```
clc; close all; clear;
```

Rigoni Luca 247451

Renewable Energy Convertion Systems - Assigment 1 - Energy evaluation of an off-shore Wind Turbine

Table of Contents

Rigoni Luca 247451 1
Static analysis 2
1) Long term statistics - yearly statistics 2
2) Stall regulated turbine 4
Tip Speed Ratio 4
Power factor curve: 4
Power profile 6
Betz limit 7
a) Stall-regulated Turbine Power Curves 9
Average Annual Energy Production - stall regulated 10
b) Pitch-regulated Turbine Power Curves 12
c1) Steady state energy losses 14
Theory: Electro Mechanical Conversion 14
a) Electro Mechanical Energy Conversion 14
b) WT Categorization 15
c) Power Converter considerations: 15
Theory: Modelling PMSM 15
c2) Turbine average electromechanical efficiency over a year, for the case of pitch regulated variable speed turbine 24
Dynamic Analysis 25
Constant input 1: 25
Constant input 2: 26
Ramp input 3: 26
Ramp input 4: 26
Kaimal distribution - short term statistics 27
Automatic simulink simulations 31
Simulink output 32
Anemometer noise 42
Automatic simulink simulations with ANEMOMETER NOISE 43

Consider the DTU 10 MW reference WT

The DTU Wind Turbine is a reference turbine not necessarly meant for installation, but rather meant to be used as a case of study for subsystems and controllers design the DTU turbine has a target power of 10 MW and is meant for installation offshore.

Assumptions:

- Known areodynamics: the C_p "Power Coefficient" curve
- Floating dynamics not considered, because Wind Turbine is bottom fixed.
- yaw dynamics are not considered, because the rotor axis is always aligned to the wind direction.

Numerical data:

```
nob = 3;                % nº of blades
V_ci = 4;               % cut-in wind speed [m/s]
V_co = 25;              % cut-off wind speed [m/s]
R_rotor = 178.3/2;      % rotor radius [m]
A_rotor = pi*R_rotor^2; % rotor area [m^2]
H_hub = 119;            % hub height [m]
rho_air = 1.225;        % air density [kg/m^3]
V_rated = 11.4;         % rated wind speed [m/s]
Jeq = 1.56e8;           % equivalent inertia at rotor [kg*m^2]
B_r = 2e5;              % rotor damping (equivalent losses) coefficient [N*m*s/rad]

p = 320;                % Pole pairs [-]
m_flux = 19.49;         % Magnetic flux1 [Wb]
R_s = 64*10^-3;         % Stator resistance [Ohm]
L_q = 1.8*10^-3;        % q-axis inductante [H]
L_d = 1.8*10^-3;        % d-axis inductante [H]
f_s = 1;                % Switching frequency of the inverter [kHz]
```

Extract useful data:

```
raw_data = readcell('Wind_data.csv');
data = raw_data(2:end);
data(strcmp(data, 'NULL')) = {-1}; % Replace 'NULL' with -1
raw_numericData = cell2mat(data);
[size_raw_numericData_r, size_raw_numericData_c] = size(raw_numericData);
numericData = raw_numericData(raw_numericData ~= -1);
[size_numericData_r, size_numericData_c] = size(numericData);
```

Considerations on time:

```
dt = 10*60; % [s]
T_time_s = size_numericData_r * dt;
T_time_h = T_time_s / (60*60);
T_time_y = T_time_h / (24*360);
disp(['Total acquisition time: ', num2str(T_time_y), ' years'])
```

```
Total acquisition time: 1.9709 years
```

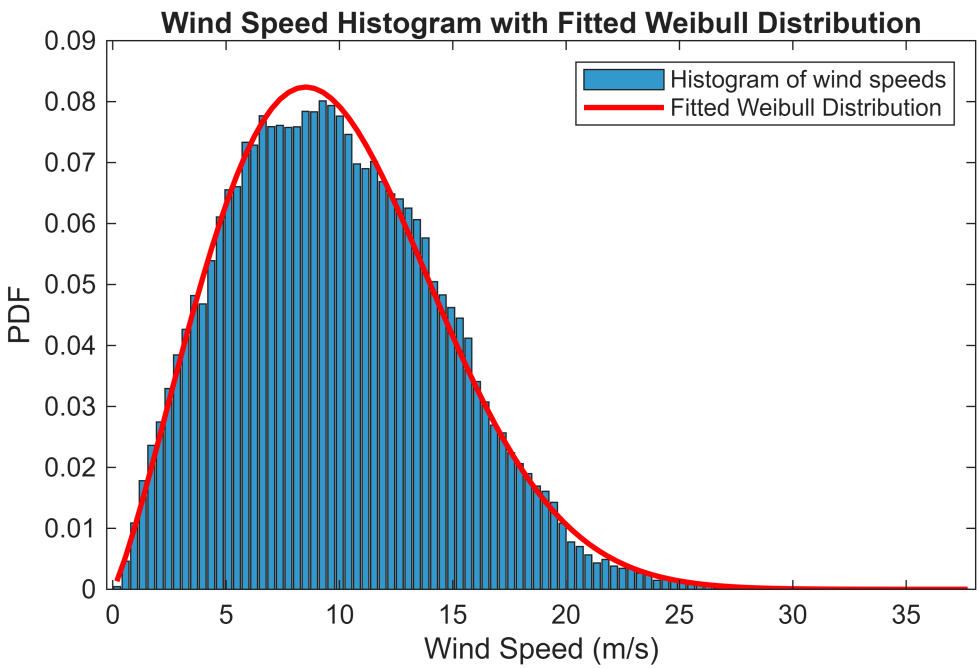
Static analysis

1) Long term statistics - yearly statistics

Generate a PDF of the wind speed over a year and find fit parameters of Weibull distribution

```
wind_speed = numericData;

num_bins = 100;
[counts, edges] = histcounts(wind_speed, num_bins, 'Normalization', 'pdf');
```

```
bin_centers = (edges(1:end-1) + edges(2:end)) / 2;

% Fit the wind speed data to a Weibull distribution
weibull_fit = fitdist(wind_speed, 'Weibull');

figure;
% histogram(wind_speed, 'Normalization','probability');
bar(bin_centers, counts,  'FaceColor', [0.2, 0.6, 0.8]);
hold on;

% Plot the fitted Weibull distribution over the histogram
x_values = linspace(min(wind_speed), max(wind_speed), 100);  % Create a range of x
values
y_values = pdf(weibull_fit, x_values);  % Compute the Weibull PDF for these x values

plot(x_values, y_values, 'r-', 'LineWidth', 2);  % Plot the Weibull PDF (red line)

xlabel('Wind Speed (m/s)');
ylabel('PDF');
title('Wind Speed Histogram with Fitted Weibull Distribution');
legend('Histogram of wind speeds', 'Fitted Weibull Distribution');
hold off;
```




```
% Get the parameters of the Weibull distribution
shape_param = weibull_fit.a;  % Shape parameter (k)
scale_param = weibull_fit.b;  % Scale parameter (lambda)

% Display the parameters
disp(['Shape parameter (k): ', num2str(shape_param)]);
```

```
Shape parameter (k): 11.1661
```

```
disp(['Scale parameter (lambda): ', num2str(scale_param)]);
```

```
Scale parameter (lambda): 2.2131
```

2) Stall regulated turbine

A stall regulated wind turbine is used whan I want to limit the power output passively at high wind speeds without needing any acttive pitch control.

This design relies on the areodynamic properties of the blades: as the wind speed increases, the flow separation (stall) naturally reduces lift, preventing excessive power generation.

It's a simple solution that require very low maintenace costs.

Tip Speed Ratio

$$\lambda = \frac{\omega R}{V_\infty}$$

- ω is the rotor angular velocity
- R is the rotor radius
- V_∞ is the undisturbed wind speed

It is a key parameter in Blade Eleemtn Momentum (BEM) theory, it determines the areodynamic efficiency of the rotor. An optimal Tip Speed Ratio ensures the bes tradeoff between power output and areodynamic performance.

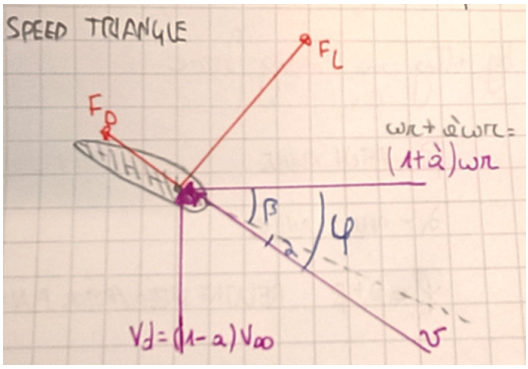
Calculate the mode of the Weibull distribution, the most frequent wind speed value:

```
[max_pdf, idx] = max(y_values);
mode_weibull = x_values(idx);
disp(['Most frequent wind speed: ', num2str(mode_weibull),' [m/s]']);
```

```
Most frequent wind speed: 8.52 [m/s]
```

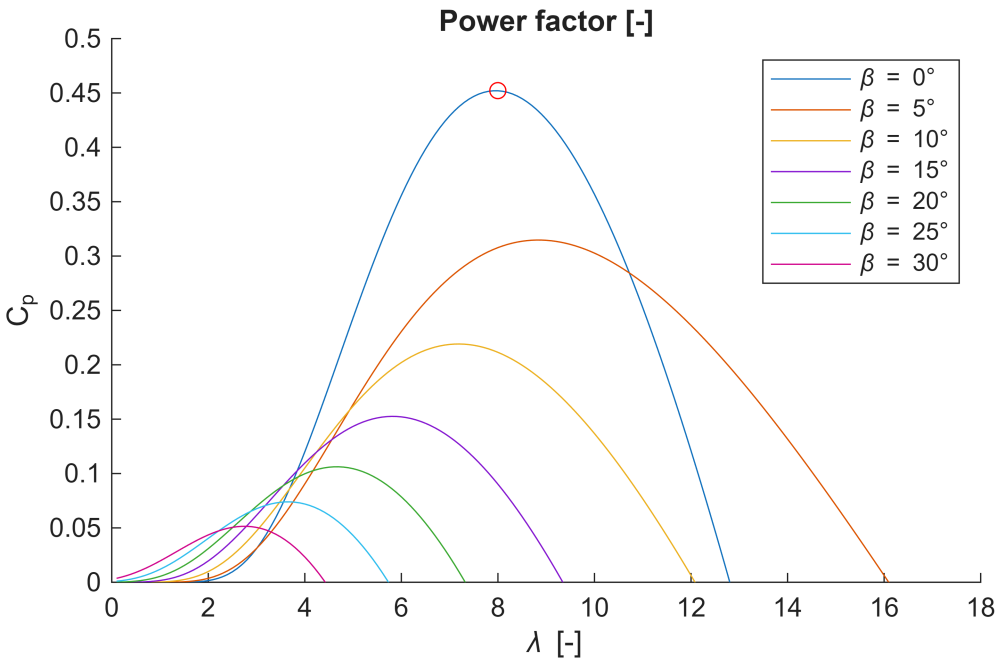
Power factor curve:

- increasing pitch angle β the peak is reduced due to drag coefficient that increase
- amall angles of β allow operation at high λ values, meaning higher rotor speed, as they come with reduction in drag forces (lowering the attack angle α)



- lowering λ too much increase a lot α that leads to STALL
- increasing λ too much lower a lot α and DRAG effects become dominant

```
% -- BUILD AXIS
theta_vec = (0:5:30)';          % blades pitch angle axis [deg]
lambda_sweep = (0.1:0.1:18)';   % tip speed ratio axis []

% -- CREATE MAP
Cp_map = zeros(length(lambda_sweep),length(theta_vec));
figure
hold on
for i=1:1:length(theta_vec)
    theta = theta_vec(i);
    % -- EXPONENTIAL FITTING FORMULA
    Cp_map(:,i) = PowerFactor(lambda_sweep,theta);
    plot(lambda_sweep,Cp_map(:,i),'-')
    leg{i} = ['\beta = ',num2str(theta),'°'];
end

% -- FIND PEAK POWER FACTOR VALUE lambda and pitch
% lambda corresponding to the peak (pitch = 0°)
lambda_opt = lambda_sweep(find(~(Cp_map(:,1) - max(Cp_map(:,1)))));
% one_over_beta_opt = 1./(lambda_opt) - 0.035;
% Cp_opt = 0.5 * (116 * one_over_beta_opt - 5)*exp(-21*one_over_beta_opt);

Cp_opt = max(Cp_map(:,1));      % optimal Cp for beta = 0
% clear('one_over_beta_opt');
Trated = Cp_opt/lambda_opt*0.5*rho_air*A_rotor.*V_rated^2*R_rotor;

plot(lambda_opt,Cp_opt,'or')
hold off
ylim([0 0.5])
title('Power factor [-]')
xlabel(' \lambda [-]')
ylabel('C_p')
legend(leg)
```



So we can extract the optimal value for lambda and the corresponding Power Factor:

```
disp(['Tip Speed ratio optimal value: ', num2str(lambda_opt),' [-]']);
```

```
Tip Speed ratio optimal value: 8 [-]
```

```
disp(['Power factor optimal value: ', num2str(Cp_opt),' [-]']);
```

```
Power factor optimal value: 0.45201 [-]
```

Power profile

Wintd turbines can be **fixed-speed** or **variable-speed**

- **fixed-speed** wind turbine:

rotor operates at constant angular speed

this solution was used in the first stages of wind power, nowadays it is applicable only in small scale WT, whereas large (MW-scale) turbines are typically speed controlled

- **variable-speed** wind turbine:

rotor speed is controlled to maximise the areodynamic performance

the speed control is performed at the electric **drive level**

- WT can be also equipped with a **pitch-regulation mechanism,** that allows the blades to rotate and changing the pitch angle β. Pitch regulation is used to limit the mechanical power avaiable to the rotor in order to prevent overloads

- if the wind turbine is not equipped with pitch control, the blades are designed in sucha way that, at high wind speeds, the blades stall and the power output falls (**stall-regulated WT**)

- the main advantage to design a WT with pitch regulation is to achieve nearly constant power output over a wide range of wind speeds.

It becomes necessary the pitch control over the rated output power of 10 MW in the DTU Wind Turbine, so let's define the working regions of a WT of a variable speed + pitch control:

```
% Generate array of wind speeds ranging from 0 to CutOff wind speed
wind_speed_sweep = (0:0.1:V_co+5)';

% Define working regions I II III IV
I1 = find(wind_speed_sweep < V_ci);
I2 = find(wind_speed_sweep >= V_ci & wind_speed_sweep <= V_rated);
I3 = find(wind_speed_sweep > V_rated & wind_speed_sweep <= V_co);
I4 = find(wind_speed_sweep > V_co);
```

Compute the rated power through the definition of Power factor:

$$C_p = \frac{P}{\frac{1}{2}\rho V_\infty{}^3 A_d}$$

- P is the actual power
- $\frac{1}{2}\rho V_\infty{}^3 A_d$ is the available power of fluid stream with the same area of the rotor

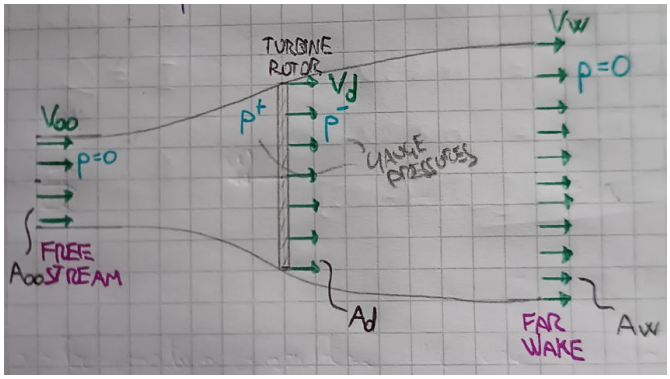
```
Prated = 0.5*rho_air*V_rated^3*A_rotor*Cp_opt;

% Define Maxinum Power for each region
Pm = 0*wind_speed_sweep;
Pm(I2) = 0.5*rho_air*wind_speed_sweep(I2).^3*A_rotor*Cp_opt;
Pm(I3) = Prated; %*ones(size(I3));

% Compute available wind power for every wind speed
Pw = 1/2*rho_air*wind_speed_sweep.^3*A_rotor;
Pw2 = 1/2*rho_air*wind_speed.^3*A_rotor;
```

Betz limit

Theoretical maxinum power that a WT can extract from a wind stream:

Assumptions:

- uniform pressure
- unoform speed
- incompressible fluid



The wind speed reduces becose the turbine "removes" kinetic energy from fluid stream.

$V_\infty > V_w$

$V_d = (1-a)\,V_\infty$ where a is the **axial induction factor**

$\dot{m} = A_\infty V_\infty \rho = A_d\,V_d\,\rho = A_w V_w\,\rho$ constant mass flow rate

PRESSURE $F_T = (p^+ - p^-)A_d$

MOMENTUM $F_T = \dot{m}\,(V_\infty - V_w) = A_d\,V_d\,\rho\,(V_\infty - V_w) = A_d\,(1-a)\,V_\infty\,\rho\,(V_\infty - V_w)$

Apply Bernoulli: $p_1 + \rho g\,z_1 + \frac{1}{2}\rho\,V_1^2 = p_2 + \rho g\,z_2 + \frac{1}{2}\rho\,V_2^2$

$$\begin{cases} \frac{1}{2}\rho\,V_\infty^2 = p^+ + \frac{1}{2}\rho\,V_d^2 \\ p^- + \frac{1}{2}\rho\,V_d^2 = \frac{1}{2}\rho\,V_w^2 \end{cases} \quad \text{--> } p^+ - p^- = \frac{1}{2}\rho\left(V_\infty^2 - V_w^2\right)$$

$F_T = \frac{1}{2}\rho\left(V_\infty^2 - V_w^2\right)A_d = A_d\,(1-a)\,V_\infty\,\rho\,(V_\infty - V_w)$ --> $V_w = (1-2a)\,V_\infty$

$$\begin{cases} V_d = (1-a)\,V_\infty \\ V_w = (1-2a)\,V_\infty \end{cases}$$

Thrust:

$F_T = \frac{1}{2}\rho\left(V_\infty^2 - V_w^2\right)A_d$

$F_T = 2\rho A_d\,a\,(1-a)V_\infty^2$

Power:

$P = F_T\,V_d$

$P = F_T\,(1-a)V_\infty$

$P = 2\rho A_d\,a\,(1-a)^2 V_\infty^3$

Find maxinum value:

$$\frac{\mathrm{d}}{\mathrm{d}a}P = 0 \quad \text{--> } a = \frac{1}{3}$$

$$P_{\mathrm{MAX}} = \frac{8}{27}\rho A_d V_\infty^3$$

Max Power factor

$$C_P = \frac{16}{27} \approx 0.59$$

```
Pb = 16/27 * Pw;

figure;
hold on
plot(wind_speed_sweep,Pw/1e6,'k','linewidth',1) % Available wind power
plot(wind_speed_sweep,Pb/1e6,'b','linewidth',1) % Betz limit
plot(wind_speed_sweep,Pm/1e6,'--r','linewidth',1) % Rated limit

ylim([0 15])
grid on
xlabel('\itV_w\rm [m/s]','FontSize',12,'fontname','times new roman')
ylabel('\itP_r\rm [MW]','FontSize',12,'fontname','times new roman')
set(gca,'FontSize',12,'fontname','times new roman')
```

a) Stall-regulated Turbine Power Curves

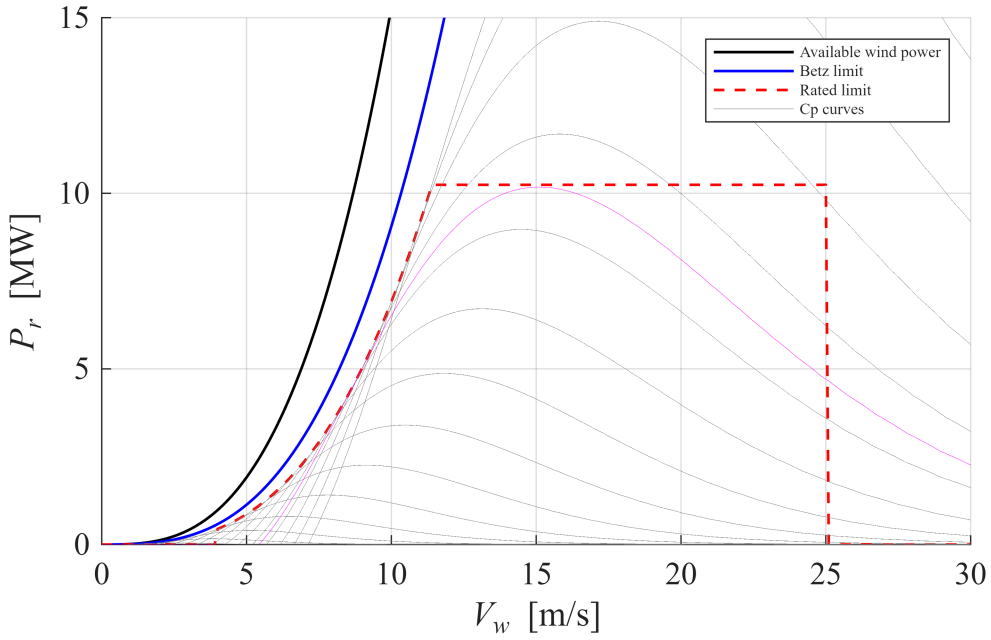
We are interested in the maxinum turbine rotation velocity without exceeding the limit of 10 MW of produced energy.

```
% powers obtained for different angular speeds
Omega_v = linspace(V_rated/R_rotor*1,V_rated/R_rotor*8,14)'; % vector of angular
speeds
for i = 1:14 % for each angular speed
    Po = Pw.*interp1(lambda_sweep,Cp_map(:,1),Omega_v(i)*R_rotor./
wind_speed_sweep,'linear');
    plot(wind_speed_sweep,1e-6*Po,'color',[0.5 0.5 0.5],'linewidth',0.1)
end
```

Given this figure choose the stall regulated rotor constant speed such that the peak do not exceed the rated power of 10 MW.

```
Omega_c = 0.955*Omega_v(11);
Pc = Pw.*interp1(lambda_sweep,Cp_map(:,1),Omega_c*R_rotor./
wind_speed_sweep,'linear');
Pc(isnan(Pc)) = 0;
plot(wind_speed_sweep,Pc/1e6,'color',[1 0 1],'linewidth',0.15) % Pink Stall
regulated
```

```
legend('Available wind power','Betz limit','Rated limit','Cp
curves','Location','northeast','FontSize',6);
hold off;
```




```
disp(['Rotor optimal angular velocity: ', num2str(Omega_c), ' rad/s']);
```

```
Rotor optimal angular velocity: 0.77969 rad/s
```

Average Annual Energy Production - stall regulated

In order to perform a reliable estimate of the Genrated Power some assumptions has to be made:

- The turbine behaviour is ideal, meaning the power produced a ta certain wind speed can be read out from the theoretical power curve
- All damping and generater losses can be neglected

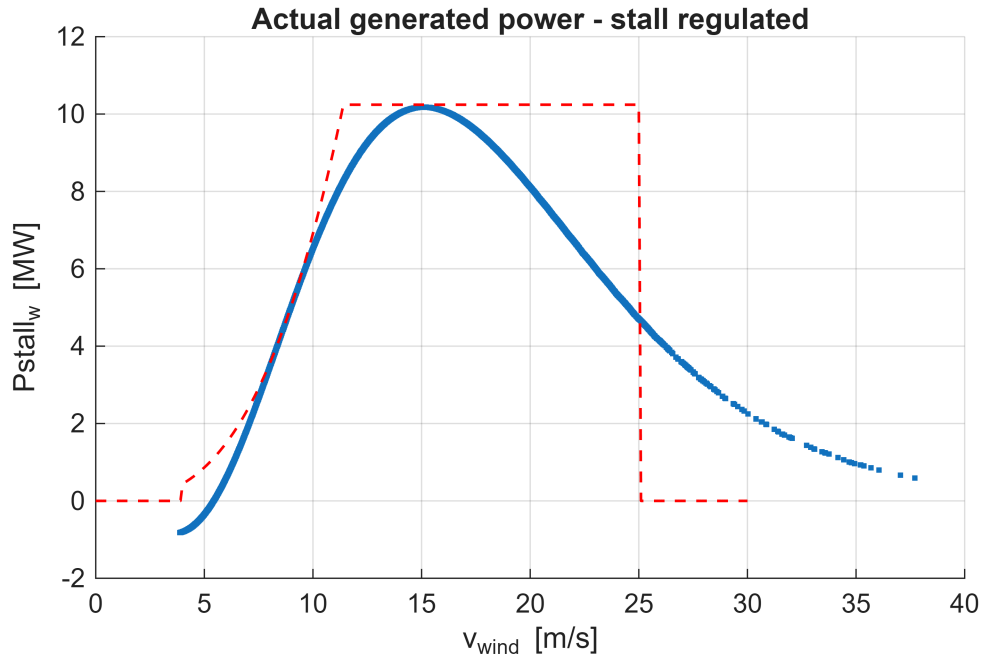
Consider the **Available Wind Power**:

```
Pwa = 0.5*rho_air*wind_speed.^3*A_rotor; % [W]
```

Compute the **Generated Power** output for a stall-regulated WT using linear interpolation of the power coefficient map (Cp_map) at the istantaneous tip-speed ratio:

```
Pg = Pwa.*interp1(lambda_sweep, Cp_map(:,1), Omega_c*R_rotor./wind_speed); % [W]
% Power generated = Fluid stream power * Cp (evaluated at actual lambda);
% lambda = wR   /  actual wind speed
%        (const) (at each time istant)

figure;
hold on;
plot(wind_speed, Pg/1e6, '.', 'LineWidth', 1);
```

```
plot(wind_speed_sweep,Pm/1e6,'--r','linewidth',1)
title('Actual generated power - stall regulated');
xlabel('v_{wind} [m/s]');
ylabel('Pstall_{w} [MW]');
grid on;
hold off;
```




```
Pg(isnan(Pg)) = 0;
```

One wind speed sample evey 10 min -> consider constant power for 10 min

```
energy_per_interval_sr_MWs = Pg * 10*60 / 1e6; % [MW * s]
energy_per_interval_sr_MWh = energy_per_interval_sr_MWs / (60*60); % [MWh] 3600s =
1h
```

Sum the total energy produce in each time interval

```
E_tot_sr_MWs = sum(energy_per_interval_sr_MWs(energy_per_interval_sr_MWs > 0)); %
[MWs]
E_tot_sr_MWh = sum(energy_per_interval_sr_MWh(energy_per_interval_sr_MWh > 0)); %
[MWh]

E_tot_1year_sr_TJ = E_tot_sr_MWs / T_time_y * 1e-6; % [TJ]
E_tot_1year_sr_MWh = E_tot_sr_MWh / T_time_y; % [MWh]

disp(['Stall Regulated WT'])
```

```
Stall Regulated WT
```

```
disp(['Average annual energy: ', num2str(E_tot_1year_sr_TJ), ' TJ/year']);
```

```
Average annual energy: 162.933 TJ/year
```

```
disp(['Average annual energy: ', num2str(E_tot_1year_sr_MWh), ' MWh/year']);
```

```
Average annual energy: 45259.1637 MWh/year
```

b) Pitch-regulated Turbine Power Curves

Assume again a perfect behaviour:

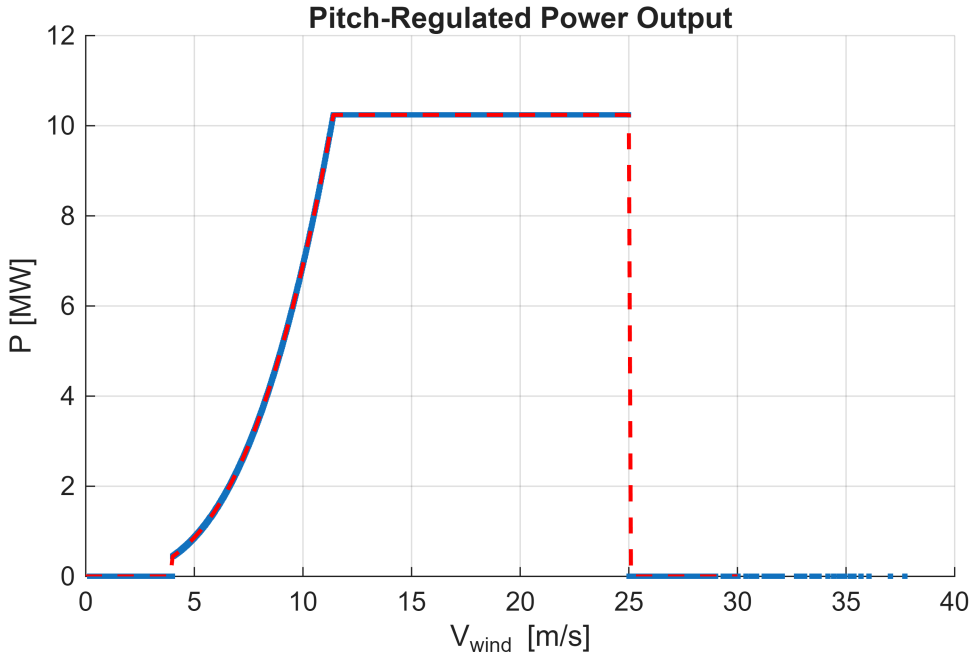
- the ideal working point is immediately reached in the Maxinum Power Point Tracking region II (MPPT)
- rated power of 10 MW is constantly kept in region III at rated conditions

Assume to cosnider no power generated:

- below cut-in wind speed (region I) because of lack of power to overcome inertia
- above cut-off wind speed (region IV) because stall is necessary to prevent oveload damages

NOTE: this is possible due to variable rotor speed ω that allows always the ideal tip-speed ratio λ to maximise Power Factor C_p

```
n_sample = length(wind_speed);
P2 = zeros(n_sample,1);
for i = 1:n_sample
    if wind_speed(i) >= V_ci && wind_speed(i) <= V_rated
        P2(i) = 0.5 * rho_air * (wind_speed(i).^3) * A_rotor * Cp_opt;
    elseif wind_speed(i) >= V_rated && wind_speed(i) <= V_co
        P2(i) = Prated;
    end
end

figure;
hold on;
plot(wind_speed, P2/1e6, '.')
plot(wind_speed_sweep,Pm/1e6,'--r','linewidth',1.5)
title('Pitch-Regulated Power Output');
xlabel('V_{wind} [m/s]');
ylabel('P [MW]');
grid on
hold off;
```




```
sum_of_powers = 0; % [W]
for i = 1:n_sample
    if wind_speed(i) >= V_ci && wind_speed(i) <= V_rated
        sum_of_powers = sum_of_powers +
0.5*rho_air*(wind_speed(i)^3)*A_rotor*Cp_opt;
    elseif wind_speed(i) >= V_rated && wind_speed(i) <= V_co
        sum_of_powers = sum_of_powers + Prated;
    end
end

sum_of_energies = sum_of_powers * 10 * 60; % [J]
E_tot_1year_pr_TJ = sum_of_energies / T_time_y / 1e12; % [TJ]
E_tot_1year_pr_MWh = E_tot_1year_pr_TJ * 1e6 / 3600; %

disp(['Annual Energy Production - Stall Regulated'])
```

```
Annual Energy Production - Stall Regulated
```

```
disp(['Average annual energy: ', num2str(E_tot_1year_sr_TJ), ' TJ/year']);
```

```
Average annual energy: 162.933 TJ/year
```

```
disp(['Average annual energy: ', num2str(E_tot_1year_sr_MWh), ' MWh/year']);
```

```
Average annual energy: 45259.1637 MWh/year
```

```


disp(['Annual Energy Production - Pitch Regulated'])
```

```
Annual Energy Production - Pitch Regulated
```

```
disp(['Average annual energy: ', num2str(E_tot_1year_pr_TJ), ' TJ/year']);
```

```
Average annual energy: 181.3094 TJ/year
```

```
disp(['Average annual energy: ', num2str(E_tot_1year_pr_MWh), ' MWh/year']);
```

```
Average annual energy: 50363.71 MWh/year
```

```

```

Compare harvested power:

```
eta = E_tot_1year_pr_TJ / E_tot_1year_sr_TJ - 1;
disp('The pitch regulated steady state regulation allow the turbine to');
```

```
The pitch regulated steady state regulation allow the turbine to
```

```
disp(['harvest ', num2str(eta*100), '% more energy compared to the stall regulated
WT']);
```

```
harvest 11.2785% more energy compared to the stall regulated WT
```

c1) Steady state energy losses

Joule and **viscous** are the main camponent that cause energy losses in the system. Let's compute them in function of the wind speed.

We consider the equation of motion. Since, for simplicity, we assume steady state conditions, the rotor has constant angular speed and no acceleration.

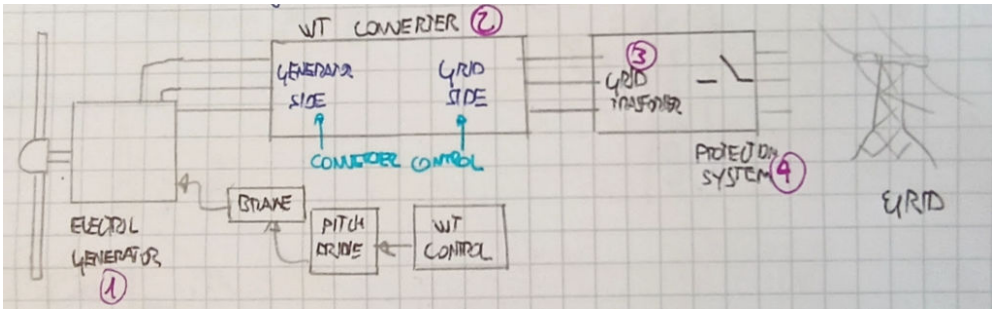
$$\cancel{J_r\dot{\omega}} + B_r\omega = T_w^r - T_m^r$$

- J_r is the equivalent moment of inertia (rotor + transmission + generator)
- B_r is the damping coefficient (due to bearing and gears)
- T_w is the wins generated torque
- T_m is the controllable torque applied by the machine

Theory: Electro Mechanical Conversion

a) Electro Mechanical Energy Conversion

Scheme of connection betweeen the generator and the grid:



1. **Electric Generator** performs conversion from mechanical to electrical power, thanks to the rotating magnetic flux acting on the 3 phase coils
2. **Power Converters** the Rectifier and the Inverter (AC->DC; DC->AC) have decouple the voltages from the generator side to the grid side.
3. **Grid Trasformes** is a step-up voltage trasformer in order to adapt the voltage level to the grid voltage level.
4. **Protection System** is a switch that activates in case of faults occur, in order to isolate the problematic components.

b) WT Categorization

- TYPE I: Direct Connection of Induction Generator (fixed speed, soft starter, low efficiency)
- TYPE II: Direct Connection of Induction Generator (rotor resistances allow to obtain nominal torque at different speeds)
- TYPE III: Doubly-fed Induction Generator (back-to-back configuration allow partial decouple rotor frequency from grid frequency (Rectifier + Inverter) allow variable speed operation)
- TYPE IV: Full Scale back-to-back power converter ensures full decoupling in order to allow variable speed

The DTU 10 MW WT is a TYPE IV, TYPE D Wind Turbine.

It is a Permanent Magnet Synchronous Machine (**PMSG**) with direct drive connection.

c) Power Converter considerations:

Most used converter has 3-phase 2 voltage level in back-to-back configuration, meaning there are both the Rectifier and the Inverter allowing full decoupling of the voltage frequency between generator side and grid side.

Note that by increasing the rated power level of few MW the 2L-VSC becomes unsuitable, Multi Voltage Source Converters are needed.

To increase reliability multiple power conversion lines in parallel are used

Theory: Modelling PMSM

Build a mathematical model of the PMSM to be able to design the controllers to regulate current, torque, speed in static and dynamic conditions.

Focus on the 3 electrical terminals of the stator, a set of e.m.f. is induced by a rotating magnetic flux. The 3-phase voltages are:

$$u_a(t) + R\, i_a(t) = \frac{\mathrm{d}}{\mathrm{d}t}\lambda_a(t)$$

$$u_b(t) + R\, i_b(t) = \frac{\mathrm{d}}{\mathrm{d}t}\lambda_b(t)$$

$$u_c(t) + R\, i_c(t) = \frac{\mathrm{d}}{\mathrm{d}t}\lambda_c(t)$$

- R phase resistance
- λ magnetic flux linkage
- i state phase current

[induced voltage is proportional to the rate of change of the magnetic flux]

The **flux linkage** combines:

- the magnetic flux produced by the PM in the rotor
- the magnetic flux produced by the 3-phase currents

Remember the two contributions act in opposite sense:

$$\lambda_a(t) = \lambda_{a,\mathrm{pm}}(t) - \lambda_{a,i}(t)$$

$$\lambda_b(t) = \lambda_{b,\mathrm{pm}}(t) - \lambda_{b,i}(t)$$

$$\lambda_c(t) = \lambda_{c,\mathrm{pm}}(t) - \lambda_{c,i}(t)$$

Remember the flux linkages due to the PM are sinusoidal due to the rotation and the 120° phase:

$$\lambda_{a,\mathrm{pm}}(t) = \Lambda_{\mathrm{pm}} \cos(\theta_{\mathrm{me}})$$

$$\lambda_{b,\mathrm{pm}}(t) = \Lambda_{\mathrm{pm}} \cos\left(\theta_{\mathrm{me}} - \frac{2}{3}\pi\right)$$

$$\lambda_{c,\mathrm{pm}}(t) = \Lambda_{\mathrm{pm}} \cos\left(\theta_{\mathrm{me}} - \frac{4}{3}\pi\right)$$

- $\theta_{\mathrm{me}}(t)$ is angle between axis of phase and axis of flux produced by PM

Magnetic flux produce the stator currents

Assume the stator is symmetric:

- Self inductances: $L_a = L_b = L_c = L_S$
- Mutaual inductances: $L_{\mathrm{Mab}} = L_{\mathrm{Mbc}} = L_{\mathrm{Mac}} = -|L_M|$

$$\lambda_{a,i} = L_a\, i_a + L_{\mathrm{Mab}}\, i_b + L_{\mathrm{Mac}}\, i_c = L_S\, i_a - |L_M|(i_b + i_c)$$

$$\lambda_{b,i} = L_{\mathrm{Mab}}\, i_a + L_b\, i_b + L_{\mathrm{Mbc}}\, i_c = L_S\, i_b - |L_M|(i_a + i_c)$$

$$\lambda_{c,i} = L_{\mathrm{Mac}}\, i_a + L_{\mathrm{Mbc}}\, i_b + L_c\, i_c = L_S\, i_c - |L_M|(i_a + i_b)$$

- remember: $i_a + i_b + i_c = 0$
- and set: $L = L_S + |L_M|$

$$\lambda_{a,i} = L\, i_a$$

$$\lambda_{b,i} = L\, i_b$$

$$\lambda_{c,i} = L\, i_c$$

NOTE: Apparently the magnetic flux of each phase depends only on the current of the same phase

Remember we are interested in the global magnetic flux linkage, so we obtain the expression with the 2 contributions:

$$\lambda_a = -L\, i_a + \Lambda_{\text{pm}} \cos(\theta_{\text{me}})$$

$$\lambda_b = -L\, i_b + \Lambda_{\text{pm}} \cos\left(\theta_{\text{me}} - \frac{2}{3}\pi\right)$$

$$\lambda_c = -L\, i_c + \Lambda_{\text{pm}} \cos\left(\theta_{\text{me}} - \frac{4}{3}\pi\right)$$

Now jump another step back, we are interested in the voltage relation:

$$u_a(t) + R\, i_a(t) = \frac{\mathrm{d}}{\mathrm{d}t}\lambda_a(t)$$

$$u_b(t) + R\, i_b(t) = \frac{\mathrm{d}}{\mathrm{dt}}\lambda_b(t)$$

$$u_c(t) + R\, i_c(t) = \frac{\mathrm{d}}{\mathrm{dt}}\lambda_c(t)$$

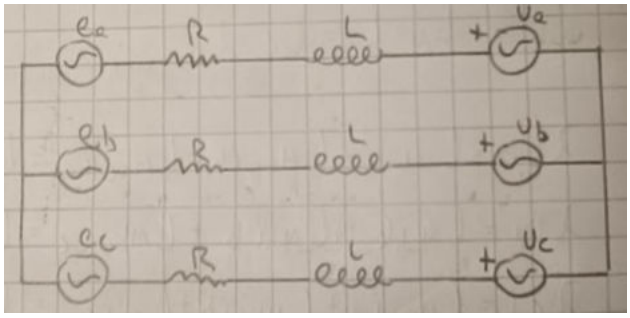
Substituting:

$$u_a + R\, i_a + L\frac{\mathrm{d}}{\mathrm{dt}} i_a = e_a \qquad e_a = \frac{\mathrm{d}}{\mathrm{dt}}\lambda_{a,\text{pm}}(t) = \Lambda_{\text{pm}} \cos\left(\theta_{\text{me}} + \frac{\pi}{2}\right)$$

$$u_b + R\, i_b + L\frac{\mathrm{d}}{\mathrm{dt}} i_b = e_b \qquad e_b = \frac{\mathrm{d}}{\mathrm{dt}}\lambda_{b,\text{pm}}(t) = \Lambda_{\text{pm}} \cos\left(\theta_{\text{me}} + \frac{\pi}{2} - \frac{2}{3}\pi\right)$$

$$u_c + R\, i_c + L\frac{\mathrm{d}}{\mathrm{dt}} i_c = e_c \qquad e_c = \frac{\mathrm{d}}{\mathrm{dt}}\lambda_{c,\text{pm}}(t) = \Lambda_{\text{pm}} \cos\left(\theta_{\text{me}} + \frac{\pi}{2} - \frac{4}{3}\pi\right)$$

Circuit representation:



Electro-Mechanical energy conversion in PMSM comes form the interaction between two magnetically coupled independent subsystems. i.e. the magnetic field from the PM and the rotating magnetic field from the stator currents.

Power balance

It can be derived from each phase voltage equation, by multiplying it by the corresponding phase current:
$P = v\, i$

$$\left[u_a + R\, i_a + L\frac{\mathrm{d}}{\mathrm{dt}} i_a = e_a\right] i_a = P_a$$

$$\left[u_b + R\, i_b + L\frac{\mathrm{d}}{\mathrm{dt}} i_b = e_b\right] i_b = P_b$$

$$\left[u_c + R\, i_c + L\frac{\mathrm{d}}{\mathrm{dt}} i_c = e_c\right] i_c = P_c$$

$$u_a i_a + u_b i_b + u_c i_c + R\left(i_a^2 + i_b^2 + i_c^2\right) + \frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{1}{2} L\left(i_a^2 + i_b^2 + i_c^2\right)\right] = e_a i_a + e_b i_b + e_c i_c$$

- $u_a i_a + u_b i_b + u_c i_c$ istantaneous power directly injected into the grid
- $R\left(i_a^2 + i_b^2 + i_c^2\right)$ dissipated power
- $\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{1}{2} L\left(i_a^2 + i_b^2 + i_c^2\right)\right]$ power used to charge the magnetic field
- $e_a i_a + e_b i_b + e_c i_c$ electro-mechanical power

Torque derivation

$$e_a i_a + e_b i_b + e_c i_c = T_m^m\, \omega_m = \frac{T_m^m\, \omega_{\mathrm{me}}}{p}$$ p->pole pairs

$$T_m^m = p\, \Lambda_{\mathrm{pm}}\left[i_a\, \cos\left(\theta_{\mathrm{me}} + \frac{\pi}{2}\right) + i_b \cos\left(\theta_{\mathrm{me}} + \frac{\pi}{2} - \frac{2}{3}\pi\right) + i_c\, \cos\left(\theta_{\mathrm{me}} + \frac{\pi}{2} - \frac{4}{3}\pi\right)\right]$$

We can expect the currents to be sinusoidal and to have a phase shift in respect to the corresponing e.m.f.

$$i_a = I_M(t) \cos\left(\theta_{\mathrm{me}} + \frac{\pi}{2} - \psi\right)$$

$$i_b = I_M(t) \cos\left(\theta_{\mathrm{me}} + \frac{\pi}{2} - \psi - \frac{2}{3}\pi\right)$$

$$i_a = I_M(t) \cos\left(\theta_{\mathrm{me}} + \frac{\pi}{2} - \psi - \frac{4}{3}\pi\right)$$

$$T_m^m = \frac{3}{2}\, p\, \Lambda_{\mathrm{pm}}\, I_M(t) \cos(\psi)$$

This is the model in the *natural reference frame* that gives immediate interpretation to the physic of the system.

```
syms p_i i_q Lambda_mg P omega_m lambda v_w R T_machine B P_loss_joule
P_loss_viscous
```

```
eqn = T_machine == 3/2*p_i*i_q*Lambda_mg;
disp(eqn);
```

$$T_{\text{machine}} = \frac{3\,\Lambda_{\text{mg}}\,i_q\,p_i}{2}$$

Under stedy-state conditions, the value of the torque can also be defined as a function of the power generated by the turbine, which has already been calculated.

```
eqn2 = T_machine == P/omega_m;
disp(eqn2);
```

$$T_{\text{machine}} = \frac{P}{\omega_m}$$

$\omega_m = \omega_r$ is the mechanical rotational speed of the rotor; it's a function of the tip speed ratio λ and the wind speed v_w

```
eqn3 = omega_m == v_w*lambda/R;
disp(eqn3);
```

$$\omega_m = \frac{\lambda\,v_w}{R}$$

Given all these quantities, the viscous and joule losses can be calculated as follows

```
eqn4 = P_loss_joule == 3/2 * i_q^2*R;
eqn5 = P_loss_viscous == B*omega_m^2;

disp(eqn5);
```

$$P_{\text{loss,viscous}} = B\,{\omega_m}^2$$

```
disp(eqn4);
```

$$P_{\text{loss,joule}} = \frac{3\,R\,{i_q}^2}{2}$$

STALL REGULATED turbine

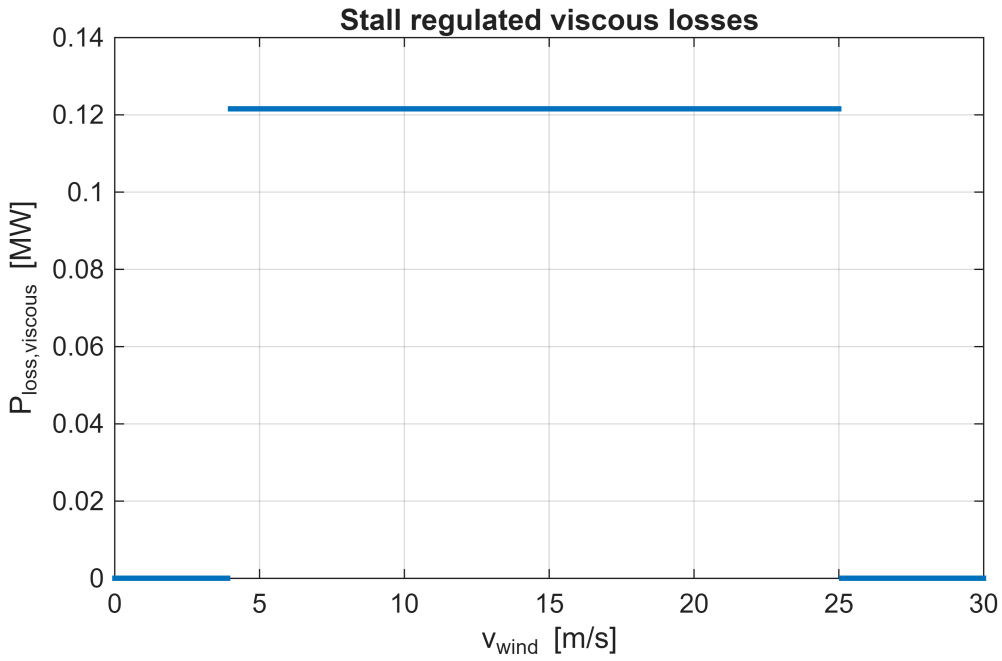
For the given set of wind speeds, ω_m is constant for stall regulation

Viscous loss stall regulated:

$$P_{\text{viscous}} = B_r\,\omega_c^2$$

```
viscous_loss_sr = B_r * Omega_c^2 / 1e6; % [MW]
% plot(wind_speed_sweep, viscous_loss_sr, '.');
P_viscous_sr = viscous_loss_sr*ones(length(wind_speed_sweep),1);
P_viscous_sr(find(wind_speed_sweep < V_ci)) = 0;
```

```
P_viscous_sr(find(wind_speed_sweep > V_co)) = 0;

figure();
plot(wind_speed_sweep, P_viscous_sr,'.','Color','#0072BD')
xlabel('v_{wind} [m/s]')
ylabel('P_{loss,viscous} [MW]')
title('Stall regulated viscous losses')
grid on
```




```
disp(['Viscous loss (Stall Regulated): ' num2str(viscous_loss_sr), ' MW']);
```

```
Viscous loss (Stall Regulated): 0.12158 MW
```

Joule loss stall regulated:

$$T_w = \frac{P}{\omega}$$

- ω constant rotor speed choosen previously
- P changing power over changing wind speed and constant rotor speed

The torque that reaches the generator is the one applied on the rotor minus the viscous damping.

$$T_g = T_w - B_r\,\omega$$

We can extract the current from the PMSG model:

$$i_q = \frac{2\,T_g}{3\,p\,\Lambda_{\mathrm{mg}}}$$

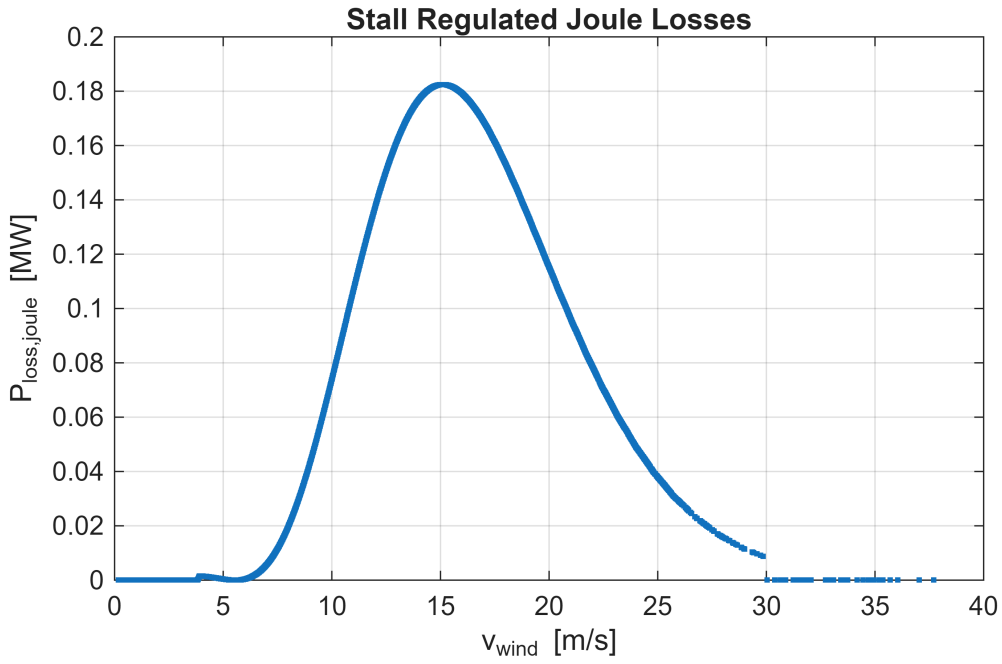
In a balanced 3-phase system the **RMS phase current** is related to the peak curent by:

$$i_{\text{RMS}} = \frac{i_q}{\sqrt{2}}$$

The **Joule losses** (copper losses) in the **three-phase stator windings** of the PMSG are given by:

$$P_{\text{joule}} = 3\, R_s\, i_{\text{RMS}}^2 = \frac{3}{2} R_s i_q^2$$

```
Pc_ws = interp1(wind_speed_sweep,Pc, wind_speed, 'linear');
Pc_ws(isnan(Pc_ws)) = 0;
T_m_sr = Pc_ws ./ Omega_c - B_r * Omega_c; % [Nm]

i_q_sr = 2/3 .* T_m_sr ./ (p*m_flux); % [A]

Joule_loss_sr_10m = 3/2 * R_s * i_q_sr.^2 /1e6 ; % [MW]
Joule_loss_sr = sum(Joule_loss_sr_10m) * (10*60) / 3600 / T_time_y; % [MWh/year] ->
è l'integrale sulla potenza specifica su ogni intervallo di tempo
plot(wind_speed, Joule_loss_sr_10m, '.');
% wind_sweep vs Pc_ws
xlabel('v_{wind} [m/s]');
ylabel('P_{loss,joule} [MW]');
title('Stall Regulated Joule Losses');
grid on
```



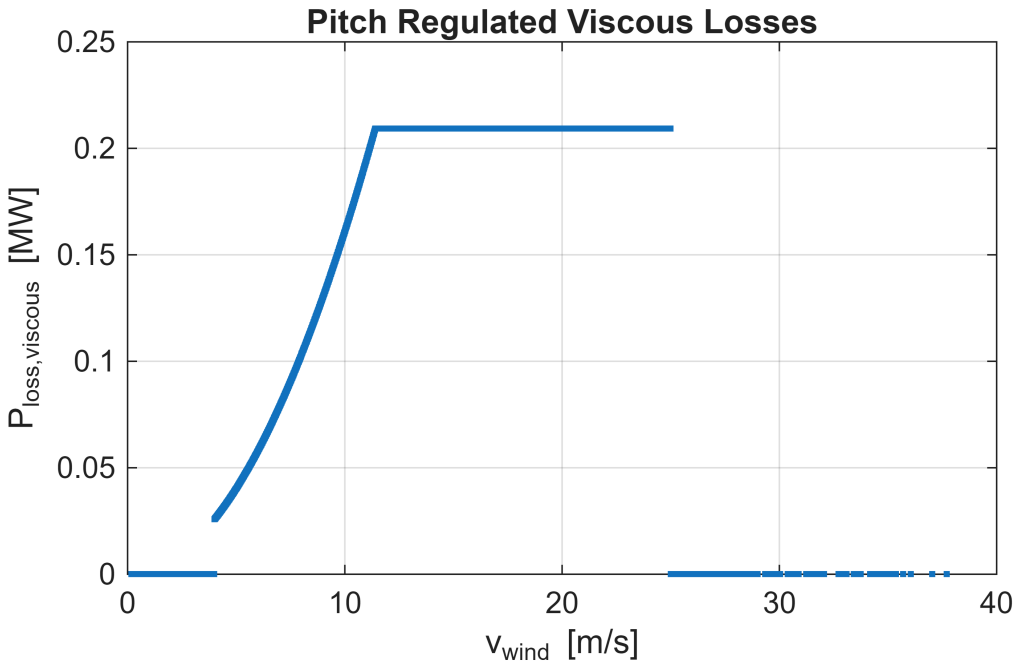

```
disp(['Joule loss (Stall Regulated): ', num2str(Joule_loss_sr),' MWh/year'])
```

```
Joule loss (Stall Regulated): 643.3941 MWh/year
```

PITCH REGULATED turbine

The mechanical speed ω_m will be changing in respect to the wind speed acording to saturation regions:

```
omega = 0*wind_speed;
for i = 1:length(wind_speed)
    if wind_speed(i)>=V_ci && wind_speed(i)<=V_rated
        omega(i) = wind_speed(i)*lambda_opt/(R_rotor);
    elseif wind_speed(i) > V_rated && wind_speed(i) <= V_co
        omega(i) = V_rated*lambda_opt/(R_rotor);
    end
end
viscous_loss_pr_10m = B_r * omega.^2 / 1e6; % [MW]
viscous_loss_pr = sum(viscous_loss_pr_10m) *(10*60) / 3600 / T_time_y; % [MWh/year]
plot(wind_speed, viscous_loss_pr_10m, '.');
xlabel('v_{wind} [m/s]');
ylabel('P_{loss,viscous} [MW]');
title('Pitch Regulated Viscous Losses');
grid on
```




```
disp(['Viscous loss (Pitch Regulated): ', num2str(viscous_loss_pr), ' MW']);
```

```
Viscous loss (Pitch Regulated): 1138.9626 MW
```

The joule losses are the following:

```
% Define Maxinum Power for each region
I1_pr = find(wind_speed<V_ci);
I2_pr = find(wind_speed>=V_ci & wind_speed<=V_rated);
I3_pr = find(wind_speed>V_rated & wind_speed<=V_co);
I4_pr = find(wind_speed>V_co);
P_pr = 0*wind_speed;
```

```
P_pr(I2_pr) = 0.5*rho_air*wind_speed(I2_pr).^3*A_rotor*Cp_opt;
P_pr(I3_pr) = Prated; %*ones(size(I3));

% T_gen_pr = P_pr./omega; % generator torque calculation for the pitch regulated
turbine
% plot(wind_speed,T_gen_pr,'.','LineWidth',0.1)
% title('Torque vs Wind Speed for the pitch regulated turbine')
% xlabel('v_{wind} [m/s]')
% ylabel('T_{w} [Nm]')
% xlim([0 30])

% extract true Torque from Pc power curve for stall regulated P = T*omega->
% T = P/omega; omega constant but P changes
```

$$T_w = \frac{P}{\omega}$$

- ω changing (optimal) rotor speed
- P changing power over changing wind speed and changing rotor speed

The torque that reaches the generator is the one applied on the rotor minus the viscous damping.

$$T_g = T_w - B_r\,\omega$$

We can extract the current from the PMSG model:

$$i_q = \frac{2\,T_g}{3\,p\,\Lambda_{\mathrm{mg}}}$$

In a balanced 3-phase system the **RMS phase current** is related to the peak curent by:

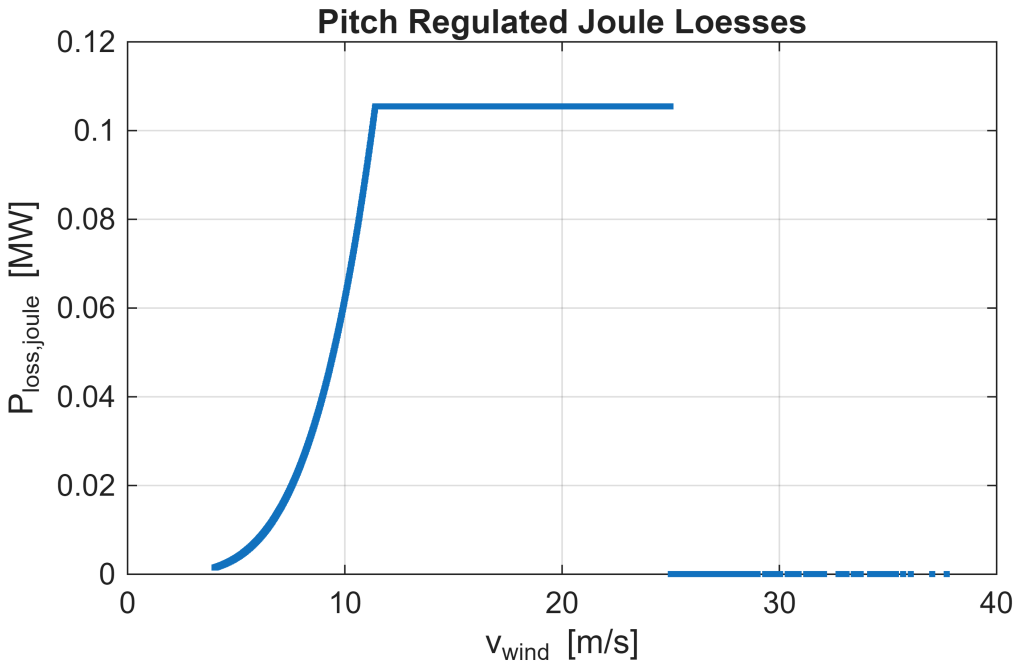
$$i_{\mathrm{RMS}} = \frac{i_q}{\sqrt{2}}$$

The **Joule losses** (copper losses) in the **three-phase stator windings** of the PMSG are given by:

$$P_{\mathrm{joule}} = 3\,R_s\,i_{\mathrm{RMS}}^2 = \frac{3}{2}R_s i_q^2$$

```
T_m_pr = P_pr ./ omega - B_r * omega; % [Nm]
% T_m_sr = Trated - B_r * Omega_c;  % [Nm]
i_q_pr = 2/3 .* T_m_pr ./ (p*m_flux); % [A]

Joule_loss_pr_10m = 3/2 * R_s * i_q_pr.^2 / 1e6; % [MW]
Joule_loss_pr_10m(find(wind_speed > V_co)) = 0;

Joule_loss_pr = sum(Joule_loss_pr_10m) *(10*60) / 3600 / T_time_y; % [MWh/year] ->
è l'integrale sulla potenza specifica su ogni intervallo di tempo

figure();
plot(wind_speed, Joule_loss_pr_10m, '.')
```

```
title('Pitch Regulated Joule Loesses');
xlabel('v_{wind} [m/s]');
ylabel('P_{loss,joule} [MW]');
grid on
```



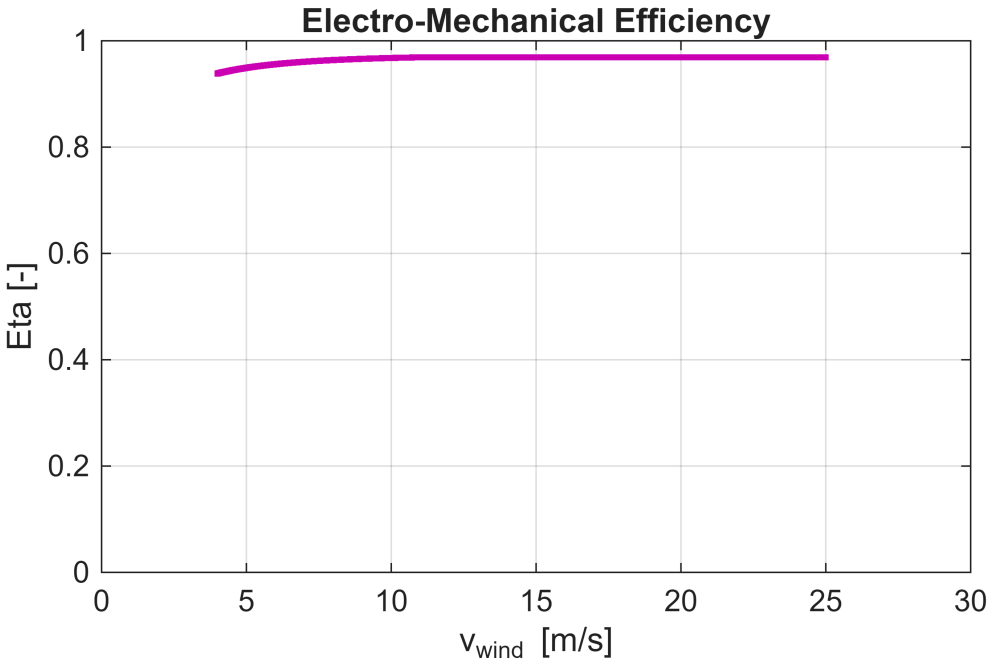

```
disp(['Joule loss (Pitch Regulated): ', num2str(Joule_loss_pr),' MWh/year'])
```

```
Joule loss (Pitch Regulated): NaN MWh/year
```

c2) Turbine average electromechanical efficiency over a year, for the case of pitch regulated variable speed turbine

$$\eta(v_w) = \frac{P_{\text{nominal}} - P_{\text{loss,viscous}} - P_{\text{loss,joule}}}{P_{\text{nominal}}}$$

```
% eta = (P_pr - viscous_loss_pr_10m - Joule_loss_pr_10m) ./ (Pw2);
eta = (P_pr/1e6 - viscous_loss_pr_10m - Joule_loss_pr_10m) ./ (P_pr/1e6);

plot(wind_speed, eta, '.', 'Color',[0.8 0 0.7]);
xlabel('v_{wind} [m/s]')
ylabel('Eta [-]')
title('Electro-Mechanical Efficiency')
ylim([0 1])
xlim([0 30])
grid on
```




```
% MPPT_indexes = find(wind_speed > V_ci & wind_speed < V_rated);
eta_mean = mean(eta(~isnan(eta) & isfinite(eta)));
disp(['Mean efficiency: ',num2str(eta_mean)]);
```

```
Mean efficiency: 0.96428
```

Dynamic Analysis

Constant input 1:

```
% Simulation time setup
T_sim = 600;            % sim time [s]
dT = 0.1;               % fixed size time interval [s]
time = (0:dT:T_sim)';      % time column vector [s]
N_samples = size(time);

% define a value
V_wind_const = 0.8*V_rated;
% define column vector
Vw_constant = ones(N_samples) * V_wind_const;
Vw_timeseries = timeseries( Vw_constant,time);
omega_0 = lambda_opt*V_wind_const / R_rotor; % omega star

% figure();
% plot(time, Vw_constant)
% xlabel('Time [s]')
% ylabel('Vw constant [m/s]')
% title('Vw constant 1')
```

Constant input 2:

```
% Simulation time setup
T_sim = 3*60;            % sim time [24h * 60 m/h] [m] -> [s]
dT = 10;               % fixed size time interval [m] -> [s]
time = (0:dT:T_sim)';      % time column vector
N_samples = size(time);

% define a value
V_wind_const = 0.8*V_rated;
% define column vector
Vw_constant = ones(N_samples) * V_wind_const;
Vw_timeseries = timeseries(time, Vw_constant);
omega_0 = lambda_opt*1.2*V_ci / R_rotor; % IC

% figure();
% plot(time , Vw_constant, '.')
% xlabel('Time [s]')
% ylabel('Vw constant [m/s]')
% title('Vw constant 2')
```

Ramp input 3:

```
% Simulation time setup
T_sim = 3*60;            % sim time [24h * 60 m/h] [m] -> [s]
dT = 10;               % fixed size time interval [m] -> [s]
time = (0:dT:T_sim)';      % time column vector
N_samples = size(time);

V_end = V_rated-0.5;
% V_end = V_co - 3; % final wind speed
Vw_ramp = linspace(1.1*V_ci,V_end,length(time))';

Vw_timeseries = timeseries(Vw_ramp, time);

omega_0 = Vw_ramp(1)*lambda_opt/R_rotor; % IC

% figure();
% plot(time, Vw_ramp, '.')
% xlabel('Time [s]')
% ylabel('Vw ramp [m/s]')
% title('Vw ramp')
```

Ramp input 4:

```
% Simulation time setup
T_sim = 10*60;            % sim time [24h * 60 m/h] [m] -> [s]
dT = 10;               % fixed size time interval [m] -> [s]
time = (0:dT:T_sim)';      % time column vector
N_samples = size(time);
```

```
V_end = 18;
% V_end = V_co - 3; % final wind speed
Vw_ramp = linspace(1.1*V_ci,V_end,length(time))';

Vw_timeseries = timeseries(Vw_ramp, time);

omega_0 = Vw_ramp(1)*lambda_opt/R_rotor; % IC

% figure();
% plot(time, Vw_ramp, '.')
% xlabel('Time [s]')
% ylabel('Vw ramp [m/s]')
% title('Vw ramp')
```

Kaimal distribution - short term statistics

Test the system response in the presence of multichromatic pressure signals

Compute fewsignificant wind speeds:

```
VW_mean = mean(wind_speed)
```

```
VW_mean =
9.8905
```

```
V10a = 6 % region II  MPPT
```

```
V10a =
6
```

```
V10b = 10 % region III MPPT
```

```
V10b =
10
```

```
V10c = 17 % region III rotor speed saturation
```

```
V10c =
17
```

Compute the kaimal PSD:

```
% syms I_ V10_ l_ f_
% S_kaimal = (I_^2 * V10_ * l_) / ((1 + 1.5 * f_*l_/V10_)^(5/3)) % [m^2/s]
```

$$S(f) = \frac{I^2 V_{10}\, l}{\left(1 + 1.5\frac{f\, l}{V_{10}}\right)^{\frac{5}{3}}}$$

- $l = \begin{cases} 20\, h & h < 30\, m \\ 600\, m & h \geq 30\, m \end{cases}$

- $I = \frac{\sigma}{V_{10}}$ turbolence intensity
- σ wind speed std.dev.
- $f$frequency

Compute the constant parameters:

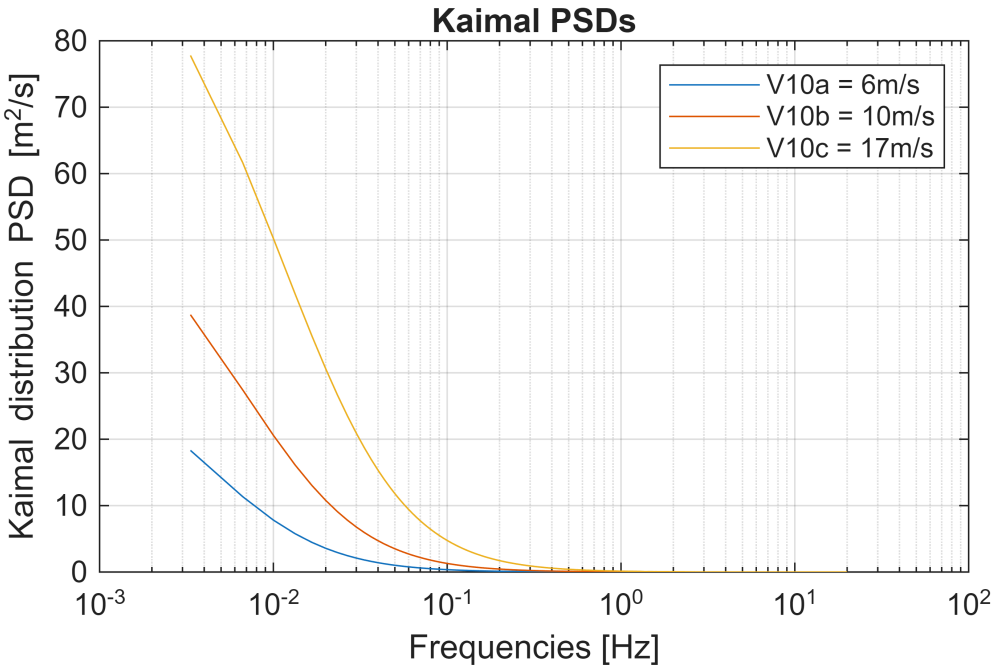
```
I = 0.1; % turbolence intensity; I = sigma/V10;
l = 600; % [m]
```

Discretise the frequency range, with frequencies and a step Δ.

```
fs = 20; % [Hz] sampling frequency
T_sim = 60*5; % [s] time interval
% T_sim = 60;
N = T_sim*fs; % number of points to compute fft on
deltaf = 1/T_sim; % frequency increment;
```

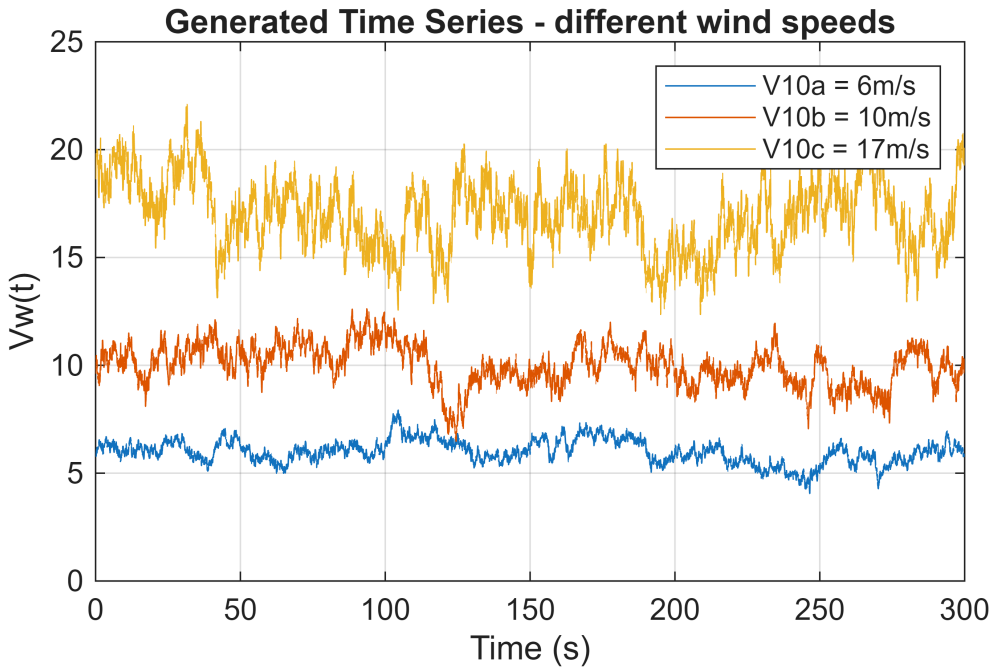
Buid the PSD:

```
frequencies = deltaf:deltaf:(fs); % frequencies from deltaf to fs/2 (Nyquist)

S_kaimal_f_a = ((I^2 * V10a * l)./(1 + (1.5 * frequencies* l)/V10a).^(5/3))';
S_kaimal_f_b = ((I^2 * V10b * l)./(1 + (1.5 * frequencies* l)/V10b).^(5/3))';
S_kaimal_f_c = ((I^2 * V10c * l)./(1 + (1.5 * frequencies* l)/V10c).^(5/3))';

figure();
semilogx(frequencies, S_kaimal_f_a);
hold on;
semilogx(frequencies, S_kaimal_f_b);
semilogx(frequencies, S_kaimal_f_c);
title('Kaimal PSDs')
xlabel('Frequencies [Hz]')
ylabel('Kaimal distribution PSD [m^2/s]')
legend(['V10a = ',num2str(V10a),'m/s'],['V10b = ',num2str(V10b),'m/s'],['V10c =
',num2str(V10c),'m/s'])
grid on;
hold off;
```




```
num_frequencies = length(frequencies);
% randomly generated phases
phases_a = 0 + (2*pi - 0) * rand(1, num_frequencies)'; % random phases [0,2*pi]
phases_b = 0 + (2*pi - 0) * rand(1, num_frequencies)';
phases_c = 0 + (2*pi - 0) * rand(1, num_frequencies)';


% A superimposition of armonics
amplitudes_a = sqrt(2*deltaf*(S_kaimal_f_a)); % amplitude for i-th component of the
spectrum
amplitudes_b = sqrt(2*deltaf*(S_kaimal_f_b)); % amplitude for i-th component of the
spectrum
amplitudes_c = sqrt(2*deltaf*(S_kaimal_f_c)); % amplitude for i-th component of the
spectrum

t = (0:N-1)'/fs;     % time vector

p_t_a = ones(N, 1)*V10a;  % initialize p(t) vector
p_t_b = ones(N, 1)*V10b;  % initialize p(t) vector
p_t_c = ones(N, 1)*V10c;  % initialize p(t) vector

% superimposition of armonics
for i = 1:N
    p_t_a = p_t_a + amplitudes_a(i)*sin(2*pi*frequencies(i)*t + phases_a(i));
    p_t_b = p_t_b + amplitudes_b(i)*sin(2*pi*frequencies(i)*t + phases_b(i));
    p_t_c = p_t_c + amplitudes_c(i)*sin(2*pi*frequencies(i)*t + phases_c(i));
end

figure();
```

```
plot(t, p_t_a);
hold on;
plot(t, p_t_b);
plot(t, p_t_c);
xlabel('Time (s)');
ylabel('Vw(t)');
ylim([0 25])
title('Generated Time Series - different wind speeds');
legend(['V10a = ',num2str(V10a),'m/s'],['V10b = ',num2str(V10b),'m/s'],['V10c =
',num2str(V10c),'m/s'])
grid on;
```



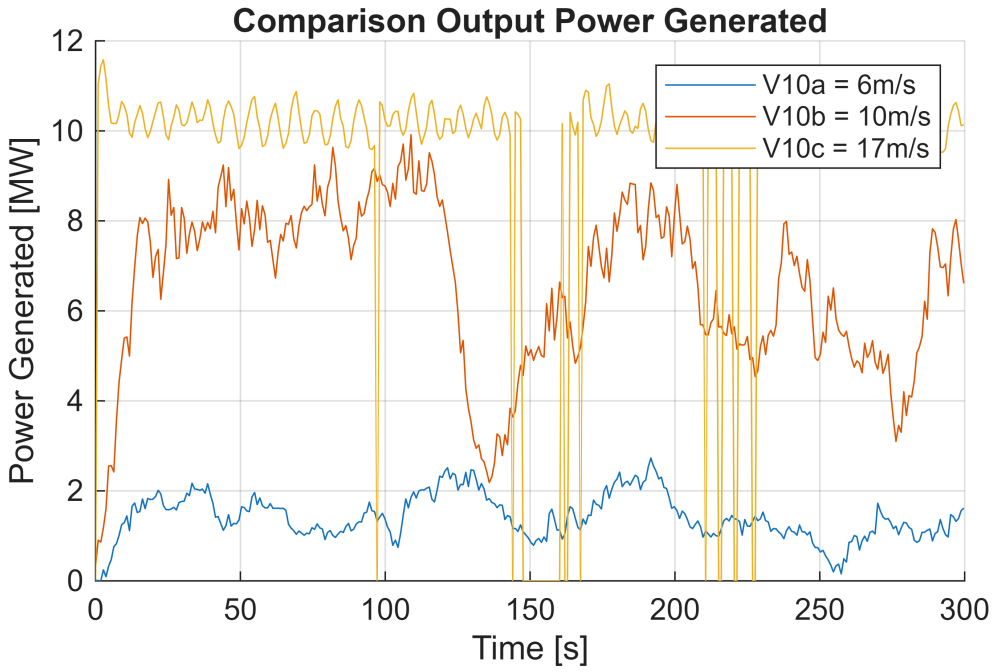

```
Vw_timeseries_a = timeseries(p_t_a, t);
Vw_timeseries_b = timeseries(p_t_b, t);
Vw_timeseries_c = timeseries(p_t_c, t);

omega_0_a = lambda_opt*V10a/R_rotor;
omega_0_b = lambda_opt*V10b/R_rotor;
omega_0_c = lambda_opt*V_rated/R_rotor;

pidParams_a.P = 11990384.5947164;
pidParams_a.I = 1377156.84973809;
pidParams_a.D = -7577964.66256918;
pidParams_a.N = 0.217260825582008;

pidParams_b.P = 11990384.5947164;
pidParams_b.I = 1377156.84973809;
pidParams_b.D = -7577964.66256918;
pidParams_b.N = 0.217260825582008;
```

```
pidParams_c.P = 0;
pidParams_c.I = 109663244653.931;
pidParams_c.D = 0;
pidParams_c.N = 100;
```

Automatic simulink simulations

```
% Store the time series in a cell array for easy iteration

inputData = {Vw_timeseries_a, Vw_timeseries_b, Vw_timeseries_c};
inputData_1 = {omega_0_a, omega_0_b, omega_0_c};
inputData_2 = {pidParams_a, pidParams_b, pidParams_c};
inputData_3 = {0.9, 0.9, 0.9}; % fixed-step size [s]

% Initialize a cell array to store output data
PowerGenerated = cell(1,3);

% modelName = 'Pitch_regulated_WT_sim_dynamic_3_2'; % Replace with the actual
Simulink model name
modelName = 'PR_WT_sim_4_1';

% Run Simulink model for each time series
for i = 1:length(inputData)
    % Assign the current time series to the Simulink input
    assignin('base', 'inputSignal', inputData{i}); % inputSignal should match the
variable used in Simulink
    assignin('base', 'omega_0',     inputData_1{i});
    assignin('base', 'pidParams',   inputData_2{i});
    assignin('base', 'step_size',   inputData_3{i});
    % Run the simulation
    simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');

    % Extract output data from the simulation
    PowerGenerated{i} = simOut.get('PowerGenerated');
    Pitch{i} = simOut.get('Pitch');

    % Save output with a unique name in the workspace
    assignin('base', ['PowerGenerated_' num2str(i)], PowerGenerated{i});
    assignin('base', ['Pitch_' num2str(i)], Pitch{i});
end

% Plot the three outputs
figure;
hold on;
plot(PowerGenerated_1.Time, PowerGenerated_1.Data/1e6, 'DisplayName', 'V10a');
plot(PowerGenerated_2.Time, PowerGenerated_2.Data/1e6, 'DisplayName', 'V10b');
plot(PowerGenerated_3.Time, PowerGenerated_3.Data/1e6, 'DisplayName', 'V10c');
xlabel('Time [s]');
ylabel('Power Generated [MW]');
title('Comparison Output Power Generated');
```

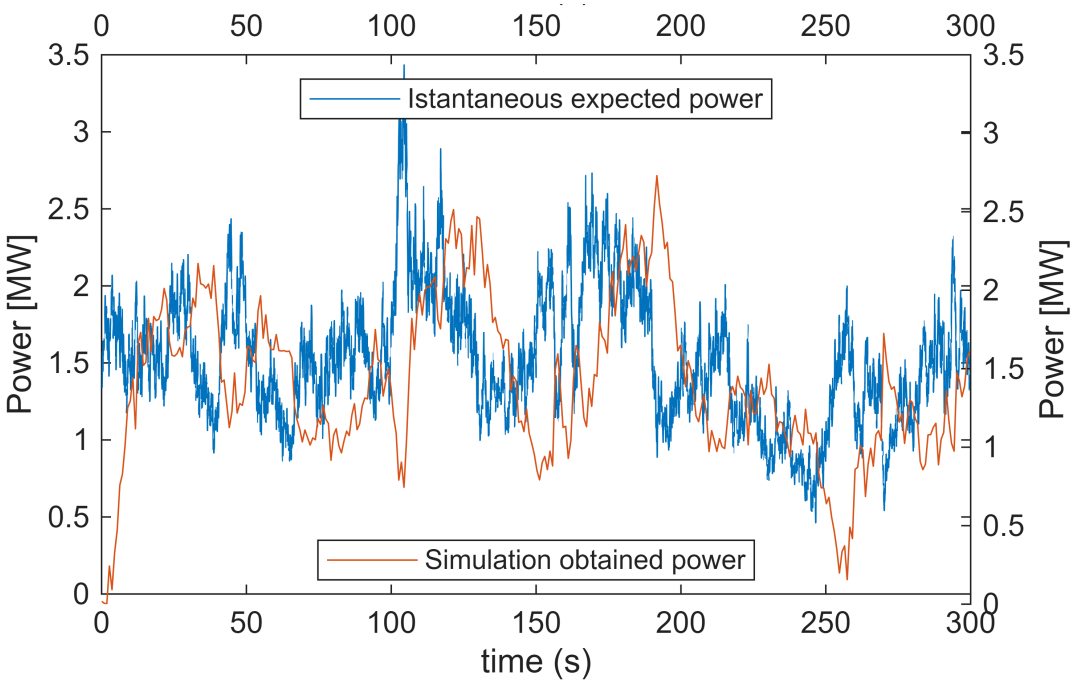
```
legend(['V10a = ',num2str(V10a),'m/s'],['V10b = ',num2str(V10b),'m/s'],['V10c =
',num2str(V10c),'m/s'])
grid on;
```



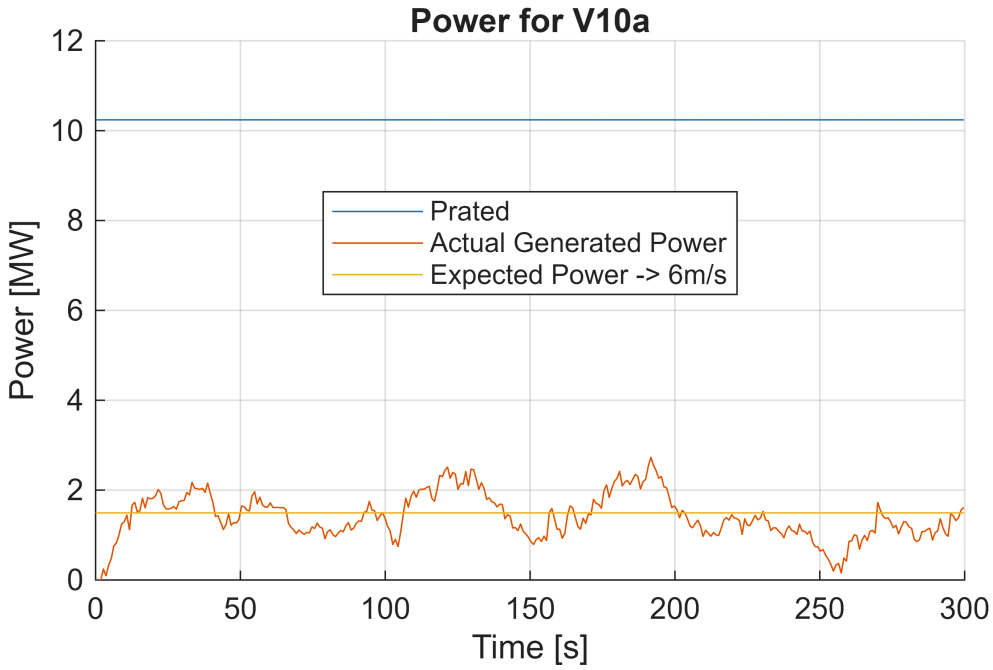
Simulink output

```
% Prated horizontal line --> Prated
Prated_vecta = ones(size(PowerGenerated_1.Time))*Prated;
% Expected Power Generated A
P_V10a = 1/2*rho_air*V10a.^3*A_rotor * Cp_opt;
P_V10a_vect = ones(size(PowerGenerated_1.Time))*P_V10a;

% Prated horizontal line --> Prated
Prated_vectb = ones(size(PowerGenerated_2.Time))*Prated;
% Expected Power Generated B
P_V10b = 1/2*rho_air*V10b.^3*A_rotor * Cp_opt;
P_V10b_vect = ones(size(PowerGenerated_2.Time))*P_V10b;

% Prated horizontal line --> Prated
Prated_vectc = ones(size(PowerGenerated_3.Time))*Prated;

% Expected Power Generated C
P_V10c = 1/2*rho_air*V10c.^3*A_rotor * Cp_opt;
P_V10c_vect = ones(size(PowerGenerated_3.Time))*P_V10c;

% Compute mean power
mean_power_V10a = mean(PowerGenerated_1.Data)/1e6; % [MW]
mean_power_V10b = mean(PowerGenerated_2.Data)/1e6; % [MW]
mean_power_V10c = mean(PowerGenerated_3.Data)/1e6; % [MW]

%
```

```
% Vw_timeseries_a
n_sample_a = length(Vw_timeseries_a.Data);
P_3_a = zeros(n_sample_a,1);
for i = 1:n_sample_a
    if Vw_timeseries_a.Data(i) >= V_ci && Vw_timeseries_a.Data(i) <= V_rated
        P_3_a(i) = 0.5 * rho_air * (Vw_timeseries_a.Data(i).^3) * A_rotor * Cp_opt;
    elseif Vw_timeseries_a.Data(i) >= V_rated && Vw_timeseries_a.Data(i) <= V_co
        P_3_a(i) = Prated;
    end
end
figure();
ax1 = axes; % First axes
plot(Vw_timeseries_a.Time, P_3_a/1e6,'Color','#0072BD');
xlabel('time (s)');
ylabel('Power [MW]');
legend('Istantaneous expected power','Location','north')
hold on;
ax2 = axes; % Second axes
plot(PowerGenerated_1.Time, PowerGenerated_1.Data/1e6,'Color','#D95319')
xlabel('time (s)');
ylabel('Power [MW]');
legend('Simulation obtained power','Location','south')

% Positioning the second axes on top
ax2.Position = ax1.Position;
ax2.Color = 'none'; % Make background transparent
ax2.YAxisLocation = 'right'; % Move y-axis to the right
ax2.XAxisLocation = 'top';   % Move x-axis to the top
ax2.Box = 'off'; % Remove extra box lines
linkaxes([ax1, ax2], 'y');
hold off
```




```
figure();
hold on;
plot(PowerGenerated_1.Time, Prated_vecta/1e6)
plot(PowerGenerated_1.Time, PowerGenerated_1.Data/1e6)
plot(PowerGenerated_1.Time, P_V10a_vect/1e6)
grid on;
title('Power for V10a');
xlabel('Time [s]')
ylabel('Power [MW]')
legend('Prated','Actual Generated Power',['Expected Power -> ', num2str(V10a),'m/
s'],'Location','best')
hold off;
```



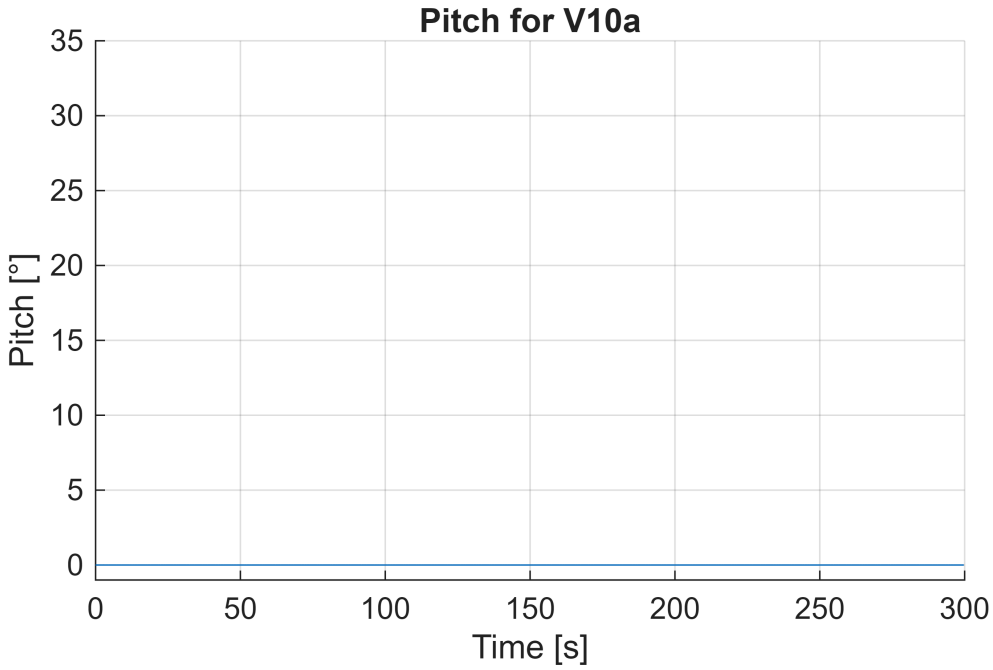

```
disp(['Expected power: ',num2str(P_V10a / 1e6),' MW'])
```

```
Expected power: 1.4931 MW
```

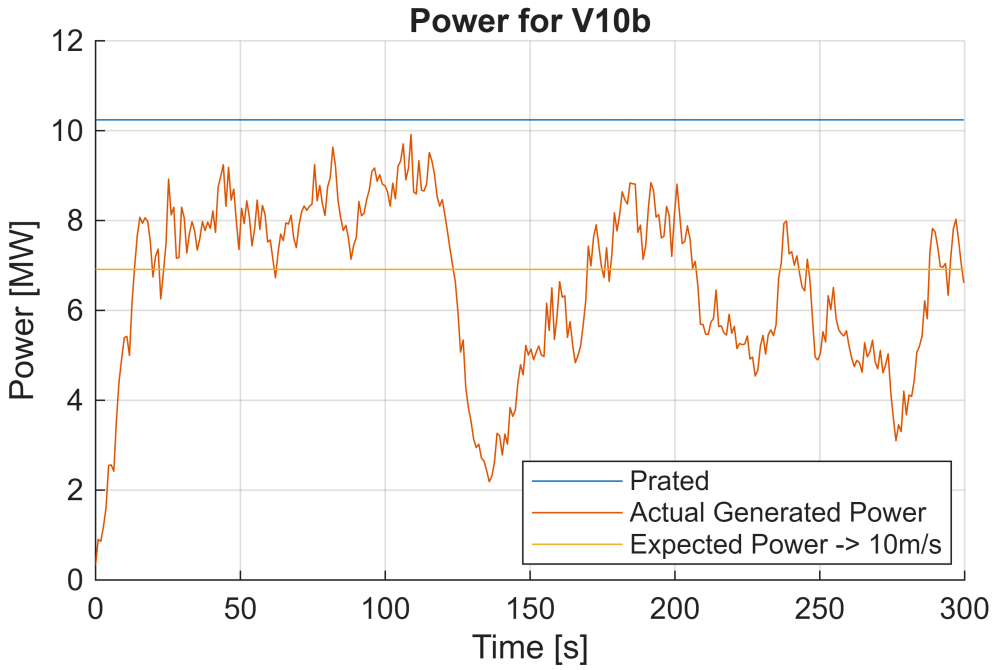
```
disp(['Mean power:     ',num2str(mean_power_V10a),' MW'])
```

```
Mean power:     1.4134 MW
```

```
figure();
hold on;
plot(Pitch_1.Time, Pitch_1.Data);
grid on;
title('Pitch for V10a');
xlabel('Time [s]')
ylabel('Pitch [°]')
ylim([-1 35])
hold off;
```




```
figure();
hold on;
plot(PowerGenerated_2.Time, Prated_vectb/1e6)
plot(PowerGenerated_2.Time, PowerGenerated_2.Data/1e6)
plot(PowerGenerated_2.Time, P_V10b_vect/1e6)
grid on;
title('Power for V10b');
xlabel('Time [s]')
ylabel('Power [MW]')
legend('Prated','Actual Generated Power',['Expected Power -> ', num2str(V10b),'m/
s'],'Location','best')
hold off;
```




```
disp(['Expected power: ',num2str(P_V10b),' MW'])
```

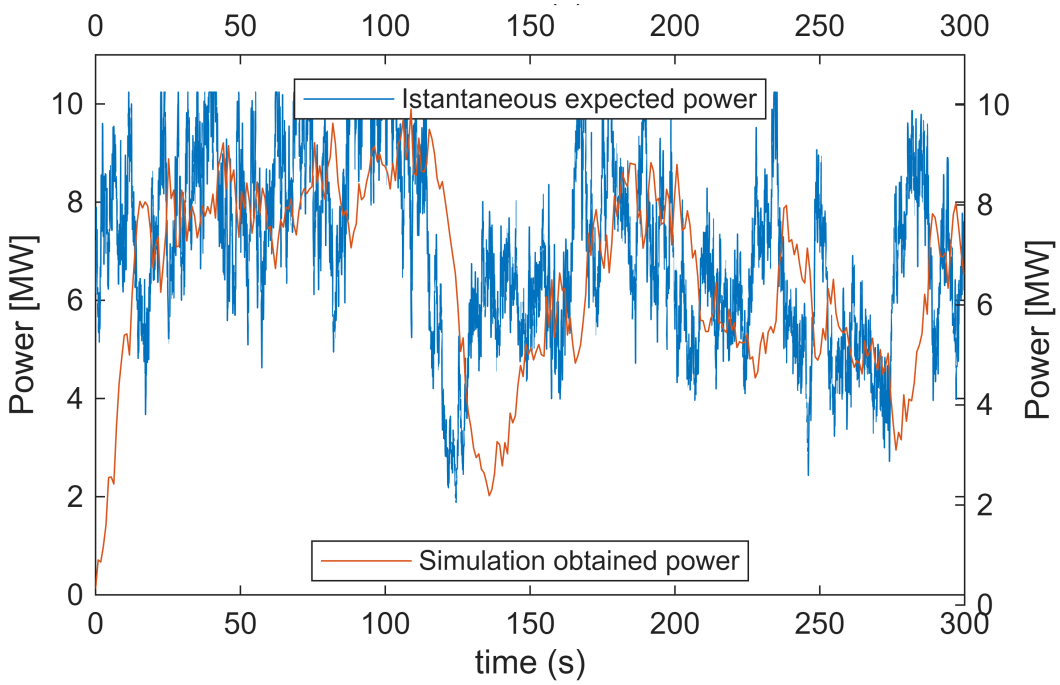
```
Expected power: 6912635.8471 MW
```

```
disp(['Mean power: ',num2str(mean_power_V10b),' MW'])
```

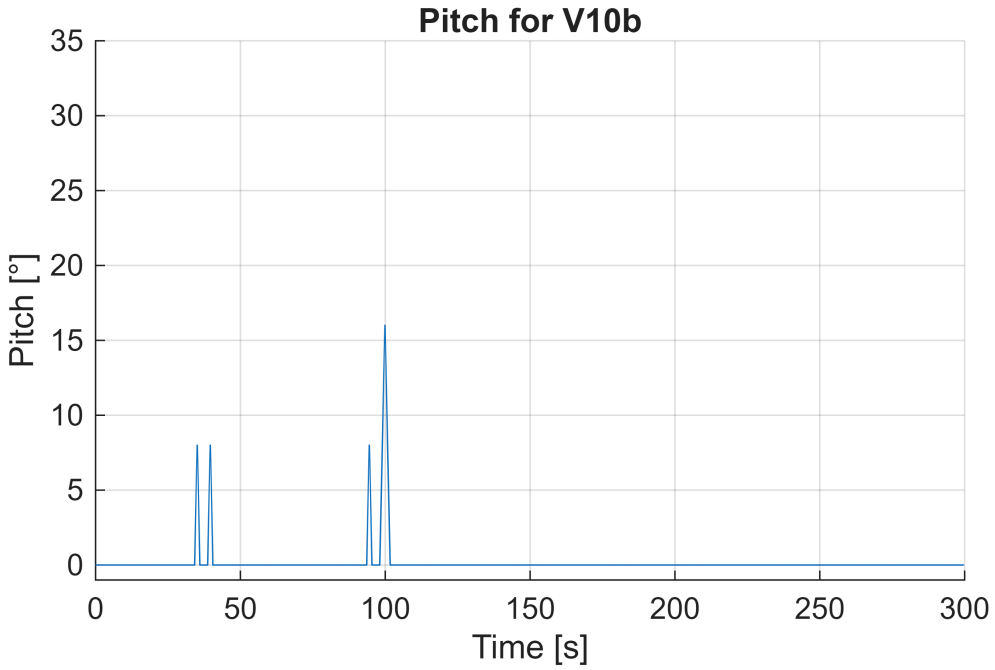
```
Mean power: 6.5896 MW
```

```
% Vw_timeseries_b
n_sample_b = length(Vw_timeseries_b.Data);
P_3_b = zeros(n_sample_b,1);
for i = 1:n_sample_b
    if Vw_timeseries_b.Data(i) >= V_ci && Vw_timeseries_b.Data(i) <= V_rated
        P_3_b(i) = 0.5 * rho_air * (Vw_timeseries_b.Data(i).^3) * A_rotor * Cp_opt;
    elseif Vw_timeseries_b.Data(i) >= V_rated && Vw_timeseries_b.Data(i) <= V_co
        P_3_b(i) = Prated;
    end
end
figure();
ax1 = axes; % First axes
plot(Vw_timeseries_b.Time, P_3_b/1e6,'Color','#0072BD');
xlabel('time (s)');
ylabel('Power [MW]');
legend('Istantaneous expected power','Location','north')
hold on;
ax2 = axes; % Second axes
plot(PowerGenerated_2.Time, PowerGenerated_2.Data/1e6,'Color','#D95319')
xlabel('time (s)');
ylabel('Power [MW]');
```

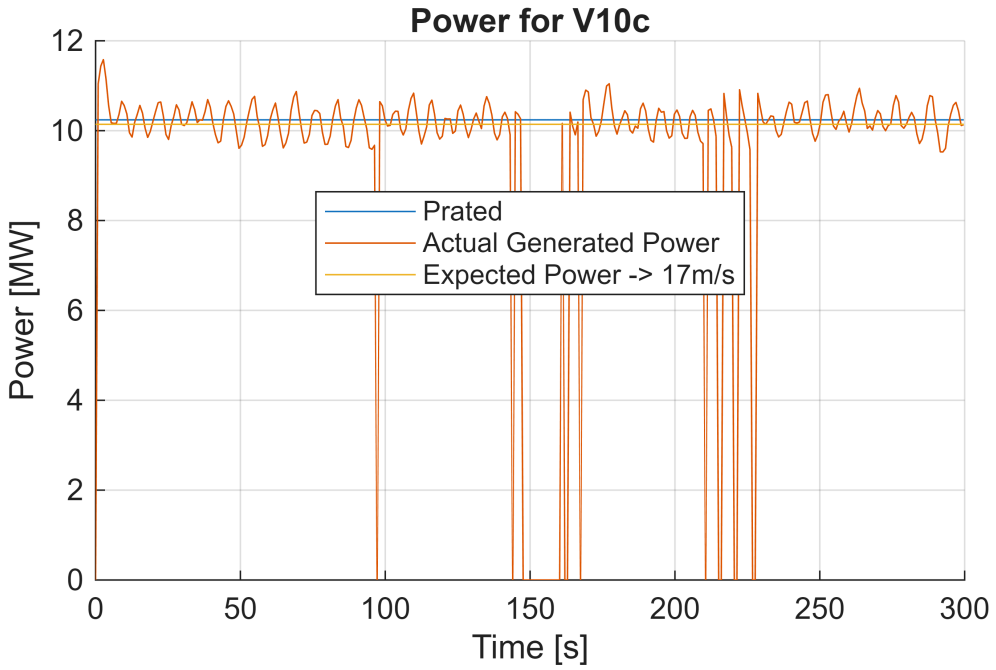
```
legend('Simulation obtained power','Location','south')

% Positioning the second axes on top
ax2.Position = ax1.Position;
ax2.Color = 'none'; % Make background transparent
ax2.YAxisLocation = 'right'; % Move y-axis to the right
ax2.XAxisLocation = 'top';   % Move x-axis to the top
ax2.Box = 'off'; % Remove extra box lines
linkaxes([ax1, ax2], 'y');
hold off
```




```
figure();
hold on;
plot(Pitch_2.Time, Pitch_2.Data);
grid on;
title('Pitch for V10b');
xlabel('Time [s]')
ylabel('Pitch [°]')
ylim([-1 35])
hold off;
```




```
figure();
hold on;
plot(PowerGenerated_3.Time, Prated_vectc/1e6)
plot(PowerGenerated_3.Time, PowerGenerated_3.Data/1e6)
plot(PowerGenerated_3.Time, (Prated_vectc-0.1*1e6)/1e6)
grid on;
title('Power for V10c');
xlabel('Time [s]')
ylabel('Power [MW]')
legend('Prated','Actual Generated Power',['Expected Power -> ', num2str(V10c),'m/
s'],'Location','best')
hold off;
```

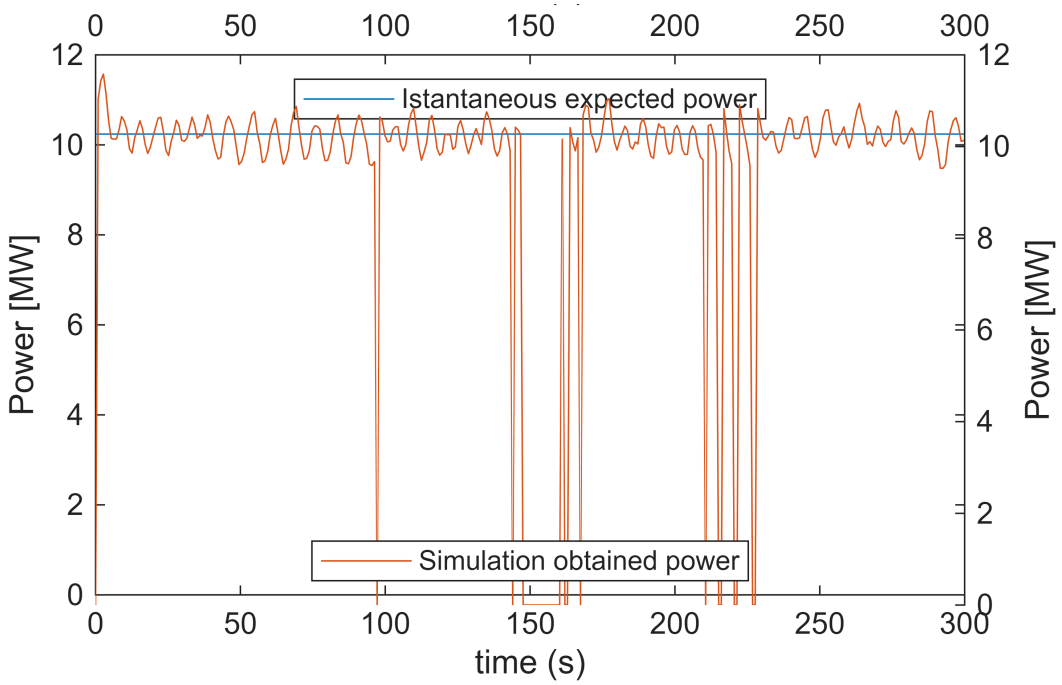



```
disp(['Mean power: ',num2str(mean_power_V10c),' MW'])
```

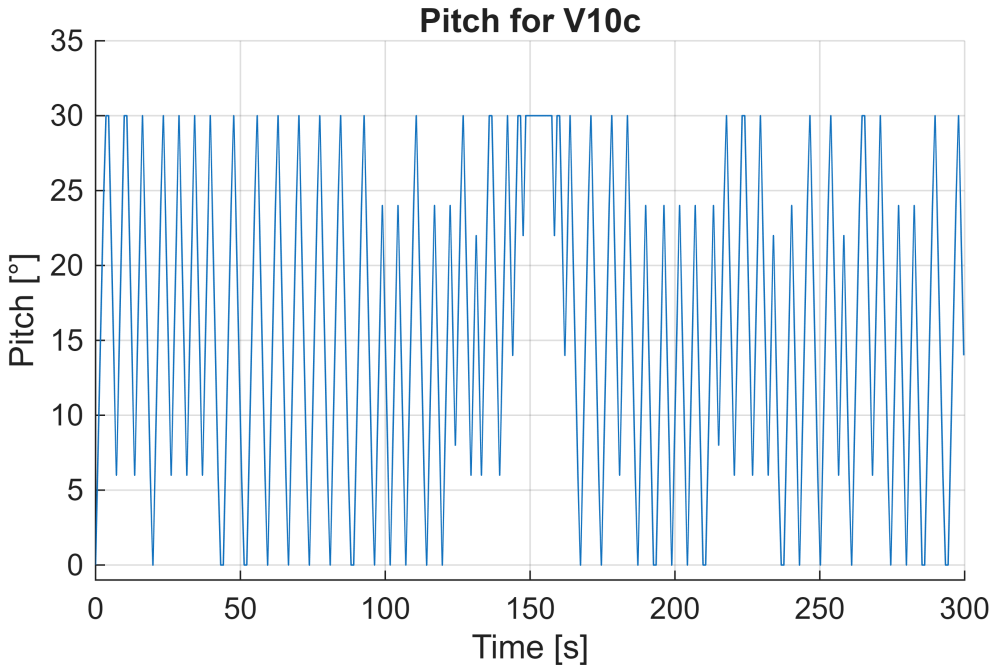
```
Mean power: 9.3861 MW
```

```
% Vw_timeseries_c
n_sample_c = length(Vw_timeseries_c.Data);
P_3_c = zeros(n_sample_c,1);
for i = 1:n_sample_c
    if Vw_timeseries_c.Data(i) >= V_ci && Vw_timeseries_c.Data(i) <= V_rated
        P_3_c(i) = 0.5 * rho_air * (Vw_timeseries_c.Data(i).^3) * A_rotor * Cp_opt;
    elseif Vw_timeseries_c.Data(i) >= V_rated && Vw_timeseries_c.Data(i) <= V_co
        P_3_c(i) = Prated;
    end
end
figure();
ax1 = axes; % First axes
plot(Vw_timeseries_c.Time, P_3_c/1e6,'Color','#0072BD');
xlabel('time (s)');
ylabel('Power [MW]');
legend('Istantaneous expected power','Location','north')
hold on;
ax2 = axes; % Second axes
plot(PowerGenerated_3.Time, PowerGenerated_3.Data/1e6,'Color','#D95319')
xlabel('time (s)');
ylabel('Power [MW]');
legend('Simulation obtained power','Location','south')

% Positioning the second axes on top
```

```
ax2.Position = ax1.Position;
ax2.Color = 'none'; % Make background transparent
ax2.YAxisLocation = 'right'; % Move y-axis to the right
ax2.XAxisLocation = 'top';   % Move x-axis to the top
ax2.Box = 'off'; % Remove extra box lines
linkaxes([ax1, ax2], 'y');
hold off
```

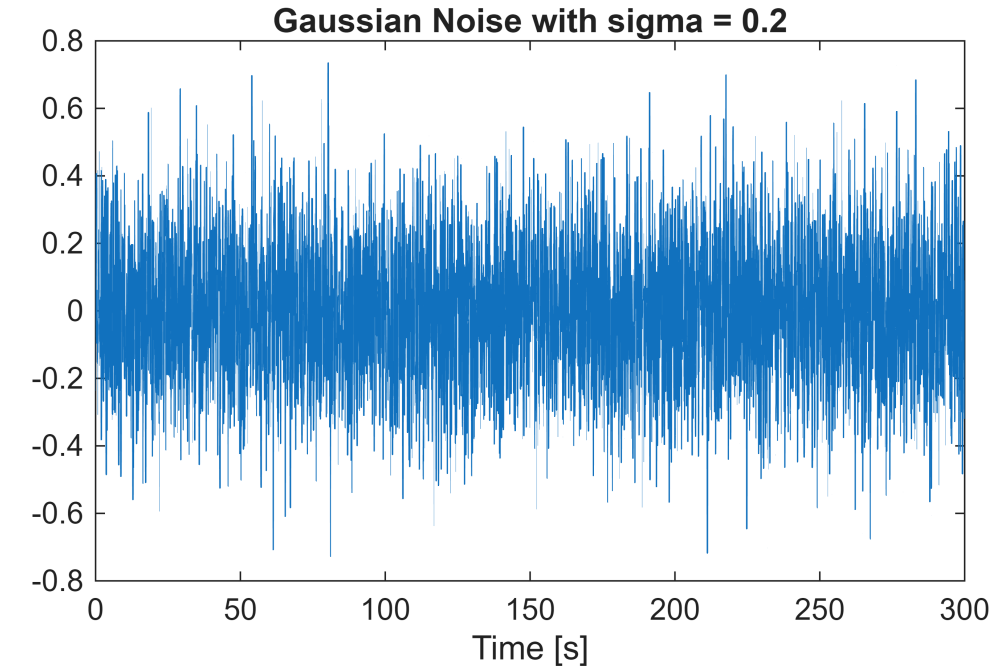



```
figure();
hold on;
plot(Pitch_3.Time, Pitch_3.Data);
grid on;
title('Pitch for V10c');
xlabel('Time [s]')
ylabel('Pitch [°]')
ylim([-1 35])
hold off;
```



Anemometer noise

```
% Vw_timeseries_a
sigma = 0.2;
anemometer_noise = sigma * randn(size(Vw_timeseries_a.Time));
figure();
plot(Vw_timeseries_a.Time, anemometer_noise)
xlabel('Time [s]')
title(['Gaussian Noise with sigma = ', num2str(sigma)])
```

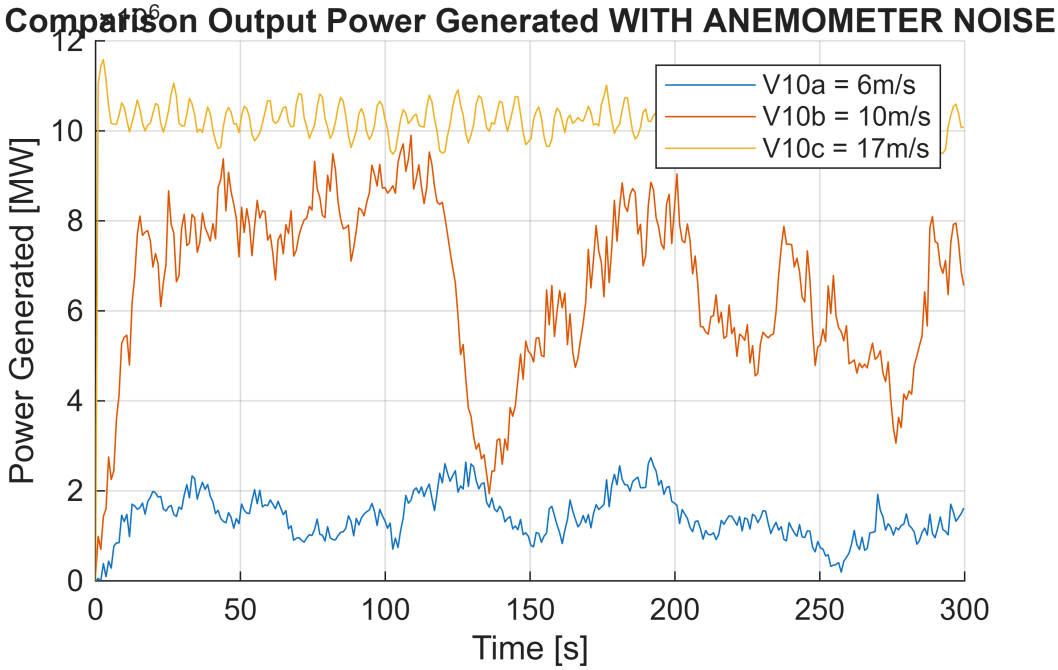


```
Vw_timeseries_a_noise = timeseries(Vw_timeseries_a.Data + anemometer_noise,
Vw_timeseries_a.Time);
Vw_timeseries_b_noise = timeseries(Vw_timeseries_b.Data + anemometer_noise,
Vw_timeseries_b.Time);
Vw_timeseries_c_noise = timeseries(Vw_timeseries_c.Data + anemometer_noise,
Vw_timeseries_c.Time);

% figure();
% hold on
% plot(Vw_timeseries_a_noise.Time, Vw_timeseries_a_noise.Data)
% plot(Vw_timeseries_a.Time, Vw_timeseries_a.Data)
% hold off
%
% figure();
% hold on
% plot(Vw_timeseries_b_noise.Time, Vw_timeseries_b_noise.Data)
% plot(Vw_timeseries_b.Time, Vw_timeseries_b.Data)
% hold off
%
% figure();
% hold on
% plot(Vw_timeseries_c_noise.Time, Vw_timeseries_c_noise.Data)
% plot(Vw_timeseries_c.Time, Vw_timeseries_c.Data)
% hold off
```

Automatic simulink simulations with ANEMOMETER NOISE

```
% Store the time series in a cell array for easy iteration
inputData = {Vw_timeseries_a_noise, Vw_timeseries_b_noise, Vw_timeseries_c_noise};
% inputData_1 = {omega_0_a, omega_0_b, omega_0_c};
% Initialize a cell array to store output data
PowerGenerated_noise = cell(1,3);

% modelName = 'Pitch_regulated_WT_sim_dynamic_3_2'; % Replace with the actual
Simulink model name
modelName = 'PR_WT_sim_4_1';

% Run Simulink model for each time series
for i = 1:length(inputData)
    % Assign the current time series to the Simulink input
    assignin('base', 'inputSignal', inputData{i}); % inputSignal should match the
variable used in Simulink
    assignin('base', 'omega_0',     inputData_1{i});
    assignin('base', 'pidParams',   inputData_2{i})
    % Run the simulation
    simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');

    % Extract output data from the simulation
    PowerGenerated_noise{i} = simOut.get('PowerGenerated');
    Pitch{i} = simOut.get('Pitch');
```

```
    % Save output with a unique name in the workspace
    assignin('base', ['PowerGenerated_noise_' num2str(i)], PowerGenerated_noise{i});
    assignin('base', ['Pitch_' num2str(i)], Pitch{i});
end

% Plot the three outputs
figure;
hold on;
plot(PowerGenerated_noise_1.Time, PowerGenerated_noise_1.Data, 'DisplayName',
'V10a');
plot(PowerGenerated_noise_2.Time, PowerGenerated_noise_2.Data, 'DisplayName',
'V10b');
plot(PowerGenerated_noise_3.Time, PowerGenerated_noise_3.Data, 'DisplayName',
'V10c');
xlabel('Time [s]');
ylabel('Power Generated [MW]');
title('Comparison Output Power Generated WITH ANEMOMETER NOISE');
legend(['V10a = ',num2str(V10a),'m/s'],['V10b = ',num2str(V10b),'m/s'],['V10c =
',num2str(V10c),'m/s'])
grid on;
hold off;
```

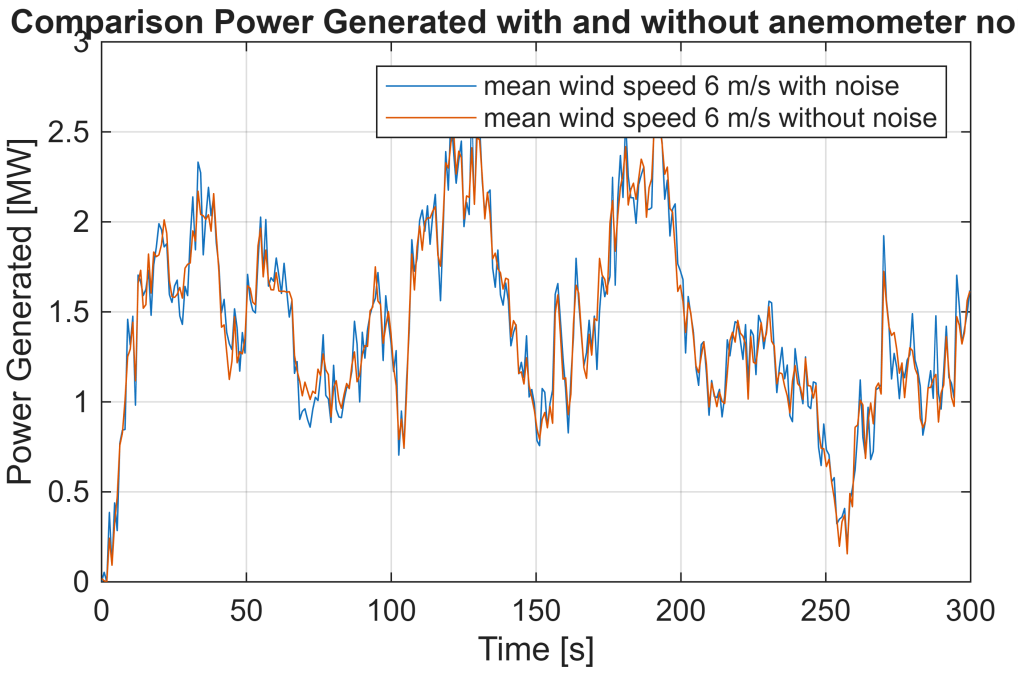


Let's understand how much have the anemometer nosie changed the power result:

RMSE = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_2[i] - y_1[i])^2}$

```
RMSE_a = sqrt(sum((PowerGenerated_noise_1.Data - PowerGenerated_1.Data).^2)/
size(PowerGenerated_1.Data, 1))/1e6; % [MW]
```

```
RMSE_b = sqrt(sum((PowerGenerated_noise_2.Data - PowerGenerated_2.Data).^2)/
size(PowerGenerated_2.Data, 1))/1e6; % [MW]
RMSE_c = sqrt(sum((PowerGenerated_noise_3.Data - PowerGenerated_3.Data).^2)/
size(PowerGenerated_3.Data, 1))/1e6; % [MW]
figure();
plot(PowerGenerated_noise_1.Time, PowerGenerated_noise_1.Data/1e6, 'DisplayName',
'V10a');
hold on;
plot(PowerGenerated_1.Time, PowerGenerated_1.Data/1e6, 'DisplayName', 'V10b');
title('Comparison Power Generated with and without anemometer noise');
xlabel('Time [s]');
ylabel('Power Generated [MW]');
legend(['mean wind speed ',num2str(V10a),' m/s with noise'],['mean wind speed
',num2str(V10a),' m/s without noise'])
grid on;
```




```
% Compute average power of noisy signals
mean_power_V10a_noise = mean(PowerGenerated_noise_1.Data)/1e6; % [MW]
mean_power_V10b_noise = mean(PowerGenerated_noise_2.Data)/1e6; % [MW]
mean_power_V10c_noise = mean(PowerGenerated_noise_3.Data)/1e6; % [MW]
% Coefficient Of Variation Root Mean Sqaure Error
COV_RMSE_a = RMSE_a / mean_power_V10a * 100;
COV_RMSE_b = RMSE_b / mean_power_V10b * 100;
COV_RMSE_c = RMSE_c / mean_power_V10c * 100;

disp(['-'])
```

-

```
disp(['Case A: mean wind speed -> ',num2str(V10a),' m/s'])
```

```
Case A: mean wind speed -> 6 m/s
```

```
disp(['Expected power:        ',num2str(P_V10a/1e6),' MW'])
```

```
Expected power:        1.4931 MW
```

```
disp(['Mean power:            ',num2str(mean_power_V10a),' MW'])
```

```
Mean power:            1.4134 MW
```

```
disp(['Mean power with noise: ',num2str(mean_power_V10a_noise),' MW'])
```

```
Mean power with noise: 1.4186 MW
```

```
disp(['RMSE: ',num2str(RMSE_a),' MW (',num2str(COV_RMSE_a, '%.3f'),'%) for V10a for
10% anemometer random noise']);
```

```
RMSE: 0.10865 MW (7.687%) for V10a for 10% anemometer random noise
```

```
disp(['-'])
```

```
-
```

```
disp(['Case B: mean wind speed -> ',num2str(V10b),' m/s'])
```

```
Case B: mean wind speed -> 10 m/s
```

```
disp(['Expected power:        ',num2str(P_V10b/1e6),' MW'])
```

```
Expected power:        6.9126 MW
```

```
disp(['Mean power:            ',num2str(mean_power_V10b),' MW'])
```

```
Mean power:            6.5896 MW
```

```
disp(['Mean power with noise: ',num2str(mean_power_V10b_noise),' MW'])
```

```
Mean power with noise: 6.5783 MW
```

```
disp(['RMSE: ',num2str(RMSE_b),' MW (',num2str(COV_RMSE_b, '%.3f'),'%) for V10b for
10% anemometer random noise']);
```

```
RMSE: 0.18597 MW (2.822%) for V10b for 10% anemometer random noise
```

```
disp(['-'])
```

```
-
```

```
disp(['Case C: mean wind speed -> ',num2str(V10c),' m/s'])
```

```
Case C: mean wind speed -> 17 m/s
```

```
disp(['Expected power:        ',num2str(Prated/1e6),' MW'])
```

```
Expected power:        10.2414 MW
```

```
disp(['Mean power:            ',num2str(mean_power_V10c),' MW'])
```

```
Mean power:             9.3861 MW
```

```
disp(['Mean power with noise: ',num2str(mean_power_V10c_noise),' MW'])
```

```
Mean power with noise: 10.2333 MW
```

```
disp(['RMSE: ',num2str(RMSE_c),' MW (',num2str(COV_RMSE_c, '%.3f'),'%) for V10c for 10% anemometer random noise']);
```

```
RMSE: 2.8926 MW (30.818%) for V10c for 10% anemometer random noise
```

```
disp(['-'])
```

```
-
```

```
% sistema fattore 3/2
```