Technical University of Denmark

Wind energy:

# Generation of time series from a spectrum:

Generation of Wind times series from the Kaimal spectrum

Generation of wave times series from the JONSWAP spectrum

Emmanuel Branlard

February 2010

Risø DTU
National Laboratory for Sustainable Energy

# Contents

## Introduction

In this report, stochastic wind and wave times series are generated from a given spectrum. The Kaimal spectrum is used for wind times series, and the JONSWAP spectrum is used for wave times series. After presenting the models, the generation of times series from these spectra will be discussed. Eventually, the spectra of the time series will be calculated in order to check that it indeed matches the input spectra, thus validating the algorithm employed.

# Part I

# Wind time series generation

## 1 Kaimal spectrum

The Kaimal spectrum will be used as it is a simple approximation for the wind spectrum[2]. This model contain the typical $-5/3$ power law for high frequencies, and it introduces a relation ship between the spectrum amplitude and three parameters relevant for the wind : the height above ground $z$, the mean wind speed $U$ and the friction velocity $u_*$. The expression of the Kaimal spectrum is as follow:

$$S_{uu}(f) = u_*^2 \frac{52.5z/U}{(1+33n)^{5/3}} \quad \text{or} \quad S_{uu}(\omega) = u_*^2 \frac{52.5z/U}{(1+33\times 2\pi n)^{5/3}} \tag{1}$$

where $n = fz/U$. The set of parameters used in this report is: $z = 70$m, $U = 20$m/s and $u_* = 1$m/s. The spectrum for a sampling frequency of $f_s = 10$Hz and $N = 10^5$ values is displayed on figure 1.

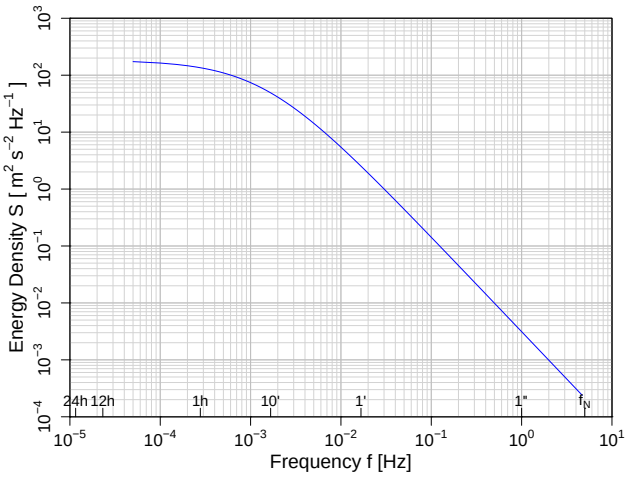



Figure 1: The Kaimal spectrum with the typical $-5/3$ power law at high frequencies

## 2 Wind Time series generations

**First approach** No direct relationship exists to determine the Fourier coefficients from the spectrum so that these coefficients have to be generated randomly. Nevertheless, the standard deviation of each coefficient is determined by the spectrum, that is:

$$\sigma_{X_l}^2 = \frac{T}{2\pi} S_{uu}(\omega_l) \tag{2}$$

where $T = N/f_s$. As a results of this, each Fourier coefficient $X_l$ is generated using a normal distribution of mean 0 and standard deviation $\sigma_{X_l}$. The mean value of the time series is entirely determined by the first Fourier coefficient. Indeed, this coefficient corresponds to a frequency of 0, and is thus not multiplied by any sine or cosine function. For this reason, the first Fourier coefficient is set to the value $c_0 = \frac{TU}{2\pi}$. The inverse Fourier transform is then applied on the series of randomly generated Fourier coefficients to determine the series in the time domain:

$$x_m = 2\pi f_s \frac{1}{N} \sum_{l=0}^{N-1} X_l \exp\left(i \frac{2\pi l m}{N}\right) = 2\pi f_s \frac{1}{N} \text{ ifft}_R(X) = 2\pi f_s \text{ ifft}_M(X) \tag{3}$$

where $\text{ifft}_R$ and $\text{ifft}_M$ are the inverse Fourier transformation respectively in *R* and *Matlab*. Nevertheless, this method will return time series of complex coefficients. Taking the real part will solve this issue, or, one can use the second approach suggested below.

**Second Approach** The method above can be made more realistic by using complex numbers for the Fourier coefficients, so that the time series is actually expressed in real numbers. For this matter $N/2$ coefficients $a_n$ and $N/2$ coefficients $b_n$ are generated to compose the Fourier coefficients vector $c_n$ of length $N$ as:

$$(c_n)_{0..N-1} = [\frac{TU}{2\pi} \ ; \ a_1 + ib_1 \ ; \ ... \ ; \ a_{N/2-1} + ib_{N/2-1} \ ; \ a_{N/2} + 0 \ ; \ a_{N/2-1} - ib_{N/2-1} \ ; \ ... \ ; \ a_1 - ib_1] \tag{4}$$

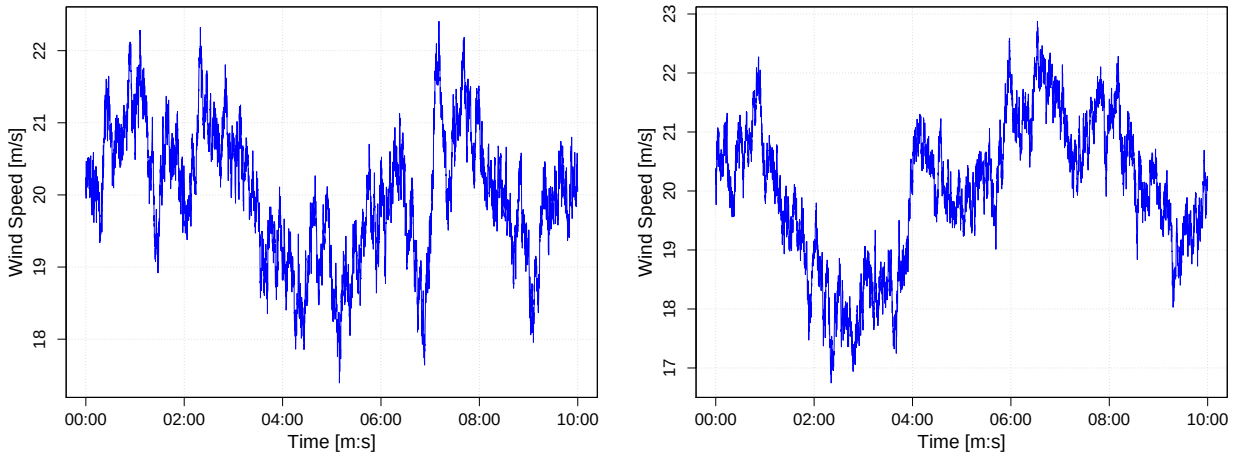
The generation of the random numbers $a_n$ and $b_n$ is such that the standard deviation is distributed equally among the real and imaginary part(no systematic time lag):

$$\sigma_{a_n}^2 = \sigma_{b_n}^2 = \frac{1}{2}\sigma_{X_n}^2 \tag{5}$$

**Examples** Four different 10 minutes samples at 10Hz generated from this method is displayed on figure 2. Their mean is exactly $U = 20$ m/s, whereas their standard deviation varies between 0.90 and 1.23 m/s. The samples resemble quite well real wind time series.

## 3 Final verifications

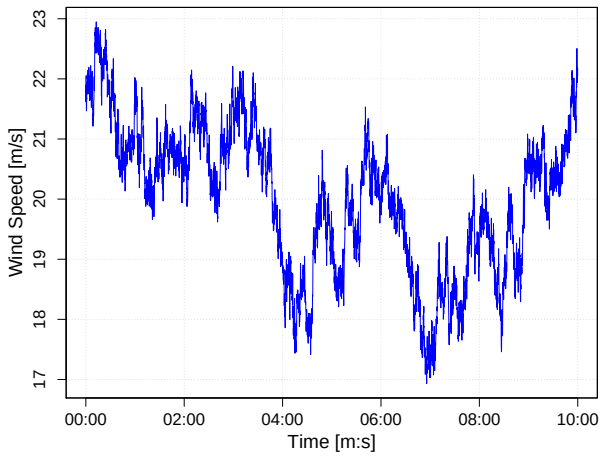
It is expected that the spectrum calculated from the generated time series matches the Kaimal spectrum from which they were derived. For a 10 minutes sample, result are displayed on figure 3a. Good agreement is found with the Kaimal spectrum, and with a higher number of points, the match is even clearer (see figure 3b).



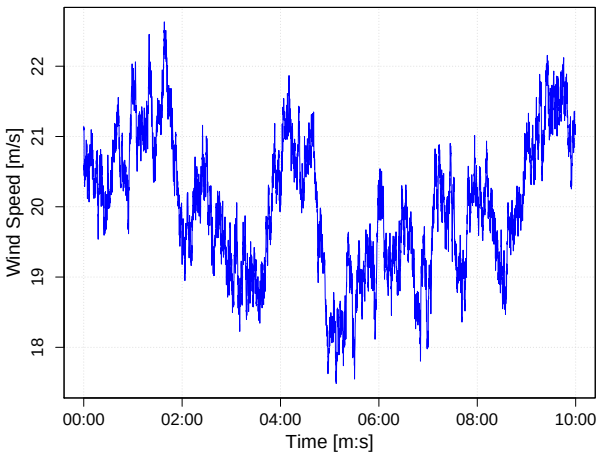

(a)




(b)

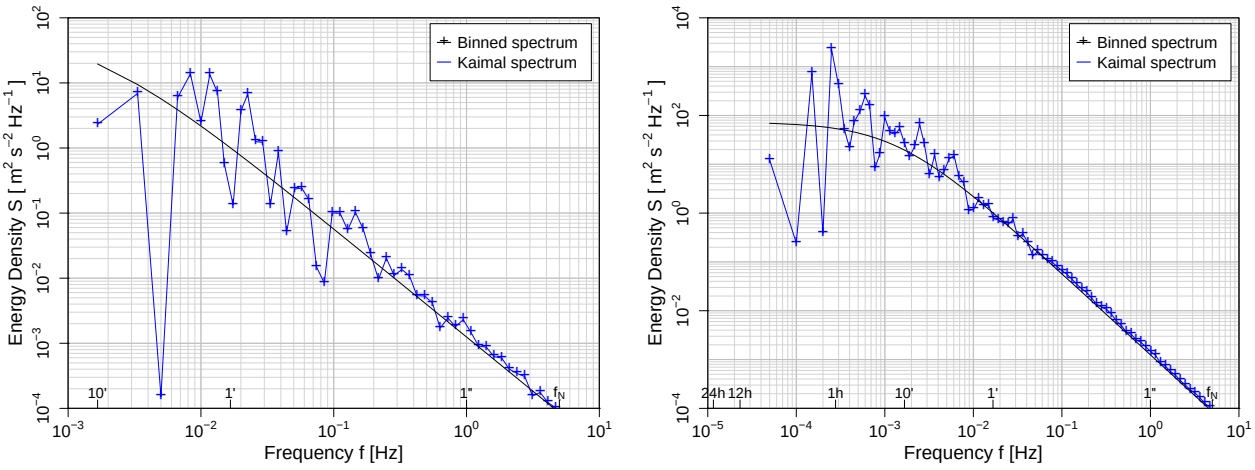



(c)




(d)

Figure 2: Four examples of time series generated from the Kaimal spectrum. The samples seem to reproduce realistic wind signals




(a) (b)

Figure 3: Comparison between the time series spectrum and the Kaimal spectrum from which they were generated. (a) 10 minutes sample of $N = 6.10^3$ points - (b) Sample of $2.10^5$ points

# Part II

# Wave time series generation

## 4 JONSWAP spectrum

The JONSWAP spectrum is implemented following the guidance in the IEC 61400-3 standard[p. 82][4]:

$$S_{JS}(f) = 0.3125 \cdot H_s^2 \cdot T_P \cdot \left(\frac{f}{f_p}\right)^{-5} \cdot \exp\left[-1.25 \cdot \left(\frac{f}{f_p}\right)^{-4}\right] \cdot (1 - 0.287 \log\gamma) \cdot \gamma^{\exp\left[-0.5\left(\frac{\frac{f}{f_p}-1}{\sigma}\right)^2\right]} \quad (6)$$

Where the peak-shape parameter $\gamma$ is assumed to be equal to 3.3, and $\sigma = 0.07$ for frequencies below the peak frequency $f_p = 1/T_p$ and $\sigma = 0.09$ in the opposite case. The values for $T_p$, the wave peak period and $H_s$, the significant wave height, are obtained from measurements. The statistical moments from the spectra are defined as:

$$m_i = \int_0^{\inf} S_{JS}(f) f^i df \quad (7)$$

The spectral moment of order 0 is related with the standard deviation as:

$$\sigma^2 = m_0 \quad (8)$$

If the above parameters are not used, and a simpler JONSWAP formula is used, a scaling factor can be derived to normalize the spectra. For a 50-year sea state spectra, the significant wave height approximation $H_s \approx 4\sigma$ can be used. The normalization coefficient $k$ (if required) is consequently:

$$k = \frac{\frac{H_s}{4}^2}{\int_0^{f_{cut}} S_{JS}(f) df} \quad (9)$$

In this report, 50 years extreme values obtained from measurements at a specific locations are used: $H_{s,50} = 8.1$m and $T_{p,50} = 12.70$

# 5 Wave time series generation

The following decomposition is applied to simulate a wave elevation time series $\eta(t)$ from a spectrum:

$$\eta(t) = \sum_p a_p \cos\left(\omega_p t - k_p x + \epsilon_p\right) \tag{10}$$

$$a_p = \sqrt{2 S_\eta(f_p) \Delta f} \tag{11}$$

$$\epsilon_p = \text{rand}(0, 2\pi) \tag{12}$$

The amplitude of the Fourier Component derived from the JONSWAP spectrum are found in Figure 4.

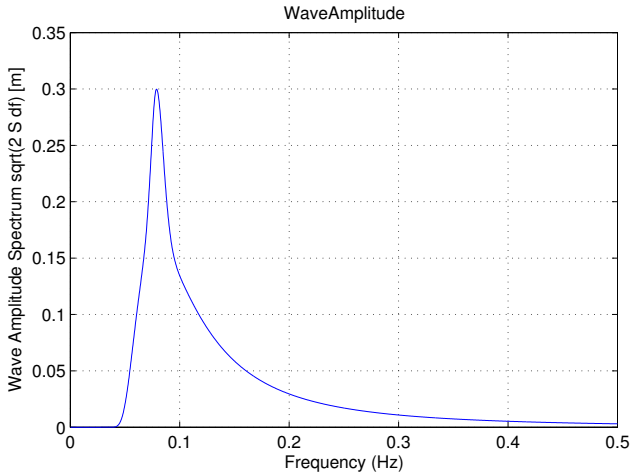



Figure 4: Amplitude of Fourier component $a_p$

If position different than $x = 0$ have to be used, then the dispersion relation

$$\omega^2 = gk \tanh(kh) \tag{13}$$

is solved for each wave component as shown in Figure 5. Resulting time series are displayed in Figure 6.

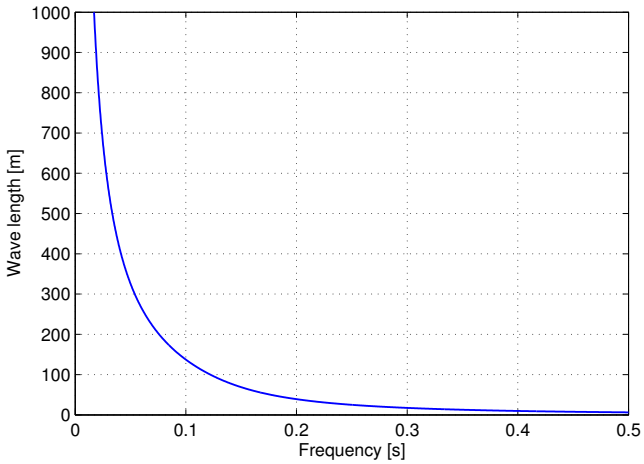



Figure 5: Dispersion Relation.

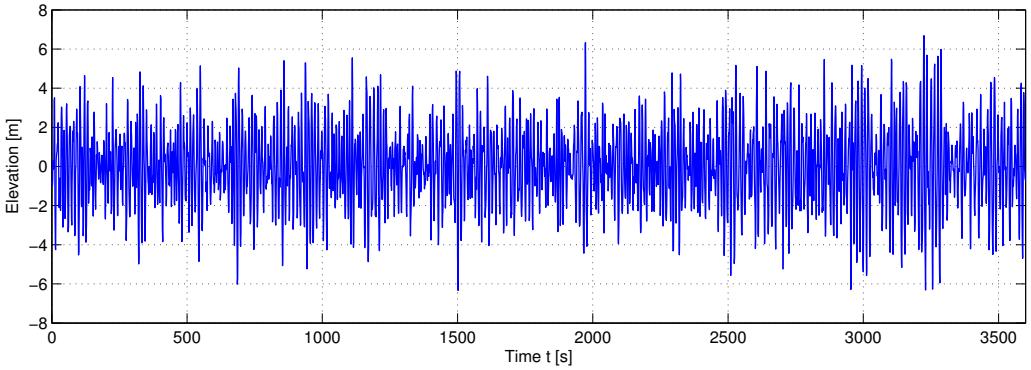



Figure 6: Example of simulated time series for Wave Elevation.

## 6 Final verification

The input spectra (JONSWAP) and the time series spectra are compared in Figure 7. The agreement is complete showing that the procedure of generation of time series from the spectra is valid. As a result of this the time series will satisfy the expected statistics.

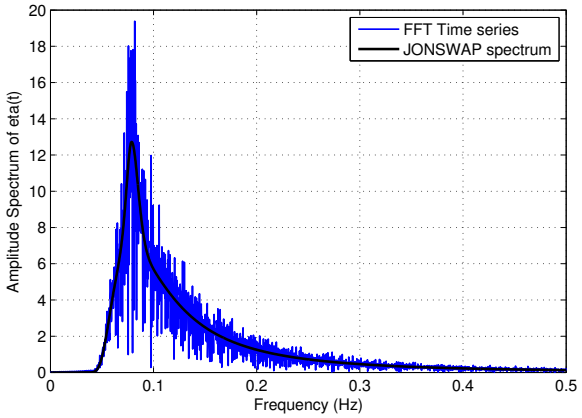



Figure 7: Spectrum Comparison.

## Conclusion

In this short analysis, we have been introduced to the generation of time series from a spectrum, which is a method commonly used in wind energy. One can use the Kaimal spectrum as in this report, nevertheless, is has been previously seen, that a real wind sample does not follow the Kaimal spectrum exactly. Thus this method will generate signals which are really close to real signals but still not perfect. Real spectrum can also be used to generate time series. But of course, if a real wind spectrum is available it means that time series are already available so the interest of the method seem limited in this case. The generation of time series from a spectrum can be used while performing statistics on extreme gusts for instance, and assessment of mean gust shapes[3][7]. Another application could be generation of data when not enough measurements are available for assessments of 50 year extremes, nevertheless, this method would be rather uncertain[1]. Eventually, a main application is for aeroelastic code such as FLEX[8] that require different wind input to test the wind turbine response.

For this matter the Kaimal spectrum is really useful as it is a function of a site-specific parameter $u_*$ as well as the height and the main wind speed.

## References

[1] Po Wen Cheng. *A reliability based methodology for extreme responses of offshore wind turbines.* DU Wind, Delft University wind Energy Research Institute, 2002.

[2] C Dyrbye and S. O. Hansen. *Wind loads on structures.* John Wiley & Sons, 1997.

[3] W. Bierbooms G.Cr. Larsen and K.S. Hansen. *Mean Gust Shapes.* Risø-R-1133(EN) - Risø National Laboratory, Roskilde, Denmark, December 2003.

[4] IEC. *IEC 61400-3 Wind turbines Part 3: Design requirements for offshore wind turbines.* 2009.

[5] J. Mann. *Wind profiles and turbulence spectra.* Risø DTU - Course 45701, 2010.

[6] Nick Jenkins Tony Burton, David Sharpe and Ervin Bossanyi. *Wind Energy Handbook.* J. Wiley & Sons, 2001.

[7] Po-Wen Cheng W. Bierbooms. Stochastic gust model for design calculation of wind turbines. *Journal of wind energineering and industrial aerodynamics, Vol.90 Pages 1237-1251*, 2002.

[8] S. Øye. *Fix Dynamisk, aeroelastisk beregning af vindmøllevinger.* Report AFM83-08, Fluid Mechanics, DTU, 1983.

# APPENDIX

# A R code

## A.1 Main Source code

```
#############################################################
# Initialization
#############################################################
setPlottingMethod("pdf")
setFigsPath("figs/")
setwd("/media/Work/Wind Ressources and loads/Exercise4/")
source("code/MyWTlib.r")
source("code/Spectrum.r")
source("code/MyLegend.r")
library(sfsmisc)

#############################################################
# Numerical values
#############################################################
U=20  # average wind speed
uf=1  # friction velocity
z=70  # height

#############################################################
# Kaimal spectrum
#############################################################
fs=10  #sampling frequency in Hz
N=6*10^3 #length of the time series
f <-(1:floor(N/2))*fs/N;
t=seq(0,(N)/fs,1/fs)
T=N/fs;
n=f*z/U
S=uf^2*(52.5*z/U)/(1+33*2*pi*z/U*f)^(5/3)
S2=T/(2*pi)*uf^2*(52.5*z/U)/(1+33*2*pi*z/U*f)^(5/3)

#############################################################
# Samples generation
#############################################################
x1=generateWSFromSpectrum2(S,N,fs,U)
beginPlot("sample1",showTitle=F);
        plotWSSample(fs=fs,WS=x1);
endPlot()

#############################################################
# calculating spectrum back
#############################################################
R1=getAveragedSpectrum(t(x1),fs)
beginPlot("SpecBackShort",showTitle=F)
        plot.loglog(R1$fb,R1$Sb,type="o",col=4,pch="+", xaxs="i",yaxs="i",
                xlab="Frequency f [Hz]",
                ylab=expression("Energy Density S ["~m^2~s^-2~Hz^-1~"]"))
        plot.add(f,S,pch="",type="o",col=1)
        axis.freq(fs);
        legend("topright",c("Binned spectrum","Kaimal spectrum"),col=c
            (1,4),pch=c("+",""),lty=1,bg="white",inset=0.015,cex=0.85)
endPlot()
```

### A.2 Sample generation

```
#################################################################
# First approach
#################################################################
generateWSFromSpectrum=function(S,N,fs,U){
        T=N/fs;
        X=numeric(N)
        for(l in 1:(N/2)){
                X[l]=rnorm(1 ,mean = 0, sd = sqrt(T/(2*pi)*S[l]) );
        }
        x= Re((fft(c(T*U/(2*pi),X),inverse=T)/N ))*2*pi*fs
        print(c(mean(x),sd(x))
        return(x)
}

#################################################################
# Second approach
#################################################################
generateWSFromSpectrum2=function(S,N,fs,U){
        T=N/fs;
        I=complex(real=0,imaginary=1)
        a=numeric(N/2)
        b=numeric(N/2)
        for(l in 1:(N/2)){
                a[l]=rnorm(1 ,mean = 0, sd = sqrt(T/(2*pi)*S[l])/sqrt(2));
                b[l]=rnorm(1 ,mean = 0, sd = sqrt(T/(2*pi)*S[l])/sqrt(2));
        }
        X=numeric(N)
        X[2:(N/2)]=(a+I*b)[1:(N/2-1)]/2
        X[((N/2+2):N)]=(a-I*b)[(N/2-1):1]/2
        X[1]=T*U/(2*pi);
        X[N/2+1]=a[N/2]
        x= Re((fft(X,inverse=T)/N ))*2*pi*fs
        print(mean(x))
        print(sd(x))
        return(x)
}
```

# B Matlab code

### B.1 Mains

```
InitClear

%% Wind spectrum
vt=linspace(0,3600,3601)
% ---------------------- Kaimal spectrum
[ vUw_Kaimal, fraw, Sraw,fKailman,SKailman,fBin,SBin,fWelch,SWelch ] =
    fStocWind(2*vt(end));
vUw_Kaimal=vUw_Kaimal(1:length(vt)) ;


% ---------- Stochastic Times series
figure,plot(vt,vUw_Kaimal ,'Color',[0 0 0.7]),box on, grid on,title('Wind
    Time Series'),xlabel('t [s]'),ylabel('Wind speed [m/s]');
```

```
12
13 % ---------- Spectra of Stochastic Times series
14 figure,
15 loglog(fraw,Sraw,'-','Color',[0.66 0.66 0.66])
16 hold on, % stupid matlab
17 loglog(fWelch,SWelch,'-','Color',[0.35 0.35 0.35])
18 loglog(fKailman,SKailman,'k-','linewidth',2)
19 loglog(fBin,SBin,'r-')
20 title('GeneratedTimeSeriesKaimalSpectrum')
21 xlabel('Frequency [Hz]')
22 ylabel('PSD of Uw [m^2/s^2 s]')
23 grid on
24 lg=legend('Raw spectrum','pWelch spectrum','Kaimal spectrum','Bin-averaged
       spectrum');
25 set(lg,'location','southwest')
```

```
1  InitClear
2
3
4  dt=0.1 % should be 1/(2df)
5  vt=0:dt:600;
6  %% wave spectra
7  % ---------------------- Jonswap spectrum
8  Hs=6;
9  Tp=10;
10 SigmaApprox=Hs/4;
11
12 % Here frequencies are prescribed indepenbdently of time vector, It's best
       to declare time vector first and use a Nyquist cut-off
13 % It's likely that there will be periodicity it the time vector is longer
14 df=0.005;       % smallest frequency
15 fHighCut=0.2;
16 [ vf_Jonswap,S_Jonswap ] = fJonswap( Hs,Tp,df,fHighCut ); % returns vectos
       of wave frequencies and amplitudes
17 N        = length(vf_Jonswap)       ; % number of component for the sum
      that will be used to compute eta(t)  see slides 02 page 10
18 vphases_Jonswap = rand(1,N)*2*pi ; % random phases between [0 2*pi]
19 vA_Jonswap      = sqrt(2*S_Jonswap*df)    ; % scaled amplitudes according
      to Jonswap spectrum
20
21 % Dispersion for all frequencies (only useful if x!=0)
22 [ vk ] = fgetDispersion( vf_Jonswap(ip),h,g );
23
24 % ---------- Stochastic Times series
25 eta=zeros(1,length(vt));
26 for it=1:length(vt)
27     eta_Jonswap(it) =  sum(vA_Jonswap.*cos((2*pi*vf_Jonswap)*vt(it) - vk*x
           +vphases_Jonswap)); % water elevation, summation of waves
28 end
29 figure,plot(vt,eta_Jonswap,'Color',[0 0 0.7]),box on, grid on,title('Wave
      Time Series'),xlabel('t [s]'),ylabel('Water elevation [m]');
30 [ SBin_eta,fBin_eta,Sraw_eta,fraw_eta ] = fSpectrum( vt,eta_Jonswap,0);
31
32 % ---------- Spectra of Stochastic Times series
33 figure,
34 Ikp=S_Jonswap>10^-5;
35 loglog(fraw_eta,Sraw_eta,'-','Color',[0.66 0.66 0.66])
36 hold on, % stupid matlab
37 % loglog(fWelch,SWelch,'-','Color',[0.35 0.35 0.35])
```

```
38 loglog(vf_Jonswap(Ikp),S_Jonswap(Ikp),'k','linewidth',2)
39 loglog(fBin_eta,SBin_eta,'r-')
40 title('GeneratedTimeSeriesJonswapSpectrum')
41 xlabel('Frequency [Hz]')
42 ylabel('PSD of eta [m^2 s]')
43 grid on
44 lg=legend('Raw spectrum','Jonswap spectrum','Bin-average spectrum');
45 set(lg,'location','southwest')% axis tight
46
47
48
49
50
51 %% Jonswap for 50 year sea state
52 Hs=8.1;
53 Tp=12.70;
54 SigmaApprox=Hs/4;
55
56 nt=3601;
57 vt=linspace(0,3600,nt); % time vector [s]
58 T = vt(2)-vt(1);           % Sample time
59 Fs = 1/T;            % Sampling frequency [Hz}
60 df=1/max(vt);        % smallest frequency
61 fHighCut=Fs/2;       % Nyquist
62 [ vf_Jonswap,S_Jonswap ] = fJonswap( Hs,Tp,df,fHighCut );
63
64 % integration of the spectrum
65 Area=trapz(vf_Jonswap,S_Jonswap);
66 normalization_factor=(Hs/4)^2/Area;
67 S_Jonswap_=normalization_factor*S_Jonswap;
68 Area_=trapz(vf_Jonswap,S_Jonswap_);
69 figure(222)
70 plot(vf_Jonswap,S_Jonswap,'k-'),hold all
71 plot(vf_Jonswap,S_Jonswap_,'b.')
72 grid on
73 xlim([0 0.3])
74 legend('Function','Normalized')
75 xlabel('Frequencies [Hz]')
76 ylabel('JONSWAP Spectral density S [m2.s]')
77
78
79 %% Summation of various wave components using Jonswap spectrum
80 N       = length(vf_Jonswap)       ; % number of component for the sum
      that will be used to compute eta(t)  see slides 02 page 10
81 vphases_Jonswap = rand(1,N)*2*pi ; % random phases between [0 2*pi]
82 vA_Jonswap      = sqrt(2*S_Jonswap*df)    ; % scaled amplitudes according
      to Jonswap spectrum
83 % Dispersion for all frequencies
84 [ vk_Jonswap ] = fgetDispersion( vf_Jonswap,h,g );
85 vf      = vf_Jonswap       ; % vector of frequencies
86 vphases = vphases_Jonswap ; % random phases between [0 2*pi]
87 vA      = vA_Jonswap       ;
88 vk      = vk_Jonswap       ;
89 % param
90 nz=10;
91 [ eta Ftot Mtot Fdrag Finertia vz Stored_u Stored_dudt dFtot dMtot dFdrag
      dFinertia]=  fHydroCalcFinal(vt,q,vf,vk,vA,vphases,g,rhow,h,zBot,D,Cm,
      CD,nz,bWheeler,bFloating,bVzTo0);
92
```

```
93 %% Verification of spectra
94 % Fs = 1/T;            % Sampling frequency [Hz}
95 % df=1/max(vt);        % smallest frequency
96 % fHighCut=Fs/2;       % Nyquist ?
97 % Matlab method
98 nt=length(vt);                  % Length of signal
99 NFFT = 2^nextpow2(nt); % Next power of 2 from length of y
100 Y = fft(eta,NFFT)/nt;
101 Ab=2*abs(Y(1:NFFT/2+1)); % Single sided Amplitude spectrum
102 fb = Fs/2*linspace(0,1,NFFT/2+1);
103 psb=Ab.^2/2/df;
104 % Areab=trapz(fb,Sb);
105 % Area=trapz(f,S);
106
107 % Double Sided -> times 2
108 fB2=[1:nt]*df-df;
109 aB2=abs(fft(eta))/nt; % double sided amplitude spectrum
110 psEtaBo2=aB2.^2/df;psEta2(1)=0;
111
112 figure,hold all
113 plot(fb,psb)
114 plot(fB3,psEtaB2*2)
115 plot(vf_Jonswap,S_Jonswap,'k')
116 legend('Matlab','PSD*2','Jonswap S')
117 xlim([0 0.3]);
118 ylabel('PSD of \eta [m^2/Hz]'),
119 xlabel('Hz');
120
121
122 %%
123 % Plot single-sided amplitude spectrum.
124 figure,grid on,hold all,box on
125 hold all
126 plot(fb,Ab)
127 plot(fB2,aB2*2)
128 plot(vf_Jonswap,vA_Jonswap,'k','LineWidth',2)
129 xlim([0 0.5])
130 ylabel('Single Sided Amplitude Spectrum of eta(t) - a')
131 xlabel('Frequency (Hz)')
132 legend('From Time series','From JONSWAP spectrum')
133 title('SpectrumComparison')
```

## B.2 Functions

```
1 function [ f,S,a ] = fJonswap( Hs,Tp,df,fHighCut )
2 %Returns the Jonswap spectrum
3 %
4 % ---------- Inputs
5 % Hs : significant wave height
6 % Tp : peak period
7 % df : frequency resolution
8 % fHightCut : Highest frequency
9 % ---------- Outputs
10 % f : frequencies
11 % S : spectrum
12 % a : scaled amplitudes for cosine Fourier Series of the form  a.*cos(
       omega t + phi)
```

```
13
14
15 fp=1/Tp; % from previous question
16
17 %% get gamma value according to standards
18
19 if (Tp/sqrt(Hs)<=3.6)==1
20     gamm=5;
21 elseif (Tp/sqrt(Hs)>=3.6)==1 && (Tp/sqrt(Hs)<=5)==1
22     gamm=exp(5.75-1.15*(Tp/sqrt(Hs)));
23 elseif  (Tp/sqrt(Hs)>3.6)==1
24     gamm=1;
25 else
26     disp('Something is wrong with your inputs, model not valid')
27 end
28
29 %% or, prescribe it
30 % gamm=3.3;
31
32
33 %% sigma
34 vFreq=df:df:fHighCut;
35 for iFreq=1:length(vFreq)
36     freq=vFreq(iFreq);
37     if (freq<=fp)==1
38         sigma=0.07;
39     elseif (freq>fp)==1
40         sigma=0.09;
41     end
42     S(iFreq)=0.3125*Hs^2*Tp*((freq/fp)^(-5))*exp(-1.25*(freq/fp)^(-4))*
           (1-0.287*log(gamm))*(gamm)^(exp(-0.5*(((freq/fp)-1)*(1/sigma))^2));
43 end
44
45 % outputs
46 f=vFreq;
47 a= sqrt(2*S*df) ;  % scaled amplitudes according to Jonswap spectrum
```

```
1  function [ Uw, fraw, Sraw,fKailman,SKailman,fBin,SBin,fWelch,SWelch ] =
       fStocWind(Tend )
2  % Generates a wind speed time series from Kaimal spectrum, taking as input
       the length of the time series
3  % This function needs some cleanup
4
5  %% Get theoretical Kaimal spectrum
6  % inputs to the spectrum
7  V10=8;
8  I=0.14;
9  l=340.2;
10
11 % Tmanu=0:0.1:500
12 T=Tend; % duration of time serie in sec
13 % df=1/dt;
14 fHighCut= 10; % cut off frequency in Hz
15 df=1/T;
16 f = 0:df:fHighCut; % freq. vector [Hz]
17
18 % kaimal spectrum
19 Sw = 4*I^2*V10*l ./ (1+(6*f*l)/V10).^(5/3);
20
```

```
%% Generate time serie
% Create two random normal distributed variables
N=T*fHighCut+1;
a_m=0;
a_std=1;
a=a_m+a_std*randn(floor(N/2)+1,1);

b_m=0;
b_std=1;
b=b_m+b_std*randn(floor(N/2)+1,1);

% Create first part of the complex vector
xlhalf1 =a+b*j;
xlhalf1(1)=0;

for i=1:N/2
    wj=2*pi*f(i)/2; % omega
    S_Kaimal_j = 4*I^2*V10*l ./ (1+(6*wj*l)/V10).^(5/3); % theoretical
        spectrum S(omega)
    sigma_K_j=sqrt(T/(2*pi)*S_Kaimal_j);
    xlhalf1(i)=xlhalf1(i)*sigma_K_j;
end

% Create second part of the complex vector
xlhalf2=flipud(xlhalf1(2:end));

% Total complex vector
xl=[xlhalf1;xlhalf2];

% Fourier inverse
Uw=ifft(xl);

% Remove zero imaginairy part and normalize
Uw=sqrt(2)*pi*fHighCut*real(Uw);

%% Plot spectrum and verify agreement
[Sm,F1] = spectrum(Uw,fHighCut,1,2*pi*fHighCut);
% [S_q26,F1_q26] = spectrum_ewma(u,Fsu,K,omegau);
NN = length(Sm);
[Smav,F1av]=spectrum_average(Sm(2:NN/2),F1(2:NN/2));

A=trapz(F1av,Smav)*2*pi*2;
B=var(Uw);
C=mean(Uw);
verif=A-B; % check diff integrate with var


% other method using pwelch
fN= length(Uw)/(2*T); % frequency
nw = 16; % Number of windows (adjust until noise is removed sufficiently)
nfft = round(length(Uw)./nw); % Choose window length
[palt,falt] = pwelch(Uw,nfft); % Normalized results based on Welch's
    method
nfac = fN./pi; falt = falt.*nfac; palt = palt./nfac; % De-normalize
    results

% figure
% loglog(F1(2:floor(NN/2)),2*pi*2*Sm(2:floor(NN/2)),'-','Color',[0.6 0.6
    0.6])  % used to be y
```

```
% hold on
% loglog(falt,palt,'-','Color',[0.25 0.25 0.25]) % used to be g
% loglog(f,Sw,'k-','linewidth',2) % used to be b
% loglog(F1av,2*pi*2*Smav,'-','linewidth',1.5,'Color',fColrs(2)) % used to be k
% title('GeneratedTimeSeriesKaimalSpectrum')
% xlabel('Frequency [Hz]')
% ylabel('PSD')
% grid on
% lg=legend('Raw spectrum','pWelch spectrum','Kaimal spectrum','Bin-averaged spectrum');
% set(lg,'location','southwest')
% % axis tight


%% Preparing outputs
fraw=F1(2:floor(NN/2));
Sraw=2*pi*2*Sm(2:floor(NN/2));
fKailman=f;
SKailman=Sw;
fBin=F1av;
SBin=2*pi*2*Smav;
fWelch=falt;
SWelch=palt;


end







function [S,F1] = spectrum(u,fs,K,omega)
n=floor(length(u)/K); % compute the size of the sample
j=1;
F1 = [0:n-1]/n*fs; % Nyquist frequency
%-----------Splitting the time series-----------------------
for i=1:K
    x(1:length(u(j:j+n-1)),i)=u(j:j+n-1)';
    j=j+n;
end
%-----------------calculating the spectrum-----------------
for i=1:length(x(1,:))
    X1(1:length(x(:,i)),i)=(abs((fft(x(:,i))')).^2)/(omega*length(x(:,1)));
end
%---------------- mean of X1 ----------------------------------------
for i=2:length(x(:,1))
    S(i)=nanmean(X1(i,:));
end
end



function [S_average,f_average]=spectrum_average(S,F1)
% Average neighboring frequency bins.
% INPUTS: Power spectrum and Nyquist Frequency
```

```
% OUTPUTS: Power spectrum and Nyquist Frequency after averaging
    neighboring
% frequency bins.


bins = 15; % tipically between 10 and 20
a = 10^(1/bins); % distance between neighbours
x_x=log10(F1(2));
fo=10^(floor(x_x)); %detection of the lowest bound of the range
F_Lower = F1(1);
F_Higher = a*fo; % higher limit of the first bin
j=0;
while (F_Lower<F1(length(F1)))
    fi=F1((F1>=F_Lower)&(F1<F_Higher));
    Si=S((F1>=F_Lower)&(F1<F_Higher));
    if (~isempty(Si))

        j=j+1;
        S_average(j)=mean(Si);
        f_average(j)=mean(fi);
    end
    F_Lower=F_Higher;
    F_Higher =a*F_Lower;
end

end
```

```
function [vk]=fgetDispersion(vf,h,g)
% Solves for the dispersion relation once and for all
% h: water depth >0
% g: gravity 9.81

for ip=1:length(vf) % loop on frequencies
    if h<100
        [ vk(ip) ] = fkSolve( vf(ip),h,g );
    else
        % Not solving the dispersion relation since we are in deep water
        vOmega=2*pi*vf;
        vk=vOmega.^2/g;
    end
end
end


function [ k ] = fkSolve( f,h,g )
warning off

omega=2*pi*f;
eq=@(k) -omega^2+g*k.*tanh(k*h);
% k=fsolve(eq,(omega^2)/g);
k=fzero(eq,[0 100]); % seems a bit faster


warning on
end
```

```
function [ SBin,fBin,Sraw,fraw ] = fSpectrum( t,q,bPlot )
%returns the spectrum of a time series
% This function needs a lot of cleanup
```

```
4
5  if t(1)~=0
6      t=t-t(1);
7  end
8
9  N_samples= length(q); % number of samples
10 dt= t(end); % time increment of 10 minutes (600 seconds)
11 Fs= N_samples/(dt); % frequencyy
12 % for the annual pattern
13 K=1; % splitting factor
14
15 omega=2*pi*Fs;
16
17 [S_,F_] = spectrum(q,Fs,K,omega);
18 NN = length(S_);
19 % binning the spectrum
20 [S,f]=spectrum_average(S_(2:floor(NN/2)),F_(2:floor(NN/2)));
21
22
23 %%
24 if bPlot==1
25     figure()
26     loglog(F_(2:floor(NN/2)),2*pi*2*S_(2:floor(NN/2)),'-','Color',[0.66
           0.66 0.66])
27     hold on
28     loglog(f,2*pi*2*S,'k-','linewidth',2)
29     xlabel('Frequency [Hz]')
30     grid on
31     legend('Raw PSD','Average PSD')
32 end
33
34 % Outputs
35 SBin=2*pi*2*S;
36 fBin=f;
37 Sraw=2*pi*2*S_(2:floor(NN/2));
38 fraw=F_(2:floor(NN/2));
39 end
40
41 %
42 % sig=q;
43 % figure;subplot(2,1,1)
44 % plot(t,sig,'r')
45 % % ylabel('\eta [m]'); legend('\eta [m]')
46 % xlabel('t [s]')
47 % % Fast Fourier Transform
48 % dfFFT=1/t(end);
49 % fFFT=[1:length(t)]*dfFFT-dfFFT;
50 % a=abs(fft(sig))/length(t);
51 % psSig=a.^2/dfFFT;psSig(1)=0;
52 % subplot(2,1,2),hold all
53 % plot(fFFT,psSig)
54 % plot(F_(2:floor(NN/2)),2*pi*2*S_(2:floor(NN/2)),'m-')
55 % set(gca,'xlim',[0 0.3])
56 % % ylabel('PSD, \eta [m^2/Hz]'),
57 % % legend('PSD, \eta [m^2/Hz]')
58 % xlabel('Hz');
59
60
61
```

```
62 function [S,F1] = spectrum(u,fs,K,omega)
63 n=floor(length(u)/K); % compute the size of the sample
64 j=1;
65 F1 = [0:n-1]/n*fs; % Nyquist frequency
66 %-----------Splitting the time series-----------------------
67 for i=1:K
68     x(1:length(u(j:j+n-1)),i)=u(j:j+n-1)';
69     j=j+n;
70 end
71 %-------------------calculating the spectrum-----------------
72 for i=1:length(x(1,:))
73     X1(1:length(x(:,i)),i)=(abs((fft(x(:,i))')).^2)/(omega*length(x(:,1)))
       ;
74 end
75 %----------------- mean of X1 ---------------------------------
76 for i=2:length(x(:,1))
77     S(i)=nanmean(X1(i,:));
78 end
79 end
80
81
82
83 function [S_average,f_average]=spectrum_average(S,F1)
84 % Average neighboring frequency bins.
85 % INPUTS: Power spectrum and Nyquist Frequency
86 % OUTPUTS: Power spectrum and Nyquist Frequency after averaging
     neighboring
87 % frequency bins.
88
89
90 bins = 15; % tipically between 10 and 20
91 a = 10^(1/bins); % distance between neighbours
92 x_x=log10(F1(2));
93 fo=10^(floor(x_x)); %detection of the lowest bound of the range
94 F_Lower = F1(1);
95 F_Higher = a*fo; % higher limit of the first bin
96 j=0;
97 while (F_Lower<F1(length(F1)))
98     fi=F1((F1>=F_Lower)&(F1<F_Higher));
99     Si=S((F1>=F_Lower)&(F1<F_Higher));
100    if (~isempty(Si))
101
102        j=j+1;
103        S_average(j)=mean(Si);
104        f_average(j)=mean(fi);
105    end
106    F_Lower=F_Higher;
107    F_Higher =a*F_Lower;
108 end
109
110 end
```