CONTINUOUS-SYSTEM SIMULATION BY USE OF DIGITAL COMPUTERS: A STATE-OF-THE-ART SURVEY AND PROSPECTIVES FOR DEVELOPMENT.

François E. Cellier, Institute for Automatic Control,
The Swiss Federal Institute of Technology Zurich,
Physikstr. 3, CH-8006 Zürich, Switzerland

I) ABSTRACT

This paper briefly describes and surveys existing methods and program packages for the simulation of continuous systems described by ordinary differential equations. The specific problems arising in selecting the digital computer as a tool to perform simulation studies are outlined. The major demands on and features of simulation packages such as algorithms for numerical integration, the sorting procedure, the associated procedural language and the associated macro language are discussed in detail, and a comparison of several commonly used simulation packages is given in tabular form. General trends for the development of new simulation packages as well as some personal suggestions towards this, close the paper.

II) INTRODUCTION

Unlike analog computers the digital computer is from its design not suited for being used as a tool for the simulation of continuous systems. There are two reasons for this:

1) Since digital computers are discrete by their nature, integration is hard to perform on them. It is necessary to define algorithms which allow quasicontinuous operations. The suitability of an algorithm highly depends on the problem to be solved.For this reason a general purpose simulation package should contain several integration algorithms, out of which one may choose the best suited algorithm in accordance with the specific problem to be solved. Besides, it is desirable to design a package in a way such that a user may easily insert a self-coded integration routine into the system without need for further knowledge of the specific compiler.
2) Physical processes are basically parallel whereas the digital computer can only execute one command after the other in a strictly procedural way. To enable the user to describe his problem in a straightforward manner a sorting algorithm is, therefore, required to bring the structural statements describing the process into the requisite sequence. Any simulation package showing these attributes can thus be divided into two parts:
   a) a preprocessor which "understands" and processes the user-supplied code by translating it from a problem adequate form into a procedure adequate form which can be handled by
   b) an algorithmic part which is activated only afterwards to perform the simulation.

Despite both of the above mentioned disadvantages the digital computer is these days much more frequently applied to simulation problems than analog- or hybrid computers. For this three reasons may be quoted:

1) Most simulation users have a digital computer at their disposal but only few have access to analog- or hybrid computers.
2) Once a simulation package exists coding of a specific problem by use of an existing simulation package is even easier done on a digital than on an analog or hybrid computer, especially if the model contains any decisions or other logic besides the pure state equations.
3) The scaling problem of analog computers [1] can almost be omitted.

The procedure for performance of simulation on a digital computer is outlined in Fig. 1. To carry out the entire coding of the procedure described is complicated and time consuming. However the more difficult parts of it such as the integration algorithm are taken care of by simulation packages if used. Three types of simulation packages can be distinguished:

1) general purpose procedural package
2) general purpose parallel package
3) special purpose simulation package

The functions taken care of by these packages are indicated by dashed areas in Fig. 1. This figure has been reproduced with some modifications from [2] with the friendly permission of the editor.

The procedure of Fig. 1 can be summarized by the following:

1) derivation of a model from a given physical system.
2) problem adequate formulation of the model.
3) procedure adequate formulation of the model.
4) integration of the model into a simulation program.

Most of the effort goes into (1) and (4). A general purpose sequential simulation package such as GASP-IV or FORSIM-V takes care of (4). A general purpose parallel simulation package such as CSMP-III, CSSL-III, MIMIC or DARE-P accomplishes the functions (3) and (4). If a problem can be formulated in a special purpose simulation language (e.g. network analysis problems may be solved by use of ECAP, LISA, ASTAP or SCEPTRE), the function (2) is processed automatically as well to a large extend.

Careful considerations are required to determine the package best suited for the solution of a specific problem. It is impossible to give a general solution to this problem since it highly depends upon:

1) the availability of a certain package at a certain installation.
2) the experience of the programmer.
3) the problem itself.

Highly specialized simulation packages are easy to use. This greatly reduces the time needed for describing the problem to the computer in a proper way. This ease in handling the package, however, is paid in terms of higher execution time and lower flexibility. For a skilled simulation analyst availability of a sorting option may be more of a hindrance than of a help

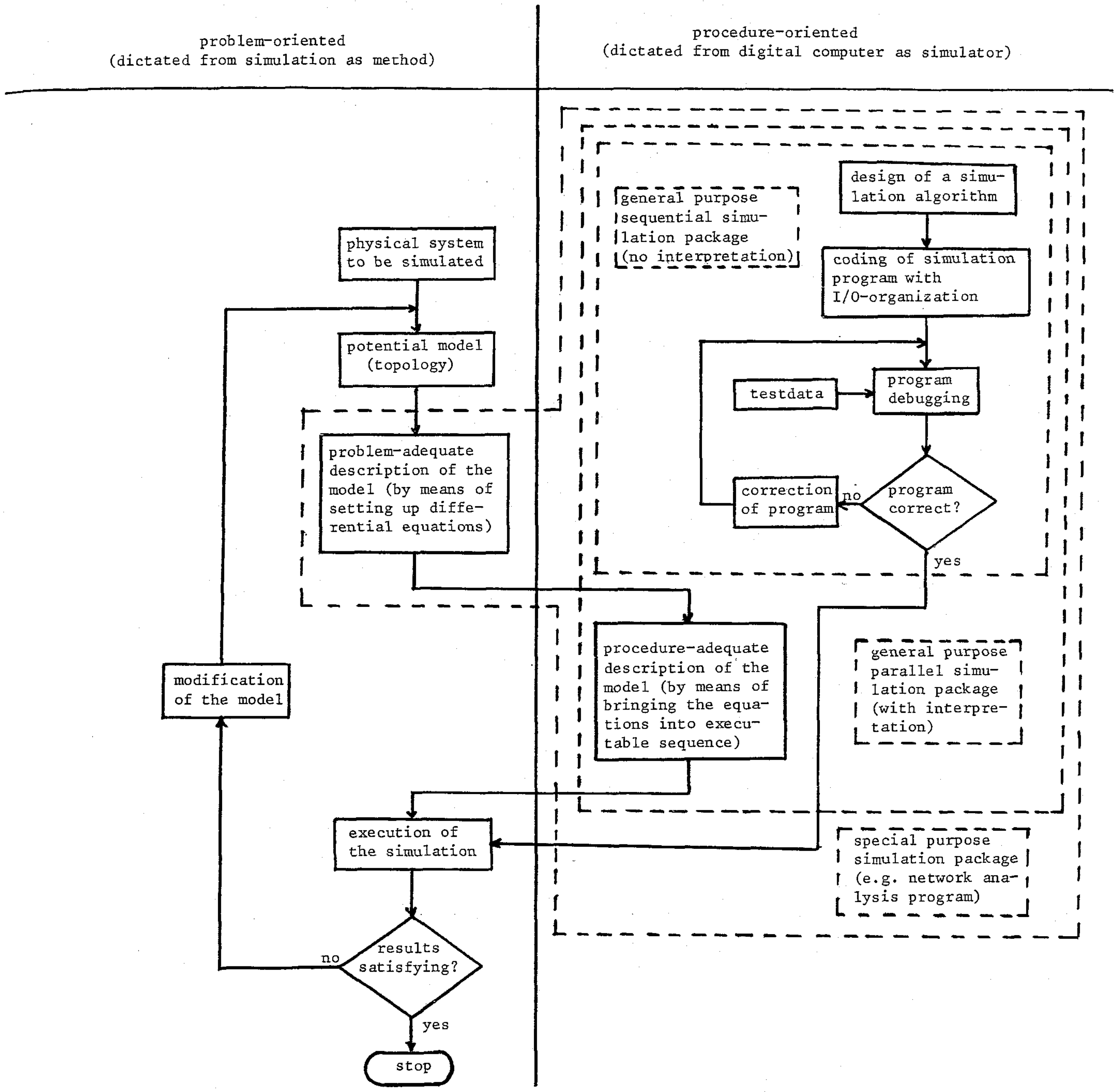
problem-oriented
(dictated from simulation as method)
procedure-oriented
(dictated from digital computer as simulator)
physical system to be simulated
potential model (topology)
problem-adequate description of the model (by means of setting up differential equations)
modification of the model
procedure-adequate description of the model (by means of bringing the equations into executable sequence)
execution of the simulation
results satisfying?
no
yes
stop
general purpose sequential simulation package (no interpretation)
design of a simulation algorithm
coding of simulation program with I/O-organization
testdata
program debugging
correction of program
no
program correct?
yes
general purpose parallel simulation package (with interpretation)
special purpose simulation package (e.g. network analysis program)

Fig. 1 Steps from the physical system to the execution of a digital simulation program

whereas the novice user may find it very helpful since he may have great difficulty in arranging his statements into the sequence required by the integration algorithm.

The following chapters will give a more detailed description of the above mentioned. The problem of stochastic process simulation will not be discussed although stochastic signals may be involved in continuous system simulation, since the problem is similar to stochastic simulation of discrete systems where it is treated by two other papers of this course [3,4].

III) NUMERICAL INTEGRATION

The integration algorithm is the "heart" of each program for continuous system simulation and at the same time is the most difficult part to be coded. At most installations, however, the engineer will find library routines performing integration of ordinary differential equations, so he need not be concerned with this problem. Since the formulation of scientific integration algorithms, these days, is normally done by mathematicians, the author of this paper will not go into details of such algorithms. He restricts himself to characterize the most commonly used algorithms and to give some considerations concerning the selection procedure. So the integration routine will be treated as a black box of the form shown in Fig. 2.

The integration algorithm has certain known attributes and calls a user supplied subprogram in which the system to be simulated is modeled by a set of state equations plus possibly some additional logic.

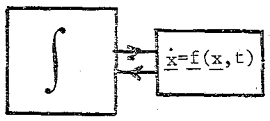

Fig. 2 Black box representing integration algorithm for integration of ordinary differential equations

It is necessary to classify the different types of algorithms to illustrate their specific attributes.

One distinguishes between

a) explicit algorithms

b) implicit algorithms

Explicit algorithms are *memory-functions* which means that computation of the output vector $\underline{x}$ at time t does not depend upon the input vector $\underline{\dot{x}}$ at time t but is computed by use of information $(\underline{x},\underline{\dot{x}})$ which has entirely been evaluated at past values of time. Implicit integration algorithms belong to the *history-functions* which means that the computation of $\underline{x}(t)$ requires knowledge of $\underline{\dot{x}}(t)$ in addition. Such algorithms are implicit since they introduce algebraic loops

$$\underline{x}(t) = f_1(\underline{\dot{x}}(t),\underline{x}(t-\Delta t),\underline{\dot{x}}(t-\Delta t),\underline{x}(t-2\Delta t),\ldots)$$
$$\underline{\dot{x}}(t) = f_2(\underline{x}(t),t)$$

The first equation describes the integration algorithm, the second describes the user supplied state equations. Both equation sets define an algebraic loop since $\underline{x}(t)$ is a direct function of $\underline{\dot{x}}(t)$ and vis versa.

General purpose simulation packages always make use of explicit integration since the better numerical stability behaviour of implicit algorithms normally does not justify the much higher computation time needed for carring out an entire iteration for the latter at each time step to solve the algebraic loop. Frequently a combination of explicit and implicit algorithm is used by taking an explicit method for predicting the next value of $\underline{x}$ and improve the value obtained by using an implicit method for correction. These methods are called predictor-corrector-methods.

Predictor: $\underline{x}^P(t) = f_1(\underline{x}(t-\Delta t),\underline{\dot{x}}(t-\Delta t),\ldots)$

$\underline{\dot{x}}^P(t) = f_2(\underline{x}^P(t),t)$

Corrector: $\underline{x}(t) = f_3(\underline{x}^P(t),\underline{x}(t-\Delta t),\underline{\dot{x}}(t-\Delta t),\ldots)$

$\underline{\dot{x}}(t) = f_2(\underline{x}(t),t)$

The combined algorithm is explicit and, therefore, does not require any iteration. (Example: Adams-Bashforth-Predictor,Adams-Moulton-Corrector Methods).

Linear Network Analysis Programs normally make use of the trapezoidal rule (first order, implicit) since iteration can be reduced to a matrix inversion in this special case. This gives better stability behaviour (as implicit methods usually do) and at the same time makes the sorting procedure superfluous which also is a big advantage since complex networks often anyway involve algebraic loops.

One also distinguishes between

a) one step methods

b) multistep methods

One step methods compute $\underline{x}$ at time t out of information on $\underline{x}$ and $\underline{\dot{x}}$ at one time step back $(t-\Delta t)$ and possibly at additional values between $(t-\Delta t)$ and t. From this class of methods the Runge-Kutta-Methods (1st order=EULER, 2nd order=improved EULER and method of HEUN; coefficients calculated up to 8. order) are best known.

Multistep methods make use of information which lies further back in time, but do not consider any information at values of time different from $t,t-\Delta t,t-2\Delta t,\ldots$ From this class of methods the Adams-Bashforth and Adams-Moulton methods of various orders (1st order=trapezoidial rule, 2nd order=method of SIMPSON) are best known. Also MILNE's method belongs to this class.

From what has been stated above it is evident that one step methods are selfstarting, whereas multistep methods need to be started by performing the first k steps ($k \triangleq$ number of time steps back needed for evaluation of $\underline{x}$) using a one step method or by defining an iteration procedure to obtain the back information required. Multistep methods are very useful in case of systems under investigation which do not involve any switching actions since these methods normally require less computational time for a given accuracy. In case of switching actions forming part of the system, one step methods should be used by adjusting the step size in a way that no switching activity takes place in the middle of a step. Multistep methods would have to be restarted at each instant of time a discrete event takes place. This is necessary since all integration algorithms are defined only for continuous systems. In case of systems involving switching activities multistep methods are, therefore, not recommended.

Special methods have to be used for the treatment of stiff systems ($\triangleq$ systems with widespread time constants). The best known algorithm to integrate stiff systems has been reported by C. William Gear [5].

A crucial problem is the selection of optimal step size and order. The step size should in most cases be optimized automatically to obtain minimal computation time for a given accuracy. Various algorithms have been reported on how to compute the optimal step size as a function of time by approximation of the integration error. If, for example, a Runge-Kutta algorithm of 4th order is used, 4 evaluations of the state equations are required to compute $\underline{x}(t)$. An approximation of the local integration error can be obtained for the price of only one additional evaluation of the state equations. So the computational time is increased by 25 %. This increase is normally justified since the loss in computation time if too small a step size is used will mostly be much higher than 25 %. Besides, the optimal step size for a specific simulation is not necessarily constant but may vary with time. On the other hand the EULER - algorithm requires only one evaluation of the state equations whereas an approximation of the integration error requires two more evaluations. In this case the computational time is increased by 200 % which for most applications will not pay out. Summarizing: low order methods should rather be used with fixed step size whereas higher order methods under all circumstances should be used together with an algorithm to compute the optimal step size.

The optimal order of the algorithm to be chosen depends very much upon the accuracy required. Low accuracy suggests application of low order algorithms, whereas high accuracy demands the choice of higher order methods. If the step size has to be reduced frequently due to switching activities taking place, low order algorithms also are preferable. In case of dynamic nonlinear programming problems where each evaluation of the performance index of the optimization procedure involves a whole simulation run the required accuracy for the simulation itself should be chosen to be a function of the gradient of the performance index. As long as the solution of the nonlinear programming problem is still far away from its optimum the accuracy requirements of the simulation runs are low. They become more stringent the more the optimum is approached. This is illustrated in Fig. 3. For such problems, therefore, also the order of the integration algorithm should be computed automatically.

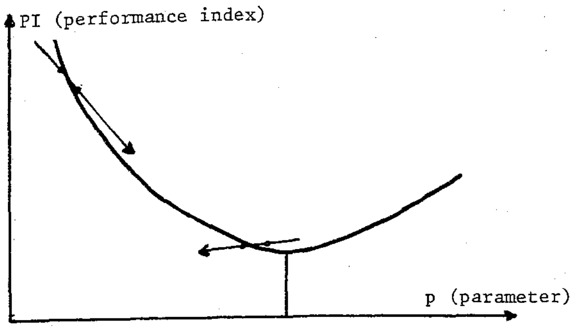

Fig. 3 The performance index of a nonlinear programming problem as a function of a parameter. Errors in the computation of the gradient are more stringent the more the optimum is approached

IV) THE SORTING PROCEDURE AND THE STRUCTURING OF MODELS. THE ASSOCIATED PROCEDURAL LANGUAGE.

Most of the modern simulation languages are *parallel languages* which means that they enable the user to enter his structural statements in any sequence regardless of the relationships between them. This feature can be obtained by dividing the simulation package into two parts, an *interpretative part* which "understands" the structural statements and sorts them to the required sequence and an *algorithmic part* which performs the numerical integration. A package showing this feature allows even unexperienced simulation users to describe their models in a straightforward manner and to simulate complex processes successfully in a short period of time. The structural statements are to be described in a way such that each simulation parameter appears once and only once at the left side of the equal sign and the sorting algorithm will sort the equations in such a way that all simulation parameters appearing on the right side of any equation have been computed in a statement placed above. The only exception to this rule are memory functions which can be computed without knowledge of the input parameters in advance. If this procedure cannot be carried out successfully, algebraic loops are in the system which require special treatment [6].

For a broad class of problems the above described procedure is optimal. Nevertheless, as soon as any additional logic and especially branching activities are needed for proper description of the model the sorting procedure becomes meaningless, since in such a case the model itself is not entirely parallel. The consequence of this is that the user of a parallel simulation package should, however, be given the possibility to describe parts of his model in a procedural way. He must have the possibility to split up his model into different *sections*, sort-sections in which he can describe parallel structured system behaviour and which are processed by the interpretative package and nosort-sections in which he can describe branching activities and other logical decisions in an *associated procedural language* and which are skipped by the interpretative package.

To preserve full flexibility of a parallel language the user should have the possibility to intervene directly between the interpretative package and the algorithmic package. This is only possible in a comfortable way if the interpretative package does not translate the user supplied parallel code directly into assembly language but into a procedural high-level language (e.g. FORTRAN-IV, ALGOL, PASCAL, PL/I). It is, therefore, advisable to code the interpretative package as a *preprocessor* (e.g. CSMP-III, CSSL-III, DARE-P) and not as a *compiler* (e.g. MIMIC). A second advantage of using a preprocessor is that the probability for a compiler to a *user-oriented* language (e.g. FORTRAN) to work faultless is much higher than for a compiler to a *problem-oriented* language (e.g. MIMIC) since the latter is much less frequently used. Being a versatile user of digital computers the author of this paper lost long ago his believe in the infallibility of compiler programs and , therefore, considers this point to be most important. On the other hand the user has a good chance of being able to correct crimes committed by missbehaving preprocessors. The procedural language into which the preprocessor translates the user supplied parallel code is called *intermediate language*. It is highly recommended to use a package which preprocesses the user supplied code into one of the commonly used procedural languages (e.g. FORTRAN) (intermediate language) using the same procedural language as associated procedural language and allowing full programmability in and full compatibility with this language.

Since simulation is often embedded in another encompassing problem (e.g. parameter identification) the user should have the possibility to split up his program into *segments*. Segmenting here means the facility to call an entire simulation program, making use of all of the features mentioned above, as a subprogram.

V) OPEN ENDED OPERATOR SET AND MODULAR PROGRAMMING. THE ASSOCIATED MACRO LANGUAGE.

Since the sequence of physical models does not correspond to the sequence of statements in a procedural simulation model, it is only possible to describe physical models by simulation modules if these modules are coded as macros [6]. The first activity of the preprocessor must be to replace all calls to macros by the macros' definitions before the sorting algorithm is activated. It is not sufficient to code the module as a subprogram since the entire definition body has to be integrated into the program and not only the address where the module is stored as in the case of a subprogram. Therefore, modular programming can only be performed by use of an *associated macro language*.

VI) THE SYSTEM LIBRARIES.

The simulation package should provide the user with a run-time library consisting of currently used system subprograms (e.g. hysteresis function) and with a symbolic library of system macros (e.g. lead-lag compensator) to simplify the coding of models. These two libraries may be called the third part of the simulation package. The composed package has now the structure given in Fig. 4. The upper part of the figure is problem-oriented (high-level, parallel), the medium part of it is user-oriented (high-level, sequential) and the under part is machine-oriented (low-level, sequential). At the left side the user supplied parts of the simulation program are given, whereas on the right side the system part of the simulation program ($\hat{=}$ the simulation package) is figured. Dashed areas characterize the functions handled by the three parts of the simulation package: the preprocessor, the execution package an the system libraries. The two boxes "compiler" and "link/load" have directly nothing to do with the simulation package.

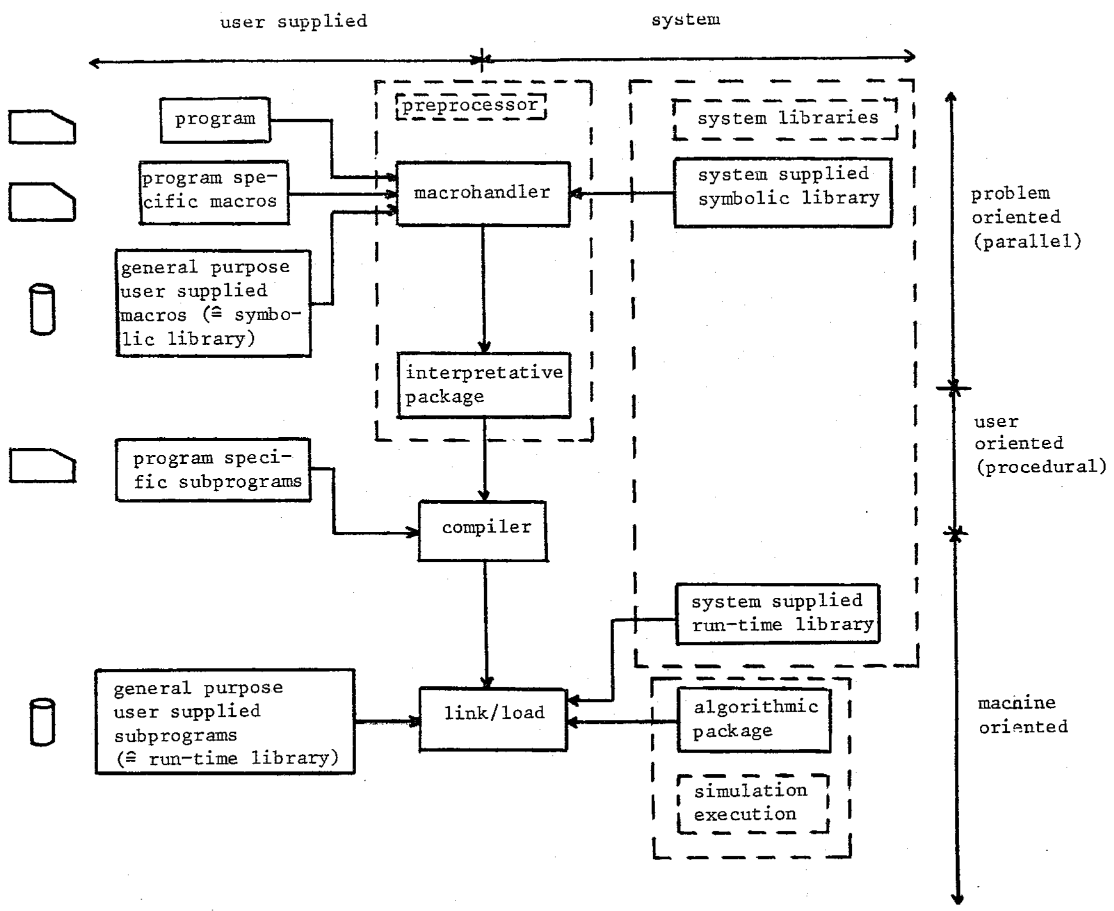
user supplied
system
program
program specific macros
general purpose user supplied macros (≙ symbolic library)
preprocessor
macrohandler
interpretative package
system libraries
system supplied symbolic library
problem oriented (parallel)
user oriented (procedural)
program specific subprograms
compiler
system supplied run-time library
general purpose user supplied subprograms (≙ run-time library)
link/load
algorithmic package
simulation execution
machine oriented

Fig. 4 Diagram showing the architecture of a simulation program

VII) COMPARISON BETWEEN SOME OF THE CURRENTLY USED SIMULATION LANGUAGES.

In the previous chapters the different types of simulation languages have been characterized and their most important attributes have been explained. In the following six of the most commonly used packages are briefly described and compared in a tabular form (Tab. 1).

MIMIC [7,8]

MIMIC is a general purpose parallel simulation language using a compiler which directly translates the user supplied parallel code into machine oriented procedural code. No associated procedural language is available but, however, some additional logic can be coded by use of logical control variables (LCV). Any structural statement can be preceded by a LCV. If this is the case the statement is only executed at times t for which the LCV has the value 'true'. There exist different versions of MIMIC running on different computers. For a long time MIMIC has been the only general purpose parallel simulation language running on CDC-installations and, therefore, has been widely used and accepted although it is neither very flexible nor very convenient. During 1974 the new simulation package DARE-P has been reported which can also be used on CDC-installations. Since DARE-P is in almost every aspect superior to MIMIC, it is expected to replace it in the near future.

CSMP-III [9]

The CSMP-III language together with its predecessor CSMP/S-360 are these days the most frequently used simulation packages. CSMP, like MIMIC, is a general purpose parallel simulation package. It uses a preprocessor which translates the user supplied parallel code into a FORTRAN-IV subroutine (UPDATE). FORTRAN is, therefore, used as intermediate language and at the same time as associated procedural language. The operator set is open ended (associated macro language) and the possibilities for structuring are generous. The CSMP simulation runs may not be called as subprograms (no segmentation), but [6] describes how library routines for the solution of nonlinear programming problems may, nevertheless, be used for optimization problems. The user can choose the integration algorithm out of a big variety of algorithms; however, discrete events may not be handled conveniently by any of them [10]. A very extensive run-time library enables the user to model complex systems easily. The CSMP language is an IBM-product and, therefore, in general restricted to such installations.

CSSL-III [11]

The CSSL-III language is similar to CSMP-III. It also uses a preprocessor which translates the user supplied parallel code into a FORTRAN-IV main program which makes the FORTRAN translation even easier to understand than in the case of CSMP-III. The associated macro language of CSSL-III is an interpretative language and, thus, much more comfortable and general than in the case of CSMP-III whose macro language is more of a macro handler than of a real "language". CSSL-III gives the user the possibility of segmenting his program. So this package is the most sophisticated package of all of them. CSSL-III runs only on CDC-installations.

	languages / features	MIMIC [7,8]	CSMP-III [9]	CSSL-III [11]	DARE-P [12]	GASP-IV[1] [13]	FORSIM-V[1] [14]
	INTEGRATION						
	selection of routines	x[2]	x	x	x		x
	dummy routine		x		x		
	STRUCTURING						
*	sort - option	x	x	x	x		
*	associated procedural language		x	x	x		
	nosort - option		x	x[3]		x	x
*	procedures[4]		x	x	x		
	initial and terminal section	x[5]	x	x	x[6]	x	x
	subprograms	x[7]	x	x	x	x	x
*	associated intermediate language		x	x	x		
	segmentation			x	x	x	x
	MODULARITY						
*	associated macro language		x	x	x[8]		
*	interpretative macro language[9]			x			
	SYSTEM LIBRARIES						
*	symbolic library		x	x			
	run-time library	x	x	x	x	x[10]	x
	INPUT/OUTPUT FACILITIES						
	numerical output	x	x	x	x	x	x
	graphical output	x	x	x	x	x	x
	crossplots	x	x[11]	x	x		
	3-dimensional graphical output		x				
	parameter handling	x	x	x	x	x	x
	ERROR DETECTING AIDS						
	diagnostics	x	x	x	x	x	x
	DEBUG - facility		x	x		x	x
	ADDITIONAL						
	partial differential equations						x
	discrete events					x	

Tab. 1 Comparison of six different simulation packages for the simulation of continuous systems described by ordinary differential equations.

[1] GASP-IV and FORSIM-V are both procedural languages. All features marked by an asterisk (*) in column 1 are not applicable to these languages.

[2] The original MIMIC does not allow any selection of integration routines. A new version developed at the Swiss Federal Institute of Technology Zurich, Switzerland offers 12 different integration routines to the user [8].

[3] Not generally possible, but the sorting region (DERIVATIVE) is embedded into a non-sorting region (DYNAMIC).

[4] A procedure is treated as a single statement of the encompassing parallel region. It will be rearranged as one block whereas the statements forming the procedure maintain their sequence.

[5] By use of logical control variables (LCV).

[6] Initial and terminal section are combined to the so called *logic block* from which the dynamic section (≙ *derivative block*) is called by the CALL RUN - statement.

[7] MIMIC allows -- dependent on the actual version being used -- three to five user supplied FORTRAN-functions (TR1 ÷ TR5) with a maximum of six non-subscribed parameters each. No general compatibility.

[8] A macro handler similar to the one of CSMP-III will be available soon from the Swiss Federal Institute of Technology Zurich, Switzerland. It does not exist in the original version of DARE-P [15].

[9] An interpretative macro language interprets certain commands on the macro level. A macro loop, for example, will generate the included statements as many times as the loop is carried out.

[10] The GASP-IV package has extensive and versatile run-time library routines for discrete simulation whereas the routines for continuous simulation are rather limited.

[11] Only available in combination with a CALCOMP - plotter. Since CSMP-III gives full compatibility with FORTRAN-IV a subroutine to accomplish crossplots may, however, easily be implemented.

CSSL-III has, for all that, not been implemented so far at many installations, since the potential user has to pay a very high amount of money to obtain it, whereas DARE-P is available at a nominal cost. Besides, the interpretative macro language is very expensive in terms of computational time and, therefore, extensive use of this feature is not recommended.

DARE-P [12]

DARE-P is a general purpose parallel simulation language, developed at the University of Arizona, Tucson AR, U.S.A.. DARE-P is entirely written in ANSI-FORTRAN-IV and, therefore, almost system independent. Adjustments, however, have to be made for the linking of the integration algorithms to the program, if other than CDC-6000-series installations are used. DARE-P is not as flexible as CSMP-III or CSSL-III and its documentation is at the actual stage rather poor, but it is, however, in almost every aspect superior to MIMIC and -- since available at a nominal cost -- even comparable to CSSL-III. The author highly recommends this package to users with a CDC-installation.

GASP-IV [13]

GASP-IV is a general purpose procedural simulation package, consisting of a FORTRAN-IV subroutine package. GASP-IV allows simulation of combined continuous and discrete systems. The user has to code the main program and a number of subroutines dependent on the problem to be solved. GASP-IV has been developed at the Purdue University, Lafayette Ind., U.S.A.. This package is unfit for the novice user, because he has to think of many things which are taken care of by parallel simulation languages. On the other hand the package is highly recommended to skilled users since its architecture and documentation are excellent. Concerning the treatment of continuous system simulation the package is somewhat poor since it offers only one integration algorithm and since its run-time library is rather limited. GASP-IV may be obtained from Pritsker Associates Inc., Lafayette Ind., U.S.A. at a nominal price. The package is entirely written in ANSI-FORTRAN-IV and, therefore, almost machine independent. Adjustments may be necessary to obtain an optimal random number generator. Independent on whether the GASP-IV package is available to a potential user of a simulation package or not the author recommends the book [13] to everybody who wants to acquire profound knowledge on simulation techniques.

FORSIM-V [14]

The FORSIM-V package is a general purpose procedural simulation language allowing simulation of continuous systems described by mixed ordinary and partial differential equations. FORSIM-V is a FORTRAN-IV written program to which the user has to add a subroutine (UPDATE) containing the structural statements of his model. Since FORSIM-V is not entirely ANSI-FORTRAN-IV coded, some modifications are required for use on other that CDC-6000-series installations. Various integration algorithms are at the user's disposal for integrating his system over time and various algorithms may be used for derivating his system over space. FORSIM-V has been developed by the Atomic Energy of Canada Ltd., Chalk River, Ontario, Canada and is available from there at a nominal cost.

The author does not claim this survey on existing packages to be complete. There exist other packages like DYNAMO-II [16], a package developed for system dynamics and still used by many economists and biologists, SL-I [17], a package running only on Xerox installations, SLANG [18] and PROSE [19], two packages developed at TRW, Corp.. These packages are surveyed in [20]. The author, however, considers them to be of minor importance partly due to restrictions in their applicability and partly due to restrictions in general availability. Many more packages have been developed at various places which have not been used except for their original installation.

VIII) PROSPECTIVES FOR DEVELOPMENT

First attempts towards digital simulation of continuous systems were reported in the late fifties. Until 1965 over 20 program packages had become available. By then the SCi Simulation Software Committee collected the different idees brought up at the different places. As a result this committee propagated the SCi Continuous System Simulation Language (CSSL). This report has been published in *Simulation*, December 1967 [21]. It contains merely idees from MIMIC -- together with DYNAMO the only two "survivors" of the generation before 1967 --,and from DSL/90 [22] -- the predecessor of CSMP. Since, in 1967 still many engineers faced with simulation problems had not much of experience in utilizing digital computers, CSSL tries by every possible means to help the potential user describe his problem to the computer in a straightforward manner keeping him as far away as possible from problems of numerical mathematics, while providing the skilled user with various possibilities of structuring his model to guarantee high flexibility. Almost all of the languages reported later on (like CSMP-III, CSSL-III or DARE-P) base very much upon the suggestions given by the SCi Simulation Software Committee in the report mentioned above.

On the other hand the situation changed quite a bit since 1967 in the following ways.

1) More and more of young engineers and even a remarkable number of biologists, economists etc. become very well acquainted with digital computation by the time they leave university. For these users parallel languages often are more of a hindrance than of a help. For such users the optimal solution is coding their problems in a procedural language like FORTRAN-IV by using a package of system provided routines for integration of state equations, for easy handling of Input/Output, for file handling etc. For this reason some of the brand new simulation languages like GASP-IV (reported: November 1973) or FORSIM-V (reported: November 1974) are procedural packages.

   According to the authors opinion an optimal package of this class of simulation packages, however, does not exist yet. A few suggestions toward this goal include the use of GASP-IV as a basis for the development of a more optimal package while

   a) modifying the subroutine GASP in such a way that the user can choose integration and iteration algorithm according to his specific problem

   b) adding the subroutines PARSET and PARFIN and some secondary subroutines of the FORSIM-V package for handling of partial derivatives (which requires reorganization of the common blocks of the GASP-IV package) and

   c) adding a good run-time library like the one of CSMP for simplification of describing complex models.

   The author feels that such a package would be welcomed by a large number of simulation users.

2) There has been a remarkable development of computer techniques since 1967. For this reason the user of simulation techniques today can be offered much higher comfort than the SCi Software Committee could propagate at that time. None of the packages charac-

terized above is, for example, really suited for computer aided design. Procedural packages may be used together with an optimization package for parameter identification problems. Parallel packages, normally, should not be used for this purpose, even if they provide the user with the possibility for segmenting, since single simulation runs for complex models in most cases cost more than sfr. 10.-- which makes optimization illusory. They may be used for the layout of systems if there exists a limited number of possibilities for realization of a system out of which the best has to be chosen.

A program package suited for computer aided design must have the following two attributes.

a) The package must be *interactive* to provide the user with the possibility of changing the model structure dynamically to obtain an optimal solution easily and in a short period of time.

b) The package must run on a *process computer* to keep the costs of the simulation in reasonable limits.

Significant efforts toward this goal have been made at the University of Arizona with the package DARE-I [23], an interactive simulation package with graphical display running on a PDP-9 process computer. An overview over the different packages of the DARE family is given in [24]. Efforts in this direction are also being made at the Swiss Federal Institute of Technology Zurich, Switzerland where an interactive simulation package is under development for utilization on PDP-11 process computers. This package is written in the macro language of the PDP-11 which guarantees expediency in execution. Many efforts, however, are still needed before satisfactory results can be obtained to the solution of this problem.

IX) FINAL REMARKS

This paper has surveyed methods and program packages for the simulation of continuous systems described by ordinary differential equations. Section 8 dealt with the author's opinion of desirable developments of the field during the near future. It is hoped that some of the suggestions given there would stimulate simulation experts to impel the field's development. In that case this paper would be of use to the novice user of simulation techniques as well as to the specialist.

X) REFERENCES

[1] M.H.Hamza: Introduction to Analog Computation. *Proc. SIMULATION'75*

[2] A.Schöne: Simulation technischer Systeme, Bd. 1 *Carl Hanser Verlag* München 1974

[3] J.Kohlas: Random Number Generation. *Proc. SIMULATION'75*

[4] J.P.C.Kleijnen: Statistical Design and Analysis of Simulation Experiments. *Proc. SIMULATION'75*

[5] C.W.Gear: Numerical Initial Value Problems in Ordinary Differential Equations. *Prentice Hall Series in Automatic Computation* 1971.

[6] F.E.Cellier, B.A.Ferroni: Modular, Digital Simulation of Electro/Hydraulic Drives using CSMP. *Proc. 1974, SCSC (Summer Computer Simulation Conference)* AFIPS Press 1974

[7] MIMIC Digital Simulation Language. Ref. Manual *Control Data, Corp.* Sunnyvale California 1968

[8] J.Halin: MIMIC-Manual des Institutes für Reaktortechnik der Eidgenössischen Technischen Hochschule Zürich. *Swiss Federal Institute of Technology.Zurich* Zürich, 1973

[9] IBM Continuous System Modeling Program III (CSMP-III) Program Reference Manual, Form SH19-7001-0 *IBM* White Plains, New York 1971

[10] F.E.Cellier, D.F.Rufer: Algorithm suited for the Solution of Initial Value Problems Occuring in Engineering. *Proc. SIMULATION'75*

[11] CDC Continuous System Simulation Language III (CSSL-III) User's Guide, Form 17304400 Rev. A *Control Data, Corp.* Sunnyvale, California 1971

[12] J.J.Lucas, J.V.Wait: DARE-P User's Manual *CSRL Report 255* University of Arizona, College of Engineering, Dept. of Electrical Engineering, Computer Science Research Laboratory, Tucson, AR. U.S.A. 1974

[13] A.A.B.Pritsker: The GASP-IV Simulation Language. *John Wiley* New York, London, Sidney, Toronto 1974

[14] M.B.Carver: FORSIM A FORTRAN Package for the Automated Solution of Coupled Partial and/or Ordinary Differential Equation Systems. User's Manual. *Atomic Energy of Canada, Ltd.* Chalk River Nuclear Laboratories, Chalk River, Ontario, Canada 1974

[15] A.Blitz, F.E.Cellier: Modular Simulation by Use of DARE-P. To be published.

[16] A.L.Pugh: DYNAMO-II. User's Manual *M.I.T. Press* Cambridge Mass. U.S.A. 1970

[17] SL-I Reference Manual *Xerox Data Systems* El Secundo, California 1970

[18] J.M.Thames: SLANG a Problem Solving Language for Continuous-Model Simulation and Optimization. *Proc. of the 24th ACM National Conference* San Francisco 1969

[19] J.M.Thames: PROSE a Problem Level Programming System. *Solveware Associates* San Pedro, California 1973

[20] R.N.Nilsen, W.J.Karplus: Continuous System Simulation Languages A State-of-the-Art Survey *Annales de l'Association Internationale pour le Calcul Analogique, N° 1* 1974 (January)

[21] The SCi Continuous System Simulation Language (CSSL) *Simulation Vol. 9 No. 6* 1967 (December)

[22] W.M.Syn, R.N.Linebarger: DSL/90 A Digital Simulation Program for Continuous System Modeling *AFIPS Conference Proceedings vol 28 1966 SJCC*

[23] G.A.Korn: Project DARE Differential-Analyzer Replacement by On-Line Digital Simulation *Proc. AFIPS/FJCC 1969* AFIPS Press, Montvale, New Jersey 1969

[24] G.A.Korn: New Techniques for Continuous-System Simulation *Automatic Control Theory and Applications* Acta Press, Calgary, Canada vol. 2 No. 1 1974 (January)