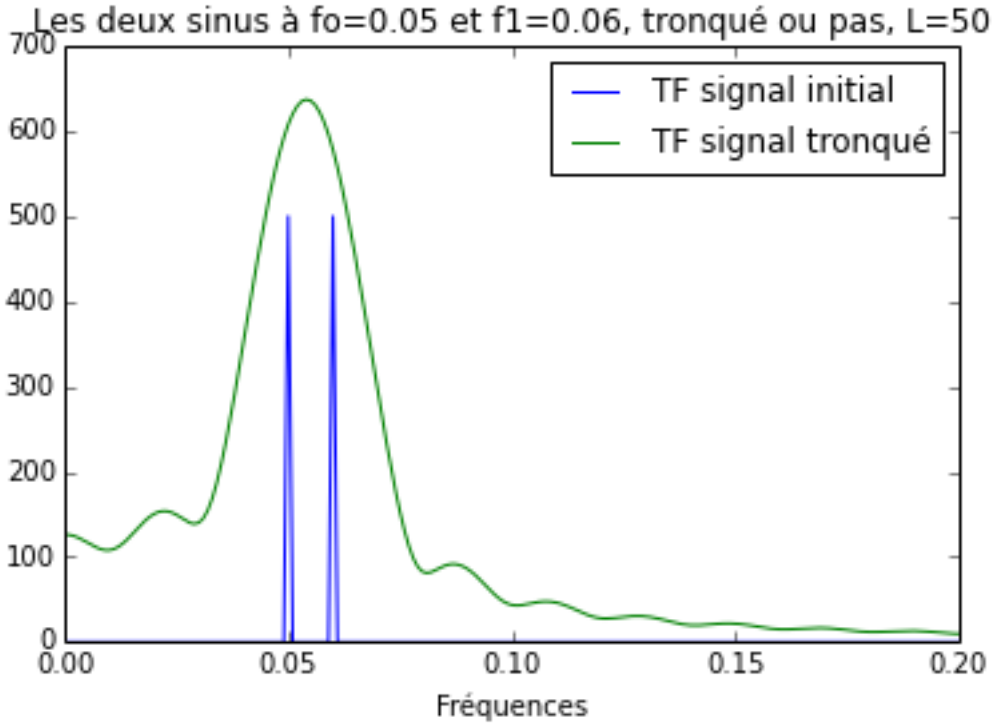
In [97]:
```
fo=0.05
f1=0.06

t=arange(N)
x[t]=sin(2*pi*fo*t)+sin(2*pi*f1*t)

for L in [100, 50]:
        x_tronq=zeros(N)
        x_tronq[arange(L)]=x[arange(L)]
        figure()
        plot(t,x,t,x_tronq)
```

```
title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L
xlabel("Temps")
#
figure()
plot(f,abs(fftshift(fft(x,N))),label='TF signal initial')
plot(f,N/L*abs(fftshift(fft(x_tronq,N))),label='TF signal tronqué')
title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L
xlabel("Fréquences")
xlim([0, 0.2])
legend()
```









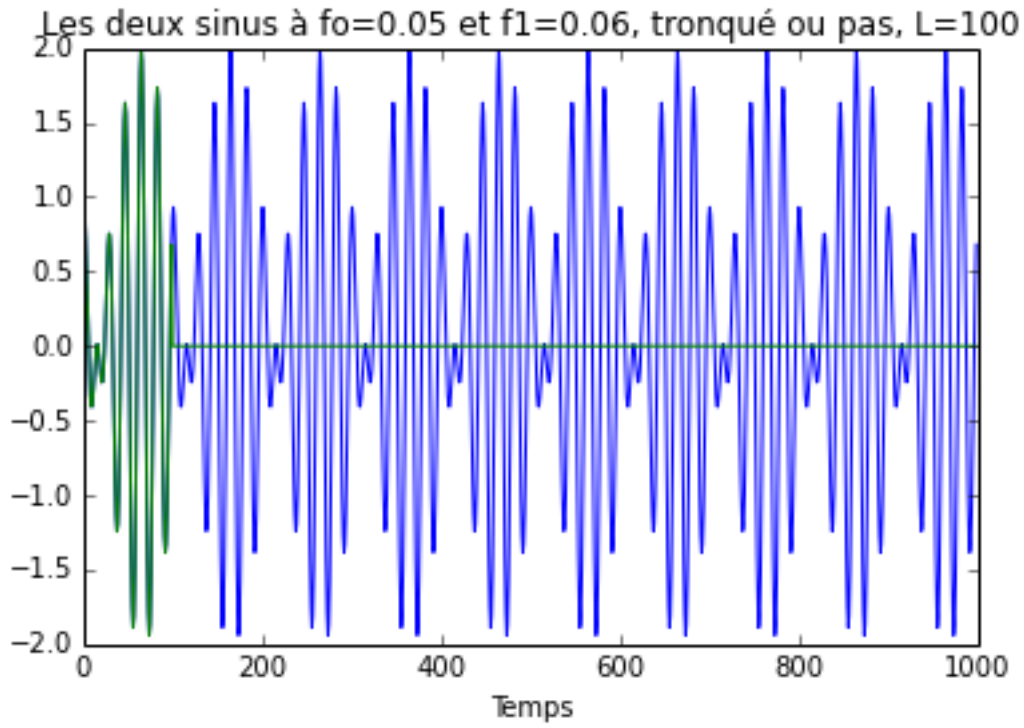
Du fait du recouvrement entre les sinus cardinaux, on note un *déplacement des maxima* et pour L=50, la perte des deux pics. On peut également noter, avec cet écart de fréquences, que le résultat est hautement sensible à la phase des signaux.Expérimentez en modifiant la phase de la seconde sinusoïde, par exemple en premant $\phi = \pi/4$, puis $\phi = 2\pi/3$, dans le modèle

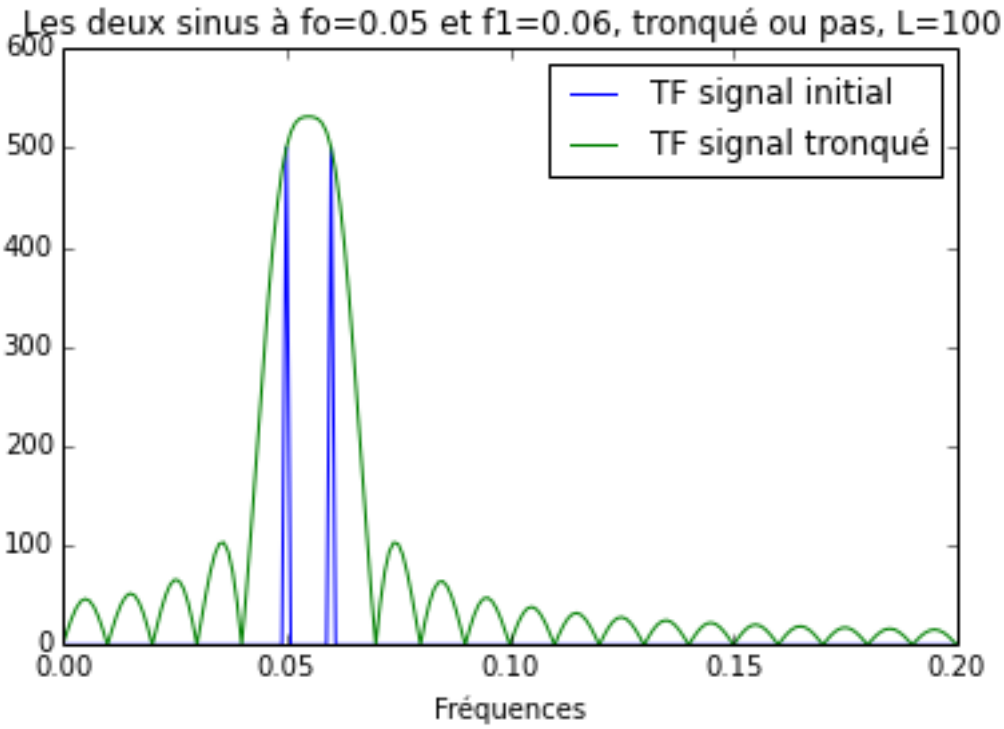
$$x(t) = \sin(2\pi f_O t) + \sin(2\pi f_1 t + \phi) \tag{1}$$

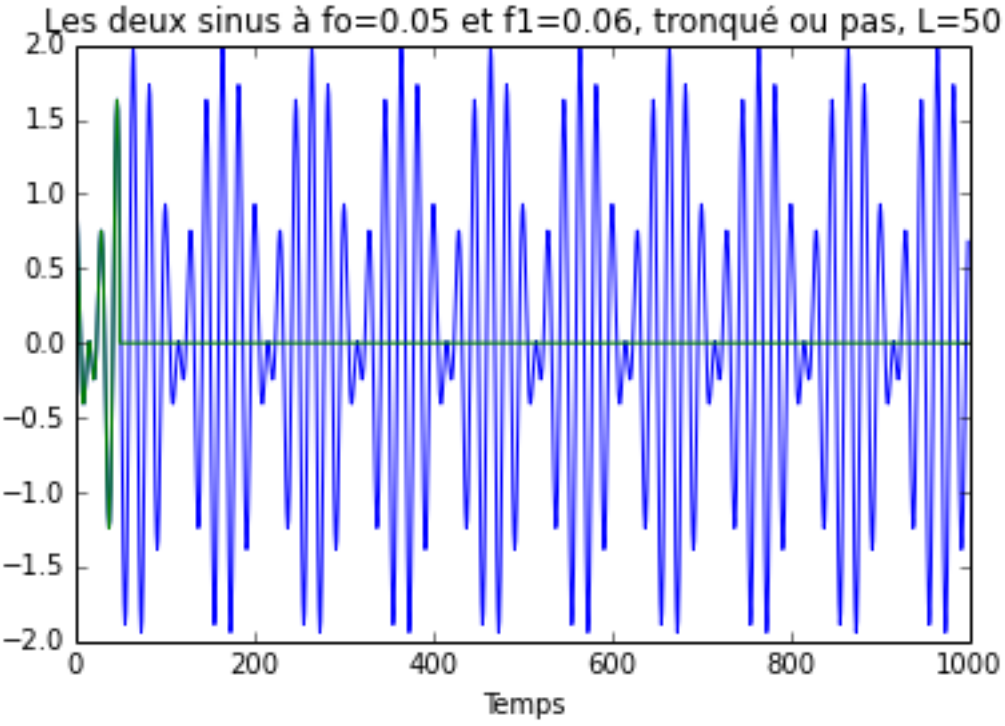
avec $f_0 = 0.05$ et 0.06.

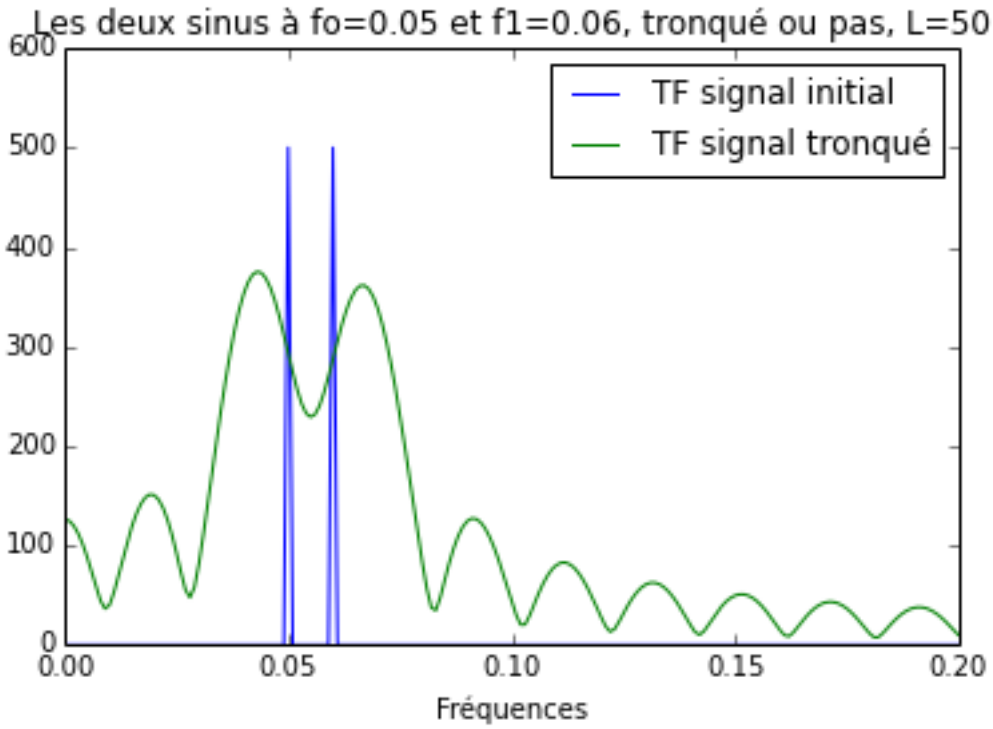
In [98]:

```
fo=0.05
f1=0.06

t=arange(N)
x[t]=sin(2*pi*fo*t)+sin(2*pi*f1*t+2*pi/3)

for L in [100, 50]:
        x_tronq=zeros(N)
        x_tronq[arange(L)]=x[arange(L)]
        figure()
        plot(t,x,t,x_tronq)
        title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L
        xlabel("Temps")
        #
        figure()
        plot(f,abs(fftshift(fft(x,N))),label='TF signal initial')
        plot(f,N/L*abs(fftshift(fft(x_tronq,N))),label='TF signal tronqué')
        title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L
        xlabel("Fréquences")
        xlim([0, 0.2])
        legend()
```




Les deux sinus à fo=0.05 et f1=0.06, tronqué ou pas, L=100
TF signal initial
TF signal tronqué
600
500
400
300
200
100
0
0.00
0.05
0.10
0.15
0.20
Fréquences


Les deux sinus à fo=0.05 et f1=0.06, tronqué ou pas, L=50
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
200
400
600
800
1000
Temps



En fait, on montre que la ** limite de résolution ** de Fourier est en $1/L$, où L est la longuuer d'observation. On ne peut pas espérer distinguer deux événements (deux raies typiquement si ils sont espacés en fréquence de moins de $1/L$).

1.2 Masquage et fenêtrage

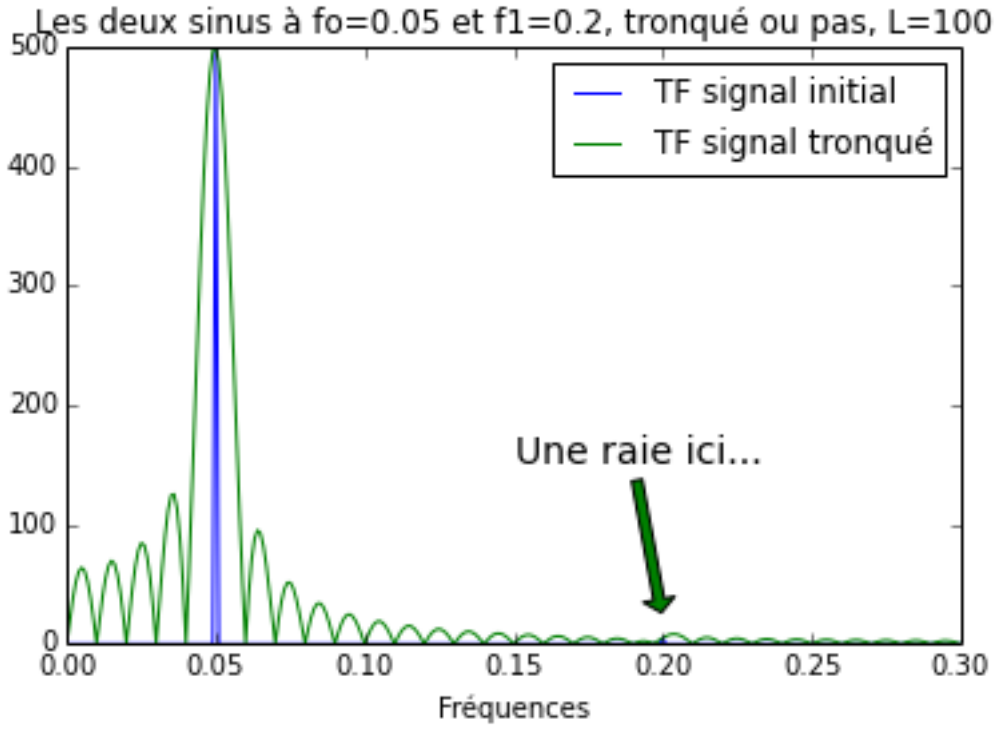
Un autre problème est celui du masquage des signaux faibles. Pour le constater, il suffit par exemple de considérer la somme de deux sinusoïdes de fréquences respectives $f_0 = 0.05$ et $f_1 = 0.2$ et d'amplitudes $A_0 = 1$ et $A_1 = 0.01$. Simulez ce signal et considérez sa transformée de Fourier. Vous aurez intérêt à tracer la représentation en décibels : prendre $20\log(|X(f)|)$,

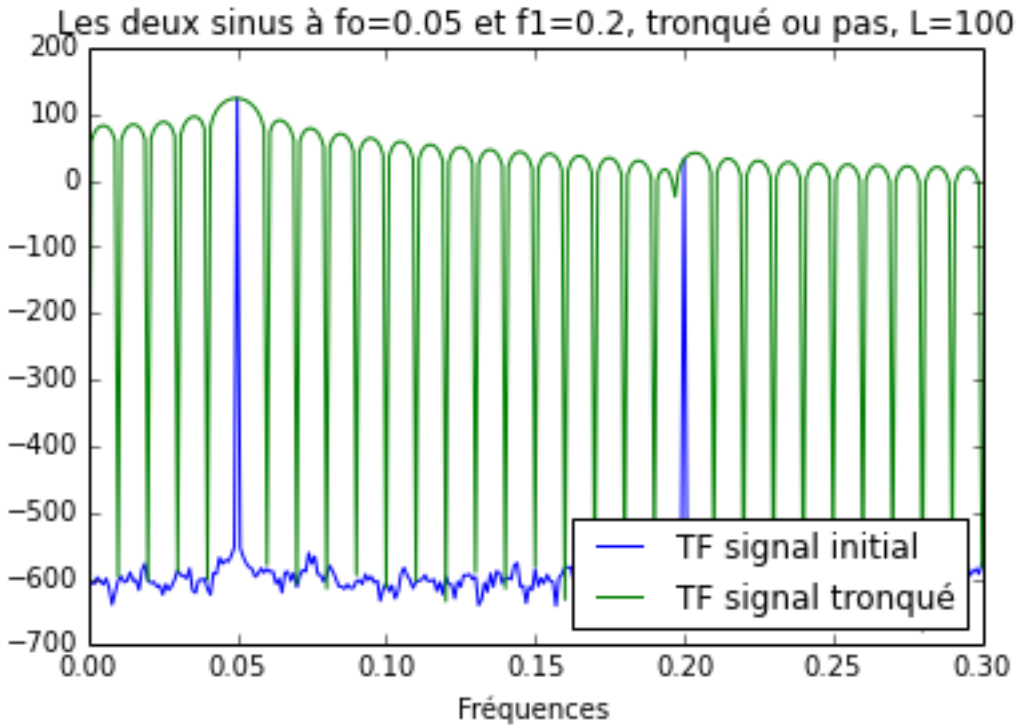
In [137]:
```
fo=0.05
f1=0.2
A0, A1=1, 0.01

t=arange(N)
x[t]=A0*sin(2*pi*fo*t)+A1*sin(2*pi*f1*t)

L=100
x_tronq=zeros(N)
x_tronq[arange(L)]=x[arange(L)]


fig,ax=subplots()
ax.plot(f,abs(fftshift(fft(x,N))),label='TF signal initial')
ax.plot(f,N/L*abs(fftshift(fft(x_tronq,N))),label='TF signal tronqué')
title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L))
xlabel("Fréquences")
xlim([0, 0.3])
legend()
ax.annotate('Une raie ici...',
            xy=(0.2,20),
            xytext=(0.15,150),
            arrowprops={'facecolor':'green','shrink':0.05},fontsize=14)


figure()
plot(f,20*log(abs(fftshift(fft(x,N)))),label='TF signal initial')
```

```
plot(f,20*log(N/L*abs(fftshift(fft(x_tronq,N)))),label='TF signal tronqué')
title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L))
xlabel("Fréquences")
xlim([0, 0.3])
legend(loc=4)
```

```
<matplotlib.legend.Legend at 0x7f66e17ff350>
```

Out [137]:





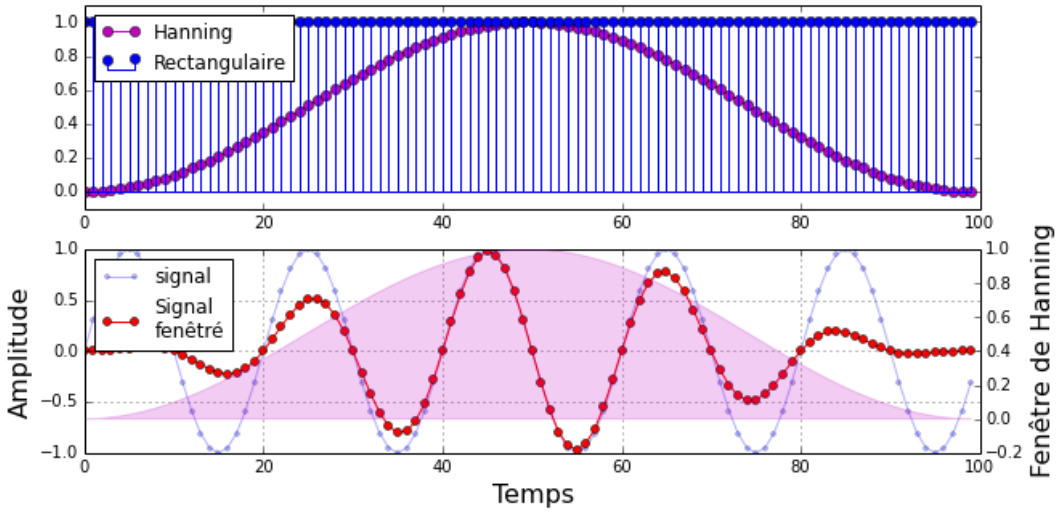
Plutôt que la pondération rectangulaire, prendre une autre fenêtre de pondération, comme la fenêtre de Hamming, et définissez le signal tronqué comme `x_tronq[arange(L)]=x[arange(L)]*sp.hanning(L)`.

Pour celà, importez le module signal de scipy, par `import scipy.signal as sp` Consultez l'aide sur `sp.hanning`, puis comparez là à la fenêtre rectangulaire `sp.boxcar`.

Tracez les représentations en fréquence, en échelle log. Conclure.

In [147]:
```
import scipy.signal as sp
# help(sp)  Pour avoir toute l'aide
# help(sp.hamming)
```

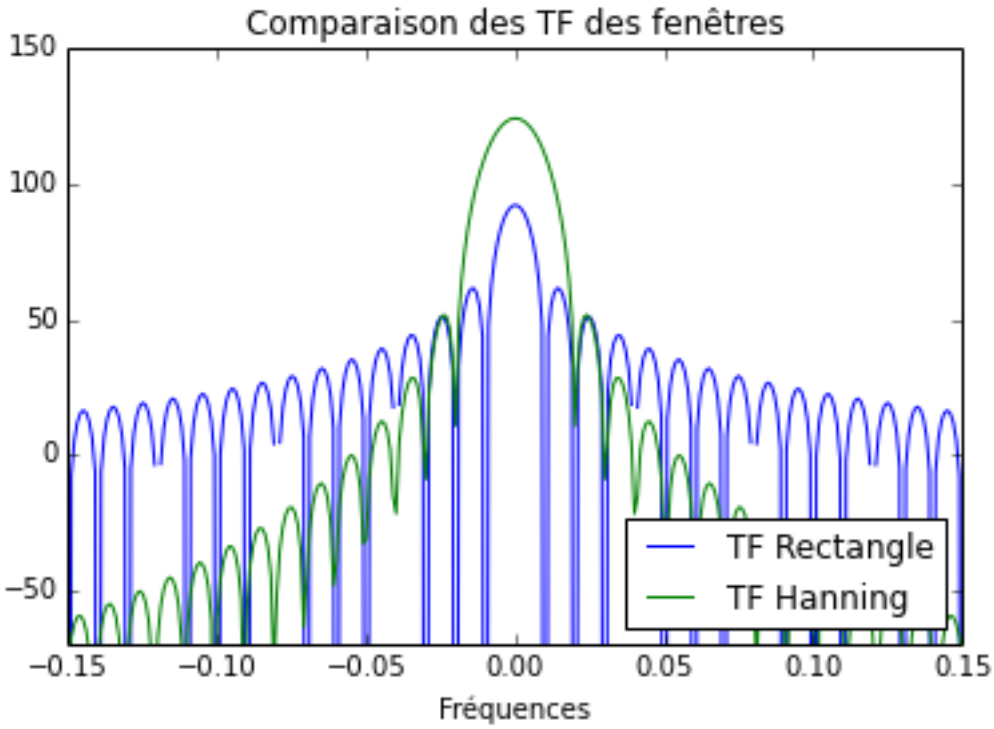
In [163]:
```
fig,axs = subplots(2,1)
fig.set_size_inches((10,5))

window = sp.hanning(L)
rectwin = sp.boxcar(L)
ax = axs[0]

ax.plot(arange(L),window,'m-o',label='Hanning')
ax.stem(arange(L),rectwin,basefmt='b-',label='Rectangulaire')
ax.set_ylim(ymax = 1.1,ymin=-0.1)
ax.legend(loc=2)

t=arange(L)
x=zeros(L)
x[t]=A0*sin(2*pi*fo*t)

ax = axs[1]
ax.plot(t,x,'-o',label='signal',ms=3.,alpha=0.3)
ax.plot(t,window*x,'-or',label='Signal\nfenêtré',ms=5.)
ax.set_ylabel('Amplitude',fontsize=16)
ax.set_xlabel('Temps',fontsize=16)
ax.legend(loc=2)
ax.grid()
ax2 = ax.twinx()
ax2.fill_between(t,window,alpha=0.2,color='m')
ax2.set_ylabel('Fenêtre de Hanning',fontsize=16);
```



Comparaison dans le domaine fréquentiel :

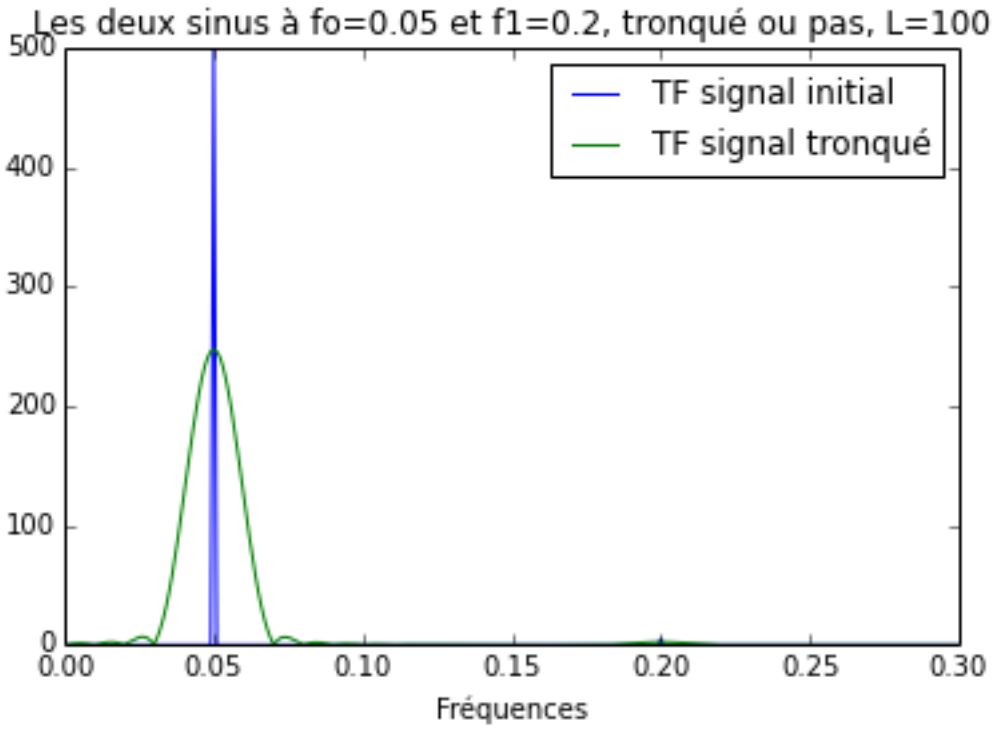
In [173]:
```
figure()
plot(f,20*log(abs(fftshift(fft(sp.boxcar(L),N)))),label='TF Rectangle')
plot(f,20*log(N/L*abs(fftshift(fft(sp.hanning(L),N)))),label='TF Hanning')
title("Comparaison des TF des fenêtres")
xlabel("Fréquences")
xlim([-0.15, 0.15])
ylim([-70, 150])
legend(loc=4)
```

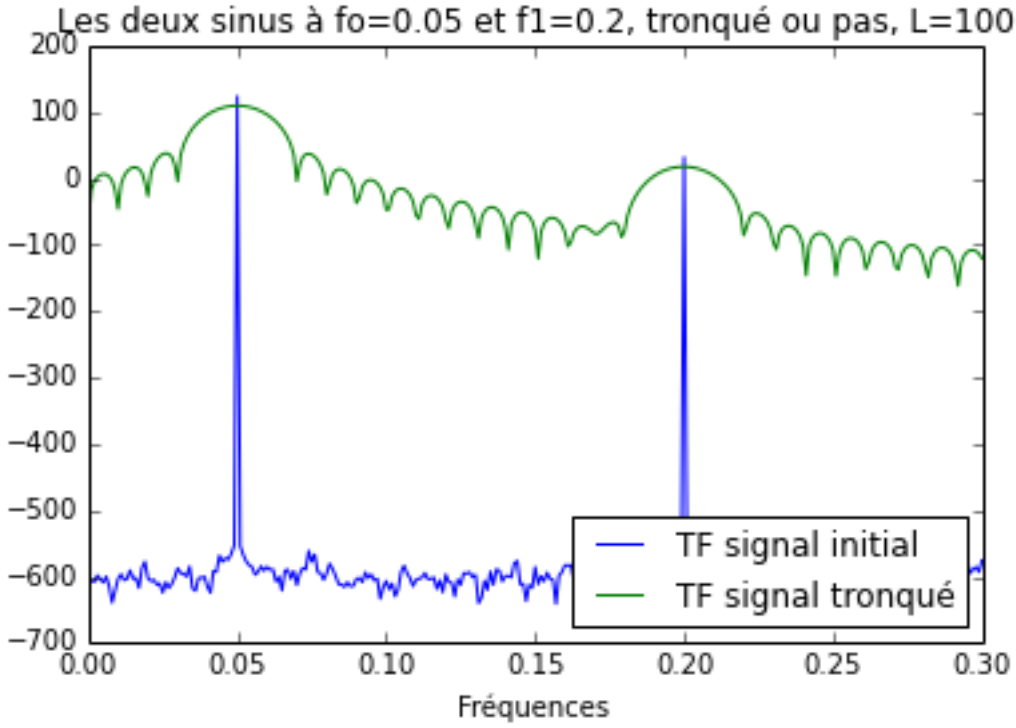
Out [173]:
```
<matplotlib.legend.Legend at 0x7f66d141eed0>
```



In [169]:
```
fo=0.05
f1=0.2
A0, A1=1, 0.01

t=arange(N)
x=zeros(N)
x[t]=A0*sin(2*pi*fo*t)+A1*sin(2*pi*f1*t)

L=100
x_tronq=zeros(N)
x_tronq[arange(L)]=x[arange(L)]*hanning(L)

figure()
plot(f,abs(fftshift(fft(x,N))),label='TF signal initial')
plot(f,N/L*abs(fftshift(fft(x_tronq,N))),label='TF signal tronqué')
title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L))
xlabel("Fréquences")
xlim([0, 0.3])
legend()

figure()
plot(f,20*log(abs(fftshift(fft(x,N)))),label='TF signal initial')
plot(f,20*log(N/L*abs(fftshift(fft(x_tronq,N)))),label='TF signal tronqué')
title("Les deux sinus à fo={} et f1={}, tronqué ou pas, L={}".format(fo,f1,L))
xlabel("Fréquences")
xlim([0, 0.3])
legend(loc=4)
```

Out [169]:
```
<matplotlib.legend.Legend at 0x7f66d18c3310>
```





1.3 Interpolation de Shannon

Il s'agit de vérifier la formule d'interpolation de Shannon pour un signal correctement échantillonné :

$$x(t) = \sum_{n=-\infty}^{+\infty} x(nT_e) \frac{\sin(\pi F_e(t - nT_e))}{\pi F_e(t - nT_e)} \tag{2}$$

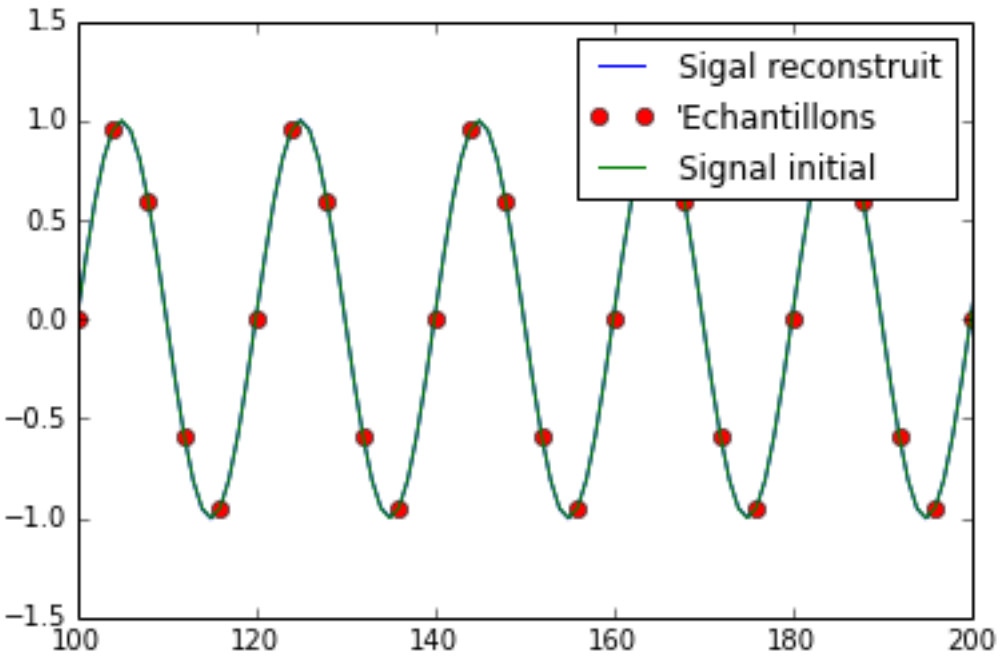
Pour ce faire, vous créerez une sinusoïde de fréquence f_0, que vous échantillonnerez à 4 échantillons par période

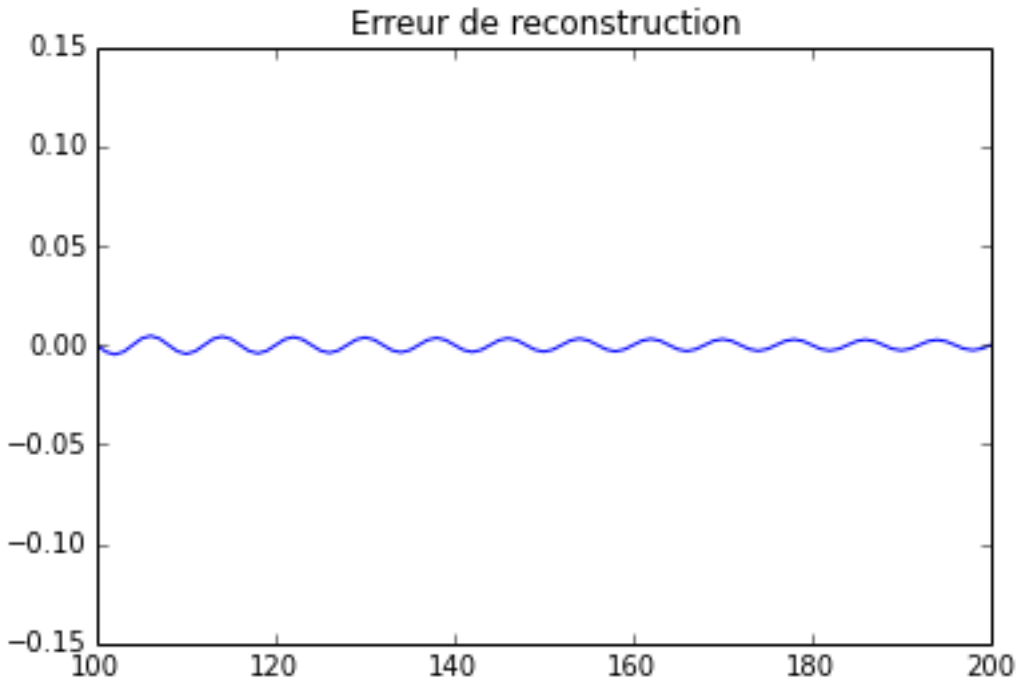
($F_e = 4f_0$), vous implanterez la formule d'interpolation et comparerez l'approximation (nombre fini de termes dans la somme) au signal initial.

In [198]:
```
N=2000
t=arange(N)
fo=0.05 #--> 1/fo=20 points par période
x=sin(2*pi*fo*t)
ts=arange(0,N,4) # 5 echantillons par période
num=size(ts) # nombre d'échantillons
xe=sin(2*pi*fo*ts)

Te,Fe=4,1/4
x_rec=zeros(N)
for n in range(num):
    x_rec=x_rec+xe[n]*sinc(Fe*(t-n*Te)) #! la définition du sinc contient le pi

plot(t,x_rec,label="Sigal reconstruit")
plot(ts,xe,'ro',label="\'Echantillons")
plot(t,x,'-g',label="Signal initial")
xlim([100,200])
legend()

figure()
plot(t,x-x_rec)
title("Erreur de reconstruction")
xlim([100,200])
```

```
(100, 200)
```

Out [198]:



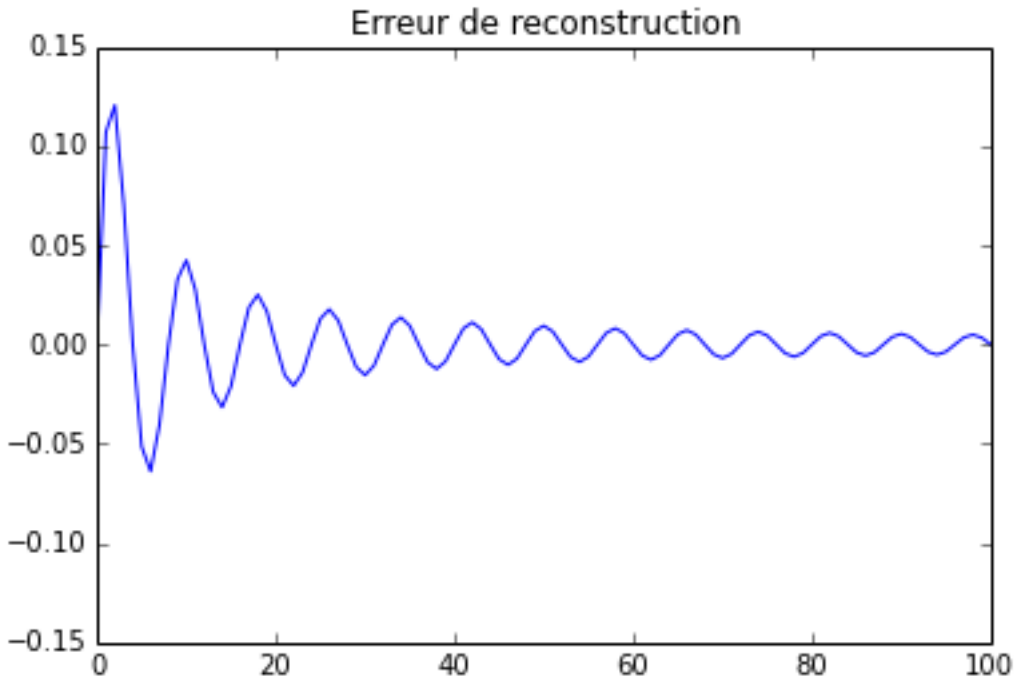


Si on regarde attentivement, on voit que l'erreur est plus importante sur les bords de l'intervalle. En effet, dans le théorème de Shannon et la formule de reconstruction, le signal est supposé de durée infinie (car à bande limitée), ce qui n'est pas le cas ici, et il manque ainsi des termes en sinc pour parfaire la reconstruction.

In [199]:
```
figure()
plot(t,x-x_rec)
title("Erreur de reconstruction")
xlim([0,100])
```

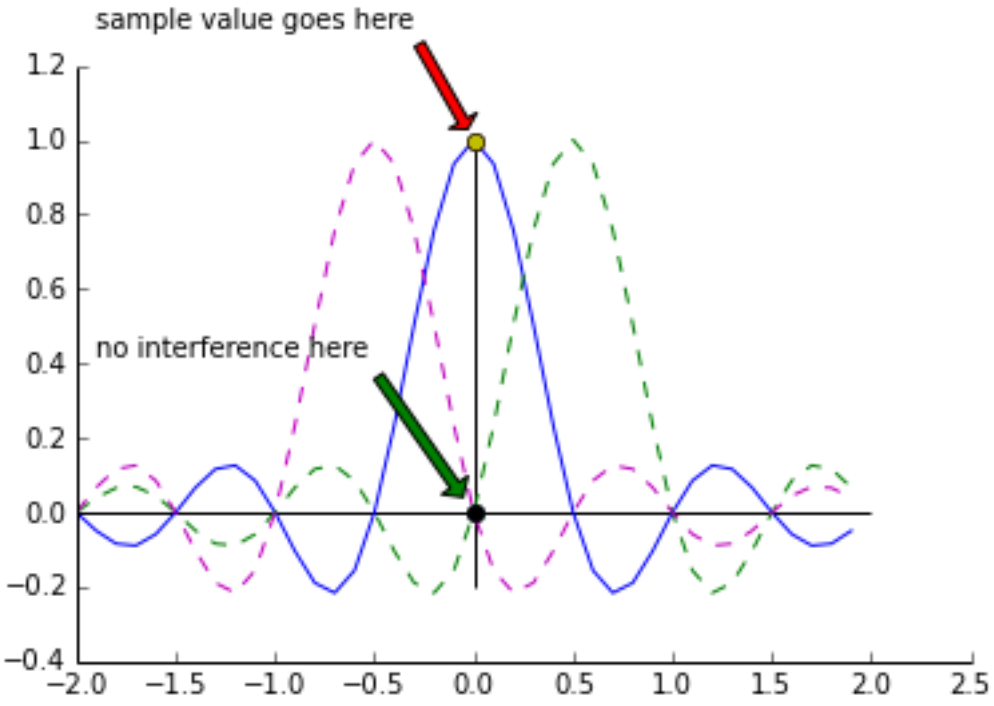
```
(0, 100)
```

Out [199]:



Une jolie figure, extraite de Python for Signal Processing, José Unpingco, 2014. Cool, non ?

In [217]:
```
t=arange(-2,2,0.1)
fig = figure()
ax = fig.add_subplot(111) # create axis handle
k=0
fs=2 # makes this plot easier to read
ax.plot (t,sinc( k - fs * t),
         t,sinc( k+1 - fs * t),'--',k/fs,1,'o',(k)/fs,0,'o',
         t,sinc( k-1 - fs * t),'--',k/fs,1,'o',(-k)/fs,0,'o'
)
ax.hlines(0,-2,2)
ax.vlines(0,-.2,1)
ax.annotate('sample value goes here',
            xy=(0,1),
            xytext=(-2+.1,1.3),
            arrowprops={'facecolor':'red','shrink':0.1},
            )
ax.annotate('no interference here',
            xy=(0,0),
            xytext=(-2+.1,0.42),
            arrowprops={'facecolor':'green','shrink':0.1},
            )
ax.spines['top'].set_color('none') #jfb
ax.xaxis.set_ticks_position('bottom') #jfb
ax.spines['right'].set_color('none') #jfb
ax.yaxis.set_ticks_position('left') #jfb
#ax.spines['left'].set_position('center')
#ax.spines['bottom'].set_position('center')
```



1.4 La TFD : échantillonnage de la TF

La TFD peut être comprise simplement comme résultant de l'échantillonnage de la TF à fréquence continue d'une séquence discrète, sur une grille régulière. Il est possible d'illustrer très simplement ceci par une petite expérimentation :

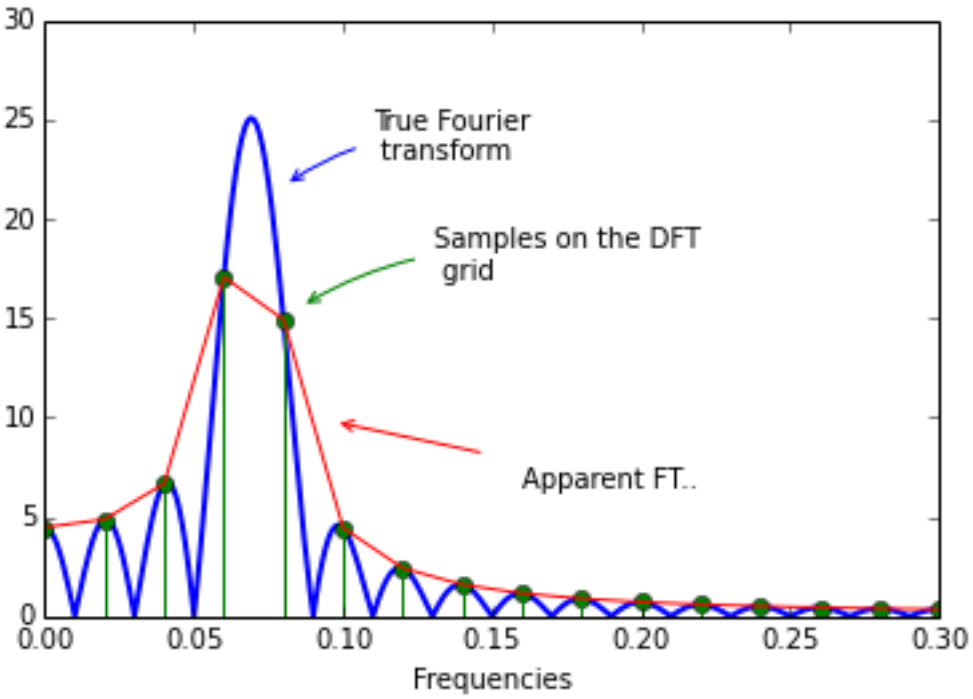
1. Créez une sinusoïde de fréquence $f_0 = 0.07$, sur $N = 50$ points, et
2. Calculez et tracez (fonction `plot`) sa transformée de Fourier `Sz` calculée sur 1000 points `Sz=fft(s,1000)`, et ce en fonction d'un vecteur de fréquences `f` correctement défini sur $[0, 1$
3. Sur le même graphique, représentez (fonction `stem`) la TF calculée uniquement sur N points `S=fft(s)`, et ce en fonction d'un vecteur de fréquences `f2` correctement défini sur $[0, 1$

4. Enfin, ajoutez sur le graphique cette même TF où les points sont reliés par des segments de doite (tout simplement la fonction `plot`)

In [3]:
```
from numpy.fft import fft, ifft
##
# experiments on DFT: the DFT as sampled FT
N=50          # Fourier resolution: 1/N
fo=0.07       # not on the Fourier grid
t=arange(N)
s=sin(2*pi*fo*t)
Sz=fft(s,1000)
f=arange(1000)/1000
plot(f,abs(Sz),lw=2, color="blue")
S=fft(s)
f2=arange(N)/N
stem(f2,abs(S),lw=2,linefmt='g-', markerfmt='go')
plot(f2,abs(S),'r-')
xlabel("Frequencies")

# ici on s'amuse un brin à mettre des annotations et des flèches
annotate("True Fourier\n transform", xy=(0.075,21), xytext=(0.11,23),
         arrowprops=dict(arrowstyle="->",
                         color="blue",
         connectionstyle="arc3,rad=0.2",
         shrinkA=5, shrinkB=10))

annotate("Samples on the DFT\n grid", xy=(0.08,15), xytext=(0.13,17),
         arrowprops=dict(arrowstyle="->",
                         color="green",
         connectionstyle="arc3,rad=0.2",
         shrinkA=5, shrinkB=10))
annotate("Apparent FT..", xy=(0.09,10), xytext=(0.16,6.5),
         arrowprops=dict(arrowstyle="->",
                         color="red",
         connectionstyle="arc3,rad=-0.0",
         shrinkA=15, shrinkB=10))
xlim([0, 0.3])
```

```
(0, 0.3)
```

Out [3]:



On constate ainsi, que sans précautions ni analyse, il est aisé de se laisser abuser par l'allure d'une TF. Faire un bourrage de zéros (i.e. calculer la TF de la séquence complétée par des zéros) permet souvent d'éviter de mauvaises interprétations.

1.5 Spectrogramme

Pour des signaux dont le contenu fréquentiel varie au cours du temps, plutôt que faire une analyse globale, qui finalement revient à moyenner les variations de fréquences, on eut utiliser une *transformée de Fourier à court terme*. Celle-ci consiste à calculer le TF d'une une portion localisée du signal, après pondération par une fenêtre (afin d'éviter les lobes secondaires de la fenere rectangulaire –cf plus haut). On obtient alors une transformée de Fourier, qui dépend non seulement de la fréquence, mais aussi de la localisation temporelle

$$X(f,\tau) = \mathrm{TF}\{x(n).w(n-\tau)\} = \sum_n x(n).w(n-\tau)e^{-j2\pi fn} \tag{3}$$

où $w(n-\tau)$ désigne la fenêtre $w(n)$ décalée de τ. Par suite, il est possible de construire une représentation conjointe, *en temps et en fréquence*, en collectionnant l'ensemble des transformées à court terme. Cette représentation conjointe, sous la forme d'une image du contenu fréquentiel en fonction du temps, s'appelle un *Spectrogramme*.Vous allez chercher à contruire un spectrogramme élémentaire, puis annalyser quelques signaux. Pour ce faire, vous créerez une fonction `spectro`, qui parcourera tous les échantillons `n` du signal d'entrée et, pour tout n, devra :

1. Extraire une portion du signal de départ `x` et le pondérer par une fenêtre : `xw=x[n:n+L]*window`
2. Calculer la TF correspondante et ranger le résultat dans un tableau Xftmp=fft(xw,N) Xwf[0 :N/2+1,k]=Xftmp[0 :N/2+1] Une fois que la boucle sur le temps aura été effectuée, et que vous aurez correctement initialisé les différentes variables (notamment celle-ci : `Xwf=zeros((N/2+1,Nx))+0j`) :
3. Il ne restera plus alors qu'à afficher la matrice résultante sous la forme d'une image imshow(abs(Xwf), aspect='auto', origin='lower',extent=(0, Nx, 0, 0.5))

Vous pourrez tester ce spectrogramme sur différents signaux : un cri (bat.wav) de chauve-souris, un son (gong.wav) de gong ou un chant (bruant.wav) de bruant.

Pour ce faire, vous importerez le module qui va bien

```
from scipy.io.wavfile import read as wavread, write as wavwrite
```

et lirez le fichier par :

```
(r, x) =wavread('bat.wav')
```

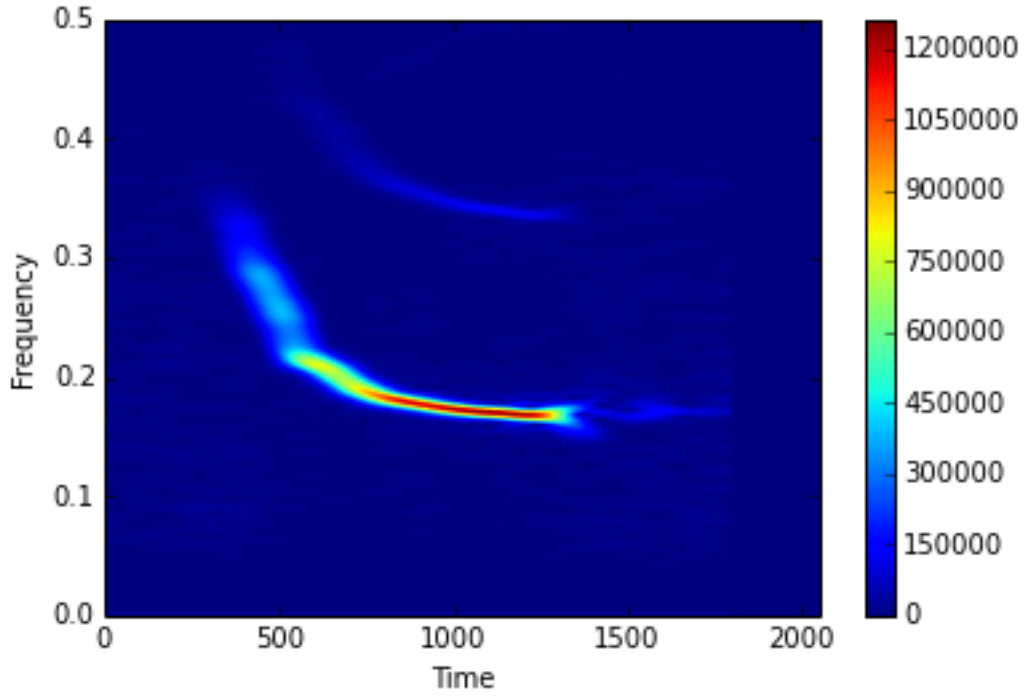
Vous représenterez à la fois le spectrogramme et le signal en fonction du temps.

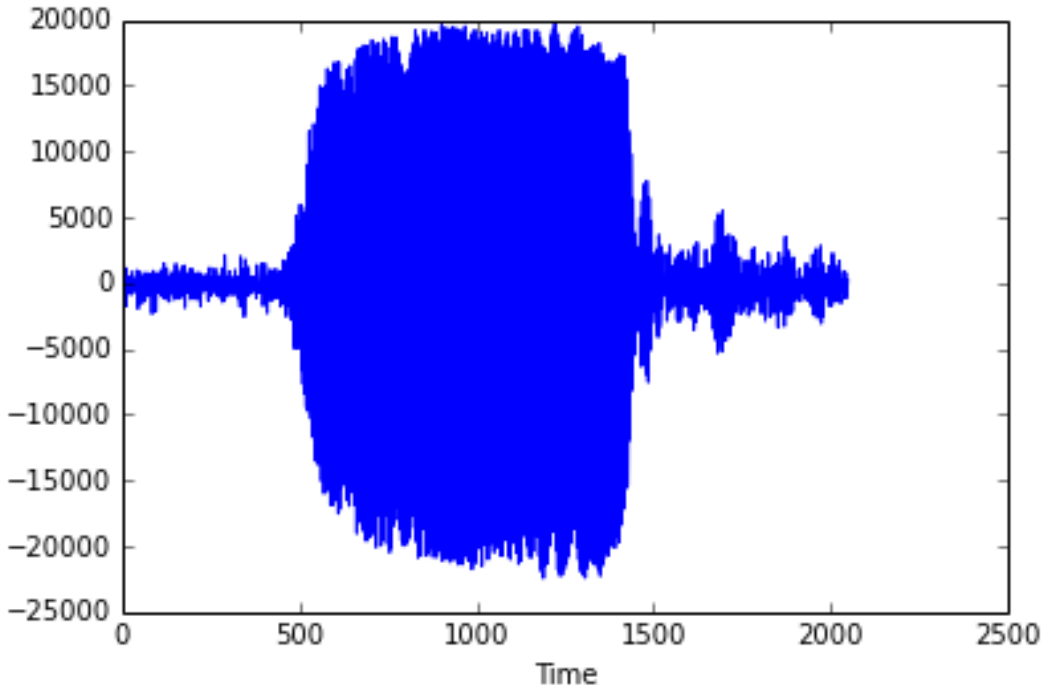
Une implantation basique est la suivante :

In [5]:
```
def spectrobasic(x,L=256, N=1024, window=hanning(256)):
    """
    Paramètres :
    ============
    x : signal d'entrée \n
    L : longueur des fenêtres (par défaut L=256) \n
    N : nombre de poins de calcul de la TF \n
    window : fenêtre de pondération --- par défaut hanning(256)
    """
    Nx=size(x)
    k=0
    Xwf=zeros((N/2+1,Nx))+0j #!! force complex type !!
    for n in range(0,Nx-L):
        xw=x[n:n+L]*window
        Xftmp=fft(squeeze(xw),N)
        Xwf[0:N/2+1,k]=Xftmp[0:N/2+1]
        k+=1
    figure()
    imshow(abs(Xwf), aspect='auto', origin='lower',extent=(0, Nx, 0, 0.5) )
    xlabel("Time")
    ylabel("Frequency")
    colorbar()
    figure()
    plot(x)
    xlabel("Time")
    return Xwf
```

Et le test sur le son de chauve-souris :

In [6]:
```
from scipy.io.wavfile import read as wavread, write as wavwrite
(r, x) =wavread('bat.wav')
out=spectrobasic(x)
```

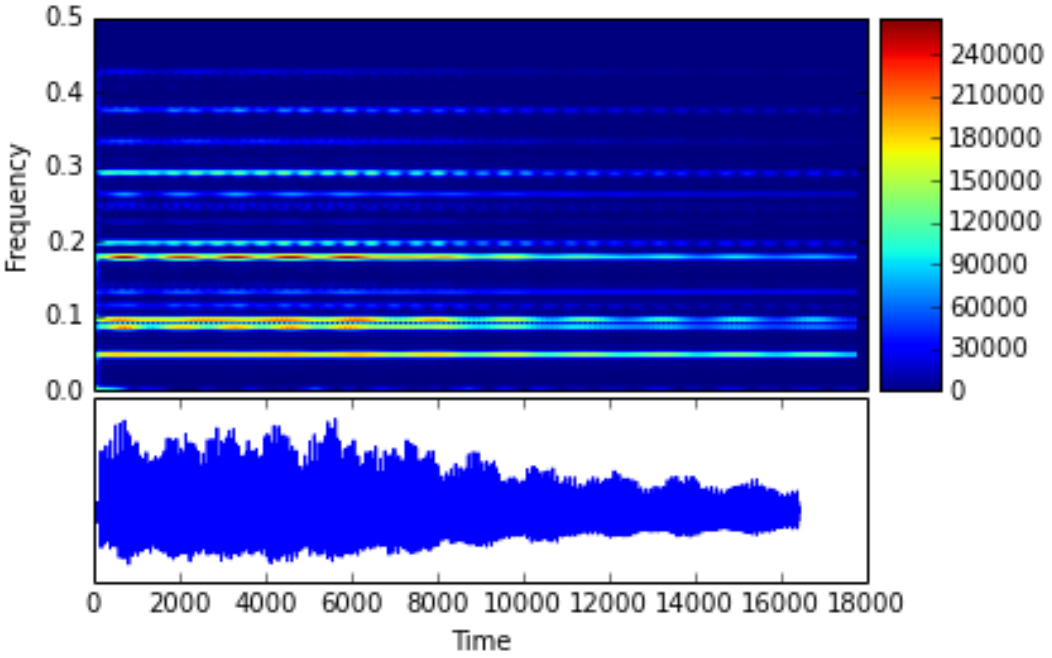




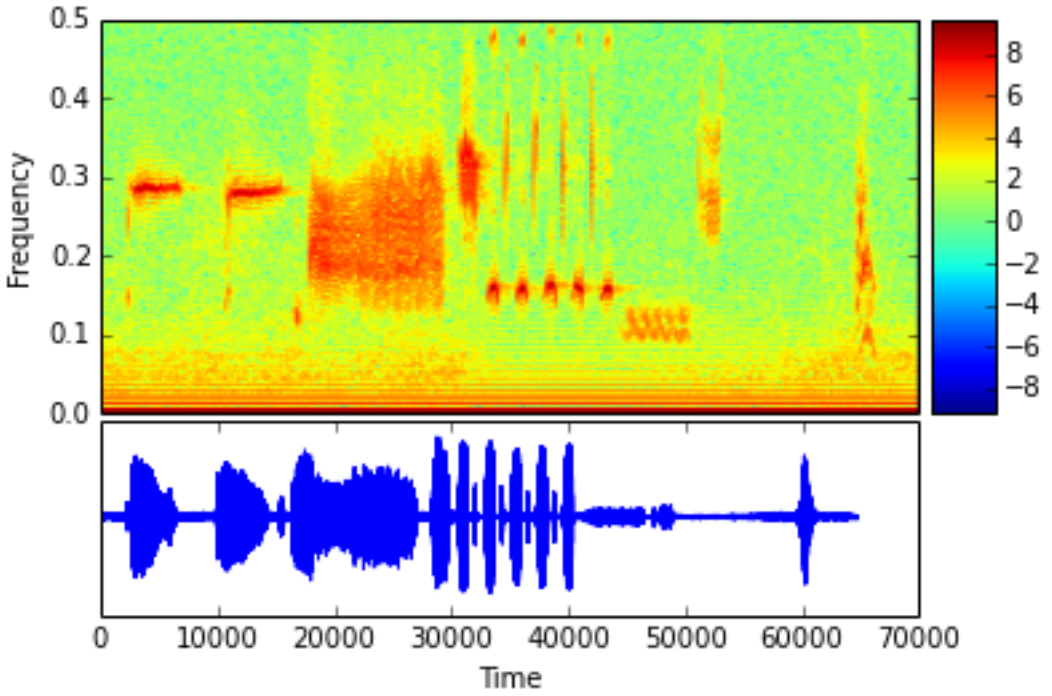
On observe ainsi que le son de chauve souris semble composé, à chaque instant de deux fréquences preque pures, qui décroissent en fonction du temps... Une autre implantation, plus complète, est fournie ci-dessous. On peut notamment faire varier le type de fenêtre de pondération, la longueur de recouvrement lors du calcul de TF à court terme successives,

In [29]:
```
def isodd(num):
        return num & 1 and True or False

def spectrogram(x,L=256, Loverlap=None, N=1024, window=None, logPlot=False):
    """
    """
    if Loverlap==None:
        Loverlap=L-1
    if window==None:
        window=hanning(L)
    elif callable(window):
        window=window(L)
    Nx=size(x)
    k=0
    T=size(range(0,Nx,L-Loverlap))

    central_freq=((N+1)/2 if isodd(N) else ((N)/2+1))
    Xwf=zeros((central_freq,T))+0j #!! force complex type !!
    for n in range(0,Nx-L-1,L-Loverlap):
        xw=x[n:n+L]*window
        Xftmp=fft(squeeze(xw),N)
        Xwf[0:central_freq,k]=Xftmp[0:central_freq]
        k+=1
    figure()

    import matplotlib.gridspec as gridspec
    # Utilisé pour faire des subplots arbitraires, avec recouvrement éventuel et tou
    G = gridspec.GridSpec(12, 12)
    #fig=figure(figsize=(10,4))
    ax1 = subplot(G[0:8, 0:11])
    if logPlot:
        # Plot in log scale -- can be pretty
        imshow(log(abs(Xwf)+1e-4), aspect='auto', cmap='jet',origin='lower',extent=
    else:
        imshow(abs(Xwf), aspect='auto', cmap='jet',origin='lower',extent=(0, Nx, 0,
    #xlabel("Time")
    xticks([])
    ylabel("Frequency")

    ax2 = subplot(G[8:12, 0:11])
    plot(x)
    xlabel("Time")
    yticks([])
# Et  maintenant la colorbar
    cbaxes = subplot(G[0:8, 11])
    colorbar(ax=ax1, cax = cbaxes)
    return Xwf
```

In [21]:
```
(r, x) =wavread('gong.wav')
X=spectrogram(x,Loverlap=255,window=bartlett)
```



In [30]:
```
(r, x) =wavread('bruant.wav')
X=spectrogram(x,Loverlap=0,window=bartlett,logPlot=True)
```



In []: