DYN, Nad lesíkem 27, CZ-160 00 Prague 6, `dyn@virtual.cvut.cz`

DYNAST

User's Guide

Heřman Mann, Michal Ševčenko

Version 3.7.21

Release Date: November 29, 2004

Table of contents

Chapter 1
About DYNAST

1.1 What DYNAST can do for you

DYNAST is a versatile computer program for efficient solving of nonlinear algebro-differential equations as well as for modeling, simulation and analysis of general nonlinear dynamic systems. For linear systems, DYNAST provides also semisymbolic analysis both in time and frequency domains.

DYNAST input language permits for describing the dynamic systems under consideration in the following ways:

- by a set of equations submitted in a natural textual form without converting them into a block diagram
- by a block diagram without any restrictions like algebraic loops
- by a multipole diagram portraying directly the configuration of the analyzed real dynamic system

The three above approaches can be freely combined.

DYNAST users are not required to have any knowledge of computer programming. Its input language is strongly problem as well as object oriented, and it is very versatile yet simple to use. The input data is directly interpreted by the program, so there is no compilation delay. DYNAST is accompanied by a graphical user's environments for easier program control, for plotting the output results, and even for submitting the analyzed diagrams in a graphical form.

The computational procedures built-in DYNAST belongs to the most effective ones and, at the same time, they are robust enough not to require much of user's knowledge of numerical mathematics when solving prevailing class of problems. Despite the fact that DYNAST uses numerical solution procedures only, for linear problems it can yield results in a closed semisymbolic form. Many operations are automated, which helps to make DYNAST a user-friendly tool.

DYNAST is easily transportable to a wide variety of computers as it is coded in the C++ language.

1.2 Solving equations

DYNAST is capable of solving systems of ordinary nonlinear and nonstationary first-order *algebro-differential equations* in the implicit form

$$\mathbf{f}(\mathbf{x}(t), \dot{\mathbf{x}}(t), t) = \mathbf{0} \tag{1.1}$$

where $\mathbf{f}(.)$ is the vector of functions, $\mathbf{x}(t)$ is the vector of the solved variables and $\dot{\mathbf{x}}(t)$ is the vector of their derivatives with respect to the independent variable t.

DYNAST also solves systems of nonlinear algebraic equations

$$\mathbf{f}(\mathbf{x}(t),t)=\mathbf{0} \tag{1.2}$$

just as a special case of (1.1). As the DYNAST input language permits for submitting the equations into the program directly in the textual form, there is no need to represent them first by a block diagram. The equations can be specified by means of a wide variety of functions including boolean functions and functions defined by a table. The expressions can be branched taking into account user specified conditions put on the solution.

To solve 1.1 in the interval $t\in\langle t_0,t_1\rangle$ for given initial conditions $\mathbf{x}(t_0)$ DYNAST uses a stiff-stable integration method. During the integration, DYNAST continuously optimizes both the integration step length as well as the order of the method to minimize the computational time while respecting the specified admissible computational error. DYNAST is robust enough even for sharply nonlinear and/or nonstationary equations the number of which changes during their solution.

The solution can start either from initial conditions specified by the user, or from the initial conditions corresponding to a steady-state of the system – either static or periodic, both of which can be computed automatically. In the the former case the fast Fourier spectral analysis can then be applied to the resulting periodic-steady state responses.

When the transient solution is interrupted, the last solution vector is automatically stored and than it can be used anytime later as a starting point for further computations. The solution can also be interrupted automatically after the occurrence of a specific event.

In case of quasistatic analysis, during which some of the system or ambient parameters is supposed to vary slowly in a specified range, the independent variable t represents the varied parameter. The integration procedure built-in DYNAST is then used to control the independent parameter variations.

1.3 Analyzing block diagrams

Block diagrams represent graphically nothing but sets of equations. The DYNAST variety of blocks is very narrow comparing to other simulation programs. Yet DYNAST permits for very versatile block diagram modeling thanks to the possibility to specify the function of the individual blocks by flexible mathematical expressions.

DYNAST also offers the possibility to set up the analyzed block diagrams not only from its built-in block elements, but also from much more sophisticated multiinput multioutput macroblocks specified by the user and stored in independent data files. These macroblocks can be nested in a hierarchical way.

As DYNAST formulates the block diagram equations simultaneously, it has no problems with the ‘fast’ or ‘algebraic’ loops as it is common to most of the other simulation programs. The block diagrams may be combined with equations as well as with multipole diagram elements.

In case of the nonlinear block diagram analysis, DYNAST can carry out similar operation as in the case of the algebro-differential equation solution described above. Besides that, DYNAST is able to linearize automatically the equations of nonlinear block diagrams for their small-excitation analysis in the vicinity of a computed or user specified operating point.

In case of linear or linearized block diagrams, DYNAST makes possible to compute numerically their rational transfer functions and transforms of initial-condition responses semisymbolic form, i.e. with symbolic Laplace s operator and numerical polynomial roots and coefficients. For the semisymbolic transfer functions the program than can compute semisymbolic frequency and time characteristics. The frequency characteristics of distributed-parameter systems DYNAST computes directly without resorting to any transfer functions.

1.4 Modeling real systems

Modeling of real systems is easy when using DYNAST, as it is based on the multipole modeling approach. The multipole models can be submitted to DYNAST in a graphical form isomorphic with the geometric configuration of the modeled real system. Therefore, a multipole model can be set up in a kit-like fashion based on mere inspection of the real system in the same way in which the system has been assembled from its real components.

Multipole models are represented graphically by multipole diagrams consisting of symbols for the individual multipoles. The symbols are interconnected by line segments representing energy interactions between the real components in a system. The variety of the multipole models can range from 'pure' two-pole elements like resistors, capacitors, inductors, dampers, inertors and springs up to sophisticated models of complex real components and subsystems.

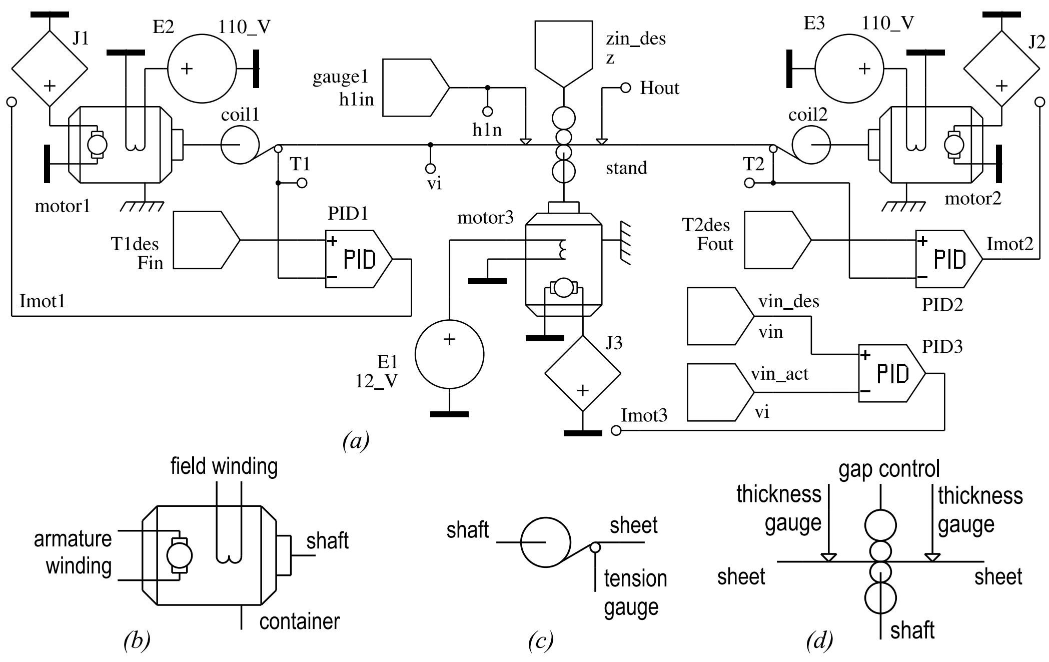


Figure 1.1 (*a*) Rolling mill for metal strips. Multipole models of (*b*) DC motor, (*c*) strip coiler, (*d*) rolling stand.

Fig. 1.1*a* shows a multipole model of a cold-rolling mill. Multipole models of the individual mill subsystems are shown in Fig. 1.1*b* through *d*. The DC motors driving the coilers with metal-strip coils are controlled for a constant tension in the strip. The DC motor driving the milling rollers is controlled for a constant strip velocity. The gap between the rollers can be adjusted continuously by a hydraulic ram. The DC motor field windings are supplied by sources of voltage and the armature windings by controlled sources of current.

The multipoles in Fig. 1.1*a* are combined with block diagrams modeling the controllers forming a dynamic diagram. Note the difference between the line segments interconnecting the blocks and the multipoles. The latter line segments represent in Figure 1.1*a* bidirectional energy transfers between components via idealized electrical conductors, shafts and the metal strip. Each of the line segments is associated with a pair of conjugate variables the product of which represents the transferred power. The segment interconnections respect physical laws governing the energy interactions.

The line segments interconnecting the blocks are associated with one mathematical variable only propagating just in one direction only. The interactions respect algebraic rules, but ignore any physical laws. Obviously, while the block diagrams are nothing but graphical representations of equations, the multipole diagrams are mappings of real system geometric configurations onto their topological representations.

1.5 Designing control

DYNAST can be easily used as a *modeling toolbox for MATLAB*. While MATLAB is well suited to control design, DYNAST is capable of automated equation formulation even for very realistic models of real systems. You can combine these two programs to exploit advantages of both.

Notably, you can implement and analyze a nonlinear model of the plant to be controlled in DYNAST. Then, for example, you may ask DYNAST to linearize the model, compute transfer-function poles and zeros of the plant model, and export them for the plant-control synthesis to MATLAB. Finally, to verify the complete controlled system you can use DYNAST again after augmenting the plant model by the resulting control configuration. During this design phase, you may use DYNAST to consider also plant nonlinearities as well as the non-ideal features of the controllers and sensors in various operation regimes of the control system. If the designed control is digital, you may verify it by interconnecting DYNAST with SIMULINK so that these two packages can communicate with each other at each time step.

For example, also the cold-rolling mill shown in Figure 1.1*a* has been controlled in a similar way. The hydraulic ram adjusting the roller gap can be controlled digitally across the Internet by a sophisticated adaptive controller implemented in Simulink and respecting the data coming across the Internet from the modeled gages and motors. During the simulation, the thickness of the incoming metal strip was varied randomly while a considerable transportation lag between the gage measurement and the rolling process was taken into account.

Chapter 2
DYNAST simulation system

2.1 DYNAST Solver

2.1.1 Sections of DYNAST Solver

The DYNAST simulation system consists of two separate parts – DYNAST Solver and a DYNAST working environment. DYNAST Solver is composed of several sections sharing common data as shown in Fig. 2.1. The section SYSTEM reads in the system-model description in the form of a set of algebro-differential equations, a block diagram, a multipole diagram, or in a form combining freely these approaches.

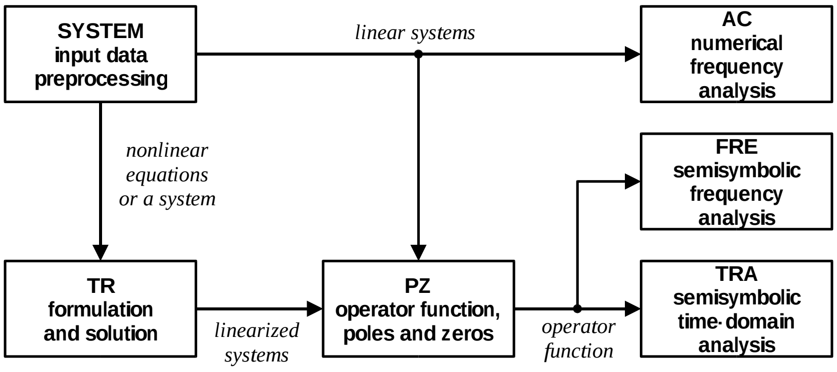


Figure 2.1 DYNAST solver

Nonlinear systems of equations can be solved, and nonlinear diagrams analyzed, in the TR section. This section computes system transient responses and system steady states, either static or periodic. In the former case, DYNAST automatically sets the time derivatives of all variables to zero. The static steady-states can be computed also for a system- or ambient-parameter sweeped through an interval. The transient responses can start either from initial conditions specified by the user, or from initial conditions corresponding to a static or periodic steady-state of the system. Fourier spectral analysis can be than applied to the periodic steady-state responses.

The TR section provides also automatic linearization of the analyzed nonlinear system. The resulting linearized system model can be subjected to small-excitation analysis in the vicinity of its user-specified or computed quiescent operating point. This analysis, yielding operator functions representing either system transfer functions or the transform of system initial-state responses, can be provided by the PZ section. The resulting operator functions are available in a semisymbolic form with the Laplace operator *s* as a symbol, and with the polynomial roots and coefficients as numbers. For such operator functions, DYNAST can than compute semisymbolic- and numeric-form time-domain characteristics using the TRA section, the FRE evaluates frequency characteristics numerically. For linear systems, DYNAST provides also an option for their direct numerical frequency analysis in the AC section. This approach allows for the analysis of distributed-parameter dynamic systems (using hyperbolic and other frequency-dependent functions).

2.1.2 Input language of DYNAST Solver

DYNAST input-language syntax is described in detail in the following chapters. Here only the general structure of the language is given.

DYNAST input-language is composed of **statements** coded in ASCII characters and both upper and lower case **letters** may be used in them. DYNAST, however, is not case sensitive and converts all the letters into the upper-case ones. Each of the statements is terminated by the semicolon character '`;`'. A statement may continue on several lines, and there may be several statements placed in one line. The maximum number of characters in one line including spaces is 80.

The statements consist of items like DYNAST input-language keywords and identifiers, user-defined identifiers and other user-specified items separated by **delimiters** made by characters which are neither letters nor numbers, i.e., by characters like '`/`', '`-`', '`=`', '`,`' and '`.`' or by spaces. The underscore character '`_`', however, is treated as a letter. It is fully insignificant if only one or more subsequent spaces are used as a delimiter.

The **user-defined identifiers** denoting variables and other items of the analyzed systems can be up to 8 alphabetical characters long. Any additional characters are ignored. The identifiers may not contain any non-numerical characters or spaces.

If there is a colon '`:`' in a line, the part of the line to the right of this character will be considered as a **comment** and ignored as such by the program. (This feature may be used for deactivating some statements during the input data debugging procedure.) When the colon '`:`' is preceded by the asterisk '`*`', the comment will be used by DYNAST as a **problem title** in the resulting tables and graphs.

2.1.3 Data-files of DYNAST Solver

When solving a problem, DYNAST communicates with the data-files of the following type:

- input data **problem file** specifying the system model to be analyzed as well as the required mode of analysis
  – file-name extension: `prb`
- input-data **submodel files** specifying component or subsystem models in the form of superblocks or multipoles
  – file-name extension: `mod`
- an **output-data file** containing a copy of the input data, the computed results, and any eventual error messages
  – file-name extension: `O`
- an input/output data file storing the **last-solution vector**, which can be used as an initial-condition vector for subsequent analysis
  – default file-name extension: `inc`

Table 2.1 DYNAST sections

Section	Purpose
`SYSTEM`	Reading in the description of the system to be analyzed in the form of • a set of algebro-differential equations • a block diagram • a multipole diagram • any combination of the above Library models may be used for superblocks and multipoles.
`TR`	Transient analysis and its special cases like • transient analysis with the initial conditions determined by the static equilibrium analysis • static analysis with a parameter being varied • periodic steady-state and subsequent fourier spectral analysis • determination of events and intervals between them • linearization of the analyzed system
`PZ`	Semisymbolic analysis in the Laplace transform domain, i.e. the computation of poles, zeros and polynomial coefficients of rational • transfer functions • transforms of initial-state responses for linear or linearized systems.
`TRA`	Semisymbolic and numerical analysis of time characteristics and responses.
`FRE`	Numerical evaluation of frequency-characteristics components for semisymbolic transfer functions.
`AC`	Direct numerical point-by-point frequency analysis of linear or linearized block or multipole diagrams.

All the files are text files.

The overall structure of the input problem-file is following:

```
*SYSTEM
⋮
data for the section SYSTEM
⋮
*section;
⋮
data for the next section
⋮
*section;
```

```
:
*END;
```

Each of the **input-data sections** is introduced by the statement `*`*section*`;` where *section* is a **section identifier** preceded by the asterisk ‘`*`’ without any space. The section identifiers are listed in Table 2.1. The statement `*END;` terminates the input data. Any data behind this statement is ignored by DYNAST.

During one DYNAST execution, several systems can be submitted, and each system can be analyzed repeatedly with its parameters modified.

2.2 DYNAST for MS Windows

2.2.1 DynShell working environment

The DYNAST Shell or DynShell has been designed for MS Windows to provide a user-friendly support for a wide variety of tasks. It suits to users of different levels of qualification and experience. All operations are supported by a context sensitive help system. There is a built-in syntax analyzer continuously checking the submitted data. Dialog windows (wizards) allow for submitting data without knowledge of the input language. Multipole and block diagrams can be submitted in a graphical form using a built-in schematic editor. The same tool can be used to create submodel symbols. The diagrams can be imported also from the OrCAD schematic editor and converted to the DYNAST language.

Resulting data can be plotted in various arrangements and the plots can be exported in the Encapsulated PostScript format. DynShell can communicate with a server-based automated LaTeX documentation system generating reports for simulation experiments in the PostScript, PDF and HTML formats. DynShell has also the capability to act as a modeling toolbox for MATLAB and Simulink. The simulation results can be used for animation of 3D models formed using VRML- and Java-based visualization tools.

2.2.2 Working with DynShell

You can start DynShell by simple clicking the DynShell icon on the Windows desktop. Unless some open work windows have been stored in your DynShell main window from the previous session, the main window will be empty and the number of menus in will be very small. The reason is that there are several different views of the DynShell main window, each of them with its own set of tools specialized for work activity with a particular type of document. Table 2.2 gives a survey of DynShell views and related work windows, documents and file extensions.

There are different pulldown menus, toolbar buttons, or key shortcuts for different types of document. To get help about a menu entry or a tool, press `Shift-F1` and click on the menu entry or toolbar button. Note that the toolset changes as you switch between documents.

If you are new to DYNAST, we strongly recommend to you to open the Help menu first and to examine all the items available there to support your work with DYNAST. We also recommend you to pay attention to the Options chosen in the Preferences menu. In this menu, you can also select the toolbars that you want to use for different tasks.

Table 2.2 DynShell work windows

Work activity	Work window shows	File extension
Problem browsing	list of problem files	
Submodel browsing	list of submodel specification	
Problem editing	text of problem specification	`.prb`
Submodel editing	text of submodel specification	`.mod`
Diagram creating	diagram of system model	`.dia`
Symbol designing	symbol library	`.lbr`
Output examining	text of output data	`.o`
Output plotting	plot of output data	`.o`
Document processing	text of Latex source	`.tex`

2.2.3 Browsing files

From the View menu, choose Problem List to open the list of al the problem files stored in the Input data directory chosen in the Preferences menu. It allows you browsing through the files in this directory as well as in its subdirectories. The green OK mark denotes the files without any syntax error. The red cross indicates a syntax error in the problem file, or in any of the submodels called from the problem file. Clicking a file name in the list opens the work window with the file. The erroneous statements are red underlined there, and pointing the cursor at such a statement activates an error message.

If the column Diagram is not empty, there is a diagram file in the current directory with the same file name as the problem file. The OK characters indicate that the diagram corresponds exactly to the problem file netlist, the "out-of-date" statement indicates differences. Clicking the nonempty diagram box in the list opens a work window with the diagram. In a similar way, from the View menu you can open the list of all the submodel files. From there, you can access all the submodel files and symbol libraries stored in the chosen directory and its subdirectories.

2.2.4 Saving and restoring the state of the application

The **Save Screen Layout** and **Load Screen Layout**s allow you to save the state of the DynShell to a file, and to restore it later. The state means the position of the main frame, positions and properties of opened documents (such as cursor and scroller positions), variables displayed in the plot viewer, etc. If you close DynShell the layout is saved, and then it is restored when you start DynShell again.

2.3 Access to DYNAST across the Internet

DYNAST Solver has been installed on a Linux computer connected to the Czech Technical University server. It can be accessed via `http://virtual.cvut.cz/dyn/` in a Web-based, on-line and e-mail modes.

Web-based access. DYNCAD - a special Web-based working environment has been developed in the form of a Java applet. It allows for setting up multipole and block diagrams directly on the Web. A wide variety of symbols of twopoles from different physical domains as well as symbols of energy transducers and blocks are available there. For more complex system components, models stored in the DYNCAD libraries can be utilized, and also, users can define their own models and symbols. DYNCAD converts diagrams into the DYNAST input language and sends the data to DYNAST across the Internet. Users can open their free private accounts in DYNCAD and store there their simulation problems. DYNCAD is able to export the set-up diagrams into PostScript and to send them to the users by e-mail.

After the computational results are sent back to a client computer by the DYNAST server, there are two options for a graphical display of the computed responses on the client's screen. The plots generated on the DYNAST server in HTML are sent to the client computer as a Web-page. Or, the output data from DYNAST can be visualized on the client computer by means of a Java applet. The Web-based mode function is illustrated by Fig. 2.2.

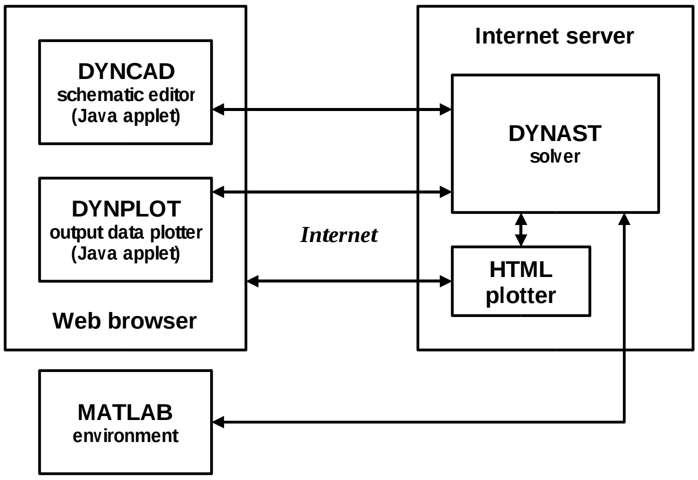


Figure 2.2 DYNAST Web-based

On-line access. DYNAST can be utilized in an even more comfortable and user-friendly way using the on-line access mode illustrated in Fig. 2.3. This mode requires, however, downloading and installing the DynShell software package forming DYNAST user's environment for PC computers with MS Windows.

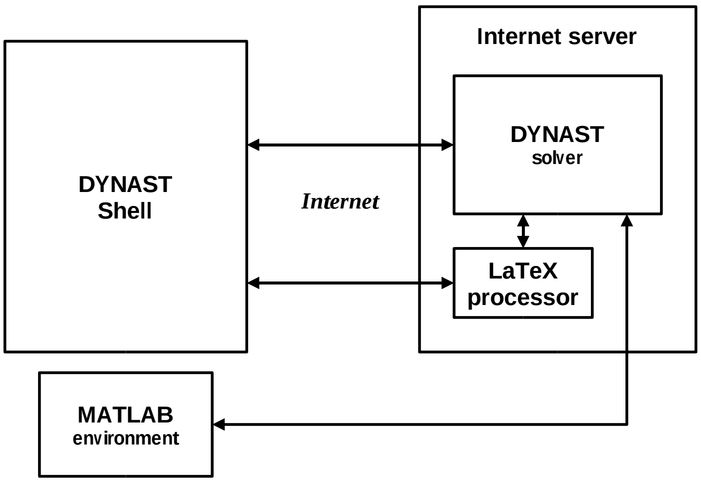


Figure 2.3 DYNAST on-line

E-mail access. The e-mail access was designed for those with a limited access to the Internet. The users can send a file with the input data for DYNAST containing equations or the netlist of a dynamic diagram to the e-mail address: `DYNAST@icosym.cvut.cz` with the subject: `compute`. They then receive the results sent them back to their e-mail address automatically.

2.4 Communication with MATLAB

Using either DYNCAD or DynShell, the plant model can be easily set up in a graphical form. DYNAST can be then used to simulate the plant and to validate its open-loop model. If the model is nonlinear, DYNAST is capable of linearizing it. DYNAST then can compute the required plant transfer-function poles and zeros, and export them to MATLAB in an M-file. After designing an analog control within the MATLAB environment, the DYNAST model of the plant to be controlled can be augmented by the designed control structure and thoroughly verified by DYNAST.

In the case of a digital control design, there is another option for the design verification. After designing the digital control in the MATLAB environment, the resulting control structure can be implemented in Simulink while the controlled plant model remains in DYNAST. Simulink installed on the client computer can then communicate with the remote DYNAST at each time step across the Internet exploiting the Simulink S-function.

Chapter 3
Submitting systems of equations

3.1 Systems of equations

You can use DYNAST to solve simultaneously systems of ordinary algebro-differential implicit-form equations within a given interval of an independent variable for given or computed initial conditions. DYNAST is also capable of evaluating variables specified by explicit-form equations.

The equations may be nonlinear as well as nonstationary. All the equations solved simultaneously may be purely algebraic, purely differential, or a mixture of both. The differential equations are assumed to be of the first order. (Note that any differential equation of the *n*-th order can be easily converted into *n* first-order equivalent equations using a simple substitution.)

Equations are submitted to DYNAST in a natural textual form without the necessity to convert them into a block diagram. In a problem specification, equations can be mixed with blocks, physical elements or library submodels. Utilizing the integrator block also integro-differential equations can be solved by DYNAST.

3.2 Implicit equations

3.2.1 Implicit algebro-differential equations

An **implicit algebro-differential equation** submitted to DYNAST is assumed to be in the form

$$f\,(x_1, x_2, \ldots, x_n, \dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n, p_1, p_2, \ldots, t) = 0$$

where $f\,(\cdot)$ is a known function, $x_1(t), x_2(t), \ldots, x_n(t)$ are the **solved variables**, $\dot{x}_1(t), \dot{x}_2(t), \ldots, \dot{x}_n(t)$ are their derivatives with respect to t, and $p_1(t), p_2(t), \ldots$ are some variables or parameters which have been evaluated already. The independent variable t should be denoted by the identifier ‘`TIME`’, which stands indeed in most cases for the physical quantity called time, but it may also represent some other known independent variable of any physical dimension.

As DYNAST solves the implicit equations shown above for the given initial conditions $x_1(t_0), x_2(t_0), \ldots, x_n(t_0)$ simultaneously, the equations can be arranged in any order. The equations cannot be solved, however, if they are singular.

3.2.2 Implicit algebraic equations

DYNAST can solve the implicit algebro-differential equations even if the time derivatives of all the solved variables $x_i(t)$ are zero, i.e., if these are purely **algebraic equations**

$$f(x_1, x_2, \ldots, x_n, p_1, p_2, \ldots, t) = 0$$

In this case, the option to specify initial conditions can be used instead for submitting **initial estimates of the solved variables** $x_1^0(i_0), x_2^0(t_0), \ldots, x_n^0(t_0)$ at $t = t_0$. This allows for

- speeding up the iterative solution process
- determining which of the solutions is going to be computed in case of **multiple-solution equations**
- avoiding the equation singularity at the start of the computation

If no initial estimate to the solution is submitted, DYNAST takes it as zero by default.

3.2.3 Equation singularity

If the solved problem is specified by a set of equations exclusively (i.e., without blocks or physical elements), the number of the submitted implicit algebro-differential equations must be equal to the number of the unknown solved variables. If these numbers are different, or if the equations are linearly dependent, DYNAST indicates this situation by the error message '`SYSTEM IS SINGULAR`'.

It may happen that the algebraic equations are singular just for the initial estimate of the solution applied at the start of the computation. If this is the case, the initial condition option can be utilised for submitting an initial-solution estimate removing the equation singularity.

3.3 Explicit equations

An **explicit equation** is assumed to be of the form

$$y = g(x_1, x_2, \ldots, x_n, \dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n, p_1, p_2, \ldots, t)$$

where $y(t)$ is an **evaluated variable** or an **evaluated parameter**, $g(\cdot)$ is a known function, t is an independent variable, $x_1(t), x_2(t), \ldots, x_n(t)$ are variables acting as solved variables in a preceding set of implicit equations or dynamic diagram, $\dot{x}_1(t), \dot{x}_2(t), \ldots, \dot{x}_n(t)$ are derivatives of the solved variables with respect to t, and $p_1(t), p_2(t), \ldots$ are some variables or parameters which have been evaluated already in a preceding explicit equation.

DYNAST does not solve the explicit equations simultaneously with implicit equations, it evaluates them only. This means, that a value for the variable or parameter defined by an explicit equation is computed after substituting values for the equation arguments resulting from the last (but not current) iteration.

3.4 Specification of equations using the wizard

1. From the File menu, choose New and open a Problem File.
2. From the System menu, choose Insert Equation.
3. Choose either the Explicit or Implicit equation option.

Specification of an explicit equation

1. Enter an identifier for the evaluated variable in the Left-Hand-Side box.
2. Enter the rest of the equation in the Right-Hand-Side box. To do this, you can choose Expressions as well as the suffix buttons. For details see the next chapter.
3. Choose Insert.

Specification of an implicit equation

1. Enter the equation using the Right-Hand-Side box. As in the former case, you can choose Expressions as well as the suffix buttons.
2. Choose Insert.
3. If DYNAST indicates an Unknown identifier for a variable, you should specify it either as a solved or evaluated variable. If this is the latter case, you should specify it by an explicit equation after choosing the second radio button. In the former case, choose the OK button.
4. Choose Insert again.

3.5 Specification of equations using the text editor

Equation specification should be included into the `SYSTEM` section of the DYNAST input data. Specification of implicit equations should be preceded by a declaration list of all the solved variables by a statement of the form:

`SYSVAR` *solved* `[,` *solved...*`];`

solved

is a user-defined identifier of a solved variable

Each of the **implicit equations** can be entered into DYNAST using the statement

`0 =` *expression*`;`

‘`0 =`’

is the key-string introducing the implicit-equation statement

expression

is a symbolic expression specifying the relation set to zero by the implicit equation. Any evaluated variable or parameter must be specified before it is used as an argument in the expression.

In the expressions, a **solved-variable derivative** with respect to *t* is denoted

`VD.`*solved*

solved is the identifier of the solved variable

Each of the explicit equations can be entered into DYNAST using the statement

evaluated = *expression* `;`

evaluated

is a user-defined identifier of the evaluated parameter

expression

is a numeric constant or symbolic expression. Any evaluated variable or parameter must be specified before it is used as an argument in the expression.

3.6 Examples

Linear constant-coefficient algebraic equations

$$\begin{aligned} 2x_1 - 7x_2 + 4x_3 &= 9 \\ x_1 + 9x_2 - 6x_3 &= 1 \\ -3x_1 + 8x_2 + 5x_3 &= 6 \end{aligned}$$

converted into the implicit form and solved for the unknown variables x_1, x_2 and x_3:

```
*SYSTEM;
SYSVAR x1, x2, x3;
0 =   2*x1 - 7*x2 + 4*x3 - 9;
0 =     x1 + 9*x2 - 6*x3 - 1;
0 = - 3*x1 + 8*x2 + 5*x3 - 6;
*TR; DC; PRINT x1, x2, x3; RUN; *END;
```

□

Nonlinear algebraic equations

$$\begin{aligned} 0.5\sin(u \cdot v) - \frac{v}{4\pi} - 0.5u &= 0 \\ \left(1 - \frac{1}{4\pi}\right)\left(e^{2u} - e\right) + e \cdot \left(\frac{3v}{\pi} - 2u\right) &= 0 \end{aligned}$$

solved for the variables u and v:

```
*SYSTEM;
SYSVAR u, v;
0 = .5*sin(u*v) - v/4pi - .5*u;
0 = (1 - 1/4pi)*(exp(2*u) - exp(1)) +
    exp(1)*(3*v/1pi - 2*u);
*TR; DC; PRINT u, v; RUN; *END;
```

□

An oscillating variable $f = A(t) \cdot \sin 2\pi\omega t$ evaluated at 201 points for $-0.5 \leq t \leq 0.5$: for the decaying amplitude $A(t) = 200 \cdot e^{\zeta t}$, where $\omega = 5$ and $\zeta = -4.46$:

```
*SYSTEM;
zeta = -4.46; A = 200*exp(zeta*time); omega = 5;
f = A*sin(2pi*omega*time);
*TR; tr -.5 .5; PRINT (201) A, f; RUN; *END;
```

□

Time-variable algebraic equations

$$0.3 \cdot \cos\varphi_2 + 0.55 \cdot \cos\varphi_3 + 0.4 \cdot \cos\varphi_4 = 0.6$$
$$0.3 \cdot \sin\varphi_2 + 0.55 \cdot \sin\varphi_3 + 0.4 \cdot \sin\varphi_4 = 0$$

solved for the variables φ_3 and φ_4 in the interval $0 \leq t \leq 1$, assuming that $\varphi_2 = 2\pi t$:

```
*SYSTEM;
SYSVAR phi3, phi4;
0 = .3*cos(2pi*time) + .55*cos(phi3) +
    .4*cos(phi4) - .6;
0 = .3*sin(2pi*time) + .55*sin(phi3) +
    .4*sin(phi4);
*TR; TR 0 1; PRINT phi3, phi4; RUN; *END;
```

□

Explicit, or normal-form, nonlinear differential equations

$$\begin{aligned} \dot{r} &= -0.1r + mf \\ \dot{m} &= 0.1r - m + 0.1f^2 - mf \\ \dot{f} &= 0.1r + 2m - mf - 0.2f^2 - 1000f \end{aligned}$$

converted into the implicit form and solved for the variables r, m and f in the interval $0 \leq t \leq 12$:

```
*SYSTEM; SYSVAR r, m, f;
0 = - VD.r - .1*r + m*f;
0 = - VD.m + .1*r - m + .1*f**2 - m*f;
0 = - VD.f + .1*r + 2*m - m*f - .2*f**2 - 1000*f;
*TR;TR 0 12; INIT f = 9.975, m = 1.674, r = 84.99;
PRINT r, m, f; RUN; *END;
```

Here, $f(0) = 9.975, m(0) = 1.674, r(0) = 84.99$ are initial conditions of the solution.

□

The second-order time-dependent Bessel's differential equation

$$t^2\ddot{y}+t\dot{y}+(t^2-n^2)y=0$$

can be easily converted into two first-order differential equations:

$$y_D=\dot{y}$$
$$t^2\dot{y}_D+ty_D+(t^2-n^2)y=0$$

and submitted for $n=1$, assuming that $y(0)=0$ and $\dot{y}(0)=0.5$:

```
*SYSTEM; n = 1; SYSVAR y, yD;
0 = yD - VD.y;
0 = time**2*VD.yD + time*yD + (time**2 - n**2)*y;
*TR; TR 0 10; INIT yD=.5; PRINT y; RUN;
*END;
```

□

The third-order differential equation

$$\overset{(3)}{x}+a_1\ddot{x}+a_2\dot{x}+a_3x=b_0+b_1\ddot{u}+b_2\dot{u}+b_3u$$

where $u=5\exp(-41)$, $x(0)=10, \dot{x}(0)=25$ and $\ddot{x}(0)=-80$, solved for x in the interval $0\le t\le 0.6$:

```
*SYSTEM; *: Third-order differential equation
:Third-order equation coefficients:
a1 = 6; a2 = 11; a3 = 6; b0 = 0; b1 = 3;
    b2 = 2; b3 = 1;
u = 5*exp(-4*time); :excitation
:Coefficients of three equivalent first-order
:equations:
c0 = b0;
c1 = b1 - a1*c0;
c2 = b2 - a1*c1 - a2*c0;
c3 = b3 - a1*c2 - a2*c1 - a3*c0;
:Three first-order equivalent equations:
SYSVAR x1, x2, x3;
0 = VD.x1 - x2 - c1*u;
0 = VD.x2 - x3 - c2*u;
0 = VD.x3 + a3*x1 + a2*x2 + a1*x3 - c3*u;
x = x1 + c0*u; :third-order equation solution
*TR; TR 0 .6; INIT x1 = 10, x2 = 10; PRINT u, x;
RUN; *END;
```

Here, the initial conditions of the substituted variables:

$$\begin{aligned}
x_1(0) &= x(0) - c_0 u(0) \\
x_2(0) &= \dot{x}(0) - c_0 \dot{u}(0) - c_1 u(0) \\
x_3(0) &= \ddot{x}(0) - c_0 \ddot{u}(0) - c_1 \dot{u}(0) - c_2 u(0)
\end{aligned}$$

□

Chapter 4
Entering symbolic expressions

4.1 Symbolic expressions

DYNAST language allows you to specify problems, either in the form of equations, block diagrams or physical diagrams, using **symbolic expressions** made up of

- numeric constants
- variables and parameters
- operators, arithmetic and logical
- functions, standard and user-defined

4.2 Numeric constants

Numeric constants can be specified either in the common format

fraction [E *exponent*]

or, in a more user-friendly way, as

fraction [*suffix_unit*]

fraction

is a fractional portion of the numeric constant. The decimal point can be placed in any position in it. If the value of this portion is integer, the decimal point may be omitted.

exponent

specifies an integer number n such that 10^n is within the range of the computer arithmetics

suffix

can be a **scale suffix** the variety of which is shown in Table 4.1. There must not be any space between the number fractional portion and its scale suffix.

unit

can be a **unit suffix** i.e., an arbitrary string of up to 8 alphanumeric characters which is separated from the preceeding scale suffix or fractional portion by the underscore character '_' without any space. The unit suffix is ignored by DYNAST.

Examples:

Table 4.1 Scale suffixes

Suffix	Scale factor	Value
`T`	tera	10^{12}
`G`	giga	10^{9}
`ME`	mega	10^{6}
`K`	kilo	10^{3}
`M`	mili	10^{-3}
`U`	micro	10^{-6}
`N`	nano	10^{-9}
`P`	pico	10^{-12}
`F`	femto	10^{-15}
`PI`	Ludolphian number	π

The following numeric constants are all legal numbers:

```
-3.4 .3 67.08E-10 6.3K 5N 1K_VOLT 100_OHM -1PI
```

Please, pay attention to the fact that `PI` is a scalling factor. This means that it must be always preceded by a fractional portion in numeric form. Note also, that the string `1FARAD` is interpreted by DYNAST as 10^{-15}, whereas the string `1_FARAD` is understood as the numeric constant 1.0.

4.3 Variables and parameters

Table 4.2 gives a survey of DYNAST variables and parameters. The value of the **independent variables** `TIME` and `FREQ` is controlled by DYNAST in the user-specified range automatically. The values of the global independent parameter `TEMP` as well as the values of the independent system parameters are specified by the user in the form of an explicit parameter definition equation, which is then evaluated by DYNAST using just a substitution, not a solution, procedure. On the contrary, evaluation of the dependent primary variables requires solution of a set of implicit-form equations or analyzing a block or multipole diagram. To evaluate a dependent secondary variable no equation need be solved, but the dependent primary variables must be solved before already.

The character string '`V.`' preceding a node identifier can be omitted if the first character in the node identifier is a letter, not a number. Further details about the variables and parameters are given in the subsequent chapters.

Table 4.2 Variables and parameters

Format	Variable or parameter	By default
	Independent	
`TIME`	variable of transient analysis	0
`FREQ`	variable of frequency analysis	0
`TEMP`	global parameter	300
	Solved	
variable	variable of implicit equation	
`VD.`*variable*	derivative of implicit-equation variable	
`V.`*node*	node (across) variable in diagram	
`VD.`*node*	derivative of node (across) variable in diagram	
`I.`*element*	through variable of Z-class element	
`ID.`*element*	through-variable derivative of Z-class element	
	Evaluated	
variable	variable or parameter defined by explicit equation	
`V.`*element*	across variable of Z-class element	
`I.`*element*	through variable of Y-class element	
parameter	element parameter	1
event	event variable	10^{32}
interval	interval variable	

4.4 Operators

4.4.1 Arithmetic operators

The DYNAST variety of **operators** is shown in Table 4.3.

Table 4.3 Arithmetic operators

Operator	Operation
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`**`	power
`%`	differentiation

4.4.2 Logical operators

The variety of **logical operators** available in DYNAST language are in Table 4.4.

Table 4.4 Logical operators

Operator	Operation
`<=`	less or equal
`>`	greater than
`>=`	greater or equal
`&`	logical AND
`!`	logical OR
`=`	equality
`'`	logical NOT
`<>`	not equal
`<`	less than

The value of a **logical expression** is either 0 or 1. The logical expressions must be enclosed in parenthesis ‘`()`’. The operands to a logical operator can be numeric constants or symbolic expressions.

Example:

The logical expression

$$X = (A\,\mathrm{OR}\,B)\,\mathrm{AND}\,(C\,\mathrm{OR}\,D)$$

can be submitted as

```
X = (A ! B)&(C ! D);
```

Using the logical functions you can also specify **branched expressions**.

Example:

For instance, the branched expression

$$y = \begin{cases} 10 & \text{for } z \le 2 \\ 3z + 4 & z > 2 \end{cases} \tag{4.1}$$

can be specified as

```
Y = 10*(Z <= 2) + (3*Z + 4)*(Z > 2);
```

4.5 Functions

The functions allowed in the symbolic expressions are classified in the DYNAST language as

- standard, divided into basic and derived functions
- user-defined polynomial, impulse and tabular functions
- altered, i.e. composed, trimmed, and periodic

4.5.1 Standard functions

The complete list of DYNAST **standard functions** allowed in symbolic expressions is given in Table 4.5.

Table 4.5 Standard functions

Type identifier	Function
`ABS`	absolute value
`EXP`	exponential
`SIN`	sinus
`COS`	cosinus
`TAN`	tangents
`ATAN`	arcustangents
`SINH`	hyperbolic sinus
`COSH`	hyperbolic cosinus
`TANH`	hyperbolic tangens
`LOG`	natural logarithm
`LOG10`	decadic logarithm
`E10`	decadic exponential
`SQRT`	square root
`INT`	integer part
`SGN`	signum
`CTN`	cotangents
`ASIN`	arcussinus
`ACOS`	arcuscosinus
`ACTN`	arcuscotangents
`COTGH`	hyperbolic cotangents

Example:

The mathematical relations

$$x = a \cdot \sin(b^2 + 3)$$
$$y = \frac{3}{\sqrt{a^2 + 10\cos 5(a+b)^2}}$$
$$z = \frac{d^2}{dxdy} x^3 e^{\tan y}$$

can be given to DYNAST in the form of the following symbolic expressions

```
X =  A * SIN(B**2 + 3);
Y = 3/SQRT(A**2 + 10*COS(5*(A + B)**2));
Z = X**3*EXP(TAN(Y))%X%Y;
```

respectively.

4.5.2 User-defined functions

DYNAST allows you to define also your own functions. The type identifiers of such **user-defined functions** are given in Table 4.6.

Table 4.6 User-defined functions

Type	Function
`POLY`	polynomial given by coefficients
`ROOT`	polynomial given by roots
`PULSE`	impulse function
`TAB`	function given by a table
`TAB3D`	function with two arguments given by a table

A user-defined function can be specified by the **function-definition statement**

function / *type* / *list* ;

function

is a user-defined identifier of the function

type

is the function-type identifier in accordance with Table 4.6 placed between slashes ‘`/ /`’

list

is a list of parameters of the function separated by comas ‘`,`’.

The function-definition statement must precede the expressions in which the function is used. Note, please, that the statement definition defines the function *function*$(\cdot)$ independently from its argument. Thus, each user-defined function can be used in the same problem with different arguments.

In a symbolic expression, the user-defined function should be refered to by the string

$$\textit{function}(\textit{argument})$$

function

is a user-defined identifier of the function

argument

is a numeric constant or a symbolic expression enclosed in parentheses ‘( )’

4.5.3 Polynomial function

A real **polynomial function**

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots = k(x - x_1)(x - x_2)\ldots$$

can be submitted using the function-definition statement either in terms of its coefficients or roots.

Specification in terms of coefficients:

type

is specified as `POLY`

list

is the list of the polynomial coefficients

$$a_0, a_1, a_2, \ldots$$

given in an ascending order by real numeric constants or symbolic expressions

Specification in terms of roots:

type

is specified as `ROOT`

list

is the list

$$k, x_1, x_2, \ldots$$

giving the polynomial multiplicative factor k and the polynomial roots x_i as real numeric constants or symbolic expressions. If the specified real polynomial has a pair of complex conjugate roots

$$x_k = \mathrm{Re}\, x_k + j\,\mathrm{Im}\, x_k,\ \bar{x}_k = \mathrm{Re}\, x_k - j\,\mathrm{Im}\, x_k$$

only one of the roots needs to be submitted using the form

$$[\mathrm{Re}\,x_k, \mathrm{Im}\,x_k]$$

where the real and imaginary part of the root is given in the brackets '`[ ]`' as a numeric constant. The sign of the imaginary part can be arbitrary.

The order of the polynomial is determined by DYNAST automatically from the number of the submitted coefficients or roots.

Examples:

The polynomial

$$y = x^3 + x = x(x - j)(x + j)$$

can be specified refering either to its coefficients as

```
F /POLY/ 0, 1, 0, 1; Y = F(x);
```

or, refering to its roots, as

```
G /ROOT/ 1, 0, [0, 1]; Y = G(x);
```

4.5.4 Impulse function

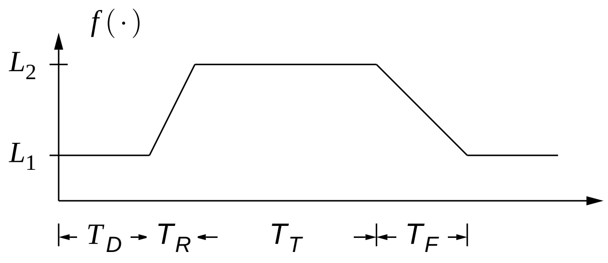


Figure 4.1 Impulse function

Fig. 4.1 shows the shape of the **impulse function** which can be submitted using the function-definition statement where

type

is specified as `PULSE`

list

is the list of the **impulse-function parameters** (see Fig. 4.1)

`L1 =` *value*, `L2 =` *value*, `TD =` *value*, `TR =` *value*, `TT =` *value*, `TF =` *value*

submitted in an arbitrary order

value

is a numeric constant or a symbolic expression specifying the parameter value. By default, $L_2 = 1$, the default values of all the other parameters are zeroes.

Example:

An impulse $y = f(t)$ with a zero rise-time T_R can be submitted as

```
TRAP /PULSE/ L2 = 10, TD = 5U,
    TT = 20U, TF  = 10U;
Y = TRAP(TIME);
```

4.5.5 Tabular function

DYNAST allows also for submitting a function $y = f(x)$ by the sequence of its discrete points using the function-definition statement where

type

is specified as `TAB`

list

is the list of the discrete-point coordinates

$$x_1,\ y_1,\ x_2,\ y_2,\ \ldots,\ x_n, y_n$$

where the argument values x_i and the corresponding function values y_i are given as numeric constants or symbolic expressions. The argument values should be given in the ascending order, that is, there should be $x_i \leq x_{i+1}$ for each i. A functional discontinuity $y_{i-} \neq y_{i+}$ at x_i must be given by two pairs of values, i.e. by the values x_i, y_{i-} and x_i, y_{i+}

If the function argument is outside of the specified points, i.e. if either $x < x_1$ or $x > x_n$, the function value is defined as follows:

$$f(x) = \begin{cases} y_1 - (x_1 - x)\dfrac{y_2 - y_1}{x_2 - x_1} & \text{for } x < x_1 \\ y_n + (x - x_n)\dfrac{y_n - y_{n-1}}{x_n - x_{n-1}} & \text{for } x > x_n \end{cases}$$

, i.e. the first and the last segment of the tabular function is extrapolated to the left and to the right of the interval (x_1, x_n), respectively.

You may replace the list of discrete-point coordinates *list* with statement `FILE=`*filename*. The point coordinates should be then placed in file *filename*`.ftn`. The file has the same format as *list* with the exception that commas separating individual points may be omitted. This allows you e.g. to import some data computed by DYNAST to an input file, just by copying part of DYNAST output file to function file.

Example:

The impulse function from the previous example can be submitted as a tabular function

```
TRAP /TAB/ 0,0, 5U,0, 5U,10, 25U,10, 35U,0;
Y = TRAP(TIME);
F /TAB/ FILE=t;
X = F(TIME);
```

Note the functional discontinuity at `TIME` $= 5\mu$s. The statement on the third line of the example fetches specification of the tabular function from the file `t.ftn`.

4.5.6 Tabular function with two arguments

The tabular function can also have two arguments. The function $z = f(x,y)$ can be submitted by sequence of discrete-point coordinates using the function-definition statement where

type

is specified as `TAB3D`

list

is specially-formated list of points defining the shape of the function:

$$\begin{array}{lccccccl}
 & [& x_1, & x_2, & \dots & , x_n &], \\
y_1, & [& z_{11}, & z_{21}, & \dots & , z_{n1} &], \\
y_2, & [& z_{12}, & z_{22}, & \dots & , z_{n2} &], \\
\vdots & & & & \ddots & & \\
y_m, & [& z_{1m}, & z_{1m}, & \dots & , z_{nm} &]
\end{array}$$

where the argument values x_i and y_j and the corresponding function values z_{ij} are given as numeric constants or symbolic expressions. The argument values should be given in the ascending order, that is, there should be $x_i \le x_{i+1}$ for each i and $y_j \le y_{j+1}$ for each j. A functional discontinuities must be handled similarly as discontinuities of tabular function described above.

You may replace the list of discrete-point coordinates *list* with statement `FILE=`*filename*. The point coordinates should be then placed in file *filename*`.ftn`. The file has the same format as *list* with the exception that commas separating individual points may be omitted.

4.5.7 Altered functions

New functions can be also acquired in the DYNAST language by altering the basic and user-defined functions, i.e. by augmenting or trimming them, or by making the periodic. In such a way, a function

$g(\cdot)$ can be converted into a function $f(\cdot)$ with some additional features. Then, $g(\cdot)$ makes the **kernel function** of $f(\cdot)$.

The **function-alteration statement** takes the form

function / *type* / [*list*,] *altlist*;

function

is a user-defined identifier of the function

type

is the function-type identifier of a basic function (see Table 4.5) or of a user-defined function or of a user-defined function (see Table 4.6). The identifier is placed between slashes '`/ /`'

list

is a list of function-definiton parameters in the case of user-defined function

altlist

is the list of function-alteration parameters given in an arbitrary order and separated by comas '`,`'.

Obviously, the function-alteration statement differs from the function-definition statement just by the additional list of parameters *altlist*. Both statements can be combined into one common statement with the parameter lists *list* and *altlist* mixed freely.

4.5.8 Composed function

A kernel function $g(x)$ can be composed to form a more general function

$$f(x) = A + Bg^E(Cx + D)$$

using the function-alteration statement where

altlist

is the list of the function-composing parameters

`A =` *value*, `B =` *value*, `C =` *value*, `D =` *value*, `E =` *value*

value

is a numeric constant or symbolic expression. The **default values** of the parameters are $A = D = 0, B = C = E = 1$

Examples:

The relation

$$i = I_0(e^{\theta v} - 1)$$

can be submitted as an composed exponential function:

```
FUN /EXP/ A = -1, C = THETA;
I = I0*FUN(V);
```

The function

$$f(x) = 3\left[z(x) + 5z^3(x) - 8z^5(x)\right]^2 - 4$$

in which

$$z(x) = 3x - 1$$

can be submitted as an composed user-defined polynomial function

$$g(z) = z + 5z^3 - 8z^5$$

using the statement

```
F /POLY/ 0, 1, 0, 5, 0, -8, C = 3, D = -1, A = -4,
    E = 2, B = 3;
```

4.5.9 Trimmed function

A kernel function $g(x)$ trimmed and linearly extrapolated outside the interval $L < x < U$ to form the function

$$f(x) = \begin{cases} f(U) + S_U(x - U) & \text{for} \quad x > U \\ g(x) & \quad\quad\;\; L < x < U \\ f(L) + S_L(x - L) & \quad\quad\;\; x < L \end{cases} \tag{4.5}$$

can be submitted using the function-alteration statement where

altlist

is the list of the function-trimming parameters

`L` = *value*, `U` = *value*, `SL` = *value*, `SU` = *value*

value

is a numeric constant or a symbolic expression. The **default values** of the parameters `L` and `U` are L = $-\infty$, U = ∞.

For a finite value of L and U, the corresponding slope S_L and S_U is by default

$$S_L = \left(\frac{df}{dx}\right)_{x=L} \quad S_U = \left(\frac{df}{dx}\right)_{x=U}$$

Thus, the function $f(x)$ is by default continuous at the points $x = L$ and $x = U$.

Examples:

The relation

$$i = I_0(e^{\theta \cdot v} - 1)$$

linearized outside the interval $0 < v < v_1$ in such a way that its derivative is continuous, i.e. that

$$i = \begin{cases} I_0(e^{\theta \cdot v} - 1) & \text{for} \quad \theta < v < v_1 \\ I_0(e^{\theta} - 1) + I_0 \theta e^{\theta} \cdot v & v \geq v_1 \\ I_0 \theta v & v \leq \theta \end{cases}$$

can be submitted as an composed and trimmed exponential function:

```
FUN /EXP/ A = -1, C = THETA, L = 0, U = V1;
I = IA*FUN(V);
```

4.5.10 Periodic function

A function $g(x)$ can be converted into a periodic function $f(x)$ such that

$$f(x) = g(x) \text{for } 0 \leq x < P, \; f(x + k \cdot P) = f(x)$$

where k is an integer and P is a period of $f(x)$, using the function-alteration statement where

altlist

gives the period `P =` *value* where *value* is a numeric constant or a symbolic expression. The default value of the period is $P = \infty$

The exception is the impulse function $g(x)$, which is converted to $f(x)$ so that $f(x) = g(x)$ for $T_D \leq x < P + T_D$

Example:

A user-defined impulse function can be made periodic just by adding the specification of the period into the list of its parameters, for example:

```
F1 /PULSE/ TD=-2,TR=1,TT=2,TF=1, P=5;
Y = F1(time);
```

The same periodic impulse function can be submitted as the altered tabular function:

```
F2 /TAB/ 0,1, 1,1, 2,0, 3,0, 4,1, 5,1, P=5;
Y = F2(TIME);
```

4.5.11 Random number generator

Random variables can be generated using random number generators with a normal or uniform distribution. The random numbers can used as an argument in a symbolic expression (without any prior declaration). The specification of such a random-waveform function is either of the form

$$\texttt{normal}(\textit{mean value}, \textit{variation}, \textit{delta})$$

or

$$\texttt{uniform}(\textit{lower}, \textit{upper}, \textit{delta})$$

This function behaves in fact like a tabular function with the argument `time` defined at time points separated from each other by *delta* s. All parameters – *mean value*, *variation*, *lower*, *upper*, *delta* – must be numeric constants or constant expressions.

Example:

A variable representing a sequence of random numbers with normal distribution and unit variation at time points displaced by 10ms can be submitted as

```
A = NORMAL(0, 1, 10m);
```

The signal may look like this:

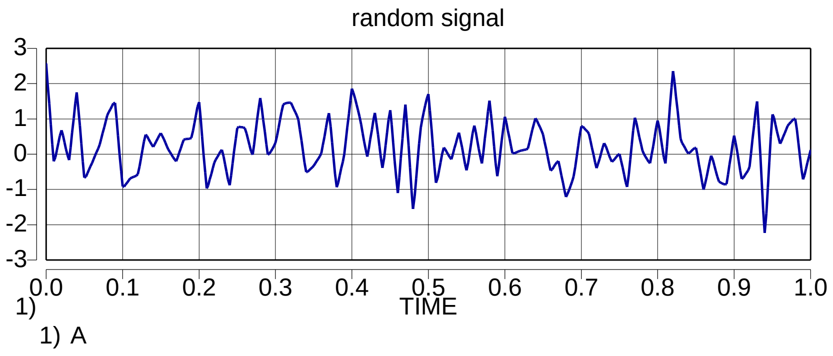


Figure 4.2 A random signal with distribution $N(0,1)$ and sample time 10ms

4.5.12 Events

An occurence of a specific event is indicated by DYNAST during numerical transient analysis by the changes of the **event variable** assigned to a particular event. Until the event occurs, the value of the associated event variable is 10^{32}. At the moment of the event occurence, the event variable drops to the value the `TIME` variable aquired at this moment.

The **event order** is an integer number indicating the number-of-times the expression must become true for the event to occur. The default values of event orders are 1.

Several events can be specified by a statement of the form:

`EVENT` *event* [(*order*)] = *expression*[, *event* [(*order*)] = *expression*...];

event

is a user-defined identifier of an event as well as of the event variable associated with that event. If this identifier is set to `STOP`, the DYNAST run is terminated at the moment of the event occurrence.

expression

is a logical expression which defines the actual event – the event occurs when *expression* becomes true

order

is a positive integer number *n* indicating the event order.

- If the value of *order* is $n > 1$, the event variable changes its value from 10^{32} to the value of the `TIME` variable when its *expression* becomes true for the *n*-th time only, and than it keeps this value constant up to the end of the analysis
- In case *order* is set to ‘0’, the corresponding event variable changes its value to a new `TIME` variable value each time its *expression* becomes true again

Example:

An event `A` representing the moment at which “*a variable VAR1 becomes less than another variable VAR2*” can be submitted by the statement

```
EVENT A = VAR1 < VAR2;
```

The specification of an event *B* implying that “*the event A has already occurred and that the variable Y ascended above the level 1500 for the second time after A has occured*” might look as

```
EVENT B (2) = (A < 1G)*(Y > 1.5K)*(VD.Y > 0);
```

4.5.13 Intervals

By the **interval** between some events we understand the difference between the `TIME` variable values corresponding to two event occurrences.

To specify several intervals we can use the following statement:

`INTRV` *interval* = *expression* [, *interval* = *expression*...] ;

interval

is a user-defined identifier of the interval

expression

is a symbolic expression which defines the `TIME` variable difference between the event occurences

Examples:

Thus, the statements invoking the computation of events `E1` and `E2` together with the interval `I` between them might be given as

```
EVENT  E1 = W < 0.5,  E2 = W < 0.3 ;
INTRV  I = E1 - E2;
```

The width `WP` of a positive overshoot of the variable `VOUT` above the level 1.5 may be specified as follows

```
EVENT  T1 = (VOUT > 1.5)*(VD.OUT > 0),
T2 = (T1 < 1E32)*(VOUT < 1.5)*(VD.OUT < 0);
INTRV  WP = T2 - T1;
```

4.6 Specification of expressions using the wizards

The Expression wizard can be open either directly from the System menu, or from the other wizards in this menu for entering equations, blocks, elements or submodels. The Expression wizard facilitates submitting operators, functions and variable identifiers. After specifying a new variable or a user function, the list of variable identifiers or functions is automatically updated.

To specify an impulse, tabular, polynomial, or altered user-defined function, open the corresponding dialog box from the System menu.

Chapter 5
Submitting basic blocks

5.1 Basic blocks

Comparing with the conventional block-diagram simulators, the variety of DYNAST basic blocks looks rather poor at the first sight. In fact, however, the blocks are very versatile as their constitutive relations can be characterized by symbolic expressions in a very flexible way. In block diagrams processed by the conventional simulators, block inputs must be connected to the outputs of some other blocks. But in block diagrams analyzed by DYNAST the block inputs can be associated with any variables or parameters of the block diagram. The DYNAST block variety includes even implicit blocks the input of which can by directly associated with their output variable. Also unlike the conventional block-diagram simulators, DYNAST has no problems with ‘algebraic loops’ in block diagrams, the diagrams may be even composed from blocks characterized by algebraic relations exclusively.

Besides the one-output basic blocks, the block diagrams analyzed by DYNAST may consist also from multi-input multi-output superblocks stored as library models. The blocks can be combined with explicit or implicit equations and with physical elements.

5.2 Variety of basic blocks

The variety of basic **blocks** ‘built-in’ DYNAST is shown in Table 5.1. Different **block types** are characterized by specific constitutive relations between a **block-output variable** $y(t)$ and **block-input variables** $z_i(t)$. The basic blocks have one output only, but they may have several inputs, except for the transfer, delay and sample&hold blocks.

Explicit blocks are characterized by a constitutive relation which may be linear, nonlinear and time or parameter dependent. Also input-variable derivatives $\dot{z}_i(t) = dz_i(t)/dt$ are allowed as arguments of the relation. If there is no input variable specified in the relation, the explicit block behaves as an autonomous **block source** of its output variable $y(t)$.

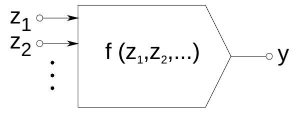
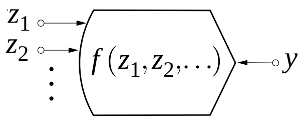
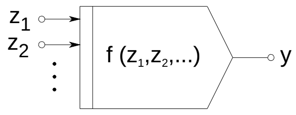
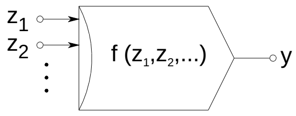
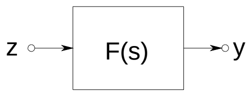
The **implicit block** is more versatile than the explicit block as its constitutive relation is of implicit form. This block makes an exception among the other basic blocks as as also its output variable $y(t)$ and/or its derivative $\dot{y}(t) = dy(t)/dt$ may become an argument of its constitutive relation.

In the case of the **integrators** and **differentiators**, the constitutive relation is assumed to be linear without time derivatives of input variables. The **transfer blocks** are characterized by real rational transfer functions $F(s)$ of the Laplace operator s. It is assumed that $F(s) = K \cdot M(s)/N(s)$ where

$$M(s) = s^m + a_{m-1}s^{m-1} + \ldots + a_0$$

is a numerator polynomial of the transfer function $F(s)$, and

Table 5.1 Basic blocks

Type	Block	Symbol	Constitutive relation
BS	explicit block		$y = f(z_1, \dot{z}_1, z_2, \dot{z}_2, \ldots, t)$
BO	implicit block		$f(y, \dot{y}, z_1, \dot{z}_1, z_2, \dot{z}_2, \ldots, t) = 0$
BI	integrator		$y = \int (k_1 z_1 + k_2 z_2 + \ldots) dt + y_0$
BD	differentiator		$y = \frac{d}{dt}(k_1 z_1 + k_2 z_2 + \ldots)$
BT	transfer block		$Y(s) = F(s) \cdot Z(s), \ \ F(s) = K \frac{M(s)}{N(s)}$
BX	delay block		$y(t) = z(t - t_D)$
BH	S-H block		

$$N(s) = s^n + b_{n-1} s^{n-1} + \ldots + b_0$$

is its denominator polynomial. K is a constant multiplicative factor of the transfer function.

Notice, that in the time-domain the transfer block is characterized by the n-th order ordinary differential equation

$$\overset{(n)}{y} + b_{n-1} \overset{(n-1)}{y} + \ldots + b_0 y = K(\overset{(m)}{z} + a_{m-1} \overset{(m-1)}{z} + \ldots + a_0 z)$$

with the initial conditions $\overset{(i)}{y}_0 = 0, i = 0, 1, \ldots, n-1$.

The **delay blocks** are characterized by the relation $y(t) = z(t - t_D)$, where T_D is the transportation delay of the block. If the transient analysis starts at $t = t_0$, $y(t) = D$ during the first integration step, where D is a user-specified constant or expression.

A **sample&hold block** samples its input signal at specified time points, and keeps the sampled value at its output for the interval between the two sampling instants.

5.3 Basic blocks in block diagrams

In the DYNAST interpretation of **block diagrams** it is assumed that each **block output** is coalesced with a **diagram node**node. If only one block output is coalesced with the node, then the **node variable** is identical to the block output variable.

In general, coalescing of several block outputs is forbidden as it is inconsistent with the block-diagram principles. If, however, outputs of two or more static blocks are coalesced with a node in a block diagram analyzed by DYNAST, the node is assumed to be acting as a **summing node** the node variable of which equals to the sum of the coalesced output variables. The advantage of summing nodes is in simplifying the input data preparation, and also in reducing the number of equations formed and solved simultaneously by DYNAST. But the individual block output variables are not available in the analysis results.

The block inputs can be associated, besides node variables, with any variables or parameters of the submitted block diagram. Just the transfer-block inputs can be associated with nodes only. No block-input variable should be left unspecified as this would result into the error message '`SYSTEM IS SINGULAR`' indicating the singularity of the equations formed by DYNAST automatically for the analyzed block diagram.

5.4 Variables of basic blocks

DYNAST allows for the analysis of block diagrams combined with equations and physical elements. The input variables z_i taking place in the constitutive relations characterizing the blocks can be any variable declared as a solved variable by the '`SYSVAR`' statement as well as any evaluated variable or parameter specified already by an explicit equation.

5.5 Specification of blocks in a diagram

Once a block symbol is placed into a diagram, you can open the dialog box related to the block by pointing the cursor at the symbol and clicking the right button of the mouse. There choose Edit properties and fill in the text fields.

5.6 Specification of blocks in a text file

Specification of a block should be included into the `SYSTEM` section of a problem file. It can be also included in a submodel file. From the System menu, choose Insert block to open the wizard.

Basic blocks are specified in the DYNAST input language by a statement of the form

block [>*type*] *node* [= *expression*];

block

is a user-defined identifier of the specified block

type

is a two-character string indicating the block type shown in Table 5.1. If the identifier *block* is chosen in such a way, that its first two characters coincide with *type*, the string '`> type`' can be omitted.

node

is a user-defined identifier of the node with which the block output is associated

expression

is a numeric constant or a symbolic expression specifying the block constitutive relation with the exception of transfer blocks. If this expression is unity, the corresponding string '`= 1`' can be omitted.

The **ransfer-block specification** differs from the specification of the other basic blocks by the form of the In this case, *expression* is of the form

[*factor*][`*`][*numerator*] [`/`*denominator*]`*`*input*`;`

factor

is a numeric constant corresponding to the multiplicative factor K of the block transfer function $F(s)$. If the value of the multiplicative factor K is unity, the corresponding character string '` 1 *`' can be omitted.

numerator

is an identifier of the numerator polynomial $M(s)$ of the block transfer function $F(s)$. If this polynomial is unity, the corresponding character string '`* 1`' can be omitted.

denominator

is an identifier of the denominator polynomial $N(s)$ of the block transfer function $F(s)$. If this polynomial is unity, the corresponding character string '`/ 1`' can be omitted.

input

is a user-defined identifier of the primary variable with which the block input is associated.

The specification of the transfer-block polynomials has to precede the specification of the transfer block.

The *expression* is of the form

input `AFTER` *delay* [`DEFAULT` *default*] `;`

input

is a user-defined identifier of the input variable of the block.

delay

is a numeric constant or symbolic expression specifying the transportation delay t_D of the block.

default

is a numeric constant or symbolic expression D specifying the output-variable during the first integration step, for which the block input is not defined. The value of D is zero by default.

The *expression* is of the form

block [`> BH`] *output* `=` *input* `PERIOD` *period* [`SHIFT` *shift*] `;`

input

is a user-defined identifier of the block-input node.

period

is a numeric constant representing length of the sampling period T.

shift

is a numeric constant representing the time instant t_1 of the first sample, it is zero by default.

The parameters *period* and *shift* are evaluated only at the beginning of the transient analysis, then they remain constant.

Examples:

The relation

$$u(t) = ae^{bv(t)} + \frac{d}{dt}c$$

can be implemented by a nonlinear explicit block specified as

```
EXP > BS u = a*exp(b*v) + VD.c;
```

or as

```
BSexp u = a*exp(b*v) + VD.c;
```

The same relation can be implemented also by the implicit block

```
BO_EXP u = a*exp(b*v) + VD.c - u;
```

The statement

`BS X = 1;` or `BS X;`

enters a block source generating the unit-step waveform of the variable $x(t)$.

The block

```
BS x = k*time*(time > 0)&(time < t0);
```

generates a ramp waveform of $x(t)$ for $0 < t < t_0$, outside this interval $x(t) = 0$.

Both the integrator

```
INTEGR > BI xOUT = x1 + 3*x2;
```

and the implicit block

```
INTEGR > BO xOUT = VD.xOUT - x1 - 3*x2;
```

implement the relation

$$x_{OUT} = \int (x_1 + 3x_2)dt$$

A transfer block implementing the transformed relation

$$Y(s) = 5\frac{s^2+1}{(s+1-j)(s+1+j)}Z(s)$$

can be specified as

```
M /poly/ 1,0,1; N /root/ 1, [-1,1];
BT Y = 5*M/N (Z);
```

or as

```
M /poly/ 5,0,5; N /root/ 1, [-1,1];
BT Y = M/N (Z);
```

Example:

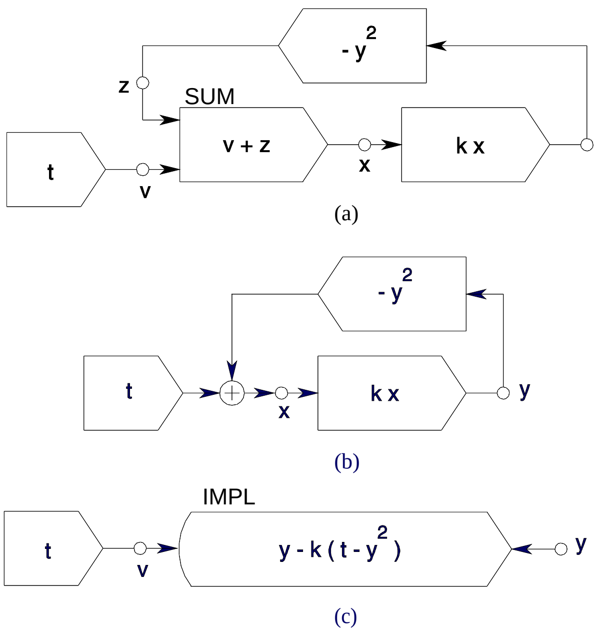


Figure 5.1 Quadratic-feedback block diagram: (*a*) summator, (*b*) implicit-summator, and (*c*) imlicit-block version.

The block diagram shown in Fig. 5.1*a* represents a simple static system with quadratic feedback. The system is excited by the ramp function $v = t$. The block diagram consists of four blocks, the output nodes of which are denoted by the identifiers v, x, y, z. The block diagram structure and its numerical time-domain analysis can be specified by the following input data file:

```
*SYSTEM; *: Quadratic feedback
k = 10;                   :parameter
BSgen v = time;           :signal source
SUM > BS x = v + z;       :summator
BSforw y = k*x;           :feedforward branch
BSback z = - y**2;        :feedback branch
*TR; dc 0 10;             :time analysis
PRINT v, x, y, z;
RUN; *END;
```

In the input data file

```
*: Quadratic feedback - implicit-summator version
*SYSTEM; k = 10;
BSgen x = time;
BSforw y = k*x;
BSback x = - y**2;
*TR; dc 0 10; PRINT x, y;
RUN; *END;
```

corresponding to the block diagram shown in Fig. 5.1*b*, the summator SUM was replaced by an implicit summator. It is formed there by coalescencing the outputs of the blocks BSgen and BSback at the same node x. The output variables of the two blocks are inaccessible, but the number of describing equations decreased by one.

Example:

The following block-diagram specification

```
*: Quadratic feedback - implicit-block version
*SYSTEM; k = 10;
BSgen v = time;
IMPL > BO y = y - k*(v - y**2); :implicit block
*TR; dc 0 10; PRINT v, y;
RUN; *END;
```

requires formulation of two equations only. The corresponding block diagram utilizing the implicit block IMPL characterized by the equation $y - K(t - y^2) = 0$.

Example:

For instance, Bessel's equation can be submitted in the following way utilizing differentiators:

```
N = 1; SYSVAR X; BD XD = X ; BD XDD = XD;
0 = TIME**2*XDD + TIME*XD
  + (TIME**2 - N**2)*X;
*TR; TR 0 10; INIT yD=.5; PRINT y; RUN;
*END;
```

Example:

A delay block delivering the input `A` delayed by 1 s to node `B` can be specified as

```
BX B = A AFTER 1;
```

During the first integration step the output-node variable is zero.

Example:

A sample&hold block sampling the input `A` at points 0.5 s, 1.5 s, 2.5 s etc. and delivering its output stair-like waveform to node `B` can be specified as

```
BH B = A PERIOD 1 SHIFT 0.5;
```

During the first 0.5 s the variable `B` is zero.

Chapter 6
Creating and editing diagrams

6.1 Block and multipole diagrams

Besides equations, models of dynamic systems can be submitted to DYNAST in the graphical form of diagrams consisting from interconnected graphical symbols of multipole models and blocks.

Recollect that blocks represent algebraic relations between their inputs and outputs. The line segments interconnecting blocks are associated with just one variable propagating in one direction only - from a block output to a block input. The multipoles, on the other hand, portray energy interactions of real system components. The line segments interconnecting multipole symbols are associated with two variables propagating in both directions (the products of these two variables represent the flows of energy of the interactions).

In DYNAST, one line segment may support even a vector of variables or variable pairs. Also, you even may to encapsulate a diagram, associate it with a symbol, and use it as a part of another diagram in a hierarchical way.

Once you create a diagram and submit it to DYNAST, the program is capable of

- converting the diagram graphical form into its textual form called netlist
- forming a set of algebro-differential equations underlying the diagram
- solving the equations and plotting the desirable variables

6.2 Creating diagrams

6.2.1 Placing parts into a diagram

You can start creating a diagram after opening the diagram window in DYNSHELL. Another alternative is creating the diagram in OrCAD and importing it then into DYNSHELL.

A **diagram** consists of parts interconnected by line segments and annotated by texts. **Parts** are instances of the following graphical symbols placed into a diagram:

- symbols of two-pole *physical elements* and one-output *basic blocks*
- symbols of submodels, i.e., of general multipoles, superblocks, or their combinations
- symbols for across-variable references, like the electrical ground, the reference mechanical frame, the references for temperature or pressure, etc.

These graphical symbols are stored in symbol libraries. Part's dynamic behavior is either defined in the DYNAST language (this is the case of basic blocks, physical elements and references), or it is described in an independent text file (subsymbol files), or it may be represented by a diagram portraying the internal configuration of the related submodel.

A part can interact with the rest of the diagram via its **pins**. The pins are symbolized by short line segments sticking out from the part symbols. There are scalar and vector pins. A **scalar pin** supports just one variable if it represents a block input or output, or two complementary power variables if it represents a multipole pole. A **vector pin** supports either several block output variables, or several pairs of power variables.

To place a part into a diagram follow these steps:

1. From the Place menu, choose Part.
2. Browse through the symbols in the Place part window.
3. Select a symbol using the left mouse button.
4. Drag the symbol by mouse into the appropriate location in the diagram.
5. Before placing the symbol, you may press the R, X or Y keys to rotate the symbol or to miror it.
6. Using the left mouse button, place the symbol.
7. Unless you want to create another instance of the same symbol, press the the right mouse button. Otherwise, repeat the procedure starting from step 4.

After placing a part, you may set its identifier, parameters and comments using Edit properties of part in the dialog window that opens after double clicking the part.

6.2.2 Interconnecting parts

Pins of the parts placed into a diagram are interconnected by connectors or buses. A **connector** interconnects scalar pins, whereas a **bus**, consisting of several connectors, interconnects vector pins. Both connectors and buses are represented by line segments, the line segments related to busses are more thick then those of connectors. To place a connector or a bus, click the respective button in the Place menu, then click the left mouse button at the diagram location where you want the connection to start, and click it again at the location you want it to end. Unless you want the connection to continue to some other direction, click the right mouse button.

Any of the individual scalar connectors in a bus can be interconnected with a pin or node outside the bus by a **vector entry**. Each of the individual connectors in a bus is associated with an identifier. The vector entry you want to connect to a connector inside a bus must be denoted by the connector identifier. You will find Vector entry in the Place menu. The pins are interconnected either directly or via nodes. A **node**, symbolized by a dot, indicates the interconnection of either at least three scalar pins and/or connectors, or at least three vector pins and/or buses. The diagram editor in DYNSHELL denotes the nodes automatically except the case of crossing of two connectors or buses. In such a case, it is up to the user to decide if there is a node at the crossing or not. In the former case, the user must indicate the interconnection by placing a **junction** symbolized by a small full square at the crossing. Note, that several dots or junctions may symbolize only one node if they are interconnected directly by connectors or buses.

To be able to convert diagrams into a netlist text file, DYNSHELL requires all nodes denoted by identifiers, either numbers or strings of some other characters. For this reason, DYNSHELL numbers the nodes in a diagram automatically. But you can associate any node with an identifier of your choice by placing a **label** at the node. From the Place menu, choose Node Label. You can rotate or mirror the label in a similar way as the parts. Double clicking the Node Label symbol opens the dialog box Label properties with a text field for you to write in the node identifier.

6.3 Editing a diagram

The diagram editor allows for moving, copying and deleting the graphical objects in a diagram, and for changing properties of the objects. The editor can be used also for adding or editing texts in the diagram. To edit a diagram, the diagram editor must be in the *select mode* enabled by choosing **Select** from the Edit menu. Note that all editing operations can be cancelled by clicking **Undo** in the Edit menu.

Selecting objects. To edit an object, select it by pointing the cursor at it and press the left mouse button. To edit a group of objects, select each of the individually while pressing the Ctrl key. To select all objects in a rectangular region, drag the mouse with the left button pressed over the region. You may also select all objects in the diagram by clicking **Select All** in the Edit menu.

Moving objects. Selected objects can be dragged to a new position by the mouse with the left button pressed. By default, the diagram editor tries to preserve the interconnection of the selected parts with the remainder of the diagram. If you do not want the interconnections preserved, hold the Alt key during the dragging.

Copying objects. Selected objects can be copied by dragging them to a new position while pressing the Ctrl key. You may also use **Copy** and **Paste** from the Edit menu – this allows you to copy objects even between different documents.

Deleting objects. To delete selected objects, use **Delete** from the Edit menu, or just press the Delete key.

Editing properties of objects.

Some objects have properties specified by texts. These properties are displayed in the diagram as *tags*. You may edit the properties of an object by double-clicking the object, or by double clicking of one of its tags.

Editing texts in a diagram.

You may place short texts into diagrams. First from the Place menu, choose text note and place the label note into a position in the diagram. Then double click the label note to open the **Text Note dialog** box.

Changing the diagram title.

The diagram *title* should be in agreement with the title of the coresponding problem file. You may change the title after opening the Change Title **Diagram Title dialog** dialog from the Edit menu.

6.4 Synchronization of a diagram with its netlist

You can see the netlist of the open diagram after clicking Open Input Text from the View menu. Now you may work in parallel with both the textual and graphical representation of the system model. If you open both the documents, they are automatically compared by DynShell. If there is a difference between them, you will be prompted about it and asked whether you want to update the activated document to consolidate both documents.

If the activated document is a netlist, it is updated in the following way:

- the title is updated from the graphical document
- the statements present both in the textual and graphical document are updated from the graphical document
- the statements present only in the graphical document are added to the textual document
- the statements present only in the textual document and managed by the diagram editor (i.e., elements, blocks and submodels, but not equations) are removed from the textual document

If the activated document is a diagram, it is updated as follows:

- the title is updated from the textual document
- if only values of a parameter in both documents differ, the parameter value is updated in accordance with the textual document
- if types of blocks or elements in statement differ, the corresponding block or element is removed from the diagram
- the statements present only in the textual document are ignored, but a warning is issued
- the statements present only in the graphical document are removed from it

The synchronization process normally takes place when either textual or graphical document is activated and the direction of synchronization is determined automatically according to the document time stamps. However, you can invoke the synchronization explicitly by selecting Synchronize from Edit menu. You may then select the synchronization direction regardless time stamps of the activated documents.

Chapter 7
Submitting physical elements

7.1 Physical elements

You can submit to DYNAST **multipole model** of a real dynamic system set up directly from physical elements and based on mere inspection of the real system without the necessity to formulate any equations, or to construct any graph or block diagram. Equations characterizing the dynamics of such a physical system model are then formulated automatically by DYNAST in a way respecting the physical laws governing the mutual energetic interactions between the system components.

The physical models analyzed by DYNAST may be linear, nonlinear and time or parameter dependent. The physical elements can be combined with library models of complex system modules as well as with blocks and equations.

7.1.1 Variety of physical elements

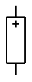
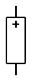
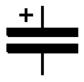
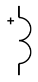
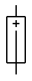
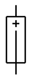
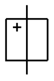
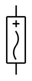
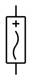
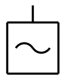
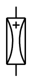
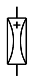
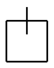
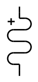
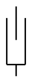
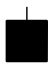
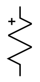
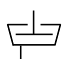
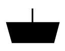
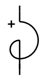
The variety of **physical elements** which is at your disposal in the DYNAST input language is given in Table 7.1. The element **constitutive relations** characterize different **element types** in terms of the element variables, *p* denotes the elements parameters. The **element classes** Y and Z given in Table 7.1 differ from each other by the way in which DYNAST treats their variables (either as solved or evaluated).

Table 7.1 Physical elements

Type	Element	Constitutive relation	Class
J	through-variable source	$i = p$	Y
E	across-variable source	$v = p$	Z
G	conductor or damper	$i = p \cdot v$	Y
R	resistor	$v = p \cdot i$	Z
C	capacitor or mass	$i = p \cdot dv/dt$	Y
L	inductor or spring	$v = p \cdot di/dt$	Z
OA	operational amplifier	idealized	Z
S	ideal switch	$i = 0$ or $v = 0$	Z

Our symbols for through-variable and across-variable sources are shown in Fig. 7.1*a* and *b*, respectively (for the sources, we are using the same symbols in all energy domains). Fig. 7.1*c* shows the symbol of the element representing an idealized operational amplifier. Graphical symbols for the other elements in different energy domains are shown in Table 7.2.

Table 7.2 Symbols of physical-model elements

Type	G	R	C	L
Element	Conductor	Resistor	Capacitor	Inductor
Electric				
Magnetic				
Thermal				
Fluid or acoustic				
Element	Damper		Mass	Spring
Mechanic translatory				
Mechanic rotary				

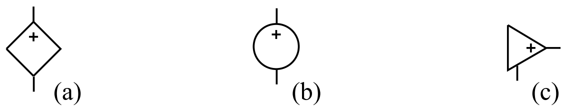


Figure 7.1 (*a*) Through-variable source, (*b*) across-variable source, (*c*) operational amplifier.

7.1.2 Element physical variables

Each of the physical elements is associated with an **element-through variable** *i* and an **element-across variable** *v*. Examples of **physical quantities** corresponding to these variables in several energy domains and their physical units are given in Table 7.3. These two **element-power variables** complement each other in the sense that their product

$$P(t) = i(t) \cdot v(t)$$

represents the power consumed in the element. (Instead of introducing a new notation for generic through and across variables, we are using the notation common for these variables in the electrical domain.) Table 7.3 gives also the corresponding pairs of **energy variables**, i.e. the through and across variables integrated with respect to time. The vacancies in Table 7.3 correspond to variables which are not used in general.

The criterion for selecting certain physical quantity either as a through or an across variable is based unambiguously on the way in which it is measured. This is illustrated by Fig. 7.2 for a generic physical element. Measurement of the element-through variable *i* requires connecting a measuring instrument ‘in series’ with the element, i.e., the measuring instrument must be included between the element and the rest of the system. Measurement of an element-across variable requires attaching a measuring instrument to

Table 7.3 Pairs of through- and across-variables

Energy domain	Power variables through i	Power variables across v	Energy variables through $\int idt$	Energy variables across $\int vdt$
Electrical	Electrical current A	electrical voltage V	electrical charge C	flux linkage $V.s$
Magnetic	magnetic flow rate Wb/s	magnetic voltage A	magnetic flow Wb	
Thermal	entropy flow W/K	temperature K	entropy J/K	
Fluid or acoustic	volume flow m^3/s	pressure N/m^2	volume m^3	pressure momentum $N.s/m^2$
Translatory	force N	velocity m/s	momentum $N.s$	displacement m
Rotary	torque $N.m$	angular velocity rad/s	angular momentum $N.m.s$	angular displacement rad

the element ‘in parallel’ without intervening into its interconnection with the system. Besides this, each pole is associated with a **pole across variable**, i.e., with an across variable of the pole with respect to an **across-variable reference**. Measurement of pole-across variables is shown in Fig. 7.2*c*.

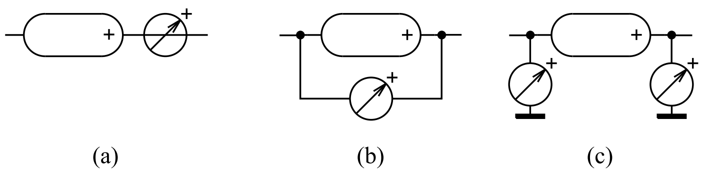
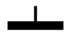
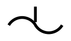
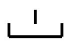
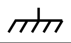


Figure 7.2 Measurement of (*a*) element-through variables, (*b*) element-across variables, and (*c*) pole-across variables.

Table 7.4 gives typical across-variable references and their graphical symbols in several energy domains. In the case of thermal, fluid or acoustics capacitors as well as translatory or rotary masses, the element-across variables must be always considered with respect to a related reference. In other words, one pole of these two-pole elements must be always associated with a reference. The graphical symbols of these elements given in Table 7.2 do not show the references, however, just to simplify the notation.

Table 7.4 Across-variable references

Energy domain	Reference across variable	Reference symbol
Electrical or magnetic	electrical-ground voltage	
Thermal	zero-point on a temperature scale	
Fluid or acoustic	free-atmosphere pressure	
Mechanical	absolute-frame velocity	

7.1.3 Parameters of physical elements

Table 7.1 shows that in the case of a source, the parameter p corresponds directly to the source nominal variable, either i or v. Relations in terms of i and v as well as physical dimensions and units for the parameters p of the other physical elements in different energy domains are given in Table 7.5.

Table 7.5 Constitutive parameters of physical elements

Element	`G`	`R`	`C`	`L`
p	$\frac{i}{v}$	$\frac{v}{i}$	$\frac{i}{dv/dt}$	$\frac{v}{di/dt}$
Electrical	conductance S	resistance Ω	capacitance F	inductance H
Magnetic	conductance Ω	resistance S	permeance H	
Thermal	entropic conductance W/K^2	entropic resistance K^2/W	entropic capacitance J/K	
Fluid or acoustic	conductance $m^3/(Pa.s)$	resistance $Pa.s/m^3$	capacitance m^3/Pa	inertance $Pa.s^2/m^3$
Mechanical translatory	damping $N.s/m$		mass kg	compliance m/N
Mechanical rotary	torsional damping $N.m.s/rad$		moment of inertia m^2kg/rad	torsional compliance $rad/(N.m)$

In the case of **ideal elements**, the parameter p is constant. The elements given in Table 7.1 need not be ideal, however. DYNAST allows also for **nonlinear** as well as for general **controlled elements** the parameters of which are functions of variables or parameters of some other elements, equations or blocks.

In general, the parameter p of a controlled nonlinear and time-variable element can be given by an expression of the form

$$p = f(z_1, z_2, \ldots, \dot{z}_1, \dot{z}_2, \ldots, t) \tag{7.1}$$

where $z_1, z_2, \ldots$ are **element controlling variables** or **parameters**, $\dot{z}_1, \dot{z}_2, \ldots$ are their time derivatives, and t denotes time.

The OA-type element shown in Fig. 7.1*c* and representing an ideal **operational amplifier** is considered as a physical element controlled by the input voltage v_{in}. In this case, the element-through and element-across variable corresponds to the amplifier output current i and output voltage v, respectively. As this operational-amplifier model is assumed to be ideal, its voltage-amplification factor is infinite. The assumption that the values of system variables are finite implies that $v_{in} = 0$. Therefore, the polarity of v_{in} is irrelevant.

7.1.4 Element and variable orientation

To simplify the determination of element-variable polarities in multipole diagrams as much as possible, we are assuming that the variable orientation is related unambiguously to the orientation of the physical elements. Since the polarity references for through variables and across variables can be assigned to each element in an arbitrary way, they can be combined into one reference and fixed to each element symbol once for ever. (In the case of elements with asymmetric characteristics such element orientation is mandatory.)

To be able to assign an **element-polarity reference** to each of the elements, we are associating one of their two poles with the $+$ sign, and the other one with the $-$ sign. In the element symbols given in Fig. 7.1 and in Table 7.2, the $-$ sign is not shown, however, in order to simplify the notation. In the case of the elements with an asymmetric symbol even the $+$ sign is omitted for the same reason, but it is always assumed to be associated with the pin shown in Table 7.2 in the upper position. The pins of the one-pin symbols are always the $+$ sign pins.

To interrelate the reference of a physical element and the references of its variables in a multipole diagram, we are adopting the following **Element Polarity Convention**:

1. The element-across variable $v = v_+ - v_-$, where v_+ is the pole-across variable of the element +pole, and v_- is the pole-across variable of the –pole of the element. Thus, the value of the element-across variable v is positive if $v_+ > v_-$, otherwise v is negative.
2. Then the positive polarity of the element-through variable i is determined by the assumption that the power P consummed (dissipated or accumulated) in the element is positive, i.e., $P = i \cdot v > 0$.

7.1.5 Arrow convention for non-mechanical variables

This Element Polarity Convention is fully sufficient for the unambiguous determination of element-variable polarities. In some situations, however, it might be useful to denote the variable orientation by arrows, and we also need the arrows to indicate the variable orientation in geometric schemes of the modeled real systems. In such cases, we are using arrows with a full arrowhead for through variables, and arrows with an empty arrowhead for across variables. If the actual polarity of any of the variables is opposite with respect to the direction of its arrow, the variable value is taken as negative. Unfortunately, however, the traditional conventions for the arrow orientation in the mechanical-energy domain are different from those common in the non-mechanical domains (like electrical, magnetic, thermal, fluid or acoustic).

Traditionally, it is assumed in the non-mechanical domains that the arrow indicating the assumed positive polarity of through variables shows the direction of the actual flow of a medium through the components (i.e., the flow of mass particles in the case of fluid or acoustic domain, the flow of positively charged particles in the case of electrical domain, etc.). An unambiguous arrow association with across variables is not so well established in the non-mechanical domains, arrows pointing at the direction of both increasing and decreasing across variables are commonly used.

Choosing the latter possibility, and taking into consideration our Element Polarity Convention, we are arriving at the following **Arrow Convention for Non-Mechanical Variables** in multipole diagrams:

- The empty-head arrow indicating the assumed positive polarity of a pole-across variable is pointing always away from the pole towards the reference.
- The empty-head and full-head arrows indicating the assumed positive polarity of the across and through variable of an element, respectively, are pointing both away from the +pole to the −pole of the element.

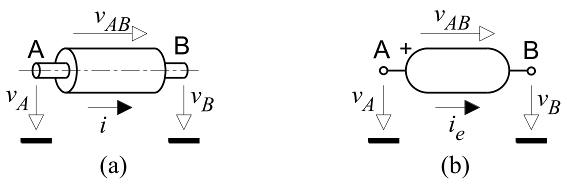


Figure 7.3 (*a*) A generic non-mechanical physical element, (*b*) a corresponding two-entry real component.

Fig. 7.3*a* shows a generic symbol for a physical element from a non-mechanical energy domain. Once the +pole of the element was denoted by the identifier A, the assumed positive polarities of all the across and through variables associated with the element and with its poles are set by our Element Polarity Convention and there is no need for denoting the polarities by any arrows.

Let us now assume that the element models with a sufficient accuracy the real component shown in Fig. 7.3*b*, and that each of its two entries A and B corresponds to the element pole denoted by the same identifier. If we now want to have the orientation of all the variables of the component in a full agreement with the orientation of the corresponding element variables, we must assume the component positive polarities shown by arrows drawn in conformity with the Arrow Convention for Non-Mechanical Variables. If, however, any of the variable polarities will associated with the component in the opposite way, also the sign of the variable will be opposite to the sign of its element counterpart.

Note, that the through variable i_e entering the component via the entry A and leaving it via the entry B corresponds to the element-through variable. The across variables v_A and v_B of the component entries with respect to a reference correspond to the element pole variables, and the across variable v_e of the entry A with respect to the entry B corresponds to the element-across variable.

7.1.6 Arrow convention for mechanical variables

In the case of mechanical variables, the arrows of across variables – velocities – are commonly chosen in agreement with direction of the motions with which they are associated. Similarly, the arrows of through variables – forces or torques – show direction of the motion which they are causing. Thus, the orientation of mechanical variables must be always considered with respect to a coordinate system.

Sticking to this tradition, and taking into consideration our Element Polarity Convention, we are arriving at the **Arrow Convention for Mechanical Variables** in multipole diagrams:

- The empty-head arrow indicating the assumed positive orientation of a pole-across variable points always towards the pole away from the reference-frame symbol.
- The empty-head and the full-head arrows indicating the assumed positive polarity of the across and through variable of an element, respectively, are both pointing towards the element +pole away from its −pole.

Fig. 7.4*a* shows a generic symbol for a physical element from the mechanical translatory domain. The symbol is augmented by arrows showing positive polarities of the element velocity $\dot{x}_e$, the element force F_e, and the pole velocities $\dot{x}_A$ and $\dot{x}_B$ in conformity with the Arrow convention for Mechanical Variables taking into consideration the element orientation (this time, the pole B_x was taken as the +pole).

Let us assume that the element models a real component shown in Fig. 7.4*b*. The component translatory entries A and B, moving along the axes *x* 'painted' on the mechanical reference frame, correspond to the element poles A_x and B_x, respectively. The orientation of the axes *x* as well as of all the component variables shown by arrows in Fig. 7.4*b* is in conformity with the orientation of the element in Fig. 7.4*a* modeling the component.

Obviously, the element velocity $\dot{x}_e$ corresponds to the relative velocity $\dot{x}_{BA}$ of the entry B, corresponding to the element +pole, with respect to the entry A. The element force F_e corresponds to the external force counterbalancing the component internal forces by stretching it.

Note, that we are assuming that the *x*-axes direction determines the positive orientation of all the forces and velocities (as well as deflections and accelerations) associated with the motion of the component terminals.

Similarly, Fig. 7.4*d* shows a real component with two energy entries rotational the mechanical entries of which rotate around the *x*-axis fixed to the reference frame. The corresponding physical element with arrows indicating the assumed positive directions of its variables is given in Fig. 7.4*c*. We assume again that once the positive direction of this rotation has been chosen, it determines the positive orientation of all the variables associated with the rotary motion (torques as well as angular velocities, angular deflections, angular accelerations, etc.). Thus the empty arrow indicates the positive direction of the relative angular velocity $\dot{\varphi}_e$ of the component terminals, the full arrow denotes the positive direction of the elementtorque τ_e counterbalancing the external torques acting on the component.

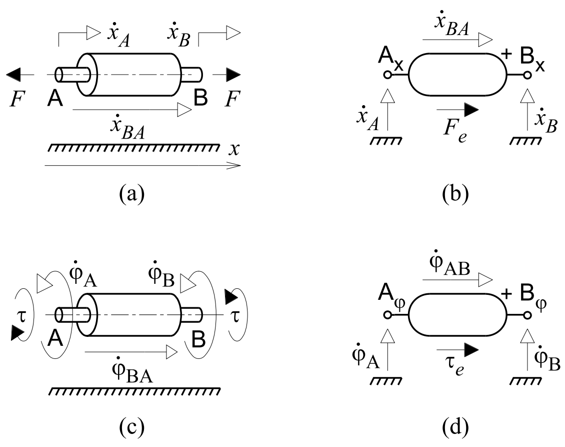


Figure 7.4 Generic (*a*) translational and (*c*) rotational physical elements, (*b*) and (*d*) the corresponding real two-entry components, respectively.

7.1.7 Interaction of physical elements

So far, we have discussed properties of individual physical elements. In this section we will explain the way in which the elements can be interrelated to form the physical model of a complete system.

The physical model of a system model can be displayed graphically in the form of a multipole diagram set up from symbols of the physical elements the poles of which are interconnected by line segments. The line segments represent *ideal connections* which neither dissipate, nor accumulate any energy.

In a system model analyzed by DYNAST, physical elements can be interacting in the following ways:

- via system model nodes
- in a common series configuration
- via inductive coupling

Besides this, the element parameters can be controlled by variables or parameters of some other elements, blocks or equations.

7.1.8 Node interactions

Each of the nodes of a physical system model is associated with a node-across variable, i.e., with the across variable of the node with respect to a reference node of the system model. As the reference node represents the across-variable reference in the related energy domain, its node variable is always zero.

Let us assume that poles of k elements are coalesced with the node N. Than, according to Postulate of Continuity,

$$\pm i_{N1} \pm i_{N2} \ldots \pm i_{Nk} = 0$$

where i_{Nj} is the through variable of the j-th of the k elements. The $+$ sign ($-$ sign) is valid if the element $+$pole ($-$pole) is coalesced with the node N.

At the same time,

$$v_N = v_{p1} = v_{p2} = \ldots = v_{pk}$$

where v_N is the node-across variable of the node N, and v_{pj}, $j = 1,2,\ldots k$ is the pole-across variable of the j-th-element pole coalesced with the node N.

Thus, if the +pole of an element is coalesced with the node N, and the –pole of the same element is coalesced with the node M, then the element-across variable

$$v_e = v_N - v_M$$

where v_N and v_M is the node-across variable of the node N and M, respectively. This is in agreement with Element Polarity Convention as well as with Postulate of Compatibility.

7.1.9 Series configuration

If several Z-class elements form in the system model a **series configuration** without any branching, DYNAST allows for submitting all the elements in a simplified way without a introducing nodes between the elements. Only the nodes between which the whole series is placed are considered. One of the elements in the series, chosen as the **series-representative**, is submitted as connected between these two nodes. Then the other elements in the series are submitted referring to the series-representative element. Besides simplifying the model description, this option also reduces the number of equations formulated and solved by DYNAST.

Let us assume that in a series of k elements the first element is taken as the series representative. Then the through variable of the series i_S is identical to the through variable i_{S1} of the first element in the series, and, in conformity with Postulate of Continuity,

$$i_S = i_{S1} = \pm i_{S2} = \ldots = \pm i_{Sk}$$

where the through variable i_{Sj} of the j-th element in the series, $j = 2,3,\ldots,k$ is taken with the $+$ sign (– sign) if the polarity of the j-th element is in agreement with (oposite to) respect to the polarity of the first element in the series.

In the same time, in conformity with Postulate of Compatibility, the across variable of the series

$$v_S = v_{S1} \pm v_{S2} \ldots \pm v_{S2}$$

where the signs are related again to polarities of the individual elements in the series with respect to the polarity of the first element.

7.1.10 Inductive coupling

DYNAST allows also to enter mutual **inductive couplings** between several inductors, i.e., L-type elements. The inductive coupling between inductors of the self-inductances L_a and L_b can be characterized either by their **mutual inductance** M_{ab}, or by the **inductance-coupling factor**

$$K_{ab} = \pm M_{ab} / \sqrt{L_a L_b}$$

The mutual inductance M_{ab} as well as the inductance-coupling factor K_{ab} between inductors L_a, L_b is positive if the + poles of the inductors correspond both to the 'wounding beginnings' (or both to the 'wounding ends') of the modeled coils.

7.1.11 Parameter control

The constitutive parameters of the submitted elements can be specified by expressions as functions with arguments corresponding to variables of some other elements in the system model. The arguments can be also defined by blocks or equations.

Note please, that the sources can be also used as **ideal measuring instruments** or **ideal sensors**. A zero-value through-variable source (as well as any Y-class element) behaves in fact as an **open connection**. When connected between two nodes, it can be used for 'sensing', or 'picking-up', the across variable between the nodes. Similarly, a zero-value across-variable source (as well as any Z-class element) behaves as a **short connection**. When connected in a series with an element, it can be used for 'picking-up' the element-through variable.

7.2 Specification of physical elements in a diagram

Once a physical element is placed into a diagram, you can open the dialog box related to the element by pointing the cursor at the symbol and clicking the mouse right button. There choose Edit properties and fill in the text fields.

7.3 Specification of physical elements in a text file

Specification of a physical element should be included into the SYSTEM section of a problem file. It can be also included in a submodel file. From the System menu, choose Insert Element to open the wizard.

7.3.1 Node-to-node element interaction

element [> *type*] $+$*node* [- $-$*node*] [= *expression*];

element

is a user-defined identifier of the specified element

type

is a one- or two-character string indicating the element type in accordance with Table 7.1. If the *element* identifier is chosen in such a way, that its first characters coincide with its *type*, the string '> *type*' can be omitted.

$+$***node*** **and** $-$***node***

are identifiers of the diagram nodes between which the specified element is situated. If the element is submitted in such a way that its +pole coincides with the $+$*node* the variables of the element obey the Element-Variable Polarity Convention. Otherwise, the polarities of all the element variables are opposite.
Each reference node must be denoted by the character '0'. (Note, that merging of several reference

nodes in no way effects the element variables.) If *–node* coincides with the reference node, the corresponding string '`- 0`' can be omitted.

parameter

is a numeric constant or a symbolic expression specifying the element constitutive parameter in accordance with Table 7.1. In case the parameter is the unity, the corresponding string '`= 1`' can be omitted. The parameter of an ideal switch must be a logical expression. The switch is closed, when the expression is true, or open otherwise.

7.3.2 In-series element interaction

A Z-class element which is in series with another Z-class element submitted already and considered as the **series-representative**, can be submitted in the following way:

element [> *type*] - *series* [= *expression*];

element

is a user-defined identifier of the specified Z-class element

type

is a one- or two-character string indicating the element type in accordance with Table 7.1. If the *element* identifier is chosen in such a way, that its first characters coincide with its *type*, the string '> *type*' can be omitted.

series

is the identifier of the Z-class element in the series chosen as the series-representative and submitted already in the node-to-node fashion assuming that its poles are coalesced with the nodes between which the series is situated.

parameter

is a numeric constant or a symbolic expression giving the value of the specified element constitutive parameter in accordance with Table 7.1. In the case the value is unity, the corresponding string '`= 1`' can be omitted.

Example:

For instance, specification of a series interconnection of four elements between nodes 13 and 0 might look as

```
E6 13 = 10; R7 - E6 = .1K; R - E6 = 1K;
L8 - E6 = 2M;
```

or as

```
RI7 13 = .1K; E6 - RI7 = 10; R - RI7 = 1K;
L8 -RI7 = 2M;
```

but also as

```
BR > L8 13 = 2M; E6 - BR = 10;
R7 - BR = .1K; R - BR = 1K;
```

7.3.3 Inductive couplings

coupling [> *type*] *inductor* - *inductor* [= *parameter*];

coupling

is a user-defined identifier of the specified inductive coupling. The first character of the identifier *coupling* should be either `M` for the mutual inductance or `K` for the coupling factor of the inductive coupling. `M` stands for for the case when the coupling is specified by the value of its mutual inductance, whereas `K` indicates, that the inductance-coupling factor is given.

inductor

is the identifier of one of the `L`-type elements participating in the inductive coupling

parameter

is a numeric constant or a symbolic expression giving the value for the mutual inductance or coupling coefficient of the appropriate polarity. In case the value is the unity, the corresponding string ‘`= 1`’ can be omitted.

Example:

Three terminals representing closed loops with a mutual inductive coupling inside the second loop and another one between the second and third loop may be specified as

```
: 3rd loop
L 0 = 20M; OUT > R-L = 1.5K; M L-L2 = 1M;
: 1st loop
IN > E 0; RI-IN = 10.;
E 0 = 50*I.RI; R2-E = 1K; L1-E = 8M;
: 2nd loop
L2 > L-E = 3M; K L1-L2 = - 0.95M;
: 3rd loop
L 0 = 20M; OUT > R-L = 1.5K; M L-L2 = 1M;
```

The analysis of the terminal diagram specified in this way requires formulation and solution of three equations only.

Chapter 8
Submitting and creating submodels

8.1 Multipole and superblock submodels

When setting up a model of your system to be analyzed by DYNAST, you are not limited to the basic blocks and twopole physical elements or equations only. For complex components and subsystems, you can also use **submodels** in the form of multi-input multi-output superblocks and multipoles. Each submodel is stored in an independent data file which can be used in a system model wherever it is needed.

The dynamic features of a submodel can be defined by a block and/or multipole diagram combined eventually with a set of algebro-differential equations. The submodels can even be nested in a hierarchical way, i.e., they themselves can be built out of submodels.

8.2 Submitting ready-made submodels

8.2.1 Specification of submodels in a diagram

Once the symbol of a submodel is placed into a diagram, you can open the dialog box related to the submodel by pointing the cursor at its symbol and clicking the mouse right button. There choose Edit properties and fill in the text fields.

8.2.2 Specification of submodels in a text file

Specification of a submodel should be included into the `SYSTEM` section of a problem file. It can be also included in a text file of a submodel. From the System menu, choose Insert Submodel to open the wizard.

A submodel can be specified using the following statement:

[*modul* >] `@`*model interface* [`-` *interface*. . .] [`/` [*parameter* =] *value* [`,` [*parameter* =] *value*. . .]]`;`

modul

is a user-defined identifier of the modeled system component

model

is the identifier of the submodel used for the component, it is preceded by the character ‘`@`’ without any space.

interface

is a user-defined identifier of a system node or of a system branch – in the later case the identifier is preceded by the string ‘`I.`’. The *interface* identifiers are separated by the ‘`-`’ or ‘`,`’ characters.

Identifiers of the system nodes and branches can differ from the identifiers of the library-model interface nodes and branches. However, both lists must contain the same number of identifiers ordered in a way corresponding to their mutual interaction. Thus neither the system reference-node identifiers '`0`' can be omitted in the list. If a library-model node or branch does have its counterpart in the system, it can be created by adding a fictitious block or two-pole element to the system.

parameter

is an identifier of a library-model external parameter. The list of the library-model external parameters is separated from the list of its *interface* identifiers by a slash '`/`'.

value

is a numeric constant or a symbolic expression specifying the value of the corresponding external library-model parameter. If *value* is not specified, the parameter acquires its default value specified within the library-model file (or zero, if neither the default value has been specified).

Note:

If the external parameters and their values are submitted in the same order as in their list within the submodel specification, the strings '*parameter* `=`' can be omitted.

The same submodel can be used for several different components in a system. Submodels can be also used in specifications of other submodels, they can be even nested in this way up to 8 hierarchical levels. The statement for using a submodel in a model specification is the same as the statement for using it in the specification of a system.

Example:

A system in the form of a cascade of two double RC differentiating electrical circuits can be submitted using the sub`CRCR` model in the following way (see the data file of the `CRCR` submodel in the following section):

```
*SYSTEM; E 1; RLOAD 3 = 100;
CIR1 > @CRCR  1,2,3;
CIR2 > @CRCR  2,0,2 / 10k,.1u;
```

The first submodel `CIR1` is called without external parameter specification. It means that its parameters retain their default values $a = 10^3$ and $b = 10$. The second submodel identified as `CIR2` acquires the parameter values $a = 10^3$ and $b = 10^{-7}$.

The system of two ideal coupling or transformer submodels `TRAFOID` in a serio-parallel combination can be given as (see the data file of the `TRAFOID` submodel in the following section):

```
*SYSTEM; E 1; RILOAD 0 = 100;
TR1 > @TRAFOID  1-0,I.RILOAD / N=50;
TR2 > @TRAFOID  1-0,I.RILOAD / N=-20;
```

From the system specification you can refer to the following library model variables:

- `V.`*modul.node* ... node-across variable
- `V.`*modul.element* ... Y-class element-across variable
- `I.`*modul.element* ... Z-class element-through variable

8.3 Submitting new submodels

8.3.1 Specification of submodel properties using the diagram editor

Once the symbol of a submodel is placed into a diagram, you can open the dialog box related to the submodel by pointing the cursor at its symbol and double clicking the left mouse button. There choose Edit properties and fill in the text fields.

8.3.2 Specification of a new submodel using the text editor

Each submodel is to be stored in an individual text file *model*`.MOD` of the following structure:

model interface [`-` *interface*...] $\big[$`/` *parameter* [`=` *value*] $\big[$`,` *parameter* [`=` *value*]...$\big]$ $\big]$`;`
⋮
model specification statements
⋮
`EO@;`

model

is a user-defined identifier of the submodel.

interface

is a user-defined identifier of a library-model node or branch via which the submodel can interact with the rest of the system (the reference node ‘`0`’ need not be given here). The *interface* identifiers are separated by the ‘`-`’ or ‘`,`’ characters.

parameter

is a user-defined identifier of a submodel external parameter the value of which can be specified outside the submodel file. The list of the library-model external parameters is separated from the list of its *interface* identifiers by a slash ‘`/`’.

value

is a numeric constant specifying the default value of the related external parameter (the parameter acquires this value if its value is not specified outside the library-model file). If the string ‘`=` *value*’ is not specified for a parameter, its default value is considered zero.

model specification statements

are identical to the statements for entering equations, blocks and twopole elements into the SYSTEM section of the problem file. From within a library-model specification you can refer to the following system variables or parameters:

- across variables of interface nodes
- through variables of interface branches
- library-model external parameters
- internal independent variables (like `TIME`, `TEMP`, `FREQ`)

`EO@`

is the string terminating the library-model specification.

Examples:

A submodel `CRCR` of a double-RC differentiating circuit

```
CRCR IN-OUT, REF / a=1k, b=10;
C1 IN-INT = b; C2 INT-OUT = b/2;
R1 INT-REF = a; R2 OUT = b/2;
EO@;
```

`IN`, `REF` and `OUT` are interface nodes of the model, `INT` is its internal node. $a = 10^3$ and $b = 10$ are external parameters and their default values.

Let us consider a ball moving freely along the vertical y-axes. The dynamics of its motion can be

```
BODY A, Y / m=1, k=1;
mass > C A = m; gravity > J A = temp*m;
air > G A = abs(V.A)*k; BI y = v.A;
EO@;
```

A primary variable specified by an implicit primary equation within the model description cannot be referred to from outside of the model. This problem can be overcome, however, by replacing the primary equation by an operation block with the output node identifier identical to the identifier of the given variable. Similarly, replacing a secondary equation by an explicit block allows for referring to the corresponding secondary variable.

```
BODY A, Y / M=1, K=1;
BO y = VD.y - v.A;
BO A = VD.A*m + sgn(v.A)*k*v.A**2 + temp*m;
EO@;
```

The submodel `TRAFOID` is stored in the file `TRAFOID.MOD` and represents an ideal transformer

```
TRAFOID A-B,BR / N=1;
E-BR = (V.A-V.B)*N; J A-B = -I.BR*N;
EO@;
```

`A`, `B` are the interfacing node identifiers and `BR` is the identifier of the interfacing branch.

The file *model*`.MOD` of a submodel *model* can be created and included directly into the appropriate directory using an ASCII editor. But it is also possible to include the fail there automatically assuming that the model description has been entered within the SYSTEM section of a problem file preceded by the `DEFMAC` command. This option is advantageous for ‘debugging’ of new submodels as will be shown in the next section.

Example:

After processing the following input file by DYNAST the submodel files `CRCR.MOD` and `TRAFOID.MOD` are created and included into the directory reserved for this purpose automatically.

```
*SYSTEM;
DEFMAC
CRCR IN, REF, OUT / a=1k,b=10;
R1  INNER-REF = a; R2  OUT-REF = b/2;
C1 IN-INNER = b; C2 INNER - OUT = b/2;
EO@;
DEFMAC
TRAFOID A-B,BR / N=1;
E-BR = (V.A-V.B)*N; J A-B = -I.BR*N;
EO@;
RUN; *END;
```

8.3.3 Specification of a new submodel using the diagram editor

You may want to encapsulate a diagram so that you could use it as a part of another diagram. Then you would need to specify which of the encapsulated diagram nodes should become associated with the new part pins. Such nodes must be labeled by the External port symbol from the Place menu.

To develop the corresponding submodel file follow these steps:

1. In the Edit menu press the Is model button.
2. From the View menu, choose Open input to open the submodel textual file.
3. When DYNSHELL prompts you that the file is out-of-date press Yes to have it updated.
4. Specify the submodel external parameters using the text editor.

You can see the textual representation of the submodel diagram press **Submodel** in the Edit menu and choose Open Input Text from the View menu. Now you may work in parallel with both the textual and the graphical representation of the submodel. If you switch between the documents, they are automatically compared and updated by DYNSHELL. If there is a difference between the activated and the corresponding document, you will be prompted about it and asked whether you want to update the activated document to consolidate both documents.

Then you should associate the submodel text file with a graphic symbol. To associate it with a new symbol, open an existing or new symbol library from the DYNSHELL File menu, design the symbol using the symbol editor, and add the symbol to the library.

8.3.4 Designing graphical symbols for submodels

Editor of graphical symbols.

The symbol editor allows you to design and edit libraries of graphical symbols that can be used in the diagram editor (see chapter 6). For each symbol you may edit its size, appearance, pins and other properties.

The window of the symbol editor is divided into two panes. The left pane displays thumbnails of all symbols in the library, and allows you to manipulate the library on the symbol level. The right pane displays the selected symbol, and allows you to edit its properties.

Manipulating symbols in the library.

Manipulation with symbols can be done in the left pane of the symbol editor window. You may add new symbols using **New symbol** from the Edit menu, create a duplicate of the selected symbol using **Duplicate symbol**, or delete the selected symbol using **Delete**. You may also change the order of symbols in the library by dragging a symbol with the mouse, or duplicate symbols by dragging a symbol while pressing the Ctrl key.

All operations on the symbol level can be undone by **Undo** from the Edit menu, but be sure that the left pane of the symbol editor is selected, otherwise operations done in the right pane will be undone. It is also possible to edit properties of a symbol by double-clicking its thumbnail in the left pane.

Editing a symbol.

To create a valid symbol, you should define its

- size (Bounding Box)
- appearance (by placing graphical elements)
- pins
- properties (name, comment, shortcut and DYNAST statement)

Adding new graphical elements.

You may use the Place menu to add new graphical elements to a symbol; you may add lines, rectangles, polygons, circles, arcs, and pins.

To place a line, polygon or filled polygon, use **Polyline**, **Polygon** or **Filled Polygon**, respectively. Click for each endpoint of the line or polygon. To start a new line or polygon, press the right mouse button.

To place a rectangle or a filled rectangle, use **Rectangle** or **Filled Rectangle**, respectively. Define the positions of the rectangle corners by a double click.

To place a circle or a filled circle, use the **Circle** or **Filled Circle**, respectively. Use a double click to define position of the center of the circle and its radius.

To place an arc, use **Arc**. Use a mouse click to define the center and two endpoints of the arc. Between the endpoints the arc will appear in a clockwise direction.

To place a text note, use **Text** from the Place menu. Use mouse clicks to place as many instances of the text as you need; when finished, press the right mouse button. If you double-click the text with the mouse, you may change its properties in the **Text Properties dialog**.

Adding new pins.

To place a pin, use **Pin** from the Place menu. Use mouse click to define the “active” endpoint of the pin – the “passive” endpoint will be always at the bounding box of the symbol. When you fix the position of the pin, the **Pin Properties dialog** appears. Here you have to fill-in name and type of the pin. The pin can be either scalar – then you should fill-in only the type of the pin, or vector – then you should fill in a vector of pin names and corresponding types.

Editing existing objects.

The symbol editor provides features for editing the symbol, i.e. for moving, copying and deleting objects, and for changing properties of pins. To edit a symbol, the symbol editor must be in the *select mode*. This mode can be enabled using **Select** from the Edit menu.

You may use the mouse to select, move and copy the objects in the symbol body in the same way as in the **diagram editor**. Moreover, if you select a single object, you may modify its shape by dragging its manipulation points (small color boxes). If nothing is selected, you may change the size of the bounding box by dragging its manipulation point in its lower-right corner.

Properties of pins and texts may be changed by double-clicking them with the mouse.

All editing operations can be undone by **Undo** from the Edit menu, but be sure that the right pane of the symbol editor is selected, otherwise operations done in the left pane will be undone.

Editing properties of the symbol.

You may edit the properties of the symbol using the Symbol Properties from the Edit menu, or by double-clicking the editor window. The **Symbol Properties dialog** appears. Here you specify following properties of the symbol:

- the name
- the shortcut, i.e. the default name of parts created from the symbol
- the DYNAST statement, i.e. type of DYNAST statement to which the symbol should be converted, usually a model
- the description (displayed in the library browser of the diagram editor)

Library of symbols are used in the diagram editor to represent models. Most often, you will need to connect your library symbols to library models, specified in separate .mod files. To do this, type `@`*model* to the **Type** field of the dialog, where *model* is the file name of your library model without path and extension, e.g. `diode`.

Sometimes you will need to connect your library to a DYNAST built-in construct, i.e. physical element or block. Then enter the identifier of the construct to the **Type** field, e.g. ‘R’ or ‘BI’.

Chapter 9
Invoking nonlinear analysis

9.1 Nonlinear analysis

Nonlinear analysis capability of DYNAST allows you to solve sets of nonlinear algebro-differential equations for given initial conditions and/or to analyze diagrams consisting of basic blocks, physical elements or submodels. In the case of diagrams, the underlying equations are formed automatically by DYNAST. The same analysis procedure can be also applied to sets of nonlinear algebraic equations and to static diagrams (i.e., diagrams the components of which are characterized by algebraic relations only).

By choosing different analysis modes you can compute transient and steady-state quiescent or periodic responses, and you can also ask for static or static parameter-sweeped analysis as well as for Fourier analysis of the responses. The initial conditions, at which the analysis starts, can be specified by yourself, they can also be taken from the previous analysis, or they can be computed by DYNAST to correspond to a steady state, either quiescent or periodic.

Besides various responses, the nonlinear analysis yields the equations, either submitted by the user, or formed by DYNAST for a diagram, linearized at the last solution point so that they can be utilized for the subsequent linear analysis.

9.2 Modes of nonlinear analysis

There are the following **nonlinear analysis modes** available to you in DYNAST:

- **transient analysis**, i.e., solving a set of algebro-differential equations or analyzing a dynamic diagram for the independent variable t growing from t_{min} to t_{max} and starting from the initial conditions $\mathbf{x}(t_{min}) = \mathbf{x}_0$ where $\mathbf{x}(t)$ is the vector of solved variables
- **static analysis**, i.e. solving a set of algebraic equations or analyzing a static diagram
- **quiescent steady-state analysis**, i.e. computation of the system transient response for $t \to \infty$ assuming that this response is constant

  In this case, DYNAST automatically converts any algebro-differential equation into an algebraic equation and any dynamic diagram into a static diagram by

  - setting all derivatives with respect to the independent variable t to zero
  - setting the Laplace operator s in transfer block functions to zero
  - setting the output variables of all differentiators to zero
  - ignoring all integrators (this may result in singularity of the underlying equations as the output variable of an integrator becomes undetermined if its output does not control any part of the system)

  - ignoring all C-type physical elements
  - replacing all L-type physical elements by an ideal connection

- static or quiescent steady-state **parameter-sweeped analysis**, i.e., solving the nonlinear algebraic equations for a parameter p growing from a value p_{min} up to a value p_{max} If t is taken as the parameter p, this becomes the **quasi-static analysis**
- **periodic steady-state analysis**, i.e. computation of the system transient response for $t \to \infty$ assuming that this response is periodic due to a periodic excitation and/or self-oscillations of the system
- **Fourier analysis** yielding the frequency spectrum of the periodic steady-state responses, and the nonlinear **distortion factor**

$$d = \sqrt{\sum_i A_i^2 - A_1^2} \Big/ \sqrt{\sum_i A_i^2}$$

  where A_i is the amplitude of the i-th harmonic of the spectrum

9.3 Initial conditions of nonlinear analysis

The nonlinear transient analysis of a system can be started from the **initial conditions** of the system solved variables

- set to **zero** (by default)
- **specified** by the user
- taken from the **residual solution** remaining in the computer operational memory from the previous nonlinear analysis execution during the current DYNAST run
- **loaded** from a nonlinear analysis residual solution file saved during a DYNAST run
- corresponding to a steady state, either **quiescent periodic**

In the static or quiescent steady-state analysis modes, the initial-condition option can be utilized for submitting the **solution initial estimate** to speed up the computation, and also to single out the required solution from other solutions in the case of multiple-solution problems.

9.4 Output of nonlinear analysis results

The nonlinear analysis results are saved into the DYNAST output file in a tabular form from which they can be plotted on the computer screen or on other devices, they can be also saved in the print-plot form. The variables or parameters of the solved set of equations or analyzed diagram which you want to be saved are referred to as the **output variables**.

The size of the steps at which the output variables are evaluated is changed automatically by DYNAST during the computation to minimize the computational time while maintaining the required accuracy. The output variables can be saved into the output file either as they are computed, or, before saving, their values can be recalculated at evenly distributed points of the independent variable using linear interpolation. The number of the points is specified by the user.

The solution vector and the set of linearized equations resulting from the last iteration of the numerical analysis is always stored temporarily in the computer operational memory as the **residual solution** until

- the residual solution is **erased** by the user
- DYNAST run has been **terminated**

The residual solution can be also saved into a residual-solution file. Each of the residual solutions, either stored temporarily or saved in a file, can be used in a subsequent nonlinear analysis to specify initial conditions, or, in a subsequent linear analysis, the system linearized equations.

9.5 Specification of nonlinear analysis using the wizard

Activate either the problem window, or the diagram window, in which the system to be analyzed is specified. In the Analysis menu choose Numerical nonlinear analysis and open the corresponding wizard. After filling in its text fields execute the analysis by choosing Analysis or Analysis&Plot from the Run menu.

9.6 Specification of nonlinear analysis using the text editor

Specification of the nonlinear analysis should be included in the TR section of the DYNAST problem file. Activate the problem window in which the system to be analyzed is specified. If the system is specified in an active diagram window, choose Problem file from the menu View.

The section can be open by the '`*TR;`' command, and closed by the '`*END;`' command or by a command opening some other section. In between, the following statements can be submitted and repeated several times (some of them can be omitted):

`*TR;`
[*modification of system parameters*]
[*erasing the previous solution and statements*]
mode of the analysis
[*initial conditions or solution estimate*]
output of analysis results
[*saving the last solution*]
analysis execution command `RUN`
[*computational control*]
⋮
`*END` *or another section opening command*

9.6.1 Modes of the analysis

The actual nonlinear analysis execution can be activated by the statement which can be combined with a list of parameters for the control of the numerical-analysis computational procedures.

If the statement specifying the required mode of analysis is omitted, the statement stored temporarily by DYNAST from the previous computation specified up to its execution up to the ‘`RUN`’ command within the current `TR` section of the input data (if any) shall apply also to this analysis.

9.6.2 Transient analysis

`TR` [`TIME`] *min max*`;`

min and ***max***

are numeric constants determining the lower and upper limits of the independent variable *t*

9.6.3 Transient analysis starting from quiescent steady-state

`DCTR` [`TIME`] *min max*`;`

min and ***max***

are numeric constants determining the lower and upper limits of the independent variable *t*

9.6.4 Static or quiescent steady-state analysis

`DC;`

9.6.5 Static or quiescent steady-state parameter-sweep analysis

`DC` [*parameter*] *min max*`;`

parameter

is the user-defined identifier of a parameter of an equation, basic block, physical element or library model. If omitted, the identifier ‘`TIME`’ is assumed by default in this place

min and ***max***

are numeric constants determining the lower and upper limits of the sweeped parameter

9.6.6 Fourier analysis

`FOUR` *period* [`,` *harmonics* [`,` *points*]]`;`

period

is a numeric constant giving the period of the analyzed periodic response

harmonics

is an integer numeric constant determining the required number of harmonics in the computed frequency spectrum. Its default value is 10.

points

is an integer numeric constant determining the number of points within one period at which the frequency spectrum of the waveforms is evaluated. The default value is 128.

Examples:

Next problem file invokes Fourier analysis of the solution of a nonlinear differential equation. Ten harmonics are computed and a histogram of the corresponding spectrum can be plotted. First, however, transient analysis over a sufficient number of periods should be performed to reach a steady-state periodic solution of the equation.

```
*SYSTEM; SYSVAR x;
0 = 10*cos(100pi*time) - x**3 - .02*VD.x;
*TR; tr 0 .1 ; INIT x=1;
PRINT (100) x; RUN;
FOUR 20m; RUN; *END;    :Fourier analysis
```

9.6.7 Initial conditions of nonlinear analysis

9.6.8 User-specified initial conditions

computations

`INIT` *variable* = *value* [, *variable* = *value*...];

variable

denotes

- the user-defined identifier of a solved variable
- the user-defined identifier of the across variable of a C-type physical element
- the command '`!XALL`' if the initial conditions of all the solved variables are to be set to the same value
- the command '`!XMAX`' to set the initial value of the solution-vector norm for the computational control

value

is a numeric constant. By default, it is zero (under the condition that there is no nonzero value available in the residual solution already)

9.6.9 Residual initial conditions

If no value is specified for the initial condition of a solved variable, DYNAST takes its value from the last residual solution available in the computer operational memory unless if it exists and if it has not been erased from the memory by the statement ‘`ERASE`’.

9.6.10 Loaded initial conditions

`LOAD` *name*[.*extension*]`;`

name

is the name of the solution-vector file in which a residual solution vector has been saved

extension

is a three-character string denoting the file-name extension. If it is the default string ‘`INC`’, it can be omitted.

Example:

For instance, the initial conditions may be set as

```
INIT X1 = -3, V.3 = 10k, NODE3 = .01,
I.LOAD = 1_A, !XMAX = 1;
```

9.6.11 Output variables

`PRINT` [[`(`*points*`)`] *variable* [`,` *variable*...]]`;`

points

is an integer numeric constant specifying the number of the independent-variable points evenly distributed over the interval of the analysis at which the values of the output variables are listed in the output file. If the string ‘`(`*points*`)`’ is omitted, the output-variable values are listed at the unevenly distributed points of the independent variable where they have been actually computed (in the former case, the output variables are evaluated from the computed values using a linear interpolation)

variable

is a user-specified identifier of a variable or parameter specified in the SYSTEM section. If no *variable* is specified, all variables and parameters will be saved into the output file

Examples:

The statement

```
PRINT (21) X1, X2, Y;
```

shall cause the output of a four-column table with 21 lines containing values of the variables denoted as `X1`, `X2` and `Y` at 21 evenly spaced points of the variable `TIME`.

9.6.12 Computational control

The analysis execution command ‘`RUN`’ can be augmented by the specification of computational-control parameters:

`RUN` [*control* = *value* [, *control* = *value*. . .]];

control

is the identifier of a computation-control parameter

value

is a numeric constant specifying the parameter value

Examples:

The following problem file demonstrates solution of nonlinear algebro-differential equations for initial conditions determined by static analysis.

```
*SYSTEM; SYSVAR i, v;
0 = i - 2*VD.v - 5;
0 = 10 - v - i**2;
*TR; DCTR 0 10;
PRINT; RUN; *END;
```

9.6.13 Saving the residual solution

`SAVE` *name*[.*extension*];

name

is the name of the residual solution file

extension

is a string denoting the file-name extension which is ‘`INC`’ by default

9.6.14 Modification of system parameters

`MODIFY` *parameter* = *value* [, *parameter* = *value*. . .];

parameter

denotes the user-defined identifier of a parameter of an equation, basic block, physical element or library model specified in the preceding `SYSTEM` section of the DYNAST problem file

value

is a numeric constant for the new parameter value

9.6.15 Erasing the residual solution and statements

The residual solution remaining in the computer operational memory from the previous computation, and also all the statements submitted in the current TR section already can be erased by the statement

```
RESET;
```

9.7 Plotting simulation results

As a result of analysis, DYNAST Solver creates an output text file. The output file usually contains one or more tables of analyzed data. Each column of the tables corresponds to the *independent* variable of the analysis (such as time or frequency).

The **Plot viewer window** in DYNSHELL is capable of plotting the results in a graphical way. To open Plot Viewer for a given output file, choose **Result Plot** from the View menu.

Selecting variables for plotting. First you should specify the independent variable of the plot, and the dependent variables that you want to display. By default, the first variable from the first analysis is considered as independent, and the second variable is considered as dependent. To change this, use **Set variables** –, which displays the **Plot - Select Variables dialog**.

In this dialog you should first select which plot (table) of the current output file you want to display selecting from the **Plot** list.

Then you should select the independent variable and dependent variables. Independent variable should be selected from the **Independent variable** list. One or more dependent variables may be selected in the **Dependent variable** list. You may also select independent and dependent variables by pressing the right mouse button in the Dependent variable list.

Importing plots. You may also display variables from two or more output-file tables in one plot. To do this, you must *import a plot* to the current plot. You may import plots from other output files, different plots from the current output file, or even the current plot (this is useful if you want to display two or more variables each with its own independent variable).

To import a plot,

1. From the Plot menu, choose Set variables.
2. Press the **Import** button.
3. Choose the output file you want to import.
4. Choose the table of variables which you would like later display from the imported output file.
5. Confirm your choice and return to the Select variables dialog. There you can see the imported variables at the end of the list of dependent variables.

6. Press the right mouse button in the Dependent variables list. Then you may still change the independent variable of the imported table, set dependent variables, or remove the imported table from the plot viewer.

Changing plot properties.

The plot appearance can be customized in many ways. To change the layout of the plot, use from the Axis Menu:

- the **Common Y** – toggles *Common Y* mode, when all dependent variables share the same independent variable.
- **Zero Y offset** – toggles the *Zero Offset Y* mode, when each dependent variable has its own axis with individual scale, but all zero points are aligned at the same vertical position.
- **Multiple Y** – toggles the *Multiple Y* mode, when all curves are displayed in individual plots, one above another.

To change the way how curves are displayed, use

- the **Log X** – to display independent axis in logarithmic instead of linear scale
- the **Discrete X** – to display histogram instead of smooth curves

You may also display the **point marks** on the curves using the **Point marks**. To set the mark density use the *Point Marks Occurrence* field on the Plot Viewer Toolbar.

Zooming.

By default, the ranges of all variables are automatically adjusted to ensure that all curves fit into the drawing area. You may customize the ranges by

- dragging the mouse over a region that you want to zoom in
- using the **Custom Range -** this allows to change the scales of the displayed variables manually
- using **Full View** – restores the scales to their original values

All these zooming actions can be undone using the **Undo Zoom** from the Axis menu. The undone zooming action can be redone using the **Redo Zoom**.

Cursor & tracing features.

If you use **Reference Cursor**, a window with values of dependent and independent variable corresponding to the mouse position appears.

The **Curve Tracing** allows you to observe coordinates of points on the chosen curve. After activating Tracing, click the curve you want to trace. Then move mouse left or right to see the tracing point along the curve. Press `ESC` key to stop tracing.

Annotating a plot. The plot viewer allows you to attach short text notes to plots. To add a new note, use **Text** from the Plot menu. It displays the **Plot - Text Properties dialog**.

The dialog defines properties of the text note. Enter the text of the note to the **Text field**. Fields **Font face** and **Font size** allow you to define font face and font size for the whole text. You may also apply formatting attributes (such as bold, italics, script) to parts of the note by selecting the part in the Text field, and pressing one of the buttons on the **Font style** toolbar.

Each note has a reference point that is placed in the coordinate system of a curve. The reference point is displayed in the plot viewer as a small red box. The **Alignment** buttons allows you to define the relative position of the reference point with respect to the note. The **Align to curve** list allows you to select to which curve the note will be aligned. This information is useful for repositioning of the textual note when some of properties of the plot change.

To end editing the current note, press the **Close** button. To delete the current note from the plot, press the **Delete** button. Use the mouse to define the position of the reference point of the note in the coordinate system of the given curve. To change properties of a note already defined in the plot, click it with the mouse.

To make the definition of the notes permanent, you may save the layout of all windows opened in DYNSHELL using the **Save Screen Layout**.

Chapter 10 Invoking numerical frequency analysis

10.1 Numerical frequency analysis

The purpose of frequency analysis is to compute the response of a linear or linearized system to the harmonic (sinusoidal) excitation in the steady state. In this DYNAST section a set of complex algebraic equations is formed and solved numerically at each frequency point resulting at numerical values of various frequency-characteristic components. In the semisymbolic analysis section the resulting frequency characteristics are in the form of a rational function with the frequency variable as a symbol. Unlike the semisymbolic analysis, the numerical frequency analysis is not limited to models with frequency-independent parameters only. Thus it can be applied even to linear models with distributed parameters.

The harmonic excitation must be applied to the analyzed system models using special physical elements: sources of sinusoidal across-variable type FE and/or sources of sinusoidal through-variable type FJ. All independent across-variable sources type E in the system model behave as ideal connectors, whereas all independent through-variable sources type J behave as nonexistent. Outputs of all blocks type BS act as across-variable references.

10.2 Specification of numerical frequency analysis using the wizard

Activate either the problem window, or the diagram window in which the system to be analyzed is specified. From the Analysis menu, choose Numerical frequency analysis and open the wizard. After filling in its text fields execute the analysis by choosing Analysis or Analysis&Plot from the Run menu.

10.3 Specification of numerical frequency analysis using the text editor

10.3.1 Specification of the system excitation

Syntax for the FE and FJ elements that should be applied to the system model specified in the SYSTEM section is similar to that of the other physical elements described in the, i.e.

element [> *type*] +*node* [- −*node*] [= *amplitude* [@ *phase*]];

element [> FE] - *series* [= *amplitude* [@ *phase*]];

element

is a user-defined identifier of the specified element

type

is the string specifying the element type, either `FE` or `FJ`

+node* and *-node

are identifiers of the diagram nodes between which the specified element is situated

series

is the identifier of the Z-class element to which this element is connected in-series

amplitude

is the numeric constant or symbolic expression specifying the amplitude of the harmonic source

phase

is the numeric constant or symbolic expression specifying the phase of the harmonic source in radians, which is zero by default

Example:

A harmonic source of voltage with the amplitude of 2.2 V and the phase of 180 degrees, connected between nodes `A` and `B`, can be entered as

```
FE A-B = 2.2 @ 1pi;
```

10.3.2 Specification of the analysis

Specification of the numerical frequency analysis should be included in the AC section of the DYNAST problem file. Activate the problem window in which the system to be analyzed is specified. If the system is specified in an active diagram window, choose Problem file from the View menu.

```
*AC;
properties of the analysis
specification of analysis results
RUN; : execution of the analysis
```

DYNAST evaluates the response of the system either at equidistant frequency points within a user-defined frequency range, or at user-defined frequency points. Thus the analysis can be specified either as

$$\texttt{FREQ}\left[\texttt{/[LIN]}\right][\textit{min max}]\texttt{;}$$

or as

$$\texttt{FREQ} = f_1,\ f_2, \ldots;$$

min* and *max

are numeric constants determining lower and upper limits of the frequency interval

f_1, f_2, . . .

are numeric constants specifying individual frequency points

‘/LIN’

, or just the ‘/’, is the string setting a linear scale for the frequency axis. By default, this scale is logarithmic.

If the statement FREQ is missing, DYNAST assumes the default frequency range from 10^{-1} to 10^{1}.

10.3.3 Output of the analysis

Tabular output for various components of the harmonic response of the system can be specified by the statement

PRINT [(*points*)] *component*.*variable* [, *component*.*variable*...];

where the identifier *variable* stands for the solved variable. The identifiers *component* are listed in Table 10.1.

Table 10.1 Components of reponses of AC analysis

Identifier	Component	Identifier	Component
MOD	modulus	DB	modulus in dB
RAD	phase in radians	RE	real part
DEG	phase in degrees	IM	imaginary part

Examples:

The statement

```
PRINT (60) MOD.I.R1, MOD.A;
```

results in a table giving modules of variables I.R1 and A at 60 points.

Chapter 11 Invoking semisymbolic analysis

11.1 Semisymbolic analysis

Semisymbolic analysis can be applied to linear time-invariant lumped-parameter system models composed from multipoles and/or blocks combined eventually with algebro-differential equations. The semisymbolic analysis can be also used for small-signal analysis of nonlinear models within the vicinity of a quiescent operating point. Such a point can be located and the system model can be linearized at it using the DYNAST nonlinear-system analysis.

DYNAST provides operator functions for a given system model in a semisymbolic rational form. The operator functions may represent either the system transfer functions, or the Laplace transforms of the system zero-input responses to a given initial state. The operator functions can be converted into semisymbolic time-domain responses for given input excitation and initial state of the system. The time-domain responses as well as various components of frequency characteristics of the semisymbolic transfer functions are also evaluated numerically.

The semisymbolic analysis is based on the computational scheme shown in Fig. 11.1.

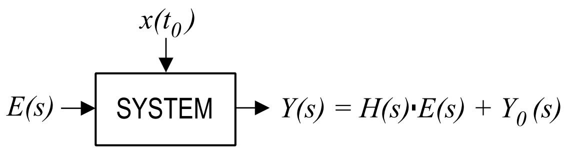


Figure 11.1 Computation of operational functions.

where $E(s)$ is the Laplace transform of a system excitation $e(t)$, and $Y(s)$ is the Laplace transform of the system response $y(t)$. For a given system model, DYNAST computes operator functions representing

- the system transfer function $H(s)$, i.e., the Laplace transform of the system response to an input stimulus $e(t)$ in the form of a Dirac impuls while the system initial state vector $x(t_0)$ is zero
- the Laplace transform of the system response $Y_0(s)$ to an initial state $x(t_0)$ while the excitation $e(t)$ is zero

The resulting operator functions are expressed in the semisymbolic rational form

$$F(s) = K\frac{(s-z_1)(s-z_2)\ldots}{(s-p_1)(s-p_2)\ldots} = K\frac{a_0+a_1s+a_2s+\ldots}{b_0+b_1s+b_2s+\ldots}$$

with the Laplace operator represented by the symbol s, and with numerically expressed multiplicative factor K, polynomial roots z_1, z_2,..., p_1, p_2,... (i.e., operator-function poles and zeros), and polynomial coefficients a_0, a_1, a_2, ..., b_0, b_1, b_2....

The system response $y(t)$ to an excitation $e(t)$ for its given rational Laplace tranform $E(s)$ and for an initial state $x(t_0)$ is then expressed in the semisymbolic form

$$y(t) = \sum_i a_i e^{b_i t} \cos(\omega_i t - \varphi_i)$$

where t stands as a symbol, whereas a_i, b_i, ω_i and φ_i are numerical constants.

DYNAST also evaluates various components of the semisymbolic responses and frequency characteristics numerically and displays them in different graphical forms.

11.2 Specification of semisymbolic analysis using the wizard

Activate either the problem window, or the diagram window, in which the system to be analyzed is specified. From the Analysis menu, choose Semisymbolic analysis and open the corresponding wizard. After filling in its text fields execute the analysis by choosing Analysis or Analysis&Plot from the Run menu.

11.3 Specification of semisymbolic analysis using the text editor

Activate the problem window in which the system to be analyzed is specified. If the system is specified in an active diagram window, choose Problem file from the menu View.

11.3.1 Operator functions

The required operator functions can be specified in the `PZ`-section using the statement

`TRAN` *function* [`,` *function* ...] `;`

where *function* can be of the form

identifier `=` *response* `/` *stimulus* [`coef`]

or

identifier `=` *block* [`coef`]

identifier

is a user-specified identifier of the required operator function

response

can represent

- the identifier of a primary variable (either node-across variable or through variable of a Z-class element)

- `V.`*element*, which is the across-variable of a Y-class element

stimulus

, separated from *response* by the character ‘`/`’, may represent:

- the source of an input stimulus in the form of the Dirac impuls if *function* is a transfer function. This can be:
  - *source*, which is the identifier of a `J`- or `E`-type element, or the identifier of a `BS`-type block assuming that this element or block has been specified in the SYSTEM section already (regardless of their actual specification there, they are considered as Dirac-impulse sources in this analysis)
  - `BS.`*node*, which is a BS-type block additionaly connected by its output to a node of the identifier *node*
  - `E.`*element*, which is an E-type element additionaly connected in series to a Z-type element of the identifier *element*
  - `J.`*element*, which is a J-type element additionaly connected in parallel to a Y-type element of the identifier *element*
- the Laplace transform of the system zero-input response to an initial state. Then *stimulus* represents the character string ‘`INIT`’.

`COEF`

is the string indicating the requirement to compute also polynomial coefficients for the operator function besides the polynomial roots.

When a transform of the system zero-input response is specified, the **initial state** is submitted by the separate statement

`INIT` *variable* `=` *value* `[,` *variable* `=` *value...*`];`

variable

can represent:

- `V.`*element*, which is the across-variable of an C-type element of the identifier *element*
- `I.`*element* which is the through-variable of a L-type element of the identifier *element*

value

is a numerical constant

The values of all the sources and variable initial states not mentioned in the specification of an operator function are considered as zero during the analysis of the function.

The actual execution of the analysis can be activated by the command

```
RUN;
```

Examples:

Some examples of operator functions:

```
TRAN T1 = I.RIOUT/EIN COEF, T2 = V.LOAD/INIT, T3 = BT13;
INIT V.C4 = .5, I.L0 = 1.2M, V.BI = 1;
```

Operator function T1 represents a transfer admittance, the function T2 is a Laplace transform of a response to initial conditions and the function T3 corresponds to the transfer function of the block BT13.

11.3.2 Operator-function time-responses

The required semisymbolic-form time-responses of some of the operator functions specified in the PZ-section can be submitted in the TRA-section using the statement

SYMB *expression* [, *expression* ...];

where *expression* is of the form

[*stimulus*]*transfer-function* [+ *init-function*]

or

init-function

stimulus

is the string indicating the stimulus of the transfer function; it may be either

- empty – then the impulse response of the transfer function is computed
- STEP. – then the step response of the transfer function is computed
- *block*. – then the response is defined by a previously defined BT *block*

transfer-function

is the identifier of a transfer function specified in the PZ-section

init-function

is the identifier of an operator function specified in the PZ-section, which was defined as response to the initial state of the system; it should have the same output specification as the *transfer-function*; the *init-function* may be also submitted alone (without the transfer function)

To evaluate numerically the time responses at specific time points, the points can be submitted either by the statement

TIME *min max*;

or by the statement

TIME = t_1, t_2,...;

min and ***max***

are numeric constants specifying the lower and upper limits of the time interval

$\boldsymbol{t_1, t_2, \ldots}$

are numeric constants giving the time points specified individually

If this statement is missing, DYNAST determines the **time interval** of the analysis **automatically**. For its upper limit DYNAST takes a multiple of the largest operator-function time constant, zero is then taken for the lower limit.

The **tabular output** of the time responses or of their combinations can be specified by the statement

`PRINT` *expression* [`,` *expression* . . .]`;`

where *expression* has the same form as the *expression* of the `SYMB` command.

The actual time-characteristics analysis execution can be activated by the command

```
RUN;
```

Example:

The statement

```
  PRINT (10) TF, STEP.TF + TFINIT;
```

results in table of 10 lines for the impulse characteristics of the transfer `TF` and also for the sum of the transfer `TF` step response and of the corresponding initial condition response (its transform denoted as `TFINIT`).

11.3.3 Frequency analysis of transfer functions

Computation of frequency characteristics of the semisymbolic transfer functions acquired in the `PZ` section can be specified in the FRE section.

Frequency is specified either as

`FREQ` [`/[LIN]`] [*min max*]`;`

or as

`FREQ =` f_1, f_2, . . .`;`

min and ***max***

are numeric constants determining lower and upper limits of the frequency interval

$\boldsymbol{f_1, f_2, \ldots}$

are numeric constants specifying individual frequency points

'/LIN'

, or just the '/', is the string setting a linear scale for the frequency axis. By default, this scale is logarithmic.

If the statement `FREQ` is missing, DYNAST determines the **frequency range automatically** taking into consideration the smallest and largest time constants of the analyzed problem.

Table 11.1 Frequency characteristics components

Identifier	Component	Identifier	Component
`MOD`	modulus	`DEL`	group delay
`DB`	modulus in dB	`SLO`	modulus slope
`RAD`	phase in radians	`RE`	real part
`DEG`	phase in degrees	`IM`	imaginary part

Tabular output for various components of the frequency characteristics is to be specified by the statement

`PRINT` [`(`*points*`)`] *component*`.`*function* [`,` *component*`.`*function*...]`;`

The identifiers *component* are listed in Table 11.1.

The actual frequency-characteristics analysis execution can be activated by the command

```
RUN;
```

Examples:

The statement

```
PRINT (60) MOD.ADMIT, MOD.IMPED;
```

results in a table giving modules of transfer functions `ADMIT` and `IMPED` at 60 points.

Chapter 12 Documenting problems and submodels

12.1 The documentation system

The documentation system facilitates considerably publishing problems solved by DYNAST as well as documenting the DYNAST submodels. The system was designed to produce documents in a standardized form and in an appropriate typographical quality with many operations automated. For this reason, the system is based on the LaTeX typesetting system and the produced documents are formated in PostScript (for printing), HTML (for electronic presentation), and PDF (for both).

You can exploit the documentation system directly from the DynShell menu either across the Internet on our server, or you can freely install it on your computer. A part of it – the latex2html package - requires, however, a UNIX compatible operational system. Before you start using the documentation system, you should configure it (see **Configuring Documentation System**). By default, the documentation system is configured for remote usage.

12.2 The documentation statements

The documentation statements are placed directly in the problem file together with the input data in the case of problem documents, or directly in the submodel data files in the case of submodel documents.

The documentation statements are denoted by colon characters, respecting the following rules:

- text between a colon '`:`' and the end of the line is treated as a comment and ignored both by the solver and by the documentation system
- a text to the right of a double colon '`::`' provides more detailed information for the statement preceding these two characters (e.g., the description of a parameter or of a submodel interface)
- the triple colon '`:::`' indicates that the text to the right of it on the same line is either a documentation statement or a text in LaTeX format

12.2.1 Special statements

The documents are underlined is by LaTeX formatted text combined with special statements. These statements must be always placed at the beginning of the line immediately after the three colons. Some of these statements have parameters.

12.2.2 Simple statements

Simple statements allow specifying a logical structure of the document. Each of the statements creates a new paragraph with a corresponding caption. The list of the simple statements shown in table 12.1 is stored in the file `template\dyn2tex.ini`, so it can be easily modified.

Table 12.1 Simple statements

Statement	Section caption
`:::EXCI`	System excitation
`:::TASK`	Task
`:::ORIG`	Example origin
`:::VARI`	Model variables
`:::SYST`	System
`:::MODEL`	Model
`:::ASSUM`	Modeling assumptions
`:::ANALY`	Model analysis
`:::RESUL`	Results
`:::PURP`	Purpose
`:::CONCL`	Conclusion

The FIG statement.

Syntax:

`:::FIG[`*suffix*,*width*`]{`*title*`}`

The `FIG` statement inserts a figure in Encapsulated PostScript format from an external file. The file must be placed in the same folder as the document file. The name of the file must be formed by the name of the document file followed by a suffix *suffix*.

If the parameter *width* (in millimeters) is entered, the figure is scaled to the given width (preserving the aspect ratio). Otherwise the original size is used. You may also specify the percentage of the original size.

The figure will get the title *title*.

All parameters are optional.

Example:

```
: File problem.prb
:::FIG[a]{the figure}
```

inserts an image from file `problema.eps`, preserves its original size, and adds title ‘the figure’.

```
:::FIG[,80%]
```

inserts an image from file `problem.eps`, and scales it down to 80% of the original size.

The DIAGRAM statement.

Syntax:

```
:::DIAGRAM[width]{title}
```

The `DIAGRAM` statement generates a figure in EPS format corresponding to the diagram of the current simulation experiment or library model, and inserts it to the document. The diagram file must be placed in the same folder as the document file, and must have the same name.

If the parameter *width* (in millimeters) is entered, the figure is scaled to the given width (preserving the aspect ratio). Otherwise the original size is used. You may also specify the percentage of the original size.

The figure will get the title *title*.

All parameters are optional.

Example:

```
: File problem.prb
:::DIAGRAM{the diagram}
```

inserts an image of the diagram `problem.dia`, preserves its original size, and adds title ‘the diagram’.

```
:::DIAGRAM[80%]
```

inserts an image of the diagram `problem.eps`, and scales it down to 80% of the original size.

The SYMBOL statement.

Syntax:

```
:::SYMBOL{library}{symbol}{title}
```

The `SYMBOL` statement generates a figure in EPS format corresponding to the library symbol, and inserts it to the document.

The parameter *library* specifies the name of the library, The parameter *symbol* specifies the name of the symbol in the library.

The figure will get the title *title*.

Example:

```
: File problem.prb
:::SYMBOL{electric}{R_ELE}{Symbol}
```

inserts an image of the library symbol R_ELE from the library file `electric.lbr`, and assigns a title ‘Symbol’.

The PLOT statement.

Syntax:

```
:::PLOT[parameter=value,parameter=value...] lpartitle}
```

The `PLOT` statement generates a figure in EPS format corresponding to the output of the current simulation experiment, and inserts it to the document. The output file must be placed in the same folder as the document file, and must have the same name. If the output file does not exists, it is created by running the DYNAST simulator on the current simulation experiment.

The figure will get the title *title*.

By default, the figure has dimensions 50×50mm, and displays the relationship of the second quantity on the first quantity of the first table of the output file. You may change the default behavior by submitting one or more parameters listed in table 12.2.

Table 12.2 Parameters of the PLOT statement

Parameter [Default]	Value
w[50]	the width of the figure in milimeters
h[50]	the height of the figure in milimeters
plot[1]	which plot (table) of the output file will be displayed, 1 denotes the first table
indep[0]	independent quantity, counts from zero
deps[1]	dependent quantities, count from zero; you may specify more quantities separated by the plus (‘+’), or ranges separated by the hyphen (‘-’), such as ‘1-5+10’ for quantities 1,2,3,4,5 and 10
range*n*	custom range of the *n*-th quantity (quantities counts from zero); *n* may be specified as range by two integers separated by the hyphen—range is then set for all quantities in the range; the range specification has the form *lower*`..`*upper*, where *lower* and *upper* are constants
import	imports another plot to the current plot; the import specification has the form *file*`;`*plot*`;`*indep*, where *file* is the name of the output file to import from without extension (empty string stands for the current file), *plot* is the index of the plot (table) in the file (counts from one), and *indep* is the index of the quantity to be considered as indepentent (counts from zero)
grid[yes]	‘no’ disables the grid
multiple[yes]	‘no’ disables the Multiple Y mode
common[no]	‘yes’ enables the Common Y mode
zero[no]	‘yes’ enables the Zero Offset Y mode
log[no]	‘yes’ enables the Log X mode
discrete[no]	‘yes’ enables the Discrete X mode
marks[no]	‘yes’ enables the point marks
occ[1]	sets the density of point marks, the bigger number, the lower density
title[yes]	‘no’ disables drawing of the plot title

Example:

```
: File problem.prb
:::PLOT[w=100,h=80]{The plot}
```

inserts an image generated from the output file `problema.o`, with size 100×80mm, displaying the first plot of the output file, with the zeroth variable as independent, and the first one as dependent.

```
:::PLOT[indep=2,deps=3-6+8-9]{The plot}
```

sets the second variable as independent and variables 3,4,5,6,8,9 as dependents

```
:::PLOT[log=yes,discrete=yes]{The plot}
```

sets the modes Log X and Discrete X on

```
:::PLOT[range0=0..1E6,range1-4=-1..3]{The plot}
```

sets the range of the zeroth variable from 0 to 10^6, and ranges of variables 1,2,3,4 from -1 to 3

```
:::PLOT[import=b;2;5,deps=1-20]{The plot}
```

imports the second plot of the file `b.o` to the current plot, and sets the fifth quantity of the plot as independent (the dependent quantities 1–20 now refer to the concatenation of quantities from the original plot and the imported plot)

See also:

Changing properties of the plot

The SITE statement.

Syntax:

```
:::SITE
::: site name ... site description
::: site name ... site description
::: ⋮
```

The `SITE` statement creates a section with description of subsystem sites of interaction. This description consists of several lines. Each line defines one site of interaction. The definition contains the name of the site and the description of the site, separated by ellipsis.

Example:

Input

```
:::SITE
::: A ... wheel
::: B ... car suspension
::: C ... car body
```

Output

Sites of interaction

A	wheel
B	car suspension
C	car body

The PARA statement.

Syntax:

```
:::PARA
```

The `PARA` statement inserts list of parameters. This list is taken from the active part of the data file. The parameters should be defined using following form:

1. *parameter* = *value*; ::[*dimension*] *description*
2. *parameter* = *value*; ::[*dimension*] *description* ::: *p*[*d*] = *v*
3. *parameter* = *value*; ::[*dimension*] *description* :::%

The first form adds the following line to the table of the parameters:

parameter = *value* [*dimension*] *description*

Using the second form you may redefine the appearance of items *parameter*, *dimension value* using items *p d v* respectively. These items may contain an arbitrary fragment of LaTeX code, which will be processed in LaTeX math mode.

By using the percentage `%` after three colons you can disable adding of a line to the table of parameters.

The three colons and the text behind them will not be inserted to the output generated by `DATA` statement.

Example:

Input

```
:::para
:::data
*SYSTEM;
x = 10;       :: [m] displacement
i = sqrt(2);  :: [A] current           ::: = \sqrt{2}
v = 5;        :: [m/s] velocity        ::: [\frac{m}{s}]
omega = 5pi;  :: [rad/s^-1] ang. vel.  ::: \omega = 5\pi
beta = 10pi;  :: [rad] angle           :::%
*END;
```

Output

System parameters

$x = 10$	$[\mathrm{m}]$	displacement
$i = \sqrt{2}$	$[\mathrm{A}]$	current
$v = 5$	$\left[\frac{m}{s}\right]$	velocity
$\omega = 5\pi$	$\left[\mathrm{rad/s}^{-1}\right]$	ang. vel.

Input data

```
*SYSTEM;
x = 10;       :: [m] displacement
i = sqrt(2);  :: [A] current
v = 5;        :: [m/s] velocity
omega = 5pi;  :: [rad/s^-1] ang. vel.
beta = 10pi;  :: [rad] angle
*END;
```

The DATA statement.

Syntax:

```
:::DATA
```

The DATA statement inserts the active part of the data file (simulation data) to the document. The document (lines or line parts beginning with three colons) is removed.

When creating the electronic document to a simulation experiment, the DATA statement inserts a form that can be used for remote simulation of the experiment.

For usage example see PARA statement.

The MODP statement.

Syntax:

```
:::MODP
```

The MODP statement inserts a list of modules (instances of submodels) to the document. Each module is accompanied by a list of parameters that were modified from their default values.

Example:

Input

```
:::MODP
chass1 > @ROD2 PX7-PX13,PY7-PY13,OM3/L=az1-a13;
chass2 > @ROD3 PX13-PX37,PY13-PY37,OM3/m=m3,J=J3;
spring20 > @SPRING1 0-PX2,PY20-PY2/c=k20,d=d20;
```

Output

Module parameters

Module: `chass1` — *Rod*

Submodel: `ROD2` — *Weightless rigid rod link in polar coordinates*

L = az1-a13	[m]	rod length

Module: `chass2` — *Rod*

Submodel: `ROD3` — *Mass rigid rod link*

m = m3	[kg]	mass
J = J3	[kg.m^2]	moment of inertia

Module: `spring20` — *Spring*

Submodel: `SPRING1` — *Linear weightless spring with damping*

c = k20	[N.m^{-1}]	spring rate
d = d20	[N.s.m^{-1}]	damping constant

The INTER statement. EXTP

Syntax:

```
:::INTER
:::EXTP
```

The statements INTER and EXTP can be used only for documenting of submodels. They insert the specification of the submodel interface and of the external parameters to the document.

Example:

Input

```
ROD3 ::Mass rigid rod link
:with two pin joints
Ax-  :: mechanical inlet in the $x$-direction
Bx,  :: mechanical inlet in the $x$-direction
Ay-  :: mechanical inlet in the $y$-direction
By,  :: mechanical inlet in the $y$-direction
om/  :: mechanical inlet of rotation
LA=1,:: [m]  distance from A and G :::L_A
LB=1,:: [m]  distance from B and G :::L_B
m=0, :: [kg] mass
J=0, :: [kg.m^2] moment of inertia
g=9.81; :: [m.s^-2] acceleration of gravity
::: INTER
::: EXTP
```

Output

Interface

Ax	mechanical inlet in the *x*-direction
Bx	mechanical inlet in the *x*-direction
Ay	mechanical inlet in the *y*-direction
By	mechanical inlet in the *y*-direction
om	mechanical inlet of rotation

External Parameters

$L_A = 1$	[m]	distance between joint A and G
$L_B = 1$	[m]	distance between joint B and G
$\mathtt{m} = 0$	[kg]	mass
$\mathtt{J} = 0$	$\left[\mathrm{kg.m}^2\right]$	moment of inertia
$\mathtt{g} = 9.81$	$\left[\mathrm{m.s}^{-2}\right]$	acceleration of gravity

The INCL statement.

Syntax:

```
:::INCL[category]{filename}
```

The INCL statement affects only the electronic form of the document. It inserts a reference to an applet, which displays the *DYNCAD* schematic editor and loads the file *filename* into it. Files (diagrams) may be organized to categories. To use a category other than the default, you must name of the category.

Example:

```
:::INCL{d1}
```

Inserts a reference to the diagram d1 from the default category

```
:::INCL[mechanical]{d2}
```

Inserts a reference to the diagram `d2` from category `mechanical`.

12.3 Documenting subsystems

A **subsystem** describes some part of a dynamic system that may have several **submodels**. Models may differ in interface, parameters, way of modeling of physical phenomena, etc.

The document describing a subsystem must be written in LaTeX. It contains only the document (not data), so it has no colons for distinguishing data from document. It may contain the special statements `FIG`, `SITE`, and the statement `CAPTION`.

The CAPTION statement.

Syntax:

`CAPTION` *name of the subsystem*

The statement `CAPTION` inserts the heading of a subsystem. It should be the first statement of each subsystem file document.

12.4 Processing the documents

The documentation system can generate an output in PostScript for printing, in HTML for electronic presentation, or in PDF format for both.

Conversion to PostScript can be done locally on a MS Windows platform or on a server equipped with the necessary software. Conversion to HTML and PDF can be done only remotely.

To create a document for a given file, you may choose from the Documentation menu

- **Publish document in PostScript**
- **Convert document to PDF**
- **Convert document to HTML**

To view previously created documentat, you may use from the same menu

- **View document in PostScript**
- **View document in PDF**
- **View document in HTML**

Chapter 13 Inside DYNAST

13.1 Formulation methods

As we have already mentioned, DYNAST is capable, besides other things, of solving systems of ordinary nonlinear and nonstationary first-order algebro-differential equations in the implicit form

$$\mathbf{f}(\mathbf{x}(t),\dot{\mathbf{x}}(t),t)=\mathbf{0} \tag{13.1}$$

where $\mathbf{f}(.)$ is the vector of functions, $\mathbf{x}(t)$ is the vector of the solved variables and $\dot{\mathbf{x}}(t)$ is the vector of their derivatives with respect to the independent variable t.

In case of static systems or in case of static or quasistatic analysis of dynamic systems, (13.1) boils down to to the system of nonlinear algebraic equations

$$\mathbf{f}(\mathbf{x}(t),t)=\mathbf{0} \tag{13.2}$$

The DYNAST procedure for solving these equations described further is based on their iterative linearization at discrete points of t. Thus, at the point $t=t_{n+1}$ and at the $(k+1)$-st iteration of the procedure, DYNAST formulates for a given set of (13.2) in fact the linear algebraic equations

$$\begin{aligned}&\left[\left(\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\right)_{n+1}^{(k)}+\gamma\left(\frac{\partial\mathbf{f}}{\partial\dot{\mathbf{x}}}\right)_{n+1}^{(k)}\right]\Delta\mathbf{x}_{n+1}^{(k+1)}\\&=-\mathbf{f}\left(\mathbf{x}_{n+1}^{(k)},\dot{\mathbf{x}}_{n+1}^{(k)},t_{n+1}\right)\end{aligned} \tag{13.3}$$

where γ is a variable computational factor. On the right-hand side we can see the approximation of the vector function $\mathbf{f}(\mathbf{x}(t),\dot{\mathbf{x}}(t),t)$ value as obtained at the previous iteration. In large parentheses are the jacobians of the vector function $\mathbf{f}(.)$ with respect to $\mathbf{x}_{n+1}^{(k)}$ and $\dot{\mathbf{x}}_{n+1}^{(k)}$. These are the approximations of the vectors $\mathbf{x}(t_{n+1})$ and $\dot{\mathbf{x}}(t_{n+1})$ at the k-th iteration of the solution process (13.3). DYNAST computes the jacobian elements automatically using a built in symbolic differentiation procedure.

The automatic formulation of diagram representing equations in DYNAST is based in principle on the modified method of nodal voltages. To make it applicable to the whole range of problems we wanted DYNAST to solve, we had to generalize the method

- to both nonlinear t domain as well as to linear s domain analysis
- to one port as well as to multiport elements
- to node as well as to port specified element interconnections (thanks to this a ‘series’ interconnection of several elements can be characterized just by one equation)
- to electrical as well as to nonelectrical elements.

Later, the formulation method was extended also to block diagrams and to their combinations with port diagrams [4], [5]. As DYNAST formulates all the primary port and block diagram equations simultaneously, it has no problems with the 'fast' or 'algebraic' loops as it is common to most of the other simulation programs.

The contributions to the jacobians as well as to the right-hand side of (13.3) corresponding to individual block and port diagram elements are preprogrammed in DYNAST in the form of 'matrix-stamps'. The Table 13.1 shows, how many primary equations DYNAST has to formulate for different items representing a dynamic system.

Table 13.1 Number of equations requirements

Type	Item	Equations
	first-order algebro-differential equation	1
	nonzero node	1
BS	explicit block	1
BO	implicit block	1
BI	integrator	1
BD	differentiator	1
BT	*n*-th order transfer block	$n+1$
J	through-variable source	0
R	resistor	0
G	conductor/damper	0
C	capacitor/inertor	0
E	across-variable source	1
RI	resistor	1
L	inductor/spring	1
OA	operational amplifier	1

For linear dynamic systems represented by a block and/or port diagram the corresponding equations are formulated using the matrix-stamps in the form

$$(\mathbf{A}_0(s)+s.\mathbf{A}_1(s)).\mathbf{X}(s)=\mathbf{B}(s) \tag{13.4}$$

where s is the complex Laplace operator. $\mathbf{A}_0(s)$ and $\mathbf{A}_1(s)$ are square matrices, $\mathbf{B}(s)$ is a column vector. In case of lumped-parameter dynamic systems these matrices are constant.

For the analysis of small excitations around a computed or user specified operating point of nonlinear lumped-parameter dynamic systems, DYNAST is able to determine the corresponding linearized (13.4) automatically. This is done very easily just by replacing the factor γ by the Laplace operator s in (13.3) after having the iteration process converged.

13.2 Computational methods

To solve the system of (13.1) in an interval $t \in \langle t_0, t_1 \rangle$ for given initial conditions $\mathbf{x}(t_0)$ during the transient analysis DYNAST uses the implicit multistep integration method characterized by the backward-differentiation formula

$$\mathbf{x}_{n+1} = hb_{-1}\dot{\mathbf{x}}_{n+1} + \sum_{i=0}^{r-1} a_i \mathbf{x}_{n-i} \tag{13.5}$$

The length of the integration steps h and, at the same time, the order of the method r DYNAST continuously optimizes during the integration with respect to the actual shape of the resulting responses. The aim is to minimize the computational time while respecting the admissible computational error [6]. The coefficients a_i and b_i in (13.5) are chosen in such a way, as to make the integration procedure numerically stable even for dynamic systems with a wide spread of time constants (stiff-stable method) [7]. The order of the integration varies within the range of 1 and 6.

In the discretized and linearized (13.3) the value of the integration factor $\gamma = 1/(hb_{-1})$ depends both on the integration step length h, as well as on the order of the integration method r. The solution $\Delta\mathbf{x}_{n+1}^{(k+1)} = \mathbf{x}_{n+1}^{(k+1)} - \mathbf{x}_{n+1}^{(k)}$ of (13.3) is used to correct the approximation of the vector $\mathbf{x}(t_{n+1})$ obtained at the k-th iteration. After finishing the iteration process, the initial estimate of the next step solution is made in accordance with the formula (13.5).

In the static cases represented by the systems of nonlinear algebraic (13.2), when the second jacobian in (13.3) is zero, the integration procedure described above turns into the modified well-known Newton-Raphson method. In case of quasistatic analysis, during which some of the system or ambient parameters is supposed to vary slowly in a specified range, the independent variable t represents the varied parameter. The integration procedure built-in DYNAST is then used there to control the independent parameter variations.

The linearized system of algebraic (13.3) is solved at each iteration by the LU algorithm, i.e. by the Gauss elimination method modification based on the decomposition of the system matrix into an upper and lower triangular matrix. This decomposition as well as the jacobian evaluation, however, is not performed at each iteration step. To make the computation faster, this is done only when the iteration convergence slows down too much.

Considerable savings of computational time and memory are achieved by matrix sparsity exploitation. The sparsity of jacobian matrices growths usually very fast with the complexity of dynamic systems (13.3). DYNAST stores in the computer operational memory the nonzero matrix elements only. The arithmetic operations resulting in zeroes are excluded in advance and the matrices are rearranged to minimize the matrix fill-in during their the LU decomposition.

To accelerate computation of periodic responses of weakly damped dynamic systems the iterative ε-algorithm [8] is utilized. For the spectral analysis of the steady-state periodic responses the fast Fourier analysis algorithm [9] is at the users disposal.

In case of the linear dynamic system analysis in the frequency domain, the corresponding (13.4) can be solved by DYNAST numerically at discrete frequency points. This approach is especially useful for distributed-parameter dynamic systems.

Besides that, however, for linear or linearized lumped-parameter dynamic systems, DYNAST makes possible to compute numerically their rational transfer functions and transforms of initial-condition responses. And what is important, the results are in the semisymbolic form, i.e. with symbolic Laplace *s* operator and numerical polynomial roots and coefficients.

For linear lumped parameter systems, for which the matrices $\mathbf{A}_0$, $\mathbf{A}_1$ and $\mathbf{B}$ are constant, (13.4) represent the generalized eigenvalue problem. The solutions of this problem in terms of *s*, correspond to the transfer function poles (i.e. denominator polynomial roots) of the analyzed dynamic system. An algorithm was developed to transform (13.4) to some other generalized eigenvalue problems the solutions of which correspond to the zeroes (i.e. numerator polynomial roots) of required transfer functions [10].

Before their actual solution, the generalized eigenvalue problems DYNAST reduces first into standard ones using a sparse matrix transformation procedure [11]. Thus the matrix $\mathbf{A}_1$, which is in engineering problems much more sparse usually than the matrix $\mathbf{A}_0$, is turned into a unit matrix. The standard eigenvalue problem is then solved iteratively by a very robust QR-algorithm. From the resulting transfer function poles and zeroes it is than very easy to derive their polynomial coefficients.

For the semisymbolic transfer functions the program than can numerically compute frequency and time characteristics in the semisymbolic form again. To do this, DYNAST makes use of closed-form formulas and does not thus have to resort to any approximations. Before that, however, the transfer functions have to be decomposed into sums of partial fractions.

13.3 Computational hints

- Try to keep the number of equations as low as possible, but do not oversimplify your model. DYNAST prefers models closed to nature
- When using block or port diagrams take into your consideration Table 13.1.
- Remember, that for DYNAST the dynamical models are easier to solve than the static ones
- Model the nonlinear relations by functions as smooth as possible, prefer functions which have their first derivatives continuous
- Start with simple models and make them more elaborate later
- When debugging a large model, deactivate first most of it by the semicolon characters : and remove these than gradually
- In case of ‘`SYSTEM IS SINGULAR`’ message check the number of your primary variables with respect to your primary equations, and also your initial conditions
- Avoid printplotting of results unless you really need it. Plotting is faster.

13.4 Computation control

Table 13.2 shows the parameters for the **computational control** in DYNAST, the values of which can be changed by the user in case of necessity.

Those are parameters of the section TR, parameters `WPRINT` and `WPLOT` apply also to the sections AC, PZ, FRE.

Table 13.2 Parameters for the computational control

Identifier	Parameter	Default value
`EPS`	permitted relative solution error	5.10^{-4}
`DCEPS`	permitted relative solution error in static analysis	10^{-6}
`MAXIT`	permitted number of iterations in one integration step	50
`DAMP`	iteration damping factor	0
`FLUF`	LU decomposition control factor	1
`MIN`	relative minimal permitted size of integration steps	10^{5}
`MAX`	relative maximal permitted length of integration steps	10
`FEPS`	prediction control factor	5
`KMAX`	maximal permitted prediction order	6
`C1...C6`	prediction damping factors of the order 1 to 6	1
`CYCLE`	maximal permitted number of the integration steps	32600
`WPRINT`	number of characters in one table line (from 30 to 120)	105
`WPLOT`	number of characters in one semiplot line (from 40 to 120)	80

During the transient analysis DYNAST continuously evaluates the relative computational error with respect to a norm related to the maximum absolute values reached by primary variables since the computation started. The initial value of this norm can be set by a numeric constant using the command `!XMAX` (see Chapter 8).

Chapter 14
DYNAST as a modeling toolbox for MATLAB

14.1 DYNAST & MATLAB in control design

DYNAST can be easily used as a *modeling toolbox for MATLAB*. While MATLAB is well suited to control design, DYNAST is capable of automated equation formulation even for very realistic models of real systems. You can combine these two programs to exploit advantages of both. Notably, you can implement and analyze a nonlinear model of the plant to be controlled in DYNAST. Then, for example, you may ask DYNAST to linearize the model, compute transfer-function poles and zeros of the plant model, and export them for the plant-control synthesis to MATLAB. Finally, to verify the complete controlled system you can use DYNAST again after augmenting the plant model by the resulting control configuration. During this design phase, you may use DYNAST to consider also plant nonlinearities as well as the non-ideal features of the controllers and sensors in various operation regimes of the control system. If the designed control is digital, you may verify it by interconnecting DYNAST with SIMULINK so that these two packages can communicate with each other at each time step.

How to do it, both in the case of analog and digital control, is described in this chapter. Even the DYNAST sitting on our server (or any other server) can communicate with MATLAB or SIMULINK installed on your own computer across the Internet. See the instructions and examples linked from `http://virtual.cvut.cz/dyn/`

14.2 Exporting transfer functions from DYNAST to MATLAB

Using the semisymbolic analysis DYNAST capability, you can compute poles and zeros the plant transfer functions necessary for the plant control design. A window with the output file showing the analysis results in a textual form opens in DYNSHELL automatically as soon as the computation is completed. From the View menu, choose then Operator functions to see the all the computed operator functions.

If you want to export all the computed transfer functions, select the problem title in the Operator functions box and click the Export to MATLAB button. If you want to export only some of the functions select them one by one and click the button for each of them separately. Do not forget, however, that your MATLAB should be configured properly and running before you start the export.

14.3 Controlling a plant model in DYNAST from SIMULINK

Using a controlling structure represented by a block diagram implemented in SIMULINK you can control a plant model implemented in DYNAST. In the SIMULINK block diagram, the DYNAST plant model is represented by a block called S-function. This block provides communication between SIMULINK and DYNAST.

14.3.1 Preparing the plant model in DYNAST

The DYNAST problem file describing the plant model must specify numerical transient analysis of the model. The analysis time interval must be at least as long as the time interval specified for the simulation in SIMULINK. In addition, the problem file for the plant model must include a statement defining the plant input and output signals. This statement should be placed at the end of the problem file, and it should be of the following form:

`: MATLAB interface spec: '`*inputs*`';'`*outputs*`'`

where *inputs* is a comma-separated list of input variables, and *outputs* is a comma-separated list of output variables. Each of the input variables should be some of the model parameters (i.e., parameter of an equation, element, block or submodel). Each of the output variables should be defined in the PRINT statement within the problem file (see **Specification of output variables**).

Example:

```
: MATLAB interface spec: 'force';'V.x,V.xdot'
```

14.3.2 Preparing the control diagram in SIMULINK

In SIMULINK, set up the block diagram of the control system with an S-function block from the SIMULINK Nonlinear library representing the plant model. Then specify the following parameters for the S-function block:

- set **S-function name** to `dynPlantL`
- set **S-function parameters** to *filename*, *sample* where *filename* is the name of the problem file with specifying the plant model, *sample* is the length of the fixed step size used during the simulation. In the specification of *filename*, you may use the `%DATA%` string as a substitution for the data directory of the DYNAST Shell environment.

  Example:

  ```
  '%DATA%\matlab\BallBeam.prb',0.02
  ```

14.4 Installing support files

Before using the DYNAST-MATLAB interface, some files must be installed in MATLAB. The following instructions describe the procedure for installing the DYNAST as a toolbox into MATLAB 5.2 for Windows. For other versions of MATLAB, the procedure may be slightly different.

1. copy directory *DYNAST*`\matlab\toolbox` to your *MATLAB* directory
2. run MATLAB
3. use **File - Set Path** command to add the path *MATLAB*`\toolbox\dynast` to the list of the paths accessible for MATLAB

DYNAST is the directory where DYNAST for Windows is installed.
MATLAB is the directory where MATLAB for Windows is installed.

To verify that MATLAB is prepared to utilize the DYNAST toolbox, type `HELP dynPlant` in MATLAB prompt. You should see the SIMULINK Help for the `dynPlant` S-function.

Chapter 15
Installing and configuring DYNAST

15.1 Installing DYNAST on Windows

15.1.1 System requirements

- IBM compatible PC computer
- 32MB RAM and 15MB of free disk space
- MS Windows 95, 98, Millenium, NT, 2000, or XP
- MS Internet Explorer 4.0 or higher
- Recommended: connection to the Internet

15.1.2 Installation procedure

1. Exit any existing copies of DYNAST you have running.
2. Download the `dynast.exe` file, or insert the DYNAST CD into your CD-ROM drive.
3. From the Start menu, choose Run, and type in either the path to the `dynast.exe` file, or to the `setup.exe` file that is in the root folder on the CD.
4. Read the welcome message and make sure that your computer meets the program requirements, then press the Next button.
5. If you don't have an older version of DYNAST installed, skip to step 6.. If DYNAST is already installed, select whether you want to overwrite the old version, or whether you want to install DYNAST to another folder; press the Next button.
6. Enter the target path for the installation, then press the Next button.
7. If you are not installing the Professional version of the DYNAST, skip to step . Select whether you want to install the device driver for the hardware key. The device driver is required for proper function of the Professional version of the DYNAST solver. Note that the installation of the device driver requires administration privileges under Windows NT, 2000 or XP.
8. Select whether you want to install the device driver for the hardlock. The device driver is required for proper function of the Professional version of the DYNAST solver. Note that the installation of the device driver requires administration privileges under Windows NT, 2000, or XP.
9. Confirm settings by pressing the Finish button. If DYNAST is already installed and is being overwritten, make sure it is not running.
10. Wait while setup installs files to your system.

11. In the *Registering shell types* dialog, select which extensions you want to associate with DYNAST. The association allows you to handle DYNAST files from within Windows environment. The dialog also shows whether an extension is not already associated with another application.
12. Let the installer install the HTML Help Update and COMCTL32 Update. It is OK if your system says that it doesn't need the updates.
13. Finish the installation by pressing the OK button.

15.1.3 Uninstalling DYNAST

Uninstalling removes all installed files. Files created by the user are *not* removed. The uninstallation procedure is the following:

1. From the Start menu, choose Settings, Control Panel.
2. In the Control Panel, choose Add/Remove Programs
3. Click DYNAST for Windows

15.2 Configuring DynShell

The main site for configuring DYNAST Shell is the **IDS-WIZARDS-CAPTION-SHEET-PREFS**, displayed by the **Options** command of the Preferences menu. The dialog consists of several pages, described in the following sections.

15.2.1 The text editor

The **Editor dialog** allows you to set up properties of the text editor. You may set up the way in which the editor will format text that is automatically inserted to the editor. The text is inserted to the editor, for example, by the commands from the System and Analysis menus.

You can set up the maximum **width** to which text being inserted should be formatted, and the **hanging indent**, i.e. number of spaces that will be inserted at the beginning of all lines except the first one.

15.2.2 The plot viewer

The **Plot dialog** allows you to set up properties of the plot viewer. You can set up plot properties independently for three output devices: the display, the printer, and the PostScript converter. Plots are accompanied by texts. You can select the **Font Face** and **Font Size** for the text.

Plots contains several elements like the frame, axes, curves etc. You can assign attributes (color, line width, and point marks) to them. Table 15.1 displays the list of the other elements of plots for which you can set their properties.

To change properties of some elements, select them in the **Symbols** list. Click mouse with Ctrl key to toggle an element, drag mouse or click with Shift key to select a range of elements. Then select available properties for selected elements.

Table 15.1 Properties of plot elements

Element	Color	Width	Point mark
Background	√		
Plot back.	√		
Frame	√	√	
Axis	√	√	
Grid	√	√	
Point mark		√	
Text	√		
Title	√		
Curve *i*	√	√	√

The **Colors** list displays the default set of colors. To customize the colors, press the **Custom colors** button.

15.2.3 The external tools

The **Tools dialog** allows you to set up some external tools utilized by DYNAST Shell.

The **documentation system** produces PostScript, PDF and HTML outputs. If you want to view these outputs, you must set up PostScript, PDF and HTML viewer.

The recommended viewer for PostScript, PDF and HTML is **GhostView**, **Adobe Acrobat**, and **MS Internet Explorer** respectively. You may also use **Netscape Navigator** for viewing HTML.

All above applications are freeware, and can be downloaded from following sites:

Table 15.2 External applications and their sites

GhostView	http://www.cs.wisc.edu/~ghost/
Acrobat	http://www.adobe.com/products/acrobat/
MS Internet Explorer	http://www.microsoft.com/windows/ie/
Netscape	http://home.netscape.com/browsers/

User can also configure DYNAST Shell to run its own custom tools (e.g. the MATLAB program). To do that, add custom menu items to the **Menu contents** list, and for each menu item (user tool), fill-in

- full path to the tool
- argument for the tool
- folder in which the tool will be executed

For the latter two, you may use symbolic names (macros) to express some useful strings, such as the full path to the file corresponding to the active document opened in DYNAST Shell. To insert a macro, press the ‘ $>$ ’ button right to the input field.

To run a tool configured in this page, select it from the Run menu.

15.2.4 The documentation system

The **LaTeX dialog** allows you to set up properties of the **documentation system**. If your computer is connected to the Internet, you can use the remote documentation server that is currently available at the address `icosym.cvut.cz:3000`. The remote documentation server can produce documents in PostScript, PDF and HTML formats.

You should enter there the e-mail address that will be presented in the headings of all the documents generated in HTML. You may customize conversion to HTML by setting the command line to LaTeX2HTML conversion program; see LaTeX2HTML documentation for more details (http://www.latex2html.org/). You may also specify what should be done after the conversion. The documentation system may open the LaTeX code, and/or the conversion outcome.

15.2.5 The folders

The **Folders dialog** allows you to set up folders for file types DynShell works with.

The **Problem Folder** is the folder that is referred to by several commands, such as **Open**, **List of Problems**, etc.

The **Submodel Folders** are the folders storing text files, diagram files and symbol libraries for submodels. These folders are searched when a reference to a model is made from a problem or submodel file model (if the model was not found within the folder of this file, or in its subfolders). The same folders are search through by the diagram editor.

15.2.6 Exporting transfer functions from DYNAST to MATLAB

Using the semisymbolic analysis capability of DYNAST, you can compute poles and zeros the plant transfer functions necessary for the plant control design. A window with the output file showing the analysis results in a textual form opens in DYNSHELL automatically as soon as the computation is completed. From the View menu, choose then Operator functions to see the all the computed operator functions.

If you want to export all the computed transfer functions, select the problem title in the Operator functions box and click the Export to MATLAB button. If you want to export only some of the functions select them one by one and click the button for each of them separately. Do not forget, however, that your MATLAB should be configured properly and running before you start the export.

15.2.7 Controlling a plant model in DYNAST from SIMULINK

You can verify control of a plant using the plant model implemented in DYNAST and controlled from a block diagram representing the control structure in Simulink. The DYNAST plant model is represented in the Simulink block diagram by the block called S-function. This block provides communication between SIMULINK and DYNAST. This communication can even take place across the Internet.

15.2.7.1 Preparing the plant model in DYNAST

Specify the numerical nonlinear transient analysis of the plant model in a DYNAST problem file. The time interval of the transient analysis must be at least as long as the time interval specified for the simulation in SIMULINK. In addition, the problem file must include a statement defining the plant input and output signals. This statement placed at the end of the problem file should be of the following form:

`: MATLAB interface spec: '`*inputs*`';'`*outputs*`'`

where *inputs* is a comma-separated list of input variables, and *outputs* is a comma-separated list of output variables. Each of the input variables should be one of the model parameters (e.g., parameter of a source or some other element, block, equation or submodel). Each of the output variables should one of the variables defined in the PRINT statement in the problem file (see **Specification of output variables**).

Example:

```
: MATLAB interface spec: 'force';'V.x,V.xdot'
```

15.2.7.2 Preparing the control diagram in SIMULINK

In SIMULINK, set up the block diagram of the control system with an S-function block from the SIMULINK Nonlinear library representing the plant model. Then specify the following parameters for the S-function block:

- set **S-function name** to `dynPlantL`
- set **S-function parameters** to *filename*, *sample*

where *filename* is the name of the problem file with specifying the plant model, *sample* is the length of the fixed step size used during the simulation. In the specification of *filename*, you may use the `%DATA%` string as a substitution for the data folder of the DYNAST Shell environment.

Example:

```
'%DATA%\matlab\BallBeam.prb',0.02
```

15.2.7.3 Installing support files

Before using the DYNAST-MATLAB interface, some files must be installed in MATLAB. The following instructions describe the procedure for installing the DYNAST as a toolbox into MATLAB 5.2 for Windows. For other versions of MATLAB, the procedure may be slightly different.

1. Copy folder *DYNAST*`\matlab\toolbox` to your *MATLAB* folder
2. Run MATLAB
3. Use **File - Set Path** command to add the path *MATLAB*`\toolbox\dynast` to the list of the paths accessible for MATLAB

DYNAST is the folder where DYNAST for Windows is installed.
MATLAB is the folder where MATLAB for Windows is installed.

To verify that MATLAB is prepared to utilize the DYNAST toolbox, type `HELP dynPlant` in MATLAB prompt. You should see the SIMULINK Help for the `dynPlant` S-function.

[1] Rubner-Petersen, T.: *Nonlinear Analysis Program NAP3* (an unfinished project). DTH, Lyngby 1980

[2] Mann, H.: *Computer applications in electrical engineering design* (in Czech). SNTL Publishing House, Prague 1984

[3] Oliva Z.: *Some algorithms for electronic circuit analysis* (in Czech). PhD. thesis, Czech Technical University, Prague 1986

[4] Mann, H.: *Multipoles, multiports and operational blocks*. Proc. European Conf. Circuit Theory and Design, London 1974

[5] Mann, H.: *Analysis of combined circuit-block diagrams*. Proc. Int. Symp. on Circuits and Systems ISCAS IEEE, Rome 1982, 639-642

[6] Rubner-Petersen, T.: *ALGDIF – a FORTRAN IV subroutine for solution and perturbated solutions of algebraic-differential equations*. Research Report, DTH, Lyngby 1979

[7] Gear, C.W: *Numerical initial value problems in ordinary differential equations.* Prentice-Hall, Englewood Cliffs, N.J. 1971

[8] Skelboe, S.: *Time-domain steady-state analysis of nonlinear electrical systems*. Proc. IEEE 70 (1982), 1210-1228

[9] Brigham, E.O.: *The Fast Fourier Transform*. Prentice-Hall, Englewood Cliffs, N.J. 1974

[10] Mann, H.: *An algorithm for the formulation of state-space equations*. Proc. 1979 Int. Symp. on Circuits and Systems ISCAS IEEE, Tokyo 1979, 161-162

[11] Rubner-Petersen,T.: *SFORM1 and SFORM2 – two FORTRAN IV subroutines for sparse matrix transformation of the general eigenproblem to standard form*. Research Report IT-41, DTH, Lyngby 1979

[12] Mann, H. et al.: *Computer-aided design of dynamic systems*. CSVTS House of Engineering, Prague 1986

[13] Mann, H.: *Theory of mechanical systems II.* Textbook, Technical University, Brno 1990