# A Collection of Matlab Routines for Control System Analysis and Synthesis

## First edition

Gustafsson, Kjell; Lilja, Mats; Lundh, Michael

1990

CODEN: LUTFD2/(TFRT-7454)/1-16/(1990)

# A Collection of Matlab Routines for Control System Analysis and Synthesis

Kjell Gustafsson
Mats Lilja
Michael Lundh

First edition

Department of Automatic Control
Lund Institute of Technology
July 1990

**Department of Automatic Control**
**Lund Institute of Technology**
P.O. Box 118
S-221 00 Lund Sweden

*Document name*
INTERNAL REPORT

*Date of issue*
July 1990

*Document Number*
CODEN: LUTFD2/(TFRT–7454)/1-16/(1990)

*Author(s)*
Kjell Gustafsson, Mats Lilja, Michael Lundh

*Supervisor*

*Sponsoring organisation*

*Title and subtitle*
A Collection of Matlab Routines for Control System Analysis and Synthesis

*Abstract*

A collection of Matlab routines for control system analysis and synthesis is described. The routines have evolved during several years of frequent Matlab use. The report includes a brief description of all the routines and examples of their intended use.

*Key words*
control system analysis, controller design, frequency response, LQG, pole placement, root locus, model reduction, simulation

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*

*ISBN*

*Language*
English

*Number of pages*
16

*Recipient's notes*

*Security classification*

*The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.*

# 1. Introduction

Any devoted Matlab user soon ends up extending the available commands with new routines. Many of these are special routines for specific projects but some might be of interest also to other users. This report describes a set of routines that we believe belong to the second category. The routines have evolved during a couple of years of frequent Matlab use, and by presenting them we hope to save at least someone from redoing our work.

The number of routines is fairly large and for ease of use they have been divided into a couple of different toolboxes:

| | |
|---|---|
| FRBOX | Generating, manipulating and plotting frequency response data |
| PPBOX | Pole placement design and simulation of closed loop systems |
| RLBOX | Pole-zero and root loci plots |
| FRLSBOX | Approximation and design in the frequency domain |
| LQGBOX | LQG design of controller and estimator |

By using help directly on a box-name, e.g. `help frbox`, one gets general information on all the routines in the box. All individual routines are also extensively documented (use `help`). Some of the boxes include a *demo*-file that demonstrates how the routines may be used.

In order to make the collection of routines more useful we welcome suggestions for changes and/or new routines to include.

### How to Use the Toolboxes

The information in this subsection is specific to the setup at the Department of Automatic Control, Lund, Sweden. When setting up Matlab using

```
setup matlab
```

an environment variable `MATLABBOXES` is defined. This environment variable contains the path to all the functions described in this report. By using it together with `MATLABSYSPATH`, an environment variable containing the path to the functions in the Matlab distribution, each user can tailor his/her own `MATLABPATH`. A common setup is to have

```
setenv MATLABPATH ~/matlab:${MATLABBOXES}:${MATLABSYSPATH}
```

in the .login file. The MATLABBOXES path also includes a directory called `misc`, containing some locally customized m-files, e.g. `print`, and m-files of general interest.

FRBOX and PPBOX use a few global variables. These can be defined by executing `frbox` and ppbox, i.e. include the line

```
frbox, ppbox
```

in your `startup.m file`.

# 2. FRBOX

This is a collection of routines to calculate, plot, and manipulate the frequency response of a system. The data structure used is matrices on the form

$$\begin{pmatrix} \omega & G_1(i\omega) & G_2(i\omega) & \dots \end{pmatrix} \tag{1}$$

The first column contains a vector of frequency points $\omega$ [rad/s], and the other columns consist of frequency responses evaluated at these frequency points. The multicolumn format makes it easy to represent and manipulate several related frequency responses at the same time, e.g. several measurements of the transfer function of an unknown system, the transfer function of a system evaluated for different parameter values, or all transfer functions for a MIMO system.

The routines in FRBOX can be divided into different categories having the frequency response data structure (1) in common. Some routines generate frequency responses, some plot them, and, finally, there are routines for manipulating frequency responses.

### Generating a Frequency Response

FRBOX contains the following commands for generating frequency responses

| | |
|---|---|
| `frc` | Frequency response from continuous-time polynomial description, $G(s) = b(s)/a(s)e^{-s\tau}$ |
| `frd` | Frequency response from discrete-time polynomial description, $H(z) = b(z)/a(z)$ |
| `frcss` | Frequency response from continuous-time state space description, $G(s) = (C(sI - A)^{-1}B + D)e^{-s\tau}$ |
| `frdss` | Frequency response from discrete-time state space description, $H(z) = C(zI - A)^{-1}B + D$ |
| `frcsys` | Frequency response from continuous-time system description, as `frcss` but $A, B, C, D$ given as one system matrix |
| `frdsys` | Frequency response from discrete-time system description, as `frdss` but $A, B, C, D$ given as one system matrix |
| `frcpid` | Frequency response of continuous-time P, PI, PD, and PID controller with and without filter on the derivative part |
| `frdpid` | Frequency response of discrete-time P, PI, PD, and PID controller with and without filter on the derivative part |
| `frtust` | Frequency response of the Tustin approximation of a continuous-time system on polynomial form |
| `frcasymp` | The Bode amplitude asymptotes from continuous-time polynomial description |
| `svcss` | Singular value response from continuous-time state space description |
| `svcsys` | Singular value response from continuous-time system description |

As an example of how to use the routines, consider a call to `frc`

```
fr = frc(b,a,tau,lgw1,lgw2,n)
```

After this call `fr` consists of two columns. The first contains n logarithmically spaced points between $10^{lgw1}$ and $10^{lgw2}$, and the second the corresponding values of $b(i\omega)/a(i\omega)e^{-i\omega\tau}$. The argument n is optional with default value equal to 50. If the calling sequence

```
fr = frc(b,a,tau,wvec)
```

is used, the response is instead calculated for the frequency points in `wvec`.

When using frc it is possible to have several rows in b, a, and tau. For each row $k$ the response of $b_k(i\omega)/a_k(i\omega)e^{-i\omega\tau_k}$ is calculated and stored in column $k+1$ of fr.

The other routines have similar syntax. Typically the first arguments are used to define the system, e.g. state space matrices, numerator/denominator, and the last ones to define the frequency points. In the case of discrete-time systems lgw2 can be omitted or supplied as [ ], and it then defaults to half the sampling frequency.

### Plotting a Frequency Response

FRBOX includes routines for plotting a frequency response in many different formats. The input is frequency responses on the form (1), and the following types of plot formats can be produced

| | |
|---|---|
| ampl | Make a new amplitude plot |
| amsh | Show an amplitude curve in a previous amplitude plot |
| amgrid | Plot grid and mark the unit gain line in an amplitude plot |
| phpl | Make a new phase plot |
| phsh | Show a phase curve in a previous phase plot |
| phgrid | Plot grid and mark the $-180°$ line in a phase plot |
| bopl | Make a new Bode plot |
| bosh | Show a Bode curve in a previous Bode plot |
| bogrid | Plot grid and mark the unit gain and the $-180°$ line in a Bode plot |
| nipl | Make a new Nichols plot |
| nish | Show a Nichols curve in a previous Nichols plot |
| nigrid | Plot grid and mp-circles in a Nichols plot |
| nypl | Make a new Nyquist plot |
| nysh | Show a Nyquist curve in a previous Nyquist plot |
| nygrid | Plot grid and mark real and imaginary axis in a Nyquist plot |
| evpl | Make an evaluation plot (four different plots that helps evaluating a closed loop system) |
| evsh | Show an evaluation curve in a previous evaluation plot |
| evgrid | Plot grid and mark unit gain line in an evaluation plot |

Consider amplitude plots as an example of how to use the routines. ampl is called using the format

```
ampl(fr1,fr2,fr3,fr4,scale)
```

and an amplitude plot will be done using the frequency responses in fr1 – fr4. The responses fr1 – fr4 may contain different number of columns and/or frequency points. Both fr2 – fr4 and scale are optional. If scale is omitted the plot scale will be chosen such that all the responses are fully visible on the screen.

After an amplitude plot has been done amsh can be used to add new curves. The calling format is

```
amsh(fr,option)
```

causing the amplitude of the responses in `fr` to be added to the plot currently on screen. If the optional argument `option` is submitted it will be used as plot option. A grid and a line marking unit gain can be added to an amplitude plot using `amgrid`.

The routines for Bode and phase plots are completely analogous to the amplitude plot routines. The routines for Nyquist and Nichols are also similar, but they include one extra argument: `wmark`. As an example take

```
nipl(fr1,fr2,fr3,fr4,wmark,scale)
```

The parameter `wmark` specifies which frequency points to mark. If `wmark` is empty, e.g. [ ], no points are marked. The special choice `wmark = '125'` marks the 1, 2, and 5 point in each decade.

The evaluation plot routine `evpl` differs from the other plot routines. It takes the frequency response of a controller and a process as input and produce four plots based on that. These plots make it possible to quickly compare different designs in terms of transfer functions from reference value to output, from load disturbance to output, and from measurement noise to control signal.

### Manipulating a Frequency Response

FRBOX includes some routines for manipulating frequency responses on the format (1)

| | |
|---|---|
| `finv` | Invert a frequency response |
| `fadd` | Add two frequency responses |
| `fsub` | Subtract one frequency response from another |
| `fmul` | Multiply two frequency responses |
| `fdiv` | Divide one frequency response with another |
| `fclose` | Calculate closed loop frequency response from open loop response |
| `fopen` | Calculate open loop frequency response from closed loop response |
| `fsens` | Calculate sensitivity function from open loop frequency response |

The manipulation routines are all rather straight forward to use. The only thing worth commenting is how frequency response data with several columns is treated. As an example take the call

```
fr = fadd(fr1,fr2)
```

If `fr1` and `fr2` contain the same number of frequency responses, then each single response in `fr1` is added to the corresponding response in `fr2`. If on the other hand `fr1` or `fr2` contain only one response this one is used for all the responses in the other variable.

### Extracting Data from a Frequency Response

It might be necessary to interpolate in a frequency response in order to find points of interest. One situation is when calculating the amplitude and phase margins. To facilitate these operations FRBOX includes the following routines

| | |
|---|---|
| `fpick` | Pick out points from a frequency response |
| `levcross` | Compute level crossings in a table |
| `ampcross` | Compute frequencies of amplitude level crossing |

| | |
|---|---|
| `phacross` | Compute frequencies of phase level crossing |
| `fmarg` | Calculate amplitude and phase margin from frequency response |
| `bandwidth` | Compute the bandwidth from a frequency response |

The routine `levcross` is a general routine for calculating points (using interpolation) where a data vector cross a certain level. This routine is then used as a subroutine by most of the other routines.

### Frequency Response Format Conversion

The System Identification Toolbox in Matlab uses a different data format than the one in FRBOX (1). Use the following routine to translate between the two formats.

| | |
|---|---|
| `id2fr` | Convert a System Identification Toolbox frequency file to FRBOX format |

### Miscellaneous Information

FRBOX uses a global variable `glob_scales` to store the scales of different subplots. This variable is declared global by executing FRBOX. If a plot routine is called without previously having defined `glob_scales` a message is printed on the screen.

All frequency plotting scales and markings in Nyquist and Nichols plots are done in Hz instead of rad/s if the variable `glob_hz` has a value different from 0 or [ ].

### Bugs

Sometimes different (and erroneous) scaling is used on the screen and in the meta file. It is due to the way MATLAB handles its automatic scaling. Normally the problem can be avoided by using a larger plot window on screen.

MATLAB sometimes chooses an automatic scaling that one cannot get through the axis command. It is then impossible to add new curves with the `*sh` commands. The problem can be avoided by specifying a scale in the `*pl` command or manipulating `glob_scales`.

When using routines that plot in several plot windows you will be left in a window different from 111. This is an inconvenience but has to be done since `subplot(111)` empties the plot buffer, i.e. no hard copies.

## 3. PPBOX

This is a collection of routines for pole placement design and simulation of continuous-time or discrete-time closed loop systems. In addition to this there are routines for polynomial creation and manipulation.

A system is represented as a fraction of polynomials

$$G = \frac{B}{A} \tag{2}$$

where $B$ and $A$ may be interpreted as continuous-time polynomials in the Laplace operator $s$ or as discrete-time polynomials in the forward shift operator

$q$. The data structure for a polynomial $P$ is a row vector with the polynomial coefficients. A transfer function is then described by two row vectors.

It is sometimes of interset to consider transfer functions where one or more parameters may take different values. Such a family of transfer functions is represented by two matrices $BB$ and $AA$. They have equal number of rows and each pair of row vectors $BB(j,:)$ and $AA(j,:)$ represents a transfer function.

### Polynomials

PPBOX contains the following routines for creation and manipulation of polynomials and systems.

| | |
|---|---|
| `addpoly` | Add two polynomials, $P(\cdot) = P_1(\cdot) + P_2(\cdot)$ |
| `polyc` | Create continuous-time polynomial, $P(s) = \prod_{k=1}^{n_c} \left(s^2 + 2\omega_k\zeta_k s + \omega_k^2\right) \prod_{k=1}^{n_r} (s + r_k)$ |
| `polybutt` | Create continuous-time Butterworth polynomial. |
| `polybess` | Create continuous-time Bessel polynomial. |
| `pade` | Pade approximation of time delay, $G(s) = B(s)/A(s) \approx e^{-s\tau}$ |
| `sample` | Sampling of a continuous-time system, $G(s) = B_c(s)/A_c(s)$ to yield $H(q) = B_d(q)/A_d(q)$ |
| `polyc2d` | Mapping of continuous-time characteristic polynomial $A_c(s)$ to discrete-time counterpart $A_d(q)$. |
| `mksysp` | Defines a family of transfer functions. The system is defined by $$G(s) = \frac{B_p(s)B_0(s)}{A_p(s)A_0(s)}$$ where the coefficients of $B_0$ and $A_0$ are known accurately and the coefficients of $B_p$ and $A_p$ belong to intervals $b_i \in b_{i0} \pm \delta b_i$ and $a_i \in a_{i0} \pm \delta a_i$. Using the intervals where the $j$ uncertain coefficients belong, $2^j$ different transfer functions are formed, each representing a corner of the convex set in coefficient space that defines the transfer function family. |

These routines may also be useful together with FRBOX.

### Transfer Function Manipulation

Some routines to manipulate transfer functions are included.

| | |
|---|---|
| `gadd` | Calculate the sum of two rational transfer functions |
| `stabpartc` | Separate a continuous-time rational transfer function into stable and unstable partial fractions |
| `stabpartd` | Separate a discrete-time rational transfer function into stable and unstable partial fractions |

In each case a rational function is represented by the corresponding pair of polynomials. The stability regions used in `stabpartc` and `stabpartd` are the open left half plane and the open unit circle respectively.

### Polynomial Synthesis

Two routines for polynomial synthesis are available.

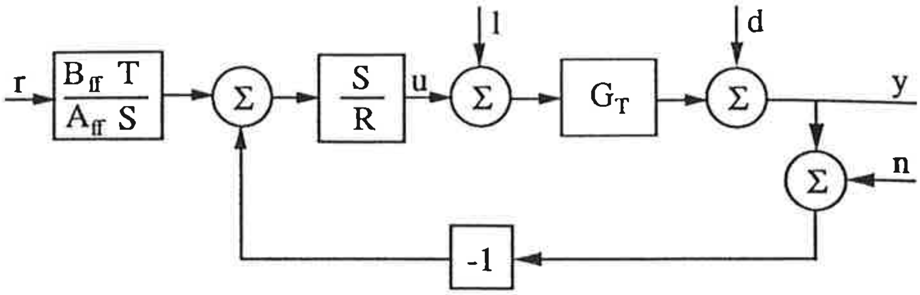


**Figure 1.** The Simulated Closed Loop System

| | |
|---|---|
| `rstc` | Polynomial synthesis for continuous-time systems |
| `rstd` | Polynomial synthesis for discrete-time systems |

They are identical with exception for how the DC-gain is normalized. A controller $Ru = Tr - Sy$ is calculated when the open system is defined by the transfer function $G = B^+B^-/A$. The polynomial $B^+$ will be cancelled and must therefore be stable and well damped. The closed loop characteristic polynomial is $A_m A_o$ where the poles in $A_o$ are cancelled in the transfer function from reference to output. This transfer function equals $G_m = kB^- B_{m1}/A_m$, where $k$ is chosen to make $G_m(0) = 1$ in the continuous-time case and $G_m(1) = 1$ in the discrete-time case. It is possible to force $R$ and $S$ to include certain factors. These are specified by `ar` and `as`. The routine `rstc` is called by

```
[r,s,t] = rstc(bplus,bminus,a,bm1,am,ao,ar,as)
```

and `as` or both `ar` and `as` may be omitted. The call to `rstd` is similar.

### Simulation and plotting

PPBOX also includes routines for simulation. Only pure continuous-time systems or pure discrete-time systems can be simulated. Simulation of a continuous-time system controlled by a discrete-time controller should be performed in SIMNON.

| | |
|---|---|
| `yusimc` | Simulation of continuous-time systems |
| `yusimd` | Simulation of discrete-time systems |
| `yusignals` | Generation of signals affecting the closed loop system |
| `yustairs` | Prepare signals for stair-step plots for discrete-time systems |
| `yupl` | Plot of time response in a new diagram |
| `yush` | Plot of time response in an existing diagram |

The simulation routines `yusimc` and `yusimd` simulate a closed loop system with four inputs and two outputs. The input signals are reference, input disturbance, output disturbance and measurement noise. The output signals are the process output and the control signal. The closed loop system is found in figure 1. Simulation of a continuous-time closed loop system is performed by the call

```
tryu = yusimc(bb,aa,r,s,t,bff,aff,trldn)
```

The open system $G_T = BB/AA$ is controlled by a two degree of freedom controller, that is given by

$$Ru = T\left(\frac{B_{ff}}{A_{ff}}\right) r - Sy \tag{3}$$

The open loop system is specified by the matrices bb and aa. If these only have one row each the simulation is done for this single system. If bb has $n$ rows and aa has 1 row then $n$ systems bb(i,:)/a are simulated with the same controller, and similarly if bb has 1 row and aa has $n$ rows. If bb and aa both have $n$ rows then $n$ systems bb(i,:)/aa(i,:) are simulated. This is convenient for simulation of systems with parametric uncertainty. The output tryu from yusimc and yusimd is a matrix where the columns should be interpreted as

$$(time \quad r \quad y_1 \quad u_1 \quad y_2 \quad u_2 \quad \cdots) \tag{4}$$

where $r$ is the reference input and each pair $(y_i, u_i)$ is the process output signal and the control signal for the $i$:th system. The full input to the simulation routines is a matrix with five columns defining

$$(time \quad r \quad l \quad d \quad n) \tag{5}$$

A convenient way of forming this matrix is to use yusignals. This routine has many options that are described in the help text. A simple way to use yusimc is to specify the simulation time tmax.

```
tryu = yusimc(bb,aa,r,s,t,bff,aff,tmax)
```

Then the reference signal $r = 1$, an input disturbance $l = -1$ affects the system from tmax/3, and an output disturbance $d = -1$ affects the system from tmax*2/3. No noise is present. Other options are available. For more information see the help text.

The routine yusimd has similar syntax. A parameter defining the sampling interval is optional.

The simulation result may be plotted using yupl. This routine is called using the format

```
yupl(tryu1,tryu2,tryu3,tryu4,scale)
```

giving a plot of the time responses tryu1 – tryu4. The responses tryu1 – tryu4 may contain different number of columns and/or time points. Both tryu2 – tryu4 and scale are optional. If scale is omitted the plot scale will be chosen such that all the responses are fully visible on the screen.

After a time response plot has been done yush can be used to add new curves. The calling format is

```
yush(tryu,option)
```

causing the time responses in tryu to be added to the plot currently on screen. If the optional argument option is submitted it will be used as plot option.

### File Output

In some cases it is desirable to transfer parameters to other programs, e.g. SIMNON, TOOLBOX. Some routines provide this. Text files are created on SIMNON parameter file format or on MATLAB .m script-file format.

| | |
|---|---|
| rst2sim | Write simnon parameter file with RST-regulator. |
| p2sim | Write simnon parameter file with polynomials. |

| | |
|---|---|
| `p2mat` | Write matlab .m script-file with polynomials. |

### Miscellaneous Information

A global variable (`glob_scale`) is used. It is defined by executing PPBOX or FRBOX.

### Bugs

The use of noise when simulating a continuous-time system is not correct since the noise is only present at the instants when outputs are calculated. The same scaling problems occur here as in FRBOX.

## 4. RLBOX

RLBOX contains routines to make pole-zero plots, and to calculate and plot root loci.

### Making Pole-zero Plots

| | |
|---|---|
| `pzpl` | Make a new pole-zero plot |
| `pzsh` | Show poles-zeros in a previous pole-zero plot |
| `pzgrid` | Make grid and plot the real and imaginary axis or the unit circle in a pole-zero plot |
| `mark` | A general routine for plotting and indexing markings |

The pole-zero plot routines take two polynomials defining a transfer function as input, calculate the poles and zeros, and finally plots them using `mark`. The relation between `pl`, `sh`, and `grid` is similar to the plot routines in FRBOX. Normally `pzpl` makes a square plot with equal scales on the x and y axis, but by supplying a scale argument (optional) this can be changed.

### Calculating and Plotting Root Loci

| | |
|---|---|
| `rootlocus` | Calculate root locus |
| `rloc1` | Plot full root locus including start and end points |
| `rloc2` | Plot root locus around a nominal point |
| `symloc` | Plot stable part of symmetric LQ locus for continuous-time system |
| `dsymloc` | Plot stable part of symmetric LQ locus for discrete-time system |

The routine `rootlocus` uses the implicit function theorem in an attempt to make the roots not vary to much between consecutive $k$-values. `rootlocus` is seldom used directly, but functions rather as a subroutine for the other root loci plot routines.

The routines `symloc` and `dsymloc` are used to plot the so called LQ root locus, i.e. the closed loop poles one gets when solving the LQ problem minimizing

$$\int_0^\infty \left(y^2(t) + \rho u^2(t)\right)\, dt.$$

Normally the root loci plot routines make a square plot with equal scales on the x and y axis. This can be changed by supplying a scale argument (optional).

**pzgrid**, **pzsh**, and **mark** can be used to make a grid and/or mark specific points in a root locus plot. The Matlab routine **zgrid** may be of interest when plotting poles-zeros of a discrete-time system.

# 5. FRLSBOX

This is a collection of routines for calculating transfer functions and controllers by using least squares fitting in the frequency domain. Functions for optimal Hankel norm approximation and Padé approximation are also included.

### Least Squares Approximation

| | |
|---|---|
| `lsbac` | Fitting a rational function $B(s)/A(s)$ to a frequency response (continuous-time version). |
| `lsbad` | Fitting a rational function $B(z)/A(z)$ to a frequency response (discrete-time version). |
| `lsbatau` | Fitting $e^{-\tau s}B(s)/A(s)$ to a frequency response |
| `lsrstc` | Calculation of a controller of type $Ru = -Sy + Tr$ by least squares fitting to a closed loop transfer function (continuous-time version). |
| `lsrstd` | Calculation of a controller of type $Ru = -Sy + Tr$ by least squares fitting to a closed loop transfer function (discrete-time version). |
| `bafit` | Fitting a rational function to a frequency response. Used by `lsbac` and `lsbad`. |
| `rstfit` | Calculation of a controller of type $Ru = -Sy + Tr$ by least squares fitting to a closed loop transfer function. Used by `lsrstc` and `lsrstd`. |

The function `bafit` computes the rational function $B(s)/A(s)$ which minimizes the loss function

$$J(B,A) = \sum_{k=1}^{N} W_k \left| A(z_k) g_k - B(z_k) \right|^2$$

where $z_k$ are the approximation points and $g_k$ are the corresponding complex frequency response values. In `lsbac` the approximation points are given by $z_k = i\omega_k$ where $\omega_k$ are the frequencies of approximation (continuous-time). The discrete-time version is called `lsbad` where $z_k = e^{i\omega_k}$. Both `lsbac` and `lsbad` uses the FRBOX frequency response format. A generalization of `lsbac` is found in `lsbatau` where a time delay is included in the model, i.e. the loss function is modified to

$$J(B,A,\tau) = \sum_{k=1}^{N} W_k \left| A(i\omega_k) g_k - e^{-i\omega_k \tau} B(i\omega_k) \right|^2$$

This gives a non-quadratic problem which is solved iteratively.

Least squares approximation is also used in `rstfit` where the parameters in the control law $Ru = -Sy + Tr$ are computed to minimize the loss function

$$J(R,S,T) = \sum_{k=1}^{N} |E(z_k)|^2$$

where

$$E(s) = \frac{G_m(s) - G_{cl}(s)}{G_{cl}(s)}$$

is the relative closed loop model error. The desired closed loop transfer function is $G_m$ and the actual closed loop transfer function is given by

$$G_{cl} = \frac{G(s)T(s)}{R(s) + G(s)S(s)}$$

where $G(s)$ is the process transfer function. In `lsrstc` and `lsrstd` the approximation points are specialized to $z_k = i\omega_k$ (continuous-time) and $z_k = e^{i\omega_k}$ (discrete-time), respectively.

### Transfer Function Approximation

| | |
|---|---|
| `hankelu` | Unweighted Hankel norm approximation of $B(s)/A(s)$ |
| `hankelw` | Weighted Hankel norm approximation of $B(s)/A(s)$ |
| `padeappr` | Padé approximation of any order of $e^{-Ts}B(s)/A(s)$ |
| `sylvester` | Compute a Sylvester matrix of two polynomials |

Approximation of stable rational transfer functions by minimization of the Hankel norm of the error is performed by `hankelu` and `hankelw`. These functions computes the optimal approximation directly from the coefficients of the transfer function $B(s)/A(s)$ without transforming the problem to state space. The reason to have a separate function for the unweighted case is that some unnecessary computations are eliminated to get the answer quicker.

The function padeappr computes the Padé approximant of specified order of a function of type

$$G(s) = e^{-Ts}\frac{B(s)}{A(s)}$$

The function `sylvester` generates the Sylvester matrix of specified order to two polynomials. This function is used by `hankelu`, `hankelw` and padeappr.

### Lead Compensator Design

| | |
|---|---|
| `leadcomp` | Compute a lead compensator |

Given a frequency response, a lead compensator is calculated such that the cut off frequency (the frequency for which the magnitude passes 1 from above) is increased by a certain factor. The compensator consists of a number of identical first (or second) order filters. This number is determined by the phase lead required to increase the cut off frequency by the specified factor.

# 6. LQGBOX

This collection of routines is written to facilitate the design of continuous-time and discrete-time LQ(G) controllers. They not only solve the LQ(G) problem but also provide output that help calculating the resulting controller. In contrast to the routines in the control toolbox of MATLAB cross terms are allowed in the loss function. The routines are also written such that it is easy to change Riccati equation solver.

### Regulator and Estimator Design

| | |
|---|---|
| `lqrc` | Continuous-time linear quadratic regulator |
| `lqrd` | Discrete-time linear quadratic regulator |
| `lqec` | Continuous-time linear quadratic estimator |
| `lqed` | Discrete-time linear quadratic estimator |

These routines solve the linear quadratic control/estimator problem for continuous-time and discrete-time systems. They also calculate variables that are needed when calculating the final controller. Take as an example `lqrc`

```
[L,lr,S] = lqrc(A,B,C,D,Q1,Q2,Q12)
```

The routine calculates $L$ such that the control law $u(t) = -Lx(t)$ minimizes

$$\int_0^\infty \left(x^T(t)Q_1x(t) + 2x^T(t)Q_{12}u(t) + u^T(t)Q_2u(t)\right)\, dt$$

for the system

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t)\\ y(t) &= Cx(t) + Du(t)\end{aligned}$$

Strictly speaking, the matrices $C$ and $D$ are not needed to solve the LQ-problem, but using them the routine can also calculate $l_r$ such that $u(t) = l_r y_r(t) - Lx(t)$ gives steady state gain equal to one from $y_r(t)$ to $y(t)$.

Similarly, the routine `lqed` provides output that makes it is easy to design Kalman filters both with and without direct term.

### Calculation of Complete Controller

| | |
|---|---|
| `lqgc` | Complete continuous-time controller from `lqrc`, `lqec` results |
| `lqgd` | Complete discrete-time controller from `lqrd`, `lqed` results |

After having designed both the feedback law and the estimator one needs to connect them to get the final controller. The two routines `lqgc` and `lqgd` take the output from the control/estimator design routines and calculate a state space description of the resulting controller.

### Riccati Equation Solver

| | |
|---|---|
| `care` | General dispatch routine for continuous-time Riccati solvers |

`dare` General dispatch routine for discrete-time Riccati solvers

The usefulness of a set of LQG design routines hinges on the quality of the Riccati equation solver. All the routines in LQGBOX that needs to solve such an equation will call either `care` (continuous-time algebraic Riccati equation) or `dare` (discrete-time algebraic Riccati equation). These two routines work as dispatch routines. They examine the global variable `caretype` (or `daretype`) and then call the corresponding solver. If `caretype` (or `daretype`) are undefined the default solver is called.

The number of different solvers currently implemented are not especially impressive, but due to the structure of `care` and `dare` it is easy to add new ones. The solvers currently available can be found by doing `help` on `care` and `dare`.

### Sampling of Continuous-time Loss Function

`lqgsamp` Samples loss function and continuous-time noise description

When designing a discrete-time controller using LQG one needs a discrete-time loss function. Often it may be more natural to define the specifications as a continuous-time loss function, and then translate it to discrete-time. This can be done using `lqgsamp`.

### Miscellaneous

Currently there is a name conflict between `lqrc` in LQGBOX and a similar routine in the robust toolbox by Safanov.

## 7. References

Matlab is described in

THE MATHWORKS (1990): *Pro-Matlab, User's Guide.*

The pole placement design (PPBOX) and the LQG-design (LQGBOX) are described in

ÅSTRÖM, K. J., and WITTENMARK, B. (1990): *Computer Controlled Systems, Theory and Design, 2nd ed.*, Prentice-Hall, Englewood Cliffs.

The following reference contains some examples on using root loci for control system design. The symmetric root locus is also treated.

FRANKLIN, G. F, POWELL, J. D., and EMAMI-NAEINI, A. (1986): *Feedback Control Systems,* Addison-Wesley.

The design methods implemented in FRLSBOX are developed and described in

LILJA, M. (1989): *Controller Design by Frequency Domain Approximation*, PhD Thesis, TFRT-1031, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

The routines are used for practical controller design in

Gustafsson, K. and Bernhardsson, B. (1990): "Control Design for Two Lab-processes: The Flexible Servo, The Fan and the Plate," Internal Report, TFRT-7456, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.