Control and operations of tokamaks
Exercise Session 6 - Kinetic control
Solutions

Lecturer: F. Felici
Instructors: D. Biek, R. Coosemans, S. Marchoni, P. Molina, F. Pastore
EPFL - SPC

February 2023

1 Simulating the plasma 0D energy balance

```
clear
close all
```

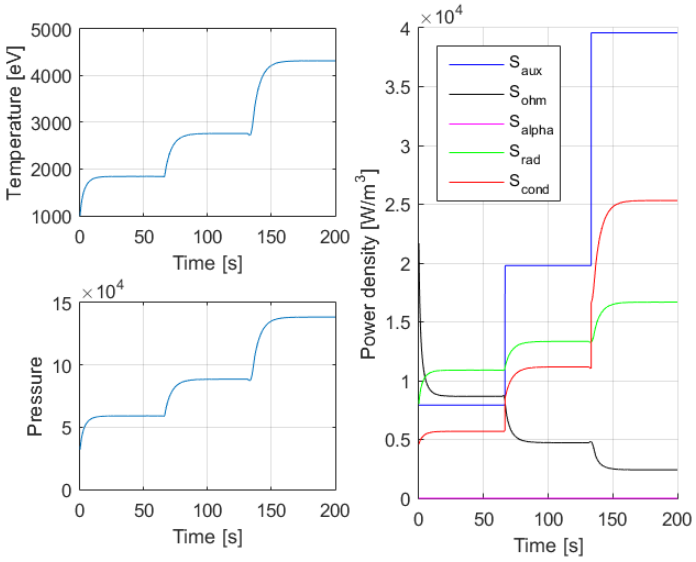
Define constants and wrap in struct

```
p.n = 1E20; % density [#/m^3]
p.R_0 = 8;
p.a = 2;
p.kappa = 2;
p.B_0 = 7;
p.A = 2;
p.Z_eff = 1.5;
p.f_DT = 0;
p.I_p = 15; % plasma current [MA]

% Define time span, input signal and initial condition
t_span = [0 200]; % time span [s]
P_aux = [10E6*ones(1,800) 25E6*ones(1,800) 50E6*ones(1,801)];
% apply 20 MW input power
time = linspace(t_span(1),t_span(end),size(P_aux,2));
% time grid for input power
T_0 = 1E3; % initial temperature [eV]

% Simulate thermal energy balance and compute power density sources and pressure
[~,T] = ode45(@(t,T) thermal_energy_balance(t,T,time,P_aux,p),time,T_0);
S = sources(T,P_aux',p);
```

Plot temperature evolution

```
fig = figure;
```

```
subplot(2,2,1)
plot(time,T)
grid on; xlabel('Time [s]'); ylabel('Temperature [eV]');
subplot(2,2,3)
plot(time,S.pressure)
grid on; xlabel('Time [s]'); ylabel('Pressure');
subplot(2,2,[2 4])
hold on
plot(time,S.S_aux,'b')
plot(time,S.S_ohm,'k')
plot(time,S.S_alpha,'m')
plot(time,S.S_rad,'g')
plot(time,S.S_cond,'r')
legend('S_{aux}','S_{ohm}','S_{alpha}','S_{rad}','S_{cond}','Location','Northwest')
grid on; xlabel('Time [s]'); ylabel('Power density [W/m^3]');
```



Functions used

This is the function thermal_energy_balance.m

```
type thermal_energy_balance
```

```
function dT_eVdt = thermal_energy_balance(t,T,time,P_in,p)

q_e = 1.60217657E-19; % electron charge [coulomb] or Boltzmann constant [J/eV]
P_aux = interp1(time,P_in,t); % Interpolate the input signal (time,u) at time t
S = sources(abs(T),P_aux,p);

dT_eVdt = 1/(3*p.n*q_e)*(S.S_alpha+S.S_ohm+S.S_aux-S.S_rad-S.S_cond);
end
```

This is the function sources.m

```
type sources
```

```
function S = sources(T,P_aux,p)

a = [-21.38, -25.20, -7.101e-2 1.938e-4 4.925e-6 -3.984e-8];
alp = 0.2935;
E_alpha = 4.5E6; % [eV]
q_e = 1.60217657E-19; % electron charge [coulomb] or Boltzmann constant [J/eV]
V = (2*pi*p.R_0)*(pi*p.kappa*p.a^2); % plasma volume [m^3]
epsilon = p.a/p.R_0;

sigmav = 1e-6*exp(a(1)./((T/1000).^alp) + a(2) + a(3)*(T/1000) + ...
         a(4)*(T/1000).^2 + a(5)*(T/1000).^3 + a(6).*(T/1000).^4);

P_alpha = V * p.f_DT/(1+p.f_DT)^2*q_e*E_alpha*p.n^2*sigmav;
P_ohm = (5.6E4/(1-1.31*epsilon^0.5+0.46*epsilon))*...
    ((p.R_0*p.I_p.^2)./(p.a^2*p.kappa*(T/1000).^(3/2)));

tau_e = 0.145*p.I_p^(0.93)*...
              p.R_0^(1.39)*...
              p.a^(0.58)*...
              p.kappa^(0.78)*...
              (p.n/1E20)^(0.41)*...
              p.B_0^(0.15)*...
              p.A^(0.19)*...
              ((P_aux+P_alpha+P_ohm)/1E6).^(-0.69);
pressure = 2*p.n*q_e*T;
%%
S.S_aux = P_aux/V;
S.S_ohm = P_ohm/V;
S.S_alpha = P_alpha/V;
S.S_rad = 5.35E3*p.Z_eff*(p.n/1E20)^2*sqrt(T/1000);
S.S_cond = 3/2*pressure./tau_e;
S.tau_e = tau_e;
S.pressure = pressure;
S.P_neutron = 4*P_alpha;
end
```

2 Control of plasma β

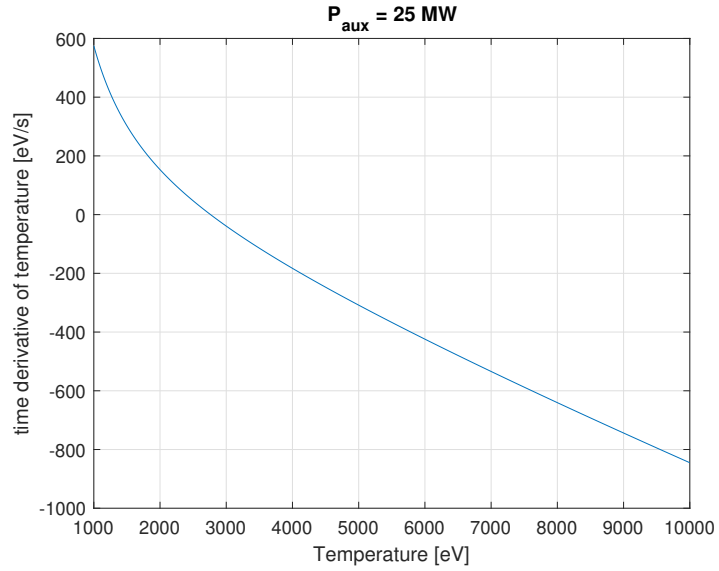
```
clear
close all
```

Define constants and wrap in struct

```
p.n = 1E20; % density [#/m^3]
p.R_0 = 8;
p.a = 2;
p.kappa = 2;
p.B_0 = 7;
p.A = 2;
p.Z_eff = 1.5;
p.f_DT = 0;
p.I_p = 15; % plasma current [MA]
```

Draw plot of $\partial T/\partial t$ vs T

```
T_plot = linspace(1e3,10e3,1001);
dTdt = thermal_energy_balance(0,T_plot,[0 1],[25E6 25E6],p);
figure;
plot(T_plot,dTdt); grid on
title('P_{aux} = 25 MW')
xlabel('Temperature [eV]'); ylabel('time derivative of temperature [eV/s]')
```




```
% Define the operating point
P_0 = 25E6; % [W]
T_0 = 2763; % [eV] read out from steady-state solution at P_aux = 25MW
T = T_0;
P_aux = P_0;
```

Apply linearization to write the equation in the following form:

$$\frac{\partial \delta T}{\partial t} = \frac{1}{3en}\left(\frac{\partial S}{\partial T}\delta T + \frac{\partial S}{\partial P_{aux}}\delta P\right) = K_T \delta T + K_P \delta P_{aux}. \tag{1}$$

Note that we write δT in eV and δP_{aux} in W. This linearisation is performed in the main script by calling

```
% Define the linearized system
[G,KT,KP]=linearise_model(T_0,P_0,p);
```

This calls the function *linearise_model.m*. In this function, the linearisation for the Ohmic power and the auxiliary power are to be completed by the students. Next to calculating the linearisation itself, the linearised equation is also converted into the linearised model of the system $G(s)$ as

$$G(s) = \frac{\delta T}{\delta P_{aux}}(s) = \frac{K_P}{s - K_T}. \tag{2}$$

The construction of this transfer function is also to be added to the script by the students. The completed scripts is shown later under "Functions used".

Next, we check the temperature linearisation by calculating the slope

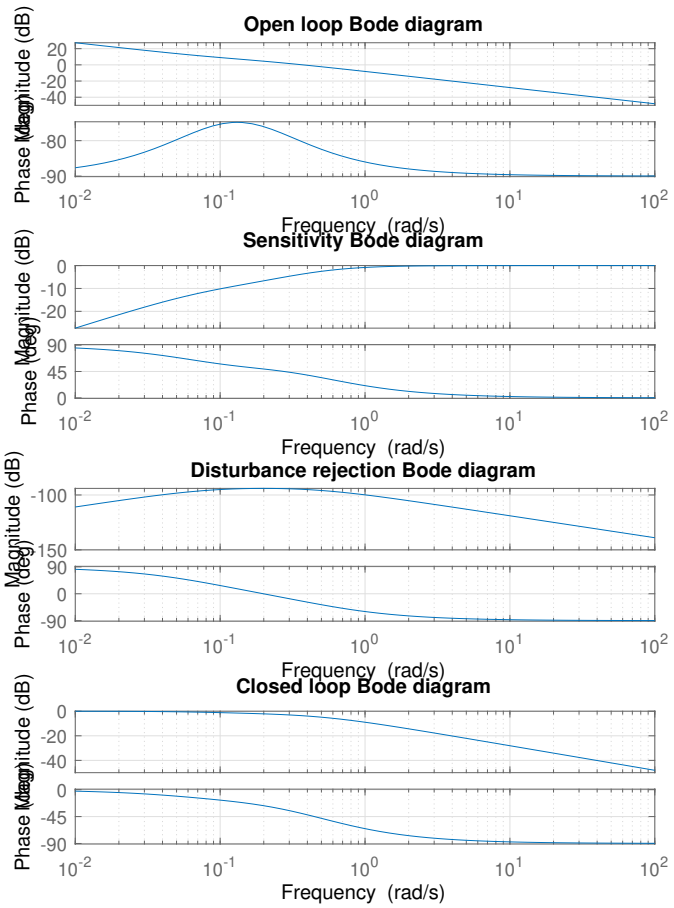
$$\frac{\partial}{\partial T}\left(\partial T/\partial t\right)_{T_0} \approx \frac{\left(\partial T/\partial t\right)_{T_1} - \left(\partial T/\partial t\right)_{T_{-1}}}{T_1 - T_{-1}} = -0.171\mathrm{s}^{-1}. \tag{3}$$

It is understood that T_{-1} and T_1 are temperatures just smaller and just larger than the operating point temperature $T_0 = 2763$eV. This value matches nicely the value of K_T for this operating point. (Something similar could presumably be done for K_P as well, but I haven't done this yet...)

Design linear controller
Even though the controller will serve to control the full nonlinear system, we design it based on the linearised system. Zero steady state error is basically ensured automatically by the integrator. For measurement (output) noise rejection, we look at the magnitude of the transfer function $S = \frac{1}{1+CG}$. The plateau at high frequency is there no matter the exact tuning of the parameters. The transition to this 0dB region is determined by the Ti parameter. To evaluate the disturbance (input noise) rejection, we consider the transfer function $DR = \frac{G}{1+CG}$. In order to get 10dB disturbance rejection at 1Hz, Ti and/or k need to be sufficiently high. At these values required for noise rejection, the closed loop bandwith (magnitude of closed loop transfer function CL) is ensured automatically.
Disclaimer: the controller proposed here fulfills the requirements, but it is not per se completely optimal. Note that we reached the control specification with just a PI controller, i.e. we did not need the derivative part to reach these.

```
% Design linear controller
close all
s = tf('s');
k=3500;
Ti=10;
C=k*(1+s*Ti)/s; % PI feedback block

% compute some transfer functions
OL = G*C;
S = 1/(1+OL);
DR = G/(1+OL);  % disturbance rejection
```

```
CL=feedback(OL,tf(1));    % CL feedback system
figure
subplot(411)
bode(OL,{1E-2,1E2}); grid on
title('Open loop Bode diagram')
subplot(412)
bode(S,{1E-2,1E2}); grid on
title('Sensitivity Bode diagram')
subplot(413)
bode(DR,{1E-2,1E2}); grid on
title('Disturbance rejection Bode diagram')
subplot(414)
bode(CL,{1E-2,1E2}); grid on
title('Closed loop Bode diagram')
display(['dcgain equals ' num2str(dcgain(CL))])
```

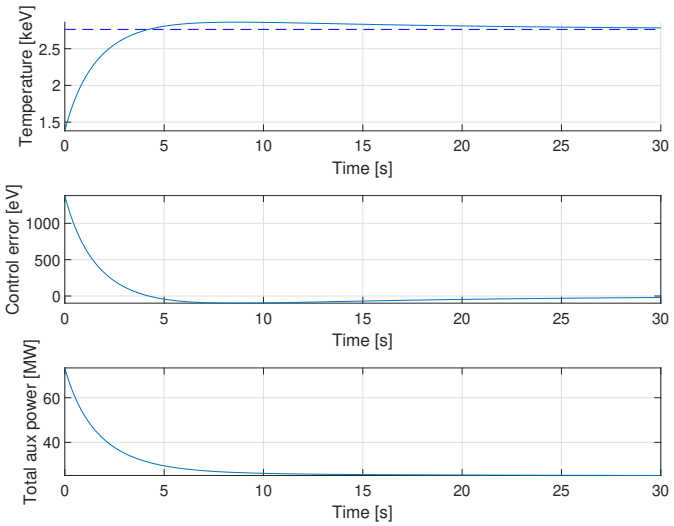


Simulate linear controller on nonlinear plant model

```
% Convert controller to state space form for simulation
[num,den] = tfdata(C);
[A_c,B_c,C_c,D_c] = tf2ss(cell2mat(num),cell2mat(den));
C_ss.A = A_c; C_ss.B = B_c; C_ss.C = C_c; C_ss.D = D_c;
C_ss.T_0 = T_0; C_ss.P_0 = P_0;

t_span = [0 30]; % time span [s]
time = linspace(t_span(1),t_span(end),500); % time grid for input power
T_init = 0.5*T_0;
[~,x] = ode45(@(t,x) CL_thermal_energy_balance(t,x,time,C_ss,p),time,[T_init 0 0]);
T = x(:,1);
z = x(:,2);
e = C_ss.T_0 - T; % control error: setpoint is T_0
```

Plot temperature and controller state

```
fig = figure;
subplot(211)
hold on
plot(time,T/1E3);
plot([time(1) time(end)],[T_0 T_0]/1E3,'b--'); grid on
grid on
xlabel('Time [s]'); ylabel('Temperature [keV]')
subplot(212)
plot(time,e); grid on
xlabel('Time [s]'); ylabel('Control error [eV]')
set(fig,'PaperPositionMode','auto');
```



Functions used

This is the function linearise_model.m

```
type linearise model

function [G,KT,KP,H,S]=linearise_model(T0,P0,p)

%% linearise sources

S=sources(T0,P0,p);

Salpha=S.S_alpha(end);
Sohm=S.S_ohm(end);
Saux=S.S_aux(end);
Srad=S.S_rad(end);
Scond=S.S_cond(end);
SH=Salpha+Sohm+Saux;    % total heating power

% redefine some constants
q_e = 1.60217657E-19;
a = [-21.38, -25.20, -7.101e-2 1.938e-4 4.925e-6 -3.984e-8];
alp = 0.2935;
Sexp = -alp*a(1)./((T0/1000).^alp) + a(3)*(T0/1000) + ...
         2*a(4)*(T0/1000).^2 + 3*a(5)*(T0/1000).^3 + 4*a(6).*(T0/1000).^4;
V = (2*pi*p.R_0)*(pi*p.kappa*p.a^2);

% construct KT
KTohm=-3/2*Sohm/(3*p.n*q_e*T0);
KTrad=-Srad/2/(3*p.n*q_e*T0);
KTcond=(-1+3/2*0.69*Sohm/SH-0.69*Sexp*Salpha/SH)*Scond/(3*p.n*q_e*T0);
KTalpha=Sexp*Salpha/(3*p.n*q_e*T0);
KT=KTohm+KTrad+KTcond+KTalpha;

% contruct KP
KPaux=1/(3*p.n*q_e*V);
KPcond=-0.69*Scond/SH/(3*p.n*q_e*V);
KP=KPaux+KPcond;

%% construct transfer functions

s = tf('s');
G=KP/(s-KT);  % delta(Paux) to delta(T) transfer function

H=4*V*Sexp*Salpha/T0;    % delta(T) to delta(neutron power) transfer function
end
```

This is the function CL_thermal_energy_balance.m

```
type CL_thermal_energy_balance

function dxdt = CL_thermal_energy_balance(t,x,time,C_ss,p)
```

```
q_e = 1.60217657E-19; % electron charge [coulomb] or Boltzmann constant [J/eV]
V = 2*pi*p.kappa*p.R_0*p.a^2; % plasma volume [m^3]
T = x(1); % plant state
z = x(2:end); % controller state

%% Linear controller
e = C_ss.T_0 - T; % control error: setpoint is T_0
dzdt =   C_ss.A*z + C_ss.B*e;
dP_aux = C_ss.C*z + C_ss.D*e;
S_aux = (dP_aux+C_ss.P_0)/V;

%% Nonlinear plant
S = sources(abs(T),dP_aux+C_ss.P_0,p);
dT_eVdt = 1/(3*p.n*q_e)*(S.S_alpha+S.S_ohm+S_aux-S.S_rad-S.S_cond);

%% Total system time derivative
dxdt = [dT_eVdt; dzdt];
end
```

3 Burn control

```
clear
close all
```

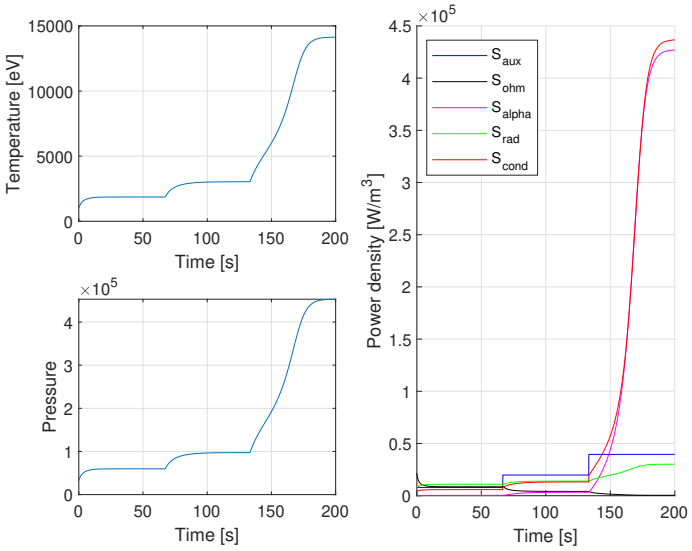
Define constants and wrap in struct

```
p.n = 1E20; % density [#/m^3]
p.R_0 = 8;
p.a = 2;
p.kappa = 2;
p.B_0 = 7;
p.A = 2;
p.Z_eff = 1.5;
p.f_DT = 1;
p.I_p = 15; % plasma current [MA]
```

Apply input power ramp for this case with a nonzero DT-fraction

```
% Define time span, input signal and initial condition
t_span = [0 200]; % time span [s]
P_aux = [10E6*ones(1,800) 25E6*ones(1,800) 50E6*ones(1,801)]; % apply 20 MW input power
time = linspace(t_span(1),t_span(end),size(P_aux,2)); % time grid for input power
T_0 = 1E3; % initial temperature [eV]

% Simulate thermal energy balance and compute power density sources and pressure
[~,T] = ode45(@(t,T) thermal_energy_balance(t,T,time,P_aux,p),time,T_0);
S = sources(T,P_aux',p);
```

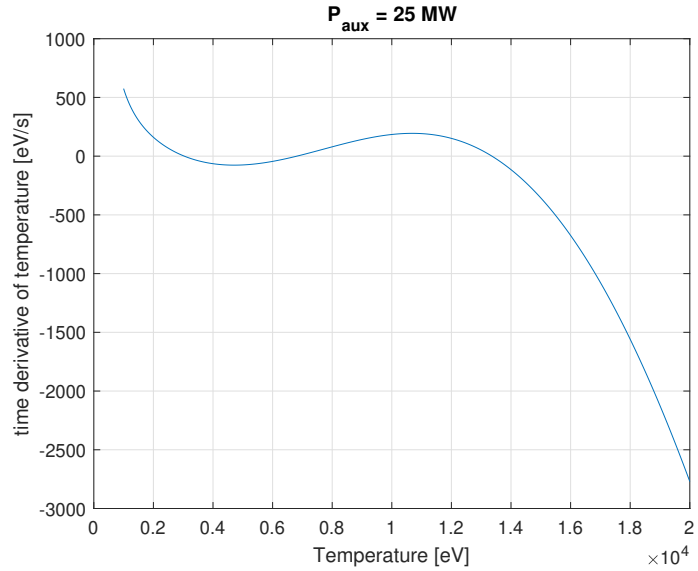
```
% Plot temperature evolution
fig = figure;
subplot(2,2,1)
plot(time,T)
grid on; xlabel('Time [s]'); ylabel('Temperature [eV]');
subplot(2,2,3)
plot(time,S.pressure)
grid on; xlabel('Time [s]'); ylabel('Pressure');
subplot(2,2,[2 4])
hold on
plot(time,S.S_aux,'b')
plot(time,S.S_ohm,'k')
plot(time,S.S_alpha,'m')
plot(time,S.S_rad,'g')
plot(time,S.S_cond,'r')
legend('S_{aux}','S_{ohm}','S_{alpha}','S_{rad}','S_{cond}','Location','Northwest')
grid on; xlabel('Time [s]'); ylabel('Power density [W/m^3]');
set(fig,'PaperPositionMode','auto');
print -depsc exercise_5_1_simulation
```



Draw plot of $\partial T/\partial t$ vs T
It is observed that the middle equilibrium point is unstable. At this point, a slight increase of the temperature will lead to a positive dT/dt, causing the temperature to increase further etc.

```
T_plot = linspace(1e3,20e3,1001);
dTdt = thermal_energy_balance(0,T_plot,[0 1],[25E6 25E6],p);
fig = figure;
plot(T_plot,dTdt); grid on
title('P_{aux} = 25 MW')
xlabel('Temperature [eV]'); ylabel('time derivative of temperature [eV/s]')
```

```
set(fig,'PaperPositionMode','auto');
print -depsc exercise_6_3_phase_plot
```



Linearize around each stationary point using the function *linearise_model.m* the we elaborated before. We can again check the match between these results and the result we would get from finite differences on the previous plot.
Since we now want to control the reactor using a reference signal for the neutron power and in response to a measurement of the neutron power, we derive the transfer function from the auxiliary power to the neutron power:

$$\frac{\delta P_n}{\delta P_{aux}}(s) = \frac{\delta P_n}{\delta T}(s)\frac{\delta T}{\delta P_{aux}}(s) = H(s)G(s). \tag{4}$$

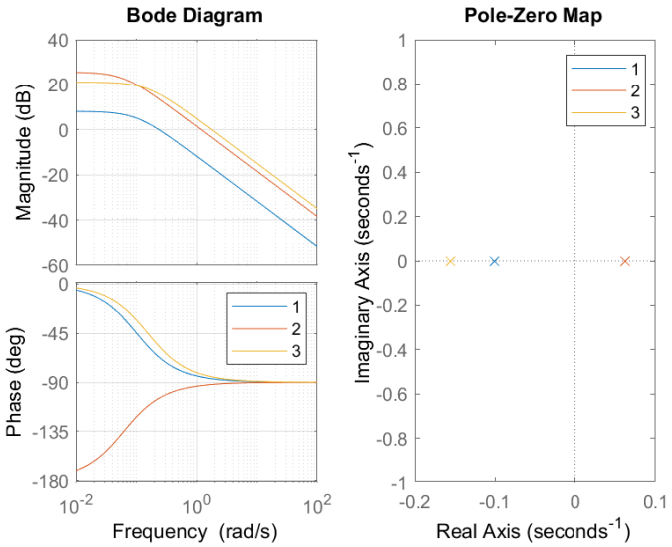
Here we defined the additional transfer function $H(s)$. For our linearised system, this turns out to be just a constant which is also already filled out in the completed version of the file *linearise_model.m*. Furthermore, we calculate the set point P_neutron of the neutron power corresponding to each of the operating points.

```
clear T_0
P_0 = 25E6; % [W]

% These are the equilibrium temperatures, from low to high
T_0{1} = 3038; % [eV] read out from phase plot (P_aux = 25MW)
T_0{2} = 6800; % [eV] read out from phase plot (P_aux = 25MW)
T_0{3} = 1.335E4; % [eV] read out from phase plot (P_aux = 25MW)

% Linearize the output relation at each stationary point
for i = 1:length(T_0)
    [G{i},A{i},B{i},H{i},S]=linearise_model(T_0{i},P_0,p);
    P_neutron{i} = 4*S.S_alpha*V;
end
```

```
% check with phase plot results (P_aux = 25MW)
% A{1} = -0.09; % stable
% A{2} = 0.067; % unstable
% A{3} = -0.08; % stable
```

The linearised system we are thus designing a controller for is H(s)G(s). As can be seen from the pole-zero map, the second operating point is indeed open loop unstable (pole in the RHP).

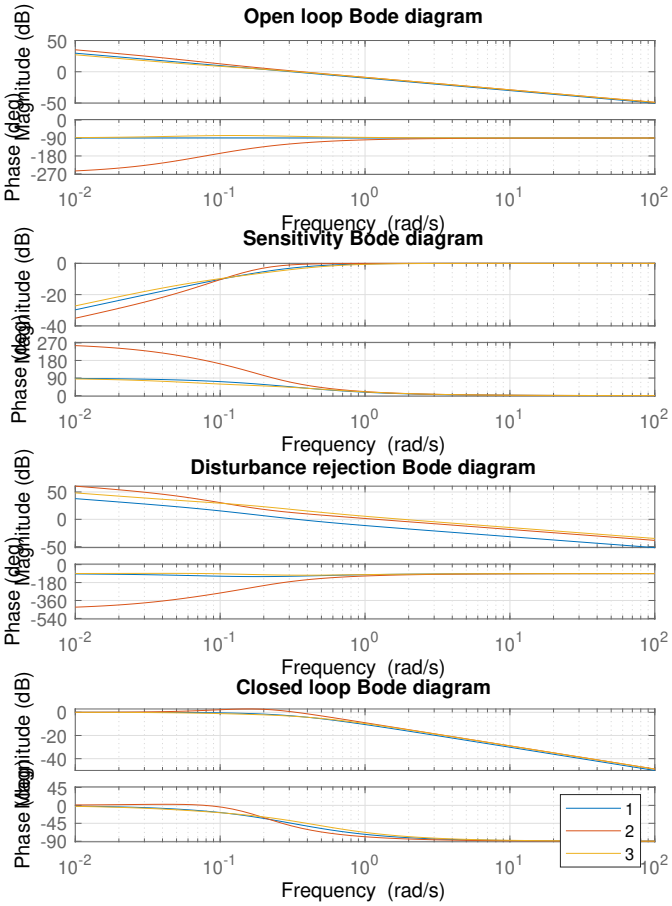
```
figure
subplot(121)
bode(H{1}*G{1},H{2}*G{2},H{3}*G{3},{1E-2,1E2}); grid on
legend('1','2','3')
subplot(122)
pzmap(H{1}*G{1},H{2}*G{2},H{3}*G{3})
legend('1','2','3')
```

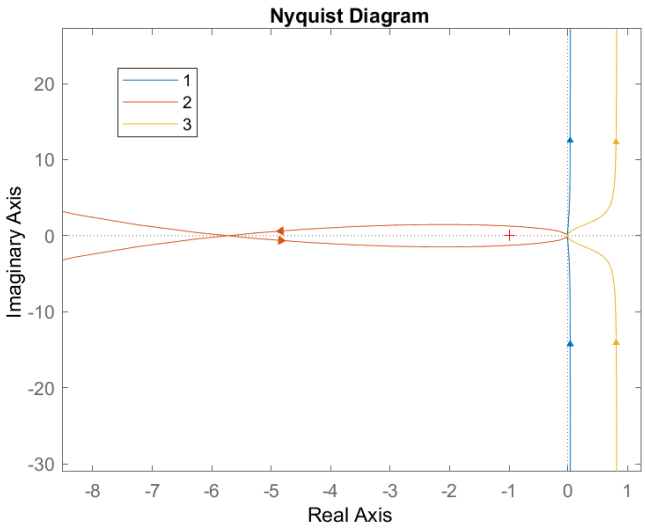


Design linear controllers in the vicinity of each stationary point
All the comments made about the controller design for the beta-control still hold identically for the burning plasma case considered here. In addition, we can remark that the unstable second operating point is stabilised by closing the control loop, at the condition that the gain is sufficiently high. This can be seen in the Nyquist plot below (1 counter-clockwise encirclement of (-1,0) required to stabilise the unstable pole of the open loop system).
Disclaimer: the controllers proposed here fulfill the requirements, but are not per se completely optimal again. We again observe that a PI controller suffices to attain the requirements.

```
close all
clear S
```

```
s = tf('s');
C{1}=0.12*(1+s*10)/s;
C{2}=0.03*(1+s*10)/s; % note that stability has to be ensured here
C{3}=0.02*(1+s*10)/s;

for i = 1:3
    OL{i} = H{i}*G{i}*C{i};
    S{i} = 1/(1+OL{i});
    DR{i} = H{i}*G{i}/S{i};  % disturbance rejection
    CL{i}=feedback(OL{i},tf(1));
end
figure
subplot(411)
bode(OL{1},OL{2},OL{3},{1E-2,1E2}); grid on
title('Open loop Bode diagram')
subplot(412)
bode(S{1},S{2},S{3},{1E-2,1E2}); grid on
title('Sensitivity Bode diagram')
subplot(413)
bode(DR{1},DR{2},DR{3},{1E-2,1E2}); grid on
title('Disturbance rejection Bode diagram')
subplot(414)
bode(CL{1},CL{2},CL{3},{1E-2,1E2}); grid on
title('Closed loop Bode diagram')
legend('1','2','3')
figure
nyquist(OL{1},OL{2},OL{3});
legend('1','2','3')
display(['dcgain operating point 1 equals ' num2str(dcgain(CL{1}))])
display(['dcgain operating point 2 equals ' num2str(dcgain(CL{2}))])
display(['dcgain operating point 3 equals ' num2str(dcgain(CL{3}))])
```


Open loop Bode diagram
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
Sensitivity Bode diagram
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
Disturbance rejection Bode diagram
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
Closed loop Bode diagram
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
1
2
3



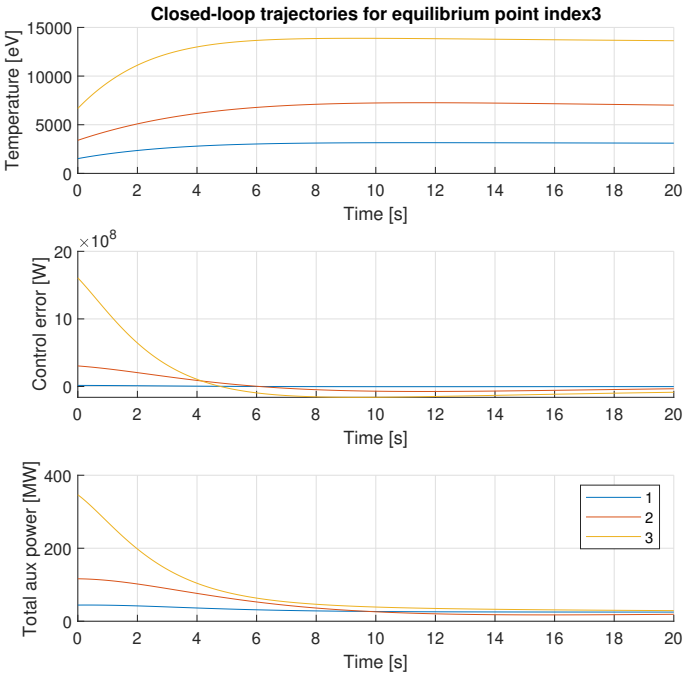
Simulate closed-loop system with three controllers close to each stationary point. Note that we now use a different function to simulate the closed loop system response since the control signal and the response are now the neutron power instead of the temperature (*CL_neutral_power.m* instead of *CL_thermal_energy_balance* before).

```
clear T e z
for i = 1:3
        [num,den] = tfdata(C{i});
    [A_c,B_c,C_c,D_c] = tf2ss(cell2mat(num),cell2mat(den));
    C_ss.A = A_c; C_ss.B = B_c; C_ss.C = C_c; C_ss.D = D_c;
    C_ss.P_0 = P_0; C_ss.P_neutron = P_neutron{i};

    %% Simulate linear controller on nonlinear plant model
    t_span = [0 20]; % time span [s]
    time = linspace(t_span(1),t_span(end),500); % time grid for input power
    T_init = 0.5*T_0{i};
    [~,x] = ode45(@(t,x) CL_neutral_power(t,x,time,C_ss,p), ...
        time,[T_init zeros(1,size(A_c,1))]);
    T{i} = x(:,1);
    z{i} = x(:,2:end);
    Salpha = sources(abs(T{i}),0,p);   % reconstruct neutron power
    e{i} = C_ss.P_neutron - 4*Salpha.S_alpha*V; % control error: setpoint is P_neutral;
    dP_aux{i} = (C_ss.C*x(:,2:end)')' + C_ss.D*(e{i});
    P_aux{i} = dP_aux{i}+C_ss.P_0;
end
```

Plot closed-loop performance

```
fig=figure;
subplot(311)
```

```
hold on
plot(time,T{1}); grid on
plot(time,T{2});
plot(time,T{3});
xlabel('Time [s]'); ylabel('Temperature [eV]')
title(strcat('Closed-loop trajectories for equilibrium point index',num2str(i)));
subplot(312)
hold on
plot(time,e{1}); grid on
plot(time,e{2});
plot(time,e{3});
xlabel('Time [s]'); ylabel('Control error [W]')
subplot(313)
hold on
plot(time,P_aux{1}/1E6);
plot(time,P_aux{2}/1E6);
plot(time,P_aux{3}/1E6);
legend('1','2','3')
grid on; xlabel('Time [s]'); ylabel('Total aux power [MW]')
set(fig,'PaperPositionMode','auto');
```



Functions used

This is the function CL_neutral_power.m

```
type CL_neutral_power

function  dxdt = CL_neutral_power(t,x,time,C_ss,p)
q_e = 1.60217657E-19; % electron charge [coulomb]
V = 2*pi^2*p.kappa*p.R_0*p.a^2; % plasma volume [m^3]
T = x(1); % plant state
z = x(2:end); % controller state

% Compute neutral power assumed to be plant output in the current state
Salpha = sources(abs(T),0,p);
P_neutron=4*Salpha.S_alpha*V;

%% Linear controller
e = C_ss.P_neutron - P_neutron; % control error: setpoint is P_neutron_0
dzdt =   C_ss.A*z + C_ss.B*e;
dP_aux = C_ss.C*z + C_ss.D*e;
S_aux = (dP_aux+C_ss.P_0)/V;

%% Nonlinear plant
S = sources(abs(T),dP_aux+C_ss.P_0,p);
dT_eVdt = 1/(3*p.n*q_e)*(S.S_alpha+S.S_ohm+S_aux-S.S_rad-S.S_cond);

%% Total system time derivative
dxdt = [dT_eVdt; dzdt];
end
```