A Collection of Matlab Routines for Control System Analysis and Synthesis

The Code

Gustafsson, Kjell; Lilja, Mats; Lundh, Michael

1990

CODEN: LUTFD2/(TFRT-7455)/1-133/(1990)

A Collection of Matlab Routines for Control System Analysis and Synthesis

Kjell Gustafsson
Mats Lilja
Michael Lundh

The code

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-7455)/1-133/(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 — The Code

Abstract

A collection of Matlab routines for control system analysis and synthesis is listed. The routines have evolved during several years of frequent Matlab 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
119

Security classification

Recipient's notes

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. FRBOX

```
-------------------- /regler/matlab/frbox/amgrid.m --------------------

function amgrid
% AMGRID Plot grid and mark unity gain in an amplitude plot.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:44:33 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

hold on;
loglog([1E-10; 1E10],[1; 1]);            % unity gain line
grid;
hold off;

-------------------- /regler/matlab/frbox/ampcross.m --------------------

function [omega,pha] = ampcross(fr,level)
% AMPCROSS Computation of amplitude level crossing frequencies
%
%       OMEGA = AMPCROSS(FR,LEVEL)
%
%       Given a frequency response FR, all frequencies for which
%       the amplitude crosses LEVEL are computed.
%
%       [OMEGA,PHA] = AMPCROSS(FR,LEVEL)
%
%       gives the phase at the crossing frequencies.

% Mats Lilja
% LastEditDate : Wed Mar 14 17:11:08 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

w = fr(:,1);
g = fr(:,2);
[omega,pha] = levcross([w abs(g) arg(g)],level);


-------------------- /regler/matlab/frbox/ampl.m --------------------

function ampl(fr1,fr2,fr3,fr4,scale)
% AMPL  Plot the amplitude of a frequency response.
%
%       ampl(fr1,fr2,fr3,fr4,scale)
%
%       An amplitude plot is done from the frequency responses in fr1 - fr4.
%       The arguments fr2 - fr4 are optional. fr1 - fr4 is allowed to have
%       different number of data points and columns.
%
%       The optional argument scale is used to affect the scaling of the
%       plot. It takes the form [wmin wmax amin amax]. All values are
%       given in 10-logarithm, i.e. amin = -1, amax = 2 corresponds to an
%       amplitude scale from 0.1 to 100. scale can be used even if fr2 -
%       fr4 are omitted.
%
%       The scale of the amplitude plot is stored in the variable glob_scale,
%       which must be declared as global. The scale is used to be able to
%       draw (using AMSH) consecutive plots in the same diagram.
```

```
%
%       The frequency axis will be marked in Hz if the global variable
%       glob_hz has a value differing from 0.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:45:22 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if nargin~=5
  scale = eval(['fr' int2str(nargin)]);
end
scaleinfo = all(size(scale) == [1 4]);

ind = sprintf('(:,2:%g)',size(fr1)*[0;1]);
if glob_hz,
  plotamp = ['loglog(fr1(:,1)/(2*pi),abs(fr1' ind ')'];
else
  plotamp = ['loglog(fr1(:,1),abs(fr1' ind ')'];
end;

for k=2:nargin-scaleinfo,
  kstr = int2str(k);
  eval(['ind = sprintf(''(:,2:%g)'',size(fr' kstr ')*[0;1]);'])
  if glob_hz,
    plotamp = [plotamp ',fr' kstr '(:,1)/(2*pi),abs(fr' kstr ind ')'];
  else
    plotamp = [plotamp ',fr' kstr '(:,1),abs(fr' kstr ind ')'];
  end
end
plotamp = [plotamp ');'];

subplot(111)
if scaleinfo, axis(scale); end
eval(plotamp);
glob_scale = axis;
if ~scaleinfo, axis; end

ylabel('Magnitude');
if glob_hz,
  xlabel('Frequency [Hz]');
else
  xlabel('Frequency [rad/s]');
end
```

-------------------- /regler/matlab/frbox/amsh.m --------------------

```
function amsh(fr,option)
% AMSH  Show the amplitude of a frequency response in a previous plot.
%
%       amsh(fr,option)
%
%       An amplitude plot is done from the frequency responses in fr using
%       the plotoption option. The argument option is optional.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:45:44 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN
```

```
if frcheck, return, end

if ~any(size(glob_scale)==[1 4])
  error('No previous amplitude plot');
end;

hold on;
ind = size(fr)*[0;1];
if nargin == 1,
  if glob_hz,
    loglog(fr(:,1)/(2*pi),abs(fr(:,2:ind)));
  else
    loglog(fr(:,1),abs(fr(:,2:ind)));
  end
elseif nargin == 2,
  if glob_hz,
    loglog(fr(:,1)/(2*pi),abs(fr(:,2:ind)),option);
  else
    loglog(fr(:,1),abs(fr(:,2:ind)),option);
  end;
end;
hold off;

-------------------- /regler/matlab/frbox/arg.m --------------------

function phase = arg(g)
% ARG   Calculate the argument of g.
%
%       phase = arg(g)
%
%       The argument is given in degrees and an attempt is done to keep
%       it continuous over the transitions at 180 and -180 degrees.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:46:27 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

phase = unwrap(angle(g))*180/pi;

-------------------- /regler/matlab/frbox/bandwidth.m --------------------

function omega_b = bandwidth(fr)
% BANDWIDTH Computes bandwidth from a frequency response
%
%       OMEGA_B = BANDWIDTH(FR)
%
%       Given a frequency response FR, the frequency OMEGA_B at which
%       the amplitude has dropped 3dB compared to the stationary gain.
%       The stationary gain is taken as the amplitude at the first
%       frequency in FR.

% Mats Lilja
% LastEditDate : Wed Mar 14 14:42:49 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

omegas = levcross([fr(:,1) abs(fr(:,2)/fr(1,2))],1/sqrt(2));
if length(omegas) == 1,
  omega_b = omegas;
else,
```

```
  error('The bandwidth is not well-defined');
end;


-------------------- /regler/matlab/frbox/bogrid.m --------------------

function bogrid
% BOGRID Plot grid and mark unity gain and -180 degree phase in bode diagram.

% Kjell Gustafsson
% LastEditDate : Wed Mar 21 09:51:12 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

subplot(211);
axis(glob_scale(1,:));
loglog([1E-10; 1E10],[1; 1]);            % unity gain line
grid;

subplot(212);
axis(glob_scale(2,:));
semilogx([1E-10; 1E10],[-180; -180]);    % -180 degree phase line
grid;
axis;

subplot(211);   % To make 'title' appear at the top

-------------------- /regler/matlab/frbox/bopl.m --------------------

function bopl(fr1,fr2,fr3,fr4,scale)
% BOPL  Plot a bode plot.
%
%       bopl(fr1,fr2,fr3,fr4,scale)
%
%       A bode plot is done from the frequency responses in fr1 - fr4.
%       The arguments fr2 - fr4 are optional. fr1 - fr4 is allowed to have
%       different number of data points and columns.
%
%       The optional argument scale is used to affect the scaling of the
%       plot. It takes the form [wmin wmax amin amax phmin phmax]. wmin,
%       wmax, amin and amax are given in 10-logarithm, i.e. amin = -1,
%       amax = 2 corresponds to an amplitude scale from 0.1 to 100.
%       scale can be used even if fr2 - fr4 are omitted.
%
%       The scales of the amplitude and the phase plot are stored in
%       the variable glob_scale. which must be declared as global. The
%       scales are used to be able to draw consecutive plots (using BOSH)
%       in the same diagram.
%
%       The frequency axis will be marked in Hz if the global variable
%       glob_hz has a value differing from 0.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:46:48 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if nargin~=5
```

```
  scale = eval(['fr' int2str(nargin)]);
end
scaleinfo = all(size(scale) == [1 6]);

ind = sprintf('(:,2:%g)',size(fr1)*[0;1]);
if glob_hz,
  plotamp = ['loglog(fr1(:,1)/(2*pi),abs(fr1' ind ')'];
  plotpha = ['semilogx(fr1(:,1)/(2*pi),arg(fr1' ind ')'];
else
  plotamp = ['loglog(fr1(:,1),abs(fr1' ind ')'];
  plotpha = ['semilogx(fr1(:,1),arg(fr1' ind ')'];
end

for k=2:nargin-scaleinfo,
  kstr = int2str(k);
  eval(['ind = sprintf(''(:,2:%g)'',size(fr' kstr ')*[0;1]);'])
  if glob_hz,
    plotamp = [plotamp ',fr' kstr '(:,1)/(2*pi),abs(fr' kstr ind ')'];
    plotpha = [plotpha ',fr' kstr '(:,1)/(2*pi),arg(fr' kstr ind ')'];
  else
    plotamp = [plotamp ',fr' kstr '(:,1),abs(fr' kstr ind ')'];
    plotpha = [plotpha ',fr' kstr '(:,1),arg(fr' kstr ind ')'];
  end
end
plotamp = [plotamp ');'];
plotpha = [plotpha ');'];

clg;
subplot(211)
if scaleinfo, axis(scale(1,1:4)); end
eval(plotamp);
glob_scale = axis;
if ~scaleinfo, axis; end
ylabel('Magnitude');

subplot(212);
if scaleinfo, axis(scale(1,[1,2,5,6])); end
eval(plotpha);
glob_scale = [glob_scale; axis];
if ~scaleinfo, axis; end
ylabel('Phase [deg]');

if glob_hz,
  xlabel('Frequency [Hz]');
else
  xlabel('Frequency [rad/s]');
end

subplot(211);   % To make `title' appear at the top


-------------------- /regler/matlab/frbox/bosh.m --------------------

function bosh(fr,option)
% BOSH  Show a bode plot in a previously drawn bode diagram.
%
%       bosh(fr,option)
%
%       A bode plot is done from the frequency responses in fr using
%       plotoption option. The argument option is optional.
%
%       The scales of the amplitude and the phaseplot are taken from the
```

```
%       variable glob_scale. This variable is assumed having been declared
%       as global.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:46:57 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if ~all(size(glob_scale)==[2 4])
  error('No previous Bode plot');
end

ind = size(fr)*[0;1];

subplot(211);
axis(glob_scale(1,:));
if nargin == 1,
  if glob_hz,
    loglog(fr(:,1)/(2*pi),abs(fr(:,2:ind)));
  else
    loglog(fr(:,1),abs(fr(:,2:ind)));
  end
elseif nargin == 2,
  if glob_hz,
    loglog(fr(:,1)/(2*pi),abs(fr(:,2:ind)),option);
  else
    loglog(fr(:,1),abs(fr(:,2:ind)),option);
  end;
end;

subplot(212);
axis(glob_scale(2,:));
if nargin == 1,
  if glob_hz,
    semilogx(fr(:,1)/(2*pi),arg(fr(:,2:ind)));
  else
    semilogx(fr(:,1),arg(fr(:,2:ind)));
  end
elseif nargin == 2,
  if glob_hz,
    semilogx(fr(:,1)/(2*pi),arg(fr(:,2:ind)),option);
  else
    semilogx(fr(:,1),arg(fr(:,2:ind)),option);
  end;
end;
axis;

subplot(211)   % To make 'title' appear at the top

-------------------- /regler/matlab/frbox/evgrid.m --------------------

function evgrid
% EVGRID plots grid and mark unity gain on the plots done by EVPL

% Kjell Gustafsson
% LastEditDate : Wed Mar 21 16:09:36 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end
```

```
subplot(221);
axis(glob_scale(1,:));
loglog([1E-10; 1E10],[1; 1]);          % unity gain line
grid;

subplot(222);
axis(glob_scale(2,:));
loglog([1E-10; 1E10],[1; 1]);          % unity gain line
grid;

subplot(223);
axis(glob_scale(3,:));
loglog([1E-10; 1E10],[1; 1]);          % unity gain line
grid;

subplot(224);
axis(glob_scale(4,:));
loglog([1E-10; 1E10],[1; 1]);          % unity gain line
grid;

axis;

-------------------- /regler/matlab/frbox/evpl.m --------------------

function evpl(gp,gfb,gff,gl,gn,gz,scale)
% EVPL  plots different frequency responses to evaluate a closed loop system
%
%       evpl(gp,gfb,gff,gl,gn,gz,scale)
%       evpl(gp,gfb,gff,gl,gn,scale)
%       evpl(gp,gfb,gff,gl,scale)
%       evpl(gp,gfb,gff,scale)
%
%       Three plots depicting the transfer functions from r to y (or z), from
%       l to y (or z), and from n to u are plotted. The fourth plot shows
%       a measure of how much multiplicative uncertainty the loop may sustain
%       without becoming unstable.
%
%       The noise and load disturbance inputs can be weighted by supplying
%       gl and/or gn. A nonempty gz tells that the evaluation should be done
%       using z rather than y. gl, gn, and gz are all optional. The value []
%       is interpreted as 1.
%
%       The optional argument scale is used to affect the scaling of the
%       plots. It takes the form [wmin wmax amin amax]. wmin,
%       wmax, amin and amax are given in 10-logarithm, i.e. amin = -1,
%       amax = 2 corresponds to an amplitude scale from 0.1 to 100. scale
%       can also consist of four rows. Each individual row is then used for
%       its corresponding plot.
%
%       The scales of the plots are stored in the variable glob_scale,
%       which must be declared as global. The scales are used to be able
%       to draw consecutive plots (using EVSH) in the same diagram.
%
%       The frequency axis will be marked in Hz if the global variable
%       glob_hz has a value differing from 0.
%
%
%                             | l
%                             |
%                             v
%                          -------       --------
```

```
%                          |     |       |      |           z
%                          | Gl  |   --->| Gz   |------------
%                          |     |   |   |      |
%                          -------   |   --------
%                             |      |
%              ----------     |      |   ---------
%         r    |       |      v   u  |   |       |          y
%         ---->| Gff   |----->+--------->| Gp    |------------
%              |       |      ^          |       |      |
%              ----------     |          ---------      |
%                             |                         |
%                             |          ---------      |    ---------
%                             |          |       |      v    |       |   n
%                             -----------| -Gfb  |<-----+<---| Gn    |<---
%                                        |       |           |       |
%                                        ---------           ---------

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:47:19 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if nargin==7
  nogl = all(size(gl)==[0 0]);
  nogn = all(size(gn)==[0 0]);
  nogz = all(size(gz)==[0 0]);
elseif nargin==6
  nogl = all(size(gl)==[0 0]);
  nogn = all(size(gn)==[0 0]);
  if size(gz)==[1 4] | size(gz)==[4 4]
    nogz = 1;
    scale = gz;
  else
    nogz = all(size(gz)==[0 0]);
    scale = [];
  end
elseif nargin==5
  nogl = all(size(gl)==[0 0]);
  if size(gn)==[1 4] | size(gn)==[4 4]
    nogn = 1;
    scale = gn;
  else
    nogn = all(size(gn)==[0 0]);
    scale = [];
  end
  nogz = 1;
elseif nargin==4
  if size(gl)==[1 4] | size(gl)==[4 4]
    nogl = 1;
    scale = gl;
  else
    nogl = all(size(gl)==[0 0]);
    scale = [];
  end
  nogn = 1;
  nogz = 1;
else
  nogl = 1;
  nogn = 1;
  nogz = 1;
```

```
  scale = [];
end

if size(scale)==[0 0]
  scaleinfo = 0;
else
  scaleinfo = 1;
  if size(scale)==[1 4]
    scale = [scale; scale; scale; scale];
  end
end

gloop = fclose(fmul(gp,gfb));

if nogz
  gzr = fmul(gloop,fdiv(gff,gfb));
else
  gzr = fmul(fmul(gloop,fdiv(gff,gfb)),fdiv(gz,gp));
end

if nogl
  gzl = fdiv(gzr,gff);
else
  gzl = fmul(gzr,fdiv(gl,gff));
end

% The following calculations have the wrong sign, but since we only
% will plot the magnitude that does not matter.

if nogn
  gun = fdiv(gloop,gp);
else
  gun = fmul(gloop,fdiv(gn,gp));
end

grob = finv(gloop);

subplot;

subplot(221);
ind = size(gzr)*[0;1];
if scaleinfo, axis(scale(1,:)); end
if glob_hz
  loglog(gzr(:,1)/(2*pi),abs(gzr(:,2:ind)));
else
  loglog(gzr(:,1),abs(gzr(:,2:ind)));
end
glob_scale = axis;
if ~scaleinfo, axis; end

if glob_hz
  xlabel('Frequency [Hz]');
else
  xlabel('Frequency [rad/s]');
end
ylabel('Magnitude');
if nogz
  title('r --> y');
else
  title('r --> z');
end
```

```
subplot(222);
ind = size(gzl)*[0;1];
if scaleinfo, axis(scale(2,:)); end
if glob_hz
  loglog(gzl(:,1)/(2*pi),abs(gzl(:,2:ind)));
else
  loglog(gzl(:,1),abs(gzl(:,2:ind)));
end
glob_scale = [glob_scale; axis];
if ~scaleinfo, axis; end

if glob_hz
  xlabel('Frequency [Hz]');
else
  xlabel('Frequency [rad/s]');
end
ylabel('Magnitude');
if nogz
  title('l --> y');
else
  title('l --> z');
end

subplot(223);
ind = size(gun)*[0;1];
if scaleinfo, axis(scale(3,:)); end
if glob_hz
  loglog(gun(:,1)/(2*pi),abs(gun(:,2:ind)));
else
  loglog(gun(:,1),abs(gun(:,2:ind)));
end
glob_scale = [glob_scale; axis];
if ~scaleinfo, axis; end

if glob_hz
  xlabel('Frequency [Hz]');
else
  xlabel('Frequency [rad/s]');
end
ylabel('Magnitude');
title('n --> u');

subplot(224);
ind = size(grob)*[0;1];
if scaleinfo, axis(scale(4,:)); end
if glob_hz
  loglog(grob(:,1)/(2*pi),abs(grob(:,2:ind)));
else
  loglog(grob(:,1),abs(grob(:,2:ind)));
end
glob_scale = [glob_scale; axis];
if ~scaleinfo, axis; end

if glob_hz
  xlabel('Frequency [Hz]');
else
  xlabel('Frequency [rad/s]');
end
ylabel('Magnitude');
title('Robustness to mult uncert');

-------------------- /regler/matlab/frbox/evsh.m --------------------
```

```
function evsh(gp,gfb,gff,gl,gn,gz,option)
% EVSH  shows new frequency responses in a previous plot by EVPL
%
%       evsh(gp,gfb,gff,gl,gn,gz,option)
%       evsh(gp,gfb,gff,gl,gn,option)
%       evsh(gp,gfb,gff,gl,option)
%       evsh(gp,gfb,gff,option)
%
%       Three plots depicting the transfer functions from r to y (or z), from
%       l to y (or z), and from n to u are plotted. The fourth plot shows
%       a measure of how much multiplicative uncertainty the loop may sustain
%       without becoming unstable.
%
%       The noise and load disturbance inputs can be weighted by supplying
%       gl and/or gn. A nonempty gz tells that the evaluation should be done
%       using z rather than y. gl, gn, and gz are all optional. The value []
%       is interpreted as 1.
%
%       The curves are plotted using the optional argument option. The plot
%       scales are taken from the global variable glob_scale.
%
%       The frequency axis will be marked in Hz if the global variable
%       glob_hz has a value differing from 0.
%
%                             | 1
%                             |
%                             v
%                          -------       --------
%                           |    |       |      |           z
%                           | Gl |   --->| Gz   |-------------
%                           |    |   |   |      |
%                          -------   |   --------
%                             |      |
%              ---------      |      |   ---------
%         r    |       |      v   u  |   |       |          y
%         ---->|  Gff  |----->+--------->|  Gp   |------------
%              |       |      ^          |       |      |
%              ---------      |          ---------      |
%                             |                         |
%                             |          ---------      |    ---------
%                             |          |       |      v    |       |   n
%                             -----------| -Gfb  |<-----+<---|  Gn   |<---
%                                        |       |           |       |
%                                        ---------           ---------

% Kjell Gustafsson
% LastEditDate : Thu Mar 15 14:59:51 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if ~all(size(glob_scale)==[4 4])
  error('No previous evaluation plot');
end

if nargin==7
  nogl = all(size(gl)==[0 0]);
  nogn = all(size(gn)==[0 0]);
  nogz = all(size(gz)==[0 0]);
elseif nargin==6
```

```
  nogl = all(size(gl)==[0 0]);
  nogn = all(size(gn)==[0 0]);
  if isstr(gz)
    nogz = 1;
    option = gz;
  else
    nogz = all(size(gz)==[0 0]);
    option = '0';
  end
elseif nargin==5
  nogl = all(size(gl)==[0 0]);
  if isstr(gn)
    nogn = 1;
    option = gn;
  else
    nogn = all(size(gn)==[0 0]);
    option = '0';
  end
  nogz = 1;
elseif nargin==4
  if isstr(gl)
    nogl = 1;
    option = gl;
  else
    nogl = all(size(gl)==[0 0]);
    option = '0';
  end
  nogn = 1;
  nogz = 1;
else
  nogl = 1;
  nogn = 1;
  nogz = 1;
  option = '0';
end

gloop = fclose(fmul(gp,gfb));

if nogz
  gzr = fmul(gloop,fdiv(gff,gfb));
else
  gzr = fmul(fmul(gloop,fdiv(gff,gfb)),fdiv(gz,gp));
end

if nogl
  gzl = fdiv(gzr,gff);
else
  gzl = fmul(gzr,fdiv(gl,gff));
end

% The following calculations have the wrong sign, but since we only
% will plot the magnitude that does not matter.

if nogn
  gun = fdiv(gloop,gp);
else
  gun = fmul(gloop,fdiv(gn,gp));
end

grob = finv(gloop);

subplot(221);
```

```
axis(glob_scale(1,:));
ind = size(gzr)*[0;1];
if glob_hz
  if option=='0'
    loglog(gzr(:,1)/(2*pi),abs(gzr(:,2:ind)));
  else
    loglog(gzr(:,1)/(2*pi),abs(gzr(:,2:ind)),option);
  end
else
  if option=='0'
    loglog(gzr(:,1),abs(gzr(:,2:ind)));
  else
    loglog(gzr(:,1),abs(gzr(:,2:ind)),option);
  end
end

subplot(222);
axis(glob_scale(2,:));
ind = size(gzl)*[0;1];
if glob_hz
  if option=='0'
    loglog(gzl(:,1)/(2*pi),abs(gzl(:,2:ind)));
  else
    loglog(gzl(:,1)/(2*pi),abs(gzl(:,2:ind)),option);
  end
else
  if option=='0'
    loglog(gzl(:,1),abs(gzl(:,2:ind)));
  else
    loglog(gzl(:,1),abs(gzl(:,2:ind)),option);
  end
end

subplot(223);
axis(glob_scale(3,:));
ind = size(gun)*[0;1];
if glob_hz
  if option=='0'
    loglog(gun(:,1)/(2*pi),abs(gun(:,2:ind)));
  else
    loglog(gun(:,1)/(2*pi),abs(gun(:,2:ind)),option);
  end
else
  if option=='0'
    loglog(gun(:,1),abs(gun(:,2:ind)));
  else
    loglog(gun(:,1),abs(gun(:,2:ind)),option);
  end
end

subplot(224);
axis(glob_scale(4,:));
ind = size(grob)*[0;1];
if glob_hz
  if option=='0'
    loglog(grob(:,1)/(2*pi),abs(grob(:,2:ind)));
  else
    loglog(grob(:,1)/(2*pi),abs(grob(:,2:ind)),option);
  end
else
  if option=='0'
    loglog(grob(:,1),abs(grob(:,2:ind)));
```

```
  else
    loglog(grob(:,1),abs(grob(:,2:ind)),option);
  end
end
axis;

-------------------- /regler/matlab/frbox/fadd.m --------------------

function fr = fadd(fr1,fr2)
% FADD  Adds two frequency responses (parallel connection of two systems).
%
%       fr = fadd(fr1,fr2)
%
%       Each frequency response in fr1 is added to the corresponding response
%       in fr2. If fr1 or fr2 contain only one response it is used for each
%       response in the other variable.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:47:37 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[n1,m1] = size(fr1);
[n2,m2] = size(fr2);

if n1~=n2
  error('Not the same number of frequency points.');
elseif any(fr1(:,1)-fr2(:,1)>eps)
  error('Different frequency points.');
elseif m1==2
  fr = [fr1(:,1) fr1(:,2)*ones(1,m2-1)+fr2(:,2:m2)];
elseif m2==2
  fr = [fr1(:,1) fr1(:,2:m1)+fr2(:,2)*ones(1,m1-1)];
elseif m1==m2
  fr = [fr1(:,1) fr1(:,2:m1)+fr2(:,2:m2)];
else
  error('Inconsistent number of columns in fr1 and fr2');
end

-------------------- /regler/matlab/frbox/fclose.m --------------------

function fr = fclose(fro)
% FCLOSE Calculates the frequency response of the closed loop system
%       resulting from unity gain feedback of the loop transfer fro.
%
%       fr = fclose(fro)

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:47:45 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[n,m] = size(fro);
fr = [fro(:,1) fro(:,2:m)./(1+fro(:,2:m))];

-------------------- /regler/matlab/frbox/fdiv.m --------------------

function fr = fdiv(fr1,fr2)
% FDIV  Divides two frequency responses (serial removal of one system
%       from another).
%
%       fr = fdiv(fr1,fr2)
```

```
%
%       Each frequency response in fr1 is divided by the corresponding
%       response in fr2. If fr1 or fr2 contain only one response it is used
%       for each response in the other variable.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:47:56 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[n1,m1] = size(fr1);
[n2,m2] = size(fr2);

if n1~=n2
  error('Not the same number of frequency points.');
elseif any(fr1(:,1)-fr2(:,1)>eps)
  error('Different frequency points.');
elseif m1==2
  fr = [fr1(:,1) (fr1(:,2)*ones(1,m2-1))./fr2(:,2:m2)];
elseif m2==2
  fr = [fr1(:,1) fr1(:,2:m1)./(fr2(:,2)*ones(1,m1-1))];
elseif m1==m2
  fr = [fr1(:,1) fr1(:,2:m1)./fr2(:,2:m2)];
else
  error('Inconsistent number of columns in fr1 and fr2');
end

-------------------- /regler/matlab/frbox/finv.m --------------------

function fri = finv(fr)
% FINV  Invertes a frequency response
%
%       fri = finv(fro)

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:48:06 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[n,m] = size(fr);
fri = [fr(:,1) ones(fr(:,2:m))./fr(:,2:m)];

-------------------- /regler/matlab/frbox/fmarg.m --------------------

function [gm,phm,wgm,wphm] = fmarg(fr)
%FMARG  Gain margin, phase margin, and associated frequencies
%
%       [gm,phm,wgm,wphm] = fmarg(fr)
%
%       Calculates the gain and phase margins for the frequency responses
%       in fr. Each row in the output arguments corresponds to one column
%       in fr. An undefined gain och phase margin is represented with NaN.

% Kjell Gustafsson
% LastEditDate : Mon Jun 25 17:35:47 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

gm = [];
phm = [];
wgm = [];
wphm = [];
```

```
for k=2:size(fr)*[0;1]
  [wphmtmp,phmtmp] = ampcross([fr(:,1) fr(:,k)],1);
  [wgmtmp,gmtmp] = phacross([fr(:,1) fr(:,k)],-180);
  if size(gmtmp)==[0 0], gmtmp = NaN; end
  if size(phmtmp)==[0 0], phmtmp = NaN; end
  if size(wgmtmp)==[0 0], wgmtmp = NaN; end
  if size(wphmtmp)==[0 0], wphmtmp = NaN; end
  gm = [gm; 1/gmtmp];
  phm = [phm; 180+phmtmp];
  wgm = [wgm; wgmtmp];
  wphm = [wphm; wphmtmp];
end
```

-------------------- /regler/matlab/frbox/fmul.m --------------------

```
function fr = fmul(fr1,fr2)
% FMUL  Multiplies two frequency responses (serial connection of two systems).
%
%       fr = fmul(fr1,fr2)
%
%       Each frequency response in fr1 is multiplied with the corresponding
%       response in fr2. If fr1 or fr2 contain only one response it is used
%       for each response in the other variable.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:48:16 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[n1,m1] = size(fr1);
[n2,m2] = size(fr2);

if n1~=n2
  error('Not the same number of frequency points.');
elseif any(fr1(:,1)-fr2(:,1)>eps)
  error('Different frequency points.');
elseif m1==2
  fr = [fr1(:,1) (fr1(:,2)*ones(1,m2-1)).*fr2(:,2:m2)];
elseif m2==2
  fr = [fr1(:,1) fr1(:,2:m1).*(fr2(:,2)*ones(1,m1-1))];
elseif m1==m2
  fr = [fr1(:,1) fr1(:,2:m1).*fr2(:,2:m2)];
else
  error('Inconsistent number of columns in fr1 and fr2');
end
```

-------------------- /regler/matlab/frbox/fopen.m --------------------

```
function fro = fopen(frc)
% FOPEN Calculates the frequency response of the loop transfer
%       corresponding to a closed loop frequency response frc.
%
%       fro = fopen(frc)

% Kjell Gustafsson
% LastEditDate : Mon Mar 26 10:50:11 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[n,m] = size(frc);
```

```
fro = [frc(:,1) frc(:,2:m)./(1-frc(:,2:m))];
```

```
--------------------- /regler/matlab/frbox/fpick.m --------------------

function fr_picked = fpick(fr,omega)
% FPICK Picks out values at certain frequencies of a frequency response.
%
%       FR_PICKED = FPICK(FR,OMEGA)
%
%       Makes linear interpolation in the frequency response FR at
%       the frequencies OMEGA. Frequencies in OMEGA outside the frequency
%       interval of FR are skipped and a warning text will appear.

% Mats Lilja
% LastEditDate : Tue Mar 13 14:29:14 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

w = sort(omega(:));
w_min = min(fr(:,1));
w_max = max(fr(:,1));
skip = find(w<w_min | w>w_max);
if sum(size(skip)) > 0,
  disp('Skipping frequencies outside the frequency interval:');
  skipped_frequencies = w(skip),
  w(skip) = [];
end;
g = table1(fr,w);
fr_picked = [w g];
```

```
-------------------- /regler/matlab/frbox/frbox.m --------------------

% FRBOX -- A collection of routines to generate, plot, and manipulate
%          the frequency response of a system. Written by Kjell Gustafsson
%          and Michael Lundh, but in most cases originating from Mats Lilja.
%
% Generate
%
%    FRC, FRD          - frequency response from polynomial description
%    FRCSS, FRDSS      - frequency response from state space description
%    FRCSYS, FRDSYS    - frequency response from system description
%    FRCPID, FRDPID    - frequency response of PID controller
%    FRTUST            - frequency response of Tustin approximation
%    FRCASYMP          - amplitude asymptots from polynomial description
%    SVCSS             - singular value response from state space description
%    SVCSYS            - singular value response from system description
%
%    A frequency response is represented as a matrix [ w G1(i*w) G2(i*w) ... ]
%    with w in rad/s.
%
% Plot
%
%    BOPL, BOSH, BOGRID  - Bode plot
%    AMPL, AMSH, AMGRID  - amplitude plot
%    PHPL, PHSH, PHGRID  - phase plot
%    NYPL, NYSH, NYGRID  - Nyquist plot
%    NIPL, NISH, NIGRID  - Nichols plot
%    EVPL, EVSH, EVGRID  - plots to help evaluate a closed loop system
%
%    The *PL routines plot a new diagram from 1 up to 4 frequency responses.
%    A *SH routine adds to a previous plot, and *GRID plots grid and markings.
%
```

```
% Manipulate
%
%    FINV                      - invert a frequency response
%    FADD, FSUB, FMUL, FDIV    - various connections of two systems
%    FCLOSE, FOPEN, FSENS      - closed loop, open loop, and sensitivity function
%
% Extracting data
%
%    FPICK                               - pick out points from frequency response
%    LEVCROSS, AMPCROSS, PHACROSS        - compute level crossings
%    FMARG                               - amplitude and phase margin
%    BANDWIDTH                           - 3 dB bandwidth
%
% Conversion
%
%    ID2FR   - System Identification Toolbox data format to FRBOX format
%
% Miscellaneous
%
%    Some global variables are used. They are defined by executing FRBOX.
%    The names of the global variables start by 'glob_'. The frequency
%    plotting scale will be marked in Hz if the variable glob_hz has a
%    value different from 0 or [].
%    Try FRDEMO
%
% Bugs
%
%    Sometimes different (and erroneous) scaling on screen and in meta file.
%    Normally taken care of by using a larger plot window on screen.
%
%    Sometimes MATLAB chooses scales that one can not get through the axis
%    command. It is then impossible to add new curves with the *SH commands.
%
%    When using routines that plots 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.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 09:59:25 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

global glob_scale;
global glob_hz;
```

--------------------- /regler/matlab/frbox/frc.m ---------------------

```
function fr = frc(b,a,tau,lgw1,lgw2,n)
% FRC   Computes the frequency response of a continuous time
%       transfer function.
%
%       fr = frc(b,a,tau,lgw1,lgw2,n)
%       fr = frc(b,a,tau,wvec)
%
%       The value of G(s) = b(s)/a(s)*exp(-tau*s) is calculated either for
%       the frequencies in wvec [rad/s] or for n logarithmically spaced
%       frequency points [rad/s] between 10^lgw1 and 10^lgw2. The argument
%       n is optional with default value 50.
%
%       If b, a, and tau contain several rows the frequency response will be
%       calculated for each triplet b(i,:), a(i,:), and tau(i).
%
```

```
%       The output fr takes the form [ w  G1(iw) G2(iw) ... ] where Gi
%       corresponds to row i of a, b, and tau.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:49:05 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

i = sqrt(-1);

if nargin==6
  w = logspace(lgw1,lgw2,n)';
elseif nargin==5
  w = logspace(lgw1,lgw2)';
else
  w = lgw1(:);
end

fr = [w gval(b,a,tau,i*w)];

-------------------- /regler/matlab/frbox/frcasymp.m --------------------

function fr = frcasymp(b,a,tau,lgw1,lgw2)
% FRCASYMP Computes the Bode diagram asymptotes
%
%       fr = frcasymp(b,a,tau,lgw1,lgw2)
%
%       The frequency responses corresponding to the amplitude asymptotes of
%       G(s) = b(s)/a(s)*exp(-s*tau) between 10^lgw1 and 10^lgw2 are
%       calculated. No attempt is made to calculate a phase asymptote, i.e.
%       the phase of fr will be 0.
%
%       The output fr takes the form [ w  G(iw)].

% Kjell Gustafsson
% LastEditDate : Fri Mar  9 01:36:29 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

w1 = 10^lgw1;
w2 = 10^lgw2;

% Find out how many poles and zeros at the origin

inda = find(a>eps);
na = length(a)-inda(length(inda));
indb = find(b>eps);
nb = length(b)-indb(length(indb));

% Calculate the slope of the low frequency asymptote and
% its magnitude for w = 1

slope0 = nb-na;
mag0 = b(indb(length(indb)))/a(inda(length(inda)));

if na==length(a)-1 & nb==length(b)-1
  % only poles and zeros at the origin
  fr = [w1 mag0*w1^slope0; w2 mag0*w2^slope0];
else
  % Calculate non origin poles and zeros

  p = roots(a(1:length(a)-na));
```

```
  z = roots(b(1:length(b)-nb));

  % Sort poles and zeros in magnitude order, calculate asymptote slopes

  [wvec,ind] = sort(abs([p; z]));
  slope = [-ones(length(p),1); ones(length(z),1)];
  slope = cumsum(slope(ind)) + slope0;

  % Calculate magnitudes for each frequency in wvec

  mag = zeros(wvec);
  mag(1) = mag0*wvec(1)^slope0;
  for k = 2:length(wvec)
    mag(k) = mag(k-1)*(wvec(k)/wvec(k-1))^slope(k-1);
  end

  % Strip out the region between lgw1 and lgw2

  if w1<=wvec(1)
    mag = [mag(1)*(w1/wvec(1))^slope0; mag];
    wvec = [w1; wvec];
  else
    ind = find(wvec>w1);
    mag = [mag(ind(1))*(w1/wvec(ind(1)))^slope(ind(1)-1); mag(ind)];
    wvec = [w1; wvec(ind)];
  end

  ind = find(wvec<w2);
  mag = [mag(ind); mag(ind(length(ind)))*..
           (w2/wvec(ind(length(ind))))^slope(ind(length(ind))-1)];
  wvec = [wvec(ind); w2];

  fr = [wvec mag];
end
```

-------------------- /regler/matlab/frbox/frcheck.m --------------------

```
function notdef = frcheck
% FRCHECK checks if the global variables needed for FRBOX have been defined

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:49:12 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if exist('glob_hz')~=1 | exist('glob_scale')~=1
  disp('Defining necessary global variables. Please reissue command.');
  frbox;
  notdef = 1;
else
  notdef = 0;
end
```

-------------------- /regler/matlab/frbox/frcpid.m --------------------

```
function [fry,frref] = frcpid(k,ti,td,n,b,lgw1,lgw2,np)
% FRCPID Computes the frequency response of a continuous-time PID controller
%
%        [fry,frref] = frcpid(k,ti,td,n,b,lgw1,lgw2,np)
%        [fry,frref] = frcpid(k,ti,td,n,b,wvec)
```

```
%
%       There are two transfer functions associated with a PID controller:
%
%         Gr(s) = k*(b + 1/(ti*s))                    (reference value, frref)
%         Gy(s) = k*(1 + 1/(ti*s) + s*td/(s*td/n+1))) (measured value, fry)
%
%       with the control signal formed as u = Gr r - Gy y.
%       The frequency response for these transfer functions are calculated
%       for the values in wvec [rad/s] or np logarithmically spaced frequency
%       points [rad/s] between 10^lgw1 and 10^lgw2. The argument np is optional
%       with default value 50. The output frref may be omitted.
%
%       k, ti, td, n, and/or b may contain several rows. The frequency response
%       for each set k(i), ti(i), td(i), n(i) and b(i) is calculated. If any of
%       the parameters is given as a single value while the others contain
%       several rows, then this single value is used for all the rows in the
%       other parameters.
%
%       The output fr takes the form [ w  G1(iw) G2(iw) ...] where Gi
%       corresponds to row i of k, ti, td, n and b.
%
%       The frequency response of a PI controller can be calculated
%       by setting td to []. The filter on the D-part is removed by setting
%       n to [] and a PD controller is calculated by setting ti to [].

% Kjell Gustafsson
% LastEditDate : Wed Jun 13 08:38:02 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

i = sqrt(-1);

if nargin==8
  w = logspace(lgw1,lgw2,np)';
elseif nargin==7
  w = logspace(lgw1,lgw2)';
else
  w = lgw1(:);
end

nk = length(k);
nti = length(ti);
ntd = length(td);
nn = length(n);
nb = length(b);

maxn = max([nk nti ntd nn nb]);

if [nk nti ntd nn nb]==maxn | [nk nti ntd nn nb]==1 | [nk nti ntd nn nb]==0
  if nk==1, k = kron(k,ones(maxn,1)); end
  if nti==1, ti = kron(ti,ones(maxn,1)); end
  if ntd==1, td = kron(td,ones(maxn,1)); end
  if nn==1, n = kron(n,ones(maxn,1)); end
  if nb==1, b = kron(b,ones(maxn,1)); end
else
  error('Inconsistent number of rows in k, ti, td, n, and b')
end

k = k(:);
ti = ti(:);
td = td(:);
n = n(:);
```

```
b = b(:);

if [nti ntd]==0
  % P
  fry = [w gval(k,1,0,i*w)];
  frref = [w gval(k.*b,1,0,i*w)];
elseif [nti nn]==0
  % PD without filter
  fry = [w gval([k.*td k],1,0,i*w)];
  frref = [w gval(k.*b,1,0,i*w)];
elseif nti==0
  % PD with filter
  fry = [w gval([k.*(n+ones(maxn,1)) k.*n./td],[ones(maxn,1) n./td],0,i*w)];
  frref = [w gval(k.*b,1,0,i*w)];
elseif ntd==0
  % PI
  fry = [w gval([k k./ti],[1 0],0,i*w)];
  frref = [w gval([k.*b k./ti],[1 0],0,i*w)];
elseif nn==0
  % PID without filter
  fry = [w gval([k.*td k k./ti],[1 0],0,i*w)];
  frref = [w gval([k.*b k./ti],[1 0],0,i*w)];
else
  % PID
  fry = ...
    [w gval([k.*(n+ones(maxn,1)) k.*(n./td+ones(maxn,1)./ti) k.*n./(ti.*td)],...
    [ones(maxn,1) n./td zeros(maxn,1)],0,i*w)];
  frref = [w gval([k.*b k./ti],[1 0],0,i*w)];
end


-------------------- /regler/matlab/frbox/frcss.m --------------------

function fr=frcss(a,b,c,d,tau,lgw1,lgw2,n)
% FRCSS Computes the frequency response of a continuous time MIMO state
%       space system.
%
%       fr = frcss(a,b,c,d,tau,lgw1,lgw2,n)
%       fr = frcss(a,b,c,d,tau,wvec)
%
%       The value of G(s) = ( c (sI-a)^(-1)  b + d )*exp(-tau*s) is calculated
%       either for the frequencies in wvec [rad/s] or for n logarithmically
%       spaced frequency points [rad/s] between 10^lgw1 and 10^lgw2. The
%       argument n is optional with default value 50.
%
%       The output fr takes the form  [  w   G11(iw) G12(iw) ... Gmn(iw) ]
%       with Gij as the transfer function from input j to output i.

% Michael Lundh (original)
% Kjell Gustafsson (revision)
% LastEditDate : Thu Mar  8 16:59:06 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin==8
  w = logspace(lgw1,lgw2,n);
elseif nargin==7
  w = logspace(lgw1,lgw2);
else
  w = lgw1(:);
end
```

```
i = sqrt(-1);

error(abcdchk(a,b,c,d));
[no,ns] = size(c);
ni = size(b)*[0;1];

% Balance A
[t,a] = balance(a);
b = t \ b;
c = c * t;

% Reduce A to Hessenberg form
[p,a] = hess(a);
c = c*p;

gg = [];
for iu=1:ni
  g = ltifr(a,p'*b(:,iu),i*w);
  g = c*g + diag(d(:,iu)) * ones(no,length(w));
  gg = [gg g.'];
end;

%---- now sorted G11 G21 G31 .. G12 G22 .. .. Gmn  -- modify
fr = [];
for i=1:no
  fr = [fr gg(:,i:no:no*ni)];
end

%---- add time delay
fr = diag(exp(-i*w*tau))*fr;

fr = [w(:) fr];

-------------------- /regler/matlab/frbox/frcsys.m --------------------

function fr=frcsys(sys,nsys,tau,lgw1,lgw2,n)
% FRCSYS Computes the frequency response of a continuous time MIMO system.
%
%       fr = frcsys(sys,nsys,tau,lgw1,lgw2,n)
%       fr = frcsys(sys,nsys,tau,wvec)
%
%       The value of G(s) = ( c (sI-a)  b + d )*exp(-tau*s) is calculated
%       either for the frequencies in wvec [rad/s] or for n logarithmically
%       spaced frequency points [rad/s] between 10^lgw1 and 10^lgw2. The
%       argument n is optional with default value 50.
%
%       sys is a system description on the form [a b; c d] with nsys as the
%       number of states.
%
%       The output fr takes the form  [  w   G11(iw) G12(iw) ... Gmn(iw) ]
%       with Gij as the transfer function from input j to output i.

% Kjell Gustafsson
% LastEditDate : Wed Apr  4 09:09:41 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[na,ma]=size(sys);

a = sys(1:nsys,1:nsys);
b = sys(1:nsys,(nsys+1):ma);
c = sys((nsys+1):na,1:nsys);
```

```
d = sys((nsys+1):na,(nsys+1):ma);

if nargin==6
  fr = frcss(a,b,c,d,tau,lgw1,lgw2,n);
elseif nargin==5
  fr = frcss(a,b,c,d,tau,lgw1,lgw2);
else
  fr = frcss(a,b,c,d,tau,lgw1);
end
```

-------------------- /regler/matlab/frbox/frd.m --------------------

```
function fr = frd(b,a,tsamp,lgw1,lgw2,n)
% FRD   Computes the frequency response of a discrete time
%       transfer function.
%
%       fr = frd(b,a,tsamp,lgw1,lgw2,n)
%       fr = frd(b,a,tsamp,lgw1,[],n)
%       fr = frd(b,a,tsamp,lgw1)
%       fr = frd(b,a,tsamp,wvec)
%
%       The value of H(z) = b(z)/a(z) is calculated either for the frequencies
%       in wvec [rad/s] or for n logarithmically spaced frequency points
%       [rad/s] between 10^lgw1 and 10^lgw2. The argument n is optional with
%       default value 50. If lgw2 is supplied as an empty matrix or omitted it
%       is taken as half the sampling frequency (pi/tsamp).
%
%       If b and a contain several rows the frequency response will be
%       calculated for each pair b(i,:) and a(i,:).
%
%       The output fr takes the form
%       [ w  H1(exp(iw*tsamp)) H2(exp(iw*tsamp)) ... ] where Hi corresponds
%       to row i of a and b.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:50:04 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

i = sqrt(-1);

if nargin==6
  if size(lgw2)~=[0 0]
    w = logspace(lgw1,lgw2,n)';
  else
    w = logspace(lgw1,log10(pi/tsamp),n)';
  end
elseif nargin==5
  w = logspace(lgw1,lgw2)';
elseif length(lgw1)==1
  w = logspace(lgw1,log10(pi/tsamp))';
else
  w = lgw1(:);
end

fr = [w gval(b,a,zeros(size(a)*[1;0],1),exp(i*w*tsamp))];
```

-------------------- /regler/matlab/frbox/frdemo.m --------------------

```
% Kjell Gustafsson
% LastEditDate : Thu Jul 19 11:10:10 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
```

```
% Lund Institute of Technology, Lund, SWEDEN

frbox
echo on
clc
%       This example demonstrates some of the functions in FRBOX

%       Through out the example we will use the process G = 2/(s+1)*exp(-s)
%       Let's start by calculating its frequency response.

fp = frc(2,[1 1],1,-1,1);

pause   % Strike RETURN to make a Bode plot of G

bopl(fp)
title('Bode plot for the process G = 2/(s+1)*exp(-s)')

pause   % Strike RETURN to add grid

bogrid, pause

clc
%       Calculate the gain and phase margin of the process

[gm,phm,wgm,wphm] = fmarg(fp)

pause   % Strike RETURN to continue

clc
%       Use Ziegler-Nichols to calculate PID parameters

k = 0.6*gm, ti = 2*pi/wgm/2, td = ti/4,

pause   % Strike RETURN to continue

clc
%       We want to investigate the closed loop system when using the
%       PID controller. Calculate the frequency response for the PID
%       controller, but vary the gain by 25 % around its nominal value.

[fy,fr] = frcpid(k*[0.75; 1; 1.25],ti,td,5,1,-1,1);

%       The PID controller has two different frequency responses; one from
%       measured value and one from reference value. We start by making
%       a Bode plot of the response from measured value. For completeness
%       we also include the response of the process.

pause   % Strike RETURN for Bode plot

bopl(fy,fp),
title('Bode plot of PID controller, k = [0.75 1 1.25]*knom'),

%       We also want to see the response from reference value. To compare
%       we add it in the current diagram.

pause   % Strike RETURN to add response from reference value

bosh(fr), pause

clc
%       By using FMUL we can calculate the loop gain. We plot it in
%       a Nichols plot.
```

```
pause   % Strike RETURN for a Nichols plot of the loop gain

nipl(fmul(fy,fp))
title('Nichols plot of the loop gain'),  pause

%       The plot scales aren't too good, so we make a new plot with different
%       scaling. We also mark the frequency corresponding to process phase lag
%       equal to -180 degrees.

pause   % Strike RETURN for a new Nichols plot

nipl(fmul(fy,fp),wgm,[-200 0 -1 1])
title('Nichols plot of the loop gain'),  pause

pause   % Strike RETURN to add grid and Mp-circles

nigrid, pause

clc
%       Make a Bode plot of the sensitivity function

pause   % Strike RETURN for a Bode plot of the sensitivity function

bopl(fsens(fmul(fp,fy)))
title('Bode plot of sensitivity function'), pause

clc
%       When evaluating control systems there is a number of different
%       transfer functions worth investigating. EVPL is a routine that
%       plots some of them. For futher information consult the help text
%       of EVPL

pause   % Strike RETURN for an evaluation plot

evpl(fp,fy,fr)

pause   % Strike RETURN to end

echo off
subplot

--------------------- /regler/matlab/frbox/frdpid.m --------------------

function [fry,frref] = frdpid(k,ti,td,n,b,tsamp,lgw1,lgw2,np)
% FRDPID Computes the frequency response of a discrete time PID controller
%
%       [fry,frref] = frdpid(k,ti,td,n,b,tsamp,lgw1,lgw2,np)
%       [fry,frref] = frdpid(k,ti,td,n,b,tsamp,wvec)
%
%       There are two transfer functions associated with a PID controller:
%
%         Gr(s) = k*(b + 1/(ti*s))                     (reference value, frref)
%         Gy(s) = k*(1 + 1/(ti*s) + s*td/(s*td/n+1))) (measured value, fry)
%
%       with the control signal formed as u = Gr r - Gy y. A standard
%       discretization is to use forward Euler for the I-part and backward
%       Euler for the derivative, i.e. 1/(ti*s) -> tsamp/(ti*(z-1)) and
%       s*td/(s*td/n+1) -> n*gamma*(z-1)/(z-gamma) with gamma = td/(td+n*tsamp)
%
%       Using these discretizations, the frequency response for the two
%       transfer functions are calculated for the values in wvec [rad/s] or np
```

```
%       logarithmically spaced frequency points [rad/s] between 10^lgw1 and
%       10^lgw2. The argument np is optional with default value 50. If lgw2 is
%       supplied as an empty matrix or omitted it is taken as half the
%       sampling frequency (pi/tsamp). The output frref may be omitted.
%
%       k, ti, td, n, and/or b may contain several rows. The frequency response
%       for each set k(i), ti(i), td(i), n(i) and b(i) is calculated. If any of
%       the parameters is given as a single value while the others contain
%       several rows, then this single value is used for all the rows in the
%       other parameters.
%
%       The output fr takes the form [ w  H1(iw) H2(iw) ...] where Hi
%       corresponds to row i of k, ti, td, n and b.
%
%       The frequency response of a PI controller can be calculated
%       by setting td to []. The filter on the D-part is removed by setting
%       n to [] and a PD controller is calculated by setting ti to [].

% Lars Rundqwist
% LastEditDate : Wed Jun 13 11:51:42 1990
% Copyright (c) 1990 by Lars Rundqwist and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN
% Modification of FRCPID (using FRD) by Kjell Gustafsson

i = sqrt(-1);

if nargin==9
  if size(lgw2)~=[0 0]
    w = logspace(lgw1,lgw2,np)';
  else
    w = logspace(lgw1,log10(pi/tsamp),np)';
  end
elseif nargin==8
  if size(lgw2)~=[0 0]
    w = logspace(lgw1,lgw2)';
  else
    w = logspace(lgw1,log10(pi/tsamp))';
  end
elseif length(lgw1)==1
  w = logspace(lgw1,log10(pi/tsamp))';
else
  w = lgw1(:);
end

nk = length(k);
nti = length(ti);
ntd = length(td);
nn = length(n);
nb = length(b);

maxn = max([nk nti ntd nn nb]);

if [nk nti ntd nn nb]==maxn | [nk nti ntd nn nb]==1 | [nk nti ntd nn nb]==0
  if nk==1, k = kron(k,ones(maxn,1)); end
  if nti==1, ti = kron(ti,ones(maxn,1)); end
  if ntd==1, td = kron(td,ones(maxn,1)); end
  if nn==1, n = kron(n,ones(maxn,1)); end
  if nb==1, b = kron(b,ones(maxn,1)); end
else
  error('Inconsistent number of rows in k, ti, td, n, and b')
end
```

```
k = k(:);
ti = ti(:);
td = td(:);
n = n(:);
b = b(:);

gamma=td./(n*tsamp+td);
onesm=ones(maxn,1);

if [nti ntd]==0
  % P
  fry = [w gval(k,1,0,exp(i*w*tsamp))];
  frref = [w gval(k.*b,1,0,exp(i*w*tsamp))];
elseif [nti nn]==0
  % PD without filter
  fry = [w gval([k.*td/tsamp k-k.*td/tsamp],1,0,exp(i*w*tsamp))];
  frref = [w gval(k.*b,1,0,exp(i*w*tsamp))];
elseif nti==0
  % PD with filter
  fry = [w gval([k.*(n.*gamma+onesm) -k.*(n.*gamma+gamma)],[onesm -gamma], ...
          0,exp(i*w*tsamp))];
  frref = [w gval(k.*b,1,0,exp(i*w*tsamp))];
elseif ntd==0
  % PI
  fry = [w gval([k  k.*(tsamp*onesm./ti-onesm)],[1 -1],0,exp(i*w*tsamp))];
  frref = [w gval([k.*b  k.*(tsamp*onesm./ti-b)],[1 -1],0,exp(i*w*tsamp))];
elseif nn==0
  % PID without filter
  fry = [w gval(...
    [k.*td/tsamp k.*(onesm-2*td/tsamp) k.*(tsamp*onesm./ti-onesm+td/tsamp)],...
    [1 -1],0,exp(i*w*tsamp))];
  frref = [w gval([k.*b  k.*(tsamp*onesm./ti-b)],[1 -1],0,exp(i*w*tsamp))];
else
  % PID
  fry = ...
    [w gval(...
    [k.*(n.*gamma+onesm) k.*(tsamp*onesm./ti-onesm-gamma-2*n.*gamma)..
       k.*(gamma-tsamp*gamma./ti+n.*gamma)],..
    [onesm -(onesm+gamma) gamma],0,exp(i*w*tsamp))];
  frref = [w gval([k.*b  k.*(tsamp*onesm./ti-b)],[1 -1],0,exp(i*w*tsamp))];
end

-------------------- /regler/matlab/frbox/frdss.m --------------------

function fr=frdss(a,b,c,d,tsamp,lgw1,lgw2,n)
% FRDSS Computes the frequency response of a discrete time MIMO state
%       space system.
%
%       fr = frdss(a,b,c,d,tsamp,lgw1,lgw2,n)
%       fr = frdss(a,b,c,d,tsamp,lgw1,[],n)
%       fr = frdss(a,b,c,d,tsamp,lgw1)
%       fr = frdss(a,b,c,d,tsamp,wvec)
%
%       The value of H(q) = c (qI-a)^(-1) b + d is calculated either for the
%       frequencies in wvec [rad/s] or for n logarithmically spaced frequency
%       points [rad/s] between 10^lgw1 and 10^lgw2. The argument n is optional
%       with default value 50. If lgw2 is supplied as an empty matrix or
%       omitted it is taken as half the sampling frequency (pi/tsamp).
%
%       The output fr takes the form
%       [ w  H11(exp(iw*tsamp)) H12(exp(iw*tsamp)) ... Hmn(exp(iw*tsamp)]
%       where Hij corresponds to the transfer function from input j to
```

```
%       output i.

% Michael Lundh (original)
% Kjell Gustafsson (revision)
% LastEditDate : Wed Mar  7 14:50:30 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin==8
  if size(lgw2)~=[0 0]
    w = logspace(lgw1,lgw2,n)';
  else
    w = logspace(lgw1,log10(pi/tsamp),n)';
  end
elseif nargin==7
  w = logspace(lgw1,lgw2)';
elseif length(lgw1)==1
  w = logspace(lgw1,log10(pi/tsamp))';
else
  w = lgw1(:);
end

eiwh = exp(sqrt(-1)*w*tsamp);

error(abcdchk(a,b,c,d));
[no,ns] = size(c);
ni      = size(b)*[0;1];

% Balance A
[t,a] = balance(a);
b = t \ b;
c = c * t;

% Reduce A to Hessenberg form
[p,a] = hess(a);
c     = c*p;

gg = [];
for iu=1:ni
  g = ltifr(a,p'*b(:,iu),eiwh);
  g = c*g + diag(d(:,iu)) * ones(no,length(w));
  gg = [gg g.'];
end;

%---- now sorted G11 G21 G31 .. G12 G22 .. .. Gmn  -- MODIFY
fr = w(:);
for i=1:no
  fr = [fr gg(:,i:no:no*ni)];
end

-------------------- /regler/matlab/frbox/frdsys.m --------------------

function fr=frdsys(sys,nsys,tsamp,lgw1,lgw2,n)
% FRDSYS Computes the frequency response of a discrete time MIMO system.
%
%       fr = frdsys(sys,nsys,tsamp,lgw1,lgw2,n)
%       fr = frdsys(sys,nsys,tsamp,lgw1,[],n)
%       fr = frdsys(sys,nsys,tsamp,lgw1)
%       fr = frdsys(sys,nsys,tsamp,wvec)
%
%       The value of H(q) = c (qI-a)^(-1) b + d is calculated either for the
%       frequencies in wvec [rad/s] or for n logarithmically spaced frequency
```

```
%       points [rad/s] between 10^lgw1 and 10^lgw2. The argument n is optional
%       with default value 50. If lgw2 is supplied as an empty matrix or
%       omitted it is taken as half the sampling frequency (pi/tsamp).
%
%       sys is a system description on the form [a b; c d] with nsys as the
%       number of states.
%
%       The output fr takes the form
%       [ w  H11(exp(iw*tsamp)) H12(exp(iw*tsamp)) ... Hmn(exp(iw*tsamp)]
%       where Hij corresponds to the transfer function from input j to
%       output i.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:50:39 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[na,ma]=size(sys);

a = sys(1:nsys,1:nsys);
b = sys(1:nsys,(nsys+1):ma);
c = sys((nsys+1):na,1:nsys);
d = sys((nsys+1):na,(nsys+1):ma);

if nargin==6
  fr = frdsys(a,b,c,d,tsamp,lgw1,lgw2,n);
elseif nargin==5
  fr = frdsys(a,b,c,d,tsamp,lgw1,lgw2);
else
  fr = frdsys(a,b,c,d,tsamp,lgw1)
end

-------------------- /regler/matlab/frbox/frtust.m --------------------

function fr = frtust(b,a,tsamp,lgw1,lgw2,n)
% FRTUST Computes the frequency response of the Tustin approximation of
%       a continuous time transfer function.
%
%       fr = frtust(b,a,tau,tsamp,lgw,n)
%
%       The frequency response of the Tustin approximation of
%       G(s) = b(s)/a(s) is calculated for the frequency points [rad/s] in
%       wvec of for n logarithmically spaces frequency points between 10^lgw1
%       and 10^lgw2. The argument n is optional with default value 50. If lgw2
%       is supplied as an empty matrix or omitted it is taken as half the
%       sampling frequency (pi/tsamp).
%
%       If b and a contain several rows the frequency response will be
%       calculated for each pair b(i,:) and a(i,:).
%
%       The output fr takes the form
%       [ w  H1(exp(iw*tsamp)) H2(exp(iw*tsamp)) ... ] where Hi corresponds
%       to row i of a and b.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:51:07 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

i = sqrt(-1);

if nargin==6
```

```
  if size(lgw2)~=[0 0]
    w = logspace(lgw1,lgw2,n)';
  else
    w = logspace(lgw1,log10(pi/tsamp),n)';
  end
elseif nargin==5
  w = logspace(lgw1,lgw2)';
elseif length(lgw1)==1
  w = logspace(lgw1,log10(pi/tsamp))';
else
  w = lgw1(:);
end

wtust = 2/tsamp*((exp(i*w*tsamp)-1)./(exp(i*w*tsamp)+1));
fr = [w gval(b,a,0,wtust)];

-------------------- /regler/matlab/frbox/fsens.m --------------------

function fr = fsens(fro)
% FSENS Calculates the sensitivity frequency response corresponding
%       to the loop transfer fro.
%
%       fr = fsens(fro)

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:51:55 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[n,m] = size(fro);
fr = [fro(:,1) ones(fro(:,2:m))./(1+fro(:,2:m))];

-------------------- /regler/matlab/frbox/fsub.m --------------------

function fr = fsub(fr1,fr2)
% FSUB  Subtracts two frequency responses (parallel connection with opposite
%       sign of two systems).
%
%       fr = fsub(fr1,fr2)
%
%       Each frequency response in fr2 is subtracted from the corresponding
%       response in fr1 (fr1-fr2). If fr1 or fr2 contain only one response it
%       is used for each response in the other variable.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:52:02 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[n1,m1] = size(fr1);
[n2,m2] = size(fr2);

if n1~=n2
  error('Not the same number of frequency points.');
elseif any(fr1(:,1)-fr2(:,1)>eps)
  error('Different frequency points.');
elseif m1==2
  fr = [fr1(:,1) fr1(:,2)*ones(1,m2-1)-fr2(:,2:m2)];
elseif m2==2
  fr = [fr1(:,1) fr1(:,2:m1)-fr2(:,2)*ones(1,m1-1)];
elseif m1==m2
  fr = [fr1(:,1) fr1(:,2:m1)-fr2(:,2:m2)];
```

```
else
  error('Inconsistent number of columns in fr1 and fr2');
end

-------------------- /regler/matlab/frbox/gval.m --------------------

function gvec = gval(b,a,tau,wvec)
% GVAL  evaluates a set of transfer functions at a vector of frequency points
%
%       gvec = gval(b,a,tau,wvec)
%
%       The value of G(s) = b(s)/a(s)*exp(-tau*s) is calculated for
%       the complex frequency points in wvec [rad/s]. The frequency reponse
%       can be calculated for several transfer functions by making  b, a,
%       and tau contain several rows. Column k in gvec corresponds to row k
%       in a, b, and tau. If a, b, or tau is given as a single value while
%       the other parameters contain several rows then this single value will
%       be used for all the rows in the other parameters.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:52:14 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

na = size(a)*[1;0];
nb = size(b)*[1;0];
nt = length(tau);
maxn = max([na nb nt]);

if [na nb nt]==maxn | [na nb nt]==1
  if na==1, a = kron(a,ones(maxn,1)); end
  if nb==1, b = kron(b,ones(maxn,1)); end
  if nt==1, tau = kron(tau,ones(maxn,1)); end
else
  error('Inconsistent number of rows in a, b, and tau')
end

gvec = zeros(length(wvec),maxn);   % reserve space in advance for speed

for k=1:maxn,
  gvec(:,k) = ..
    exp(-tau(k)*wvec(:)).*polyval(b(k,:),wvec(:))./polyval(a(k,:),wvec(:));
end

-------------------- /regler/matlab/frbox/id2fr.m --------------------

function fr = id2fr(idfr)
% ID2FR Convert Identification Toolbox frequency files to FRBOX format
%
%       FR = ID2FR(IDFR)
%
%       Convert an Identification Toolbox frequency file IDFR
%       to the format [omega G].

% Mats Lilja
% LastEditDate : Tue Mar 20 10:53:51 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[nrows,ncols] = size(idfr);
idfr1 = idfr(1,:);
idfr2 = idfr(2:nrows,:);
```

```
freqind = find(idfr1 > 100);
ampind = find(idfr1 < 20);
phaind = find(idfr1 > 10 & idfr1 < 50);
nfre = length(freqind);
namp = length(ampind);
npha = length(phaind);
if max(abs([nfre-namp,nfre-npha])) > 0,
  error('Different number of frequency, amplitude or phase columns');
end;
nfre = length(freqind);
if nfre > 1,
  disp('Picking all frequency responses with the same frequency column');
  disp('as the first frequency response.');
end;
omega = idfr2(:,freqind);
omega1 = omega(:,1);
ind = freqind(1);
for j=2:nfre,
  if sum(abs(omega1-omega(:,j))) < eps,
    ind = [ind idfr1(freqind(j))-100];
  end;
end;
npicked = length(ind);
if npicked == 1,
  pluralis = [];
else,
  pluralis = 's';
end;
disp([sprintf('%g frequency response',npicked) pluralis ' picked.']);
fr = omega1;
for k=ind,
  aind = find(idfr1==k);
  ampk = idfr2(:,aind);
  pind = find(idfr1==20+k);
  phak = idfr2(:,pind);
  fr = [fr ampk.*exp(sqrt(-1)*phak*pi/180)];
end;
```

-------------------- /regler/matlab/frbox/levcross.m --------------------

```
function [xcross,zvalues] = levcross(table,level)
% LEVCROSS Compute level crossings in a table
%
%       XCROSS = LEVCROSS(TABLE,LEVEL)
%
%       Given a two column table TABLE, [ X  Y ], all values in X
%       for which Y crosses LEVEL are computed.
%
%       If TABLE consists of three or more columns [ X  Y  Z ]
%
%       [XCROSS,ZVALUES] = LEVCROSS(TABLE,LEVEL)
%
%       gives the corresponding values of Z as well

% Mats Lilja
% LastEditDate : Mon Jun 25 17:25:56 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

x = table(:,1);
y = table(:,2);
[nrows,ncols] = size(table);
```

```
z = table(:,3:ncols);
err = y - level;
indices = find(diff(sign(err))~=0);
if length(indices)<1,
  disp('No crossing of this level');
end;
xcross = [];
zvalues = [];
for ind1=indices',
  ind2 = min([length(x),ind1+1]);
  yxz = [y x z];
  yxz1 = yxz(ind1:ind2,:);
  xz = table1(yxz1,level);
  xcross = [xcross ; xz(1)];
  zvalues = [zvalues ; xz(2:ncols-1)];
end;

-------------------- /regler/matlab/frbox/mpcirc.m --------------------

function z = mpcirc(mp,n);
% MPCIRC Calculates mp-circles.
%
%       z = mpcirc(mp,n)
%
%       Calculates mp-circles corresponding to the values in the row
%       vector mp. n states the number of points on each circle. n is
%       optional, and if not given taken as 40.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:52:31 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin == 1,
  n = 40;
end
i = sqrt(-1);
uc = exp(i*(0.00001:2*(pi-0.000011)/(n-1):(2*pi-0.00001)))';
ucs = kron(uc,mp);
z = ucs./(ones(ucs) - ucs);

-------------------- /regler/matlab/frbox/nigrid.m --------------------

function nigrid(mp)
% NiGRID Plot grid and mark the mp-circles in a Nichols plot.
%
%       nigrid(mp)
%
%       The argument mp decides which mp-circles to draw. If mp is not
%       supplied the circle corresponding to log10(|G|) = [-0.5 -0.2
%       -0.1 -0.05 0 0.05 0.1 0.2 0.5] are drawn.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:52:43 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

niscale = axis;

hold on;
grid;
```

```
phtemp = niscale(1);
if nargin==0,
  mp = [10^0.5 10^0.2 10^0.1 10^0.05 1 10^(-0.05)..
    10^(-0.1) 10^(-0.2) 10^(-0.5)];
end;
mpval = mpcirc(mp,100);
mpabs = abs(mpval);
mparg = arg(mpval);
mparg = mparg - 360*ones(100,1)*(max(mparg)>0);
while phtemp<niscale(2)+359,
  semilogy(mparg+ceil(phtemp/360)*360,mpabs,'-');
  phtemp = phtemp+360;
end
hold off;

-------------------- /regler/matlab/frbox/nipl.m --------------------

function nipl(fr1,fr2,fr3,fr4,wmark,scale)
% NIPL  Plot a Nichols plot.
%
%       nipl(fr1,fr2,fr3,fr4,wmark,scale)
%       nipl(fr1,fr2,fr3,fr4,[],scale)
%       nipl(fr1,fr2,fr3,fr4,wmark)
%       nipl(fr1,fr2,fr3,fr4)
%
%       A Nichols plot is done from the frequency responses in fr1 - fr4.
%       The arguments fr2 - fr4 are optional. fr1 - fr4 is allowed to have
%       different number of data points and columns.
%
%       The frequency points in wmark are marked in the plot. If wmark is
%       omitted or has the value [] no marking is done. If wmark equals
%       '125' then the 1, 2 and 5 frequency points in each decade are marked
%       with +, *, and o, respectively. Due to numerics it may happen that
%       end points are not marked. If the global variable glob_hz has a value
%       differing from 0, then the marked points will be in [Hz] while in
%       [rad/s] otherwise. The mp-circle corresponding to log10(|G|) = 0.1
%       is plotted.
%
%       The scales of the plot can be affected by supplying the argument
%       scale. It takes the form [phmin phmax amin amax]. amin and amax are
%       given in 10-logarithm, i.e. amin = -1, amax = 2 corresponds to an
%       amplitude scale from 0.1 to 100.
%
%       To draw consecutive plots in the same diagram use NISH.

% Kjell Gustafsson
% LastEditDate : Tue Apr 17 08:24:49 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

% Determine if the scale and/or mark argument is present

if nargin==1
  scaleinfo = 0;
  markinfo = 0;
  markarg = 0;
  wmark = [];
else
  if nargin<6
    if nargin==5
```

```
      scale = wmark;
    else
      scale = eval(['fr' int2str(nargin)]);
    end
    wmark = eval(['fr' int2str(nargin-1)]);
  end
  if min(size(wmark))>1
    scaleinfo = 0;
    wmark = scale;
    wsize = size(wmark);
    if min(wsize)==1
      markinfo = 1;
      markarg = 1;
    elseif wsize==[0 0]
      markinfo = 0;
      markarg = 1;
    else
      markinfo = 0;
      markarg = 0;
    end
  else
    if size(scale)==[1 4]
      scaleinfo = 1;
    else
      error('Incorrect scale argument');
    end
    markarg = 1;
    if size(wmark)==[0 0]
      markinfo = 0;
    else
      markinfo = 1;
    end
  end
end

% Construct a command that calculates which decades a frequency response spans

if glob_hz,
  decstr = ['dec = 2*pi*'..
    'exp(log(10)*(floor(log10(wmin/(2*pi))):ceil(log10(wmax/(2*pi))))'');'];
else
  decstr = 'dec = exp(log(10)*(floor(log10(wmin)):ceil(log10(wmax)))'');';
end


% Calulate the markings for the first response. The interpolation has to be
% done on the phase-amplitude data to handle phase wrap correctly.
% Unfortunately this makes the code rather involved.

ind1 = size(fr1)*[0;1];
ind2 = 2*(ind1-1);
f = [fr1(:,1) abs(fr1(:,2:ind1)) arg(fr1(:,2:ind1))];

if markinfo
  if ~isstr(wmark)
    ptemp = table1(f,wmark);
    if size(ptemp)~=[0 0]
      pt1 = ptemp(:,1:ind1-1);
      pt2 = ptemp(:,ind1:ind2);
      p = [pt1(:) pt2(:)];
    else
      p = [];
```

```
    end
  else
    wmin = min(f(:,1));
    wmax = max(f(:,1));
    eval(decstr);
    w1 = dec;
    w2 = 2*dec;
    w5 = 5*dec;
    ptemp = table1(f,w1(find((w1 >= wmin) & (w1 <= wmax))));
    if size(ptemp)~=[0 0]
      pt1 = ptemp(:,1:ind1-1);
      pt2 = ptemp(:,ind1:ind2);
      p1 = [pt1(:) pt2(:)];
    else
      p1 = [];
    end
    ptemp = table1(f,w2(find((w2 >= wmin) & (w2 <= wmax))));
    if size(ptemp)~=[0 0]
      pt1 = ptemp(:,1:ind1-1);
      pt2 = ptemp(:,ind1:ind2);
      p2 = [pt1(:) pt2(:)];
    else
      p2 = [];
    end
    ptemp = table1(f,w5(find((w5 >= wmin) & (w5 <= wmax))));
    if size(ptemp)~=[0 0]
      pt1 = ptemp(:,1:ind1-1);
      pt2 = ptemp(:,ind1:ind2);
      p5 = [pt1(:) pt2(:)];
    else
      p5 = [];
    end
  end
end

f1 = f;
istr1 = sprintf('(:,2:%g)',ind1);
istr2 = sprintf('(:,%g+1:%g+1)',ind1,ind2);
plotnich = ['semilogy(f1' istr2 ',f1' istr1 ];

% Loop to handle the rest of the responses similar to the first one

for k=2:nargin-scaleinfo-markarg,
  kstr = int2str(k);
  f = eval(['fr' kstr]);
  ind1 = size(f)*[0;1];
  ind2 = 2*(ind1-1);
  f = [f(:,1) abs(f(:,2:ind1)) arg(f(:,2:ind1))];

  if markinfo
    if ~isstr(wmark)
      ptemp = table1(f,wmark);
      if size(ptemp)~=[0 0]
        pt1 = ptemp(:,1:ind1-1);
        pt2 = ptemp(:,ind1:ind2);
        p = [p; pt1(:) pt2(:)];
      end
    else
      wmin = min(f(:,1));
      wmax = max(f(:,1));
      eval(decstr);
      w1 = dec;
```

```
        w2 = 2*dec;
        w5 = 5*dec;
        ptemp = table1(f,w1(find((w1 >= wmin) & (w1 <= wmax))));
        if size(ptemp)~=[0 0]
          pt1 = ptemp(:,1:ind1-1);
          pt2 = ptemp(:,ind1:ind2);
          p1 = [p1; pt1(:) pt2(:)];
        end
        ptemp = table1(f,w2(find((w2 >= wmin) & (w2 <= wmax))));
        if size(ptemp)~=[0 0]
          pt1 = ptemp(:,1:ind1-1);
          pt2 = ptemp(:,ind1:ind2);
          p2 = [p2; pt1(:) pt2(:)];
        end
        ptemp = table1(f,w5(find((w5 >= wmin) & (w5 <= wmax))));
        if size(ptemp)~=[0 0]
          pt1 = ptemp(:,1:ind1-1);
          pt2 = ptemp(:,ind1:ind2);
          p5 = [p5; pt1(:) pt2(:)];
        end
      end
    end

    eval(['f' kstr ' = f;']);
    istr1 = sprintf('(:,2:%g)',ind1);
    istr2 = sprintf('(:,%g+1:%g+1)',ind1,ind2);

    plotnich = [plotnich ',f' kstr istr2 ',f' kstr istr1];
  end

  plotnich = [plotnich ');'];

  subplot(111)
  if scaleinfo, axis(scale); end
  eval(plotnich);

  hold on;
  if markinfo
    if ~isstr(wmark)
      semilogy(p(:,2),p(:,1),'x')
    else
      semilogy(p1(:,2),p1(:,1),'+')
      semilogy(p2(:,2),p2(:,1),'*')
      semilogy(p5(:,2),p5(:,1),'o')
    end
  end
  hold off;

  % Plot mp-circle

  niscale = axis;
  mp = mpcirc(10^0.1,100);
  mpabs = abs(mp);
  mparg = arg(mp);
  mparg = mparg - 360*ones(100,1)*(max(mparg)>0);
  phtemp = niscale(1);
  hold on;
  while phtemp<niscale(2)+359,
    semilogy(mparg+ceil(phtemp/360)*360,mpabs,-180+ceil(phtemp/360)*360,1,'+');
    phtemp = phtemp+360;
  end
  hold off;
```

```
ylabel('Magnitude');
xlabel('Phase [deg]');
if isstr(wmark)
  if glob_hz,
    text(0.96,0.89,'Hz','sc');
  else
    text(0.96,0.89,'rad','sc');
  end
  text(0.96,0.86,'1 +','sc');
  text(0.96,0.83,'2 *','sc');
  text(0.96,0.80,'5 o','sc');
end
```

-------------------- /regler/matlab/frbox/nish.m --------------------

```
function nish(fr,wmark,option)
% NISH  Show a Nichols plot in a previously drawn Nichols diagram.
%
%       nish(fr,wmark,option)
%       nish(fr,[],option)
%       nish(fr,wmark)
%       nish(fr)
%
%       A Nichols plot is done from the frequency responses in fr using
%       plotoption option. The argument option is optional.
%
%       The frequency points in wmark are marked in the plot. If wmark is
%       omitted or has the value [] no marking is done. If wmark equals
%       '125' then the 1, 2 and 5 frequency points in each decade are marked
%       with +, *, and o, respectively. Due to numerics it may happen that
%       end points are not marked. If the global variable glob_hz has a value
%       differing from 0, then the marked points will be in [Hz] while in
%       [rad/s] otherwise.

% Kjell Gustafsson
% LastEditDate : Fri Mar 30 08:27:01 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if nargin>1
  markinfo = size(wmark)~=[0 0];
else
  markinfo = 0;
end

% Calulate the markings. The interpolation has to be done on the
% phase-amplitude data to handle phase wrap correctly. Unfortunately
% this makes the code rather involved.

ind1 = size(fr)*[0;1];
ind2 = 2*(ind1-1);
f = [fr(:,1) abs(fr(:,2:ind1)) arg(fr(:,2:ind1))];

if markinfo
  if ~isstr(wmark)
    ptemp = table1(f,wmark);
    if size(ptemp)~=[0 0]
      pt1 = ptemp(:,1:ind1-1);
      pt2 = ptemp(:,ind1:ind2);
```

```
      p = [pt1(:) pt2(:)];
    else
      p = [];
    end
  else
    wmin = min(f(:,1));
    wmax = max(f(:,1));
    if glob_hz,
      dec = 2*pi*..
        exp(log(10)*(floor(log10(wmin/(2*pi))):ceil(log10(wmax/(2*pi))))');
    else
      dec = exp(log(10)*(floor(log10(wmin)):ceil(log10(wmax)))');
    end
    w1 = dec;
    w2 = 2*dec;
    w5 = 5*dec;
    ptemp = table1(f,w1(find((w1 >= wmin) & (w1 <= wmax))));
    if size(ptemp)~=[0 0]
      pt1 = ptemp(:,1:ind1-1);
      pt2 = ptemp(:,ind1:ind2);
      p1 = [pt1(:) pt2(:)];
    else
      p1 = [];
    end
    ptemp = table1(f,w2(find((w2 >= wmin) & (w2 <= wmax))));
    if size(ptemp)~=[0 0]
      pt1 = ptemp(:,1:ind1-1);
      pt2 = ptemp(:,ind1:ind2);
      p2 = [pt1(:) pt2(:)];
    else
      p2 = [];
    end
    ptemp = table1(f,w5(find((w5 >= wmin) & (w5 <= wmax))));
    if size(ptemp)~=[0 0]
      pt1 = ptemp(:,1:ind1-1);
      pt2 = ptemp(:,ind1:ind2);
      p5 = [pt1(:) pt2(:)];
    else
      p5 = [];
    end
  end
end

hold on;
if nargin~=3
  semilogy(f(:,ind1+1:ind2+1),f(:,2:ind1));
else
  semilogy(f(:,ind1+1:ind2+1),f(:,2:ind1),option);
end
if markinfo
  if ~isstr(wmark)
    semilogy(p(:,2),p(:,1),'x')
  else
    semilogy(p1(:,2),p1(:,1),'+')
    semilogy(p2(:,2),p2(:,1),'*')
    semilogy(p5(:,2),p5(:,1),'o')
  end
end
hold off;

-------------------- /regler/matlab/frbox/nygrid.m --------------------
```

```
function nygrid
% NYGRID Plot grid and mark axis in a Nyquist plot.

% Kjell Gustafsson
% LastEditDate : Mon Jun 25 15:50:02 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

axis('square');
hold on;
plot([-1E10; 1E10],[0; 0]);        % real axis
plot([0; 0],[-1E10; 1E10]);        % imaginary axis
grid;
hold off;
axis('normal');
```

-------------------- /regler/matlab/frbox/nypl.m -------------------

```
function nypl(fr1,fr2,fr3,fr4,wmark,scale)
% NYPL  Plot a Nyquist plot.
%
%       nypl(fr1,fr2,fr3,fr4,wmark,scale)
%       nypl(fr1,fr2,fr3,fr4,[],scale)
%       nypl(fr1,fr2,fr3,fr4,wmark)
%       nypl(fr1,fr2,fr3,fr4)
%
%       A Nyquist plot is done from the frequency responses in fr1 - fr4.
%       The arguments fr2 - fr4 are optional. fr1 - fr4 is allowed to have
%       different number of data points and columns.
%
%       The frequency points in wmark are marked in the plot. If wmark is
%       omitted or has the value [] no marking is done. If wmark equals
%       '125' then the 1, 2 and 5 frequency points in each decade are marked
%       with +, *, and o, respectively. Due to numerics it may happen that
%       end points are not marked. If the global variable glob_hz has a value
%       differing from 0, then the marked points will be in [Hz] while in
%       [rad/s] otherwise.
%
%       By default the plot scale is chosen as [-2 1 -2 1], i.e. a square
%       with the -1 point in the center. The scaling can be affected by
%       supplying the argument scale. scale may take the following forms
%       [xmin xmax ymin ymax],
%       [xmin xmax ymin]     interpreted as [xmin xmax ymin ymin+(xmax-xmin)],
%       [xmin xmax]          interpreted as [xmin xmax xmin xmax]
%       [xmin]               interpreted as [-xmin xmin -xmin xmin]
%
%       To draw consecutive plots in the same diagram use NYSH.

% Kjell Gustafsson
% LastEditDate : Thu May  3 16:12:32 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

% Determine if the scale and/or mark argument is present

if nargin==1
  scaleinfo = 0;
  markinfo = 0;
  markarg = 0;
  wmark = [];
```

```
else
  if nargin<6
    if nargin==5
      scale = wmark;
    else
      scale = eval(['fr' int2str(nargin)]);
    end
    wmark = eval(['fr' int2str(nargin-1)]);
  end
  if min(size(wmark))>1
    scaleinfo = 0;
    wmark = scale;
    wsize = size(wmark);
    if min(wsize)==1
      markinfo = 1;
      markarg = 1;
    elseif wsize==[0 0]
      markinfo = 0;
      markarg = 1;
    else
      markinfo = 0;
      markarg = 0;
    end
  else
    scaleinfo = 1;
    [ns,ms] = size(scale);
    if ns==1,
      if ms==1,
        scale = [-scale scale -scale scale];
      elseif ms==2,
        scale = [scale scale];
      elseif ms==3,
        scale = [scale scale(3)+(scale(2)-scale(1))];
      elseif ms==4,
        scale = scale;
      else
        error('Incorrect scale argument');
      end
    else
      error('Incorrect scale argument');
    end
    markarg = 1;
    if size(wmark)==[0 0]
      markinfo = 0;
    else
      markinfo = 1;
    end
  end
end

% Construct a command that calculates which decades a frequency response spans

if glob_hz,
  decstr = ['dec = 2*pi*'..
    'exp(log(10)*(floor(log10(wmin/(2*pi))):ceil(log10(wmax/(2*pi))))'');'];
else
  decstr = 'dec = exp(log(10)*(floor(log10(wmin)):ceil(log10(wmax)))'');';
end

% Calulate the markings for the first response.

if markinfo
```

```
  if ~isstr(wmark)
    ptemp = table1(fr1,wmark);
    if size(ptemp)~=[0 0]
      p = ptemp(:);
    else
      p = [];
    end
  else
    wmin = min(fr1(:,1));
    wmax = max(fr1(:,1));
    eval(decstr);
    w1 = dec;
    w2 = 2*dec;
    w5 = 5*dec;
    ptemp = table1(fr1,w1(find((w1 >= wmin) & (w1 <= wmax))));
    if size(ptemp)~=[0 0]
      p1 = ptemp(:);
    else
      p1 = [];
    end
    ptemp = table1(fr1,w2(find((w2 >= wmin) & (w2 <= wmax))));
    if size(ptemp)~=[0 0]
      p2 = ptemp(:);
    else
      p2 = [];
    end
    ptemp = table1(fr1,w5(find((w5 >= wmin) & (w5 <= wmax))));
    if size(ptemp)~=[0 0]
      p5 = ptemp(:);
    else
      p5 = [];
    end
  end
end

ind = sprintf('(:,2:%g)',size(fr1)*[0;1]);
plotny = ['plot(real(fr1' ind '),imag(fr1' ind ')'];

% Loop to handle the rest of the responses similar to the first one

for k=2:nargin-scaleinfo-markarg,
  kstr = int2str(k);
  f = eval(['fr' kstr]);

  if markinfo
    if ~isstr(wmark)
      ptemp = table1(f,wmark);
      if size(ptemp)~=[0 0]
        p = [p; ptemp(:)];
      end
    else
      wmin = min(f(:,1));
      wmax = max(f(:,1));
      eval(decstr);
      w1 = dec;
      w2 = 2*dec;
      w5 = 5*dec;
      ptemp = table1(f,w1(find((w1 >= wmin) & (w1 <= wmax))));
      if size(ptemp)~=[0 0]
        p1 = [p1; ptemp(:)];
      end
      ptemp = table1(f,w2(find((w2 >= wmin) & (w2 <= wmax))));
```

```
        if size(ptemp)~=[0 0]
          p2 = [p2; ptemp(:)];
        end
        ptemp = table1(f,w5(find((w5 >= wmin) & (w5 <= wmax))));
        if size(ptemp)~=[0 0]
          p5 = [p5; ptemp(:)];
        end
      end
    end

    eval(['ind = sprintf(''(:,2:%g)'',size(fr' kstr ')*[0;1]);'])
    plotny = [plotny ',real(fr' kstr ind '),imag(fr' kstr ind ')'];
end

plotny = [plotny ');'];

subplot(111)
axis('square');
if scaleinfo,
  axis(scale);
else
  axis([-2 1 -2 1]);
end
eval(plotny);

hold on;
if markinfo
  if ~isstr(wmark)
    plot(p,'x')
  else
    plot(p1,'+')
    plot(p2,'*')
    plot(p5,'o')
  end
end

ylabel('Im');
xlabel('Re');
if isstr(wmark)
  if glob_hz,
    text(0.845,0.89,'Hz','sc');
  else
    text(0.845,0.89,'rad/s','sc');
  end
  text(0.845,0.86,'1 +','sc');
  text(0.845,0.83,'2 *','sc');
  text(0.845,0.80,'5 o','sc');
end

hold off;
axis('normal');

-------------------- /regler/matlab/frbox/nysh.m --------------------

function nysh(fr,wmark,option)
% NYSH  Show a Nyquist plot in a previously drawn Nyquist diagram.
%
%       nysh(fr,wmark,option)
%       nysh(fr,[],option)
%       nysh(fr,wmark)
%       nysh(fr)
%
```

```
%       A Nyquist plot is done from the frequency responses in fr using
%       the plotoption option. The argument option is optional.
%
%       The frequency points in wmark are marked in the plot. If wmark is
%       omitted or has the value [] no marking is done. If wmark equals
%       '125' then the 1, 2 and 5 frequency points in each decade are marked
%       with +, *, and o, respectively. Due to numerics it may happen that
%       end points are not marked. If the global variable glob_hz has a value
%       differing from 0, then the marked points will be in [Hz] while in
%       [rad/s] otherwise.

% Kjell Gustafsson
% LastEditDate : Fri Mar 30 08:26:48 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if nargin>1
  markinfo = size(wmark)~=[0 0];
else
  markinfo = 0;
end

% Calulate the markings.

if markinfo
  if ~isstr(wmark)
    ptemp = table1(fr,wmark);
    if size(ptemp)~=[0 0]
      p = ptemp(:);
    else
      p = [];
    end
  else
    wmin = min(fr(:,1));
    wmax = max(fr(:,1));
    if glob_hz,
      dec = 2*pi*..
        exp(log(10)*(floor(log10(wmin/(2*pi))):ceil(log10(wmax/(2*pi))))');
    else
      dec = exp(log(10)*(floor(log10(wmin)):ceil(log10(wmax)))');
    end
    w1 = dec;
    w2 = 2*dec;
    w5 = 5*dec;
    ptemp = table1(fr,w1(find((w1 >= wmin) & (w1 <= wmax))));
    if size(ptemp)~=[0 0]
      p1 = ptemp(:);
    else
      p1 = [];
    end
    ptemp = table1(fr,w2(find((w2 >= wmin) & (w2 <= wmax))));
    if size(ptemp)~=[0 0]
      p2 = ptemp(:);
    else
      p2 = [];
    end
    ptemp = table1(fr,w5(find((w5 >= wmin) & (w5 <= wmax))));
    if size(ptemp)~=[0 0]
      p5 = ptemp(:);
    else
```

```
      p5 = [];
    end
  end
end

ind = size(fr)*[0;1];

axis('square');
hold on;
if nargin~=3
  plot(real(fr(:,2:ind)),imag(fr(:,2:ind)));
else
  plot(real(fr(:,2:ind)),imag(fr(:,2:ind)),option);
end
if markinfo
  if ~isstr(wmark)
    plot(p,'x')
  else
    plot(p1,'+')
    plot(p2,'*')
    plot(p5,'o')
  end
end
hold off;
axis('normal');

-------------------- /regler/matlab/frbox/phacross.m --------------------

function [omega,amp] = phacross(fr,level)
% PHACROSS Computation of phase level crossing frequencies
%
%       OMEGA = PHACROSS(FR,LEVEL)
%
%       Given a frequency response FR, all frequencies for which
%       the phase crosses LEVEL are computed.
%
%       [OMEGA,AMP] = PHACROSS(FR,LEVEL)
%
%       gives the amplitude at the crossing frequencies.

% Mats Lilja
% LastEditDate : Wed Mar 14 16:55:58 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

w = fr(:,1);
g = fr(:,2);
[omega,amp] = levcross([w arg(g) abs(g)],level);

-------------------- /regler/matlab/frbox/phgrid.m --------------------

function phgrid
% PHGRID Plot grid and mark -180 degree in a phase plot.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:53:43 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

hold on;
semilogx([1E-10; 1E10],[-180; -180]);    % -180 degree phase line
grid;
```

```
hold off;

-------------------- /regler/matlab/frbox/phpl.m --------------------

function phpl(fr1,fr2,fr3,fr4,scale)
% PHPL  Plot the phase of a frequency response.
%
%       phpl(fr1,fr2,fr3,fr4,scale)
%
%       An phase plot is done from the frequency responses in fr1 - fr4.
%       The arguments fr2 - fr4 are optional. fr1 - fr4 is allowed to have
%       different number of data points and columns.
%
%       The optional argument scale is used to affect the scaling of the
%       plot. It takes the form [wmin wmax phmin phmax]. wmin and wmax
%       are given in 10-logarithm, i.e. wmin = -1, wmax = 2 corresponds to
%       a frequency scale from 0.1 to 100. scale can be used even if fr2 -
%       fr4 are omitted.
%
%       The scale of the phase plot is stored in the variable glob_scale,
%       which must be declared as global. The scale is used to be able to
%       draw (using PHSH) consecutive plots in the same diagram.
%
%       The frequency axis will be marked in Hz if the global variable
%       glob_hz has a value differing from 0.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:53:52 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if nargin~=5
  scale = eval(['fr' int2str(nargin)]);
end
scaleinfo = all(size(scale) == [1 4]);

ind = sprintf('(:,2:%g)',size(fr1)*[0;1]);
if glob_hz,
  plotpha = ['semilogx(fr1(:,1)/(2*pi),arg(fr1' ind ')'];
else
  plotpha = ['semilogx(fr1(:,1),arg(fr1' ind ')'];
end

for k=2:nargin-scaleinfo,
  kstr = int2str(k);
  eval(['ind = sprintf(''(:,2:%g)'',size(fr' kstr ')*[0;1]);'])
  if glob_hz,
    plotpha = [plotpha ',fr' kstr '(:,1)/(2*pi),arg(fr' kstr ind ')'];
  else
    plotpha = [plotpha ',fr' kstr '(:,1),arg(fr' kstr ind ')'];
  end
end
plotpha = [plotpha ');'];

subplot(111);
if scaleinfo, axis(scale); end
eval(plotpha);
glob_scale = axis;
if ~scaleinfo, axis; end
```

```
ylabel('Phase [deg]');
if glob_hz,
  xlabel('Frequency [Hz]');
else
  xlabel('Frequency [rad/s]');
end
```

-------------------- /regler/matlab/frbox/phsh.m --------------------

```
function phsh(fr,option)
% PHSH  Show the phase of a frequency response in a previous plot.
%
%       phsh(fr,option)
%
%       A phase plot is done from the frequency responses in fr using
%       the plotoption option. The argument option is optional.

% Kjell Gustafsson
% LastEditDate : Wed Mar  7 14:54:00 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if frcheck, return, end

if ~any(size(glob_scale)==[1 4])
  error('No previous phase plot');
end;

hold on;
ind = size(fr)*[0;1];
if nargin == 1,
  if glob_hz,
    semilogx(fr(:,1)/(2*pi),arg(fr(:,2:ind)));
  else
    semilogx(fr(:,1),arg(fr(:,2:ind)));
  end
elseif nargin == 2,
  if glob_hz,
    semilogx(fr(:,1)/(2*pi),arg(fr(:,2:ind)),option);
  else
    semilogx(fr(:,1),arg(fr(:,2:ind)),option);
  end;
end;
hold off;
```

-------------------- /regler/matlab/frbox/svcss.m --------------------

```
function fr = svcss(a,b,c,d,lgw1,lgw2,n)
% SVCSS Computes the singular value frequency response of a
%       continuous time MIMO state space system.
%
%       fr = svcss(a,b,c,d,lgw1,lgw2,n)
%       fr = svcss(a,b,c,d,wvec)
%
%       The maximum and miminum singular values for
%       G(s) = ( c (sI-a)^(-1)  b + d ) is calculated either for the
%       frequencies in wvec [rad/s] or for n logarithmically spaced
%       frequency points [rad/s] between 10^lgw1 and 10^lgw2. The
%       argument n is optional with default value 50.
%
%       The output fr takes the form  [ w sigma_max(w) sigma_min(w) ].
```

```
% Michael Lundh (original)
% LastEditDate : Mon Jun 25 16:19:00 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin==7
  w = logspace(lgw1,lgw2,n);
elseif nargin==6
  w = logspace(lgw1,lgw2);
else
  w = lgw1(:);
end

i = sqrt(-1);

error(abcdchk(a,b,c,d));
[no,ns] = size(c);
ni = size(b)*[0;1];

% Balance A
[t,a] = balance(a);
b = t \ b;
c = c * t;

% Reduce A to Hessenberg form
[p,a] = hess(a);
c = c*p;

gg = [];
for iu=1:ni
  g = ltifr(a,p'*b(:,iu),i*w);
  g = c*g + diag(d(:,iu)) * ones(no,length(w));
  gg = [gg g.'];
end;

%---- now sorted G11 G21 G31 .. G12 G22 .. .. Gmn  -- calculate sv
fr = [];
for ii=1:length(w)
  giw=gg(ii,1:no)';
  for j=2:ni
    giw=[giw gg(ii,(1+no*(j-1)):(no*j))'];
  end
  sv=svd(giw);
  fr=[fr ; w(ii) max(sv) min(sv)];
end

-------------------- /regler/matlab/frbox/svcsys.m --------------------

function fr = svcsys(sys,nsys,lgw1,lgw2,n)
% SVCSYS Computes the singular value frequency response of a
%       continuous time MIMO state space system.
%
%       fr = svcsys(sys,nsys,lgw1,lgw2,n)
%       fr = svcsys(sys,nsys,wvec)
%
%       The maximum and miminum singular values for
%       G(s) = ( c (sI-a)^(-1)  b + d ) is calculated either for the
%       frequencies in wvec [rad/s] or for n logarithmically spaced
%       frequency points [rad/s] between 10^lgw1 and 10^lgw2. The
%       argument n is optional with default value 50.
%
```

```
%       sys is a system description on the form [a b; c d] with nsys as the
%       number of states.
%
%       The output fr takes the form  [ w sigma_max(w) sigma_min(w) ].

% Michael Lundh
% LastEditDate : Mon Jun 25 16:19:31 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin==5
  w = logspace(lgw1,lgw2,n);
elseif nargin==4
  w = logspace(lgw1,lgw2);
else
  w = lgw1(:);
end

[na,ma]=size(sys);

a = sys(1:nsys,1:nsys);
b = sys(1:nsys,(nsys+1):ma);
c = sys((nsys+1):na,1:nsys);
d = sys((nsys+1):na,(nsys+1):ma);

fr = fsvcss(a,b,c,d,w);
```

2. PPBOX

```
-------------------- /regler/matlab/ppbox/addpoly.m --------------------

function p=addpoly(p1,p2)
% ADDPOLY
%       Adds two polynomials of arbitrary degrees
%       P=ADDPOLY(P1,P2)

% Michael Lundh     LastEditDate : Wed Mar  7 16:25:03 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

dn = length(p1)-length(p2);
p  = [zeros(1,-dn) p1] + [zeros(1,dn) p2];

-------------------- /regler/matlab/ppbox/dab.m --------------------

function [x,y] = dab(a,b,c)
% DAB   Solves the Diophantine-Aryabhatta-Bezout identity
%
%       [X,Y] = DAB(A,B,C)
%
%       AX + BY = C, where A, B, C, X and Y are polynomials
%       and deg Y = deg A - 1.

% Mats Lilja      LastEditDate : Wed Mar  7 16:26:14 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

na = length(a);
nb = length(b);
nc = length(c);
ny = na - 1;
if ny<1,
  x = c/a;
  y = 0;
  return;
end;
nx = nc - ny;
c = [zeros(1,nb-nx-1) c];
nc = length(c);
nx = nc - ny;
if nx<1,
  x = 0;
  y = c/b;
  return;
end;
b = [zeros(1,nx-nb+1) b];
za = zeros(1,nx-1);
zb = zeros(1,ny-1);
ma = toeplitz([a za],[a(1) za]);
mb = toeplitz([b zb],[b(1) zb]);
m = [ma mb];
if rank(m)<min(size(m)),
  disp('Singular problem due to common factors in A and B');
end;
xy = c/m';
x = xy(1:nx);
y = xy(nx+1:nc);
```

```
-------------------- /regler/matlab/ppbox/gadd.m --------------------

function [b,a] = gadd(b1,a1,b2,a2)
% GADD  Calculates the sum of two transfer functions
%
%       [B,A] = GADD(B1,A1,B2,A2)
%
%       Computes polynomials B(s) and A(s) such that
%
%         B(s)   B1(s)   B2(s)
%         ---- = ----- + -----
%         A(s)   A1(s)   A2(s)
%

% Mats Lilja
% LastEditDate : Wed Mar 14 10:22:56 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

a = conv(a1,a2);
b = addpoly(conv(b1,a2),conv(a1,b2));

-------------------- /regler/matlab/ppbox/mksysp.m --------------------

function [bb,aa]=mksysp(b,a,db,da,b0,a0)
% MKSYSP
%       Generate polynomials for structured uncertainty.
%
%       [BB,AA]=MKSYSP(B,A,DB,DA)
%       [BB,AA]=MKSYSP(B,A,DB,DA,B0,A0)
%
%       Generate all extreme parameter combinations of the system
%
%              BP   B0
%         G = ---- ----
%              AP   A0
%
%       where B0 and A0 are known and the coefficients of BP/AP
%       satisfy
%
%         B(i)-DB(i) <= BP(i) <= B(i)+DB(i)
%         A(i)-DA(i) <= AP(i) <= A(i)+AB(i)
%
%       The output matrices BB and AA have 2^I rows, where I is the
%       numberof nonzero elements in [DB DA].  The polynomials BB(j,:)
%       and AA(j,:) defines the system corresponding to the j:th corner
%       in the polytope of polynomial coefficients.

% Michael Lundh     LastEditDate : Wed Mar  7 16:26:53 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if max(size(db))==0, db=0*b; end;
if max(size(da))==0, da=0*a; end;

ba  = [b a];
dba = [db da];

nb  = length(b);
na  = length(a);
nba = na+nb;
```

```
Index = [];
for i=1:nba,
  if abs(dba(i))>1.0e-10, Index = [Index i]; end;
end;

nIndex = length(Index);
msys   = 2^nIndex;
bbaa   = kron(ba,ones(msys,1));

%---- gray table
g=[0;1];
[m,n]=size(g);
while m<msys
  g=[zeros(m,1) g ; ones(m,1) rot90(eye(m,m))*g];
  [m,n]=size(g);
end

for m=1:msys
  for i=1:nIndex,
    bbaa(m,Index(i)) = bbaa(m,Index(i)) + (2*g(m,i)-1)*dba(Index(i));
  end
end

bb = bbaa(:,1:nb);
aa = bbaa(:,nb+1:nba);

if nargin==6
  for m=1:msys
    bbx(m,:) = conv(bb(m,:),b0);
    aax(m,:) = conv(aa(m,:),a0);
  end
  bb = bbx;
  aa = aax;
end
```

-------------------- /regler/matlab/ppbox/p2mat.m --------------------

```
function []=p2mat(fil, id1,p1, id2,p2, id3,p3, id4,p4, id5,p5, id6,p6)
% P2MAT Write a polynomials on .M file for other use.
%
%       P2MAT(FILE, ID1,P1, ID2,P2, ID3,P3, ID4,P4, ID5,P5, ID6,P6)
%
%       Polynomials are written on file FILE.M. One to six polynomials
%       may be written.

% Michael Lundh     LastEditDate : Wed Mar  7 16:27:40 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if rem(nargin-1,2)~=0, error('Wrong number of inputs'); end;

%---- open file
film=[fil '.m'];
i=0;
while exist(film)==2
  i=i+1;
  film=[fil int2str(i) '.m'];
end
disp(' ')
disp(['Out-file is ' film])
disp(' ')
```

```
time_stamp=fix(clock);
time_stamp=[sprintf('%g-%g-%g ',time_stamp(1),time_stamp(2),time_stamp(3)) ..
           sprintf('%g:%g:%g',time_stamp(4),time_stamp(5),time_stamp(6))];

fprintf(film,['%% ' film '\n'])
fprintf(film,['%% Created in Matlab at ' time_stamp '\n'])
fprintf(film,'%% \n')

%---- write polynomials
npoly = (nargin-1)/2;

for ip=1:npoly
  eval(sprintf('id=id%g;p=p%g;',ip,ip))
  str = sprintf('%.4g',p(1));
  for il=2:length(p)
    str = [str ' ' sprintf('%.4g',p(il))];
  end
  fprintf(film,[id '=[' str '] \n'])
end;

--------------------- /regler/matlab/ppbox/p2sim.m ---------------------

function []=p2sim(fil,syst, id1,p1,i1, id2,p2,i2, id3,p3,i3, id4,p4,i4)
% P2SIM Write a polynomials for use in simnon.
%
%       P2SIM(FILE,SYST, ID1,P1,I1, ID2,P2,I2, ID3,P3,I3, ID4,P4,I4)
%
%       Polynomial coefficients for simnon system SYST are
%       written on parameter-file FILE. One to four polynomials
%       may be written. Polynomial Pi is recognized in simnon
%       as IDi with index starting from Ii.

% Michael Lundh     LastEditDate : Wed Mar  7 16:28:13 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if rem(nargin-2,3)~=0, error('Wrong number of inputs'); end;

%---- open file
filt=[fil '.t'];
i=0;
while exist(filt)==2
  i=i+1;
  filt=[fil int2str(i) '.t'];
end
disp(' ')
disp(['Out-file is ' filt])
disp(' ')

time_stamp=fix(clock);
time_stamp=[sprintf('%g-%g-%g ',time_stamp(1),time_stamp(2),time_stamp(3)) ..
           sprintf('%g:%g:%g',time_stamp(4),time_stamp(5),time_stamp(6))];

fprintf(filt,['" ' filt '\n'])
fprintf(filt,['" Created in Matlab at ' time_stamp '\n'])
fprintf(filt,'" \n')
fprintf(filt,['[' syst '] \n'])

%---- write polynomials
npoly = (nargin-2)/3;

for ip=1:npoly
```

```
  eval(sprintf('i=i%g;id=id%g;p=p%g;',ip,ip,ip))
  for il=1:length(p)
    fprintf(filt,[id '%g :  %g \n'],i,p(il))
    i=i+1;
  end
end;
```

--------------------- /regler/matlab/ppbox/pade.m --------------------

```
function [b,a] = pade(l,ord)
% PADE  Computes Pade' approximation B(s)/A(s) of order ORD to exp(-sL)
%
%       [B,A] = PADE(L,ORD)
%
%       ORD is 1 or 2.

% Michael Lundh     LastEditDate : Wed Mar  7 16:28:52 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if abs(l)<eps, b=1;
               a=1;
               return;
end;

if ord==1,     b=[-l 2];
               a=[ l 2];
elseif ord==2, b=[l^2 -6*l 12];
               a=[l^2  6*l 12];
else
  error('ORD must be 1 or 2')
end
```

-------------------- /regler/matlab/ppbox/polybess.m --------------------

```
function p = polybess(n,r)
% POLYBESS
%       Computes a Bessel polynomial of specified order.
%
%       P = POLYBESS(N,R)
%
%       A Bessel polynomial of order N is computed. The zeros of
%       the polynomial are scaled with a factor of R, which means
%       that each zero has a magnitude of approximately R.

% Mats Lilja     LastEditDate : Wed Mar  7 16:29:56 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if n == 0,
  p = 1;
elseif n == 1,
  p = [1 1];
elseif n > 1,
  p0 = 1;
  p1 = [1 1];
  for j=2:n,
    p = [0 (2*j-1)*p1] + [p0 0 0];
    p0 = p1;
    p1 = p;
  end;
else,
```

```
  error('Argument must be a non-negative integer');
end
if nargin == 2,
  np = length(p) - 1;
  p = p.*(r^np/p(np+1)).^((0:np)/np);
end;

-------------------- /regler/matlab/ppbox/polybutt.m --------------------

function butterpoly = polybutt(order,radius,angle)
% POLYBUTT
%       Make continuous time polynomial.
%
%       P = POLYBUTT(ORDER,RADIUS,ANGLE)
%
%       Calculates a polynomial of order 'ORDER' with roots evenly spread
%       on a circle segment with radius 'RADIUS' in left half plane.
%       The third argument is half the opening angle of the segment.
%       The default value for 'ANGLE' is 90(1-1/'ORDER') degrees
%       (ordinary Butterworth).

% Mats Lilja     LastEditDate : Wed Mar  7 16:30:33 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin == 2 & order > 0,
  angle = 90*(1-1/order);
end;
i = sqrt(-1);
if order > 1,
  dangle = 2*angle/(order-1);
  d = dangle/10;
  butterpoly = real(poly(radius*exp(i*(180-angle:dangle:180+angle+d)*pi/180)));
elseif order == 1,
  butterpoly = [1 radius];
else
  butterpoly = 1;
end;

-------------------- /regler/matlab/ppbox/polyc.m --------------------

function ac=polyc(w,z,p);
% POLYC Make Continuous time polynomial.
%
%       P=POLYC(W,Z,R)
%
%       Polynomial formed as product of first and second order
%       systems with real zeros R(i) and complex zeros with
%       natural frequencies W(i) and damping Z(i).
%       Argument R is optional.

% Michael Lundh     LastEditDate : Wed Mar  7 16:31:15 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if length(w)~=length(z), error('length(w)~=length(z)'); end;

ac=1;
for i=1:length(w)
  ac=conv(ac,[1 2*w(i)*z(i) w(i)*w(i)]);
end
```

```
if nargin==3,
  ac=conv(ac,poly(p));
end
```

-------------------- /regler/matlab/ppbox/polyc2d.m --------------------

```
function ad=polyc2d(ac,h);
%POLYC2D
%       Make discrete time polynomial.
%
%       AD=POLYC2D(AC,H)
%
%       Discrete characteristic polynomial AD that is given when a
%       continuous system with characteristic polynomial AC is sampled
%       with sampling interval H.

% Michael Lundh     LastEditDate : Wed Mar  7 16:31:53 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

rc  = roots(ac);

ind = find(abs(imag(rc))>100*eps);
w   = [0; abs(rc(ind))];

if max(w*h)>pi, disp('Varning  max(w*h)>pi'); end;

ad  = real(poly(exp(roots(ac)*h)));
```

-------------------- /regler/matlab/ppbox/ppbox.m --------------------

```
% PPBOX -- A collection of routines for pole placement design
%          and simulation of continuous time closed loop systems.
%          Written by Michael Lundh and Mats Lilja.
%
% Polynomial Synthesis
%
%   RSTC       - Continuous time
%   RSTD       - Discrete time
%
% Simulation and plotting
%
%   YUSIMC     - Simulation of closed loop of continuous systems
%   YUSIMD     - Simulation of closed loop of discrete systems
%   YUPL       - Plot one to four time responses
%   YUSH       - Add one time response to previous plot
%   YUSIGNALS  - Generate input signals for YUSIMC and YUSIMD
%   YUSTAIRS   - Modification of simulation outputs for stair plots
%
%   Simulation of a continuous system controlled by a discrete controller
%   should be performed in SIMNON.
%
% Polynomials
%
%   ADDPOLY    - Add two polynomials
%   POLYC      - Create continuous time polynomials
%   POLYBUTT   - Create continuous time Butterworth polynomial
%   POLYBESS   - Create continuous time Bessel polynomial
%   PADE       - Pade approximation of time delay
%   MKSYSP     - Generate systems with structured uncertainty
%
% Continuous to discrete conversion
```

```
%
%   SAMPLE     - Sampling of continuous system
%   POLYC2D    - Mapping of continuous poles to discrete poles
%
% Transfer Function Manipulation
%
%   GADD       - Add two rational transfer functions
%   STABPARTC  - Separate a rational transfer function into stable and
%                unstable partial fractions (continuous time version)
%   STABPARTD  - Separate a rational transfer function into stable and
%                unstable partial fractions (discrete time version)
%
% File Output
%
%   RST2SIM    - Write simnon parameter file with RST-regulator
%   P2SIM      - Write simnon parameter file with polynomials
%   P2MAT      - Write matlab .m file with polynomials
%
% Miscellaneous
%
%    A global variable is used. It is defined by executing PPBOX.
%    The global variable is glob_scale.
%
%    Try PPDEMO for a demonstration.
%
% Bugs
%
%    The use of noise in simulation of continuous systems is not correct
%    since the noise is only present at the instants when outputs are
%    calculated.
%
%    Sometimes different (and erroneous) scaling on screen and in meta file.
%    Normally taken care of by using a larger plot window on screen.
%
%    More bugs, reported but not yet fixed are described by "type ppbox.bugs"

% Michael Lundh     LastEditDate : Fri Jul 20 10:20:31 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

global glob_scale;

-------------------- /regler/matlab/ppbox/ppcheck.m --------------------

function notdef = ppcheck
% PPCHECK checks if the global variables needed for PPBOX have been defined

% Michael Lundh     LastEditDate : Tue Mar 13 22:39:33 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if exist('glob_scale')~=1
  disp('Defining necessary global variables. Please reissue command.');
  ppbox;
  notdef = 1;
else
  notdef = 0;
end


-------------------- /regler/matlab/ppbox/ppdemo.m --------------------
```

```
% PPBOXDEMO
%       A demonstration of how PPBOX and FRBOX may be used for pole
%       placement design.

% Michael Lundh     LastEditDate : Wed Mar 21 22:42:36 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

echo off
ppbox
frbox
echo on

%---- The open system: 1/(s+1)^3
b = 1;
a = [1 3 3 1];

%---- Pole placement design
am = polybutt(3,1.2,45)
ao = polybutt(3,2.0,45)

[r,s,t]=rstc(1,b,a,1,am,ao,[1 0])

%---- Hit any key to continue
pause

%---- Simulation
yu0 = yusimc(b,a,r,s,t,30);
yupl(yu0)
title('Nominal closed loop system')

%---- Hit any key to continue
pause

%---- structured uncertainty in gain and one pole
[bb,aa] = mksysp(1,[1 1],0.4,[0 0.4],1,[1 2 1])
yuz     = yusimc(bb,aa,r,s,t,30);
yupl(yuz)

%---- Hit any key to continue
pause

yush(yu0,'+')
title('Nominal and disturbed systems')

%---- Hit any key to continue
pause

%---- frequency resonses
gp  = frc(b,a,0,-2,2);
gpz = frc(bb,aa,0,-2,2);

gff = frc(t,r,0,-2,2);
gfb = frc(s,r,0,-2,2);

lz  = fmul(gfb,gpz);

bopl(gpz)
title('Disturbed open systems')

%---- Hit any key to continue
```

```
pause

bopl(lz)
title('Disturbed compensated open systems')
%---- Hit any key to continue
pause

nypl(lz)
nygrid
title('Disturbed compensated open systems')
%---- Hit any key to continue
pause

evpl(gp,gfb,gff)
%---- Hit any key to continue
pause

evpl(gpz,gfb,gff)

echo off

-------------------- /regler/matlab/ppbox/rst2sim.m --------------------

function [] = rst2sim(fil,syst,r,s,t,p,paclibflag)
% RST2SIM
%       Write a RST controller for use in simnon.
%
%       RST2SIM(FILE,SYST,R,S,T)
%       RST2SIM(FILE,SYST,R,S,T,P)
%       RST2SIM(FILE,SYST,R,S,T,P,PACLIBFLAG)
%
%       Controller polynomial coefficients for simnon system SYST are
%       written on parameter-file FILE.
%
%       P is an optional argument defining anti-windup characteristic
%       polynomial. Default anti-windup polynomial is given if P=[].
%
%       If PACLIBFLAG is present the parameter-file suits the RST-
%       controllers CRSTREG and DRSTREG in Simnon. For more information
%       see the documentation of CRSTREG and DRSTREG in PACLIB.
%
%       Indices start from 0. First coefficient of R (and P if present
%       and non-empty) must be 1.

% Michael Lundh     LastEditDate : Wed Mar  7 16:34:01 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

rn=max(size(r));
sn=max(size(s));
tn=max(size(t));

if max([sn,tn])>rn,     error('Not Casual regulator'), end;
if abs(r(1)-1)>1.0E-10, error('R(1) not 1'),          , end;

s=[zeros(1,rn-sn) s];
t=[zeros(1,rn-tn) t];

%---- assign not entered inputs
if nargin>5
  if length(p)==0
    p  = [1 zeros(1,rn-1)]; % default p -- not good for cont regul
```

```
    aw = 0;
  else
    aw = 1;
  end
  if rn~=max(size(p)),    error('deg P ~= deg R'),end;
  if abs(p(1)-1)>1.0E-10, error('P(1) not 1'),    end;
end
if nargin==7
  if rn>6, error('PACLIB routine allows max degree=5'), end
  syst='reg'; disp(' ');disp('Forcing: syst=reg');
  r  = [r zeros(1,6-rn)];
  s  = [s zeros(1,6-rn)];
  t  = [t zeros(1,6-rn)];
  p  = [p zeros(1,6-rn)];
  rn = 6;
end

%---- open file
filt=[fil '.t'];
i=0;
while exist(filt)==2
  i=i+1;
  filt=[fil int2str(i) '.t'];
end
disp(' ')
disp(['Out-file is ' filt])
disp(' ')

time_stamp=fix(clock);
time_stamp=[sprintf('%g-%g-%g ',time_stamp(1),time_stamp(2),time_stamp(3)) ..
            sprintf('%g:%g:%g',time_stamp(4),time_stamp(5),time_stamp(6))];

fprintf(filt,['" ' filt '\n'])
fprintf(filt,['" Created in Matlab at ' time_stamp '\n'])
fprintf(filt,'" \n')
fprintf(filt,['[' syst '] \n'])

%---- write polynomials

for i=2:rn,
  fprintf(filt,'r%g :  %g \n',i-1,r(i))
end;

for i=1:rn,
  fprintf(filt,'s%g :  %g \n',i-1,s(i))
end;

for i=1:rn,
  fprintf(filt,'t%g :  %g \n',i-1,t(i))
end;

if nargin>5
  for i=2:length(p),
    fprintf(filt,'p%g :  %g \n',i-1,p(i))
  end;
  if nargin==7
    fprintf(filt,'aw :  %g \n',aw)
  end
end;

-------------------- /regler/matlab/ppbox/rstc.m --------------------
```

```
function [r,s,t]=rstc(bplus,bminus,a,bm1,am,ao,ar,as)
% RSTC  Polynomial synthesis in continuous time.
%
%       [R,S,T]=RSTC(BPLUS,BMINUS,A,BM1,AM,AO,AR,AS)
%       [R,S,T]=RSTC(BPLUS,BMINUS,A,BM1,AM,AO,AR)
%       [R,S,T]=RSTC(BPLUS,BMINUS,A,BM1,AM,AO)
%
%       Polynomial synthesis according to CCS ch 10 to
%       design a controller R(s) u(s) = T(s) r(s) - S(s) y(s)
%
%       Inputs:  BPLUS   : Part of open loop numerator
%                BMINUS  : Part of open loop numerator
%                A       : Open loop denominator
%                BM1     : Additional zeros
%                AM      : Closed loop denominator
%                AO      : Observer polynomial
%                AR      : Pre-specified factor of R,
%                          e.g integral part [1 0]**k
%                AS      : Pre-specified factor of S,
%                          e.g notch filter [1 0 w^2]
%
%       Outputs: R,S,T   : Polynomials in controller
%
%       See function DAB how the solution to the Diophantine-
%       Aryabhatta-Bezout identity is chosen.

% Michael Lundh     LastEditDate : Wed Mar 21 14:33:47 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin==7, as=1; elseif nargin==6, ar=1; as=1; end;

ae      = conv(a,ar);
be      = conv(bminus,as);
aoam    = conv(am,ao);
[r1,s1] = dab(ae,be,aoam);

r       = conv(conv(r1,ar),bplus);
s       = conv(s1,as);

bm      = conv(bminus,bm1);
t0      = am(length(am))/bm(length(bm));
t       = t0*conv(ao,bm1);

s       = s/r(1);
t       = t/r(1);
r       = r/r(1);

-------------------- /regler/matlab/ppbox/rstd.m --------------------

function [r,s,t]=rstd(bplus,bminus,a,bm1,am,ao,ar,as);
% RSTD  Polynomial synthesis in discrete time.
%
%       [R,S,T]=RSTD(BPLUS,BMINUS,A,BM1,AM,AO,AR,AS);
%       [R,S,T]=RSTD(BPLUS,BMINUS,A,BM1,AM,AO,AR);
%       [R,S,T]=RSTD(BPLUS,BMINUS,A,BM1,AM,AO);
%
%       Polynomial synthesis according to CCS ch 10 to
%       design a controller R(q) u(k) = T(q) r(k) - S(q) y(k)
%
%       Inputs:  BPLUS   : Part of open loop numerator
%                BMINUS  : Part of open loop numerator
```

```
%               A       : Open loop denominator
%               BM1     : Additional zeros
%               AM      : Closed loop denominator
%               AO      : Observer polynomial
%               AR      : Pre-specified factor of R,
%                         e.g integral part [1 -1]**k
%               AS      : Pre-specified factor of S,
%                         e.g notch filter [1 0 w^2]
%
%       Outputs: R,S,T  : Polynomials in controller
%
%       See function DAB how the solution to the Diophantine-
%       Aryabhatta-Bezout identity is chosen.

% Michael Lundh     LastEditDate : Wed Mar 21 14:35:36 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin==7, as=1; elseif nargin==6, ar=1; as=1; end;

ae      = conv(a,ar);
be      = conv(bminus,as);
aoam    = conv(am,ao);
[r1,s1] = dab(ae,be,aoam);

r       = conv(conv(r1,ar),bplus);
s       = conv(s1,as);

bm      = conv(bminus,bm1);
t0      = sum(am)/sum(bm);
t       = t0*conv(ao,bm1);

s       = s/r(1);
t       = t/r(1);
r       = r/r(1);

-------------------- /regler/matlab/ppbox/sample.m --------------------

function [bd,ad]=sample(bc,ac,cdelay,h);
% SAMPLE
%       Sampling of continuous time system.
%
%       [Bd,Ad]=SAMPLE(Bc,Ac,Tsamp);
%       [Bd,Ad]=SAMPLE(Bc,Ac,Tau,Tsamp);
%
%       Computes the discrete time transfer function H(q)=Bd(q)/Ad(q)
%       when sampling the continuous system G(s)= Bc(s)/Ac(s) or
%       G(s)= Bc(s)/Ac(s)*exp(-Tau*s).

% Mats Lilja (original)
% Michael Lundh     LastEditDate : Wed Mar  7 16:36:06 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

  while abs(bc(1))<10000*eps, bc(1)=[]; end

  [a,b,c,d]=tf2ss(bc,ac);

  if nargin==3
    h = cdelay;
    [ax,bx] = c2d(a,b,h);
    [bd,ad] = ss2tf(ax,bx,c,d,1);
```

```
    while abs(bd(1)) < 1e-12,
      bd(1) = [];
    end;
    return
  end

  if abs(d)>eps
    error('System must be strictly proper if a delay is present')
  end

  ddelay = round(cdelay/h + 0.5)-1;
  tau = cdelay - h*ddelay;
  [fi,gam]=c2d(a,b,h);
  [slask,gam0]=c2d(a,b,h-tau);
  gam1 = gam - gam0;
  bigfi = [fi gam1; 0*c 0*d];
  biggam = [gam0 ; 1 ];
  bigc = [c 0*d];
  [bd,ad]=ss2tf(bigfi,biggam,bigc,d,1);
  ee=eye(1,ddelay+1);
  ad = conv(ad,ee);
  while max(abs([bd(length(bd)) ad(length(ad))])) < 1e-12,
    ad(length(ad)) = [];
    bd(length(bd)) = [];
  end;
  while abs(bd(1)) < 1e-12,
    bd(1) = [];
  end;

-------------------- /regler/matlab/ppbox/stabpartc.m --------------------

function [Bs,As,Bu,Au,D] = stabpartc(B,A)
% STABPART Compute the stable part of a continuous time transfer function
%
%       [Bs,As] = stabpartc(B,A)
%
%       Extracts the strictly proper, stable partial fraction
%       of the transfer function B(s)/A(s). The stability region
%       is defined as the open left half plane.
%
%       [Bs,As,Bu,Au,D] = stabpartc(B,A)
%
%       Separates the transfer function B(s)/A(s) into strictly proper stable,
%       strictly proper unstable and non-proper parts according to
%
%         B(s)   Bs(s)   Bu(s)
%         ---- = ----- + ----- + D(s)
%         A(s)   As(s)   Au(s)
%

% Mats Lilja
% LastEditDate : Tue Jul 17 11:47:56 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[D,B] = deconv(B,A);
B(1:min(find(B))-1) = [];   % remove leading zeros in B
Aroots = roots(A);
As = real(poly(Aroots(find(real(Aroots)<0))));
Au = real(poly(Aroots(find(real(Aroots)>=0))));
[Bs,Bu] = dab(Au,As,B);
```

```
-------------------- /regler/matlab/ppbox/stabpartd.m --------------------

function [Bs,As,Bu,Au,D] = stabpartd(B,A)
% STABPART Compute the stable part of a discrete time transfer function
%
%       [Bs,As] = stabpartd(B,A)
%
%       Extracts the strictly causal, stable partial fraction
%       of the transfer function B(z)/A(z). The stability region
%       is defined as the open unit circle.
%
%       [Bs,As,Bu,Au,D] = stabpartd(B,A)
%
%       Separates the transfer function B(z)/A(z) into strictly causal stable,
%       strictly causal unstable and non-causal parts according to
%
%         B(z)   Bs(z)   Bu(z)
%         ---- = ----- + ----- + D(z)
%         A(z)   As(z)   Au(z)
%

% Mats Lilja
% LastEditDate : Tue Jul 17 11:53:34 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

[D,B] = deconv(B,A);
B(1:min(find(B))-1) = [];   % remove leading zeros in B
Aroots = roots(A);
As = real(poly(Aroots(find(abs(Aroots)<1))));
Au = real(poly(Aroots(find(abs(Aroots)>=1))));
[Bs,Bu] = dab(Au,As,B);



-------------------- /regler/matlab/ppbox/yupl.m --------------------

function yupl(tr1,tr2,tr3,tr4,scale)
% YUPL  Plot time responses.
%
%       YUPL(TR1,TR2,TR3,TR4,SCALE)
%
%       A plot of simulation results TR1 - TR4 from YUSIMC or YUSIMD is
%       done.  The arguments TR2 - TR4 are optional. TR1 - TR4 are
%       allowed to have different number of data points.
%
%       The optional argument SCALE is used to affect the scaling of the
%       plot. It takes the form [tmin tmax ymin ymax umin umax].
%       SCALE can be used even if TR2 - TR4 are omitted.
%
%       The scales are stored in the variable glob_scale, which must be
%       declared as global. The scales are used to be able to draw
%       consecutive plots (using YUSH) in the same diagram.

% Michael Lundh     LastEditDate : Tue Mar 13 22:44:08 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if ppcheck, return, end
```

```
if nargin==4,
  scale = tr4;
elseif nargin==3,
  scale = tr3;
elseif nargin==2,
  scale = tr2;
elseif nargin==1,
  scale = [];
end

if size(scale) == [1 6],
  npl = nargin-1;
  yscale = scale(1,1:4);
  uscale = scale(1,[1,2,5,6]);
else
  npl = nargin;
  yscale = [];
  uscale = [];
end

[m,n]=size(tr1);
t1=tr1(:,1);  r1=tr1(:,2);  y1=tr1(:,3:2+(n-2)/2);  u1=tr1(:,3+(n-2)/2:n);
do1='plot(t1,y1,t1,r1,''-'' ';
do2='plot(t1,u1';

if npl>1
  [m,n]=size(tr2);
  t2=tr2(:,1);  r2=tr2(:,2);  y2=tr2(:,3:2+(n-2)/2);  u2=tr2(:,3+(n-2)/2:n);
  do1=[do1 ',t2,y2,t2,r2,''-'' '];
  do2=[do2 ',t2,u2'];
end
if npl>2
   [m,n]=size(tr3);
  t3=tr3(:,1);  r3=tr3(:,2);  y3=tr3(:,3:2+(n-2)/2);  u3=tr3(:,3+(n-2)/2:n);
  do1=[do1 ',t3,y3,t3,r3,''-'' '];
  do2=[do2 ',t3,u3'];
end
if npl>3
  [m,n]=size(tr4);
  t4=tr4(:,1);  r4=tr4(:,2);  y4=tr4(:,3:2+(n-2)/2);  u4=tr4(:,3+(n-2)/2:n);
  do1=[do1 ',t4,y4,t4,r4,''-'' '];
  do2=[do2 ',t4,u4'];
end

do1=[do1 ')'];
do2=[do2 ')'];

clg
subplot;
subplot(211);
axis([0 1 0 1]);axis; % autorange axis
if length(yscale)>0, axis(yscale); end;
eval(do1);
ylabel('r and y');
text(0.83,0.61,'time [s]','sc')
glob_scale = axis;
axis(glob_scale);
axis;

subplot(212);
if length(uscale)>0, axis(uscale); end;
eval(do2);
```

```
ylabel('u')
text(0.83,0.11,'time [s]','sc')
glob_scale = [glob_scale ;axis];
axis(glob_scale(2,:));
axis;

subplot(211);
```

--------------------- /regler/matlab/ppbox/yush.m --------------------

```
function yush(tr,option)
% YUSH  Show an YU plot in a previously drawn YU diagram.
%
%       YUSH(TR,OPTION)
%
%       An YU plot is done from the time responses in TR using
%       plot-option OPTION. The argument option is optional.
%
%       The scales of the y and u plot are taken from the
%       variable glob_scale.  This variable is assumed
%       having been declared as global.

% Michael Lundh     LastEditDate : Wed Mar  7 17:02:29 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if length(glob_scale) < 1,
  error('No previous YU plot');
end;

[m,n]=size(tr);
t1=tr(:,1);  r1=tr(:,2);  y1=tr(:,3:2+(n-2)/2);  u1=tr(:,3+(n-2)/2:n);

subplot(211);
axis(glob_scale(1,:));
if nargin == 1,
  plot(t1,y1,t1,r1,'-');
else
  plot(t1,y1,option,t1,r1,'-');
end;

subplot(212);
axis(glob_scale(2,:));
if nargin == 1,
  plot(t1,u1)
else
  plot(t1,u1,option)
end;
axis;
subplot(211);
```

--------------------- /regler/matlab/ppbox/yusignals.m --------------------

```
function trldn=yusignals(t,rlev,rch,llev,lch,dlev,dch,nvar,nch)
% YUSIGNALS
%       Create signals for simulation.
%
%       TRLDN=YUSIGNALS(TIME,RLEV,RCH,LLEV,LCH,DLEV,DCH,NVAR,NCH)
%
%       Signal generator for TRLDN in YUSIMC and YUSIMD. A matrix
%
%         TRLDN=[t r l d n]
```

```
%
%       is created.  Its first column defines the simulation time, other
%       columns correspond to reference signal r, input disturbance l,
%       output disturbance d and measurement noise n respectively at the
%       time instants of first column.
%
%       Each signal s is defined by two vectors sLEV and sCH. Initially
%       s is zero. Changes occur at occasions in vector sCH. The signal
%       s is sLEV from time sCH.
%       NVAR is variance for Gaussian noise from time NCH.
%
%       The function may have 1, 3, 5, 7 or 9 input arguments.
%       In case of only one argument, ref=1 from time=0, l=-1
%       from max(TIME)/3, d=-1 from max(TIME)*2/3 and n=0;
%       In case of more input arguments signals not being referenced
%       to are zero.

% Michael Lundh     LastEditDate : Wed Mar 21 13:06:36 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if rem(nargin,2)~=1, error('wrong number of inputs'); end

epps = 10000*eps;
tmax = max(t);

t=t(:);
l=0*t; d=0*t; n=0*t;

narginZ = nargin;
if narginZ==1
  narginZ=7;
  rlev=1 ; rch=0;
  llev=-1; lch=tmax/3;
  dlev=-1; dch=tmax*2/3;
end

rch=[rch tmax+1];
for i = 1:length(rlev)
  ind    = find( ((rch(i)-epps)<=t) & (t<rch(i+1)) );
  r(ind) = rlev(i)*ones(1,length(ind));
end

if narginZ>3
  lch=[lch tmax+1];
  for i = 1:length(llev)
    ind    = find( ((lch(i)-epps)<=t) & (t<lch(i+1)) );
    l(ind) = llev(i)*ones(1,length(ind));
  end;
end

if narginZ>5
  dch=[dch tmax+1];
  for i = 1:length(dlev)
    ind    = find( ((dch(i)-epps)<=t) & (t<dch(i+1)) );
    d(ind) = dlev(i)*ones(1,length(ind));
  end;
end

if narginZ>7
  rand('normal');rand('seed',0)
  nn=rand(length(t),1);
```

```
  nch=[nch tmax+1];
  for i = 1:length(nvar)
    ind    = find( ((nch(i)-epps)<=t) & (t<nch(i+1)) );
    n(ind) = sqrt(nvar(i))*nn(ind);
  end;
end

trldn=[t(:) r(:) l(:) d(:) n(:)];
```

```
-------------------- /regler/matlab/ppbox/yusimc.m --------------------

function tryu=yusimc(bp,ap,r,s,t,bff,aff,trldn)
%YUSIMC Simulation of continuous time SISO system.
%
%       TRYU=YUSIMC(B,A,R,S,T,TRLDN)
%       TRYU=YUSIMC(B,A,R,S,T,BFF,AFF,TRLDN)
%
%       The strictly proper system
%
%       y(s) = B(s)/A(s)*(u(s)+l(s)) + d(s)
%
%       is controlled using
%
%       R(s)*u(s) = T(s)*BFF(s)/AFF(s)*ref(s)
%                   - S(s)*(y(s)+n(s))
%
%       The feedforward filter BFF/AFF is optional.
%
%       If B has n rows and A has 1 row then n systems B(i,:)/A are
%       simulated with the same controller.  If B has 1 row and A has
%       n rows then n systems B/A(i,:) are simulated.  If B and A both
%       have n rows then n systems B(i,:)/A(i,:) are simulated.  This
%       feature is nice for simulation of systems with parametric
%       uncertainty.
%
%       System output and control signal are calculated and stored in
%       the matrix TRYU=[TIME REF Y1 .. Yn U1 .. Un]. This time response
%       is displayed using YUPL and YUSH.
%
%       Argument TRLDN defines simulation time and external signals.
%       It may have different formats:
%         Scalar TMAX:  Simulation time defined by vector 0:TMAX with
%                       151 points.
%                       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.
%         Vector TIME:  Simulation time is defined. Signals as above.
%         Matrix TRLDN: A matrix with 2 - 5 columns.
%                       First column defines simulation time. Second
%                       column defines the reference signal r at time-
%                       instants of first column. If present column
%                       three to five define input disturbance l, output
%                       disturbance d and measurement noise n respectively.
%                       Omitted column implies that the corresponding
%                       signal is zero.
%                       The matrix TRLDN may be generated by YUSIGNALS.
%

% Michael Lundh     LastEditDate : Wed Mar  7 16:38:58 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN
```

```
if nargin==6
  trldn=bff;  bff=1;  aff=1;
end;

%---- assign trldn ----
if length(trldn)==1            % only maxtime defined
  trldn=(0:trldn/150:trldn)';
end

[m,n]=size(trldn);
if (n==1) | (m==1)             % only time specified
  trldn = trldn(:);
  m     = max(m,n);
  trldn = [trldn ones(m,1) zeros(m,3)];
  mi    = floor(m/3)+1;
  trldn(mi:m,3) = -ones(m-mi+1,1);
  mi    = floor(2*m/3)+1;
  trldn(mi:m,4) = -ones(m-mi+1,1);
else
  trldn=[trldn zeros(m,5-n)];
end

%---- RST-controller ----
[fb,hb,gb1,kb1] = tf2ss(t,r);
[fb,hb,gb2,kb2] = tf2ss(s,r);
fb  = fb' ;
hb  = hb' ;
gb1 = gb1';
gb2 = gb2';


%---- FF-filter ----
[ff,gf,hf,kf] = tf2ss(bff,aff);
mff=size(ff)*[1;0];

nbpol = size(bp)*[1;0];
napol = size(ap)*[1;0];
if napol~=nbpol
  if napol==1,     ap = kron(ap,ones(nbpol,1));
  elseif nbpol==1, bp = kron(bp,ones(napol,1));
  else error('More than 1 row in A and B and not same number of rows')
  end
end
nbpol = size(bp)*[1;0];

disp(sprintf('Simulation of %g system(s)',nbpol))
yy=[];uu=[];
for i=1:nbpol
  %---- Open system ----
  api = ap(i,:);
  bpi = bp(i,:);
  while abs(bpi(1))<10000*eps, bpi(1)=[]; end
  if length(api)<=length(bpi)
    error('B/A is not strictly proper')
  end

  [a,b,c,d]=tf2ss(bpi,api);

  aa = [a-b*kb2*c  b*hb ; -gb2*c fb];
  [maa,naa]=size(aa);
```

```
  aaa=[aa [b*kb1*hf ; gb1*hf] ; zeros(mff,naa) ff];
  bbb=[b*kb1*kf b -b*kb2 -b*kb2  ; gb1*kf 0*gb2 -gb2 -gb2 ; gf 0*[gf gf gf]];
  ccc=[ c 0*hb 0*hf ; -kb2*c hb kb1*hf];
  ddd=[0 0 1 0 ; kb1*kf 0 -kb2 -kb2];

  yu = lsim(aaa,bbb,ccc,ddd,trldn(:,2:5),trldn(:,1));
  yy = [yy yu(:,1)];
  uu = [uu yu(:,2)];
  disp(sprintf(' %g done',i))
end

tryu=[trldn(:,1:2) yy uu];

-------------------- /regler/matlab/ppbox/yusimd.m --------------------

function tryu=yusimd(bp,ap,r,s,t,bff,aff,h,tmax)
% YUSIM Simulation of discrete time SISO system.
%
%       TRYU=YUSIMD(B,A,R,S,T,TRLDN)
%       TRYU=YUSIMD(B,A,R,S,T,H,TMAX)
%       TRYU=YUSIMD(B,A,R,S,T,BFF,AFF,TRLDN)
%       TRYU=YUSIMD(B,A,R,S,T,BFF,AFF,H,TMAX)
%
%       The strictly proper system
%
%       y(z) = B(z)/A(z)*(u(z)+l(z)) + d(z)
%
%       is controlled using
%
%       R(z)*u(z) = T(z)*BFF(z)/AFF(z)*ref(z)
%                   - S(z)*(y(z)+n(z))
%
%       The feedforward filter BFF/AFF is optional.
%
%       If B has n rows and A has 1 row then n systems B(i,:)/A are
%       simulated with the same controller.  If B has 1 row and A has
%       n rows then n systems B/A(i,:) are simulated.  If B and A both
%       have n rows then n systems B(i,:)/A(i,:) are simulated.  This
%       feature is nice for simulation of systems with parametric
%       uncertainty.
%
%       System output and control signal are calculated and stored in
%       the matrix TRYU=[TIME REF Y1 .. Yn U1 .. Un]. This time response
%       is displayed using YUPL and YUSH.
%
%       Argument TRLDN defines simulation time and external signals.
%       It may have different formats:
%         Scalar TMAX:  Simulation time defined by vector 0:TMAX where
%                       TMAX is a positive integer. Sampling interval is 1.
%                       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.
%         Vector TIME:  Simulation time is defined. Signals as above.
%                       Time increment defines the sampling interval.
%         Matrix TRLDN: A matrix with 2 - 5 columns.
%                       First column defines simulation time. Second
%                       column defines the reference signal r at time-
%                       instants of first column. If present column
%                       three to five define input disturbance l, output
%                       disturbance d and measurement noise n respectively.
%                       Omitted column implies that the corresponding
```

```
%                          signal is zero.
%                          The matrix TRLDN may be generated by YUSIGNALS.
%
%       Two scalars H and TMAX may instead be given to form TRLDN.  H is
%       the sampling interval.  Simulation time id defined by vector
%       0:H:TMAX.  Signals as for scalar TMAX in the definition of TRLDN.

% Michael Lundh     LastEditDate : Wed Mar  7 16:39:54 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin==6,       trldn=bff; bff=1; aff=1;
                    if length(trldn)==1
                      trldn=(0:trldn)';
                    end
elseif nargin==7,   trldn=(0:bff:aff)'; bff=1; aff=1;
elseif nargin==8,   trldn=h;
                    if length(trldn)==1
                      trldn=(0:trldn)';
                    end
elseif nargin==9,   trldn=(0:h:tmax)';
else                error('False number of imputs')
end;

%---- assign trldn ----
[m,n]=size(trldn);
if (n==1) | (m==1)              % only time specified
  trldn = trldn(:);
  m     = max(m,n);
  trldn = [trldn ones(m,1) zeros(m,3)];
  mi    = floor(m/3)+1;
  trldn(mi:m,3) = -ones(m-mi+1,1);
  mi    = floor(2*m/3)+1;
  trldn(mi:m,4) = -ones(m-mi+1,1);
else
  trldn=[trldn zeros(m,5-n)];
end

%---- RST-controller ----
[fb,hb,gb1,kb1] = tf2ss(t,r);
[fb,hb,gb2,kb2] = tf2ss(s,r);
fb  = fb' ;
hb  = hb' ;
gb1 = gb1';
gb2 = gb2';


%---- FF-filter ----
[ff,gf,hf,kf] = tf2ss(bff,aff);
mff=size(ff)*[1;0];

nbpol = size(bp)*[1;0];
napol = size(ap)*[1;0];
if napol~=nbpol
  if napol==1,      ap = kron(ap,ones(nbpol,1));
  elseif nbpol==1, bp = kron(bp,ones(napol,1));
  else error('More than 1 row in A and B and not same number of rows')
  end
end
nbpol = size(bp)*[1;0];

disp(sprintf('Simulation of %g system(s)',nbpol))
```

```
yy=[];uu=[];
for i=1:nbpol
  %---- Open system ----
  api = ap(i,:);
  bpi = bp(i,:);
  while abs(bpi(1))<10000*eps, bpi(1)=[]; end
  if length(api)<=length(bpi)
    error('B/A is not strictly proper')
  end

  [a,b,c,d]=tf2ss(bpi,api);

  aa = [a-b*kb2*c  b*hb ; -gb2*c fb];
  [maa,naa]=size(aa);

  aaa=[aa [b*kb1*hf ; gb1*hf] ; zeros(mff,naa) ff];
  bbb=[b*kb1*kf b -b*kb2 -b*kb2  ; gb1*kf 0*gb2 -gb2 -gb2 ; gf 0*[gf gf gf]];
  ccc=[ c 0*hb 0*hf ; -kb2*c hb kb1*hf];
  ddd=[0 0 1 0 ; kb1*kf 0 -kb2 -kb2];

  yu = dlsim(aaa,bbb,ccc,ddd,trldn(:,2:5));
  yy = [yy yu(:,1)];
  uu = [uu yu(:,2)];
  disp(sprintf(' %g done',i))
end

tryu=[trldn(:,1:2) yy uu];

-------------------- /regler/matlab/ppbox/yustairs.m --------------------

function tryu2=yustairs(tryu1,ryu)
%YUSTAIRS
%       Modification of simulation outputs.
%
%       TRYU2=YUSTAIRS(TRYU1)
%       TRYU2=YUSTAIRS(TRYU1,RYU)
%
%       The function modifies the output from the routines YUSIMD to make
%       the plot more discrete like.  Additional points are added in order
%       to plot some of the signals r, y and u as stairs, as if they were
%       outputs from zero order hold circuits.
%       The optional parmeter RYU is any combination of 'u' 'y' 'r' to
%       define which signal that will be modified. Default is 'ru'.
%
%       Warning: The output from this function is recomended for plotting
%       only.

% Michael Lundh     LastEditDate : Tue Mar 20 13:11:39 1990
% Copyright (c) 1990 by Michael Lundh and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin==1; ryu='ru'; end

fixr = any('r'==ryu) | any('R'==ryu);
fixy = any('y'==ryu) | any('Y'==ryu);
fixu = any('u'==ryu) | any('U'==ryu);

[m,n] = size(tryu1);

%---- time column
ci    = [tryu1(1:m-1,1) tryu1(2:m,1)]';
tryu2 = ci(:);
```

```
%---- reference
if fixr
  ci = [tryu1(1:m-1,2) tryu1(1:m-1,2)]';
else
  ci = [tryu1(1:m-1,2) tryu1(2:m,2)]';
end
tryu2 = [tryu2 ci(:)];


for i=3:n
  if (rem(i,2)==1 & fixy) | (rem(i,2)==0 & fixu)
    ci = [tryu1(1:m-1,i) tryu1(1:m-1,i)]';
  else
    ci = [tryu1(1:m-1,i) tryu1(2:m,i)]';
  end
  tryu2 = [tryu2 ci(:)];
end
```

3. RLBOX

```
-------------------- /regler/matlab/rlbox/dsymloc.m --------------------

function locus = dsymloc(b,a,r1,r2,dsmax,scale)
% DSYMLOC Plots symmetric root locus for discrete system
%
%       locus = dsymloc(b,a,r1,r2,dsmax,scale)
%       locus = dsymloc(b,a,r1,r2,dsmax)
%
%       Plots the stable part of the root locus corresponding to
%       a(z)*a(1/z) + r*b(z)*b(1/z) for r1 < r < r2. Also the roots of
%       a (r = Inf) and the roots of b (r = 0) are plotted.
%
%       The increments in r are culculated such that abs(ds) < dsmax.
%       The r-values and the roots are stored in locus, with the r-values
%       in the first column.
%
%       scale (optional) takes the form [xmax xmin ymin ymax]. If scale is
%       omitted an attempt is done to find a plot scale that gives an aspect
%       ratio as close to 1 as possible.
%
%       Use PZGRID(1) to plot grid and unit circle.

% Kjell Gustafsson, original file due to Ulf Holmberg
% LastEditDate : Thu Jun 14 17:35:01 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

aspec = conv(a,a(length(a):-1:1));
bspec = conv(b,b(length(b):-1:1));

reldeg = length(a) - length(b);
if reldeg>=0,
  b = [0*(1:reldeg) b];
else
  a = [0*(1:-reldeg) a];
end

ainv = a(length(a):-1:1);
binv = b(length(b):-1:1);
asym = conv(b,binv);
bsym = conv(a,ainv);
while asym(length(asym))==0 & bsym(length(bsym))==0
  asym = asym(1:length(asym)-1);
  bsym = bsym(1:length(bsym)-1);
end

loc = rootlocus(bsym,asym,r1,r2,dsmax,1);

% sort out stable roots

[n,m] = size(loc);
locus = [];
for k = 1:n,
  [imvec,imindex] = sort(imag(loc(k,2:m)));
  loc(k,2:m) = loc(k,imindex+1);
  index1 = find(abs(loc(k,2:m))==1);
  index = find(abs(loc(k,2:m))<1);
  locus = [locus; loc(k,1) loc(k,index+1) loc(k,index1(1:2:length(index1))+1)];
end
```

```
rl = locus(:,2:size(locus)*[0;1]);

po = roots(aspec);
[tmp,ind] = sort(abs(po));
po = po(ind(1:length(ind)/2));
ze = roots(bspec);
[tmp,ind] = sort(abs(ze));
ze = ze(ind(1:length(ind)/2));

if reldeg>=0,
  ze = [zeros(reldeg,1); ze];
else
  po = [zeros(-reldeg,1); po];
end

if nargin<6
  Re_max = max([max(real(rl)) real(po') real(ze')]);
  Re_min = min([min(real(rl)) real(po') real(ze')]);
  Im_max = max([max(imag(rl)) imag(po') imag(ze')]);
  dRe = Re_max - Re_min;
  dIm = 2*Im_max;
  mRe = (Re_min + Re_max)/2;
  d = max(dRe,dIm);
  if d==0
    w = [-abs(Re_max) abs(Re_max)]*1.1;
  else
    w = [-d/2 d/2]*1.1;
  end
  w = [-d/2 d/2]*1.1;
  scale = [w+mRe w];
end

axis('square');
axis(scale);
plot(0,0,'i');
hold on;
plot(real(rl),imag(rl),'.');
mark(ze,4,[],[],1);
mark(po,2,[],[],1);
xlabel('Re');
ylabel('Im');
hold off;
axis('normal');

-------------------- /regler/matlab/rlbox/mark.m --------------------

function mark(pos,mtype,msize,mindex,aspect)
% MARK  plot and index markings
%
%       mark(pos,mtype,msize,mindex,aspect)
%
%       A marking of type mtype is made at position pos with size msize. The
%       marking is superindexed with the numerical value in mindex. If pos is
%       a vector then mtype, msize, and/or mindex can be supplied as scalars
%       or vectors. If they are scalars the same type (size, index) is used for
%       all values in pos. Otherwise mtype(i), msize(i), and mindex(i) are used
%       for pos(i).
%
%       mtype, msize, and mindex are all optional. If omitted they default to
%       1, 0.01, and NaN, respectively. NaN at any position in mindex omits
%       indexing at the corresponding point. Parameters inside the parameter
```

```
%       list are omitted by supplying an empty matrix, i.e. [].
%
%       Possible mark types are
%
%       1  plus         4  ring          7  hollow plus      10  natostar
%       2  cross        5  heart         8  smiley face
%       3  diamond      6  triangle      9  pentagon
%
%       aspect is an optional argument telling the aspect ratio of the screen.
%       Default value is 137/99 (the normal screen). aspect = 1 should be used
%       with axis('square').
%
%       hold will be off after having executed this routine.

% Per Persson (original routine plopp)
% Kjell Gustafsson (revised version)
% LastEditDate : Thu Jul  5 16:06:04 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin<5, aspect = 137/99; end
if nargin<4, mindex = NaN; end
if nargin<3, msize = 0.01; end
if nargin<2, mtype = 1; end

if size(mtype)==[0 0], mtype = 1; end
if size(msize)==[0 0], msize = 0.01; end
if size(mindex)==[0 0], mindex = NaN; end

lpos = length(pos);
if lpos>1
  if length(mtype)==1, mtype = mtype*ones(1,lpos); end
  if length(msize)==1, msize = msize*ones(1,lpos); end
  if length(mindex)==1, mindex = mindex*ones(1,lpos); end
end

i = sqrt(-1);
ax = axis;
axis;

for ipos=1:lpos
  dx = msize(ipos)*(ax(2) - ax(1))/aspect;
  dy = msize(ipos)*(ax(4) - ax(3));
  x = real(pos(ipos));
  y = imag(pos(ipos));
  hold on;
  dx1 = 5*dx; dy1 = 5*dy;
  if (x-dx1 > ax(1)) & (x+dx1 < ax(2)) & (y-dy1 > ax(3)) & (y+dy1 < ax(4)),
    if mtype(ipos) == 1,
      plot([(x - dx), (x + dx)], [y, y]);
      plot([x, x], [(y - dy), (y + dy)]);
    elseif mtype(ipos) == 2,
      sc = sqrt(0.5); dx1 = sc*dx; dy1 = sc*dy;
      plot([(x - dx1), (x + dx1)], [(y - dy1), (y + dy1)]);
      plot([(x - dx1), (x + dx1)], [(y + dy1), (y - dy1)]);
    elseif mtype(ipos) == 3,
      sc = sqrt(0.5); dx1 = sc*dx; dy1 = sc*dy;
      plot([(x - dx1) x], [y (y + dy1)]);
      plot([x (x + dx1)], [(y + dy1) y]);
      plot([(x - dx1) x], [y (y - dy1)]);
      plot([x (x + dx1)], [(y - dy1) y]);
    elseif mtype(ipos) == 4,
```

```
      tmp = [0:2*pi/20:2*pi]';
      res = sqrt(0.5)*[dx*cos(tmp) dy*sin(tmp)];
      plot(res(:, 1) + x, res(:, 2) + y);
    elseif mtype(ipos) == 5,
      dx1 = 1.5*dx; dy1 = 1.5*dy;
      tmp1 = [-pi/4:pi/20:3*pi/4]';
      tmp2 = [pi/4:pi/20:5*pi/4]';
      res1 = [dx1*cos(tmp1) dy1*sin(tmp1)]/sqrt(8);
      res2 = [dx1*cos(tmp2) dy1*sin(tmp2)]/sqrt(8);
      plot(res1(:, 1) + x + dx1/4, res1(:, 2) + y + dy1/4);
      plot(res2(:, 1) + x - dx1/4, res2(:, 2) + y + dy1/4);
      plot((dx1*[0 0.5] + x), (dy1*[-0.5 0] + y));
      plot((dx1*[-0.5 0] + x), (dy1*[0 -0.5] + y));
    elseif mtype(ipos) == 6,
      tmp = [0:2*pi/3:2*pi]';
      res = sqrt(0.5)*[dx*cos(tmp) dy*sin(tmp)];
      plot(res(:, 1) + x, res(:, 2) + y);
    elseif mtype(ipos) == 7,
      m1 = [(x-dx) (x-dx/4) (x-dx/4) (x+dx/4) (x+dx/4),..
            (x+dx) (x+dx) (x+dx/4) (x+dx/4) (x-dx/4) (x-dx/4) (x-dx) (x-dx)];
      m2 = [(y+dy/4) (y+dy/4) (y+dy) (y+dy) (y+dy/4) (y+dy/4),..
            (y-dy/4) (y-dy/4) (y-dy) (y-dy) (y-dy/4) (y-dy/4) (y+dy/4)];
      plot(m1, m2);
    elseif mtype(ipos) == 8,
      tmp = [0:2*pi/20:2*pi]';
      c = [dx*cos(tmp) dy*sin(tmp)];
      res = 1.5*c;
      plot(res(:, 1) + x, res(:, 2) + y);
      res = 0.2*c;
      plot(res(:, 1) + x - 0.6*dx, res(:, 2) + y + 0.5*dy);
      plot(res(:, 1) + x + 0.6*dx, res(:, 2) + y + 0.5*dy);
      tmp = [pi*1.1:2*pi/50:1.9*pi]';
      c = [dx*cos(tmp) dy*sin(tmp)];
      res = 0.9*c;
      plot(res(:, 1) + x, res(:, 2) + y - 0.2*dy);
    elseif mtype(ipos) == 9,
      tmp = [0:2*pi/5:2*pi]';
      res = sqrt(0.5)*[dx*cos(tmp) dy*sin(tmp)];
      plot(res(:, 1) + x, res(:, 2) + y);
    elseif mtype(ipos) == 10,
      plot([0, dx/4, dx, dx/4, 0, -dx/4, -dx, -dx/4, 0] + x,..
           [dy, dy/4, 0, -dy/4, -dy, -dy/4, 0, dy/4, dy] + y);
    end;
  end;
  hold off;
  if mindex(ipos) ~= NaN
    numbers(mindex(ipos), x + dx + i*(y + dy));
  end;
end;

-------------------- /regler/matlab/rlbox/numbers.m --------------------

function numbers(n,z)
% NUMBERS print numeric value in plot
%
%       numbers(n,z)
%
%       The numeric value n is printed at position z. If z is omitted it
%       defaults to 0.

% Per Persson
% LastEditDate : Thu Jul  5 15:04:02 1990
```

```
% Copyright (c) 1990 by Per Persson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin == 1, z = 0; end;
x = real(z);
y = imag(z);
pz = 0.01;
ax = axis;
axis;
r = pz*(ax(2) - ax(1));
for ix = (abs(sprintf('%g', n))),
  n = ix;
  hold on;
  if n == 49,
    [x1, y1] = plotline(x, y, 1.5*r, pi/2);
    plotline(x1, y1, 0.3*r, 5*pi/4);
    x = x + 0.8*r;
  elseif n == 50,
    plot(x + [-r/2, r/2], y + [0, 0]);
    plot(x + [-r/2, r/(2*sqrt(2))], y + [0, r*(1 - 1/(2*sqrt(2)))]);
    plotcircle(r/2, -pi/4, pi, x, y + r, '-', 10);
    x = x + 1.1*r;
  elseif n == 51,
    plot(x + [-r/2, r/2, 0], y + [1.5*r, 1.5*r, r]);
    plotcircle(r/2, -pi, pi/2, x, y + r/2, '-', 10);
    x = x + 1.1*r;
  elseif n == 52,
    plot(x + [0, 0, -0.6*r, 0.35*r], y + [0, 1.5*r, 0.7*r, 0.7*r]);
    x = x + 0.75*r;
  elseif n == 53,
    plot(x + [-0.5*r/sqrt(2), -0.5*r/sqrt(2), r/2],..
         y + [r*(1/2 + 0.5/sqrt(2)), 1.5*r, 1.5*r]);
    plotcircle(r/2, -3*pi/4, 3*pi/4, x, y + r/2, '-', 10);
    x = x + 1.1*r;
  elseif n == 54,
    plot(x + [-r/2, -r/2], y + [r/2, r]);
    plotcircle(r/2, 0, 2*pi, x, y + r/2, '-', 10);
    plotcircle(r/2, pi/6, pi, x, y + r, '-', 10);
    x = x + 1.1*r;
  elseif n == 55,
    plot(x + [-0.35*r, 0.5*r, 0], y + [1.5*r, 1.5*r, 0]);
    x = x + r;
  elseif n == 56,
    plotcircle(r/2, 0, 2*pi, x, y + r/2, '-', 10);
    fi = acos(-((r/2)^2 - (0.35*r)^2 - (0.75*r)^2)/(2*(r*0.35)*(0.75*r)));
    plotcircle(0.35*r, pi + (pi/2 - fi), -(pi/2 - fi), x, y + 1.25*r, '-', 10);
    x = x + 1.1*r;
  elseif n == 57,
    plot(x + [r/2, r/2], y + [r/2, r]);
    plotcircle(r/2, 7*pi/6, 2*pi, x, y + r/2, '-', 10);
    plotcircle(r/2, 0, 2*pi, x, y + r, '-', 10);
    x = x + 1.1*r;
  elseif n == 48,
    plot(x + [(r/2), (r/2)], y + [(r/2), r]);
    plot(x + [(-r/2), (-r/2)], y + [(r/2), r]);
    plotcircle(r/2, 0, -pi, x, y + r/2, '-', 10);
    plotcircle(r/2, 0, pi, x, y + r, '-', 10);
    x = x + 1.1*r;
  elseif n == 43,
    plot(x + [-0.5*r, 0.5*r], y + [0.55*r, 0.55*r]);
    plot(x + [0, 0], y + [0.2*r, 1.3*r]);
    x = x + 1.0*r;
```

```
  elseif n == 45,
    plot(x + [-0.5*r, 0.5*r], y + [0.55*r, 0.55*r]);
    x = x + 1.3*r;
  elseif n == 46,
    x = x - 0.3*r;
    plotcircle(0.1*r, 0, 2*pi, x, y);
    x = x + 0.8*r;
  elseif n == 101,
    x = x - 0.25*r;
    plot(x + [0.7*r, 0, 0, 0.7*r], y + [0, 0, r, r]);
    plot(x + [0, 0.6*r], y + [0.5*r, 0.5*r]);
    x = x + 1.3*r;
  end;
  hold off;
end;
```

```
-------------------- /regler/matlab/rlbox/plotcircle.m --------------------

function res = plotcircle(rad, phi1, phi2, x, y, lt, ddeg)
% PLOTCIRCLE plot a circle segment
%
%       res = plotcircle(rad, phi1, phi2, x, y, lt, ddeg)
%
%       Plots a circle segment with radius rad from angle phi1 to phi2 with
%       center x, y. The linetype is lt and the angle increment in the plot
%       is ddeg. lt and ddeg are optional with default value '-' and 1,
%       respectively. The output argument res is the plotted circle segment.

% Per Persson
% LastEditDate : Thu Jul  5 15:03:55 1990
% Copyright (c) 1990 by Per Persson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin<7, ddeg = 1; end;
if nargin<6, lt = '-'; end;

if phi2 < phi1, tmp = phi1; phi1 = phi2; phi2 = tmp; end;
tmp = [phi1:(ddeg*pi/180):phi2]';
res = rad*[cos(tmp) sin(tmp)];

hold on;
plot(res(:,1) + x, res(:,2) + y, lt);
hold off;
```

```
-------------------- /regler/matlab/rlbox/plotline.m --------------------

function [x1, y1] = plotline(x0, y0, r, angle)
% PLOTLINE plots a line
%
%       [x1,y1] = plotline(x0,y0,r,angle)
%
%       A line of length r is plotted in direction angle from x0, y0. The
%       endpoint is x1, y1.

% Per Persson
% LastEditDate : Thu Jul  5 15:03:38 1990
% Copyright (c) 1990 by Per Persson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN
```

```
hold on;
x1 = x0 + r*cos(angle);
y1 = y0 + r*sin(angle);
plot([x0 x1], [y0 y1]);
hold off;
```

-------------------- /regler/matlab/rlbox/polyder.m --------------------

```
function derpol = polyder(pol,n);
% POLYDER Differentiate polynomials
%
%       derpol = polyder(pol,n)
%
%       Computes the n:th derivative of the polynomial pol. The argument
%       n is optional. It is taken as 1 if omitted.

% Kjell Gustafsson, stolen from Mats Lilja
% LastEditDate : Wed Mar  7 14:56:26 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin == 1,
  n = 1;
end;
if n < 1,
  derpol = pol;
else,
  pol = polyder(pol,n-1);
  np = length(pol)-1;
  nn = np:-1:0;
  derpol = nn.*pol;
  derpol = derpol(1:np);
end;
if length(derpol) < 1,
  derpol = 0;
end;
```

-------------------- /regler/matlab/rlbox/pzgrid.m --------------------

```
function pzgrid(disc)
% PZGRID draw grid in pole-zero plot
%
%       pzgrid(disc)
%
%       If the (optional) argument disc is supplied a unit circle will be
%       drawn. When plotting the circle Matlab prints a warning. Don't mind!
%
%       The Matlab function zgrid might be useful when plotting discrete-time
%       poles and zeros.

% Kjell Gustafsson
% LastEditDate : Fri Jun  8 13:53:56 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

axis('square');
hold on;
grid
if nargin==1
  uc = exp(sqrt(-1)*(0:pi/50:2*pi));
  plot([uc; uc([2:101,2])],'-');         % unit circle
```

```
else
  plot([-1E10; 1E10],[0; 0]);          % real axis
  plot([0; 0],[-1E10; 1E10]);          % imaginary axis
end
hold off
axis('normal');
```

```
-------------------- /regler/matlab/rlbox/pzpl.m --------------------

function pzpl(b,a,pmark,zmark,msize,scale)
% PZPL  pole-zero plot
%
%       pzpl(b,a,pmark,zmark,msize,scale)
%       pzpl(b,a,pmark,zmark,msize)
%       pzpl(b,a,pmark,zmark)
%       pzpl(b,a)
%
%       Plots the poles and zeros of the transfer function b(s)/a(s). Poles
%       and zeros are marked with mark type pmark and zmark, respectively.
%       Possible mark types are
%
%       1  plus        4  ring         7  hollow plus      10  natostar
%       2  cross       5  heart        8  smiley face
%       3  diamond     6  triangle     9  pentagon
%
%       If pmark and/or zmark are omitted or supplied as [] they default to
%       2 (cross) and 4 (ring). The marks have the size msize which defaults
%       to 0.01 if msize is omitted or supplied as [].
%
%       scale (optional) takes the form [xmax xmin ymin ymax]. If scale is
%       omitted an attempt is done to find a plot scale that gives an aspect
%       ratio as close to 1 as possible.

% Mats Lilja (original)
% Kjell Gustafsson
% LastEditDate : Thu Jul  5 15:00:12 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin<6
  r = [roots(b); roots(a)];
  Re_max = max(real(r));
  Re_min = min(real(r));
  Im_max = max(imag(r));
  dRe = Re_max - Re_min;
  dIm = 2*Im_max;
  mRe = (Re_min + Re_max)/2;
  d = max(dRe,dIm);
  if d==0
    w = [-abs(Re_max) abs(Re_max)]*1.1;
  else
    w = [-d/2 d/2]*1.1;
  end
  scale = [w+mRe w];
end

if nargin<3, pmark = 2; end
if nargin<4, zmark = 4; end
if nargin<5, msize = 0.01; end

if size(pmark)==[0 0], pmark = 2; end
if size(zmark)==[0 0], zmark = 4; end
```

```
if size(msize)==[0 0], msize = 0.01; end

axis('square');
axis(scale);
plot(0,0,'i');
mark(roots(b),zmark,msize,[],1);
mark(roots(a),pmark,msize,[],1);
xlabel('Re');
ylabel('Im');
axis('normal');

-------------------- /regler/matlab/rlbox/pzsh.m --------------------

function pzsh(b,a,pmark,zmark,msize)
% PZSH  add to a previous pole-zero plot
%
%       pzsh(b,a,pmark,zmark,msize)
%       pzsh(b,a,pmark,zmark)
%       pzsh(b,a)
%
%       Plots the poles and zeros of the transfer function b(s)/a(s). Poles
%       and zeros are marked with mark type pmark and zmark, respectively.
%       Possible mark types are
%
%       1  plus        4  ring          7  hollow plus      10  natostar
%       2  cross       5  heart         8  smiley face
%       3  diamond     6  triangle      9  pentagon
%
%       If pmark and/or zmark are omitted or supplied as [] they default to
%       2 (cross) and 4 (ring). The marks have the size msize which defaults
%       to 0.01 if msize is omitted or supplied as [].

% Kjell Gustafsson
% LastEditDate : Thu Jul  5 15:02:37 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin<3, pmark = 2; end
if nargin<4, zmark = 4; end
if nargin<5, msize = 0.01; end

if size(pmark)==[0 0], pmark = 2; end
if size(zmark)==[0 0], zmark = 4; end
if size(msize)==[0 0], msize = 0.01; end

axis('square');
mark(roots(b),zmark,msize,[],1);
mark(roots(a),pmark,msize,[],1);
axis('normal');

-------------------- /regler/matlab/rlbox/rlbox.m --------------------

% RLBOX -- A collection of routines to make pole-zero plots, and to calculate
%          and plot root loci. Written by Kjell Gustafsson, but in many cases
%          originating from Mats Lilja and Per Persson.
%
% Pole-zero plots
%
%   PZPL     - pole-zero plot
%   PZSH     - add to a previous pole-zero plot
%   PZGRID   - make grid in pole-zero plot
%   MARK     - general routine for plotting and indexing markings
```

```
%
% Root loci
%
%   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 system
%   DSYMLOC     - plot stable part of symmetric LQ locus for discrete system
%   ROOTLOCUS   - calculate root locus
%
%   All the root locus plot routines use ROOTLOCUS to calculate the root
%   locus. PZGRID can be used to add grid and/or unit circle to all the
%   root loci routines.
%
% Miscellaneous
%
%   Try RLDEMO

% Kjell Gustafsson
% LastEditDate : Thu Mar 22 22:16:51 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN


-------------------- /regler/matlab/rlbox/rldemo.m --------------------

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:12:17 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

echo on
clc
%       This example demonstrates some of the functions in RLBOX

%       Consider the continuous time system b(s)/a(s) where

b = poly(-4)
a = poly([0 -2 -6])

pause   % Strike RETURN to get a pole-zero plot

pzpl(b,a)
title('Pole-zero plot of the system b(s)/a(s)');

pause   % Strike RETURN to add a grid

pzgrid, pause

clc
%       Let's use a proportional controller to control the system.
%       The variation of the closed loop poles can be seen in a root
%       locus plot. RLOC1 plots the root locus including start and end
%       points.

pause   % Strike RETURN for a plot of the full root locus

rloc1(b,a,0.01,40,0.2);
title('Root locus of the system b(s)/a(s)'),  pause

clc
%       Suppose we are interested in the pole positions for gain variations
%       around the nominal value knom = 2. RLOC2 can be used to plot just
```

```
%       a part of the root locus.

        % Strike RETURN for a plot of the root locus for k = 1..4 with
pause   % the nominal value knom marked.

rloc2(b,a,1,2,4,0.2);
title('Root locus of b(s)/a(s) around the nominal point k = 2'), pause

clc
%       When designing continuous time LQ controllers it is interesting to
%       see the pole variations as function of rho when using the cost
%       function J = integ( y^2 + rho u^2 ). This can be achieved using
%       SYMLOC.

pause   % Strike RETURN for a plot of the continuous time rho root locus

symloc(b,a,0.01,10000,0.4);
title('Symmetric rho root locus for b(s)/a(s)'), pause

clc
%       There is a corresponding discrete time symmetric rho root locus. It
%       can be plotted using DSYMLOC. As discrete system we will use the
%       sampled double integrator (CCS Example 11.2)

bd = poly(-1)
ad = poly([1 1])

pause   % Strike RETURN for a plot of the discrete time rho root locus

dsymloc(bd,ad,1e-4,1e4,0.07);
title('Symmetric rho root locus for bd(z)/ad(z)'), pause

pause   % Strike RETURN to add grid and unit circle
pzgrid(1),

pause   % Strike RETURN to end

echo off

-------------------- /regler/matlab/rlbox/rloc1.m --------------------

function locus = rloc1(b,a,k1,k2,dsmax,scale);
% RLOC1 Plots full rootlocus including start and end points
%
%       locus = rloc1(b,a,k1,k2,dsmax,scale)
%       locus = rloc1(b,a,k1,k2,dsmax)
%
%       Plots the root locus of a(s) + k*b(s) for k1 < k < k2. Also the
%       roots of a (start points) and the roots of b (end points) are plotted.
%
%       The increments in k are chosen such that abs(ds) < dsmax.
%       The k-values and the roots are stored in locus, with the k-values
%       in the first column.
%
%       scale (optional) takes the form [xmax xmin ymin ymax]. If scale is
%       omitted an attempt is done to find a plot scale that gives an aspect
%       ratio as close to 1 as possible.

% Kjell Gustafsson, derived from routines by Mats Lilja
% LastEditDate : Thu Jun 14 17:34:15 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN
```

```
locus = rootlocus(b,a,k1,k2,dsmax);
rl = locus(:,2:size(locus)*[0;1]);
po = roots(a);
ze = roots(b);

if nargin<6
  Re_max = max([max(real(rl)) real(po') real(ze')]);
  Re_min = min([min(real(rl)) real(po') real(ze')]);
  Im_max = max([max(imag(rl)) imag(po') imag(ze')]);
  dRe = Re_max - Re_min;
  dIm = 2*Im_max;
  mRe = (Re_min + Re_max)/2;
  d = max(dRe,dIm);
  if d==0
    w = [-abs(Re_max) abs(Re_max)]*1.1;
  else
    w = [-d/2 d/2]*1.1;
  end
  w = [-d/2 d/2]*1.1;
  scale = [w+mRe w];
end

axis('square');
axis(scale);
plot(0,0,'i');
hold on;
plot(real(rl),imag(rl),'.');
mark(ze,4,[],[],1);
mark(po,2,[],[],1);
xlabel('Re');
ylabel('Im');
hold off;
axis('normal');

-------------------- /regler/matlab/rlbox/rloc2.m --------------------

function locus = rloc2(b,a,k1,knom,k2,dsmax,scale);
% RLOC2 Plots root locus around a nominal point.
%
%       locus = rloc2(b,a,k1,knom,k2,dsmax,scale)
%       locus = rloc2(b,a,k1,knom,k2,dsmax)
%
%       Plots the root locus of a(s) + k*b(s) for k1 < k < k2. The nominal
%       roots corresponding to knom are marked with a *.
%
%       The increments in k are chosen such that abs(ds) < dsmax.
%       The k-values and the roots are stored in locus, with the k-values
%       in the first column.
%
%       scale (optional) takes the form [xmax xmin ymin ymax]. If scale is
%       omitted an attempt is done to find a plot scale that gives an aspect
%       ratio as close to 1 as possible.

% Kjell Gustafsson, derived from routines by Mats Lilja
% LastEditDate : Thu Jun 14 17:34:33 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

locus = rootlocus(b,a,k1,k2,dsmax);
reldeg = length(a) - length(b);
if reldeg>=0,
```

```
  b = [0*(1:reldeg) b];
else
  a = [0*(1:-reldeg) a];
end
index = sum(locus(:,1)<knom);
locus = [locus(1:index,:); [knom roots(a+knom*b)']; ..
  locus(index+1:length(locus(:,1)),:)];
rl = locus(:,2:size(locus)*[0;1]);

if nargin<7
  Re_max = max(max(real(rl)));
  Re_min = min(min(real(rl)));
  Im_max = max(max(imag(rl)));
  dRe = Re_max - Re_min;
  dIm = 2*Im_max;
  mRe = (Re_min + Re_max)/2;
  d = max(dRe,dIm);
  if d==0
    w = [-abs(Re_max) abs(Re_max)]*1.1;
  else
    w = [-d/2 d/2]*1.1;
  end
  w = [-d/2 d/2]*1.1;
  scale = [w+mRe w];
end

axis('square');
axis(scale);
plot(0,0,'i');
hold on;
plot(real(rl),imag(rl),'.')
mark(rl(index+1,:),2,[],[],1);
xlabel('Re');
ylabel('Im');
hold off;
axis('normal');

-------------------- /regler/matlab/rlbox/rootlocus.m --------------------

function locus = rootlocus(b,a,k1,k2,dsmax,ucirc);
% ROOTLOCUS Calculates rootlocus
%
%       locus = rootlocus(a,b,k1,k2,dsmax,ucirc)
%       locus = rootlocus(a,b,k1,k2,dsmax)
%
%       Calculates the root locus of a(s) + k*b(s) for k1 < k < k2. The
%       increment in k is calculated such that abs(ds) < dsmax for the
%       fastest root branch. This is done using the implicit function
%       theorem.
%
%       The roots and the corresponding k-values are stored in locus, with
%       the gains in the first column.
%
%       All k (k1 < k < k2) giving rise to multiple poles are calculated
%       and and the corresponding roots are included in locus.
%
%       If ucirc (optional) is supplied (any value) only the roots inside
%       the unit circle will be forced to obey dsmax.

% Kjell Gustafsson, derived from routines by Mats Lilja
% LastEditDate : Wed Mar 14 22:23:59 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
```

```
% Lund Institute of Technology, Lund, SWEDEN

na = length(a);
nb = length(b);
d = na - nb;
if d>=0,
  b = [0*(1:d) b];
else
  a = [0*(1:-d) a];
end
da = polyder(a);
db = polyder(b);
k = k1;
epsi = 1e-10;
i = 1;
while k<=k2,
  kvec(i) = k;
  ri = roots(a+k*b)';
  rvec(i,:) = ri;
  if nargin==6
    ind = find(abs(ri)<=1);
    ri = ri(ind);
  end
  fs = polyval(da+k*db,ri);
  fk = polyval(b,ri);
  dk = dsmax*min(abs(fs./fk))+epsi;
  k =  k + dk;
  i =  i + 1;
end;
locus = [kvec' rvec];

% multiple roots

dab = conv(da,b);
adb = conv(a,db);
if nb>1,
  mpol = dab - adb;
else
  mpol = dab;
end;
multroots = roots(mpol);
bmult = polyval(b,multroots);
amult = polyval(a,multroots);
ind = find(bmult~=0);
kmult = -amult(ind)./bmult(ind);
mr = [];
for j=1:length(kmult),
  if abs(imag(kmult(j)))<1e-9,
    kj = real(kmult(j));
    if kj>k1 & kj<k2,
      index = sum(locus(:,1)<kj);
      locus = [locus(1:index,:); [kj roots(a+kj*b)'];..
        locus(index+1:length(locus(:,1)),:)];
    end;
  end;
end;

-------------------- /regler/matlab/rlbox/symloc.m --------------------

function locus = symloc(b,a,r1,r2,dsmax,scale)
% SYMLOC Plots symmetric root locus for continuous system
%
```

```
%       locus = symloc(b,a,r1,r2,dsmax,scale)
%       locus = symloc(b,a,r1,r2,dsmax)
%
%       Plots the stable part of the root locus corresponding to
%       a(s)*a(-s) + r*b(s)*b(-s) for r1 < r < r2. Also the roots of
%       a (r = Inf) and the roots of b (r = 0) are plotted.
%
%       The increments in r are calculated such that abs(ds) < dsmax.
%       The r-values and the roots are stored in locus, with the r-values
%       in the first column.
%
%       scale (optional) takes the form [xmax xmin ymin ymax]. If scale is
%       omitted an attempt is done to find a plot scale that gives an aspect
%       ratio as close to 1 as possible.
%
%       Use PZGRID to plot grid and real/imaginary axis.

% Kjell Gustafsson, original file due to Ulf Holmberg
% LastEditDate : Thu Jun 14 17:34:51 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

na = length(a);
nb = length(b);
aminus = a;
bminus = b;
if na>1,
  aminus(1,na-1:-2:1) = -a(1,na-1:-2:1);
end
if nb>1,
  bminus(1,nb-1:-2:1) = -b(1,nb-1:-2:1);
end

loc = rootlocus(conv(b,bminus),conv(a,aminus),r1,r2,dsmax);

% sort out stable roots

[n,m] = size(loc);
locus = [];
for k = 1:n,
  [imvec,imindex] = sort(imag(loc(k,2:m)));
  loc(k,2:m) = loc(k,imindex+1);
  zindex = find(real(loc(k,2:m))==0);
  index = find(real(loc(k,2:m))<0);
  locus = [locus; loc(k,1) loc(k,index+1) loc(k,zindex(1:2:length(zindex))+1)];
end

rl = locus(:,2:size(locus)*[0;1]);

po = roots(conv(a,aminus));
[tmp,ind] = sort(real(po));
po = po(ind(1:length(ind)/2));
ze = roots(conv(b,bminus));
[tmp,ind] = sort(real(ze));
ze = ze(ind(1:length(ind)/2));
if nargin<6
  Re_max = max([max(real(rl)) real(po') real(ze')]);
  Re_min = min([min(real(rl)) real(po') real(ze')]);
  Im_max = max([max(imag(rl)) imag(po') imag(ze')]);
  dRe = Re_max - Re_min;
  dIm = 2*Im_max;
  mRe = (Re_min + Re_max)/2;
```

```
  d = max(dRe,dIm);
  if d==0
    w = [-abs(Re_max) abs(Re_max)]*1.1;
  else
    w = [-d/2 d/2]*1.1;
  end
  w = [-d/2 d/2]*1.1;
  scale = [w+mRe w];
end

axis('square');
axis(scale);
plot(0,0,'i');
hold on;
plot(real(rl),imag(rl),'.');
mark(ze,4,[],[],1);
mark(po,2,[],[],1);
xlabel('Re');
ylabel('Im');
hold off;
axis('normal');
```

4. FRLSBOX

```
--------------------- /regler/matlab/frlsbox/bafit.m --------------------

function [b,a,the_error] = bafit(z,g,nb,na,weighting)
% BAFIT Fits a rational function to a given process frequency response.
%
%       [B,A,THE_ERROR] = BAFIT(Z,G,NB,NA,WEIGHTING)
%
%       Fits a rational transfer function B(s)/A(s) to the
%       complex frequency response G at the complex frequencies Z
%       using least squares. Deg A = NA and deg B = NB.
%       Weighting may be included (default is unity weighting).
%       The magnitude of the (weighted) closed loop transfer
%       function error is given by THE_ERROR.

% Mats Lilja
% LastEditDate : Thu Jul 19 11:08:02 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

nz = length(z);
ng = length(g);
if nz ~= ng,
  error('Different lengths of vectors Z and G');
end;
if nargin<5,
  weighting = ones(1,z);
end;
nweight = length(weighting);
if nz ~= nweight,
  error('Wrong number of weightings');
end;
i = sqrt(-1);
f = abs(weighting(:))';
if 2*nz < na+nb+1,
  error('Too few frequencies.');
end;
zj = ones(1,nz);
zb = zj;
for j=1:nb,
  zj = z.*zj;
  zb = [zj ; zb];
end;
zj = ones(1,nz);
za = [];
for j=1:na,
  za = [zj ; za];
  zj = z.*zj;
end;
zn = zj;
ga = diag(g.*f);
gb = diag(f);
m = [-za*ga ; zb*gb];
l = zn*ga;
mm = [imag(m) real(m)];
ll = [imag(l) real(l)];
x = ll/mm;
a = [1 x(1:na)];
b = x(na+1:na+nb+1);
the_error = abs(g - polyval(b,z)./polyval(a,z));
```

```
-------------------- /regler/matlab/frlsbox/frlsbox.m --------------------

% FRLSBOX -- A collection of routines for calculating transfer functions
%            and controllers by using least squares fitting in the
%            frequency domain. Some Hankel norm approximation procedures
%            are also included. Written by Mats Lilja.
%
% Least squares fitting
%
%   LSBAC    - Fitting a frequency response to a rational function B(s)/A(s)
%              (continuous time version).
%   LSBAD    - Fitting a frequency response to a rational function B(z)/A(z)
%              (discrete time version).
%   LSBATAU  - Fitting a frequency response to exp(-tau s)B(s)/A(s)
%   LSRSTC   - Calculation of a controller of type Ru = -Sy + Tr
%              by least squares fitting of a closed loop transfer function
%              (continuous time version).
%   LSRSTD   - Calculation of a controller of type Ru = -Sy + Tr
%              by least squares fitting of a closed loop transfer function
%              (discrete time version).
%   BAFIT    - Fitting a frequency response to a rational function.
%              Used by LSBAC and LSBAD.
%   RSTFIT   -  Calculation of a controller of type Ru = -Sy + Tr
%               by least squares fitting of a closed loop transfer function.
%               Used by RSTC and RSTD.
%
% Frequency response manipulation
%
%   FPICK     - Pick out points from a frequency response.
%   AMPCROSS  - Compute frequencies of amplitude level crossing.
%   PHACROSS  - Compute frequencies of phase level crossing.
%   BANDWIDTH - Compute the bandwidth from a frequency response.
%   ID2FR     - Convert an Ident. Toolbox frequency file to FRBOX format.
%
% Transfer function approximation
%
%   HANKELU  - Unweighted optimal Hankel norm approximation.
%   HANKELW  - Weighted optimal Hankel norm approximation.
%   PADEAPPR - Pade' approximation of exp(-tau s)B(s)/A(s).
%
% Miscellaneous
%
%   SYLVESTER - Compute a Sylvester matrix of two polynomials.
%   LEVCROSS  - Compute level crossings in a table.
%   LEADCOMP  - Compute a lead compensator.
%

% Mats Lilja
% LastEditDate : Fri Jul 20 10:19:44 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

-------------------- /regler/matlab/frlsbox/frlsdemo1.m --------------------

% Mats Lilja
% LastEditDate : Thu Jul 19 16:03:43 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

echo off
frbox
```

```
ppbox
clc
echo on

%   In this session, functions in FRLSBOX will be used
%   for approximation of the transfer function G(s)=1/(s+1)^8.

b = 1;
a = poly(-ones(1,8));

%   The first approximation method will be least squares
%   fitting of some points on the Nyquist curve to a rational
%   function of lower order.
%   Generate the frequency response of the system.

fr = frc(b,a,0,-2,2,100);

pause   % Strike RETURN to see the Nyquist plot of G

nypl(fr);
nygrid;

pause   % Strike RETURN to continue

clc
%   First, the frequencies for which the Nyquist curve is to be fitted
%   must be chosen. To see this, the axis intersection frequencies
%   of the Nyquist curve are computed.

waxes = table1([arg(fr(:,2)) fr(:,1)],-[90 180 270 360]),

pause   % Strike RETURN to continue

clc
%   Let's start with a second order approximation.
%   with relative degree one. This means that the
%   numerator polynomial is of degree one and the
%   denominator polynomial of degree 2, i.e. a
%   rational approximation of degree (1,2).
%   To see the effect of choosing too high approximation
%   frequencies the (-360) degree frequency 1 rad/s is included.
%   Try, for example, Omega={0.1 0.3 1}.

w = [0.1 0.3 1];
f = frc(b,a,0,w);
[bh,ah] = lsbac(f,1,2)

%   Notice that the approximation is unstable.
%   Calculate the frequency response of the approximation

frh = frc(bh,ah,0,-2,2,100);

pause   % Strike RETURN to see the Nyquist plots of both systems

nypl(fr,frh,w);
nygrid;pause;

%   The curve fit is rather bad.

pause;  % Strike RETURN to continue.

clc
```

```
%   Increase model order to 3 (a (2,3) approximation).

[bh,ah] = lsbac(f,2,3)

%   Calculate the frequency response

frh = frc(bh,ah,0,-2,2,100);

pause    % Strike RETURN to see the Nyquist plots

nypl(fr,frh,w);
nygrid;

%   Slightly better ...

pause    % Strike RETURN to continue

clc
%   Another way to get a better approximation
%   is to decrease the approximation frequencies.
%   Choose, for example Omega = {0.1 0.2 0.4} and return to
%   approximation by a second order transfer function.
%   Notice that 0.4 is near the (-180) degree frequency.

w = [0.1 0.2 0.4];
f = frc(b,a,0,w);
[bh,ah] = lsbac(f,1,2)
frh = frc(bh,ah,0,-2,2,100);

pause    % Strike RETURN to see the Nyquist plots

nypl(fr,frh,w);
nygrid;

pause    % Strike RETURN to continue

%   The error is rather large at the lowest frequency
%   Introducing the frequency weighting {1 0.3 0.1} gives:

[bh,ah] = lsbac(f,1,2,[1 0.3 0.1])
frh = frc(bh,ah,0,-2,2,100);

pause    % Strike RETURN to see the new Nyquist curve

nysh(frh,w,'-.');

pause    % Strike RETURN to continue

clc
%   Increasing approximation model order to three gives:

[bh,ah] = lsbac(f,2,3)
frh = frc(bh,ah,0,-2,2,100);

pause    % Strike RETURN to see the Nyquist plots

nypl(fr,frh,w);
nygrid; pause

%   The Nyquist curves fit very well together.
%   To get a better view of how well, the absolute error is plotted
```

```
pause   % Strike RETURN to see the approximation error magnitude

ampl(fsub(fr,frh));
amgrid;

pause   % Strike RETURN to continue

clc
%   The error magnitude has a peak around 1 rad/s
%   An approximation method which gives a much more
%   uniform error magnitude is optimal Hankel norm
%   approximation. A third order (unweighted)
%   approximation is computed and the error magnitude
%   curve is displayed.

[bh,ah] = hankelu(b,a,3)

frh = frc(bh,ah,0,-2,2,100);

pause   % Strike RETURN to see the error magnitude

amsh(fsub(fr,frh),'--'); pause;

%   It is also possible to shape the error by using
%   frequency weighted Hankel norm approximation.
%   Assume that the error should be approximately 4 times
%   smaller at frequencies below 1 rad/s than at frequencies
%   above 1 rad/s. The weighting is then chosen as (s - 2)/(s - 0.5).

[bh,ah] = hankelw(b,a,3,[1 -2],[1 -0.5])
pause;
frh = frc(bh,ah,0,-2,2,100);

pause   % Strike RETURN to see the error magnitude

amsh(fsub(fr,frh),'--');

pause   % Strike RETURN to continue

clc
%   A classical approximation method is Pade' approximation.
%   In this method a rational function Gh(s) of degree (m,n) is
%   calculated such that G(s) - Gh(s) = O(s^(m+n+1)) which
%   means that n+m first taylor coefficient of G(s) and Gh(s)
%   coincide. Let's compute a Pade' approximation of degree (2,3).

[bh,ah] = padeappr(b,a,0,2,3)
frh = frc(bh,ah,0,-2,2,100);

pause   % Strike RETURN to see the error magnitude

ampl(fsub(fr,frh));

%   Notice the small error at low frequecncies.

pause;  % Strike RETURN to continue

clc
%   It is possible to include a time delay in the approximate model.
%   This is done by using a non-linear least squares method.
%   A second order system with time delay is fitted to G(s)
%   at the frequencies Omega = {0.1 0.3 1} with unity weighting (default).
```

```
w = [0.1 0.3 1]
f = frc(b,a,0,w);
[bh,ah,tauh] = lsbatau(f,1,2)

%   Unfortunately, this gives a negative time delay.
%   The Nyquist curve fitting becomes very strange.

frh = frc(bh,ah,tauh,-2,2,100);

pause;  % Strike RETURN to see Nyquist plots

nypl(fr,frh,w);
nygrid;

pause;  % Strike RETURN to continue

clc
%   One remedy this is to decrease weighting of
%   higher frequencies. Weightings are chosen as {1 0.5 0.1}.

[bh,ah,tauh] = lsbatau(f,1,2,[1 0.5 0.1])

%   A positive time delay is obtained.

frh = frc(bh,ah,tauh,-2,2,100);

pause;  % Strike RETURN to see Nyquist plots

nypl(fr,frh,w);
nygrid;

%   Alternatively, the start value of the time delay (default=0)
%   can be slightly increased. Choose the value 0.1, for example:

[bh,ah,tauh] = lsbatau(f,1,2,[1 1 1],0.1)
frh = frc(bh,ah,tauh,-2,2,100);

pause;  % Strike RETURN to see new Nyquist plot

nysh(frh,w,':');

%   Notice the bad fitting at low frequencies due to the unity
%   weighting {1 1 1} used in this case.

echo off;

-------------------- /regler/matlab/frlsbox/frlsdemo2.m --------------------

% Mats Lilja
% LastEditDate : Thu Jul 19 19:29:56 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

echo off;
frbox;
ppbox;
clc;
echo on;

%   This session demonstrates methods for designing controllers
%   given some points on the Nyquist curve of a system.
```

```
%   In this example the system has the transfer function
%   G(s)=1/(s+1)^8.

b = 1;
a = poly(-ones(1,8));

pause   % Strike RETURN to continue

clc
%   First, an indirect method will be used. The plant transfer
%   function is first approximated by a low order model. A controller
%   based on ordinary pole placement design of the low order plant
%   model is then computed. A third order model should be suitable.
%   Choose approximation frequencies Omega = {0.01 0.2 0.4}:

w = [0.01 0.2 0.4];
f = frc(b,a,0,w);
[bh,ah] = lsbac(f,2,3)

pause   % Strike RETURN to continue

clc
%   A second order controller will be computed by solving the polynomial
%   pole placement (DAB) equation AR + BS = A_m A_o. In order to choose
%   poles of closed loop system (zeros of A_m and A_o), check the poles
%   of the approximate model:

roots(ah)

pause   % Strike RETURN to continue

clc
%   Make the closed loop model poles slightly faster. Take for example

am = polybutt(3,0.5)
ao = polybutt(2,1)
[r,s,t] = rstc(1,bh,ah,1,am,ao,1)

pause   % Strike RETURN to continue

clc
%   Check stability of the actual closed loop:

clpoles = roots(addpoly(conv(a,r),conv(b,s)))

pause   % Strike RETURN to continue

%   Check Nyquist curves of the specified and actual closed loop
%   transfer function (ref --> y):

frcl = frc(conv(b,t),addpoly(conv(a,r),conv(b,s)),0,-2,2,100);
bm = b/b(length(b))*am(length(am));
frm = frc(bm,am,0,-2,2,100);

pause   % Strike RETURN to see closed loop Nyquist curves

nypl(frcl,frm,w);
nygrid;

%   Check closed loop performance:

tryu = yusimc(b,a,r,s,t,160);
```

```
pause;  % Strike RETURN to view the closed loop time response

yupl(tryu);

pause;  % Strike RETURN to continue

clc
%   Introduce integration in the controller. The degree of the
%   polynomial A_m A_o must be increased. Choose to increase deg A_o:

ao = polybutt(3,1)
[r,s,t] = rstc(1,bh,ah,1,am,ao,[1 0])

%   The controller is in this case unstable.

pause;  % Strike RETURN to continue

clc
%   Check closed loop stability:

clpoles = roots(addpoly(conv(a,r),conv(b,s)))

%   The actual closed loop is unstable!

pause;  % Strike RETURN to continue

clc
%   Respecify the performance requirements!

am = polybutt(3,0.4);
ao = polybutt(3,0.7);
[r,s,t] = rstc(1,bh,ah,1,am,ao,[1 0])
pause;

%   Is the system closed loop stable?

clpoles = roots(addpoly(conv(a,r),conv(b,s)))

pause;  % Strike RETURN to continue

clc
%   Check closed loop Nyquist curves.

frcl = frc(conv(b,t),addpoly(conv(a,r),conv(b,s)),0,-2,2,100);
bm = bh/bh(length(bh))*am(length(am)); % Stationary gain G_m(0)=1
frm = frc(bm,am,0,-2,2,100);

pause;  % Strike RETURN to see the Nyquist curves

nypl(frcl,frm,w);
nygrid;

%   Closed loop performance?

tryu = yusimc(b,a,r,s,t,160);

pause;  % Strike RETURN to have a look at the closed loop response

yupl(tryu);

%   To check robustness the open loop Nyquist curve is plotted.
```

```
frl = frc(conv(b,s),conv(a,r),0,-2,2,100);

pause;  % Strike RETURN to see the Nyquist curve of the loop.

nypl(frl,w);
nygrid;

pause;  % Strike RETURN to continue

clc
%   Using the indirect design method sketched above, the computation
%   of a second order controller with integration requires a second
%   order process model.

[bh,ah] = lsbac(f,1,2,[1 0.3 0.1])

pause;

%   Specify nominal closed loop poles in butterworth patterns:

am = polybutt(2,0.3);
ao = polybutt(2,0.5);
[r,s,t] = rstc(1,bh,ah,1,am,ao,[1 0])

pause;  % Strike RETURN to continue

clc
%   Check closed loop Nyquist curves.

frcl = frc(conv(b,t),addpoly(conv(a,r),conv(b,s)),0,-2,2,100);
bm = bh/bh(length(bh))*am(length(am));
frm = frc(bm,am,0,-2,2,100);

pause;  % Strike RETURN to see closed loop Nyquist curves

nypl(frcl,frm,w);
nygrid;

%   The curve fitting is bad at higher frequencies.

pause;  % Strike RETURN to continue

clc
%   Check the closed loop performance.

tryu = yusimc(b,a,r,s,t,160);

pause;  % Strike RETURN to view the closed loop time response

yupl(tryu);

%   The performance is not very impressive.

pause;  % Strike RETURN to continue

clc
%   In the direct method the controller parameters are computed
%   directly from frequency response data without solving
%   any polynomial equation.
%   This makes it possible to use a third order closed loop
%   specification when computing a second order controller
```

```
%   with integration. A third order process model is first
%   computed.

[bh,ah] = lsbac(f,2,3)

pause;  % Strike RETURN to continue

clc
%   The numerator polynomial from this transfer function is
%   used in the third order specification.

am = polybutt(3,0.4);
bm = bh/bh(length(bh))*am(length(am));
[r,s,t] = lsrstc(f,bm,am,ao,2,2,[1 0])

pause;  % Strike RETURN to continue

clc
%   Check closed loop Nyquist curves.

frcl = frc(conv(b,t),addpoly(conv(a,r),conv(b,s)),0,-2,2,100);
frm = frc(bm,am,0,-2,2,100);

pause;  % Strike RETURN to see Nyquist curves

nypl(frcl,frm,w);
nygrid;

%   The curve fitting is better in this case.

pause;  % Strike RETURN to contimue

%   What about the closed loop performance?

tryu = yusimc(b,a,r,s,t,160);
yupl(tryu);

pause;

%   The performance is better than in the indirect method.
echo off


-------------------- /regler/matlab/frlsbox/hankelu.m --------------------

function [Bhat,Ahat,sigm,signature] = hankelu(B,A,nred)
% HANKELU Compute a unweighted Hankel norm approximation
%         of a rational transfer function
%
%         [BHAT,AHAT,SIGM] = HANKELU(B,A,NRED)
%
%         An unweighted optimal Hankel norm approximation BHAT(s)/AHAT(s)
%         of order NRED is computed for B(s)/A(s). The vector SIGM contains
%         the Hankel singular values.

% Mats Lilja
% LastEditDate : Tue Mar 13 17:56:46 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

na = length(A);
nb = length(B);
```

```
m1 = sylvester(A,-B,na-1,na-1);
m2 = diag(1-rem(1:2*na-2,2)*2)*sylvester(0,A,na-1,na-1);
[alfa,beta,q,z,v] = qz(m1,m2);
alfa = diag(alfa);
beta = diag(beta);
nn = na:2*na-2;
d = alfa(nn)./beta(nn);
v = v(:,nn);
[dd,index] = sort(-abs(d));
d = d(index);
v = v(:,index);
sigm = abs(d);
signature = d./sigm;
v = real(v/diag(v(na,:)));
bh = v(1:na-1,nred+1)';
ah = v(na:2*na-2,nred+1)';
[bh,ah] = stabpart(bh,ah);
% This gives the unique strictly proper optimal Hankel norm approximation.
% Next we choose a direct term so that the error magnitude at omega=infinity
% equals the error magnitude at omega=0. This is a heuristic attempt to
% minimize the Tchebycheff norm of the error.
G0 = B(nb)/A(na);
if nb < na,
  Ginf = 0;
else
  Ginf = B(1)/A(1);
end;
Gh0 = bh(length(bh))/ah(length(ah));
Ahat = ah;
direct_term = (Ginf + G0 - Gh0)/2;
Bhat = [0 bh] + ah*direct_term;

-------------------- /regler/matlab/frlsbox/hankelw.m --------------------

function [Bhat,Ahat,sigm] = hankelw(B,A,nred,Bw,Aw,toler)
% HANKELW Compute a weighted Hankel norm approximation
%         of a rational transfer function
%
%         [BHAT,AHAT] = HANKELW(B,A,NRED,BW,AW)
%
%         A weighted optimal Hankel norm approximation BHAT(s)/AHAT(s)
%         of order NRED is computed for B(s)/A(s). The weighting function
%         BW(s)/AW(s) must have all poles and zeros in the open right half
%         plane and degrees of the polynomials AW and BW must be the same.

% Mats Lilja
% LastEditDate : Tue Mar 13 18:04:57 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin < 6,
  toler = 1e-12;
end;
Aw = Aw/Aw(1);
Bw = Bw/Bw(1);
aaw = conv(A,Aw);
bbw = conv(B,Bw);
[Bpol,Apol] = stabpart(bbw,aaw);
na = length(Apol);
nb = length(Bpol);
nn1 = 1:(na-1);
nn2 = na:(2*na-2);
```

```
m1 = sylvester(Apol,-Bpol,na-1,na-1);
m2 = -diag(1-rem(1:2*na-2,2)*2)*sylvester(0,Apol,na-1,na-1);
[alfa,beta,q,z,v] = qz(m1,m2);
alfa = diag(alfa);
beta = diag(beta);
d = alfa(nn2)./beta(nn2);
v = v(:,nn2);
[dd,ii] = sort(-abs(d));
d = d(ii);
v = v(:,ii);
v = real(v/diag(v(na,:)));
sigm = abs(d);
signature = d./sigm;
ab = real(v(:,nred+1))';
ab = ab/ab(na);
bb = conv(ab(nn1),Aw);
aa = conv(ab(nn2),Bw);
[bb,aa] = stabpart(bb,aa);
[q,r] = deconv(aa,Bw);
if max(abs(r))<1e-12;
  aa = q;
  bb = dab(Bw,aa,bb);
end;
ah = aa;
[qq,bh] = deconv(bb,aa);
bh(1:min(find(abs(bh)>eps))-1) = [];
Ahat = ah;
Bhat = bh;
G0 = B(length(B))/A(length(A));
if length(B) < length(A),
  Ginf = 0;
else
  Ginf = B(1)/A(1);
end;
Gh0 = bh(length(bh))/ah(length(ah));
if length(bh) < length(ah),
  Ghinf = 0;
else
  Ghinf = bh(1)/ah(1);
end;
Gw0 = Bw(length(Bw))/Aw(length(Aw));
if length(Bw) < length(Aw),
  Gwinf = 0;
else
  Gwinf = Bw(1)/Aw(1);
end;
heur_direct = (Ginf*Gwinf + Gw0*(G0 - Gh0))/(Gwinf + Gw0);
Bhat = addpoly(Bhat,heur_direct*Ahat);

-------------------- /regler/matlab/frlsbox/leadcomp.m --------------------

function [s,r,wc,phi_m] = leadcomp(fr,n_speed,maxlead,zeta)
% LEADCOMP Compute a lead compensator from a frequency response
%
%       [S,R] = LEADCOMP(FR,N_SPEED)
%
%       Given a frequency response FR, a lead compensator
%       S(s)/R(s) is computed which increases the cut frequency by a
%       factor of N_SPEED without changing the phase margin.
%       The compensator consists of cascaded identical first order
%       compensators, where the number of compensators is determined
%       by the amount of phase lead required.
```

```
%
%       [S,R] = LEADCOMP(FR,N_SPEED,MAXLEAD)
%
%       limits the maximum phase lead for each first order compensator
%       to be MAXLEAD (default value = 90 degrees).
%
%       [S,R] = LEADCOMP(FR,N_SPEED,MAXLEAD,ZETA)
%
%       gives a number of identical second order compensators,
%       each with a relative damping ZETA.
%
%       [S,R,OMEGA_C,PHI_M] = LEADCOMP(FR,N_SPEED)
%
%       gives the (old) cut frequency OMEGA_C and the phase margin PHI_M.

% Mats Lilja
% LastEditDate : Thu Mar 15 17:08:46 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin < 3,
  maxlead = 90;
end;
if nargin < 4,
  zeta = 1;
end;
w = fr(:,1);
g = fr(:,2);
[wc,phi1] = ampcross(fr,1);
phi_m = 180 + phi1;
wc2 = wc*n_speed;
gwc2 = table1([w abs(g) arg(g)],wc2);
amp2 = gwc2(1);
phi2 = gwc2(2);
dphi = phi1 - phi2;
if zeta == 1,
  order = ceil(dphi/maxlead);
  th = pi*(dphi/order+90)/360;
  N = (tan(th))^2;
  b = wc2/sqrt(N);
  r = poly(-N*b*ones(1,order));
  s = poly(-b*ones(1,order))*(sqrt(N))^order/amp2;
else,
  order2 = ceil(dphi/maxlead/2);
  th = pi*(dphi/order2)/360;
  alfa = zeta*tan(th);
  N = (alfa+sqrt(1+alfa^2))^2;
  b = wc2/sqrt(N);
  one2 = ones(1,order2);
  r = polyc(N*b*one2,zeta*one2);
  s = polyc(b*one2,zeta*one2)*(N^order2)/amp2;
end;

-------------------- /regler/matlab/frlsbox/lsbac.m --------------------

function [b,a,the_error] = lsbac(fr,nb,na,weighting)
% LSBAC Fits a rational function to a given continous time
%       process frequency response.
%
%       [B,A,THE_ERROR] = LSBAC(FR,NB,NA,WEIGHTING)
%
%       Fits a rational transfer function B(s)/A(s) to the
```

```
%       continuous time frequency response FR using least squares.
%       Deg A = NA and deg B = NB. Weighting may be included (default is
%       unity weighting). The magnitude of the (weighted) closed
%       loop transfer function error is given by THE_ERROR.

% Mats Lilja
% LastEditDate : Tue Jun 12 19:54:44 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

nfr = size(fr)*[1;0];
if nargin<4,
  weighting = ones(1,nfr);
end;
nweight = length(weighting);
if nfr ~= nweight,
  error('Wrong number of weightings');
end;
i = sqrt(-1);
w = fr(:,1).';
z = i*w;
g = fr(:,2).';
[b,a,the_error] = bafit(z,g,nb,na,weighting);

-------------------- /regler/matlab/frlsbox/lsbad.m --------------------

function [b,a,the_error] = lsbad(fr,nb,na,weighting)
% LSBAD Fits a rational function to a given discrete time
%       process frequency response.
%
%       [B,A,THE_ERROR] = LSBAD(FR,NB,NA,WEIGHTING)
%
%       Fits a rational transfer function B(s)/A(s) to the
%       discrete time frequency response FR using least squares.
%       Deg A = NA and deg B = NB. Weighting may be included (default is
%       unity weighting). The magnitude of the (weighted) closed
%       loop transfer function error is given by THE_ERROR.

% Mats Lilja
% LastEditDate : Tue Jun 12 19:54:31 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

nfr = size(fr)*[1;0];
if nargin<4,
  weighting = ones(1,nfr);
end;
nweight = length(weighting);
if nfr ~= nweight,
  error('Wrong number of weightings');
end;
i = sqrt(-1);
w = fr(:,1).';
z = exp(i*w);
g = fr(:,2).';
[b,a,the_error] = bafit(z,g,nb,na,weighting);

-------------------- /regler/matlab/frlsbox/lsbatau.m --------------------

function [b,a,tau,the_error,logging] = lsbatau(fr,nb,na,weighting,tau0,dtaumax)
% LSBATAU Least squares fitting of a rational function with
%       a time delay.
```

```
%
%       [B,A,TAU,THE_ERROR,LOGGING] = LSBATAU(FR,NB,NA,WEIGHTING,TAU0,DTAUMAX)
%
%       Computes a least-squares fitting
%
%                      - TAU s  B(s)
%            G(s) = e           ----
%                               A(s)
%
%       to the frequency response data given in FR.
%       where the degrees of the polynomials A and B are NA and NB
%       respectively. The time delay TAU is found by using a
%       modified Newton-Raphson algorithm. Weighting may be included
%       (default is unity weighting). DTAUMAX is the iteration termination
%       treshold value for the magnitude of the time delay increment
%       (default value = 0.01). TAU0 is the initial value of TAU
%       (default value = 0). The magnitude of the (weighted) closed
%       loop transfer function error is given by THE_ERROR.
%       The history of the iterations are saved in LOGGING.

% Mats Lilja
% LastEditDate : Tue Mar 20 15:02:56 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

nfr = size(fr)*[1;0];
if nargin < 4,
  weighting = ones(1,nfr);
end;
if nargin < 5,
  tau0 = 0;
end;
if nargin < 6,
  dtaumax = 0.01;
end;
nweight = length(weighting);
if nfr ~= nweight,
  error('Wrong number of weightings');
end;
i = sqrt(-1);
w = fr(:,1).';
zvec = i*w;
gvec = fr(:,2).';
pvec = abs(weighting(:))';
if 2*nfr < na+nb+2,
  error('Too few frequencies.');
end;
nn = 2*nfr;
en = eye(nn);
s = conj([zvec conj(zvec)])';
pw = pvec/max(pvec);
filt = diag([pw pw]);
g = diag([gvec conj(gvec)]);
zj = ones(nn,1);
psib = zj;
for j=1:nb;
  zj = s.*zj;
  psib = [zj psib];
end;
zj = ones(nn,1);
psia = zj;
for j=1:(na-1);
```

```
  zj = s.*zj;
  psia = [zj psia];
end;
zn = s.*zj;
gam = filt*g*zn;
tau = tau0;
dtau = 10000;
logg = [];
my = 1;
while abs(dtau)>dtaumax,
  count = count + 1;
  ew = diag(exp(-s*tau));
  fi = filt*[-g*psia ew*psib];
  [u,sig,v] = svd(fi);
  u1 = u(:,1:min(size(sig)));
  sig = sig(1:min(size(sig)),:);
  fiff = (v/sig)*u1';
  dew = -diag(s)*ew;
  dfi = filt*[ 0*psia dew*psib];
  d2ew = -diag(s)*dew;
  d2fi = filt*[ 0*psia d2ew*psib];
  p = en - fi*fiff;
  p1 = dfi*fiff;
  p2 = d2fi*fiff;
  q =  -p*p1;
  dp = q + q';
  dq = p*( 2*p1*p1 - p2 ) + q'*q + q*q';
  d2p = dq + dq';
  gpg = gam'*p*gam;
  gdpg = gam'*dp*gam;
  gd2pg = gam'*d2p*gam;
  logg = [logg ; [tau gpg gdpg gd2pg]];
  dtau = -real(gdpg/(0.6*abs(gd2pg)+0.4*gd2pg));
  tau = tau + dtau;
  dtau;
  tau;
end;
ew = diag(exp(-s*tau));
fi = filt*[-g*psia ew*psib];
[u,s,v] = svd(fi);
sig = s(1:min(size(s)),:);
u1 = u(:,1:min(size(s)));
fiff = (v/sig)*u1';
theta = real(fiff*gam);
logging = real(logg);
a = [1 theta(1:na)'];
b = theta(na+1:na+nb+1)';
tau = real(tau);
the_error = abs(gvec - exp(-tau*zvec).*polyval(b,zvec)./polyval(a,zvec));

--------------------- /regler/matlab/frlsbox/lsrstc.m --------------------

function [r,s,t,the_error]=lsrstc(fr,bm,am,ao,nr,ns,r1,s1,weighting)
% LSRSTC Fits a continuous time controller to a specified closed
%       loop model given a frequency response of the process.
%
%       [R,S,T,THE_ERROR]=LSRSTC(FR,BM,AM,AO,NR,NS,R1,S1,WEIGHTING)
%
%       The frequency response FR on the form [w G(iw)]
%       is used for least squares fitting of controller parameters
%       in a controller structure given by Ru = Tr - Sy.
%       The closed loop system (r --> y) is specified
```

```
%       by the transfer function BM(s)/AM(s). For convenience, the
%       polynomial BM is multiplied by a constant factor in order to
%       get a closed loop stationary gain of 1 (i.e. BM(0)/AM(0)=1).
%       Factors to be included in R are specified by the polynomial R1.
%       Similarly, S1 pre-specifies factors of S.
%       Deg R = NR and deg S = NS. The observer polynomial is given by
%       AO (T = const.*AO). Default WEIGHTING is unity weighting
%       while R1 and S1 both defaults to 1.
%       The magnitude of the (weighted) closed loop transfer
%       function error is given by THE_ERROR.

% Mats Lilja
% LastEditDate : Thu Jul 19 18:20:48 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

nfr = size(fr)*[1;0];
if nargin < 9,
  weighting = ones(1,nfr);
end;
if nargin < 8,
  s1 = 1;
end;
if nargin < 7,
  r1 = 1;
end;
i = sqrt(-1);
w = fr(:,1).';
z = i*w;
g = fr(:,2).';
% To get staionary gain = 1 from r to y
bm = bm/bm(length(bm)).*am(length(am));
gm = polyval(bm,z)./polyval(am,z);
[r,s,t,the_error]=rstfit(z,g,gm,ao,nr,ns,r1,s1,weighting);

-------------------- /regler/matlab/frlsbox/lsrstd.m --------------------

function [r,s,t,the_error]=lsrstd(fr,bm,am,ao,nr,ns,r1,s1,weighting)
% LSRSTD Fits a discrete time controller to a specified closed
%       loop model given a frequency response of the process.
%
%       [R,S,T,THE_ERROR]=LSRSTD(FR,BM,AM,AO,NR,NS,R1,S1,WEIGHTING)
%
%       The frequency response FR on the form [w H(exp(iw))]
%       is used for least squares fitting of controller parameters
%       in a controller structure given by Ru = Tr - Sy.
%       The closed loop system (r --> y) is specified
%       by the transfer function BM(z)/AM(z). For convenience, the
%       polynomial BM is multiplied by a constant factor in order to
%       get a closed loop stationary gain of 1 (i.e. BM(1)/AM(1)=1).
%       Factors to be included in R are specified by the polynomial R1.
%       Similarly, S1 pre-specifies factors of S.
%       Deg R = NR and deg S = NS. The observer polynomial is given by
%       AO (T = const.*AO). Default WEIGHTING is unity weighting
%       while R1 and S1 both defaults to 1.
%       The magnitude of the (weighted) closed loop transfer
%       function error is given by THE_ERROR.

% Mats Lilja
% LastEditDate : Thu Jul 19 18:21:12 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN
```

```
nfr = size(fr)*[1;0];
if nargin < 9,
  weighting = ones(1,nfr);
end;
if nargin < 8,
  s1 = 1;
end;
if nargin < 7,
  r1 = 1;
end;
i = sqrt(-1);
w = fr(:,1).';
z = exp(i*w);
g = fr(:,2).';
% To get staionary gain = 1 from r to y
bm = bm/sum(bm)*sum(am);
gm = polyval(bm,z)./polyval(am,z);
[r,s,t,the_error]=rstfit(z,g,gm,ao,nr,ns,r1,s1,weighting);
```

-------------------- /regler/matlab/frlsbox/padeappr.m --------------------

```
function [bh,ah] = padeappr(b,a,tau,nbh,nah)
% PADEAPPR Compute a Pade' approximation of (rational function)*exponential
%
%        [BH,AH] = PADEAPPR(B,A,TAU,NBH,NAH)
%
%        Calculates a Pade' approximation BH(s)/AH(s) with
%        deg BH = NBH and deg AH = NAH of the system
%
%           -TAU s  B(s)
%          e        ------
%                   A(s)

% Mats Lilja
% LastEditDate : Mon Mar 19 15:44:02 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

exppol = 1;
taylor_term = 1;
for j=1:(nah+nbh),
  taylor_term = taylor_term*tau/j;
  exppol = [taylor_term exppol];
end;
e = conv(a,exppol);
m = sylvester(b,-e,nah+1,nbh+1);
m(1:nbh,:) = [];
l = -m(:,1);
m(:,1) = [];
x = (m\l)';
ah = [1 x(1:nah)];
bh = x(nah+1:nah+nbh+1);
```

-------------------- /regler/matlab/frlsbox/rstfit.m --------------------

```
function [r,s,t,the_error]=rstfit(z,g,gm,ao,nr,ns,r1,s1,weighting)
% RSTFIT Fits a controller to a specified closed loop model
%        given a frequency response of the process.
%
%        [R,S,T,THE_ERROR]=RSTFIT(Z,G,GM,AO,NR,NS,R1,S1,WEIGHTING)
```

```
%
%       The complex frequency response G, given at the
%       complex frequencies Z, is used for least squares fitting
%       of controller parameters in the controller structure
%       Ru = Tr - Sy. The closed loop system (r --> y) is specified
%       by the frequency response GM, given at the frequencies Z.
%       Factors to be included in R are specified by the polynomial R1.
%       Similarly, S1 pre-specifies factors of S.
%       Deg R = NR and deg S = NS. The observer polynomial is given by
%       AO (T = const.*AO). Default WEIGHTING is unity weighting
%       while R1 and S1 both defaults to 1.
%       The magnitude of the (weighted) closed loop transfer
%       function error is given by THE_ERROR.

% Mats Lilja
% LastEditDate : Fri Jul 20 10:13:11 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

nz = length(z);
ng = length(g);
if nz ~= ng,
  error('Different lengths of vectors Z and G');
end;
if nargin < 9,
  weighting = ones(1,nz);
end;
if nargin < 8,
  s1 = 1;
end;
if nargin < 7,
  r1 = 1;
end;
nweight = length(weighting);
if nz ~= nweight,
  error('Wrong number of weightings');
end;
nr2 = nr - length(r1) + 1;
if nr2 < 0,
  error('Degree of R1 higher than degree of R');
end;
ns2 = ns - length(s1) + 1;
if ns2 < 0,
  error('Degree of S1 higher than degree of S');
end;
i = sqrt(-1);
f = abs(weighting);
Npoints = 2*nz;
Nparameters = nr2 + ns2 + 2;
%%%%
if Npoints < Nparameters
  error('Too few approximation points.');
end;
aovec = polyval(ao,z);
gr = diag(f.*polyval(r1,z).*gm./aovec./g);
gs = diag(f.*polyval(s1,z).*gm./aovec);
gt = diag(f);
one = ones(1,nz);
mt = one;
ms = one;
zs = one;
for j=1:ns2,
```

```
  zs = z.*zs;
  ms = [zs ; ms];
end;
mr = [];
zr = one;
for j=1:nr2,
  mr = [zr ; mr];
  zr = z.*zr;
end;
l = zr*gr;
m = [-mr*gr ; -ms*gs ; mt*gt];
mm = [real(m) imag(m)];
ll = [real(l) imag(l)];
x = ll/mm;
r2 = [1 x(1:nr2)];
r = conv(r1,r2);
s2 = x(nr2+1:nr2+ns2+1);
s = conv(s1,s2);
t = x(nr2+ns2+2)*ao;
num = g.*polyval(t,z);
den = polyval(r,z) + g.*polyval(s,z);
the_error = abs(num./den - gm);

-------------------- /regler/matlab/frlsbox/sylvester.m --------------------

function [m] = sylvester(a,b,nx,ny)
% SYLVESTER Computes a Sylvester matrix of two polynomials
%
%       M = SYLVESTER(A,B,N1,N2)
%
%       The Sylvester matrix of order (N1,N2) is computed
%       for the polynomials A and B.

% Mats Lilja
% LastEditDate : Tue Mar 13 14:29:55 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

na = length(a);
nb = length(b);
m = na - nb - ny + nx;
a = [zeros(1,-m) a];
b = [zeros(1,m) b];
za = zeros(1,nx-1);
zb = zeros(1,ny-1);
ma = toeplitz([a za],[a(1) za]);
mb = toeplitz([b zb],[b(1) zb]);
m = [ma mb];
```

5. LQGBOX

```
-------------------- /regler/matlab/lqgbox/care.m --------------------

function S = care(A,B,Q1,Q2,Q12)
% CARE  dispatch routine for continuous-time algebraic Riccati equation solver
%
%       S = care(A,B,Q1,Q2,Q12)
%       S = care(A,B,Q1,Q2)
%
%       Depending on the value of the global variable caretype the
%       corresponding solver is called to solve the continuous-time
%       algebraic equation
%
%          S A + A' S - (S B + Q12) Q2^(-1) (Q12' + B' S) + Q1 = 0
%
%       Possible choices for caretype are: 'eigen', 'schur', and 'gener'.
%       If caretype is undefined it is taken as 'schur'.
%
%       Q12 is optional. If omitted it is taken as the empty matrix.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:19:44 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin<5,
  Q12 = zeros(size(A)*[1;0],size(B)*[0,1]);
end

if ~exist('caretype')==1
  caretype = 'schur';
end

if caretype=='eigen'
  % Direct calculation of the eigenvalues to the Hamiltonian
  if Q12==0
    S = careeig(A,B/Q2*B',Q1);
  else
    S = careeig(A-B/Q2*Q12',B/Q2*B',Q1-Q12/Q2*Q12');
  end
elseif caretype =='schur'
  % Use Schur form to calculate S
  if Q12==0
    S = careschur(A,B/Q2*B',Q1);
  else
    S = careschur(A-B/Q2*Q12',B/Q2*B',Q1-Q12/Q2*Q12');
  end
elseif caretype =='gener'
  % CTRL-C algorithm using QZ
  error('Generalized Schur form Riccati solver not yet implemented');
else
  error('Riccati solver in caretype undefined');
end

% residual check

Serr = S*A + A'*S - (S*B + Q12)/Q2*(Q12' + B'*S) + Q1;
disp('');
disp(['1-norm of the Riccati solution residual is ' num2str(norm(Serr,1)) ])
disp('');
```

-------------------- /regler/matlab/lqgbox/careeig.m --------------------

```
function X = careeig(F,G,H)
%CAREEIG continuous-time algebraic Riccati equation solver using direct
%       eigenvalue calculation
%
%       X = careeig(F, G, H)
%
%       returns the stablizing solution (if it exists) to the continuous-time
%       Riccati equation:
%
%                 X F + X' F - X G X + H = 0
%
%       assuming G is symmetric and nonnegative definite and H is
%       symmetric.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:20:02 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

ham = [F -G; -H -F'];

[n,m] = size(F);
[v,d] = eig(ham);
d = diag(d);
[d,index] = sort(real(d));
if (~( (d(n)<0) & (d(n+1)>0) ))
  error('Can''t order eigenvalues, System may be uncontrollable/unobservable.')
end

% select vectors with eigenvalues in the left half plane
x1 = v(1:n,index(1:n));
x2 = v((n+1):(2*n),index(1:n));
X = real(x2/x1);
```

-------------------- /regler/matlab/lqgbox/careschur.m --------------------

```
function X = careschur(F,G,H)
%CARESCHUR continuous-time algebraic Riccati equation solver using Schur form
%
%       X = careschur(F, G, H)
%
%       returns the stablizing solution (if it exists) to the continuous-time
%       Riccati equation:
%
%                 X F + F' X - X G X + H = 0
%
%       assuming G is symmetric and nonnegative definite and H is
%       symmetric.

% This is nothing but the routine are.m in the control systems toolbox. I have
% changed the name and the documentation to make it fit my routines.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:20:21 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

%  -- check for correct input problem --
[nr,nc] = size(F); n = nr;
```

```
if (nr ~= nc), error('Nonsquare F matrix'), end;
[nr,nc] = size(G);
if (nr~=n | nc~=n), error('Incorrectly dimensioned G matrix'), end;
[nr,nc] = size(H);
if (nr~=n | nc~=n), error('Incorrectly dimensioned H matrix'), end;

[q,t] = schur([F -G; -H -F']*(1.0+eps*eps*sqrt(-1)));
tol = 10.0*eps*max(abs(diag(t)));       % ad hoc tolerance
ns = 0;
%
% Prepare an array called index to send message to ordering routine
% giving location of eigenvalues with respect to the imaginary axis.
% -1 denotes open left-half-plane
%  1 denotes open right-half-plane
%  0 denotes within tol of imaginary axis
%
for i = 1:2*n,
  if (real(t(i,i)) < -tol),
    index = [ index -1 ];
    ns = ns + 1;
  elseif (real(t(i,i)) > tol),
    index = [ index 1 ];
  else,
    index = [ index 0 ];
  end;
end;
if (ns ~= n),
  error('No solution: (A,B) may be uncontrollable or no solution exists');
end;
[q,t] = schord(q,t,index);
X = real(q(n+1:n+n,1:n)/q(1:n,1:n));


-------------------- /regler/matlab/lqgbox/dare.m --------------------

function S = dare(A,B,Q1,Q2,Q12)
% DARE  dispatch routine for discrete-time algebraic Riccati equation solver
%
%       S = dare(A,B,Q1,Q2,Q12)
%       S = dare(A,B,Q1,Q2)
%
%       Depending on the value of the global variable daretype the
%       corresponding solver is called to solve the discrete-time
%       algebraic equation
%
%       A' S A - S - (A' S B + Q12) (Q2 + B' S B)^(-1) (Q12' + B' S A) + Q1 = 0
%
%       Possible choices for daretype are: 'eigen', 'itera',and 'gener'.
%       If daretype is undefined it is taken as 'itera'.
%
%       Q12 is optional. If omitted it is taken as the empty matrix.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:20:36 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin<5,
  Q12 = zeros(size(A)*[1;0],size(B)*[0;1]);
end

if ~exist('daretype')==1
```

```
  daretype = 'itera';
end

if daretype=='eigen'
  % Direct calculation of the eigenvalues to the Hamiltonian
  if Q12==0
    S = dareeig(A,B,Q2,Q1);
  else
    S = dareeig(A-B/Q2*Q12',B,Q2,Q1-Q12/Q2*Q12');
  end
elseif daretype =='itera'
  % Use iterative method to calculate S
  if Q12==0
    S = dareiter(A,B,Q2,Q1);
  else
    S = dareiter(A-B/Q2*Q12',B,Q2,Q1-Q12/Q2*Q12');
  end
elseif daretype =='gener'
  % CTRL-C algorithm using QZ
  error('Generalized Schur form Riccati solver not yet implemented');
else
  error('Riccati solver in daretype undefined');
end

% check residual

Serr = A'*S*A - S - (A'*S*B + Q12)/(Q2 + B'*S*B)*(Q12' + B'*S*A) + Q1;
disp('');
disp(['1-norm of the Riccati solution residual is ' num2str(norm(Serr,1)) ])
disp('');

-------------------- /regler/matlab/lqgbox/dareeig.m --------------------

function X = dareeig(F,G1,G2,H)
%DAREEIG discrete-time algebraic Riccati equation solver using direct
%       eigenvalue calculation
%
%       X = dareeig(F, G1, G2, H)
%
%       returns the stablizing solution (if it exists) to the discrete-time
%       Riccati equation:
%
%         F' X F - X - F' X G1 (G2 + G1' X G1)^(-1) G1' X F + H = 0
%
%       assuming G is symmetric and nonnegative definite and H is
%       symmetric.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:20:51 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

G = G1/G2*G1';
invF = inv(F);
n = length(F);

ham = [F+G*invF'*H -G*invF'; -invF'*H invF'];

[v,d] = eig(ham);
d = diag(d);
[d,index] = sort(abs(d));
if (~( (d(n)<1) & (d(n+1)>1) ))
```

```
  error('Can''t order eigenvalues, System may be uncontrollable/unobservable.')
end

% select vectors with eigenvalues inside unit circle
x1 = v(1:n,index(1:n));
x2 = v((n+1):(2*n),index(1:n));
X = real(x2/x1);
```

```
-------------------- /regler/matlab/lqgbox/dareiter.m --------------------

function X = dareiter(F,G1,G2,H)
%DAREITER discrete-time algebraic Riccati equation solver using an iterative
%        doubling algorithm
%
%        X = dareiter(F, G1, G2, H)
%
%        returns the stablizing solution (if it exists) to the discrete-time
%        Riccati equation:
%
%          F' X F - X - F' X G1 (G2 + G1' X G1)^(-1) G1' X F + H = 0
%
%        assuming G is symmetric and nonnegative definite and H is
%        symmetric.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:21:11 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

Pnew = F';
Xnew = H;
Wnew = -G1/G2*G1';

X = 0*Xnew;
k = 1;

while norm(Xnew-X,1) > 1e-10*norm(X,1)
  P = Pnew;
  X = Xnew;
  W = Wnew;
  temp1 = (eye(length(X))-X*W)\P;
  temp2 = (eye(length(X))-W*X)\P';
  Pnew = P*temp1;
  Xnew = X + P*X*temp2;
  Wnew = W + P'*W*temp1;
  k = k + 1;
  if k>100
    error('No convergence');
  end
end

X = Xnew;
```

```
-------------------- /regler/matlab/lqgbox/lqec.m --------------------

function [K,P] = lqec(A,C,R1,R2,R12)
% LQEC  Linear quadratic estimator for continuous-time systems
%
%        [K,P] = lqec(A,C,R1,R2,R12)
%        [K,P] = lqec(A,C,R1,R2)
%
%        A gain matrix K is calculated to minimze the variance of the
```

```
%       estimation error in the stationary Kalman filter
%          .
%          x = Ax + Bu + K(y - Cx - Du)
%
%       given the continuous-time system
%          .
%          x = Ax + Bu + v
%          y = Cx + Du + e
%
%       with
%
%          E{v} = 0, E{e} = 0, E{vv'} = R1, E{ee'} = R2, E{ve'} = R12
%
%       Also returned is P, the steady-state the solution to the associated
%       algebraic Riccati equation. P equals the steady state covariance of
%       Kalman filter prediction error.
%
%       The cross-term R12 is optional, If omitted it is regarded as zero.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:21:22 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

% check arguments

error(abcdchk(A,ones(length(A),1),C));

[ma,na] = size(A);
[mc,nc] = size(C);
[mr1,nr1] = size(R1);
if (ma ~= mr1) | (na ~= nr1)
  error('A and R1 must be the same size')
end
[mr2,nr2] = size(R2);
if (mr2 ~= nr2) | (mc ~= mr2)
  error('C and R2 must be consistent')
end

if nargin == 5
  [mr12,nr12] = size(R12);
  if (mr12 ~= ma) | (nr12 ~= nr2)
    error('R12 must be consistent with R1 and R2')
  end
else
  R12 = zeros(mr1,nr2);
end

R = [R1 R12; R12' R2];
% Check if R is positive semi-definite and symmetric
if any(eig(R) < -eps) | (norm(R'-R,1)/norm(R,1) > eps)
  error('[R1 R12; R12'' R2] must be symmetric and positive semi-definite')
end

% Check if R2 is positive definite and symmetric
if any(eig(R2) <= -eps) | (norm(R2'-R2,1)/norm(R2,1) > eps)
  error('R2 must be symmetric and positive definite')
end

% Calculate solution to Riccati equation

P = care(A',C',R1,R2,R12)';
```

```
K = (P*C'+R12)/R2;

-------------------- /regler/matlab/lqgbox/lqed.m --------------------

function [K,Kf,Kv,P,Pf] = lqed(Phi,C,R1,R2,R12)
% LQED  Linear quadratic estimator for discrete-time systems
%
%       [K,Kf,Kv,P,Pf] = lqed(Phi,C,R1,R2,R12)
%       [K,Kf,Kv,P,Pf] = lqed(Phi,C,R1,R2)
%
%       The gain matrices K and Kf are calculated to minimize the variance
%       of the estimation error in the stationary Kalman filter
%
%          x(k+1|k) = Phi x(k|k-1) + Gam u(k) + K (y(k) - C x(k|k-1))
%          x(k|k) = x(k|k-1) + Kf (y(k) - C x(k|k-1))
%          v(k|k) = Kv (y(k) - C x(k|k-1))
%
%       given the discrete-time system
%
%          x(k+1) = Phi x(k) + Gam u(k) + v(k)
%          y(k)   = C x(k) + e(k)
%
%       with
%
%          E{v} = 0, E{e} = 0, E{vv'} = R1, E{ee'} = R2, E{ve'} = R12
%
%       Also returned is P, the steady-state solution to the associated
%       algebraic Riccati equation. P equals the steady state covariance of
%       x(k+1|k) while Pf equals the covariance after the measurment update,
%       i.e. x(k|k)
%
%       The cross-term R12 is optional, If omitted it is regarded as zero.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:22:14 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

% check arguments

error(abcdchk(Phi,ones(length(Phi),1),C));

[ma,na] = size(Phi);
[mc,nc] = size(C);
[mr1,nr1] = size(R1);
if (ma ~= mr1) | (na ~= nr1)
  error('Phi and R1 must be the same size')
end
[mr2,nr2] = size(R2);
if (mr2 ~= nr2) | (mc ~= mr2)
  error('C and R2 must be consistent')
end

if nargin == 5
  [mr12,nr12] = size(R12);
  if (mr12 ~= ma) | (nr12 ~= nr2)
    error('R12 must be consistent with R1 and R2')
  end
else
  R12 = zeros(mr1,nr2);
end
```

```
R = [R1 R12; R12' R2];
% Check if R is positive semi-definite and symmetric
if any(eig(R) < -eps) | (norm(R'-R,1)/norm(R,1) > eps)
  error('[R1 R12; R12'' R2] must be symmetric and positive semi-definite')
end

% Check if R2 is positive definite and symmetric
if any(eig(R2) <= -eps) | (norm(R2'-R2,1)/norm(R2,1) > eps)
  error('R2 must be symmetric and positive definite')
end

% Calculate solution to Riccati equation

P = dare(Phi',C',R1,R2,R12)';
Kf = P*C'/(C*P*C'+R2);                 % (11.50), p353, CCS
Kv = R12/(C*P*C'+R2);                  % (11.50), p353, CCS
K  = Phi*Kf+Kv;                        % Remark 4, p352, CCS
Pf = P - Kf*C*P;                       % (11.50), p353, CCS

-------------------- /regler/matlab/lqgbox/lqgbox.m --------------------

% LQGBOX -- A collection of routines to solve continuous and discrete-time
%           LQG problems. Written by Kjell Gustafsson
%
% Regulator
%
%    LQRC  - continuous-time linear quadratic regulator
%    LQRD  - discrete-time linear quadratic regulator
%
% Estimator
%
%    LQEC  - continuous-time linear quadratic estimator
%    LQED  - discrete-time linear quadratic estimator
%
% Complete controller
%
%    LQGC  - complete continuous-time controller from LQRC, LQEC results
%    LQGD  - complete discrete-time controller from  LQRD, LQED results
%
% Riccati equation solver
%
%    CARE  - general dispatch routine for continuous-time Riccati solvers
%    DARE  - general dispatch routine for discrete-time Riccati solvers
%
% Sampling
%
%    LQGSAMP  - sample loss function and continuous-time noise description
%
% Miscellaneous
%
%    When calling CARE (or DARE) they will examine the variable 'caretype'
%    ('daretype') to determine what kind of solver to use. If this variable
%    is undefined a Schurform solver is used for continuous-time Riccati
%    equations and an iterative solver for the discrete-time case.
%
%    Some rainy day one should implement solvers that use generalized
%    Schur form and QZ factorization (see Ctrl-C manual).
%
%    Currently there is a name conflict between LQRC in LQGBOX and a similar
%    routine in the robust toolbox by Safanov.

% Kjell Gustafsson
```

```
% LastEditDate : Wed Jul  4 11:22:26 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN


--------------------- /regler/matlab/lqgbox/lqgc.m --------------------

function [Ac,Bc,Cc,Dc] = lqgc(A,B,C,D,L,lr,K)
% LQGC  Calculates a continuous-time linear quadratic gaussian controller
%       from data given by LQRC and LQEC.
%
%       [Ac,Bc,Cc,Dc] = lqgc(A,B,C,D,L,lr,K)
%
%       A continuous-time LQG controller can be implemented through
%           .
%          xhat = A xhat + B usat + K ( y - C xhat - D usat )
%          u = lr yr - L xhat
%          usat = sat u
%
%       The controller can also be expressed as (excluding the saturation)
%
%          u = Gff yr - Gff y
%
%       with
%
%          Gff = Cc ( sI - Ac )^(-1) B yr + D yr
%          Gfb = Cc ( sI - Ac )^(-1) B y  + D y
%
%       Note that the 'feedback minus sign' is not included in Gf

% Kjell Gustafsson
% LastEditDate : Mon Aug  6 08:33:27 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

% Construct controller

Ac = A - B*L - K*C + K*D*L;
Byr = K*D-B;
By  = K;
Cc = L;
Dyr = lr;
Dy = zeros(D');

-------------------- /regler/matlab/lqgbox/lqgd.m --------------------

function [Lx,Ly,Phic,Gamy,Gamyr,Cc,Dy,Dyr]=lqgd(Phi,Gam,C,L,Lv,lr,K,Kf,Kv,dir)
% LQGD  Calculates a discrete-time linear quadratic gaussian controller
%       from data given by LQRD and LQED.
%
%       [Phic,Gamy,Gamyr,Cc,Dy,Dyr] = lqgd(Phi,Gam,C,L,Lv,K,Kf,Kv,dir)
%       [Phic,Gamy,Gamyr,Cc,Dy,Dyr] = lqgd(Phi,Gam,C,L,Lv,K,Kf,Kv)
%
%       A discrete-time LQG controller can be implemented through
%
%          x(k+1|k) = Phi x(k|k-1) + Gam usat(k) + K (y(k) - C x(k|k-1)
%          u(k) = lr yr(k) - Lx x(k|k-1) - Ly y(k)
%          usat(k) = sat u(k)
%
%       where the values of Lx and Ly depend on whether the controller
%       contains a direct term or not. If the parameter 'dir' is supplied
%       (any non-empty value) the direct term case is chosen.
```

```
%
%       The controller can also be expressed as (excluding the saturation)
%
%          u(k) = Hff(q) yr(k) - Hfb(q) y(k)
%
%       with
%
%          Hff(q) = Cc ( qI - Phic )^(-1) Gam yr + D yr
%          Hfb(q) = Cc ( qI - Phic )^(-1) Gam y  + D y
%
%       Note that the 'feedback minus sign' is not included in Hfb.

% Kjell Gustafsson
% LastEditDate : Mon Aug  6 08:33:30 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

% check arguments

if nargin<10
  % no direct term
  Ly = zeros(size(Gam)*[0;1],size(C)*[1;0]);
  Lx = L;
else
  % direct term
  Ly = L*Kf + Lv*Kv;
  Lx = L - Ly*C;
end

% controller dynamics

Phic = Phi - Gam*Lx - K*C;
Gamy  = K-Gam*Ly;
Gamyr = -lr*Gam;
Cc = Lx;
Dy  = Ly;
Dyr = lr;


-------------------- /regler/matlab/lqgbox/lqgsamp.m --------------------

function [Phi,Gam,Q1,Q2,Q12,R1,Je] = lqgsamp(A,B,h,Q1c,Q2c,Q12c,R1c)
% LQGSAMP Transform continuous-time LQG problem to the corresponding
%       discrete-time LQG problem
%
%       [Phi,Gam,Q1,Q2,Q12,R1,Js] = lqgsamp(A,B,h,Q1c,Q2c,Q12c,R1c)
%
%       The continuous-time loss function
%
%         Jc = Integral {x' Q1c x + 2 x' Q12c u + u' Q2c u} dt
%
%       subject to the constraint equation
%
%         dx = A x dt + B u dt + dw,      E{w} = 0, E{ww'} = R1c
%
%       is translated into the corresponding discrete-time loss function
%
%         Jd = Sum {x' Q1 x + 2 x' Q12 u + u' Q2 u}
%
%       subject to the constraint equation
%
%         x(k+1) = Phi x(k) + Gam u(k) + v(k),   E{v} = 0, E{vv'} = R1
```

```
%
%       Also returned is Je, an extra term that should be added to the
%       discrete loss function. Je captures the effect of the variation of
%       v during the sampling interval.

% Bo Bernhardsson, original authour
% Kjell Gustafsson, modifications
% LastEditDate : Wed Jul  4 11:23:18 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

% ---- Check the input

error(abcdchk(A,B));
[na,nb] = size(b);

if size(R1c)~=size(A),
   error('R1c and A must have same size');
end

% ---- Calculate extended system matrices
AB = [A B ; zeros(nb,na+nb)];
Qc = [Q1 Q12; Q12' Q2];

% ---- Calculate start values for doubling algorithm
j = max(ceil(log(h*norm(a)/5+eps)/log(2)),3);
delta = h/2^j;
n = zeros(qc);
m = delta*Qc;
mv = delta*R1c;
s = m;
sv = mv;
t = n;
k = 1;

while max(norm(m,1),norm(n,1))>1e-10,
  n = delta*k/(k+1)*m;
  m = delta/(k+1)*(AB'*m+m*AB);
  mv = delta/(k+1)*(A*mv+mv*A');
  s = s+m;
  sv = sv+mv;
  t = t+n;
  k = k+1;
  if k>100
    error('No convergence');
  end;
end;

% ---- The doubling algorithm

ea = expm(AB*delta);
for i=1:j,
  t = t+ea'*(t+delta*2^(i-1)*s)*ea;
  s = s+ea'*s*ea;
  sv = sv+ea(1:na,1:na)*sv*ea(1:na,1:na)';
  ea = ea*ea;
end

Q1 = s(1:na,1:na);
Q12 = s(na+1:na+nb,1:na);
Q2 = s(na+1:na+nb,na+1:na+nb);
```

```
Je = trace((h*s-t)*[R1c zeros(na,nb); zeros(nb,na+nb)]);

% ---- sample system
[Phi,Gam] = c2d(A,B,h);
```

-------------------- /regler/matlab/lqgbox/lqrc.m --------------------

```
function [L,lr,S] = lqrc(A,B,C,D,Q1,Q2,Q12)
% LQRC  Linear quadratic regulator design for continuous-time systems
%
%       [L,lr,S] = lqrc(A,B,C,D,Q1,Q2,Q12)
%       [L,lr,S] = lqrc(A,B,C,D,Q1,Q2)
%
%       A state feedback gain matrix L is calculated such that the feedback
%       law u = -Lx minimizes the cost function:
%
%               J = Integral {x'Q1x + 2x'Q12u + u'Q2u} dt
%
%       subject to the constraint equation:
%               .
%               x = Ax + Bu
%               y = Cx + Du
%
%       When using a Kalman filter, the control signal is formed as
%
%          u(k) = lr yr(k) - L x
%
%       Here, lr is calculated to give steady state gain 1 from yr to y.
%
%       Also returned is S, the steady-state solution to the associated
%       algebraic Riccati equation.
%
%       The cross-term Q12 is optional, If omitted it is regarded as zero.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:23:32 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

% check arguments

error(abcdchk(A,B));

[ma,na] = size(A);
[mb,nb] = size(B);
[mq1,nq1] = size(Q1);
if (ma ~= mq1) | (na ~= nq1)
  error('A and Q1 must be the same size')
end
[mq2,nq2] = size(Q2);
if (mq2 ~= nq2) | (nb ~= mq2)
  error('B and Q2 must be consistent')
end

if nargin == 7
  [mq12,nq12] = size(Q12);
  if (mq12 ~= ma) | (nq12 ~= nq2)
    error('Q12 must be consistent with Q1 and Q2')
  end
```

```
else
  Q12 = zeros(mq1,nq2);
end

Q = [Q1 Q12; Q12' Q2];
% Check if Q is positive semi-definite and symmetric
if any(eig(Q) < -eps) | (norm(Q'-Q,1)/norm(Q,1) > eps)
  error('[Q1 Q12; Q12'' Q2] must be symmetric and positive semi-definite')
end

% Check if Q2 is positive definite and symmetric
if any(eig(Q2) <= eps) | (norm(Q2'-Q2,1)/norm(Q2,1) > eps)
  error('Q2 must be symmetric and positive definite')
end

% Calculate solution to Riccati equation

S = care(A,B,Q1,Q2,Q12);
L = Q2\(Q12'+B'*S);

% Calculate lr to adjust steady state gain

lr = 1/(C*inv(-A+B*L)*B+D);

-------------------- /regler/matlab/lqgbox/lqrd.m --------------------

function [L,Lv,lr,S] = lqrd(Phi,Gam,C,Q1,Q2,Q12)
% LQRD  Linear quadratic regulator design for discrete-time systems
%
%       [L,Lv,lr,S] = lqrd(Phi,Gam,C,Q1,Q2,Q12)
%       [L,Lv,lr,S] = lqrd(Phi,Gam,C,Q1,Q2)
%
%       A state feedback gain matrix L is calculated such that the feedback
%       law u = -Lx minimizes the cost function:
%
%          J = Integral {x'Q1x + 2x'Q12u + u'Q2u} dt
%
%       subject to the constraint equation:
%
%          x(k+1) = Phi x(k) + Gam u(k)
%          y(k)   = C x(k)
%
%       When using a Kalman filter, the control signal is formed as
%
%          u(k) = lr yr(k) - L x(k|k-1)               (no direct term)
%          u(k) = lr yr(k) - L x(k|k) - Lv v(k|k)     (direct term)
%
%       Here, lr is calculated to give steady state gain 1 from yr to y.
%
%       Also returned is S, the steady-state solution to the associated
%       algebraic Riccati equation.
%
%       The cross-term Q12 is optional, If omitted it is regarded as zero.

% Kjell Gustafsson
% LastEditDate : Wed Jul  4 11:23:43 1990
% Copyright (c) 1990 by Kjell Gustafsson and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

% check arguments

error(abcdchk(Phi,Gam));
```

```
[ma,na] = size(Phi);
[mb,nb] = size(Gam);
[mq1,nq1] = size(Q1);
if (ma ~= mq1) | (na ~= nq1)
  error('Phi and Q1 must be the same size')
end
[mq2,nq2] = size(Q2);
if (mq2 ~= nq2) | (nb ~= mq2)
  error('Gam and Q2 must be consistent')
end

if nargin == 6
  [mq12,nq12] = size(Q12);
  if (mq12 ~= ma) | (nq12 ~= nq2)
    error('Q12 must be consistent with Q1 and Q2')
  end
else
  Q12 = zeros(mq1,nq2);
end

Q = [Q1 Q12; Q12' Q2];
% Check if Q is positive semi-definite and symmetric
if any(eig(Q) < -eps) | (norm(Q'-Q,1)/norm(Q,1) > eps)
  error('[Q1 Q12; Q12'' Q2] must be symmetric and positive semi-definite')
end

% Check if Q2 is positive semi-definite and symmetric
if any(eig(Q2) < -eps) | (norm(Q2'-Q2,1)/norm(Q2,1) > eps)
  error('Q2 must be symmetric and positive semi-definite')
end

% Calculate solution to Riccati equation

S = dare(Phi,Gam,Q1,Q2,Q12);
L  = (Q2 + Gam'*S*Gam)\(Gam'*S*Phi + Q12');  % (Remark 2), p341 CCS
Lv = (Q2 + Gam'*S*Gam)\(Gam'*S);             % p353 CCS

% Calculate lr to adjust steady state gain

lr = 1/(C*inv(eye(Phi)-Phi+Gam*L)*Gam);
```

6. MISC

```
-------------------- /regler/matlab/misc/fstab.m --------------------

function B = fstab(A);
%FSTAB  FSTAB(A) stabilizes a MONIC polynomial with respect to the
%       unit circle, i.e. roots whose magnitudes are greater than
%       one are reflected into the unit circle.  The result is a monic
%       polynomial as well.  Used by YULEWALK.
%
%       Original routine crashed if deg(A) = 0. Fixed /KG
if length(A)~=1,
  v = roots(A); ind=(abs(v)>eps);
  vs = 0.5*(sign(abs(v(ind))-1)+1);
  v(ind) = (1-vs).*v(ind) + vs./ (conj(v(ind)));
  B = real(poly(v));
else
  B = A;
end

-------------------- /regler/matlab/misc/matlab.m --------------------

% Master startup M-file, executed by MATLAB at startup time.  On
% multi-user or networked systems, the system manager can put here
% any messages, definitions, etc. that apply to all users.

disp('             HELP, DEMO, INFO, and TERMINAL are available')

% Now execute user's own startup M-file:
if exist('startup')
       startup
end

% Load ACSL exchange file, if it exists. (Uncomment if you own ACSL).
%if exist('acsl_matlab.tmp')
%       load acsl_matlab.tmp
%       delete acsl_matlab.tmp
%end

% Transfer to Protoblock, if it exists. (Uncomment if you own Protoblock).
%if exist('protoblock.m')
%        system_dependent(3);
%end

-------------------- /regler/matlab/misc/mfiles.m --------------------

function mfiles(string,option)
% MFILES Listing of matlab files in the search path.
%
%       MFILES(STRING)
%
%       lists all files in the MATLAB search path which matches
%       STRING concatenated with ".m" according to the rules for
%       file substitution in c-shell (do "man csh" in unix).
%
%       MFILES(STRING,OPTION)
%
%       gives the complete path name for each file, whatever the
%       second argument OPTION may be.
%
%       Example 1: Find all MATLAB files with names starting with
```

```
%       "bo" followed by precisely two more characters.
%
%         mfiles('bo??')
%
%       which may give the result
%
%         bopl
%         bosh
%         bode
%
%       Example 2: Find all MATLAB files with names containing the
%       string "rst" and supply the full path names.
%
%         mfiles('*rst*',1)
%
%       which could give, for example
%
%         /regler/matlab/ppbox/rst2sim.m
%         /regler/matlab/ppbox/rstc.m
%         /regler/matlab/ppbox/rstd.m
%         /regler/matlab/frlsbox/lsrstc.m
%         /regler/matlab/frlsbox/lsrstd.m
%         /regler/matlab/frlsbox/rstfit.m
%

% Mats Lilja
% LastEditDate : Fri Jul 20 13:08:06 1990
% Copyright (c) 1990 by Mats Lilja and Department of Automatic Control,
% Lund Institute of Technology, Lund, SWEDEN

if nargin < 2,
  dostrip = '| sed ''s#.*/##;s#[.]m##''';
else,
  dostrip = [];
end;
do1=['! (\ls `echo $MATLABPATH | sed ''s#:#/' string '.m #g;s#$#/'];
do2=[string '.m #''` ' dostrip ' > /dev/tty ) >& /dev/null '];
doit = [do1 do2];
eval(doit);
```

-------------------- /regler/matlab/misc/print.m --------------------

```
function print
%PRINT  Send the graph currently on the screen to a local printer.
%       PRINT is an M-file that you may wish to customize to indicate
%       your printer type and destination.

% Apollo is different:
if strcmp(computer,'APOLLO')
        meta metatmp            % Put current plot into temporary metafile
        !gpp metatmp -dps       % Invoke GPP, creating device specific output
        delete metatmp.met      % Delete temporary metafile
        !prf -transparent metatmp.ps    % Spool output to printer
        return
end

% Unix is all the same:
t = fix(clock);
fname = ['mtmp' sprintf('%02.0f%02.0f%02.0f',t(4),t(5),t(6))]; % unique name
eval(['meta ' fname])                     % Current plot to temporary metafile
gpp_cmd = ['gpp ' fname ' -dps'];         % gpp cmd and options
rm_cmd  = ['\rm ' fname '.met'];                  % remove temporary meta file
```

```
lpr_cmd = ['lpr -r ' fname '.ps'];       % print cmd with file remove
semi = ';';
eval(['!('   gpp_cmd   semi   rm_cmd   semi   lpr_cmd   ') &'])
```

```
-------------------- /regler/matlab/misc/simnon.m --------------------

function simnon(linsys,dt,a,b,c,d,x0,consys,opt)

% ** simnon(linsys,dt,a,b,c,d,x0,consys,opt) ** Converts a Matlab (MathWorks)
% linear time-invariant system into its Simnon (Dept. of Automatic Control,
% Lund Institute of Technology, Lund SWEDEN) equivalent
%
% linsys  name of Simnon linear system description (file linsys.t)
% dt      sampling interval (0 for continuous time)
% a,b,c,d system matrices (d is optional; default 0)
% x0      initial state vector (row or column; optional; default 0)
% consys  name of Simnon connecting system template, merely defining linsys's
%         inputs as 0 (file consys.t; should be edited by the user; optional)
% opt     optimization = 0 or 1. If 1, redundant adds and mults will be
%         removed; use opt=0 if structure matters (optional; default 1)
%
% Note    Simnon needs a CONNECTING system to drive the input signals
%         For large systems auxiliary variables named _1, _2, etc may appear
%         To omit an optional argument, don't enter it or specify it as []
%
% Example: Continuous a,b,c-system (with/without connecting system template)
% >> simnon('sc1', 0, [-2 -1; 1 0], [0; 1], [1 0], [], [], 'cc1')
% >> simnon('sc1', 0, [-2 -1; 1 0], [0; 1], [1 0])

% Tomas Schonthal 1988-09-14
% Department of Automatic Control, Lund Institute of Technology, Lund SWEDEN

% Initialize and configure
version='V1.00';
nr_aux=0;
add_sav=0;
mul_sav=0;
discrete=dt>0;
d_supplied=nargin>=6;
if d_supplied
  d_supplied=size(d)~=[0 0];
end
x0_supplied=nargin>=7;
if x0_supplied
  x0_supplied=size(x0)~=[0 0];
end
consys_supplied=nargin>=8;
if consys_supplied
  consys_supplied=size(consys)~=[0 0];
end
opt_supplied=nargin>=9;
if opt_supplied
  opt_supplied=size(opt)~=[0 0];
end
if ~opt_supplied
  opt=1;
end

% Check input
if nargin<5
  disp('**Too few arguments; must at least have: linsys,dt,a,b,c')
```

```
  return
end
if consys_supplied
  if size(consys)==size(linsys)
    if consys==linsys
      disp('**consys coincides with linsys')
      return
    end
  end
end
if size(dt)~=[1 1]
  disp('**Non-scalar sampling interval')
  return
end
if dt<0
  disp('**Negative sampling interval')
  return
end
[nx,i]=size(a);
if nx<=0
  disp('**Empty A-matrix')
  return
end
if i~=nx
  disp('**A-matrix not square')
  return
end
[i,nu]=size(b);
if nu<=0
  disp('**Empty B-matrix')
  return
end
if i~=nx
  disp('**Incompatible A and B-matrices')
  return
end
[ny,i]=size(c);
if ny<=0
  disp('**Empty C-matrix')
  return
end
if i~=nx
  disp('**Incompatible A and C-matrices')
  return
end
if d_supplied
  [i,nu2]=size(d);
  if i~=ny | nu2~=nu
    disp('**Incompatible C and D or B and D-matrices')
    return
  end
else
  nu2=0;
  d=[];
end
if x0_supplied
  [i,j]=size(x0);
  if j>i
    x0=x0';
    [i,j]=size(x0);
  end
  if i~=nx | j~=1
```

```
    disp('**Incompatible X0-vector')
    return
  end
end

% Looks OK, so delete any previous version of the output file
file=[linsys '.t'];
if exist(file)==2
  eval(['delete ' file])
end

disp(['--Begin linear system ' linsys ', file ' file '; simnon.m ' version])
if opt
  disp('**Code optimization attempted; opt=0 overrides')
else
  disp('**Code optimization NOT attempted; opt=1 overrides')
end
if discrete
  dxnx='nx';
  fprintf(file,['DISCRETE SYSTEM ' linsys])
else
  dxnx='dx';
  fprintf(file,['CONTINUOUS SYSTEM ' linsys])
end
time_stamp=fix(clock);
time_stamp=[sprintf('%g-%g-%g ',time_stamp(1),time_stamp(2),time_stamp(3)) ..
            sprintf('%g:%g:%g',time_stamp(4),time_stamp(5),time_stamp(6))];
fprintf(file,'\n\n" Linear, time-invariant Simnon system')
fprintf(file,'\n" Note: Needs a CONNECTING system to drive input signals')
fprintf(file,['\n\n" Created in Matlab by simnon.m ' version ' at ' ..
             time_stamp])
fprintf(file,['\n" Dept. of Automatic Control, ' ..
              'Lund Institute of Technology, Lund SWEDEN\n'])
if opt
  fprintf(file,'\n" **Code optimization attempted; opt=0 overrides\n')
else
  fprintf(file,'\n" **Code optimization NOT attempted; opt=1 overrides\n')
end

disp('  --Input, output, state, der (new, time, tsamp) declarations')
simnon3(file,'INPUT','u',nu)
simnon3(file,'OUTPUT','y',ny)
simnon3(file,'STATE','x',nx)
if discrete
  simnon3(file,'NEW','nx',nx)
  fprintf(file,'\nTIME t')
  fprintf(file,'\nTSAMP ts')
else
  simnon3(file,'DER','dx',nx)
end
fprintf(file,'\n')

disp('  --Matrix elements -> parameter assignments')
disp('    --A-matrix')
simnon2(a,nx,nx,file,'a',opt)
disp('    --B-matrix')
simnon2(b,nx,nu,file,'b',opt)
disp('    --C-matrix')
simnon2(c,ny,nx,file,'c',opt)
if d_supplied
  disp('    --D-matrix')
  simnon2(d,ny,nu,file,'d',opt)
```

```
end

disp('  --Der (new) variable assignments')
for i=1:nx,
  [nr_aux add_sav mul_sav]=simnon4(file,dxnx,i,nx,nu,'a','b', ..
                                   [nr_aux add_sav mul_sav],opt,a,b);
end
fprintf(file,'\n')

disp('  --Output variable assignments')
for i=1:ny,
  [nr_aux add_sav mul_sav]=simnon4(file,'y',i,nx,nu2,'c','d', ..
                                   [nr_aux add_sav mul_sav],opt,c,d);
end
fprintf(file,'\n')

term_tot=nx*(nx+nu)+ny*(nx+nu2);
add_sav_pct=fix((100*add_sav)/term_tot);
mul_sav_pct=fix((100*mul_sav)/term_tot);
disp(sprintf('    --%g axiliary variables introduced',nr_aux))
disp(sprintf('    --%g redundant adds removed (%g%%)',add_sav,add_sav_pct))
disp(sprintf('    --%g redundant mults removed (%g%%)',mul_sav,mul_sav_pct))
if opt
  fprintf(file,'\n" --%g redundant adds removed (%g%%)',add_sav,add_sav_pct)
  fprintf(file,'\n" --%g redundant mults removed (%g%%)\n',mul_sav,mul_sav_pct)
end

if discrete
  disp('  --Tsamp update')
  fprintf(file,'\nts=t+dt')
  fprintf(file,'\ndt: %g',dt)
  fprintf(file,'\n')
end
if x0_supplied
  disp('  --X0-vector')
  simnon2(x0,nx,1,file,'x',0)
end
fprintf(file,'\nEND\n')

if consys_supplied
  file=[consys '.t'];
  if exist(file)==2
    eval(['delete ' file])
  end

  disp(['--Begin connecting system template ' consys ', file ' file])
  fprintf(file,['CONNECTING SYSTEM ' consys])
  fprintf(file,['\n\n" Drives Simnon system ' linsys ' with zero inputs'])
  fprintf(file,['\n" Note: This is merely a template and should ' ..
                'therefore be edited by the user'])
  fprintf(file,['\n\n" Created in Matlab by simnon.m ' version ' at ' ..
               time_stamp])
  fprintf(file,['\n" Dept. of Automatic Control, ' ..
                'Lund Institute of Technology, Lund SWEDEN\n'])

  disp('  --Input variable assignments')
  for i=1:nu,
    fprintf(file,[sprintf(['\nu%g[' linsys ']='],i) sprintf('u%gp',i)])
  end
  fprintf(file,'\n')

  disp('  --Parameter assignments for input levels')
```

```
  for i=1:nu,
    fprintf(file,'\nu%gp: 0',i)
  end

  fprintf(file,'\nEND\n')
end
disp('--Done')
```

-------------------- /regler/matlab/misc/simnon2.m --------------------

```
function simnon2(matrix,rows,columns,file,type_string,opt)

% Help routine to simnon.m
% Converts a Matlab matrix to a sequence of Simnon parameter assignments

% Tomas Schonthal 1988-09-14
% Dept. of Automatic Control, Lund Institute of Technology, Lund SWEDEN

par_pri=0;

for i=1:rows,
  for j=1:columns,
    r=matrix(i,j);
    if ~opt | (r~=0 & abs(r)~=1)
      par_pri=1;
      if type_string=='x'
        fprintf(file,['\n' type_string '%g: %g'],i,r)
      else
        fprintf(file,['\n' type_string '%g_%g: %g'],i,j,r)
      end
    end
  end
end
if par_pri
  fprintf(file,'\n')
end
```

-------------------- /regler/matlab/misc/simnon3.m --------------------

```
function simnon3(file,declaration,variable,n)

% Help routine to simnon.m
% Writes an INPUT, OUTPUT, STATE, DER or NEW declaration

% Tomas Schonthal 1988-09-14
% Department of Automatic Control, Lund Institute of Technology, Lund SWEDEN

format_string=[' ' variable '%g'];

for i=1:n,
  if rem(i,10)==1
    buff=['\n' declaration];
  end
  buff=[buff sprintf(format_string,i)];
  if rem(i,10)==0 | i==n
    fprintf(file,buff)
  end
end
```

-------------------- /regler/matlab/misc/simnon4.m --------------------

```
function [nr_aux,add_sav,mul_sav]=simnon4(file,variable,i,nx,nu,ac_nam, ..
```

```
                                         bd_nam,old_stat,opt,ac_mat,bd_mat)
% Help routine to simnon.m
% Writes an equation

% Tomas Schonthal 1988-09-14
% Department of Automatic Control, Lund Institute of Technology, Lund SWEDEN

char_lim_1=71;
char_lim_2=79;

ac_nam=[ac_nam sprintf('%g_',i)];
bd_nam=[bd_nam sprintf('%g_',i)];
equation=['\n' variable sprintf('%g',i) '='];
nr_aux=old_stat(1);
add_sav=old_stat(2);
mul_sav=old_stat(3);
no_terms=1;

for columns=1:nx+nu,
  if columns<=nx
    matrix_nam=ac_nam;
    vector_nam='x';
    j=columns;
    r=ac_mat(i,j);
  else
    matrix_nam=bd_nam;
    vector_nam='u';
    j=columns-nx;
    r=bd_mat(i,j);
  end
  full_term= ~opt | (r~=0 & abs(r)~=1);
  half_term=opt & abs(r)==1;
  if full_term
    term=[matrix_nam sprintf('%g*',j) vector_nam sprintf('%g',j)];
  elseif half_term
    term=[vector_nam sprintf('%g',j)];
    mul_sav=mul_sav+1;
  else
    term=[];
    add_sav=add_sav+1;
    mul_sav=mul_sav+1;
  end
  if columns<nx+nu
    char_lim=char_lim_1;
  else
    char_lim=char_lim_2;
  end
  if max(size(equation))+max(size(term))>char_lim
    nr_aux=nr_aux+1;
    fprintf(file,[equation sprintf('+_%g',nr_aux)])
    equation=sprintf('\n  _%g=',nr_aux);
    no_terms=1;
  end
  if ~no_terms & (full_term | (half_term & r==1))
    term=['+' term];
  elseif r==-1
    term=['-' term];
  end
  if size(term)~=[0 0]
    equation=[equation term];
    no_terms=0;
  end
```

```
end

if no_terms
  equation=[equation '0'];
end
fprintf(file,equation)
```