Technische Universität Berlin
Faculty for Electrical Engineering and Computer Science
Professor Energieversorgungsnetze und Integration Erneuerbarer Energien
Renewable Energy Technology for WS 20/21

Laboratory Wind Assignment

Renewable Energy Technology WS 20/21
Professor: Kai Strunz
Submission date: 28 February 2021

Name	Matriculation number
Heesun Joo	451995
Hutomo Rachmat Saleh	461980

Structure

Task 1: Electric System Modeling and Current Command Synthesizer

1.1 Calculate ω_e, T_e, v_s

1.2 Plot current i_s vs T_e

1.3 Transform current into dq-Frame

Task 2: Control and Stability in Wind Energy Converters Using Transfer Function Analysis

2.1 Calculate $K_{d,I}$, $K_{d,P}$, $K_{q,I,}$ $K_{q,P}$ for $\tau_i = 3\ ms$

2.2 Block diagram of durrent control

Task 3: Torque Generation and Turbine-Rotor Interaction Process

3.1 Torque to speed and torque to power change

3.2 Turbine-rotor interaction process

3.3 Torque Generation as a subsystem

Task 4: Nonlinear Wind Turbine Model

4.1 Design of the power controller

4.2 Power command synthesizer and power measurement

4.3 Simulation Case 1

4.4 Simulation Case 2

Task 1: Electric System Modeling and Current Command Synthesizer

Description:

Our first task is to model the electric system and current command synthesizer. Here, we calculate the rated electrical angular frequency $\boldsymbol{\omega_e}$, rated electrical torque $\mathbf{T_e}$ as well as rated generator voltage $\mathbf{v_s}$. We then want to observe the changes in the current for the corresponding d- and q-Axis vs the varying Te. Lastly we are to transform the normal current into the dq-Frame.

Task 1.1:

The formulas used are all obtained from lecture and can be seen in the following code:

```
%% Assignment 1

wm = n / 60 * 2 * pi;
we = wm * (p / 2);
Te_rated = (Ptur * p) / (2 * we);
v_sd = 0;
v_sq = Phi_m * we;
v_s = 1 / sqrt(2) * sqrt(v_sd^2 + v_sq^2);
v_s_phase = sqrt(3) * v_s;
Te = 0:1e3:Te_rated;

i_sq_mat = zeros(1, length(Te));
i_sd_mat = zeros(1, length(Te));
for i=2:length(Te)
    % Newton-Rhapson Method w/ 10 Iteration
    % Using equations 25 & 26
    for iteration=1:10
        f = i_sq_mat(i)^4 + Phi_m * Te(i) * i_sq_mat(i) /
(3/2*p/2*(Lsd-Lsq)^2) ...
            - (Te(i) / (3/2*p/2*(Lsd-Lsq)))^2;
        df = 4*i_sq_mat(i)^3 + Phi_m*Te(i) / (3/2*p/2*(Lsd-Lsq)^2);
        di = f / df;
        i_sq_mat(i) = i_sq_mat(i) - di;
    end

    i_sd_mat(i) = -Te(i) / (3/2 * p/2 * (Lsd-Lsq) * i_sq_mat(i)) ...
                  + Phi_m / (Lsd-Lsq);
end
```

The results are:

ω_e : 188.4956 1/s

T_e : 1.6711 MNm

v_s : 3.9939 kV

Task 1.2:

The resulting plot of the code above is shown in Figure 1.2.1. We chose 1ms as our step size as it is small enough for the result to be reasonably accurate but still large to lessen the computational burden. As we can see the approximate i_{sq} and i_{sd} reference value at rated T_e is around **710.7 A** and **58 A** respectively.

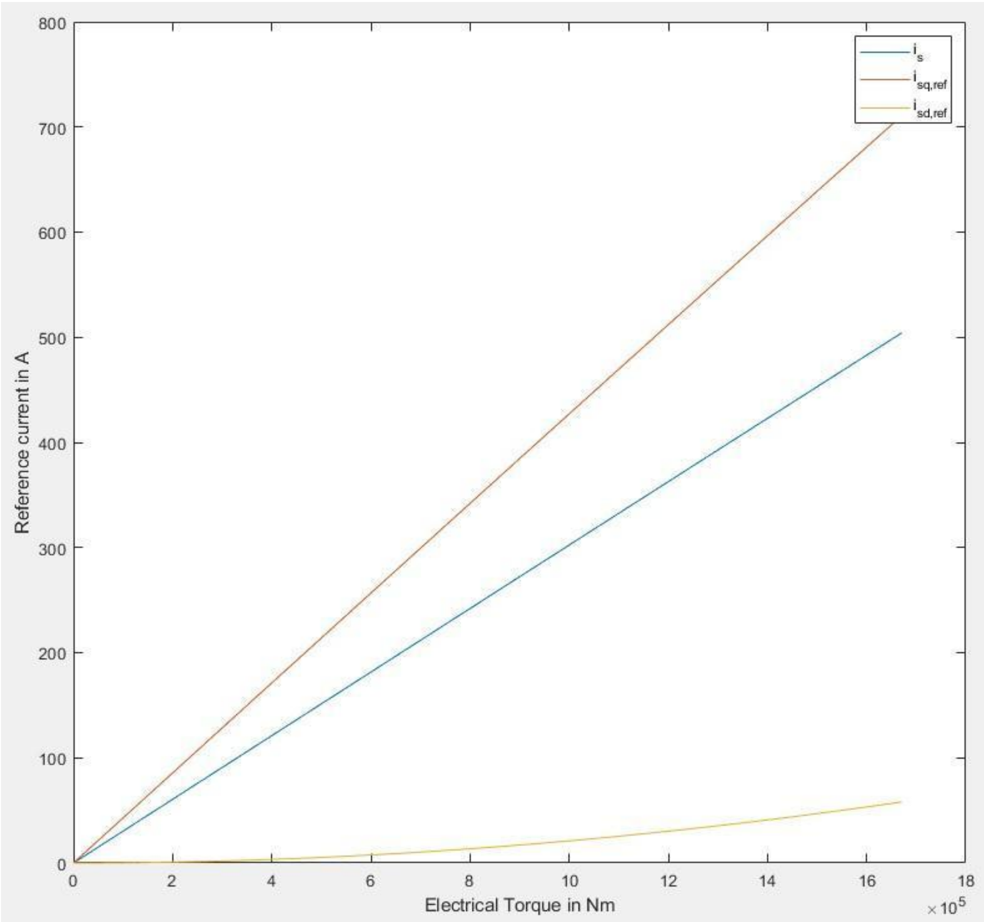

Figure 1.2.1. Plot of i_s, $i_{sd,ref}$ and $i_{sq,ref}$ over time

Task 1.3:

In this task, we applied the derivation shown in the Clarke Transform in slide 7.

Transformation to d-q-Frame :

$$\begin{pmatrix} i_{sd} \\ i_{sq} \end{pmatrix} = \frac{2}{3} \begin{pmatrix} \cos\theta & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ -\sin\theta & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \end{pmatrix} \begin{pmatrix} i_{sa} \\ i_{sb} \\ i_{sc} \end{pmatrix}$$

$\Rightarrow$ assume $\theta = 0^\circ$

$$\begin{pmatrix} i_{sd} \\ i_{sq} \end{pmatrix} = \begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} i_{sa} \\ i_{sb} \\ i_{sc} \end{pmatrix}$$

$$\begin{pmatrix} i_{sd} \\ i_{sq} \end{pmatrix} = \begin{pmatrix} \frac{2}{3} i_{sa} & -\frac{1}{3} i_{sb} & -\frac{1}{3} i_{sc} \\ & \frac{1}{\sqrt{3}} i_{sb} & -\frac{1}{\sqrt{3}} i_{sc} \end{pmatrix}$$

$$\Rightarrow \frac{3}{4}(i_{sd} + i_{sq}) = \frac{3}{4}\left(\frac{2}{3} i_{sa} + \left(\frac{1}{\sqrt{3}} - \frac{1}{3}\right) i_{sb} - \left(\frac{1}{\sqrt{3}} + \frac{1}{3}\right) i_{sc}\right)$$

$$\overset{!}{=} \frac{1}{2} \cdot \frac{3}{2}\left(\frac{2}{3} i_{sa} + \frac{3-\sqrt{3}}{3\sqrt{3}} i_{sb} - \frac{3+\sqrt{3}}{3\sqrt{3}} i_{sc}\right)$$

$$\frac{3}{4}(i_{sd} + i_{sq}) \overset{!}{=} \frac{1}{2}\left(i_{sa} + \frac{6-2\sqrt{3}}{\sqrt{3}} i_{sb} - \frac{6+2\sqrt{3}}{\sqrt{3}} i_{sc}\right)$$

$$\Rightarrow \frac{3}{4}(M_d i_{sd} + M_q i_{sq}) = \frac{1}{2}\left(M_a i_{sa} + M_b i_{sb} + M_c i_{sc}\right) = i_{dc}$$

Task 2: Control and Stability in Wind Energy Converters Using Transfer Function Analysis

Description:

As our second task, we focus on building controller models in simulink. We are to calculate the parameters of the controller **Kd,i**, **Kd,p**, **Kq,i**, **Kq,p** for **τ**i = 3ms. After that we are to build a simple controller model as shown in the project overview. The simulink model is shown in Figure 2.1.1.

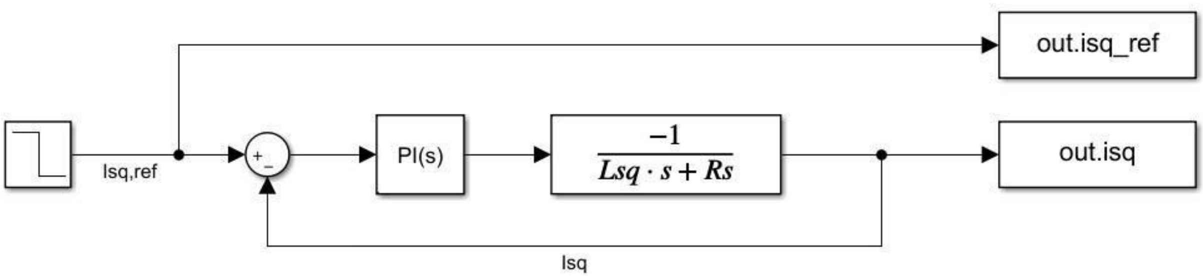

Figure 2.1.2. Plot of isq and isq,ref over time

Task 2.1:

For the simulink model we followed the Figure shown in the assignment overview. The $i_{sq,ref}$ is fed using a step up block with an initial value of $i_{sq,ref}$ and a final value of $i_{sq,ref}$ - Δi_{sq}. The values of the PI Controller are first calculated in code and exported to our simulink model.

a) The resulting plot of the controller is shown in Figure 2.1.2. We can see that the controller follows the reference value but with slight deviation that is expected.

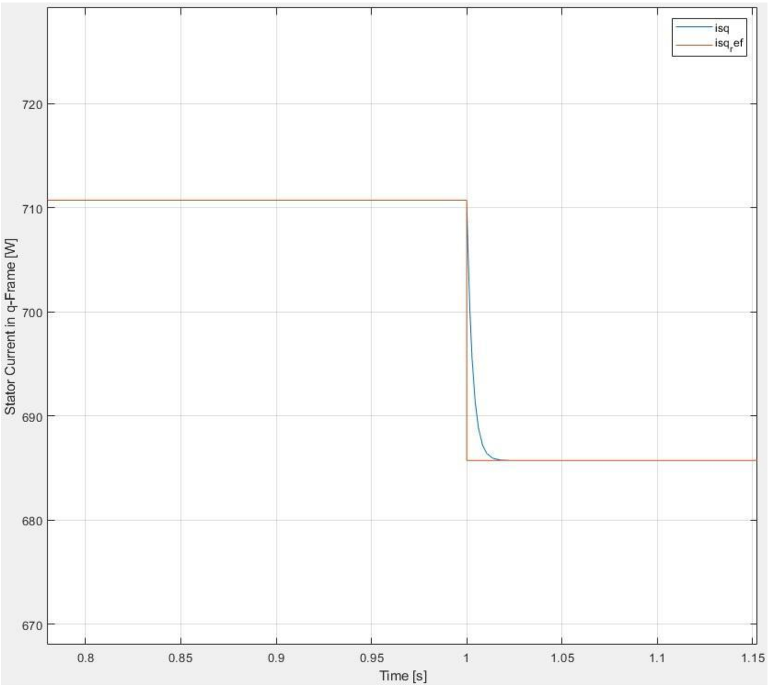

Figure 2.1.2. Plot of isq and isq,ref over time

b) Since both isq and isd has been calculated simultaneously in Task 1, each isq is coupled with isd. Using the find() function, the index can be found from the isq array from Task 1 and thus the corresponding value of isd can be also be found. The results are as follows:

i_{sd} : 53.9894 A

Δi_{sd} : 4.0161 A

The code is shown below.

```
% 2.1 b
matrix_diff = abs(i_sq_mat - i_sq); % Find difference of isq value
index = matrix_diff == min(matrix_diff); % Get index of minimum value
i_sd_ref = i_sd_mat(end); % Get last value of isd_ref
i_sd = i_sd_mat(index); % Use index to find corresponding isd
delta_d = i_sd_ref - i_sd; % Calculate delta
```

c) Figure 2.1.3 shows an enlarged image of the previous plot and has been modified to show the τ_i and the change of current during this time step. The value measured shown in the figure is as expected.

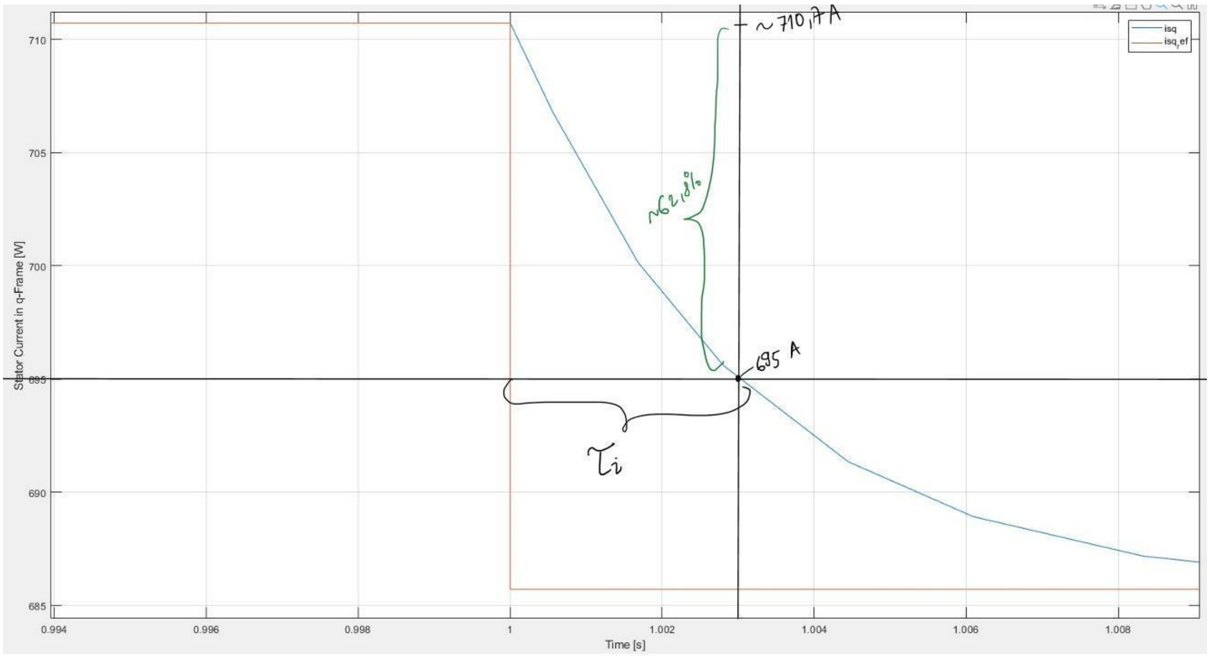

Figure 2.1.3. Enlarged view of isq

d) No because as shown in equation (17) in the paper. The controller parameters are strictly dependent on the values of Rs and Lsd for d-Frame and Lsq for q-Frame.

Task 2.2:

In this task, our main task is to recreate the block diagram of current control. All the variables are imported from the .m code and the results of m_d, m_q and v_{sq}, v_{sd}.

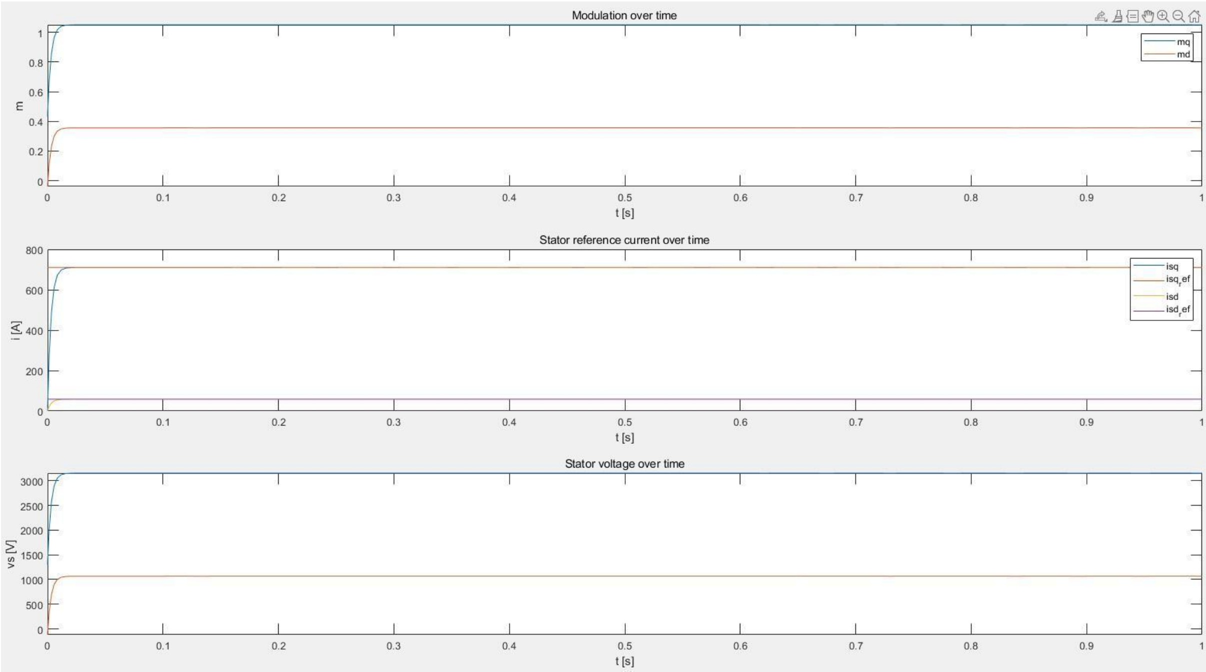

Figure 2.2.1. Plot of i_{sq}, i_{sd}, m_q, m_d, v_{sq} & v_{sd}

c) As we can observe in the plot diagram in Figure 2.2.2. The difference between the output Power calculated with the produced Idc and the Ptur given converges to zero. This shows that the expected power generation is as expected.

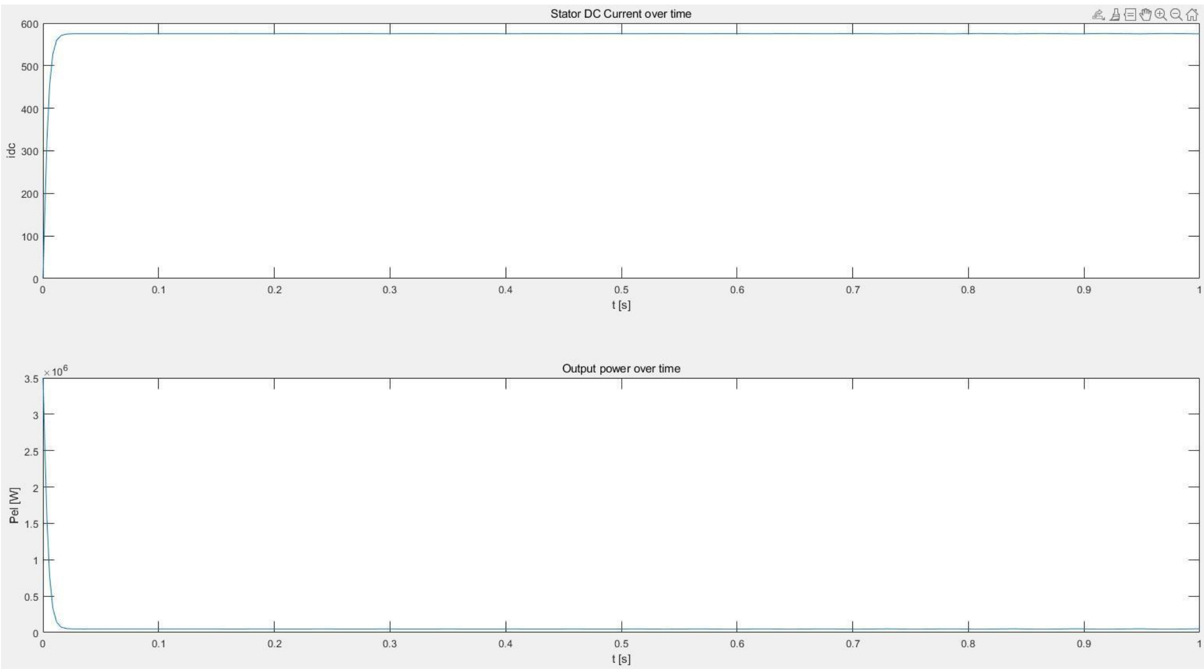

Figure 2.2.2. Course of i_{dc} and P_{el} over time

Task 3: Torque Generation and Turbine-Rotor Interaction Process

Description:

Since we calculated the value of the overall inertia J, the optimal operation points and the corresponding time, we are able to build block diagrams and to observe the Torque-Rotor Interaction Process and Torque Generation by building some simulations.

Task 3.1:

Using the simulation about a torque change under a stable wind speed, we can explain the principal for the reaction of speed and power on a torque change. The resulting plot is shown in Figure 3.1.4.

a) According to two equations for H and S_base, the overall inertia J is calculated, because i_{sq} and i_{sd} have been calculated in the Task 1. The values of L_{sd} , L_{sq} and ϕ_m have been given. The code is shown below.

```
% 3.1 a
Sb = Pel;
J = H * 2 * Sb / wm^2;
```

The results are:

J : 9.5749e+06 $kg \cdot m^2$

b) Since there is the assumption that $P_{e,ref} = P_{tur,opt}$ with a wind speed of 10 m/s, so that some optimal operating points such as $P_{tur,opt}$, $\omega_{e,opt}$, $T_{e,opt}$ can be calculated. The code is shown below.

```
% 3.1 b
wm_opt = lambda_opt * Vw_3 / r; % mechanical angular velocity [rpm]
we_opt = p / 2 * wm_opt; % electrical angular velocity [rpm]
Ct_opt = c0 + c1*lambda_opt + c2*lambda_opt^2;
Cp_opt = lambda_opt * Ct_opt;
Ptur_opt = 4 / p^3 * pi * rho * r^5 * we_opt^3 * Cp_opt / lambda_opt^3; %
[W]
Pe_opt = Ptur_opt;
Te_opt = Pe_opt * p / (2 * we_opt);
```

The results are:

$P_{tur,opt}$ **: 2.1051 MW**

$\omega_{e,opt}$ **: 120.6000 1/s**

$T_{e,opt}$ **: 1.5710 MNm**

c) For the operating point, we can calculate the corresponding time constants τ_w and τ_z. The code is shown below.

```
% 3.1 c
a1 = c0 * pi * rho * r^3;
a2 = c1 * pi * rho * r^4 / p;
a3 = 4 * c2 * pi * rho * r^5 / p^2;
D = 0; % No damping torque
tau_w = - 2 * J / (p * (a2*Vw_3 + a3*we_opt - D*2/p));
tau_z = tau_w / (1 - p * tau_w * Te_opt / (2 * we_opt * J));
```

The results are:

τ_w **: 8.8265 s**

τ_z **: -109.3302 s**

d) We implemented both block diagrams shown below.

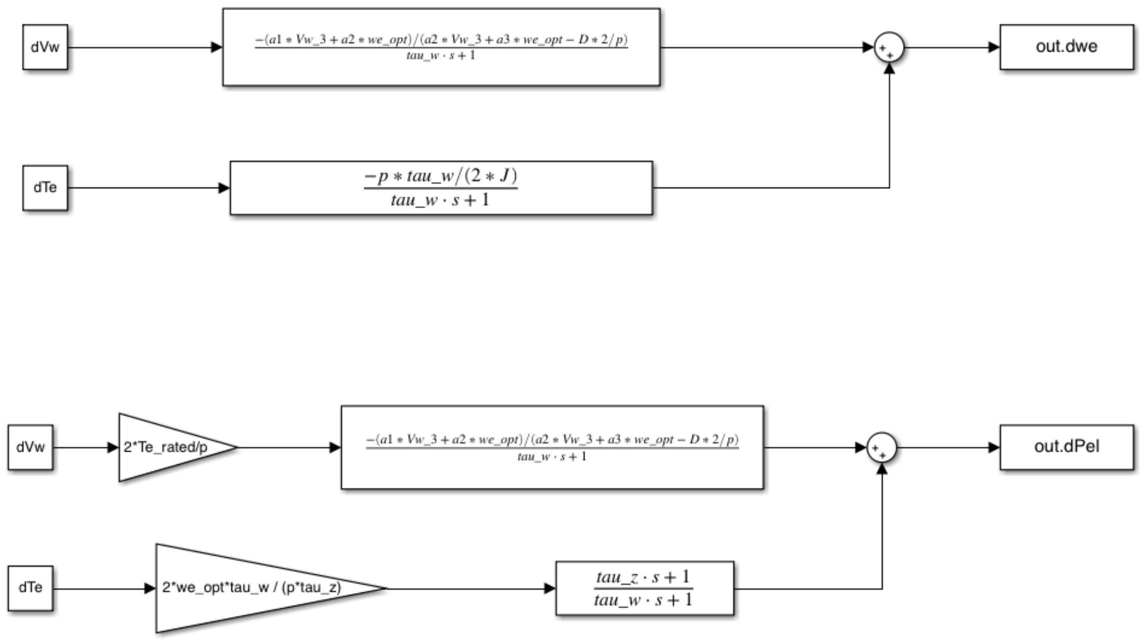

Figure 3.1.1 Plot for a torque change

There are both plots regarding the resulting speed change and power change in Figure 3.1.5. Both plots below show that the change of the speed and power has been decreased markedly from 0s to around 20s on a torque change. Moreover, we can observe that there is a more dramatic decrease for the torque change than for the constant torque. That's why the velocity is proportional to torque. But the change of speed and power is in inverse proportion to torque. If the torque has the bigger value, the change of speed has minus value and the change of power is also dropped.

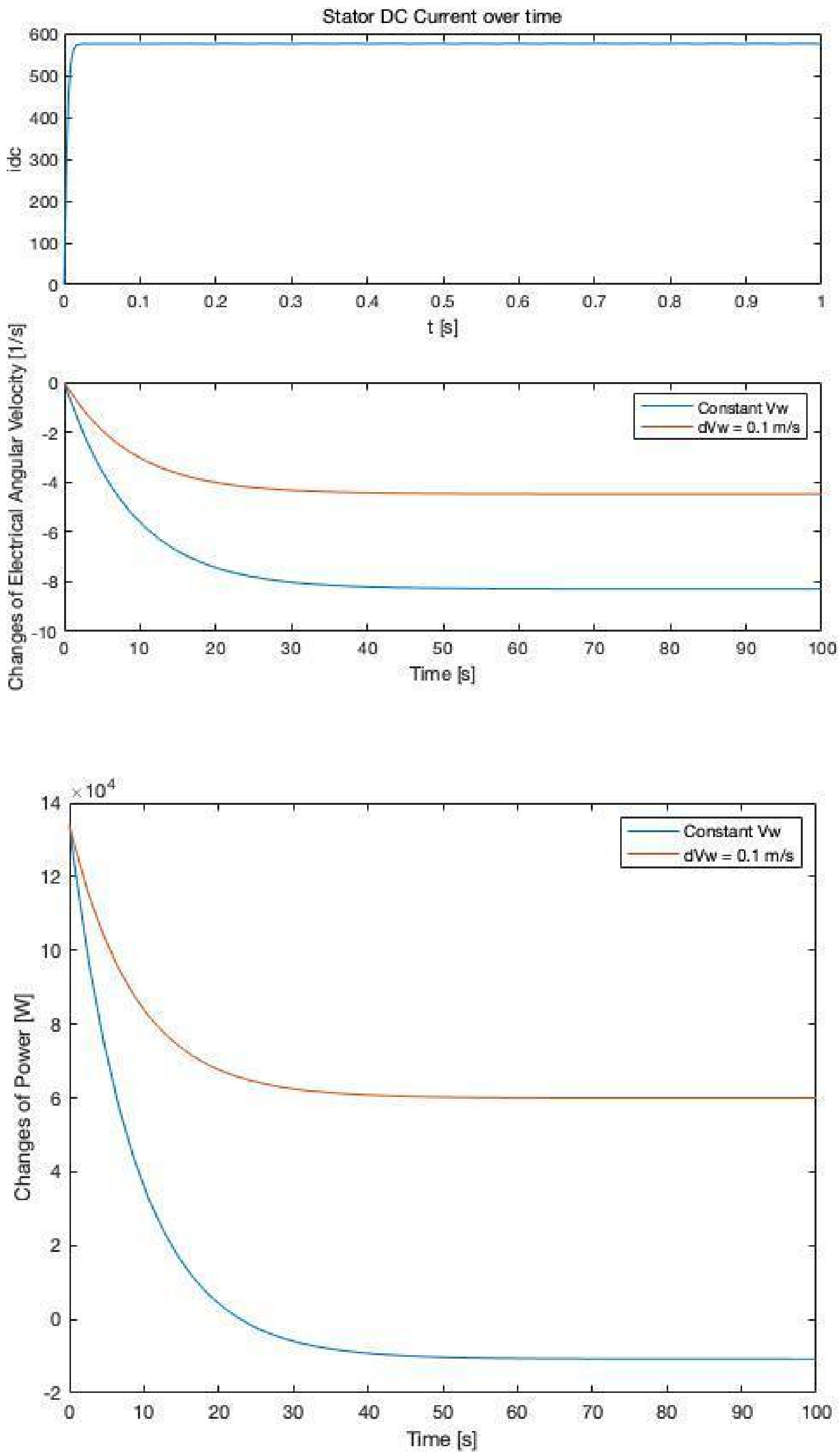

Figure 3.1.2 Speed change and Power change on a torque change

Task 3.2:

As implementing both blocks of the turbine-rotor interaction process in Simulink, we can define the initial value of the integrator in the Simulink model and observe the T_{tur} in comparison to T_e. The simulink model is shown in Figure 3.2.1.

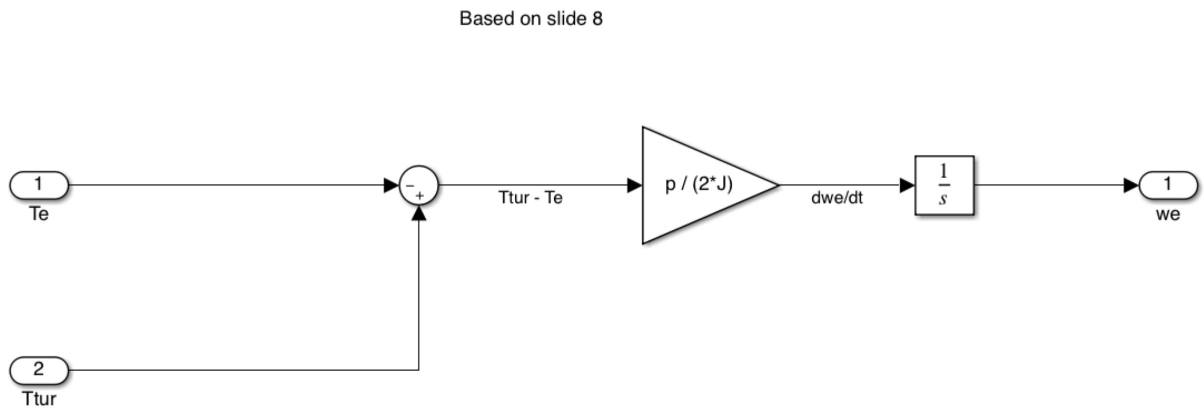

Figure 3.2.1 Turbine-rotor interaction Process

a) There is a assumption for speed $V_w = 10\ m/s$ and the T_e under V_w. We built up the plot about T_{tur} in comparison to T_e. Since we have set up the initial value with $\omega_{e,opt}$ simulating using T_e with the value of its optimal value will result in the plot below. The plot does not converge since the value does not change.

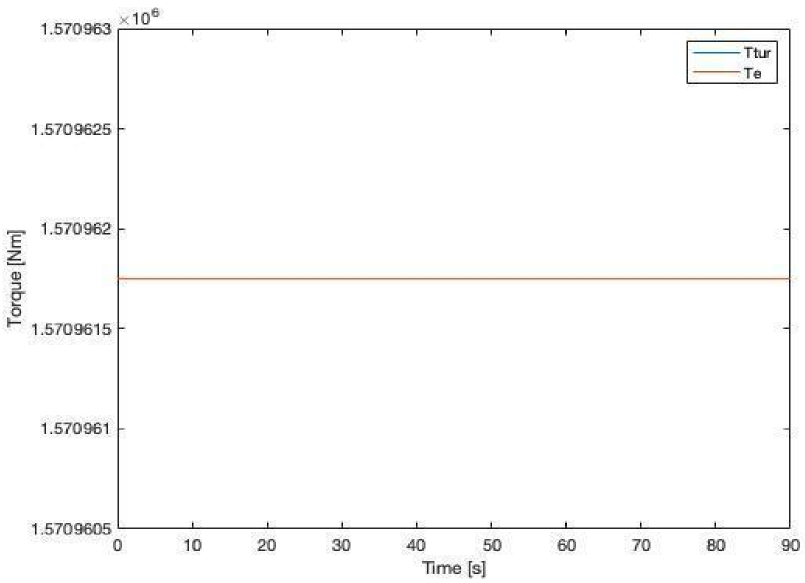

Figure 3.2.2 Optimal Torque under the wind speed 10m/s

b) In this task, we can calculate the torque for the turbine using the equation (5) and (6) shown in the assignment overview. The resulting simulink model is shown in Figure 3.2.3.

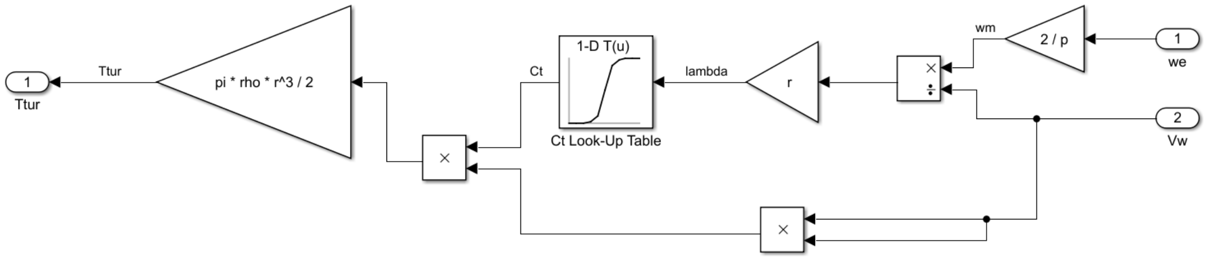

Figure 3.2.3 Simulink block of turbine part

c) For rotor we can calculate ω_e by using the equation (7) about the overall inertia J. The end result of the simulink block for this task is shown in Figure 3.2.1.

Task 3.3:

We are able to implement the torque generation in Simulink. By simulating it, we can figure out an appropriate simulation time for our torque generation and observe the ω_e. The simulink model is shown in Figure 3.3.1.

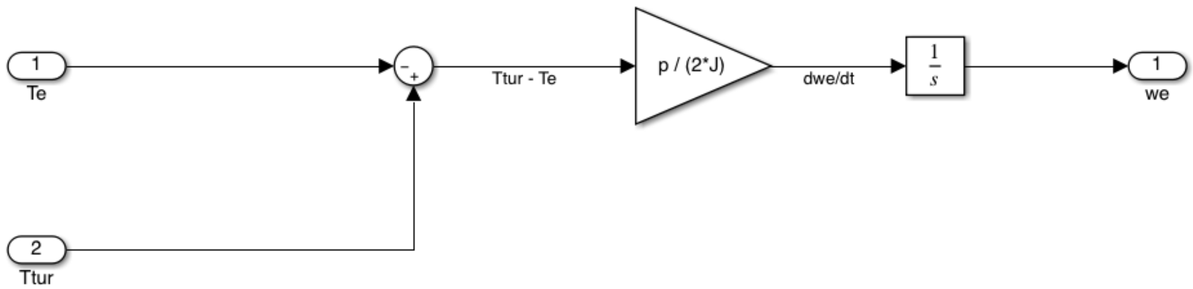

Figure 3.3.1 Simulink block for torque generation

a) We simulated the optimal current values i_{sq} and i_{sd} under a wind speed $V_w = 10\ m/s$. The i_{sq} and i_{sd} value is calculated with the Newton-Rhapson method done in our first task using optimal variables. Then we can get the appropriate time to work the torque generation. The suitable time is 60s. The code for this task is shown below. The code is shown below.

```
% 3.3 a
% Same calculation as Assignment 1
Te = 0:1e3:Te_opt;
i_sq_opt = zeros(1, length(Te));
i_sd_opt = zeros(1, length(Te));
for i=2:length(Te)
```

```
    for iteration=1:10
        f = i_sq_opt(i)^4 + Phi_m * Te(i) * i_sq_opt(i) /
(3/2*p/2*(Lsd-Lsq)^2) - (Te(i) / (3/2*p/2*(Lsd-Lsq)))^2;
        df = 4*i_sq_opt(i)^3 + Phi_m*Te(i) / (3/2*p/2*(Lsd-Lsq)^2);
        di = f / df;
        i_sq_opt(i) = i_sq_opt(i) - di;
    end
    i_sd_opt(i) = -Te(i) / (3/2 * p/2 * (Lsd-Lsq) * i_sq_opt(i)) ...
                  + Phi_m / (Lsd-Lsq);
end
i_sq_opt = i_sq_opt(end);
i_sd_opt = i_sd_opt(end);
```

b) As the turbine-rotor interaction process, we made a plot about ω_e. The plot is shown below in Figure 3.3.2.

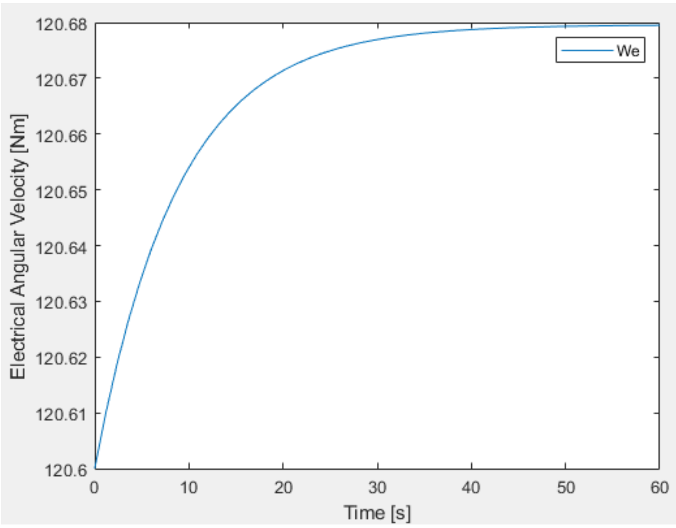

Figure 3.3.2 Plot of ω_e over time

The plot above shows that the value converges to around 120.68, even though the initial value is already set as our calculated optimal ω_e of 120.6. The reason for this is most likely due to the system calculating its own optimal ω_e value using our optimal stator current in dq-Frame. The difference between the initial value and the end value is marginal.

Task 4: Nonlinear Wind Turbine Model

Description:

Using our models and data in assignments 1-3, our last task is to combine them and run simulations. The subsystem power synthesizer and controller are to be constructed in 4.1 and 4.2. In Task 4.3 and 4.4 the models are combined and run with static as well as varying wind speeds. The graphs are to be observed and explained.

Task 4.1:

The simulink models of the power controller are shown in Fig. 4.1.1

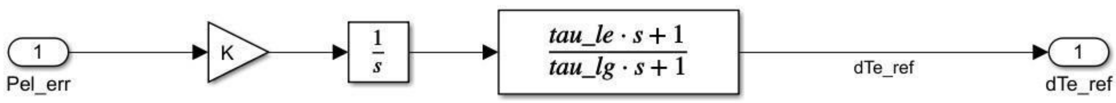

Figure 4.1.1. Simulink Block of Power Controller

The lead τ_{le} and lag τ_{lg} time constant compensates the process due to τ_{ω} and τ_{z}. Which is why the value is the same. The closed-loop time constant is given as 5% of the mechanical lag time $\tau_{le} = \tau_{\omega}$.

We then apply a sudden change of reference power of -50kW to the turbine. We coupled the power controller with the turbine-rotor interaction process block made in our previous task. As input we used a step function which introduces our spike in power at t = 5s. The whole block is shown in Fig. 4.1.2.

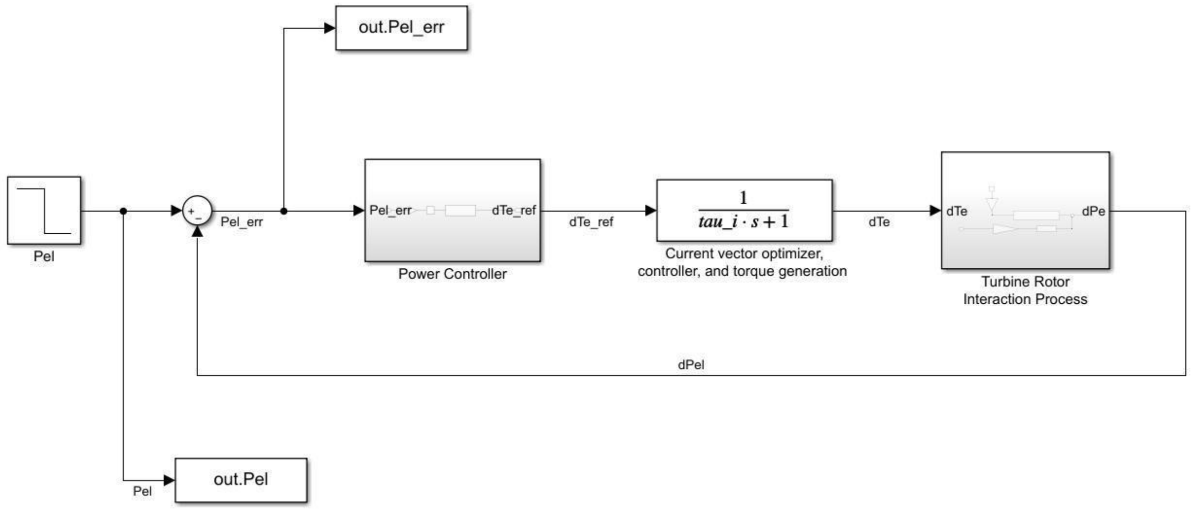

Figure 4.1.2. Complete Simulink Block of Task 4.1

As we can see in Figure 4.1.3 the error converges to 0 as it should be. We can observe a small spike at the time of t = 5s. This is due to the sudden power change specified for this task. The controller then restabilizes the value and the error converges to zero again.

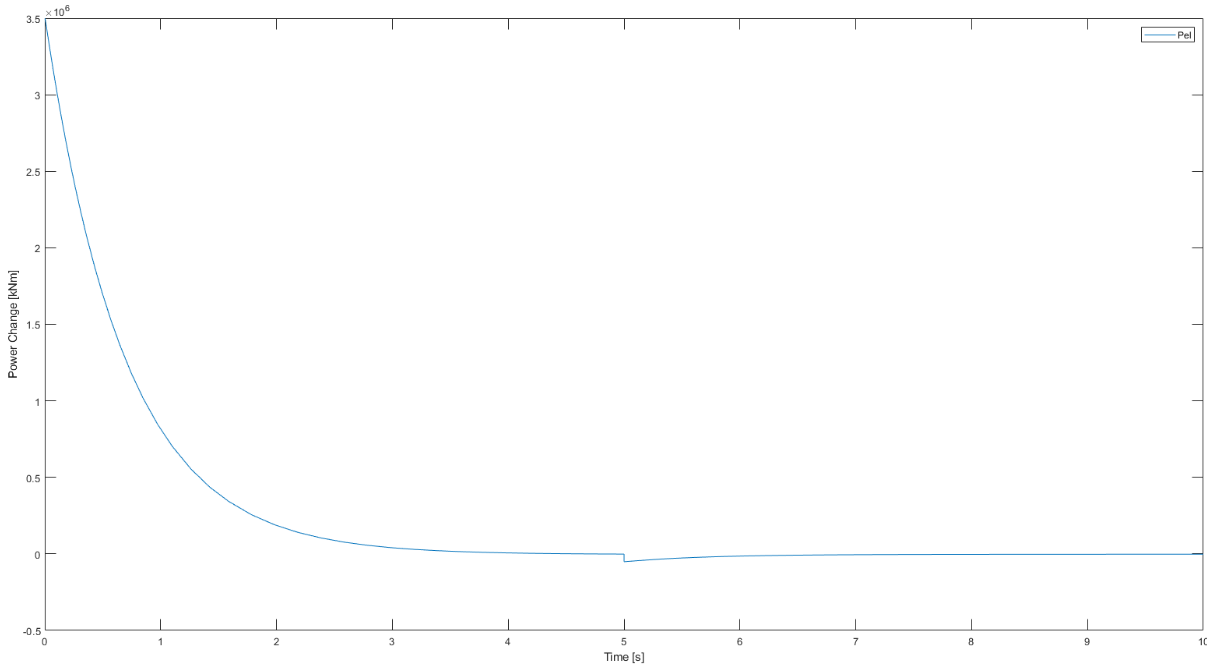

Figure 4.1.3. Change in Pel over time

Task 4.2:

In this task, we built the power command synthesizer and power measurement in Simulink. There is no simulation involved. The simulink blocks are shown in Figure 4.2.1 and 4.2.2

blocks are derived from the equations shown on slides 54 and 58 from chapter 8 respectively.

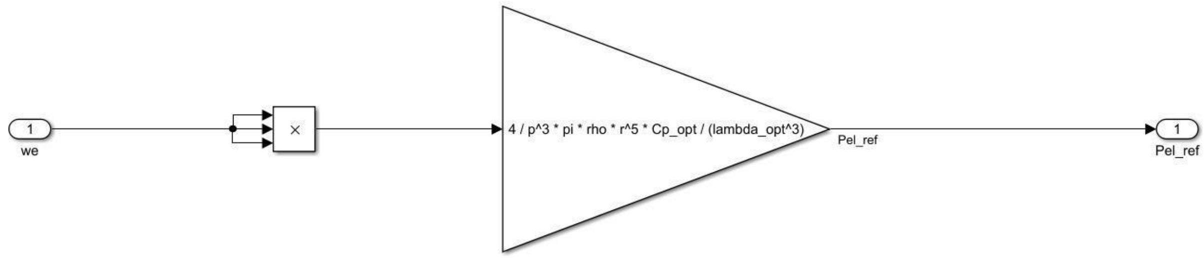

Figure 4.2.1. Simulink Block of Power Command Synthesizer

With the assumption of non-changing i_{sd} and i_{sq} given from the task, the PL becomes zero. Thus the Power P_e is measured only from the difference between P_{ter} and P_L.

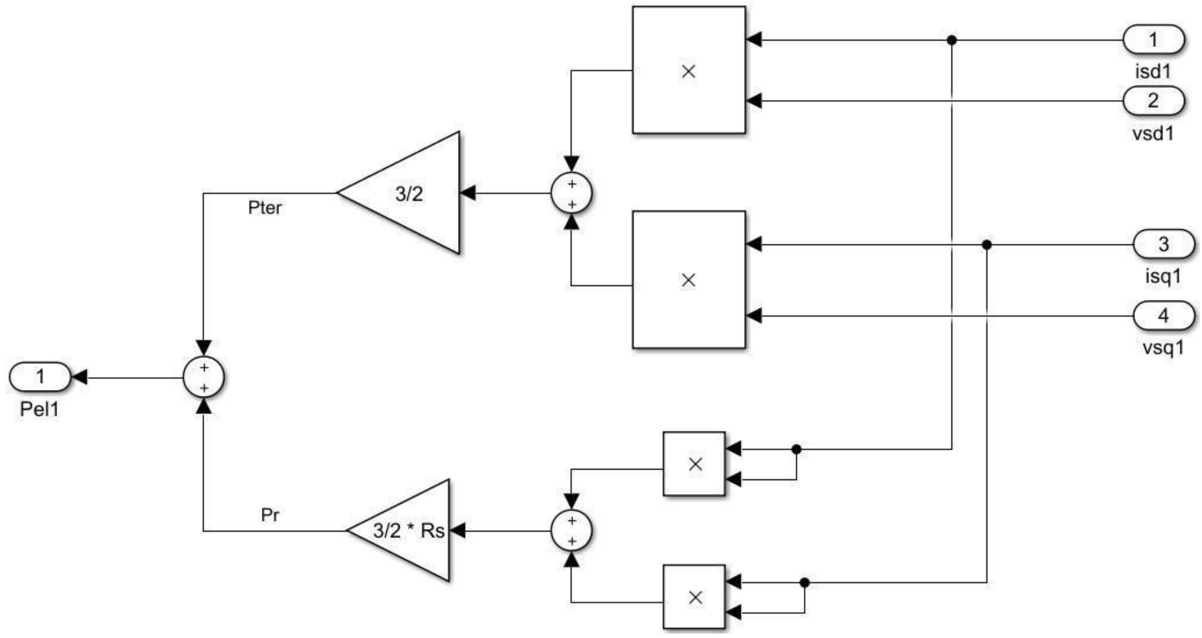

Figure 4.2.2. Simulink Block of Power Measurement

Task 4.3:

Using the blocks made from the previous tasks, we now build the whole wind turbine system. The complete simulink model is shown in Figure 4.3.1.

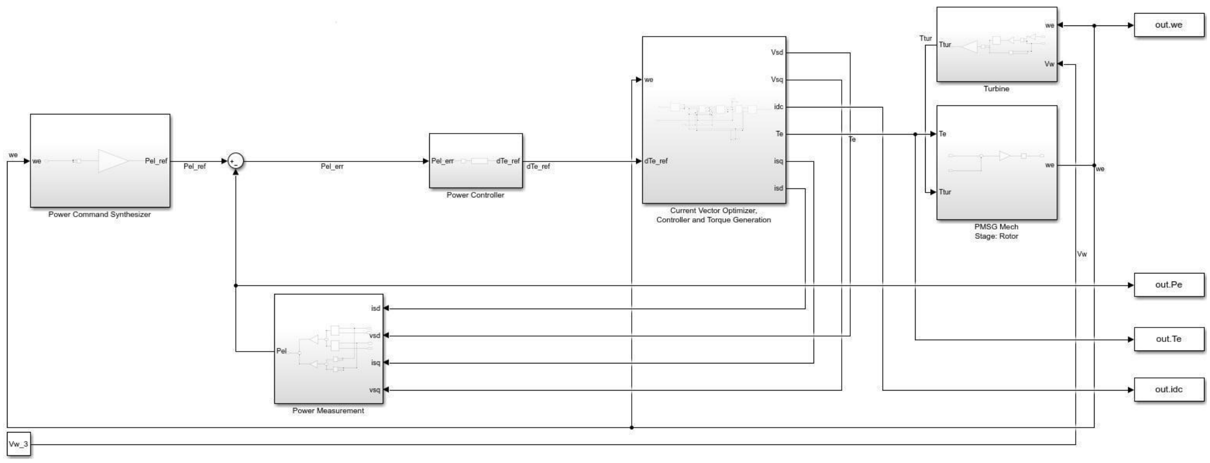

Figure 4.3.1. Complete Wind Turbine System in Simulink

The block from Task 4.1 and 4.2 is added with the simulink block from 3.3. The simulation is to be executed with a constant wind speed of 10 m/s. We are to observe the course of power P_e, electric angular velocity ω_e and torque T_e over 100s. The resulting plot is shown in Figure 4.3.2.

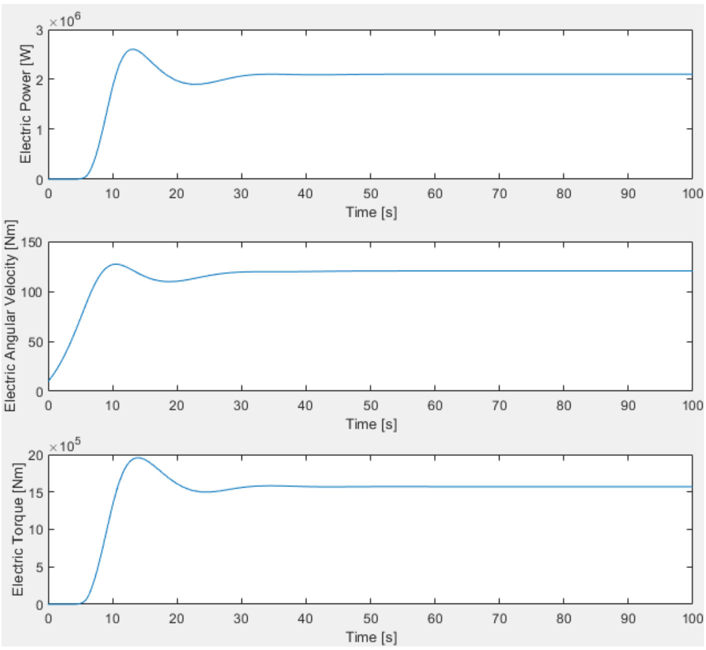

Figure 4.3.2. Plot of P_e, ω_e and T_e

As expected, after the initial transient, the course of the values becomes stable after some time. This shows that the system works as intended.

Task 4.4:

Different from the simulation on Task 4.3, we are to simulate with a fluctuating wind speed around 10 m/s. The fluctuation is created via code using a random generated number with rand() and we set up the lower and upper limit to ±0.5 m/s. To control the result of the RNG, we used the default seed from matlab. We set the wind speed course for 300s, converted the array into a time-series format and exported it into a .mat file. The .mat file is then imported to our simulink model. The code used in this task is shown below.

```
% 4.4
% Generate wind noise
t_44 = (1:300);
rng('default')
noise = -0.5 + 1*rand(1, length(t_44));
Vw = Vw_3 + noise;
Vw_ts = timeseries(Vw', t_44);
save("wind_speed_fluct.mat", 'Vw_ts', '-v7.3')
simout_44 = sim('Task4_4.slx');
```

The simulink block stays almost identical to Figure 4.3.1, the only difference being the wind speed block is changed into a “from File” block. The result of the simulation can be seen in Figure 4.4.1.

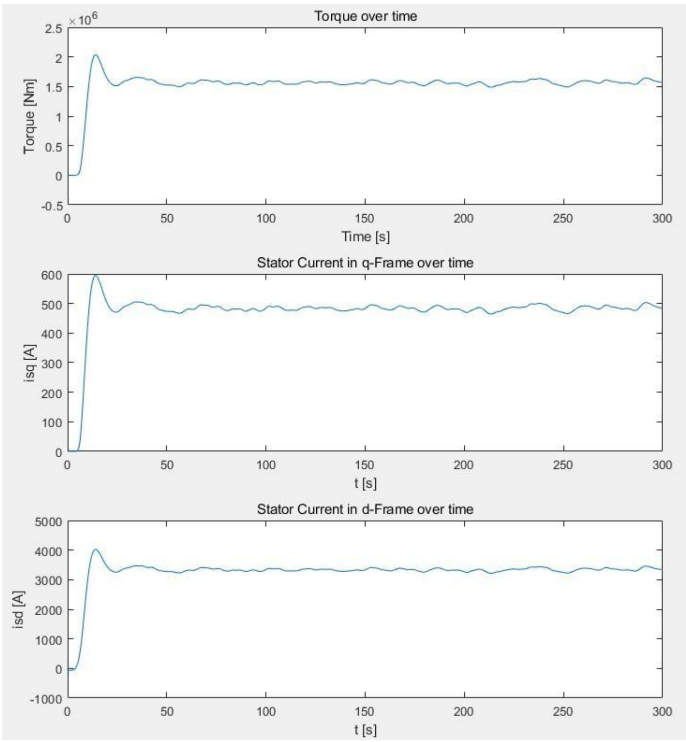

Figure 4.4.1. Plot of P_e, ω_e and T_e

In comparison to Figure 4.3.2, the resulting plot with fluctuation wind speed results in a fluctuation of values. Nonetheless, throughout the whole course the value stays relatively to the desired value. The Figure 4.4.2 shows the relation between P_e with ω_e as well as C_p with T_e.

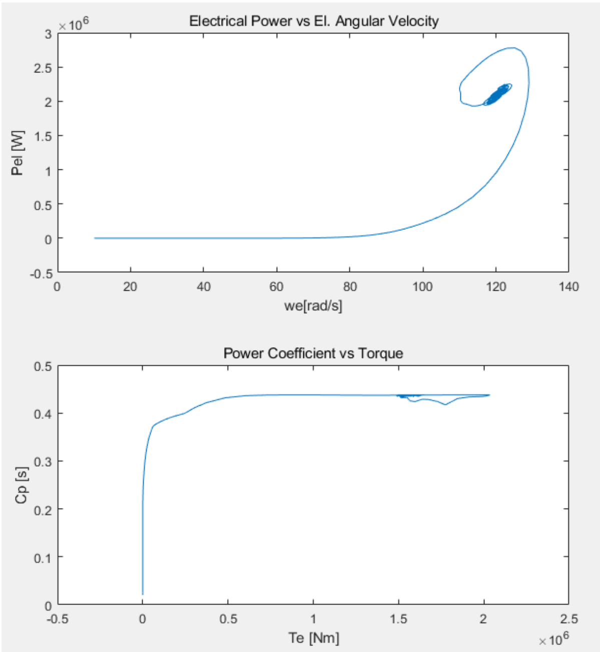

Figure 4.4.2. Plot of P_e, ω_e and T_e

The figure shows that both P_e with ω_e as well as C_p and T_e converges to a desired value. But due to the fluctuations no definite point and a spiral due to the values moving back and forth is observed.

For the task 4.4.e we are to compare the minimal and maximal value of Pe with the expected Pe from theory. In order to find the minimal and maximal value of the wind speed, we would have to find it using the min() and max() function. The resulting min and max value of the wind speed are **9.5046** and **10.4961** m/s. We then use the same method to find the value of P_e, the only difference is that we need to start searching the value after the transient has passed. The resulting min and max P_e are **1.923** and **2.239** MW. To check whether the min/max wind speed value does not appear during transient, we looked up the index using find().

The code used for this task is shown below.

```
% 4.4 e
% Get minimal and maximal value of Vw and its corresponding Pe.
min_Vw = min(Vw);
max_Vw = max(Vw);
min_Pe = min(Pel_44(200:end));
max_Pe = max(Pel_44(200:end));
min_index = find(Pel_44(200:end) == min_Pe);
max_index = find(Pel_44(200:end) == max_Pe);
min_Cp = Cp_44(min_index);
max_Cp = Cp_44(max_index);

min_Pe_theory = 1 / 2 * pi * rho * r^2 * min_Vw^3 * min_Cp;
max_Pe_theory = 1 / 2 * pi * rho * r^2 * max_Vw^3 * max_Cp;
```

The P_e from theory can be achieved by following the formula stated in Slide 54. The resulting min and max from theory are **1.6739** and **2.4229** MW. The results are quite far from each other. This is probably due to the deviation that occurs from each part of the simulink models used.

The full code used in this project is as follows:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Wind Assignment %%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear
close all

%{
    RET - Wind Assignment

    Submitted by:
    Heesun Jo
    Hutomo Saleh
%}

%% Parameters

Ptur = 3.5e6; % Rated power [W]
Pel = Ptur;
Vw = 12; % Rated wind speed [m/s]
n = 20; % Rated mechanical angular velocity [rpm]
Cp_opt = 0.4378; % Maximum power coefficient
r = 50; % Rotor radius [m]
lambda_opt = 6.7; % Optimal tip speed ratio
H = 6; % Inertia constant of turbine & PMSG [s]
p = 180; % Number of Poles
Rs = 60e-3; % Stator resistance [Ohm]
Lsd = 6e-3; % Stator d-axis inductance [H]
Lsq = 8e-3; % Stator q-axis inductance [H]
Phi_m = 17.3; % Flux induced by magnets [Wb]
V_dc = 6.5e3; % DC Voltage [V]
rho = 1.225; % Air density [kg/m3]

%% Assignment 1

wm = n / 60 * 2 * pi;
we = wm * (p / 2);
Te_rated = (Ptur * p) / (2 * we);
v_sd = 0;
v_sq = Phi_m * we;
v_s = 1 / sqrt(2) * sqrt(v_sd^2 + v_sq^2);
v_s_phase = sqrt(3) * v_s;
Te = 0:1e3:Te_rated;

i_sq_mat = zeros(1, length(Te));
i_sd_mat = zeros(1, length(Te));
for i=2:length(Te)
    % Newton-Rhapson Method w/ 10 Iteration
    % Using equations 25 & 26
    for iteration=1:10
        f = i_sq_mat(i)^4 + Phi_m * Te(i) * i_sq_mat(i) /
(3/2*p/2*(Lsd-Lsq)^2) ...
            - (Te(i) / (3/2*p/2*(Lsd-Lsq)))^2;
        df = 4*i_sq_mat(i)^3 + Phi_m*Te(i) / (3/2*p/2*(Lsd-Lsq)^2);
        di = f / df;
        i_sq_mat(i) = i_sq_mat(i) - di;
```

```matlab
    end

    i_sd_mat(i) = -Te(i) / (3/2 * p/2 * (Lsd-Lsq) * i_sq_mat(i)) ...
                  + Phi_m / (Lsd-Lsq);
end

% Plot
figure(1)
i_s = (i_sq_mat.^2 + i_sd_mat.^2).^(1/2) / sqrt(2);
plot(Te, i_s, Te, i_sq_mat, Te, i_sd_mat);
xlabel('Electrical Torque in Nm')
ylabel('Reference current in A')
legend('i_{s}', 'i_{sq,ref}', 'i_{sd,ref}')

%% Assignment 2

% Calculate controller parameters
tau_i = 3e-3; % [s]
Kq_p = - Lsq / tau_i;
Kd_p = - Lsd / tau_i;
Kq_i = Rs * Kq_p / Lsq;
Kd_i = Rs * Kd_p / Lsd;

% 2.1 a
delta_q = -25;
i_sq_ref = i_sq_mat(end); % Get last value of isq_ref
i_sq = i_sq_ref + delta_q;
simout_21 = sim('Task2_1.slx');
t_21 = simout_21.tout;
isq_21 = simout_21.isq;
isq_ref_21 = simout_21.isq_ref;

figure(1)
plot(t_21, isq_21, t_21, isq_ref_21)
grid on
xlabel('Time [s]')
ylabel('Stator Current in q-Frame [W]')
legend('isq', 'isq_ref')

% 2.1 b
matrix_diff = abs(i_sq_mat - i_sq); % Find difference of isq value
index = matrix_diff == min(matrix_diff); % Get index of minimum value
i_sd_ref = i_sd_mat(end); % Get last value of isd_ref
i_sd = i_sd_mat(index); % Use index to find corresponding isd
delta_d = i_sd_ref - i_sd; % Calculate delta

% 2.1 c
index = 43;
i_sq_calc = i_sq_mat(end-index);

% 2.2
Vdc = 6e3; % Rated DC Voltage [V]

simout_22 = sim('Task2_2.slx');
t_22 = simout_22.tout;
isq_22 = simout_22.isq;
```

```
isd_22 = simout_22.isd;
isq_ref_22 = simout_22.isq_ref;
isq_ref_22 = isq_ref_22*ones(1, length(t_22));
isd_ref_22 = simout_22.isd_ref;
isd_ref_22 = isd_ref_22*ones(1, length(t_22));
mq_22 = simout_22.mq;
md_22 = simout_22.md;
vsq_22 = simout_22.vsq;
vsd_22 = simout_22.vsd;
idc_22 = simout_22.idc;
Pel_22 = simout_22.Pel;

figure("name", "Plot of isq, isd, mq, md, vsq & vsd");
subplot(311)
plot(t_22, mq_22, t_22, md_22)
subtitle("Modulation over time")
xlabel('t [s]')
ylabel('m')
legend('mq', 'md')
subplot(312)
plot(t_22, isq_22, t_22, isq_ref_22, t_22, isd_22, t_22, isd_ref_22)
subtitle("Stator reference current over time")
xlabel('t [s]')
ylabel('i [A]')
legend('isq', 'isq_ref', 'isd', 'isd_ref')
subplot(313)
plot(t_22, vsq_22, t_22, vsd_22)
subtitle("Stator voltage over time")
xlabel('t [s]')
ylabel('vs [V]')

figure("name", "Stator current and output power");
subplot(211)
plot(t_22, idc_22)
subtitle("Stator DC Current over time")
xlabel('t [s]')
ylabel('idc')
subplot(212)
plot(t_22, Pel_22)
subtitle("Output power over time")
xlabel('t [s]')
ylabel('Pel [W]')

%% Assignment 3

% Parameters
Vw_3 = 10; % Wind speed for 3rd Task [m/s]
c0 = 2.25e-2;
c1 = 2.18e-2;
c2 = -0.23e-2;

% 3.1 a
Sb = Pel;
J = H * 2 * Sb / wm^2;
% 3.1 b
wm_opt = lambda_opt * Vw_3 / r; % mechanical angular velocity [rpm]
```

```
we_opt = p / 2 * wm_opt; % electrical angular velocity [rpm]
Ct_opt = c0 + c1*lambda_opt + c2*lambda_opt^2;
Cp_opt = lambda_opt * Ct_opt;
Ptur_opt = 4 / p^3 * pi * rho * r^5 * we_opt^3 * Cp_opt / lambda_opt^3; %
[W]
Pe_opt = Ptur_opt;
Te_opt = Pe_opt * p / (2 * we_opt);
% 3.1 c
a1 = c0 * pi * rho * r^3;
a2 = c1 * pi * rho * r^4 / p;
a3 = 4 * c2 * pi * rho * r^5 / p^2;
D = 0; % No damping torque
tau_w = - 2 * J / (p * (a2*Vw_3 + a3*we_opt - D*2/p));
tau_z = tau_w / (1 - p * tau_w * Te_opt / (2 * we_opt * J));
% 3.1 d
dVw = 0;
dTe = 1e5;
simout_31 = sim('Task3_1.slx');
dwe1 = simout_31.dwe;
dPel1 = simout_31.dPel;
t_31 = simout_31.tout;

dVw = 0.1; % New wind speed [m/s]
simout_31 = sim('Task3_1.slx');
dwe2 = simout_31.dwe;
dPel2 = simout_31.dPel;

% Plot
figure(3)
plot(t_31, dwe1, t_31, dwe2);
xlabel('Time [s]')
ylabel('Changes of Electrical Angular Velocity [1/s]')
legend('Constant Vw', 'dVw = 0.1 m/s')
figure(4)
plot(t_31, dPel1, t_31, dPel2);
xlabel('Time [s]')
ylabel('Changes of Power [W]')
legend('Constant Vw', 'dVw = 0.1 m/s')

% 3.2 a b c
Te = Te_opt; % Because of Vw = 10 m/s
lambda = (0:1e-2:20);
Ct = c0 + c1.*lambda + c2*lambda.^2;

simout_32 = sim('Task3_2.slx');
Ttur_32 = simout_32.Ttur;
t_32 = simout_32.tout;
figure(5)
plot(t_32, Ttur_32, t_32, Te*ones(1, length(t_32)));
xlabel('Time [s]')
ylabel('Torque [Nm]')
legend('Ttur', 'Te')

% 3.3 a
% Same calculation as Assignment 1
Te = 0:1e3:Te_opt;
```

```
i_sq_opt = zeros(1, length(Te));
i_sd_opt = zeros(1, length(Te));
for i=2:length(Te)
    for iteration=1:10
        f = i_sq_opt(i)^4 + Phi_m * Te(i) * i_sq_opt(i) /
(3/2*p/2*(Lsd-Lsq)^2) - (Te(i) / (3/2*p/2*(Lsd-Lsq)))^2;
        df = 4*i_sq_opt(i)^3 + Phi_m*Te(i) / (3/2*p/2*(Lsd-Lsq)^2);
        di = f / df;
        i_sq_opt(i) = i_sq_opt(i) - di;
    end
    i_sd_opt(i) = -Te(i) / (3/2 * p/2 * (Lsd-Lsq) * i_sq_opt(i)) ...
                  + Phi_m / (Lsd-Lsq);
end
i_sq_opt = i_sq_opt(end);
i_sd_opt = i_sd_opt(end);
% 3.3 b
simout_33 = sim('Task3_3.slx');
we_33 = simout_33.we;
t_33 = simout_33.tout;
figure(6)
plot(t_33, we_33)
xlabel('Time [s]')
ylabel('Electrical Angular Velocity [Nm]')
legend('We')

%% Assignment 4

% 4.1
tau_le = tau_w;
tau_lg = tau_z;
tau_pl = 0.05 * tau_w;
K = p * tau_z / (tau_pl * 2*we * tau_w);
Pel_err = -50e3; % Change in reference power [W]

simout_41 = sim('Task4_1.slx');
Pel_err_plot = simout_41.Pel_err;
t_41 = simout_41.tout;
figure(7)
plot(t_41, Pel_err_plot)
xlabel('Time [s]')
ylabel('Power Change [kNm]')
legend('Pel')
% 4.3
Te = 0:1e3:Te_rated;
simout_43 = sim('Task4_3.slx');
Te_43 = simout_43.Te;
Pe_43 = simout_43.Pe;
we_43 = simout_43.we;
t_43 = simout_43.tout;

figure("name", "Plot of Te, Pel and we");
subplot(311)
plot(t_43, Pe_43)
xlabel('Time [s]')
ylabel('Electric Power [W]')
subplot(312)
```

```
plot(t_43, we_43)
xlabel('Time [s]')
ylabel('Electric Angular Velocity [Nm]')
subplot(313)
plot(t_43, Te_43)
xlabel('Time [s]')
ylabel('Electric Torque [Nm]')
% 4.4
% Generate wind noise
t_44 = (1:300);
rng('default')
noise = -0.5 + 1*rand(1, length(t_44));
Vw = Vw_3 + noise;
Vw_ts = timeseries(Vw', t_44);
save("wind_speed_fluct.mat", 'Vw_ts', '-v7.3')
simout_44 = sim('Task4_4.slx');

t_44 = simout_44.tout;
Te_44 = simout_44.Te;
isq_44 = simout_44.isq;
isd_44 = simout_44.isd;
Pel_44 = simout_44.Pel;
Cp_44 = simout_44.Cp;
we_44 = simout_44.we;

figure("name", "Plot of Te, isd and isq");
subplot(311)
plot(t_44, Te_44)
subtitle("Torque over time")
xlabel('Time [s]')
ylabel('Torque [Nm]')
subplot(312)
plot(t_44, isq_44)
subtitle("Stator Current in q-Frame over time")
xlabel('t [s]')
ylabel('isq [A]')
subplot(313)
plot(t_44, isd_44)
subtitle("Stator Current in d-Frame over time")
xlabel('t [s]')
ylabel('isd [A]')

figure("name", "Plot of Pe vs we and Cp vs Te");
subplot(211)
plot(we_44, Pel_44)
subtitle("Electrical Power vs El. Angular Velocity")
xlabel('we[rad/s]')
ylabel('Pel [W]')
subplot(212)
plot(Te_44, Cp_44)
subtitle("Power Coefficient vs Torque")
xlabel('Te [Nm]')
ylabel('Cp [s]')

figure("name", "Wind Fluctuation")
plot(Vw)
```

```
title("Wind Fluctuation over Time")
xlabel('Time [s]')
ylabel('Wind Speed [m/s]')

% 4.4 e
% Get minimal and maximal value of Vw and its corresponding Pe.
min_Vw = min(Vw);
max_Vw = max(Vw);
min_Pe = min(Pel_44(200:end));
max_Pe = max(Pel_44(200:end));
min_index = find(Pel_44(200:end) == min_Pe);
max_index = find(Pel_44(200:end) == max_Pe);
min_Cp = Cp_44(min_index);
max_Cp = Cp_44(max_index);

min_Pe_theory = 1 / 2 * pi * rho * r^2 * min_Vw^3 * min_Cp;
max_Pe_theory = 1 / 2 * pi * rho * r^2 * max_Vw^3 * max_Cp;
```