Optimization-Based Design of Nonlinear Systems Using CONSOLE and SIMNON

by

M. Fan
Optimization-Based Design of Nonlinear Systems Using
CONSOLE and SIMNON *

Michael Ko-Hui Fan
Systems Research Center
University of Maryland, College Park, MD 20742

November 22, 1987

Abstract

The most challenging task when designing a nonlinear system is that of coming up with an appropriate system 'structure'. This task calls extensively upon the engineer's ingenuity, creativity, intuition and experience. After a structure has been (maybe temporarily) selected, it remains to determine the 'best' values of a number of 'design parameters'. The input from the engineer is still essential here, as tradeoffs are likely to be encountered. However, except in the simplest cases, achieving anything close to optimal would be impossible without the support of numerical optimization and simulation tools. In this report, we present an interface between CONSOLE, an interactive, optimization-based design package, and SIMNON, an interactive program for simulation of systems governed by ordinary differential equations and difference equations. We show, by means of an example, how to set up a design problem for nonlinear systems by using CONSOLE and SIMNON.

1 Introduction

The most challenging task when designing a nonlinear system is that of coming up with an appropriate system 'structure'. This task calls extensively upon the engineer's ingenuity, creativity, intuition and experience. After a structure has been (maybe temporarily) selected, it remains to determine the 'best' values of a number of 'design parameters'. The input from the engineer is still essential here, as tradeoffs are likely to be encountered. However, except in the simplest cases, achieving anything close to optimal would be impossible without the support of numerical optimization and simulation tools.

*Many thanks are due to Dr. A.L. Tits and Dr. W.T. Nye for various useful suggestions in the preparation of this report and to Y.C. Pati, L.S. Wang and Dr. W.S. Levine for their helps in understanding SIMNON. The author is also thankful to Prof. K.J. Åström of the Lund Institute of Technology for making SIMNON available to the Systems Research Center. This research was supported by the National Science Foundation under grants No. DMC-84-20740 and CDR-85-00108.
**Figure 1: Block diagram of the design example**

CONSOLE is a software tandem for interactive, optimization-based design of general engineering systems. It was recently developed at the University of Maryland, College Park [1]. SIMNON is an interactive program for simulation of systems governed by ordinary differential equations and difference equations. It was developed at the Lund Institute of Technology in 1975 [2].

In this report, we present an interface between CONSOLE (Version 1.0) and SIMNON (Version 4) under Unix systems. The interface is made possible mainly due to the special structure of CONSOLE. CONSOLE was designed in such a way that any simulator can be used without complicated interface procedures. By way of a design example, we show how an optimization-based design of a nonlinear system can be set up and how the procedure to determine the best values of a number of design parameters can proceed. This report does not intend to cover any details peculiar to CONSOLE or SIMNON used as individual packages; instead, we discuss various features of the union of CONSOLE and SIMNON. The reader is referred to [1] and [2] for a full explanation of each package.

## 2 A Design Example: Problem Setup

The design example is a regulation problem. The overall system configuration is depicted in Figure 1. Given the second order linear time-invariant system

\[ G(s) = \frac{s + 1}{s^2} \]

with the actuator in saturation. The design task is to find a suitable PI controller

\[ D(s) = K_p + \frac{K_i}{s} \]

with unity feedback such that the output of the overall system tracks a unit step signal asymptotically. Furthermore, the rise time, the settling time and the overshoot of the output are required to be small, i.e., the system is to be regulated such that its output with respect to a unit step input lies in the given shaded area in Figure 2. Since the controller has already been chosen (a PI controller), it simply remains to choose the values of \( K_p \) and \( K_i \) in order to meet the specifications given above.
Two files are necessary for the design problem setup. The first file is the Problem Description File which is the input file for CONSOLE. It defines the design specifications (i.e., declarations of design parameters, declarations of objectives, etc.). The second file is the System Description File. It is the input file for SIMNON and describes the nonlinear system configuration (i.e., describes the differential equations and differences equations the systems follow and assigns the values of various parameters, etc.) Suppose we have named the Problem Description File design and the System Description File servo.t. The corresponding Problem Description File design follows:

```
design_parameter Kp init=1 variation=5
design_parameter Ki

initialization {
    /* declares initialization routine */
    simmon("syst servo"); /* reads system description */
    simmon("store y"); /* output y needs to be stored */
    simu (0.0, 20.5, .1); /* declares integration range and step size */
}

functional_objective "upper bound"
for t from 0 to 20 by .1
minimize {
    double output();
    return output("y",t);
}

if (t <= 4) return 1.05;
else return 1.01;

if (t <= 4) return 1.1;
else return 1.02;
}

functional_objective "lower bound"
for t from 2 to 20 by .1
maximize {
    double output();
    return output("y",t);
}

if (t < 4) return 0.9;
else return 0.99;

bad_curve ={
```
Figure 3: Overall structure of CONSOLE with SIMNON

```plaintext
if (t < 4) return 0.85;
else return 0.95;
}
```

The following is the corresponding System Description File.

```plaintext
CONTINUOUS SYSTEM servo
STATE x1 x2 x3
DER dx1 dx2 dx3
init x1:0
init x2:0
init x3:0
dx1 = x2
dx2 = if (e > 0.4) then 0.4 else if (e < -0.4) then -0.4 else e
dx3 = r - y
e = (r - y)*Kp + x3*Ki
y = x1+x2
r:1
Kp:0 "assign arbitrary values for design parameters"
Ki:0
END
```

3 How to invoke CONSOLE and SIMNON

CONSOLE is composed of two main programs: CONVERT and SOLVE. Figure 3 illustrates the over-
all structure of CONSOLE with SIMNON. The Problem Description File is the input to CONVERT. CONVERT checks the description for all possible syntax errors and some logic errors and then generates two files. One file is a data file which contains, among other things, the values and names of design parameters, the names of specifications, etc. The other file is an object file. It contains a compiled version of the various specifications (objectives, functional objectives, constraints and functional constraints). Both files are input to SOLVE, together with the object files of SIMNON and the interface. SOLVE will iterate together with the user to obtain an acceptable solution. At the same time the user can also communicate with SIMNON interactively (see below). To invoke CONVERT, the user may type

```convert design```

Some messages appear on the terminal indicating various operations CONVERT is performing. If any error occurs, the user should correct it and run CONVERT again. The next step is to invoke SOLVE and SIMNON by typing

```solve-simon design```

`solve-simon` is a file containing command scripts. In Section 5, we will discuss how to construct such file. Again, some messages appear on the terminal indicating various operations SOLVE is performing. If no error occurs, SOLVE prompts

```<o>```

showing that SOLVE and SIMNON are ready to receive commands from the user to perform optimization.

Figure 4 shows the time response of the output $y$ for initial values of $K_p$ and $K_i$. After 4 optimization iterations in SOLVE, a set of design parameters is obtained and the corresponding system output satisfies the desired specifications. The time response of the output $y$ corresponds to the new values of $K_p$ and $K_i$ is given in Figure 5.

## 4 Features of the Union of CONSOLE and SIMNON

As mentioned in the introduction, this report does not intend to cover any details peculiar to CONSOLE or SIMNON used as individual packages. We assume that the user has some basic understanding in using CONSOLE and SIMNON separately. Below, we list the important things to be remembered when using CONSOLE and SIMNON together.

1. Any design parameter declared in the Problem Description File should be also a parameter in the System Description File, i.e., its value (in the System Description File) can be changed by the command `par` in SIMNON. The initial values for design parameters in the System Description File are not important, but they have to be given. In the example above, they are set zero.
Figure 4: Output with $K_p = 1$, $K_i = 0$

Figure 5: Output with $K_p = 24.6$, $K_i = -0.05$
2. While CONSOLE is case sensitive, SIMNON is not. Therefore, the user should not declare two design parameters such as ABC and abc since they are treated differently in SOLVE but the same in SIMNON.

3. The user should first run SIMNON alone to obtain a bug-free System Description File.

4. Three C routines are available to be accessed in the Problem Description File. They are (and their arguments)

   simnon (string)
   char *string;

   simu (from, to, stepsize)
   double from, to, stepsize;

   output (oname, time)
   char *oname;
   double time;

   The routine simnon can be used to send commands from CONSOLE to SIMNON (commands to SIMNON can also be sent from a terminal, see below). For example, an execution of the statement

   simnon ("syst servo")

   in a Problem Description File causes SIMNON to execute the command

   syst servo

   to read a system configuration from the file servo.t. It should be kept in mind that no error could occur whenever the routine simnon is executed, since otherwise this would cause the program to abort. The routine simu is used to declare the integration interval and step size in SIMNON. Whenever CONSOLE requests a simulation to SIMNON, it sends the command

   simu from to stepsize

   to SIMNON, where from, to and stepsize represent the values given by the command simu in the Problem Description File. Finally, the routine output is used to get the simulation result from SIMNON. It returns the double precision value of output oname at time t. The output oname must have been declared to be stored in SIMNON (see below).

5. In the Problem Description File, an initialization routine must be declared by the command initialization. It should also contain at least three function calls as we see in the file design earlier. When the initialization routine is executed, it first reads the system configuration from the file servo.t (simnon("syst servo"));, then requests to store the value of the output y
during simulation (simnon("store y");) and finally declares the integration interval and step size (simu(0.0, 20.5, 0.1);). The order for the first two statements is important. Note that the integration interval declared above is from 0.0 to 20.5 while in the Problem Description File design, the two functional objectives are only declared for t from 0 to 20 (see above). This is because sometimes SIMNON does not finish the integration at the exact value of 20, but rather at a close but smaller value. To ensure the output over the entire range of t (time) in functional specifications can be obtained, we usually request a slightly larger integration range.

6. The step sizes declared for integration and declared for functional specifications need not coincide. However, it is suggested that the step size for integration should be no larger than the ones for functional specifications. The value of output for a given t is computed by linearly interpolating the simulation results from SIMNON.

7. When a simulation is performed in SIMNON, the following appears on the screen

```
SIMNON
```

A subsequent right bracket ‘]’ indicates the completion of the simulation.

8. When a simulation is being performed, it cannot be stopped by an interrupt signal from the user (as it can if SIMNON is used alone) since the interrupt mechanism is superseded by the one in CONSOLE, where the program can only stop when the optimization algorithm has finished a complete iteration (see [1]).

9. SIMNON can be called interactively when the command sim in SOLVE is issued. It brings the user from CONSOLE into SIMNON. In this state, almost all the usual SIMNON commands can be issued. However, it is strongly suggested that only those commands which display information be used. Commands such as syst, simu, store and par should be avoided.

10. In SIMNON, the description of a continuous or discrete subsystem can be made as a FORTRAN subroutine having a special structure. The link between SIMNON and such subroutines is achieved through a FORTRAN subroutine called systs [2]. When SOLVE is invoked, the corresponding object files should appear as arguments. For example,

```
solve-simmon design systs.o subsys1.o subsys2.o
```

where subsys1.o and subsys2.o contain the object code of two subsystems. The order of the three object files in the above line is not important.
11. If the integration step size is too large during simulations in SIMNON, the interface will detect this and may request a new simulation with a smaller step size until the simulation is completed. For the subsequent simulations, the original value of the step size will still be used.

5 Implementation of the Interface

As mentioned above, the file solve-simnon contains command scripts. Its content is as follows:

```bash
#!/bin/csh -f
# Shell script to invoke SOLVE and SIMNON

if ( $#argv < 1 ) then       ## check number of arguments (must >= 1)
    echo 'usage: solve-simnon file1 file2 ...'
    exit(1)
endif

setenv textliblev2 /usr/simnon/machelp/simnon/  ## for help in SIMNON
setenv textliblev3 /usr/simnon/machelp/progpac/  ## for terminal setting

solve $argv simnon1 simnon2.a

exit(0)
```

where we assume that the on-line facility of SIMNON resides in the directory

```
/usr/simnon/machelp/
```

The files simnon1 and simnon2.a contain the interface between CONSOLE and SIMNON plus SIMNON itself. They can be constructed by the procedure below.

1. Extract the file systs.o from the library archive file simlib.a (part of SIMNON):
   ```
ar x simlib.a systs.o
   ```

2. Construct the file simnon2.a:
   ```
ar rv simnon2.a systs.o
ranlib simnon2.a
   ```

3. Delete the file systs.o from the file simlib.a:
   ```
ar d simlib.a systs.o
ranlib simlib.a
   ```

4. Construct the file simnon1:
   ```
ld -r -S -x -o simnon1 -u _logger_ -u _sdelay_ -u _sfunc_ -u _sifile_ \\
   -u _snoise_ -u _sopta_ -u _stime_ \\
consim.o simnon.o extsub.o unimpl.o simlib.a \\
edit.a intr.a plo.a files.a string.a
   ```

where the files extsub.o unimpl.o edit.a intr.a plo.a files.a string.a are also parts of SIMNON, the file consim.o is the interface routine (between CONSOLE and SIMNON) and simnon.o is a revised SIMNON main program. Lists of the source files consim.c and simnon.f are given in Appendices A and B respectively.
#include <stdio.h>
#include <math.h>
#include <ctype.h>

extern struct _despar {
    char *name;
    double variation;
    int min_soft_or_hard;
    int max_soft_or_hard;
    double min_good;
    double min_bad;
    double max_good;
    double max_bad;
};

extern int numdes; 
extern struct _despar **despars;
extern double pdelta;

#define NO 0
#define YES 1
#define alloc(n,p) (p *) malloc(n, sizeof(p))

static int needsimu=YES, numout, numpt, pline, interac=NO, trace=NO;
static char **outnam, buf[6][80];
static double *simout[2000], *zsv, simu_from=0, simu_to=1, simu_stepsize=.01;

#define SIM_GRAD 101
#define SIM_INIT 102
#define SIM_INTA 103
#define SIM_INTR 104
#define SIM_ITER 105
#define SIM_NOTR 106
#define SIM_PUPD 107
#define SIM.Quit 108
#define SIM_RSTT 109
#define SIM.TRAC 110

simlat(type,z)
int type;
double *z;
{
    if (type == SIM_INIT) { sim_init(z); return; }
    if (type == SIM_INTA) { sim_inta(z); return; }
    if (type == SIM_NOTR) { sim_notr(z); return; }
    if (type == SIM_PUPD) { sim_pupd(z); return; }
    if (type == SIM.Quit) { sim_quit(z); return; }
    if (type == SIM_RSTT) { sim_rset(z); return; }
    if (type == SIM.TRAC) { sim_trac(z); return; }
}

sim_init()
{
    int i;

    pdelta = 1.0e-3;
    zsv = alloc (numdes, double);
    for (i = 0 ; i < numdes ; i++)
        zsv[i] = -1.0e20;

    sminon("=");
    _solve_setint(); /* must reset again since SIMNOW sets it too */
return;
}

sim_quit()
{
    int i;
    printf("cleaning SIMNON ...\n");
    i = 1;
    logpri_(s{i});
    unlink(" "); /* remove temporary files created by SIMNON */
    unlink("store.d");
    unlink("store.dt");
    return;
}

sim_inta()
{
    interact = YES;
    printf("enter SIMNON, type \"stop\" to leave\n");
    smmain();
    printf("back to SOLVE\n");
    interact = NO;
    return;
}

sim_pupd(x)
double z[];
{
    int i;
    char str[80];

    for (i = 0 ; i < numdes ; i++) {
        if (fuzcmp(x[i],zsv[i]) == 0) continue;
        zsv[i] = x[i];
        sprintf(str, "par %s: %.9e", despars[i]->name, z[i]);
        simmon(str);
        needsimu = YES;        /* need simulation */
    }
    return;
}

sim_rset()
{
    int i;

    for (i = 0 ; i < numdes ; i++)
        zsv[i] = -1.0e20;
}

sim_trac() { trace = YES; }
sim_notr() { trace = NO; }

typred(outbuf, nchar, inbuf, eof)
char *outbuf, *nchar, *inbuf;
int eof;
{
    int i;

    if (!interac) goto L1;

    for (i = 0 ; i < *nchar ; i++)
        putchar(outbuf[i]);

    i = 0;

    L1:
while ((inbuf[i++] = getchar()) != '\n')
  if (inbuf[i-1] == EOF) {
    printf ("stop\n");
    rewind(stdin);
    strcpy(inbuf, "stop\n");
    *nchar = strlen(inbuf);
    return;
  }
  *nchar = i;
  return;
}

L1:

if (outbuf[i] != '>') { /* not simon prompt, i.e., */
  for (i = 0; i < *nchar; i++) /* something wrong */
    putchar(outbuf[i]);
}

if (buf[pline][0] == '#') error ("SIMON ERROR");
strcpy(inbuf, buf[pline++]);
*nchar = strlen(inbuf);
if (trace && strncmp(inbuf,"stop",4) != 0) printf ("%s", inbuf);
return;
}

simu(from, to, steps) double from, to, steps;
{
  simu_from = from;
  simu_to = to;
  simu_stepsize = stepsize;
  return;
}

double output(onam, t) char *onam;

double t;
{
  int i, k, ptri, ptrt, ptrb, order;
  double r, factor;
  char str[80];
  double *p1, *p2;

  factor = 1;

  if (fzcmp(t, simu_from) < 0 || fzcmp(t, simu_to) > 0)
    error ("requested time \%e is out of integration range", t);

  if (needsimul) {
    sprintf(str, "simu %f %f %f", simu_from, simu_to, simu_stepsize);
    if (!trace) printf ("[SIMON]"); fflush(stdout);
    simon(str);
    if (!trace) printf (""); fflush(stdout);
    needsimul = NO;
  }

  for (k = 0; k < numout; k++)
    if (!strncmp(onam, outnam[k])) break;

  if (k == numout) /* output name not found */
    error ("\"%s\" not stored in simon", onam);

  L1:
if (fuzcmp(t, simout[0][0]) < 0 || fuzcmp(t, simout[numpnt-1][0]) > 1) {
    factor = 10;
    printf("simulation failed, trying smaller stepsize (x\n",
        simu_stepsize/factor);
    sprintf(str, "simu %f %f", simu_from, simu_to, simu_stepsize/factor);
    if (itrace) printf("[SIMMON]"); flush(stdout);
    simmon(str);
    if (itrace) printf("\n"); flush(stdout);
    goto L1;
}

ptrt = 0;
ptrb = numpnt-1;

while (1) {
    if (ptrb == ptrt+1) {
        p1 = simout[ptrt];
        p2 = simout[ptrb];
        return (p2[k+1] - (p2[k+1]-p1[k+1])/(p2[0]-p1[0])*(p2[0]-t));
    }
    ptri = (ptrt+ptrb)/2;
    order = fuzcmp(t, simout[ptri][0]);
    if (order == 1) ptrt = ptri;
    else if (order == 0) return simout[ptri][k+1];
    else ptrb = ptri;
}

fuzcmp(a,b)
double a, b;
{
    if (a + 1e-8 < b) return -1;
    if (a - 1e-8 > b) return 1;
    return 0;
}

error(s1, s2)
char *s1, *s2;
{
    printf("\nERROR: ");
    printf(s1, s2);
    printf("\n\nabove message from the interface between SOLVE and SIMMON\n");
    sim_quit(); /* clean up */
    exit(1);
}

wrreal_a,b,c)
float *b;
int *a, *c;
{
    int k, i;
    char *index();

    if (++numpnt + numout == 0) return;

    if (numpnt <= 0) {
        K = numpnt + numout - 1;
        if (outnam[k] := NULL) return;
        outnam[k] = alloc (8, char);
        strncpy(outnam[k], b, 8);
        *(index(outnam[k], ' ')) = '\0';
        return;
    }
if (numpt > 2000) error ("too many simulation points");

if (simout[numpt-1] == NULL)
    simout[numpt-1] = alloc(numout+1, double);

for (i = 0 ; i <= numout ; i++)
    simout[numpt-1][i] = b[i];

return;
}

wrint_(a,b,c)
int *a, *b, *c;
{
    if (outnam == NULL) {
        if ((numout = b[1]-1) < 1) error ("no simulation output");
        outnam = alloc(numout, char *);
    }
    numpt = numout-1;
    return;
}

simon(s)
char *s;
{
    pline = 0;
    sprintf(buf[0], "%s\n", s);
    strcpy(buf[1], "stop\n");
    strcpy(buf[2], "#");
    smmain_();
    return;
}

7 Appendix B: Program simon.f

c This file is a revised SIMON main program for the purpose of use
cc in conjunction with the CAD package CONSOLE. It is now a
cc subroutine named 'smmain'. Its original form is preserved
cc by capital letters and the modifications are indicated by
cc lower case letters. All comments in the original file have been
cc removed.
cc subroutine smmain

LOGICAL LDUM1,LDUM2,LDUM3,LDUM4,LDUM5,LDUM6,ITV,IERASE,MACSUS, 1
   LDUM7
COMMON /ALLCOM/ IDD(3)
COMMON /PSCODE/ IDUM1(16000)
COMMON /VARTB1/ VARS(2,1000)
COMMON /VARTB2/ IPWTS(1000)
COMMON /VARTB3/ ITYPES6(1000)
COMMON /VALUES/ DUM3(1200)
COMMON /SYSIN1/ IDUM4,DUM4(50)
COMMON /SYSIN2/ IDUM42(50)
COMMON /SYSIN3/ IDUM43(26)
COMMON /SYSIN4/ IDUM44(26)
COMMON /EXTCOM/ IDUM6(4),DUM6(3)
COMMON /ENTRYS/ IDUM6(5)
COMMON /ENTRY/ IDUM7
COMMON /PRTS1/ IDUM8(302)
COMMON /PRTS2/ IDUM9(300)
COMMON /PWTSS/ IDUM10(300)
COMMON /PNTS4/ IDUM101(25)
COMMON /MESSS/ IDUM11
COMMON /SIMS/ LDUM1(10)
COMMON /PLT/ IDUM13(12),DUM13(20)
COMMON /STOVAR/ DUM136(4),IDUM136(101)
COMMON /AXINF/ DUM14(28)
COMMON /AXPAR/ IDUM146(10)
COMMON /TEXPAR/ IDUM148(7),DUM148(2)
COMMON /ERRWEI/ DUM18(301)
COMMON /ALG/ IDUM16
COMMON /MARKS/ LDUM2,IDUM17(6),DUM17(2),LDUM7
COMMON /USER/ LDUM3(7),IDUM18
COMMON /DESTIN/ IDUM19(2)
COMMON /SYSSTS/ IDM191
COMMON /HALLOC/ IDM192
COMMON /TIME/ DUM20
COMMON /CMVAR/ LDUM6,IDUM23(9)
COMMON /XPPUT/ IDUM24(50)
COMMON /COND/ LDUN4(26)
COMMON /LIMITS/ MPSC,NVAR,NVAL,MX,MSYS
COMMON /SIMARG/ LDUN6(2),IDUM26,DUM26(3)
COMMON /STIFF/ IDUM27(3)
COMMON /EDFILG/ FILEMN(2)
COMMON /PRLTC/IAFIL,IAIDUM(7)
COMMON /BREPAR/IBREAK,NTPLEX
COMMON /CSIERR/INODE,INODEX

logical init
common /intrf/ init
data init /.false./

if (init) go to 66

CALL IBDATA

MPSC=16000
MVAR=1200
MX=300
MVAR=1000
MSYS=26
CALL LOGPRI(-1)
CALL ISIMN
init = .true.

66 continue

MODE=1
10 CALL ESIMN(MODE,MACSUS)

GOTO(1,2,3,4,5,MODE)

1 CONTINUE
 c CALL LOGPRI(1)
 c STOP
 c CALL EXIT
 return

2 CALL SIMNSY
 CALL EQORD
 CALL VCODE
 GO TO 10

3 CALL SIMU
 GO TO 10
ITV=.FALSE.
CALL EDITOR(FILENM,.FALSE.,.FALSE.,.FALSE.,.FALSE.,ITV,IERASE,IRET,
1     .FALSE.)
IF(IRET.EQ.2) GO TO 45
GO TO 10

CALL LOGPRI(2)
c STOP
c CALL EXIT
return
END

Reference
