A Simnon Tutorial

Åström, Karl Johan

1985

Document Version:
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Åström, K. J. (1985). *A Simnon Tutorial*. (Research Reports TFRT-3176). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

CODEN: LUTFD2/(TFRT-3176)/1-87/(1985)

A SIMNON TUTORIAL

Karl Johan Åström

Department of Automatic Control
Lund Institute of Technology
July 1985

A SIMNON TUTORIAL

Karl Johan Åström

Department of Automatic Control
Lund Institute of Technology

Revised edition, July 1985

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
REPORT

Date of issue
July 1985

Document Number
CODEN: LUTFD2/(TFRT-3176)/1–87/(1985)

Author(s)
Karl Johan Åström

Supervisor

Sponsoring organisation

Title and subtitle
A Simnon Tutorial

Abstract

The purpose of this report is to give a tutorial introduction to the simulation language SIMNON.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
87

Recipient's notes

Security classification

Contents

1. Introduction

Simnon is a special programming language for simulating dynamical systems. Systems may be described as ordinary differential equations, as difference equation or as combinations of such equations. Models of this type are common in mathematics, biology, economics and in many branches of engineering. Simnon requires a computer with a graphics terminal. The results are displayed as curves on the terminal. The language has an interactive implementation which makes it easy for a user to work with the system. Simnon may be used in a very simple way to find solutions to difference or differential equations. This requires only six commands. There are 38 additional commands in the system. They also allow optimization, introduction of experimental data and parameter fitting.

The purpose of this report is to provide an introduction to the simulation language. The conceptual framework is first described in Chapter 2. The report then proceeds by examples. Chapter 3 describes how to solve simple differential equations. Solution of difference equations is described in Chapter 4. Chapter 5 describes simulation of more complicated systems which are obtained by combining subsystems described by differential or difference equations. It is useful to note that all of this can be accomplished by about a dozen commands. Some advanced features are described in Chapter 6. A few remarks on the implementation are given in Chapter 7. Each chapter is provided with exercises. Do not forget to experiment and test at a terminal as you progress with the reading. Remember that you can always type HELP and that there is also a manual which gives a detailed description of the language constructs. Also remember that a good way to learn the language is to start by learning how to master a few commands and expand the vocabulary gradually.

2. The Conceptual Framework

Simnon is an interactive language for simulating dynamical systems. The system may be described by ordinary differential equations, or difference equations. It is also possible to simulate systems which consist of interconnected subsystems. This is useful in order to structure a large system. Simnon may also be used for other purposes e.g. to graph functions to fit models to data etc.

The simulation language has facilities to edit system descriptions, to integrate differential equations, to store and retrieve data, to show the solutions as graphs, and to change parameters and initial conditions.

DESCRIPTIONS OF DYNAMICAL SYSTEMS

To use Simnon it is necessary to have a basic understanding of dynamical systems. In particular to be familiar with the notions of input, output and state. The generic form of a continuous time system is

$$\begin{cases} \dfrac{dx}{dt} = f(x, u) \\ y = g(x, u) \end{cases} \tag{2.1}$$

where u is a vector of inputs, y a vector of outputs and x is the state vector. A system like (2.1) is specified as a CONTINUOUS SYSTEM in Simnon. The analogous form for a discrete time system is

$$\begin{cases} x(t_{k+1}) = f(x(t_k), u(t_k)) \\ y(t_k) = g(x(t_k), u(t_k)). \quad k = 1,2,\ldots \end{cases} \tag{2.2}$$

A system like (2.2) is specified as a DISCRETE SYSTEM. Simnon allows a system or a subsystem to be described by either of the forms (2.1) or (2.2). It is also

possible to have interconnected systems where each subsystem has the form (2.1) or (2.2). Connections are described as a CONNECTING SYSTEM.

INTERACTION PRINCIPLES

Simnon gives information to the user via a graphical screen which can show curves, text and numbers. It is also possible to get a hard copy of a picture and to list system descriptions and data. Simnon receives information from the user by commands from the keyboard. The commands have the form

```
CMND arg1 arg2 ...
```

where CMND is the name of the command and arg1, arg2, etc. are the arguments. The name is a combination of up to eight characters. The arguments may be identifiers or numbers. Spaces are used as separators. A command is terminated by carriage return (CR).

Default values

It is desirable that commands are both short and flexible. One possibility to achieve these conflicting goals is to allow variations of a command which are selected by the arguments. A description of the simulation command illustrates the idea.

The simulation command

The different forms of the command SIMU which executes a simulation of a system are illustrated in the syntax diagram Fig. 1. The diagram implies that any form of the command which is obtained by traversing the graph in the directions of the arrows is allowed. For example the command

```
SIMU    0    100
```

simulates a system from time 0 to time 100. If we want to repeat the simulation a second time with different parameters it suffices to write

```
SIMU
```

The previous values of the arguments, i.e. 0 and 100, are then used. Most commands are provided with default arguments. These arguments are then used unless otherwise stated.

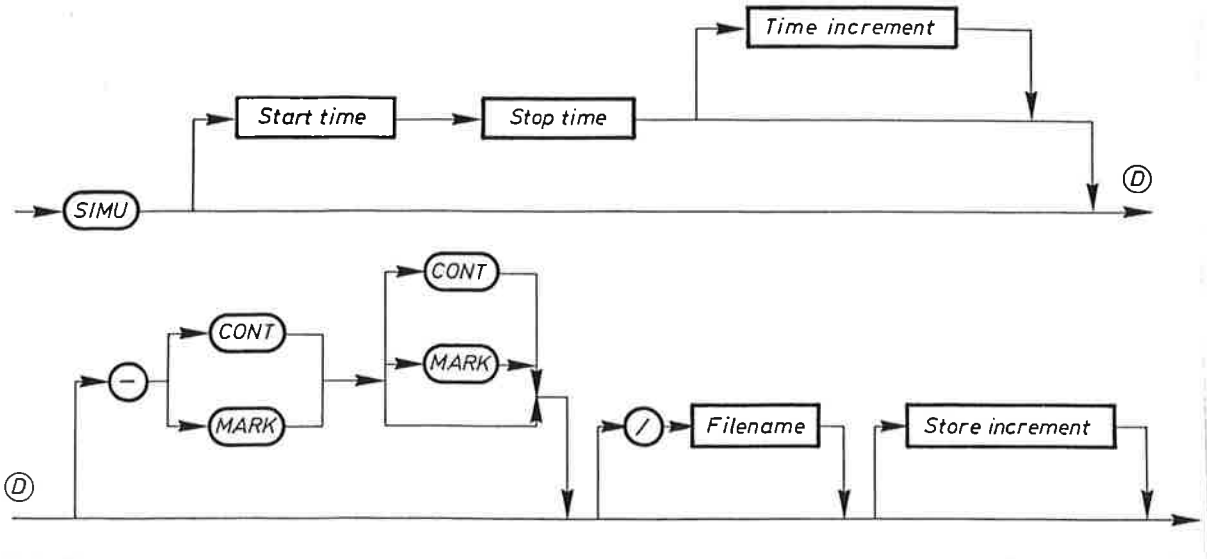


Figure 1. Syntax diagram for the command SIMU.

It follows from Fig. 1 that start and stop times and the initial time increment may be specified. It is also possible to mark curves by the optional argument MARK. A simulation may also be continued by using the end conditions of a previous simulation as initial values. This is done by the command option CONT. The results of a simulation may also be stored in a file. The time spacing between the stored values is specified by increment.

The syntax for the simulation command may also be described as follows:

```
SIMU [<start time> <stop  time> [<increment>]]
     [-{CONT|MARK}] [/<filename>[<increment>]]
```

This description is called the Backus-normal form (BNF), or the Bachus-Naur form. <...> denotes an argument i.e. a number or an identifier. [a1 a2 ...] denotes optional arguments which may be omitted and {a1|a2|...} denotes that one of the alternative arguments must be chosen. An asterisk (*) after an argument denotes that it may be repeated. The syntax for any command is obtained by typing the command HELP followed by the name of the command. The syntax is then shown in the Backus-Naur notation. The syntax of the commands is given in Appendix A.

EXERCISES

1. Learn how to log in and log out of your computer system and how to get access to mass storage areas. Get help from your systems manager if necessary. Document what you are doing so that you can repeat it on your own.

2. On the Vax implementations you start Simnon simply by typing Simnon. Simnon answers by the prompt >. You can exit Simnon by typing STOP. Practice this.

3. Try to find out about Simnon by using the command HELP.

4. Use the HELP command to find out how the command AXES works. Experiment with the AXES command to find out how it works.

3. Differential Equations

Solution of ordinary nonlinear differential equations is a simple application of Simnon. In such applications Simnon can be viewed simply as a calculator for differential equations. There are two minor differences compared to a calculator. Simnon can display curves. In a calculator a function is activated by pushing a dedicated key. In Simnon functions are activated by typing a command on the keyboard. How Simnon may be used to generate solutions to an ordinary differential equation is illustrated by an example.

THE PROBLEM

Suppose that we would like to know the character of the solution to the van der Pol equation

$$\frac{d^2y}{dt^2} + a(y^2 - b)\frac{dy}{dt} + y = 0 \qquad (3.1)$$

for different initial conditions and different values of the parameters a and b. The van der Pol equation is a model for an electronic oscillator.

The path towards the solution to the problem can be divided in the following steps.

1. Enter system descriptions
2. Simulate
3. Analyse the results
4. Change parameters and initial conditions

where steps 2, 3 and 4 are iterated until a satisfactory result is obtained. The different steps will now be described in some detail.

```
continuous system VDPOL
"The van der Pol equation
state x y
der dx dy
dy=x
dx=a*x*(b-y*y)-y
a: 1
b: 1
END
```

Listing 1. A Simnon system for Equation (3.2).

ENTER THE SYSTEM DESCRIPTION

The equation (3.1) is first rewritten in the standard state space form (2.1). Since the equation (3.1) is of second order an extra variable is introduced. The equation (3.1) can then be written as

$$\frac{dy}{dt} = x$$
$$\frac{dx}{dt} = ax(b - y^2) - y \qquad (3.2)$$

The equations are now in standard state space form, which is the format necessary to use Simnon. A file which describes the system should now be prepared. This file which is labeled VDPOL is listed in Listing 1. The first line indicates that it is a continuous time system with the name VDPOL. The state variables x and y and their derivatives dx and dy are declared. The differential equations are then defined. Notice the strong similarity with (3.2). Finally values are assigned to the parameters a and b. Simnon separates between parameters and variables. Parameters may be assigned values in a system description using the notation ':' for assignment. Parameter values may be reassigned interactively using the command PAR. Variables are defined using the notation '='.

The file can be edited using any editor you are familiar with. Assuming the standard editor you type EDIT VDPOL.T if you are in VMS and $EDIT VDPOL.T if you are in Simnon. In the VAX/VMS system all Simnon system files have the extension '.T'. This also applies to Macro files (Chapter 6). There is also a simple line oriented editor built into Simnon which can be used to enter the file. This

editor is invoked by the command

```
EDIT <filename>
```

where the argument <filename> is an identifier i.e. a letter possibly followed by letters or digits. The facilities in the editor are described in Appendix A. The syntax of the system descriptions is given in Appendix B. The command

```
LIST <filename>
```

lists a system description on the terminal.

SIMULATION

To run a system it must first be activated. This is done by the command

```
SYST VDPOL
```

If there are any errors during the compilation an error message is given and the system enters the Simnon editor so that the error may be corrected. If you want to use another editor simply type LEAVE to exit Simnon's editor.

If we would like to see the solution curves as they are integrated we must first draw axes on the screen. This is done with the command

```
AXES H 0 20 V -6 6
```

which means that the ranges of the horizontal (H) and vertical (V) axes are chosen as (0, 20) and (-6, 6).

The command

```
PLOT x y
```

instructs the program to plot the variables x and y as functions of time.

To perform a simulation it is necessary to give appropriate initial conditions to the state variables. The command

```
INIT x: 1
```

assigns the initial value 1 to the state variable x. Initial values are automatically set to zero if no assignments are made. We are now ready to perform a simulation. The command

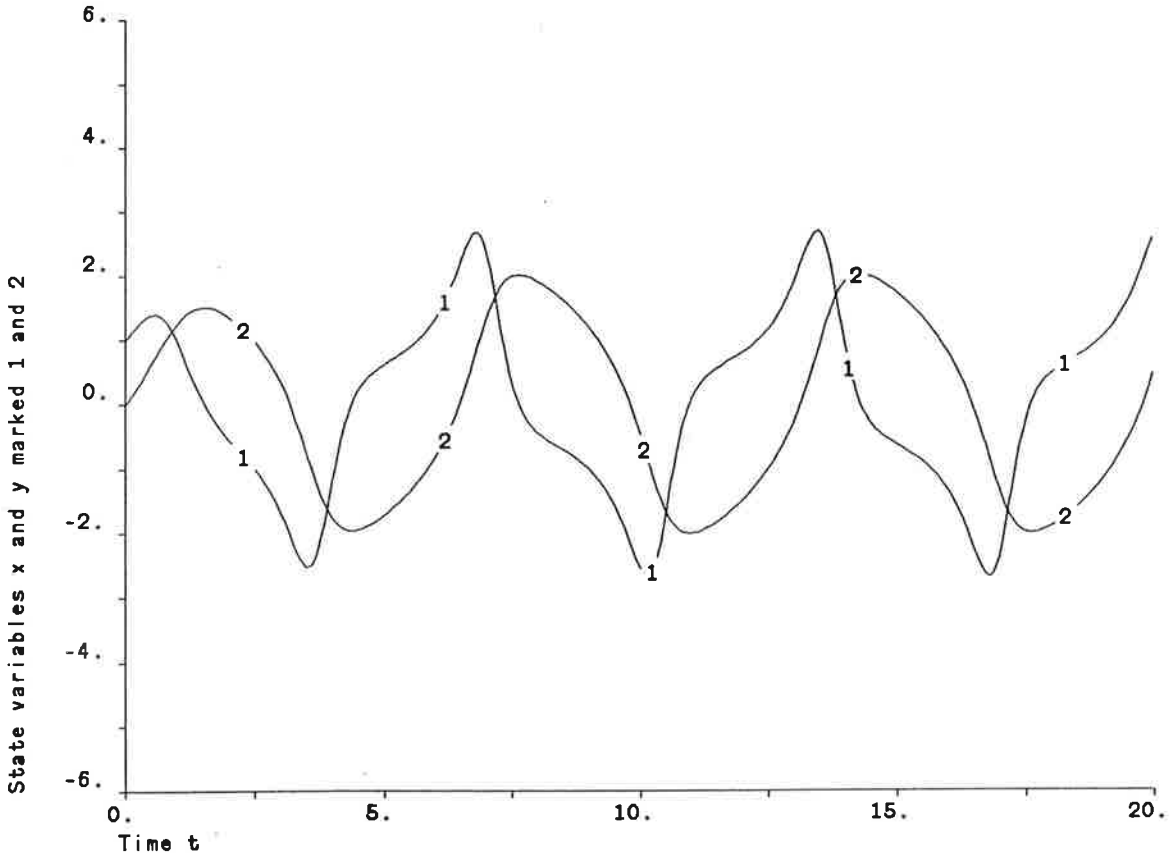


Figure 2. Simulation of the van der Pol equation for a=1 and b=1 with initial values x(0)=1 and y(0)=0.

```
SIMU 0 20 - MARK
```

activates a simulation from time 0 to time 20, and the result shown in Fig. 2 is obtained. The argument -MARK implies that the curves are labeled with integers 1,2,... in the order they appear in the plot command.

HOW TO INTERRUPT A SIMULATION

Some times when you make a simulation you will find that the results are wrong at the beginning. It is then useful to be able to break the simulation immediately. There are facilities for doing this in Simnon. The details are implementation dependent. On the standard Vax systems the simulation is interrupted by typing CTRL-C.

CHANGING PARAMETERS

Suppose that it is of interest to explore how the character of the solution is influenced by the parameters a and b. The command

```
PAR b: 2
```

assigns the value 2 to the parameter b. The command

```
SIMU
```

now generates a new simulation. Notice that it is not necessary to specify any arguments in the command SIMU. The previously given values 0 and 20 are automatically used as default values. The use of default values simplifies the user interaction considerably.

It is now easy to continue to explore how the solution is influenced by the parameters of the solution by a repeated use of the commands PAR and SIMU.

Current values of all states, derivatives, variables and parameters may be displayed. The command

```
DISP
```

displays all current data. Selective displays of the parameter a, the state x and the variable y is done by the command

```
DISP a x y
```

The display may also be directed to the line printer. A simple way to find out how the command DISP works is to type

```
HELP DISP
```

The help function can be applied to all commands.

STORING AND EDITING RESULTS OF A SIMULATION

It may be useful to store some results, to compare results from different simulations and to plot different state variables in different diagrams. Suppose for example that we would like to compare the results for the parameter sets a=1, b=1 with a=1, b=2. Two data files are first generated. The command

```
PAR a: 1
PAR b: 1
```

sets the parameters. The command

```
STORE x y
```

tells that the state variables x and y should be stored. The command

```
SIMU/B1
```

then performs a simulation and stores x and y in a file called B1. The value of b is set to 2 by

```
PAR b: 2
```

and the command

```
SIMU/B2
```

simulates and stores the results in file B2. The command

```
SPLIT 2 1
```

splits the screen into two windows. The command

```
ASHOW x/B2
```

plots the variable x from file B2 in the first window using automatic scaling. The command

```
SHOW x/B1
```

plots the variable x from file B1 in the same window. The commands

```
ASHOW y/B2
SHOW  y/B1
```

plots the variable y from files B2 and B1 in the second window. To document the results it is useful to generate a hardcopy of the curves obtained. The details depend on the hardware. On a normal installation the command

```
HCOPY
```

sends a copy of the picture on the screen to the plotter queue. The results shown in Fig. 3 are then obtained. The command

```
SPLIT 1 1
```

clears the screen and resets plotting to one window which covers the full screen.

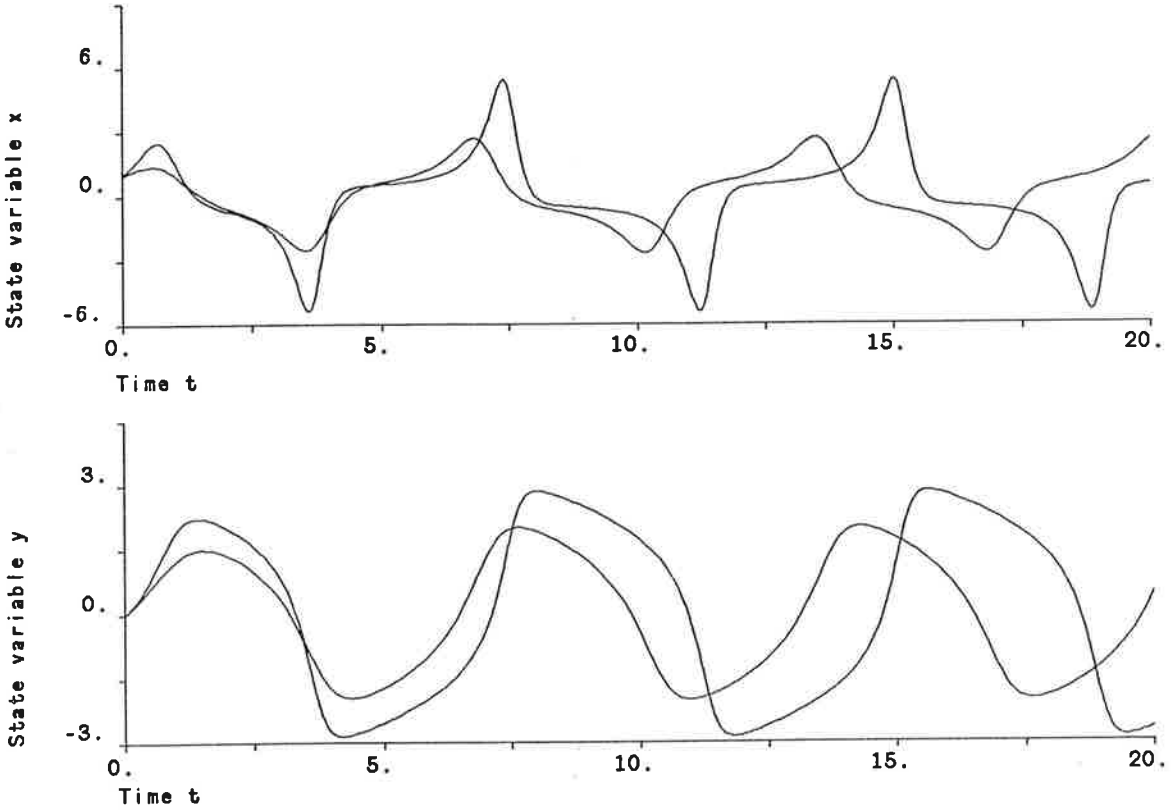


Figure 3. Solution of the van der Pol equation for b = 1 and b = 2.

PHASE PLANE PLOTS

For two-dimensional differential equations it is useful to visualize the results as phase plane plots. This may be done simply by

```
AXES H -4 4 V -3 3
SHOW y(x)/B1
```

which generates Fig. 4.

GENERALITIES

The general way of using Simnon as a "calculator for differential equations" will now be given. The generic form of a system description is given in Listing 2. The system description starts with CONTINUOUS SYSTEM <Identifier>. It is terminated by a line which contains END. An identifier is a a sequence of letters and digits

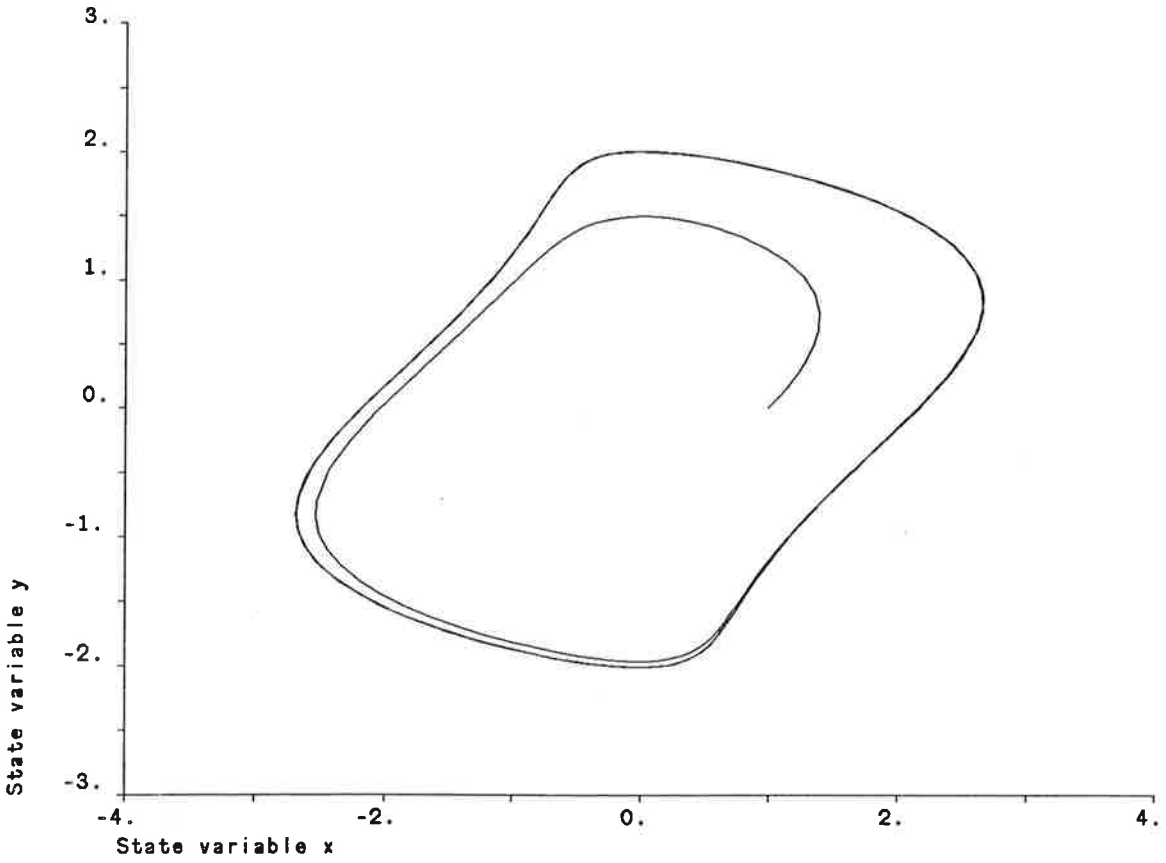


Figure 4. Phase plane plot of the van der Pol equation for a = 1, b = 1, x(0) = 1, y(0) = 0.

```
CONTINUOUS SYSTEM <Identifier>
"General differential equation          Comment
state <Identifier>...<Identifier>  }
der   <Identifier>...<Identifier>  } Declarations
time  <Identifier>                 }
computation of auxiliary variables  }
computation of derivatives          } Body
parameter assignment                }
initial value assignment            }
END
```

Listing 2. Generic form of system description for simulation of differential equations.

where the first character must be a letter. Both upper and lower case letters may be used although the compiler does not distinguish between them. The

system description has two parts, a declaration and a body.

There are three types of declarations. A time variable may be declared for simulation of time varying differential equations. This is done by the keyword TIME followed by an identifier. The state variables and their derivatives are declared by the keywords STATE and DER followed by a list of the state variables and the derivatives. associated by their sequential order in the lists.

The body of the system description specifies the derivatives of the state variables in terms of state variables and parameters. Auxiliary variables may also be used. The body also contains assignment of parameters and initial values. The order of the statements in the body is unimportant. A variable may only be defined once.

Expressions and operators

The expressions available in Simnon are similar to those in a procedural language like Algol or Pascal. An expression may be a string, a numeric constant or a variable. It may also be combinations of variables, operators and functions. Conditional expressions of the form IF...THEN...ELSE are also permitted. Simnon has arithmetic, relational and logical operators. All variables are floating point numbers. The numbers are written in the conventional way as 4, 1.1 or 6E7. The result of boolean expressions is 1.0 if it is true and 0.0 if it is false.

Arithmetic operators

The arithmetic operators are addition, subtraction, multiplication, division and exponentiation. They are denoted as

```
+  -  *  /  ^
```

respectively.

Relational operators

There are three relational operators, namely equal, greater than, and smaller than. They are denoted by

```
=  >  <
```

Logical operators

The logical operators are AND, OR and NOT.

Functions

The following functions are available:

abs(x)	absolute value
sign(x)	$\begin{cases} -1 & x<0 \\ 0 & x=0 \\ 1 & x>0 \end{cases}$
int(x)	largest integer less than x
mod(x,y)	x mod y
max(x,y)	largest of x and y
min(x,y)	smallest of x and y
sqrt(x)	square root of x, $x \geq 0$
exp(x)	exponential function
ln(x)	natural logarithm of x
log(x)	logarithm (base 10) of x
sin(x)	sine of x (x is in radians)
cos(x)	cosine of x (x is in radians)
tan(x)	tangent of x (x is in radians)
arcsin(x)	arcsine
arccos(x)	arccos
atan(x)	arctangent of x result in $(-\pi/2, \pi/2)$
atan2(x,y)	arctangent of x/y result in $(-\pi, \pi)$
sinh(x)	hyperbolic sine
cosh(x)	hyperbolic cosine
tanh(x)	hyperbolic tan

SUMMARY

It has been demonstrated that Simnon may be used to generate solutions to ordinary differential equations in a very simple way. To do this the differential equations are first written as a system of first order equations like

$$\frac{dx}{dt} = f(x,t).$$

A continuous time system is then generated in Simnon by declaring the state variables and the derivatives. The function f, which defines the right hand side of the differential equation, is then introduced using ordinary mathematical expressions and assignments of parameters. There is no vector notation so all equations must be written in scalar notation. Any editor may be used. There is also a special editor incorporated into Simnon, which is invoked by the command

EDIT. The command LIST can be used to list files. A simulation is executed using six basic commands: SYST, AXES, PLOT, INIT, PAR and SIMU.

In order to edit, manipulate and document results from several different simulations it is, however, also useful to use six additional commands, namely STORE, SHOW, DISP, SPLIT, AREA and HCOPY.

The HELP command is useful in order to see what the commands do.

EXERCISES:

1. Learn how to use an editor on your system. Practice by editing the system in Listing 1.

2. Learn how to enter and exit from Simnon. Use the command LIST to list the system you have edited on the screen and on the line printer. (Note in the Vax implementation you may invoke a program called PRG from Simnon by typing $PRG without leaving Simnon.)

3. Repeat the simulation described in this chapter on your own.

4. Change parameters with the command PAR and initial conditions with the command INIT and investigate how the solutions to the van der Pool equations change.

5. Experiment with the commands AXES, SPLIT, SHOW, ASHOW and AREA.

6. Use the HELP command to investigate the basic simulation commands AXES, PLOT, INIT, PAR, SYST and SIMU.

7. Introduce a formal error in the program in Listing 1 by changing the assignment of dx to dx = a*x*(b-y*y-y. Try to simulate the incorrect program. (When an error is detected the command EDIT is automatically executed. You may correct the mistake using the line editor. The simulation is then automatically continued when you exit the editor by typing E. You can also leave the editor by the command LEAVE and use your favourite editor for the correction.)

8. Use the HELP command to investigate the auxiliary commands STORE, SHOW, DISP, ASHOW, SPLIT and AREA.

9. Look at the commands LIST, LP and HCOPY and learn how to get hardcopies of listings and plots on your system.

10. Consider the following van der Pol equation

$$\varepsilon\ \ddot{y} + (y^2-1)\ \dot{y} + y = a$$

Investigate the limit cycles obtained for $\epsilon = 0.05$ and $0.993 \leq a \leq 1$.

11. The following equations called the Volterra Lotka equations represent a simple model for the development of two competing species

$$\frac{dx}{dt} = (a - by)x$$

$$\frac{dy}{dt} = (cx - d)y$$

Make a Simnon program to study the equations. Start with the following nominal values: a = 2.7, b = 0.7, c = 1 and d = 3.

12. A simple model for a satellite orbit is given by

$$\frac{d^2r}{dt^2} - r\left[\frac{d\varphi}{dt}\right]^2 = -\frac{k}{r^2}$$

$$\frac{d^2\varphi}{dt^2} + \frac{2}{r}\frac{dr}{dt}\frac{d\varphi}{dt} = 0$$

where r is the radius from the center of the earth, φ the azimut angle and k a gravitational constant. Simulate the equations.

13. Learn how to interrupt a simulation on your installation.

14. Simulate the differential equation

$$\frac{dx}{dt} = a(y-x), \qquad x(0) = -8$$

$$\frac{dy}{dt} = bx - y - xz, \qquad y(0) = -8$$

$$\frac{dz}{dt} = -cz + xy, \qquad z(0) = 24$$

where a = 10, b = 28 and c = 8/3. Look in particular at z as a function of x. The equation, which is called the Lorenz equation, is an example of a deterministic system, which has a very irregular (chaotic) behavior.

4. Difference Equations

There are also facilities for simulating difference equations in Simnon. This is done analogously to simulation of differential equations. An example is first given and the general principles are then stated.

EXAMPLE

Consider the following difference equation

$$x_{k+1} = x_k + r * x_k * (1-x_k), \quad k = 0,1,\ldots \tag{4.1}$$

This is a simple model of a population dynamics. The variable x_k denotes the number of individuals at time k in a normalized scale. There is an equilibrium value x=1 at which the population remains constant. For $x_k \approx 0$ the population increases with the factor 1+r in each generation. For $x_k > 1$ the population will decrease. Assume that it is of interest to investigate how the population changes with time.

A difference equation is characterized as a DISCRETE SYSTEM in Simnon. The description of a system which simulates the difference equation (4.1) is given in Listing 3.

The state variable x is declared in the same way as for continuous time system using the STATE declaration. The declaration NEW is used to declare the variable nx, which denotes the new value of the state variable. NEW is thus the analog of DER for continuous time systems.

When simulating difference equations it is necessary to provide a mechanism for making the state variables change at certain sampling times. For example the state

```
discrete system POPDYN
"Simple model for population dynamics
state x
new nx
time k
tsamp ts
nx=x+r*x*(1-x)
ts=k+1
x: 0.5
r: 2
END
```

Listing 3. A Simnon system for simulation of the difference equation (4.1).

variable in equation (4.1) changes periodically at k = 1, 2, ... In the general case there may, however, be irregular sampling. The mechanism used to describe this in Simnon is to introduce a variable TSAMP which gives the next sampling time.

In Listing 3 the first assignment statement is simply a definition of the right hand side of the difference equation (4.1). The second line: ts=k+1 assigns the value k+1 to the TSAMP variable ts. This tells that the system will be activated next at time k+1.

SIMULATION

Discrete time systems are run in the same way as continuous time systems. A simulation is thus executed by the commands

```
syst   popdyn
split  3 1
axes   H 0 80 V 0 1
plot   x
simu   0 80
```

This generates the uppermost curve in Fig. 5. The commands

```
axes
par r: 2.70
simu
axes
par r: 2.83
simu
```

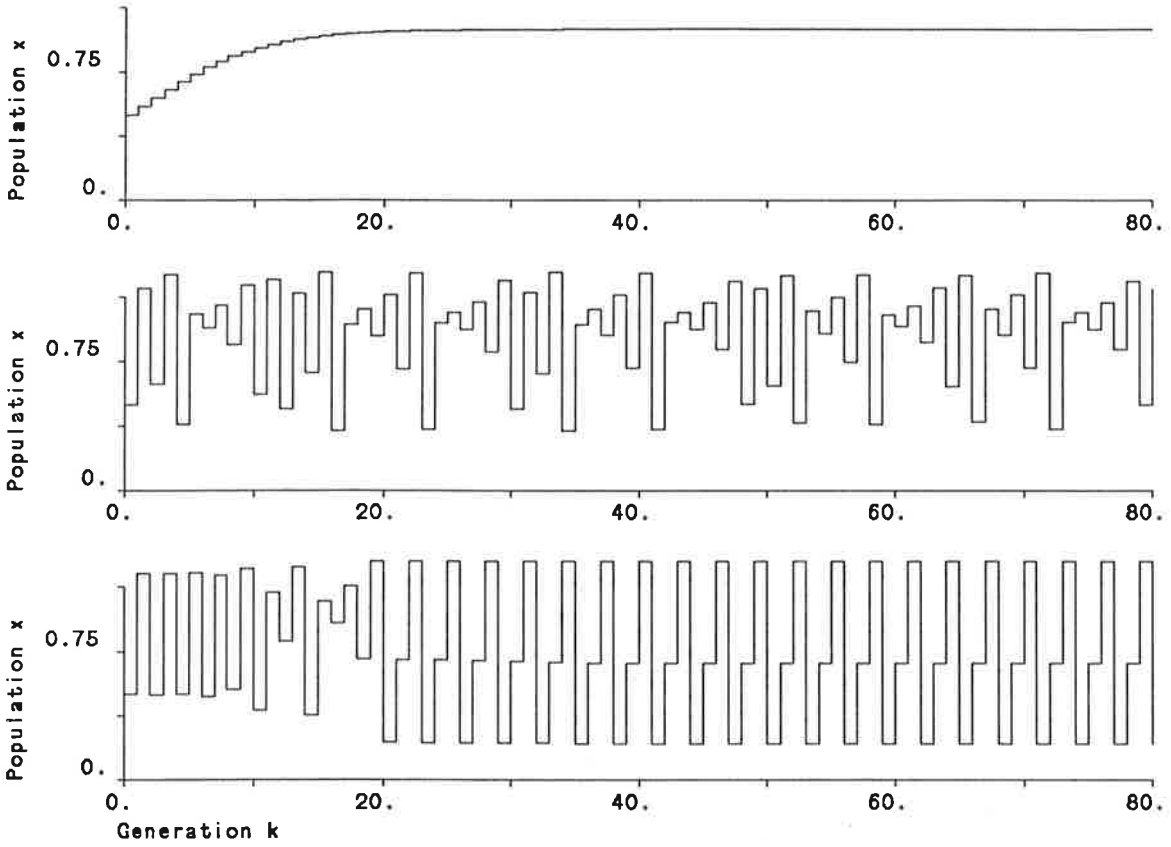


Figure 5. Simulation of population dynamics.

repeats the simulation for r = 2.70 and r = 2.83, and plots the corresponding curves shown in Fig. 5. Notice that the behaviour of the solutions change drastically with moderate changes in the parameters.

GRAPHS OF FUNCTIONS

Simnon is conveniently used to obtain a graph of a function. Assume for example that we want a graph of the polynomial

$$f(x) = x(x+3)(x+2)(x-2)(x-3).$$

This is accomplished by the following system.

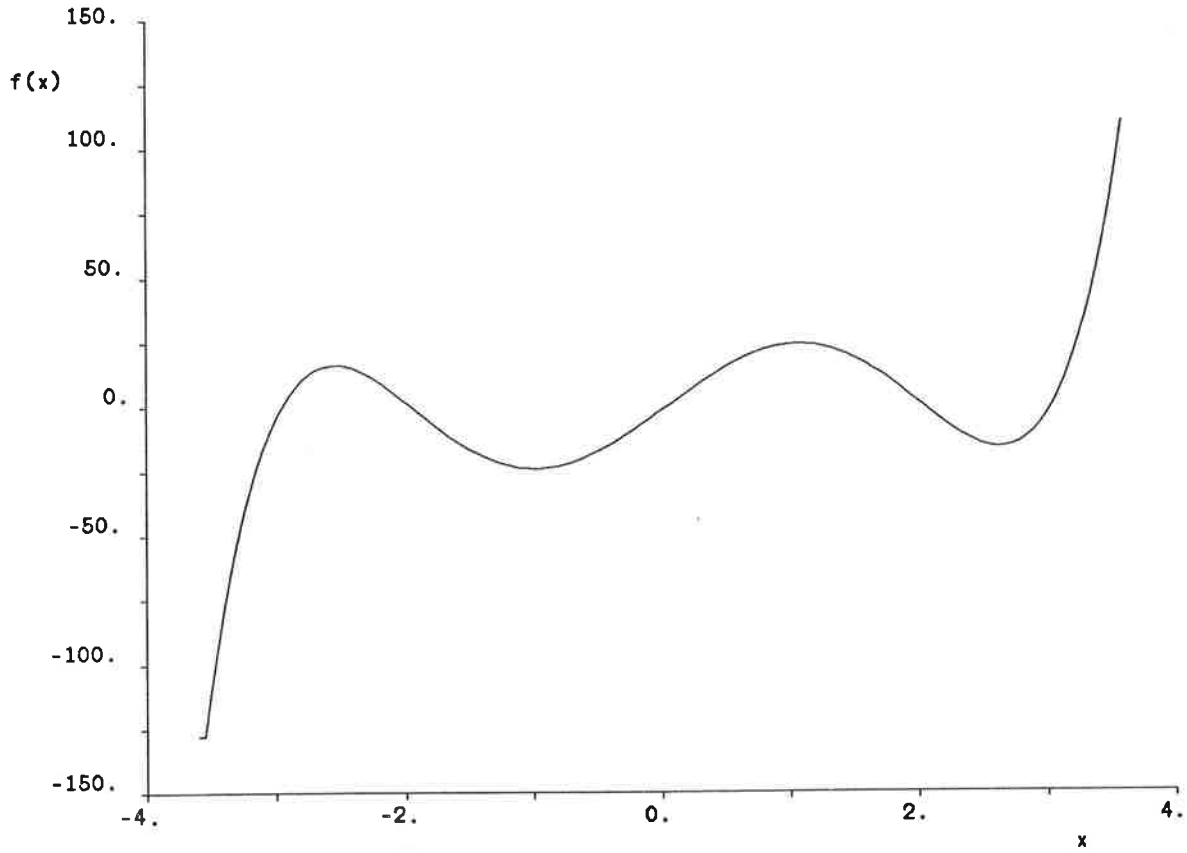


Figure 6. Graph of the function f(x) = x(x+3)(x+2)(x-2)(x-3) generated by Simnon.

```
discrete system POL
time x
tsamp z
f=(x+3)*(x+2)*x*(x-2)*(x-3)
z=x+dx
dx: 0.05
END
```

The commands

```
syst pol
store f
simu -3.6 3.6
ashow f
```

will then generate Fig. 6.

Simnon is also conveniently used to generate level curves, field plots and conformal maps. To illustrate such applications consider for example the problem

of finding the image of the lines arg z = $\pi/2$ and arg z = $3\pi/4$ in the conformal map

$$G(z) = e^{z} = e^{x=iy} = e^{x}(\cos y + i \sin y).$$

This may be accomplished by the system

```
discrete system EXPZ
time r
tsamp s
fi=alfa*pi/180
x=r*cos(fi)
y=r*sin(fi)
ReG=exp(x)*cos(y)
ImG=exp(x)*sin(y)
s=r+dr
dr: 0.01
pi: 3.1415926
alfa: 90
END
```

The commands

```
syst EXPZ
axes h -1 1 v 0 1.5
plot ImG(ReG)
simu 0 3.14
par alpha: 135
simu 0 4.44
```

then generate the graph shown in Fig. 7.

GENERALITIES

The general form of a description of a discrete time system is given in Listing 4. It is analogous to the continuous time system. The only semantic difference is that the sampling variable TSAMP must be assigned and that der is replaced by new.

SUMMARY

Difference equations are simulated in the same way as differential equations. The system description DISCRETE SYSTEM is used instead of CONTINUOUS SYSTEM.

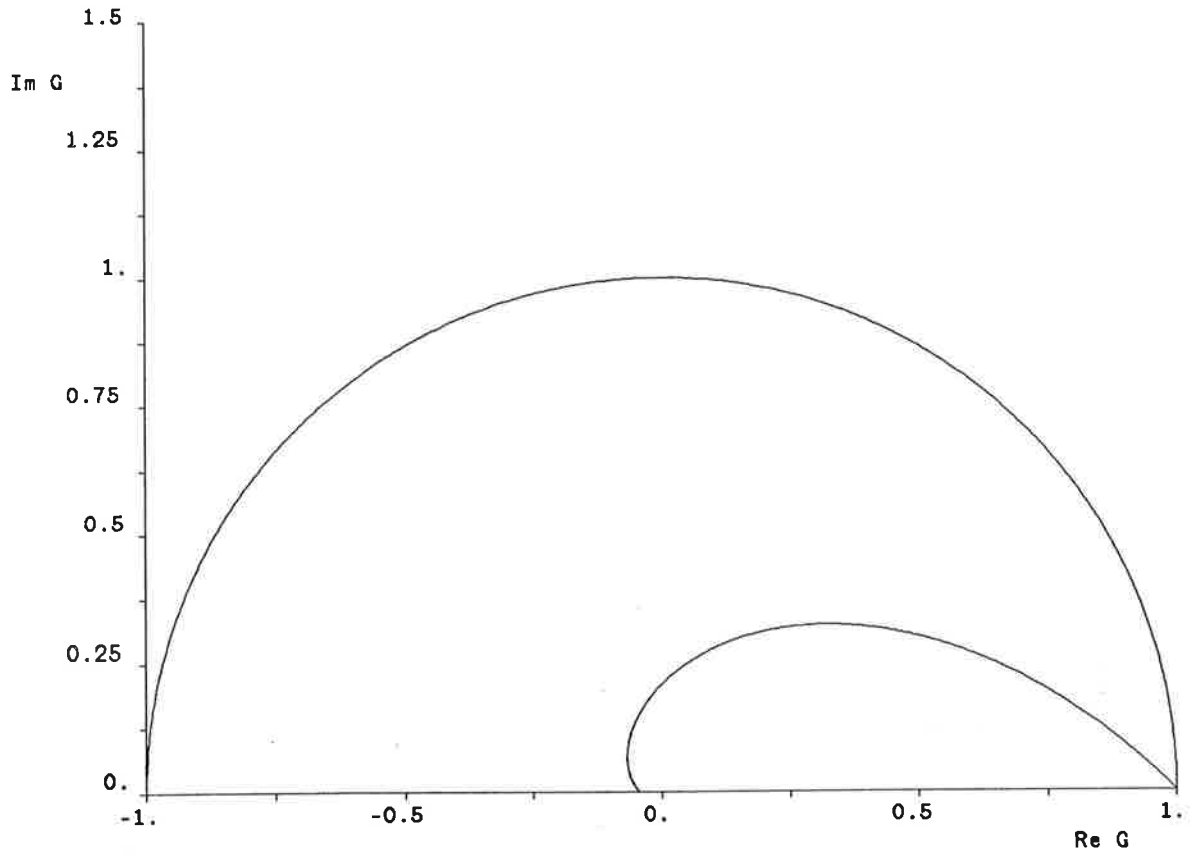


Figure 7. Graph of the map of the rays arg z = $\pi/1$ and arg z = $3\pi/4$ in the map $G(z) = e^z$.

```
discrete system <identifier>
"general difference equation                    comment
state  <identifier>...<identifier>   }
new    <identifier>...<identifier>   } declarations
time   <identifier>                  }
tsamp  <identifier>                  }
computation of auxiliary variables   }
computation of new values of the states } body
update the variable Tsamp.           }
parameter assignment                 }
initial value assignment             }
END
```

Listing 4. Structure of Simnon system for simulating difference equations.

The simulation commands are the same as for differential equations, i.e. six basic commands AXES, PLOT, INIT, PAR, SYST and SIMU and six auxiliary commands STORE, SHOW, DISP, SPLIT, AREA and HCOPY.

EXERCISES

1. The difference equation

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$$

is a well known algorithm for computing the square root of a. Write a discrete system in Simnon and explore the algorithm.

2. Modify the simulation program in exercise 1 so that the stationary solution to the equation is computed and plotted.

3. The following is a simple Keynesian model of a macro economic system

$$\begin{aligned} y(t) &= c(t) + i(t) + g(t) \\ c(t) &= a\ y(t-1) \\ i(t) &= b\ [c(t) - c(t-1)] \end{aligned}$$

where y is the gross national product, c consumption, i investment and g government expenditures. Write the equations as a system of first order equations and simulate the equations. Investigate the response of the system to a sudden increase in government spending. What are the influences of the parameters a and b? Try the values a = 0.75 and b = 0.5, 1 and 2.

4. Investigate the properties of the following difference equation

$$\begin{aligned} x(t+1) &= r(x^2 + y^2)\ \exp\ [-0.1(x^2 + y^2)] \\ y(t+1) &= x(t) \end{aligned}$$

for different values of the parameter r.

5. Combination of Systems

It is often useful to structure a large problem into smaller subproblems. In simulation this is done by decomposing a large system into interconnected subsystems. A subsystem is often represented as a box with inputs and outputs and the interconnections are represented by directed lines between the boxes. Such a structure can be represented in Simnon. This is done by adding declarations of inputs and outputs to the system descriptions. The subsystems can then be described as a CONTINUOUS SYSTEM or a DISCRETE SYSTEM. A special type of system is used to describe the interconnections. This system is called a CONNECTING SYSTEM.

Consider the control system shown in Fig. 8 which is a combination of two subsystems

```
discrete system REG
input yref y
output u
.
.
.
END

continuous system PROC
input u
output y
.
.
.
END
```

and the system which describes the interconnection is given by

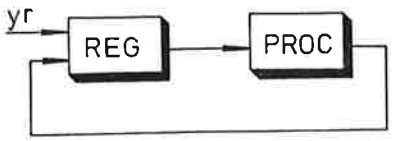


Figure 8. Block diagram of an interconnected system.

```
connecting system CON
"Connecting system for simulation of process PROC
"with PI regulation by system PIREG
time t
yr[pireg]=1
y[pireg]=y[proc]
u[proc]=u[pireg]
END
```

Notice that states, variables, and parameters are local variables in each subsystem. Variables in different subsystems may be specified by adding the system name in square brackets after the identifier. Also notice that expressions may be used to describe the interconnections. Constructions like

```
y[reg] = if t<1 then 0 else sin(k*y[proc])
```

are thus possible.

The simulation of an interconnected system is done using the same commands as was used to simulate difference or differential equations. The only difference is that it is necessary to activate all subsystems that describe the interconnected system. This is done by the command

```
SYST SYS1 SYS2 CON
```

The connecting system must be the last system in the list. The order of the systems is otherwise irrelevant. Continuous and discrete system may be mixed freely.

AN EXAMPLE - SIMULATION OF A COMPUTER CONTROL SYSTEM

A continuous time process with a computer control system is conveniently described as an interconnected system. The process may be represented as a CONTINUOUS SYSTEM and the control computer as a DISCRETE SYSTEM.

```
discrete system PIREG
"PI regulator with anti-windup
input yr y
output u
state i
new ni
time t
tsamp ts
e=yr-y
v=k*e+i
u=if v<ulow then ulow else if v<uhigh then v  else uhigh
ni=i+k*e*h/ti+u-v
ts=t+h
k: 1
ti: 1
h: 0.5
ulow: -1
uhigh: 1
END


connecting system CON
"Connecting system for simulation of process PROC
"with PI regulation by system PIREG
time t
yr[pireg]=1
y[pireg]=y[proc]
u[proc]=u[pireg]
END
```

Listing 5. Simnon description of a simple control loop consisting of a continuous time process and a discrete time PI regulator.

Listing 5 describes a feedback loop consisting of a continuous time process called PROC and a digital PI regulator called PIREG. The process is an integrator with input saturation. The interconnections are described by the connecting system CON.

The following annotated dialogue illustrates how Simnon is used.

Command	Action
syst proc pireg con	Activate the system.
store yr y[proc] upr	Select variables to be stored.
simu 0 40	Simulate.
split 2 1	Form two screen windows.
ashow y yr	Draw y and yr with automatic scaling in first window.
ashow upr	Draw upr with automatic scaling in second window.

Notice that the names are local to each subsystem. To distinguish between variables that occur in different subsystem the name of the subsystem is written in square brackets as in y[proc]. Variables can be transmitted between subsystem by declaring them as inputs and outputs.

The results of the simulation are shown by the oscillatory curves in Fig. 9. The discrete nature of the control actions generated by the computer are clearly visible in the curves. These curves show that there is a considerable overshoot due to windup at the integral. This is avoided by telling the regulator what the process limitations are. The commands

```
par  ulow: -0.1
par  uhigh: 0.1
```

changes the necessary parameters. The commands

```
simu 0 40
area 1 1
show y yr
area 2 1
show upr
```

shows that the overshoot is reduced significantly. Compare Fig. 9.

GENERALITIES

Simnon allows three types of system descriptions, namely CONTINUOUS SYSTEM, DISCRETE SYSTEM and CONNECTING SYSTEM. The discrete and the continuous

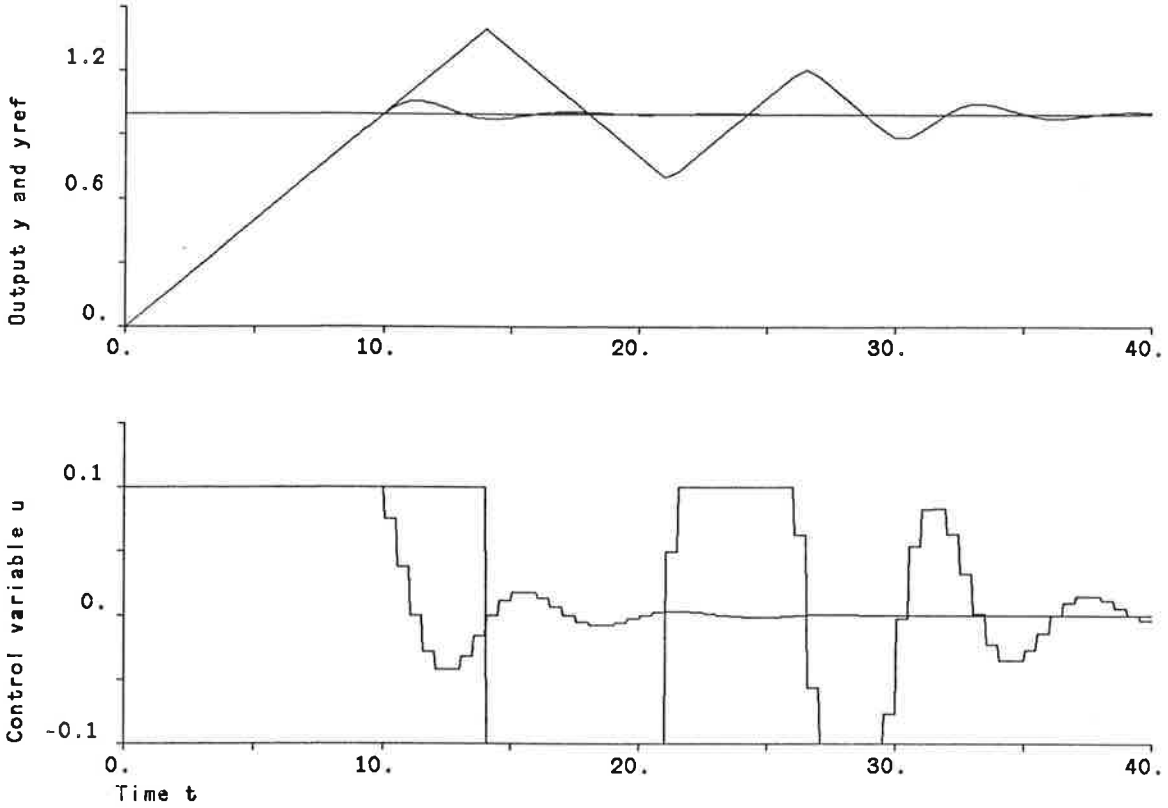


Figure 9. Results of simulation of process control with a PI regulator. The curves with a large overshoot correspond to an ordinary regulator. The other set of curves are obtained with a regulator with overshoot inhibition.

systems may be simulated individually provided that no inputs and outputs are declared. Interconnected systems may also be described by using the connecting system. The complete syntax for the system descriptions is given in Appendix B.

EXERCISES

1. Look at the syntax of the commands SYST and SIMU using the help command. What are the differences between simulation of single systems and interconnected systems.

2. The HELP command has an hierarchical structure. Explore this experimentally.

3. Use the command HELP LANGUAGE STRUCT to find the form of the different system descriptions.

4. Assume that the variable y is used in two subsystems in an interconnected system. Construct a simple test example to find out what happens. What diagnosis is produced? How can the variables be separated?

5. Consider the system in the example. Repeat the simulation on your own computer. Investigate the consequences of changing the sampling period.

6. Study the structure of the system descriptions.

6. Advanced Features

Simnon may be used in many different ways. So far we have described problems which may be solved by using only a dozen commands. This is sufficient for many applications. For more demanding problems there are however several additional features. These may all be explored by using the HELP command or by reading the manual. A brief description of some useful features will be given here to indicate some possibilities.

MACROS

Commands are normally read from the terminal. It is, however, useful to have the option of reading a sequence of commands from a file instead. The construction

```
MACRO NAME
Command1
Command2
Command3
END
```

thus indicates that the commands 1, 2 and 3 are not executed directly but stored on a file. The command sequence is then activated simply by typing NAME.

As an illustration we give the following macro which generates Fig. 9.

```
macro FIG9
"Generates Fig. 9
syst proc pireg con
store yr y[proc] upr
simu 0 40/wup
par ulow: -0.1
par uhigh: 0.1
simu /nowup
```

```
split 2 1
ashow y/wup
show yr y/nowup
ashow upr/wup
show upr/nowup
mark a 2.5 0
mark "Time t
mark a 1 1
mark v "Control variable u
mark a 1 7.5
mark v "Output y and yref
END
```

Notice that lower case letters may be used to get a more readable code, although Simnon does not distinguish them from upper case letters. Macros for generating all figures for this report are given in Appendix D.

Macros are useful for documentation. They are also convenient for simplification of a dialogue. Command sequences, that are commonly used, may be defined as macros. A simple macro call will then activate a whole sequence of commands. Macros may also be used to generate new commands.

The usefulness of macros may be extended considerably by introducing commands to control the program flow in a macro, facilities for handling local and global variables and by allowing macros to have arguments. By having commands for reading the keyboard and for writing on the terminal it is also possible to implement menu driven dialogues using macros.

Even a casual user is strongly recommended to learn simple uses of the macro facility.

INTEGRATION ALGORITHMS

Differential equations are solved in Simnon using numeric integration routines. A predictor corrector formula by Hamming with automatic step length adjustment is normally used. The initial value of the step length is chosen as one hundredth of the integration interval. A different initial step size may be chosen by an optional argument in the command SIMU. In the algorithm the step length is reduced until the difference between the prediction and the correction is sufficiently small. The tolerance may be set by the command ERROR.

It is also possible to choose other integration algorithms by the command

```
ALGOR {HAMPC|RK|RKFIX|DAS}
```

where HAMPC is Hammings predictor corrector method, RK is a fourth order Runge-Kutta algorithm with automatic steplength adjustment. RKFIX is also a fourth order Runge-Kutta algorithm but it has a fixed step length. DAS is an algorithm for integration of stiff equations, i.e. differential equations with both slow and fast modes. Further details on the different algorithms are given in the Simnon manual.

FORTRAN SYSTEMS

The Simnon language is simple, easy to use and reliable because of all the diagnostics that is built into it. The language has however a limited expression power. There are no possibilities to control program flow, there are no arrays or other data types and there are no procedures. For models whose descriptions require a more powerful language it is possible to interface Fortran routines to Simnon. This also makes it possible to use library subroutines like Eispac and Linpac in the simulations. The Simnon manual describes in detail how to do this.

STANDARD SYSTEMS

There are several standard systems in Simnon. A list of the available systems is normally displayed on the screen when the system is started. A few of the systems are listed below.

DELAY	- time delay
FUNC1	- nonlinearity defined by a table
LOGGER	- logging of stored variables
NOISE1	- white gaussian noise generator
OPTA	- optimizer
IFILE	- discrete system which reads output variables from file

There are also more sophisticated systems like linear quadratic gaussian regulators, self-tuning regulators in some implementations. A description of the standard systems is given in the manual. The command HELP SYSTEMS also gives

information about the systems. The standard systems are often implemented as Fortran systems.

GLOBAL VARIABLES

All variables in Simnon systems are local. When using Fortran systems or standard systems it may, however, be necessary to transfer global data. Global variables are used for this purpose. The global variables are set by a LET command. An example illustrates the use of global variables and standard systems.

Example 6.1 - Use of standard systems

Assume that we want to include a function defined by a table in a simulation. This is conveniently done by the function FUNC1. This function has one input u and one output y. The function has one global parameter N.FUNC1 which gives the number of entries in the table. The arguments are specified by the local parameters ui1, ui2, ... The corresponding function values are specified by the arguments gi1, gi2, ... The local parameter ORDER specifies staircase (order = 0) or linear (order = 1) interpolation.

The following dialogue illustrates how the function may be used.

```
let n.func1 = 4
syst FUNC1 FUNCPLOT
par ui1: -3
par gi1: -1
par ui2: -1
par gi2: -2.8
par ui3: 1
par gi3: 1.5
par ui4: 4
par gi4: 2
par order: 1
```

The global parameter n.func1 must be assigned a value before the system is activated by the SYST command. The local parameters may be changed by the PAR command as parameters in ordinary Simnon systems.

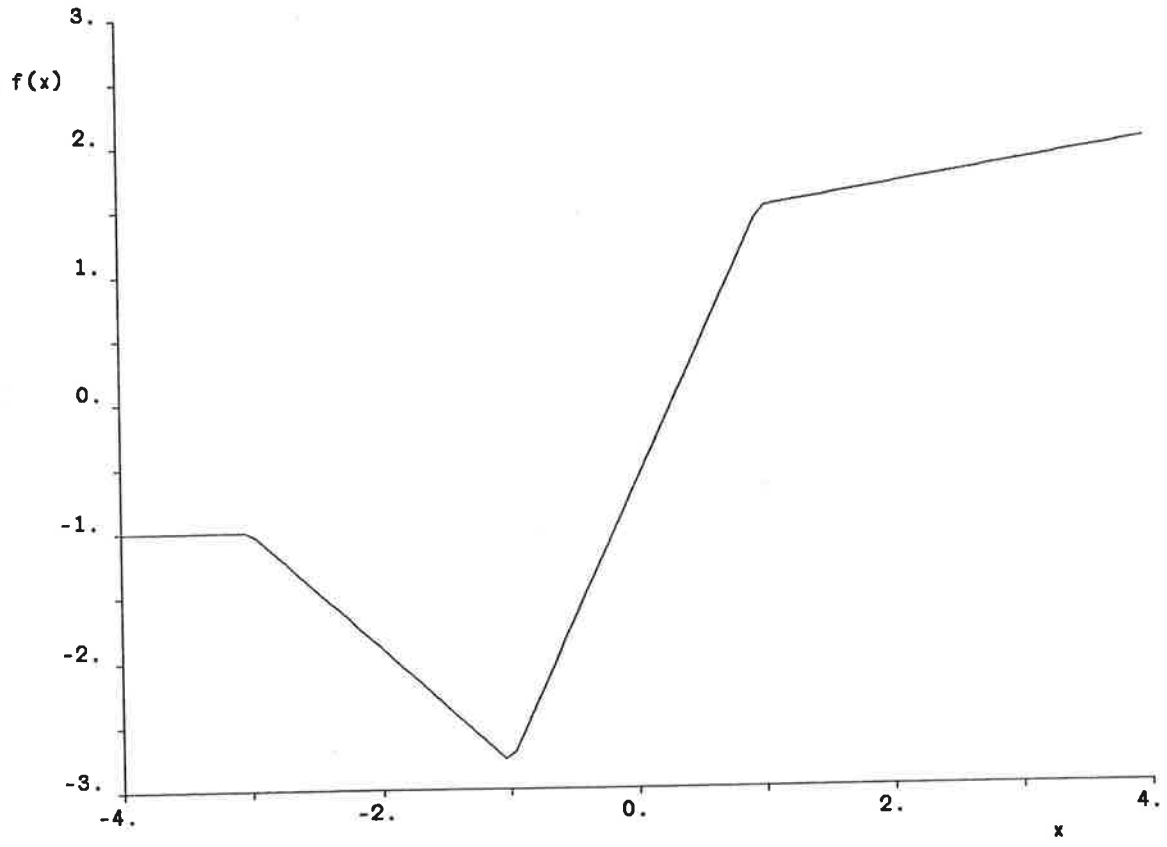


Figure 10. Graph of a function defined by a table generated using the Simnon function FUNC1.

With

```
connecting system FUNCPLOT
time x
u[FUNC1]=x
yp=y[FUNC1]
END
```

the commands

```
axes h -4 4 v -3 3
plot yp
simu -4 4
```

then generates a graph of the function, see Fig. 10.

ORGANIZATION OF LARGE SIMULATIONS

There are several facilities which are useful when working with large systems or with large amounts of data. System descriptions and macros are stored as text files. These files have the file extension .T on the VAX/VMS system. The subsystems also have a name which is the identifier given on the first line of the system description. See Appendix B. Notice that the file name and the system name may be different. This is very useful when simulating different versions of a model because the same macros and the same connecting systems may be used. The selection of a particular model is done when the systems are activated by the command SYST. An illustration is given in the macro FIG9 which is listed in the appendix. A PI regulator with the name REG is stored in a file called PIREG and a system with the name PROC is stored in a file called INTEGR. The systems are activated by the command

```
SYST INTEGR PIREG CON
```

The variables of the system are labeled by [REG] and [PROC] respectively. This makes it possible to use a standard connecting system for different processes and regulators.

SAVING AND RETRIEVING PARAMETERS AND INITIAL VALUES

The command SAVE stores parameters and initial values in a file. The values may be retrieved by the command GET. These commands are very useful when working with large models, because parameters and initial values are not introduced manually. Assume for example that the systems FUNC1 and FUNCPLOT have been activated as in Example 6.1 by the command

```
SYST FUNC1 FUNCPLOT
```

The command

```
SAVE FUNCPAR
```

then generates a file FUNCPAR.T of the form

```
[FUNC1]
UI1: -3.
UI2: -1.
UI3: 1.
UI4: 4.
GI1: -1.
GI2: -2.8
GI3: 1.5
GI4: 2.
order: 1.
[FUNCPLOT]
```

It is convenient to edit files of this type for parameters and initial values when simulating large systems.

DOCUMENTATION

It is often useful to keep a running log of an interactive session. The command

```
SWITCH LOG ON
```

generates a file of all commands in a session.

The scale factors in the graphs are computed by an algorithm. The same axes are obtained in horizontal or vertical direction if the horizontal range is divisible by four and the vertical by 3. It is possible to get scale factors 1, 2 and 5 only by the command

```
TURN S125 ON
```

It is possible to add text to the axes by the command MARK. This is used in the macros which generate the curves in the report.

It is also useful to have access to operating system commands under Simnon. In the Vax implementation this is done simply by typing $ followed by any operating system command.

7. Implementation

When using Simnon it is helpful to have some insight into how the program works and how it is implemented. A brief description is given in this chapter.

HOW A SIMULATION IS ORGANIZED

When the command SYST is given the equations describing the system to be simulated are first sorted so that all calculations are done in the correct order. It is possible to store the sorted equations in a text file by using the optional argument [/<filename>]. The command

```
SYST INTEGR PIREG CON/DUMMY
```

generates a file called DUMMY with the content:

```
SORTED INITIAL EQUATIONS

SORTED TIME-INDEP. EQUATIONS
CON       yr[REG] = 1

SORTED DERIVATIVE EQUATIONS
PROC      y = x
CON       y[REG] = y[PROC]
REG       e = yr - y
          v = k*e + i
          u = IF v<ulow THEN ulow ELSE IF v<uhigh THEN v ELSE uhigh
CON       u[PROC] = u[REG]
PROC      upr = IF u<-0.1 THEN -0.1 ELSE IF u<0.1 THEN u ELSE 0.1
          dx = upr

SORTED CONTINUOUS EQUATIONS

SORTED DISCRETE EQUATIONS
REG       ni = i + k*e*h/ti + u - v
          ts = t + h
```

The subsystem name is given to the left and the equation to the right. The equations are sorted so that the calculations may be performed in sequential order. The equations are then brought to the standard form of a system of first order differential equations, which are then integrated using the chosen integration routine. The state variables of discrete time systems are constant between the sampling instants. They may change discontinuously at the sampling instants. The integration is therefore carried out only to the nearest sampling instant. The states of the discrete systems are then changed and the integration is continued.

ALGEBRAIC LOOPS

When systems are interconnected it may happen that the equations can not be sorted so that the variables may be obtained by sequential computations. A simple example, which illustrates what may happen is given below.

```
continuous system S1
Input u
Output y
y = u
END

continuous system S2
Input u
Output y
y = u
END

connecting system C
yr = 1
u[S1] = yr + k*y[s2]
u[S2] = y[S1]
k: 0.5
END
```

When the command

```
SYST S1 S2 C
```

is given the error diagnosis "algebraic loop detected" is given. The attempt to sort the equations result in

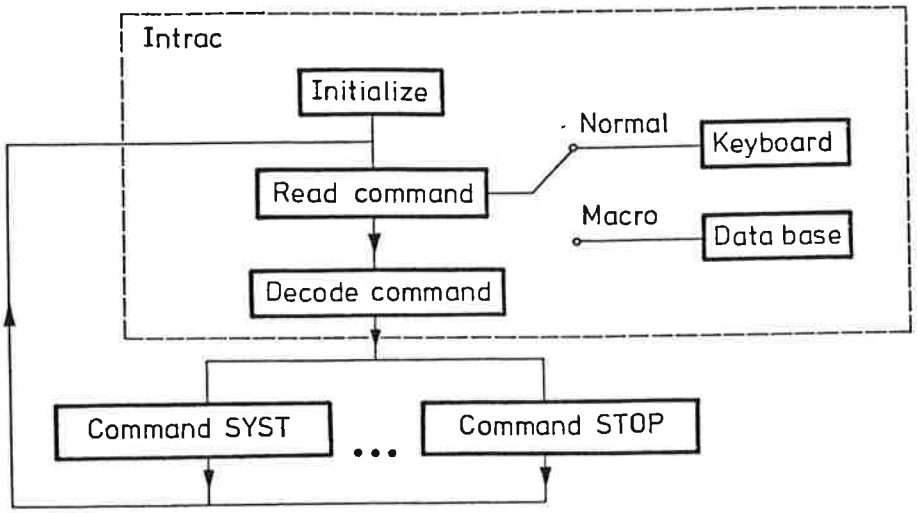


Figure 11. Skeleton flow chart for Simnon.

```
S1    y = u
C     u[S2] = u[S1]
S2    y = u
C     u[S1] = yr + k*y[S2]
S1    y = u
```

The variable u[S1] thus can not be solved sequentially. A simple calculation shows that the variable is given by the following algebraic equation:

```
u[S1] = yr + k*u[S1]
```

This explains the name algebraic loop. In this particular case it is easy to solve the equation if $k \neq 1$. In the general case the algebraic equation may be nonlinear. It is then difficult both to determine if a solution exist and to find it if it does. Simnon will not attempt to find a solution. It will just give an error message. The presence of an algebraic loop is often an indication of poor modeling. Models with algebraic loops may be integrated with the routine DAS.

HOW THE PROGRAM IS ORGANIZED

A schematic flow chart for Simnon is shown in Fig. 11. The main loop reads a command, decodes it and performs the required actions. All parts of Fig. 7 except the action routines are implemented as a package of subroutines called Intrac. These subroutines perform command decoding, file handling, listing and plotting. Intrac also contains the macro facility. A summary of the commands in Intrac and

Simnon is given in Appendix E.

The source code for Intrac is about 7000 lines of Fortran code, which compiles to about 90 kbytes of code. Simnon itself has a source code of about 25 000 lines of Fortran code, which compiles into 360 kbytes of code.

8. References

Simnon was developed in connection with a project for computer aided design of control systems. See

Åström, K J (1983):
Computer aided modeling, analysis and design of control systems - A perspective. Control Systems Magazine, May, pp 4-16.

Simnon inherited many ideas from the tradition of analog and digital simulation in the control engineering field. When the work on Simnon was started there were a number of digital simulation languages like CSSL and CSMP available. These are described in the book

Korn, G. A. (1978):
Digital Continuous System Simulation. Prentice Hall. Englewood Cliffs, New Jersey

which contains much useful information on simulation. Languages like CSMP and CSSL were, however, implemented using batch calculations. Since the languages largely were inspired by analog simulation techniques they also inherited several constraints imposed by the analog hardware. For example the state space notation which is so natural was not supported by proper language constructs. When Simnon was developed the main idea was to provide a tool which supports the state space notation with good man-machine interaction based on interactive computing.

The first version of Simnon was defined and implemented by Hilding Elmqvist as an MS dissertation in 1972. The work was continued by Elmqvist and a usable language and a manual were available in 1975:

Elmqvist, H (1975):
SIMNON, an interactive simulation program for nonlinear systems.

Report TFRT-3091, Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden.

A good description of Simnon is also found in

Elmqvist, H (1977):
SIMNON - An interactive simulation program for nonlinear systems. Simulation '77, Montreux, Switzerland, June 1977.

A new Simnon manual by Elmqvist is in preparation.

The Simnon language has been used extensively in teaching at the department of Automatic Control at Lund Institute of Technology ever since. It is increasingly being used at other schools and industries.

An innovative use of Simnon is given in the textbook

Åström, K J and Wittenmark, B (1984): Computer Control Theory.

All graphs for simulation results in the book are generated by Simnon Macros which are accessible to the students. They can then easily change parameters and modify graphs.

The interaction principles used in Simnon, which are based on commands and macros were developed in a more general CAD context. See

Wieslander, J (1979):
Interaction in computer aided analysis and design of control systems. PhD thesis, Report TFRT-1019, Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden.

and

Wieslander, J (1980a):
Interactive programs - General guide. Report TFRT-3156, Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden.

The program Intrac, which is the core of the interaction with the user, is described in

Wieslander, J and Elmqvist, H (1978):
INTRAC, A communication module for interactive programs. Language manual. Report TFRT-3149, Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden.

APPENDIX A

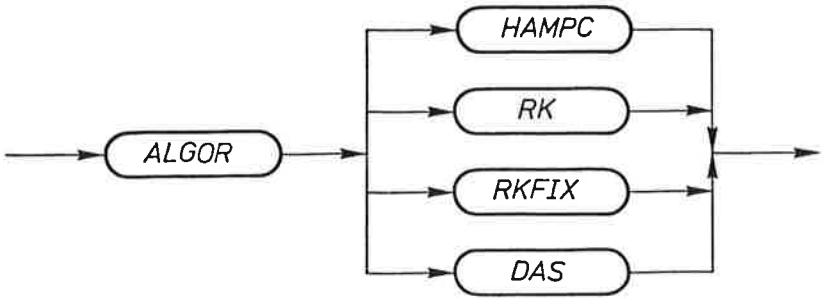
Syntax for Simnon Commands

A list of the commands, their syntax and brief descriptions are given below. Some Intrac commands (denoted by †) are also included in the list. More details about the commands are obtained using the command HELP. All Intrac commands are listed in Appendix D. The list is valid for implementations on VAX-11 systems. The following notations are used:

{op1 | ... | opn}	Defines different alternatives of which one must be given.
[....]	Parts within squared brackets are optional and can be omitted.
{....}*	A star indicates that the previous part can be repeated.
<....>	Defines arguments for the commands.

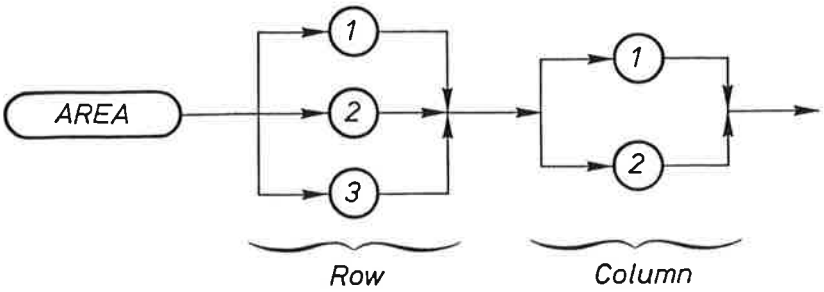
ALGOR {HAMPC|RK|RKFIX|DAS}

To select integration routine.
HAMPC - Hamming predictor corrector (default)
RK - Runge-Kutta variable step size
RKFIX - Runge-Kutta fixed step size
DAS - Integration routine for stiff systems



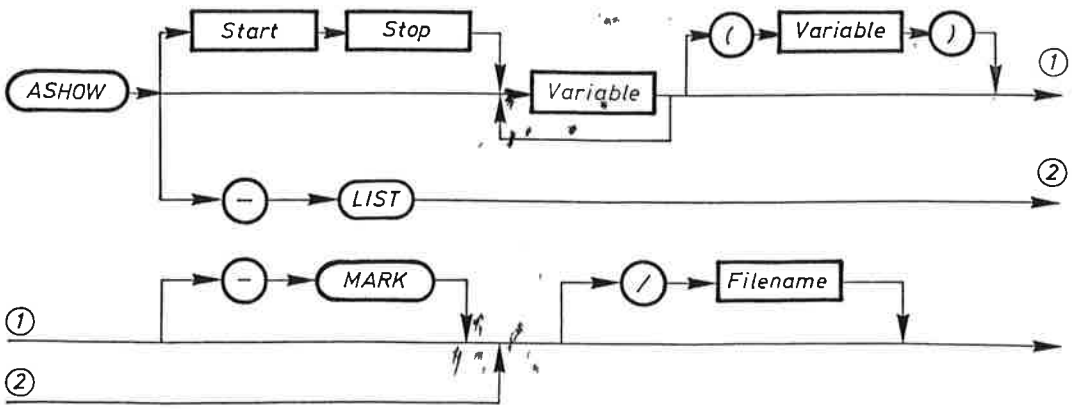
AREA <row> <column>

To define the plotting area to be used next, see SPLIT.



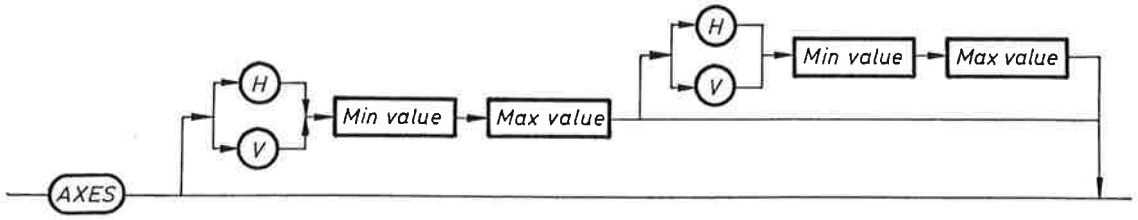
ASHOW {[<start><stop>] {<variable>}* [(<variable>)] [-MARK]|-LIST}
[/<filename>]

To plot stored variables with automatic scaling. Similar to SHOW.



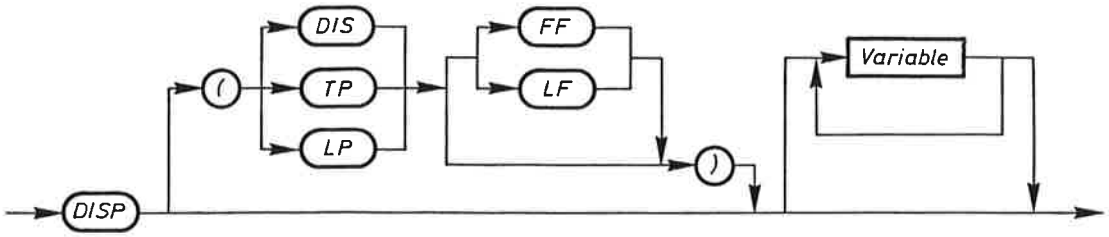
AXES [<axis spec> [<axis spec>]]

To draw axes.
<axis spec>::= {H|V} <min value> <max value>
H - horizontal, V - vertical



DISP [({DIS|TP|LP}[{FF|LF}])] [{<variable>}*]

To display variables.
DIS - display (default)
TP - teleprinter
LP - line printer
FF - form feed (default when no variables are specified)
LF - line feed (default when variables are specified)
If no variables are given all variables are displayed.



EDIT <filename>

To edit a file. The editor has two modes, INPUT and EDIT. The mode is changed by entering an empty line. In the edit mode it is possible to change the current file. The following commands are available:

A[PPEND] <string>	- Append string to current line
B[OTT]	- Line pointer to bottom
C[HANG] /x/y/	- Replace string x by string y
D[EL] <integer>	- Delete n lines
DIS [ON|OFF]	- Echo check enable/disable
E[XIT]	- Leave editor with updated file
F[IND] <string>	- Find string at the beginning of a line
I[NS] <string>	- Insert a line below current line
L[OC] <string>	- Locate string anywhere in a line
LEAVE	- Leave the editor without update
N[EXT] <integer>	- Move line pointer down n lines
O[VERL] <integer>	- Overlay n lines by new text
P[RINT] <integer>	- Print n lines on display
R[ETYP] <string>	- Retype current line
T[OP]	- Move line pointer to the top of the file

ERROR <error bound>

To choose error bound for integration routine. Default value is 0.001.

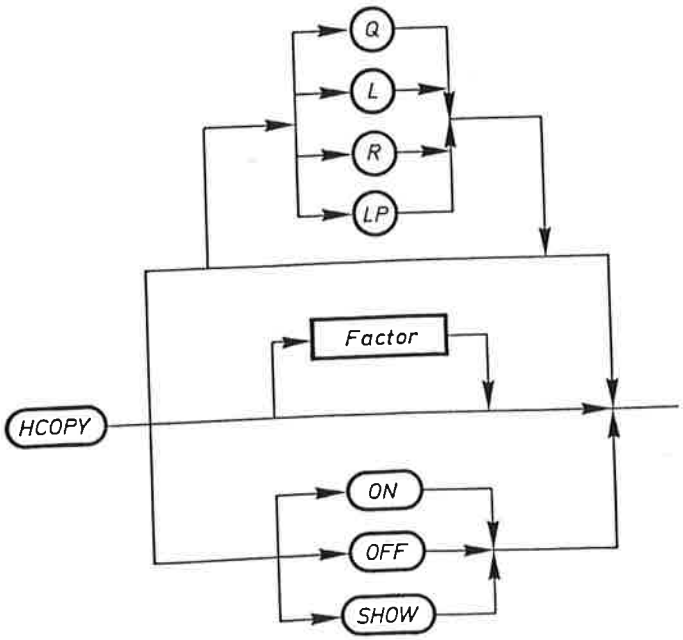
GET <filename>

To get parameter values and initial values from a file that previously has been stored using SAVE or edited.

HCOPY {[<MODE>]|[<factor>]|<SWITCH>}

MODE::={Q|L|R|LP}
SWITCH::={ON|OFF|SHOW}

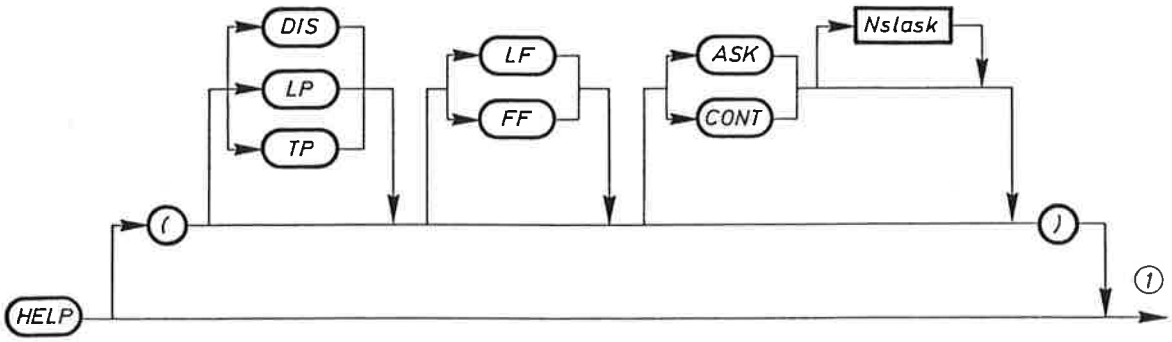
Make a hardcopy of curves on display. The hardcopy is scaled with <factor>, which may be in the interval (0.5, 1.6). The comment is obtained on the hardcopy too.

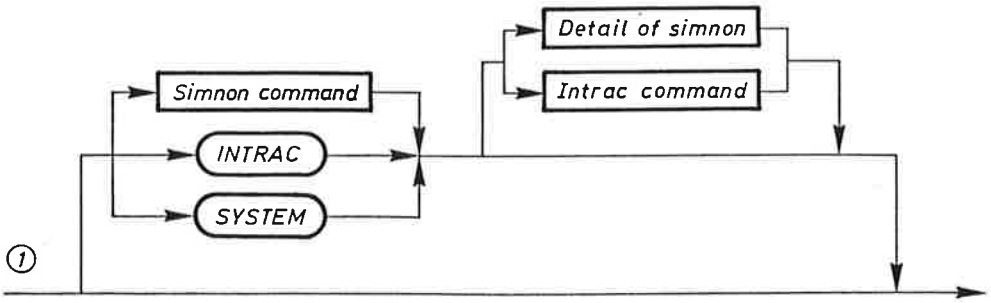


HELP [([DEV][FEED][PROMT[NSLASK]])][KEY1[KEY2..]..]

DEV::={DIS|LP|TP}
FEED::={LF|FF}
PROMT::={ASK|CONT}
KEY1::={SIMNON COMMAND|INTRAC|SYSTEM}
KEY2::={DETAIL OF SIMNNON|INTRAC COMMAND}

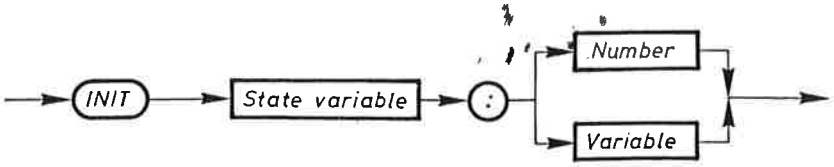
To get more information about the commands, the editor, Intrac, Simnon, and the standard systems. A menu of commands is obtained by typing HELP.





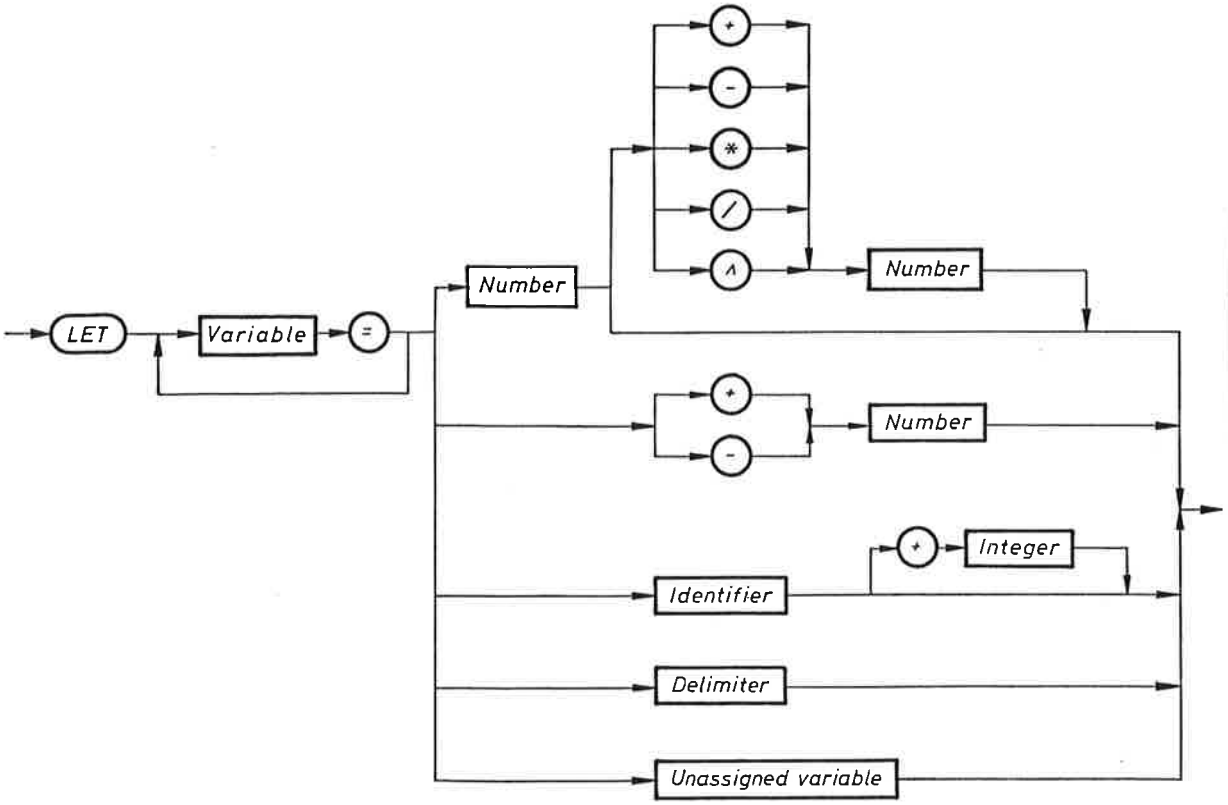
INIT <state variable>:{<number>|<variable>}

To change the initial value of a state variable.



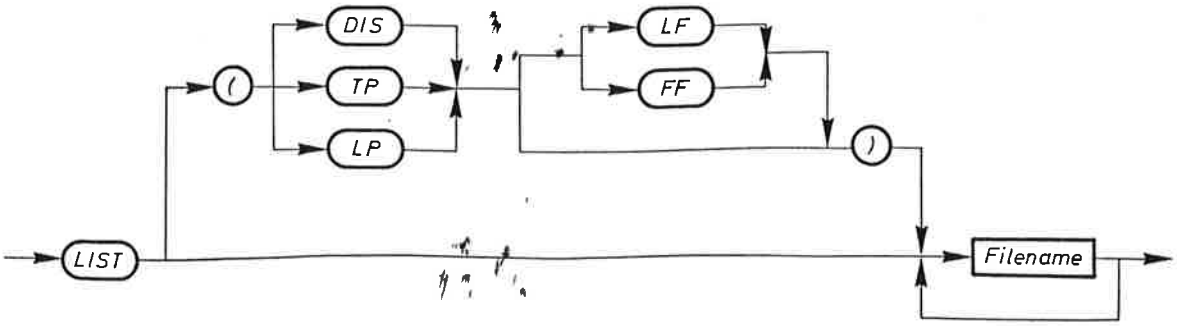
LET {<variable>=}* {<number> [<operator> <number>]
|{+|-} <number>| <identifier> [+<integer>]
|<delimiter> |<unassigned variable>}
<operator>::= {+|-|*|/|^}

An Intrac command that also is used to set global parameters for the standard systems.



LIST [({DIS|TP|LP}[{FF|LF}])] {<filename>}*

To list textfiles. The first arguments are the same as for the command DISP. When typing the command LIST the files are sent to a print file. Type the command LP to initiate the printing.



LP

Initiate listing on lineprinter of the print file generated by the commands DISP, LIST, LOG or PRINT.

MARK

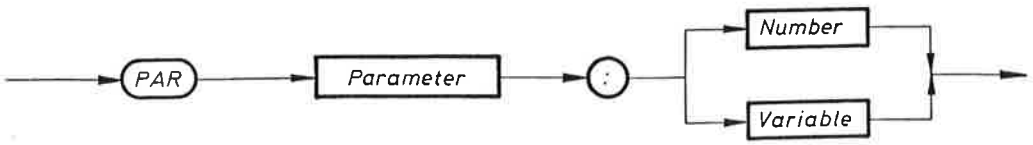
To introduce text into a plot. For syntax type HELP INTRAC MARK.

NEWS

To obtain news about Simnon.

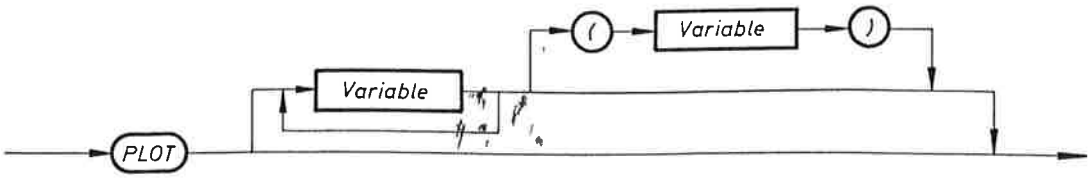
PAR <parameter> : {<number>|<variable>}

To change a parameter value.



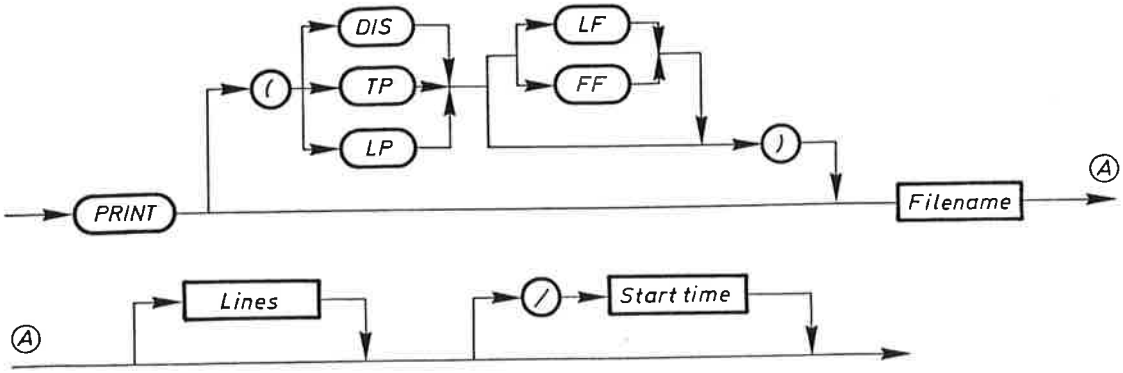
PLOT [{<variable>} * [(<variable>)]]

To select variables to be plotted when making the command SIMU.
Examples: PLOT X1 X2 gives X1 and X2 as functions of time while PLOT X1(X2) gives X1 as function of X2.



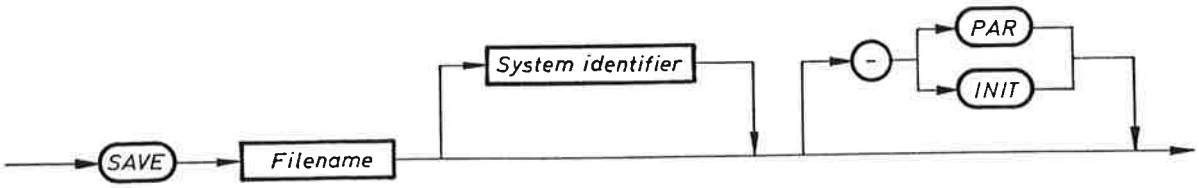
PRINT [({DIS|TP|LP} [{FF|LF}])] <filename> [<lines>]
[/<start time>]

To list file generated with the commands STORE+SIMU. <lines> lines starting from <start time> will be printed. The other parameters are the same as for DISP.



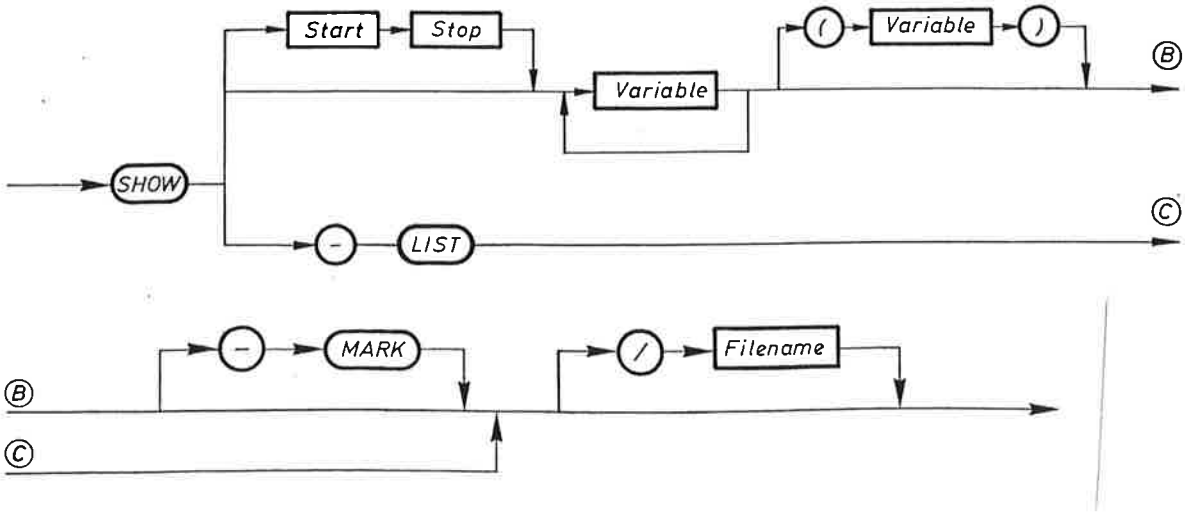
SAVE <filename> [<systemname>] [-{PAR|INIT}]

To save parameter values and initial values for a given system <systemname> on a file named <filename> to be used by the command GET. Only parameters or initial values are saved with the option PAR or INIT.



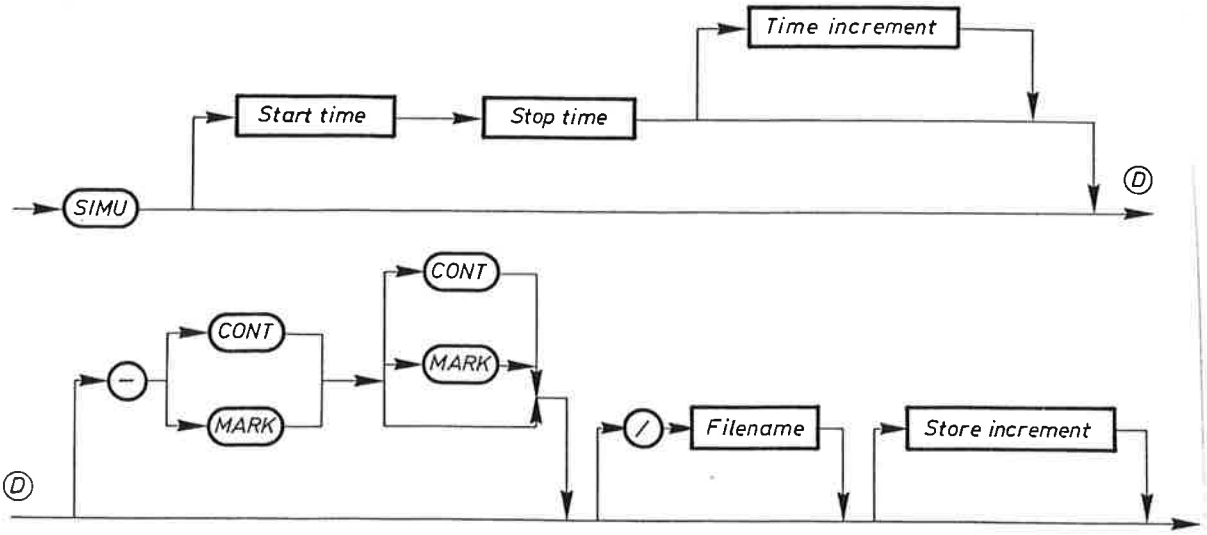
SHOW {[<start><stop>] {<variable>}* [(<variable>)] [-MARK]|-LIST}
[/<filename>]

To plot stored variables from file <filename>. To be used with the command STORE. The specified variables are plotted from <start> to <stop> time. If MARK is used the different variables are numbered on the plot. The LIST-option lists the names of all variables.



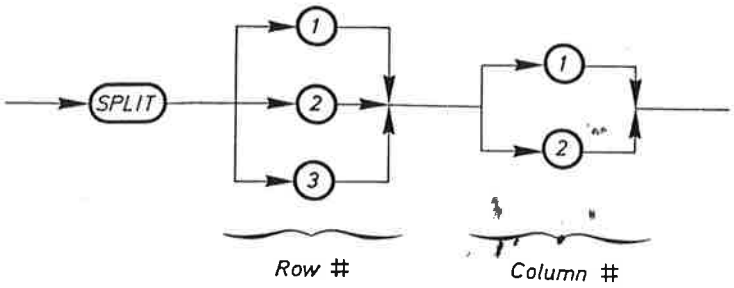
SIMU [<start time> <stop time> [<time increment>]]
[-{CONT|MARK}[CONT|MARK]] [/<filename>][<store increment>]

To simulate the system from <start time> to <stop time> using the maximum stepsize <increment> (default (stop time - start time)/100). Using MARK the variables defined by PLOT are numbered. With CONT the simulation is continued with the previously obtained state variables as initial values. When specifying <filename> then the plotted variables are stored in <filename> with <increment> as sampling interval.



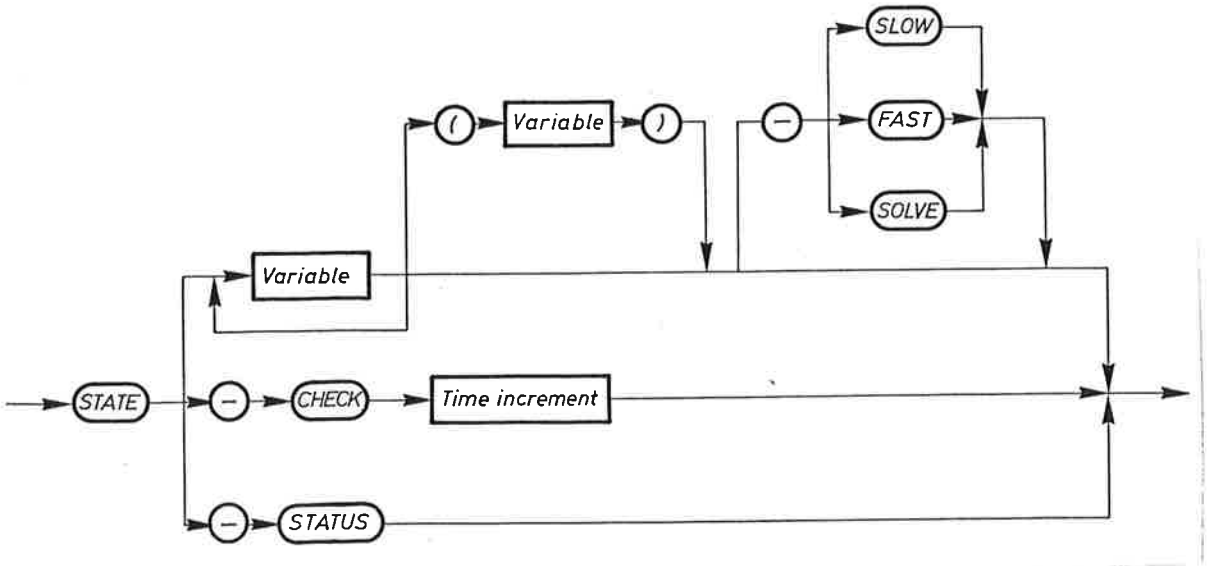
SPLIT <rows> <columns>

To split the screen into maximum six plotting areas.
<rows>:={1|2|3} default 1
<colums>:={1|2} default 1



STATE {{<variable>}* [(<variable>)] [-<option>]|
-CHECK <time increment> |-STATUS}
<option>::={SLOW|FAST|SOLVE}

A special command, which is only used with the integration routine DAS.

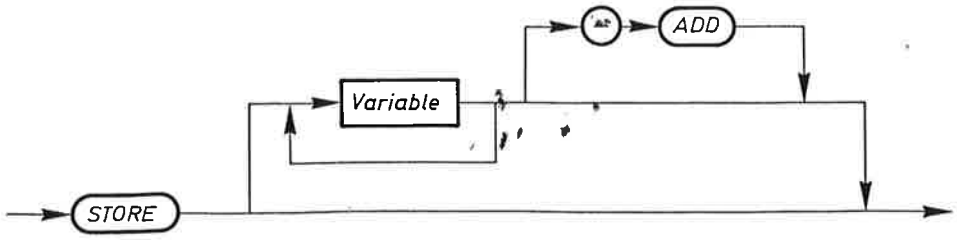


STOP †

To leave Simnon.

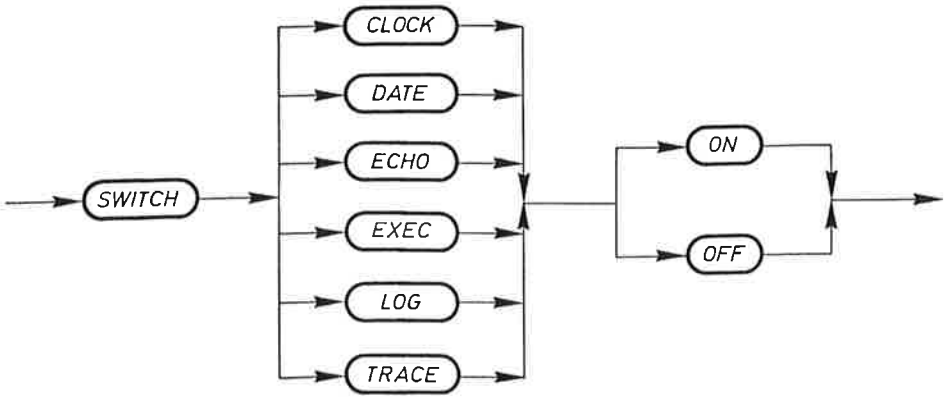
STORE [{<variable>}* [-ADD]]

To select variables to be stored at each simulation. With ADD new variables can be added to a previously defined list of variables. The variables can be displayed using ASHOW or SHOW and printed using PRINT.



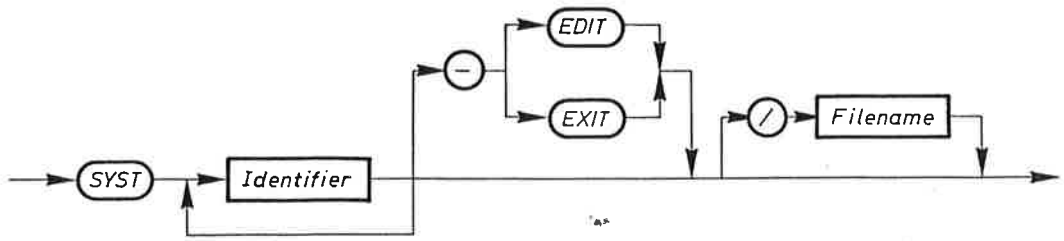
SWITCH {CLOCK|DATE|ECHO|EXEC|LOG|TRACE} {ON|OFF} †

To control the execution of Intrac.
CLOCK - Adds time to hardcopy output. Default OFF.
DATE - Adds date to hardcopy output. Default OFF.
ECHO - Macro commands are echoed. Default OFF.
EXEC - Commands executed while typed during macro generation. Default OFF.
LOG - Executed commands are logged on line printer. Default OFF.
TRACE - Affects ECHO and LOG.



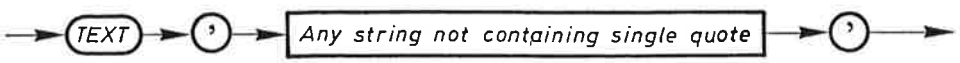
SYST {<identifier>}* [-<option>] [/<filename>]
<option>:={EDIT|EXIT}

To define the system. The subsystems are compiled. If there are several subsystems the last one has to be a connecting system. EDIT means that the compiler goes into the editor for each file. If filename is specified the sorted equations are written into a text file.



TEXT '<any string not containing single quote>'

To include text on paper plot.



TURN {DARK|DIS|NOCODE|OVFLO|PLCOM|S125|TIMING} {ON|OFF}

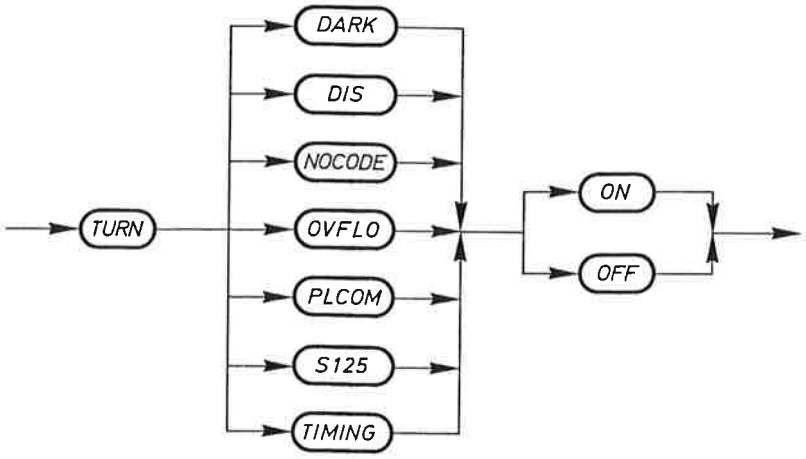
To turn on and off switches.

S125 Selects a limited set of scale factors (1., 2., 5.) when choosing scales on axes. Default OFF.

DARK ON means that plotted curves will not be connected between the sampling instants. Default OFF.

DIS Informs SIMNON if the user has a graphic display. Default ON.

OVFLO ON means that the simulation stops if overflow occurs in the calculations. Default OFF.



APPENDIX B

Syntax of System Descriptions

Simnon allows three types of systems: CONTINUOUS SYSTEM, DISCRETE SYSTEM and CONNECTING SYSTEM. The following Bachus-Naur notations are used to describe the syntax.

< >	syntactic unit
::=	denotes
|	exclusive or
[ ]	optional element
{ }	compulsory element
*	repetition

The following syntactic elements are needed to describe the systems.

LETTERS

```
<letter>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|X|Y|Z
           a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|x|y|z
<digit>::= 0|1|2|3|4|5|6|7|8|9
```

IDENTIFIERS

```
<identifier>::=<letter>|<identifier><letter>|
               <identifier><digit>
```

VARIABLES

```
<system identifier>::=<identifier>
<simple variable>::=<identifier>
<variable>::=<simple variable>|<simple variable>[system
             identifier>]
```

The generic form of the system descriptions are:

```
CONTINUOUS SYSTEM   <system identifier>
[INPUT  <simple variable>*]
[OUTPUT <simple variable>*]
[STATE  <simple variable>*]
[DER    <simple variable>*]
[TIME   <simple variable>]

[INITIAL
Computation of initial values for state variables
SORT]

[Computation of auxiliary variables]
[Computation of output variables]
[Computation of derivatives]
Parameter assignment
[Initial value assignment]
END


DISCRETE SYSTEM  <system identifier>
[INPUT  <simple variable*]
[OUTPUT <simple variable>*]
[STATE  <simple variable>*]
[NEW    <simple variable>*]
[TIME   <simple variable>]      "denotes the new values of
                                 the states
TSAMP   <simple variable>       "denotes the next sampling
                                 instant

[INITIAL
[Computation of initial values for state variables]
[Computation of initial values for output variables]
[Computation of initial values for the TSAMP-variable]
SORT]


[Computation of auxiliary variables]
[Computation of output variables]
[Computation of new values of the states]
Updating of the TSAMP-variable
[Modification of states in continuous subsystems]
Parameter assignment
[Initial value assignment]
END
```

```
CONNECTING SYSTEM  <system identifier>
[TIME  <simple variable>]
[Computation of auxiliary variables]
[Computation of input variables]
[Parameter assignments]
END
```

Note that the order of the computations and assignment is unimportant because the equations will be sorted automatically. The section Initial makes it possible to compute initial values to state outputs and TSAMP. These variables can normally not be assigned an expression.

APPENDIX C
Standard Systems

This appendix describes the standard systems DELAY, FUNC1, IFILE, LOGGER, NOISE1, OPTA, and STIME. These systems are written in Fortran and linked into Simnon.

DELAY

This discrete time system simulates pure time delays. Old values of the signal to be delayed are stored in a vector. Delayed values are then generated by interpolation. The system admits two interpolation schemes due to Hermite and Aitken. In Aitken's scheme a Lagrange polynomial is fitted to the stored values. In Hermites method the values of the derivatives are stored together with the function values. The delayed values are then determined using an interpolation polynomial, which agrees with the function and its derivative at the stored points. Hermite's method requires that derivatives of the function are also stored.

The function admits delay of many signals. It has the following global variables.

n1.delay	Number of variables using Hermit interpolation
n2.delay	Number of variables using Aitken interpolation
space.delay	Number of elements in the allocation area to be used for saving old values. Try to use as many as possible, SIMNON gives an error message if too much space is used.

These variables must be assigned by a LET command before the system is activated.

The system has the following local variables.

INPUT:

u1,u2,...	Variables to be delayed (n1+n2)
du1,du2,...	Derivatives of the variables (n1)
td1,td2,...	Delay times (see below) (n1+n2)

OUTPUT:

y1,y2...	The delayed variables (n1+n2)

The outputs from the system DELAY are all zero for t<td, but other values can easily be set in the equations (see example below). The following example illustrates how the system DELAY can be used.

Example: Assume that the following systems have been defined.

```
continuous system SYS1
  ...
END

connecting system CONN
time t
td1[delay]=t-5
u1[delay]=sin(t)
u[sys1] = if t<5 then 1 else y1[delay]
END
```

The global variables for the system delay are assigned by the commands

```
let n1.delay=0
let n2.delay=1
let space.delay=450
```

The systems are activated by the command

```
syst sys1 delay conn
```

The signal u[sys1] is then given by

$$u[sys1]\ (t) = \begin{cases} 1 & \text{if } t < 5 \\ \sin(t-5) & \text{if } t \geq 5 \end{cases}$$

FUNC1

This is a continuous time function which makes it easy to introduce functions in tabular form. The system has one global variable:

```
n.func1
```

which gives the number of tabulated function values. The local variables are:

INPUT:	
u	Argument value
OUTPUT:	
y	Function value
PAR:	
ui1, ui2, ...	Table of argument values
gi1, gi2, ...	Table of function values
order	Order of interpolation (0 or 1)

The following example illustrates how the function is used.

Example: Assume that the following systems are defined.

```
continuous system SYS1
 ...
END

connecting system CONN
time t
u[func1]=t
u[sys1]=y[func1]
END
```

The global variable of the system FUNC1 is assigned by the command

```
let n.func1=5
```

The systems are activated by the command

```
syst sys1 func1 conn
```

The parameters of the system FUNC1 are defined by the commands

```
par ui1: 0
par gi1: 0
par ui2: 1
par gi2: 1
par ui3: 2
par gi3: 4
par ui4: 3
par gi4: 9
par ui5: 4
par gi5: 16
par order: 1              "Use linear interpolation
```

The parameters of the system FUNC1 are saved on file TAB1.T by the command

```
save tab1 func1
```

Another illustration is given in Example 6.1.

IFILE

This is a discrete time system which reads variables stored in a datafile. It is useful for example when comparing simulation models with real data. The system has two global variables:

n.ifile	Number of columns to be read from file or equivalently the number of output variables
fname.ifile	File name for input data file

The local variables are

OUTPUT:

c1	1st column in the file
c2	2nd column in the file
...	...

PAR:

dt1	Time for first input relative to the start time (default: 0.0)
dt	Distance between inputs (default: 1.0)

TSAMP:

ts	Time for next input

The actual value of n.ifile is fixed during the execution of the SYST command. The value of fname.ifile can be changed between simulations. The simulation terminates if the input file is exhausted.

Several systems with different system identifiers can be active simultaneously. The global variables should then contain actual system identifiers. The following example illustrates the use of IFILE.

Example. Assume that data is stored in the file FDAT.D, which has 10 columns and that we want to use the third column as input to a simulation. Let the following systems be defined.

```
continuous system SYS1
  ...
END

connecting system CONN
time t
u[sys1]=c3[ifile]
END
```

The global variables of the system IFILE are assigned by the command

```
let n.ifile=10
let fname.ifile=fdat
```

and the systems are activated by the command

```
syst sys1 ifile conn
```

The file used as input to IFILE can be created in Simnon using a store command. It can also be generated in the system identification package Idpac.

LOGGER

This is a discrete time system which can be used to sample and save arbitrary Simnon variables on a file.

The system has one global variable:

```
file.npoint
```

which gives the number of data points stored in the current STORE file with name <file>. The global variable is set by LOGGER after each simulation. It can be displayed by the WRITE command. (This value must be known if the generated file

is to be used by Idpac.)

The local variables are:

PAR:

dt1	Time for first sampling relative to the start time (default: 0.0)
dt	Sampling interval. If dt=0.0, no fixed sampling is done, and all points are saved (default: 0.0)

TSAMP:

ts	Time for next sampling

When it is desired to store data the system LOGGER should be included as a system in the SYST command. Since it has neither INPUTs nor OUTPUTs, it need not be connected in the CONNECTING SYSTEM. A connecting system must, nevertheless, be present even if only one system is compiled together with LOGGER. The connecting system can then be empty (3 lines). The following example illustrates how the system LOGGER can be used.

Example. Assume that a system SYS1 is simulated, which has a variable x and that it is desired to store the values of x every 10th second starting at time t = 2.0. Assume that the following systems are defined.

```
continuous system SYS1
  ...
END

connecting system CONN
time t
END
```

The systems are activated by the command

```
syst sys1 logger conn
```

The parameters of the system LOGGER are set by the commands

```
par dt[logger]: 10
par dt1[logger]: 2
```

The command request the variable x of SYS1 to be stored.

```
store x[sys1]
```

The following simulation command executes the simulation with the option that the

variable is stored in FIL1

```
simu 0 1500 / fil1
```

Notice that the number of stored variables are available as the global variable fil1.npoint. The value of this variable is found as follows:

```
write fil1.npoint
149
```

NOISE1

This is a discrete time system, which generates a sequence of independent random vectors. The components of the vector have either a normal or rectangular distribution. The system has the global variables

n.noise1	Number of outputs
nodd.noise1	Initial value for the generator (should be an odd, positive integer)

The local variables are:

OUTPUT:	
e1, e2, ...	The noise vector
PAR:	
stdev1, stedv2, ...	Standard deviations for the outputs (default: 1.0)
dt1	Time for first output relative to start time (default: 0.0)
dt	Distance in time between outputs (default: 1.0)
same	Reset switch. When same > 0.5, the noise generator is reset (default: 1.0)
rect	Type switch. If rect > 0.5, rectangular noise in the interval (0, stdev) is generated instead of white noise (default: 0.0)
TSAMP:	
ts	Time for next output

Note that the switches same and rect influence the entire noise vector. Control of individual components is not possible.

The global variable nodd.noise1 is updated at the end of each simulation. The value of nodd.noise1 can be changed between the simulations. The following

example illustrates how to use the noise generator.

Example. Generate two white noise input signals to the system SYS1. The signals should start at t = 0.5 and change with a sampling period of 3 time units. Assume that the following systems are available.

```
continuous system SYS1
 ...
END

connecting system CONN
time t
u1[sys1]=e1[noise1]
u2[sys1]=e2[noise1]
END
```

The global variables of the system NOISE1 are assigned by the commands

```
let n.noise1=2
let nodd.noise1=25831
```

The systems are activated by the command

```
syst noise1 sys1 conn
```

and the parameters dt and dt1 of NOISE1 are set by

```
par dt:3
let dt1:0.5
```

STIME

This system stores clocktime and cpu-time in the two Simnon variables MSCLOCK and MSCPU. It is useful for timing of simulations and for investigating computational efficiency. No connecting system is needed. It is sufficient to include STIME in the SYST command. The variables MSCLOCK and MSCPU are set to zero at the start of each simulation.

OPTA

This discrete time system is a tool for the problem of minimizing the function J(p) subject to the constraints

$$g_i(p) \leq 0 \qquad i = 1, \ldots, m.$$

The system OPTA performs the minimization recursively. The system has old values of the parameters as states. It accepts values of J and g as inputs and it generates new values of the parameters, which will give a smaller loss function as outputs.

The global variables are:

npar.opta	Number of parameters (max 10)
ncons.opta	Number of constraints (max 10)

The major local variables are:

INPUT:

loss	The value of the function to be optimized
con1,con2	The values of the constraints

OUTPUT:

p1,p2	The new values of the parameters
tbeg	This variable is set to the sampling time

PAR:

tinc	Sampling period

There are also several additional parameters, which will influence the optimization. They are listed below.

XM1 XM2 ... (1 1 ..) Scaling factors, see HH and EPS

LAM1 LAM2 ... (0 0 ..) Lagrange multipliers, initial values

DFN (-0.5) Controls initial step at the first linear minimization; should give an estimate of the likely reduction in function value, Δf; there are two possibilities:
DFN<0 DFN itself is an estimate of Δf
DFN<0 ABS(DFN)·f is taken as an estimate of Δf

TINC (1) Length of sampling interval of OPTA

HH (0.005) Step length for calculation of the gradient by differences; the step length for each component Pi is XMI·HH

EPS (0.01) Stopping criterion for unconstrained minimization - is satisfied

when the change in each component Pi is less than XMi·EPS

PRIN (1) Controls printout on line printer; every ABS(PRIN):th iteration is printed; if PRIN<0 only function values are printed; if PRIN>0 also P and the gradient are printed; if PRIN=0 there is no printout

EVMAX (10000) Maximum number of function evaluations

CEQ (0) Number of equality constraints

C (1) Constant used in the modified function; only used for constrained problems

DELTA (0.01) Stopping criterion for constrained minimization - satisfied when TEST is less than DELTA

RESET (1) The states are reset to their initial values if RESET>0

DARK (1) There is no trace on the display at the sampling points of OPTA if DARK>0

MODE (1) Controls the initialization of the approximation of the second derivative, H
MODE=1 H is set equal to the identity matrix initially
MODE=3 the H-matrix from the previous minimization is used

LPLOT There is no plotting on the display when the optimization routine is calculating derivatives by differentiation if lplot>0.0

For a detailed discussion of these parameters and their use we refer to Glad (1974).

The system OPTA can be used in many different ways. Typical applications are to adjust regulator parameters for optimized performance and to adjust model parameters in connection with model fitting. In such cases it is necessary to perform a simulation in order to obtain the values of lossfunctions and constraints.

A starting value of the parameter vector p is given to initialize the optimization. The system is then simulated for this parameter value. The criterion and the constraints are evaluated. The optimization routine then uses the value of the loss function J and the constraint g to compute a new value of the parameter vector. The process is then repeated with the new parameter. The procedure is illustrated by Fig. C-1. The criterion J and the constraint vector g are connected from the system to the optimizer while the output of the optimizer is the parameter vector p, which is an input to the system. At each sampling point of

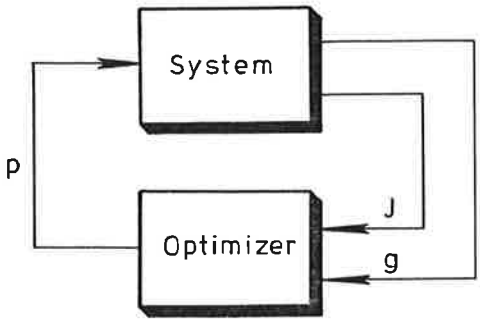


Figure C-1

the optimizer it uses the current values of J and g to compute the next value of p. If the criterion or constraints depend explicitly on the parameter vector p, a vector p_d, containing delayed values of p, is used. The system can consist of a number of subsystems. The connections between these subsystems, as well as connections needed to form J and g from output of the subsystems, are made in the connecting system.

An example illustrates how OPTA is used.

Example. Consider a control system, whose block diagram is shown in Fig. C-2. Let the purpose of the controller be to keep x_2 as small as possible despite the disturbance v, which is an impulse disturbance. This is equivalent to setting $x_1(0)=1$. The criterion is

$$J = \int_0^T x_2^2 \, dt$$

It is also assumed that the total control effort is limited.

$$g = \int_0^T u^2 \, dt - u_{lim} \leq 0 \qquad u_{lim} = 0.5$$

The gains K_p and K_d should be chosen to satisfy these demands. The Simnon programs required to solve the problem are given below.

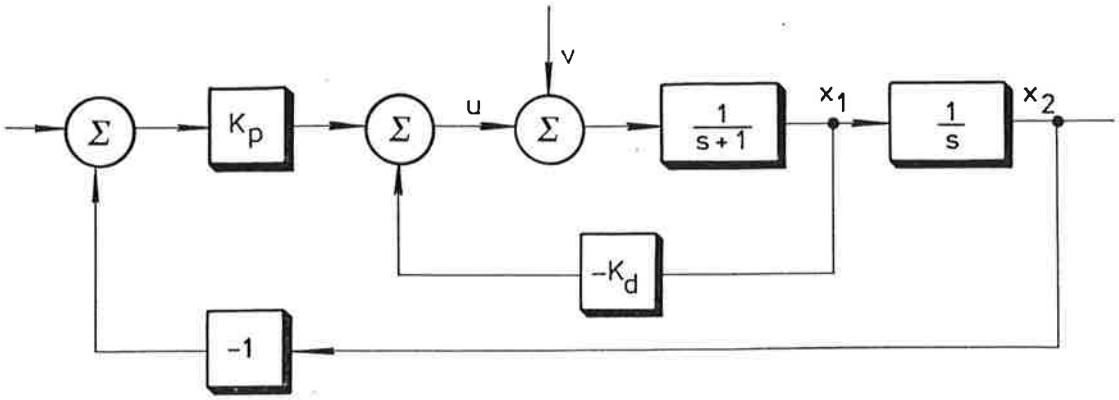


Figure C-2

```
continuous system IMP
state x1 x2  z w
der dx1 dx2 dz dw
input kd kp
output y
output y=x2
y=x2
dynamics
u=-kd*x1-kp*x2
dx1=-x1+u
dx2=x1
dz=u*u
dw=x2*x2
END

connecting system CONN
time tim
wt: 10
loss[opta]=wt*w[imp]
ulim: .5
con1[opta]=z[imp]-ulim
kd[imp]=p1[opta]
kd[imp]=p2[opta]
t = tim - tbeg[opta]
END
```

To do the optimization the global variables of OPTA are first assigned by the commands

```
let npar.opta=2
let ncons.opta=1
```

The systems are then activated by the command

```
syst imp opta conn
```

Initial values of states and parameters are assigned by

```
init x1: 1
init pi1: 2
init pi2: 2
```

The sampling period of the system OPTA is set by the command

```
par tinc: 10
```

The command

```
par prin: 5
```

tells that parameters, function values and gradients are printed. The optimization is then executed by

```
plot y con1 (t)
axes h 0 10 v -.1 .5
simu 0 10000 1
```

APPENDIX D

Macros for Generating the Figures

This appendix gives Simnon macros for generating the figures in the report.

```
macro FIG2
"Generates Fig. 2
syst vdpol
split 1 1
axes h 0 20 v -6 6
plot x y
store x y
init x: 1
simu 0 20 -mark/b1
mark a 2.5 0
mark "Time t
mark a 1 1
mark v "State variables x and y marked 1 and 2
END

continuous system VDPOL
"The van der Pol equation
state x y
der dx dy
dy=x
dx=a*x*(b-y*y)-y
a: 1
b: 1
END

macro FIG3
syst vdpol
init x: 1
store x y
simu 0 20/b1
par b: 2
simu/b2
split 2 1
axes h 0 20 v -6 6
show x /b1
```

```
show x /b2
axes v -3 3
show y /b1
show y /b2
mark a 2.5 0
mark "Time t
mark a 2.5 6.5
mark "Time t
mark a 1 1
mark v "State variable y
mark a 1 7.5
mark v "State variable x
END

macro FIG4
"Phase plane for the van der Pol equation
syst vdpol
init x:1
split 1 1
axes h -4 4 v -3 3
plot y(x)
simu 0 20
mark a 2.5 0
mark "State variable x
mark a 1 1
mark v "State variable y
END

macro FIG5
"Simulation of population dynamics 0 20
syst popdyn
split 3 1
axes h 0 80 v 0 1
par r:0.2
plot x
simu 0 80
mark a 2.5 0
mark "Generation k
mark a 1 1
mark v "Population x
mark a 1 5.5
mark v "Population x
mark a 1 10
mark v "Population x
axes
par r:2.70
simu
axes
par r:2.83
simu
END
```

```
discrete system POPDYN
"Simple model for population dynamics
state x
new nx
time k
tsamp ts
nx=x+r*x*(1-x)
ts=k+1
x: 0.5
r: 2
END

macro FIG6
syst POL
store f
simu -3.6 3.6
split 1 1
ashow f
mark a 17 0
mark "x
mark a 0.5 12
mark "f(x)
END

discrete system POL
time x
tsamp z
f=(x+3)*(x+2)*x*(x-2)*(x-3)
z=x+dx
dx: 0.05
END

macro FIG7
"Generates Fig. 7
syst EXPZ
split 1 1
axes h -1 1 v 0 1.5
plot ImG(ReG)
simu 0 3.14
par alfa: 135
simu 0 4.44
mark a 17 0
mark "Re G
mark a 0.5 12
mark "Im G
END

discrete system EXPZ
time r
tsamp s
fi=alfa*pi/180
x=r*cos(fi)
```

```
y=r*sin(fi)
ReG=exp(x)*cos(y)
ImG=exp(x)*sin(y)
s=r+dr
dr: 0.01
pi: 3.1415926
alfa: 90
END

macro FIG9
"Generates Fig. 9
syst proc pireg con
store yr y[proc] upr
simu 0 40/wup
par ulow: -0.1
par uhigh: 0.1
simu /nowup
split 2 1
ashow y/wup
show yr y/nowup
ashow upr/wup
show upr/nowup
mark a 2.5 0
mark "Time t
mark a 1 1
mark v "Control variable u
mark a 1 7.5
mark v "Output y and yref
END

continuous system PROC
"Integrator with input saturation
input u
output y
state x
der dx
upr=if u<-0.1 then -0.1 else if u<0.1 then u else 0.1
dx=upr
y=x
END

discrete system PIREG
"PI regulator with anti-windup
input yr y
output u
state i
new ni
time t
tsamp ts
e=yr-y
v=k*e+i
u=if v<ulow then ulow else if v<uhigh then v  else uhigh
```

```
ni=i+k*e*h/ti+u-v
ts=t+h
k: 1
ti: 1
h: 0.5
ulow: -1
uhigh: 1
END


connecting system CON
"Connecting system for simulation of process PROC
"with PI regulation by system PIREG
time t
yr[pireg]=1
y[pireg]=y[proc]
u[proc]=u[pireg]
END


macro FIG10
"Macro for generating Fig. 10
let n.func1=4
syst FUNC1 FUNCPLOT
get funcpar
split 1 1
axes h -4 4 v -3 3
plot yp
simu -4 4
mark a 17 0
mark "x
mark a 1 12
mark "f(x)
END


connecting system FUNCPLOT
time x
u[FUNC1]=x
yp=y[FUNC1]
END


[FUNC1]
UI1: -3.
UI2: -1.
UI3: 1.
UI4: 4.
GI1: -1.
GI2: -2.8
GI3: 1.5
GI4: 2.
order: 1.
[FUNCPLOT]
```

APPENDIX E

Intrac and Simnon Commands

This appendix lists all the commands in Intrac and Simnon.

INTRAC COMMANDS

1. Input and output

READ - Read variable from keyboard
WRITE - Write variables
SWITCH - Utility command
STOP - Stop execution and return to OS

2. Assignment

FREE - Releases assigned global variables
LET - Assigns global variables

3. Control of program flow

LABEL L - declaration of label
GOTO L - transfer control
IF..GOTO - Transfer control
FOR..TO - Loop
NEXT V

4. Macro

DEFAULT - Assigns default values
MACRO - Macro definition
FORMAL - Declaration of formal arguments
END - End of macro definition
SUSPEND - Suspend execution of macro
RESUME - Resume execution of macro

SIMNON COMMANDS

1. Input and output

EDIT - Edit system description
DISP - Display parameters

GET - Get parameter and initial values
LIST - List files
PRINT - Print files
SAVE - Save parameter values and initial values in a file
STOP - Stop

2. Graphic output

AREA - Select window on screen
ASHOW - Plot stored variables with automatic scaling
AXES - Draw axes
HCOPY - Hardcopy of the screen
MARK - Write text on axes of graphs
SHOW - Plot stored variables
SPLIT - Split screen into windows
TEXT - Transfer text string to graph

3. Simulation

ALGOR - Select integration algorithm
ERROR - Choose error bound for integration routine
INIT - Change initial values of state variables
PAR - Change parameters
PLOT - Choose variables to be plotted
SIMU - Simulate
STATE - Special command for the integration routine DAS
STORE - Choose variables to be stored
SYST - Activate systems

4. Auxiliary

HELP - Gives guidance
NEWS - Gives news about Simnon
TURN - Set switches

Index