control 2.2.0

Control Systems Package for GNU Octave

Lukas F. Reichlin

Table of Contents

1 Examples

1.1 MDSSystem

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

1.2 optiPID

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

2 Linear Time Invariant Models

2.1 dss

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

Create or convert to descriptor state-space model.

Inputs

sys LTI model to be converted to state-space.

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

b Input matrix (n-by-m).

c Measurement matrix (p-by-n).

d Feedthrough matrix (p-by-m).

e Descriptor matrix (n-by-n).

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

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

Outputs

sys Descriptor state-space model.

Equations

```
  .
E x = A x + B u
  y = C x + D u
```

ss, tf

2.2 frd

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

Create or convert to frequency response data.

Inputs

sys LTI model to be converted to frequency response data. If second argument *w* is omitted, the interesting frequency range is calculated by the zeros and poles of *sys*.

H Frequency response array (p-by-m-by-lw). H(i,j,k) contains the response from input j to output i at frequency k. In the SISO case, a vector (lw-by-1) or (1-by-lw) is accepted as well.

w Frequency vector (lw-by-1) in radian per second [rad/s]. Frequencies must be in ascending order.

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

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

Outputs

sys Frequency response data object.

dss, ss, tf

2.3 ss

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

Create or convert to state-space model.

Inputs

sys LTI model to be converted to state-space.

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

b Input matrix (n-by-m).

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

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

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

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

Outputs

sys State-space model.

Example

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

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

sys.b =
        u1
   V    10
   A    11
   kJ   12

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

sys.d =
         u1
   y1     0
   y2     0
   y3     0

Continuous-time model.
octave:5>
```

tf, dss

2.4 tf

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

Create or convert to transfer function model.

Inputs

sys LTI model to be converted to transfer function.

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

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

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

Outputs

sys Transfer function model.

Example

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

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

        1
 y1:  -----
      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 ...

       0.095
 y1:  -------
      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 ...

      s^2 + 5 s + 7
 y1:  -------------
      s^2 + 5 s + 6

          s + 7
 y2:  -------------
      s^2 + 8 s + 6

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

        1
 y1:  -----
      s + 2

      s^2 + 5 s + 5
 y2:  -------------
      s^2 + 3 s + 2

Continuous-time model.
octave:8>
```

ss, dss

2.5 zpk

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

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

Inputs

sys LTI model to be converted to transfer function.

z Cell of vectors containing the zeros for each channel. z{i,j} contains the zeros from input j to output i. In the SISO case, a single vector is accepted as well.

p Cell of vectors containing the poles for each channel. p{i,j} contains the poles from input j to output i. In the SISO case, a single vector is accepted as well.

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 *tsam* is not specified, a continuous-time model is assumed.

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

Outputs

sys Transfer function model.

tf, ss, dss, frd

3 Model Data Access

3.1 @lti/dssdata

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

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

Inputs

sys Any type of LTI model.

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

Outputs

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

b Input matrix (n-by-m).

c Measurement matrix (p-by-n).

d Feedthrough matrix (p-by-m).

e Descriptor matrix (n-by-n).

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

3.2 @lti/frdata

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

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

Inputs

sys Any type of LTI model.

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

Outputs

H Frequency response array (p-by-m-by-lw). H(i,j,k) contains the response from input j to output i at frequency k. In the SISO case, a vector (lw-by-1) is possible as well.

w Frequency vector (lw-by-1) in radian per second [rad/s]. Frequencies are in ascending order.

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

3.3 @lti/get

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

Access property values of LTI objects.

3.4 @lti/set

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

Set or modify properties of LTI objects.

3.5 @lti/ssdata

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

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

Inputs

sys Any type of LTI model.

Outputs

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

b Input matrix (n-by-m).

c Measurement matrix (p-by-n).

d Feedthrough matrix (p-by-m).

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

3.6 @lti/tfdata

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

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

Inputs

sys Any type of LTI model.

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

Outputs

num Cell of numerator(s). Each numerator is a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. num{i,j} contains the numerator polynomial from input j to output i. In the SISO case, a single vector is possible as well.

den Cell of denominator(s). Each denominator is a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. den{i,j} contains the denominator polynomial from input j to output i. In the SISO case, a single vector is possible as well.

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

3.7 @lti/zpkdata

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

Access zero-pole-gain data.

Inputs

sys Any type of LTI model.

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

Outputs

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

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

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

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

4 Model Conversions

4.1 @lti/c2d

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

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

Inputs

sys Continuous-time LTI model.

tsam Sampling time in seconds.

method Optional conversion method. If not specified, default method "*zoh*" is taken.

"*zoh*" Zero-order hold or matrix exponential.

"*tustin*", "*bilin*"
Bilinear transformation or Tustin approximation.

"*prewarp*" Bilinear transformation with pre-warping at frequency *w0*.

Outputs

sys Discrete-time LTI model.

4.2 @lti/d2c

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

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

Inputs

sys Discrete-time LTI model.

method Optional conversion method. If not specified, default method "*zoh*" is taken.

"*zoh*" Zero-order hold or matrix logarithm.

"*tustin*", "*bilin*"
Bilinear transformation or Tustin approximation.

"*prewarp*" Bilinear transformation with pre-warping at frequency *w0*.

Outputs

sys Continuous-time LTI model.

4.3 @lti/prescale

[*scaledsys*, *info*] = prescale (*sys*) [Function File]

Prescale state-space model. Uses SLICOT TB01ID and TG01AD by courtesy of NICONET e.V.. 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 = diag (info.SL)`.

info.SR Right scaling factors. `Tr = diag (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.
```

4.4 @lti/xperm

sys = xperm (*sys*, *st_idx*) [Function File]

Reorder states in state-space models.

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

```
  u    +         +--------+             y
------>(+)----->|  sys1  |-------+------->
        ^ -      +--------+       |
        |                         |
        |        +--------+       |
        +-------|  sys2  |<------+
                 +--------+
```

5.5 @lti/lft

sys = `lft` (*sys1*, *sys2*) [Function File]
sys = `lft` (*sys1*, *sys2*, *nu*, *ny*) [Function File]

Linear fractional tranformation, also known as Redheffer star product.

Inputs

sys1 Upper LTI model.

sys2 Lower LTI model.

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

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

Outputs

sys Resulting LTI model.

Block Diagram

```
             .............sys..............
             :         +--------+         :
      w1 ------------>|        |------------> z1
             :         |  sys1  |         :
             : u +---->|        |-----+ y :
             :   |     +--------+     |   :          Lower LFT
             :   |                    |   :
             :   |     +--------+     |   :          lft (sys1, sys2)
             :   +-----|  sys2  |<----+   :
             :         +--------+         :
             :............................:
```

```
             .............sys..............
             :         +--------+         :
             : u +---->|  sys1  |-----+ y :
             :   |     +--------+     |   :          Upper LFT
             :   |                    |   :
             :   |     +--------+     |   :          lft (sys1, sys2)
             :   +-----|        |<----+   :
             :         |  sys2  |         :
      z2 <------------|        |<------------ w2
             :         +--------+         :
             :............................:
```

```
          .............sys.............                         =
          :         +--------+         :
    w1 ------------>|        |------------> z1
          :         |  sys1  |         :
          : u +---->|        |-----+ y :
          :   |     +--------+     |   :
          :   |                    |   :         lft (sys1, sys2, nu, ny)
          :   |     +--------+     |   :
          :   +-----|        |<----+   :
          :         |  sys2  |         :
    z2 <------------|        |<------------ w2
          :         +--------+         :
          :............................:
```

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

    | -1      5      0 |
M = | pi      0      7 |
```

5.7 @lti/parallel

sys = parallel (*sys1*, *sys2*) [Function File]

Parallel connection of two LTI systems.

Block Diagram

```
    ..........................
    :      +--------+        :
    :  +-->|  sys1  |---+    :
 u  :  |   +--------+   | +  :  y
-------+                O--------->
    :  |   +--------+   | +  :
    :  +-->|  sys2  |---+    :
    :      +--------+        :
    :.........sys............:

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

```
    .....................................
 u  :  +--------+ y1    u2  +--------+  :  y
------>|  sys1  |---------->|  sys2  |------->
    :  +--------+           +--------+  :
    :.................sys.................

sys = series (sys1, sys2)
```

```
    .....................................
    :                   v2  +--------+  :
    :            ---------->|        |  :  y
    :  +--------+ y1    u2  |  sys2  |------->
 u  :  |        |---------->|        |  :
------>|  sys1  |       z1  +--------+  :
    :  |        |---------->            :
    :  +--------+                       :
    :.................sys.................

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

$$C_o = [BABA^2BldotsA^{n-1}B]$$

6.2 @lti/dcgain

k = dcgain (*sys*) [Function File]

DC gain of LTI model.

Inputs

sys LTI system.

Outputs

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

freqresp

6.3 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 $aWc + mWc' + bb' = 0$.

6.4 hsvd

hsv = hsvd (*sys*) [Function File]
hsv = hsvd (*sys*, "*offset*", *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. Uses SLICOT AB13AD by courtesy of NICONET e.V.

6.5 @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.

6.6 isctrb

bool = isctrb (*sys*) [Function File]
bool = isctrb (*sys*, *tol*) [Function File]
bool = isctrb (*a*, *b*) [Function File]
bool = isctrb (*a*, *b*, *e*) [Function File]
bool = isctrb (*a*, *b*, [], *tol*) [Function File]
bool = isctrb (*a*, *b*, *e*, *tol*) [Function File]

Logical check for system controllability. Uses SLICOT AB01OD and TG01HD by courtesy of NICONET e.V.

Inputs

sys LTI model.

a State transition matrix.

b Input matrix.

e Descriptor matrix.

tol Optional roundoff parameter. Default value is 0.

Outputs

bool = 0 System is not controllable.

bool = 1 System is controllable.

isobsv

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

Inputs

sys LTI system.

a State transition matrix.

c Measurement matrix.

e Descriptor matrix.

tol Optional tolerance for stability. Default value is 0.

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

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

Outputs

bool = 0 System is not detectable.

bool = 1 System is detectable.

See `isstabilizable` for description of computational method. isstabilizable, isstable, isctrb, isobsv

6.8 @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.

6.9 @lti/isminimumphase

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

Determine whether LTI system is minimum phase. If a square system *P* is minimum-phase, its inverse *P^-1* is stable.

6.10 isobsv

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

Logical check for system observability. Uses SLICOT AB01OD and TG01HD by courtesy of NICONET e.V.

Inputs

sys LTI model.

a State transition matrix.

c Measurement matrix.

e Descriptor matrix.

tol Optional roundoff parameter. Default value is 0.

Outputs

bool = 0 System is not observable.

bool = 1 System is observable.

isctrb

6.11 @lti/issiso

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

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

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

Inputs

sys LTI system.

a State transition matrix.

b Input matrix.

e Descriptor matrix.

tol Optional tolerance for stability. Default value is 0.

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

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

Outputs

bool = 0 System is not stabilizable.

bool = 1 System is stabilizable.

Method

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

isdetectable, isstable, isctrb, isobsv

6.13 @lti/isstable

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

Determine whether LTI system is stable.

6.14 @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. Uses SLICOT AB13BD and AB13DD by courtesy of NICONET e.V.

6.15 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

$$O_b = left[matrixCcrCAcrCA^2cr^dotscrCA^{n-1}ight]$$

6.16 @lti/pole

p = pole (*sys*) [Function File]

Compute poles of LTI system.

Inputs

sys LTI model.

Outputs

p Poles of *sys*.

6.17 pzmap

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

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

Inputs

sys LTI model.

Outputs

p Poles of *sys*.

z Transmission zeros of *sys*.

6.18 @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.

6.19 @lti/zero

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

Compute transmission zeros and gain of LTI model.

Inputs

sys LTI model.

Outputs

z Transmission zeros of *sys*.

k Gain of *sys*.

7 Model Simplification

7.1 @lti/minreal

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

Minimal realization or zero-pole cancellation of LTI models.

7.2 @lti/sminreal

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

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

Inputs

sys State-space model.

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

Outputs

sys Reduced state-space model.

minreal

8 Time Domain Analysis

8.1 covar

`[p, q] = covar (sys, w)` [Function File]

Return the steady-state covariance.

Inputs

sys LTI model.

w Intensity of white noise inputs which drive *sys*.

Outputs

p Output covariance.

q State covariance.

lyap, dlyap

8.2 gensig

`[u, t] = gensig (sigtype, tau)` [Function File]
`[u, t] = gensig (sigtype, tau, tfinal)` [Function File]
`[u, t] = gensig (sigtype, tau, tfinal, tsam)` [Function File]

Generate periodic signal. Useful in combination with lsim.

Inputs

sigtype = "*sin*"
Sine wave.

sigtype = "*cos*"
Cosine wave.

sigtype = "*square*"
Square wave.

sigtype = "*pulse*"
Periodic pulse.

tau Duration of one period in seconds.

tfinal Optional duration of the signal in seconds. Default duration is 5 periods.

tsam Optional sampling time in seconds. Default spacing is tau/64.

Outputs

u Vector of signal values.

t Time vector of the signal.

lsim

8.3 impulse

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

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

Inputs

sys LTI model.

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

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

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

Outputs

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

t Time row vector.

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

initial, lsim, step

8.4 initial

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

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

Inputs

sys State-space model.

x0 Vector of initial conditions for each state.

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

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

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

Outputs

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

t Time row vector.

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

Example

```
                                 .
Continuous Time:    x = A x ,    y = C x ,    x(0) = x0

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

=

impulse, lsim, step

8.5 lsim

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

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

Inputs

sys LTI model. System must be proper, i.e. it must not have more zeros than poles.

u Vector or array of input signal. Needs `length(t)` rows and as many columns as there are inputs. If *sys* is a single-input system, row vectors *u* of length `length(t)` are accepted as well.

t Time vector. Should be evenly spaced. If *sys* is a continuous-time system and *t* is a real scalar, *sys* is discretized with sampling time `tsam = t/(rows(u)-1)`. If *sys* is a discrete-time system and *t* is not specified, vector *t* is assumed to be `0 : tsam : tsam*(rows(u)-1)`.

x0 Vector of initial conditions for each state. If not specified, a zero vector is assumed.

method Discretization method for continuous-time models. Default value is zoh (zero-order hold). All methods from `c2d` are supported.

Outputs

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

t Time row vector. It is always evenly spaced.

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

impulse, initial, step

8.6 step

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

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

Inputs

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.

impulse, initial, lsim

9 Frequency Domain Analysis

9.1 bode

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

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

Inputs

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

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

Outputs

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

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

w Vector of frequency values used.

nichols, nyquist, sigma

9.2 bodemag

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

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

Inputs

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

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

Outputs

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

w Vector of frequency values used.

bode, nichols, nyquist, sigma

9.3 @lti/freqresp

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

Evaluate frequency response at given frequencies.

Inputs

sys LTI system.

w Vector of frequency values.

Outputs

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

dcgain

9.4 margin

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

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

Inputs

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

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

Outputs

gamma Gain margin (as gain, not dBs).

phi Phase margin (in degrees).

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

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

Equations

```
CONTINUOUS SYSTEMS                                                   =
Gain Margin
          _            _
L(jw) = L(jw)      BTW: L(jw) = L(-jw) = conj (L(jw))

num(jw)   num(-jw)
------- = --------
den(jw)   den(-jw)

num(jw) den(-jw) = num(-jw) den(jw)

imag (num(jw) den(-jw)) = 0
imag (num(-jw) den(jw)) = 0


Phase Margin                                                         =
          |num(jw)|
|L(jw)| = |-------| = 1
          |den(jw)|
  _     2      2
z z = Re z + Im z

num(jw)   num(-jw)
------- * -------- = 1
den(jw)   den(-jw)

num(jw) num(-jw) - den(jw) den(-jw) = 0

real (num(jw) num(-jw) - den(jw) den(-jw)) = 0
```

```
DISCRETE SYSTEMS                                                         =
Gain Margin
                                 jwT          log z
L(z) = L(1/z)      BTW: z = e    --> w = -----
                                               j T
num(z)   num(1/z)
------ = --------
den(z)   den(1/z)

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


Phase Margin                                                             =
         |num(z)|
|L(z)| = |------| = 1
         |den(z)|

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

num(z)   num(1/z)
------ * -------- = 1
den(z)   den(1/z)

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


PS: How to get L(1/z)                                                    =
         4       3       2
p(z) = a z  +  b z  +  c z  +  d z  +  e

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

          -4                      2       3       4
       = z   ( a  +  b z  +  c z  +  d z  +  e z  )

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

roots

9.5 nichols

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

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

Inputs

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

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

Outputs

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

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

w Vector of frequency values used.

bode, nyquist, sigma

9.6 nyquist

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

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

Inputs

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

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

Outputs

re Vector of real parts. Has length of frequency vector *w*.

im Vector of imaginary parts. Has length of frequency vector *w*.

w Vector of frequency values used.

bode, nichols, sigma

9.7 sigma

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

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

Inputs

sys LTI system. Multiple inputs and/or outputs (MIMO systems) make practical sense.

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

ptype = 0 Singular values of the frequency response *H* of system *sys*. Default Value.

ptype = 1 Singular values of the frequency response `inv(H)`; i.e. inversed system.

ptype = 2 Singular values of the frequency response `I + H`; i.e. inversed sensitivity (or return difference) if `H = P * C`.

ptype = 3 Singular values of the frequency response `I + inv(H)`; i.e. inversed complementary sensitivity if `H = P * C`.

Outputs

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

w Vector of frequency values used.

bodemag, svd

10 Pole Placement

10.1 place

f = place (*sys*, *p*) [Function File]
f = place (*a*, *b*, *p*) [Function File]
[*f*, *nfp*, *nap*, *nup*] = place (*sys*, *p*, *alpha*) [Function File]
[*f*, *nfp*, *nap*, *nup*] = place (*a*, *b*, *p*, *alpha*) [Function File]

Pole assignment for a given matrix pair (A,B) such that `p = eig (A-B*F)`. If parameter *alpha* is specified, poles with real parts (continuous-time) or moduli (discrete-time) below *alpha* are left untouched. Uses SLICOT SB01BD by courtesy of NICONET e.V.

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.

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`.

nap The number of assigned eigenvalues. `nap = n-nfp-nup`.

nup The number of uncontrollable eigenvalues detected by the eigenvalue assignment algorithm.

Note

```
Place is also suitable to design estimator gains:
        L = place (A.', C.', p).'
        L = place (sys.', p).'   # useful for discrete-time systems
```

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

```
  u    +         +---+       +-------+             y
------>(+)----->| k |----->| SISO |-------+------->
        ^ -      +---+       +-------+       |
        |                                    |
        +------------------------------------+
```

=

11 Linear-Quadratic Control

11.1 dlqr

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

Linear-quadratic regulator for discrete-time systems.

Inputs

sys Continuous or discrete-time LTI model.

a State transition matrix of discrete-time system.

b Input matrix of discrete-time system.

q State weighting matrix.

r Input weighting matrix.

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

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

Outputs

g State feedback matrix.

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

l Closed-loop poles.

Equations

```
x[k+1] = A x[k] + B u[k],   x[0] = x0

        inf
J(x0) = SUM (x' Q x  +  u' R u  +  2 x' S u)
        k=0

L = eig (A - B*G)
```

=

dare, care, lqr

11.2 estim

`est = estim (sys, l)` [Function File]
`est = estim (sys, l, sensors, known)` [Function File]

Return state estimator for a given estimator gain.

Inputs

sys LTI model.

l State feedback matrix.

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

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

Outputs

est State-space model of estimator.

kalman, place

11.3 kalman

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

Design Kalman estimator for LTI systems.

Inputs

sys Nominal plant model.

q Covariance of white process noise.

r Covariance of white measurement noise.

s Optional cross term covariance. Default value is 0.

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

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

Outputs

est State-space model of the Kalman estimator.

g Estimator gain.

x Solution of the Riccati equation.

Block Diagram

```
                                 u  +-------+         ^                =
      +--------------------------->|       |-------> y
      |    +-------+     +       y  |  est  |         ^
u ----+--->|       |----->(+)------>|       |-------> x
           |  sys  |       ^ +      +-------+
w -------->|       |       |
           +-------+       | v

Q = cov (w, w')     R = cov (v, v')     S = cov (w, v')
```

care, dare, estim, lqr

11.4 lqr

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

a State transition matrix of continuous-time system.

b Input matrix of continuous-time system.

q State weighting matrix.

r Input weighting matrix.

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

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

Outputs

g State feedback matrix.

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

l Closed-loop poles.

Equations

```
.
x = A x + B u,   x(0) = x0

        inf
J(x0) = INT (x' Q x  +  u' R u  +  2 x' S u)  dt
         0

L = eig (A - B*G)
```

=

care, dare, dlqr

12 Robust Control

12.1 augw

P = 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

```
                                                                      =
        | W1 | -W1*G |     z1 = W1 r  -  W1 G u
        | 0  |  W2   |     z2 =          W2   u
    P = | 0  |  W3*G |     z3 =          W3 G u
        |----+-------|
        | I  |    -G |     e  =    r  -     G u
```

```
                                                    +------+  z1      =
            +-------------------------------------->|  W1  |----->
            |                                       +------+
            |                                       +------+  z2
            |                 +-------------------->|  W2  |----->
            |                 |                     +------+
 r   +    e |   +---------+ u |   +---------+  y    +------+  z3
----->(+)---+-->|  K(s)   |----+-->|  G(s)   |----+---->|  W3  |----->
       ^ -      +---------+       +---------+    |     +------+
       |                                         |
       +-----------------------------------------+
```

```
                +--------+                                            =
                |        |-----> z1 (p1x1)         z1 = W1 e
 r (px1) ----->|  P(s)  |-----> z2 (p2x1)         z2 = W2 u
                |        |-----> z3 (p3x1)         z3 = W3 y
 u (mx1) ----->|        |-----> e (px1)           e = r - y
                +--------+
```

```
            +--------+
    r ----->|        |-----> z
            |  P(s)  |
    u +---->|        |-----+ e
      |     +--------+     |
      |                    |
      |     +--------+     |
      +-----|  K(s)  |<----+
            +--------+
```

```
Reference:
Skogestad, S. and Postlethwaite I.
Multivariable Feedback Control: Analysis and Design
Second Edition
Wiley 2005
Chapter 3.8: General Control Problem Formulation
```

h2syn, hinfsyn, mixsyn

12.2 h2syn

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

H-2 control synthesis for LTI plant. Uses SLICOT SB10HD and SB10ED by courtesy of NICONET e.V.

Inputs

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

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

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

Outputs

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

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

gamma H-2 norm of *N*.

Block Diagram

=

```
gamma = min||N(K)||             N = lft (P, K)
          K        2

                +---------+
        w ----->|         |-----> z
                |  P(s)   |
        u +---->|         |-----+ v
          |     +---------+     |
          |                     |
          |     +---------+     |
          +-----|  K(s)   |<----+
                +---------+

                +---------+
        w ----->|  N(s)   |-----> z
                +---------+
```

augw, lqr, dlqr, kalman

12.3 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. Uses SLICOT SB10FD and SB10DD by courtesy of NICONET e.V.

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*.

Block Diagram

=

```
gamma = min||N(K)||             N = lft (P, K)
         K        inf

               +--------+
       w ----->|        |-----> z
               |  P(s)  |
       u +---->|        |-----+ v
         |     +--------+     |
         |                    |
         |     +--------+     |
         +-----|  K(s)  |<----+
               +--------+

               +--------+
       w ----->|  N(s)  |-----> z
               +--------+
```

augw, mixsyn

12.4 mixsyn

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

Solve stacked S/KS/T H-inf 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*.

Block Diagram

```
                                                                       =
                                    | W1 S   |
gamma = min||N(K)||             N = | W2 K S | = lft (P, K)
         K         inf              | W3 T   |
```

```
                                                      +------+  z1     =
            +---------------------------------------->|  W1  |----->
            |                                         +------+
            |                                         +------+  z2
            |                 +---------------------->|  W2  |----->
            |                 |                       +------+
 r   +    e |   +---------+ u |   +---------+  y      +------+  z3
----->(+)---+-->|  K(s)   |---+-->|  G(s)   |---+---->|  W3  |----->
       ^ -      +---------+       +---------+   |     +------+
       |                                        |
       +----------------------------------------+
```

```
               +---------+                                             =
               |         |-----> z1 (p1x1)          z1 = W1 e
 r (px1) ----->|  P(s)   |-----> z2 (p2x1)          z2 = W2 u
               |         |-----> z3 (p3x1)          z3 = W3 y
 u (mx1) ----->|         |-----> e (px1)            e = r - y
               +---------+
```

```
               +---------+                                             =
       r ----->|         |-----> z
               |  P(s)   |
       u +---->|         |-----+ e
         |     +---------+     |
         |                     |
         |     +---------+     |
         +-----|  K(s)   |<----+
               +---------+
```

```
               +---------+                                             =
       r ----->|  N(s)   |-----> z
               +---------+
```

```
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)
```

```
Reference:
Skogestad, S. and Postlethwaite I.
Multivariable Feedback Control: Analysis and Design
Second Edition
Wiley 2005
Chapter 3.8: General Control Problem Formulation
```

hinfsyn, augw

12.5 ncfsyn

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

Normalized Coprime Factor (NCF) H-infinity synthesis. Compute positive feedback controller using the McFarlane/Glover Loop Shaping Design Procedure. Uses SLICOT SB10ID, SB10KD and SB10ZD by courtesy of NICONET e.V.

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*.

info Structure containing additional information.

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

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

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

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

Block Diagram of N

```
             ^ z1              ^ z2
             |                 |
 w1  +       |   +---------+   |          +---------+
----->(+)---+-->|   Ks    |----+--->(+)---->|   Gs    |----+
       ^ +       +---------+         ^      +---------+    |
       |                          w2 |                     |
       |                                                   |
       +---------------------------------------------------+
```

13 Matrix Equation Solvers

13.1 care

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

Solve continuous-time algebraic Riccati equation (ARE). Uses SLICOT SB02OD and SG02AD by courtesy of NICONET e.V.

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

```
                          -1                                                =
A'X + XA - XB R  B'X + Q = 0

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

      -1
G = R  B'X

      -1
G = R  (B'X + S')

L = eig (A - B*G)
```

```
                          -1
A'XE + E'XA - E'XB R   B'XE + Q = 0

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

      -1
G = R   B'XE

      -1
G = R   (B'XE + S)

L = eig (A - B*G, E)
```

=

dare, lqr, dlqr, kalman

13.2 dare

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

Solve discrete-time algebraic Riccati equation (ARE). Uses SLICOT SB02OD and SG02AD by courtesy of NICONET e.V.

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 discrete-time Riccati equation (n-by-n).

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

g Corresponding gain matrix (m-by-n).

Equations

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

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

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

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

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

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

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

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

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

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

care, lqr, dlqr, kalman

13.3 dlyap

x = dlyap (*a*, *b*) [Function File]
x = dlyap (*a*, *b*, *c*) [Function File]
x = dlyap (*a*, *b*, [], *e*) [Function File]

Solve discrete-time Lyapunov or Sylvester equations. Uses SLICOT SB03MD, SB04QD and SG03AD by courtesy of NICONET e.V.

Equations

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

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

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

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. Uses SLICOT SB03OD and SG03BD by courtesy of NICONET e.V.

Equations

```
A U' U A'  -  U' U  +  B B'  =  0           (Lyapunov Equation)        =

A U' U A'  -  E U' U E'  +  B B'  =  0      (Generalized Lyapunov Equation)
```

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. Uses SLICOT SB03MD, SB04MD and SG03AD by courtesy of NICONET e.V.

Equations

```
AX + XA' + B = 0      (Lyapunov Equation)                              =

AX + XB + C = 0       (Sylvester Equation)

AXE' + EXA' + B = 0   (Generalized Lyapunov Equation)
```

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. Uses SLICOT SB03OD and SG03BD by courtesy of NICONET e.V.

Equations

```
A U' U  +  U' U A'  +  B B'  =  0           (Lyapunov Equation)        =

A U' U E'  +  E U' U A'  +  B B'  =  0      (Generalized Lyapunov Equation)
```

lyap, dlyap, dlyapchol

14 Overloaded Operators

14.1 @lti/horzcat

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

14.2 @lti/inv

Inversion of LTI objects.

14.3 @lti/minus

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

14.4 @lti/mldivide

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

14.5 @lti/mpower

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

14.6 @lti/mrdivide

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

14.7 @lti/mtimes

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

14.8 @lti/plus

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

14.9 @lti/subsasgn

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

14.10 @lti/subsref

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

14.11 @lti/transpose

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

14.12 @lti/uminus

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

14.13 @lti/vertcat

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

15 Miscellaneous

15.1 strseq

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

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

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

15.2 test_control

test_control [Script File]

Execute all available tests at once.

15.3 BMWengine

sys = BMWengine () [Function File]
sys = BMWengine ("*scaled*") [Function File]
sys = BMWengine ("*unscaled*") [Function File]

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

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


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


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


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

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

15.4 Boeing707

sys = Boeing707 () [Function File]

Creates a linearized state-space model of a Boeing 707-321 aircraft at v=80 m/s ($M = 0.26$, $G_{a0} = -3^{circ}$, $\pi lpha_0 = 4^{circ}$, $kappa = 50^{circ}$).

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

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

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

15.5 WestlandLynx

sys = WestlandLynx () [Function File]

Model of the Westland Lynx Helicopter about hover.

```
INPUTS                                                          =
main rotor collective
longitudinal cyclic
lateral cyclic
tail rotor collective


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


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

```
Reference:                                                                  =
Skogestad, S. and Postlethwaite I.
Multivariable Feedback Control: Analysis and Design
Second Edition
Wiley 2005
http://www.nt.ntnu.no/users/skoge/book/2nd_edition/matlab_m/matfiles.html
```