

control 2.3.54

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.
- Frequency-weighted model and controller reduction.
- System identification by subspace 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
1.3	Anderson	1
1.4	Madievski	1
2	Linear Time Invariant Models	2
2.1	dss	2
2.2	filt	2
2.3	frd	3
2.4	ss	4
2.5	tf	5
2.6	zpk	7
3	Model Data Access	9
3.1	@lti/dssdata	9
3.2	@lti/filtdata	9
3.3	@lti/frdata	10
3.4	@lti/get	10
3.5	@lti/set	10
3.6	@lti/ssdata	10
3.7	@lti/tfdata	11
3.8	@lti/zpkdata	11
4	Model Conversions	12
4.1	@lti/c2d	12
4.2	@lti/d2c	12
4.3	@lti/prescale	12
4.4	@lti/xperm	13
5	Model Interconnections	14
5.1	@lti/append	14
5.2	@lti/blkdiag	14
5.3	@lti/connect	14
5.4	@lti/feedback	14
5.5	@lti/lft	15
5.6	@lti/mconnect	16
5.7	@lti/parallel	17
5.8	@lti/series	17
6	Model Characteristics	18
6.1	ctrb	18
6.2	ctrbf	18
6.3	@lti/dcgain	18
6.4	gram	19
6.5	hsvd	19

6.6	@lti/isct	19
6.7	isctrb	19
6.8	isdetectable	20
6.9	@lti/isdt	20
6.10	@lti/isminimumphase	21
6.11	isobsv	21
6.12	@lti/issiso	21
6.13	isstabilizable	22
6.14	@lti/isstable	22
6.15	@lti/norm	23
6.16	obsv	23
6.17	obsvf	23
6.18	@lti/pole	24
6.19	pzmap	24
6.20	@lti/size	24
6.21	@lti/zero	25
7	Model Simplification	26
7.1	@lti/minreal	26
7.2	@lti/sminreal	26
8	Time Domain Analysis	27
8.1	covar	27
8.2	gensig	27
8.3	impulse	28
8.4	initial	28
8.5	lsim	29
8.6	step	29
9	Frequency Domain Analysis	31
9.1	bode	31
9.2	bodemag	31
9.3	@lti/freqresp	32
9.4	margin	32
9.5	nichols	34
9.6	nyquist	34
9.7	sensitivity	35
9.8	sigma	36
10	Pole Placement	37
10.1	place	37
10.2	rlocus	38
11	Linear-Quadratic Control	39
11.1	dlqe	39
11.2	dlqr	40
11.3	estim	40
11.4	kalman	41
11.5	lqe	42
11.6	lqr	42

12	Robust Control	44
12.1	augw	44
12.2	fitfrd	45
12.3	h2syn	45
12.4	hinfyn	46
12.5	mixsyn	47
12.6	ncfsyn	49
13	Matrix Equation Solvers	51
13.1	care	51
13.2	dare	52
13.3	dlyap	53
13.4	dlyapchol	54
13.5	lyap	54
13.6	lyapchol	54
14	Model Reduction	55
14.1	bstmodred	55
14.2	btamodred	57
14.3	hnamodred	59
14.4	spamodred	61
15	Controller Reduction	65
15.1	btaconred	65
15.2	cfconred	67
15.3	fwcfconred	68
15.4	spaconred	69
16	Experimental Data Handling	72
16.1	iddata	72
16.2	@iddata/cat	73
16.3	@iddata/detrend	73
16.4	@iddata/diff	73
16.5	@iddata/fft	73
16.6	@iddata/filter	74
16.7	@iddata/get	74
16.8	@iddata/iff	74
16.9	@iddata/merge	75
16.10	@iddata/nkshift	75
16.11	@iddata/plot	75
16.12	@iddata/resample	75
16.13	@iddata/set	75
16.14	@iddata/size	76
17	System Identification	77
17.1	arx	77
17.2	moen4	78
17.3	moesp	80
17.4	n4sid	82

18	Overloaded LTI Operators	84
18.1	@lti/ctranspose	84
18.2	@lti/horzcat	84
18.3	@lti/inv	84
18.4	@lti/minus	84
18.5	@lti/mldivide	84
18.6	@lti/mpower	84
18.7	@lti/mrdivide	84
18.8	@lti/mtimes	84
18.9	@lti/plus	84
18.10	@lti/subsasgn	84
18.11	@lti/subsref	85
18.12	@lti/transpose	85
18.13	@lti/uminus	85
18.14	@lti/uplus	85
18.15	@lti/vertcat	85
19	Overloaded IDDATA Operators	86
19.1	@iddata/horzcat	86
19.2	@iddata/subsasgn	86
19.3	@iddata/subsref	86
19.4	@iddata/vertcat	86
20	Miscellaneous	87
20.1	options	87
20.2	strseq	87
20.3	test_control	87
20.4	BMWengine	88
20.5	Boeing707	88
20.6	WestlandLynx	89

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.

1.3 Anderson

Frequency-weighted coprime factorization controller reduction.

1.4 Madievski

Frequency-weighted controller reduction.

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

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

See also: `ss`, `tf`.

2.2 filt

`sys = filt (num, den, ...)` [Function File]
`sys = filt (num, den, tsam, ...)` [Function File]

Create discrete-time transfer function model from data in DSP format.

Inputs

`num` Numerator or cell of numerators. Each numerator must be a row vector containing the coefficients of the polynomial in ascending powers of z^{-1} . `num{i,j}` contains the numerator polynomial from input `j` to output `i`. In the SISO case, a single vector is accepted as well.
`den` Denominator or cell of denominators. Each denominator must be a row vector containing the coefficients of the polynomial in ascending powers of z^{-1} . `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, default value -1 (unspecified) is taken.

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

Outputs

`sys` Discrete-time transfer function model.

Example

$$H(z^{-1}) = \frac{3 z^{-1}}{1 + 4 z^{-1} + 2 z^{-2}}$$

```
octave:1> H = filt ([0, 3], [1, 4, 2])
```

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

$$y1: \frac{3 z^{-1}}{1 + 4 z^{-1} + 2 z^{-2}}$$

Sampling time: unspecified
Discrete-time model.

See also: `tf`.

2.3 frd

<code>sys = frd (sys)</code>	[Function File]
<code>sys = frd (sys, w)</code>	[Function File]
<code>sys = frd (H, w, ...)</code>	[Function File]
<code>sys = frd (H, w, tsam, ...)</code>	[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.

See also: `dss`, `ss`, `tf`.

2.4 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

<code>sys</code>	LTI model to be converted to state-space.
<code>a</code>	State transition matrix (n-by-n).
<code>b</code>	Input matrix (n-by-m).
<code>c</code>	Measurement matrix (p-by-n). If <code>c</code> is empty <code>[]</code> or not specified, an identity matrix is assumed.
<code>d</code>	Feedthrough matrix (p-by-m). If <code>d</code> is empty <code>[]</code> or not specified, a zero matrix is assumed.
<code>tsam</code>	Sampling time in seconds. If <code>tsam</code> is not specified, a continuous-time model is assumed.
<code>...</code>	Optional pairs of properties and values. Type <code>set (ss)</code> for more information.

Outputs

<code>sys</code>	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>

```

See also: `tf`, `dss`.

2.5 `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

<code>sys</code>	LTI model to be converted to transfer function.
<code>num</code>	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. <code>num{i,j}</code> contains the numerator polynomial from input <code>j</code> to output <code>i</code> . 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. <i>den</i> {i,j} contains the denominator polynomial from input j to output i. In the SISO case, a single vector is accepted as well.
<i>tsam</i>	Sampling time in seconds. If <i>tsam</i> is not specified, a continuous-time model is assumed.
...	Optional pairs of properties and values. Type <code>set (tf)</code> 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 ...

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

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 ...

$$y1: \frac{0.095}{z - 0.9}$$

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>
```

See also: ss, dss.

2.6 zpk

<code>s = zpk ("s")</code>	[Function File]
<code>z = zpk ("z", tsam)</code>	[Function File]
<code>sys = zpk (sys)</code>	[Function File]
<code>sys = zpk (k)</code>	[Function File]
<code>sys = zpk (z, p, k, ...)</code>	[Function File]
<code>sys = zpk (z, p, k, tsam, ...)</code>	[Function File]
<code>sys = zpk (z, p, k, tsam, ...)</code>	[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

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

See also: `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/filtdata

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

Access discrete-time transfer function data in DSP format. 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 discrete-time 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 ascending powers of z^{-1} . `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 ascending powers of z^{-1} . `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 `tsam` is not specified, -1 is returned.

3.3 @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.

3.4 @lti/get

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

Access property values of LTI objects.

3.5 @lti/set

`set (sys)` [Function File]
`set (sys, "property", value, ...)` [Function File]
`retsys = set (sys, "property", value, ...)` [Function File]

Set or modify properties of LTI objects. If no return argument `retsys` is specified, the modified LTI object is stored in input argument `sys`. `set` can handle multiple properties in one call: `set (sys, 'prop1', val1, 'prop2', val2, 'prop3', val3)`. `set (sys)` prints a list of the object's property names.

3.6 @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.

3.7 @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.

3.8 @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", "bilinear" 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", "bilinear" 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

$$Es = Tl * E * Tr$$

$$As = Tl * A * Tr$$

$$Bs = Tl * B$$

$$Cs = C * Tr$$

$$Ds = D$$

=

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.

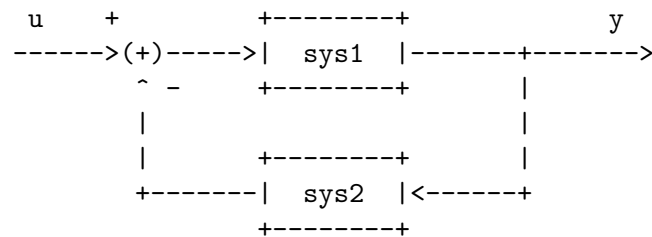
Inputs

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 transformation, also known as Redheffer star product.

Inputs

`sys1` Upper LTI model.

`sys2` Lower LTI model.

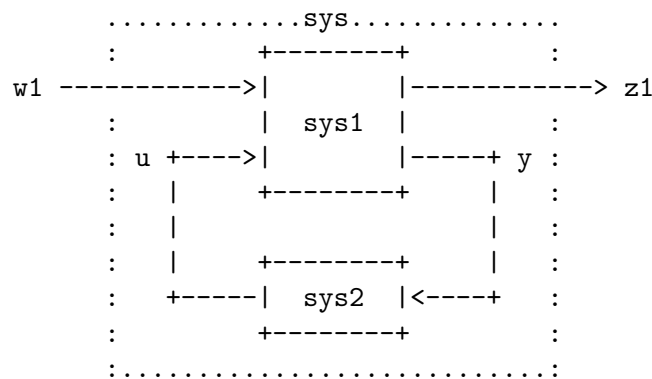
`nu` The last `nu` inputs of `sys1` are connected with the first `nu` outputs of `sys2`. If not specified, `min (m1, p2)` is taken.

`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

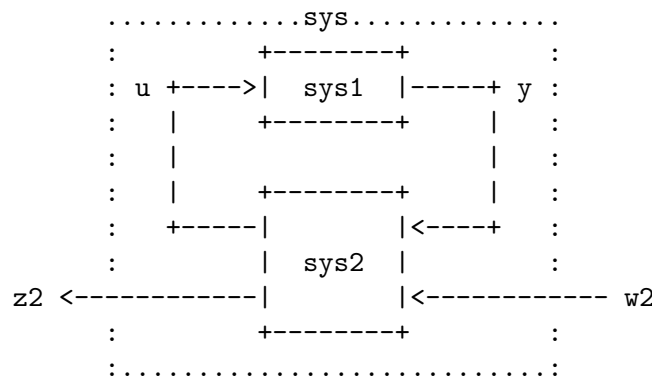
`sys` Resulting LTI model.

Block Diagram



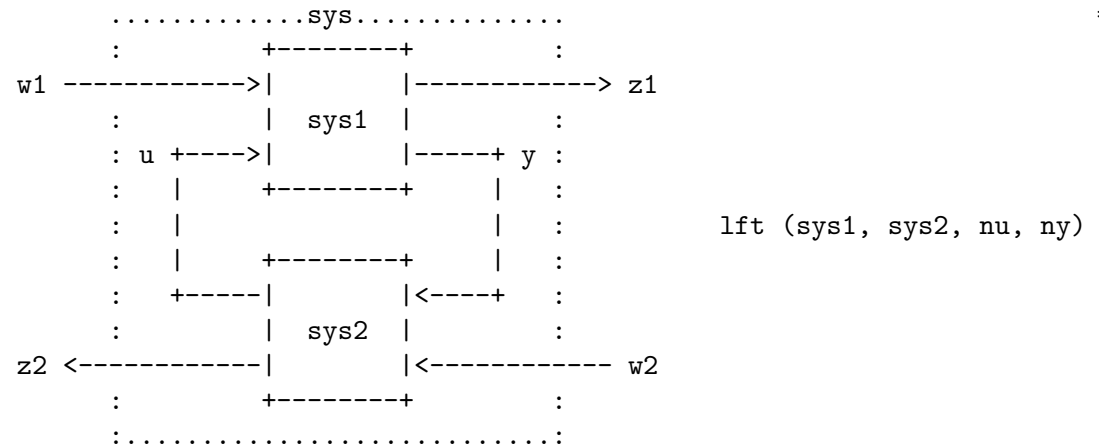
Lower LFT

`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:

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

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

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

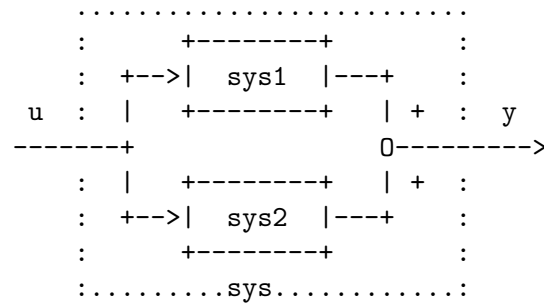
5.7 @lti/parallel

`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)`

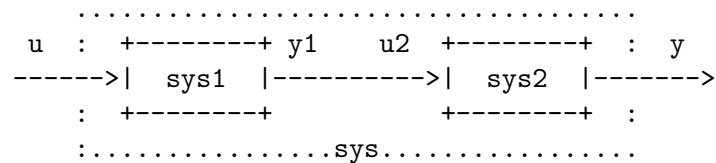
[Function File]

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

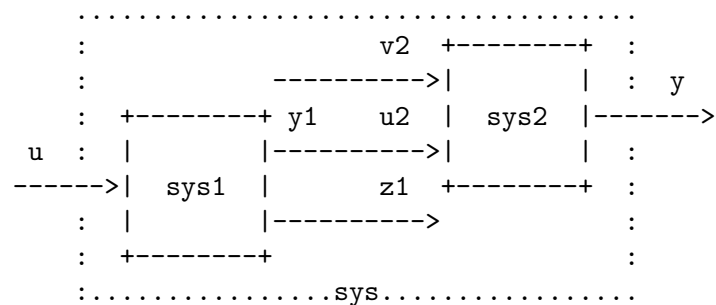
[Function File]

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

$$A_{\text{bar}} = T_c \setminus A * T_c, \quad B_{\text{bar}} = T_c \setminus B, \quad C_{\text{bar}} = C * T_c \quad =$$

and the transformed system has the form

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

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 $[p, m]$.

See also: 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. If `e` is empty `[]` or not specified, an identity matrix is assumed.

`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.](#)

See also: `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. If *e* is empty [] or not specified, an identity matrix is assumed.
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.

See also: `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 $|\text{sys}(j\omega)|$.

Inputs

`sys` LTI system.

`tol` Optional tolerance. Default value is 0.

Outputs

`bool = 0` System is not minimum phase.

`bool = 1` System is minimum phase.

<code>real (z) < -tol*(1 + abs (z))</code>	continuous-time	=
<code>abs (z) < 1 - tol</code>	discrete-time	

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. If `e` is empty `[]` or not specified, an identity matrix is assumed.

`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.](#)

See also: `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. If *e* is empty [] or not specified, an identity matrix is assumed.

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

```

See also: 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.

Inputs

sys LTI system.

tol Optional tolerance for stability. Default value is 0.

Outputs

bool = 0 System is not stable.

bool = 1 System is stable.

real (p) < -tol*(1 + abs (p)) continuous-time
abs (p) < 1 - tol discrete-time

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 *Ob*=obsv(*A*,*C*) has rank *r* <= *n* = SIZE(*A*,1), then there is a similarity transformation *Tc* such that *To* = [*t1*;*t2*] where *t1* is *c* and *t2* is orthogonal to *t1*

$$\bar{A} = T_o \setminus A * T_o, \quad \bar{B} = T_o \setminus B, \quad \bar{C} = C * T_o \quad =$$

and the transformed system has the form

$$\bar{A} = \begin{bmatrix} A_o & 0 \\ \hline A_{21} & A_{no} \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} B_o \\ \hline B_{no} \end{bmatrix}, \quad \bar{C} = [C_o \mid 0]. \quad =$$

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]

`pzmap (sys1, sys2, ..., sysN)` [Function File]

`pzmap (sys1, 'style1', ..., sysN, 'styleN')` [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.

See also: 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 Gaussian white noise inputs which drive *sys*.

Outputs

p Output covariance.
q State covariance.

See also: `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.

See also: `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.

See also: `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: $x[k+1] = A x[k]$, $y[k] = C x[k]$, $x[0] = x0$

See also: 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.

See also: 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.

See also: `impulse`, `initial`, `lsim`.

9 Frequency Domain Analysis

9.1 bode

`bode (sys)` [Function File]
`bode (sys1, sys2, ..., sysN)` [Function File]
`bode (sys1, sys2, ..., sysN, w)` [Function File]
`bode (sys1, 'style1', ..., sysN, 'styleN')` [Function File]
`[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. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where `wmin` and `wmax` denote minimum and maximum frequencies in rad/s.

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.

See also: `nichols`, `nyquist`, `sigma`.

9.2 bodemag

`bodemag (sys)` [Function File]
`bodemag (sys1, sys2, ..., sysN)` [Function File]
`bodemag (sys1, sys2, ..., sysN, w)` [Function File]
`bodemag (sys1, 'style1', ..., sysN, 'styleN')` [Function File]
`[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. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where `wmin` and `wmax` denote minimum and maximum frequencies in rad/s.

Outputs

`mag` Vector of magnitude. Has length of frequency vector `w`.
`w` Vector of frequency values used.

See also: `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)`.

See also: `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. A more robust criterion to assess the stability of a feedback system is the sensitivity `Ms` computed by command `sensitivity`.

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

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

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

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

$$\begin{aligned} \text{imag} (\text{num}(j\omega) \text{ den}(-j\omega)) &= 0 \\ \text{imag} (\text{num}(-j\omega) \text{ den}(j\omega)) &= 0 \end{aligned}$$

=

Phase Margin

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

$$z^2 = Re\ z + Im\ z$$

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

$$num(j\omega) num(-j\omega) - den(j\omega) den(-j\omega) = 0$$

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

DISCRETE SYSTEMS

Gain Margin

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

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

$$num(z) den(1/z) - num(1/z) den(z) = 0$$

Phase Margin

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

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

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

$$num(z) num(1/z) - den(z) den(1/z) = 0$$

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

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

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

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

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

See also: roots.

9.5 nichols

`nichols (sys)` [Function File]

`nichols (sys1, sys2, ..., sysN)` [Function File]

`nichols (sys1, sys2, ..., sysN, w)` [Function File]

`nichols (sys1, 'style1', ..., sysN, 'styleN')` [Function File]

`[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. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where `wmin` and `wmax` denote minimum and maximum frequencies in rad/s.

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.

See also: bode, nyquist, sigma.

9.6 nyquist

`nyquist (sys)` [Function File]

`nyquist (sys1, sys2, ..., sysN)` [Function File]

`nyquist (sys1, sys2, ..., sysN, w)` [Function File]

`nyquist (sys1, 'style1', ..., sysN, 'styleN')` [Function File]

`[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. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where *wmin* and *wmax* denote minimum and maximum frequencies in rad/s.

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.

See also: `bode`, `nichols`, `sigma`.

9.7 sensitivity

`[Ms, ws] = sensitivity (L)` [Function File]
`[Ms, ws] = sensitivity (P, C)` [Function File]
`[Ms, ws] = sensitivity (P, C1, C2, ...)` [Function File]

Return sensitivity margin *Ms*. The quantity *Ms* is simply the inverse of the shortest distance from the Nyquist curve to the critical point -1. Reasonable values of *Ms* are in the range from 1.3 to 2.

$$M_s = ||S(j\omega)||_{\infty}$$

If no output arguments are given, the critical distance $1/M_s$ is plotted on a Nyquist diagram. In contrast to gain and phase margin as computed by command `margin`, the sensitivity *Ms* is a more robust criterion to assess the stability of a feedback system.

Inputs

L Open loop transfer function. *L* can be any type of LTI system, but it must be square.

P Plant model. Any type of LTI system.

C Controller model. Any type of LTI system.

C1, C2, ...
 If several controllers are specified, command `sensitivity` computes the sensitivity *Ms* for each of them in combination with plant *P*.

Outputs

Ms Sensitivity margin *Ms* as defined in [1]. Scalar value. If several controllers are specified, *Ms* becomes a row vector with as many entries as controllers.

ws The frequency [rad/s] corresponding to the sensitivity peak. Scalar value. If several controllers are specified, *ws* becomes a row vector with as many entries as controllers.

Algorithm

Uses SLICOT AB13DD by courtesy of **NICONET e.V.**

References

[1] Aström, K. and Hägglund, T. (1995) PID Controllers: Theory, Design and Tuning, Second Edition. Instrument Society of America.

9.8 sigma

<code>sigma (sys)</code>	[Function File]
<code>sigma (sys1, sys2, ..., sysN)</code>	[Function File]
<code>sigma (sys1, sys2, ..., sysN, w)</code>	[Function File]
<code>sigma (sys1, 'style1', ..., sysN, 'styleN')</code>	[Function File]
<code>[sv, w] = sigma (sys)</code>	[Function File]
<code>[sv, w] = sigma (sys, w)</code>	[Function File]
<code>[sv, w] = sigma (sys, [], ptype)</code>	[Function File]
<code>[sv, w] = sigma (sys, w, ptype)</code>	[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. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where *wmin* and *wmax* denote minimum and maximum frequencies in rad/s.

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.

See also: `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 .

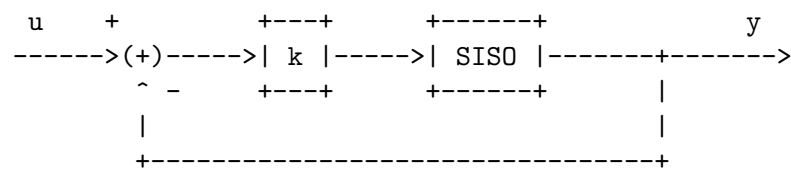
increment The increment used in computing gain values.

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 dlqe

$[m, p, z, e] = \text{dlqe}(a, g, c, q, r)$ [Function File]
 $[m, p, z, e] = \text{dlqe}(a, g, c, q, r, s)$ [Function File]
 $[m, p, z, e] = \text{dlqe}(a, [], c, q, r)$ [Function File]
 $[m, p, z, e] = \text{dlqe}(a, [], c, q, r, s)$ [Function File]

Kalman filter for discrete-time systems.

$$x[k] = Ax[k] + Bu[k] + Gw[k] \quad (\text{State equation})$$

$$y[k] = Cx[k] + Du[k] + v[k] \quad (\text{Measurement Equation})$$

$$E(w) = 0, E(v) = 0, \text{cov}(w) = Q, \text{cov}(v) = R, \text{cov}(w, v) = S$$

Inputs

a State transition matrix of discrete-time system (n-by-n).
g Process noise matrix of discrete-time system (n-by-g). If *g* is empty [], an identity matrix is assumed.
c Measurement matrix of discrete-time system (p-by-n).
q Process noise covariance matrix (g-by-g).
r Measurement noise covariance matrix (p-by-p).
s Optional cross term covariance matrix (g-by-p), $s = \text{cov}(w, v)$. If *s* is empty [] or not specified, a zero matrix is assumed.

Outputs

m Kalman filter gain matrix (n-by-p).
p Unique stabilizing solution of the discrete-time Riccati equation (n-by-n). Symmetric matrix.
z Error covariance (n-by-n), $\text{cov}(x(k|k)-x)$
e Closed-loop poles (n-by-1).

Equations

$$x[k|k] = x[k|k-1] + M(y[k] - Cx[k|k-1] - Du[k])$$

$$x[k+1|k] = Ax[k|k] + Bu[k] \quad \text{for } S=0$$

$$x[k+1|k] = Ax[k|k] + Bu[k] + G*S*(C*P*C' + R)^{-1}*(y[k] - C*x[k|k-1]) \quad \text{for non-zero } S$$

$$E = \text{eig}(A - A*M*C) \quad \text{for } S=0$$

$$E = \text{eig}(A - A*M*C - G*S*(C*P*C' + R)^{-1}*C) \quad \text{for non-zero } S$$

See also: dare, care, dlqr, lqr, lqe.

11.2 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

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

Outputs

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

Equations

$$\mathbf{x}[k+1] = \mathbf{A} \mathbf{x}[k] + \mathbf{B} \mathbf{u}[k], \quad \mathbf{x}[0] = \mathbf{x}_0 \quad =$$

$$J(\mathbf{x}_0) = \sum_{k=0}^{\infty} (\mathbf{x}' \mathbf{Q} \mathbf{x} + \mathbf{u}' \mathbf{R} \mathbf{u} + 2 \mathbf{x}' \mathbf{S} \mathbf{u})$$

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

See also: `dare`, `care`, `lqr`.

11.3 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

<code>sys</code>	LTI model.
<code>l</code>	State feedback matrix.
<code>sensors</code>	Indices of measured output signals <code>y</code> from <code>sys</code> . 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.

See also: *kalman*, *place*.

11.4 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.

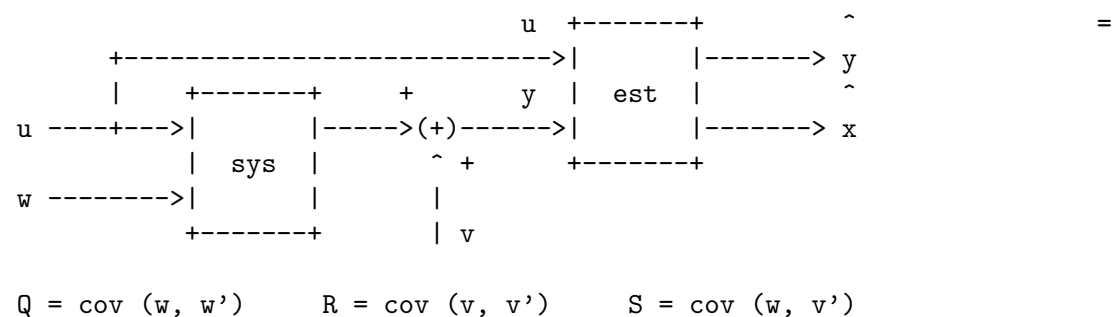
Outputs

est State-space model of the Kalman estimator.

g Estimator gain.

x Solution of the Riccati equation.

Block Diagram



See also: *care*, *dare*, *estim*, *lqr*.

11.5 lqe

<code>[l, p, e] = lqe (sys, q, r)</code>	[Function File]
<code>[l, p, e] = lqe (sys, q, r, s)</code>	[Function File]
<code>[l, p, e] = lqe (a, g, c, q, r)</code>	[Function File]
<code>[l, p, e] = lqe (a, g, c, q, r, s)</code>	[Function File]
<code>[l, p, e] = lqe (a, [], c, q, r)</code>	[Function File]
<code>[l, p, e] = lqe (a, [], c, q, r, s)</code>	[Function File]

Kalman filter for continuous-time systems.

$$\begin{aligned} \dot{x} &= Ax + Bu + Gw && \text{(State equation)} \\ y &= Cx + Du + v && \text{(Measurement Equation)} \\ E(w) &= 0, E(v) = 0, \text{cov}(w) = Q, \text{cov}(v) = R, \text{cov}(w,v) = S \end{aligned}$$

Inputs

<code>sys</code>	Continuous or discrete-time LTI model (p-by-m, n states).
<code>a</code>	State transition matrix of continuous-time system (n-by-n).
<code>g</code>	Process noise matrix of continuous-time system (n-by-g). If <code>g</code> is empty <code>[]</code> , an identity matrix is assumed.
<code>c</code>	Measurement matrix of continuous-time system (p-by-n).
<code>q</code>	Process noise covariance matrix (g-by-g).
<code>r</code>	Measurement noise covariance matrix (p-by-p).
<code>s</code>	Optional cross term covariance matrix (g-by-p), <code>s = cov(w,v)</code> . If <code>s</code> is empty <code>[]</code> or not specified, a zero matrix is assumed.

Outputs

<code>l</code>	Kalman filter gain matrix (n-by-p).
<code>p</code>	Unique stabilizing solution of the continuous-time Riccati equation (n-by-n). Symmetric matrix. If <code>sys</code> is a discrete-time model, the solution of the corresponding discrete-time Riccati equation is returned.
<code>e</code>	Closed-loop poles (n-by-1).

Equations

$$\begin{aligned} \dot{x} &= Ax + Bu + L(y - Cx - Du) \\ E &= \text{eig}(A - L \cdot C) \end{aligned}$$

See also: `dare`, `care`, `dlqr`, `lqr`, `dlqe`.

11.6 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]

`[g, x, l] = lqr (a, b, q, r, [], e)` [Function File]
`[g, x, l] = lqr (a, b, q, r, s, e)` [Function File]

Linear-quadratic regulator.

Inputs

`sys` Continuous or discrete-time LTI model (p-by-m, n states).
`a` State transition matrix of continuous-time system (n-by-n).
`b` Input matrix of continuous-time system (n-by-m).
`q` State weighting matrix (n-by-n).
`r` Input weighting matrix (m-by-m).
`s` Optional cross term matrix (n-by-m). If `s` is not specified, a zero matrix is assumed.
`e` Optional descriptor matrix (n-by-n). If `e` is not specified, an identity matrix is assumed.

Outputs

`g` State feedback matrix (m-by-n).
`x` Unique stabilizing solution of the continuous-time Riccati equation (n-by-n).
`l` Closed-loop poles (n-by-1).

Equations

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

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

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

See also: `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

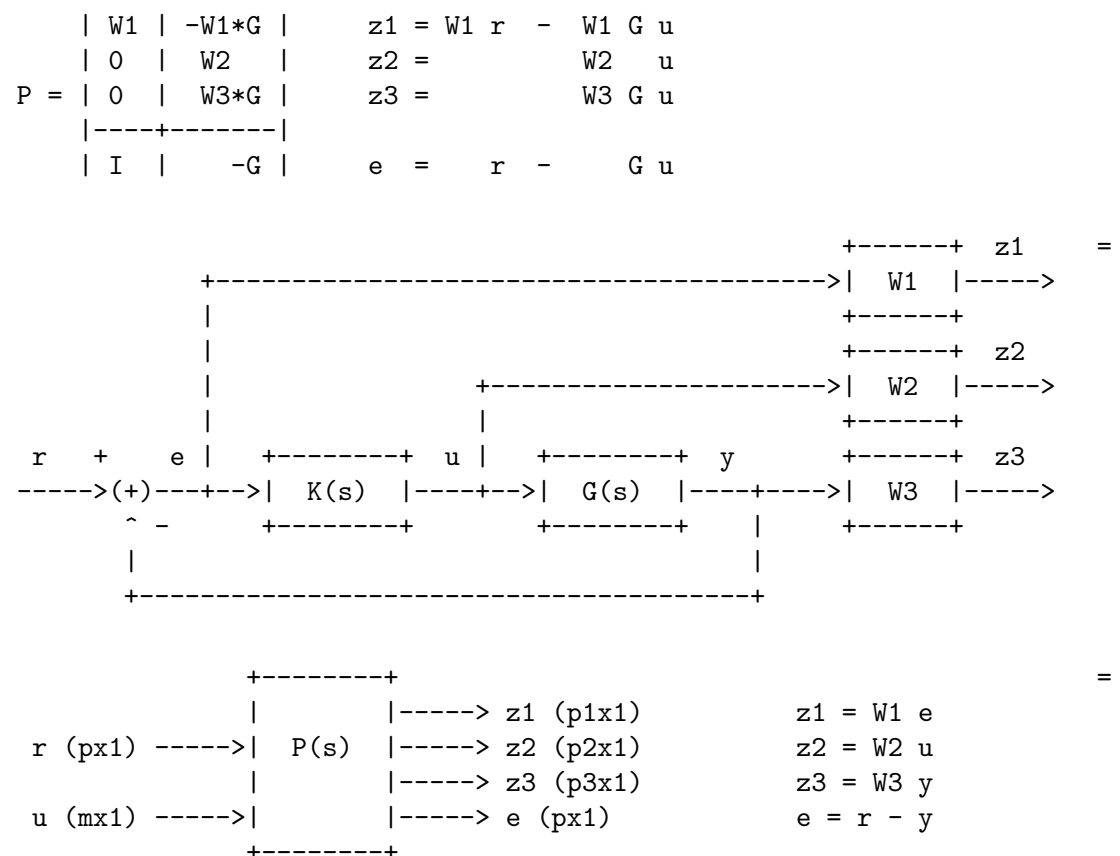
- G LTI model of plant.
- $W1$ LTI model of performance weight. Bounds the largest singular values of sensitivity S . Model must be empty [], SISO or of appropriate size.
- $W2$ LTI model to penalize large control inputs. Bounds the largest singular values of KS . Model must be empty [], SISO or of appropriate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

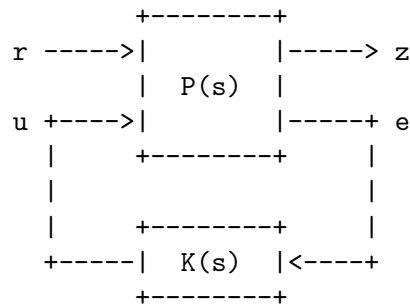
All inputs must be proper/realizable. Scalars, vectors and matrices are possible instead of LTI models.

Outputs

P State-space model of augmented plant.

Block Diagram





References

[1] Skogestad, S. and Postlethwaite I. (2005) *Multivariable Feedback Control: Analysis and Design: Second Edition*. Wiley.

See also: h2syn, hinfsyn, mixsyn.

12.2 fitfrd

`[sys, n] = fitfrd (dat, n)` [Function File]

`[sys, n] = fitfrd (dat, n, flag)` [Function File]

Fit frequency response data with a state-space system. If requested, the returned system is stable and minimum-phase.

Inputs

- | | |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>dat</i> | LTI model containing frequency response data of a SISO system. |
| <i>n</i> | The desired order of the system to be fitted. <code>n <= length(dat.w)</code> . |
| <i>flag</i> | The flag controls whether the returned system is stable and minimum-phase. |
| 0 | The system zeros and poles are not constrained. Default value. |
| 1 | The system zeros and poles will have negative real parts in the continuous-time case, or moduli less than 1 in the discrete-time case. |

Outputs

- | | |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>sys</i> | State-space model of order <i>n</i> , fitted to frequency response data <i>dat</i> . |
| <i>n</i> | The order of the obtained system. The value of <i>n</i> could only be modified if inputs <code>n > 0</code> and <code>flag = 1</code> . |

Algorithm

Uses SLICOT SB10YD by courtesy of [NICONET e.V.](#)

12.3 h2syn

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

H-2 control synthesis for LTI plant.

Inputs

- | | |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>P</i> | Generalized plant. Must be a proper/realizable LTI model. |
| <i>nmeas</i> | Number of measured outputs <i>v</i> . The last <i>nmeas</i> outputs of <i>P</i> are connected to the inputs of controller <i>K</i> . The remaining outputs <i>z</i> (indices 1 to <i>p-nmeas</i>) 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.

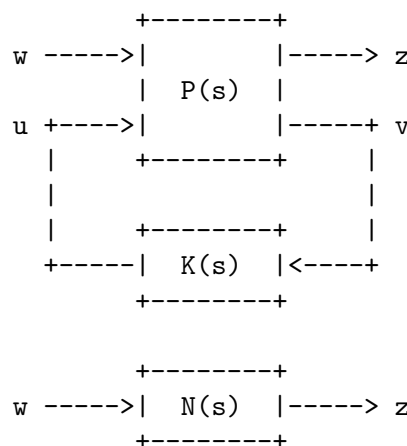
N State-space model of the lower LFT of P and K .

γ H-2 norm of N .

$rcond$ Vector $rcond$ contains estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller K . For details, see the description of the corresponding SLICOT algorithm.

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.](#)

See also: `augw`, `lqr`, `dlqr`, `kalman`.

12.4 hinfsyn

`[K, N, gamma, rcond] = hinfsyn (P, nmeas, ncon)` [Function File]

`[K, N, gamma, rcond] = 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.

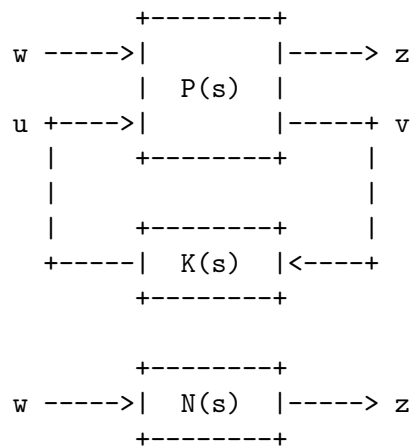
N State-space model of the lower LFT of P and K .

gamma L-infinity norm of N .

rcond Vector *rcond* contains estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller K . For details, see the description of the corresponding SLICOT algorithm.

Block Diagram

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



Algorithm

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

See also: `augw`, `mixsyn`.

12.5 mixsyn

`[K, N, gamma, rcond] = 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.

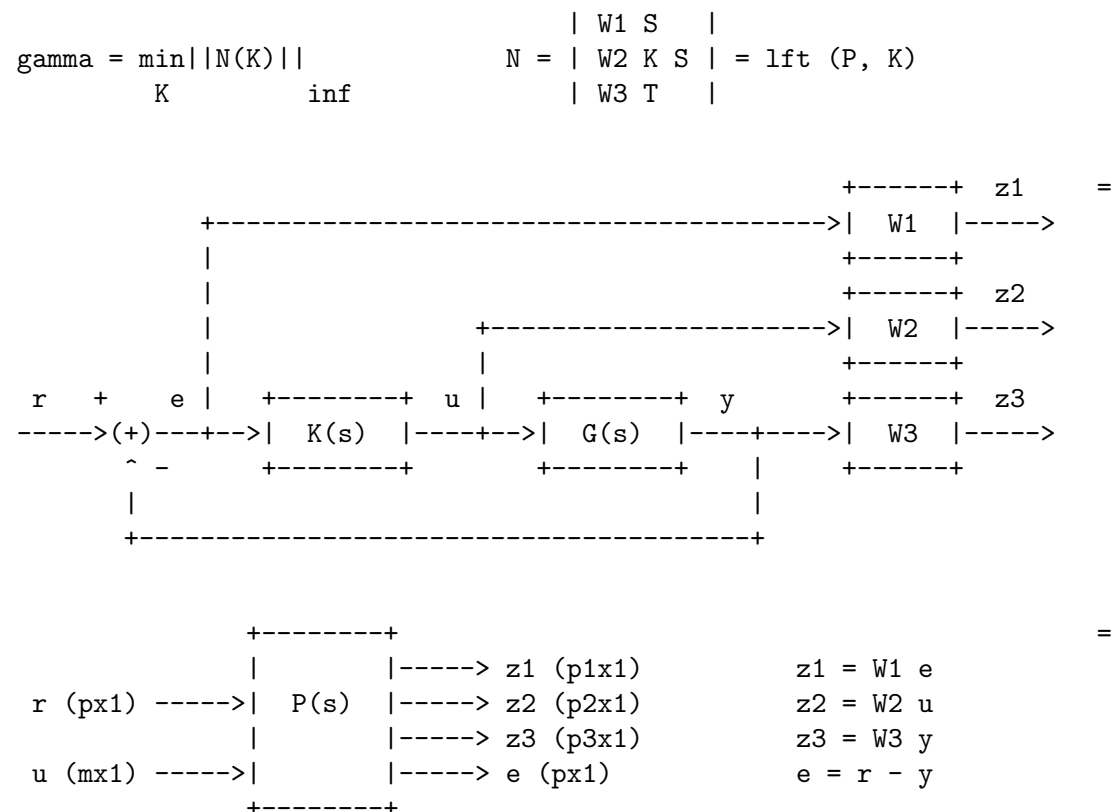
- W2* LTI model to penalize large control inputs. Bounds the largest singular values of KS . Model must be empty $[]$, SISO or of appropriate size.
- W3* LTI model of robustness and noise sensitivity weight. Bounds the largest singular values of complementary sensitivity T . Model must be empty $[]$,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

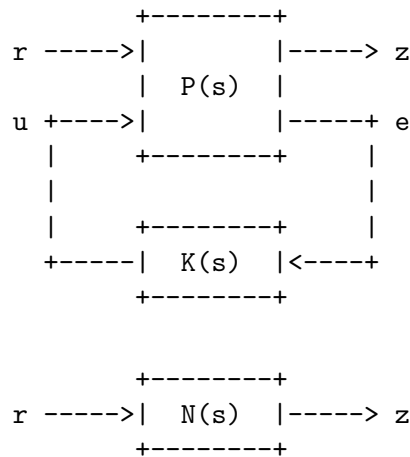
All inputs must be proper/realizable. Scalars, vectors and matrices are possible instead of LTI models.

Outputs

- K* State-space model of the H-infinity (sub-)optimal controller.
- N* State-space model of the lower LFT of P and K .
- gamma* L-infinity norm of N .
- rcond* Vector *rcond* contains estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller K . For details, see the description of the corresponding SLICOT algorithm.

Block Diagram





```

Extended Plant:  P = augw (G, W1, W2, W3)
Controller:      K = mixsyn (G, W1, W2, W3)
Entire System:   N = lft (P, K)
Open Loop:       L = G * K
Closed Loop:     T = feedback (L)

```

Algorithm

Relies on commands `augw` and `hinfsyn`, which use SLICOT SB10FD and SB10DD by courtesy of [NICONET e.V.](#)

References

[1] Skogestad, S. and Postlethwaite I. (2005) *Multivariable Feedback Control: Analysis and Design: Second Edition*. Wiley.

See also: `hinfsyn`, `augw`.

12.6 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.

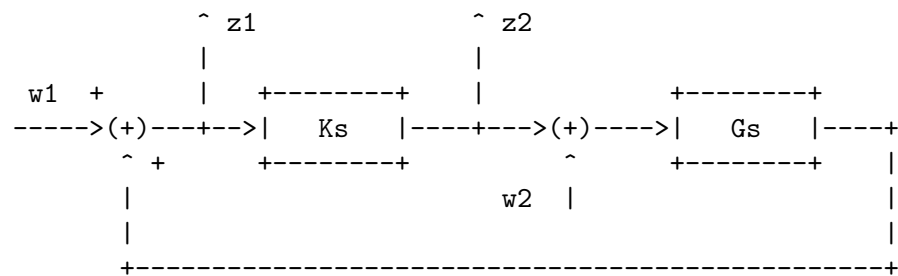
info.emax Nugap robustness. $\mathbf{emax} = \mathbf{inv}(\gamma)$.

info.Gs Shaped plant. $\mathbf{Gs} = \mathbf{W2} * \mathbf{G} * \mathbf{W1}$.

info.Ks Controller for shaped plant. $\mathbf{Ks} = \mathbf{ncfsyn}(\mathbf{Gs})$.

info.rcond Estimates of the reciprocal condition numbers of the Riccati equations and a few other things. For details, see the description of the corresponding SLICOT algorithm.

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

a Real matrix (n-by-n).

b Real matrix (n-by-m).

q Real matrix (n-by-n).

r Real matrix (m-by-m).

s Optional real matrix (n-by-m). If *s* is not specified, a zero matrix is assumed.

e Optional descriptor matrix (n-by-n). If *e* is not specified, an identity matrix is assumed.

Outputs

x Unique stabilizing solution of the continuous-time Riccati equation (n-by-n).

l Closed-loop poles (n-by-1).

g 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.](#)

See also: 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.](#)

See also: 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.](#)

See also: 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.](#)

See also: 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.](#)

See also: 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.](#)

See also: lyap, dlyap, dlyapchol.

14 Model Reduction

14.1 bstmodred

`[Gr, info] = bstmodred (G, ...)` [Function File]
`[Gr, info] = bstmodred (G, nr, ...)` [Function File]
`[Gr, info] = bstmodred (G, opt, ...)` [Function File]
`[Gr, info] = bstmodred (G, nr, opt, ...)` [Function File]

Model order reduction by Balanced Stochastic Truncation (BST) method. The aim of model reduction is to find an LTI system Gr of order nr ($nr < n$) such that the input-output behaviour of Gr approximates the one from original system G .

BST is a relative error method which tries to minimize

$$\|G^{-1}(G - G_r)\|_{\infty} = \min$$

Inputs

G LTI model to be reduced.
 nr The desired order of the resulting reduced order system Gr . If not specified, nr is chosen automatically according to the description of key 'order'.
 \dots Optional pairs of keys and values. "key1", value1, "key2", value2.
 opt Optional struct with keys as field names. Struct opt can be created directly or by command `options`. `opt.key1 = value1`, `opt.key2 = value2`.

Outputs

Gr Reduced order state-space model.
 $info$ Struct containing additional information.
 $info.n$ The order of the original system G .
 $info.ns$ The order of the α -stable subsystem of the original system G .
 $info.hsv$ The Hankel singular values of the phase system corresponding to the α -stable part of the original system G . The ns Hankel singular values are ordered decreasingly.
 $info.nu$ The order of the α -unstable subsystem of both the original system G and the reduced-order system Gr .
 $info.nr$ The order of the obtained reduced order system Gr .

Option Keys and Values

'order', 'nr' The desired order of the resulting reduced order system Gr . If not specified, nr is the sum of `NU` and the number of Hankel singular values greater than `MAX(TOL1, NS*EPS)`; nr can be further reduced to ensure that `HSV(NR-NU) > HSV(NR+1-NU)`.
 'method' Approximation method for the H-infinity norm. Valid values corresponding to this key are:
 'sr-bta', 'b' Use the square-root Balance & Truncate method.

<code>'bfsr-bta', 'f'</code>	Use the balancing-free square-root Balance & Truncate method. Default method.
<code>'sr-spa', 's'</code>	Use the square-root Singular Perturbation Approximation method.
<code>'bfsr-spa', 'p'</code>	Use the balancing-free square-root Singular Perturbation Approximation method.
<code>'alpha'</code>	Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix $G.A$. For a continuous-time system, $ALPHA \leq 0$ is the boundary value for the real parts of eigenvalues, while for a discrete-time system, $0 \leq ALPHA \leq 1$ represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time systems and 1 for discrete-time systems.
<code>'beta'</code>	Use $[G, \text{beta} \cdot I]$ as new system G to combine absolute and relative error methods. $BETA > 0$ specifies the absolute/relative error weighting parameter. A large positive value of $BETA$ favours the minimization of the absolute approximation error, while a small value of $BETA$ is appropriate for the minimization of the relative error. $BETA = 0$ means a pure relative error method and can be used only if $\text{rank}(G.D) = \text{rows}(G.D)$ which means that the feedthrough matrix must not be rank-deficient. Default value is 0.
<code>'tol1'</code>	If <code>'order'</code> is not specified, <code>tol1</code> contains the tolerance for determining the order of reduced system. For model reduction, the recommended value of <code>tol1</code> lies in the interval $[0.00001, 0.001]$. $\text{tol1} < 1$. If $\text{tol1} \leq 0$ on entry, the used default value is $\text{tol1} = NS \cdot \text{EPS}$, where NS is the number of ALPHA-stable eigenvalues of A and EPS is the machine precision. If <code>'order'</code> is specified, the value of <code>tol1</code> is ignored.
<code>'tol2'</code>	The tolerance for determining the order of a minimal realization of the phase system (see <code>METHOD</code>) corresponding to the ALPHA-stable part of the given system. The recommended value is $\text{TOL2} = NS \cdot \text{EPS}$. $\text{TOL2} \leq \text{TOL1} < 1$. This value is used by default if <code>'tol2'</code> is not specified or if $\text{TOL2} \leq 0$ on entry.
<code>'equil', 'scale'</code>	Boolean indicating whether equilibration (scaling) should be performed on system G prior to order reduction. Default value is true if $G.\text{scaled} == \text{false}$ and false if $G.\text{scaled} == \text{true}$. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can not be done by the equilibration option or the <code>prescale</code> command because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

BST is often suitable to perform model reduction in order to obtain low order design models for controller synthesis.

Approximation Properties:

- Guaranteed stability of reduced models
- Approximates simultaneously gain and phase
- Preserves non-minimum phase zeros
- Guaranteed a priori error bound

$$\|G^{-1}(G - G_r)\|_{\infty} \leq 2 \sum_{j=r+1}^n 1 + \sigma_j - \sigma_j - 1$$

Algorithm

Uses SLICOT AB09HD by courtesy of [NICONET e.V.](#)

14.2 btamodred

```
[Gr, info] = btamodred (G, ...) [Function File]
[Gr, info] = btamodred (G, nr, ...) [Function File]
[Gr, info] = btamodred (G, opt, ...) [Function File]
[Gr, info] = btamodred (G, nr, opt, ...) [Function File]
```

Model order reduction by frequency weighted Balanced Truncation Approximation (BTA) method. The aim of model reduction is to find an LTI system Gr of order nr ($nr < n$) such that the input-output behaviour of Gr approximates the one from original system G .

BTA is an absolute error method which tries to minimize

$$\|G - G_r\|_{\infty} = \min$$

$$\|V (G - G_r) W\|_{\infty} = \min$$

where V and W denote output and input weightings.

Inputs

G	LTI model to be reduced.
nr	The desired order of the resulting reduced order system Gr . If not specified, nr is chosen automatically according to the description of key 'order'.
\dots	Optional pairs of keys and values. "key1", value1, "key2", value2.
opt	Optional struct with keys as field names. Struct opt can be created directly or by command <code>options</code> . <code>opt.key1 = value1</code> , <code>opt.key2 = value2</code> .

Outputs

Gr	Reduced order state-space model.
$info$	Struct containing additional information.
$info.n$	The order of the original system G .
$info.ns$	The order of the <i>alpha</i> -stable subsystem of the original system G .
$info.hsv$	The Hankel singular values of the <i>alpha</i> -stable part of the original system G , ordered decreasingly.
$info.nu$	The order of the <i>alpha</i> -unstable subsystem of both the original system G and the reduced-order system Gr .
$info.nr$	The order of the obtained reduced order system Gr .

Option Keys and Values

'order', 'nr'	The desired order of the resulting reduced order system Gr . If not specified, nr is chosen automatically such that states with Hankel singular values $info.hsv > toll$ are retained.
---------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

'left', 'output'	LTI model of the left/output frequency weighting V . Default value is an identity matrix.
------------------	---------------------------------------------------------------------------------------------

- 'right', 'input'** LTI model of the right/input frequency weighting W . Default value is an identity matrix.
- 'method'** Approximation method for the L-infinity norm to be used as follows:
- 'sr', 'b'** Use the square-root Balance & Truncate method.
 - 'bfsr', 'f'** Use the balancing-free square-root Balance & Truncate method. Default method.
- 'alpha'** Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix $G.A$. For a continuous-time system, $\text{ALPHA} \leq 0$ is the boundary value for the real parts of eigenvalues, while for a discrete-time system, $0 \leq \text{ALPHA} \leq 1$ represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time systems and 1 for discrete-time systems.
- 'tol1'** If **'order'** is not specified, *tol1* contains the tolerance for determining the order of the reduced model. For model reduction, the recommended value of *tol1* is $c \cdot \text{info.hsv}(1)$, where c lies in the interval $[0.00001, 0.001]$. Default value is $\text{info.ns} \cdot \text{eps} \cdot \text{info.hsv}(1)$. If **'order'** is specified, the value of *tol1* is ignored.
- 'tol2'** The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given model. $\text{TOL2} \leq \text{TOL1}$. If not specified, $\text{ns} \cdot \text{eps} \cdot \text{info.hsv}(1)$ is chosen.
- 'gram-ctrb'** Specifies the choice of frequency-weighted controllability Grammian as follows:
- 'standard'** Choice corresponding to a combination method [4] of the approaches of Enns [1] and Lin-Chiu [2,3]. Default method.
 - 'enhanced'** Choice corresponding to the stability enhanced modified combination method of [4].
- 'gram-obsv'** Specifies the choice of frequency-weighted observability Grammian as follows:
- 'standard'** Choice corresponding to a combination method [4] of the approaches of Enns [1] and Lin-Chiu [2,3]. Default method.
 - 'enhanced'** Choice corresponding to the stability enhanced modified combination method of [4].
- 'alpha-ctrb'** Combination method parameter for defining the frequency-weighted controllability Grammian. $\text{abs}(\text{alphac}) \leq 1$. If $\text{alphac} = 0$, the choice of Grammian corresponds to the method of Enns [1], while if $\text{alphac} = 1$, the choice of Grammian corresponds to the method of Lin and Chiu [2,3]. Default value is 0.
- 'alpha-obsv'** Combination method parameter for defining the frequency-weighted observability Grammian. $\text{abs}(\text{alphao}) \leq 1$. If $\text{alphao} = 0$, the choice of Grammian corresponds to the method of Enns [1], while if $\text{alphao} = 1$, the choice of Grammian corresponds to the method of Lin and Chiu [2,3]. Default value is 0.

'*equil*', '*scale*'

Boolean indicating whether equilibration (scaling) should be performed on system G prior to order reduction. This is done by state transformations. Default value is true if `G.scaled == false` and false if `G.scaled == true`. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` command because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Approximation Properties:

- Guaranteed stability of reduced models
- Lower guaranteed error bound
- Guaranteed a priori error bound

$$\sigma_{r+1} \leq \|(G - G_r)\|_\infty \leq 2 \sum_{j=r+1}^n \sigma_j$$

References

- [1] Enns, D. *Model reduction with balanced realizations: An error bound and a frequency weighted generalization*. Proc. 23-th CDC, Las Vegas, pp. 127-132, 1984.
- [2] Lin, C.-A. and Chiu, T.-Y. *Model reduction via frequency-weighted balanced realization*. Control Theory and Advanced Technology, vol. 8, pp. 341-351, 1992.
- [3] Sreeram, V., Anderson, B.D.O and Madieviski, A.G. *New results on frequency weighted balanced reduction technique*. Proc. ACC, Seattle, Washington, pp. 4004-4009, 1995.
- [4] Varga, A. and Anderson, B.D.O. *Square-root balancing-free methods for the frequency-weighted balancing related model reduction*. (report in preparation)

Algorithm

Uses SLICOT AB09ID by courtesy of **NICONET e.V.**

14.3 hnamodred

[*Gr*, *info*] = hnamodred (*G*, ...) [Function File]
 [*Gr*, *info*] = hnamodred (*G*, *nr*, ...) [Function File]
 [*Gr*, *info*] = hnamodred (*G*, *opt*, ...) [Function File]
 [*Gr*, *info*] = hnamodred (*G*, *nr*, *opt*, ...) [Function File]

Model order reduction by frequency weighted optimal Hankel-norm (HNA) method. The aim of model reduction is to find an LTI system Gr of order nr ($nr < n$) such that the input-output behaviour of Gr approximates the one from original system G .

HNA is an absolute error method which tries to minimize

$$\|G - G_r\|_H = \min$$

$$\|V (G - G_r) W\|_H = \min$$

where V and W denote output and input weightings.

Inputs

G LTI model to be reduced.

nr The desired order of the resulting reduced order system Gr . If not specified, nr is chosen automatically according to the description of key "order".

... Optional pairs of keys and values. "key1", value1, "key2", value2.
opt Optional struct with keys as field names. Struct *opt* can be created directly or by command *options*. *opt.key1* = value1, *opt.key2* = value2.

Outputs

Gr Reduced order state-space model.
info Struct containing additional information.
 info.n The order of the original system *G*.
 info.ns The order of the *alpha*-stable subsystem of the original system *G*.
 info.hsv The Hankel singular values corresponding to the projection $\text{op}(V) * G1 * \text{op}(W)$, where *G1* denotes the *alpha*-stable part of the original system *G*. The *ns* Hankel singular values are ordered decreasingly.
 info.nu The order of the *alpha*-unstable subsystem of both the original system *G* and the reduced-order system *Gr*.
 info.nr The order of the obtained reduced order system *Gr*.

Option Keys and Values

'order', 'nr' The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is the sum of *info.nu* and the number of Hankel singular values greater than $\max(\text{tol1}, \text{ns} * \text{eps} * \text{info.hsv}(1))$;
 'method' Specifies the computational approach to be used. Valid values corresponding to this key are:
 'descriptor' Use the inverse free descriptor system approach.
 'standard' Use the inversion based standard approach.
 'auto' Switch automatically to the inverse free descriptor approach in case of badly conditioned feedthrough matrices in *V* or *W*. Default method.
 'left', 'v' LTI model of the left/output frequency weighting. The weighting must be anti-stable. $\|V (G - G_r) \dots\|_H = \min$
 'right', 'w' LTI model of the right/input frequency weighting. The weighting must be anti-stable. $\|\dots (G - G_r) W\|_H = \min$
 'left-inv', 'inv-v' LTI model of the left/output frequency weighting. The weighting must have only antistable zeros. $\|inv(V) (G - G_r) \dots\|_H = \min$
 'right-inv', 'inv-w' LTI model of the right/input frequency weighting. The weighting must have only antistable zeros. $\|\dots (G - G_r) inv(W)\|_H = \min$
 'left-conj', 'conj-v' LTI model of the left/output frequency weighting. The weighting must be stable. $\|conj(V) (G - G_r) \dots\|_H = \min$
 'right-conj', 'conj-w' LTI model of the right/input frequency weighting. The weighting must be stable. $\|\dots (G - G_r) conj(W)\|_H = \min$

'left-conj-inv', 'conj-inv-v'

LTI model of the left/output frequency weighting. The weighting must be minimum-phase. $\|conj(inv(V)) (G - G_r) \dots\|_H = min$

'right-conj-inv', 'conj-inv-w'

LTI model of the right/input frequency weighting. The weighting must be minimum-phase. $\|\dots (G - G_r) conj(inv(W))\|_H = min$

'alpha'

Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix *G.A*. For a continuous-time system, ALPHA ≤ 0 is the boundary value for the real parts of eigenvalues, while for a discrete-time system, $0 \leq \text{ALPHA} \leq 1$ represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time systems and 1 for discrete-time systems.

'tol1'

If 'order' is not specified, *tol1* contains the tolerance for determining the order of the reduced model. For model reduction, the recommended value of *tol1* is $c \cdot \text{info.hsv}(1)$, where *c* lies in the interval $[0.00001, 0.001]$. $tol1 < 1$. If 'order' is specified, the value of *tol1* is ignored.

'tol2'

The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given model. $tol2 \leq tol1 < 1$. If not specified, $ns \cdot \text{eps} \cdot \text{info.hsv}(1)$ is chosen.

'equil', 'scale'

Boolean indicating whether equilibration (scaling) should be performed on system *G* prior to order reduction. Default value is true if *G.scaled* == false and false if *G.scaled* == true. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the *prescale* command because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Approximation Properties:

- Guaranteed stability of reduced models
- Lower guaranteed error bound
- Guaranteed a priori error bound

$$\sigma_{r+1} \leq \|(G - G_r)\|_\infty \leq 2 \sum_{j=r+1}^n \sigma_j$$

Algorithm

Uses SLICOT AB09JD by courtesy of **NICONET e.V.**

14.4 spamodred

[Gr, info] = spamodred (G, ...)	[Function File]
[Gr, info] = spamodred (G, nr, ...)	[Function File]
[Gr, info] = spamodred (G, opt, ...)	[Function File]
[Gr, info] = spamodred (G, nr, opt, ...)	[Function File]

Model order reduction by frequency weighted Singular Perturbation Approximation (SPA). The aim of model reduction is to find an LTI system *Gr* of order *nr* ($nr < n$) such that the input-output behaviour of *Gr* approximates the one from original system *G*.

SPA is an absolute error method which tries to minimize

$$\|G - G_r\|_\infty = \min$$

$$\|V (G - G_r) W\|_\infty = \min$$

where V and W denote output and input weightings.

Inputs

G	LTI model to be reduced.
nr	The desired order of the resulting reduced order system G_r . If not specified, nr is chosen automatically according to the description of key 'order'.
\dots	Optional pairs of keys and values. "key1", value1, "key2", value2.
opt	Optional struct with keys as field names. Struct opt can be created directly or by command <code>options</code> . <code>opt.key1 = value1</code> , <code>opt.key2 = value2</code> .

Outputs

G_r	Reduced order state-space model.
$info$	Struct containing additional information.
$info.n$	The order of the original system G .
$info.ns$	The order of the <i>alpha</i> -stable subsystem of the original system G .
$info.hsv$	The Hankel singular values of the <i>alpha</i> -stable part of the original system G , ordered decreasingly.
$info.nu$	The order of the <i>alpha</i> -unstable subsystem of both the original system G and the reduced-order system G_r .
$info.nr$	The order of the obtained reduced order system G_r .

Option Keys and Values

'order', 'nr'	The desired order of the resulting reduced order system G_r . If not specified, nr is chosen automatically such that states with Hankel singular values $info.hsv > toll$ are retained.
'left', 'output'	LTI model of the left/output frequency weighting V . Default value is an identity matrix.
'right', 'input'	LTI model of the right/input frequency weighting W . Default value is an identity matrix.
'method'	Approximation method for the L-infinity norm to be used as follows:
'sr', 's'	Use the square-root Singular Perturbation Approximation method.
'bfsr', 'p'	Use the balancing-free square-root Singular Perturbation Approximation method. Default method.
'alpha'	Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix $G.A$. For a continuous-time system, $ALPHA \leq 0$ is the boundary value for the real parts of eigenvalues, while for a discrete-time system, $0 \leq ALPHA \leq 1$ represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time systems and 1 for discrete-time systems.

- 'tol1'** If **'order'** is not specified, *tol1* contains the tolerance for determining the order of the reduced model. For model reduction, the recommended value of *tol1* is $c \cdot \text{info.hsv}(1)$, where c lies in the interval $[0.00001, 0.001]$. Default value is $\text{info.ns} \cdot \text{eps} \cdot \text{info.hsv}(1)$. If **'order'** is specified, the value of *tol1* is ignored.
- 'tol2'** The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given model. $\text{TOL2} \leq \text{TOL1}$. If not specified, $\text{ns} \cdot \text{eps} \cdot \text{info.hsv}(1)$ is chosen.
- 'gram-ctrb'** Specifies the choice of frequency-weighted controllability Grammian as follows:
- 'standard'** Choice corresponding to a combination method [4] of the approaches of Enns [1] and Lin-Chiu [2,3]. Default method.
- 'enhanced'** Choice corresponding to the stability enhanced modified combination method of [4].
- 'gram-obsv'** Specifies the choice of frequency-weighted observability Grammian as follows:
- 'standard'** Choice corresponding to a combination method [4] of the approaches of Enns [1] and Lin-Chiu [2,3]. Default method.
- 'enhanced'** Choice corresponding to the stability enhanced modified combination method of [4].
- 'alpha-ctrb'** Combination method parameter for defining the frequency-weighted controllability Grammian. $\text{abs}(\text{alphac}) \leq 1$. If $\text{alphac} = 0$, the choice of Grammian corresponds to the method of Enns [1], while if $\text{alphac} = 1$, the choice of Grammian corresponds to the method of Lin and Chiu [2,3]. Default value is 0.
- 'alpha-obsv'** Combination method parameter for defining the frequency-weighted observability Grammian. $\text{abs}(\text{alphao}) \leq 1$. If $\text{alphao} = 0$, the choice of Grammian corresponds to the method of Enns [1], while if $\text{alphao} = 1$, the choice of Grammian corresponds to the method of Lin and Chiu [2,3]. Default value is 0.
- 'equil', 'scale'** Boolean indicating whether equilibration (scaling) should be performed on system G prior to order reduction. Default value is true if $G.\text{scaled} == \text{false}$ and false if $G.\text{scaled} == \text{true}$. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the **prescale** command because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

References

- [1] Enns, D. *Model reduction with balanced realizations: An error bound and a frequency weighted generalization*. Proc. 23-th CDC, Las Vegas, pp. 127-132, 1984.
- [2] Lin, C.-A. and Chiu, T.-Y. *Model reduction via frequency-weighted balanced realization*. Control Theory and Advanced Technology, vol. 8, pp. 341-351, 1992.
- [3] Sreeram, V., Anderson, B.D.O and Madieviski, A.G. *New results on frequency weighted balanced reduction technique*. Proc. ACC, Seattle, Washington, pp. 4004-4009, 1995.

- [4] Varga, A. and Anderson, B.D.O. *Square-root balancing-free methods for the frequency-weighted balancing related model reduction*. (report in preparation)

Algorithm

Uses SLICOT AB09ID by courtesy of **NICONET e.V.**

15 Controller Reduction

15.1 btaconred

```
[Kr, info] = btaconred (G, K, ...) [Function File]
[Kr, info] = btaconred (G, K, ncr, ...) [Function File]
[Kr, info] = btaconred (G, K, opt, ...) [Function File]
[Kr, info] = btaconred (G, K, ncr, opt, ...) [Function File]
```

Controller reduction by frequency-weighted Balanced Truncation Approximation (BTA). Given a plant G and a stabilizing controller K , determine a reduced order controller K_r such that the closed-loop system is stable and closed-loop performance is retained.

The algorithm tries to minimize the frequency-weighted error

$$\|V (K - K_r) W\|_{\infty} = \min$$

where V and W denote output and input weightings.

Inputs

G LTI model of the plant. It has m inputs, p outputs and n states.

K LTI model of the controller. It has p inputs, m outputs and nc states.

ncr The desired order of the resulting reduced order controller K_r . If not specified, ncr is chosen automatically according to the description of key 'order'.

\dots Optional pairs of keys and values. "key1", value1, "key2", value2.

opt Optional struct with keys as field names. Struct opt can be created directly or by command options. $opt.key1 = value1$, $opt.key2 = value2$.

Outputs

K_r State-space model of reduced order controller.

$info$ Struct containing additional information.

$info.ncr$ The order of the obtained reduced order controller K_r .

$info.ncs$ The order of the alpha-stable part of original controller K .

$info.hsvc$ The Hankel singular values of the alpha-stable part of K . The ncs Hankel singular values are ordered decreasingly.

Option Keys and Values

'order', 'ncr' The desired order of the resulting reduced order controller K_r . If not specified, ncr is chosen automatically such that states with Hankel singular values $info.hsvc > toll$ are retained.

'method' Order reduction approach to be used as follows:

'sr', 'b' Use the square-root Balance & Truncate method.

'bfsr', 'f' Use the balancing-free square-root Balance & Truncate method. Default method.

'weight' Specifies the type of frequency-weighting as follows:

'none' No weightings are used ($V = I$, $W = I$).

'left', 'output'

Use stability enforcing left (output) weighting

$$V = (I - GK)^{-1}G, \quad W = I$$

'right', 'input'

Use stability enforcing right (input) weighting

$$V = I, \quad W = (I - GK)^{-1}G$$

'both', 'performance'

Use stability and performance enforcing weightings

$$V = (I - GK)^{-1}G, \quad W = (I - GK)^{-1}$$

Default value.

'feedback' Specifies whether K is a positive or negative feedback controller:

'+'

Use positive feedback controller. Default value.

'-'

Use negative feedback controller.

'alpha'

Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix $K.A$. For a continuous-time controller, $ALPHA \leq 0$ is the boundary value for the real parts of eigenvalues, while for a discrete-time controller, $0 \leq ALPHA \leq 1$ represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time controllers and 1 for discrete-time controllers.

'tol1'

If *'order'* is not specified, *tol1* contains the tolerance for determining the order of the reduced controller. For model reduction, the recommended value of *tol1* is $c \cdot \text{info.hsvc}(1)$, where c lies in the interval $[0.00001, 0.001]$. Default value is $\text{info.ncs} \cdot \text{eps} \cdot \text{info.hsvc}(1)$. If *'order'* is specified, the value of *tol1* is ignored.

'tol2'

The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given controller. $TOL2 \leq TOL1$. If not specified, $\text{ncs} \cdot \text{eps} \cdot \text{info.hsvc}(1)$ is chosen.

'gram-ctrb'

Specifies the choice of frequency-weighted controllability Grammian as follows:

'standard'

Choice corresponding to standard Enns' method [1]. Default method.

'enhanced'

Choice corresponding to the stability enhanced modified Enns' method of [2].

'gram-obsv'

Specifies the choice of frequency-weighted observability Grammian as follows:

'standard'

Choice corresponding to standard Enns' method [1]. Default method.

'enhanced'

Choice corresponding to the stability enhanced modified Enns' method of [2].

'*equil*', '*scale*'

Boolean indicating whether equilibration (scaling) should be performed on G and K prior to order reduction. Default value is false if both `G.scaled == true`, `K.scaled == true` and true otherwise. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` command because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Algorithm

Uses SLICOT SB16AD by courtesy of **NICONET e.V.**

15.2 cfconred

<code>[Kr, info] = cfconred (G, F, L, ...)</code>	[Function File]
<code>[Kr, info] = cfconred (G, F, L, ncr, ...)</code>	[Function File]
<code>[Kr, info] = cfconred (G, F, L, opt, ...)</code>	[Function File]
<code>[Kr, info] = cfconred (G, F, L, ncr, opt, ...)</code>	[Function File]

Reduction of state-feedback-observer based controller by coprime factorization (CF). Given a plant G , state feedback gain F and full observer gain L , determine a reduced order controller Kr .

Inputs

G	LTI model of the open-loop plant (A,B,C,D). It has m inputs, p outputs and n states.
F	Stabilizing state feedback matrix (m -by- n).
L	Stabilizing observer gain matrix (n -by- p).
ncr	The desired order of the resulting reduced order controller Kr . If not specified, ncr is chosen automatically according to the description of key ' <i>order</i> '.
\dots	Optional pairs of keys and values. " <i>key1</i> ", <i>value1</i> , " <i>key2</i> ", <i>value2</i> .
<i>opt</i>	Optional struct with keys as field names. Struct <i>opt</i> can be created directly or by command <code>options</code> . <code>opt.key1 = value1</code> , <code>opt.key2 = value2</code> .

Outputs

Kr	State-space model of reduced order controller.
<i>info</i>	Struct containing additional information.
<i>info.hsv</i>	The Hankel singular values of the extended system?!?. The n Hankel singular values are ordered decreasingly.
<i>info.ncr</i>	The order of the obtained reduced order controller Kr .

Option Keys and Values

'*order*', '*ncr*'

The desired order of the resulting reduced order controller Kr . If not specified, ncr is chosen automatically such that states with Hankel singular values *info.hsv* $> tol1$ are retained.

'*method*' Order reduction approach to be used as follows:

'*sr-bta*', '*b*'

Use the square-root Balance & Truncate method.

<code>'bfsr-bta', 'f'</code>	Use the balancing-free square-root Balance & Truncate method. Default method.
<code>'sr-spa', 's'</code>	Use the square-root Singular Perturbation Approximation method.
<code>'bfsr-spa', 'p'</code>	Use the balancing-free square-root Singular Perturbation Approximation method.
<code>'cf'</code>	Specifies whether left or right coprime factorization is to be used as follows:
<code>'left', 'l'</code>	Use left coprime factorization. Default method.
<code>'right', 'r'</code>	Use right coprime factorization.
<code>'feedback'</code>	Specifies whether F and L are fed back positively or negatively:
<code>'+'</code>	$A+BK$ and $A+LC$ are both Hurwitz matrices.
<code>'-'</code>	$A-BK$ and $A-LC$ are both Hurwitz matrices. Default value.
<code>'tol1'</code>	If <code>'order'</code> is not specified, <code>tol1</code> contains the tolerance for determining the order of the reduced system. For model reduction, the recommended value of <code>tol1</code> is <code>c*info.hsv(1)</code> , where <code>c</code> lies in the interval $[0.00001, 0.001]$. Default value is <code>n*eps*info.hsv(1)</code> . If <code>'order'</code> is specified, the value of <code>tol1</code> is ignored.
<code>'tol2'</code>	The tolerance for determining the order of a minimal realization of the coprime factorization controller. $TOL2 \leq TOL1$. If not specified, <code>n*eps*info.hsv(1)</code> is chosen.
<code>'equil', 'scale'</code>	Boolean indicating whether equilibration (scaling) should be performed on system G prior to order reduction. Default value is true if <code>G.scaled == false</code> and false if <code>G.scaled == true</code> . Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can not be done by the equilibration option or the <code>prescale</code> command because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Algorithm

Uses SLICOT SB16BD by courtesy of [NICONET e.V.](#)

15.3 fwcfconred

<code>[Kr, info] = fwcfconred (G, F, L, ...)</code>	[Function File]
<code>[Kr, info] = fwcfconred (G, F, L, ncr, ...)</code>	[Function File]
<code>[Kr, info] = fwcfconred (G, F, L, opt, ...)</code>	[Function File]
<code>[Kr, info] = fwcfconred (G, F, L, ncr, opt, ...)</code>	[Function File]

Reduction of state-feedback-observer based controller by frequency-weighted coprime factorization (FW CF). Given a plant G , state feedback gain F and full observer gain L , determine a reduced order controller Kr by using stability enforcing frequency weights.

Inputs

G	LTI model of the open-loop plant (A,B,C,D). It has m inputs, p outputs and n states.
F	Stabilizing state feedback matrix (m -by- n).

L	Stabilizing observer gain matrix (n-by-p).
ncr	The desired order of the resulting reduced order controller Kr . If not specified, ncr is chosen automatically according to the description of key 'order'.
...	Optional pairs of keys and values. "key1", value1, "key2", value2.
opt	Optional struct with keys as field names. Struct opt can be created directly or by command options. $opt.key1 = value1$, $opt.key2 = value2$.

Outputs

Kr	State-space model of reduced order controller.
$info$	Struct containing additional information.
$info.hsv$	The Hankel singular values of the extended system!?. The n Hankel singular values are ordered decreasingly.
$info.ncr$	The order of the obtained reduced order controller Kr .

Option Keys and Values

'order', 'ncr'	The desired order of the resulting reduced order controller Kr . If not specified, ncr is chosen automatically such that states with Hankel singular values $info.hsv > tol1$ are retained.
'method'	Order reduction approach to be used as follows:
'sr', 'b'	Use the square-root Balance & Truncate method.
'bfsr', 'f'	Use the balancing-free square-root Balance & Truncate method. Default method.
'cf'	Specifies whether left or right coprime factorization is to be used as follows:
'left', 'l'	Use left coprime factorization.
'right', 'r'	Use right coprime factorization. Default method.
'feedback'	Specifies whether F and L are fed back positively or negatively:
'+'	$A+BK$ and $A+LC$ are both Hurwitz matrices.
'-'	$A-BK$ and $A-LC$ are both Hurwitz matrices. Default value.
'tol1'	If 'order' is not specified, $tol1$ contains the tolerance for determining the order of the reduced system. For model reduction, the recommended value of $tol1$ is $c*info.hsv(1)$, where c lies in the interval $[0.00001, 0.001]$. Default value is $n*eps*info.hsv(1)$. If 'order' is specified, the value of $tol1$ is ignored.

Algorithm

Uses SLICOT SB16CD by courtesy of [NICONET e.V.](#)

15.4 spaconred

$[Kr, info] = \text{spaconred}(G, K, \dots)$	[Function File]
$[Kr, info] = \text{spaconred}(G, K, ncr, \dots)$	[Function File]
$[Kr, info] = \text{spaconred}(G, K, opt, \dots)$	[Function File]
$[Kr, info] = \text{spaconred}(G, K, ncr, opt, \dots)$	[Function File]

Controller reduction by frequency-weighted Singular Perturbation Approximation (SPA). Given a plant G and a stabilizing controller K , determine a reduced order controller Kr such that the closed-loop system is stable and closed-loop performance is retained.

The algorithm tries to minimize the frequency-weighted error

$$\|V (K - K_r) W\|_{\infty} = \min$$

where V and W denote output and input weightings.

Inputs

G	LTI model of the plant. It has m inputs, p outputs and n states.
K	LTI model of the controller. It has p inputs, m outputs and nc states.
ncr	The desired order of the resulting reduced order controller K_r . If not specified, ncr is chosen automatically according to the description of key 'order'.
\dots	Optional pairs of keys and values. "key1", value1, "key2", value2.
opt	Optional struct with keys as field names. Struct opt can be created directly or by command <code>options</code> . <code>opt.key1 = value1</code> , <code>opt.key2 = value2</code> .

Outputs

K_r	State-space model of reduced order controller.
$info$	Struct containing additional information.
$info.ncr$	The order of the obtained reduced order controller K_r .
$info.ncs$	The order of the alpha-stable part of original controller K .
$info.hsvc$	The Hankel singular values of the alpha-stable part of K . The ncs Hankel singular values are ordered decreasingly.

Option Keys and Values

'order', 'ncr'	The desired order of the resulting reduced order controller K_r . If not specified, ncr is chosen automatically such that states with Hankel singular values $info.hsvc > tol1$ are retained.
'method'	Order reduction approach to be used as follows:
'sr', 's'	Use the square-root Singular Perturbation Approximation method.
'bfsr', 'p'	Use the balancing-free square-root Singular Perturbation Approximation method. Default method.
'weight'	Specifies the type of frequency-weighting as follows:
'none'	No weightings are used ($V = I$, $W = I$).
'left', 'output'	Use stability enforcing left (output) weighting
	$V = (I - GK)^{-1}G, \quad W = I$
'right', 'input'	Use stability enforcing right (input) weighting
	$V = I, \quad W = (I - GK)^{-1}G$
'both', 'performance'	Use stability and performance enforcing weightings
	$V = (I - GK)^{-1}G, \quad W = (I - GK)^{-1}$
	Default value.

- 'feedback'** Specifies whether K is a positive or negative feedback controller:
- '+'** Use positive feedback controller. Default value.
 - '-'** Use negative feedback controller.
- 'alpha'** Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix $K.A$. For a continuous-time controller, $\text{ALPHA} \leq 0$ is the boundary value for the real parts of eigenvalues, while for a discrete-time controller, $0 \leq \text{ALPHA} \leq 1$ represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time controllers and 1 for discrete-time controllers.
- 'tol1'** If **'order'** is not specified, *tol1* contains the tolerance for determining the order of the reduced controller. For model reduction, the recommended value of *tol1* is $c \cdot \text{info.hsvc}(1)$, where c lies in the interval $[0.00001, 0.001]$. Default value is $\text{info.ncs} \cdot \text{eps} \cdot \text{info.hsvc}(1)$. If **'order'** is specified, the value of *tol1* is ignored.
- 'tol2'** The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given controller. $\text{TOL2} \leq \text{TOL1}$. If not specified, $\text{ncs} \cdot \text{eps} \cdot \text{info.hsvc}(1)$ is chosen.
- 'gram-ctrb'** Specifies the choice of frequency-weighted controllability Grammian as follows:
- 'standard'** Choice corresponding to standard Enns' method [1]. Default method.
 - 'enhanced'** Choice corresponding to the stability enhanced modified Enns' method of [2].
- 'gram-obsv'** Specifies the choice of frequency-weighted observability Grammian as follows:
- 'standard'** Choice corresponding to standard Enns' method [1]. Default method.
 - 'enhanced'** Choice corresponding to the stability enhanced modified Enns' method of [2].
- 'equil', 'scale'** Boolean indicating whether equilibration (scaling) should be performed on G and K prior to order reduction. Default value is false if both `G.scaled == true`, `K.scaled == true` and true otherwise. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` command because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Algorithm

Uses SLICOT SB16AD by courtesy of **NICONET e.V.**

16 Experimental Data Handling

16.1 iddata

<code>dat = iddata (y)</code>	[Function File]
<code>dat = iddata (y, u)</code>	[Function File]
<code>dat = iddata (y, u, tsam, ...)</code>	[Function File]
<code>dat = iddata (y, u, [], ...)</code>	[Function File]

Create identification dataset of output and input signals.

Inputs

y Real matrix containing the output signal in time-domain. For a system with p outputs and n samples, y is a n -by- p matrix. For data from multiple experiments, y becomes a e -by-1 or 1-by- e cell vector of $n(i)$ -by- p matrices, where e denotes the number of experiments and $n(i)$ the individual number of samples for each experiment.

u Real matrix containing the input signal in time-domain. For a system with m inputs and n samples, u is a n -by- m matrix. For data from multiple experiments, u becomes a e -by-1 or 1-by- e cell vector of $n(i)$ -by- m matrices, where e denotes the number of experiments and $n(i)$ the individual number of samples for each experiment. If u is not specified or an empty element `[]` is passed, *dat* becomes a time series dataset.

tsam Sampling time. If not specified, default value -1 (unspecified) is taken. For multi-experiment data, *tsam* becomes a e -by-1 or 1-by- e cell vector containing individual sampling times for each experiment. If a scalar *tsam* is provided, then all experiments have the same sampling time.

... Optional pairs of properties and values.

Outputs

dat iddata identification dataset.

Option Keys and Values

`'expname'` The name of the experiments in *dat*. Cell vector of length e containing strings. Default names are `{'exp1', 'exp2', ...}`

`'y'` Output signals. See 'Inputs' for details.

`'outname'` The name of the output channels in *dat*. Cell vector of length p containing strings. Default names are `{'y1', 'y2', ...}`

`'outunit'` The units of the output channels in *dat*. Cell vector of length p containing strings.

`'u'` Input signals. See 'Inputs' for details.

`'inname'` The name of the input channels in *dat*. Cell vector of length m containing strings. Default names are `{'u1', 'u2', ...}`

`'inunit'` The units of the input channels in *dat*. Cell vector of length m containing strings.

`'tsam'` Sampling time. See 'Inputs' for details.

`'timeunit'` The units of the sampling times in *dat*. Cell vector of length e containing strings.

`'name'` String containing the name of the dataset.

`'notes'` String or cell of string containing comments.

`'userdata'` Any data type.

16.2 @iddata/cat

`dat = cat (dim, dat1, dat2, ...)` [Function File]

Concatenate iddata sets along dimension *dim*.

Inputs

dim Dimension along which the concatenation takes place.

- 1 Concatenate samples. The samples are concatenated in the following way: `dat.y{e} = [dat1.y{e}; dat2.y{e}; ...]` `dat.u{e} = [dat1.u{e}; dat2.u{e}; ...]` where *e* denotes the experiment. The number of experiments, outputs and inputs must be equal for all datasets. Equivalent to `vertcat`.
- 2 Concatenate inputs and outputs. The outputs and inputs are concatenated in the following way: `dat.y{e} = [dat1.y{e}, dat2.y{e}, ...]` `dat.u{e} = [dat1.u{e}, dat2.u{e}, ...]` where *e* denotes the experiment. The number of experiments and samples must be equal for all datasets. Equivalent to `horzcat`.
- 3 Concatenate experiments. The experiments are concatenated in the following way: `dat.y = [dat1.y; dat2.y; ...]` `dat.u = [dat1.u; dat2.u; ...]` The number of outputs and inputs must be equal for all datasets. Equivalent to `merge`.

dat1, dat2, ...

iddata sets to be concatenated.

Outputs

dat Concatenated iddata set.

See also: `horzcat`, `merge`, `vertcat`.

16.3 @iddata/detrend

`dat = detrend (dat)` [Function File]

`dat = detrend (dat, ord)` [Function File]

Detrend outputs and inputs of dataset *dat* by removing the best fit of a polynomial of order *ord*. If *ord* is not specified, default value 0 is taken. This corresponds to removing a constant.

16.4 @iddata/diff

`dat = diff (dat)` [Function File]

`dat = diff (dat, k)` [Function File]

Return *k*-th difference of outputs and inputs of dataset *dat*. If *k* is not specified, default value 1 is taken.

16.5 @iddata/fft

`dat = fft (dat)` [Function File]

`dat = fft (dat, n)` [Function File]

Transform iddata objects from time to frequency domain using a Fast Fourier Transform (FFT) algorithm.

Inputs

dat iddata identification dataset in time domain. In order to preserve signal power and noise level, the FFTs are normalized by multiplying each transform by the square root of the signal length.

16.9 @iddata/merge

`dat = merge (dat1, dat2, ...)` [Function File]

Concatenate experiments of iddata datasets. The experiments are concatenated in the following way: `dat.y = [dat1.y; dat2.y; ...]` `dat.u = [dat1.u; dat2.u; ...]` The number of outputs and inputs must be equal for all datasets.

16.10 @iddata/nkshift

`dat = nkshift (dat, nk)` [Function File]

`dat = nkshift (dat, nk, 'append')` [Function File]

Shift input channels of dataset *dat* according to integer *nk*. A positive value of *nk* means that the input channels are delayed *nk* samples. By default, both input and output signals are shortened by *nk* samples. If a third argument *'append'* is passed, the output signals are left untouched while *nk* zeros are appended to the (shortened) input signals such that the number of samples in *dat* remains constant.

16.11 @iddata/plot

`plot (dat)` [Function File]

`plot (dat, exp)` [Function File]

Plot signals of iddata identification datasets on the screen. The signals are plotted experiment-wise, either in time- or frequency-domain. For multi-experiment datasets, press any key to switch to the next experiment. If the plot of a single experiment should be saved by the `print` command, use `plot(dat,exp)`, where *exp* denotes the desired experiment.

16.12 @iddata/resample

`dat = resample (dat, p, q)` [Function File]

`dat = resample (dat, p, q, n)` [Function File]

`dat = resample (dat, p, q, h)` [Function File]

Change the sample rate of the output and input signals in dataset *dat* by a factor of *p/q*. This is performed using a polyphase algorithm. The anti-aliasing FIR filter can be specified as follows: Either by order *n* (scalar) with default value 0. The band edges are then chosen automatically. Or by impulse response *h* (vector). Requires the signal package to be installed.

Algorithm

Uses commands `fir1` and `resample` from the signal package.

References

- [1] J. G. Proakis and D. G. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, 4th ed., Prentice Hall, 2007. Chap. 6
- [2] A. V. Oppenheim, R. W. Schaffer and J. R. Buck, Discrete-time signal processing, Signal processing series, Prentice-Hall, 1999

16.13 @iddata/set

`set (dat)` [Function File]

`set (dat, "property", value, ...)` [Function File]

`dat = set (dat, "property", value, ...)` [Function File]
 Set or modify properties of iddata objects. If no return argument *dat* is specified, the modified LTI object is stored in input argument *dat*. **set** can handle multiple properties in one call: `set (dat, 'prop1', val1, 'prop2', val2, 'prop3', val3)`. `set (dat)` prints a list of the object's property names.

16.14 @iddata/size

`nvec = size (dat)` [Function File]
`ndim = size (dat, dim)` [Function File]
`[n, p, m, e] = size (dat)` [Function File]
 Return dimensions of iddata set *dat*.

Inputs

dat iddata set.
dim If given a second argument, **size** will return the size of the corresponding dimension.

Outputs

nvec Row vector. The first element is the total number of samples (rows of *dat.y* and *dat.u*). The second element is the number of outputs (columns of *dat.y*) and the third element the number of inputs (columns of *dat.u*). The fourth element is the number of experiments.
ndim Scalar value. The size of the dimension *dim*.
n Row vector containing the number of samples of each experiment.
p Number of outputs.
m Number of inputs.
e Number of experiments.

17 System Identification

17.1 arx

<code>[sys, x0] = arx (dat, n, ...)</code>	[Function File]
<code>[sys, x0] = arx (dat, n, opt, ...)</code>	[Function File]
<code>[sys, x0] = arx (dat, opt, ...)</code>	[Function File]
<code>[sys, x0] = arx (dat, 'na', na, 'nb', nb)</code>	[Function File]

Estimate ARX model using QR factorization.

$$A(q)y(t) = B(q)u(t) + e(t)$$

Inputs

dat iddata identification dataset containing the measurements, i.e. time-domain signals.

n The desired order of the resulting model sys.

... Optional pairs of keys and values. 'key1', value1, 'key2', value2.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by command `options`. `opt.key1 = value1`, `opt.key2 = value2`.

Outputs

sys Discrete-time transfer function model. If the second output argument *x0* is returned, *sys* becomes a state-space model.

x0 Initial state vector. If *dat* is a multi-experiment dataset, *x0* becomes a cell vector containing an initial state vector for each experiment.

Option Keys and Values

'na' Order of the polynomial A(q) and number of poles.

'nb' Order of the polynomial B(q)+1 and number of zeros+1. $nb \leq na$.

'nk' Input-output delay specified as number of sampling instants. Scalar positive integer. This corresponds to a call to command `nkshift`, followed by padding the B polynomial with *nk* leading zeros.

Algorithm

Uses the formulae given in [1] on pages 318-319, 'Solving for the LS Estimate by QR Factorization'. For the initial conditions, SLICOT IB01CD is used by courtesy of [NICONET e.V.](#)

References

[1] Ljung, L. (1999) System Identification - Theory for the User Second Edition Prentice Hall, New Jersey.

17.2 moen4

`[sys, x0, info] = moen4 (dat, ...)` [Function File]
`[sys, x0, info] = moen4 (dat, n, ...)` [Function File]
`[sys, x0, info] = moen4 (dat, opt, ...)` [Function File]
`[sys, x0, info] = moen4 (dat, n, opt, ...)` [Function File]

Estimate state-space model using combined subspace method: MOESP algorithm for finding the matrices A and C, and N4SID algorithm for finding the matrices B and D. If no output arguments are given, the singular values are plotted on the screen in order to estimate the system order.

Inputs

dat iddata set containing the measurements, i.e. time-domain signals.
n The desired order of the resulting state-space system *sys*. If not specified, *n* is chosen automatically according to the singular values and tolerances.
 \dots Optional pairs of keys and values. 'key1', value1, 'key2', value2.
opt Optional struct with keys as field names. Struct *opt* can be created directly or by command *options*. *opt.key1* = value1, *opt.key2* = value2.

Outputs

sys Discrete-time state-space model.
x0 Initial state vector. If *dat* is a multi-experiment dataset, *x0* becomes a cell vector containing an initial state vector for each experiment.
info Struct containing additional information.
 info.K Kalman gain matrix.
 info.Q State covariance matrix.
 info.Ry Output covariance matrix.
 info.S State-output cross-covariance matrix.
 info.L Noise variance matrix factor. $LL' = Ry$.

Option Keys and Values

'n' The desired order of the resulting state-space system *sys*. $s > n > 0$.
 's' The number of block rows *s* in the input and output block Hankel matrices to be processed. $s > 0$. In the MOESP theory, *s* should be larger than *n*, the estimated dimension of state vector.
 'alg', 'algorithm' Specifies the algorithm for computing the triangular factor R, as follows:
 'C' Cholesky algorithm applied to the correlation matrix of the input-output data. Default method.
 'F' Fast QR algorithm.
 'Q' QR algorithm applied to the concatenated block Hankel matrices.
 'tol' Absolute tolerance used for determining an estimate of the system order. If *tol* ≥ 0 , the estimate is indicated by the index of the last singular value greater than or equal to *tol*. (Singular values less than *tol* are considered as zero.) When *tol* = 0, an internally computed default value, $tol = s * eps * SV(1)$, is used, where *SV*(1) is the maximal singular value, and *eps* is the relative machine precision. When *tol* < 0, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default value is 0.

'*rcond*' The tolerance to be used for estimating the rank of matrices. If the user sets *rcond* > 0, the given value of *rcond* is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/*rcond* is considered to be of full rank. If the user sets *rcond* ≤ 0, then an implicitly computed, default tolerance, defined by *rcond* = m*n**eps*, is used instead, where *eps* is the relative machine precision. Default value is 0.

'*confirm*' Specifies whether or not the user's confirmation of the system order estimate is desired, as follows:

true User's confirmation.
false No confirmation. Default value.

'*noiseinput*' The desired type of noise input channels.

'*n*' No error inputs. Default value.

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

'*e*' Return *sys* as a (p-by-m+p) state-space model with both measured input channels *u* and noise channels *e* with covariance matrix *R_y*.

$$x_{k+1} = Ax_k + Bu_k + Ke_k$$

$$y_k = Cx_k + Du_k + e_k$$

'*v*' Return *sys* as a (p-by-m+p) state-space model with both measured input channels *u* and white noise channels *v* with identity covariance matrix.

$$x_{k+1} = Ax_k + Bu_k + KLv_k$$

$$y_k = Cx_k + Du_k + Lv_k$$

$$e = Lv, LL^T = R_y$$

'*k*' Return *sys* as a Kalman predictor for simulation.

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + K(y_k - \hat{y}_k)$$

$$\hat{y}_k = C\hat{x}_k + Du_k$$

$$\hat{x}_{k+1} = (A - KC)\hat{x}_k + (B - KD)u_k + Ky_k$$

$$\hat{y}_k = C\hat{x}_k + Du_k + 0y_k$$

Algorithm

Uses SLICOT IB01AD, IB01BD and IB01CD by courtesy of **NICONET e.V.**

17.3 moesp

`[sys, x0, info] = moesp (dat, ...)` [Function File]
`[sys, x0, info] = moesp (dat, n, ...)` [Function File]
`[sys, x0, info] = moesp (dat, opt, ...)` [Function File]
`[sys, x0, info] = moesp (dat, n, opt, ...)` [Function File]

Estimate state-space model using MOESP algorithm. MOESP: Multivariable Output Error State sPace. If no output arguments are given, the singular values are plotted on the screen in order to estimate the system order.

Inputs

dat iddata set containing the measurements, i.e. time-domain signals.
n The desired order of the resulting state-space system *sys*. If not specified, *n* is chosen automatically according to the singular values and tolerances.
 \dots Optional pairs of keys and values. 'key1', value1, 'key2', value2.
opt Optional struct with keys as field names. Struct *opt* can be created directly or by command `options`. `opt.key1 = value1`, `opt.key2 = value2`.

Outputs

sys Discrete-time state-space model.
x0 Initial state vector. If *dat* is a multi-experiment dataset, *x0* becomes a cell vector containing an initial state vector for each experiment.
info Struct containing additional information.
 info.K Kalman gain matrix.
 info.Q State covariance matrix.
 info.Ry Output covariance matrix.
 info.S State-output cross-covariance matrix.
 info.L Noise variance matrix factor. $LL' = Ry$.

Option Keys and Values

'n' The desired order of the resulting state-space system *sys*. $s > n > 0$.
 's' The number of block rows *s* in the input and output block Hankel matrices to be processed. $s > 0$. In the MOESP theory, *s* should be larger than *n*, the estimated dimension of state vector.
 'alg', 'algorithm'
 Specifies the algorithm for computing the triangular factor *R*, as follows:
 'C' Cholesky algorithm applied to the correlation matrix of the input-output data. Default method.
 'F' Fast QR algorithm.
 'Q' QR algorithm applied to the concatenated block Hankel matrices.
 'tol' Absolute tolerance used for determining an estimate of the system order. If $tol \geq 0$, the estimate is indicated by the index of the last singular value greater than or equal to *tol*. (Singular values less than *tol* are considered as zero.) When $tol = 0$, an internally computed default value, $tol = s * eps * SV(1)$, is used, where *SV*(1) is the maximal singular value, and *eps* is the relative machine precision. When $tol < 0$, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default value is 0.

- '*rcond*' The tolerance to be used for estimating the rank of matrices. If the user sets $rcond > 0$, the given value of $rcond$ is used as a lower bound for the reciprocal condition number; an m -by- n matrix whose estimated condition number is less than $1/rcond$ is considered to be of full rank. If the user sets $rcond \leq 0$, then an implicitly computed, default tolerance, defined by $rcond = m*n*eps$, is used instead, where eps is the relative machine precision. Default value is 0.
- '*confirm*' Specifies whether or not the user's confirmation of the system order estimate is desired, as follows:
- true* User's confirmation.
- false* No confirmation. Default value.
- '*noiseinput*' The desired type of noise input channels.
- '*n*' No error inputs. Default value.

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

- '*e*' Return *sys* as a (p-by-m+p) state-space model with both measured input channels *u* and noise channels *e* with covariance matrix R_y .

$$x_{k+1} = Ax_k + Bu_k + Ke_k$$

$$y_k = Cx_k + Du_k + e_k$$

- '*v*' Return *sys* as a (p-by-m+p) state-space model with both measured input channels *u* and white noise channels *v* with identity covariance matrix.

$$x_{k+1} = Ax_k + Bu_k + KLv_k$$

$$y_k = Cx_k + Du_k + Lv_k$$

$$e = Lv, \quad LL^T = R_y$$

- '*k*' Return *sys* as a Kalman predictor for simulation.

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + K(y_k - \hat{y}_k)$$

$$\hat{y}_k = C\hat{x}_k + Du_k$$

$$\hat{x}_{k+1} = (A - KC)\hat{x}_k + (B - KD)u_k + Ky_k$$

$$\hat{y}_k = C\hat{x}_k + Du_k + 0y_k$$

Algorithm

Uses SLICOT IB01AD, IB01BD and IB01CD by courtesy of [NICONET e.V.](#)

17.4 n4sid

`[sys, x0, info] = n4sid (dat, ...)` [Function File]
`[sys, x0, info] = n4sid (dat, n, ...)` [Function File]
`[sys, x0, info] = n4sid (dat, opt, ...)` [Function File]
`[sys, x0, info] = n4sid (dat, n, opt, ...)` [Function File]

Estimate state-space model using N4SID algorithm. N4SID: Numerical algorithm for Subspace State Space System IDentification. If no output arguments are given, the singular values are plotted on the screen in order to estimate the system order.

Inputs

dat iddata set containing the measurements, i.e. time-domain signals.
n The desired order of the resulting state-space system *sys*. If not specified, *n* is chosen automatically according to the singular values and tolerances.
 \dots Optional pairs of keys and values. 'key1', value1, 'key2', value2.
opt Optional struct with keys as field names. Struct *opt* can be created directly or by command `options`. `opt.key1 = value1`, `opt.key2 = value2`.

Outputs

sys Discrete-time state-space model.
x0 Initial state vector. If *dat* is a multi-experiment dataset, *x0* becomes a cell vector containing an initial state vector for each experiment.
info Struct containing additional information.
 info.K Kalman gain matrix.
 info.Q State covariance matrix.
 info.Ry Output covariance matrix.
 info.S State-output cross-covariance matrix.
 info.L Noise variance matrix factor. $LL' = Ry$.

Option Keys and Values

'n' The desired order of the resulting state-space system *sys*. $s > n > 0$.
 's' The number of block rows *s* in the input and output block Hankel matrices to be processed. $s > 0$. In the MOESP theory, *s* should be larger than *n*, the estimated dimension of state vector.
 'alg', 'algorithm'
 Specifies the algorithm for computing the triangular factor *R*, as follows:
 'C' Cholesky algorithm applied to the correlation matrix of the input-output data. Default method.
 'F' Fast QR algorithm.
 'Q' QR algorithm applied to the concatenated block Hankel matrices.
 'tol' Absolute tolerance used for determining an estimate of the system order. If $tol \geq 0$, the estimate is indicated by the index of the last singular value greater than or equal to *tol*. (Singular values less than *tol* are considered as zero.) When $tol = 0$, an internally computed default value, $tol = s * eps * SV(1)$, is used, where *SV*(1) is the maximal singular value, and *eps* is the relative machine precision. When $tol < 0$, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default value is 0.

- '*rcond*' The tolerance to be used for estimating the rank of matrices. If the user sets $rcond > 0$, the given value of $rcond$ is used as a lower bound for the reciprocal condition number; an m -by- n matrix whose estimated condition number is less than $1/rcond$ is considered to be of full rank. If the user sets $rcond \leq 0$, then an implicitly computed, default tolerance, defined by $rcond = m*n*eps$, is used instead, where eps is the relative machine precision. Default value is 0.
- '*confirm*' Specifies whether or not the user's confirmation of the system order estimate is desired, as follows:
- true* User's confirmation.
- false* No confirmation. Default value.
- '*noiseinput*' The desired type of noise input channels.
- '*n*' No error inputs. Default value.

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

- '*e*' Return *sys* as a (p-by-m+p) state-space model with both measured input channels *u* and noise channels *e* with covariance matrix R_y .

$$x_{k+1} = Ax_k + Bu_k + Ke_k$$

$$y_k = Cx_k + Du_k + e_k$$

- '*v*' Return *sys* as a (p-by-m+p) state-space model with both measured input channels *u* and white noise channels *v* with identity covariance matrix.

$$x_{k+1} = Ax_k + Bu_k + KLv_k$$

$$y_k = Cx_k + Du_k + Lv_k$$

$$e = Lv, LL^T = R_y$$

- '*k*' Return *sys* as a Kalman predictor for simulation.

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + K(y_k - \hat{y}_k)$$

$$\hat{y}_k = C\hat{x}_k + Du_k$$

$$\hat{x}_{k+1} = (A - KC)\hat{x}_k + (B - KD)u_k + Ky_k$$

$$\hat{y}_k = C\hat{x}_k + Du_k + 0y_k$$

Algorithm

Uses SLICOT IB01AD, IB01BD and IB01CD by courtesy of **NICONET e.V.**

18 Overloaded LTI Operators

18.1 @lti/ctranspose

Conjugate transpose or pertransposition of LTI objects. Used by Octave for "sys'". For a transfer-function matrix G , G' denotes the conjugate of G given by $G'(-s)$ for a continuous-time system or $G'(1/z)$ for a discrete-time system. The frequency response of the pertransposition of G is the Hermitian (conjugate) transpose of $G(j\omega)$, i.e. $\text{freqresp}(G', \omega) = \text{freqresp}(G, \omega)'$. **WARNING:** Do **NOT** use this for dual problems, use the transpose "sys.'" (note the dot) instead.

18.2 @lti/horzcat

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

18.3 @lti/inv

Inversion of LTI objects.

18.4 @lti/minus

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

18.5 @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".

18.6 @lti/mpower

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

18.7 @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".

18.8 @lti/mtimes

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

18.9 @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".

18.10 @lti/subsasgn

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

18.11 @lti/subsref

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

18.12 @lti/transpose

Transpose of LTI objects. Used by Octave for `"sys.{'}"`. Useful for dual problems, i.e. controllability and observability or designing estimator gains with `lqr` and `place`.

18.13 @lti/uminus

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

18.14 @lti/uplus

Unary plus of LTI object. Used by Octave for `"+sys"`.

18.15 @lti/vertcat

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

19 Overloaded IDDATA Operators

19.1 @iddata/horzcat

`dat = [dat1, dat2, ...]` [Function File]
`dat = horzcat (dat1, dat2, ...)` [Function File]

Horizontal concatenation of iddata datasets. The outputs and inputs are concatenated in the following way: `dat.y{e} = [dat1.y{e}, dat2.y{e}, ...]` `dat.u{e} = [dat1.u{e}, dat2.u{e}, ...]` where e denotes the experiment. The number of experiments and samples must be equal for all datasets.

19.2 @iddata/subsasgn

Subscripted assignment for iddata objects. Used by Octave for "dat.property = value".

19.3 @iddata/subsref

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

19.4 @iddata/vertcat

`dat = [dat1; dat2; ...]` [Function File]
`dat = vertcat (dat1, dat2, ...)` [Function File]

Vertical concatenation of iddata datasets. The samples are concatenated in the following way: `dat.y{e} = [dat1.y{e}; dat2.y{e}; ...]` `dat.u{e} = [dat1.u{e}; dat2.u{e}; ...]` where e denotes the experiment. The number of experiments, outputs and inputs must be equal for all datasets.

20 Miscellaneous

20.1 options

`opt = options ("key1", value1, "key2", value2, ...)` [Function File]
 Create options struct `opt` from a number of key and value pairs. For use with order reduction commands.

Inputs

key, property The name of the property.
value The value of the property.

Outputs

opt Struct with fields for each key.

Example

```
octave:1> opt = options ("method","spa", "tol", 1e-6)           =
opt =

    scalar structure containing the fields:

    method = spa
    tol = 1.0000e-06

octave:2> save filename opt                                   =
octave:3> # save the struct 'opt' to file 'filename' for later use
octave:4> load filename
octave:5> # load struct 'opt' from file 'filename'
```

20.2 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"}
```

20.3 test_control

`test_control` [Script File]

Execute all available tests at once. The Octave control package is based on the **SLICOT** library. SLICOT needs BLAS and LAPACK libraries which are also prerequisites for Octave itself. In case of failing tests, it is highly recommended to use **Netlib's reference BLAS** and **LAPACK** for building Octave. Using ATLAS may lead to sign changes in some entries of the state-space matrices. In general, these sign changes are not 'wrong' and can be regarded as the result of state transformations. Such state transformations (but not input/output transformations) have no influence on the input-output behaviour of the system. For better numerics, the control package uses such transformations by default when calculating the

frequency responses and a few other things. However, arguments like the Hankel singular Values (HSV) must not change. Differing HSVs and failing algorithms are known for using Framework Accelerate from Mac OS X 10.7.

20.4 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

20.5 Boeing707

```
sys = Boeing707 () [Function File]
```

Creates a linearized state-space model of a Boeing 707-321 aircraft at $v=80$ m/s ($M = 0.26$, $G_{\alpha 0} = -3^\circ$, $\alpha_0 = 4^\circ$, $\kappa = 50^\circ$).

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

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

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

20.6 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]

References

- [1] Skogestad, S. and Postlethwaite I. (2005) *Multivariable Feedback Control: Analysis and Design: Second Edition*. Wiley. http://www.nt.ntnu.no/users/skoge/book/2nd_edition/matlab_m/matfiles.html