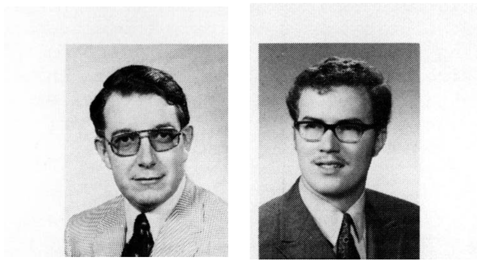

by

Edward E. L. Mitchell and Joseph S. Gauthier
Mitchell and Gauthier Associates
P.O. Box 685
Concord, Massachusetts 01742

Advanced Continuous Simulation Language (ACSL)

EDWARD E. L. MITCHELL graduated from Trinity College, Cambridge, England and received his PhD from the University of Liverpool. His interest in simulation is of long standing. He was associated with EAI's Princeton Computation Center until 1968. After moving to Raytheon, he was responsible for developing digital and hybrid missile models. He was program chairman of the 1971 SCSC and for two years chaired the SCS Committee on Continuous System Simulation Languages. Since early 1975 he has been developing a simulation language with a basic aim of (1) making its model-definition section machine-independent and (2) making models written in the language executable on as large a variety of computers as possible.

JOSEPH S. GAUTHIER is a graduate of the University of Rochester, where he received his BS in physics. During college he was introduced to the computer as a research tool and worked on the computing-center staff and as a software consultant for a local company. Upon graduation he joined Raytheon and worked as a scientific applications programmer on digital and hybrid missile simulations while taking graduate-level courses in computer science at MIT. Since early 1975 he has applied his software systems experience to the development of a simulation system capable of meeting the needs of a large proportion of those in the simulation community.

ABSTRACT
This paper describes ACSL, a continuous system simulation language designed to help the engineer or scientist with a mathematical model analyze the behavior of his system. The computer calculates dynamic responses and interactively provides pictures (plots) and tabular display of selected variables. The user has a choice of two Runge-Kutta integration algorithms and two variable stepsize algorithms which includes Gear's implicit method for stiff systems. Any other programmable algorithm can be readily incorporated. The main features of the language are illustrated by a detailed example.

INTRODUCTION

The ACSL (pronounced "axle") system is designed for modelling the behaviour of continuous systems described by time-dependent, nonlinear differential equations and transfer functions. Typical areas of application are control-system design, electrical-circuit analysis, missile and aircraft simulation, and fluid-flow and heat-transfer analysis. Program preparation can be from block-diagram interconnections or conventional FORTRAN statements or a mixture of both.

Highlights of the language are its macro capability, independent error control on each integrator, free-form input, and generation of functions of up to three variables. Many simulation-oriented operators such as variable time-delay, dead zone, backlash, and quantization are included and made readily accessible.

Macro capability allows constructing representative blocks (for instance, a transistor, an actuator, or an element in a heat-transfer problem) for systems containing state variables. Access is possible to any FORTRAN subroutine either predefined in a library or included in the ACSL model definition.

The powerful array capability allows breakdown of partial differential equations into sets of ordinary differential equations. Macro operations can pick up array dimensions so that size changes can be kept in one area in the simulation program. An array integration operator complements this facility so that vector and matrix integrations can be set up with a single statement.

Independent error control on each integrator allows faster execution by relaxing the accuracy specification on high-frequency variables. At the end of each simulation run the variable-step integration routine reports the effect each state variable had on controlling the integration stepsize so as to allow appropriate action to be taken. Relative error (MERROR) and absolute error allowable (XERROR) can both be specified individually for each state variable. For instance, relative errors on states A, B, C would be specified by the statement:

```
MERROR  A = Ø.ØØ1, B = 1.ØE-6, C = Ø.1
```

In this statement, the value for the first state becomes the default for all other unspecified states.

The integration executive picks the larger of the relative error (state value times fraction) or the absolute error and adjusts the stepsize so that the actual error lies within this bound for all states. Increasing the error allowed usually results in a larger stepsize and hence the use of less computer time—until the stability limit is approached.

ACSL is compared with other well-known simulation languages in Table 1 in a qualitative fashion. CSSL III is supported by Control Data Corporation and is available on CDC 6000 and CYBER machines. CSMP is a product of IBM Corporation and runs on 360/370 systems. MIMIC is an example of an older simulation language—it was originally developed by John Sansom for Wright Patterson Air Force Base in 1964 for an IBM 7094 but versions exist for most machines. It was by far the best of all the simulation systems developed through the 1960's.

THE *ACSL* LANGUAGE

The basic structure of ACSL follows the specification established by the SCi Technical Committee on Continuous System Simulation Language (CSSL).[1]

The simulation program is defined in a model-definition section which consists of four parts:

1. INITIAL—Statements that are performed once; typically they lead to the calculation of initial conditions on the state variables, e.g., launch angles of a missile which depend—after a fairly lengthy calculation—on target position and velocity.

2. DYNAMIC—Statements that are performed at every communication (data output) interval and whenever the data recording operation takes place. A simple example would be the conversion of radians to degrees. Efficiency is improved if those calculations relevant to data recording operations only are collected into one block and performed together. Fixed time-step operations (as in sampled-data systems or digital-computer control algorithms) can be modelled in this section.

3. DERIVATIVE—Describes the calculations that determine the derivatives of all the state variables. This section allows the integration routine selected to advance the state of the system with respect to its independent variable, usually time. The statements in this section need not be ordered but will be automatically sorted into the correct sequence so that intermediate values are calculated prior to their use.

4. TERMINAL—Statements that are performed once at the end of the simulation run. For instance, mean and standard deviation calculations for Monte Carlo sequences and the radial miss-distance of a missile from a target.

The model definition is processed by the system translator and changed into a FORTRAN program that interfaces with an executive library. This FORTRAN program in execution reads what we call model drive cards; these are sequences of commands that exercise the simulation. A complete dictionary of user symbols is built up so that data can be accessed by name:

```
DISPLAY  K9, DRAG, RT(2)
SET  K1 = Ø.3, FUNCTN = 1ØØ*1.77
```

With these two commands, the executive is first asked to print the value currently in the variables K9, DRAG, and the second element of the vector RT. Then a value of 0.3 is moved (set) into the variable K1 and 1.77 is moved into the first 100 slots of an array called FUNCTN. If sufficient slots are not available in the array, an error will be reported, values outside the array bound will not be set and processing will continue.

Table 1

Relative merits of selected simulation languages

	ACSL	CSSL III	CSMP/360	MIMIC
Object language	FORTRAN	FORTRAN	FORTRAN	Assembly
Macro processor	Extensive	Extensive	Usual	No
Free-form coding	Unrestrict	Unrestrict	Unrestrict	No
Run-time options	Extensive	Average	Average	Nominal
Stiff integration algorithm	Yes	No	Yes	No
Vector integration	Yes	No	Yes	No
Symbolic input	All names	All names	Some	None
Run-time procedures(1)	Yes	No	No	No
Translation speed	Fast	Slow	Average	Average
Dynamic storage(2)	Yes	No	No	No
Full syntax analysis(3)	Yes	No	No	Yes
Fully buffered I/O(4)	Yes	No	Yes	Yes
Time-sharing use(5)	Yes	No	No	No
Program size	Average	Large	Average	Small

Notes:

(1) Run-time procedures involve the prestoring of collections of commands, i.e., START, PLOT, PRINT..., making these accessible by a single word. These procedures can be nested to any depth.

(2) Dynamic storage means the allocation of table space according to need at the time the program is run, rather than compiling in fixed-length tables that must be sized for the most demanding task. Storage sharing can reduce total requirements if the space is needed at different times. For instance, the Gear's Stiff Integration algorithm requires an N by N array (N is the number of state variables in the simulation) to store the Jacobean or linearised state transition matrix, which is only needed while the simulation model is being run (integrating the differential equations). Printer plotting needs about a 2000-word region in order to build up the page image since the printer cannot be reversed. This space is only needed after the simulation run on a PLOT command so that this area can be shared with the integration routine requirements. Dynamic tables make this sharing automatic.

(3) A full syntax analysis eliminates the need for understanding the intermediate FORTRAN program. CSSL III and CSMP/360 rely on the FORTRAN compiler to catch most of the syntactical and procedural errors. However, this program typically contains a large number of generated variables (Z0999 etc.) and is reordered from the original source statements of the model definition. The average user finds it difficult to correlate errors reported in this FORTRAN program with the original text. The full source syntax analysis identifies errors where they occur.

(4) Fully buffered input and output (I/O) doesn't mean much in terms of central processor usage but is significant when considering turn-around time. Using conventional ping-pong buffer techniques, all the I/O operations can be overlapped with central-processor calculations, leading to a more efficient program.

(5) Most time-sharing systems allow text editing and submittal of jobs that can write data on the terminal. Time-sharing in this list, however, implies true interaction where simulation commands can be issued one by one and data displayed and modified on-line. Good practice requires means of separating high-volume data—detailed printouts, line printer plots—from low-volume data—value display—and a means of routing of the high-volume data to a local line printer, if available.

The executive library allows recording of any named variable in the computer's memory, changing any symbolic constant, and producing line-printer, pen, or CRT plots, depending on local equipment.

During model execution, procedures can be defined that represent a number of statements, each procedure being invoked by a single user-chosen name. The simulation run (integration sequence) is initiated by the command START.

Following this command, the integration routine takes over and advances the state variables until one of the termination conditions is met. The integration algorithm is selectable from four established as part of the basic system. The user may also incorporate other integration algorithms of his own choice.

The four integration algorithms included in the system are as follows:

1. Runge-Kutta Fourth Order (RK-4). A dependable fixed-step method that may not be the fastest but has almost no trouble with stability provided the integration stepsize is chosen sufficiently small.

2. Runge-Kutta Second Order (RK-2). Similar to the Fourth Order above but not quite as accurate. For some systems with small controlling time constants, this can be almost twice as fast as RK-4.

3. Adams-Moulton Variable Order, Variable Step. As implemented by Gear[2] this algorithm chooses the order (from one to six) and stepsize so as to take the largest step commensurate with satisfying the specified error criteria.

4. Gear's Stiff-Variable Order, Variable Step (Implicit). Also implemented by Gear,[2] this method is particularly suited to modelling systems that contain characteristic roots that differ by many decades. When the higher modes have died out, the algorithm can take steps many times larger than the shortest time constant, an operation not possible with any other method.

Unlike other CSSL languages, no limits exist for any of the internal tables since they all can grow to fill the available core space. Most simulation languages establish arbitrary limits on such things as number of symbols, number of state variables, labels etc. These table sizes are typically built into the translator at compile time and can only be changed by system programming personnel updating the source cards and recompiling. In ACSL no such artificial limits exist. As a corollary small programs can be executed in a smaller field-length since the simulation program does not have to be configured to accommodate the largest conceivable program any user may submit. Typical program sizes for ACSL are $55000_8 (22K_{10})$ on a CDC 6000 machine, $70000_8 (28K_{10})$ on a UNIVAC 1108.

THE MODEL EQUATIONS

An interesting example of how ACSL simplifies the task of solving a nonlinear differential equation is given by an investigation of Van der Pol's equation.[3] Describing the characteristics of a feedback oscillator, this equation contains a nonlinear damping term that causes the self-excited oscillations and limits their amplitude. Analytical solutions can be made by the method of successive approximations, but even when the series representation is obtained, time response and phase-plane diagrams are tedious to compute. ACSL provides straightforward problem solutions and easily obtained pictures of the interaction between the different variables in the system. We shall follow a Van der Pol program step by step in some detail in order to illustrate the flexibility of the run-time commands. At small signal values positive feedback causes the amplitude of the oscillation to grow, but the finite output capability of the amplifier means that the amplifier's gain must become smaller and so limit the amplitude of the oscillation. Van der Pol's equation describes a particular case of this sort. His equation is usually given in parametric form as follows:

$$\ddot{x} - \lambda(1 - x^2)\dot{x} + x = 0$$

where λ determines the growth rate of the oscillation (λ is positive and typically ranges from 0.1 to 10.0). For small x ($<<1$), the damping term or coefficient of $\dot{x}$ is $-\lambda$ and, since λ is a positive number, any oscillation will grow exponentially. When the amplitude of x becomes sufficiently large, the x^2 in the $(1 - x^2)$ term will dominate, changing the sign of the damping and leading to a stable limit cycle. Since the oscillator is nonlinear, harmonics will be generated. The object of the program we shall examine is to determine the frequency content of the output wave form. Since we see a continuous oscillation, only discrete frequencies will be present, and in actual fact it can be shown that they are the odd harmonics of the fundamental. In Rogers and Connolly[4] an approximate solution is obtained which shows that the limit cycle fundamental frequency ω_o is given by

$$\omega_o = 1 - \lambda^2/8$$

and the amplitude of the first three harmonics by

$$a_0 = 2$$

$$a_1 = 0$$

$$a_2 = 2\lambda/9$$

When the spectrum analyser is applied to a system like this, the result is more representative of the analyser itself, with its associated window, rather than of the waveform under investigation.

In ACSL, the oscillator equation becomes transformed into two statements by changing the original equation into two first-order differential equations, thus:

$$\dot{x} = \int[\lambda(1 - x^2)\dot{x} - x]\, dt$$

$$x = \int\dot{x}\, dt$$

The corresponding ACSL statements are

```
XD = INTEG(LA*(1.Ø - X**2)*XD - X, XDIC)
X = INTEG(XD, XIC)
```

where XD is the first derivative ($\dot{x}$)

XDIC is the initial condition on XD

XIC is the initial condition on X

LA is the growth rate parameter (λ).

Figure 4 shows the waveform generated for XIC = 0.1, XDIC = 0.0 and λ = 1.0, and the initial growth to a constant-amplitude limit cycle is clearly seen. Now in order to determine the harmonic content of the waveform, the components in phase with and in quadrature with (90° out of phase) a driving sine wave are determined. The root-mean-square (rms) value of these two quantities determines the signal power. The in-phase and quadrature components are determined by multiplying the waveform by $\sin(\omega t)$ and $\cos(\omega t)$ and integrating over a whole number of cycles. The ACSL statements for this operation are

```
P = INTEG(X*COS(WNEXT*T), Ø.Ø)

Q = INTEG(X*SIN(WNEXT*T), Ø.Ø)
```

```
***********ADVANCED  CONTINUOUS  SIMULATION  LANGUAGE***********
    ACSL TRANSLATOR VERSION 1 LEVEL 1A  75/Ø4/19.

PROGRAM VAN DER POL-S EQUATION
     '--------------PROGRAM CONSTANTS'
      CONSTANT       LA      = 1.Ø      , KW      = 1.Ø     ...
                   , WMN     = Ø.5      , WMX     = 5.Ø     ...
                   , NCYCLE  = 2Ø.Ø     , TSTOP   = 5Ø.Ø    ...
                   , PI      = 3.141593
     '--------------STATE INITIAL CONDITIONS'
      CONSTANT       XIC     = Ø.Ø      , XDIC    = Ø.Ø
     '--------------DEFINE COMMUNICATION INTERVAL'
      CINTERVAL      CINT    = Ø.Ø1
     '--------------PROGRAM INITIALIZATION'
INITIAL
      WNEXT   = WMN
      PZ      = QZ      = Ø.Ø
END$ ' OF INITIAL '
     '--------------ITERATE OVER ALL COMMUNICATION INTERVALS'
DYNAMIC
     '--------------INTERATE OVER INTEGRATION STEPS'
DERIVATIVE
     '--------------VAN DER POL-S EQUATION'
      XD      = INTEG(LA*(1.Ø - X**2)*XD - X, XDIC)
      X       = INTEG(XD, XIC)
     '--------------IN-PHASE AND QUADRATURE COMPONENTS'
      P       = INTEG(X*COS*WNEXT*T), Ø.Ø)
      Q       = INTEG(X*SIN(WNEXT*T), Ø.Ø)
END$ ' OF DERIVATIVE SECTION '
     '--------------NEXT FREQUENCY IN SWEEP'
      W       = WNEXT
      WNEXT   = W*KW
      LOGW    = ALOG1Ø(W)
     '--------------SPECTRAL DENSITY'
      PSD     = SQRT((P - PZ)**2 + (Q - QZ)**2)*W/(NCYCLE*PI)
      PZ      = P
      QZ      = Q
     '--------------COMMUNICATION INTERVAL OVER NCYCLE-S'
      CINT    = RSW(KW .EQ. 1.Ø, CINT, 2.Ø*PI*NCYCLE/WNEXT)
     '--------------TERMINATE ON TIME OR FREQUENCY LIMIT'
      TERMT(T .GE. TSTOP .OR. W .GT. WMX)
END$ ' OF DYNAMIC SECTION '
END$ ' OF PROGRAM '
```

Figure 1 - ACSL source-model definition statements for Van der Pol oscillator and spectrum analyser

THE MODEL DEFINITION

If we look at the listing of the simulation program, Figure 1, the first section defines constants that are to be used in the model. These are really parameters since any variable can be referred to at run-time by name and given another value. Statements are continued by adding an ellipsis (...) at the end of the line. Statements that are surrounded by quotes are treated as comments and ignored.

The communication interval is the interval over which the integration routine advances the states before returning for any data logging and executing the DYNAMIC code section. This interval is normally considered the smallest observation interval. Its value is defined in the statement

```
CINTERVAL CINT = Ø.Ø1
```

which establishes the name CINT for the variable and presets the value to 0.01. This variable will be changed later to ensure that the integration for the spectral density is performed over a whole number of cycles. The initial section is bracketed by the INITIAL...END statements and consists of statements initializing the frequency scan WNEXT to the minimum frequency WMN and the phase and quadrature base-level integrals PZ and QZ to zero.

The DYNAMIC section contains embedded within it the DERIVATIVE section that specifies the state variables and the code to calculate their derivatives. This information is used by the integration routine to advance the state step by step within the communication interval (CINT). The four INTEG statements in the DERIVATIVE section specify the derivatives of the four states XD, X, P and Q. In the rest of the DYNAMIC section the frequency is advanced geometrically by the constant multiplier KW, although this is preset to 1.0

```
W = WNEXT

WNEXT = KW*W
```

The geometric advance means equally spaced points on a logarithmic plot using LOGW, which is the logarithm of frequency (base 10) shown, i.e.,

```
LOGW = ALOG1Ø(W)
```

showing the natural use of standard FORTRAN subroutines. All variables are considered to be real unless explicitly named as integer variables. The power spectral density is obtained from the change in the integrals P and Q over a communication interval, which is made to consist of a whole number of cycles (NCYCLE) of the reference frequency W. The ACSL statement for this becomes

```
PSD = SQRT((P - PZ)**2 + (Q - QZ)**2)*W/(NCYCLE*PI)
```

The base-level integrals PZ and QZ are then set to P and Q so that the change over the next communication interval can be obtained.

The next statement shows that the communication interval is made a whole number of cycles when the frequency is being swept, but with the option to leave the communication interval unchanged. The function RSW (real switch) has three arguments, and the result is the second argument if the first is .TRUE. (logical expressions), else the third argument. Thus the statement

```
CINT = RSW(K.EQ.1.Ø, CINT, 2.Ø*PI*NCYCLE/WNEXT)
```

will not change the value of CINT if KW is equal to 1.0 (KW.EQ.1.Ø will have the value .TRUE.). If it is not, then the third argument will be selected; it is a time equal to NCYCLE cycles of frequency WNEXT.

The last statement of the program determines the stopping condition. When the argument of the TERMT operator becomes .TRUE., the simulation run will stop, and control will revert back to the run-time drive cards. The statement

```
TERMT(T.GE.TSTOP.OR.W.GT.WMX)
```

says to stop on time exceeding a stop time *or* on frequency's being swept up to the maximum value. Blocks are terminated by the appropriate number of END's. It may be noted that more than one statement may be placed on a line, but if they are, they must

```
SET TITLE = '****** VAN DER POL-S EQUATION ******'
PREPAR T, X, XD
'-----EVALUATE RESPONSE OF SYSTEM'
SET XIC = Ø.1 $ START
PRINT 'NCIPRN' = 5ØØ, 'ALL'
PLOT 'XHI' = TSTOP, X, 'LO' = -5.Ø, 'HI' = 5.Ø
PLOT 'XAXIS' = X, 'XLO' = -5.Ø, 'XHI' = 5.Ø, ...
    XD, 'HI' = 5.Ø, 'LO' = -5.Ø
'-----NOW FIND PSD BY SWEEPING W FROM WMN TO WMX'
PREPAR 'CLEAR'. W, LOGW, PSD
OUTPUT W, PSD, 'NCIOUT' = 5Ø
'-----RESET IC-S FOR PSD ANALYSIS'
SET XIC = X, XDIC = XD
'-----TERMINATE ON FREQUENCY LIMIT ONLY'
SET TSTOP = 1.ØE99, KW = 1.Ø1
START
PLOT 'XAXIS' = LOGW, PSD
PLOT 'XAXIS' = W, 'XLO' = Ø.Ø, 'XHI' = WMX, PSD
STOP
```

Figure 2 - Run-time drive cards used to exercise model

be separated by a dollar sign ($). Thus the line:

```
END$ ' OF DYNAMIC SECTION '
```

is really two statements and since the second is a quoted string, it is treated as a comment.

The model definition cards are accepted by the ACSL translator which produces a load module that reads run-time drive cards via a number of FORTRAN intermediate subroutines and an extensive library. These run-time drive cards are shown in Figure 2. Figure 3 shows the program output, which includes the echoed commands. Figure 3 would be a typical sample of the output generated at an online terminal except that it lacks a "prompt" character—usually a question mark (?)—which would appear before every input command. For CDC 6000 machines under the KRONOS operating system, this prompt character is issued to remind the on-line user that the program expects him to type something.

The executive system makes provision for separating high-volume output from low-volume output and routing the former to a nearby line printer if one is available.

These run-time commands in sequence are as follows:

```
SET TITLE = '****** VAN DER POL-S EQUATION ******'
```

The Hollerith string enclosed in the quotation marks is moved into the 120-character title array. This string will be used on subsequent plot and page headings.

The command

```
PREPAR T, X, XD
```

tells the system to save the value of the argument variables on a scratch file when the simulation is run: The interpretation is "get ready to save.' Any command statement that is completely quoted, for example,

```
'-----EVALUATE RESPONSE OF SYSTEM'
```

is treated as a comment and ignored.

The SET command

```
SET XIC = Ø.1 $ START
```

changes the value of the initial condition on the variable *X*. Remember it was preset to zero in the model-definition section by a CONSTANT statement. This SET takes the value 0.1 and places it into the variable location XIC; it will be left there until changed again later in the sequence.

More than one command can be placed on a card by separating the individual statements by a dollar sign ($). The START command actually runs the simulation, passing control to the INITIAL region of the model definition. The return to read the next command occurs only when the argument of the TERMT operator becomes true; the exception is running online when the interrupt or break-key will regain control at the command level. The simulation at this time has KW = 1.Ø; so the communication interval CINT will remain 0.01, and it will run to a stop time TSTOP of 50.0 seconds since the frequency W is not changing and so will never reach WMX. In the beginning of each communication interval, the values of the variables listed on the PREPAR list have been saved on a scratch file.

The next command

```
PRINT 'NCIPRIN' = 5ØØ, 'ALL'
```

asks to have the data recorded and listed in column form.

```
ACSL RUN-TIME EXEC  VERSION 1 LEVEL 1A    75/Ø4/19.

SET TITLE = '****** VAN  DER  POL-S EQUATION ******'
PREPAR T, X, XD
'-----EVALUATE RESPONSE OF SYSTEM'
SET XIC = Ø.1 $ START
PRINT 'NCIPRN' = 5ØØ, 'ALL'

LINE          T              X            XD
   Ø  Ø.              1.ØØØØE-Ø1  Ø.
 5ØØ  5.ØØØØE+ØØ  2.9174E-Ø1  1.2Ø95E+ØØ
1ØØØ  1.ØØØØE+Ø1 -1.5435E+ØØ  7.4349E-Ø1
15ØØ  1.5ØØØE+Ø1 -1.147ØE+ØØ -2.5641E+ØØ
2ØØØ  2.ØØØØE+Ø1  1.5456E+ØØ -7.5798E-Ø1
25ØØ  2.5ØØØE+Ø1  1.16Ø1E+ØØ  2.554ØE+ØØ
3ØØØ  3.ØØØØE+Ø1 -1.5417E+ØØ  7.6Ø49E-Ø1
35ØØ  3.5ØØØE+Ø1 -1.173ØE+ØØ -2.5434E+ØØ
4ØØØ  4.ØØØØE+Ø1  1.5379E+ØØ -7.63ØØE-Ø1
45ØØ  4.5ØØØE+Ø1  1.1859E+ØØ  2.5324E+ØØ
5ØØØ  5.ØØØØE+Ø1 -1.534ØE+ØØ  7.6552E-Ø1
PLOT 'XHI' = TSTOP, X, 'LO' = -5.Ø, 'HI' = 5.Ø
PLOT 'XAXIS' = X, 'XLO' = -5.Ø, 'XHI' = 5.Ø, ...
    XD, 'HI' = 5.Ø, 'LO' = -5.Ø
'-----NOW FIND PSD BY SWEEPING W FROM WMN TO WMX'
PREPAR 'CLEAR', W, LOGW, PSD
OUTPUT W, PSD, 'NCIOUT' = 5Ø
'-----RESET IC-S FOR PSD ANALYSIS'
SET XIC = X, XDIC = XD
'-----TERMINATE ON FREQUENCY LIMIT ONLY'
SET TSTOP = 1.ØE99, KW = 1.Ø1
START
         W 5.ØØØØØE-Ø1        PSD Ø.
         W 8.22316E-Ø1        PSD 4.72587E-Ø2
         W 1.35241E+ØØ        PSD 1.54585E-Ø2
         W 2.22421E+ØØ        PSD 8.17491E-Ø2
         W 3.658Ø1E+ØØ        PSD 3.9541ØE-Ø2
         W 5.Ø2955E+ØØ        PSD 2.33419E-Ø2
PLOT 'XAXIS' = LOGW, PSD
PLOT 'XAXIS' = W, 'XLO' = Ø.Ø, 'XHI' = WMX, PSD
STOP
```

Figure 3 - Run-time output under the run-time drive cards

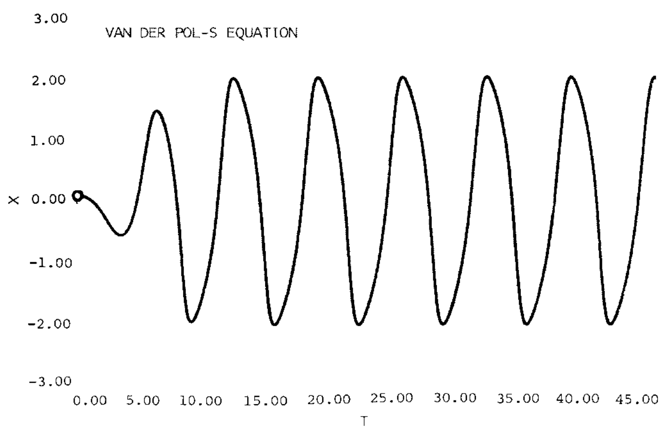

Figure 4 - Line plot of limit-cycle growth. Displacement (X) versus time (T)

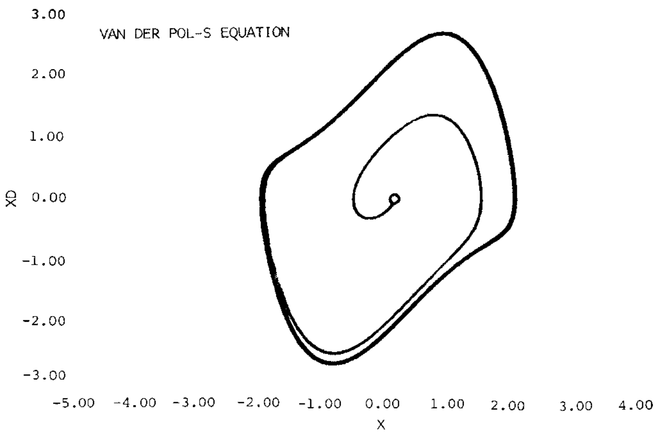

Figure 5 - Line plot of phase-plane trajectory. Rate (XD) versus displacement (XD)

The above command means to print all the variables saved on the scratch file (those on the PREPAR list) at the end of each set of 500 communication intervals. Individual variables (instead of ALL) can be named in this command and printed out every *n* communication intervals. For this example the print interval was made very large to reduce the amount of data.

Next, two line plots are made: The first (Figure 4) is a time plot in which the *x*-axis is time. If no *x*-axis variable is specified, then the first variable on the PREPAR list is used, in this case T.

```
PLOT 'XHI' = TSTOP, X, 'LO' = -5.Ø, 'HI' = 5.Ø
```

This command specifies that the *x*-axis maximum value is to be the number contained in TSTOP (= 5.0 seconds), the variable to be plotted is X, and scales are specified by the numbers following the subcommands 'LO' and 'HI.' Any number of previously PREPARed variables can be plotted and scales may or may not be specified individually. The plot could have been made by

```
PLOT X
```

in which case the system would have chosen its own scale factors, usually rounded so that they are some power of ten times 1, 2, 4, or 5.

A phase-plane plot (Figure 5) is obtained by changing the *x*-axis variable and plotting velocity (XD) against displacement (X):

```
PLOT 'XAXIS' = XD, 'XLO' = -5.Ø, 'XHI' = 5.Ø,...
          X, 'LO' = -5.Ø, 'HI' = 5.Ø
```

Note that continuation of commands is by the ellipsis (...) just as in the model definition section.

Next is the changeover so that the model can be used to determine the spectrum. The list of variables to be saved is changed by

```
PREPAR 'CLEAR', W, LOGW, PSD
```

The subcommand 'CLEAR' erases the previous list and then adds the frequency W, LOGW, and power spectrum PSD to the list to be recorded on the scratch file.

```
OUTPUT W, PSD, 'NCIOUT' = 5Ø
```

establishes a list of variables whose values are to be displayed during the simulation run. The subcommand 'NCIOUT' (number of communication intervals per output) reduces the print frequency to every 5Ø intervals to cut down the amount of paper used.

Next the initial conditions on displacement and velocity are changed by

```
SET XIC = X, XDIC = XD
```

Since the previous run was for 50 seconds, the limit cycle had become well established (see plot, Figure 4). In order to avoid the initial transient, this statement says to use the current value in *X* and *XD* and place them in the variable *XIC* and *XDIC*, respectively. Now any subsequent START will continue the limit cycle.

The command

```
SET TSTOP = 1.ØE99, KW = 1.Ø1
```

effectively makes the time stop infinite and sets the frequency multiplier so that it advances by 1% at each interval. In this way the frequency W will be swept between WMN and WMX.

Having set up all the conditions for the run, the START command transfers control to the model definition section. For each value of W, the communication interval will be adjusted to be a whole number (NCYCLE) of cycles, and the integration will result in one value of the power spectrum. As the program runs, the values of the variables mentioned on the PREPAR list are recorded and saved on the scratch file. About 230 communication intervals are needed to sweep the frequency through the range desired. The values listed for W and PSD are recorded at the OUTPUT frequency, every 50 intervals.

Now the spectrum plots (Figure 6) are made by

```
PLOT 'XAXIS' = W, 'XLO' = Ø.Ø, 'XHI' = WMX, PSD
```

The system picks its own scale factors for PSD. Note that the scales of the abscissae are specified by the subcommands 'XLO' and 'XHI.' Where a symbol is used (as in 'XHI' = WMX), the number contained in the variable is used (5.0 in this case). The simulation run is completed by the command STOP, which terminates the session.

At this point, any named variable or variables could have been changed by the SET command, and the model exercised again by START. The window used in the spectral analysis could be extended by

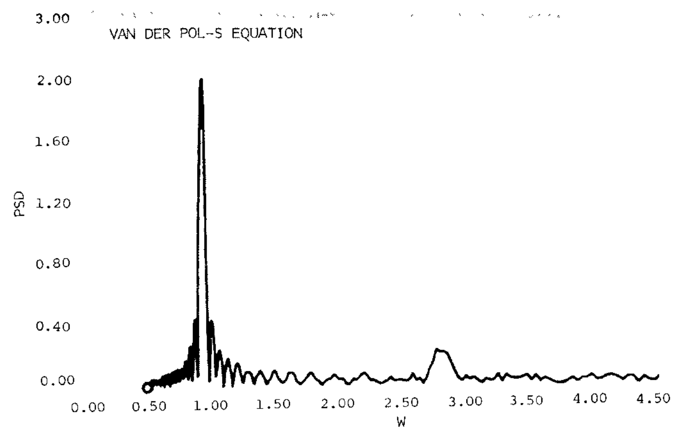

Figure 6 - Line plot of power spectrum of oscillator output (PSD) versus frequency (W)

```
SET NCYCLE = 100.0
```

which would have the effect of sharpening the spikes in the PSD plot at the fundamental and third harmonic. If the spectrum analyser is made too narrow, the frequency sweep may be too coarse and the peaks may be missed.

```
SET WMX = 10.0
```

would extend the frequency sweep to cover the fifth, seventh, and ninth harmonics.

Other commands available to help exercise the model at run-time include the following:

CONTIN	- Continue from the last state value
RANGE	- Determine maximum and minimum values of a list of variables
SAVE	- Save current state of all variables and constants
RESTOR	- Restore situation to conditions at a prior SAVE
DISPLAY	- List current value of named variables or arrays
PROCED	- Define a collection of commands (procedures) under a user defined name

To explain the action of the last command, the PROCEDure, consider making five runs, each time changing the constant K5 followed by two plots and two columnar prints. A single run would require these cards:

```
SET K5 = Ø.1
START
PLOT X, 'LO' = -5.Ø, 'HI' = 5.Ø, Y, Z, 'SAME'
PLOT V1, V2, V3
PRINT T, X, Y, V1, 'NCIOUT' = 2
PRINT T, V2, V3, V4, 'NCIOUT' = 1Ø
```

Now each subsequent run would require duplication of five unchanging cards. The PROCEDure feature collects all these cards into a unit and gives them a name by:

```
PROCED GOPLOT
START
PLOT X, 'LO' = -5.Ø, 'HI' = 5.Ø, Y, Z, 'SAME'
PLOT V1, V2, V3
PRINT T, X, Y, V1, 'NCIOUT' = 2
PRINT T, V2, V3, V4, 'NCIOUT' = 1Ø
END
...
```

Now the name GOPLOT becomes a new command that stands for the command sequence within the PROCEDure block—delimited by PROCED ... END. The command sequence for the set of runs changing K5 is now:

```
SET K5 = Ø.1 $ GOPLOT
SET K5 = Ø.2 $ GOPLOT
SET K5 = Ø.3 $ GOPLOT
SET K5 = Ø.4 $ GOPLOT
SET K5 = Ø.5 $ GOPLOT
```

with a significant saving of input volume. PROCEDures may refer to other PROCEDures within themselves with no nesting limitation.

CONCLUSIONS

ACSL has been implemented with the design engineer in mind so that results can be quickly and efficiently obtained. On-line debugging is recommended due to the ease with which the program can be run for two or three cycles, after which variable values can be DISPLAYed at will. For full runs, on-line plotters such as a ZETA plotter or Tektronix scope provide pictures and help problem solving. With only a simple terminal, the slower character plots are available, but these suffer from resolution problems when the page is only 72 characters wide.

ACSL has been designed to reduce the user's need to understand the operating system, but the degree of success varies. Under KRONOS on CDC 6000 machines, only two control cards are necessary in addition to the standard job, account, and charge cards, i.e.,

```
JOB
ACCOUNT(XYZ)
CHARGE(your project)
GET(RUNACSL/UN=ACSLSYS)
CALL,RUNACSL(MODEL=MDLFILE)
```

Under SCOPE on the same computer, about twelve control cards are needed. For the Univac EXEC 8 about seven cards are needed.

REFERENCES

1 *The SCi Continuous System Simulation Language (CSSL)* *Simulation* December 1967 pp. 281-303

2 GEAR, C.W. *Numerical Initial Value Problems in Ordinary Differential Equations* Prentice-Hall Englewood Cliffs, New Jersey 1971

3 VAN DER POL, *On Relaxation Oscillators* *Philosophical Magazine* vol. 17 1926 p.986

4 ROGERS, A.E. CONNOLLY, T.W. *Analog Computation in Engineering Design* McGraw Hill New York 1960 pp. 146-149