

control 2.2.4

Control Systems Package for GNU Octave

Lukas F. Reichlin

Copyright © 2009-2012, Lukas F. Reichlin lukas.reichlin@gmail.com

This manual is generated automatically from the texinfo help strings of the package's functions.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the same conditions as for modified versions.

Preface

The GNU Octave control package from version 2 onwards was developed by Lukas F. Reichlin and is based on the proven open-source library SLICOT. This new package is intended as a replacement for control-1.0.11 by A. Scottedward Hodel and his students. Its main features are:

- Reliable solvers for Lyapunov, Sylvester and algebraic Riccati equations.
- Pole placement techniques as well as H_2 and H_∞ synthesis methods.
- Overloaded operators due to the use of classes introduced with Octave 3.2.
- Support for descriptor state-space models and non-proper transfer functions.
- Improved MATLAB compatibility.

Acknowledgments

The author is indebted to several people and institutions who helped him to achieve his goals. I am particularly grateful to Luca Favatella who introduced me to Octave development as well as discussed and revised my early draft code with great patience. My continued support from the FHNW University of Applied Sciences of Northwestern Switzerland, where I could work on the control package as a semester project, has also been important. Furthermore, I thank the SLICOT authors Peter Benner, Vasile Sima and Andras Varga for their advice.

Using the help function

Some functions of the control package are listed with a leading `@lti/`. This is only needed to view the help text of the function, e.g. `help norm` shows the built-in function while `help @lti/norm` shows the overloaded function for LTI systems. Note that there are LTI functions like `pole` that have no built-in equivalent.

When just using the function, the leading `@lti/` must **not** be typed. Octave selects the right function automatically. So one can type `norm(sys, inf)` and `norm(matrix, inf)` regardless of the class of the argument.

Table of Contents

1	Examples	1
1.1	MDSSystem	1
1.2	optiPID	1
2	Linear Time Invariant Models	2
2.1	dss	2
2.2	frd	2
2.3	ss	3
2.4	tf	4
2.5	zpk	6
3	Model Data Access	8
3.1	@lti/dssdata	8
3.2	@lti/frdata	8
3.3	@lti/get	8
3.4	@lti/set	9
3.5	@lti/ssdata	9
3.6	@lti/tfdata	9
3.7	@lti/zpkdata	10
4	Model Conversions	11
4.1	@lti/c2d	11
4.2	@lti/d2c	11
4.3	@lti/prescale	11
4.4	@lti/xperm	12
5	Model Interconnections	13
5.1	@lti/append	13
5.2	@lti/blkdiag	13
5.3	@lti/connect	13
5.4	@lti/feedback	13
5.5	@lti/lft	14
5.6	@lti/mconnect	15
5.7	@lti/parallel	16
5.8	@lti/series	16
6	Model Characteristics	17
6.1	ctrb	17
6.2	ctrbf	17
6.3	@lti/dcgain	17
6.4	gram	18
6.5	hsvd	18
6.6	@lti/isct	18
6.7	isctrb	18
6.8	isdetectable	19
6.9	@lti/isdt	19

6.10	@lti/isminimumphase	20
6.11	isobsv	20
6.12	@lti/issiso	20
6.13	isstabilizable	20
6.14	@lti/isstable	21
6.15	@lti/norm	21
6.16	obsv	22
6.17	obsvf	22
6.18	@lti/pole	22
6.19	pzmap	23
6.20	@lti/size	23
6.21	@lti/zero	23
7	Model Simplification	24
7.1	@lti/minreal	24
7.2	@lti/sminreal	24
8	Time Domain Analysis	25
8.1	covar	25
8.2	gensig	25
8.3	impulse	26
8.4	initial	26
8.5	lsim	27
8.6	step	27
9	Frequency Domain Analysis	29
9.1	bode	29
9.2	bodemag	29
9.3	@lti/freqresp	29
9.4	margin	30
9.5	nichols	31
9.6	nyquist	32
9.7	sigma	32
10	Pole Placement	34
10.1	place	34
10.2	rlocus	35
11	Linear-Quadratic Control	36
11.1	dlqr	36
11.2	estim	36
11.3	kalman	37
11.4	lqr	38
12	Robust Control	39
12.1	augw	39
12.2	h2syn	40
12.3	hinfsyn	41
12.4	mixsyn	42
12.5	ncfsyn	44

13	Matrix Equation Solvers	45
13.1	care	45
13.2	dare	46
13.3	dlyap	47
13.4	dlyapchol	48
13.5	lyap	48
13.6	lyapchol	48
14	Overloaded Operators	49
14.1	@lti/horzcat	49
14.2	@lti/inv	49
14.3	@lti/minus	49
14.4	@lti/mldivide	49
14.5	@lti/mpower	49
14.6	@lti/mrdivide	49
14.7	@lti/mtimes	49
14.8	@lti/plus	49
14.9	@lti/subsasgn	49
14.10	@lti/subsref	49
14.11	@lti/transpose	49
14.12	@lti/uminus	50
14.13	@lti/vertcat	50
15	Miscellaneous	51
15.1	strseq	51
15.2	test_control	51
15.3	BMWengine	51
15.4	Boeing707	52
15.5	WestlandLynx	52

1 Examples

1.1 MDSSystem

Robust control of a mass-damper-spring system. Type `which MDSSystem` to locate, `edit MDSSystem` to open and simply `MDSSystem` to run the example file.

1.2 optiPID

Numerical optimization of a PID controller using an objective function. The objective function is located in the file `optiPIDfun`. Type `which optiPID` to locate, `edit optiPID` to open and simply `optiPID` to run the example file.

2 Linear Time Invariant Models

2.1 dss

`sys = dss (sys)` [Function File]
`sys = dss (d)` [Function File]
`sys = dss (a, b, c, d, e, ...)` [Function File]
`sys = dss (a, b, c, d, e, tsam, ...)` [Function File]

Create or convert to descriptor state-space model.

Inputs

`sys` LTI model to be converted to state-space.
`a` State transition matrix (n-by-n).
`b` Input matrix (n-by-m).
`c` Measurement matrix (p-by-n).
`d` Feedthrough matrix (p-by-m).
`e` Descriptor matrix (n-by-n).
`tsam` Sampling time in seconds. If `tsam` is not specified, a continuous-time model is assumed.
`...` Optional pairs of properties and values. Type `set (dss)` for more information.

Outputs

`sys` Descriptor state-space model.

Equations

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{u} \\ \mathbf{y} &= \mathbf{C} \mathbf{x} + \mathbf{D} \mathbf{u} \end{aligned}$$

ss, tf

2.2 frd

`sys = frd (sys)` [Function File]
`sys = frd (sys, w)` [Function File]
`sys = frd (H, w, ...)` [Function File]
`sys = frd (H, w, tsam, ...)` [Function File]

Create or convert to frequency response data.

Inputs

`sys` LTI model to be converted to frequency response data. If second argument `w` is omitted, the interesting frequency range is calculated by the zeros and poles of `sys`.
`H` Frequency response array (p-by-m-by-lw). `H(i,j,k)` contains the response from input `j` to output `i` at frequency `k`. In the SISO case, a vector (lw-by-1) or (1-by-lw) is accepted as well.
`w` Frequency vector (lw-by-1) in radian per second [rad/s]. Frequencies must be in ascending order.

tsam Sampling time in seconds. If *tsam* is not specified, a continuous-time model is assumed.

... Optional pairs of properties and values. Type `set (frd)` for more information.

Outputs

sys Frequency response data object.

dss, ss, tf

2.3 ss

<code>sys = ss (sys)</code>	[Function File]
<code>sys = ss (d)</code>	[Function File]
<code>sys = ss (a, b)</code>	[Function File]
<code>sys = ss (a, b, c)</code>	[Function File]
<code>sys = ss (a, b, c, d, ...)</code>	[Function File]
<code>sys = ss (a, b, c, d, tsam, ...)</code>	[Function File]

Create or convert to state-space model.

Inputs

sys LTI model to be converted to state-space.

a State transition matrix (n-by-n).

b Input matrix (n-by-m).

c Measurement matrix (p-by-n). If *c* is empty `[]` or not specified, an identity matrix is assumed.

d Feedthrough matrix (p-by-m). If *d* is empty `[]` or not specified, a zero matrix is assumed.

tsam Sampling time in seconds. If *tsam* is not specified, a continuous-time model is assumed.

... Optional pairs of properties and values. Type `set (ss)` for more information.

Outputs

sys State-space model.

Example

```

octave:1> a = [1 2 3; 4 5 6; 7 8 9];
octave:2> b = [10; 11; 12];
octave:3> stname = {"V", "A", "kJ"};
octave:4> sys = ss (a, b, [], [], "stname", stname)

```

```

sys.a =
      V      A      kJ
      V      1      2      3
      A      4      5      6
      kJ      7      8      9

```

```

sys.b =
      u1
      V      10
      A      11
      kJ      12

```

```

sys.c =
      V      A      kJ
      y1      1      0      0
      y2      0      1      0
      y3      0      0      1

```

```

sys.d =
      u1
      y1      0
      y2      0
      y3      0

```

Continuous-time model.

```
octave:5>
```

tf, dss

2.4 tf

```

s = tf ("s") [Function File]
z = tf ("z", tsam) [Function File]
sys = tf (sys) [Function File]
sys = tf (num, den, ...) [Function File]
sys = tf (num, den, tsam, ...) [Function File]

```

Create or convert to transfer function model.

Inputs

<i>sys</i>	LTI model to be converted to transfer function.
<i>num</i>	Numerator or cell of numerators. Each numerator must be a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. <i>num</i> {i,j} contains the numerator polynomial from input j to output i. In the SISO case, a single vector is accepted as well.
<i>den</i>	Denominator or cell of denominators. Each denominator must be a row vector containing the coefficients of the polynomial in descending powers of the transfer

function variable. `den{i,j}` contains the denominator polynomial from input `j` to output `i`. In the SISO case, a single vector is accepted as well.

`tsam` Sampling time in seconds. If `tsam` is not specified, a continuous-time model is assumed.

... Optional pairs of properties and values. Type `set (tf)` for more information.

Outputs

`sys` Transfer function model.

Example

```
octave:1> s = tf ("s");
octave:2> G = 1/(s+1)
```

Transfer function "G" from input "u1" to output ...

$$\begin{array}{r} 1 \\ y1: \text{-----} \\ s + 1 \end{array}$$

Continuous-time model.

```
octave:3> z = tf ("z", 0.2);
octave:4> H = 0.095/(z-0.9)
```

Transfer function "H" from input "u1" to output ...

$$\begin{array}{r} 0.095 \\ y1: \text{-----} \\ z - 0.9 \end{array}$$

Sampling time: 0.2 s
Discrete-time model.

```
octave:5> num = {[1, 5, 7], [1]; [1, 7], [1, 5, 5]};
octave:6> den = {[1, 5, 6], [1, 2]; [1, 8, 6], [1, 3, 2]};
octave:7> sys = tf (num, den)
```

Transfer function "sys" from input "u1" to output ...

$$y1: \frac{s^2 + 5s + 7}{s^2 + 5s + 6}$$

$$y2: \frac{s + 7}{s^2 + 8s + 6}$$

Transfer function "sys" from input "u2" to output ...

$$y1: \frac{1}{s + 2}$$

$$y2: \frac{s^2 + 5s + 5}{s^2 + 3s + 2}$$

Continuous-time model.

```
octave:8>
```

ss, dss

2.5 zpk

```
s = zpk ("s") [Function File]
z = zpk ("z", tsam) [Function File]
sys = zpk (sys) [Function File]
sys = zpk (k) [Function File]
sys = zpk (z, p, k, ...) [Function File]
sys = zpk (z, p, k, tsam, ...) [Function File]
sys = zpk (z, p, k, tsam, ...) [Function File]
```

Create transfer function model from zero-pole-gain data. This is just a stop-gap compatibility wrapper since zpk models are not yet implemented.

Inputs

<i>sys</i>	LTI model to be converted to transfer function.
<i>z</i>	Cell of vectors containing the zeros for each channel. <i>z</i> _{i,j} contains the zeros from input <i>j</i> to output <i>i</i> . In the SISO case, a single vector is accepted as well.
<i>p</i>	Cell of vectors containing the poles for each channel. <i>p</i> _{i,j} contains the poles from input <i>j</i> to output <i>i</i> . In the SISO case, a single vector is accepted as well.
<i>k</i>	Matrix containing the gains for each channel. <i>k</i> (<i>i,j</i>) contains the gain from input <i>j</i> to output <i>i</i> .

tsam Sampling time in seconds. If *tsam* is not specified, a continuous-time model is assumed.

. . . Optional pairs of properties and values. Type `set (tf)` for more information.

Outputs

sys Transfer function model.

tf, ss, dss, frd

3 Model Data Access

3.1 @lti/dssdata

`[a, b, c, d, e, tsam] = dssdata (sys)` [Function File]
`[a, b, c, d, e, tsam] = dssdata (sys, [])` [Function File]

Access descriptor state-space model data. Argument `sys` is not limited to descriptor state-space models. If `sys` is not a descriptor state-space model, it is converted automatically.

Inputs

`sys` Any type of LTI model.
`[]` In case `sys` is not a dss model (descriptor matrix `e` empty), `dssdata (sys, [])` returns the empty element `e = []` whereas `dssdata (sys)` returns the identity matrix `e = eye (size (a))`.

Outputs

`a` State transition matrix (n-by-n).
`b` Input matrix (n-by-m).
`c` Measurement matrix (p-by-n).
`d` Feedthrough matrix (p-by-m).
`e` Descriptor matrix (n-by-n).
`tsam` Sampling time in seconds. If `sys` is a continuous-time model, a zero is returned.

3.2 @lti/frdata

`[H, w, tsam] = frdata (sys)` [Function File]
`[H, w, tsam] = frdata (sys, "vector")` [Function File]

Access frequency response data. Argument `sys` is not limited to frequency response data objects. If `sys` is not a frd object, it is converted automatically.

Inputs

`sys` Any type of LTI model.
`"v", "vector"` In case `sys` is a SISO model, this option returns the frequency response as a column vector (lw-by-1) instead of an array (p-by-m-by-lw).

Outputs

`H` Frequency response array (p-by-m-by-lw). `H(i,j,k)` contains the response from input `j` to output `i` at frequency `k`. In the SISO case, a vector (lw-by-1) is possible as well.
`w` Frequency vector (lw-by-1) in radian per second [rad/s]. Frequencies are in ascending order.
`tsam` Sampling time in seconds. If `sys` is a continuous-time model, a zero is returned.

3.3 @lti/get

`get (sys)` [Function File]
`value = get (sys, "property")` [Function File]

Access property values of LTI objects.

3.4 @lti/set

`set (sys)` [Function File]
`sys = set (sys, "property", value)` [Function File]
 Set or modify properties of LTI objects.

3.5 @lti/ssdata

`[a, b, c, d, tsam] = ssdata (sys)` [Function File]
 Access state-space model data. Argument `sys` is not limited to state-space models. If `sys` is not a state-space model, it is converted automatically.

Inputs

`sys` Any type of LTI model.

Outputs

`a` State transition matrix (n-by-n).
`b` Input matrix (n-by-m).
`c` Measurement matrix (p-by-n).
`d` Feedthrough matrix (p-by-m).
`tsam` Sampling time in seconds. If `sys` is a continuous-time model, a zero is returned.

3.6 @lti/tfdata

`[num, den, tsam] = tfdata (sys)` [Function File]
`[num, den, tsam] = tfdata (sys, "vector")` [Function File]
`[num, den, tsam] = tfdata (sys, "tfpoly")` [Function File]

Access transfer function data. Argument `sys` is not limited to transfer function models. If `sys` is not a transfer function, it is converted automatically.

Inputs

`sys` Any type of LTI model.

`"v", "vector"`

For SISO models, return `num` and `den` directly as column vectors instead of cells containing a single column vector.

Outputs

`num` Cell of numerator(s). Each numerator is a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. `num{i,j}` contains the numerator polynomial from input `j` to output `i`. In the SISO case, a single vector is possible as well.
`den` Cell of denominator(s). Each denominator is a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. `den{i,j}` contains the denominator polynomial from input `j` to output `i`. In the SISO case, a single vector is possible as well.
`tsam` Sampling time in seconds. If `sys` is a continuous-time model, a zero is returned.

3.7 @lti/zpkdata

`[z, p, k, tsam] = zpkdata (sys)` [Function File]

`[z, p, k, tsam] = zpkdata (sys, "v")` [Function File]

Access zero-pole-gain data.

Inputs

sys Any type of LTI model.

"v", "vector"

For SISO models, return *z* and *p* directly as column vectors instead of cells containing a single column vector.

Outputs

z Cell of column vectors containing the zeros for each channel. *z*{i,j} contains the zeros from input j to output i.

p Cell of column vectors containing the poles for each channel. *p*{i,j} contains the poles from input j to output i.

k Matrix containing the gains for each channel. *k*(i,j) contains the gain from input j to output i.

tsam Sampling time in seconds. If *sys* is a continuous-time model, a zero is returned.

4 Model Conversions

4.1 @lti/c2d

`sys = c2d (sys, tsam)` [Function File]
`sys = c2d (sys, tsam, method)` [Function File]
`sys = c2d (sys, tsam, "prewarp", w0)` [Function File]

Convert the continuous lti model into its discrete-time equivalent.

Inputs

`sys` Continuous-time LTI model.
`tsam` Sampling time in seconds.
`method` Optional conversion method. If not specified, default method "zoh" is taken.
 "zoh" Zero-order hold or matrix exponential.
 "tustin", "bilin" Bilinear transformation or Tustin approximation.
 "prewarp" Bilinear transformation with pre-warping at frequency `w0`.

Outputs

`sys` Discrete-time LTI model.

4.2 @lti/d2c

`sys = d2c (sys)` [Function File]
`sys = d2c (sys, method)` [Function File]
`sys = d2c (sys, "prewarp", w0)` [Function File]

Convert the discrete lti model into its continuous-time equivalent.

Inputs

`sys` Discrete-time LTI model.
`method` Optional conversion method. If not specified, default method "zoh" is taken.
 "zoh" Zero-order hold or matrix logarithm.
 "tustin", "bilin" Bilinear transformation or Tustin approximation.
 "prewarp" Bilinear transformation with pre-warping at frequency `w0`.

Outputs

`sys` Continuous-time LTI model.

4.3 @lti/prescale

`[scaledsys, info] = prescale (sys)` [Function File]
 Prescale state-space model. Frequency response commands perform automatic scaling unless model property `scaled` is set to `true`.

Inputs

`sys` LTI model.

Outputs

scaledsys Scaled state-space model.

info Structure containing additional information.

info.SL Left scaling factors. $Tl = \text{diag}(\text{info.SL})$.

info.SR Right scaling factors. $Tr = \text{diag}(\text{info.SR})$.

Equations

$$\begin{aligned} Es &= Tl * E * Tr \\ As &= Tl * A * Tr \\ Bs &= Tl * B \\ Cs &= C * Tr \\ Ds &= D \end{aligned}$$

=

For proper state-space models, Tl and Tr are inverse of each other.

Algorithm

Uses SLICOT TB01ID and TG01AD by courtesy of [NICONET e.V.](#)

4.4 @lti/xperm

sys = *xperm* (*sys*, *st_idx*)

Reorder states in state-space models.

[Function File]

5 Model Interconnections

5.1 @lti/append

`sys = append (sys1, sys2)` [Function File]
 Group LTI models by appending their inputs and outputs.

5.2 @lti/blkdiag

`sys = blkdiag (sys1, sys2)` [Function File]
 Block-diagonal concatenation of LTI models.

5.3 @lti/connect

`sys = connect (sys, cm, inputs, outputs)` [Function File]
 Arbitrary interconnections between the inputs and outputs of an LTI model.

5.4 @lti/feedback

`sys = feedback (sys1)` [Function File]
`sys = feedback (sys1, "+")` [Function File]
`sys = feedback (sys1, sys2)` [Function File]
`sys = feedback (sys1, sys2, "+")` [Function File]
`sys = feedback (sys1, sys2, feedin, feedout)` [Function File]
`sys = feedback (sys1, sys2, feedin, feedout, "+")` [Function File]
 Feedback connection of two LTI models.

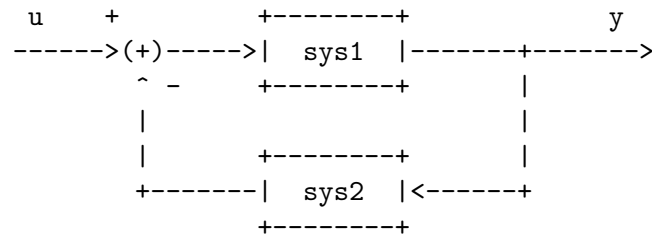
Inputs

sys1 LTI model of forward transmission. `[p1, m1] = size (sys1)`.
sys2 LTI model of backward transmission. If not specified, an identity matrix of appropriate size is taken.
feedin Vector containing indices of inputs to *sys1* which are involved in the feedback loop. The number of *feedin* indices and outputs of *sys2* must be equal. If not specified, `1:m1` is taken.
feedout Vector containing indices of outputs from *sys1* which are to be connected to *sys2*. The number of *feedout* indices and inputs of *sys2* must be equal. If not specified, `1:p1` is taken.
 "+" Positive feedback sign. If not specified, "-" for a negative feedback interconnection is assumed. *+1* and *-1* are possible as well, but only from the third argument onward due to ambiguity.

Outputs

sys Resulting LTI model.

Block Diagram



5.5 @lft/lft

`sys = lft (sys1, sys2)`

[Function File]

`sys = lft (sys1, sys2, nu, ny)`

[Function File]

Linear fractional tranformation, also known as Redheffer star product.

Inputs

`sys1` Upper LTI model.

`sys2` Lower LTI model.

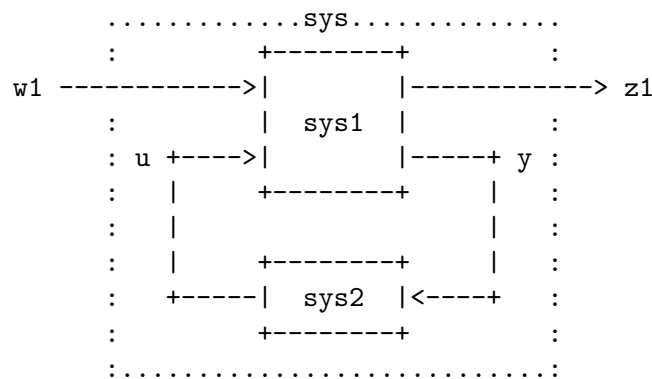
`nu` The last `nu` inputs of `sys1` are connected with the first `nu` outputs of `sys2`. If not specified, `min (m1, p2)` is taken.

`ny` The last `ny` outputs of `sys1` are connected with the first `ny` inputs of `sys2`. If not specified, `min (p1, m2)` is taken.

Outputs

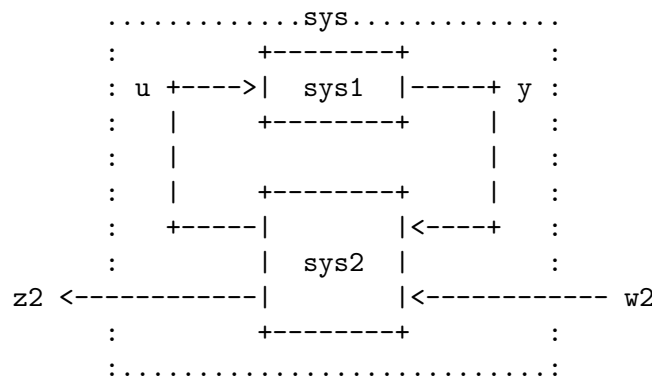
`sys` Resulting LTI model.

Block Diagram



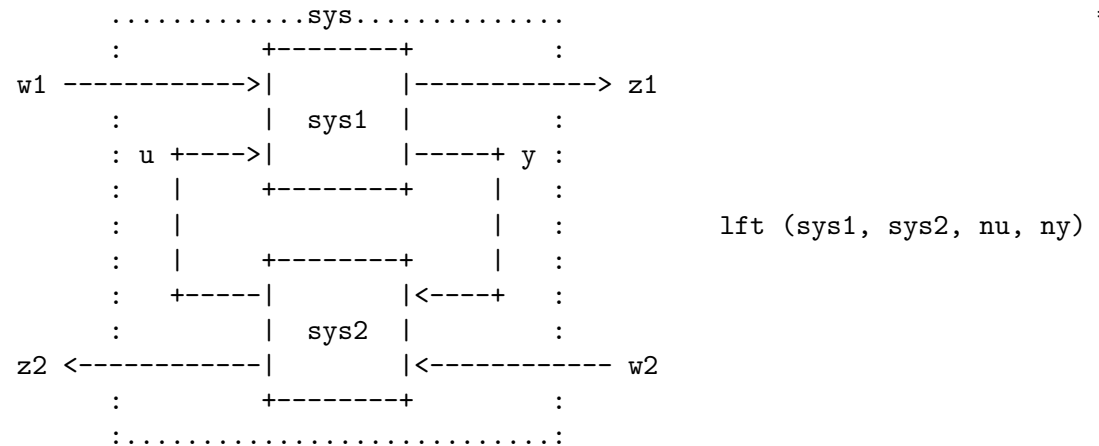
Lower LFT

`lft (sys1, sys2)`



Upper LFT

`lft (sys1, sys2)`



5.6 @lti/mconnect

`sys = mconnect (sys, m)` [Function File]

`sys = mconnect (sys, m, inputs, outputs)` [Function File]

Arbitrary interconnections between the inputs and outputs of an LTI model.

Inputs

`sys` LTI system.

`m` Connection matrix. Each row belongs to an input and each column represents an output.

`inputs` Vector of indices of those inputs which are retained. If not specified, all inputs are kept.

`outputs` Vector of indices of those outputs which are retained. If not specified, all outputs are kept.

Outputs

`sys` Interconnected system.

Example

Solve the system equations of
 $y(t) = G e(t)$
 $e(t) = u(t) + M y(t)$
 in order to build
 $y(t) = H u(t)$
 The matrix M for a (p-by-m) system G
 has m rows and p columns (m-by-p).

Example for a 3x2 system:

$u1 = -1*y1 + 5*y2 + 0*y3$

$u2 = pi*y1 + 0*y2 - 7*y3$

$M = \begin{bmatrix} -1 & 5 & 0 \\ pi & 0 & 7 \end{bmatrix}$

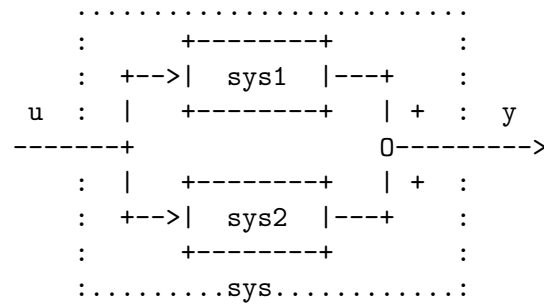
5.7 @lti/parallel

`sys = parallel (sys1, sys2)`

[Function File]

Parallel connection of two LTI systems.

Block Diagram



`sys = parallel (sys1, sys2)`

5.8 @lti/series

`sys = series (sys1, sys2)`

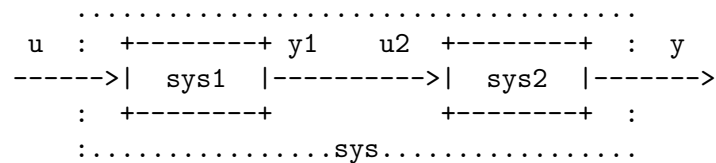
[Function File]

`sys = series (sys1, sys2, outputs1, inputs2)`

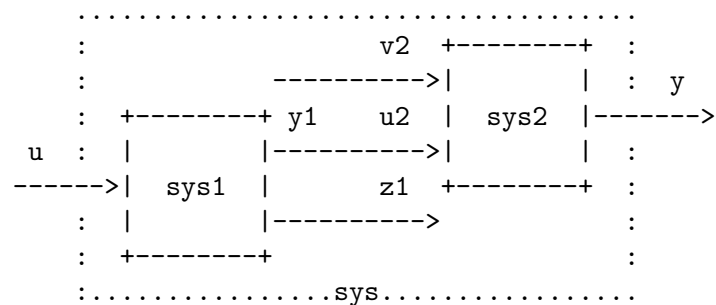
[Function File]

Series connection of two LTI models.

Block Diagram



`sys = series (sys1, sys2)`



`outputs1 = [1]`

`inputs2 = [2]`

`sys = series (sys1, sys2, outputs1, inputs2)`

6 Model Characteristics

6.1 ctrb

`co = ctrb (sys)` [Function File]
`co = ctrb (a, b)` [Function File]

Return controllability matrix.

Inputs

`sys` LTI model.
`a` State transition matrix (n-by-n).
`b` Input matrix (n-by-m).

Outputs

`co` Controllability matrix.

Equation

$$C_o = [B \ AB \ A^2B \ \dots \ A^{n-1}B]$$

6.2 ctrbf

`[sysbar, T, K] = ctrbf (sys)` [Function File]
`[sysbar, T, K] = ctrbf (sys, tol)` [Function File]
`[Abar, Bbar, Cbar, T, K] = ctrbf (A, B, C)` [Function File]
`[Abar, Bbar, Cbar, T, K] = ctrbf (A, B, C, TOL)` [Function File]

If $C_o = \text{ctrb}(A, B)$ has rank $r \leq n = \text{SIZE}(A, 1)$, then there is a similarity transformation T_c such that $T_c = [t_1 \ t_2]$ where t_1 is the controllable subspace and t_2 is orthogonal to t_1

$$Abar = T_c \ A \ T_c^{-1}, \quad Bbar = T_c \ B, \quad Cbar = C \ T_c^{-1} \quad =$$

and the transformed system has the form

$$Abar = \begin{bmatrix} | & A_c & A_{12} & | \\ \hline & & & \\ | & 0 & A_{nc} & | \end{bmatrix}, \quad Bbar = \begin{bmatrix} | & B_c & | \\ \hline & & \\ | & 0 & | \end{bmatrix}, \quad Cbar = [C_c \ | \ C_{nc}]. \quad =$$

where (A_c, B_c) is controllable, and $C_c(sI - A_c)^{-1}B_c = C(sI - A)^{-1}B$. and the system is stabilizable if A_{nc} has no eigenvalues in the right half plane. The last output K is a vector of length n containing the number of controllable states.

6.3 @lti/dcgain

`k = dcgain (sys)` [Function File]
 DC gain of LTI model.

Inputs

`sys` LTI system.

Outputs

`k` DC gain matrix. For a system with m inputs and p outputs, the array k has dimensions $[p, m]$.

`freqresp`

6.4 gram

`W = gram (sys, mode)` [Function File]

`Wc = gram (a, b)` [Function File]

`gram (sys, "c")` returns the controllability gramian of the (continuous- or discrete-time) system `sys`. `gram (sys, "o")` returns the observability gramian of the (continuous- or discrete-time) system `sys`. `gram (a, b)` returns the controllability gramian Wc of the continuous-time system $dx/dt = ax + bu$; i.e., Wc satisfies $aWc + mWc' + bb' = 0$.

6.5 hsvd

`hsv = hsvd (sys)` [Function File]

`hsv = hsvd (sys, "offset", offset)` [Function File]

`hsv = hsvd (sys, "alpha", alpha)` [Function File]

Hankel singular values of the stable part of an LTI model. If no output arguments are given, the Hankel singular values are displayed in a plot.

Algorithm

Uses SLICOT AB13AD by courtesy of [NICONET e.V.](#)

6.6 @lti/isct

`bool = isct (sys)` [Function File]

Determine whether LTI model is a continuous-time system.

Inputs

`sys` LTI system.

Outputs

`bool = 0` `sys` is a discrete-time system.

`bool = 1` `sys` is a continuous-time system or a static gain.

6.7 isctrb

`[bool, ncon] = isctrb (sys)` [Function File]

`[bool, ncon] = isctrb (sys, tol)` [Function File]

`[bool, ncon] = isctrb (a, b)` [Function File]

`[bool, ncon] = isctrb (a, b, e)` [Function File]

`[bool, ncon] = isctrb (a, b, [], tol)` [Function File]

`[bool, ncon] = isctrb (a, b, e, tol)` [Function File]

Logical check for system controllability. For numerical reasons, `isctrb (sys)` should be used instead of `rank (ctrb (sys))`.

Inputs

`sys` LTI model. Descriptor state-space models are possible.

`a` State transition matrix.

`b` Input matrix.

`e` Descriptor matrix.

`tol` Optional roundoff parameter. Default value is 0.

Outputs

`bool = 0` System is not controllable.

bool = 1 System is controllable.
ncon Number of controllable states.

Algorithm

Uses SLICOT AB01OD and TG01HD by courtesy of **NICONET e.V.**

isobsv

6.8 isdetectable

bool = isdetectable (*sys*) [Function File]
bool = isdetectable (*sys*, *tol*) [Function File]
bool = isdetectable (*a*, *c*) [Function File]
bool = isdetectable (*a*, *c*, *e*) [Function File]
bool = isdetectable (*a*, *c*, [], *tol*) [Function File]
bool = isdetectable (*a*, *c*, *e*, *tol*) [Function File]
bool = isdetectable (*a*, *c*, [], [], *dflg*) [Function File]
bool = isdetectable (*a*, *c*, *e*, [], *dflg*) [Function File]
bool = isdetectable (*a*, *c*, [], *tol*, *dflg*) [Function File]
bool = isdetectable (*a*, *c*, *e*, *tol*, *dflg*) [Function File]

Logical test for system detectability. All unstable modes must be observable or all unobservable states must be stable.

Inputs

sys LTI system.
a State transition matrix.
c Measurement matrix.
e Descriptor matrix.
tol Optional tolerance for stability. Default value is 0.
dflg = 0 Matrices (*a*, *c*) are part of a continuous-time system. Default Value.
dflg = 1 Matrices (*a*, *c*) are part of a discrete-time system.

Outputs

bool = 0 System is not detectable.
bool = 1 System is detectable.

Algorithm

Uses SLICOT AB01OD and TG01HD by courtesy of **NICONET e.V.** See `isstabilizable` for description of computational method. `isstabilizable`, `isstable`, `isctrb`, `isobsv`

6.9 @lti/isdt

bool = isdt (*sys*) [Function File]
Determine whether LTI model is a discrete-time system.

Inputs

sys LTI system.

Outputs

bool = 0 *sys* is a continuous-time system.
bool = 1 *sys* is a discrete-time system or a static gain.

6.10 @lti/isminimumphase

`bool = isminimumphase (sys)` [Function File]

`bool = isminimumphase (sys, tol)` [Function File]

Determine whether LTI system is minimum phase. The zeros must lie in the left complex half-plane. The name minimum-phase refers to the fact that such a system has the minimum possible phase lag for the given magnitude response $|\text{sys}(j\omega)|$.

6.11 isobsv

`[bool, nobs] = isobsv (sys)` [Function File]

`[bool, nobs] = isobsv (sys, tol)` [Function File]

`[bool, nobs] = isobsv (a, c)` [Function File]

`[bool, nobs] = isobsv (a, c, e)` [Function File]

`[bool, nobs] = isobsv (a, c, [], tol)` [Function File]

`[bool, nobs] = isobsv (a, c, e, tol)` [Function File]

Logical check for system observability. For numerical reasons, `isobsv (sys)` should be used instead of `rank (obsv (sys))`.

Inputs

`sys` LTI model. Descriptor state-space models are possible.

`a` State transition matrix.

`c` Measurement matrix.

`e` Descriptor matrix.

`tol` Optional roundoff parameter. Default value is 0.

Outputs

`bool = 0` System is not observable.

`bool = 1` System is observable.

`nobs` Number of observable states.

Algorithm

Uses SLICOT AB01OD and TG01HD by courtesy of [NICONET e.V.](#)

isctrb

6.12 @lti/issiso

`bool = issiso (sys)` [Function File]

Determine whether LTI model is single-input/single-output (SISO).

6.13 isstabilizable

`bool = isstabilizable (sys)` [Function File]

`bool = isstabilizable (sys, tol)` [Function File]

`bool = isstabilizable (a, b)` [Function File]

`bool = isstabilizable (a, b, e)` [Function File]

`bool = isstabilizable (a, b, [], tol)` [Function File]

`bool = isstabilizable (a, b, e, tol)` [Function File]

`bool = isstabilizable (a, b, [], [], dflg)` [Function File]

`bool = isstabilizable (a, b, e, [], dflg)` [Function File]

```
bool = isstabilizable (a, b, [], tol, dflg) [Function File]
bool = isstabilizable (a, b, e, tol, dflg) [Function File]
```

Logical check for system stabilizability. All unstable modes must be controllable or all uncontrollable states must be stable.

Inputs

sys LTI system.

a State transition matrix.

b Input matrix.

e Descriptor matrix.

tol Optional tolerance for stability. Default value is 0.

dflg = 0 Matrices (*a*, *b*) are part of a continuous-time system. Default Value.

dflg = 1 Matrices (*a*, *b*) are part of a discrete-time system.

Outputs

bool = 0 System is not stabilizable.

bool = 1 System is stabilizable.

Algorithm

Uses SLICOT AB01OD and TG01HD by courtesy of [NICONET e.V.](#)

```
* Calculate staircase form (SLICOT AB01OD)
* Extract unobservable part of state transition matrix
* Calculate eigenvalues of unobservable part
* Check whether
  real (ev) < -tol*(1 + abs (ev))    continuous-time
  abs (ev) < 1 - tol                 discrete-time
```

isdetectable, isstable, isctrb, isobsv

6.14 @lti/isstable

```
bool = isstable (sys) [Function File]
bool = isstable (sys, tol) [Function File]
```

Determine whether LTI system is stable.

6.15 @lti/norm

```
gain = norm (sys, 2) [Function File]
[gain, wpeak] = norm (sys, inf) [Function File]
[gain, wpeak] = norm (sys, inf, tol) [Function File]
```

Return H-2 or L-inf norm of LTI model.

Algorithm

Uses SLICOT AB13BD and AB13DD by courtesy of [NICONET e.V.](#)

6.16 obsv

`ob = obsv (sys)` [Function File]
`ob = obsv (a, c)` [Function File]

Return observability matrix.

Inputs

`sys` LTI model.
`a` State transition matrix (n-by-n).
`c` Measurement matrix (p-by-n).

Outputs

`ob` Observability matrix.

Equation

$$O_b = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

6.17 obsvf

`[sysbar, T, K] = obsvf (sys)` [Function File]
`[sysbar, T, K] = obsvf (sys, tol)` [Function File]
`[Abar, Bbar, Cbar, T, K] = obsvf (A, B, C)` [Function File]
`[Abar, Bbar, Cbar, T, K] = obsvf (A, B, C, TOL)` [Function File]

If $O_b = \text{obsv}(A, C)$ has rank $r \leq n = \text{SIZE}(A, 1)$, then there is a similarity transformation T_c such that $T_o = [t_1; t_2]$ where t_1 is c and t_2 is orthogonal to t_1

$$Abar = T_o \ A \ T_o^{-1}, \quad Bbar = T_o \ B, \quad Cbar = C \ T_o \quad =$$

and the transformed system has the form

$$Abar = \begin{bmatrix} | & A_o & 0 & | \\ \hline & & & \\ | & A_{21} & A_{no} & | \end{bmatrix}, \quad Bbar = \begin{bmatrix} | & B_o & | \\ \hline & & \\ | & B_{no} & | \end{bmatrix}, \quad Cbar = [C_o \ | \ 0]. \quad =$$

where (A_o, B_o) is observable, and $C_o(sI - A_o)^{-1}B_o = C(sI - A)^{-1}B$. And system is detectable if A_{no} has no eigenvalues in the right half plane. The last output K is a vector of length n containing the number of observable states.

6.18 @lti/pole

`p = pole (sys)` [Function File]
 Compute poles of LTI system.

Inputs

`sys` LTI model.

Outputs

`p` Poles of `sys`.

6.19 pzmap

`pzmap (sys)` [Function File]

`[p, z] = pzmap (sys)` [Function File]

Plot the poles and zeros of an LTI system in the complex plane. If no output arguments are given, the result is plotted on the screen. Otherwise, the poles and zeros are computed and returned.

Inputs

sys LTI model.

Outputs

p Poles of *sys*.

z Transmission zeros of *sys*.

6.20 @lti/size

`nvec = size (sys)` [Function File]

`n = size (sys, dim)` [Function File]

`[p, m] = size (sys)` [Function File]

LTI model size, i.e. number of outputs and inputs.

Inputs

sys LTI system.

dim If given a second argument, `size` will return the size of the corresponding dimension.

Outputs

nvec Row vector. The first element is the number of outputs (rows) and the second element the number of inputs (columns).

n Scalar value. The size of the dimension *dim*.

p Number of outputs.

m Number of inputs.

6.21 @lti/zero

`z = zero (sys)` [Function File]

`[z, k] = zero (sys)` [Function File]

Compute transmission zeros and gain of LTI model.

Inputs

sys LTI model.

Outputs

z Transmission zeros of *sys*.

k Gain of *sys*.

7 Model Simplification

7.1 @lti/minreal

`sys = minreal (sys)` [Function File]
`sys = minreal (sys, tol)` [Function File]

Minimal realization or zero-pole cancellation of LTI models.

7.2 @lti/sminreal

`sys = sminreal (sys)` [Function File]
`sys = sminreal (sys, tol)` [Function File]

Perform state-space model reduction based on structure. Remove states which have no influence on the input-output behaviour. The physical meaning of the states is retained.

Inputs

`sys` State-space model.
`tol` Optional tolerance for controllability and observability. Entries of the state-space matrices whose moduli are less or equal to `tol` are assumed to be zero. Default value is 0.

Outputs

`sys` Reduced state-space model.
`minreal`

8 Time Domain Analysis

8.1 covar

`[p, q] = covar (sys, w)` [Function File]
 Return the steady-state covariance.

Inputs

`sys` LTI model.
`w` Intensity of white noise inputs which drive `sys`.

Outputs

`p` Output covariance.
`q` State covariance.
`lyap, dlyap`

8.2 gensig

`[u, t] = gensig (sigtype, tau)` [Function File]
`[u, t] = gensig (sigtype, tau, tfinal)` [Function File]
`[u, t] = gensig (sigtype, tau, tfinal, tsam)` [Function File]
 Generate periodic signal. Useful in combination with `lsim`.

Inputs

`sigtype = "sin"`
 Sine wave.
`sigtype = "cos"`
 Cosine wave.
`sigtype = "square"`
 Square wave.
`sigtype = "pulse"`
 Periodic pulse.
`tau` Duration of one period in seconds.
`tfinal` Optional duration of the signal in seconds. Default duration is 5 periods.
`tsam` Optional sampling time in seconds. Default spacing is `tau/64`.

Outputs

`u` Vector of signal values.
`t` Time vector of the signal.
`lsim`

8.3 impulse

```
[y, t, x] = impulse (sys) [Function File]
[y, t, x] = impulse (sys, t) [Function File]
[y, t, x] = impulse (sys, tfinal) [Function File]
[y, t, x] = impulse (sys, tfinal, dt) [Function File]
```

Impulse response of LTI system. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI model.

t Time vector. Should be evenly spaced. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

tfinal Optional simulation horizon. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

dt Optional sampling time. Be sure to choose it small enough to capture transient phenomena. If not specified, it is calculated by the poles of the system.

Outputs

y Output response array. Has as many rows as time samples (length of *t*) and as many columns as outputs.

t Time row vector.

x State trajectories array. Has **length (t)** rows and as many columns as states.

initial, lsim, step

8.4 initial

```
[y, t, x] = initial (sys, x0) [Function File]
[y, t, x] = initial (sys, x0, t) [Function File]
[y, t, x] = initial (sys, x0, tfinal) [Function File]
[y, t, x] = initial (sys, x0, tfinal, dt) [Function File]
```

Initial condition response of state-space model. If no output arguments are given, the response is printed on the screen.

Inputs

sys State-space model.

x0 Vector of initial conditions for each state.

t Optional time vector. Should be evenly spaced. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

tfinal Optional simulation horizon. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

dt Optional sampling time. Be sure to choose it small enough to capture transient phenomena. If not specified, it is calculated by the poles of the system.

Outputs

y Output response array. Has as many rows as time samples (length of *t*) and as many columns as outputs.

t Time row vector.

x State trajectories array. Has **length** (t) rows and as many columns as states.

Example

Continuous Time: $\dot{x} = A x$, $y = C x$, $x(0) = x0$

Discrete Time: $x[k+1] = A x[k]$, $y[k] = C x[k]$, $x[0] = x0$

impulse, lsim, step

8.5 lsim

`[y, t, x] = lsim (sys, u)` [Function File]
`[y, t, x] = lsim (sys, u, t)` [Function File]
`[y, t, x] = lsim (sys, u, t, x0)` [Function File]
`[y, t, x] = lsim (sys, u, t, [], method)` [Function File]
`[y, t, x] = lsim (sys, u, t, x0, method)` [Function File]

Simulate LTI model response to arbitrary inputs. If no output arguments are given, the system response is plotted on the screen.

Inputs

sys LTI model. System must be proper, i.e. it must not have more zeros than poles.
 u Vector or array of input signal. Needs **length**(t) rows and as many columns as there are inputs. If sys is a single-input system, row vectors u of length **length**(t) are accepted as well.
 t Time vector. Should be evenly spaced. If sys is a continuous-time system and t is a real scalar, sys is discretized with sampling time $tsam = t / (rows(u) - 1)$. If sys is a discrete-time system and t is not specified, vector t is assumed to be $0 : tsam : tsam * (rows(u) - 1)$.
 $x0$ Vector of initial conditions for each state. If not specified, a zero vector is assumed.
 $method$ Discretization method for continuous-time models. Default value is `zoh` (zero-order hold). All methods from `c2d` are supported.

Outputs

y Output response array. Has as many rows as time samples (length of t) and as many columns as outputs.
 t Time row vector. It is always evenly spaced.
 x State trajectories array. Has **length** (t) rows and as many columns as states.
 impulse, initial, step

8.6 step

`[y, t, x] = step (sys)` [Function File]
`[y, t, x] = step (sys, t)` [Function File]
`[y, t, x] = step (sys, tfinal)` [Function File]
`[y, t, x] = step (sys, tfinal, dt)` [Function File]

Step response of LTI system. If no output arguments are given, the response is printed on the screen.

Inputs

<i>sys</i>	LTI model.
<i>t</i>	Time vector. Should be evenly spaced. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.
<i>tfinal</i>	Optional simulation horizon. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.
<i>dt</i>	Optional sampling time. Be sure to choose it small enough to capture transient phenomena. If not specified, it is calculated by the poles of the system.

Outputs

<i>y</i>	Output response array. Has as many rows as time samples (length of <i>t</i>) and as many columns as outputs.
<i>t</i>	Time row vector.
<i>x</i>	State trajectories array. Has length (t) rows and as many columns as states.

impulse, initial, lsim

9 Frequency Domain Analysis

9.1 bode

`[mag, pha, w] = bode (sys)` [Function File]

`[mag, pha, w] = bode (sys, w)` [Function File]

Bode diagram of frequency response. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI system. Must be a single-input and single-output (SISO) system.

w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system.

Outputs

mag Vector of magnitude. Has length of frequency vector *w*.

pha Vector of phase. Has length of frequency vector *w*.

w Vector of frequency values used.

nichols, nyquist, sigma

9.2 bodemag

`[mag, w] = bodemag (sys)` [Function File]

`[mag, w] = bodemag (sys, w)` [Function File]

Bode magnitude diagram of frequency response. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI system. Must be a single-input and single-output (SISO) system.

w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system.

Outputs

mag Vector of magnitude. Has length of frequency vector *w*.

w Vector of frequency values used.

bode, nichols, nyquist, sigma

9.3 @lti/freqresp

`H = freqresp (sys, w)` [Function File]

Evaluate frequency response at given frequencies.

Inputs

sys LTI system.

w Vector of frequency values.

Outputs

H Array of frequency response. For a system with *m* inputs and *p* outputs, the array *H* has dimensions [*p*, *m*, length (*w*)]. The frequency response at the frequency *w*(*k*) is given by *H*(:,:,*k*).

dcgain

9.4 margin

`[gamma, phi, w_gamma, w_phi] = margin (sys)` [Function File]

`[gamma, phi, w_gamma, w_phi] = margin (sys, tol)` [Function File]

Gain and phase margin of a system. If no output arguments are given, both gain and phase margin are plotted on a bode diagram. Otherwise, the margins and their corresponding frequencies are computed and returned.

Inputs

sys LTI model. Must be a single-input and single-output (SISO) system.

tol Imaginary parts below *tol* are assumed to be zero. If not specified, default value `sqrt (eps)` is taken.

Outputs

gamma Gain margin (as gain, not dBs).

phi Phase margin (in degrees).

w_gamma Frequency for the gain margin (in rad/s).

w_phi Frequency for the phase margin (in rad/s).

Equations

CONTINUOUS SYSTEMS

Gain Margin

$$L(j\omega) = \bar{L}(j\omega) \quad \text{BTW: } \bar{L}(j\omega) = L(-j\omega) = \text{conj} (L(j\omega))$$

$$\frac{\text{num}(j\omega)}{\text{den}(j\omega)} = \frac{\text{num}(-j\omega)}{\text{den}(-j\omega)}$$

$$\text{num}(j\omega) \text{ den}(-j\omega) = \text{num}(-j\omega) \text{ den}(j\omega)$$

$$\text{imag} (\text{num}(j\omega) \text{ den}(-j\omega)) = 0$$

$$\text{imag} (\text{num}(-j\omega) \text{ den}(j\omega)) = 0$$

Phase Margin

$$|L(j\omega)| = \frac{|\text{num}(j\omega)|}{|\text{den}(j\omega)|} = 1$$

$$z^2 = \text{Re } z + j \text{Im } z$$

$$\frac{\text{num}(j\omega)}{\text{den}(j\omega)} * \frac{\text{num}(-j\omega)}{\text{den}(-j\omega)} = 1$$

$$\text{num}(j\omega) \text{ num}(-j\omega) - \text{den}(j\omega) \text{ den}(-j\omega) = 0$$

$$\text{real} (\text{num}(j\omega) \text{ num}(-j\omega) - \text{den}(j\omega) \text{ den}(-j\omega)) = 0$$

DISCRETE SYSTEMS

Gain Margin

$$L(z) = L(1/z) \quad \text{BTW: } z = e^{j\omega T} \quad \rightarrow \quad \omega = \frac{\log z}{jT}$$

$$\frac{\text{num}(z)}{\text{den}(z)} = \frac{\text{num}(1/z)}{\text{den}(1/z)}$$

$$\text{num}(z) \text{den}(1/z) - \text{num}(1/z) \text{den}(z) = 0$$

Phase Margin

$$|L(z)| = \frac{|\text{num}(z)|}{|\text{den}(z)|} = 1$$

$$L(z) L(1/z) = 1$$

$$\frac{\text{num}(z)}{\text{den}(z)} * \frac{\text{num}(1/z)}{\text{den}(1/z)} = 1$$

$$\text{num}(z) \text{num}(1/z) - \text{den}(z) \text{den}(1/z) = 0$$

PS: How to get $L(1/z)$

$$p(z) = a z^4 + b z^3 + c z^2 + d z + e$$

$$p(1/z) = a z^{-4} + b z^{-3} + c z^{-2} + d z^{-1} + e$$

$$= z^{-4} (a + b z + c z^2 + d z^3 + e z^4)$$

$$= (e z^4 + d z^3 + c z^2 + b z + a) / (z^4)$$

roots

9.5 nichols

`[mag, pha, w] = nichols(sys)` [Function File]

`[mag, pha, w] = nichols(sys, w)` [Function File]

Nichols chart of frequency response. If no output arguments are given, the response is printed on the screen.

Inputs

`sys` LTI system. Must be a single-input and single-output (SISO) system.

`w` Optional vector of frequency values. If `w` is not specified, it is calculated by the zeros and poles of the system.

Outputs

mag Vector of magnitude. Has length of frequency vector *w*.
pha Vector of phase. Has length of frequency vector *w*.
w Vector of frequency values used.
 bode, nyquist, sigma

9.6 nyquist

`[re, im, w] = nyquist (sys)` [Function File]
`[re, im, w] = nyquist (sys, w)` [Function File]

Nyquist diagram of frequency response. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI system. Must be a single-input and single-output (SISO) system.
w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system.

Outputs

re Vector of real parts. Has length of frequency vector *w*.
im Vector of imaginary parts. Has length of frequency vector *w*.
w Vector of frequency values used.
 bode, nichols, sigma

9.7 sigma

`[sv, w] = sigma (sys)` [Function File]
`[sv, w] = sigma (sys, w)` [Function File]
`[sv, w] = sigma (sys, [], ptype)` [Function File]
`[sv, w] = sigma (sys, w, ptype)` [Function File]

Singular values of frequency response. If no output arguments are given, the singular value plot is printed on the screen;

Inputs

sys LTI system. Multiple inputs and/or outputs (MIMO systems) make practical sense.
w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system.
ptype = 0 Singular values of the frequency response *H* of system *sys*. Default Value.
ptype = 1 Singular values of the frequency response *inv(H)*; i.e. inversed system.
ptype = 2 Singular values of the frequency response *I + H*; i.e. inversed sensitivity (or return difference) if *H* = *P * C*.
ptype = 3 Singular values of the frequency response *I + inv(H)*; i.e. inversed complementary sensitivity if *H* = *P * C*.

Outputs

sv Array of singular values. For a system with *m* inputs and *p* outputs, the array *sv* has `min (m, p)` rows and as many columns as frequency points `length (w)`. The singular values at the frequency *w(k)* are given by `sv (:,k)`.

w Vector of frequency values used.

bodemag, svd

10 Pole Placement

10.1 place

`f = place (sys, p)` [Function File]
`f = place (a, b, p)` [Function File]
`[f, info] = place (sys, p, alpha)` [Function File]
`[f, info] = place (a, b, p, alpha)` [Function File]

Pole assignment for a given matrix pair (A, B) such that $p = \text{eig}(A - B * F)$. If parameter *alpha* is specified, poles with real parts (continuous-time) or moduli (discrete-time) below *alpha* are left untouched.

Inputs

sys LTI system.
a State transition matrix (n-by-n) of a continuous-time system.
b Input matrix (n-by-m) of a continuous-time system.
p Desired eigenvalues of the closed-loop system state-matrix $A - B * F$. `length(p) <= rows(A)`.
alpha Specifies the maximum admissible value, either for real parts or for moduli, of the eigenvalues of A which will not be modified by the eigenvalue assignment algorithm. `alpha >= 0` for discrete-time systems.

Outputs

f State feedback gain matrix.
info Structure containing additional information.
info.nfp The number of fixed poles, i.e. eigenvalues of A having real parts less than *alpha*, or moduli less than *alpha*. These eigenvalues are not modified by `place`.
info.nap The number of assigned eigenvalues. `nap = n - nfp - nup`.
info.nup The number of uncontrollable eigenvalues detected by the eigenvalue assignment algorithm.
info.z The orthogonal matrix z reduces the closed-loop system state matrix $A + B * F$ to upper real Schur form. Note the positive sign in $A + B * F$.

Note

Place is also suitable to design estimator gains:

```

L = place (A.', C.', p).';
L = place (sys.', p).'; # useful for discrete-time systems

```

Algorithm

Uses SLICOT SB01BD by courtesy of [NICONET e.V.](#)

10.2 rlocus

`rlocus (sys)` [Function File]

`[rldata, k] = rlocus (sys, increment, min_k, max_k)` [Function File]

Display root locus plot of the specified SISO system.

Inputs

sys LTI model. Must be a single-input and single-output (SISO) system.

min_k Minimum value of k .

max_k Maximum value of k .

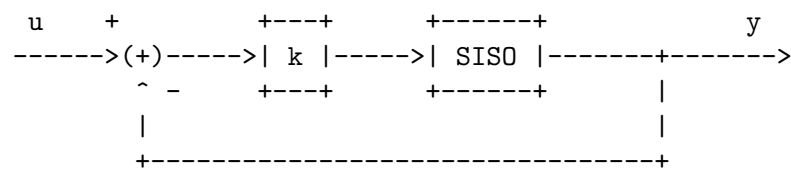
increment The increment used in computing gain values.

Outputs

rldata Data points plotted: in column 1 real values, in column 2 the imaginary values.

k Gains for real axis break points.

Block Diagram



11 Linear-Quadratic Control

11.1 dlqr

<code>[g, x, l] = dlqr (sys, q, r)</code>	[Function File]
<code>[g, x, l] = dlqr (sys, q, r, s)</code>	[Function File]
<code>[g, x, l] = dlqr (a, b, q, r)</code>	[Function File]
<code>[g, x, l] = dlqr (a, b, q, r, s)</code>	[Function File]
<code>[g, x, l] = dlqr (a, b, q, r, [], e)</code>	[Function File]
<code>[g, x, l] = dlqr (a, b, q, r, s, e)</code>	[Function File]

Linear-quadratic regulator for discrete-time systems.

Inputs

sys Continuous or discrete-time LTI model.

a State transition matrix of discrete-time system.

b Input matrix of discrete-time system.

q State weighting matrix.

r Input weighting matrix.

s Optional cross term matrix. If *s* is not specified, a zero matrix is assumed.

e Optional descriptor matrix. If *e* is not specified, an identity matrix is assumed.

Outputs

g State feedback matrix.

x Unique stabilizing solution of the discrete-time Riccati equation.

l Closed-loop poles.

Equations

$$x[k+1] = A x[k] + B u[k], \quad x[0] = x_0$$

$$J(x_0) = \sum_{k=0}^{\infty} (x' Q x + u' R u + 2 x' S u)$$

$$L = \text{eig} (A - B \cdot G)$$

dare, care, lqr

11.2 estim

<code>est = estim (sys, l)</code>	[Function File]
<code>est = estim (sys, l, sensors, known)</code>	[Function File]

Return state estimator for a given estimator gain.

Inputs

sys LTI model.

l State feedback matrix.

sensors Indices of measured output signals *y* from *sys*. If omitted, all outputs are measured.

known Indices of known input signals *u* (deterministic) to *sys*. All other inputs to *sys* are assumed stochastic. If argument *known* is omitted, no inputs *u* are known.

Outputs

est State-space model of estimator.

kalman, *place*

11.3 kalman

`[est, g, x] = kalman (sys, q, r)` [Function File]
`[est, g, x] = kalman (sys, q, r, s)` [Function File]
`[est, g, x] = kalman (sys, q, r, [], sensors, known)` [Function File]
`[est, g, x] = kalman (sys, q, r, s, sensors, known)` [Function File]

Design Kalman estimator for LTI systems.

Inputs

sys Nominal plant model.

q Covariance of white process noise.

r Covariance of white measurement noise.

s Optional cross term covariance. Default value is 0.

sensors Indices of measured output signals *y* from *sys*. If omitted, all outputs are measured.

known Indices of known input signals *u* (deterministic) to *sys*. All other inputs to *sys* are assumed stochastic. If argument *known* is omitted, no inputs *u* are known.

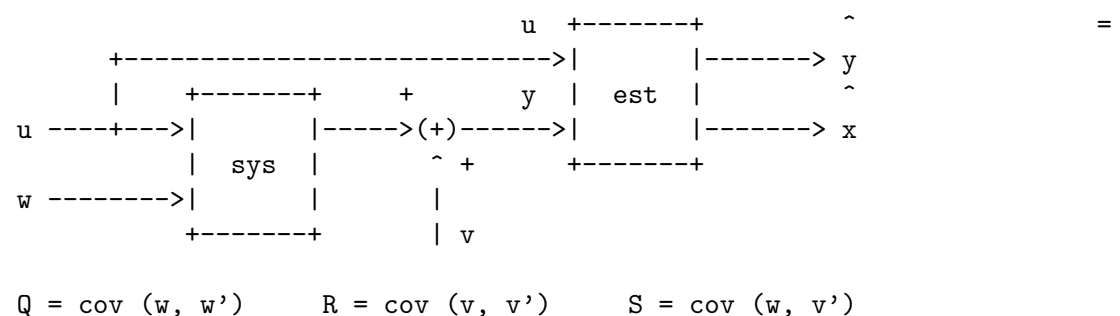
Outputs

est State-space model of the Kalman estimator.

g Estimator gain.

x Solution of the Riccati equation.

Block Diagram



care, *dare*, *estim*, *lqr*

11.4 lqr

<code>[g, x, l] = lqr (sys, q, r)</code>	[Function File]
<code>[g, x, l] = lqr (sys, q, r, s)</code>	[Function File]
<code>[g, x, l] = lqr (a, b, q, r)</code>	[Function File]
<code>[g, x, l] = lqr (a, b, q, r, s)</code>	[Function File]
<code>[g, x, l] = lqr (a, b, q, r, [], e)</code>	[Function File]
<code>[g, x, l] = lqr (a, b, q, r, s, e)</code>	[Function File]

Linear-quadratic regulator.

Inputs

<code>sys</code>	Continuous or discrete-time LTI model.
<code>a</code>	State transition matrix of continuous-time system.
<code>b</code>	Input matrix of continuous-time system.
<code>q</code>	State weighting matrix.
<code>r</code>	Input weighting matrix.
<code>s</code>	Optional cross term matrix. If <code>s</code> is not specified, a zero matrix is assumed.
<code>e</code>	Optional descriptor matrix. If <code>e</code> is not specified, an identity matrix is assumed.

Outputs

<code>g</code>	State feedback matrix.
<code>x</code>	Unique stabilizing solution of the continuous-time Riccati equation.
<code>l</code>	Closed-loop poles.

Equations

$$\dot{\mathbf{x}} = \mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{u}, \quad \mathbf{x}(0) = \mathbf{x}_0$$

$$J(\mathbf{x}_0) = \int_0^{\infty} (\mathbf{x}' \mathbf{Q} \mathbf{x} + \mathbf{u}' \mathbf{R} \mathbf{u} + 2 \mathbf{x}' \mathbf{S} \mathbf{u}) \, dt$$

$$\mathbf{L} = \text{eig} (\mathbf{A} - \mathbf{B} \mathbf{G})$$

`care`, `dare`, `dlqr`

12 Robust Control

12.1 augw

$P = \text{augw}(G, W1, W2, W3)$ [Function File]

Extend plant for stacked S/KS/T problem. Subsequently, the robust control problem can be solved by h2syn or hinfsyn.

Inputs

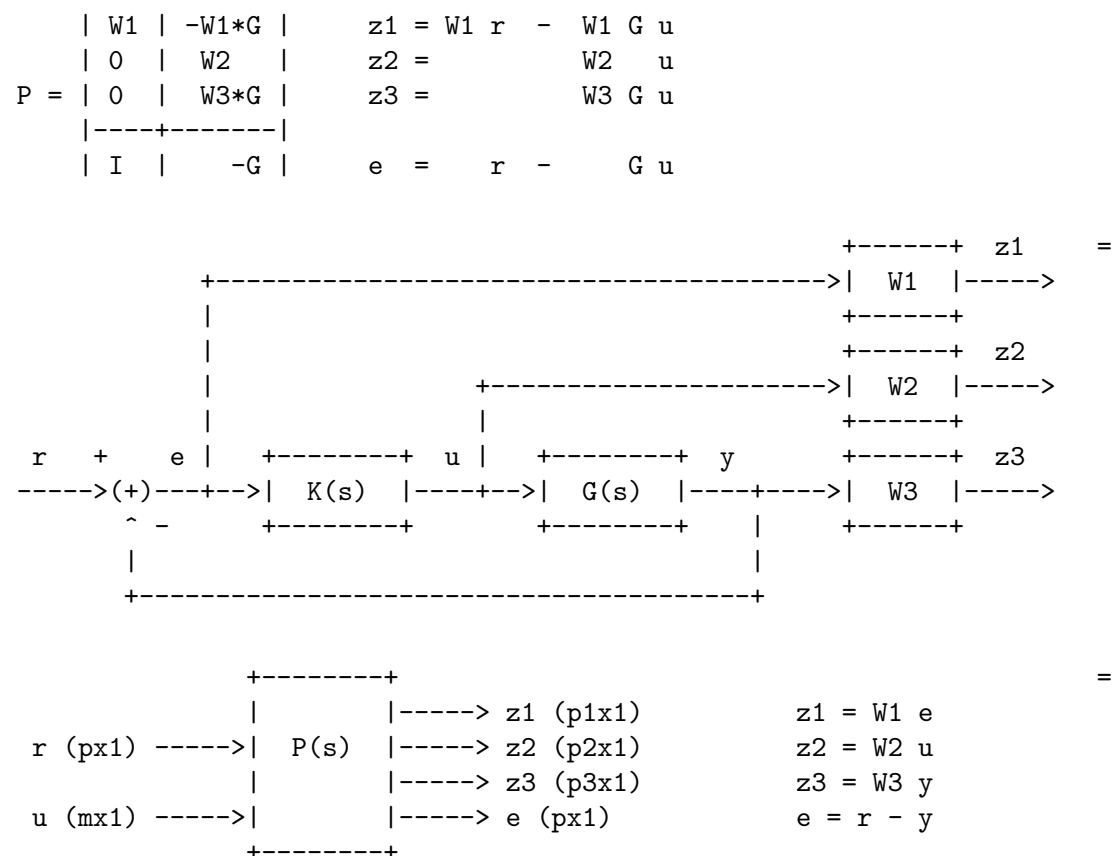
- G LTI model of plant.
- $W1$ LTI model of performance weight. Bounds the largest singular values of sensitivity S . Model must be empty [], SISO or of appropriate size.
- $W2$ LTI model to penalize large control inputs. Bounds the largest singular values of KS . Model must be empty [], SISO or of appropriate size.
- $W3$ LTI model of robustness and noise sensitivity weight. Bounds the largest singular values of complementary sensitivity T . Model must be empty [], SISO or of appropriate size.

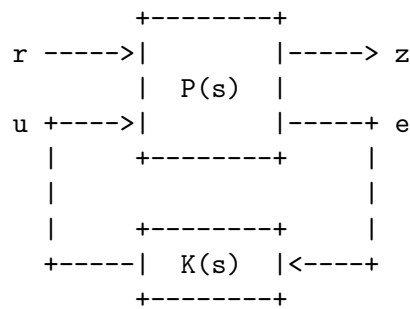
All inputs must be proper/realizable. Scalars, vectors and matrices are possible instead of LTI models.

Outputs

P State-space model of augmented plant.

Block Diagram





Reference:

Skogestad, S. and Postlethwaite I.

Multivariable Feedback Control: Analysis and Design

Second Edition

Wiley 2005

Chapter 3.8: General Control Problem Formulation

h2syn, hinfyn, mixsyn

12.2 h2syn

`[K, N, gamma, rcond] = h2syn (P, nmeas, ncon)`

[Function File]

H-2 control synthesis for LTI plant.

Inputs

P Generalized plant. Must be a proper/realizable LTI model.

nmeas Number of measured outputs *v*. The last *nmeas* outputs of *P* are connected to the inputs of controller *K*. The remaining outputs *z* (indices 1 to *p*-*nmeas*) are used to calculate the H-2 norm.

ncon Number of controlled inputs *u*. The last *ncon* inputs of *P* are connected to the outputs of controller *K*. The remaining inputs *w* (indices 1 to *m*-*ncon*) are excited by a harmonic test signal.

Outputs

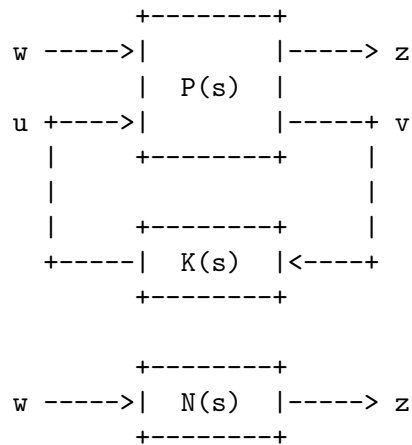
K State-space model of the H-2 optimal controller.

N State-space model of the lower LFT of *P* and *K*.

gamma H-2 norm of *N*.

Block Diagram

$$\gamma = \min_K \|N(K)\|_2 \quad N = \text{lft}(P, K)$$



Algorithm

Uses SLICOT SB10HD and SB10ED by courtesy of **NICONET e.V.**

augw, lqr, dlqr, kalman

12.3 hinfsyn

$[K, N, \gamma, rcond] = \text{hinfsyn}(P, nmeas, ncon)$ [Function File]

$[K, N, \gamma, rcond] = \text{hinfsyn}(P, nmeas, ncon, gmax)$ [Function File]

H-infinity control synthesis for LTI plant.

Inputs

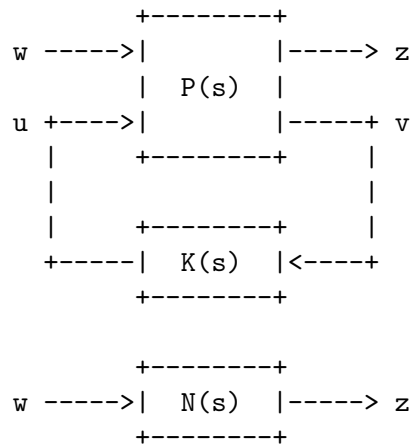
- P Generalized plant. Must be a proper/realizable LTI model.
- $nmeas$ Number of measured outputs v . The last $nmeas$ outputs of P are connected to the inputs of controller K . The remaining outputs z (indices 1 to $p-nmeas$) are used to calculate the H-infinity norm.
- $ncon$ Number of controlled inputs u . The last $ncon$ inputs of P are connected to the outputs of controller K . The remaining inputs w (indices 1 to $m-ncon$) are excited by a harmonic test signal.
- $gmax$ The maximum value of the H-infinity norm of N . It is assumed that $gmax$ is sufficiently large so that the controller is admissible.

Outputs

- K State-space model of the H-infinity (sub-)optimal controller.
- N State-space model of the lower LFT of P and K .
- γ L-infinity norm of N .

Block Diagram

$$\gamma = \min_K \|N(K)\| \quad N = \text{lft}(P, K)$$

**Algorithm**

Uses SLICOT SB10FD and SB10DD by courtesy of [NICONET e.V.](#)

augw, mixsyn

12.4 mixsyn

`[K, N, gamma] = mixsyn (G, W1, W2, W3, ...)` [Function File]

Solve stacked S/KS/T H-infinity problem. Bound the largest singular values of S (for performance), $K S$ (to penalize large inputs) and T (for robustness and to avoid sensitivity to noise). In other words, the inputs r are excited by a harmonic test signal. Then the algorithm tries to find a controller K which minimizes the H-infinity norm calculated from the outputs z .

Inputs

- G LTI model of plant.
- $W1$ LTI model of performance weight. Bounds the largest singular values of sensitivity S . Model must be empty `[]`, SISO or of appropriate size.
- $W2$ LTI model to penalize large control inputs. Bounds the largest singular values of $K S$. Model must be empty `[]`, SISO or of appropriate size.
- $W3$ LTI model of robustness and noise sensitivity weight. Bounds the largest singular values of complementary sensitivity T . Model must be empty `[]`, SISO or of appropriate size.
- `...` Optional arguments of `hinfsyn`. Type `help hinfsyn` for more information.

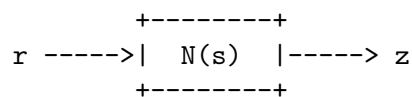
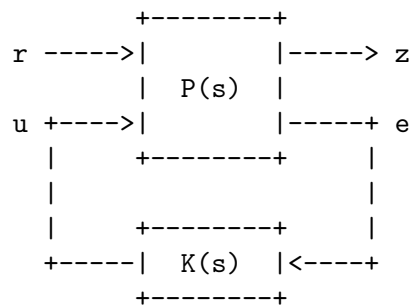
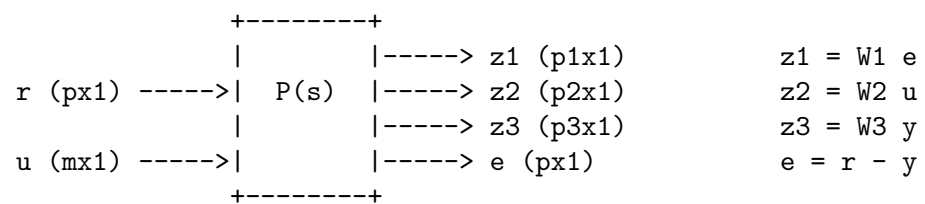
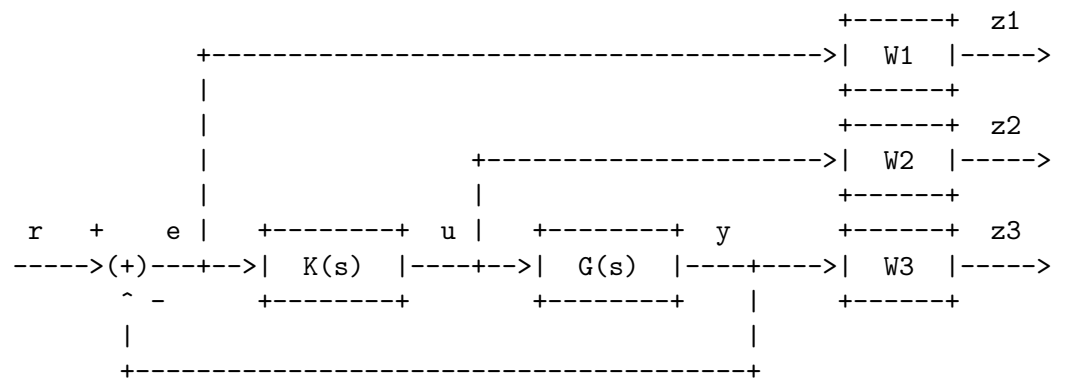
All inputs must be proper/realizable. Scalars, vectors and matrices are possible instead of LTI models.

Outputs

- K State-space model of the H-infinity (sub-)optimal controller.
- N State-space model of the lower LFT of P and K .
- γ L-infinity norm of N .

Block Diagram

$$\gamma = \min_K \|N(K)\| \quad N = \begin{bmatrix} W_1 S \\ W_2 K S \\ W_3 T \end{bmatrix} = \text{lft}(P, K)$$



Extended Plant: $P = \text{augw}(G, W_1, W_2, W_3)$
 Controller: $K = \text{mixsyn}(G, W_1, W_2, W_3)$
 Entire System: $N = \text{lft}(P, K)$
 Open Loop: $L = G * K$
 Closed Loop: $T = \text{feedback}(L)$

Reference:

Skogestad, S. and Postlethwaite I.

Multivariable Feedback Control: Analysis and Design

Second Edition

Wiley 2005

Chapter 3.8: General Control Problem Formulation

hinfsyn, augw

12.5 ncfsyn

`[K, N, gamma, info] = ncfsyn (G, W1, W2, factor)` [Function File]

Loop shaping H-infinity synthesis. Compute positive feedback controller using the McFarlane/Glover normalized coprime factor (NCF) loop shaping design procedure.

Inputs

G LTI model of plant.

W1 LTI model of precompensator. Model must be SISO or of appropriate size. An identity matrix is taken if *W1* is not specified or if an empty model [] is passed.

W2 LTI model of postcompensator. Model must be SISO or of appropriate size. An identity matrix is taken if *W2* is not specified or if an empty model [] is passed.

factor **factor** = 1 implies that an optimal controller is required. **factor** > 1 implies that a suboptimal controller is required, achieving a performance that is *factor* times less than optimal. Default value is 1.

Outputs

K State-space model of the H-infinity loop-shaping controller.

N State-space model of the closed loop depicted below.

gamma L-infinity norm of *N*. **gamma** = **norm** (*N*, **inf**).

info Structure containing additional information.

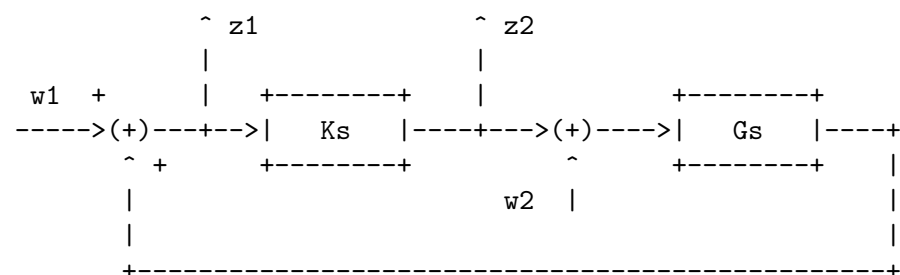
info.emax Nugap robustness. **emax** = **inv** (**gamma**).

info.Gs Shaped plant. **Gs** = **W2** * **G** * **W1**.

info.Ks Controller for shaped plant. **Ks** = **ncfsyn** (**Gs**).

info.rcond Estimates of the reciprocal condition numbers of the Riccati equations.

Block Diagram of N



Algorithm

Uses SLICOT SB10ID, SB10KD and SB10ZD by courtesy of **NICONET e.V.**

13 Matrix Equation Solvers

13.1 care

<code>[x, l, g] = care (a, b, q, r)</code>	[Function File]
<code>[x, l, g] = care (a, b, q, r, s)</code>	[Function File]
<code>[x, l, g] = care (a, b, q, r, [], e)</code>	[Function File]
<code>[x, l, g] = care (a, b, q, r, s, e)</code>	[Function File]

Solve continuous-time algebraic Riccati equation (ARE).

Inputs

<i>a</i>	Real matrix (n-by-n).
<i>b</i>	Real matrix (n-by-m).
<i>q</i>	Real matrix (n-by-n).
<i>r</i>	Real matrix (m-by-m).
<i>s</i>	Optional real matrix (n-by-m). If <i>s</i> is not specified, a zero matrix is assumed.
<i>e</i>	Optional descriptor matrix (n-by-n). If <i>e</i> is not specified, an identity matrix is assumed.

Outputs

<i>x</i>	Unique stabilizing solution of the continuous-time Riccati equation (n-by-n).
<i>l</i>	Closed-loop poles (n-by-1).
<i>g</i>	Corresponding gain matrix (m-by-n).

Equations

$$A'X + XA - XB R^{-1} B'X + Q = 0$$

$$A'X + XA - (XB + S) R^{-1} (B'X + S') + Q = 0$$

$$G = R^{-1} B'X$$

$$G = R^{-1} (B'X + S')$$

$$L = \text{eig} (A - B*G)$$

$$\begin{aligned}
 & A'XE + E'XA - E'XB R^{-1} B'XE + Q = 0 \\
 & A'XE + E'XA - (E'XB + S) R^{-1} (B'XE + S') + Q = 0 \\
 & G = R^{-1} B'XE \\
 & G = R^{-1} (B'XE + S) \\
 & L = \text{eig} (A - B*G, E)
 \end{aligned}$$

Algorithm

Uses SLICOT SB02OD and SG02AD by courtesy of [NICONET e.V.](#)

dare, lqr, dlqr, kalman

13.2 dare

<code>[x, l, g] = dare (a, b, q, r)</code>	[Function File]
<code>[x, l, g] = dare (a, b, q, r, s)</code>	[Function File]
<code>[x, l, g] = dare (a, b, q, r, [], e)</code>	[Function File]
<code>[x, l, g] = dare (a, b, q, r, s, e)</code>	[Function File]

Solve discrete-time algebraic Riccati equation (ARE).

Inputs

<i>a</i>	Real matrix (n-by-n).
<i>b</i>	Real matrix (n-by-m).
<i>q</i>	Real matrix (n-by-n).
<i>r</i>	Real matrix (m-by-m).
<i>s</i>	Optional real matrix (n-by-m). If <i>s</i> is not specified, a zero matrix is assumed.
<i>e</i>	Optional descriptor matrix (n-by-n). If <i>e</i> is not specified, an identity matrix is assumed.

Outputs

<i>x</i>	Unique stabilizing solution of the discrete-time Riccati equation (n-by-n).
<i>l</i>	Closed-loop poles (n-by-1).
<i>g</i>	Corresponding gain matrix (m-by-n).

Equations

$$A'XA - X - A'XB (B'XB + R)^{-1} B'XA + Q = 0 \quad =$$

$$A'XA - X - (A'XB + S) (B'XB + R)^{-1} (B'XA + S') + Q = 0$$

$$G = (B'XB + R)^{-1} B'XA$$

$$G = (B'XB + R)^{-1} (B'XA + S')$$

$$L = \text{eig} (A - B*G)$$

$$A'XA - E'XE - A'XB (B'XB + R)^{-1} B'XA + Q = 0 \quad =$$

$$A'XA - E'XE - (A'XB + S) (B'XB + R)^{-1} (B'XA + S') + Q = 0$$

$$G = (B'XB + R)^{-1} B'XA$$

$$G = (B'XB + R)^{-1} (B'XA + S')$$

$$L = \text{eig} (A - B*G, E)$$

Algorithm

Uses SLICOT SB02OD and SG02AD by courtesy of [NICONET e.V.](#)

care, lqr, dlqr, kalman

13.3 dlyap

$x = \text{dlyap} (a, b)$ [Function File]

$x = \text{dlyap} (a, b, c)$ [Function File]

$x = \text{dlyap} (a, b, [], e)$ [Function File]

Solve discrete-time Lyapunov or Sylvester equations.

Equations

$$AXA' - X + B = 0 \quad (\text{Lyapunov Equation}) \quad =$$

$$AXB' - X + C = 0 \quad (\text{Sylvester Equation})$$

$$AXA' - EXE' + B = 0 \quad (\text{Generalized Lyapunov Equation})$$

Algorithm

Uses SLICOT SB03MD, SB04QD and SG03AD by courtesy of [NICONET e.V.](#)

dlyapchol, lyap, lyapchol

13.4 dlyapchol

`u = dlyapchol (a, b)` [Function File]

`u = dlyapchol (a, b, e)` [Function File]

Compute Cholesky factor of discrete-time Lyapunov equations.

Equations

$$A U' U A' - U' U + B B' = 0 \quad (\text{Lyapunov Equation}) \quad =$$

$$A U' U A' - E U' U E' + B B' = 0 \quad (\text{Generalized Lyapunov Equation})$$

Algorithm

Uses SLICOT SB03OD and SG03BD by courtesy of [NICONET e.V.](#)

`dlyap`, `lyap`, `lyapchol`

13.5 lyap

`x = lyap (a, b)` [Function File]

`x = lyap (a, b, c)` [Function File]

`x = lyap (a, b, [], e)` [Function File]

Solve continuous-time Lyapunov or Sylvester equations.

Equations

$$AX + XA' + B = 0 \quad (\text{Lyapunov Equation}) \quad =$$

$$AX + XB + C = 0 \quad (\text{Sylvester Equation})$$

$$AXE' + EXA' + B = 0 \quad (\text{Generalized Lyapunov Equation})$$

Algorithm

Uses SLICOT SB03MD, SB04MD and SG03AD by courtesy of [NICONET e.V.](#)

`lyapchol`, `dlyap`, `dlyapchol`

13.6 lyapchol

`u = lyapchol (a, b)` [Function File]

`u = lyapchol (a, b, e)` [Function File]

Compute Cholesky factor of continuous-time Lyapunov equations.

Equations

$$A U' U + U' U A' + B B' = 0 \quad (\text{Lyapunov Equation}) \quad =$$

$$A U' U E' + E U' U A' + B B' = 0 \quad (\text{Generalized Lyapunov Equation})$$

Algorithm

Uses SLICOT SB03OD and SG03BD by courtesy of [NICONET e.V.](#)

`lyap`, `dlyap`, `dlyapchol`

14 Overloaded Operators

14.1 @lti/horzcat

Horizontal concatenation of LTI objects. If necessary, object conversion is done by `sys_group`. Used by Octave for "[sys1, sys2]".

14.2 @lti/inv

Inversion of LTI objects.

14.3 @lti/minus

Binary subtraction of LTI objects. If necessary, object conversion is done by `sys_group`. Used by Octave for "sys1 - sys2".

14.4 @lti/mldivide

Matrix left division of LTI objects. If necessary, object conversion is done by `sys_group` in `mtimes`. Used by Octave for "sys1 \ sys2".

14.5 @lti/mpower

Matrix power of LTI objects. The exponent must be an integer. Used by Octave for "sys^{int}".

14.6 @lti/mrdivide

Matrix right division of LTI objects. If necessary, object conversion is done by `sys_group` in `mtimes`. Used by Octave for "sys1 / sys2".

14.7 @lti/mtimes

Matrix multiplication of LTI objects. If necessary, object conversion is done by `sys_group`. Used by Octave for "sys1 * sys2".

14.8 @lti/plus

Binary addition of LTI objects. If necessary, object conversion is done by `sys_group`. Used by Octave for "sys1 + sys2". Operation is also known as "parallel connection".

14.9 @lti/subsasgn

Subscripted assignment for LTI objects. Used by Octave for "sys.property = value".

14.10 @lti/subsref

Subscripted reference for LTI objects. Used by Octave for "sys = sys(2:4, :)" or "val = sys.prop".

14.11 @lti/transpose

Transpose of LTI objects. Used by Octave for "sys.'".

14.12 @lti/uminus

Unary minus of LTI object. Used by Octave for "-sys".

14.13 @lti/vertcat

Vertical concatenation of LTI objects. If necessary, object conversion is done by sys_group. Used by Octave for "[sys1; sys2]".

15 Miscellaneous

15.1 strseq

`strvec = strseq (str, idx)` [Function File]

Return a cell vector of indexed strings by appending the indices *idx* to the string *str*.

```
strseq ("x", 1:3) = {"x1"; "x2"; "x3"}
strseq ("u", [1, 2, 5]) = {"u1"; "u2"; "u5"}
```

15.2 test_control

`test_control` [Script File]

Execute all available tests at once.

15.3 BMWengine

`sys = BMWengine ()` [Function File]

`sys = BMWengine ("scaled")` [Function File]

`sys = BMWengine ("unscaled")` [Function File]

Model of the BMW 4-cylinder engine at ETH Zurich's control laboratory.

OPERATING POINT

Drosselklappenstellung	alpha_DK = 10.3 Grad
Saugrohrdruck	p_s = 0.48 bar
Motordrehzahl	n = 860 U/min
Lambda-Messwert	lambda = 1.000
Relativer Wandfilminhalt	nu = 1

INPUTS

U_1 Sollsignal Drosselklappenstellung	[Grad]
U_2 Relative Einspritzmenge	[-]
U_3 Zuendzeitpunkt	[Grad KW]
M_L Lastdrehmoment	[Nm]

STATES

X_1 Drosselklappenstellung	[Grad]
X_2 Saugrohrdruck	[bar]
X_3 Motordrehzahl	[U/min]
X_4 Messwert Lamba-Sonde	[-]
X_5 Relativer Wandfilminhalt	[-]

OUTPUTS

Y_1 Motordrehzahl	[U/min]
Y_2 Messwert Lambda-Sonde	[-]

```

SCALING
U_1N, X_1N    1 Grad
U_2N, X_4N, X_5N, Y_2N    0.05
U_3N    1.6 Grad KW
X_2N    0.05 bar
X_3N, Y_1N    200 U/min

```

15.4 Boeing707

`sys = Boeing707 ()` [Function File]
 Creates a linearized state-space model of a Boeing 707-321 aircraft at $v=80$ m/s ($M = 0.26$, $G_{a0} = -3^\circ$, $\alpha_0 = 4^\circ$, $\kappa = 50^\circ$).

System inputs: (1) thrust and (2) elevator angle.

System outputs: (1) airspeed and (2) pitch angle.

Reference: R. Brockhaus: *Flugregelung* (Flight Control), Springer, 1994.

15.5 WestlandLynx

`sys = WestlandLynx ()` [Function File]
 Model of the Westland Lynx Helicopter about hover.

```

INPUTS
main rotor collective
longitudinal cyclic
lateral cyclic
tail rotor collective

STATES
pitch attitude      theta      [rad]
roll attitude       phi        [rad]
roll rate (body-axis) p        [rad/s]
pitch rate (body-axis) q        [rad/s]
yaw rate            xi         [rad/s]
forward velocity    v_x        [ft/s]
lateral velocity     v_y        [ft/s]
vertical velocity    v_z        [ft/s]

OUTPUTS
heave velocity      H_dot      [ft/s]
pitch attitude       theta      [rad]
roll attitude        phi        [rad]
heading rate         psi_dot    [rad/s]
roll rate            p          [rad/s]
pitch rate           q          [rad/s]

```

Reference:

Skogestad, S. and Postlethwaite I.

Multivariable Feedback Control: Analysis and Design

Second Edition

Wiley 2005

http://www.nt.ntnu.no/users/skoge/book/2nd_edition/matlab_m/matfiles.html