# Maple SYREP

Version 2.1

For Maple V, Releases 4 and 5, Maple 6, and Maple 7.

A software package for the modeling and analysis of lumped-parameter linear systems.

## User Reference Manual: Part II

## Function Descriptions

Derek Rowell
Department of Mechanical Engineering
Massachusetts Institute of Technology
October, 2002

# 1. Argument conventions:

In all following SYREP function descriptions the arguments follow the guidelines:

| | |
|---|---|
| { } | Indicates an optional argument. In some cases the argument is optional for SISO systems,and must be supplied for MIMO systems. When multiple arguments are specified within the brackets, all or none must be supplied. |
| `[arg]` | Indicates that `arg` is a Maple *list* in the format `[item1, item2,...,itemN]`. The items themselves may be lists. |
| `sys` | The name assigned to the system at its creation, for example `sys := LGraph_to_system(graph,outputs):` |
| `{inp,out}` | Input and output variable names. Optional for SISO systems but mandatory for MIMO systems. |
| `var` | A variable name for use in the output display, for example the Laplace variable in a transfer function. Usually a default name is available. |
| `xrange` | A Maple range specification of the form `min..max`, for example `Root_locus(mysys,B,0..150,200,-10..0,-20..20)` |

# 2. Alphabetical Function Descriptions:

## • A_matrix(sys)

Returns a system's state-space **A** matrix

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

**Example:**

```
This_system := Transfer_function_to_system(K/(m*s^2 + B*s + K)):
A_matrix(This_system);
```

**See Also:** B_matrix(), C_matrix(), D_matrix(), E_matrix(), F_matrix(), System_order(), State_variables(), System_inputs(), System_outputs()

## • Accel_error_constant(sys{,inp,out})

Returns the acceleration error constant $\lim_{s\to 0} s^2 G(s)$ of the transfer function $G(s)$ between the given input and output.

**Arguments:**

`sys` A Maple-Syrep system.
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)

**Comments:**

Used for specifying the steady-state error behavior in control system design.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
K := 10:   m := 5:   B := 4:
Accel_error_constant(Mech_sys);
```

**See Also:** Position_error_constant(), Velocity_error_constant()

## • Ackerman(plant,char_poly)

Uses Ackerman's formula to return the state feedback gain matrix to achieve the closed- loop pole placement specified by the roots of the supplied characteristic polynomial.

**Arguments:**

| | |
|---|---|
| `plant` | A Maple-Syrep SISO system representing the plant. |
| `char_poly` | The desired characteristic polynomial for the closed-loop system. |

**Comments:**

Defines a state-feedback gain matrix ( $1 \times n$, where $n$ is the system order) that defines the closed-loop system to have the same poles as the supplied characteristic polynomial.
The system must be controllable.

**Example:**

```
ol := Transfer_function_to_system(5/(s*(s+1)*(s+2)),s)
Gains := Ackerman(My_plant, (s+5)*(s^2 + 2*s + 5)):
cl := State_feedback(ol, Gains):
System_poles(cl);
```

**See Also:** State_feedback(), Controllability(), Controllability_matrix()

## • B_matrix(sys)

Returns a system's state-space **B** matrix

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |

**Comments:**

**Example:**

```
This_system := Transfer_function_to_system(K/(m*s^2 + B*s + K)):
B_matrix(This_system);
```

**See Also:** A_matrix(), C_matrix(), D_matrix(), E_matrix(), F_matrix(), System_order(), State_variables(), System_inputs(), System_outputs()

## • Bode_magnitude(sys,frange{,inp,out})

Displays a Bode magnitude plot (in decibels) of the frequency response function between a specified input and output.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `frange` | A Maple "range" of the form $\omega_{min} \ldots \omega_{max}$ indicating the frequency range (rad/s) for the plot. |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |

**Comments:**

The system `sys` must be fully evaluated, that is all parameters must have numeric values.
Because of the logarithmic frequency scale, the lower frequency limit cannot be 0.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
Bode_magnitude(hydraulic, 0.1...1000, u_1, y_2);
```

**See Also:** Bode_phase(), Polar_plot(), LogMag_phase()

## • Bode_phase(sys,frange{,inp,out})

Displays a Bode phase plot (in degrees) of the frequency response function between a specified input and output.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `frange` | A Maple "range" of the form $\omega_{min} \ldots \omega_{max}$ indicating the frequency range (rad/s) for the plot. |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |

**Comments:**

The system `sys` must be fully evaluated, that is all parameters must have numeric values. Because of the logarithmic frequency scale, the lower frequency limit cannot be 0.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
Bode_phase(hydraulic, 0.1...1000, u_1, y_2);
```

**See Also:** Bode_magnitude(), Polar_plot(), LogMag_phase()

## • C_matrix(sys)

Returns a system's state-space **C** matrix

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |

**Comments:**

**Example:**

```
This_system := Transfer_function_to_system(K/(m*s^2 + B*s + K)):
C_matrix(This_system);
```

**See Also:** A_matrix(), B_matrix(), D_matrix(), E_matrix(), F_matrix(), System_order(), State_variables(), System_inputs(), System_outputs()

## • Cascade_systems(sys1,sys2{,inp,out})

Generates a Maple-Syrep description description for two non-loading SISO systems connected in series, yhat is $G(s) = G_1(s)G_2(s)$.

**Arguments:**

| | |
|---|---|
| `sys1` | A Maple-Syrep SISO system. |
| `sys2` | A Maple-Syrep SISO system. |
| `inp` | Optional (see comments below). Defines the name of the input variable of the combined systems. |
| `out` | Optional (see comments below). Defines the name of the output variable of the combined systems. |

**Comments:**

Both `sys1` and `sys2` must be SISO.

If `inp` and `out` are specified the new system will have its input and output variables assigned these names; if absent the input variable name is taken from `sys1` and the output variable name from `sys2`.

**Example:**

```
first_order := TF_to_system(1/(tau*s+1),s):
second_order := TF_to_system(wn^2/(s^2 + 2*zeta*wn*s + wn2̂),s):
new_system := Parallel_systems(first_order, second_order);
```

**See Also:** SISO(), Parallel_systems(), Closed_loop_system(), Open_loop_system()

## • Closed_loop_system(plant,ctrl,feedback{,inp,out})

Returns a Maple-Syrep system description for a closed-loop SISO system with output feedback and controller and feedback dynamics..

**Arguments:**

| | |
|---|---|
| `plant` | A Maple-Syrep SISO system representing the plant. |
| `ctrl` | A Maple-Syrep SISO system representing the controller. |
| `feedback` | Optional. A Maple-Syrep SISO system representing the sensor/feedback dynamics. If omitted, a unity feedback system ($H(s) = 1$) is assumed. |
| `inp` | Optional. The name of the closed-loop input variable. The default name is `u_1`. |
| `out` | Optional. The name of the closed-loop output variable. The default name is `y_1`. |

**Comments:**

All three elements must be SISO.

**Example:**

```
Plant := TF_to_system(10/(s^2 + 2*s + 10),s):
Sensor := TF_to_system(1/(0.5*s+1), s):
cl := Closed_loop_system(Plant, PID(10, 0, 3), Sensor, reference, output):
plot(Step_response(cl), t=0..5);
```

**See Also:** Open_loop_system(), PID(), Gain(), Integrator(), Differentiator()

## • Controllability(sys)

Returns the rank of the system controllability matrix.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |

**Comments:**

A system is controllable if the controllability matrix has rank $n$ (the system order).

**Example:**

```
A:=matrix(2,2,[-1,1,-4,-4]):  B:=matrix(2,1,[0,4]):
C:=matrix(1,2,[0,1]):
hydraulic:=Matrices_to_system(A,B,C):
Controllability(hydraulic);
```

**See Also:** Controllability_matrix(), Observability_matrix(), Observability()

## • Controllability_matrix(sys)

Returns the system controllability matrix.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |

**Comments:**

The $(n \times nr)$ controllability matrix is defined $[\mathbf{B}\ \mathbf{AB}\ \mathbf{A}^2\mathbf{B} \ldots \mathbf{A}^{n-1}\mathbf{B}\ ]$.

**Example:**

```
A:=matrix(2,2,[-1,1,-4,-4]):  B:=matrix(2,1,[0,4]):
C:=matrix(1,2,[0,1]):
hydraulic:=Matrices_to_system(A,B,C):
Controllability_matrix(hydraulic);
```

**See Also:** Controllability(), Observability_matrix(), Observability()

## • D_matrix(sys)

Returns a system's state-space **D** matrix

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

**Example:**

```
This_system := Transfer_function_to_system(K/(m*s^2 + B*s + K)):
D_matrix(This_system);
```

**See Also:** A_matrix(), B_matrix(), C_matrix(), E_matrix(), F_matrix(), System_order(), State_variables(), System_inputs(), System_outputs()

## • Damping_ratio(sys)

Returns the damping ratio $\zeta$ of a second-order system.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

Generates an error message if the system is not second-order.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
Damping_ratio(Mech_system);
```

**See Also:** Time_constant(), Natural_frequency()

## • Differential_equation(sys{,inp,out})

Displays the differential equation relating a single output to a single input

**Arguments:**

`sys` A Maple-Syrep system
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)

**Comments:**

The differential equation is displayed in differential operator ($S$) notation.

**Example:**

```
second_order := TF_to_system(10/(s**2 + 5*s + 10), s, Vin, Vout):
Differential_equation(second_order, Vin, Vout);
```

**See Also:** Transfer_function(), Transfer_function_matrix()

## • Differentiator(K)

Generates a Maple-Syrep description description for a differentiator element $G(s) = Ks$.

**Arguments:**

`K` The differentiator gain.

**Comments:**

Useful in feedback control system design.

**Example:**

```
Plant := TF_to_system(10/(s^2 + 2*s + 10),s);
Inner_loop := Closed_loop_system(Plant, Gain(1), Differentiator(Kv));
Outer_loop := Closed_loop_system(Inner_loop, Gain(10), Gain(1));
Transfer_function(Outer_loop);
Root_locus(Outer_loop, Kv ,0...5, 200);
```

**See Also:** Gain(), Integrator(), PID(), Closed_loop_system(), Open_loop_system()

## • Display_digits(n)

Sets the display precision for numeric results.

**Arguments:**

`n` The number of digits to be displayed in numeric outputs.

**Comments:**

**Example:**

```
Display_digits(4):
```

**See Also:**

## • E_matrix(sys)

Returns a system's state-space **E** matrix

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

Note: The system in the example below does not have a **E** matrix.

**Example:**

```
This_system := Transfer_function_to_system(K/(m*s^2 + B*s + K)):
E_matrix(This_system);
```

**See Also:** A_matrix(), B_matrix(), C_matrix(), D_matrix(), F_matrix(), System_order(), State_variables(), System_inputs(), System_outputs()

## • Elements_to_system([elements],[constraints],[outputs]{,verbose})

Generates a Maple-Syrep description for a system specified by a set of elemental equations and constraint relationships.

**Arguments:**

| | |
|---|---|
| `[elements]` | A list of elemental equations. |
| `[constraints]` | A list of constraint equations. |
| `[outputs]` | A list of the system output variables. |
| `verbose` | Optional. Controls the verbosity of the output during computation of the system description. If absent, the system is silent. |

**Comments:**

See the Maple-Syrep User's Manual for the specification of the system structure through a a set of elemental and consraint equations.

**Example:**

```
elements := [dot(vC)=(1/C)*iC, dot(iL)=L*iL, vR=R*iR]:
constraints := [iC= L, iR=iL, vL=Vin-vC-vR]:
output := [vC, iC]:
series_RLC := Elements_to_system(elements, constraints, output, verbose);
```

**See Also:** Lgraph_to_system(), Zgraph_to_system(), TF_to_system(), Matrices_to_system()

## • F_matrix(sys)

Returns a system's state-space **F** matrix

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

Note: The system in the example below does not have a **F** matrix.

**Example:**

```
This_system := Transfer_function_to_system(K/(m*s^2 + B*s + K)):
F_matrix(This_system);
```

**See Also:** A_matrix(), B_matrix(), C_matrix(), D_matrix(), E_matrix(), System_order(), State_variables(), System_inputs(), System_outputs()

## • Final_value(sys,stepsize{,inp,out})

Returns the steady-state response to a step input of magnitude `stepsize`.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `stepsize` | The magnitude of the input step. |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |

**Comments:**

If the system is stable, returns the steady-state response.
If the system is unstable, or has poles at the origin, returns $\infty$.
If the system is an oscillator (poles on the imaginary axis), returns a *range* of the form $y_{min} \ldots y_{max}$ indicating the extremes of the oscillation.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
K := 10:   m := 5:   B := 4:
Final_value(Mech_system,1.0);
```

**See Also:** Step_response()

## • Find_magnitude(sys,mag{,inp,out}{,frange})

Returns the frequencies (rad/s) at which the frequency response function has a specified magnitude.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `mag` | The target magnitude (linear scale). |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |
| `frange` | Optional. A Maple range of the form $\omega_{min} \ldots \omega_{max}$ indicating the range of frequencies over which to search. The default is $0 \ldots 10000$. |

**Comments:**

The system `sys` must be fully evaluated, that is all parameters must have numeric values.
The argument *mag* must also have a numeric value.
If maple cannot solve the equations it returns an expression of the form `fsolve{...}`. In such cases it often helps to use `Bode_magnitude()` to estimate the approximate frequency, and then specify `frange` to bracket this frequency.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
Find_magnitude(hydraulic, 0.5, u_1, y_2, 0...100);
```

**See Also:** Find_phase(), Gain_margin(), Phase_margin()

## • Find_phase(sys,phase{,inp,out}{,frange})

Returns the frequencies (rad/s) at which the frequency response function has the the given phase *phase*.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `phase` | The target phase (rad). |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |
| `frange` | Optional. A Maple range of the form $\omega_{min} \ldots \omega_{max}$ indicating the range of frequencies over which to search. The default is $0 \ldots 10000$. |

**Comments:**

The system `sys` must be fully evaluated, that is all parameters must have numeric values.
The argument *phase* must also have a numeric value.
If maple cannot solve the equations it returns an expression of the form `fsolve{...}`. In such cases it often helps to use `Bode_phase()` to estimate the approximate frequency, and then specify `frange` to bracket this frequency.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
Find_phase(hydraulic, -Pi/4, u_1, y_2, 0...1000);
```

**See Also:** Find_magnitude(), Gain_margin(), Phase_margin()

## • Frequency_response(sys{,inp,out}{, var})

Returns the frequency response function beween a single input and a single output.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |
| `var` | The frequency response variable. The default is $\omega$. |

**Comments:**

The frequency response at a particular frequency may be found by substituting a numerical value for the argument `var`.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
```

```
FR := Frequency_response(hydraulic, u_1, y_2);
FR100 := Frequency_response(hydraulic, u_1, y_1, 100);
```

**See Also:** Frequency_response_matrix(), Frequency_response_magnitude(), Frequency_response_phase(), Bode_magnitude(), Bode_phase(). Polar_plot(), LogMag_phase()

## • Frequency_response_magnitude(sys{,inp,out}{, var})

Returns the frequency response magnitude function between a single input and a single output.

**Arguments:**

- `sys` A Maple-Syrep system.
- `inp` A system input for a MIMO system. (Optional for a SISO system.)
- `out` A system output for a MIMO system. (Optional for a SISO system.)
- `var` The frequency response variable. The default is $\omega$.

**Comments:**

The response magnitude at a particular frequency may be found by substituting a numerical value for the argument `var`.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
FRmag := Frequency_response_magnitude(hydraulic, u_1, y_2);
FRmag100 := Frequency_response_magnitude(hydraulic, u_1, y_1, 100);
```

**See Also:** Frequency_response_matrix(), Frequency_response(), Frequency_response_phase(), Bode_magnitude(), Bode_phase(), Polar_plot(), LogMag_phase()

## • Frequency_response_matrix(sys{, var})

Returns the frequency response matrix ($m \times r$) for a system with $m$ outputs and $r$ inputs, in which each element is the frequency response function between a single input and a single output.

**Arguments:**

- `sys` A Maple-Syrep system.
- `var` The frequency response variable. The default is $\omega$.

**Comments:**

The frequency response matrix at a particular frequency may be found by substituting a numerical value for the argument `var`.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
FRmat := Frequency_response_matrix(hydraulic);
FRmat100 := Frequency_response_matrix(hydraulic, 100);
```

**See Also:** Frequency_response(), Frequency_response_magnitude(), Frequency_response_phase(), Bode_magnitude(), Bode_phase(), Polar_plot(), LogMag_phase()

## • Frequency_response_phase(sys{,inp,out}{, var})

Returns the frequency response phase (radians) function between a single input and a single output.

**Arguments:**

`sys` A Maple-Syrep system.
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)
`var` The frequency response variable. The default is $\omega$.

**Comments:**

The response phase at a particular frequency may be found by substituting a numerical value for the argument `var`.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
FRphase := Frequency_response_phase(hydraulic, u_1, y_2);
FRphase100 := Frequency_response_phase(hydraulic, u_1, y_2, 100);
```

**See Also:** Frequency_response_matrix(), Frequency_response(), Frequency_response_magnitude(), Bode_magnitude(), Bode_phase(), Polar_plot(), LogMag_phase()

## • Gain(K)

Generates a Maple-Syrep description for a constant gain element $G(s) = K$.

**Arguments:**

`K` The system gain.

**Comments:**

Useful in feedback control system design. The following example uses the gain block to set up a proportional controller with a variable gain ($G_c(s) = K$) with unity feedback. The `Root_locus()` procedure is then used to construct a root-locus plot.

**Example:**

```
Plant := TF_to_system(10/(s^2 + 2*s + 10),s);
cl := Closed_loop_system(Plant,Gain(K),Gain(1));
Root_locus(cl,K,0...100,100)
```

**See Also:** Integrator(), Differentiator(), PID(), Closed_loop_system(), Open_loop_system()

## • Gain_margin(sys)

Returns the gain margin (dB) of an open-loop SISO system.

**Arguments:**

`plant` A Maple-Syrep SISO system representing the open-loop system.

**Comments:**

**Example:**

```
Open_loop := Transfer_function_to_system(5/(s*(s+1)*(s+2)),s)
Gain_margin(Open_loop);
```

**See Also:** Phase_margin(), Find_magnitude(), Find_phase()

## • Impluse_response(sys{,inp,out})

Returns the system impulse response function between a given input and output.

**Arguments:**

`sys` A Maple-Syrep system.
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)

**Comments:**

It is assumed that the system is at rest at time $t = 0$.

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s);
y_impulse := Impulse_response(my_system);
plot(y_impulse, t=0..10);
```

**See Also:** Step_response(), Ramp_response(), System_response()

## • Integrator(K)

Generates a Maple-Syrep description description for an integrator element $G(s) = K/s$.

**Arguments:**

`K` The integrator gain.

**Comments:**

Useful in feedback control system design.

**Example:**

```
Velocity := TF_to_system(1/(2*s + 1),s):
Position := Cascade_systems(Velocity, Integrator(1)):
Transfer_function(Position,s);
```

**See Also:** Gain(), Differentiator(), PID(), Closed_loop_system(), Open_loop_system()

## • Lgraph_to_system([lgraph],[outputs]{,verbose})

Generates a Maple-Syrep description for a system specified by a linear graph.

**Arguments:**

`[lgraph]` A list of lists defining the system structure as a linear graph, as described in the User Manual.
`[outputs]` A list of the system output variables
`verbose` Optional. Controls the "verbosity" of the output during computation of the system description. If absent, the system is silent.

**Comments:**

See the Maple-Syrep User's Manual for the format of the system structure through a linear graph.

**Example:**

```
graph := [[1,4,voltage,Vin],[1,2,D,R],[2,3,L,L],[3,4,A,C,vC,iC]]:
output := [vC]:
series_RLC := Lgraph_to_system(graph,output,verbose);
```

**See Also:** Zgraph_to_system(), TF_to_system(), Elements_to_system(), Matrices_to_system()

## • LogMag_phase(sys,frange{,inp,out}{,npoints})

Displays the log-magnitude (dB) versus phase (deg) form of the frequency response function.

**Arguments:**

`sys` A Maple-Syrep system.
`frange` A Maple "range" of the form $\omega_{min} \ldots \omega_{max}$ indicating the frequency range (rad/s) for the plot.
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)
`npoints` Optional. Defines the number of points to be used in the plot.

**Comments:**

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
LogMag_phase(hydraulic, 0...100, u_1, y_1, 200);
```

**See Also:** Bode_magnitude(), Bode_phase(), Polar_plot()

## • Matrices_to_system(A,B,{,C{,D{,E{,F}}}})

Generates a Maple-Syrep description for a system specified by a set of state-space matrices.

**Arguments:**

- `A` The system **A** matrix
- `B` The system **B** matrix.
- `C` Optional. The system **C** matrix.
- `D` Optional. The system **D** matrix.
- `E` Optional. The system **E** matrix.
- `F` Optional. The system **F** matrix.

**Comments:**

Matrices may be entered in any valid Maple format. See the Maple-Syrep User's Manual.

If the matrix **C** is not specified, it is assumed to an identity matrix of the same size as the **A** matrix, that is the output vector is defined to be the state vector.

The state variables are assigned names `x_1...x_n`, the inputs are named `u_1...u_r` and the outputs `y_1...y_m`.

Note the letter "D" is a reserved word in Maple, and cannot be used as a variable name. In the example below, the **D** matrix is named "D1".

**Example:**

```
A := matrix(2, 2, [0, 1/C, -1/L, -R/L]):
B := matrix(2, 1, [0, 1/L]):
C := matrix(3, 2, [ -1, -R, 0, 1, 1, 0]):
D1 := matrix(3, 1, [1, 0, 0]):
series_RLC := Matrices_to_system(A, B, C, D1):
```

**See Also:** Lgraph_to_system(), Zgraph_to_system(), TF_to_system(), Elements_to_system()

## • Natural_frequency(sys)

Returns the undamped natural frequency $\omega_n$ (rad/s) of a second-order system.

**Arguments:**

- `sys` A Maple-Syrep system.

**Comments:**

Generates an error message if the system is not second-order.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
Natural_frequency(Mech_system);
```

**See Also:** Time_constant(), Damping_ratio()

## • Observability(sys)

Returns the rank of the system observability matrix.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

A system is observable if the observability matrix has rank $n$ (the system order).

**Example:**

```
A:=matrix(2,2,[-1,1,-4,-4]):  B:=matrix(2,1,[0,4]):
C:=matrix(1,2,[0,1]):
hydraulic:=Matrices_to_system(A,B,C):
Observability(hydraulic);
```

**See Also:** Controllability_matrix(), Controllability(), Observability_matrix()

## • Observability_matrix(sys)

Returns the system observability matrix.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

The $(n \times nm)$ observability matrix is defined $[\mathbf{C}^T\ \mathbf{A}^T\mathbf{C}^T\ (\mathbf{A}^T)^2\mathbf{C}^T\ \ldots(\mathbf{A}^T)^{n-1}\mathbf{C}^T\ ]$.

**Example:**

```
A:=matrix(2,2,[-1,1,-4,-4]):  B:=matrix(2,1,[0,4]):
C:=matrix(1,2,[0,1]):
hydraulic:=Matrices_to_system(A,B,C):
Observability_matrix(hydraulic);
```

**See Also:** Controllability_matrix(), Controllability(), Observability()

## • Open_loop_system(plant,ctrl,feedback{,inp,out})

Returns a Maple-Syrep system description for an open-loop SISO system with output feedback and controller and feedback dynamics..

**Arguments:**

| | |
|---|---|
| `plant` | A Maple-Syrep SISO system representing the plant. |
| `ctrl` | A Maple-Syrep SISO system representing the controller. |
| `feedback` | Optional. A Maple-Syrep SISO system representing the sensor/feedback dynamics. If omitted, a unity feedback system ($H(s) = 1$) is assumed. |
| `inp` | Optional. The name of the closed-loop input variable. The default name is `u_1`. |
| `out` | Optional. The name of the closed-loop output variable. The default name is `y_1`. |

**Comments:**

All three elements must be SISO.

**Example:**

```
Plant := TF_to_system(10/(s^2 + 2*s + 10),s):
Sensor := TF_to_system(1/(0.5*s+1), s):
Ol := Open_loop_system(Plant, PID(10, 0, 3), Sensor):
Gain_margin(ol);
Phase_margin(ol);
```

**See Also:** Closed_loop_system(), PID(), Gain(), Integrator(), Differentiator()

## • Output_IC_response(sys,ic)

Returns the response of a SISO system to a given set of initial conditions on the system output variable.

**Arguments:**

`sys` A Maple-Syrep system.

`ic` For a system of order $n$, a vector of initial conditions containing the output $y(0)$ and its first $(n-1)$ derivatives.

**Comments:**

The following example display and plots the system initial condition response to the initial output conditions $y(0) = 2,\ dy/dt|_{t=0} = 1,\ d^2y/dt^2\big|_{t=0} = 0$.

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s);
y_ic := Output_IC_response(my_system, [2,1,0]);
plot(y_ic, t=0..10);
```

**See Also:** State_IC_response()

## • Parallel_systems(sys1,sys2{,inp,out})

Generates a Maple-Syrep description for two non-loading SISO systems connected in parallel, that is $G(s) = G_1(s) + G_2(s)$.

**Arguments:**

`sys1` A Maple-Syrep SISO system.

`sys2` A Maple-Syrep SISO system.

`inp` Optional (see comments below). Defines the name of the input variable of the combined systems.

`out` Optional (see comments below). Defines the name of the output variable of the combined systems.

**Comments:**

Both `sys1` and `sys2` must be SISO.

If `inp` and `out` are specified the new system will have its input and output variables assigned these names; if absent the input and output variables will be assigned the variable names from `sys1`.

**Example:**

```
first_order := TF_to_system(1/(tau*s+1),s):
second_order := TF_to_system(wn^2/(s^2 + 2*zeta*wn*s + wn^2),s):
new_system := Cascade_systems(first_order, second_order);
```

**See Also:** SISO(), Cascade_systems(), Closed_loop_system(), Open_loop_system()

## • Phase_margin(sys)

Returns the phase margin (degrees) of an open-loop SISO system.

**Arguments:**

`plant` A Maple-Syrep SISO system representing the open-loop system.

**Comments:**

**Example:**

```
Open_loop := Transfer_function_to_system(5/(s*(s+1)*(s+2)),s)
Phase_margin(Open_loop);
```

**See Also:** Gain_margin(), Find_magnitude(), Find_phase()

## • PID(Kp,Ki,Kd)

Generates a Maple-Syrep description description for a PID (proportional + integral + derivative) controller element $G(s) = K_p + K_i/s + K_d s$.

**Arguments:**

`Kp` The controller proportional gain.
`Ki` The controller integral gain.
`Kd` The controller integral gain.

**Comments:**

Useful in feedback control system design. The following example uses a PID element to set up a PD controller with a second-order plant.

**Example:**

```
Plant := TF_to_system(10/(s^2 + 2*s + 10),s);
cl := Closed_loop_system(Plant, PID(10, 0, 0.5), Gain(1));
```

**See Also:** Gain(), Integrator(), Differentiator(), Closed_loop_system(), Open_loop_system()

## • Polar_plot(sys,frange{,inp,out}{,npoints})

Displays the frequency response function in polar (Nyquist) form.

**Arguments:**

`sys` A Maple-Syrep system.
`frange` A Maple "range" of the form $\omega_{min} \ldots \omega_{max}$ indicating the frequency range (rad/s) for the plot.
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)
`npoints` Optional. Defines the number of points to be used in the plot.

**Comments:**

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(2,2,[1,0,0,1]):
hydraulic := Matrices_to_system(A,B,C):
Polar_plot(hydraulic, 0...100, u_1, y_2, 500);
```

**See Also:** Bode_magnitude(), Bode_phase(), LogMag_phase()

## • Pole_zero_plot(sys{,inp,out})

Displays the pole-zero plot (for a fully evaluated system), between a given input and a given output.

**Arguments:**

`sys` A Maple-Syrep system.
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)

**Comments:**

The system must be fully evaluated (numeric values assigned to all parameters) to enable the plotting of poles and zeros.

**Example:**

```
This_system := TF_to_system((13*s+26)/(s^3 + 7*s2 + 19*s + 13), s):
Pole_zero_plot(This_system, u_, y_);
```

**See Also:** System_char_poly(), System_eigenvalues(), System_poles(), System_zeros(), Root_locus()

## • Position_error_constant(sys{,inp,out})

Returns the position error constant $\lim_{s \to 0} G(s)$ of the transfer function $G(s)$ between the given input and output.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |

**Comments:**

Used for specifying the steady-state error behavior in control system design.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
K := 10:   m := 5:   B := 4:
Position_error_constant(Mech_sys);
```

**See Also:** Accel_error_constant(), Velocity_error_constant()

## • Ramp_response(sys{,inp,out})

Returns the system unit-ramp response function between a given input and output.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |

**Comments:**

It is assumed that the system is at rest at time $t = 0$.

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s);
y_ramp := Ramp_response(my_system);
plot(y_ramp, t=0..10);
```

**See Also:** Step_response(), Impulse_response(), System_response()

## • Rename_inputs(sys,[newnames])

Renames the input variables of a system.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `[newnames]` | A list of names for the state vector. |

**Comments:**

The whole input vector must be renamed.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
Rename_inputs(Mech_system,[Force]):
Rename_outputs(Mech_system,[velocity]):
Rename_states(Mech_system,[x1, x2]):
```

**See Also:** Rename_outputs(), Rename_states()

## • Rename_outputs(sys,[newnames])

Renames the output variables of a system.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `[newnames]` | A list of names for the state vector. |

**Comments:**

The whole input vector must be renamed.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
Rename_inputs(Mech_system,[Force]):
Rename_outputs(Mech_system,[velocity]):
Rename_states(Mech_system,[x1, x2]):
```

**See Also:** Rename_inputs(), Rename_states()

## • Rename_states(sys,[newnames])

Renames the state variables of a system.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `[newnames]` | A list of names for the state vector. |

**Comments:**

The whole input vector must be renamed.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
Rename_inputs(Mech_system,[Force]):
Rename_outputs(Mech_system,[velocity]):
Rename_states(Mech_system,[x1, x2]):
```

**See Also:** Rename_outputs(), Rename_outputs()

## • Root_locus(sys,param,prange,nsteps{,xrange}{,yrange})

Displays a *generalized* root locus plot for any system parameter `param` over a given range.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `param` | A system parameter (system element name). |
| `prange` | A Maple range (for example 0..100) over which the the root loci will be computed. |
| `nsteps` | The number of steps over `prange` that will be used to display the root loci. |
| `xrange` | Optional. Defines the plotting range for the real axis of the $s$- plane. |
| `yrange` | Optional. Defines the plotting range for the imaginary axis of the $s$-plane. |

**Comments:**

This is a generalized root locus function. It does not assume a closed-loop system structure.
All system elements except for `param` must be assigned numeric values.
The following example illustrates the locus of the poles as a frictional damping coefficient is varied.

**Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
K := 10:   m := 5:
Root_locus(Mech_system, B, 0.50, 25, -10.0, -10..10)
```

**See Also:** System_char_poly(), System_eigenvalues(), System_poles(), System_zeros(), Pole_zero_plot(),

## • SISO(sys,inp,out)

Generates a Maple-Syrep description for a SISO system from a MIMO system.

**Arguments:**

| | |
|---|---|
| `sys1` | A Maple-Syrep system. |
| `inp` | The input variable of the system `sys` that defines the SISO system input. |
| `out` | The output variable of the system `sys` that defines the SISO system output. |

**Comments:**
Useful for interfacing MIMO systems to procedures that require a SISO description.

**Example:**
```
A:=matrix(2,2,[a11,a12,a21,a22]):  B:=matrix(2,2,[1,0,0,1]):
C:=matrix(2,2,[1,0,0,1]):
MIMO_sys := Matrices_to_system(A,B,C):
New_system := Cascade_systems(SISO(MIMO_sys, u_1, y_2), Integrator(1)):
```

**See Also:** Cascade_systems(), Parallel_systems(), Closed_loop_system(), Open_loop_system()

## • State_equations(sys)

Displays the system state and output equations in matrix form

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |

**Comments:**
Displays the system state equations.

**Example:**
```
second_order := TF_to_system(10/(s^2 + 5*s + 10), s, Vin, Vout):
State_equations(second_order);
```

**See Also:** State_equations(), Transfer_function(), Transfer_function_matrix()

## • State_feedback(plant,gain_matrix)

Returns a Maple-Syrep system description for the closed-loop system with full-state feedback as specified by the gain matrix.

**Arguments:**

| | |
|---|---|
| `plant` | A Maple-Syrep system representing the plant. |
| `gain_matrix` | A $n \times r$ matrix $\mathbf{K}$ of feedback gains such that $\mathbf{u}$ = -$\mathbf{Kx}$. |

**Comments:**
Forms a system with the new A-matrix $\mathbf{A} - \mathbf{BK}$.
The gain matrix $K$ should be $1 \times n$ where $n$ is the system order.
The system must be controllable.

**Example:**
```
ol := Transfer_function_to_system(5/(s*(s+1)*(s+2)),s)
Gains := Ackerman(My_plant, (s+5)*(s^2 + 2*s + 5)):
cl := State_feedback(ol, Gains):
System_poles(cl);
```

**See Also:** Ackerman(), Controllability(), Controllability_matrix()

## • State_IC_response(sys,ic,{out})

Returns the response of a given system output to a set of initial conditions on the system state vector.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `ic` | For a system of order $n$, a vector specifying the state at time $t = 0$ seconds, that is $x_1(0), x_2(0) \ldots x_n(0)$. |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |

**Comments:**
The following example display and plots the system initial condition response to the initial state $x_1(0) = 0$, $x_2(0) = 1$.

**Example:**

```
A := matrix(2,2,[-1,1,-4,-4]):  B := matrix(2,1,[0,4]):
C := matrix(1,2,[0,1]):
hydraulic := Matrices_to_system(A,B,C):
y_ic := State_IC_response(hydraulic, [0,1], y_1);
plot(y_ic, t=0..10);
```

**See Also:** Output_IC_response()

## • State_variables(sys)

Returns a vector containing the names of the state variables of the specified system.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

If defined in the system structure, the assigned names will be returned, otherwise the internally generated names *x_1* ... *x_n* will be returned. ($n$ is the system order).

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s);
states := State_variables(my_system);
```

**See Also:** A_matrix(), B_matrix(), C_matrix(), D_matrix(), E_matrix(), F_matrix(), System_order(), System_inputs(), System_outputs()

## • Step_response(sys{,inp,out})

Returns the system unit-step response function between a given input and output.

**Arguments:**

`sys` A Maple-Syrep system.
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)

**Comments:**

It is assumed that the system is at rest at time $t = 0$.

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s:;
y_step := Step_response(my_system);
plot(y_step, t=0..10);
```

**See Also:** Impulse_response(), Ramp_response(), System_response()

## • System_char_poly(sys{,var})

Returns the system characteristic polynomial.

**Arguments:**

`sys` A Maple-Syrep system.
`var` Optional. The characteristic polynomial variable for output display.The default is $s$.

**Comments:**

The following example returns the characteristic polynomial in the variable $\lambda$. The coefficients will be expressed in terms of the symbolic matrix elements $a_{ij}$.

**Example:**

```
A:=matrix(2,2,[a11,a12,a21,a22]):  B:=matrix(2,2,[1,0,0,1]):
C:=matrix(2,2,[1,0,0,1]):
MIMO:=Matrices_to_system(A, B, C):
System_char_poly(MIMO,lambda);
```

**See Also:** System_eigenvalues(), System_poles(), System_zeros(), Pole_zero_plot(), Root_locus()

## • System_eigenvalues(sys)

Returns the system eigenvalues (eigenvalues of the **A** matrix, or roots of the characteristic equation). (Equivalent to the function System_poles().)

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

The system usually should be fully evaluated (numeric values assigned to all parameters) for meaningful results.

**Example:**

```
A:=matrix(2,2,[0,K,-1/m,0]):  B:=matrix(2,1,[0,1/m]):
mass_spring := Matrices_to_system(A,B):
m := 5:  K := 100:
System_eigenvalues(mass_spring);
```

**See Also:** System_char_poly(), System_poles(), System_zeros(), Pole_zero_plot(), Root_locus()

## • System_exponential(sys,t)

Returns the matrix exponential $e^{\mathbf{A}t}$ of the specified system.

**Arguments:**

`sys` A Maple-Syrep system.
`t` Time (s).

**Comments:**

**Example:**

```
A:=matrix(2,2,[-1,1,-4,-4]):  B:=matrix(2,1,[0,4]):
C:=matrix(1,2,[0,1]):
hydraulic:=Matrices_to_system(A,B,C):
System_exponential(hydraulic,0.5);
```

**See Also:**

## • System_inputs(sys)

Returns a vector containing the names of the system inputs.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

If defined in the system structure, the assigned names will be returned, otherwise the internally generated names *u_1* ... *u_r* will be returned. ($r$ is the number of inputs).

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s);
inputs := System_inputs(my_system);
```

**See Also:** A_matrix(), B_matrix(), C_matrix(), D_matrix(), E_matrix(), F_matrix(), System_order(), State_variables(), System_outputs()

## • System_order(sys)

Returns the order of the specified system.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s);
ord := System_order(my_system);
```

**See Also:** A_matrix(), B_matrix(), C_matrix(), D_matrix(), E_matrix(), F_matrix(), State_variables(), System_inputs(), System_outputs()

- **System_outputs(sys)**

Returns a vector containing the names of the system outputs.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

If defined in the system structure, the assigned names will be returned, otherwise the internally generated names *y_1* ... *y_m* will be returned. ($m$ is the number of outputs).

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s);
out_vars := System_outputs(my_system);
```

**See Also:** A_matrix(), B_matrix(), C_matrix(), D_matrix(), E_matrix(), F_matrix(), System_order(), State_variables(), System_inputs()

- **System_poles(sys)**

Returns the system poles (roots of the characteristic equation, or eigenvalues of the **A** matrix). (Equivalent to the function System_eigenvalues().)

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

The system should be fully evaluated (numeric values assigned to all parameters) for meaningful results.

**Example:**

```
mass_spring := Transfer_function_to_system((K/m)/(s^2 + K/m))
m := 5:  K := 100:
System_poles(mass_spring);
```

**See Also:** System_char_poly(), System_eigenvalues(), System_zeros(), Pole_zero_plot(), Root_locus()

- **System_response(sys,function,{,inp,out})**

Returns the response of a system to an arbitrary input function.

**Arguments:**

| | |
|---|---|
| `sys` | A Maple-Syrep system. |
| `function` | The system input function. |
| `inp` | A system input for a MIMO system. (Optional for a SISO system.) |
| `out` | A system output for a MIMO system. (Optional for a SISO system.) |

**Comments:**

It is assumed that the system is at rest at time $t = 0$.

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), Fin, Vout,s);
y := System_response(my_system, 1-exp(-3*t));
plot(y, t=0..10);
```

**See Also:** Step_response(), Impulse_response(), Ramp_response()

## • System_type(sys)

Returns the system "type", that is the number of free integrators (poles at the origin of the $s$-plane) in the system transfer function.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

The system type is used for determining the error behavior of systems.

**Example:**

```
This_system := TF_to_system((13*s+26)/(s^2*(s^3 + 7*s2 + 19*s + 13)), s):
System_type(This_system);
```

**See Also:**

## • System_zeros(sys{,inp,out})

Returns a list of the system zeros (roots of the numerator of the transfer function ) between a given input and a given output.

**Arguments:**

`sys` A Maple-Syrep system.
`inp` A system input for a MIMO system. (Optional for a SISO system.)
`out` A system output for a MIMO system. (Optional for a SISO system.)

**Comments:**

The system should be fully evaluated (numeric values assigned to all parameters) for meaningful results.

**Example:**

```
A:=matrix(2,2,[-1,1,-4,-4]):  B:=matrix(2,1,[0,4]):
C:=matrix(1,2,[0,1]):
hydraulic:=Matrices_to_system(A,B,C):
System_zeros(hydraulic);
```

**See Also:** System_char_poly(), System_eigenvalues(), System_poles(), Pole_zero_plot(), Root_locus()

## • TF_to_system(transfer_function{,var}{,inp,out})

Generates a Maple-Syrep description for a SISO system specified by a transfer function.

**Arguments:**

| | |
|---|---|
| `transfer_function` | The system's transfer function. |
| `var` | Optional. The transfer function variable used in `transfer_function`. The default is $s$. |
| `inp` | Optional. The name of the input variable |
| `out` | Optional. The name of the output variable. |

**Comments:**

If `inp` and `out` are not specified they are assigned internal names *u_1* and *y_1* respectively.

**Example:**

```
my_system := TF_to_system((3*s + 1)/((s+1)*(s^2 + 2*s + 5)), s, Fin, Vout):
```

**See Also:** Lgraph_to_system(), Zgraph_to_system(), Elements_to_system(), Matrices_to_system()

- **Time_constant(sys)**

Returns the time-constant $\tau$ (s.) of a first-order system.

**Arguments:**

`sys` A Maple-Syrep system.

**Comments:**

Generates an error message if the system is not first-order.

**Example:**

```
RC_circuit := TF_to_system(1/(R*C*s + 1), s):
Time_constant(RC_circuit);
```

**See Also:** Natural_frequency(), Damping_ratio()

- **Transfer_function(sys{,inp,out}{,var})**

Returns the transfer function relating a single output to a single input for the specified system.

**Arguments:**

`sys` A Maple-Syrep system.
`inp` The selected system input variable. (Optional for a SISO system.)
`out` The selected system output variable. (Optional for a SISO system.)
`var` Optional. The transfer function variable for output display. The default is $s$.

**Comments:**

**Example:**

```
A:=matrix(2,2,[a11,a12,a21,a22]):  B:=matrix(2,1,[0,1]):
C: = matrix(1,2,[0,1]):
This_system:=Matrices_to_system(A, B, C):
Transfer_function(This_system, s);
```

**See Also:** State_equations(), Differential_equation(), Transfer_function_matrix()

- **Transfer_function_matrix(sys{,var})**

Returns the transfer function matrix relating all outputs to all inputs for the specified system.

**Arguments:**

`sys` A Maple-Syrep system.
`var` Optional. The transfer function variable for output display. The default is $s$.

**Comments:**

For a $r$ input and $m$ output system, displays a $m \times r$ symbolic matrix in which the $i, j$th element is the transfer function relating output $i$ to input $j$. In displaying the output, the denominator polynomial will often be displayed as a sub- expression of the form "%1 = ..." outside the matrix.

**Example:**

```
A:=matrix(2,2,[a11,a12,a21,a22]):  B:=matrix(2,2,[1,0,0,1]):
C:=matrix(2,2,[1,0,0,1]):
MIMO:=Matrices_to_system(A, B, C):
Transfer_function_matrix(MIMO,lambda);
```

**See Also:** State_equations(), Differential_equation(), Transfer_function()

- **Velocity_error_constant(sys{,inp,out})**

  Returns the velocity error constant $\lim_{s\to 0} sG(s)$ of the transfer function $G(s)$ between the given input and output.

  **Arguments:**

  `sys` A Maple-Syrep system.
  `inp` A system input for a MIMO system. (Optional for a SISO system.)
  `out` A system output for a MIMO system. (Optional for a SISO system.)

  **Comments:**

  Used for specifying the steady-state error behavior in control system design.

  **Example:**

```
Mech_system := TF_to_system((K)/(m*s^2 + B*s + K), s):
K := 10:   m := 5:   B := 4:
Velocity_error_constant(Mech_sys);
```

  **See Also:** Position_error_constant(), Accel_error_constant()

- **Version()**

  Returns the version identification of the current Maple-Syrep.

  **Arguments:**

  **Comments:**

  **Example:**

```
Version()
```

  **See Also:**

- **Y_parallel(Y1, Y2,...,Yn)**

  Returns the admittance of an arbitrary number of parallel connected admittance elements, $Y = Y_1 + \ldots + Y_n$.

  **Arguments:**

  `Y1...Yn` A set of $n$ comma separated admittance expressions.

  **Comments:**

  Useful in formulating system structures for the system generation procedure `Zgraph_to_system()`.

  **Example:**

```
Y1 := s*C: Y2 := 1/R:
Y := Y_series(Y1, Y2, 1/(s*L)):
```

  **See Also:** Z_series(), Z_parallel(), Y_seriws(), Z_to_Y(), Y_to_Z()

- **Y_series(Y1, Y2,...,Yn)**

  Returns the admittance of an arbitrary number of series connected admittance elements, $1/Y = 1/Y_1 + \ldots + 1/Y_n$.

  **Arguments:**

  `Y1...Yn` A set of $n$ comma separated admittance expressions.

  **Comments:**

  Useful in formulating system structures for the system generation procedure `Zgraph_to_system()`.

**Example:**

```
Y1 := s*C: Y2 := 1/R:
Y := Y_parallel(Y1, Y2, 1/(s*L)):
```

**See Also:** Z_series(), Z_parallel(), Y_parallel(), Z_to_Y(), Y_to_Z()

- **Y_to_Z(Y)**

Returns the impedance of an admittance element *Z = 1/Y*.

**Arguments:**

`Y` An admittance expression.

**Comments:**

Useful in formulating system structures for the system generation procedure `Zgraph_to_system()`.

**Example:**

```
Y1 := s*C: Y2 := 1/R:
Z := Y_to_Z(Y_series(Y1, Y2, 1/(s*L))):
```

**See Also:** Z_series(), Z_parallel(), Y_series(), Y_parallel(), Z_to_Y()

- **Z_parallel(Z1, Z2,. . . ,Zn)**

Returns the impedance of the series connection of arbitrary number of parallel connected impedance elements $1/Z = 1/Z_1 + \ldots + 1/Z_n$.

**Arguments:**

`Z1...Zn` A set of $n$ comma separated impedance expressions.

**Comments:**

Useful in formulating system structures for the system generation procedure `Zgraph_to_system()`.

**Example:**

```
Z1:=s/K: Z2:=1(/s*m):  Z3:=1/B:
Ztotal := Z_series(Z1, Z2, Z3, s/K1):
```

**See Also:** Z_parallel(), Y_series(), y_parallel(), Z_to_Y(), Y_to_Z()

- **Z_series(Z1, Z2,. . . ,Zn)**

Returns the impedance of the series connection of arbitrary number of parallel connected impedance elements $Z = Z_1 + \ldots + Z_n$.

**Arguments:**

`Z1...Zn` A set of $n$ comma separated impedance expressions.

**Comments:**

Useful in formulating system structures for the system generation procedure `Zgraph_to_system()`.

**Example:**

```
Z1:=s/K: Z2:=1(/s*m):  Z3:=1/B:
Zequiv := Z_parallel(Z1, Z2, Z3, s/K1):
```

**See Also:** Z_series(), Y_series(), Y_parallel(), Z_to_Y(), Y_to_Z()

- **Z_to_Y(Z)**

Returns the admittance of an impedance element *Y = 1/Z*.

**Arguments:**

`Z` An impedance expression.

**Comments:**

Useful in formulating system structures for the system generation procedure `Zgraph_to_system()`.

**Example:**

```
Z1:=s/K: Z2:=1(/s*m):  Z3:=1/B:
Y := Z_to_Y(Z1 + Z2 + Z3):
```

**See Also:** Z_series(), Z_parallel(), Y_series(), Y_parallel(), Y_to_Z()

## • Zgraph_to_system([zgraph],[outputs]{,verbose})

Generates a Maple-Syrep description for a SISO system specified by an impedance graph.

**Arguments:**

| | |
|---|---|
| `[zgraph]` | A list of lists defining the system structure as an impedance graph, as described in the User Manual. |
| `[output]` | A single-element list (or a single system variable) specifying the system output variable. |
| `verbose` | Optional. Controls the "verbosity" of the output during computation of the system description. If absent, the system is silent. |

**Comments:**

See the Maple-Syrep User's Manual for the format of the system structure through an impedance graph.

Note that the system must be SISO.

**Example:**

```
imp_graph := [[1,3,voltage,Vin],[1,2,Z,R + s*L],[2,3,Y,s*C,vC,iC]]:
output := [vC]:
series_RLC := Zgraph_to_system(imp_graph,output,verbose);
```

**See Also:** Lgraph_to_system(), TF_to_system(), Elements_to_system(), Matrices_to_system()