

# Package ‘control’

December 12, 2017

**Type** Package

**Title** A Control Systems Toolbox

**Version** 0.2.5

**Author** Ben C. Ubah [aut, cre]

**Maintainer** Ben C. Ubah <ubah.ben22@gmail.com>

**Imports** pracma, expm, signal, Matrix, graphics, stats

**Description** Solves control systems problems relating to time/frequency response, LTI systems design and analysis, transfer function manipulations, and system conversion.

**License** GPL-2

**BugReports** <https://github.com/benubah/control/issues>

**LazyData** TRUE

**RoxygenNote** 6.0.1

**Suggests** testthat

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2017-12-12 09:35:34 UTC

## R topics documented:

abcdchk . . . . .	3
acker . . . . .	4
append . . . . .	5
bode . . . . .	6
c2d . . . . .	7
care . . . . .	8
cloop . . . . .	9
connect . . . . .	10
ctrb . . . . .	12
damp . . . . .	13
dcgain . . . . .	14
esort . . . . .	14

feedback . . . . .	15
freqresp . . . . .	16
gensig . . . . .	17
givens_rot . . . . .	18
impulse . . . . .	18
initial . . . . .	20
issiso . . . . .	22
lsim . . . . .	22
lsimplot . . . . .	23
ltifr . . . . .	24
ltitr . . . . .	25
nyquist . . . . .	26
obsv . . . . .	28
ordschur . . . . .	28
parallel . . . . .	29
pid . . . . .	30
place . . . . .	31
pole . . . . .	32
poly2str . . . . .	33
polysub . . . . .	34
ramp . . . . .	34
selectsys . . . . .	36
series . . . . .	37
ss . . . . .	38
ss2tf . . . . .	39
ss2zp . . . . .	40
ssdata . . . . .	42
step . . . . .	43
TF . . . . .	44
tf . . . . .	46
tf2ss . . . . .	47
tf2zp . . . . .	48
tfchk . . . . .	49
tfdata . . . . .	49
zp2ss . . . . .	50
zp2tf . . . . .	51
zpk . . . . .	52
zpkdata . . . . .	53

---

abcdchk                      *State-space matrices check.*

---

### Description

abcdchk verifies the dimensions of A,B,C,D matrices in its arguments, to ascertain that they are correctly defined.

### Usage

```
abcdchk(a, b, c, d)
```

### Arguments

a	An n x n matrix
b	An n x m matrix
c	An p x n matrix
d	An p x m matrix

### Details

This is a utility function that is always invoked by other functions to ascertain the dimensions of the arguments a,b,c,d and returns a message if there is an ill-defined entry.

### Value

Returns an empty string if matrix dimensions are consistent. Otherwise it returns the associated error message

### Examples

```
A <- rbind(c(0,1), c(-10000,-4))
B <- rbind(0,1)
C <- rbind(c(1,0), c(0,1))
D <- rbind(0,0)
message <- abcdchk(A,B,C,D)
```

---

`acker`*Pole placement gain selection using Ackermann's formula*

---

**Description**

Computes the Pole placement gain selection using Ackermann's formula.

**Usage**

```
acker(a, b, p)
```

**Arguments**

<code>a</code>	State-matrix of a state-space system
<code>b</code>	Input-matrix of a state-space system
<code>p</code>	closed loop poles

**Details**

`K <- ACKER(A,B,P)` calculates the feedback gain matrix `K` such that the single input system  $\dot{x} <- Ax + Bu$

with a feedback law of  $u <- -Kx$  has closed loop poles at the values specified in vector `P`, i.e., `P <- eigen(A - B * K)`.

This method is NOT numerically stable and a warning message is printed if the nonzero closed loop poles are greater than 10 in `P`.

**Examples**

```
F <- rbind(c(0,1),c(0,0))
G <- rbind(0,1)
H <- cbind(1,0);
J <- 0
t <- 1
sys <- ss(F,G, H,J)
A <- c2d(sys,t);
j <- sqrt(as.complex(-1));
pc <- rbind(0.78+0.18*j, 0.78-0.18*j)
K <- acker(A$A, A$B, pc)
```

---

append

*Append the dynamics of a set of systems*

---

### Description

append appends the dynamics of a set of n-state-space systems together

### Usage

```
append(...)
```

### Arguments

... Variable argument for LTI system models of tf, ss or zpk class

### Details

append(sys1, sys2, sys3, ... sysN) first combines the the first two systems and then goes on to combine the resulting state-space system to the next system and so forth. This is achieved by calling the sysgroup(sys1, sys2) at each iteration to group the systems in consecutive pairs until all systems are completely appended to form one system.

sysgroup(sys1, sys2) appends only two systems and is used by append

If a system is not in state-space representation, the function tries to form a state-space representation for such system.

### Value

The function returns a state-space model of the formed appended system with A, B, C, D matrices

### See Also

[series parallel feedback connect](#)

### Examples

```
sys1 <- ss(1,2,3,4)
sys2 <- ss(2,3,4,5)
sys3 <- ss(6,7,8,9)
append(sys1, sys2, sys3)
sys4 <- tf(1, c(1,2,5))
append(sys1, sys2, sys4)
```

---

**bode***Bode Frequency Response for continuous-time Linear Systems.*

---

**Description**

bode computes the magnitude and phase of the frequency response of system `sys` at given frequencies `w`

General Usage:

```
bode(sys)
```

```
bode(sys, w)
```

```
bode(sys, w, iu)
```

```
bode(sys, w = seq(0, 100, length = 10000), iu = 1)
```

```
bodeplot(sys)
```

```
bodeplot(sys, w)
```

```
bodeplot(sys, w, subtitle)
```

**Usage**

```
bode(sys, w, iu)
```

**Arguments**

<code>sys</code>	LTI system of transfer-function, state-space and zero-pole classes
<code>w</code>	vector of range of frequencies at the response is computed in rad/sec
<code>iu</code>	number to specify an input for a MIMO state-space system. If the system has 3 inputs, then <code>iu</code> would be set to 1, set to 2 and then to 3 to obtain the bode response from input 1, 2, and 3 to the outputs. For single input systems, <code>iu</code> is always set to 1. <code>iu</code> is not needed/allowed for calls to <code>bodeplot</code>

**Details**

bode Compute the magnitude and phase of the frequency response of system `sys` at given frequencies `w`. When `sys` is a transfer function, bode computes the frequency response of the system using the `signal` package.

`bodeplot` plots the frequency response computed by `bode`. For a MIMO state-space system, `bodeplot` uses `selectsys` to obtain the bode response for each input-to-output pair and plot them individually. This means that for a 2-input, 2-output system, `bodeplot` obtains the response for input 1 to output 1, input 1 to output 2, input 2 to output 1 and input 2 to output 2. `bodeplot` uses the `subtitle` argument to allow a user assign the plot a sub-title

**Value**

A list is returned by calling `bode` containing: `w` - frequencies

`mag` - magnitude of the response

`phase` - phase of the response

A plot is returned by calling `bodeplot`

**See Also**

[nyquist](#)

**Examples**

```
bode(tf(100, c(1,6,100)))
bode(ssdata(tf(100, c(1,6,100))))

bode(tf(4, c(1,1)))
A <- rbind(c(-2, -1), c(1,0)); B <- rbind(1,0);
C <- cbind(0,1); D <- as.matrix(0);
bode(ss(A,B,C,D))

## MIMO plot
A1 <- rbind(c(0,1), c(-25,-4)); B1 <- rbind(c(1,1), c(0,1))
C1 <- rbind(c(1,0), c(0,1)); D1 <- rbind(c(0,0), c(0,0))
sys1 <- ss(A1,B1,C1,D1)
bodeplot(sys1)
# Use: par(mfrow = c(2,1)); bodeplot(selectsys(sys1,1,2)) to obtain the response for a subsystem
# of sys1 for input 1 and output 2 only
# RESET your plot layout using par(mfrow = c(1,1))
```

---

c2d

*Continuous Time model conversion to Discrete Time model.*


---

**Description**

`c2d` converts a system in continuous-time model to a discrete time model

**Usage**

```
c2d(sys, t)
```

**Arguments**

`sys` An object of transfer function, state-space or zero-pole class

`t` Sample time; a numeric value greater than 0

**Details**

c2d converts the continuous-time system:  $x = Ax + Bu$  to the discrete-time state-space system:  $x[n+1] = \Phi * x[n] + \Gamma * u[n]$  based on the method of assuming a zero-order hold on the inputs and sample time. Transfer function and zero-pole systems are converted to state-space representation before conversion to discrete-time.

**Value**

Returns the provided system (transfer function, state-space or zero-pole) in an equivalent discrete-time.

**See Also**

[ltitr](#)

**Examples**

```
## for TF
c2d(tf(c(1,-1), c(1,4,5)), 0.1)
## for ZPK
sys <- zpndata( tf(c(1,-1), c(1,4,5)) )
c2d(sys, 0.1)
c2d(zpndata( tf(c(1,-1), c(1,4,5)) ), 0.1)
```

---

care

*Continuous-time Algebraic Riccati Equation solution*

---

**Description**

Computes the unique solution to the continuous-time Riccati equation:

$$A^* X + X^* A - X^* B * R^{-1} * B^* * X + Q^* Q = 0$$

**Usage**

```
care(A, B, Q, R = 1)
```

**Arguments**

A	State-matrix of a state-space system
B	Input-matrix of a state-space system
Q	Symmetric output-matrix of a state-space system
R	Single number



**Details**

`X <- care(A, B, Q, R)` returns the stabilizing solution (if it exists) to the continuous-time Riccati equation.

The `care` function also returns the gain matrix, `G` and a vector, `L` of the closed-loop eigenvalues, where

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

$$L = \text{eig}(a - b * g)$$

**Value**

Returns the stabilizing matrix, gain and closed-loop eigenvalues in a list.

**Note**

`A`, `B` must be controllable

**Examples**

```
a <- matrix(c(-3, 2, 1, 1), byrow = TRUE, ncol = 2)
b <- matrix(c(0, 1), nrow = 2)
c <- matrix(c(1, -1), ncol = 2)
q <- t(c) %*% c
r <- 3
care(a, b, q, r)
```

---

cloop

*Closed Feedback Loops*

---

**Description**

`cloop` forms a closed feedback loop for a state-space or transfer function system

**Usage**

```
cloop(sys, e, f)
```

**Arguments**

<code>sys</code>	LTI system model of transfer-function or state-space model
<code>e</code>	inputs vector
<code>f</code>	outputs vector

**Details**

Other possible usages of `cloop`:

```
cloop(sys)
```

```
cloop(sys, sgn)
```

If `sys` is a state-space model, `cloop(sys, SGN)` produces a state-space model of the closed-loop system obtained by feeding all the outputs of the system to all the inputs. Positive feedback is used when `SGN <- 1` and negative when `SGN <- -1`.

If `sys` is a transfer function model, `cloop(sys, SGN)` produces the SISO closed loop system in transfer function form obtained by unity feedback with the sign `SGN`.

`cloop(sys,OUTPUTS,INPUTS)` forms the closed loop system obtained by feeding the specific outputs into specific outputs. The vectors `OUTPUTS` and `INPUTS` contain indices into the outputs and inputs of the system respectively. Positive feedback is assumed. To form closed loop with negative feedback, negative values are used in the vector `INPUTS`.

**Value**

Returns a closed feedback loop system

**See Also**

[feedback](#)

**Examples**

```
J <- 2.0; b <- 0.04; K <- 1.0; R <- 0.08; L <- 1e-4
P <- TF("K/(s*((J*s + b)*(L*s + R) + K^2))")
cloop(P)
cloop(ss(1,2,3,4))
```

---

connect

*Block diagram interconnections of dynamic systems*

---

**Description**

`connect` is used to form a state-space model of a system from its block diagram.

**Usage**

```
connect(sysapp, q, inputs, outputs)
```

**Arguments**

sysapp	A state-space system containing several appended systems returned from the <a href="#">append</a> function. All appended systems must be in state-space model.
q	Matrix that specifies the interconnections of the block diagram. Each row specifies a connection. The first element of each row is the number of the block. The other following elements of each row specify where the block gets its summing inputs, with negative elements used to indicate minus inputs to the summing junction. For example: <code>cbind(2,1,3)</code> means that block 2 has an input from block 1 and block 3
inputs	A column matrix specifying the inputs of the resulting aggregate system
outputs	A column matrix specifying the outputs of the resulting aggregate system

**Details**

`connect` This function requires calling the [append](#) function to group a set of unconnected dynamics system in one system object. It then uses the `q` matrix to determine the interconnections between the systems and finally specifies the inputs and outputs for the new aggregate system. This approach helps to realize a block diagram as a single system on which further analysis could be performed. See examples below.

**Value**

Returns the interconnected system, returned as either a state-space model

**See Also**

[append series parallel feedback](#)

**Examples**

```
a1 <- rbind(c(0, 0), c(1,-3))
b1 <- rbind(-2,0)
c1 <- cbind(0,-1)
d <- as.matrix(0)
a2 <- as.matrix(-5)
b2 <- as.matrix(5)
c2 <- as.matrix(1)
d2 <- as.matrix(0)
sysa1 <- ss(a1, b1, c1, d)
sysa2 <- ss(a2, b2, c2, d2)
a1 <- append(sysa1, sysa2)
connect(a1, cbind(2,1,0), cbind(1,2), cbind(1,2))
## OR
connect(append(sysa1, sysa2), cbind(2,1,0), cbind(1), cbind(2))
## Not run:
cbind(2,1,0) means that block 2 has an input from block 1 and block 0 (which doesnt exist)
cbind(1) means that block 1 is the input of the system, and cbind(2) means block 2 is the
output of the system.
if we replace cbind(2) with cbind(1,2), this means that the system has two outputs from
block 1 and 2
```

```
i.e. \code{connect(append(sysa1, sysa2), cbind(2,1,0), cbind(1), cbind(1,2))}  
## End(Not run)
```

---

ctrb

*Form Controllability Matrix*

---

## Description

ctrb forms the controllability matrix.

## Usage

```
ctrb(A, B)
```

## Arguments

A	State matrix, A
B	State matrix, B

## Details

ctrb(a, b) returns the controllability matrix,  $[B \ AB \ A^2B \ \dots \ A^{(n-1)}B]$ . If the Controllability matrix has full row rank, the system is controllable.

## Value

Returns the controllability matrix.

## See Also

[obsv](#)

## Examples

```
a1 <- rbind(c(0,0),c(1,-3))  
b1 <- rbind(-2,0)  
ctrb(a1, b1)
```

**Description**

damp computes the natural frequency and damping for continuous systems.

**Usage**

```
damp(sys, doPrint = TRUE)
```

**Arguments**

sys	A Continuous-time system of state-space, transfer-function or zero-pole-gain model.
doPrint	If TRUE prints out the results. Default is TRUE.

**Details**

A table of the eigenvalues of the matrix  $a$ , the associated damping factors, the associated natural frequency (rad/s and Hz.) is displayed by calling the function.

When the continuous system is a state-space model, the eigenvalues of the state matrix are obtained and sorted. If the system is a transfer-function, the poles of the systems are obtained and sorted. For zero-pole systems, the poles are just extracted and sorted. The sorted eigenvalues are printed to output and used to obtain the natural frequencies and damping factors.

**Value**

Returns the natural frequencies and damping factors in a list:

omegan = Natural Frequencies (rad/s) zeta = Damping Factors

**See Also**

[esort](#)

**Examples**

```
sys1 <- tf(1, c(1,2,5))  
damp(sys1)
```

---

`dcgain`*DC Gain*

---

**Description**

`dcgain` Forms the Givens rotation matrix

**Usage**

`dcgain(sys)`

**Arguments**

`sys`                    A transfer function or state-space model

**Details**

`dcgain(sys)` Computes the steady-state gain (or low frequency gain) of a continuous system.

**Value**

Returns the gain.

---

`esort`*Sort Complex Continuous Eigenvalues in Descending Order*

---

**Description**

`esort` sorts the complex continuous eigenvalues in descending order

**Usage**

`esort(p)`

**Arguments**

`p`                    A vector containing the poles of a transfer-function, zero-pole model or the eigenvalues of a state-matrix

**Details**

`esort` sorts the complex eigenvalues based on their real part. The unstable eigenvalues (positive real part) are first shown.

This function is used to sort eigenvalues and system poles in [damp](#)

**Value**

Returns the sorted eigenvalues and the corresponding indices in a list:

$s$  = sorted eigenvalues  $idx$  = index

**See Also**

[damp](#)

---

 feedback

---

*Feedback Connection of LTI systems*


---

**Description**

feedback forms a feedback connection for two LTI state-space or transfer function systems

**Usage**

```
feedback(sys1, sys2, in1, out1)
```

**Arguments**

sys1	LTI system model of transfer-function or state-space model
sys2	LTI system model of transfer-function or state-space model
in1	vector of inputs
out1	vector of outputs

**Details**

When `sys1` and `sys2` are transfer functions `feedback(sys1, sys2, SIGN)` produces the SISO closed loop system in transfer function form obtained by connecting the two SISO transfer function systems in feedback with the sign `SIGN`.

`feedback(sys1, sys2, SIGN)` produces an aggregate state-space system consisting of the feedback connection of the two systems 1 and 2. If `SIGN = 1` then positive feedback is used. If `SIGN = -1` then negative feedback is used. In all cases, the resulting system has the same inputs and outputs as system 1.

`feedback(sys1, sys2, inputs, outputs)` produces the feedback system formed by feeding all the outputs of system2 into the inputs of system 1 specified by `INPUTS1` and by feeding the outputs of system 2 specified by `OUTPUTS1` into all the inputs of system 2. Positive feedback is assumed. To connect with negative feedback, use negative values in the vector `INPUTS1`.

`feedback()` calls `fdbcksys()` to perform the feedback connection for two systems. Unity feedback calls are possible, for example, `feedback(sys1, 1)`, `feedback(1, sys1)`

**Value**

Returns the feedback system in `tf` or `ss` model

**See Also**

[cloop parallel series](#)

**Examples**

```
C <- pid(350,300,50)
P <- TF(" 1/(s^2 + 10* s + 20)")
feedback(C,P)
feedback(P,P,1)
feedback(P,P,-1)
feedback(P,P)
feedback(P,1)
feedback(TF("C*P"))
## Not run: On Octave: feedback(C*P)
```

---

freqresp

*Low level frequency response function*

---

**Description**

This function obtains the low level frequency response of a system.

**Usage**

```
freqresp(sys, w = seq(0, 100, length = 10000), iu = 1)
```

**Arguments**

<code>sys</code>	An LTI system of <code>tf</code> , <code>ss</code> and <code>zpk</code> class
<code>w</code>	a vector of frequency points
<code>iu</code>	For calls to <code>freqresp</code> , <code>iu</code> is a number specifying an input for a MIMO state-space system. If the system has 3 inputs, then <code>iu</code> would be set to 1, set to 2 and then to 3 to obtain the step response from input 1, 2, and 3 to the outputs

**Value**

`freqresp(sys, w)` returns a vector of frequencies for `sys` in complex form

**See Also**

[bode nyquist](#)

**Examples**

```
H <- freqresp(ssdata(tf(c(1,1), c(1,2,1))), (seq(0, 100, length = 10000)))
H <- freqresp(tf(c(1,1), c(1,2,1)), seq(0, 100, length = 10000))
```



---

gensig                      *Generate periodic signal*

---

### Description

gensig generates a periodic signal. More useful when used in combination with `lsim`

### Usage

```
gensig(signal, tau, tfinal, tsam)
```

### Arguments

signal	A string input containing either values of: square, sin, cos, pulse in the following format: 'sq' or 'square' - Square wave 'si' or 'sine' - Sine wave 'co' or 'cos' - Cosine wave 'pu' or 'pulse' - Periodic pulse
tau	Duration of one period in seconds. Default is 5
tfinal	Duration of the signal in seconds. Default is 30
tsam	sampling time in seconds. Default is 0.01

### Details

gensig generates a periodic signal of the following types: square, sin, cos, pulse  
Possible usage: `gensig(signal)`

### Value

Returns a list of two single column matrices, `u` and `t`  
`u` is the vector of signal values  
`t` is the time vector of the signal

### See Also

[lsim](#)

### Examples

```
## Not run: A square wave signal
sig <- gensig('square', 4, 10, 0.1)
plot(sig$t, sig$u, type = "l", col = "blue")
grid(5,5, col = "lightgray")
```

```
## Not run: A sine wave signal

sig <- gensig('sin')
plot(sig$t, sig$u, type = "l", col = "blue")
grid(5,5, col = "lightgray")
```

---

givens\_rot                      *Complex Givens Rotation*

---

### Description

givens\_rot Forms the Givens rotation matrix

### Usage

```
givens_rot(a, b)
```

### Arguments

a	Complex Square-matrix
b	complex Input-matrix

### Details

givens\_rot(a, b) returns the complex Givens rotation matrix This function is called by [ordschur](#)

### Value

Returns the complex Givens rotation matrix.

### See Also

[ordschur](#)

---

impulse                      *Impulse Response for Linear Systems*

---

### Description

impulse obtains the impulse response of the linear system:

$$dx/dt = Ax + Bu$$

$$y = Cx + Du$$

to an impulse applied to the input

**Usage**

```
impulse(sys, t, input)
impulseplot(sys, t, input)
```

**Arguments**

sys	LTI system of transfer-function, state-space and zero-pole classes
t	Time vector. If not provided, it is automatically set.
input	For calls to <code>impulse</code> , <code>input</code> is a number specifying an input for a MIMO state-space system. If the system has 3 inputs, then <code>input</code> would be set to 1, set to 2 and then to 3 to obtain the impulse response from input 1, 2, and 3 to the outputs. For single input systems, <code>input</code> is always set to 1. For calls to <code>impulseplot</code> , <code>input</code> is a vector or range for a MIMO state-space system. For example, <code>input &lt;- 1:3</code> for a system with 3-inputs

**Details**

`impulse` produces the impulse response of linear systems using `lsim`

`impulseplot` produces the impulse response as a plot against time.

These functions can handle both SISO and MIMO (state-space) models.

Other possible calls using `impulse` and `impulseplot` are:

```
impulse(sys)
impulse(sys, t)
impulseplot(sys)
impulseplot(sys, t)
```

**Value**

A list is returned by calling `impulse` containing:

- t Time vector
- x Individual response of each x variable
- y Response of the system

The matrix `y` has as many rows as there are outputs, and columns of the same size of `length(t)`. The matrix `x` has as many rows as there are states. If the time vector is not specified, then the automatically set time vector is returned as `t`

A plot of `y` vs `t` is returned by calling `impulseplot`

**See Also**

[initial step ramp](#)

**Examples**

```

res <- impulse(tf(1, c(1,2,1)))
res$y
res$t
impulse(tf(1, c(1,2,1)), seq(0, 10, 0.1))
impulseplot(tf(1, c(1,2,1)))
impulseplot(tf(1, c(1,2,1)), seq(0, 10, 0.1))

## Not run: State-space MIMO systems
A <- rbind(c(0,1), c(-25,-4)); B <- rbind(c(1,1), c(0,1));
C <- rbind(c(1,0), c(0,1)); D <- rbind(c(0,0), c(0,0))
res1 <- impulse(ss(A,B,C,D), input = 1)
res2 <- impulse(ss(A,B,C,D), input = 2)
res1$y # has two rows, i.e. for two outputs
res2$y # has two rows, i.e. for two outputs
impulseplot(ss(A,B,C,D), input = 1:2) # OR
impulseplot(ss(A,B,C,D), input = 1:ncol(D))
impulseplot(ss(A,B,C,D), seq(0,3,0.01), 1:2)

```

---

initial

*Initial Condition Response for Linear Systems*


---

**Description**

initial obtains the time response of the linear system:

$$dx/dt = Ax + Bu$$

$$y = Cx + Du$$

to an initial condition.

**Usage**

```

initial(sys, x0, t)
initialplot(sys, x0, t)

```

**Arguments**

sys	LTI system of transfer-function, state-space and zero-pole classes
x0	initial conditions as a column vector. Should have as many rows as the rows of A. where x0 is not specified, random values are assigned
t	regularly spaced time vector. If not provided, it is automatically set. For calls to initialplot, the same arguments are allowed

**Details**

`initial` produces the time response of linear systems to initial conditions using `lsim`

`initialplot` produces the time response to initial conditions as a plot against time.

The functions can handle both SISO and MIMO (state-space) models.

Other possible calls using `initial` and `initialplot` are:

```
initial(sys)
```

```
initial(sys, x0)
```

```
initialplot(sys)
```

```
initialplot(sys, x0)
```

**Value**

A list is returned by calling `initial` containing:

`x` Individual response of each `x` variable

`y` Response of the system

`t` Time vector

The matrix `y` has as many rows as there are outputs, and columns of the same size of `length(t)`.

The matrix `X` has as many rows as there are states. If the time vector is not specified, then the automatically set time vector is returned as `t`

A plot of `y` vs `t` is returned by calling `initialplot`

**See Also**

[step impulse ramp](#)

**Examples**

```
res <- initial(tf(1, c(1,2,1)))
res$y
res$t
A <- rbind(c(-2, -1), c(1,0)); B <- rbind(1,0);
C <- cbind(0,1); D <- as.matrix(0);
x0 <- matrix(c( 0.51297, 0.98127))
initialplot(ss(A,B,C,D), x0)
initialplot(tf(1, c(1,2,1)), t = seq(0, 10, 0.1))

## Not run: State-space MIMO systems
A <- rbind(c(0,1), c(-25,-4)); B <- rbind(c(1,1), c(0,1));
C <- rbind(c(1,0), c(0,1)); D <- rbind(c(0,0), c(0,0))
res <- initial(ss(A,B,C,D))
res$y # has two rows, i.e. for two outputs
initialplot(ss(A,B,C,D))
```

---

issiso *SISO / MIMO Check*

---

**Description**

issiso checks if state-space system is a single-input single-output system  
ismimo checks if state-space system is a multiple-input multiple-output system

**Usage**

issiso(sys)

**Arguments**

sys                    Dynamic system of state-space model

**Value**

Returns TRUE or FALSE

---

lsim *Time response of a Linear system*

---

**Description**

lsim Computes the time response of a Linear system described by:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

to the input time history u.

**Usage**

lsim(sys, u, t, x0)

**Arguments**

sys                    An LTI system of tf, ss and zpk class  
u                        A row vector for single input systems. The input u must have as many rows as there are inputs in the system. Each column of U corresponds to a new time point. u could be generated using a signal generator like gensig  
t                        time vector which must be regularly spaced. e.g. seq(0, 4, 0.1)  
x0                        a vector of initial conditions with as many rows as the rows of a

**Details**

`lsim(sys, u, t)` provides the time history of the linear system with zero-initial conditions.

`lsim(sys, u, t, x0)` provides the time history of the linear system with initial conditions. If the linear system is represented as a model of `tf` or `zpk` it is first converted to state-space before linear simulation is performed. This function depends on `c2d` and `ltitr`

**Value**

Returns a list of two matrices, `x` and `y`. The `x` values are returned from `ltitr` call.

**See Also**

[ltitr](#) [lsimplot](#)

**Examples**

```
signal <- gensig('square',4,10,0.1)
H <- tf(c(2, 5, 1),c(1, 2, 3))
response <- lsim(H, signal$u, signal$t)
plot(signal$t, response$y, type = "l", main = "Linear Simulation Response", col = "blue")
lines(signal$t, signal$u, type = "l", col = "grey")
grid(5,5, col = "lightgray")
## Not run: based on example at: https://www.mathworks.com/help/ident/ref/lsim.html

## Not run: MIMO system response
A <- rbind(c(0,1), c(-25,-4)); B <- rbind(c(1,1), c(0,1))
C <- rbind(c(1,0), c(0,1)); D <- rbind(c(0,0), c(0,0))
response <- lsim(ss(A,B,C,D), cbind(signal$u, signal$u), signal$t)
plot(signal$t, response$y[1,], type = "l",
     main = "Linear Simulation Response", col = "blue"); grid(7,7)
plot(signal$t, response$y[2,], type = "l",
     main = "Linear Simulation Response", col = "blue"); grid(7,7)
```

---

lsimplot

*Plot time response of an LTI system*

---

**Description**

`lsimplot` Plots the time response of a Linear system described by:

$$x = Ax + Bu$$

$$y = Cx + Du$$

to the input time history `u`.

**Usage**

```
lsimplot(sys, u, t, x0)
```

**Arguments**

<code>sys</code>	An LTI system of <code>tf</code> , <code>ss</code> and <code>zpk</code> class
<code>u</code>	A row vector for single input systems. The input <code>u</code> must have as many rows as there are inputs in the system. Each column of <code>u</code> corresponds to a new time point. <code>u</code> could be generated using a signal generator such as <code>gensig</code>
<code>t</code>	time vector which must be regularly spaced. e.g. <code>seq(0,4,0.1)</code>
<code>x0</code>	a vector of initial conditions with as many rows as the rows of <code>sys\$A</code>

**Details**

`lsimplot(sys, u, t)` plots the time history of the linear system with zero-initial conditions.

`lsimplot(sys, u, t, x0)` plots the time history of the linear system with given initial conditions.

If the linear system is represented as a model of `tf` or `zpk` it is first converted to state-space before linear simulation is performed. This function depends on `c2d` and `ltitr`

**Value**

Returns a plot for the response of the system

**See Also**

[lsim](#) [stepplot](#) [rampplot](#)

**Examples**

```
signal <- gensig('square',4,10,0.1)
H <- tf(c(2, 5, 1),c(1, 2, 3))
lsimplot(H, signal$u, signal$t)

## Not run: MIMO system response
A <- rbind(c(0,1), c(-25,-4)); B <- rbind(c(1,1), c(0,1))
C <- rbind(c(1,0), c(0,1)); D <- rbind(c(0,0), c(0,0))
lsimplot(ss(A,B,C,D), cbind(signal$u, signal$u), signal$t)
```

---

ltifr

*LTI frequency response kernel*

---

**Description**

This function computes the frequency response of the following system:

$$g(w) = (wI - A) \setminus B$$

for the complex frequencies contained in the vector `W`. The column vector `B` must have as many rows as the matrix `A`.



**Usage**

```
ltifr(A, B, w)
```

**Arguments**

A	State-space matrix, A
B	State-space input-matrix, B. B must have as many rows as the matrix A.
w	Vector of complex frequencies

**Value**

Returns the frequency response in vector. [freqresp](#) utilizes this function for state-space systems.

**See Also**

[freqresp](#)

**Examples**

```
## use \code{\link{freqresp}}
```

---

ltitr

*Time response of a Linear Time-Invariant system*


---

**Description**

ltitr Computes the time response of a Linear Time-Invariant system

**Usage**

```
ltitr(a, b, u, x0)
```

**Arguments**

a	An n x n matrix of the state-space system
b	An n x m matrix of the state-space system
u	A row vector for single input systems. The input U must have as many rows as there are inputs in the system. Each column of U corresponds to a new time point. u could be generated using a signal generator like <code>gensig</code>
x0	a vector of initial conditions with as many rows as the rows of a

**Details**

ltitr computes the time response of a Linear Time-Invariant system in state-space representation of the form:  $x[n+1] = Ax[n] + Bu[n]$  to an input, U

ltitr(a, b, u) computes the time response with zero-initial conditions since x0 is not supplied.

**Value**

Returns a matrix  $X$  which has as many rows as there are outputs  $y$  (and with  $\max(\dim(U))$  columns).

**See Also**

[lsimgensig](#)

**Examples**

```
A <- diag(1, 2)
B <- rbind(1, 1)
x0 <- rbind(-1, -2)
u <- cbind(1, 2, 3, 4, 5)
X <- ltitr(A, B, u)
X <- ltitr(A, B, u, x0)
```

```
A <- replicate(6, abs(rnorm(6)))
B <- replicate(3, abs(rnorm(6)))
U <- replicate(100, rnorm(3))
x0 <- rnorm(6)
X <- ltitr(A, B, U)
X <- ltitr(A, B, U, x0)
```

---

nyquist

*Nyquist Frequency Response for continuous-time Linear Systems.*


---

**Description**

nyquist computes the real and imaginary parts of the frequency response of system `sys` at given frequencies `w`

**Usage**

```
nyquist(sys, w, iu)
```

**Arguments**

<code>sys</code>	LTI system of transfer-function, state-space and zero-pole classes
<code>w</code>	vector of range of frequencies at the response is computed in rad/sec
<code>iu</code>	number to specify an input for a MIMO state-space system. If the system has 3 inputs, then <code>iu</code> would be set to 1, set to 2 and then to 3 to obtain the nyquist response from input 1, 2, and 3 to the outputs. For single input systems, <code>iu</code> is always set to 1. <code>iu</code> is not needed/allowed for calls to <code>nyquistplot</code>

## Details

`nyquist` Compute the real and imaginary parts of the frequency response of system `sys` at given frequencies `w`. When `sys` is a transfer function, `nyquist` computes the frequency response of the system using the signal package.

`nyquistplot` plots the frequency response computed by `nyquist`. For a MIMO state-space system, `nyquistplot` uses `selectsys` to obtain the nyquist response for each input-to-output pair and plot them individually. This means that for a 2-input, 2-output system, `nyquistplot` obtains the response for input 1 to output 1, input 1 to output 2, input 2 to output 1 and input 2 to output 2. `nyquistplot` uses the `subtitle` argument to allow a user assign the plot a sub-title

Other possible calls using `nyquist` and `nyquistplot` are:

```
nyquist(sys) nyquist(sys, w) nyquist(sys, w = seq(0, 100, length = 10000), iu = 1)
nyquistplot(sys) nyquistplot(sys, w) nyquistplot(sys, w, subtitle)
```

## Value

A list is returned by calling `nyquist` containing:

- `h.real` - real part of the frequency response

- `h.imag` - imaginary part of the frequency response

A plot is returned by calling `nyquistplot`

## See Also

[bode](#)

## Examples

```
nyquist(tf(100, c(1,6,100)))
nyquist(ssdata(tf(100, c(1,6,100))))

## Not run: MIMO plot
A1 <- rbind(c(0,1), c(-25,-4)); B1 <- rbind(c(1,1), c(0,1))
C1 <- rbind(c(1,0), c(0,1)); D1 <- rbind(c(0,0), c(0,0))
sys1 <- ss(A1,B1,C1,D1)
nyquistplot(sys1)
## Not run: Use nyquistplot(selectsys(sys1,1,2)) to obtain the response for a subsystem
of sys1 for input 1 and output 2 only.

RESET your plot layout using par(mfrow = c(1,1))

## End(Not run)
```

---

obsv	<i>Observability Matrix</i>
------	-----------------------------

---

**Description**

This function creates the observability matrix.

**Usage**

```
obsv(A, C)
```

**Arguments**

A	State-space matrix, A
C	State-space matrix, C

**Value**

obsv(A, C) returns the observability matrix, obsvm. where  $\text{obsvm} = [C \ CA \ CA^2 \ \dots \ CA^{(n-1)}]$

**See Also**

[ctrb](#)

**Examples**

```
A <- rbind(c(0,1), c(-25,-4))
C <- rbind(c(1,0), c(0,1))
obsv(A, C)
```

---

ordschur	<i>Ordered schur decomposition</i>
----------	------------------------------------

---

**Description**

ordschur Orders a schur decomposition

**Usage**

```
ordschur(Ui, Si, idx)
```

**Arguments**

Ui	Square upper-triangular matrix from schur decomposition. If Ui is not given it is set to the identity matrix.
Si	Orthogonal matrix from schur decomposition
idx	array index

**Details**

ordschur finds an orthogonal matrix, U so that the eigenvalues appearing on the diagonal of Si are ordered according to the increasing values of the array index where the i-th element of index corresponds to the eigenvalue appearing as the element Si[i, i].

ordschur could also be used in this syntax: ordschur(Si, idx)

**Value**

Returns a list of ordered (U, S)

---

parallel

*Parallel Connection of two systems*


---

**Description**

parallel connects two systems in the parallel block form below

$$| \rightarrow [\text{System1}] | \text{ u} \rightarrow + 0 \rightarrow y | \leftarrow [\text{System2}] |$$
**Usage**

```
parallel(sys1, sys2, in1, in2, out1, out2)
```

**Arguments**

sys1	LTI system object of tf, ss or zpk class
sys2	LTI system object of tf, ss or zpk class
in1	Numeric vector containing indexes to the inputs of sys1
in2	Numeric vector containing indexes to the inputs of sys2
out1	Numeric vector containing indexes to the outputs of sys1
out2	Numeric vector containing indexes to the outputs of sys2

**Details**

`psys <- parallel(sys1, sys2)` produces a state- space system consisting of the parallel connection of `sys1` and `sys2` that connects all the inputs together and sums all the outputs of the two systems.

The parallel connection is performed by appending the two systems, summing the specified inputs and outputs, and removing the, now redundant, inputs and outputs of system 2.

If `sys1` and `sys2` are transfer functions, then `parallel(sys1, sys2)` produces a parallel connection of the two transfer function systems.

`parallel(sys1, sys2, IN1, IN2, OUT1, OUT2)` connects the two systems in parallel by connecting the inputs specified by `IN1` and `IN2` and by summing the outputs specified by `OUT1` and `OUT2`. The vector `IN1` contains indexes into the input vectors of `sys1` while, `IN2` contains indexes for `sys2`, . Vectors `OUT1` and `OUT2` contain indexes for the outputs of the `sys1` and `sys2` respectively.

**Value**

The function returns a state-space model of the parallel-connected system with A, B, C, D matrices

**See Also**

[series feedback connect](#)

**Examples**

```
sys2 = ss(1,2,3,4)
sys3 = ss(6,7,8,9)
parallel(sys2, sys3)
parallel(tf(1, c(1,2,3)), ss(1,2,3,4))
parallel(tf(1, c(1,2,3)), tf(2, c(3,2,3)))
```

---

pid

*Proportional-Integral-Derivative (PID) Controller*

---

**Description**

pid Parallel form of the model of a PID controller

**Usage**

```
pid(p, i, d)
```

**Arguments**

p	Proportional gain. A real and finite value.
i	Integral gain. A real and finite value. set this to zero for PD and P-control
d	Derivative gain. A real and finite value. set this to zero for PI and P-control

**Details**

pid creates the transfer function model for a PID, PI, PD, and P-controller.

**Value**

Returns a transfer function model for the PID, PI, PD or P-controller.

**Examples**

```
C <- pid(350,300,50) # PID-control
P <- TF(" 1/(s^2 + 10* s + 20)")
T <- feedback(TF("C*P"), 1)
stepplot(T, seq(0,2,0.01))
```

```
C <- pid(300,0,0) # P-control
T <- feedback(TF("C*P"), 1)
stepplot(T, seq(0,2,0.01))
```

```
C <- pid(30,70,0) # PI-control
T <- feedback(TF("C*P"), 1)
stepplot(T, seq(0,2,0.01))
```

```
C <- pid(300,0,10) # PD-control
T <- feedback(TF("C*P"), 1)
stepplot(T, seq(0,2,0.01))
```

---

place

*Pole placement gain selection*

---

**Description**

Computes the Pole placement gain selection using Ackermann's formula.

**Usage**

```
place(a, b, p)
```

**Arguments**

a	State-matrix of a state-space system
b	Input-matrix of a state-space system
p	closed loop poles

**Details**

`K <- place(A,B,P)` calculates the feedback gain matrix `K` such that the single input system  $\dot{x} <- Ax + Bu$

with a feedback law of  $u <- -Kx$  has closed loop poles at the values specified in vector `P`, i.e.,  $P <- \text{eigen}(A - B * K)$ . This function is just a wrapper for the `acker` function.

This method is NOT numerically stable and a warning message is printed if the nonzero closed loop poles are greater than 10 in `P`.

**Examples**

```
F <- rbind(c(0,1),c(0,0))
G <- rbind(0,1)
H <- cbind(1,0);
J <- 0
t <- 1
sys <- ss(F,G, H,J)
A <- c2d(sys,t);
j <- sqrt(as.complex(-1));
pc <- rbind(0.78+0.18*j, 0.78-0.18*j)
K <- place(A$A, A$B, pc)
```

---

pole

*Obtain Poles for a System*

---

**Description**

This function obtains the poles for a given system

**Usage**

```
pole(sys)
```

**Arguments**

`sys` LTI system of `tf`, `ss` and `zpk` class

**Details**

`pole` returns the poles for a given system either a transfer function, state-space or zero-pole models. If `sys` is a transfer function, it computes the roots of the denominator. If `sys` is a state-space object, it computes the eigenvalues of the `A` matrix. If `sys` is a `zpk` object, it retrieves the poles from the object.

**Value**

The function returns a column matrix containing the poles for the given system



**Examples**

```
H1 <- tf(c(2, 5, 1),c(1, 3, 5))
pole(zpk(NULL, c(-1,-1), 1))
pole(ssdata(tf(1, c(1,2,1))))
```

---

poly2str

*Print Polynomial*

---

**Description**

Print polynomial as a character string.

**Usage**

```
poly2str(p, svar = "x", smul = "*", d = options("digits")$digits)
```

**Arguments**

p	numeric vector representing a polynomial
svar	character representing the unknown, default x.
smul	multiplication symbol, default *.
d	significant digits, default options("digits").

**Details**

Modified from package *\*pracma\**. Modification: To hide any coefficient and power that is equal to 1 So that instead of '1s^3' we have 's^3' and instead of 's^1', we have 's'

**Value**

Returns the usual string representing a polynomial in mathematics.

**Examples**

```
poly2str(c(2, -3, 1, 20, -11))
```

---

polysub                      *Subtracting Polynomials*

---

**Description**

Subtract two polynomials given as vectors

**Usage**

```
polysub(a, b)
```

**Arguments**

a                      Vector representing first polynomial.  
b                      Vector representing second polynomial.

**Details**

Simply calls polyadd from pracma package in the following manner: `pracma::polyadd(a, -b)`

**Value**

Returns a Vector representing the resulting polynomial.

**Examples**

```
polysub(c(1, 1, 1), 1)  
polysub(c(1, 1, 1), c(0, 0, 1))
```

---

ramp                      *Ramp Response for Linear Time-Invariant Systems*

---

**Description**

ramp obtains the ramp response of the linear system:

$$dx/dt = Ax + Bu$$

$$y = Cx + Du$$

**Usage**

```
ramp(sys, t, input)  
rampplot(sys, t, input)
```

**Arguments**

sys	LTI system of transfer-function, state-space and zero-pole classes
t	Time vector. If not provided, it is automatically set.
input	For calls to ramp, input is a number specifying an input for a MIMO state-space system. If the system has 3 inputs, then input would be set to 1, set to 2 and then to 3 to obtain the ramp response from input 1, 2, and 3 to the outputs. For single input systems, input is always set to 1. For calls to rampplot, input is a vector or range for a MIMO state-space system. For example, input <- 1:3 for a system with 3-inputs

**Details**

ramp produces the ramp response of linear systems using lsim

rampplot produces the ramp response as a plot against time.

These functions can handle both SISO and MIMO (state-space) models.

#' Other possible calls using ramp and rampplot are:

```
ramp(sys)
```

```
ramp(sys, t)
```

```
rampplot(sys)
```

```
rampplot(sys, t)
```

**Value**

A list is returned by calling ramp containing:

t Time vector

x Individual response of each x variable

y Response of the system

The matrix y has as many rows as there are outputs, and columns of the same size of length(t). The matrix x has as many rows as there are states. If the time vector is not specified, then the automatically set time vector is returned as t

A plot of y vs t is returned by calling rampplot

**See Also**

[initial step impulse](#)

**Examples**

```
res <- ramp(tf(1, c(1,2,1)))
res$y
res$t
ramp(tf(1, c(1,2,1)), seq(0, 6, 0.1))
rampplot(tf(1, c(1,2,1)))
rampplot(tf(1, c(1,2,1)), seq(0, 6, 0.1))
```

```
## Not run: State-space MIMO systems
A <- rbind(c(0,1), c(-25,-4)); B <- rbind(c(1,1), c(0,1));
C <- rbind(c(1,0), c(0,1)); D <- rbind(c(0,0), c(0,0))
res1 <- ramp(ss(A,B,C,D), input = 1)
res2 <- ramp(ss(A,B,C,D), input = 2)
res1$y # has two rows, i.e. for two outputs
res2$y # has two rows, i.e. for two outputs
rampplot(ss(A,B,C,D), input = 1:2) # OR
rampplot(ss(A,B,C,D), input = 1:ncol(D))
rampplot(ss(A,B,C,D), seq(0,3,0.01), 1:2)
```

---

selectsys

*Select/Remove Subsystem in State-space Model*


---

### Description

selectsys extracts a subsystem from a larger state-space system. removesys removes specified inputs, outputs, and state from a state-space system.

### Usage

```
selectsys(statesys, inputs, outputs, states)
removesys(statesys, inputs, outputs, states)
```

### Arguments

statesys	LTI system model of state-space model
inputs	single integer or vector specifying the particular inputs to be selected/removed
outputs	single integer or vector specifying the particular outputs to be selected/removed
states	single integer or vector specifying the particular states to be selected/removed

### Details

subsys <- selectsys(statesys, inputs, outputs) will extract a state space subsystem with the specified inputs and outputs.

subsys <- selectsys(statesys, inputs, outputs, states) will return the state space subsystem with the specified inputs, outputs, and states.

subsys <- removesys(statesys, inputs, outputs) will remove the specified inputs and outputs from the system.

subsys <- removesys(statesys, inputs, outputs, states) will also return a state-space model with the specified inputs, outputs, and states removed from the system.

### Value

Returns a subsystem of the state-space model

**See Also**[append](#)**Examples**

```

A <- rbind(c(33,2,5), c(23,200,2), c(9,2,45))
B <- rbind(c(4,5), c(12,5), c(82,1))
C <- rbind(c(34,56,2), c(6,2,112))
D <- rbind(c(2,0), c(0,19))
sys1 <- ss(A, B, C, D)
selectsys(sys1, 1, 1) # extract subsystem for only input 1 and output 1
selectsys(sys1, 2,2) # extract subsystem for only input 2 and output 2
selectsys(sys1, 2, 1:2) # extract subsystem for only input 1 and output 1 to 2
selectsys(sys1, 1:2, 2) # extract subsystem for only input 1 to 2 and output 2 to 2
selectsys(sys1, 2, 2, 1:2) # extract subsystem for only input 2 and output 2 but states 1 to 2
removesys(sys1, 1,2) # removes input 1 and output 2

```

series

*Series Connection of two systems***Description**

series connects two systems in the series block form below

$$u \rightarrow [\text{System1}] \rightarrow [\text{System2}] \rightarrow y$$
**Usage**

```
series(sys1, sys2, outputs, inputs)
```

**Arguments**

sys1	LTI system object of tf, ss or zpk class
sys2	LTI system object of tf, ss or zpk class
outputs	vector of outputs
inputs	vector of inputs

**Details**

seriesssys <- series(sys1, sys2) connects the two state-space systems in series such that the outputs of sys1 specified are connected to the inputs of sys2 specified by input2. If sys1 and sys2 are both transfer functions, series(systf1, systf2) produces the SISO system in transfer function form obtained by connecting the two SISO transfer function systems in series. If a system is not in state-space representation, the function tries to form a state-space representation for such system.

**Value**

The function returns a state-space model of the aggregate system with A, B, C, D matrices

**See Also**

[parallel feedback connect](#)

**Examples**

```
series(tf(1, c(1,2,3)), tf(2, c(2,3,5)))  
sys2 = ss(1,2,3,4)  
sys3 = ss(6,7,8,9)  
series(sys2, sys3)  
series(tf(1, c(1,2,3)), ss(1,2,3,4))
```

---

ss *Create State-space Model.*

---

**Description**

ss creates the model for a system represented in state-space form

**Usage**

```
ss(A, B, C, D, Ts = NULL)
```

**Arguments**

A	An $n \times n$ matrix
B	An $n \times m$ matrix
C	An $p \times n$ matrix
D	An $p \times m$ matrix
Ts	Sample time for discrete time systems

**Details**

ss creates a model object for state-space systems.

**Value**

Returns a list object of 'ss' class.

**See Also**

[tf zpk](#)

**Examples**

```

A <- rbind(c(-2, -1), c(1,0))
B <- rbind(1,0)
C <- cbind(0,1)
D <- 0;
sys <- ss(A,B,C,D)

## Not run:  OR

sys <- ss(c(-2,-1,1,0), c(1,0), c(0,1), 0)

## Not run:  Access individual state-space sys elements as

sys$A
sys$B
sys$C
sys$D

```

---

ss2tf

*State-space model conversion to Transfer function model.*


---

**Description**

ss2tf converts the model for a state-space system to transfer function representation

**Usage**

```
ss2tf(a, b, c, d, iu)
```

**Arguments**

a	An n x n matrix
b	An n x m matrix
c	An p x n matrix
d	An p x m matrix
iu	A numeric value denoting number of inputs. default value is 1. For example, if the system has three inputs (u1, u2, u3), then iu must be either 1, 2, or 3, where 1 implies u1, 2 implies u2, and 3 implies u3.

**Details**

ss2tf converts a model object in state-space form to transfer function model by calculating the transfer function of the system:  $\dot{x} = Ax + Bu$   $y = Cx + Du$

#' Other possible usages for ss2tf are: `ss2tf(a,b,c,d)` `ss2tf(sys)` `ss2tf(sys, iu)`

where sys is an object of state-space class

**Value**

Returns an object of 'tf' class containing num and den. The numerator coefficients are returned in matrix num with as many rows as outputs y.

**See Also**

[tf2ss](#) [ss2zp](#)

**Examples**

```
sys2 <- tf2ss(tf(1, c(1,2,1)))
ss2tf(sys2)

## Not run:  OR

ss2tf(sys2$A,sys2$B,sys2$C,sys2$D)

# a single input multiple output system
A <- rbind(c(0,1), c(-10000,-4)); B <- rbind(0,1); C <- rbind(c(1,0), c(0,1));
D <- rbind(0,0)
ss2tf(A, B, C, D)

# a MIMO system
A = rbind(c(0,1), c(-25,-4)); B = rbind(c(1,1), c(0,1));
C = rbind(c(1,0), c(0,1)); D = rbind(c(0,0), c(0,0))
ss2tf(A,B,C,D,1) # to obtain output for input 1
ss2tf(A,B,C,D,2) # to obtain output for input 2

## OR

systems <- vector("list", ncol(D))
for(i in 1:ncol(D)){ systems[[i]] <- ss2tf(A,B,C,D,i) }
systems
systems[[1]]
systems[[2]]
```

---

ss2zp

*State-space representation to zero-pole-gain representation*


---

**Description**

ss2zp converts a system represented in state-space form to zero-pole-gain model

**Usage**

```
ss2zp(a,b,c,d,iu)
```



**Arguments**

a	An $n \times n$ matrix
b	An $n \times m$ matrix
c	An $p \times n$ matrix
d	An $p \times m$ matrix
iu	A numeric value denoting number of inputs. default value is 1. For example, if the system has three inputs (u1, u2, u3), then iu must be either 1, 2, or 3, where 1 implies u1, 2 implies u2, and 3 implies u3.

**Details**

ss2zp converts a system represented in zero-pole form to state-space by converting from zero-pole to transfer function and from transfer function to state-space. The vector P contains the pole locations of the denominator of the transfer function.

Other possible usages for ss2zp are:

```
ss2zp(a,b,c,d)
```

```
ss2zp(sys)
```

```
ss2zp(sys, iu)
```

where sys is an object of state-space class

**Value**

Returns a list object of 'zpk' class, consisting of z, p and k. The numerator zeros are returned in the columns of matrix Z with number of columns equal to number of outputs. The gains for each numerator transfer function are returned in column vector K. P, a column vector contains the pole locations of the denominator of the transfer function.

**See Also**

[zp2ss](#) [ss2tf](#)

**Examples**

```
A <- rbind(c(-2, -1), c(1,0)); B <- rbind(1,0);
C <- cbind(0,1); D <- 0;
sys2 <- ss(A,B,C,D)
ss2zp(sys2$A,sys2$B,sys2$C,sys2$D)
ss2zp( zp2ss ( tf2zp( c(1,1,1), c(1,2,1) ) ) )

## Not run: A MIMO system
A = rbind(c(0,1), c(-25,-4)); B = rbind(c(1,1), c(0,1));
C = rbind(c(1,0), c(0,1)); D = rbind(c(0,0), c(0,0))
ss2tf(A,B,C,D,1) # to obtain output for input 1
ss2tf(A,B,C,D,2) # to obtain output for input 2

## Not run: OR
```

```
systems <- vector("list", ncol(D))
for(i in 1:ncol(D)){ systems[[i]] <- ss2zp(A,B,C,D,i) }
systems
systems[[1]]
systems[[2]]
```

---

ssdata

*Retrieve State-space data*

---

### Description

ssdata retrieves the model for a state-space system from a sys object

### Usage

```
ssdata(sys1)
```

### Arguments

sys1                    an LTI system object of tf, ss or zpk classes

### Details

ssdata retrieves a model object for a state-space system, from a sys object of tf, ss and zpk classes

### Value

Returns a list object of ss class containing A, B, C and D matrices

### See Also

[ss](#) [tfdata](#) [zpkdata](#)

### Examples

```
sys1 <- tf(c(1), c(1,2,1))
ssdata(sys1)
A <- rbind(c(-2, -1), c(1,0)); B <- rbind(1,0);
C <- cbind(0,1); D <- 0;
sys2 <- ss(A,B,C,D)
ssdata(sys2)
ss2zp(ssdata(zpk(NULL, c(-1,-1), 1)))
```

**Description**

step obtains the time response of the linear system:

$$dx/dt = Ax + Bu$$

$$y = Cx + Du$$

**Usage**

```
step(sys, t, input)
stepplot(sys, t, input)
```

**Arguments**

sys	LTI system of transfer-function, state-space and zero-pole classes
t	Time vector. If not provided, it is automatically set.
input	For calls to step, input is a number specifying an input for a MIMO state-space system. If the system has 3 inputs, then input would be set to 1, set to 2 and then to 3 to obtain the step response from input 1, 2, and 3 to the outputs. For single input systems, input is always set to 1. For calls to stepplot, input is a vector or range for a MIMO state-space system. For example, input <- 1:3 for a system with 3-inputs

**Details**

step produces the step response of linear systems using lsim

stepplot produces the step response as a plot againsts time.

The functions can handle both SISO and MIMO (state-space) models.

Other possible calls using step and stepplot are:

```
step(sys)
```

```
step(sys, t)
```

```
stepplot(sys)
```

```
stepplot(sys, t)
```

**Value**

A list is returned by calling `step` containing:

`x` Individual response of each `x` variable

`y` Response of the system

`t` Time vector

The matrix `y` has as many rows as there are outputs, and columns of the same size of `length(t)`. The matrix `X` has as many rows as there are states. If the time vector is not specified, then the automatically set time vector is returned as `t`

A plot of `y` vs `t` is returned by calling `stepplot`

**See Also**

[initial impulse ramp](#)

**Examples**

```
res <- step(tf(1, c(1,2,1)))
res$y
res$t
step(tf(1, c(1,2,1)), seq(0, 10, 0.1))
stepplot(tf(1, c(1,2,1)))
stepplot(tf(1, c(1,2,1)), seq(0, 10, 0.1))

## Not run: State-space MIMO systems

A <- rbind(c(0,1), c(-25,-4)); B <- rbind(c(1,1), c(0,1));
C <- rbind(c(1,0), c(0,1)); D <- rbind(c(0,0), c(0,0))
res1 <- step(ss(A,B,C,D), input = 1)
res2 <- step(ss(A,B,C,D), input = 2)
res1$y # has two rows, i.e. for two outputs
res2$y # has two rows, i.e. for two outputs
stepplot(ss(A,B,C,D), input = 1:2) # OR
stepplot(ss(A,B,C,D), input = 1:ncol(D))
```

---

 TF

---

*Evaluate Transfer function Expressions*


---

**Description**

TF Evaluates a given transfer function expression in the `s`-domain

**Usage**

TF(str\_expr)

**Arguments**

`str_expr`      String expression containing the transfer function

**Details**

TF Evaluates a given transfer function polynomial expression in the s-domain. The evaluation of the expressions are performed similar to symbolic math computations for polynomials. A transfer function model is created as the result of the expression evaluation. Thus, this is an alternative way of creating transfer function models following the natural math expressions found in block diagrams. It also provides an alternative way to perform system interconnections. Only transfer function models are currently supported for system interconnection using this function. System interconnections for other models could be performed using the `series`, `parallel`, `feedback` or `connect` functions. See the Examples section for further details.

**Value**

Returns an object of 'tf' class list with a transfer function. Numerator and denominator coefficients could then be retrieved from the object the same way as any other tf object

**See Also**

[tf](#) [tf2ss](#) [series](#) [parallel](#)

**Examples**

```
# Example taken from the GitHub page of Julia Control - an electric motor example
J <- 2.0
b <- 0.04
K <- 1.0
R <- 0.08
L <- 1e-4
P <- TF("K/(s*((J*s + b)*(L*s + R) + K^2))")
Cls <- TF("P/(1 + P)") # closed-loop connection

# More examples
TF("s+1")
sys1 <- tf(1, c(1, 2, 5))
sys2 <- tf(2, c(1, 2, 5))
TF("sys1 + sys2") # parallel system interconnection
TF("sys1 * sys2") # series system interconnection
TF("sys1 - sys2")
TF("sys1 - 1")
TF("sys1 + 1")
TF("sys1 - sys2 + sys2")
TF("sys1 / sys2 / sys2")
```

---

tf *Create Transfer function Model.*

---

## Description

tf creates the model for a transfer function

## Usage

```
tf(num, den, Ts = NULL)
```

## Arguments

num	A numeric vector or matrix (for multivariable systems)
den	A numeric vector or matrix (for multivariable systems)
Ts	Sample time for discrete time systems

## Details

tf creates a model object for a transfer function, Where num is the numerator and den is the denominator of the transfer function.

## Value

Returns an object of 'tf' class list with a proper transfer function or with warnings when not proper.

## See Also

[ss](#) [zpk](#) [TF](#) [tf2ss](#) [tf2zp](#)

## Examples

```
tf(1, c(1,2,1))
sys1 <- tf(1, c(1,2,1))
sys1$num
sys1$den
```

```
## Not run: for single-input multi-output systems (SIMO) each numerator row for one output
num = rbind(c(0,1,1), c(1,0,1))
den = rbind(c(1,3,2))
tf(num, den)
```

---

`tf2ss`*Transfer function model conversion to State-space model.*

---

**Description**

`tf2ss` converts the model for a transfer function to state-space representation

**Usage**

```
tf2ss(num, den)
```

**Arguments**

<code>num</code>	A numeric vector containing the coefficients of the
<code>den</code>	A numeric vector containing the coefficients of the

**Details**

`tf2ss` converts a model object for a transfer function to a state-space model, Where `num` is the numerator and `den` is the denominator of the transfer function and `sys` is a transfer function object

Another possible call is `tf2ss(sys)` where `sys` is object of transfer-function model.

**Value**

Returns an object of 'ss' class.

**See Also**

[ss2tf](#) [tf2zp](#)

**Examples**

```
tf2ss(tf(1, c(1,2,1)))  
  
## Not run: OR  
  
sys <- tf(1, c(1,2,1))  
tf2ss(sys)  
  
## Not run: OR  
  
sys2 <- tf2ss(1, c(1,2,1))
```

---

`tf2zp`*Transfer function model conversion to Zero-Pole-Gain model.*

---

### Description

`tf2zp` converts the model for a transfer function to zero-pole-gain representation

### Usage

```
tf2zp(num, den)
```

### Arguments

<code>num</code>	A numeric vector containing the coefficients of the
<code>den</code>	A numeric vector containing the coefficients of the

### Details

`tf2zp` converts a model object for a transfer function to a zero-pole model, Where `num` is the numerator and `den` is the denominator of the transfer function and `sys` is a transfer function object

Another possible call is: `tf2zp(sys)`

where `sys` is an object of transfer-function model.

### Value

Returns a list object of 'zpk' class.

### See Also

[tf2ss](#) [zp2tf](#)

### Examples

```
syszp1 <- tf2zp(c(1,1), c(1,2,1))
syszp1
syszp2 <- tf2zp(c(2,2,1), c(1,2,1))
syszp2
unclass(syszp2) # to see list of the zeros,poles and gain as vectors
tf2zp(zp2tf(c(-1,-1), c(-1,-2), 5))
```



---

tfchk	<i>Transfer function check.</i>
-------	---------------------------------

---

**Description**

tfchk verifies the structure of a transfer function

**Usage**

```
tfchk(num, den)
```

**Arguments**

num	A numeric vector
den	A numeric vector

**Details**

This is a utility function that is always invoked by other functions to verify the structure of num, den. Where num is the numerator and den is the denominator of the transfer function. If the transfer function is not proper, it returns a list with length(num) = length(den).

**Value**

Returns a list with a proper transfer function or with warnings when not proper.

**Examples**

```
tf1 <- tfchk(1, c(1,2,1))
```

---

tfdata	<i>Retrieve Transfer function data</i>
--------	--

---

**Description**

tfdata retrieves the model for a transfer function from a sys object

**Usage**

```
tfdata(sys1)
```

**Arguments**

sys1	an LTI system object of tf, ss or zpk classes
------	---

**Details**

tfdata retrieves a model object for a transfer function, from a sys object of tf, ss and zpk classes

**Value**

Returns a list object of tf class containing numerator and denominator coefficients in descending values of s. For multiple-input multiple-output systems (MIMO) a list containing tf sys objects for as many outputs is returned

**See Also**

[tf](#) [ssdata](#) [zpkdata](#)

**Examples**

```
sys1 <- zpk(NULL, c(-1,-1), 1)
tfdata(sys1)
A <- rbind(c(-2, -1), c(1,0)); B <- rbind(1,0);
C <- cbind(0,1); D <- 0
tfdata( ss(A, B, C, D) )
tfdata(ss2zp( A,B,C,D))
tfdata(tf(c(1), c(1,2,1)))

## Not run:  MIMO system
A = rbind(c(0,1), c(-25,-4)); B = rbind(c(1,1), c(0,1));
C = rbind(c(1,0), c(0,1)); D = rbind(c(0,0), c(0,0))
tfdata(ss(A,B,C,D))
```

---

zp2ss

*Convert Zero-Pole-Gain Model to State-Space Model*

---

**Description**

zp2ss converts a system represented in zero-pole form to state-space

**Usage**

```
zp2ss(z,p,k)
```

**Arguments**

z	Zero, a vector or single row matrix
p	Pole, a vector or single row matrix
k	Gain, a vector

**Details**

zp2ss converts a system represented in zero-pole form to state-space by converting from zero-pole to transfer function and from transfer function to state-space

Another possible usage is: zp2ss(sys)

where sys is an object of zero-pole-gain model.

**Value**

Returns a list object of 'ss' class.

**See Also**

[ss2zp](#) [zp2tf](#)

**Examples**

```
zp2ss(NULL, c(-1,-1), 1)
zp2ss(tf2zp(c(1,1,1), c(1,2,1)))
```

---

 zp2tf

---

*Zero-pole-gain model conversion to Transfer function model*


---

**Description**

zp2tf converts the model for a zero-pole-gain system to transfer function representation

**Usage**

```
zp2tf(z, p, k)
```

**Arguments**

z	A numeric vector containing zero locations
p	A numeric vector containing pole locations
k	A numeric vector for gain

**Details**

zp2tf converts a model object for a zero-pole-gain system to a transfer function model

Another possible usage is: zp2tf(sys)

where sys is an object of zero-pole-gain model.

**Value**

Returns a list object of 'tf' class.

**See Also**[zp2ss](#) [tf2zp](#)**Examples**

```
systf <- zp2tf(zpk(NULL, c(-1,-1), 1))
zp2tf(tf2zp(c(2,2,1), c(1,2,1)))
```

---

zpk

*Create Zero-Pole-Gain Model.*

---

**Description**

zpk creates the model for a system represented in zero-pole form

**Usage**

```
zpk(zero, pole, gain, Ts = NULL)
```

**Arguments**

zero	A vector
pole	A vector
gain	A vector
Ts	Sample time for discrete time systems

**Details**

zpk creates a model object for zero-pole systems.

**Value**

Returns a list object of 'zpk' class.

**See Also**[ss tf](#)

**Examples**

```
sys <- zpk(NULL, c(-1,-1), 1)
sys <- zpk(c(1,2), c(3,4), 5)
sys <- zpk(c(1,2), c(3+1i,4+2i), 5)

## Not run:  Access individual sys elements as
sys$z
sys$p
sys$k
```

---

zpkdata	<i>Retrieve zero-pole data from LTI system object</i>
---------	---

---

**Description**

zpkdata retrieves the model for a zero-pole-gain system from a sys object

**Usage**

```
zpkdata(sys1)
```

**Arguments**

sys1            an LTI system object of tf, ss or zpk classes

**Details**

zpkdata retrieves a model object for a zero-pole-gain system, from a sys object of tf, ss and zpk classes

**Value**

Returns a list object of zpk class containing zero, pole and gain matrices. For multivariable systems, the zeros of each system is listed as a column in the zeros matrix, the poles are listed as a column-vector as well as the gain

**See Also**

[zpk tfdata ssdata](#)

**Examples**

```
sys1 <- zpk(NULL, c(-1,-1), 1)
zpkdata(sys1)
sys3 <- tf(c(1), c(1,2,1))
zpkdata(sys3)
```

```
## Not run:  MIMO system of 2-inputs and 2-outputs
A = rbind(c(0,1), c(-25,-4)); B = rbind(c(1,1), c(0,1));
C = rbind(c(1,0), c(0,1)); D = rbind(c(0,0), c(0,0))
zpkdata(ss(A,B,C,D))

## OR
syszp <- zpkdata(ss(A,B,C,D))
syszp[[1]]
syszp[[2]]
syszp[[1]]$z # retrieve zeros of system 1 - Input 1 to Outputs 1 and 2
syszp[[2]]$z # retrieve zeros of system 2 - Input 2 to Outputs 1 and 2
```

# Index

abcdchk, 3  
acker, 4  
append, 5, 11, 37  
  
bode, 6, 16, 27  
bodeplot (bode), 6  
  
c2d, 7  
care, 8  
cloop, 9, 16  
connect, 5, 10, 30, 38  
ctrb, 12, 28  
  
damp, 13, 14, 15  
dcgain, 14  
  
esort, 13, 14  
  
fdbcksys (feedback), 15  
feedback, 5, 10, 11, 15, 30, 38  
freqresp, 16, 25  
  
gensig, 17, 26  
givens\_rot, 18  
  
impulse, 18, 21, 35, 44  
impulseplot (impulse), 18  
initial, 19, 20, 35, 44  
initialplot (initial), 20  
ismimo (issiso), 22  
issiso, 22  
  
lsim, 17, 22, 24, 26  
lsimplot, 23, 23  
ltifr, 24  
ltitr, 8, 23, 25  
  
nyquist, 7, 16, 26  
nyquistplot (nyquist), 26  
  
obsv, 12, 28  
  
ordschur, 18, 28  
  
parallel, 5, 11, 16, 29, 38, 45  
pid, 30  
place, 31  
pole, 32  
poly2str, 33  
polysub, 34  
  
ramp, 19, 21, 34, 44  
rampplot, 24  
rampplot (ramp), 34  
removesys (selectsys), 36  
  
selectsys, 36  
series, 5, 11, 16, 30, 37, 45  
ss, 38, 42, 46, 52  
ss2tf, 39, 41, 47  
ss2zp, 40, 40, 51  
ssdata, 42, 50, 53  
step, 19, 21, 35, 43  
stepplot, 24  
stepplot (step), 43  
sysgroup (append), 5  
  
TF, 44, 46  
tf, 38, 45, 46, 50, 52  
tf2ss, 40, 45, 46, 47, 48  
tf2zp, 46, 47, 48, 52  
tfchk, 49  
tfdata, 42, 49, 53  
  
zp2ss, 41, 50, 52  
zp2tf, 48, 51, 51  
zpk, 38, 46, 52, 53  
zpkdata, 42, 50, 53