Pascal Systems in SIMNON

Mårtensson, Bengt

1984

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

CODEN:LUTFD2/(TFRT-7278)/1-14/(1984)

PASCAL SYSTEMS IN SIMNON

BENGT MÅRTENSSON

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
DECEMBER 1984

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 118
S 221 00 Lund Sweden

Document name
Report

Date of issue
Decmber 1984

Document number
CODEN:LUTFD2/(TFRT-7278)/1-14/(1984)

Author(s)
Bengt Mårtensson

Supervisor

Sponsoring organization

Title and subtitle
Pascal Systems in Simnon

Abstract

Abstract:
It is demonstrated how to write Pascal systems in the VAX-VMS version of Simnon. An example in the form of an discrete time "universal stabilizing" regulator is included. Some hopefully useful tips and comments are given along the way. A proof of the stability of the regulator is contained in an appendix.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
14

Recipient's notes

Security classification

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

Pascal Systems in Simnon

Abstract:
It is demonstrated how to write Pascal systems in the VAX-VMS version of Simnon. An example in the form of an discrete time "universal stabilizing" regulator is included. Some hopefully useful tips and comments are given along the way. A proof of the stability of the regulator is contained in an appendix.

1. Introduction

It is well known that it is possible to write Fortran subsystems in Simnon. In the VAX-VMS version it is also possible to write systems in Pascal, utilizing some of the nonstandard features of Vax-Pascal. This report is intended to document this.

In what follows it is assumed that the reader is familiar with Simnon e.g. as described in [1]. Knowledge about Fortran systems in Simnon, especially chapter 7 in [2] is helpful, but believed not necessary.

General "how to do it" - info is collected in the next section. Section 3 contains as an example the system "Helmuth", which is a kind of discrete time "universal stabilizing" regulator. The fourth section contains some general comments.

The program list of "Helmuth" is presented in Appendix 1. Some other necessary files are included in the following appendices. Appendix 4 presents a neat Simnon macro, intended to show a technique of documenting simulation results depending on parameters. Finally, slightly off the track, Appendix 5 contains a full description and a proof of stability and convergence of the algorithm.

It should be stressed out that these possibilities do not necessarily generalize to other versions of Simnon. Needless to say, the command file in appendix 3 runs only on the computer at the Department of Automatic Control in Lund.

My thanks goes to Leif Andersson and Tomas Schönthal who have been very helpful when I found out many of the things which are documented here.

2. How To Do It

The reader is advised to compare with the appendices when reading this section. The description will be given in the same order as in the listings, which probably not is the most logical order.

The Pascal system is compiled as a "module", and henceforth no "begin" of a main block is permitted. It is ended by an odd "end.".

Types, variables and external procedures for Simnon are declared as in the listing in Appendix 1. The variables the Pascal system is using are declared next. The dynamic memory allocation in the standard Pascal is inhibited by declaring these variables outside all procedures. Otherwise, all variables will be lost

Pascal Systems in Simnon

Abstract:
It is demonstrated how to write Pascal systems in the VAX-VMS version of Simnon. An example in the form of an discrete time "universal stabilizing" regulator is included. Some hopefully useful tips and comments are given along the way. A proof of the stability of the regulator is contained in an appendix.

1. Introduction

It is well known that it is possible to write Fortran subsystems in Simnon. In the VAX-VMS version it is also possible to write systems in Pascal, utilizing some of the nonstandard features of Vax-Pascal. This report is intended to document this.

In what follows it is assumed that the reader is familiar with Simnon e.g. as described in [1]. Knowledge about Fortran systems in Simnon, especially chapter 7 in [2] is helpful, but believed not necessary.

General "how to do it" - info is collected in the next section. Section 3 contains as an example the system "Helmuth", which is a kind of discrete time "universal stabilizing" regulator. The fourth section contains some general comments.

The program list of "Helmuth" is presented in Appendix 1. Some other necessary files are included in the following appendices. Appendix 4 presents a neat Simnon macro, intended to show a technique of documenting simulation results depending on parameters. Finally, slightly off the track, Appendix 5 contains a full description and a proof of stability and convergence of the algorithm.

It should be stressed out that these possibilities do not necessarily generalize to other versions of Simnon. Needless to say, the command file in appendix 3 runs only on the computer at the Department of Automatic Control in Lund.

My thanks goes to Leif Andersson and Tomas Schönthal who have been very helpful when I found out many of the things which are documented here.

2. How To Do It

The reader is advised to compare with the appendices when reading this section. The description will be given in the same order as in the listings, which probably not is the most logical order.

The Pascal system is compiled as a "module", and henceforth no "begin" of a main block is permitted. It is ended by an odd "end.".

Types, variables and external procedures for Simnon are declared as in the listing in Appendix 1. The variables the Pascal system is using are declared next. The dynamic memory allocation in the standard Pascal is inhibited by declaring these variables outside all procedures. Otherwise, all variables will be lost

between successive calls.

The main procedure is of course declared global, and consists of a case statment. The structure follows from the example. Part 1 is executed when Simnon is started, parts 2-3 when the "syst" command is given. Part 2 ties the Pascal system's variable with Simnon "variables", with the name contained in the "string" (array [1..8] of char). The example is hoped to be sufficient explanation, otherwise chapter 7 of [2] should be consulted. Observe that when writing external systems in Pascal or Fortran, unlike the case when you are writing systems in the Simnon language, it is not possible to use the same identifier on a state and its initial value. This is slightly annoying.

Part 3 assigns default values to the "par"'s. Part 4 is executed once at the start of each simulation, and corresponds to the "initial" section of the Simnon language. Parts 5 and 6 are executed once at every simulation step, and corresponds to the "output" and "dynamics" sections. Part 7 is called once at every instant of time Simnon stores something in the store file. Time-consuming calculation, not needed for the dynamics- or output-sections, can be put here. Finally, section 8 is executed when the simulation is stopped. e.g. closing of files can be put here.

A Fortran source file "SYSTS.FOR" is required. See Appendix 2 for the concrete example. The variable NSYSTS should be put equal to the number of included systems. Adjust the computed GOTO statement if necessary. A call to the Pascal system is done as in the "8 CALL HELMUTH" line.

The Pascal system and SYSTS are then linked together with Simnon's object libraries into your own version of Simnon. The command file I have been using is included in Appendix 3.

To run your own Simnon version use "invoke simnon", not "run simnon". Otherwise, you will not e.g. get access to the help facility.

3. An Example: Helmuth

The system Helmuth contains a first order discrete time linear system, and a somewhat weird "adaptive" or "universal" controller that will stabilize it. It is different from other adaptive algorithms in the sense that it is not based on identification ideas, and it does not use probing signals.

The algorithm is based on a stepwise search through a dense sequence of all possible controllers, which in this case is in simple 1-1 correspondence with the real numbers R. The dense sequence on R is generated by an random number generator, which yields an uniform distribution on the interval [0,1], composed with a bijection from (0,1) to $(-\infty,\infty)$.

As can be seen from the simulation in Appendix 4, the system behaves quite wildly, but the algorithm does the job.

A full description, together with a proof of stability and convergence, is included in Appendix 5.

An algorithm of this type has, as far as the author is aware, never occured in the literature. For more results of this type, see [3].

4. Some Comments

The procedure Testifcrash is used to test weather |y| > crash, where y is the output of the system to be controlled, and crash is a parameter in the Simnon sense. If this is true, the simulation is stopped gracefully by setting the global boolean variable lstop to true. Crash should normally be set to a "very large" number, slightly less than the largest real number the computer can hold.

This is to prevent a program crash due to numerical overflow. Another use of this is to interrupt a simulation just when or before things "explode".

The Simnon macro "run", given in Appendix 4, shows how to automatically put parameter values on your hardcopy's. This is very useful for documentation purposes.

It is believed that it is evident how to use the macro, and what it does, perhaps with the following exception: If you forget the "free"-ing the following bug occurs: The first time you assign a value to the global intrac variables (i.e. a. and b.) they will be assigned the same type (i.e. integer or real) as their values. If you first make them integer, and then give them a non-integer value, the value will be truncated.

It is not possible to write a Pascal (or Fortran) system and "control" it with a "controller" with "direct term". In this case, Simnon will complain about "algebraic loop detected", regardless if it is true or not. Strictly speaking, this is a bug in Simnon, but it is a consequence of the tree-building in the equation sorting parts of Simnon.

It is tempting to call the executable code something else than "simnon". This will however confuse the "invoke" command so you lose the help utility.

Since the executable code is fairly large (> 500 blocks in VAX-VMS) and it is fairly easy to generate, it is probably a good idea to let it reside on a non-backed-up area.

5. References

[1] Åström, K J, A Simnon Tutorial, CODEN: LUTFD2/(TFRT-3168)/1-52/(1982), Department of Automatic Control, Lund Institute of Technology

[2] Elmqvist, H, Simnon Users Manual, Report TFRT-3091, Department of Automatic Control, Lund Institute of Technology

[3] Mårtensson, B, PhD - thesis, to appear (sometime!)

Appendix 1

Program List

```
Module Helmuth(output);

{Simple discrete time adaptive controller by searching a dense
sequence of controllers

Pascal system to be linked into Simnon


Author  Bengt Martensson, 841210 }



{#########Types for simnon ############################}

type
   simnonid=packed array [1..8] of char;
   destinrec=record
               idum,ipart: integer
             end;

const
   maxindex = 15;

type
   index = 1..maxindex;
   vector = array [index] of real;


{########## Variables for Simnon ####################}

var
     destin : [common] destinrec;
     t : [common(time)] real;
     lstop : [common(user)] boolean;



{############## Procedures for Simnon ####################}

[external(ident2)]Procedure Ident(stype,sysid: simnonid); external;
[external(tsamp2)]Procedure Tsamp(var v:real; vid: simnonid); external;
[external(input2)]Procedure Input(var v:real; vid: simnonid); external;
[external(outpu2)]Procedure Outpu2(var v:real; vid: simnonid); external;
[external(state2)]Procedure State(var v:real; vid: simnonid); external;
[external(init2)]Procedure Init(var v:real; vid: simnonid); external;
[external(Der2)]Procedure Der(var v:real; vid: simnonid); external;
[external(new2)]Procedure New(var v:real; vid: simnonid); external;
[external(par2)]Procedure Par(var v:real; vid: simnonid); external;
[external(var2)] Procedure auxvar(var v:real; vid: simnonid);external;
[external(inpuv2)]Procedure Inputv(var v:vector; n: integer; vid: simnonid);
                                  external;
```

```
[external(outpv2)]Procedure Outputv(var v:vector; n: integer; vid: simnonid);
                                    external;
[external(statv2)]Procedure Statev(var v:vector; n: integer; vid: simnonid);
                                    external;
[external(initv2)]Procedure Initv(var v:vector; n: integer; vid: simnonid);
                                   external;
[external(Derv2)]Procedure Derv(var v:vector; n: integer; vid: simnonid);
                                   external;
[external(newv2)]Procedure Newv(var v:vector; n: integer; vid: simnonid);
                                   external;
[external(parv2)]Procedure Parv(var v:vector; n: integer; vid: simnonid);
                                   external;
[external(varv2)] Procedure Auxvarv(var v:vector; n: integer;
                    vid: simnonid); external;


{##################################################}


var
   a,b,y,u,k,ny,l1norm,nl1norm,yinit,n,zero        : real;
   initseed,l1init,crash,oldl1norm,y0,h,ts         : real;
   seed                                            : integer;
   writtenonce                                      : boolean;


procedure Testifcrash;

{Stops the simulation gracefully if abs(y) > crash}

begin
   if (abs(y) > crash) and not writtenonce then
   begin
      writeln('Abs(y) > crash = ',crash,' Simulation stopped!');
      writtenonce := true;
      lstop := true;
   end;
end;


function tan(x : real) : real;

begin
   tan := sin(x)/cos(x);
end;


function mth$random(var seed : integer):real;external;


const
```

```
   pi = 3.1415926;


function Newk : real;
{generates a dense sequence on R}

begin
   Newk := tan(pi*mth$random(seed) - pi/2);
end;


Procedure Updategain;

begin
   if l1norm - oldl1norm > n*y0 then     {try new regulator}
   begin
      k := Newk;
      oldl1norm := l1norm;
      n := n + 1;
      y0 := abs(y);
   end;
end;




[Global] procedure Helmuth;

begin
  case destin.ipart of
  1: Ident('DISC     ','HELMUTH ');                          {identification}
  2: begin                                          {declaration of variables}
        State(y,'y         ');
        New(ny,'ny         ');
        Init(yinit,'yinit    ');
        State(l1norm,'l1norm   ');
        New(nl1norm,'nl1norm ');
        Init(l1init,'l1init   ');
        Par(a,'a          ');
        Par(b,'b          ');
        Par(initseed,'initseed');
        Par(crash,'crash    ');
        Par(h,'h          ');
        Tsamp(ts,'ts         ');
        Auxvar(k,'k          ');
        Auxvar(u,'u          ');
        Auxvar(n,'n          ');
        Auxvar(zero,'zero     ');
     end;
  3: begin                                          {Assignments of 'Par's}
        crash := 1E30;
        a := 2;
```

```
        b := 1;
        h := 1;
        initseed := 1;
        yinit := 1;
    end;
  4: begin                                          {Initial section}
        zero := 0;
        writtenonce := false;
        seed := round(initseed);
        n := 0;
        y0 := 0;
        oldl1norm := 0;
    end;
  5: begin                                           {output section}
        Testifcrash;
        Updategain;
        u := -k*y;
    end;
  6: begin                                          {dynamics section}
        ny := a*y + b*u;
        nl1norm := l1norm + abs(y);
        ts := t + h;
    end;
  7: begin                                {computation on accepted values}
    end;
  8: begin                                          {final computations}
    end;
  end;
end;

end.
```

Appendix 2

SYSTS.FOR

```
      SUBROUTINE SYSTS
C
      DIMENSION SNAM1(2),SIFIL(2),FUNC1(2)
      COMMON/DESTIN/ISYST,IDUM
      COMMON/NSYSTS/NSYST
      COMMON/NALLOC/NS
      COMMON /SYSAV/ S(10000)
      COMMON/SAVEAR/IS(42)
      COMMON/DEVICE/LKB,LTP,LLP,LDIS,LTO,LPLOT,LXXX,LDK1,LDK2,LDK3,LDK4
C
      DATA SNAM1,FUNC1/4HNOIS,4HE1   ,4HFUNC,4H1    /
      DATA SIFIL/4HIFIL,4HE    /
C
C     SIZE OF INTEGER SAVE AREAS (IS)
C
C     OPTA   14
C     SNOISE  6
C     SDELAY  9
C     SIFILE  5
C     SFUNC   4
C     LOGGER  2
C     STIME   1
C
C      ** VAX ERROR HANDLER FOR FORTRAN SYSTEM IS DECLARED HERE:
      EXTERNAL FORSYSHDL
      CALL LIB$ESTABLISH(FORSYSHDL)
C
      NSYST=8
      NS=10000
C
      GO TO (1,2,3,4,5,6,7,8),ISYST
C
 1    CALL SOPTA(IS,S)
      RETURN
C
 2    CALL SNOISE(SNAM1,IS(15),S)
      RETURN
C
 3    CALL STIME(IS(21),S)
      RETURN
C
 4    CALL SDELAY(IS(22),S)
      RETURN
C
 5    CALL SIFILE(SIFIL,LDK2,IS(31),S)
      RETURN
C
 6    CALL LOGGER(IS(36),S)
      RETURN
C
 7    CALL SFUNC(FUNC1,IS(38),S)
```

```
      RETURN
C
 8    CALL HELMUTH
      RETURN

      END
```

Appendix 3

Link Command File

```
$ on error then goto delobj
$ delete simnon.exe;*
$ pascal helmuth
$ link/nomap/executable=simnon -
  paclib:simlib/include=(simnon$main,extsub),-
  build:unimpl,-
  use:[]helmuth, -
  use:[]systs, -
  paclib:simlib/library,-
  build:every/options
$ delobj:
$ delete helmuth.obj;*
```

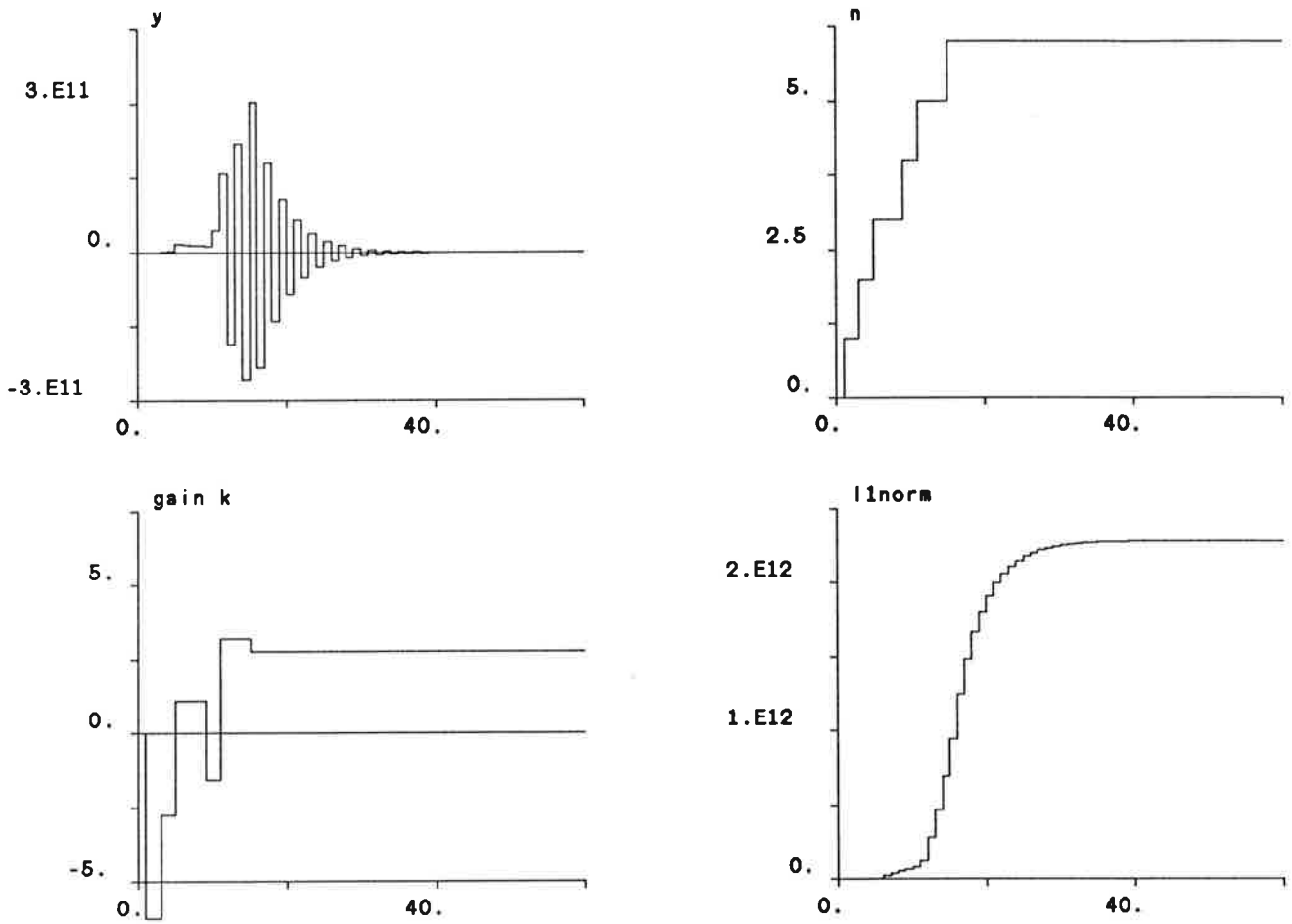
Appendix 4

Neat Macro

```
macro run ; simtime ; ain    bin
default simtime = 60
default ain = 2
default bin = 1
free a.
free b.
let a. = ain
let b. = bin
par a : a.
par b : b.
simu 0 simtime  /store 0
switch graph on

split 2 2
ashow y
show zero
text 'y'
axes v -5 5 h 0 simtime
show k
show zero
text 'gain k'
ashow n
text 'n'
ashow l1norm
text 'l1norm'

switch mark off
mark a 0 14
mark "Helmuth
mark a 0 13.5
mark : a  =  a. ; b  =  b.

end
```

Simulation Results

84.12.12 - 14:40:10 nr: 1
hcopy
Helmuth
a=2;b=1



This figure was generated by the command "Run"

Appendix 5

Full Description, Proof of Convergence of the Helmuth Regulator

Let the time $t \in N = \{0,1,...\}$. Denote the ℓ^1 norm of y(.) truncated at time t by $\Sigma(t)$, e.g.

$$\Sigma(t) = \sum_{0}^{t} |y(j)|$$

Let $(k_\nu)_1^\infty$ be a dense sequence on R. Suppose that the system to be controlled is

$$y(t+1) = ay(t) + bu(t) \qquad (*)$$

where a and b are unknown (including signs) real numbers. To make things interesting, we assume that (*) is unstable, i.e. $|a| > 1$.

The algorithm can be described in some sort of "pseudo Pascal language":

```
for n := 1 to ∞ do
begin
   y0 := |y|;
   t0 := t;
   k := kn;
   repeat
      (regulate with the control law u = -ky);
   until Σ(t) - Σ(t0) > ny0;
end
```

Proof of stability and convergence

There is an open set of k's such that the system (*) is stable when controlled by the control law u = -ky. By shrinking this open set somewhat to O, there is a real number $N > 0$ such that $\Sigma(\infty)$ exists, and is $< N|y(0)|$ for all $k \in O$ (just sum the infinite geometric series). But this means that for n in the algorithm sufficiently large and $k \in O$, the condition in the repeat statement will never be satisfied. Since (k_ν) is dense in R and therefore $k_\nu \in O$ for infinitely many ν, this proves convergence and stability.