CONTINUOUS-SYSTEM SIMULATION LANGUAGES: A STATE-OF-THE-ART SURVEY*

NILSEN Ragnar N., Ph.D. and KARPLUS Walter J., Ph.D.
Computer Science Department - School of Engineering and Applied Science
University of California, Los Angeles, California, U.S.A.

RESUME. — La simulation est l'art et la science le comportement d'un système dynamique par un modèle et d'utiliser ce modèle validé dans une suite d'expérimentations destinées à procurer une compréhension du comportement futur du système dans des conditions données. Le processus de la simulation peut donc être considéré comme comportant quatre phases: (1) définition du problème, (2) spécification du modèle, (3) validation du modèle et (4) compréhension accélérée par l'expérimentation. En vue de faciliter ce processus de simulation, on a développé différents outils parmis lesquels le calculateur numérique.

La liaison homme-machine se fait au moyen du langage qu'il utilise pour décrire son problème au calculateur. L'aisance avec laquelle il pourra résoudre son problème dépendra de la richesse tout comme de l'aptitude du langage à être spécifique et non ambigu. D'autre part, la structure du langage dépendra de la forme du modèle à étudier. Toutefois, l'obligation d'utiliser un langage donné aura une influence certaine sur la forme à choisir pour le modèle. Il est donc nécessaire de se montrer prudent dans le choix et l'utilisation de langages pour calculateurs en vue de la simulation de systèmes.

Tous les systèmes physiques sont décrits dans un ensemble espace-temps. Le choix du modèle dépend du degré d'agrégation des phénomènes individuels. On pourra choisir soit une approche continue, soit discontinue pour le développement de ce modèle. Les modèles continus sont utiles lorsque le comportement du système dépend plus de l'agrégation de la succession des événements que de l'apparition d'événements individuels. Le choix d'un modèle continu ou discret dépendra donc de la nature du système, des objectifs de la simulation et des outils dont on dispose pour réaliser cette simulation.

La nature mathématique du modèle pour la simulation continue d'un système à entraîné le développement de langages qui sont particulièrement adaptés à étudier cette classe de problèmes. En 1973, il a été proposé plus de quarante langages de simulation continue. Cet article donnera un aperçu des langages communément disponibles (soit commercialement, soit par l'intermédiaire de groupes de recherches spécialisés) pour la simulation de systèmes dynamiques décrits par des équations différentielles. Les langages envisagés sont scindés en deux groupes. Le premier comporte les langages de simulation continue pour représenter des modèles définis par un

OVERVIEW. — Simulation is the art and science of describing the behavior of a dynamic system by means of a model and using this validated model in a series of experiments designed to provide insight into the future behavior of the system under specific conditions. Therefore, the process of simulation can be visualized as consisting of four phases: (1) problem definition, (2) model specification, (3) model validation, and (4) insight acceleration through experimentation. In order to facilitate the process of simulation, one develops various tools. The digital computer is one such tool.

Man's interface to the computer is through the language with which he describes his problem to the computer. The ease with which he solves his problem depends upon the richness of the language as well as the ability of the language to be unambiguously specific. In turn, the structure of the language depends on the form of the model to be implemented. However, having to work with a given language may bias or control the form of the model. Therefore, one must be careful in selecting and using computer languages for system simulation.

All physical systems exist in the time-space continuum. The type of model one selects depends on the degree of aggregation of individual phenomena. One may choose either a continuous or a discrete approach to model development. Continuous models are useful when the behavior of the system depends more on the aggregate flow of events than upon the occurence of individual events Choice of the continuous—or discrete— event modeling approach depends on the nature of the system, the objectives of the simulation, and the tools available to implement the simulation.

Because of the mathematical nature of the models in continuous system simulation, special languages have been evolued which are specifically adapted to expressing this class of problems. Over forty continuous-system simulation languages have been developed as of 1973. This paper will survey currently available languages (available commercially or from specific research groups) for simulating dynamic systems described by differential equations. The languages surveyed in this paper are partitioned into two groups. The first group consists of general Continuous-System Simulation Languages designed to represent dynamic models defined by sets of differential equations having primarily one independent variable. Included in the first group

système d'équations différentielles à une seule variable indépendante, entr'autres MIMIC, DYNAMO, CSMP/CSMP III, CSSL III, SL-1 et PROSE. Le second groupe comprend les langages spécialement destinés à décrire les modèles définis par des équations aux dérivées partielles, entr'autres SALEM, PDEL. LEANS, DSS, PDELAN et FORSIM.

Le but de cet article est de décrire les caractéristiques fonctionnelles des langages de simulation des systèmes continus disponibles, considérées du point de vue de l'utilisateur. L'accent est mis sur les particularités du langage, les services rendus, l'apport et l'appui des fournisseurs, sa généralité, sa souplesse et son aptitude à une extension éventuelle par l'utilisateur, sa facilité d'apprentissage et enfin les économies que son emploi permet.

are MIMIC, DYNAMO, CSMP/CSMP III, CSSL-III, SL-1, and PROSE. The second group consists of languages specifically designed to represent models defined by partial differential equations. Included in the second group are SALEM, PDEL, LEANS, DSS, PDELAN and FORSIM.

The objective of this survey is to describe the functional characteristics of currently available continuous-system simulation languages from a user's point of view. Emphasis will be placed on the unique features of the language, the services provided, the degree of supplier support, the degree of generality, flexibility in terms of user ability to extend the language, the ease of learning, and finally the economics of use.

CONTINUOUS-SYSTEM SIMULATION LANGUAGES

Continuous-system simulation languages have been developed as an aid for solving problems characterized by differential equations in one or more independent variables. Their development was further motivated by (1) the need to reduce the task of programming the digital computer, (2) the requirement to provide flexibility in changing the parameters and structure of a simulation, and (3) to provide conceptual guidance in the model development and experimental stages of simulation. This section will survey languages designed to describe dynamic systems whose model is in the form of a set of ordinary differential equations.

Historically, continuous-system simulation languages evolved out of attempts at digitally emulating the behavior of the electronic differential analyzer (analog computer)[2, 3, 4, 5]. The need for independent check solutions to analog simulations as well as the potential for increased precision motivated the early digital-analog simulators. Later with the advent of floating-point hardware, the application of digital-analog simulators was expanded to provide information needed to scale the fixed-point analog simulations. Simulation languages started out as an interconnection and control language for a collection of "analog blocks" in the form of subroutines. By the mid 1960's, FORTRAN and other procedural languages had become widely accepted by the scientific community and many of the features of conventional programming were deemed both desirable and feasible for inclusion in simulation languages. Thus with DSL/90 (1965), the advent of equation-based simulation languages augmented by a block-oriented capability, began to appear. An excellent historical review of digital-analog simulators from the initial work by Selfridge in 1953 to the work of Brennan in 1964 is presented in a paper by Brennan and Linebarger[2].

Due to the profusion of digital-analog simulators in the early 1960's, a simulation software committee, not unlike the ALGOL committee, was formed within Simulation Councils, Inc. (an AFIPS member) to formulate and provide direction for language development in the field of continuous-system simulation. The work of this committee under the chairmanship of Mr. D. Augustin of McDonald Automation Company resulted in the publication of a position paper entitled "The SCi Continuous System Simulation Language (CSSL)"[7]. The CSSL report presents recommendations for the objectives, features, and structure of continuous-system simulation languages. Rather than implement a language, the CSSL report presents its recommendations in the form of a "communication language". This allowed the committee to focus on the language design from the user's point of view with little concern over implementation problems. Since the CSSL report has become more or less the standard against which to measure new simulation languages, the objectives, features and structural recommendations of the CSSL report will be reviewed in preparation for a meaningful discussion of the six languages surveyed in this section.

CSSL Report[7]

The CSSL report published in 1967 was the first attempt at defining the desirable features of a simulation language. The recommendations presented in this report are in terms of a "communication language" for documenting dynamic-system simulations. Using the language called MIMIC as a starting point, the first task of the committee was to specify the essentials of an "ideal" simulation language. Next, they posed a collection of design goals and objectives. Finally, the language details were presented along with a structural definition characteristic of most simulation procedures.

The following lists the essentials of an "ideal" simulation language as quoted from the CSSL report.

A. Applicable areas :
   1. All-digital simulation of essentially parallel systems.
   2. Digital programming for a hybrid problem.
   3. Check solutions for modern analog and hybrid computers.
   4. Communication language (documentation).

B. Applicable programmer levels:
   1. Engineer or scientist concerned with a problem, but unfamiliar with digital computer techniques.
   2. Simulation analyst.
   3. Skilled digital application programmer.

C. Applicable digital computers:
   1. All scientific computers.

The above essentials of "ideal" language will be used subsequently to measure the degree to which the languages in this survey meet or exceed these requirements.

MIMIC

MIMIC, as supplied by CDC, is a nonprocedural language used to simulate the structure and functions required in system simulation.

The MIMIC processor consists of a set of FORTRAN IV subprograms which perform the following functions, (1) compile MIMIC source programs into an intermediate format, (2) sort the intermediate program into executable order, (3) generate machine code from the sorted form, (4) execute the simulation program including control of integration routines, parameters, and end-of-run, (5) control of post-execution operations such as listing the name and final value of each variable and plotting selected results. MIMIC is coded in a fixed and rigorous format. The following shows a typical card format where LCV contains the name of a logical control variable, RESULT contains the name of a variable:

1	2 LCV	10 RESULT	19 EXPRESSION	73
		X	INT (ARG, IC)	

and EXPRESSION contains the functional transformation which defines the behavior of the result variable. MIMIC contains the standard simulation functions such as LIMITER, TRACK and STORE, ZERO ORDER HOLD, DEAD SPACE, TIME DELAY and RANDOM NUMBER. The integration is performed via a 4th order variable-step Runge Kutta subroutine with no provision for user supplied routines. Run-time control is provided through logical control variables which allow the execution of the associated expression only if the LCV is TRUE. Termination control is also provided. Diagnostics are virtually nonexistent. Input and output functions are provided requiring rigid formats. Both PRINT and PLOT output formats are provided.

Once coded, the MIMIC program is economical to run, but because of its rigid coding requirements and limited diagnostics, it takes longer to learn, code and debug using MIMIC than using some of the more recent languages. MIMIC is still available and supported by Control Data for the CDC 6000 series computers.

DYNAMO[10]

DYNAMO, developed by the industrial dynamics group at M.I.T., is a compiler for translating and executing continuous system simulations. DYNAMO was designed for the problem-oriented rather than computer-oriented user. DYNAMO was initiated in 1959, rewritten in 1965, and the current version, written for the IBM 360/370 computers, is available through Pugh Roberts and Associates in Cambridge, Massachusetts.

DYNAMO, having evolved outside of the mainstream of CSSL development contains some unique features but also has some serious deficiencies. A special subscript notation allows the user to keep track of time. The following DYNAMO statement illustrates this time subscripting as well as the fact that only Euler integration is available.

eq.type present value.K = past value.J + DT* rate of change.JK

To facilitate the sort operation required of this nonprocedural language, the user must also identify the equation type when writing each statement. A user defined MACRO capability is provided along with system Macros to handle the simulation operators.

On the plus side, DYNAMO has an extensive set of user-oriented diagnostics as well as facilities to aid in model documentation. DYNAMO is available in both a batch version for OS/360 and an interactive version using CP/CMS 360. A load and go feature is also provided to minimize the user's interface to the operating system. Input requires a somewhat rigid card format and the output functions include tabular and plotted formats. Detracting from this easy to use language is the fact that implicit equations are trapped but not handled by the system and the use of Euler integration on a digital computer leads to either large accumulated errors or long execution times or both.

DYNAMO is easy to learn, easy to use, but to the unsuspecting, may provide misleading results due to numerical analysis type problems.

CSMP/360[11] and CSMP-III with graphics[12]

CSMP/360 evolved from the expression-based simulation language DSL/90 with CSMP-III as a vastly improved version of CSMP/360. These CSMP programs accept a simulation description including the model definition and run-time instructions and produce a FORTRAN program as an intermediate result. This allows the user access to all the features (and limitations) of FORTRAN including extensive library facilities. These CSMP programs are expression-based with the option of handling block-

oriented descriptions. They meet or exceed the CSSL report specifications in most instances. They are nonprocedural and will sort the user's time dependent equations. Provision is made for user defined PROCEDURE blocks where the user wishes to define the order of execution. Both system MACROs and user defined MACROs are provided along with the ability to handle implicit algebraic equations. Extensive translator directives allow the user to define the structure of his simulation and control the run-time environment of the simulation. Over 50 simulation functions, operators, and macros are provided in CSMP/360 and over 60 in CSMP-III.

CSMP provides seven different integration schemes including provision for a user specified subroutine. CSMP-III has, in addition, algorithms to handle stiff differential equations and the provision for double precision arithmetic not found in CSMP. Both CSMP languages provide extensive diagnostic messages as well as special debugging features. Many different input/output options are available including various tabular and plot formats.

The CSMP-III graphics feature using an IBM 2250 display, allows on-line access to the input program, the data set, the translator, the compiler and link editor, and the execution phases of the simulation.

Both CSMP/360 and CSMP-III are easy to use and can be learned at various levels. However, they are expensive to run. The use of CSMP requires at least a 360/40 with floating point hardware, a disk and 102K bytes of core memory. The convenience offered by CSMP far outweighs the cost, particularly when used by non-computer oriented personnel.

CSSL-III[13, 14]

CSSL-III is the first attempt at developing a simulation language based solely upon the CSSL report. Originally developed by Programming Sciences Inc., CSSL-III is currently marketed by Control Data Corp. It is also a nonprocedural language with a NOSORT option which translates into FORTRAN. Its free-form coding and extensive function and operator set make it powerful, yet easy to use. CSSL-III has retained the logical control variable feature of MIMIC and provides for convenient structure definition as per the CSSL report.

Nine integration algorithms including two user supplied routines are provided. Extensive error control aids are incorporated into these algorithms and are available at the discretion of the user. However, double precision is not provided in this system.

A major innovation in CSSL-III is the extensive MACRO definition capability. This provides the sophisticated user with many options for creating and extending the existing language features.

CSSL-III like CSMP has extensive user oriented capabilities which reduce programming and debugging time but are paid for in terms of resources and cost-per-run. A deficiency in CSSL-III as in CSMP and SL-1 is the lack of a file management system to handle large data sets. CDC's simpler operating system language allows the user with some study to do this, but to do file manipulation using OS/360 is out of the question for the non-computer oriented user.

SL-1[15]

SL-1 was developed in 1970 by Xerox Data Systems as a super set of the CSSL report. The SL-1 is a nonprocedural language whose translator transforms the user's simulation into a FORTRAN program, then into an assembly language (SYMBOL) program and finally into machine code. The organization allows the user to introduce both FORTRAN and SYMBOL statements; SYMBOL statements being particularly useful in real-time and hybrid situations when access to I/O and interrupts is desired by the sophisticated user.

The structure and syntax of SL-1 follows the CSSL report and is similar to both CSMP and CSSL-III. Several additional features not found in other languages make SL-1 attractive for certain sophisticated applications. SL-1 allows both multiple-rate and multiple-algorithm integration where portions of the system can be run at submultiple rates with respect to other segments of the system or can be executed using different integration algorithms. SL-1 also allows for multiple independent-variables thus extending the language to the realm of partial differential equations. A major innovation is the inclusion of real-time and hybrid computer features such as interrupt driven synchronization to real-time, analog-to-digital and digital-to-analog input and output, and analog static check.

SL-1 is available only on Xerox Σ5 and Σ7 computers using XEROX FORTRAN. It is a powerful language for the sophisticated user but tends to overwhelm the novice.

SLANG[16] and PROSE[17]

SLANG and its successor PROSE are procedural simulation languages designed for the casual user as well as problem-solving professional. As a result, much attention is paid to programming ease, natural syntax rules, readability and debugging ease. Both SLANG and PROSE were designed to permit the solution of very sophisticated mathematical problems characterized by iterative solution techniques. SLANG and PROSE depart markedly from the CSSL report and related languages (CSMP, CSSL, SL-1) in that (1) they are procedural in nature, (2) they handle both continuous-and-discrete-event simulation and (3) they provide sophisticated tools for performing nonlinear optimization, solving nonlinear implicit sets of algebraic equations and generating the Jacobian Matrix $[\partial f_i/\partial x^j]$; as well as performing numerical integration. These languages were modeled after ALGOL, SOL, SIMULA and SIMSCRIPT and provide extensive problem-solving aids and extensive diagnostics.

SLANG and PROSE are mathematically based languages with extensive execution logic and com-

mand structure. Of particular interest to problem-solvers is the ability to do optimization. Algorithms are provided to optimize continuous-systems through the generation of the Jacobian Matrix at each interation; included are nonlinear programming routines which provide for event oriented optimization. At present, the only weakness appears in the input/output facilities such as tabulated results and plotted output. PROSE promises to be a rich and powerful language applicable to a broad range of simulation problems.

PARTIAL DIFFERENTIAL EQUATION LANGUAGES

Introductory Remarks:

The use of digital simulation languages for the solution of problems characterized by partial differential equations is far less wide-spread and well-developed than is the use of such languages for ordinary differential equations. Over the past fifteen years, well over forty distinct languages have been developed for the simulation of lumped-parameter systems governed by ordinary differential equations; only five languages have been introduced specifically for distributed systems governed by partial differential equations. There are a number of reasons for this situation. First, and most important, the digital simulation languages for ordinary differential equations were conceived initially as a substitute for analog computer simulations. During the 1950's and early 1960's, the electronic analog computer assumed an enormous importance in the treatment of systems of ordinary differential equations. Digital simulation languages for ordinary differential equ. therefore found a ready and well-organized market, particularly in the aerospace and process control industries. In the case of partial differential equations, on the other hand, no single dominant analog method had been developed. In fact, although electrolytic tanks and network analogs were used at a number of laboratories, may important problems governed by partial differential equations had completely defied treatment by analog methods. Even to this day, the treatment of partial differential equations is carried out by relatively isolated groups, each concerned with a single highly specialized application area.

Furthermore, the various numerical methods available for the solution of ordinary differential equations are quite similar, differing only in the detailed nature of the integration algorithms. It is therefore relatively easy to specify a digital simulation language which is general for all types of ordinary differential equations encountered in practice. By contrast, the partial differential equations arising in physics and engineering fall into a number of distinct classes, each requiring highly-specialized numerical methods for their solution. These classes include the elliptic, parabolic, hyperbolic, and biharmonic equations in one-two-and-three space dimensions and in a variety of coordinate systems.

There exist a number of basic numerical approaches to the solution of partial differential equations. These include particularly the following finite difference methods, the method of characteristics, the Monte Carlo method, and the finite element method.

The Simulation Language Approach to Partial Differential Equations

To date, all attempts to develop higher-level languages for the treatment of partial differential equations have entailed the application of finite difference approximations, involving the following steps:

1. The time and the space domains are discretized by replacing the time space continuum by an array of regularly-spaced points, the finite difference gridpoints, and attention is focussed on the dependent variables (usually potentials) at those points.
2. The partial differential equation characterizing the field is approximated by a set of finite difference equations, one for each finite difference gridpoint. Each of these algebraic equations relates the potential at a given point to the potential at adjacent points in the time and space domains.
3. Numerical algorithms are employed to solve the set of finite difference equations, thereby providing an approximate solution of the partial differential equation. In the treatment of elliptic partial differential equations, all solution values are obtained simultaneously. In the case of time-dependent equations, solutions are generalized for successive time levels, until the time domain has been completely "marched" through in s step-wise fashion.

It is the purpose of digital simulation languages for partial differential equations to assist the programmer in carrying out the above steps and to relieve him of the burden of detailed programming. The languages which have been implemented over the past five years differ primarily in the extent to which they "decouple" the programmer from the numerical algorithms.

It is possible to solve partial differential equations using digital simulation languages designed for ordinary differential equations. Abraham[18] described the application of CSMP to the treatment of parabolic partial differential equations. While solutions of satisfactory accuracy can readily be obtained by this technique, the object program is generally highly inefficient. The use of the integration routines provided in CSMP entails the solution of the partial differential equation by an explicit method. In order to obviate the danger of computational instability, (the catastrophic accumulation and growth of roundoff errors) it is necessary to make Δt very small, so that solution time becomes very large. This limits the application of this approach to models containing relatively few finite difference gridpoints in the space domain. The first concerted effort to develop higher-level language tools specifically for the treatment of partial differential equations was the SPADE project[19] in the late 1950's. The aim of this project was to produce

a set of subroutines to permit the solution of certain elliptic and parabolic partial differential equations. Because of the limited capabilities of second generation hardware and software, this project did not achieve any substantial results. It served, however, as a forerunner for similar efforts undertaken in the 1960's.

To date at least five digital simulation languages for partial differential equations have been implemented. The main points of difference between these languages are:

1. *Generality:* the number of classes and subclasses of partial differential equations which can be handled by the language.
2. *Flexibility:* the extent to which the user has the ability to specify or select that algorithm which he feels would be most effective for this specific problem.
3. *Procedure Orientation:* the object programs prepared by each of the language translators is necessarily procedural, consisting of a sequence of declarations and commands. On the other hand, partial differential equations and the finite difference models as they are originally formulated are "parallel" in that all points within the space domain are active simultaneously, In a language which is nonprocedural, the user is not required to consider the sequence or order in which calculations will be carried out on the digital computer.
4. *Target Language:* all of the languages described below are implemented as higher-level languages in that they effect the translation of the source program into another programming language - the target language. Most existing languages for partial differential equations employ FORTRAN as the target language, while at least one uses PL/I.

The digital simulation languages for partial differential equations which have successfully implemented to date are briefly described below approximately in the order in which they first appeared in the literature.

SALEM[20,21]

Developed at Lehigh University, SALEM was an experimental system including a translator and an assortment of FORTRAN subroutines. The user expressed his partial differential equation in FORTRAN-like terms together with the specified boundary and initial conditions. In addition, the user specified a call for a FORTRAN subroutine which was then invoked to solve the problem. The system was implemented for one and two-dimensional parabolic and elliptic equations as well as one-dimensional hyperbolic equations.

For example, the equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial T}{\partial y^2} = \frac{\partial T}{\partial t} \qquad (1)$$

is expressed as:

$$D2(T,X) + D2(T,Y) - 5D1(T,TM) = 0 \qquad (2)$$

For problems in one space dimension, the tri-diagonal algorithm is provided as a subroutine, while problems in two space dimensions employ the alternating direction technique. The program includes provisions for modifying the time step in accordance with truncation-error estimates, as well as diagnostics to pinpoint errors in formulation and programming. The SALEM system is no longer being maintained by its originators.

PDEL[22,23,24,25,26,27,28]

PDEL was developed at the University of California at Los Angeles in 1967, and has been continuously used and expanded since that time. Like SALEM, PDEL is completely nonprocedural in character, the user merely specifying the partial differential equation together with boundary and initial conditions; no reference to algorithms or detailed solution procedures need to be made except for the specification of the finite difference grid spacing in the space and time domains. The system is capable of solving linear and nonlinear elliptic, and parabolic partial differential equations in one two or three space dimensions as well as hyperbolic equation in one space dimension. Equation (1) is expressed in PDEL as

```
% EQUATION = 'S*PX,PX,PHI + S*PY,PY,PHI
                 = K*PT,PHI';                    (3)
% S = '1.0';
% K = '5.0';
```

PDEL is PL/1-based and is highly modular in nature. The PDEL translator invokes subroutines from a large and varied library. The tri-diagonal algorithm is used for problems in one space dimension, and subroutines for the alternating direction method (ADI) and the successive overrelaxation method are available for problems in higher space dimensions. Recent extensions of the PDEL language include modules to facilitate parameter identification and modeling. The translator includes very extensive error-diagnostic facilities.

LEANS[29,30,31,32,33,36]

LEANS was developed at Lehigh University. Compared to SALEM and PDEL, LEANS provides considerably more generality and flexibility at some sacrifice of simplicity and user convenience. The language is based upon FORTRAN IV and is subroutined oriented. The user does not write his specific equation as a part of the LEANS program. Rather he specifies the coefficients of a general equation such as

$$\begin{aligned} A_2\partial^2 u/\partial t^2 + A_1 \partial u/\partial t = {} & \\ & \partial(A_3 u)/\partial x + (1/x^c)\partial(x^c A_4 \partial u/\partial x)/\partial x + \\ & \partial(A_7 u)/\partial y + (1/y^d)\partial(y^d A_8 \partial u/\partial y)/\partial y + \\ & A_5 u + A_6 \end{aligned} \qquad (4)$$

where the coefficients c, d and e have values of 0, 1 and 2 for Cartesian, cylindrical and spherical coordinates respectively. For example, for Equation (1), the user's program would include the specifications that

$$\begin{aligned} &A_1 = 5 \\ &A_2 = A_3 = A_5 = A_6 = A_7 = 0 \\ &A_4 = A_8 = 1 \\ &c = d = e = 0 \end{aligned} \tag{5}$$

The system is therefore capable of handling elliptic, parabolic and hyperbolic equations in one, two or three dimensions and in Cartesian, cylindrical and spherical coordinates. The LEANS system is based upon differential-difference equation algorithms. Finite difference approximations are made of all partial derivatives with respect to space variables only. There results a system of simultaneous ordinary differential equations, one for each finite difference gridpoint in the space domain. A user/selected algorithm is invoked to solve this system of ordinary differential equations. In order to avoid computational instability, Δt must be made rather small in some applications. LEANS is maintained and documented by Lehigh University.

DSS[34, 35]

DSS is a FORTRAN-based simulation language developed at Lehigh University. Just as in the case of LEANS, the user specifies his partial differential equation by assigning values in a general partial differential equation, in this case

$$\begin{aligned} &1/y^b \partial(y^b A_{6j} \partial u_j/\partial y)/\partial y + \\ &1/x^a \partial(x^a A_{1j} \partial u_j/\partial x)/\partial x - \partial(A_{2j} u_j)/\partial x - \\ &A_{3j} \partial u_j/\partial t - A_{4j} u_j = A_{5j} \end{aligned} \tag{6}$$

the subscript $_j$ denotes the equation for dependent variable$_j$ in a system of simultaneous equations. Note that there is no provision for hyperbolic partial differential equations. The equation coefficients and boundary conditions may be constant, variable, or nonlinear. Three user-supplied subroutines must be provided to describe the coefficients, initial conditions and boundary conditions. DSS differs from LEANS in that all independent variables are approximated by finite difference expressions and that unconditionally + stable implicit finite difference approximations are employed. The tri-diagonal algorithm is used for one-dimensional problems while Peaceman-Rachford method is used for two-dimensional elliptic and parabolic partial differential equations. Truncation-error correction is provided with the aid of the Richardson extrapolation. Diagnostic facilities are provided for input data errors and convergence failures. The present configuration can handle five simultaneous one-dimensional partial differential equations and two simultaneous two-dimensional partial differential equations.

PDELAN[37, 38]

Developed at the National Center for Atmospheric Research, Boulder, Colorado, PDELAN represents an additional step in the direction of generality with concomitant increased demands upon the user. PDELAN is completely procedural in character and requires that the user have a considerable amount of familiarity with the details of the finite difference algorithms. The user declares the size of one or more rectangular finite difference grids, as well as the simultaneous dependent variables associated with each grid. The user must also declare and specify all finite difference operators. The DOMESH command specifies how these operators are to be applied throughout the grid. The language appears to be particularly useful for the systems of simultaneous nonlinear parabolic partial differential equations which arise in meteorology. Since the number of finite difference gridpoints in such a problem is extremely large, execution efficiency is of paramount importance, and PDELAN makes a relatively small sacrifice in running time, compared to the same program written entirely in FORTRAN.

FORSIM[39]

FORSIM was developed at the Chalk River Nuclear Laboratories, Ontario, Canada and is also a FORTRAN-based language. The program was originally conceived as a tool for systems of ordinary differential equations, and extension to partial differential equations involves the same difference-differential equation approach as is used in LEANS. As in that language, the user is required to specify the coefficients of a general partial differential equation, in this case

$$\begin{aligned} A_1 \frac{\partial^2 u}{\partial t^2} + A_2 \frac{\partial u}{\partial t} &= \frac{\partial}{\partial x}(A_3 u) + \\ &+ \frac{1}{x^c} \frac{\partial}{\partial x}\left(x^c A_4 \frac{\partial u}{\partial x}\right) + A_5 u + A_6 \end{aligned} \tag{7}$$

where the coefficients A_1 as well as the boundary conditions may be constant, functions of x and/or t, and where the exponent c defines the coordinate system. The system is designed primarily for initial value problems which fit the general format of Equation (7), but equations involving higher order or mixed derivatives may be handled by providing suitable subroutines. The user may specify the number of special divisions, and whether a 3-point or 5-point difference approximation of the spatial derivatives are to be used.

Concluding Remarks

This paper has surveyed six commercially available continuous-system simulation languages and six languages specifically designed to solve partial differential equation problems. Space has limited this survey to the more important, available, and innovative languages currently used in system simulation.

The choice of which language is most suitable for a particular problem solver depends on:

1. The extent of the problem, model complexity, and degree of experimentation required.
2. User sophistication—problem-orientation vs. computer orientation.

3. Available funding.
4. Available computer resources.

The selection is in general based on the need for a special service unique to a particular language. These special services may be summarized as follows:

MIMIC	– easy and inexpensive to use, easy to learn
DYNAMO	– language natural to non-engineering disciplines
CSMP/CSMP-III	– complete expression-oriented language with graphics
CSSL-3	– extensive set of MACRO directives
SL-1	– real-time, hybrid computer, and multiple derivative features
PROSE	– problem solving packages for optimization, nonlinear programming and parameter estimation
SALEM	– no longer supported
PDEL	– a PL/1 oriented system with considerable flexibility
LEANS	– handles elliptic, parabolic and hyperbolic equations
DSS	– will not handle hyperbolic equations
PDELAN	– structured for problems in meteorology
FORSIM	– a PDE interface to FORTRAN

The user is reminded of Hamming's statement that "The purpose of computation is insight not numbers". One uses simulation to gain insight into the behavior of systems. The models one uses are biased to a great extent by his choice of descriptive language. Therefore, the user should be familiar with constructive alternatives in order to make an intelligent selection as to which language is "best" suited for his class of problems.

BIBLIOGRAPHY

1. DELAND, E.C., G.A. BEKEY and G.P. MOORE. "Toward a Natural Simulation Language", *Proceedings of the Summer Computer Simulation Conference,* Boston, 1971, pp. 15-19.
2. BRENNAN, R.D. and R.N. LINEBARGER. "A Survey of Digital Simulation: Digital Analog Simulator Programs", *Simulation,* Vol. 3, No. 6, December 1964, pp. 22-36.
3. BRENNAN, R.D. "Continuous System Modeling Programs: State of the Art and Prospectives for Development", *Simulation Programming Languages,* North Holland Pub. Co., 1968.
4. BRENNAN, R.D. and R.N. LINEBARGER. "An Evaluation of Digital Analog Simulator Languages", *Proceedings IFIP Congress '65,* New York, 1965, Vol. 2.
5. TIECHROEW, D., J.F. LUBIN and T.D. TRUITT, "Discussion of Computer Simulation Techniques and Comparison of Languages", *Simulation,* October 1967, pp. 181-190.
6. CLANCY, J.J. and M.S. FINEBERG, "Digital Simulation Languages: a Critique and a Guide", *Proceedings of the Fall Joint Computer Conference,* 1965, Vol. 27, pp. 23-36.
7. "The SCI Continuous System Simulation Language (CSSL)", *Simulation,* Vol. 9, No. 6, December 1967, pp. 281-303.
8. PETERSON, H.E. and F.J. SANSOM. *MIMIC Programming Manual,* Air Force Systems Command, Wright Patterson Air Force Base, Ohio, 1967.
9. *MIMIC Digital Simulation Language, Reference Manual,* Control Data Corporation, Sunnyvale, California, 1968.
10. PUGH, A.L. *DYNAMO II User's Manual,* M.I.T. Press, Cambridge, Massachusetts, 1970.
11. *System/360 Continuous System Modeling Program (CSMP), User's Manual,* a type II program number 360A-CX-16X, Manual Number GH 20-0367, 1968, available from IBM, White Plains, New York.
12. *Continuous System Modeling Program III (CSMP-III) and Graphic Feature, General Information Manual,* a program product number 5734-X59, Manual Number GH 19-7000, 1971, available from IBM, White Plains, New York.
13. *CSSL-III User's Guide and Reference Manual,* Programming Sciences Corporation, Los Angeles, California, 1970.
14. *Continuous System Simulation Language Version 3, User's Guide,* Control Data Corporation, Sunnyvale, California, 1971.
15. *SL-1 Reference Manual,* Xerox Data Systems, El Secundo, California, 1970.
16. THAMES J.M. "SLANG a Problem Solving Language for Continuous-Model Simulation and Optimization", *Proceedings of the 24th ACM National Conference,* San Francisco, 1969, p. 23.
17. THAMES, J.M. "PROSE - a Problem-Level Programming System", Solveware Associates, San Pedro, California, 1973.
18. ABRAHAM, F.F. "Computer Simulation of Diffusion Problems Using the Continuous System Modeling Program Language", in *An Introduction to Computer Simulation in Applied Science,* Plenum Publ., New York, 1972, pp. 71-106.
19. YOUNG, D.M. and M.J. JUNCOSA. "SPADE, a Set of Subroutines for Solving Elliptic and Parabolic Partial Differential Equations", Rand Corporation, Santa Monica, California, P-1709, May 1959.
20. MORRIS, S.M. and W.E. SCHIESSER. "Salem - a Programming System for the Simulation of Systems Described by Partial Differential Equations", *Proceedings of AFIPS 1968, Fall Joint Computer Conference,* November 1968, Vol. 33, Pt. 1, pp. 353-357.
21. MORRIS, S.M. "SALEM - a Programming System for the Simulation of Systems Described by Partial Differential Equations", Ph.D., Lehigh University, Bethlehem, Pennsylvania, 1967.
22. CARDENAS, A.F. "A Problem Oriented Language and a Translator for Partial Differential Equations", Ph.D., University of California, Los Angeles, December 1968.
23. CARDENAS, A.F. and W.J. KARPLUS. "PDEL - A Language for Partial Differential Equations", *Communications of the ACM,* Vol. 13, No. 3, March 1970, pp. 184-191.
24. CARDENAS, A.F. and W.J. KARPLUS. "Design and Organization of a Translator for a Partial Differential Equation Language", *Proceedings of the Spring Joint Computer Conference,* Vol. 36, May 5-6, 1970, pp. 513-524.
25. KARPLUS, W.J. and A.F. CARDENAS. "Experience with the Partial Difference Equation Language PDEL: an Alternative to Hybrid Computation", *Proceedings of the Sixth International Meeting on Analog and Hybrid Computation,* AICA/IFIPS, Munich, September 1970, pp. 724-729.
26. FAIRLEY, R.E. and A.F. CARDENAS. "Batch and Interactive Simulation of Partial Differential Equation Models", *Proceedings of the 1971 Summer Computer Simulation Conference,* Boston, July 14-21, 1971.
27. TAM, W.C. "Digital Simulation Language for Distributed Parameter System Identification", Ph.D., University of California, Los Angeles, June 1972.
28. CARDENAS, A.F. "The PDEL Language and its Use to Solve Partial Differential Equation Problems", University of California, Los Angeles, Computer Science Department Report, June 1972.
29. HUTCHINSON, R.W. and W.E. SCHIESSER. "Truncation Error Correction Applied to Non-linear Partial Differential Operators", *Proceedings of the 1971 Summer Computer Simulation Conference,* Boston, July 1971, pp. 112-117.
30. SCHIESSER, W.E. "Some Recent Developments in Software for Chemical Reactor Simulation", *Proceedings of the 1971 Summer Computer Simulation Conference,* Boston,

July 1971, pp. 388-396.
SCHIESSER, W.E. "A Digital Simulation System for Mixed Ordinary/Partial Differential Equation Models", *Proceedings of IFAC Symposium on Digital Simulation of Continuous Processes,* Gyor, Hungary, September 1971, Vol. 2, pp. S2-1 to S2-9.
32. SCHIESSER, W.E. "A Digital Simulation System for Higher-Dimensional Partial Differential Equations", *Proceedings of the 1972 Summer Computer Simulation Conference,* San Diego, California, June 1972, pp. 62-72.
33. RABAS, T.J. and W.E. SCHIESSER. "A Digital Simulation System for Heat Transfer Modelled by Ordinary and Partial Differential Equations", *1972 Aiche-Asme Heat Transfer Conference,* Denver, Colorado, August 1972. (Paper No. 72-Ht-25).
34. ZELLNER, M.G. "DSS - Distributed System Simulator", Ph.D., Lehigh University, Bethlehem, Pennsylvania, 1970.
35. ZELLNER, M.G., R.W. COUGHLIN, F.P. STEIN and W.E. SCHIESSER. "A DSS Study of the Kinetics of Adsorption and Chemical Reaction", *Proceedings of the 1970 Summer Computer Simulation Conference,* Denver, Colorado, June 1970, pp. 284-294.
36. SCHIESSER, W.E. "An Introduction to LEANS-III, Lehigh Analog Simulator Version III and DSS Distributed System Simulator", Lehigh University, Bethlehem, Pennsylvania, Computing Center Report, 1973.
37. HELGASON, R. and J. GARY. *PDELAN User's Manual Version 1,* National Center for Atmospheric Research, Boulder, Colorado, Manuscript No. 71-130, 1971.
38. GARY, J. and R. HELGASON. "An Extension of FORTRAN Containing Finite Difference Operators", *Software - Practice and Experience,* Vol. 2, 1972, pp. 321-336.
39. CARVER, M.B. "A FORTRAN-Oriented Simulation System for the General Solution of Partial Differential Equations", *Proceedings of the 1973 Summer Computer Simulation Conference,* Montreal, July 1973.