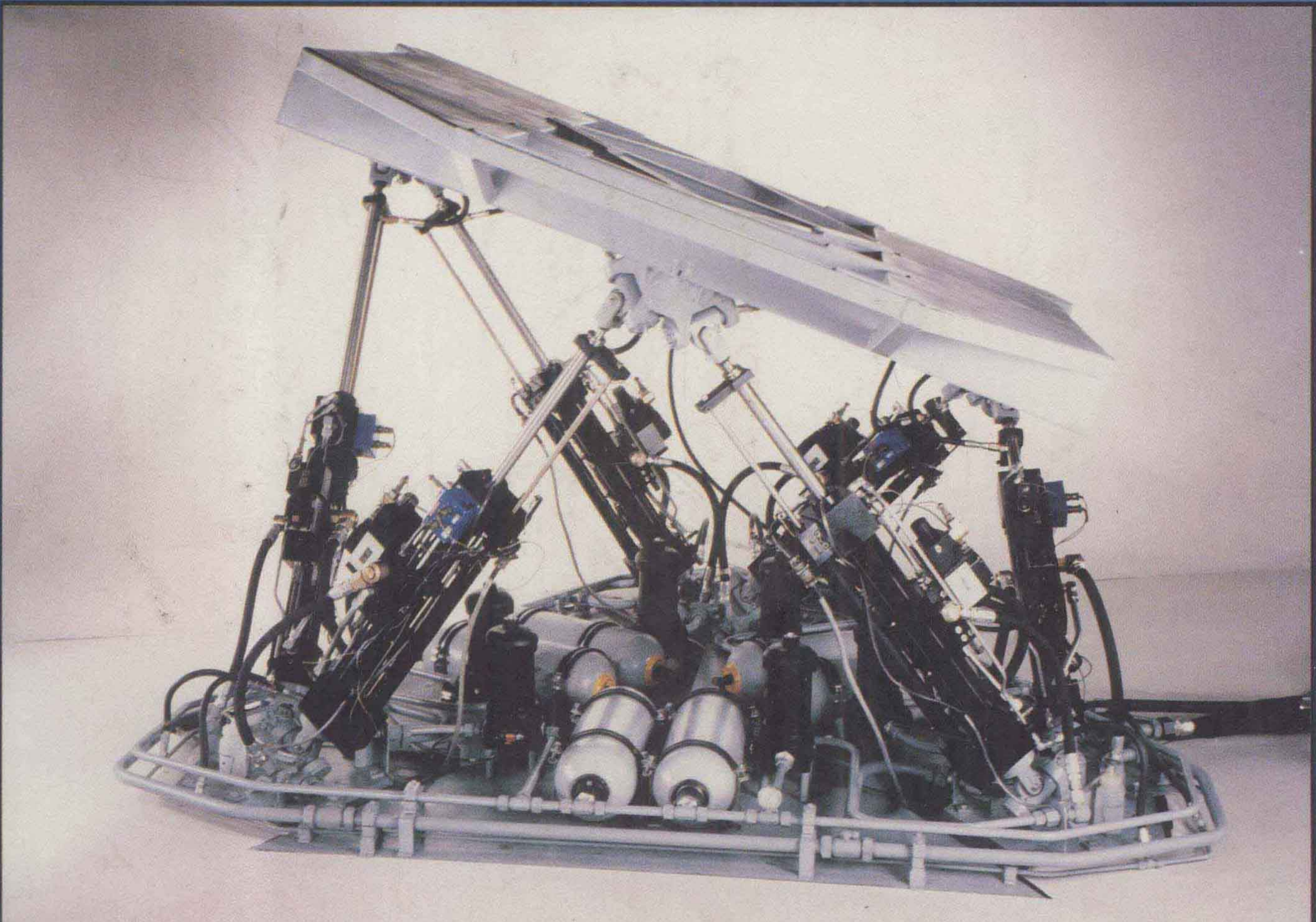
MODELING AND SIMULATION ON MICROCOMPUTERS, 1988

Edited by

Joe Hilber



THE SOCIETY FOR COMPUTER SIMULATION INTERNATIONAL

MODELING AND SIMULATION ON MICROCOMPUTERS, 1988

Proceedings of the SCS Multiconference on
Modeling and Simulation on Microcomputers
3-5 February 1988
San Diego, California

Edited by
Joe Hilber
AT&T

SCS
INTERNATIONAL

A Society for Computer Simulation International (Simulation Councils, Inc.) publication
San Diego, California

Rosemary A. Whiteside, Managing Editor

ISBN 0-911801-26-x

PRINTED IN THE UNITED STATES OF AMERICA

MODELING AND SIMULATION ON MICROCOMPUTERS, 1988

Preface

In preparing for the seventh annual Modeling and Simulation on Microcomputers Conference, I sense a changing attitude about Personal Computers and simulation. In the early years of the microcomputer revolution it seemed remarkable that significant simulation work could be undertaken with the computing power on ones desk. In the ensuing years our Proceedings have documented the rapidly evolving PC computing power, both hardware and software. Most of us now have on our desk more computing power than most university computer departments of a generation ago!

THE UBIQUITOUS MICRO

Many authors now classify their work first by the process being modeled (e.g., Transportation, Space Systems, Biomedical), the technique used (e.g., AI, Discrete, Continuous), or the application being made (e.g., Training, Robotics). and only incidentally that it was done on a microcomputer. The microcomputer has truly become ubiquitous.

Another indicator of this reality is that many your professionals, technicians, and managers entering the work force from our colleges and universities view and use microcomputers as a natural tool in their work. A growing library of "user-friendly" software make the generation of models an activity of accountants, stock brokers, marketeers, and engineers rather than solely simulation specialists.

The annual "Catalog of Simulation Software" published in the October issues of *SIMULATION* lists over 150 software packages. Almost two thirds (63%) of these are implementable on commonly available microcomputer hardware and operating systems.

WHERE THE MICRO IS THE ANSWER

On the other hand there are applications where simulation on microcomputers will continue to be remarkably well suited, potentially large markets exist and the expertise of simulation specialists is needed. In applications where portability is a necessity, low cost is a requirement, small size dictated, or speed of implementation mandatory, the microcomputer is the answer. Perhaps the future of this conference will be to highlight those areas of simulation and modeling for which the microcomputer is uniquely suited.

NETWORKING MICROS AND MAINLINE

An exciting aspect of current developments in microcomputers is the advent of personal workstations and the software to link them to massive sources of public and proprietary data. While this networking of Personal and Mainline computer power has been widely discussed, demonstrated, and even implemented in large businesses, it is only recently that IBM, AT&T, Apple, and others have begun to provide hardware and software products at prices that make them affordable for small businesses, professionals, families, and schools.

1988 MODELING AND SIMULATION ON MICROCOMPUTERS

This year's papers demonstrate that microcomputers are being applied to an exciting range of simulations. The low cost of the personal computers makes computer simulation a tool for students and individuals that normally do not have access to mainline computing power. Combining portability, low cost and the ability of network micros with the institution's main processing center has captured the best of both worlds. Animation and graphics on microcomputers continue to intrique us and three papers on the problems and opportunities in this area are included in this year's Proceedings.

THE FUTURE

One effect of reading, presenting, and discussing technical papers such as those published here is to stimulate you to become active and creative in the use of microcomputers for simulation. I urge you to share your progress at future conferences. In particular, research on and applications of modeling and simulation that make use of the special characteristics of microcomputers (portability, low cost, etc.) will be of great interest to future conferences.

Joseph E. Hilber, Editor
AT&T

Simulation Languages and Tools

CONTENTS

CONTENTS (continued)

Simulation Languages and Tools

ISBN 0-911801-26-x

PASION: The language and its environment

S. RACZYNSKI
ESCUELA DE INGENIERIA
UNIVERSIDAD PANAMERICANA
AUGUSTO RODIN 498, 03910 MEXICO,D.F.

ABSTRACT

PASION is a PASCAL-related simulation language with a process/event structure. A PASION program consists of a sequence of process declarations. At run-time the program generates objects that represent processes of the simulated system. PASION offers some features of the object-oriented languages such as inheritance and generic processes. The language environment supports graphics, interactive simulation, queue statistics and other additional simulation tools.

INTRODUCTION

The language PASION (This stands for Pascal Simulation) was created rather with didactic aims, for the simulationists who already know and use Pascal. The language is an extension of Pascal characterized by the following features.

1. Almost all Pascal structures can be used in the source program. Arithmetics, predefined and user-defined types, input/output, variables definitions, record and file handling, functions and procedures are the same as in Pascal.

2. The extension consists in relating the program to model descriptive variables (including the state variables), and the model time. This is done by - specifying the simulation model as a set of objects of "process" types, and events. The language supports the simulation by creating objects, controlling their activities and the model time through the corresponding event scheduling mechanism.

An introduction to PASION can be found [1]. Let us note that PASION is a simulation language and not a package. Consequently, some of the features which offer typical simulation packages (graphics, interactive simulation etc.), don't belong to the language but rather to its environment, i.e. the library of predefined - processes and procedures.

PROCESS/EVENT STRUCTURE OF THE MODEL

PASION suggests certain structure to be imposed on the simulation model. First of all, we have to describe the model as a set of components, descriptive variables and interaction rules. This approach, described excellently by Zeigler [2], provides certain model structure which can be coded in PASION easily. The components of the model are dynamic objects created in the operational memory due to the corresponding process declarations. The descriptive variables of the components are declared as process attributes, which can store the state of the process. The model activities (interaction rules) are described as events inside process declarations.

The general program structure is as follows:

```
PROGRAM ident1;
   global declarations
......
PROCESS ident2;
   ATR attribute specifications
......
EVENT ident3;
   event body ... ENDEV;
......
EVENT ident4;
   event body ... ENDEV;
......
PROCESSS ident5;
   ATR....
      ....
      ....
START
     main program
$
```

Where ident 1,2,... are identifiers. Each PROCESS declaration defines a process type with its attributes and events. At the run time, objects are created due to the PROCESS declarations and run in the quasi-parallel mode, executing their events. The simulation is initiated in the main program.

Let us compare PASION with two well known simula--

lation tools, namely GPSS and SIMULA. The events of PASION are similar to the blocks of GPSS, while the processes correspond to the GPSS transactions. The difference lies in the fact that the PASION events are sequences of operations coded by the programmer and the GPSS blocks are provided by the package. Compared with Simula, PASION is rather simple, although the - Process Class of Simula is similar to the Process declaration of PASION. The difference is that SIMULA - activates and desactivates processes and PASION controls the events. In other words, -- any PASION object (of a process type) can be activated, suspended, reactivated, etc., by activating its events. An event, once invoked, can not be interrupted. As for the strategy of event execution, it is an event scheduling algorithm, with some elements of process interaction and the "three-phase approach" (see [3], [4],[5]).

PASION has some of the properties of object-oriented languages, though it is not strictly object-oriented. Let us recall that a language to be object-oriented must support information hiding, data abstraction, dynamic binding and inheritance. Only two of these features are implemented in PASION, namely information hiding and inheritance. Information hiding consists in the fact that the attributes of an object are not directly visible from outside of the object. The inheritance refers to process declarations [6] and permits to agregate new properties (new attributes and events) to existing processes. Thus, certain processes share the same code, prepared earlier. Let us note that most of the recently used simulation packages and languages lack this feature which undoubtedly will be indispensable in the future simulation software.

PROCESS HIERARCHY AND INHERITANCE

While constructing complex models it is convenient to add new properties to existing processes without reediting the whole program. This feature is well - known as class hierarchy (e.g. in SIMULA) or inheritance in object-oriented languages [6], [7], [8]. It should be noted that most of the commonly used simulation languages lack this facility. The inheritance in PASION can be applied by using prefixed process declarations. For example, if PA is the name of an - existing process and we wish to create a new one, say PB, having all the properties of the process PA (this means all its attributes and events), we simply use the name PA/PB in the heading of the declaracion of PB. The complete heading may be, for example

PROCESS PA/PB,25;

where 25 is the maximal number of objects of type PB which can exist simultaneously. The translator looks for the "parent" process PA (it can reside in the same source file, or in a separate library file) and inserts all the attributes and events of PA to the declarations of PB.

In [1] an example is given, being a model of two populations of bacteria (of type BACTA and BACTB). - The BACTB bacteria can move, divide, die and eat -- those of type BACTB. Using inheritance, the declaration of BACTA can be prefixed with BACTB, supposing that BACTB was declared earlier. Thus, BACTA inherits from BACTB the ability to move, divide and die and the declaration of BACTB includes only one event "eat". This makes the program much concise.

Inheritance is one of the features of PASION which make it possible to develop application-oriented libraries of "parent" processes.

PREDEFINED PROCESSES

There are few programming languages which can be used without an appropiate support environment. Actual version of PASION is equipped with a minimal - PASION Programming Environment (MPPE) which consists of a library of predefined processes and auxiliary PASCAL procedures. It supports such features as interactive simulation mode, graphics, queue statistics etc. The core of MPPE is the library of predefined -- processes. These are generic program units which - generate PASION processes.

Predefined processes are written in PASION extended by a simple "meta-language" which permits a process to have formal parameters. the user invokes a predefined process by its name and specifies the actual parameters which are passed by name, before the program is translated to PASCAL. The user can prepare his own predefined processes and add them to the library. A formal parameter by be a constant, a variable name, a type, a procedure or process name etc., even a PASCAL reserved word. The call has the following - syntax

$\longrightarrow$ PROCESS $\longrightarrow$ pname $\longrightarrow$ LIKE $\longrightarrow$ ppname $\longrightarrow$
$\longrightarrow$ ($\rightarrow$ actual parameters $\rightarrow$), $\longrightarrow$ pnumber $\longrightarrow$;

The reserved word LIKE identifies the call, "ppname" must be the name of an existing predefined process. This process is read from the library, preprocessed, and inserted into the resulting code as a new process called "pname". The "pnumber" is the maximal number of objects of type "pname" which can be created. -- Consider for example, the following call:

```
PROCESS X1 LIKE INTERP(A,B,,,,
         C,D,,, 0.1),1;
```

The process INTERP is read from the MPPE and the corresponding declaration is generated in the source program. The generated process is named X1, and the first, the second, the 5th, the 6-th and the 10-th - formal parameters of INTERP are replaced by "A", "B", "C", "D" and "0.1" respectively.

The process INTERP is one of the processes of MPPE. It controls the interactive simulation mode and generates graphic output. The first five parameters are the names of variables to be plotted simultanously - with the simulation. The next 5 parameters are the names of variables which can be redefined at the run time. The user can interrupt the simulation, change the values of some of these variables and reactivate the program. Fig. 1 shows a typical screen generated by INTERP. The screen is divided into three windows. The graphic window is marked with I. Here the plots of the variables appear. Window II is reserved for the changes. It is active when the user interrupts the simulation and specifies the changes. Window III is the user window. Here appears any output specified by the user in the source program.

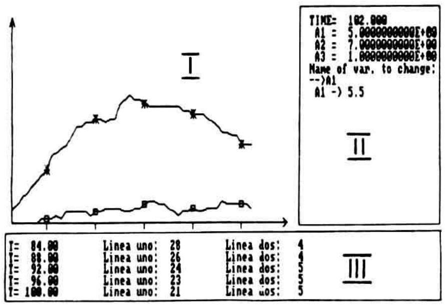
CONCLUSIONS

PASION system, considered as the language, its -- translator and its environment provides all the basic simulation tools. The system is open and can be extended using both parent and predefined processes. - PASION is completely hardware-independent, because all the translator does is preprocessing.

The resulting PASCAL code can be compiled on any system. Actually it is implemented on the IBM PC.

REFERENCES

1. RACZYNSKI, S.; "PASION - Pascal-related simulation language for small systems", SIMULATION 46(6), 1986.
2. Zeigler, B.P.; "Theory of modelling and simulation" John Wiley & Sons, 1976.
3. Hooper, J.W.; "Strategy-related characteristics of discrete-event languages and models", SIMULATION 46(4), 1986.
4. Hooper, J.W.; "Activity scanning and the three-phase approach", SIMULATION 47(5), 1986.
5. O'Keefe, R.M.; "The three-phase approach: A comment on "strategy-related characteristics of discrete-event languages and models", SIMULATION 47(5), 1986.
6. Pascoe, G.A.; "Elements of object-oriented programming", BYTE 11(8), 1986.
7. Kaehler, T. and Patterson, D.; "A taste of - Smalltalk", W.W. Norton & Co., New York, 1986.
8. Schmucker, K, Y.; "Object-oriented Languages for the Macintosh", BYTE 11(8), 1986.



```
TIME:  102.000
 A1 =  5.0000000000E+00
 A2 =  7.0000000000E+00
 A3 =  1.0000000000E+00
Name of var. to change:
-->A1
 A1 -> 5.5
```

```
T=  84.00      Linea uno:  28      Linea dos:  4
T=  88.00      Linea uno:  26      Linea dos:  4
T=  92.00      Linea uno:  24      Linea dos:  5
T=  96.00      Linea uno:  23      Linea dos:  5
T= 100.00      Linea uno:  21      Linea dos:  5
```

Modeling and Simulation on
Microcomputers 1988
©1988 by The Society for Computer
Simulation
ISBN 0-911801-26-x

LYSIS: An interactive software system for non-linear modeling and simulation

V.Z. Marmarelis and N. Herman
Biomedical Simulations Resource
University of Southern California
Los Angeles, California 90089-1451

ABSTRACT

To facilitate the broader use of "black box" modeling and simulation methods among biomedical investigators, an interactive software package--named LYSIS--is being developed by the Biomedical Simulations Resource at the University of Southern California. LYSIS allows the processing of time-series data to obtain linear and nonlinear dynamic "black box" models of physiological systems. It also allows the simulation of these models in computer studies of physiological function. The nonlinear modeling and simulation methods are placed in the framework of Wiener's theory for nonlinear systems.

INTRODUCTION

There has been a rising trend in recognizing the importance of mathematical modeling and computer simulation of physiological function in the last twenty years. Mathematical models of physiological system organs and their components have been gradually accepted as the best way of summarizing our quantitative knowledge about physiological function, acquired through experimental and clinical observations. Computer simulation of these models allows the study of functional characteristics of the corresponding physiological systems without the burdens or the costs of actual experimentation and prolonged observation. The success of this scientific/engineering endeavor hinges critically upon the validity and effectiveness of the employed modeling and simulation methodologies. The results to date have vindicated the vision of the pioneering systems physiologists as to the efficacy of this approach in advancing basic scientific knowledge and improving clinical practice. A lot remains to be done however, and the prospects of this ambitious undertaking are as exciting as the challenges are formidable.

The problem of physiological system modeling is often placed in a "black box" context, whereby the complexity of the system internal workings prevents the development of realistic mathematical models (typically in the form of differential equations) from physical or chemical principles. In this context, we seek to obtain "empirical" mathematical models, on the basis of experimental input-output data, that represent the input-output causal relationship with sufficient accuracy. It is hoped that these empirical models will reveal systemic functional properties of importance in understanding the underpinnings of physiological function, and allow the effective analysis of physiological system function under a variety of experimental or natural conditions. This task is further complicated by the presence of noise contaminating the experimental data, that calls for modeling methodologies effective in a stochastic framework. Most of the modeling efforts in this area have been limited thus far to the linear case, when system dynamics are involved. However, nonlinearities abound in physiology (especially in neurophysiology) and effective methodologies for nonlinear modeling and simulation have been slow to develop and be adopted by physiologists, owing to the complexity associated with the study of nonlinear dynamics. The occasional presence of nonstationarities is an additional complicating factor that deserves proper attention if meaningful results are to be obtained. Finally, limitations posed by experimental conditions usually make it imperative that time-efficient procedures be used in collecting the necessary experimental data.

To address this problem, methodologies based on Wiener's theory of nonlinear systems (Wiener, 1958) have been gradually adopted by pioneering systems physiologists, that offer promise in tackling a class of these formidable problems. Vision physiologists were the first to attempt this approach and other neurophysiologists (primarily in the auditory, vestibular and neuromuscular area) have been gradually attracted to this method, due to the nonlinear "black box" nature of their systems. In the process, much has been learned about the intricacies of this approach and refined variants have been developed--for a review of the basic methodology and its initial applications to physiological systems see Marmarelis and Marmarelis (1978). Applications of this method have been increasing in recent years. Some recent developments are included in a volume edited by Marmarelis (1987).

In spite of the increasing activity around this approach, its broader use has been impeded by lack of readily available software that allows the employment of the method by the non-expert. To address this need and facilitate the broader use of this method, we, at the Biomedical Simulations Resource at USC, have developed a user-friendly interactive software package named LYSIS, that allows Wiener nonlinear analysis of experimental data and related simulations studies. In the following section we summarize the computational requirements of this approach, and in the next section, we outline the basic structure and capabilities of LYSIS.

NONLINEAR WIENER MODELING

Wiener's theory is based on functional expansions when Gaussian white noise (GWN) is used as a test input. Wiener's critical contribution is in suggesting a GWN is an effective test input for identifying nonlinear systems of a very broad class and proposing relatively simple mathematical procedures for the estimation of the unknown system descriptors (kernels) from input-output data.

The input-output relation of a causal system can be seen as a mapping of the input past (and present) values onto the present value of the output. In this sense, the use of a mathematical functional (denoting this mapping) is appropriate:

$$y(t) = F[x(t'), t' \leq t] \tag{1}$$

where x is the input, y the output and F the functional representing the black-box system. The modeling task thus becomes one of obtaining an explicit mathematical description of the system functional F.

In the case of linear time-invariant systems, this functional is represented by the convolution integral:

$$y(t) = \int_0^\infty h(\tau)x(t-\tau)d\tau \tag{2}$$

where $h(\tau)$ - the impulse response function or kernel - is the sole descriptor of the system. The system identification task reduces, in this case, to obtaining an estimate of $h(\tau)$ - or its Fourier transform H(jw) - from input-output data. In the case of nonlinear time-invariant systems, a functional expansion of the system functional F can be considered. If the system functional F is analytic, then a Volterra series expansion exists of the form:

$$y(t) = \sum_{n=0}^{\infty} \int_0^\infty \dots \int_0^\infty k_n(\tau_1, \dots, \tau_n)x(t-\tau_1)\dots x(t-\tau_n)d\tau_1 \dots d\tau_n \tag{3}$$

provided the series converges.

The multiple convolution integrals of the Volterra series involve high-order kernel functions $\{k_n(\tau_1,\dots,\tau_n)\}$ which constitute the descriptors of the system nonlinear dynamics. Consequently, the system identification task is to obtain estimates of these kernels from input-output data. These kernel functions are symmetric with respect to their arguments, i.e., attain the same value for any permutation of given $(\tau_1, \dots, \tau_n)$ values. In general, the Volterra kernels of a system cannot be directly determined from input-output data unless the Volterra expansion is of finite order.

The inability to estimate the Volterra kernels in the general case of an infinite series has prompted the use of GWN test inputs and the Wiener approach to kernel estimation. Wiener suggested the orthogonalization of the Volterra series when a GWN test input is used. The functional terms of the Wiener series are constructed on the basis of a Gramm-Schmidt orthogonalization procedure requiring that the covariance between any two Wiener functionals be zero. The resulting Wiener series expansion takes the form:

$$y(t) = \sum_{n=0}^{\infty} \sum_{m=0}^{[n/2]} \frac{(-1)^m n! P^m}{(n-2m)!m!2^m} \int_0^\infty \dots \int_0^\infty h_n(\tau_1, \dots, \tau_{n-2m}, \lambda_1, \lambda_1, \dots, \lambda_m, \lambda_m)$$
$$x(t-\tau_1)\dots x(t-\tau_{n-2m})d\tau_1 \dots d\tau_{n-2m} d\lambda_1 \dots d\lambda_m \tag{4}$$

where [n/2] is the integer part of n/2 and P is the power level of the GWN input. The set of Wiener kernels $\{h_n\}$ is, in general, different from the set of Volterra kernels $\{k_n\}$. Specific relations however exist between the two sets of kernels (Marmarelis and Marmarelis, 1978).

The orthogonality of the Wiener series allows the estimation of Wiener kernels from input-output data in the general case. Lee and Schetzen (1965) proposed a simple technique for the estimation of the Wiener kernels of a system based on high-order input-output crosscorrelations:

$$h_m(\tau_1, \dots, \tau_m) = \frac{1}{m!P^m} E[y(t)x(t-\tau_1)\dots x(t-\tau_m)] \quad (for\ \tau_i \neq \tau j) \tag{5}$$

For the evaluation of the kernel at the diagonal points the m-th order response residual must be used in the crosscorrelation. The simplicity of the crosscorrelation technique led to its adoption by many investigators in modeling studies of nonlinear systems in the area of physiological systems.

A variety of important practical issues had to be explored in actual applications of the crosscorrelation technique. To name but a few: the generation of appropriate quasi-white test signals (since ideal GWN is not physically realizable); the choice of input bandwidth; the accuracy of the obtained kernel estimates as a function of input bandwidth and record length; the effect of extraneous noise and experimental imperfections, etc. An extensive study of these practical considerations can be found in Marmarelis & Marmarelis (1978).

The greatest obstacle in the broader use of the crosscorrelation technique has been the heavy computational burden associated with the estimation of high-order kernels. Obviously, the amount of required computations increases geometrically with the order of estimated kernel. This prevents the estimation of kernels above a certain order, depending of course on one's computing facilities. Typically, kernel estimation has been limited to second order, with a few attempts for third-order kernel estimation but not further. An additional practical limitation is imposed by the fact that kernel functions of more than three dimensions are difficult to inspect or interpret meaningfully. As a result, successful application of the crosscorrelation technique has been limited to weakly nonlinear systems (second or third order) to date. In an effort to reduce the computational burden, non-Gaussian (binary, ternary, etc.) quasi-white random inputs and inputs based on pseudorandom m-sequences have been used as well as deterministic inputs (e.g. sums of sinusoids of incommensurate frequencies), all of them leading to some reduction of computational effort but failing to provide the breakthrough required for the practical identification of strongly nonlinear systems.

In closing this section, we note that the Wiener approach has been extended to the case of nonlinear systems with multiple inputs and multiple outputs. This extension has led to a generalization for modeling of nonlinear systems with spatio-temporal inputs that has found interesting applications to the visual system. Finally, an extension of the Wiener approach to systems with spike inputs and/or outputs, encountered in neurophysiology, has been made, where the GWN test input is replaced by a Poisson process of impulses.

LYSIS

As mentioned in the Introduction, one practical obstacle in the broader use of the Wiener approach has been the lack of readily available and easy to use software for this purpose. The development of LYSIS aims at redressing this problem.

LYSIS (the Greek word for "solution") is an interactive software package that can be used for linear and nonlinear modeling, simulation and time-series analysis. Version I of LYSIS is currently available and incorporates some modeling and simulation methodologies for nonlinear "black box" systems appropriate for studies of physiological function.

LYSIS is written in FORTRAN-77 and runs on the VAX class of computers with VMS operating system. It is structured in modular, individually-executable interactive programs performing specific high-level tasks. Particular attention was paid to making the dialogue simple and efficient, while achieving the maximum range of executable tasks. Use of LYSIS does not require knowledge of computer programming but it does require understanding of the basic principles and theories of signal processing and system modeling.

The datasets in LYSIS are one-dimensional or two-dimensional evenly-sampled data arrays with fixed format. Data analysis can be done either in the time or the frequency domains. The results can be displayed in two or three dimensions, through flexible interactive graphics programs. Synthesis results (e.g., model predictions) can be obtained by use of the appropriate programs. Computer generated (using LYSIS programs) and experimental data (externally provided) can be used for analysis. In the latter case the experimental datasets must be converted to LYSIS format.

The first version of LYSIS is comprised of 16 programs described below. These form the basic set that can be used for the most fundamental tasks in linear and nonlinear time-series analysis and system modeling and simulation. It is also hoped that they will provide a common computational framework for investigators in this area of research, facilitating communication and interaction. As mentioned above, Version I is only the beginning in the LYSIS development and many programs must be added before we have a reasonably complete package.

LYSIS is developed by the Biomedical Simulations Resource at the University of Southern California under Grant RR01861 from the Biomedical Research Technology Program of the Division of Research Resources of the National Institutes of Health. The programs of Version I are written by Ms. Nava Herman (except for PLOT and PLOT3D written by Ms. D. Kennedy and Mr. A. Weitzenfeld, respectively) under the supervision of the Resource Director, Prof. V.Z. Marmarelis, and Associate Director, Prof. D.Z. D'Argenio. The graphics library PGPLOT was made available to us by Dr. T.J. Pearson of the California Institute of Technology, whose contribution we wish to acknowledge.

The modular (individually executable) programs included in Version I of LYSIS are:

1. GENER: generates one-dimensional datasets specified by the user; also, it operates on existing one-dimensional datasets.
2. GENER2: generates two-dimensional datasets specified by the user; also, it operates on existing two-dimensional datasets.
3. PLOT: displays one-dimensional datasets; the user can display up to six datasets simultaniously, change the range of the axes and specify labels.
4. PLOT3D: displays two-dimensional datasets; the user can change the viewing point, the range of axes and specify labels.
5. DSSTAT: computes basic statistics and the amplitude histogram of one-dimensional datasets specified by the user.
6. CORREL: computes auto- and crosscorrelation of one-dimensional datasets specified by the user.
7. CONVOL: computes the convolution of one-dimensional datasets specified by the user.
8. FFT: computes the Fast Fourier Transform of one-dimensional dataset specified by the user.
9. IFFT: computes the inverse Fast Fourier Transform of one-dimensional dataset specified by the user.
10. SPECT: computes the spectrum of one-dimensional dataset specified by the user.
11. KERNEL: computes the zero, first and second order Wiener kernels from (white-noise) input-output data.
12. MODRES: computes the model response of first and second order Wiener models (specified by the kernels).
13. NARMAX: simulates nonlinear difference equations involving three variables and displays the result.
14. ARMA: simulates linear difference equations involving two variables, and displays the results.
15. PAREST: computes parameter estimates of specified ARMA models for given input-output data (using ordinary least squares).

16. TRANS: transcribes non-LYSIS datasets into LYSIS dataset format.

EXAMPLE

Consider a second-order Wiener system with the kernels shown in Fig. 1ab. A GWN input x of 8000 datapoints is used to test the system producing the output y. The output signal in this case is generated using LYSIS program MODRES. Segments of the input-output data are shown in Fig. 2. The crosscorrelation technique is applied on these data (using LYSIS program KERNEL) to obtain estimates of the first and second order Wiener kernels shown in Fig. 3ab. The estimation variance of these kernel estimates (evident especially is the second order) is due to the random nature of the input and the finite record length used. This estimation variance can be reduced by increasing the input-output record length and/or optimizing the input bandwidth (Marmarelis & Marmarelis, 1978). The 2-D and 3-D graphics are done with LYSIS programs PLOT and PLOT3D, respectively.

An example of LYSIS programs used in the computation of the system output and estimates of the Wiener kernels (of first and second order) is given below.

* MODRES *

This program computes the response to a given input (stimulus) of a second-order Wiener model specified by given kernels. The second-order kernel is optional and leaving blank the file name of the second-order kernel will result in the computation only of the first-order model response. Likewise, the model response based only on the second-order term can be computed by leaving blank the file name of the first-order kernel.

An example session with the program follows. The user's input is in lower case letters, and the program's output is in upper case letters.

```
$run modres
NAME OF MODEL RESPONSE:
y
SPECIFY THE INPUT SIGNAL:
x
SPECIFY THE 1ST-ORDER KERNEL DATASET:
h1
SPECIFY THE 2ND-ORDER KERNEL DATASET:
h2
EVALUATION COMPLETED

FORTRAN STOP
$
```

* KERNEL *

This program computes first and the second order Wiener kernels from given input-output data over a specified range of lags (with minimum and maximum lag values determined by the user). In order to obtain meaningful results (kernels), the input data <u>must</u> be a white sequence. If all you want is the first-order kernel, you should leave blank the file name of the second-order kernel. Likewise, if you only wish to compute the second-order kernel, you should leave blank the file name of the first-order kernel. The program will display the value of the zero-order kernel (which is the mean of the output data).

```
$run kernel
NAME OF 1ST-ORDER KERNEL:
k1
NAME OF 2ND-ORDER KERNEL:
k2
DEFINE THE OUTPUT SIGNAL
y
DEFINE THE INPUT SIGNAL:
x
PLEASE ENTER START SHIFT-VALUE:
0
PLEASE ENTER END SHIFT-VALUE:
40
ZERO-ORDER KERNEL IS EQUAL TO: 1.2476
1ST KERNEL COMPLETED
2ND KERNEL COMPLETED

FORTRAN STOP
$
```

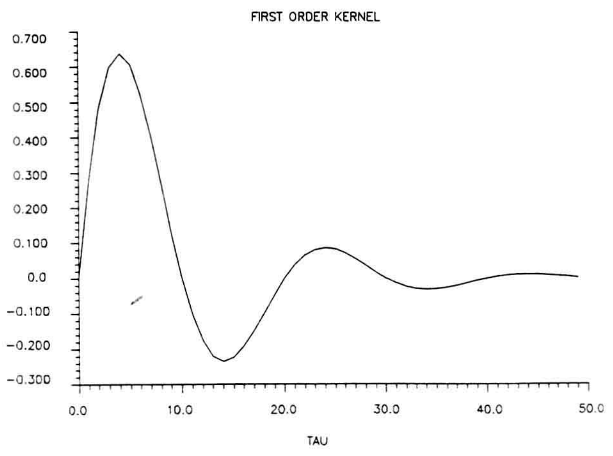


Fig. 1a: First-order Wiener kernel h_1 of simulated system.

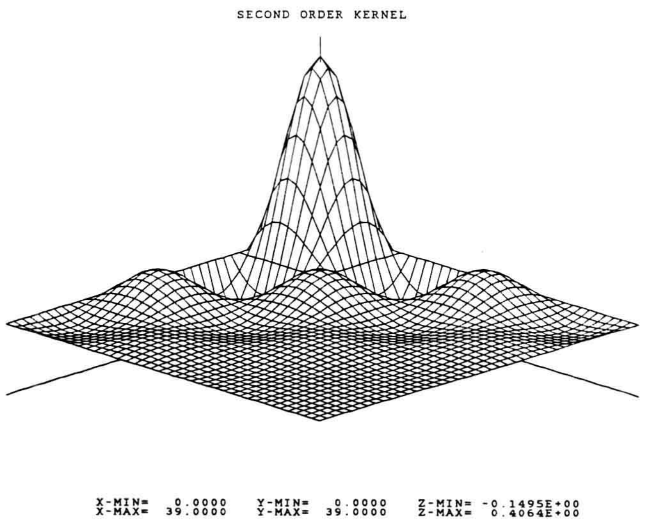


Fig. 1b: Second-order Wiener kernel h_2 of simulated system.