Maple SYREP

Version 2.1

Quick Reference

1. Loading SYREP:

At the Maple prompt:

`restart:`	(not strictly required, but a good idea)
`read '/syrepdir/syrep.m':`	(`/syrepdir/` is path to SYREP's directory; note the use of back-quotes (') and forward slashes (/).)
`Display_digits(5):`	(Set display precision to desired level.)

2. 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)`

3. Principal System Generation Procedures:

Procedure	**Function**
`LGraph_to_system([lgraph],[outputs]{,verbose})`	Generate a system description from a linear graph
`ZGraph_to_system([zgraph],[outputs]{,verbose})`	Generate a system description from an impedance graph.
`Elements_to_system([elements],[constraints], [outputs]{,verbose})`	Generate a system description from a set of elemental and constraint equations.
`TF_to_system(transfer_function{,var}{,inp,out})`	Generate a system description directly from a transfer function.
`Matrices_to_system(A,B{,C{,D{,E{,F}}}})`	Generate a system description directly from supplied state-space matrices.

3.1 Source specification sub-list: (LGraph_to_system(), ZGraph_to_system())

List Structure: [tail_node, head_node, source_type, source_variable]
Examples: `[1,2,velocity,Vsource]`, `[2,4,current,I]`, `[2,3,As,Vin]`

Across-variable Source	Through-variable source	Primary domains
`As`	`Ts`	generalized sources
`velocity`	`force`	translational
`angular_velocity`	`torque`	rotational
`voltage`	`current`	electrical
`pressure`	`flow, flow_rate`	fluid
`temperature`	`heat, heat_flow_rate`	thermal

3.2 One-port element specification sub-list: (LGraph_to_system())

List Structure: [tail_node, head_node, 1port_type, parameter {,across_variable, through_variable}]
Examples: `[3,5,inertia,J,WJ,TJ]`, `[2,4,D,3000]`, `[8,4,A,m3,vm3,Fm3]`

Element type name	Parameter	Elemental relationship	Primary domains
`A`	generalized A-type (C)	$\dot{v} = (1/C)f$	generalized
`C`	generalized capacitance (C)	$\dot{v} = (1/C)f$	generalized
`mass`	mass (m)	$\dot{v} = (1/m)f$	translational
`inertia`	rotary inertia (J)	$\dot{v} = (1/J)f$	rotational
`capacitance`	capacitance (C)	$\dot{v} = (1/C)f$	electrical,fluid,thermal
`T`	generalized T-type (L)	$\dot{f} = (1/L)v$	generalized
`L`	generalized inductance (L)	$\dot{f} = (1/L)v$	generalized
`spring`	stiffness (K)	$\dot{f} = Kv$	translational, rotational
`stiffness`	stiffness (K)	$\dot{f} = Kv$	translational, rotational
`compliance`	compliance (C)	$\dot{f} = (1/C)v$	translational, rotational
`inductance`	inductance (L)	$\dot{f} = (1/L)v$	electrical
`inertance`	inertance (I)	$\dot{f} = (1/I)v$	fluid
`D`	generalized D-type (R)	$v = Rf$	generalized
`R`	generalized resistance (R)	$v = Rf$	generalized
`G`	generalized conductance (G)	$f = Gv$	generalized
`damper`	viscous coefficient (B)	$f = Bv$	translational, rotational
`dashpot`	viscous coefficient (B)	$f = Bv$	translational, rotational
`resistance`	resistance (R)	$v = Rf$	electrical, fluid, thermal
`conductance`	conductance (G)	$f = Gv$	electrical, fluid, thermal

3.3 Impedance element sub-list: (ZGraph_to_system())

List Structure: [tail_node, head_node, branch_type, impedance {,across_variable, through_variable}]
Examples: `[2,3,Z,R+s*L,vL,iL]`, `[4,3,admittance,s*m+B]`, `[11,7,Y,13.33+s*15.1]`

Branch type name	Impedance/Admittance	Elemental relationship	Domains
`impedance`	impedance (Z)	$V(s) = ZF(s)$	all domains
`Z`	impedance (Z)	$V(s) = ZF(s)$	all domains
`addmittance`	admittance (Y)	$F(s) = YV(s)$	all domains
`Y`	admittance (Y)	$F(s) = YV(s)$	all domains

3.4 Two-port element sub-list: (LGraph_to_system(), ZGraph_to_system())

Structure: [[tail_a,head_a],[tail_b,head_b] 2port_type,ratio {,[across_a through_a], [across_b,through_b]}]
Examples: `[[2,3],[7,8],transformer,1/Km,[vback,imotor],[Wm,Tm]]`, `[[1,2],[5,3],GY,r]`

Two-port type name	Ratio	Elemental relationships	Domains
`TF` `transformer`	r	$vb = r.va$ $fb = -(1/r)fa$	all domains
`GY` `gyrator`	r	$vb = r.fa$ $fb = -(1/r)va$	all domains

4. Additional System Generation Procedures:

Procedure	Function
`Gain(K)`	Generate a system description for a gain block with gain K.
`Integrator(K)`	Generate a system description for an integrator with gain K.
`Differentiator(K)`	Generate a system description for a differentiator with gain K.
`PID(Kp,Ki,Kd)`	Generate a system description for a PID controller.
`Cascade_systems(sys1,sys2{,input,output})`	Combine two (non-loading) systems in series: $H(s) = H_1(s)H_2(s)$).
`Parallel_systems(sys1,sys2{,input,output})`	Combine two (non-loading) systems in parallel: $H(s) = H_1(s) + H_2(s)$.
`SISO(sys,inp,out)`	Generates a SISO system from a MIMO system.

5. Impedance manipulation Procedures:

Procedure	Function
Z_series(Z1, Z2, ..., Zn)	Returns the impedance of the series connection of an arbitrary number of impedances.
Z_parallel(Z1, Z2, ..., Zn)	Returns the impedance of the parallel connection of an arbitrary number of impedances.
Y_series(Y1, Y2, ..., Yn)	Returns the admittance of the series connection of an arbitrary number of admittances.
Y_parallel(Y1, Y2, ..., Yn)	Returns the admittance of the parallel connection of an arbitrary number of admittances.
Z_to_Y(Z)	Returns an admittance $Y = 1/Z$
Y_to_Z(Y)	Returns an impedance $Z = 1/Y$

6. System Description and Property Procedures:

Procedure	**Function**
`A_matrix(sys)`	Returns the system A matrix
`B_matrix(sys)`	Returns the system B matrix
`C_matrix(sys)`	Returns the system C matrix
`D_matrix(sys)`	Returns the system D matrix
`E_matrix(sys)`	Returns the system E matrix
`F_matrix(sys)`	Returns the system F matrix
`System_order(sys)`	Returns the system order
`State_variables(sys)`	Returns the vector of state-variable names
`System_Inputs(sys)`	Returns a vector of the system input variable names.
`System_Outputs(sys)`	Returns a vector of the system output variable names.
`State_equations(sys)`	Displays the system state and output equations in matrix form.
`Differential_equation(sys{,inp,out})`	Displays the differential equation (in differential (S) operator notation) relating a single output to a single input.
`Transfer_function(sys{,inp,out}{,var})`	Returns the transfer function relating a single output to a single input.
`Transfer_function_matrix(sys{,var})`	Returns the transfer function matrix relating all outputs to all inputs.
`System_char_poly(sys{,var})`	Returns the system characteristic polynomial.
`System_eigenvalues(sys)`	Returns the system eigenvalues.
`System_poles(sys)`	Returns the system poles (same as the system eigenvalues).
`System_zeros(sys{,inp,out})`	Returns the zeros of the transfer function between a given input and output.
`Pole_zero_plot(sys{,inp,out})`	Displays the pole-zero plot for the transfer function between a given input and output.
`System_type(sys)`	Returns the system "type", that is the number of free integrators (poles at the origin of the s-plane).
`Root_locus(sys,param,prange,nsteps {,xrange}{,yrange})`	Displays a generalized root-locus for the variation of any system parameter.
`Controllability_matrix(sys)`	Returns the system controllability matrix.
`Controllability(sys)`	Returns the rank of the system controllability matrix.
`Controllable(sys)`	Returns *true* if system is controllable, *false* otherwise.
`Observability_matrix(sys)`	Returns the system observability matrix.
`Observability(sys)`	Returns the rank of the system observability matrix.
`Observable(sys)`	Returns *true* if system is observable, *false* otherwise.
`Position_error_constant(sys)`	Returns the position error constant, $\lim(s \to 0)\{G(s)\}$.
`Velocity_error_constant(sys)`	Returns the velocity error constant, $\lim(s \to 0)\{sG(s)\}$.
`Accel_error_constant(sys)`	Returns the acceleration error constant, $\lim(s \to 0)\left\{s^2G(s)\right\}$.
`Time_constant(sys)`	Returns the time constantτ (s) of a first order system.
`Natural_frequency(sys)`	Returns the undamped natural frequency ω_n (rad/s) of a second order system.
`Damping_ratio(sys)`	Returns the damping ratio ζ of a second order system
`System_exponential(sys,t)`	Compute the matrix exponential $e^{\mathbf{A}t}$ for the system.

7. Time Domain Response Procedures:

Procedure	Function
`Step_response(sys{,inp,out})`	Returns the system unit step response function between a given input and output.
`Impulse_response(sys{,inp,out})`	Returns the system impulse response function between a given input and output.
`Ramp_response(sys{,inp,out})`	Returns the system ramp response function between a given input and output.
`System_response(sys,function{,inp,out})`	Returns the response of a system to an arbitrary input function.
`Final_value(sys,stepsize{,inp,out})`	Returns the final value of the response to a step of size `stepsize`.
`Output_IC_response(sys,ic)`	Returns the response of a SISO system output to a given set of initial conditions on the system output variable.
`State_IC_response(sys,ic{,out})`	Returns the response of a system output to a given set of initial conditions on the system states.

8. Control Systems Procedures::

Procedure	Function
`Closed_loop_system(plant,ctrl{,feedback}{,inp,out})`	Returns a system representing the closed-loop SISO system with output feedback, and the given controller and sensor dynamics.
`Open_loop_system(plant,ctrl{,feedback}{,inp,out})`	Returns a system representing an open-loop SISO system with output feedback, and controller and sensor dynamics.
`Phase_margin(sys)`	Returns the phase margin (deg) of an open-loop SISO system.
`Gain_margin(sys)`	Returns the gain margin (dB) of an open-loop SISO system.
`State_feedback(plant,gain_matrix)`	Generates a description for the system formed with full state feedback as specified by the gain matrix.
`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 polynomial char_poly.

9. Frequency Domain Response Procedures:

Procedure	**Function**
Frequency_response_matrix(sys{,var})	Returns the frequency response matrix relating all outputs to all inputs.
Frequency_response(sys{,inp,out}{,var})	Returns an ndividual frequency response function relating a single output to a single input.
Frequency_response_magnitude(sys{,inp,out}{,var})	Returns the magnitude function relating a single output to a single input. The value of the magnitude function at a particular frequency can be found by substituting a numerical value for 'var'.
Frequency_response_phase(sys{,inp,out}{,var})	Returns the phase function relating a single output to a single input. The phase response at a particular frequency may be found by substituting a numerical value for 'var'.
Find_magnitude(sys, mag{,inp,out}{,frange})	Returns the frequencies (rad/s) at which the frequency response has the given magnitude 'mag'.
Find_phase(sys, phase{,inp,out}{,frange})	Returns the frequencies (rad/s) at which the frequency response has the given phase 'phase'.
Bode_magnitude(sys, frange{,inp,out})	Displays a Bode magnitude plot (in decibels) of the transfer function between a given input and output.
Bode_phase(sys, frange{,inp,out})	Displays a Bode phase plot (in degrees) of the transfer function between a given input and output.
Polar_plot(sys, frange{,inp,out}{,npoints})	Displays the polar form of the frequency response function
LogMag_phase(sys, frange{,inp,out}{,npoints})	Displays the log-magnitude (dB) versus phase (deg) form of the frequency response function.

10. Utility Procedures:

Procedure	**Function**
`Version()`	Displays the version of SYREP
`Display_digits(n)`	Sets the display precision returned by SYREP procedures.