Post-processing data with Matlab®

Best Practice

TMR7 - 31/08/2015 - Valentin Chabaud
valentin.chabaud@ntnu.no

- Cleaning data
- Filtering data
- Extracting data’s frequency content

Introduction

- A trade-off between do-it-yourself philosophy, time spent on side tasks and quality of the results
- Keeping data as is and default settings while filtering / computing the power spectral density leads to inaccurate, or even misleading results which are hard to comment on. Fault is often mistakenly taken back to measurement uncertainties.
- Many possibilities in Matlab (various toolboxes and built-in functions of various complexity and flexibility)

➔ The following is only a suggestion of efficient methods to save time. Help will be preferably provided for those methods. You are however free to choose your own as long as you keep a critical eye on the underlying uncertainties.

Cleaning data

- Equipment limitations (especially in MC lab) lead to:
  - Erroneous data: Infinite (very large) or NaN (not a number).
  - Missing data: 0. Can occur for a somewhat long period of time and thus affects the results even if the mean value is small, even 0.

- Acquired data should be already uniformly sampled (constant step size). However for safety, run the function:

Selected time array

```
t=tstart:dt:tend
```

Uniformly sampled selected data

```
x=interp1(t0,x0,t)
```

Raw data and time arrays

Which also cuts the data to the desired time span.

Cleaning data cont.

- The data can be cleaned by the function:

Original data (uniformly sampled), row vector.

```
xclean=clean_data(x',CrtSTD,CrtCONV)
```

Cleaned data, row vector

Iterative outlier criterion

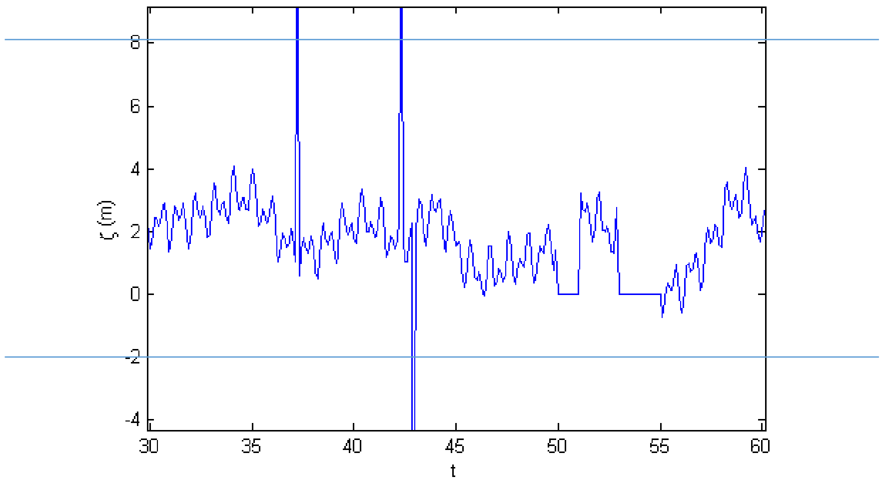
Convergence criterion

Play around with these criteria to get the desired result

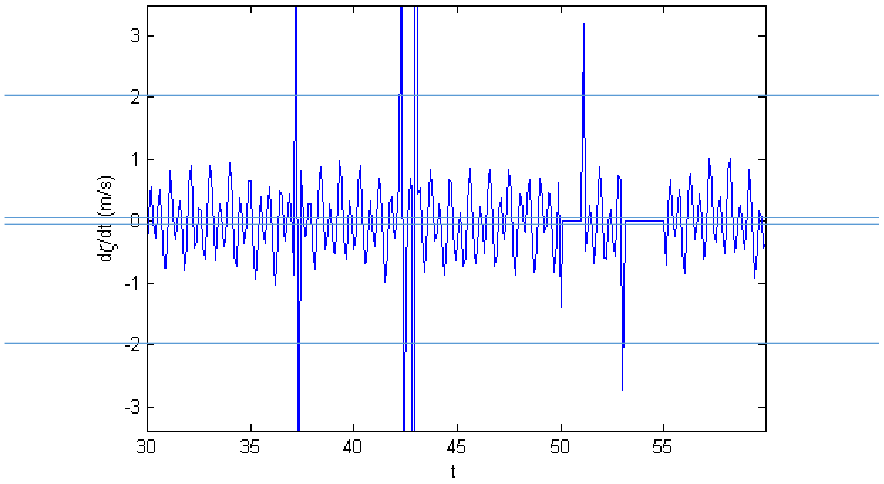
- Home made function. Tested on a limited number of time series only. Yet, always check the results! Modifications and suggestions are welcome.
- Smoothen x using `smooth(x,round(fs/fx)+1)` if sampled at fx<fs (stair-like signal)
- `clean_data` function is found in the Resource-section of the TMR7 webpage and at the end of this presentation

μ : Mean value
σ : Standard deviation

How `clean_data` works



Signal



Derivative

1) If

$$|x_i - \mu_x| \geq CrtSTD * \sigma_x$$

Or

$$|\dot{x}_i| \geq CrtSTD * \sigma_{\dot{x}}$$

Or

$$|\dot{x}_i| \leq \frac{\sigma_{\dot{x}}}{10 * CrtSTD}$$

3) Recompute μ_x and σ_x and iterate until it has converged:

$$\frac{|\sigma_{x_n} - \sigma_{x_{n-1}}|}{\sigma_{x_{n-1}}} \leq CrtCONV$$

- CrtSTD > 1
- CrtSTD large when signal has uneven amplitudes (if too small, cleaning can affect valid parts of the signal)

Then

2) **Replace** x_i by a linear interpolation of the nearest valid points

Less error is induced by **keeping** corrupt points than simply **removing** them!

Filtering data

Digital Butterworth filters:

- Most commonly used filters for this kind of application. One is already in place in the data acquisition set up, removing very high frequencies.
- Described by a transfer function $H(z) = \frac{b(1)+b(2)z^{-1}+\cdots+b(n+1)z^{-n}}{1+a(2)z^{-1}+\cdots+a(n+1)z^{-n}}$
- Designed by

'low'	low-pass filter	filters frequencies > cutoff freq.
'high'	high-pass filter	filters frequencies < cutoff freq.
'bandpass'	band-pass filter	filters frequencies outside the cutoff freq interval.

Order of the filter

```
[b,a]=butter(order,wstar,'ftype')
```

Normalized cutoff frequency
Or interval of frequencies
(bandpasss filter)

$$w^* = \frac{Cutoff\ frequency}{Nyquist\ frequency}$$

$$\frac{1}{2 * time\ step}$$

Filtering data cont.

How does it work?

A function, called gain, attenuates some parts of the frequency content of the signal.

$$x_{filt}(t) = \mathfrak{F}^{-1}\{G(\omega) * \mathfrak{F}\{x(t)\}(\omega)\}(t)$$

Filtered signal IFFT Gain FFT of the signal

- In the frequency domain, no difference is made from 2 different processes having the same frequency

➔ In order for filtering to be successful, undesired processes should have a distinct frequency content from that of the studied process.

- $G(\omega)$ must be continuous for the IFFT to exist.

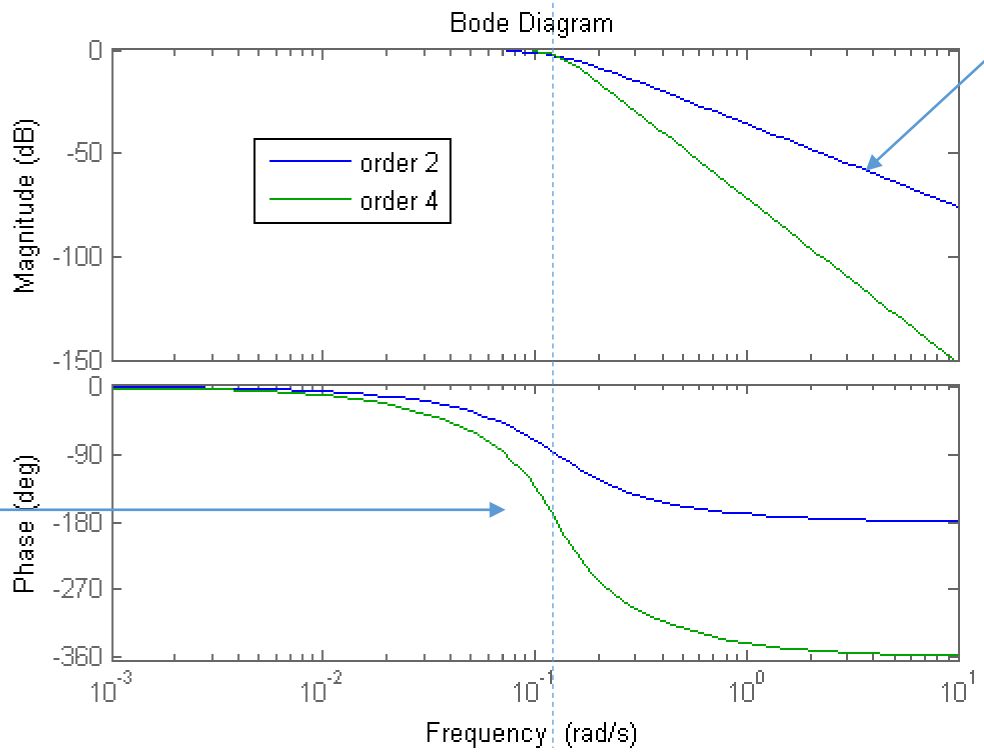
➔ The attenuation evolves gradually with the frequency. A sharp cut in the frequency content is not possible with low order filters.

Filtering data cont.

The filtering effect is best described by Bode diagrams of the filter's continuous transfer function

```
[b,a]=butter(order,wstar,'low')
Figure()
Bode(d2c(tf(b,a,dt)))
```

Cut-off frequency



Filtering induces a phase shift in the signal, increasing with the order

Slope in gain reduction:

- = «filtering strength»
- Increasing with the order
- Increasing with frequency (for a low-pass filter) from cut-off frequency

➔ The cut-off frequency should be higher than the undesired frequencies, but lower than the frequencies of interest.
Else the signal will be badly filtered or **the amplitude attenuated**!

Filtering data cont.

- A so-called “spectral gap” is needed for efficient filtering

  = No energy in the spectrum around the cut-off frequency

If this is not the case, uncertainties will be introduced, take note of them!

- To avoid phase shift (improves readability in time domain plots), use:

Original data (uniformly sampled)

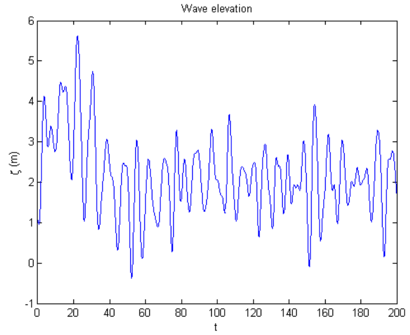
```
xfilt=filtfilt(b,a,x)
```

Filtered data

Digital filter coefficients

Extracting PSD (Power Spectral Density)

2 approaches:



Autocorrelation in time-domain

$$R_{xx}(\tau) = \int_{-\infty}^{+\infty} x(t)\, \bar{x}(t-\tau)dt \xrightarrow{\text{FFT}} S_{xx}(f) = \int_{-\infty}^{+\infty} R_{xx}(\tau)\, e^{-j2\pi x\tau}\, d\tau$$

PSD

FFT

$$\mathfrak{F}_x(f) = \int_{-\infty}^{+\infty} x(t)\ e^{-j2\pi f t}\, dt \xrightarrow{\text{Square}} S_{xx}(f) = |\mathfrak{F}_x(f)|^2$$

- Upper: `pcov` function and variants. Sensitive to signal manipulations.
- Lower: `pwelch` function and variants. Sensitive to signal length. In practice, using directly the `fft` function squared and smoothening manually is more computationally efficient.

Extracting PSD cont.

`pwelch` is the standard. However default values often lead to inaccurate results, it may excessively computationally demanding and hard to tune.

PSD
Array 1xlength(f)

Time series. Uniformly sampled.
Preferably minus mean value.

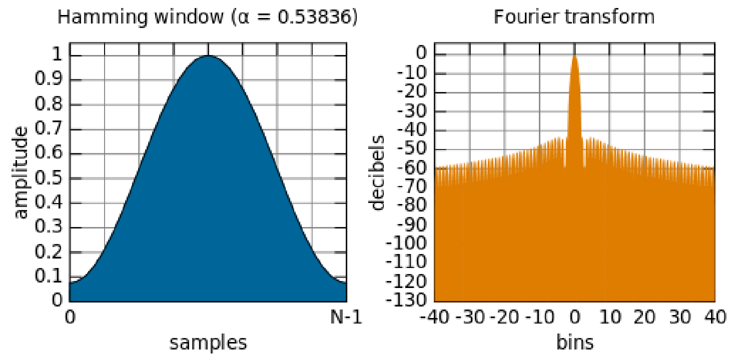
Number of overlapping samples between windows. Does not have a big influence. Window/10 is a good start.

Sampling frequency (Hz)

```
Sxx=2*pwelch(x,Window,Noverlap,f,fs)
```

Change from two-sided to one-sided PSD

Frequencies (Hz) at which you want the PSD to be computed.
Caution: Use uniform step size in frequency. If you want better accuracy at low frequencies without being too computationally demanding, use a smaller step but scale the PSD at all freqs according to a reference step size. Else it will not reflect properly the amplitude when plotted!



The signal is segmented into «windows». The FFT is computed segment by segment which are then assembled to give the PSD.
The broader the window, the finer the spectrum. The narrower, the smoother. Adjust it to get a readable yet accurate spectrum (Use values from NFFT/2 to NFFT/10).

Extracting PSD cont.

- `pwelch` may give inaccurate results for short signals with transients (oscillations in low frequencies). Try to play around with the `Noverlap` parameter.

- `psd_fft` is a home made function computing the PSD directly from the Fourier transform. It is more computationally efficient and user friendly than `pwelch`. A similar syntax is kept.

Smooting parameter (number of points in smoothing parameter).
Typical value: sampling frequency in Hz/10

```
Sxx=psd_fft(x,N,f,fs)
```

- `psd_fft.m` is found in the Resource-section of the TMR7 webpage and at the end of this presentation

Example: Irregular wave elevation

Generated from JONSWAP spectrum. The following is artificially added:

- Erroneous and missing data
- Measurement noise
- Transients
- Mean offset

Example cont. : Matlab script

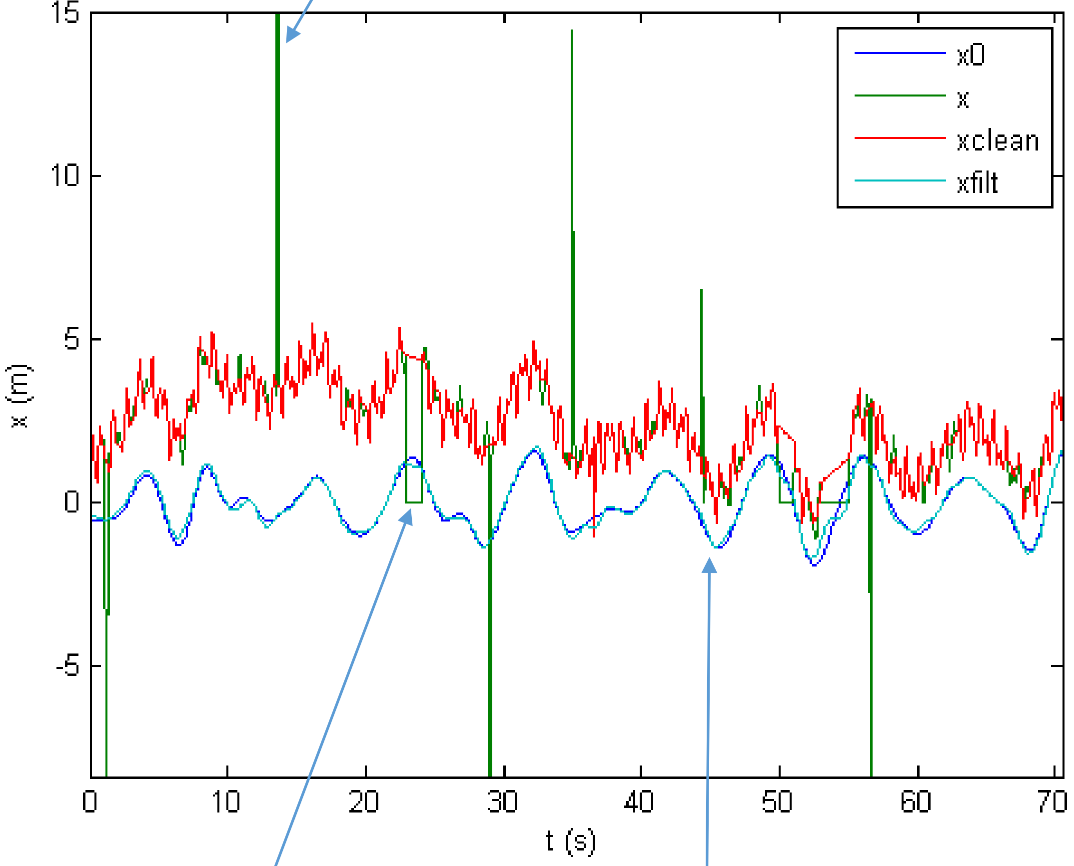
```
load('data.mat','x','time')             %Load wave elevation and time from file
duration=200;
dt=0.1;
t=0:dt:duration;
Nt=length(t);
xint=interp1(time,x,t);                 %Interpolate data
xclean=clean_data(xint,3,0.001);        %Clean data
cutoff=[0.3 4]/(2*pi);                  %Cut-off frequencies
fnyq=1/(2*dt);                          %Nyquist frequency
[b,a]=butter(4,cutoff/fnyq,'bandpass'); %Get filter coefficients
xfilt=filtfilt(b,a,xclean);             %Zero-phase filtering
df1=0.01;
df2=0.1;
f1=0.01:df1:0.99;                       %Small frequency step for low frequencies
f2=1:df2:10;                            %Large frequency step for high frequencies
f=[f1 f2];
Sxx=psd_fft(xint-mean(xint),10,f,1/dt); %PSD of unfiltered data
Sxx_filt=psd_fft(xfilt-mean(xfilt),10,f,1/dt);     %PSD of filtered data

figure(1)
plot(t,[x0 xint xclean xfilt])                    %x0: original data generated from JONSWAP

figure(2)
plot(w,jonswap,f*2*pi,Sxx/(2*pi),f*2*pi,Sxx_filt /(2*pi))
```

Example cont. : time and frequency domain plots

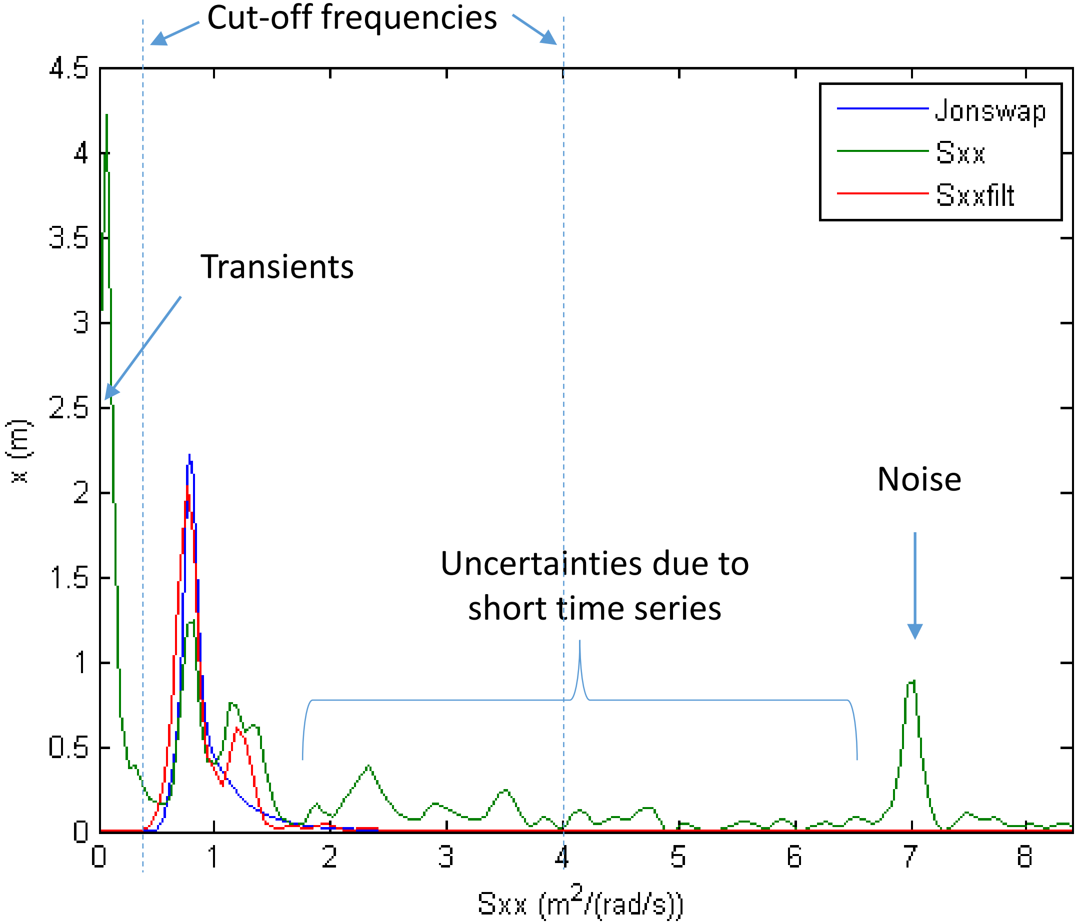
Outlier



Period of missing data

Band-pass filtering removes hig and low (including offset = 0 rad/s) frequencies

Cut-off frequencies



Transients

Noise

Uncertainties due to short time series

Uncomplete spectral gap: slightly uncertain filtering of the transients

Large spectral gaps allowing efficient filtering of the noise

Questions?

Now or later on, about this or anything related to the course, don’t hesitate.

valentin.chabaud@ntnu.no

Office G2.130

clean_data.m

```
function x=clean_data(data,CrtSTD,CrtCONV)

%Written by Valentin Chabaud. v3 - August 2015
%Removes erroneous values and outsiders from time series

x=data';
sx=std(x);
mx=mean(x);
d=diff(x);


sd=std(d);
d=[d;d(end)];
% figure(3)
% plot([data';d])
std_prev=std(x)/CrtSTD;
N=10;
```

```
while abs((std(x)-std_prev)/std_prev)>CrtCONV
    flag=0;
    ind=[];
    for i=1:length(x)
        if abs(x(i)-mx)>sx*CrtSTD || abs(d(i))>sd*CrtSTD ||
abs(d(i))<sd/CrtSTD*0.1
            if flag==0
                flag=1;
                ind=[ind;[i 0]];
            end
        else
            if flag==1
                ind(end,2)=i;
                flag=0;
            end
        end
    end
    if(ind(end,end))==0
        ind(end,end)=length(x);
    end
    y=[ones(N,1)*x(1);x;ones(N,1)*x(end)];
    for i=1:size(ind,1)
        inttot=(1:length(y))';
        intrem=ind(i,1)+N:ind(i,2)+N;
        intfit=setdiff(inttot,intrem);
        z=y(intfit);
%         f = fit(intfit, z, 'smoothingspline','SmoothingParam',
0.1);
%         y(intrem)=feval(f,intrem);
        y(intrem)=interp1(intfit,y(intfit),intrem);
        x=y(N+(1:length(x)));
    end
    std_prev=std(x);
end

x=x';
```

psd_fft.m

```
function [S,Sraw]=psd_fft(x,N,f,fs)
    %Calculate PSD from raw fft and smoothing
    %x: signal
    %N: smoothing parameters (number of points in moving average)
    %f: desired output frequencies
    %fs: sampling frequency
    %S: PSD @ frequencies f
    %Sraw: Structure with field S=PSD and field f=frequencies as defined by fft
    Nt=floor(size(x,1)/2)*2;
    S=fft(x);
    dt=1/fs;
    S=2*dt/Nt*abs(S(1:Nt/2+1,:)).^2;
    fS = 1/dt*(0:(Nt/2))/Nt;
    for i=1:size(x,2)
        S(:,i)=[S(1,i)/2;smooth(S(2:end,i),N)];
    end
    Sraw.S=S;
    Sraw.f=fS;
    S=interp1(fS,S,f);
```