```
close all; %needed to close previous plots of the SLR

%% estimated parameters (from LAB 0)
est.Beq = 1.221e-6;
est.tausf = 5.687e-3;
est.Jeq = 5.643e-7;

Req = mot.R + sens.curr.Rs;
Jeq = (mot.J + mld.Jh/gbox.N^2);

%% state-space model - preliminary definitions

%system matrices calculation
ssc.A_prime = [0 0 1 0; 0 0 0 1; 0 mld.k/(gbox.N^2*Jeq) -1/Jeq*(est.Beq+mot.↙
Kt*mot.Ke/Req) 0; 0 -mld.k/mld.Jb-mld.k/(Jeq*gbox.N^2) -mld.Bb/mld.Jb+1/Jeq*(est.↙
Beq+mot.Kt*mot.Ke/Req) -mld.Bb/mld.Jb];
ssc.B_prime = [0 0 mot.Kt*drv.dcgain/(gbox.N*Jeq*Req) -mot.Kt*drv.dcgain/(gbox.↙
N*Jeq*Req)]';
ssc.Bd_prime = [0 0 -1/(gbox.N^2*Jeq) 1/(gbox.N^2*Jeq)]';
ssc.C_prime = [1 0 0 0];
ssc.D_prime = 0;

%LTI objects to be used in the following
ssc.sysG = ss(ssc.A_prime, ssc.B_prime, ssc.C_prime, 0);
ssc.sysGp = ss(-ssc.A_prime, -ssc.B_prime, ssc.C_prime, 0);

%controller gains (Nx, Nu)
gains = inv([ssc.A_prime ssc.B_prime; ssc.C_prime ssc.D_prime])*[0; 0; 0; 0; 1];
ssc.Nx = gains(1:4, 1); %state feedforward gain
ssc.Nu = gains(5, 1); %input feedforward gain

clear gains;

%recall that the third gain, for alternative structure (Nr), must be recalculated
%each time the state feedback matrix is changed
%so it is computed separately every time in each section

% extended system matrices (for the robust tracking parts)
ssc.Ae = [0 ssc.C_prime; zeros(4,1) ssc.A_prime];
ssc.Be = [0; ssc.B_prime];
ssc.Ce = [1 0 0 0 0];

%extended systems (LTI objects) defintions
ssc.Sigma_e = ss(ssc.Ae, ssc.Be, ssc.Ce, 0);
ssc.Sigma_ep = ss(-ssc.Ae, -ssc.Be, ssc.Ce, 0);

%% Specifications
ssc.ts5 = 0.5;
ssc.Mp = 30/100;

%Specifications translation
ssc.delta = (log(1/ssc.Mp)/sqrt(pi^2 + (log(1/ssc.Mp))^2)); %see eqn (8)↙
assignment 3
ssc.wn = 3/(ssc.delta*ssc.ts5); %see eqn (8) assignment 3

%constraints for the admissible complex plane area (gray area in Assignment Fig. 4↙
and 5)
ssc.sigma = -3/ssc.ts5;
ssc.phi = atan((sqrt(1-ssc.delta^2)/ssc.delta));


%% LQR method (point 1 of Assignment)
```

```matlab

%symmetric root locus plot
figure;
rlocus(ssc.sysG*ssc.sysGp);
grid on;

%symmetric root locus data
k = (0:10:1e5);
r = rlocus(ssc.sysG*ssc.sysGp, k);
%k is an array that stores the values of the gain
%r is a matrix in which in every row I have a pole and in every column the
%value of this pole for the corresponding gain value

%This (r, k) is the standard MATLAB notation. Don't get confused with our notation
%for r.

%finding the correct gain r to use in lqr
r3 = find((real(r(3,:)) < ssc.sigma) & (abs(imag(r(3,:))./real(r(3,:))) < tan(ssc.phi)));
r6 = find((real(r(6,:)) < ssc.sigma) & (abs(imag(r(6,:))./real(r(6,:))) < tan(ssc.phi)));

r2 = find((real(r(2,:)) < ssc.sigma) & (abs(imag(r(2,:))./real(r(2,:))) < tan(ssc.phi)));
r4 = find((real(r(4,:)) < ssc.sigma) & (abs(imag(r(4,:))./real(r(4,:))) < tan(ssc.phi)));

%NOTE: I first executed r = rlocus(ssc.sysG*ssc.sysGp, k) and I looked for the
%      poles I needed. They were in rows 3 and 6. So that's why 3 and 6
%      are used above.

%the correct gain to use is 1 over the minimum element that r3 and r6 have in
%common
gain_idx = intersect(r3, r6);
gain = k(gain_idx(1));
ssc.nominal.r = 1/gain;

%for allocation of the eigs in a specific part of the complex plane
% (below -alpha = -1/tau)
tau = 0.299/5;
ssc.sysGa = ss(ssc.A_prime+1/tau*eye(4), ssc.B_prime, ssc.C_prime, 0);

%state feedback matrix calculation with lqr method
ssc.nominal.K = lqr(ssc.sysGa, 1.1*ssc.C_prime'*ssc.C_prime, 0.199*ssc.nominal.r);

%third gain (for alternative structure)
ssc.nominal.Nr = ssc.Nu + ssc.nominal.K*ssc.Nx;

%clearing temporary variables used in this section
%clear gain_idx gain;
%clear r3 r6;
%clear r k;

%TO USE THIS IN THE SIMULINK MODEL:
% substitute in State_space_LQR.slx
% - the state feedback matrix gain with ssc.nominal.K
% - the Nr gain with ssc.nominal.Nr

%% LQR method (Bryson's rule - point 3 of Assignment)

ssc.bryson.Q = diag([1/(0.3*50*pi/180)^2 1/(pi/36)^2 0 0]);
ssc.bryson.R = 1/10^2;
```

```matlab

%state feedback matrix calculation
ssc.bryson.K = lqr(ssc.sysG, ssc.bryson.Q, ssc.bryson.R);

%controller gain calculation
ssc.bryson.Nr = ssc.Nu + ssc.bryson.K*ssc.Nx;

%TO USE THIS IN THE SIMULINK MODEL:
% substitute in State_space_LQR.slx
% - the state feedback matrix gain with ssc.bryson.K
% - the Nr gain with ssc.bryson.Nr

%% Robust tracking & LQR method (point 5 of Assignment)
figure;
rlocus(ssc.Sigma_e*ssc.Sigma_ep);
grid on;

%symmetric root locus data
k = (0:10:1e5);
r = rlocus(ssc.Sigma_e*ssc.Sigma_ep, k);

%finding the correct gain r to use in lqr
r6 = find((real(r(6,:)) < ssc.sigma) & (abs(imag(r(6,:))./real(r(6,:))) < tan(ssc.phi)));
r7 = find((real(r(7,:)) < ssc.sigma) & (abs(imag(r(7,:))./real(r(7,:))) < tan(ssc.phi)));

%NOTE: Also in this case, I first executed r = rlocus(ssc.Sigma_e*ssc.Sigma_ep, k)
%      and I looked for the poles I needed. They were in rows 6 and 7.

%the correct gain to use is 1 over the minimum element that r6 and r7 have in
%common
gain_idx = intersect(r6, r7);
gain = k(gain_idx(1));
ssc.rt.r = 1/gain;

%extended state feedback matrix calculation with lqr method
ssc.rt.Ke = lqr(ssc.Sigma_e, ssc.Ce'*ssc.Ce, ssc.rt.r);

%state feedback marix and integral gain
ssc.rt.K  = ssc.rt.Ke(2:5);
ssc.rt.Ki = ssc.rt.Ke(1);

%controller gain calculation
ssc.rt.Nr = ssc.Nu + ssc.rt.K*ssc.Nx;

%clearing temporary variables used in this section
clear gain_idx gain;
clear r6 r7;
clear r k;

%TO USE THIS IN THE SIMULINK MODEL:
% substitute in State_space_robust_tracking_LQR.slx
% - the state feedback matrix gain with ssc.rt.K
% - the integral gain with ssc.rt.Ki
% - the Nr gain with ssc.rt.Nr

%% Robust tracking & LQR method with Bryson's rule (point 7 of Assignment)

ssc.bryson_rt.q11 = 1e-2; %weight associated with integrator state
ssc.bryson_rt.Q = diag([ssc.bryson_rt.q11 1/(0.3*50*pi/180)^2 1/(pi/36)^2 0 0]);
ssc.bryson_rt.R = 1/10^2;
```

```matlab

%extended state feedback matrix calculation
ssc.bryson_rt.Ke = lqr(ssc.Sigma_e, ssc.bryson_rt.Q, ssc.bryson_rt.R);

%state feedback matrix and integral gain
ssc.bryson_rt.K  = ssc.bryson_rt.Ke(2:5);
ssc.bryson_rt.Ki = ssc.bryson_rt.Ke(1);

%controller gain calculation
ssc.bryson_rt.Nr = ssc.Nu + ssc.bryson_rt.K*ssc.Nx;

%TO USE THIS IN THE SIMULINK MODEL:
% substitute in State_space_robust_tracking_LQR.slx
% - the state feedback matrix gain with ssc.bryson_rt.K
% - the integral gain with ssc.bryson_rt.Ki
% - the Nr gain with ssc.bryson_rt.Nr

%% FS-LQR (point 9 of Assignment)

%finding the angular frequency w0 of the pair of complex conjugate eigs
%The angular frequency of a complex conjugate pair p = a +/- ib is
%w0 = sqrt(a^2 + b^2) = abs(p)
polesG = pole(ssc.sysG);
p = polesG(2);
ssc.fslqr.w0 = abs(p);
%NOTE:  I previously looked at the poles and I saw that I needed the one in
%       position 2

%parameters for the matrix Q
ssc.fslqr.q22 = 0.01; %possible choices: 0.01, 1, 100 (see eqn. (29))
ssc.fslqr.theta_hbar = 5*pi/180;

%parameter for the matrix R
ssc.fslqr.r = 1/10^2; %see eqn. (20)-(21)

%system Sigma_Q'matrices (see eqn. (24) Assignment)
ssc.fslqr.Aqp = [0 1; -ssc.fslqr.w0^2 0];
ssc.fslqr.Bqp = [0; 1];
ssc.fslqr.Cqp = [sqrt(ssc.fslqr.q22)*ssc.fslqr.w0^2 0];
ssc.fslqr.Dqp = 0;

%system Sigma_Q matrices (see eqn. (25) Assignment)
ssc.fslqr.Aq = ssc.fslqr.Aqp;
ssc.fslqr.Bq = [zeros(2, 1) ssc.fslqr.Bqp zeros(2,2)];
ssc.fslqr.Cq = [zeros(1, 2); ssc.fslqr.Cqp; zeros(2,2)];
ssc.fslqr.Dq = diag([1/ssc.fslqr.theta_hbar ssc.fslqr.Dqp 0 0]);

%augmented-state model SigmaA (see eqn. (26) Assignment)
ssc.fslqr.AA = [ssc.A_prime zeros(4,2); ssc.fslqr.Bq ssc.fslqr.Aq];
ssc.fslqr.BA = [ssc.B_prime; zeros(2, 1)];
ssc.fslqr.CA = [ssc.C_prime zeros(1, 2)];
ssc.fslqr.DA = 0;

%cost matrices for the new cost function (see eqn. (27) Assignment)
ssc.fslqr.QA = [ssc.fslqr.Dq'*ssc.fslqr.Dq ssc.fslqr.Dq'*ssc.fslqr.Cq; ssc.fslqr.↙
Cq'*ssc.fslqr.Dq ssc.fslqr.Cq'*ssc.fslqr.Cq];
ssc.fslqr.RA = ssc.fslqr.r;

%state feedback matrix calculation
ssc.fslqr.K = lqr(ssc.fslqr.AA, ssc.fslqr.BA, ssc.fslqr.QA, ssc.fslqr.RA);

%controller gains calculation (see eqn. (28) Assignment)
```

```matlab
gains = inv([ssc.fslqr.AA ssc.fslqr.BA; ssc.fslqr.CA ssc.fslqr.DA])*[zeros(6,1);↙
1];
ssc.fslqr.NxA = gains(1:6, 1); %state feedforward gain
ssc.fslqr.Nu = gains(7, 1); %input feedforward gain

%clearing temporary variables used in this section
clear polesG p gains;

%% FS-LQR & robust tracking (point 11 of Assignment)

%further extended system (see eqn. (30) Assignment)
ssc.fslqr_rt.Ae = [0 ssc.fslqr.CA; zeros(6,1) ssc.fslqr.AA];
ssc.fslqr_rt.Be = [0; ssc.fslqr.BA];
ssc.fslqr_rt.Ce = [0 ssc.fslqr.CA];

ssc.fslqr_rt.Sigma_e = ss(ssc.fslqr_rt.Ae, ssc.fslqr_rt.Be, ssc.fslqr_rt.Ce, 0);

%extended cost matrix Q (see eqn. (31) Assignment)
ssc.fslqr_rt.qI = 10; %first trial, see point 12 Assignment
ssc.fslqr_rt.Qe = [ssc.fslqr_rt.qI zeros(1, 6); zeros(6,1) ssc.fslqr.QA];

%extended cost matrix R (see eqn. (31) Assignment)
ssc.fslqr_rt.Re = ssc.fslqr.RA;

%extended state feedback matrix
ssc.fslqr_rt.Ke = lqr(ssc.fslqr_rt.Sigma_e, ssc.fslqr_rt.Qe, ssc.fslqr_rt.Re);

%state feedback marix and integral gain
ssc.fslqr_rt.K  = ssc.fslqr_rt.Ke(2:7);
ssc.fslqr_rt.Ki = ssc.fslqr_rt.Ke(1);

%% Real derivative transfer function calculation (High-pass Filter)
%(Continuous)
hpf.wc = 2*pi*50;
hpf.delta = 1/sqrt(2);

hpf.num = [hpf.wc^2 0];
hpf.den = [1 2*hpf.delta*hpf.wc hpf.wc^2];
%hpf.Tf = tf(hpf.num, hpf.den);

%(Discrete)
hpfd.num = [1 0 0 0 0 0 0 0 0 0 -1];
hpfd.den = [10*1e-3 0 0 0 0 0 0 0 0 0 0];
%hpfd.Tf = tf(hpfd.num, hpfd.den, 1e-3);

%% Anti-windup parameters for robust tracking
% (not requested, but maybe it can be useful - e.g. for the challenge)
ssc.Tw = ssc.ts5/5;
ssc.Kw = 1/(ssc.Tw);
```