1-D Fourier Transforms

The Fourier Transform is method for representing a 1-dimensional vector (like a time-series) in terms of 'frequencies'. It is typically used to find periodic signals buried in noise, and to design filters that only pass through a specific range of frequencies.

There is an amazing theorem that states that any finitely sampled time-series can be perfectly reconstructed as a sum of a finite number of sinusoids. Moreover, if there are n time-points, then only n/2 sinusoids will be needed.

A sinusoid is defined by three variables, the frequency, the amplitude and the phase. The frequency of the sinusoids needed to reconstruct a time-series is determined by the length of the time series. The first sinusoid will have exactly one cycle, the second will have two cycles, and so on. There's one more sinusoid which has 'zero' frequency, sometimes called the 'dc' (from electrical engineering). This is the mean of the time-series and therefore determines the vertical offset.

Fourier transforms on discretely sampled data are almost always done through an algorithm called the 'Fast Fourier Transform', or FFT. This is a very efficient algorithm (originally developed by Gauss) that is available in all mathematical sofware programs, including Matlab.

```
fft Discrete Fourier transform.

   fft(X) is the discrete Fourier transform (DFT) of vector X.  For matrices, the fft operation is applied to each column. For N-D arrays, the fft operation operates on the first non-singleton dimension.

   fft(X,N) is the N-point fft, padded with zeros if X has less

   than N points and truncated if it has more.

   fft(X,[],DIM) or fft(X,N,DIM) applies the fft operation across the dimension DIM.

   For length N input vector x, the DFT is a length N vector X,

   with elements

                 N

     X(k) =      sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.

                n=1

   The inverse DFT (computed by IFFT) is given by

                 N

     x(n) = (1/N) sum  X(k)*exp( j*2*pi*(k-1)*(n-1)/N), 1 <= n <= N.

                k=1
```

Q1 :Définir une sinusoide d'amplitude 1 durant 5 cycles et pas de déphasage avec une période d'échantillonnage de 0.01s. Visualiser le signal temporel.

```
clear all ;
dt = .01;
maxt = 1;
t = 0:dt:(maxt-dt);
nt = length(t);  %length of t

nCycles = 5;
amp = 1;
phase = 0;
y5 = amp*cos(2*pi*t*nCycles-pi*phase/180);
figure(1)
clf
plot(t,y5); xlabel('Time (s)'); set(gca,'XTick',0:.2:maxt);
set(gca,'YTick',-1:1); ylabel('Amplitude'); set(gca,'YLim',amp*1.1*[-1,1]);
%'grid' draws dotted lines at the x and y-tick values
grid
```

Q2:La sinusoide a une fréquence de 5Hz (5cycles en 1s).
Vous allez maintenant calculer le spectre de y5 en utilisant ‘fft’

```
Y5 = fft(y5);
```

Taille de y5 et Y5 ?

```
whos y Y
```

Classe de Y ? (A note : les variables temporelles sont coder en **minuscule**, et les variables transformées -espace de Fourier- en **majuscule**)

```
Y5(1:10)'
ans =

   0.0000
  -0.0000 - 0.0000i
  -0.0000 + 0.0000i
  -0.0000 + 0.0000i
  -0.0000 + 0.0000i
  50.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 - 0.0000i
   0.0000 + 0.0000i
   0.0000 - 0.0000i
```

Since our sinusoid has 5 cycles, the 'component' in the FFT that we use to reconstruct this should thererfore be in the 6th entry of Y

Y5(6)

ans =

50.0000 - 0.0000i

This sinusoid of 5 cycles has both an amplitude and phase. Since the corresponding complex number has two parts, you can guess that there is a mapping between the real and imaginary parts of this complex number, and the corresponding amplitude and phase of the sinusoid.

This number has a real component of 50, and an imaginary component of zero. 50? This number is a convention specific to FFT. To convert it to the actual amplitude of the sinusoid component we need to scale it by **nt/2**. We'll do this now and re-plot (and rescale the axis to include negative numbers).

Maintenant ajouter au graphique une nouvelle sinusoide d'amplitude 2 et de déphasage 30°.Visualiser 7 cycles.

```
nCycles = 7;
amp = 2;  %around 1.15
phase = 30;  %degrees
y7 = amp*cos(2*pi*t*nCycles-pi*phase/180);
figure(1)
 hold on
 plot(t,y7,'r-'); xlabel('Time (s)'); set(gca,'XTick',0:.2:maxt);
set(gca,'YTick',-1:1); ylabel('Amplitude'); set(gca,'YLim',amp*1.1*[-1,1]);
axis square grid off  Y7 = fft(y7);
```

Comparer la phase de Y5 et Y7 en tracant le graph partie real vs partie imaginaire (cycle)

```
figure(2)
 clf
plot([0,2*Y5(6)/nt],'b-'); hold on
h5=plot(2*Y5(6)/nt,'bo','MarkerFaceColor','b'); axis equal  xlabel('Real
component') ylabel('Imaginary component');
hold on plot([0,2*Y7(8)/nt],'r-'); h7=
plot(2*Y7(8)/nt,'ro','MarkerFaceColor','r');  set(gca,'XLim',1.1*amp*[-
1,1]); set(gca,'YLim',1.1*amp*[-1,1]);


amp5 = 2*abs(Y5(6))/nt;
phase5 = -180*angle(Y5(6))/pi;
 amp7 = 2*abs(Y7(8))/nt;
phase7 = -180*angle(Y7(8))/pi;
```

Q5:On génére une sinusoide plus compliquée
y57=y5+y7

```
y57 = y5+y7;  figure(1) clf  hold on plot(t,y57); xlabel('Time (s)');
set(gca,'XTick',0:.2:2); set(gca,'YTick',-1:1); ylabel('Amplitude');
```

Tracer son spectre en utilisant ‘fft’

```
Y57 = fft(y57);
% We'll plot the amplitudes of the first 10 sinusoids as a 'stem' plot
id = 2:11;  %sinusoids of periods 1 through 10
figure(3)
clf
stem(2*abs(Y57(id))/nt,'fill')  xlabel('Number of cycles')
ylabel('Amplitude');
```

See how there are only two non-zero amplitudes - the two sinusoids corresponding to 5 and 7 cycles, which is how we built the sinusoid. This illustrates the 'linear' property of the fft: The fft of the sum of two time-series is the same as the sum of the fft of each time series. It also shows a conventional way to represent the results of an fft. By showing only the amplitude we've lost our phase information. Because of this, people often forget to consider the phase of the fft

Calculer le vecteur F des fréquences réduites.

Fs=1/dt;
F=(0:nt-1)*Fs/nt ;

Comparer le spectre Y57 avec l’usage de la fonction plotFFT.

Q6: Calcul de spectre “entier”
Sur tout le spectre calculé, on consate une symétrie miroir par rapport a Fs/2, les amplitudes sont doublées

```
Fs=1/dt;
F=(0:nt-1)*Fs/nt;
Figure ;
clf
stem(F,2*abs(Y57)/nt,'fill')
xlabel('Hz');
ylabel('Amplitude');
```

The fft is a 'lossless' transformation, meaning that no information is lost. Because of this (and because the sinusoids are mutually 'orthogonal') if there are nt timepoints in our time-series, then exactly nt numbers are needed to represent the transformation. The output of the fft has nt values in it - but each of these values are complex so they 'contain' two numbers. So the output of the fft has twice as many numbers as the input.

Q7:2 exemples de spectre intéressant pour l’équivalence temps-fréquence

An interesting example is the 'delta' function - zeros everywhere except for one value;

```
y = zeros(size(t)); y(20) = 1;  plotfft(t,y);
```

Another example is 'white' noise, where every time point is an independent draw from a normal distribution:

```
y = randn(size(t)); plotfft(t,y);
```

The amplitude spectrum looks like noise too. These amplitudes are also normally distributed with equal probability across all frequencies. White noise contains sinusoids of all frequencies - it's called 'white' presumably because 'white' light contains light of all wavelengths in the spectrum.

Fourier transforms are good at detecting periodic signals buried in noise that is otherwise very difficult to see in the time-series. As an example, we'll add a 12-cycle sinusoid to the random time-series that we just looked at:

```
nCycles = 12; y = y+sin(2*pi*nCycles*t); plotfft(t,y);
```

Q8:Un peu de vibration ;)

Visualiser le signal ‘handel’

```
load handel
```

The file contains two variables, the time-course of the sound, y and the sampling rate, Fs (in Hz). We can create our own time vector from this information:

```
maxt = length(y)/Fs;
t = linspace(0,maxt,length(y));
% Here's a plot of the time-course of the sound hander

Figure (4) ;
clf
hold on
 plot(t,y);
xlabel('Time (sec)');
```

L’écouter maintenant…

```
ylim = get(gca,'YLim');
timeH = plot([0,0],ylim','r-','LineWidth',2);
sound(y,Fs);  tic
while toc<maxt
     set(timeH,'XData',toc*[1,1])
drawnow
end
delete(timeH);
```

Visualier le spectre (amplitude)

```
plotfft(t,y);
```

Q9:Maintenant on zoom, on écoute et on regarde le spectre

```
tLo = .7; tHi = .9;  figure(1)
y1 = y(t>tLo & t<tHi);
 maxt = length(y1)/Fs;
t1 = t(t>tLo & t<tHi);  %linspace(0,maxt,length(y1));
sound(y1,Fs);
plotfft(t1,y1) ;
```

Q10:Maintenant afficher l'évolution du spectre au cours du temps

```
figure
spectrogram(y,256,120,256,Fs);
```

si on moyenne les spectres "locaux" avec recouvrement, on parle de periodogramme de Welch 'periodogram' et on améliore la precision du spectre

http://www.mathworks.fr/fr/help/signal/ug/spectral-analysis.html

Q11: Comment estimer un spectre à partir d'une entrée temporelle (force en function du temps) et d'une sortie temporelle (deplacement en function du temps)?

```
Ts=0.05;
Fs=1/Ts;
load data;
u=data(:,1);
ys=data(:,2);
t=0:Ts:(length(u)-1)*Ts;
figure; subplot(211);plot(t,u);subplot(212);plot(t,ys);
 [txy,f]=tfestimate(u,ys,[],[],[],Fs);
figure
 semilogx(f,20*log10(abs(txy)))
```