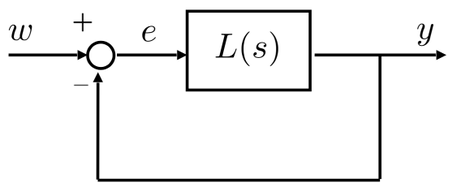
Examples and Applications of Root Loci

Table of Contents

Introduction



With this live script, we want to illustrate how one can draw a **root locus** in MATLAB and also determine some characteristic points, such as the **centroid** of the **asymptotes**, the slope **angles** of the asymptotes, and the **singular points** of the locus (points belonging to multiple branches of the root locus).

```
clear

s = tf('s');
```

A Recap: Singular Points, Angles & Centroid of the Asymptotes

We refer to the parameterisation of $L(s)$ using poles and zeros (refer to slide Part4, 28 fo the course material)

$$L(s) = \varrho \frac{\prod_{j=1}^{m}(s+z_j)}{\prod_{i=1}^{n}(s+p_i)}, \quad m \leq n$$

The Root Locus diagram shows the roots of

$$1 + L(s) = 0 \iff 1 + \varrho \frac{\prod_{j=1}^{m}(s+z_j)}{\prod_{i=1}^{n}(s+p_i)} = 0 \iff \frac{\prod_{j=1}^{m}(s+z_j)}{\prod_{i=1}^{n}(s+p_i)} = -\frac{1}{\varrho} \quad (\star)$$

in the complex plane for all $\varrho \in \mathbb{R},\ \varrho \neq 0$.

The Singular Points of a Root Locus

Let's rewrite Eq. $(\star)$ as

$$\varrho = -\frac{\prod_{i=1}^{n}(s+p_i)}{\prod_{j=1}^{m}(s+z_j)} = \gamma(s)$$

If we consider $s = x, x \in \mathbb{R}$, then the **singular points** (or **breakaway/break-in points**) of the **root locus**, belonging to the real axis of the complex plane, are the **stationary points of** $\gamma(x)$ such that

$$\bar{x} \in \mathbb{R} : \frac{d\,\gamma(x)}{dx}\Big|_{x=\bar{x}} = 0,\ \gamma(x) = -\frac{\prod_{i=1}^{n}(s+p_i)}{\prod_{j=1}^{m}(s+z_j)}\Bigg|_{s=x,\ x\in\mathbb{R}} \quad (+)$$

Note: given a singular point $\bar{x}$, the corresponding value of ϱ can be computed simply as

$$\bar{\varrho} = \gamma(\bar{x}) = -\frac{\prod_{i=1}^{n}(\bar{x}+p_i)}{\prod_{j=1}^{m}(\bar{x}+z_j)}$$

Let's rewrite Eq. $(+)$:

$$\gamma(x) = -\frac{\prod_{i=1}^{n}(x+p_i)}{\prod_{j=1}^{m}(x+z_j)} \implies \frac{d\,\gamma(x)}{dx} = 0 \longleftrightarrow \left[\prod_{j=1}^{m}(x+z_j)\right] \cdot \frac{d}{dx}\left(\prod_{i=1}^{n}(x+p_i)\right) - \left[\prod_{i=1}^{n}(x+p_i)\right] \cdot \frac{d}{dx}\left(\prod_{j=1}^{m}(x+z_j)\right) = 0$$

By solving Eq. (+), do we find every singular point of the locus? In general no, not always!

A generic point $\hat{s} \in \mathbb{C}$ belongs to the root locus described by Eq. ($\star$) if and only if there exists a real value $\hat{\varrho}$ such that

$$\frac{\prod_{j=1}^{m}(\hat{s}+z_j)}{\prod_{i=1}^{n}(\hat{s}+p_i)} = -\frac{1}{\hat{\varrho}}$$

Thus, a candidate singular point $\tilde{s} \in \mathbb{C}$ solution of

$$\tilde{s} \in \mathbb{C} : \frac{d\,\gamma(s)}{d}\Big|_{s=\tilde{s}} = 0,\ \gamma(s) = -\frac{\prod_{i=1}^{n}(s+p_i)}{\prod_{j=1}^{m}(s+z_j)} \qquad (*)$$

is a true singular point if the corresponding value $\tilde{\varrho} = \gamma(\tilde{s})$ is **real**.

- Candidate singular points (i.e. solutions of Eq. ($*$)), which are real values, are for sure singular points.
- Candidate complex-valued singular points $\tilde{s}$ must be checked: they are actual singular points if and only if $\tilde{\varrho} = \gamma(\tilde{s}) \in \mathbb{R}$.
- This does not seem to be an efficient strategy.

Let's rewrite Eq $(*)$:

$$\gamma(s) = -\frac{\prod_{i=1}^{n}(s+p_i)}{\prod_{j=1}^{m}(s+z_j)} \implies \frac{d\,\gamma(s)}{ds} = 0 \longleftrightarrow \left[\prod_{j=1}^{m}(s+z_j)\right] \cdot \frac{d}{ds}\left(\prod_{i=1}^{n}(s+p_i)\right) - \left[\prod_{i=1}^{n}(s+p_i)\right] \cdot \frac{d}{ds}\left(\prod_{j=1}^{m}(s+z_j)\right) = 0 \qquad (**)$$

Multiple Zeros and Poles as Singular Points

- Every **open-loop pole/zero** with **multiplicity greater than 1** is a **singular point** of the root locus (easy to prove from Eq. ($*$)).
- Example: let $-p_k$ be a pole with algebraic multiplicity 2

$$\prod_{i=1}^{n}(s+p_i) = (s+p_k)^2 \prod_{i\neq k}(s+p_i)$$

Thus

$$\left[\prod_{j=1}^{m}(s+z_j)\right]\cdot\frac{d}{ds}\left(\prod_{i=1}^{n}(s+p_i)\right)-\left[\prod_{i=1}^{n}(s+p_i)\right]\cdot\frac{d}{ds}\left(\prod_{j=1}^{m}(s+z_j)\right)=0 \iff \left[\prod_{j=1}^{m}(s+z_j)\right]\cdot\frac{d}{ds}\left((s+p_k)^2\prod_{i\neq k}(s+p_i)\right)-\left[(s+p_k)^2\prod_{i\neq k}(s+p_i)\right]\cdot\frac{d}{ds}\left(\prod_{j=1}^{m}(s+z_j)\right)=0$$

Then

$$\left[\prod_{j=1}^{m}(s+z_j)\right]\cdot\left\{2(s+p_k)\left[\prod_{i\neq k}(s+p_i)\right]+(s+p_k)^2\frac{d}{d\,s}\left[\prod_{i\neq k}(s+p_i)\right]\right\}-\left[(s+p_k)^2\prod_{i\neq k}(s+p_i)\right]\cdot\frac{d}{ds}\left(\prod_{j=1}^{m}(s+z_j)\right)=0$$

So, $s=-p_k$ is a solution of the polynomial equation.

Important Remark

- If the open-loop pole/zero with multiplicity greater than 1 is real-valued, it is the solution of Eq. (+).
- Otherwise, if complex, it is one of the solutions of Eq. (∗), which is an effective singular point of the locus.
- In the latter case, by efficiently exploiting this property, we have successfully determined the singular points of the locus that do not belong to the real axis without the need to solve Eq. (∗) directly.

Examples

$$L_1(s)=\varrho\frac{(s+1)^2}{(s+4)^2}\qquad L_2(s)=\varrho\,\frac{(s+1)^2(s+2)}{(s^2+2s+9)^2}$$

A Different Expression for the Singular Points Equation

You can manipulate Eq. $(**)$ to obtain an additional expression for the singular points equation.

Let's define

$$L(s)=\frac{\prod_{j=1}^{m}(s+z_j)}{\prod_{i=1}^{n}(s+p_i)}=\frac{N(s)}{\varphi(s)}$$

Thus, you can rewrite Eq, $(**)$ as

$$N(s)\cdot\frac{d}{ds}\left\{\varphi(s)\right\}-\varphi(s)\cdot\frac{d}{ds}\left\{N(s)\right\}=0 \iff \varphi(s)\cdot\frac{d}{ds}\left\{N(s)\right\}-N(s)\cdot\frac{d}{ds}\left\{\varphi(s)\right\}=0 \iff \frac{d}{d\,s}\left[L(s)\right]=0 \qquad (**ALT)$$

This formulation could be helpful if you don't have the factorised form for the polynomials in the numerator and denominator in $L(s)$.

With a Little Help from the 'help' Command!

Here is the documentation related to the *Control System Toolbox* function we will use in this live script:

```
help rlocus
```

```
rlocus - Root locus of dynamic system
   This MATLAB function calculates the root locus of SISO model sys and
   returns the resulting vector of feedback gains k and corresponding
   complex root locations r.

   Syntax
     [r,kout] = rlocus(sys)
     r = rlocus(sys,k)

     rlocus(___)

   Input Arguments
     sys - Dynamic system
       dynamic system model | model array
     k - Feedback gain values
       vector

   Output Arguments
     r - Closed-loop pole locations
       array
     kout - Feedback gain values
       vector

   Examples
     Root Locus Plot of Dynamic System
     Root Locus Plot of Multiple Dynamic System Models
     Closed-Loop Poles and Feedback Gain Values Using Root Locus
     Closed-Loop Pole Locations for a Set of Feedback Gain Values

   See also rlocusplot, tf, pole, zero, ss, zpk

   Introduced in Control System Toolbox before R2006a
   Documentation for rlocus
   Other uses of rlocus
```

```
help rloclims
```

```
rloclims  Compute adequate axis limits for root loci with asymptotes.

   [XLIMS,YLIMS] = rloclims(R) returns the X and Y axis limits
   vectors given the set of roots R (arranged column-wise).

   See also rlocus.
```

```
help rlocfind
```

```
rlocfind  Find root locus gains for a given set of roots.

   [K,POLES] = rlocfind(SYS)  is used for interactive gain
   selection from the root locus plot of the SISO system SYS
   generated by RLOCUS.  rlocfind puts up a crosshair cursor
   in the graphics window which is used to select a pole location
   on an existing root locus.  The root locus gain associated
   with this point is returned in K and all the system poles for
   this gain are returned in POLES.

   [K,POLES] = rlocfind(SYS,P)  takes a vector P of desired root
   locations and computes a root locus gain for each of these
   locations (i.e., a gain for which one of the closed-loop roots
   is near the desired location).  The j-th entry of the vector K
   gives the computed gain for the location P(j), and the j-th
   column of the matrix POLES lists the resulting closed-loop poles.
```

```
    See also  rlocus.

    Other uses of rlocfind
```

```
help rlocusplot
```

```
--- help for controllib.chart.RLocusPlot ---

 controllib.chart.RLocusPlot - Root locus plot of dynamic system
    The rlocusplot function plots the root locus of a dynamic system model
    and returns an RLocusPlot chart object.

    Creation
      Syntax
        rlp = rlocusplot(sys)
        rlp = rlocusplot(sys1,sys2,...,sysN)
        rlp = rlocusplot(sys1,LineSpec1,...,sysN,LineSpecN)
        np = rlocus(___,k)

        rlp = rlocusplot(___,plotoptions)

        rlp = rlocusplot(parent,___)

      Input Arguments
        sys - Dynamic system
          dynamic system model | model array
        LineSpec - Line style, marker, and color
          string | character vector
        k - Feedback gain values
          vector
        plotoptions - Pole-zero plot options
          pzoptions object
        parent - Parent container
          Figure object (default) | TiledChartLayout object |
          UIFigure object | UIGridLayout object | UIPanel object |
          UITab object

    Properties
      Responses - Model responses
        RootLocusResponse object | array of RootLocusResponse objects
      TimeUnit - Time units
        "seconds" | "milliseconds" | "minutes" | ...
      FrequencyUnit - Frequency units
        "rad/s" | "Hz" | "rpm" | ...
      Visible - Chart visibility
        "on" (default) | on/off logical value

    Object Functions
      addResponse - Add dynamic system response to existing response plot
      setoptions  - (Not recommended) Set options for linear analysis plot object
      getoptions  - (Not recommended) Get options for linear analysis plot object

    Examples
      Plot Root Locus

    See also rlocus, pzoptions

    Introduced in Control System Toolbox before R2006a
    Documentation for controllib.chart.RLocusPlot
```

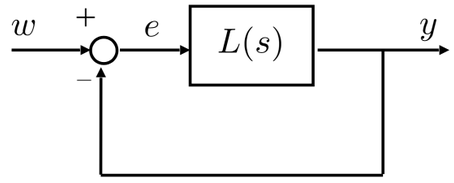
```
help sgrid
```

```
 sgrid - Generate s-plane grid of constant damping factors and natural frequencies
    This MATLAB function generates a grid of constant damping factors from 0
    to 1 in steps of 0.1 and natural frequencies from 0 to 10 rad/sec in
```

```
steps of one rad/sec for pole-zero and root locus plots.

Syntax
  sgrid
  sgrid(zeta,wn)
  sgrid(___,'new')
  sgrid(AX,___)

Input Arguments
  zeta - Damping ratio
    vector
  wn - Normalized natural frequency
    vector
  AX - Object handle
    Axes object | UIAxes object

Examples
  Generate S-Plane Grid on Root Locus Plot

See also pzmap, rlocus, zgrid

Introduced in Control System Toolbox before R2006a
Documentation for sgrid
Other uses of sgrid
```

Example: a Direct Root Locus

Consider the following block diagram



where $L(s)$ is the transfer function

$$L(s) = \varrho \frac{s+1}{s^2}, \varrho \in \mathbb{R}, \varrho > 0$$

Let's determine the **Direct RL** in MATLAB.

- **The first step**: write the closed-loop equation as

$$1 + L(s) = 0 \quad \Longrightarrow \quad 1 + \varrho \frac{\prod_{j=1}^{m}(s+z_j)}{\prod_{i=1}^{n}(s+p_i)} = 0 \quad \Longrightarrow \quad \frac{\prod_{j=1}^{m}(s+z_j)}{\prod_{i=1}^{n}(s+p_i)} = -\frac{1}{\varrho}$$

- **Notice** that the last expression separates the variable parameter (ϱ in the example) from the transfer function expression. This form of the closed-loop pole equation will be used to determine the expression of the transfer function to be entered into the MATLAB functions, allowing us to plot the required root locus.

$$1+\varrho\frac{s+1}{s^2}=0 \quad \Longrightarrow \quad \frac{s+1}{s^2}=-\frac{1}{\varrho}$$

Thus the transfer function we will use in MATLAB is:

```
L1s = (s+1)/s^2
```

```
L1s =

  s + 1
  -----
   s^2

Continuous-time transfer function.
Model Properties
```

Zeros & Poles

- **The second step**: compute the open-loop zeros and poles.
- Are there zeros or poles with multiplicity greater than 1?

Simply

```
zL1 = zero(L1s)
```

```
zL1 =
-1
```

```
pL1 = pole(L1s)
```

```
pL1 = 2×1
     0
     0
```

Notice the pole at the origin $p_1=0$, with multiplicity $n_1=2$. Thus it is a **singular point**!

Singular Points

- **The third step**: determine the **set of singular points** of the root locus, by solving Eq. (+) and looking for open-loop complex-valued zeros/poles with multiplicity greater than 1.

$$L_1(s)=\frac{s+1}{s^2} \quad \Longrightarrow \quad \gamma(x)=-\frac{x^2}{x+1},\ x\in\mathbb{R} \quad \Longrightarrow \quad \frac{d\,\gamma(x)}{dx}=0 \Longleftrightarrow \quad \frac{d}{dx}\left[-\frac{x^2}{x+1}\right]=0$$

```
syms x real
syms gamma(x)
gamma(x) = -x^2/(x+1)
```

```
gamma(x) =
```

$$-\frac{x^2}{x+1}$$

```
dot_g(x) = simplify(collect(diff(gamma,x),x))
```

```
dot_g(x) =
```

$$-\frac{x\,(x+2)}{(x+1)^2}$$

```
breakP_candidate = solve(dot_g==0)
```

```
breakP_candidate =
```

$$\begin{pmatrix}-2\\0\end{pmatrix}$$

```
breakP_C1 = breakP_candidate(1);
breakP_C2 = breakP_candidate(2);
```

Note: by solving Eq. (+) we found also the pole $p_1 = 0$ as singular point (we already knew this).

Question 1: To which root locus does the singular point $x_1 = 0$ belong?

Question 2: To which root locus does the other singular point candidate belong?

```
if (gamma(breakP_C1) >0)
   fprintf('x = %.2f is a singular point of the Direct RL.\n', breakP_C1);
else
    fprintf('x = %.2f is a singular point of the Inverse RL.\n', breakP_C1);
end
```

```
x = -2.00 is a singular point of the Direct RL.
```

```
breakPs = double(breakP_candidate);
```

Centroid & Asymptotes

Given

$$L(s) = \varrho\,\frac{\prod_{j=1}^{m}(s+z_j)}{\prod_{i=1}^{n}(s+p_i)},\ \varrho \in \mathbb{R},\ m < n$$

the center of the asymptotes (named **centroid**) is

$$x_a = \frac{1}{n-m}\left[\sum_{j=1}^{m} z_j - \sum_{i=1}^{n} p_i\right]$$

and the $(n-m)$ asymptotes (of the Direct RL) form the following angles with the real axis

$$\psi_a = \frac{(2k+1)\cdot 180^\circ}{n-m},\ k = 0, 1, \ \ldots, n-m-1$$

Thus, the centroid is

```
n = numel(pL1); % the degree of the polynomial at the denominator in the TF
m = numel(zL1); % the degree of the polynomial at the nuumerator in the TF

% the centroid
x_a = (1/(n-m))*(sum(pL1) - sum(zL1))
```

```
x_a =
1
```

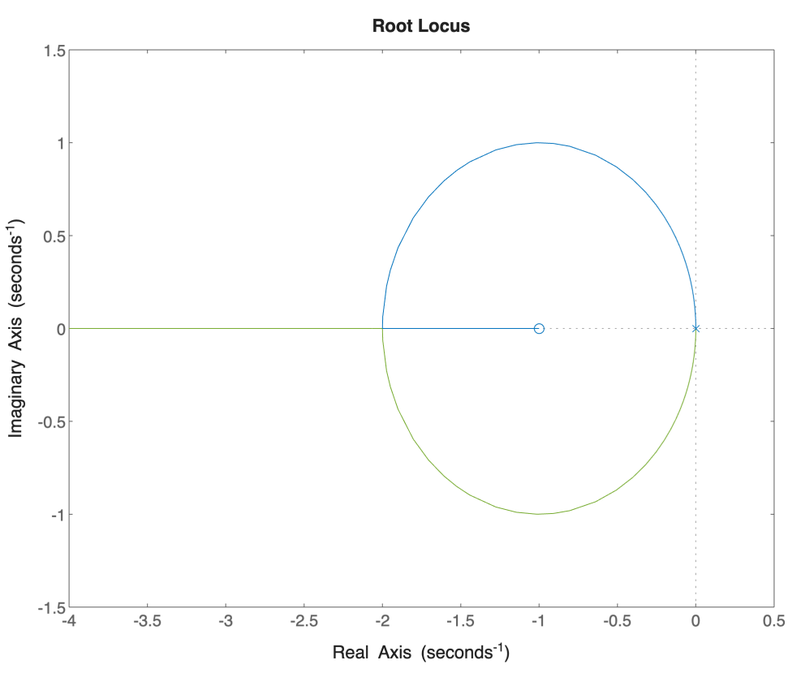
and the asymptotes of the DL form the angles

```
k = (0:n-m-1);
theta_a = (180/(n-m))*(2.*k+1)
```

```
theta_a =
180
```

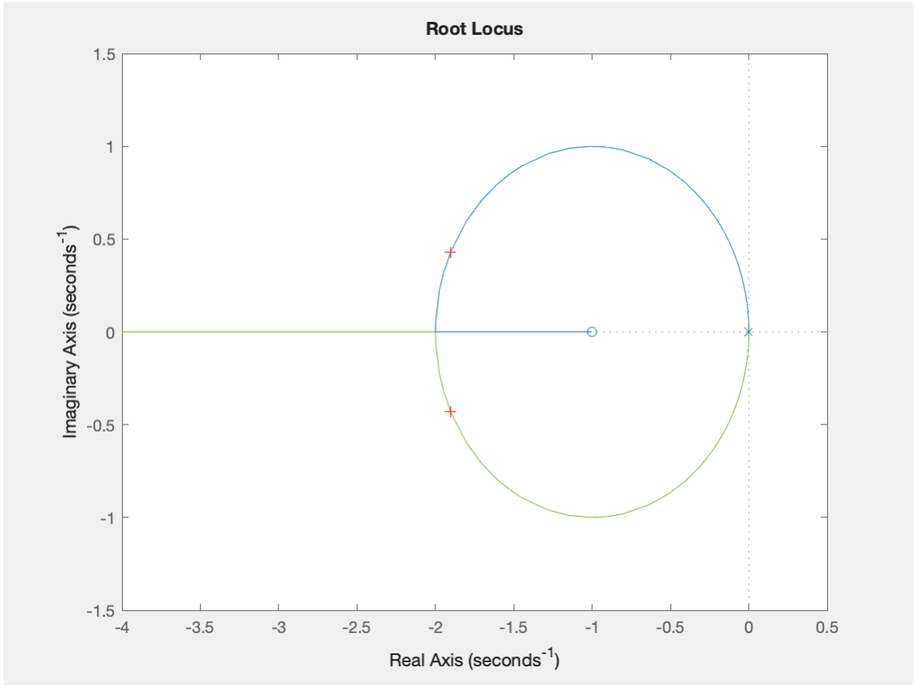
The Direct Locus

```
figure;rlocus(L1s);
```



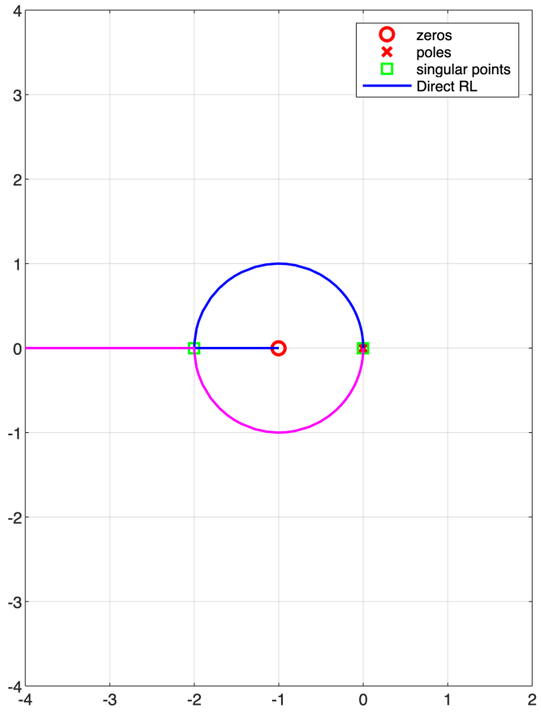

```
[KK, cc_poles] = rlocfind(L1s)
```

```
Select a point in the graphics window
```




```
selected_point =
-1.8874 + 0.4204i
KK =
3.8077
cc_poles = 2×1 complex
  -1.9039 + 0.4278i
```

```
RL1 = rlocus(L1s);
% note the structure of RL1: array(n, p)
% where n <--> number of branches (i.e. the degree of the polynomial at
%              denominator)
%       p <--> number of points (closed-loop poles) belonging to each
%              branch, i.e. number of values fo the gain rho used to
%              determine the Direct Locus
figure('Units','centimeters','Position',[0.1, 0.1, 22, 18]);
% let's start with zeros & poles and then singular points
h_s1 = plot(real(zL1),imag(zL1),'ro', ...
    real(pL1),imag(pL1),'rx', ...
    real(breakPs), imag(breakPs), 'gs', ...
    'MarkerSize',8, 'LineWidth',2); hold on;

% the Direct RL
h_rld1 = plot( real(RL1(1,:)),imag(RL1(1,:)),'b', ...
   real(RL1(2,:)),imag(RL1(2,:)),'m', ...
  'LineWidth',1.5 );

grid on; hold off; axis equal;
axis([-4,2,-4, 4]);
legend([h_s1;h_rld1(1)],'zeros','poles','singular points','Direct RL')
```



What about the Inverse Root Locus?

What about closed-loop performance?

Hands-on Examples

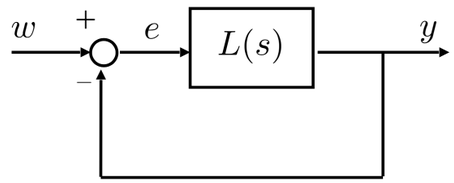
Determine the **Direct** and the **Inverse Root Locus** for the transfer functions

$$L_a(s) = \frac{(s+1)^2}{(s+2)^4} \qquad L_b(s) = \frac{(s+1)^2}{(s-1)^3} \qquad L_c(s) = \frac{s+4}{s\left(s^2+s+\frac{5}{4}\right)(s+2)^2}$$

Closed-Loop Performance Analysis Using Root Locus

A First Example

Consider the block diagram



where $L(s)$ is the transfer function

$$L(s) = \varrho \frac{s+6}{(s+1)^2}, \varrho \in \mathbb{R}, \varrho > 0$$

Let's vary the gain ϱ: what about the closed-loop performance and stability? Can we infer the step-response features when increasing the gain? Can we infer the stability margins?

```
clear variables
s = tf('s');

% the transfer function L(s)
L1s = (s+6)/((s+1)^2);

% zeros & poles
zL1 = zero(L1s);
pL1 = pole(L1s);

% asymptotes & centroid
n = numel(pL1); % the degree of the polynomial at the denominator in the TF
m = numel(zL1); % the degree of the polynomial at the nuumerator in the TF
% the centroid
x_a = (1/(n-m))*(sum(pL1) - sum(zL1))
```

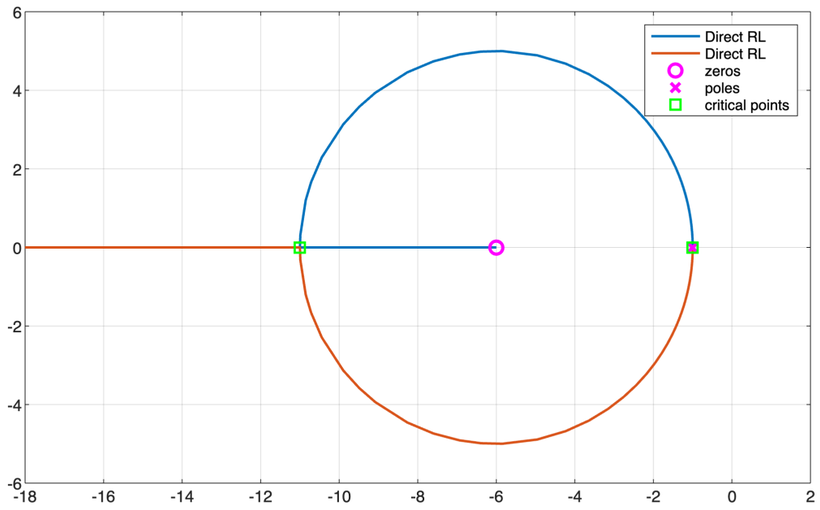
```
x_a =
4
```

```
% the asymptotes belonging to the DL
k = (0:n-m-1);
theta_a = (180/(n-m))*(2.*k+1)
```

```
theta_a =
180
```

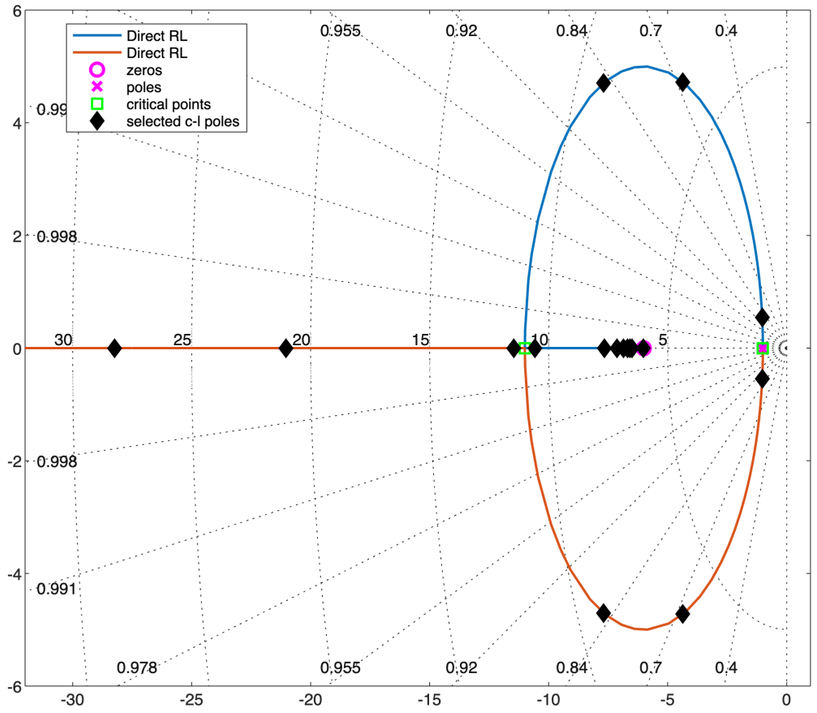
```
% ========================================================
% Let's determine the critical points using the Eq. (**ALT)
criticalPoints = DetermineCriticalPointsRL(L1s);
cP_DLset = checkCriticPointsDL(criticalPoints, L1s);
% Which critical point belongs to the Direct Root Locus?
% ========================================================

RL1 = rlocus(L1s);
% note the structure of RL1: array(n, p)
% where n <--> number of branches (i.e. the degree of the polynomial at
%              denominator)
%       p <--> number of points (closed-loop poles) belonging to each
```

```
%              branch, i.e. number of values fo the gain rho used to
%              determine the Direct Locus
hfRL1 = figure('Units','centimeters','Position',[0.1, 0.1, 22, 18]);
% the Direct RL
n = size(RL1, 1); % how many branches?
for k=1:n
    h_rld1(k,1) = plot( real(RL1(k,:)),imag(RL1(k,:)),'LineWidth',1.5 );
    hold on;
end % for k

% zeros & poles and then singular points
h_s1 = plot(real(zL1),imag(zL1),'mo', ...
    real(pL1),imag(pL1),'mx', ...
    real(cP_DLset), imag(cP_DLset), 'gs', ...
    'MarkerSize',8, 'LineWidth',2);

grid on; hold off; axis equal;
axis([-18,2,-6, 6]);
LocusLeg = cell(1,n);
for k=1:n
    LocusLeg{1,k} = 'Direct RL';
end % for k
legend([LocusLeg, {'zeros','poles','critical points'}])
```



Let's consider a set of ϱ values, visualise the corresponding closed-loop poles on the Direct Root Locus and investigate about the performance of the closed-loop system corresponding to each ϱ value:

```
rhoSET = linspace(6e-2, 6.0e1, 10); % 20 values for the gain rho, from 10^(-2) to 10^4
rhoSET = [rhoSET, 1000];
closedLoopPoles = rlocus(L1s, rhoSET);

figure('Units','centimeters','Position',[0.1, 0.1, 22, 18]);
% the Direct RL
n = size(RL1, 1); % how many branches?
for k=1:n
    h_rld1(k,1) = plot( real(RL1(k,:)),imag(RL1(k,:)),'LineWidth',1.5 );
    hold on;
end % for k

% zeros & poles and then singular points
h_s1 = plot(real(zL1),imag(zL1),'mo', ...
    real(pL1),imag(pL1),'mx', ...
    real(cP_DLset), imag(cP_DLset), 'gs', ...
    'MarkerSize',8, 'LineWidth',2);

axis([-32,1,-6, 6]);
LocusLeg = cell(1,n);
for k=1:n
    LocusLeg{1,k} = 'Direct RL';
end % for k

hold on;
n = size( closedLoopPoles,2);
for k=1:n
    hp = plot( real(closedLoopPoles(:,k)),imag(closedLoopPoles(:,k)), ...
   'LineStyle', 'none', 'Marker', 'diamond', 'MarkerSize', 8,...
   'MarkerEdgeColor','k', 'MarkerFaceColor','k');

end % for k
sgrid
hl = [h_rld1 ;h_s1; hp];
legend(hl,[LocusLeg, {'zeros','poles','critical points', 'selected c-l poles'}], 'Location', 'best')
```



Closed-Loop Performance Analysis

Let's determine the open-loop and the closed-loop transfer functions corresponding to each ϱ value:

```
r = numel(rhoSET);
OL_tf = cell(r, 1); % to store the Open-Loop TFs
CL_tf = cell(r, 1); % to store the Closed-Loop TFs

for i=1:r
    OL_tf{i} = rhoSET(i)*L1s;
    CL_tf{i} = feedback(OL_tf{i}, 1, -1);
end % for i
```

Now, let's determine the performance for each transfer function.

```
STABmargins = cell(r, 1);
STEPperformance =  cell(r, 1);
CLpoles = cell(r, 1);

for ijk = 1:r
    [GmVAL,PmVAL,Wcg,Wcp] = margin(OL_tf{ijk} );
    STABmargins{ijk} = struct('Gm', GmVAL, ...
                              'GainFreq', Wcg, ...
```

```
                                'Pm', PmVAL, ...
                                'PhaseFreq', Wcp);
    CLpoles{ijk} = pole(CL_tf{ijk});
    res = stepinfo(CL_tf{ijk}, 'SettlingTimeThreshold',0.01);
    STEPperformance{ijk} = struct('RiseT', res.RiseTime, ...
                                  'SettlingT', res.SettlingTime, ...
                                  'OverShoot', res.Overshoot);
end % for ijk
```

Finally, let's fill a table with the results:

```
results = zeros(r, 8);

for m = 1:r
    results(m, 1) = rhoSET(m);

    results(m, 2) = STABmargins{m}.Pm;
    results(m, 3) = STABmargins{m}.PhaseFreq;
    results(m, 4) = STABmargins{m}.Gm;
    results(m, 5) = STABmargins{m}.GainFreq;

    results(m, 6) = STEPperformance{m}.SettlingT;
    results(m, 7) = STEPperformance{m}.OverShoot;
    results(m, 8) = STEPperformance{m}.RiseT;


end % for m
```

```
REStable1 = array2table([results(:,1),results(:,6:8)], "VariableNames",...
    {' Gain Rho', 'Step Resp. Settling Time', ...
     'Step Resp. Overshoot', 'Step Resp. Rise Time'})
```

REStable1 = 11×4 table

	Gain Rho	Step Resp. Settling Time	Step Resp. Overshoot	Step Resp. Rise Time
1	0.0600	4.0031	0.2760	2.3720
2	6.7200	0.8245	14.3586	0.1715
3	13.3800	0.6254	11.2345	0.1045
4	20.0400	0.5441	9.2235	0.0769
5	26.7000	0.4920	7.8449	0.0613
6	33.3600	0.4521	6.8435	0.0512
7	40.0200	0.4195	6.0789	0.0440
8	46.6800	0.3918	5.4759	0.0386
9	53.3400	0.3678	4.9858	0.0345
10	60	0.3467	4.5796	0.0311
11	1000	0.0043	0.3388	0.0022

```
REStable2 = cell2table([num2cell(results(:,1)),CLpoles], "VariableNames",...
    {' Gain Rho', 'Closed-Loop Poles'})
```

REStable2 = 11×2 table

	Gain Rho	Closed-Loop Poles
1	0.0600	[-1.0300 + 0.5469i;-1.0300 - 0.5469i]
2	6.7200	[-4.3600 + 4.7234i;-4.3600 - 4.7234i]
3	13.3800	[-7.6900 + 4.7057i;-7.6900 - 4.7057i]
4	20.0400	[-11.4677;-10.5723]
5	26.7000	[-21.0375;-7.6625]
6	33.3600	[-28.2357;-7.1243]
7	40.0200	[-35.1627;-6.8573]
8	46.6800	[-41.9853;-6.6947]
9	53.3400	[-48.7553;-6.5847]
10	60	[-55.4949;-6.5051]
11	1000	[-995.9747;-6.0253]

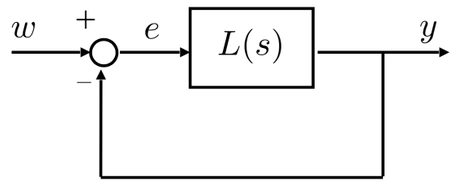
```
REStable3 = array2table([results(:,1:5)], "VariableNames",...
    {' Gain Rho', 'Phase Margin', 'Ph. Marg. Ang. Freq.', ...
     'Gaim Margin', 'G. Marg. Ang. Freq.'})
```

REStable3 = 11×5 table

	Gain Rho	Phase Margin	Ph. Marg. Ang. Freq.	Gaim Margin	G. Marg. Ang. Freq.
1	0.0600	Inf	NaN	Inf	NaN
2	6.7200	67.7187	8.2036	Inf	NaN
3	13.3800	75.3444	14.4224	Inf	NaN
4	20.0400	79.4181	20.8084	Inf	NaN
5	26.7000	81.8004	27.3006	Inf	NaN
6	33.3600	83.3330	33.8505	Inf	NaN
7	40.0200	84.3929	40.4335	Inf	NaN
8	46.6800	85.1665	47.0370	Inf	NaN
9	53.3400	85.7557	53.6661	Inf	NaN
10	60	86.2173	60.2918	Inf	NaN
11	1000	89.7708	1.0000e+03	Inf	NaN

A Second (Long) Example

Consider the block diagram



where $L(s)$ is the transfer function

$$L(s) = \varrho \frac{1}{(s+1)(s+3)}, \varrho \in \mathbb{R}, \varrho > 0$$

Let's vary the gain ϱ: what about the closed-loop performance and stability? Can we infer the step-response features when increasing the gain? Can we infer the stability margins?

What happens if we modify the transfer function, multiplying the actual transfer function by another simple first-order term with a zero and a pole? Does performance improve or worsen?

Let's compare the performance of $L(s)$ with those obtained using the following transfer functions

$$L_a(s) = \varrho \frac{s+5}{(s+1)(s+3)(s+9)}, \varrho \in \mathbb{R}, \varrho > 0 \qquad L_b(s) = \varrho \frac{s+9}{(s+1)(s+3)(s+5)}, \varrho \in \mathbb{R}, \varrho > 0$$

Which is the most performing closed-loop system?

```
clear variables
s = tf('s');

% the transfer function L(s)
L0s = 1/((s+1)*(s+3));

La_s = (s+5)/((s+1)*(s+3)*(s+9));
Lb_s = (s+9)/((s+1)*(s+3)*(s+5));

% zeros & poles
zL0 = zero(L0s);
pL0 = pole(L0s);

% asymptotes & centroid
n = numel(pL0); % the degree of the polynomial at the denominator in the TF
m = numel(zL0); % the degree of the polynomial at the nuumerator in the TF
% the centroid
x_a0 = (1/(n-m))*(sum(pL0) - sum(zL0))
```

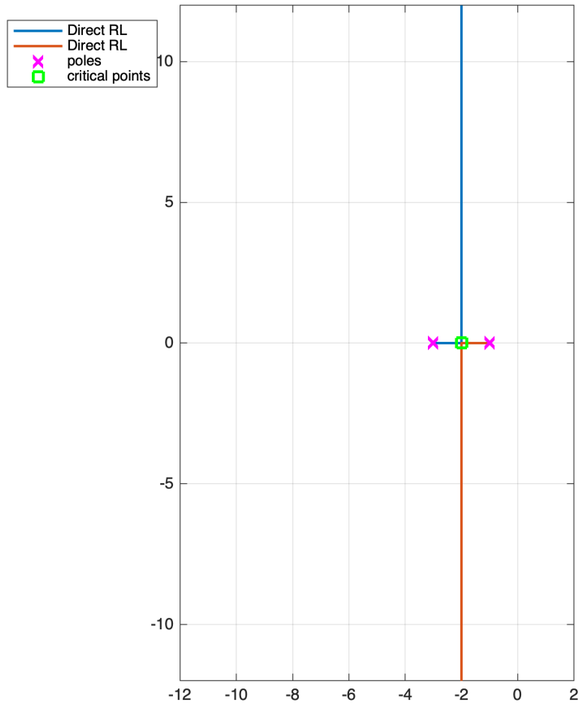
```
x_a0 =
    -2
```

```
% the asymptotes belonging to the DL
k = (0:n-m-1);
theta_a0 = (180/(n-m))*(2.*k+1)
```

```
theta_a0 = 1×2
    90   270
```

```
% ============================================================
```

```
% Let's determine the critical points using the Eq. (**ALT)
criticalPoints0 = DetermineCriticalPointsRL(L0s);
cP_DLset0 = checkCriticPointsDL(criticalPoints0, L0s);
% Which critical point belongs to the Direct Root Locus?
% ========================================================

RL0 = rlocus(L0s);
% note the structure of RL1: array(n, p)
% where n <--> number of branches (i.e. the degree of the polynomial at
%              denominator)
%       p <--> number of points (closed-loop poles) belonging to each
%              branch, i.e. number of values fo the gain rho used to
%              determine the Direct Locus
hfRL0 = figure('Units','centimeters','Position',[0.1, 0.1, 22, 18]);
% the Direct RL
n = size(RL0, 1); % how many branches?
for k=1:n
    h_rld1(k,1) = plot( real(RL0(k,:)),imag(RL0(k,:)),'LineWidth',1.5 );
    hold on;
end % for k

% zeros & poles and then singular points
h_s0 = plot(real(zL0),imag(zL0),'mo', ...
    real(pL0),imag(pL0),'mx', ...
    real(cP_DLset0), imag(cP_DLset0), 'gs', ...
    'MarkerSize',8, 'LineWidth',2);

grid on; hold off; axis equal;
axis([-12,2,-12, 12]);
LocusLeg = cell(1,n);
for k=1:n
    LocusLeg{1,k} = 'Direct RL';
end % for k
legend([LocusLeg, {'poles','critical points'}], 'Location', 'best')
```




```
% zeros & poles
zLa = zero(La_s);
pLa = pole(La_s);

% asymptotes & centroid
n = numel(pLa); % the degree of the polynomial at the denominator in the TF
m = numel(zLa); % the degree of the polynomial at the nuumerator in the TF
% the centroid
x_aA = (1/(n-m))*(sum(pLa) - sum(zLa))
```

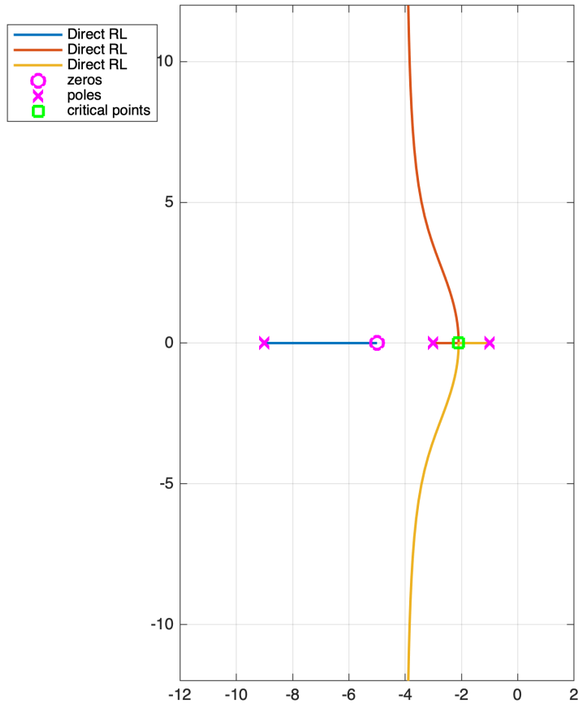
```
x_aA =
  -3.999999999999999
```

```
% the asymptotes belonging to the DL
k = (0:n-m-1);
theta_aA = (180/(n-m))*(2.*k+1)
```

```
theta_aA = 1×2
    90   270
```

```
% ========================================================
% Let's determine the critical points using the Eq. (**ALT)
criticalPoints_A = DetermineCriticalPointsRL(La_s);
```

```
cP_DLset_A = checkCriticPointsDL(criticalPoints_A, La_s);
% Which critical point belongs to the Direct Root Locus?
% ========================================================

RL_A = rlocus(La_s);
% note the structure of RL1: array(n, p)
% where n <--> number of branches (i.e. the degree of the polynomial at
%              denominator)
%       p <--> number of points (closed-loop poles) belonging to each
%              branch, i.e. number of values fo the gain rho used to
%              determine the Direct Locus
hfRL_A = figure('Units','centimeters','Position',[0.1, 0.1, 22, 18]);
% the Direct RL
n = size(RL_A, 1); % how many branches?
for k=1:n
    h_rld1(k,1) = plot( real(RL_A(k,:)),imag(RL_A(k,:)),'LineWidth',1.5 );
    hold on;
end % for k

% zeros & poles and then singular points
h_s_A = plot(real(zLa),imag(zLa),'mo', ...
    real(pLa),imag(pLa),'mx', ...
    real(cP_DLset_A), imag(cP_DLset_A), 'gs', ...
    'MarkerSize',8, 'LineWidth',2);

grid on; hold off; axis equal;
axis([-12,2,-12, 12]);
LocusLeg = cell(1,n);
for k=1:n
    LocusLeg{1,k} = 'Direct RL';
end % for k
legend([LocusLeg, {'zeros','poles','critical points'}], 'Location', 'best')
```



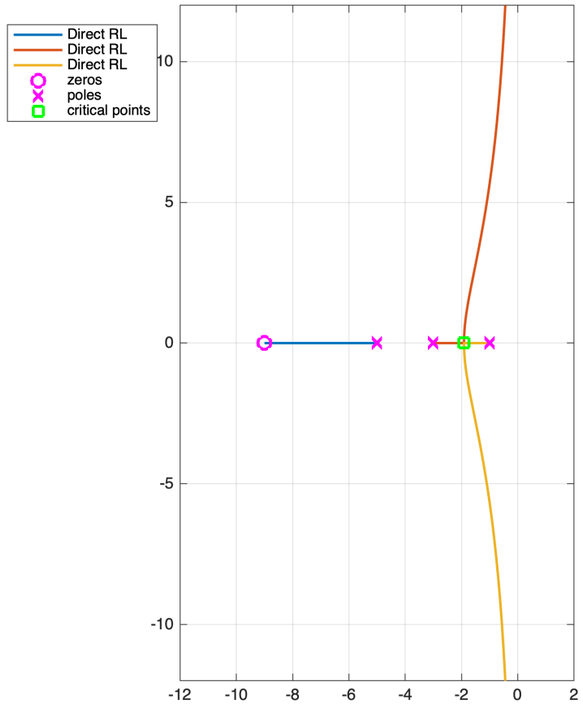

```
% zeros & poles
zLb = zero(Lb_s);
pLb = pole(Lb_s);

% asymptotes & centroid
n = numel(pLb); % the degree of the polynomial at the denominator in the TF
m = numel(zLb); % the degree of the polynomial at the nuumerator in the TF
% the centroid
x_aB = (1/(n-m))*(sum(pLb) - sum(zLb))
```

```
x_aB =
   -1.776356839400250e-15
```

```
% the asymptotes belonging to the DL
k = (0:n-m-1);
theta_aB = (180/(n-m))*(2.*k+1)
```

```
theta_aB = 1×2
    90   270
```

```
% ========================================================
% Let's determine the critical points using the Eq. (**ALT)
criticalPoints_B = DetermineCriticalPointsRL(Lb_s);
cP_DLset_B = checkCriticPointsDL(criticalPoints_B, Lb_s);
% Which critical point belongs to the Direct Root Locus?
% ========================================================

RL_B = rlocus(Lb_s);
% note the structure of RL1: array(n, p)
% where n <--> number of branches (i.e. the degree of the polynomial at
%              denominator)
%       p <--> number of points (closed-loop poles) belonging to each
%              branch, i.e. number of values fo the gain rho used to
%              determine the Direct Locus
hfRL_B = figure('Units','centimeters','Position',[0.1, 0.1, 22, 18]);
% the Direct RL
n = size(RL_B, 1); % how many branches?
for k=1:n
    h_rld1(k,1) = plot( real(RL_B(k,:)),imag(RL_B(k,:)),'LineWidth',1.5 );
    hold on;
end % for k

% zeros & poles and then singular points
h_s_B = plot(real(zLb),imag(zLb),'mo', ...
    real(pLb),imag(pLb),'mx', ...
    real(cP_DLset_B), imag(cP_DLset_B), 'gs', ...
    'MarkerSize',8, 'LineWidth',2);

grid on; hold off; axis equal;
axis([-12,2,-12, 12]);
LocusLeg = cell(1,n);
for k=1:n
    LocusLeg{1,k} = 'Direct RL';
end % for k
legend([LocusLeg, {'zeros','poles','critical points'}], 'Location', 'best')
```



Let's consider a set of ϱ values, visualise the corresponding closed-loop poles on the Direct Root Locus and investigate about the performance of the closed-loop system corresponding to each ϱ value:

```
rhoSET = linspace(6e-2, 6.0e1, 10); % 20 values for the gain rho, from 10^(-2) to 10^4
rhoSET = [rhoSET, 1000];
closedLoopPoles0 = rlocus(L0s, rhoSET);
closedLoopPolesA = rlocus(La_s, rhoSET);
closedLoopPolesB = rlocus(Lb_s, rhoSET);
```

Closed-Loop Performance Analysis

Let's determine the open-loop and the closed-loop transfer functions corresponding to each ϱ value:

```
r = numel(rhoSET);
OL0_tf = cell(r, 1); % to store the Open-Loop TFs
OLA_tf = cell(r, 1); % to store the Open-Loop TFs
OLB_tf = cell(r, 1); % to store the Open-Loop TFs
CL0_tf = cell(r, 1); % to store the Closed-Loop TFs
CLA_tf = cell(r, 1); % to store the Closed-Loop TFs
CLB_tf = cell(r, 1); % to store the Closed-Loop TFs

for i=1:r
    OL0_tf{i} = rhoSET(i)*L0s;
    OLA_tf{i} = rhoSET(i)*La_s;
    OLB_tf{i} = rhoSET(i)*Lb_s;
```

```
    CL0_tf{i} = feedback(OL0_tf{i}, 1, -1);
    CLA_tf{i} = feedback(OLA_tf{i}, 1, -1);
    CLB_tf{i} = feedback(OLB_tf{i}, 1, -1);
end % for i
```

Now, let's determine the performance for each transfer function.

```
STABmargins0 = cell(r, 1);
STEPperformance0 =  cell(r, 1);
CLpoles0 = cell(r, 1);

STABmarginsA = cell(r, 1);
STEPperformanceA =  cell(r, 1);
CLpolesA = cell(r, 1);

STABmarginsB = cell(r, 1);
STEPperformanceB =  cell(r, 1);
CLpolesB = cell(r, 1);

for ijk = 1:r
    [GmVAL,PmVAL,Wcg,Wcp] = margin(OL0_tf{ijk} );
    STABmargins0{ijk} = struct('Gm', GmVAL, ...
                               'GainFreq', Wcg, ...
                               'Pm', PmVAL, ...
                               'PhaseFreq', Wcp);
    CLpoles0{ijk} = pole(CL0_tf{ijk});
    res = stepinfo(CL0_tf{ijk}, 'SettlingTimeThreshold',0.01);
    STEPperformance0{ijk} = struct('RiseT', res.RiseTime, ...
                               'SettlingT', res.SettlingTime, ...
                               'OverShoot', res.Overshoot);

    [GmVAL,PmVAL,Wcg,Wcp] = margin(OLA_tf{ijk} );
    STABmarginsA{ijk} = struct('Gm', GmVAL, ...
                           'GainFreq', Wcg, ...
                           'Pm', PmVAL, ...
                           'PhaseFreq', Wcp);
    CLpolesA{ijk} = pole(CLA_tf{ijk});
    res = stepinfo(CLA_tf{ijk}, 'SettlingTimeThreshold',0.01);
    STEPperformanceA{ijk} = struct('RiseT', res.RiseTime, ...
                               'SettlingT', res.SettlingTime, ...
                               'OverShoot', res.Overshoot);

    [GmVAL,PmVAL,Wcg,Wcp] = margin(OLB_tf{ijk} );
    STABmarginsB{ijk} = struct('Gm', GmVAL, ...
                           'GainFreq', Wcg, ...
                           'Pm', PmVAL, ...
                           'PhaseFreq', Wcp);
    CLpolesB{ijk} = pole(CLB_tf{ijk});
    res = stepinfo(CLB_tf{ijk}, 'SettlingTimeThreshold',0.01);
    STEPperformanceB{ijk} = struct('RiseT', res.RiseTime, ...
                               'SettlingT', res.SettlingTime, ...
                               'OverShoot', res.Overshoot);
```

```
end % for ijk
```

Finally, let's fill a table with the results:

```
results0 = zeros(r, 8);
resultsA = zeros(r, 8);
resultsB = zeros(r, 8);

for m = 1:r
    results0(m, 1) = rhoSET(m);

    results0(m, 2) = STABmargins0{m}.Pm;
    results0(m, 3) = STABmargins0{m}.PhaseFreq;
    results0(m, 4) = STABmargins0{m}.Gm;
    results0(m, 5) = STABmargins0{m}.GainFreq;

    results0(m, 6) = STEPperformance0{m}.SettlingT;
    results0(m, 7) = STEPperformance0{m}.OverShoot;
    results0(m, 8) = STEPperformance0{m}.RiseT;

    resultsA(m, 1) = rhoSET(m);

    resultsA(m, 2) = STABmarginsA{m}.Pm;
    resultsA(m, 3) = STABmarginsA{m}.PhaseFreq;
    resultsA(m, 4) = STABmarginsA{m}.Gm;
    resultsA(m, 5) = STABmarginsA{m}.GainFreq;

    resultsA(m, 6) = STEPperformanceA{m}.SettlingT;
    resultsA(m, 7) = STEPperformanceA{m}.OverShoot;
    resultsA(m, 8) = STEPperformanceA{m}.RiseT;

    resultsB(m, 1) = rhoSET(m);

    resultsB(m, 2) = STABmarginsB{m}.Pm;
    resultsB(m, 3) = STABmarginsB{m}.PhaseFreq;
    resultsB(m, 4) = STABmarginsB{m}.Gm;
    resultsB(m, 5) = STABmarginsB{m}.GainFreq;

    resultsB(m, 6) = STEPperformanceB{m}.SettlingT;
    resultsB(m, 7) = STEPperformanceB{m}.OverShoot;
    resultsB(m, 8) = STEPperformanceB{m}.RiseT;


end % for m

REStable1_0 = array2table([results0(:,1),results0(:,6:8)], "VariableNames",...
    {' Gain Rho', 'Step Resp. Settling Time', ...
     'Step Resp. Overshoot', 'Step Resp. Rise Time'})
```

REStable1_0 = 11×4 table

	Gain Rho	Step Resp. Settling Time	Step Resp. Overshoot	Step Resp. Rise Time
1	0.060000000000000	4.882716967292071	0	2.334335402227898
2	6.720000000000000	2.057321517768195	7.228507786117833	0.628664483731854

	Gain Rho	Step Resp. Settling Time	Step Resp. Overshoot	Step Resp. Rise Time
3	13.380000000000001	2.171135398593359	16.763605916201850	0.402273467505006
4	20.039999999999999	2.327265776086779	23.683557377171294	0.310956902508124
5	26.699999999999999	2.108443752079773	28.953932610360901	0.259811783070024
6	33.359999999999999	2.314016720426889	33.136742174962741	0.226880635018854
7	40.020000000000003	2.178528431448700	36.562733771937815	0.202768989948948
8	46.680000000000000	2.044164502492058	39.451132268294288	0.184912892094044
9	53.340000000000003	2.269495308667151	41.946201868878738	0.171298932494013
10	60	2.172730009296917	44.090085849779051	0.160117734621057
11	1000	2.293291513688223	81.970485997654819	0.034559417699746

```
REStable1_A = array2table([resultsA(:,1),resultsA(:,6:8)], "VariableNames",...
    {' Gain Rho', 'Step Resp. Settling Time', ...
     'Step Resp. Overshoot', 'Step Resp. Rise Time'})
```

REStable1_A = 11×4 table

	Gain Rho	Step Resp. Settling Time	Step Resp. Overshoot	Step Resp. Rise Time
1	0.060000000000000	4.839773939848826	0	2.314457061487797
2	6.720000000000000	1.520240195231347	0.735599325075942	0.916461153941538
3	13.380000000000001	1.834884204512138	4.060862283283573	0.574417950189225
4	20.039999999999999	1.497190023639296	7.506962602644540	0.427661878395971
5	26.699999999999999	1.265034627903564	10.622543028122045	0.346625673435545
6	33.359999999999999	1.504726932424442	13.406496508617028	0.294894144509559
7	40.020000000000003	1.412733982737880	15.908337486904767	0.258885597852009
8	46.680000000000000	1.316615548351497	18.155794567890272	0.232083766207570
9	53.340000000000003	1.230883045339334	20.206147920977191	0.211450116380918
10	60	1.156434258885211	22.074151015221876	0.195131036851929
11	1000	1.123079487286126	67.911084105866834	0.036280441512641

```
REStable1_B = array2table([resultsB(:,1),resultsB(:,6:8)], "VariableNames",...
    {' Gain Rho', 'Step Resp. Settling Time', ...
     'Step Resp. Overshoot', 'Step Resp. Rise Time'})
```

REStable1_B = 11×4 table

	Gain Rho	Step Resp. Settling Time	Step Resp. Overshoot	Step Resp. Rise Time
1	0.060000000000000	4.867645572270012	0	2.328477768426513
2	6.720000000000000	3.252269453948275	22.446575356732712	0.455619088913752
3	13.380000000000001	3.770375674941782	37.874240339507857	0.306307226686239
4	20.039999999999999	4.279832098632355	47.378592138983635	0.244461573386099
5	26.699999999999999	4.411240241989121	53.924952968238692	0.209006183078909
6	33.359999999999999	4.929091242688632	58.859548754269483	0.185128499889900
7	40.020000000000003	5.408685246192301	62.683103620405412	0.168166238145172
8	46.680000000000000	5.857097942668095	65.769636146327002	0.154946816161271
9	53.340000000000003	6.277210417756745	68.354202864490830	0.144226821054653
10	60	6.377784624372757	70.121418429946303	0.138477429343769
11	1000	52.588974018622601	96.601564583975858	0.033031843795217

```
REStable2_0 = cell2table([num2cell(results0(:,1)),CLpoles0], "VariableNames",...
    {' Gain Rho', 'Closed-Loop Poles'})
```

REStable2_0 = 11×2 table

	Gain Rho	Closed-Loop Poles
1	0.060000000000000	[-2.969535971483266;-1.030464028516734]
2	6.720000000000000	[-2.000000000000000 + 2.391652148620279i;-2.000000000000000 - 2.391652148620279i]
3	13.380000000000001	[-2.000000000000000 + 3.518522417151836i;-2.000000000000000 - 3.518522417151836i]
4	20.039999999999999	[-2.000000000000000 + 4.363484845854286i;-2.000000000000000 - 4.363484845854286i]
5	26.699999999999999	[-2.000000000000000 + 5.069516742254630i;-2.000000000000000 - 5.069516742254630i]
6	33.359999999999999	[-2.000000000000000 + 5.688585061331157i;-2.000000000000000 - 5.688585061331157i]
7	40.020000000000003	[-2.000000000000000 + 6.246599074696568i;-2.000000000000000 - 6.246599074696568i]
8	46.680000000000000	[-2.000000000000000 + 6.758698099486320i;-2.000000000000000 - 6.758698099486320i]
9	53.340000000000003	[-2.000000000000000 + 7.234638899074370i;-2.000000000000000 - 7.234638899074370i]
10	60	[-2.000000000000000 + 7.681145747868608i;-2.000000000000000 - 7.681145747868608i]
11	1000	[-1.999999999999999 +31.606961258558208i;-1.999999999999999 -31.606961258558208i]

```
REStable2_A = cell2table([num2cell(resultsA(:,1)),CLpolesA], "VariableNames",...
    {' Gain Rho', 'Closed-Loop Poles'})
```

REStable2_A = 11×2 table

	Gain Rho	Closed-Loop Poles
1	0.060000000000000	[-8.994998959204157;-2.989915594094981;-1.015085446700868]
2	6.720000000000000	[-8.428665426260089 + 0.000000000000000i;-2.285667286869954 + 1.401954129129616i;-2.285667286869954 - 1.401954129129616i]
3	13.380000000000001	[-7.852196970146913 + 0.000000000000000i;-2.573901514926544 + 2.309430154287710i;-2.573901514926544 - 2.309430154287710i]
4	20.039999999999999	[-7.298659705068090 + 0.000000000000000i;-2.850670147465956 + 3.049842132774401i;-2.850670147465956 - 3.049842132774401i]
5	26.699999999999999	[-6.815432078993675 + 0.000000000000000i;-3.092283960503158 + 3.739956858584498i;-3.092283960503158 - 3.739956858584498i]
6	33.359999999999999	[-6.435510985882237 + 0.000000000000000i;-3.282244507058879 + 4.397843770397450i;-3.282244507058879 - 4.397843770397450i]
7	40.020000000000003	[-6.156471698266488 + 0.000000000000000i;-3.421764150866764 + 5.017922058973707i;-3.421764150866764 - 5.017922058973707i]
8	46.680000000000000	[-3.522425260432349 + 5.596372251267300i;-3.522425260432349 - 5.596372251267300i;-5.955149479135309 + 0.000000000000000i]
9	53.340000000000003	[-3.596046580005212 + 6.134936180317283i;-3.596046580005212 - 6.134936180317283i;-5.807906839989580 + 0.000000000000000i]
10	60	[-3.651274358989986 + 6.637942452305044i;-3.651274358989986 - 6.637942452305044i;-5.697451282020029 + 0.000000000000000i]
11	1000	[-3.983740929494222 +31.353362040029005i;-3.983740929494222 -31.353362040029005i;-5.032518141011554 + 0.000000000000000i]

```
REStable2_B = cell2table([num2cell(resultsB(:,1)),CLpolesB], "VariableNames",...
    {' Gain Rho', 'Closed-Loop Poles'})
```

REStable2_B = 11×2 table

	Gain Rho	Closed-Loop Poles
1	0.060000000000000	[-5.029141428689332;-2.908434592672819;-1.062423978637846]
2	6.720000000000000	[-6.165203910242457 + 0.000000000000000i;-1.417398044878764 + 3.199044676940390i;-1.417398044878764 - 3.199044676940390i]
3	13.380000000000001	[-6.597014167893301 + 0.000000000000000i;-1.201492916053348 + 4.368510408940198i;-1.201492916053348 - 4.368510408940198i]
4	20.039999999999999	[-6.873353518906489 + 0.000000000000000i;-1.063323240546754 + 5.224188990821353i;-1.063323240546754 - 5.224188990821353i]
5	26.699999999999999	[-7.075407851157328 + 0.000000000000000i;-0.962296074421337 + 5.929309560560663i;-0.962296074421337 - 5.929309560560663i]
6	33.359999999999999	[-7.233372988580459 + 0.000000000000000i;-0.883313505709771 + 6.542253827644723i;-0.883313505709771 - 6.542253827644723i]
7	40.020000000000003	[-7.362051492792832 + 0.000000000000000i;-0.818974253603584 + 7.091587968807265i;-0.818974253603584 - 7.091587968807265i]
8	46.680000000000000	[-7.469861402422731 + 0.000000000000000i;-0.765069298788641 + 7.593730685088284i;-0.765069298788641 - 7.593730685088284i]

	Gain Rho	Closed-Loop Poles
9	53.340000000000003	[-7.562065198650300 + 0.000000000000000i;-0.718967400674854 + 8.059114663382836i;-0.718967400674854 - 8.059114663382836i]
10	60	[-0.678911217360358 + 8.494843106890604i;-0.678911217360358 - 8.494843106890604i;-7.642177565279277 + 0.000000000000000i]
11	1000	[-0.087202082341053 +31.960181087211545i;-0.087202082341053 -31.960181087211545i;-8.825595835317893 + 0.000000000000000i]

```
REStable3_0 = array2table([results0(:,1:5)], "VariableNames",...
    {' Gain Rho', 'Phase Margin', 'Ph. Marg. Ang. Freq.', ...
     'Gaim Margin', 'G. Marg. Ang. Freq.'})
```

REStable3_0 = 11×5 table

	Gain Rho	Phase Margin	Ph. Marg. Ang. Freq.	Gaim Margin	G. Marg. Ang. Freq.
1	0.060000000000000	Inf	NaN	Inf	Inf
2	6.720000000000000	91.531739255485903	1.679395741877449	Inf	Inf
3	13.380000000000001	63.523992597859021	2.994179447673040	Inf	Inf
4	20.039999999999999	51.645597477992702	3.928777462588190	Inf	Inf
5	26.699999999999999	44.640134478119187	4.690196949282482	Inf	Inf
6	33.359999999999999	39.883076696080359	5.347797384294073	Inf	Inf
7	40.020000000000003	36.381779015716333	5.934592588125223	Inf	Inf
8	46.680000000000000	33.665816066866292	6.469239035480339	Inf	Inf
9	53.340000000000003	31.479552988150211	6.963458912831756	Inf	Inf
10	60	29.670510730818027	7.425171773174528	Inf	Inf
11	1000	7.248891031160707	31.542540638792200	Inf	Inf

```
REStable3_A = array2table([resultsA(:,1:5)], "VariableNames",...
    {' Gain Rho', 'Phase Margin', 'Ph. Marg. Ang. Freq.', ...
     'Gaim Margin', 'G. Marg. Ang. Freq.'})
```

REStable3_A = 11×5 table

	Gain Rho	Phase Margin	Ph. Marg. Ang. Freq.	Gaim Margin	G. Marg. Ang. Freq.
1	0.060000000000000	Inf	NaN	Inf	Inf
2	6.720000000000000	1.354475544816747e+02	0.699005856934063	Inf	Inf
3	13.380000000000001	93.442038244916205	1.939074353414786	Inf	Inf

	Gain Rho	Phase Margin	Ph. Marg. Ang. Freq.	Gaim Margin	G. Marg. Ang. Freq.
4	20.039999999999999	78.641115274470579	2.796732241424344	Inf	Inf
5	26.699999999999999	70.132338213204022	3.515247555791681	Inf	Inf
6	33.359999999999999	64.338456209697242	4.151231619156166	Inf	Inf
7	40.020000000000003	60.001468700869410	4.731016659609129	Inf	Inf
8	46.680000000000000	56.559075553445595	5.268494975807708	Inf	Inf
9	53.340000000000003	53.723069138203208	5.771036900612890	Inf	Inf
10	60	51.317813344579456	6.244997833314786	Inf	Inf
11	1000	14.352827730573289	31.111872525716603	Inf	Inf

```
REStable3_B = array2table([resultsB(:,1:5)], "VariableNames",...
    {' Gain Rho', 'Phase Margin', 'Ph. Marg. Ang. Freq.', ...
     'Gaim Margin', 'G. Marg. Ang. Freq.'})
```

REStable3_B = 11×5 table

	Gain Rho	Phase Margin	Ph. Marg. Ang. Freq.	Gaim Margin	G. Marg. Ang. Freq.
1	0.060000000000000	Inf	NaN	Inf	Inf
2	6.720000000000000	58.373547849378461	2.618202747276477	Inf	Inf
3	13.380000000000001	36.258063186589766	3.995871352692262	Inf	Inf
4	20.039999999999999	26.721259044594895	4.952059548761828	Inf	Inf
5	26.699999999999999	21.204118032071495	5.717455044252373	Inf	Inf
6	33.359999999999999	17.554040615836222	6.371134576751364	Inf	Inf
7	40.020000000000003	14.944952855090726	6.949705769763717	Inf	Inf
8	46.680000000000000	12.981271466810641	7.473795137983799	Inf	Inf
9	53.340000000000003	11.447828257135695	7.956201413110498	Inf	Inf
10	60	10.216617763166724	8.405467355554473	Inf	Inf
11	1000	0.318919971143677	31.959616305834892	Inf	Inf

Other Use of the Root Locus - Generalised Root Locus

The **root locus** can be applied to **any polynomial** with a single parameter varying linearly.

Example: a Spring-Mass-Damper System

We want to explore the influence of some parameters on the system's dynamics represented by the following characteristic polynomial

$$Ms^2 + \mu s + k = 0$$

- What if we vary the spring stiffness?
- What if we vary the damping effect?

Varying the spring stiffness: let's suppose $k \in [0, +\infty)$, and assume M and μ as constant parameters

$$p(s,k) = Ms^2 + \mu s + k = (Ms + \mu)s + k = 0 \quad \Longrightarrow \quad \frac{1}{s(Ms+\mu)} = -\frac{1}{k}$$

Varying the damping coefficient; let's suppose this time $\mu \in [0, +\infty)$, and assume M and k as constant parameters

$$p(s,\mu) = Ms^2 + \mu s + k = (Ms^2 + k) + \mu s = 0 \quad \Longrightarrow \quad \frac{s}{Ms^2+k} = -\frac{1}{\mu}$$

Important Remarks

- in the first scenario, as k increases (the spring becomes stiffer and stiffer), the poles are complex with constant real part, while the imaginary part becomes larger;
- in the second scenario, the poles become real as μ increases, and a slow dominant dynamics arises.

```
function criticalPoints = DetermineCriticalPointsRL(sysLs)
% criticalPoints = DetermineCriticalPointsRL(sysLs) determines the critical points
% of the root locus of the SISO system described by the transfer function sysLs,
% solving the equation (**ALT)
% INPUT: the SISO transfer function sysLs
% OUTPUT: criticalPoints <--> array of complex values, representing the
%                             critical points of both the Direct and
%                             Inverse Root Loci
%
% example:
% s = tf('s');
% Ls = 1/(s+1)/(s+11);
% cP = DetermineCriticalPointsRL(Ls) % cP = -6
% ------------------------------------------
    if ~issiso(sysLs)
        error('DetermineCriticalPointsRL: only applicable to SISO systems');
    end % if
    % now surely it is a SISO system described through transfer function
    %
    zL = zero(sysLs); % compute zeros and poles of sysLs
    pL = pole(sysLs);
    % let's determine the expression of the critical point equation
    [numP, denP]= tfdata(sysLs,'v'); % let's extract the polynomials
                                     % from the given transfer function
    pL = roots(denP);
    [PC_EQ, ~] = polyder(numP, denP);% compute the expression (**ALT)
    % and solve it
    PCcandidates = roots(PC_EQ);
    preliminaryCP = [];
    % Is there any pole into PCcandidates? If YES, then it is a critical
    % point for sure --> let's add to the set criticalPoints and remove
    % from the set PCcandidates
    [poleCP, poleID, ~] = intersect(PCcandidates, pL);
    if ~isempty(poleCP)
        preliminaryCP = [preliminaryCP, poleCP];
        PCcandidates(poleID) = [];
    end % if
    % Is there any zero into PCcandidates? If YES, then it is a critical
    % point for sure --> let's add to the set criticalPoints and remove
    % from the set PCcandidates
    [zeroCP, zeroID, ~] = intersect(PCcandidates, zL);
    if ~isempty(zeroCP)
        preliminaryCP = [preliminaryCP, zeroCP];
        PCcandidates(zeroID) = [];
    end % if
    LsVALS = polyval(numP, PCcandidates)./polyval(denP, PCcandidates);
    if ~isempty(imag(LsVALS))
        realCHK = (abs(imag(LsVALS))<=1e-10);
        if any(realCHK)
            LsVALS(realCHK) = real(LsVALS(realCHK));
        end % if
```

```
        realFLAG = realCHK;
    else
        realFLAG = false(size(realCHK));
    end % if
    % let's evaluate the transfer function for each critical point candidate
    %  bar_s is a critical point iff L(bar_s) is real
    if any(realFLAG)
        criticalPoints = PCcandidates(realFLAG);
    else
        criticalPoints = [];
    end % if
    criticalPoints = [criticalPoints , preliminaryCP];
end % function
```

```
function cP_DLset = checkCriticPointsDL(criticalPoints, Ls)
% given the set of critical points criticalPoints, corresponding
% to the open-loop transfer function Ls, the function
% ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,k
% checkCriticPointsDL(criticalPoints, L1s) returns the subset of critical
% points belonging to the Direct Root Locus

    zL = zero(Ls); % compute zeros and poles of sysLs
    pL = pole(Ls);
    [numP, denP]= tfdata(Ls,'v'); % let's extract the polynomials
                                     % from the given transfer function
    preliminaryCP = [];
    % Is there any pole into criticalPoints? If YES, then it ºªm,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, a critical
    % point for sure --> let's add to the set criticalPoints and remove
    % from the set PCcandidates
    [poleCP, poleID, ~] = intersect(criticalPoints, pL);
    if ~isempty(poleCP)
        preliminaryCP = [preliminaryCP, poleCP];
        criticalPoints(poleID) = [];
    end % if
    % Is there any zero into PCcandidates? If YES, then it is a critical
    % point for sure --> let's add to the set criticalPoints and remove
    % from the set PCcandidates
    [zeroCP, zeroID, ~] = intersect(criticalPoints, zL);
    if ~isempty(zeroCP)
        preliminaryCP = [preliminaryCP, zeroCP];
        criticalPoints(zeroID) = [];
    end % if
    LsVALS = polyval(numP, criticalPoints)./polyval(denP, criticalPoints);
    D_RL_FLAG = (LsVALS < 0); % the root locus eq. is L(s) = -1/rho
                              % thus if rho >0 then L(s) is a negative real
                              % value
    if any(D_RL_FLAG)
        cP_DLset = criticalPoints(D_RL_FLAG);
    else
        cP_DLset = [];
    end % if
    cP_DLset = [cP_DLset , preliminaryCP];
```

```
end % function
```