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 I

Program Description

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

Copyright ©2002 D. Rowell

Chapter 1

Introduction

1.1 Introduction

MAPLE-SYREP (SYstem REPresentation) is a package of Maple V procedures for the generation of the dynamic equations, and analysis of the dynamics of linear lumped-parameter systems. It is a closed-form, symbolic package that allows the specification of a system in several forms, including the linear graph modeling method (Rowell and Wormley, *System Dynamics, An Introduction*, Prentice Hall, 1997), impedance based methods, specification of the transfer function, and direct state-space matrix entry. An extensive set of procedures is provided for the manipulation and analysis of systems in state-space and classical input-output representations. These include system properties, time and frequency domain response analysis, and closed-loop control system design.

MAPLE-SYREP is a symbolic processor. Unlike numerical analysis packages, such as MATLAB, it performs all analyses in closed-form, and produces its results as *functions* of the supplied parameters. For example, if a first-order electrical system is specified in terms of a resistance R and a capacitance C, the step response will be computed in symbolic form and displayed as a function of R and C, such as $(1 - e^{-t/RC})$. If numeric values are assigned to the parameters, the closed form solution may be plotted as a function of time t.

It is a simple matter to perform parameter variation studies using SYREP. For example, the damping ratio of a second-order system may be plotted as a function of one of the system parameters with a one line command. Assume that a mechanical system has a transfer function

$$H(s) = \frac{K}{ms^2 + Bs + K}$$

The system may be entered into SYREP, and the effect of B on the damping ratio studied with the commands:

```
sys := TF_to_system(K/(m*s^2 + B*s + K), s):
Damping_ratio(sys);
m:=10: K:=50:
plot(Damping_ratio(sys), B=0..100);
```

The SYREP procedure `TF_to_system()` takes the given transfer function and converts it to a SYREP system structure. The second line computes and displays the damping ratio as $0.5B/\sqrt{Km}$. After assignment of numerical values to m and K, the SYREP function `Damping_ratio` returns an expression for the system's damping ratio as a function of B; the Maple `plot` function plots the expression with B varying from 0 to 100.

A numerical value may be assigned to B and the step response computed:

```
B:=5:
y_step := Step_response(sys);
```

and Maple SYREP reports that the step response is

$$y\_step = 1. - 1.e^{-.250t}\cos(2.22t) - .113e^{-.250t}\sin(2.22t)$$

Numerical packages use approximate integration techniques to compute the response of a system at a set of discrete times. They provide no insight into the components of a response waveform, or the influence of parameters upon the system performance. The system must be specified in numerical form, with full prior substitution for the system parameters.

The two approaches are complementary. Numerical methods allow for the handling of some classes of nonlinear systems; the methods used in MAPLE-SYREP are based on linear algebra, and are therefore limited to linear systems.

1.1.1 Caveat:

The current version of SYREP is experimental. Additions are being made and, as in any developmental software package of this complexity, it cannot be guaranteed to be completely free of bugs. Maple SYREP is offered without warranty of any kind, on the understanding that while it appears to be operating as designed, there is a distinct possibility that bugs continue to lurk beneath the surface.

1.1.2 Running SYREP:

Because MAPLE-SYREP is written in the Maple language, it is machine independent and runs on any computer under Maple V Release 4.00b or later (see below). It has been tested on PC's under Windows 3.1, Windows 95 and Windows NT, on Macintoshes, and on Unix workstations. It can be run on a graphical (windowed) system or on a text based terminal session, for example by Telnet to a remote computer. The output is much more attractive on a graphics system, and the Maple "worksheet" concept only applies to that mode.

A word of caution: the initial releases of Maple V Release 4 on PC's and Macintoshes had some significant bugs that prevented SYREP from running. You must be running at least Version 4.00b for SYREP to work. Earlier versions may be upgraded using an upgrade "patch" form Waterloo Maple Inc. The patch may be downloaded directly from their ftp site `ftp.maplesoft.com`, or their web site `www.maplesoft.com`.

1.1.3 Obtaining Your Own Copy of SYREP

SYREP exists as a single machine-independent file. To obtain your own copy you must download it from the web site http://web.mit.edu/drowell/www/syrep/.

1.1.4 Loading SYREP

SYREP is distributed as a Maple ".m" file which must be loaded into Maple with a "read" command. The procedure for loading SYREP is therefore to start Maple, and then to issue the command:

```
read ‘directory_path/syrep.m‘;
```

or alternatively in Maple V Release 5 as

```
read "directory_path/syrep.m";
```

where *directory_path* is the specification of where you have stored the file syrep.m. For example, on a PC the line might read

```
read ‘c:/work/maple/syrep.m‘;
```

(Note the use of back-quotes (‘) to enclose the path string, the semi-colon (;) at the end of the line to force execution of the command, and that Maple *requires* the use of forward slashes (/) in the directory specification on a DOS system.)

1.1.5 Notes on Using Maple V

1. Maple commands are generally executed as they are entered, whenever the Enter (Return) key is hit. As you enter commands they are entered into an "execution block". To enter multiple commands into a block, use the Shift-Enter combination between lines. The entire block may be executed by placing the text cursor anywhere in the block and hitting Enter. The pull-down menus (Edit and Insert) have options for manipulating execution blocks.

2. Spaces are ignored in input lines. Commands lines are ended with a semicolon or colon. Long lines may be continued on a new line (use the Shift-Enter combination to continue a line).

3. A line starting with # is ignored (is treated as a comment), similarly any text appearing after a # on a line is ignored.

4. A Maple command is not recognized until a terminating character, either a semi-colon (;) or colon (:). If the terminator is a semi-colon, the results of the command are displayed; if the command ends with a colon (:) the operation is "silent" and no output is displayed.

5. The Maple assignment is done through the := combination, for example

   ```
   Z := 3*x - 2*y;
   ```

   assigns the result of the expression $3x - 2y$ to the variable Z.

6. Maple has a number of reserved words that may impact the use of certain names. For example the name D is reserved, and unavailable for use as a matrix name in this software.

7. Maple text input is case-sensitive. Thus the names `vm` and `Vm` represent different quantities to Maple.

8. Greek letters will be displayed as a Greek symbol if they are complete and independent. Thus the name "omega" will appear on the screen as ω but "omega_n" will appear untranslated.

9. Plotting functions can only work if all system parameters except the independent variable have been assigned numerical values. For example, it is not possible to plot a step response, or invoke the SYREP function `Pole_zero_plot()` until the system parameters have been given numerical values.

1.1.6 Floating Point Precision

Maple performs floating point computations to arbitrary precision, as determined by the value of its global variable `Digits`. The default value is `Digits := 10` but this can be changed by including a statement such as

```
Digits := 15:
```

in the Maple SYREP session.

Maple results are also displayed in the chosen precision. This can make the output difficult to read; for this reason SYREP provides a procedure that will control the displayed precision of results from its procedures. The command

```
Display_digits(4):
```

will allow floating point calculations to be done internally at the precision specified by the value of `Digits` but will display only four digits in the values returned by SYREP functions.

1.1.7 Starting A Maple SYREP Session

The following shows a typical initialization of a SYREP session:

```
restart:
read `c:/mysyrepdirectory/syrep.m`:
Display_digits(3):
```

The `restart` command initializes Maple and resets all variables. It is often useful to issues a `restart` command during a session to reset Maple. Note the use of the forward slashes (/) as separators in the `read` command.

Chapter 2

Specifying the System to SYREP

2.1 Introduction

The first step in using SYREP is to specify the structure of the system to be studied. There are currently five primary methods of specifying the system structure to SYREP; each generates a SYREP system *object*, containing a mathematical representation of the system as a set of symbolic state equations

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{x} + (\mathbf{E}\dot{\mathbf{u}})$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + (\mathbf{F}\dot{\mathbf{u}})$$

where $\mathbf{x}$ is the state vector, $\mathbf{y}$ is a vector of user specified outputs, and $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}$ are constant matrices containing system parameters. (The matrices $\mathbf{E}$ and $\mathbf{F}$ are non-zero only when a system contains dependent energy storage elements, or has an improper transfer function.) In addition the system object contains descriptive information such as variable names, input and output variables and system order. All SYREP analysis functions extract system information from this common object structure.

The five methods for system definition are:

- Direct specification of the system to SYREP from a set of $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}$ matrices.
- Specification of the *transfer function* of a SISO system as a rational function in the Laplace variable.
- Specification of the structure and elements from a *linear graph* representation of a system. The system *object* is created automatically from the graph topology.
- Specification of the structure and relationships from an *impedance graph* of a system. The system *object* is created automatically from the graph topology.
- Specification of the system elements by their elemental equations, together with a set of independent constraint equations. (This method is obsolete, and is retained for tutorial reasons.)

and are implemented through the procedures defined in Table 2.1.

2.2 System Specification Directly from System Matrices

The first method of defining a system is to specify the dynamic model directly in terms of a set of state-space system matrices:

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

by the SYREP procedure:

```
system_name := Matrices_to_system(A, B, {C{, D{, E{, F}}}}):
```

Procedure	**Function**
`Matrices_to_system(A,B{,C{,D{,E{,F}}}})`	Generate a system description directly from supplied state-space matrices.
`TF_to_system(transfer_function{,var}{,inp,out})`	Generate a system description directly from a transfer 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.

Table 2.1: SYREP functions for system specification.

where $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}$ are Maple matrices. The matrices $\mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}$ are optional. The system matrix $\mathbf{A}$ must be square, and will define the system order n. The input matrix $\mathbf{B}$ must have n rows and r columns, where r is the number of inputs. If the matrices $\mathbf{C}$ and $\mathbf{D}$ are omitted, it is assumed that the output vector will be equal to the vector of state variables, that is a diagonal (identity) $\mathbf{C}$ matrix is created internally and $\mathbf{D} = \mathbf{0}$. The matrices $\mathbf{E}$ and $\mathbf{F}$ are assumed to be zero unless specified. The elements of the matrices may be symbolic or numeric in nature. Notice that the matrix arguments are expected in order, that is to include a specification of an $\mathbf{E}$ matrix both $\mathbf{C}$ and $\mathbf{D}$ must be specified.

There are several ways to specify a matrix to Maple, all using variations of the `matrix()` procedure:

`matrix(`L`)`
`matrix(`m, n, f`)`
`matrix(`m, n, lv`)`

where L is a list of lists (or vectors) [[...],..., [...]] of elements defining the rows, m, n are positive integers (row and column dimensions), f is a function (possibly a constant), used to create the entries $f(i, j)$, and lv is a list (or vector of the elements). For example:

```
matrix([[1, 2 ],[3, 4]]);
```
$\rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

```
matrix(2,2, [a, b, c, d]);
```
$\rightarrow \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

```
matrix(2,2,0);
```
$\rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

```
matrix(2, 3, [1, 2, 3, 4, 5, 6]);
```
$\rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

The following shows how these methods might be used to specify a system to SYREP:

```
A := matrix([[0, 1],[-2, -3]]); B := matrix(2,1,[0, 1]);
C := matrix(1,2,[0, 1]); D := matrix(1,1,0);
Matrices_to_system(A, B, C, D);
```

or

```
A := matrix([[-B/J, 1/(Ka*J)],[-1(Ka*L), -R/L]]); B := matrix(2,1,[0, 1/L]);
C := matrix(3,2,[-B,1/Ka,0,1,B, 0]); D := matrix(3,1,[0,0,0]);
Matrices_to_system(A, B, C, D);
```

In this method the system variables are labeled internally; inputs are named $u\_1, u\_2, \ldots u\_r$, outputs are labeled $y\_1, y\_2, \ldots y\_m$, and the state variables are named $x\_1, x\_2, \ldots x\_n$.

2.3 System Specification by Transfer Function:

The second method for specifying a single-input single-output system to MAPLE-SYREP, is through its transfer function using the procedure:

system_name := TF_to_system(transfer_function, var {, input, output }):

where the `transfer_function` is written as the ratio of a pair of univariate polynomials in the variable `var`, and `input` and `output` are optional names of the input and output variables. If no names are specified, SYREP will assign the names u_1 and y_1 to the input and output respectively. Note that although the variable names are optional, either both names or neither must be entered.

For example, a hydraulic system for which the transfer function relating pressure P_1 to input flow Q_{in} is:

$$H(s) = \frac{P_1(s)}{Q_{in}(s)} = \frac{5s+7}{3s^3+2s^2+4s+1}$$

might be specified:

```
fluid_sys := TF_to_system((5*s + 7)/(3*s^3 + 2*s^2 + 4*s +1),s, Qin, P1);
```

where the input variable is `Qin`, and the output variable is `P1`, or

```
fluid_sys := TF_to_system((5*s + 7)/(3*s^3 + 2*s^2 + 4*s +1),s);
```

if the default names are to be used. It is not necessary for the numerator and denominator polynomials to be fully expanded, for example the following specification in which the transfer function is factored into first- and second-order terms is valid:

```
fluid_sys := TF_to_system(((s+3)*(2*s+1))/(s*(s^2 + 2*s +10)*(s+5)),s);
```

The transfer function variable may be any variable, but must not have been assigned to any value (that is it must be a Maple "indeterminate"). The numerator and denominator polynomials may be written with the terms in any order, and the coefficients may be symbolic or numerical. The transfer function must be a proper fraction, that is the order of the numerator must be less than or equal to the order of the denominator. MAPLE-SYREP converts the transfer function description to a state-space formulation in a *companion* or *phase-variable* form. Internally the state variables are given the set of names $x\_1, x\_2, \ldots, x\_n$ where n is the order of the denominator polynomial.

2.4 Linear Graph Based System Specification:

The third form of system specification, using the procedure `LGraph_to_system()`, is based on the structure of a physical system. In this method the user defines the elements and interconnections to be used in a linear lumped-parameter physical model, and SYREP automatically generates a set of symbolic state-equations describing the system dynamics from the system topology.

The linear graph method of system representation can be applied to a wide range of energy domains, including mechanical translational and rotational, electrical, fluidic, and thermal systems, and energy transduction between domains (Rowell and Wormley, *System Dynamics, An Introduction*, Prentice Hall, 1997). In this representation a pair of variables, the *across-variable*, and the *through-variable*, is defined for each energy domain. In general, a set of spatially distinct points are identified in the physical system and are drawn as nodes on the graph. The lumped-parameter elements are then represented as branches joining the nodes on the graph. Through-variables are defined as variables that sum to zero at nodes (for example forces in mechanical systems (d'Alembert's principle), and currents in electrical systems (Kirchoff's current law).) Across-variables are defined as variables that sum to zero around loops in the graph (for example velocities in mechanical systems, and voltage drops in electrical systems. In this form the system is represented as a line graph in which *nodes* represent points of distinct across-variables, and *branches* (edges) of the graph represent elemental relationships between the across- and through-variable associated with each element.

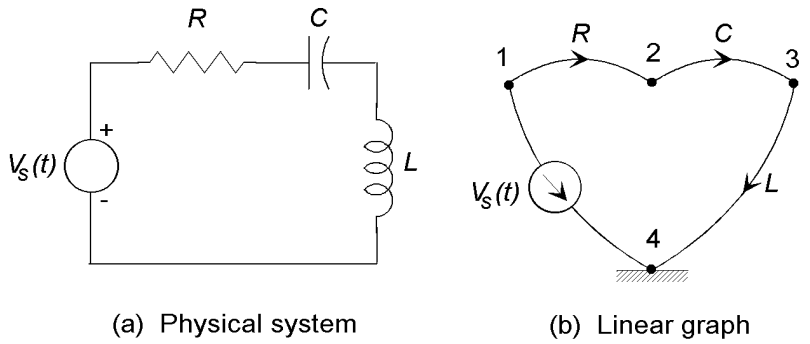


Figure 2.1: A series RLC circuit and its linear graph.

Perhaps the easiest way to understand the linear graph is to draw an analogy with the circuit diagram for electrical circuits. Figure 2.1 shows a simple series RLC circuit and its linear graph. If the electrical circuit assumes ideal components there will be a strong similarity in appearance between the linear graph and the circuit diagram. The nodes on the graph will represent the points of interconnection of components in the circuit.

In the electrical systems the across-variable is defined to be voltage, the through-variable is current. Nodes on the linear graph represent points of distinct voltage in the circuit. The interconnection between nodes is represented by the branches of the graph, and will generally correspond to the circuit elements of the circuit diagram. The difference is that in the circuit diagram the symbols represent actual components; in the linear graph the branches represent mathematical relationships between the across- and through variables.

In each energy domain a set of one or two idealized *energy storage elements* is defined; these are classified as A-type if the stored energy is a function of the across-variable, and T-type if the stored energy is a function of the through-variable. In the electrical domain the A-type storage element is the capacitance C ($E = 1/2Cv^2$) and the T-type element is the inductance L ($E = 1/2Li^2$). In addition, each energy domain has a dissipative element with an algebraic relationship between through- and across variables; the resistance R in the electrical case ($v = iR$). Two idealized source elements an *across-variable source* (voltage) and *through-variable source* (current) are also defined.

The arrow drawn on each branch indicates two assumptions related to power flow into the element:

(a) the direction of assumed across-variable *drop* (voltage drop in the electrical case), and simultaneously

(b) the direction of assumed through-variable *flow* (current direction in the electrical case.)

For sources the arrow direction indicates the assumed across-variable drop for across-variable sources, and the flow direction for through-variable sources.

In addition to the *one-port* elements defined for each energy domain, energy transduction within and between domains is described by *transforming* and *gyrating* two-port elements. A transformer has a direct algebraic relationship between the across-variables (or through-variables) in the two domains and includes elements such as levers, rack and pinion drives, dc electric motors, etc. A gyrator has an algebraic relationship between an across-variable in one domain and the through-variable in the other domain, for example a hydraulic piston/cylinder in which the force F (through-variable) is a function of the fluid pressure P (across-variable) $F = PA$ where A is the piston area.

The linear graph implicitly represents the system structure, the elemental relationships relating the through- and across-variables associated with each branch, and a set of independent loop and node constraint equations between the system variables. In the electrical case these correspond directly with a set of nodal equations (Kirchoff's Current Law), and loop equations (Kirchoff's Voltage Law).

Refer to Rowell and Wormley, Chapters 1–6, for a detailed description of system modeling using the linear graph method.

The steps required to define a system in Maple SYREP are:

1. Prepare a lumped-parameter linear graph representation of the system to be analyzed.

2. Number the nodes on the graph.

3. Define a set of output variables of interest.

4. Prepare a set of Maple lists defining the system structure and output variables.

The system generation procedure `LGraph_to_system()` generates the necessary elemental relationships and constraint equations automatically, and is invoked by a statement such as:

```
system := LGraph_to_system(graph, outputs, verbose);
```

where `graph` is the Maple list (enclosed in square brackets [ ]) defining the structure of the linear graph, `outputs` is a list of output expressions, and `verbose` is an optional argument that controls the printing of intermediate results as the equation generation proceeds.

2.4.1 The Linear Graph Specification:

The structure of the linear graph is specified by elemental connections between the numbered nodes. The data is entered as a list of sub-lists, one for each system element defining the element type and its nodal connections. For example, the electrical RLC circuit of Fig. 2.1 is specified by the list:
`[[1,4,voltage,Vs], [2,3,capacitance,C1,vC1,iC1],[3,4,inductance,vL,iL],[1,2,resistance,R,vR,iR]]`
In this case the graph list contains 4 sub-lists, each of which specifies an elemental connection between a pair of numbered nodes on the graph.

One-port elements

For one-port elements the sub-list specification is of the form:

`[tail_node, head_node, element_type, parameter_value{,across-variable, through-variable}]`

where `tail_node` and `head_node` are the numbers assigned to the tail and head nodes of the directed edge (branch) of the graph representing the element. `Element_type` is one of the element definition names known to SYREP (discussed below), `parameter_value` is the symbolic name or value assigned to the element, and `across-variable` and `through-variable` are optional names to be given to the across- and through-variables associated with the element.

In the case of sources, the specification is

`[tail_node, head_node, source_type, source_name]`

The first entry in the above example indicates that a voltage source, named `Vs` is connected with its arrow (voltage drop) from node 2 to node 1. SYREP determines the source type (across- or through-variable) through the name specified for `source_type`; `voltage` in this case.

Consider the mechanical system and its linear graph with the nodes numbered as shown in Fig. 2.2. The specification list for this system may be defined and assigned to the Maple variable `mechanical` as follows:

```
mechanical :=  [[4,3,force,Fs],
                      [3,4,mass,m2,vm2,Fm2],
                      [2,3,damper,B2],
                      [1,2,spring,K2,vK2,FK2],
                      [1,4,damper,B1],
                      [1,4,spring,K1,vK1,FK1],
                      [1,4,mass,m1,vm1,Fm1]]:
```

(Note that the statement may be spread over several lines because Maple does not recognize a statement until it encounters a colon(:) or semi-colon (;) terminator.)

The elements (edges) may be specified in any order, but the arrow directions on each graph edge should be specified by correctly ordering the nodes in each sub-list. In the above example some of the optional variable names were specified, for example on the masses m_1 and m_2 and the springs K_1 and K_2, while on the dampers no names were given. If no names are specified, SYREP assigns a pair of internal names for each branch using the convention `f_`n for the through-variable and `v_`n for the across variable, where n is

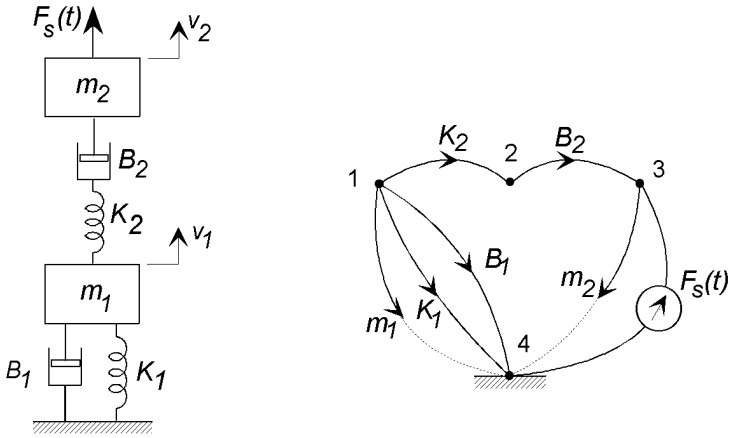


Figure 2.2: A mechanical system and its linear graph.

the element number as defined from its position in the graph list. Thus in the example the velocity and force associated with dampers $B1$ and $B2$ will be named `v_5` and `f_5` for $B1$ and `v_3` and `f_3` for $B2$. Users should avoid the use of names of this type to prevent possible conflicts. Notice also that either *both* names or *no* names must be given; it is not permissible to specify just one name.

SYREP maintains a table of known element types and automatically produces the elemental equations for these elements. The parameters associated with each named type is the commonly used physical parameter, for example the parameter associated with the type `spring` is its stiffness K, leading to the elemental relationship $\dot{F}_K = Kv_K$, whereas if the same element is specified as a `compliance` with parameter C, the elemental equation would be $\dot{F}_C = (1/C)v_C$.

SYREP names are case-insensitive, so that `Resistance` and `resistance` are equivalent. Furthermore, only the first six letters are recognized, so that `capacitor` and `capacitance` are equivalent. Tables 2.2.1 and 2.3 summarize the names recognized by SYREP for sources and one-port passive elements. In many cases the names are synonyms and are included for energy-domain clarity.

Two-port Elements

The two-port energy transducing elements (transformers and gyrators) are represented by a definition sublist with a slightly different form. Assume that the two graph branches in the two-port are specified as `a` and `b` as shown in Fig. 2.3. The element is defined by the list

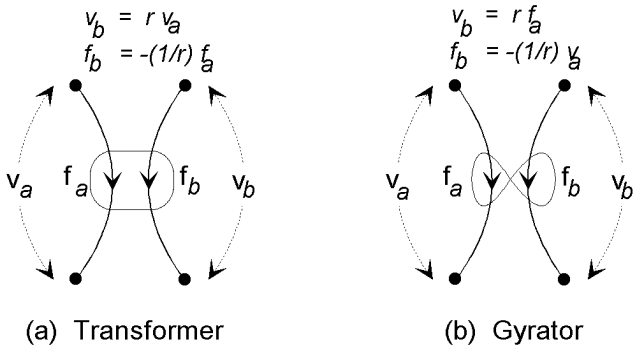


Figure 2.3: Linear graph representation of energy transducing two-port elements.

```
[[tail_a, head_a],[tail_b,head_b], name, ratio,
       {[a_var_a,t_var_a],[a_var_b,t_var_b]}]
```

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

Table 2.2: SYREP name definitions for idealized sources.

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

Table 2.3: SYREP name definitions for one-port passive elements.

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

Table 2.4: SYREP name definitions for two-port energy transducing elements.

where `tail_a`, `head_a`, `tail_b`, `head_b` are the tails and heads of the directed graph edges representing the two sides of the two-port element, `name` is the SYREP name for the two-port (defined in Table 2.4), `ratio` is the transduction constant *that defines the across-variable* on side `b` of the two-port element. The variables on each branch may be optionally named by including the pair of lists `[a_var_a,t_var_a],[a_var_b,t_var_b]`. When explicit variable names are not given, the default naming convention for two-port branches is to designate the two sides by `a` and `b`, and designate the variables as `va_`n, `fa_`n, `vb_`n, `fb_`n for the across and through variables respectively, where n is the position of the two-port in the element list.

For example, an idealized dc motor might be specified as

```
[[4,5],[6,7],transformer,1/Km,[v_back,i_motor],[Omega,T]]
```

or

```
[[4,5],[6,7],TF,1/Km]
```

indicating that the electrical side is connected between nodes 4 and 5 (with the arrow pointing to node 5), the mechanical side is connected from node 6 to node 7, In the first case the motor back emf will be designated as `v_back`, the motor current as `i_motor`, and the angular velocity and torque as `Omega` and `torque` respectively; in the second case the default names will be assigned. The element definitions `transformer` and `TF` are synonyms. As defined by the parameter `1/Km` the power conserving elemental relationships for this motor are

$$\begin{aligned} \Omega &= (1/K_m)v_{back} \\ T &= -K_m i_{motor}. \end{aligned}$$

Similarly the specification for a hydraulic piston (gyrator) might be written

```
[[1,2],[3,4],GY,1/A,[Pressure,Flow_rate],[Velocity,Force]]
```

where A is the surface area of the piston.

2.4.2 System Outputs:

The system outputs may be specified as a single item for a single-output system, or a list of items in a multiple-output system. Individual output quantities may be

(a) a system variable, either named in the element definition list, or the default SYREP name if no name was given, for example `[vm, torque, v_3]`,

(b) an expression in the form of an equation that defines a new variable as a linear combination of scaled system variables, for example

```
[x=(1/K)*FK, v_motor=v_back+vR1+vL1, y=v_3 + v_5]
```

(c) an expression that is the integral of a linear combination of system variables using the procedure `integral()`:

```
[position=integral(velocity), angle=integral(WJ)]
```

When an expression is used in the variable list, the output variable is named to the variable on the left-hand side of the expression.

When the integral function `integral()` is used, the system state vector is augmented with an extra state variable.

2.4.3 "Verbose" Output Display

The optional argument `verbose` controls the display of intermediate results as the reduction to the state equations proceeds. If absent, the process is "silent" and no output is displayed, if the word `verbose` is present SYREP displays the elemental equations, the normal tree, the number of dependent energy storage elements found, the constraint (compatibility and continuity) equations, and the final state and output equations in matrix form. Figure 2.4 shows the screen output that SYREP generates with the `verbose` option while modeling a dc moving-coil voltmeter. In the definition of the edges in the normal tree the naming convention is `e`n for one-port elements, and `e`n`a` and `e`n`b` for two-port elements, where n is the position of the element in the graph list. See Rowell and Wormley for definitions of the quantities displayed with the `verbose` output.

2.4.4 Further Examples

Figure 2.5 shows three simple systems and their specification to SYREP through the linear graph method.

2.5 System Specification by Impedance Graph

The procedure `ZGraph_to_system()` allows a single-input/single-output (SISO) system to be specified from an impedance based linear graph, as described in Chapter 13 of Rowell and Wormley. In this case the branch relationships of the linear graph are specified as algebraic relationships in the Laplace variable s; in terms of the impedance $Z(s)$ or the admittance $Y(s) = 1/Z(s)$ of the branch, as defined by the relationships

$$V(s) = Z(s)F(s) \qquad \text{or} \qquad F(s) = Y(s)V(s)$$

where $V(s)$ and $F(s)$ are the Laplace transforms of the across-variable and through-variable associated with the branch. For example, a mass element's behavior may be written as $smv_m(s) = F_m(s)$, or as $v_{m(s)} = (1/sm)F_m(s)$, defining the branch impedance $Z_m(s) = 1/sm$ and the admittance $Y_m(s) = sm$.

Preparation for the use of `ZGraph_to_system()` is similar to `LGraph_to_system()`. A lumped parameter model is prepared and expressed as a linear graph. The nodes of the graph are numbered, and the elemental relationships are specified in impedance form in a list similar to that of `LGraph_to_system()`, for example The list correponding to the series RLC circircuit of Fig. 2.1 is

```
[[1,4,voltage,Vs],[1,2,Z,R],[2,3,Y,s*C,vC,iC],[3,4,Z,s*L,vL,iL]]
```

where the list contains a sequence of sub-lists, each specifying an impedance or admittance relationship for a branch.

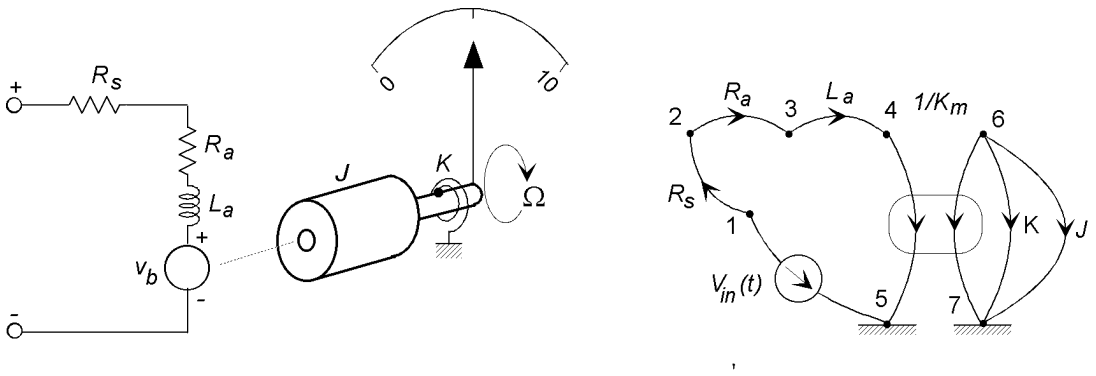
For passive one-port elements the sub-list has the form

```
[tail_node, head_node, type, expression{,across-variable, through-variable}]
```

where `tail_node` and `head_node` are the numbers assigned to the tail and head nodes of the directed edge (branch) of the graph representing the element, `type` is either `Z` if the element is represented as an impedance or `Y` if it is an admittance as specified in Table 2.5, `expression` is an expression of the impedance or admittance of the element and `across-variable` and `through-variable` are optional names to be given to the across- and through-variables associated with the element. As in the case of `LGraph_to_system()` if no names are specified, SYREP assigns names of the form `v_`n and `f_`n to the across- and through-variables on the nth element in the list.

The impedance relationships may be either

- Simple elemental impedance relationships for a lumped parameter element, such as `1/(s*m)` or `s*L`.




```
 graph := [[1,5,Voltage,Vs], [1,2,resistor,Rs],
          [2,3,resistor,Ra],[3,4,inductance,La],
          [[4,5,[6,7,],TF,1/Km], [6,7,spring,K],
          [6,7,inertia,J,WJ,TJ]]:
meter := LGraph_to_system(graph,[WJ,angle=integral(WJ)],verbose):
```

Normal tree branches: .{e5a, e2, e7, e3, e1}

Normal tree links: , {e6, e5b, e4}

Number of independent energy storage elements: 3

Number of dependent energy storage elements: 0

Elemental equations: , $\left[va\_5 = Km\, vb\_5,\ v\_2 = Ra\, f\_2,\ dot\_WJ = \frac{TJ}{J},\ v\_3 = Rs\, f\_3,\ dot\_f\_7 = K\, v\_7,\ fb\_5 = -Km\, fa\_5,\ dot\_f\_4 = \frac{v\_4}{La}\right]$

Constraint equations: , $[v\_7 - WJ = 0,\ WJ - vb\_5 = 0,\ va\_5 + v\_4 + v\_3 + v\_2 - Vs = 0,\ -fa\_5 + f\_4 = 0,\ -f\_2 + f\_4 = 0,\ -TJ - f\_7 - fb\_5 = 0,\ -f\_3 + f\_4 = 0]$

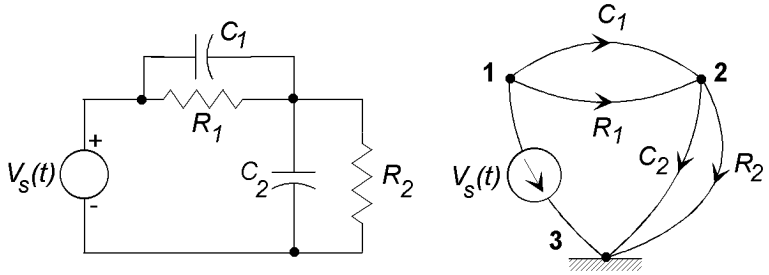
State equations:

$$\frac{\partial}{\partial t}\begin{bmatrix} WJ \\ f\_7 \\ f\_4 \\ xint\_2 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1.}{J} & \frac{Km}{J} & 0 \\ K & 0 & 0 & 0 \\ -1.\frac{Km}{La} & 0 & -1.\frac{Ra+Rs}{La} & 0 \\ 1. & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} WJ \\ f\_7 \\ f\_4 \\ xint\_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{La} \\ 0 \end{bmatrix}[Vs]$$

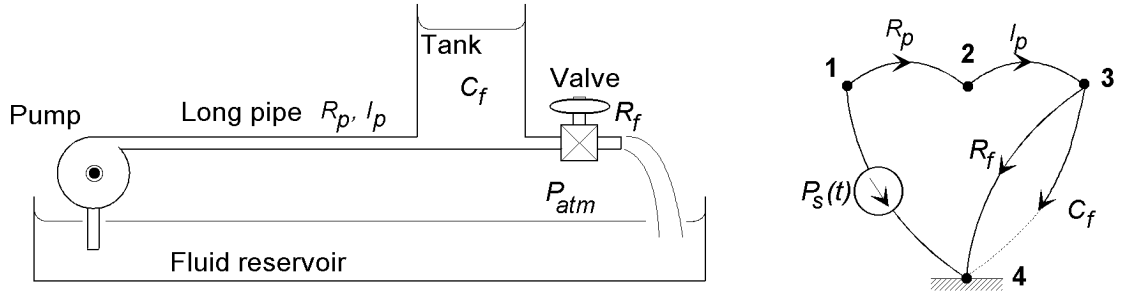
Output equations:

$$\begin{bmatrix} WJ \\ angle \end{bmatrix} = \begin{bmatrix} 1. & 0 & 0 & 0 \\ 0 & 0 & 0 & 1. \end{bmatrix}\begin{bmatrix} WJ \\ f\_7 \\ f\_4 \\ xint\_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}[Vs]$$

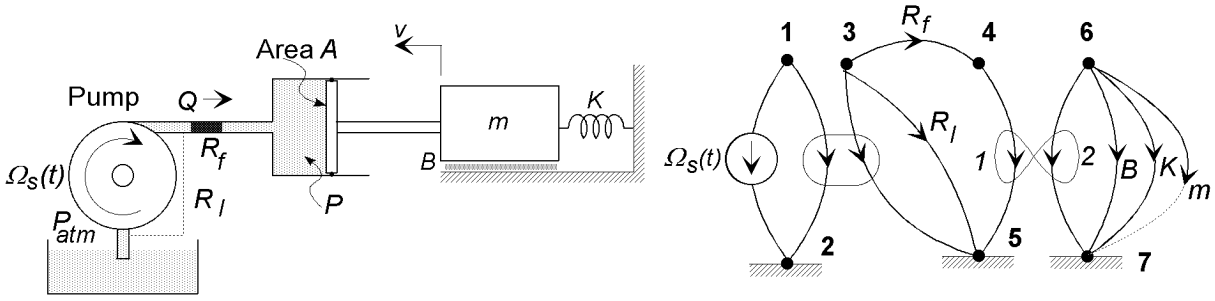
Figure 2.4: Example of SYREP output generated using the `verbose` argument.




```
circuit:=[[1,3,Voltage,Vs],
          [1,2,capacitance,C1,vC1,iC1],
          [1,2,resistance,R1,vR1,iR1],
          [2,3,capacitance,C2,vC2,iC2],
          [2,3,resistance,R2,vR2,iR2]];
lead_lag:=LGraph_to_system(circuit,[vC1,vC2],verbose):
```




```
lgraph := [[1,4,pressure,Ps],
           [1,2,resistance,Rp],
           [2,3,inertance,Ip,pIp,qIp],
           [3,4,resistance, Rf],
           [3,4,capacitance,Cf,pCf,qCf]:
LGraph_to_system(lgraph,[pCf,volume=integral(qCf)],verbose):
```




```
structure:= [[1,2,Pressure,Ps],
             [[1,2],[3,5],TF,Kp],
             [3,5,resistance,Rl],
             [3,4 ,resistance,Rf],
             [[4,5],[6,7],gyrator,1/A],
             [6,7,damper,B],
             [6,7,spring,K,vK,FK],
             [6,7,mass,m,vm,Fm]]:
out:= [pos=integral(vm), FK]:
hydraulic:=LGraph_to_system(structure,out):
```

Figure 2.5: Three systems and their SYREP specification using `LGraph_to_system()`.

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

Table 2.5: SYREP definitions impedance based graphs.

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 $Z = 1/Y$

Table 2.6: SYREP functions for impedance based graph simplification.

- A compound impedance formed by series and parallel combinations of lumped elements, for example a series combination of resistance and inductance may be specified as a single branch in the linear graph and specified in a single equation `R+s*L`. Similarly a parallel R-C combination may be specified as a single branch with impedance `R/(s*R*C+1)`.

To aid in the reduction of impedance based linear graphs, SYREP provides four procedures for manipulating series and parallel combination of impedances and admittances, as summarized in Table 2.6. Each function takes an arbitrary number of impedance expressions and returns a single expression for the equivalent combined impedance (or admittance), for example

```
Z1 := Z_series(R, s*L, 1/(s*C)):
Y := Y_parallel(B, s*m, K/s):
Z3 := Z_series(R, Z_parallel(1/(s*C), s*L)):
```

Two-port energy transducers are specified in the same way as in `LGraph_to_system()`, that is

```
 [[tail_a, head_a],[tail_b,head_b], name, ratio,
{[a_var_a,t_var_a],[a_var_b,t_var_b]}]
```

where `tail_a`, `head_a`, `tail_b`, `head_b` are the tails and heads of the directed graph edges representing the two sides of the two-port element, `name` is the SYREP name for the two-port (defined in Table 2.4), ratio is the transduction constant *that defines the across-variable* on side `b` of the two-port. The variables on each branch may be optionally named by including the pair of lists `[a_var_a,t_var_a],[a_var_b,t_var_b]`. The default naming convention is similarly `va_`n, `fa_`n, `vb_`n, `fb_`n for the across-and through variables on brances a and b of the two-port.

The system input is specified in the same manner as for `LGraph_to_system()`, using the naming conventions specified in Table 2.2.1.

2.5.1 Output Specification

`ZGraph_to_system()` allows only a single output variable to be declared. The variable may be

(a) a system variable, either named in an element definition sub-list, or the default SYREP name if no name was given.

(b) an expression that is a linear combination of scaled system variables, for example `x = (1/K)*FK`, `v_motor = v_back + vR1 + vL1`, or `y = v_3 + v_2 - v_5]`.

(c) an expression that is the integral of a linear combination of system variables using the procedure `integral()`, for example `position = integral(velocity)`, or `angle=integral(WJ)`

When an expression is used, the system output variable is named to the variable on the left-hand side of the expression. The output specifcation may be optionally specified as a single item list, so that

`sys := ZGraph_to_system(graph,[v_1]):` or `sys := ZGraph_to_system(graph,v_1):`

are both valid.

2.5.2 "Verbose" Output Display

The optional third argument `verbose` controls the display of intermediate results as the reduction to the state-space system description proceeds. If the argument is present, results will be displayed, if absent the operation is silent.

2.5.3 Internal Representation

`ZGraph_to_system()` generates a set of independent linear equations in the Laplace variable, and solves this set for the output variable. The resulting transfer function is converted to state-space description in a *companion* or *phase-variable* form. Internally the state variables are given the set of names `x_1, x_2, ...,x_N` where N is the order of the denominator polynomial.

2.5.4 An Example

The hydraulic system shown in Fig. 2.6 has a pump, characterized as a Thevenin source with a pressure source P_{in} and a series resistance R_s, connected to a long pipe with lumped inertance I_p and resistance R_p and a vertical walled tank with fluid capacitance C_f. A discharge valve is partially opened and is modeled as a linear resistance R_o. The output variable of interest is the tank pressure.

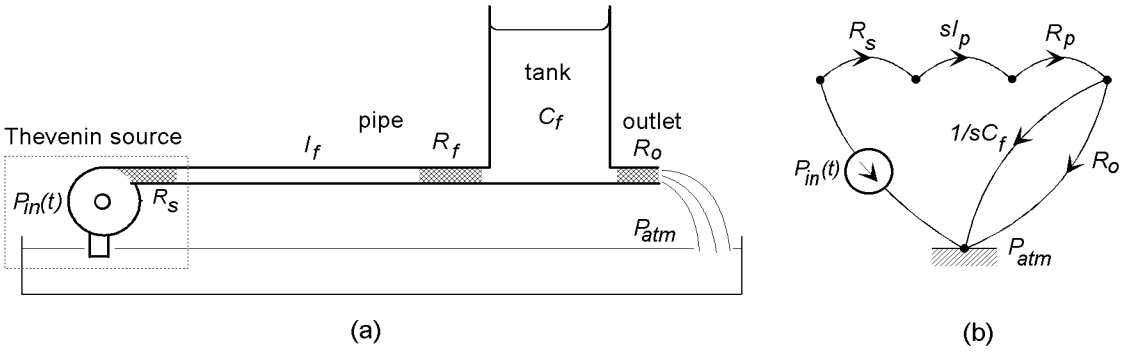


Figure 2.6: A fluid system and its linear graph.

From the linear graph the system may be specified directly:

```
zgraph := [[1,5,Pressure,Pin],
           [1,2,Z,Rs],
           [2,3,Z,s*Ip],
           [3,4,Z,Rp],
           [4,5,Y,s*Cf,PCf,QCf],
           [4,5,Z,Ro]]:
Fluid_system := ZGraph_to_system(zgraph,PCf):
```

Alternatively, the three series elements R_s, R_p, and I_p may be combined into a single impedance

$$Z_1 = R_s + R_p + sI_p$$

and the two parallel elements, C_f and R_o, combined into an equivalent impedance

$$Z_2 = \frac{R_o}{R_o C s + 1}$$

the system may be represented by a reduced linear graph containing just two passive branches, as shown in Fig. 2.7: The reduced system may be represented by just two linear equations. The system, in its reduced

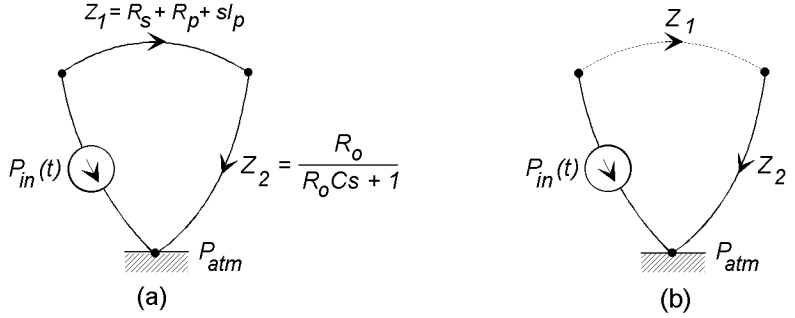


Figure 2.7: A reduced linear graph for the fluid system and its tree

form may be specified to SYREP as:

```
Z1 := Rs + Rp + s*Ip:
Z2 := Z_parallel(Ro,1/(s*Cf)]:
zgraph := [[1,3,Pressure,Pin],
           [1,2,Z,Z1],
           [2,3,Z,Z2,PZ2,QZ2]]:
Fluid_System := Zgraph_to_system(zgraph, PZ2):
```

where the output variable P_{Z_2} is equivalent to P_{C_f}.

2.6 Specification by Elemental and Constraint Equations

The fifth form of linear graph based system specification `Elements_to_system()` requires the user to provide a set of lists containing (i) the elemental equations describing the lumped parameter elements in the system, (ii) a sufficient set of constraint (compatibility and continuity) equations, and a set of output variables. This method has been largely superseded by `Lgraph_to_system`, and is retained primarily for compatibility with earlier versions SYREP. The method is based on the *normal tree* linear graph reduction method described in Chapters 5 and 6 of Rowell and Wormley, *System Dynamics, An Introduction*, Prentice Hall, 1997).

To specify a system to `Elements_to_system()` the following steps should be taken:

(a) Prepare a lumped parameter model of the system and express the model as a linear graph.

(b) Generate a normal tree and use it to define elemental causality, and a set of primary and secondary variables.

(c) Write the set of elemental equations explicitly in terms of the primary variables in the form described below.

(d) Generate the set of continuity and compatibility equations from the normal tree. Each equation should express a single secondary variable in terms of the primary variables and inputs.

The call to `Elements_to_system()` is of the form

```
system := Elements_to_system(elemental,constraints,outputs{,verbose})
```

where `elemental` is a list of elemental equations, `constraints` is a list of constraint equations, and `outputs` is a list of output quantities. For example, a second-order mechanical system might be specified as:

```
el_eqns := [ dot(vm) = (1/m)*Fm, dot(FK) = K*vK, FB = B*vB];
constraints := [ Fm = Fin - FB - FK, vK = vm, vB - vm = 0];
out := [vm, FK, FB, x = integral(vm)];
Mechanical := Elements_to_system(el_eqns, constraints, out, verbose)
```

2.6.1 Elemental Equation Specification

The elemental equations are entered in a list:

```
name := [eqn1, ...  ,eqnN]:
```

where `eqn1` to `eqnN` are the N elemental equations describing the passive elements in the system. Each of the elemental equations must be entered in SYREP with the *primary* variable on the left hand side, in one of three forms

$$\begin{aligned} \texttt{dot(primary)} &= \texttt{coefficient} * \texttt{secondary} \\ \texttt{primary} &= \texttt{coefficient} * \texttt{dot(secondary)} \\ \texttt{primary} &= \texttt{coefficient} * \texttt{secondary} \end{aligned}$$

where the notation `dot(variable)` indicates the time derivative. The order of the items on the right-hand side of the equation is important; SYREP recognizes the variable as the right-hand term.

The use of parentheses around coefficients is optional, but helps readability. Both

$$\texttt{dot(vm)} = \texttt{1/m} * \texttt{Fm} \qquad \text{and} \qquad \texttt{dot(vm)} = \texttt{(1/m)} * \texttt{Fm}$$

are acceptable.

2.6.2 Constraint Equation Specification

The constraint equations are written in a form similar to the elemental equations:

```
name := [eqn1, ...  ,eqnN]:
```

where each equation is a compatibility or continuity equation that contains only one of the *secondary* variables. The equations may be written in free form, and need not be entered in the same order as the elemental equations. The following three examples are equivalent:

`Fm = Fin - FB - FK,` `Fm - Fin + FB + FK = 0,` `Fm - Fin = -FB - FK.`

SYREP recognizes input variables as those variables appearing in the constraint equations that do not appear in the elemental equations.

2.6.3 Output Variables

The output variables are specified as a list

```
name := [out1, out2, ...  , outM]:
```

where the items `out1` to `outM` are either variables or expressions. Each entry may be either

1. A variable associated with one of the passive elements in the system, specified by its name as it appears in the elemental equations, or
2. A linear combination of system variables, written as an expression such as `y = vR1 + 3*vR2` or `Angle = (1/K)* Torque`, or
3. The integral of a variable or a linear combination of variables through the `integral()` function.

An example of an electrical system output specification with three defined outputs might be

```
Elect_out := [v_motor=v_back+vL+vR, vC1, I_in=IR1+IR2, angle=integral(WJ)]:
```

2.6.4 An Example

A sketch of a fixed field D.C. motor drive system, with its system graph model is shown in Fig. 2.8. The motor is represented as a four terminal element, generating two distinct connected graph sections in the system graph ($N_d = 2$), so that there is $B_T = N - N_d = 4$ branches in the normal tree, and two branches in the links. The system normal tree is shown in Fig. 2.8c.

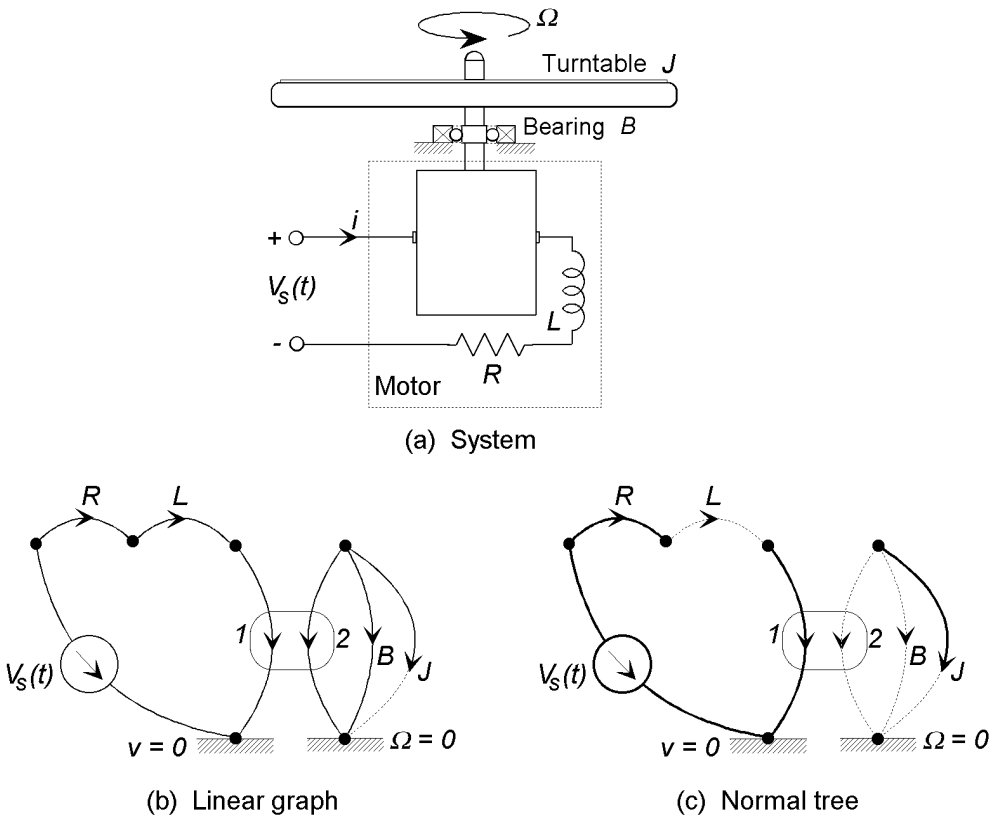


Figure 2.8: An electric motor drive, system graph, and normal tree.

From the normal tree in Fig. 2.8c:

Primary variables:	$V_s(t)$,	Ω_J,	v_R,	i_L,	T_B,	v_1,	T_2
Secondary variables:	$I_s(t)$,	T_J,	i_R,	v_L,	Ω_B,	i_1,	Ω_2
System order:	2						
State variables:	Ω_J,	i_L					

The elemental equations written in terms of primary variables are:

$$\begin{aligned} \frac{d\Omega_J}{dt} &= \frac{1}{J}T_J \\ \frac{di_L}{dt} &= \frac{1}{L}v_L \\ v_R &= Ri_R \\ T_B &= B\Omega_B \\ v_1 &= \frac{1}{K_a}\Omega_2 \\ T_2 &= -\frac{1}{K_a}i_1 \end{aligned}$$

The continuity equations are:

$$\begin{aligned} T_J &= -T_2 - T_B \\ i_R &= i_L \\ i_1 &= i_L. \end{aligned}$$

The compatibility equations are:

$$\begin{aligned} v_L &= V_s - v_R - v_1 \\ \Omega_B &= \Omega_J \\ \Omega_2 &= \Omega_J \end{aligned}$$

The system may be specified to SYREP as follows:

```
elem_eqns      :=  [ dot(WJ)=(1/J)*TJ, dot(iL)=(1/L)*vL, vR=R*iR,
                     TB=B*WB, v1=(1/Ka)*W2, T2=-(1/Ka)*i1 ]:
Constraints    :=  [ TJ=-T2-TB, iR=iL, i1=iL, vL=Vs-vR-v1,
                     WB=Wj, W2=WJ]:
Outputs        :=  [ torque=TB+TJ, WJ]:
motor          :=  Elements_to_system(elem_eqns, Constraints, Outputs)
```

2.6.5 "Verbose" Output Display

If the optional argument `verbose` is included, the system state and output equations will be displayed in matrix form.

2.7 System Generation Procedures for Elementary Systems

In addition to the five methods for specifying complex system structures, SYREP contains a set of procedures for specifying primitive systems that will typically be used in combination with other system objects. The set includes gain `Gain(K)`, differentiator `Differentiator(K)`, integrator `Integrator(K)`, and PID control `PID(Kp,Ki,Kd)` elements. Each procedure takes one or more arguments that defines the gain(s). Table 2.7 defines the blocks and their transfer functions. They are invoked by statements such as:

```
Proportional_controller := gain(10):
PID_control := PID(K,0.1*K,0):
```

Procedure	Transfer Function
`Gain(K)`	K
`Differentiator(K)`	Ks
`Integrator(K)`	$\frac{K}{s}$
`PID(Kp,Ki,Kd)`	$K_p + \frac{K_i}{s} + K_d s$
`Parallel_systems(sys1,sys2{,inp,out})`	$H_1(s) + H_2(s)$
`Cascade_systems(sys1,sys2{,inp,out})`	$H_2(s)H_1(s)$
`Open_loop_system(Plant,Control{,Feedback})`	$G_p(s)G_c(s)H(s)$
`Closed_loop_system(Plant,Control{,Feedback})`	$\frac{G_p(s)G_c(s)}{1+G_p(s)G_c(s)H(s)}$
`SISO(system,inp,out)`	
`Rename_inputs(sys),[new_names])`	
`Rename_outputs(sys),[new_names])`	
`Rename_states(sys),[new_names])`	

Table 2.7: Elementary system generation, combination, and modification procedures

2.8 System Combination Procedures

SISO system objects, defined by the system creation commands, may be combined to form new systems. `Parallel_systems(sys1,sys2)` combines two systems in parallel, that is they share the same input, and the outputs are summed to form the output variable. If the input and output variable names are not given, they are taken from `sys1`. `Cascade_systems(sys1,sys2)` combines two systems in series; the output of `sys1` drives the input of `sys2`. In this case if the input and output names are not given, the input name is taken from `sys1`, and the output name is taken from `sys2`. (For this reason alone, care should be taken to preserve the order of the systems.) It is assumed that the systems are non-loading and that connected inputs and outputs have compatible units. For example:

```
sys:= Parallel_systems(TF_to_system(1/(3*s+1),s),Integrator(1),in,out):
open_loop := Cascade_systems(PID(10,1,2), Plant):
```

The procedures `Open_loop_system()` and `Closed_loop_system()` are intended for use with the control system design procedures described in Chapter 4. They allow the the generation of new system objects from the interconnection of SISO systems in classical open-loop and closed-loop structures. For example, a PID controller with a second-order plant and first-order sensor dynamics may be specified:

```
Plant := TF_to_system(5/(s*(s+5)),s):
Sensor := TF_to_system(1/(0.1*s+1),s):
Cloop := Closed_loop_system(Plant,PID(10,.1,3),Sensor):
```

See Chapter 4 for more details.

The procedure `SISO(system,inp,out)` is provided to create a single-input/single-output system from an existing MIMO system object.

2.9 Renaming System Variables

The names of system variables may be changed through the procedures `Rename_inputs(sys,[new_names])`, `Rename_outputs(sys,[new_names])`, and `Rename_states(sys,[new_names])`. These commands create a new system object which can be assigned to a new name (leaving the original object intact), or reassigned to the same name. The argument `[new_names]` must be a Maple list (enclosed in square brackets) and and of the same length as the list being replaced. In other words these procedures replace the whole name list.

In the following example, the `TF_to_system()` procedure will create the name `u_1` for the input variable, `y_1` for the output, and `x_1, x_2` for the state variables. The rename procedures are used to assign new names:

```
sys := TF_to_system(5/((s+3)*(s+5)),s):
sys := Rename_inputs(sys,[Vin]);
sys := Rename_outputs(sys,[Vout]);
sys := Rename_states(sys,[x1,x2]);
```

In each case the system `sys` has been updated because the newly created system has been assigned the same name.

Chapter 3

System Property and Response Functions

3.1 Changing the Values of System Parameters

System parameters may be specified in either symbolic or numeric form. SYREP presents all results in closed-form expressions in terms of the symbolic names and numeric values. Numerical values may be assigned to parameters with the Maple assignment (`name := value`), such as:

```
m := 5;   or   B := 27;
```

All subsequent references to the parameters m or B will return the numerical values, and all subsequent analyses will have the numerical values substituted into the formulations. A potential problem occurs when it is desired to return to a symbolic representation for the parameter. The only way to do this in Maple is to assign the value of the parameter to its "unevaluated" value (which is its name):

```
m := 'm';   or   B:= 'B';
```

where the use of single quotes (') (not back-quotes (‘)) is essential.

A system is said to be *fully evaluated* when numerical values have been assigned to all parameters. The Maple term for an unassigned symbolic parameter is an *indeterminate*. Most SYREP procedures will return results when a system is not fully evaluated, but there are exception:

- It is not possible to plot functions (responses, pole-zeros, etc) when a system is not fully evaluated.
- Procedures that require solution of polynomials will not function properly with a system that is not full evaluated.

A set of ten functions is provided for extracting the system matrices and descriptions from the internal representation, as summarized in Table 3.1. In each case the argument `sys` is the name assigned to the system. In order to display the **A** matrix and state vector of a system named `mechanical`, the following commands would be entered at the Maple prompt:

```
A_matrix(mechanical);
State_vector(mechanical);
```

where the semi-colon line terminators instruct Maple to display the results.

3.2 System Property Functions

Table 3.1 defines the SYREP functions that return basic descriptions and properties of linear systems. In each case the argument `sys` is the name of the system being analyzed. Optional arguments are shown

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, $\lim(s \to 0)\{s^2G(s)\}$.
`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.

Table 3.1: SYREP functions to determine system properties.

in braces { and }, with the restriction all arguments within the braces must be included if any are to be specified. In the case of SISO systems it is not necessary to specify the input and output variables, and the arguments *inp* and *out* may be omitted; for MIMO systems both must be specified.

The command

```
tf := Transfer_function(mechanical,s, Fin, vm);
```

will compute and display the transfer function of the system *mechanical* output variable *vm* to an input *Fin*. The transfer function is expressed as a rational function in the variable *s*. If *mechanical* is a SISO system the statement

```
Transfer_function(mechanical, s);
```

is sufficient.

The procedure `Root_locus()` is a generalized root locus plotting procedure that plots the system poles as a function of any system parameter. The system must be fully evaluated (numerical values assigned to all parameters) with the exception of the parameter being studied. `Root_locus()` requires the specification of one or more "ranges" in its argument list in the form `min..max`. For example, to plot the variation of the poles of the system `mechanical` as the parameter B is varied from 0 to 10 in 150 steps, and plotting the results on the s-plane with a y scale from -10 to 10 and an x-scale from -20 to 0 the SYREP command is

```
Root_locus(mechanical,B,0..100,150,-20..0,-10..10);
```

These commands may be integrated into standard Maple commands. For example, plots of the damping ratio and undamped natural frequency of a second-order mechanical system as a function of the the damping coefficient B, and the spring stiffness K may be made using the standard Maple `plot` command:

```
graph := [[2,1,Force,Fs],[1,2,mass,m,vm,Fm],[1,2,dashpot,B],[1,2,spring,K]]:
system := LGraph_to_system(graph,vm):
m := 10:  K:=50:
plot(Damping_ratio(system), B=0..100);
K := 'K': B := 5:
plot(Natural_frequency(system), K=0..100);
```

In each case the all system parameters, except the independent plotting variable, are assigned numeric values.

See Part II of the User Manual for more detailed descriptions of these functions.

3.3 Time Domain Response Functions

Table 3.2 summarizes the available functions to compute thetime domain response of a system. As before *sys* is the system name, *inp* and *out* are the names of the selected input and output variables (optional for a SISO system), *function* is a specified function of time, *ic* is a Maple list of the initial values of the state variables or the output. To find and plot the response of the system to an input $u(t) = \sin(t)$,

```
y := System_response(mechanical, Vin, vm, sin(t)): plot(y,t=0..10);
```

or as a single command:

```
plot(System_response(mechanical, Vin, vm, sin(t)), t=0..10):
```

The specification of system initial conditions is in the form of a Maple *list*. For the procedure `Output_IC_response()` the list contains the output $y(0)$ at time $t = 0$ and the first $n-1$ derivatives at $t = 0$, for example a second-order system with $y(0) = 2.0$ and $\prime y(0) = 0$ is specified

```
Output_IC_response(second_order, [2.0, 0], out);
```

The procedure `State_IC_response()` list contains the values of the state variables at time $t = 0$, specified in the same order as in the state vector. A third-order system with initial values $x1(0) = 5.0$, $x2(0) = 0.0$, $x3(0) = -3.0$ would be analyzed

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.

Table 3.2: SYREP Time Domain Response Functions

```
State_IC_response(third_order, [5.0, 0.0, -3.0], out);
```

or optionally

```
State_IC_response(third_order, [5.0, 0.0, -3.0]);
```

if it is a SISO system.

See Part II of the User Manual for more detail on these functions.

3.4 Frequency-Domain Response Functions

The functions in Table 3.3 are available to examine the frequency domain response of a system. See Part II of the User Manual for more detail on these functions.

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.

Table 3.3: SYREP Frequency Domain Response Functions

Chapter 4

Control System Design Functions

Table 4.1 describes functions that are useful in the design of feedback control systems. In addition many of the system property and response functions described in Chapter 3 have direct application in control system design. In particular the `Gain(), Integrator(), Differentiator(), PID()` system generation procedures, the observability and controllability procedures, and the static error constants (`Position_error_constant()`, etc) have direct application in control system design. Consider the closed-loop system shown in Fig. 4.1,

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.

Table 4.1: SYREP Closed-loop control design functions

with controller dynamics specified by the transfer function $G_c(s)$, and sensor/feedback dynamics by $H(s)$. The function `Closed_loop_system(plant, ctrl, feedback)` generates a SYREP system object defined by the closed-loop transfer function

$$G_{cl}(s) = \frac{G_p(s)G_c(s)}{1 + G_p(s)G_c(s)H(s)}$$

where $G_p(s)$ is the transfer function of the plant `plant`, $G_c(s)$ is the transfer function of the controller `ctrl`, and $H(s)$ is the transfer function of the sensor and feedback path `feedback`. As an example, consider a plant with transfer function

$$G_p(s) = \frac{10}{s+1}$$

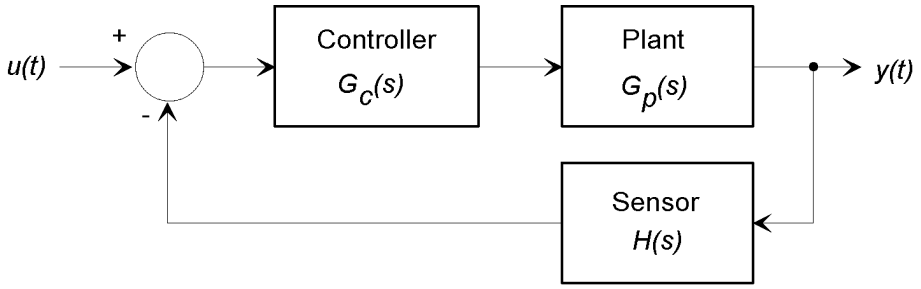


Figure 4.1: A closed-loop system.

which is to be compensated with a PID controller $G_c(s) = 1 + 0.1/s + 0.2s$. The sensor has a first-order lag characteristic

$$H(s) = \frac{1}{5s+1}$$

Suitable SYREP code to determine the transfer function and plot the closed-loop step response is:

```
Plant := TF_to_system(10/(s+1,s)):
Sensor:=TF_to_system(1/(5*s+1),s):
cl := Closed_loop_system(Plant, PID(1,0.1,0.2), Sensor, ref, response):
plot(Step_response(cl), t=0..15);
```

The function `Open_loop_system()` will return a SYREP system described by the open-loop transfer function

$$G_{ol}(s) = G_p(s)G_c(s)H(s)$$

For example to find the gain and phase margins of the above closed-loop system

```
Plant := TF_to_system(10/(s+1)):
ol := Open_loop_system(Plant,PID(1,0.1,0.2), Sensor):
GM := Gain_margin(ol):
PM := Phase_margin(ol);
```

The functions `State_feedback()` and `Ackerman()` may be used to place closed-loop eigenvalues. `Ackerman()` returns a state-feedback gain matrix to place the poles at values prescribed by the given characteristic polynomial; `State_feedback()` can use this matrix to create the closed-loop system:

```
# Enter a two-mass system connected by a spring
Graph := [[1,3,mass,m1],[2,3,mass,m2,vm2,Fm2],[1,2,spring,K],[3,1,Force,Fin]]:
out:= [vm2]:
system:=LGraph_to_system(Graph,out):
# Assign values and check controllability and observability
m1:=10:  m2:=20:  K:=50:
Observable(system);
Controllable(system);
# Find feedback gains to set closed-loop poles at (-1, -2, -5).
Gains:=Ackerman(system,(s+1)*(s+2)*(s+5)):
CL := State_feedback(system,Gains):
# Check pole placement and plot step response.
System_eigenvalues(CL);
plot(Step_response(CL, t=0..10));
```

The Maple SYREP root locus procedure `Root_locus()` is a more general tool than the classic root locus technique used in control system design. SYREP's root locus plots the loci of system poles for variation of

any parameter in the transfer function, whereas the classical root locus assumes a characteristic equation of the form

$$1 + KG_{ol}(s) = 0$$

In order to use the `Root_locus()` procedure to study closed-loop behavior with a variable controller gain, the controller should be formulated with an overall gain constant, the closed-loop system formulated, and used in `Root_locus()`. For example, the following uses a compensator with an overall gain K

```
Plant := TF_to_system(10/(s*(s^2+2*s+10),s):
Control := TF_to_system(K*(s+1)/(s+10),s)):
cl := Closed_loop_system(Plant,Control,Gain(1.5)):
Root_locus(cl, K, 0..20,50);
```

In the following a PID controller is written with an overall gain constant:

```
Plant := TF_to_system(10/(s*(s^2+2*s+10),s):
cl := Closed_loop_system(Plant,PID(K,0.1*K,0.5*K),Gain(1.5)):
Root_locus(cl, K, 0..20,50);
```

so that the controller transfer function is

$$G_c(s) = K(1 + \frac{0.1}{s} + 0.5s)$$

Notice however that SYREP's `Root_locus()` allows much more general root locus studies. For example, the effect of derivative gain control on the closed-loop poles can be plotted

```
Plant := TF_to_system(10/(s*(s^2+2*s+10),s):
cl := Closed_loop_system(Plant,PID(20,2,Kd),Gain(1.5)):
Root_locus(cl, Kd, 0..20,50);
```

See Part II of the User Manual for more detail on the control systems procedures.