DYSAC: A DIGITALLY SIMULATED ANALOG COMPUTER

J. R. Hurley
Senior Engineering Analyst
Allis-Chalmers Mfg. Co.
Milwaukee, Wisconsin

and

J. J. Skiles
Professor of Elec. Engr.
University of Wisconsin
Madison, Wisconsin

SUMMARY

A digital computer program which simulates a large electronic analog computer has been written for the CDC 1604 digital computer. In addition to providing many non-linear computing elements rarely found in electronic analog computers, the program accepts the input data in a form which may be written down directly from a block diagram or analog computer wiring diagram. Graphical output in the form of plotted curves is available by use of a digital plotter. The simplicity of the input language permits the program to be used easily by persons having no digital computer experience. This digital computer program, called DYSAC, an acronym for Digitally Simulated Analog Computer, is, in reality, a complete programming system, and as with FORTRAN, has a special language to facilitate its use.

Introduction

A large class of engineering problems involving the dynamics of systems is far more easily attacked by simulating the physical system on an analog computer, than by numerical integration of a set of differential equations. The factor which favors the analog machine is that it consists of separate components each performing some particular mathematical function continuously in time. This continuity of operation permits basic analog components to be arranged in groups which simulate complex systems by individually simulating portions of the physical system. Thus a minimum of effort is required for set-up and a physical significance may be attached to the signals at various points in the simulation.

The convenience of analog methods has prompted the development of digital computer programs which simulate the operation of analog computers yet retain the accuracy of digital devices. Selfridge has done some of the basic work in this area. His original approach has been extended by Lesh and Curl into the DEPI program. The existing simulator programs vary considerably in the degree which the input language is related to an analog computer. For example, considerable additional digital coding must be done by the user of DIDAS to set up a problem. On the other hand, Stein, Rose, and Parker have devised a digital compiler for the IBM 704 which employs FORTRAN as an intermediate in the compilation. It accepts an input language tailored expressly for the Electronics Associates PACE analog computer.

The DYSAC program to be described here, written by the first-named author for the Control Data Corporation 1604 computer,* and was designed with particular emphasis on the simplicity and clarity of the input language.

Description of the DYSAC Program

The DYSAC program has been written so that, from the user's viewpoint, there are supplies of analog computer components available

* The University of Wisconsin's CDC 1604 is a high speed, 22K core machine with four magnetic tape units. Typical speeds are 7.2 microseconds for add, 36 microseconds for floating multiply. The 1604 uses a 48 bit wordlength with 2 instructions per word.

From the collection of the Computer History Museum (www.computerhistory.org)
which may be interconnected according to directions supplied by the user. The 1154 components available include integrators, summers, limiters, square root generators, sine and cosine generators, natural logarithm and exponential generators, arbitrary (tabular) function generators of single variables, transport delay units, dividers, and relays as listed in Table I. Multiplication is performed as an auxiliary operation by the integrators and adders.

The input data to describe a problem to the DYSAC program includes the specification of the inputs to each DYSAC analog computing component and is called the patching. (This corresponds to physically patching the components of an analog computer.)

The “patching” information, the numerical data for the problem, and certain auxiliary data are used by the DYSAC program to obtain a numerical solution to the problem.

The numerical methods that are used in the DYSAC program for digital simulation differ from those inherent in analog computation with conventional analog computers in several important ways. In conventional analog computers the outputs of all computing components are calculated simultaneously and continuously in time. In a DYSAC simulation, the outputs of all analog components are calculated for discrete increments in the value of the independent variable (usually time.) For each increment in the independent variable, new outputs for each DYSAC component are calculated in sequence in the order specified in Table I; after new outputs for all components have been calculated (this constitutes an iteration), the independent variable is incremented and the process repeated iteratively until the program reaches a predetermined stopping point.

The DYSAC Input Language

The input data required to describe a simulation have been grouped into seven sections, as given in Table II.

Of the seven data sections required for a problem, five are straightforward and their purpose fairly obvious. All sections of the data are presented on standard IBM cards. The title and headings are merely alphanumeric data used to identify problems and output values. The potentiometer settings, initial values on integrators, and arbitrary tabular functions comprise the numerical data.

Section 7, supplementary machine language instructions, is not generally used, but provides a method of introducing a sequence of computer operations at the basic machine language level. This sequence is executed once each time the main program does one iteration.

<table>
<thead>
<tr>
<th>TABLE I</th>
</tr>
</thead>
<tbody>
<tr>
<td>DYSAC COMPONENTS</td>
</tr>
<tr>
<td><strong>Components</strong></td>
</tr>
<tr>
<td>Integrators</td>
</tr>
<tr>
<td>Adders</td>
</tr>
<tr>
<td>Limiters</td>
</tr>
<tr>
<td>Sq. Root Generators</td>
</tr>
<tr>
<td>Sine Generators</td>
</tr>
<tr>
<td>Cosine Generators</td>
</tr>
<tr>
<td>Log. Generators</td>
</tr>
<tr>
<td>Exp. Generators</td>
</tr>
<tr>
<td>Function Generators</td>
</tr>
<tr>
<td>Time Delay Units</td>
</tr>
<tr>
<td>Dividers</td>
</tr>
<tr>
<td>Relays</td>
</tr>
<tr>
<td>Potentiometers</td>
</tr>
<tr>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
</tr>
</tbody>
</table>
DYSAC: A DIGITALLY SIMULATED ANALOG COMPUTER

TABLE II:
DYSAC INPUT
DATA SECTIONS

1. Problem title.
2. Patching. Description of connections between components.
3. Initial values for integrators.
4. Potentiometer settings.
5. Function tables.
7. Supplementary machine language instructions.

TABLE III:
CONTROL OPTIONS

Section 2, patching, describes the interconnections between the hypothetical analog computing elements available in DYSAC. This description is formulated by giving the inputs to each component used. The components are described in the order given in Table I. The serial numbers shown in this table are used in the descriptions. The method can be best explained by illustration. The “patching” for integrator N01 could be:

N01 = A01A02 + A03.

This is read, “the input to integrator N01 is (the output of adder A01 times the output of adder A02) plus (the output of adder A03)”. Note that multiplication is performed at the input to the integrators. The patching structure for adders is identical. The equal and plus signs used in the patching have no operational significance, they serve merely as punctuation. If the input of N01 was to be the product of the outputs of A01 and A02 minus the output of A03, the standard plus sign must still be used in the patching and thus the patching would be written:

N01 = A01A02 + A03P05.

where P05 is a potentiometer set to a value of -1. Another important point is illustrated in this patching example. Pots are not considered as computational units with inputs and outputs, but are merely constants and can be used as reference values or in conjunction with multiplication as coefficients for variables. There is no limit on the number of terms which may be summed at the input of an adder or an integrator, but the number of factors in each term of the sum is restricted to 20 or less. DYSAC
adders and integrators do not cause an algebraic sign reversal as do their usual electronic analog counterparts.

The patching for other components is fixed in form. For example, a time delay unit accepts only two input signals, one providing the signal to be delayed and the other the amount of the delay:

\[ T_{01} = N_{02}A_{21}. \]

Here the output of time delay generator \( T_{01} \) is the output of integrator \( N_{02} \) delayed by the amount given by the output of adder \( A_{21} \).

**Operation of the DYSAC Program**

The major part of the DYSAC program involves dissecting the alphanumeric input data and converting it into the proper machine language instructions. Since operations of this type are not generally feasible with the FORTRAN compiler available for the CDC 1604, the DYSAC program was written in symbolic machine language. In its present state, DYSAC operates as an interpretive program, that is, during each iteration in the numerical solution the interconnections between simulated analog computing elements are scanned by the program to determine the proper calculation sequencing. After the data describing the analog configuration of a problem is read by DYSAC, one initializing pass is made upon the patching information in order to reduce it to a sequence of memory location addresses. This sequence can be very efficiently and rapidly “interpreted” repetitively by DYSAC, thus the low net computing speed usually associated with an interpretive routine is greatly increased without sacrificing the inherent flexibility of the interpreter.

All the DYSAC components of a given type are “serviced” in order, starting with the adders and progressing to the relays according to the order in Table I. Machine language instructions, if present, are processed next. The integrators are then processed and the sequence continues again with the adders, etc. The program is divided up into sub-sections which each process a particular type of component. During the development of the numerical results of a simulation problem, the mathematical operations implied by the patching description for each component must be performed at each iteration. In order to save computing time during this interpretive iterative solution, the entire patching sequence is translated once by the DYSAC program into a more easily used form before the numerical solution starts. The sequence of serial numbers and punctuation marks is translated into a sequence of numbers by replacing each alphanumeric serial number with the address of the memory location in which the output of that component is stored. Certain numerical codes which can not be confused with permissible component output addresses are used to replace the plus signs and periods which occur in the patching. Serial numbers appearing to the left of an equal sign as well as the equal signs themselves, are deleted from the translated patching sequence. However, a count is made of the total number of each type of component defined in the patching. Then, since all the components were described in order, there can be no ambiguity concerning which component is described by any portion of the sequence of translated patching.

The translated patching sequence is used by the interpreter at each iteration during the numerical solution. The operation of the program can be shown, in principle, by considering the processing of the hypothetical dividers. The dividers follow the time delay units in the processing sequence. Patching for dividers is fixed in format. For example:

\[ D_{01} = N_{01}A_{02}. \]

Here the output of divider \( D_{01} \) is to be the output of integrator \( N_{01} \) divided by the output from adder \( A_{02} \). After patching translation, numbers equal to memory addresses of the quantities which are to be the numerator and denominator occupy consecutive locations in the translated patching sequence. These addresses are taken from a patching table and inserted into “load accumulator” and “floating divide” instructions when the divider \( D_{01} \) is to be processed. These two instructions are then executed and are followed by an instruction which stores the result in the memory location allocated to \( D_{01} \). The program then counts having processed a divider, checks to determine if there are any more dividers in the simulation, and on this basis either proceeds to the portion of DYSAC which processes the next type of component (relays) or continues processing dividers. This is done by increasing the address of the “store output of \( D_{01} \)” instruction so the next time it
functions, it will store the result in the location corresponding to D02. Then the index which is used as a tally to count through the translated patching is increased by 3 (translating the patching for a divider results in 3 consecutive words, a numerator address, denominator address, and a period symbol) and the process is repeated by inserting addresses taken from the patching sequence into “load accumulator” and “floating divide” instructions.

The processing of other components makes similar use of instruction skeletons as outlined for the divider processing.

The sine, cosine, logarithm, exponential, and square root functions are computed with the usual rational approximation subroutines and provide approximately 10 place accuracy.

Special Features

The time delay units produce true transport delays. The delay time can be a continuously variable quantity generated by some other component in the simulation. (This method of delay realization is far superior to DYSAC simulation of usual analog computer techniques such as the use of a Padé approximation, although such delay simulations are straightforward with DYSAC, if desired.)

The arbitrary function generators operate with tables which contain pairs of coordinates of points of the function. An extra degree of flexibility is provided by a feature which enables the function generators to operate with any of three methods of interpolation. The methods available are “1 point” or “boxcar”, 2 point or linear, and 3 point or parabolic. Figure 1 illustrates these interpolation methods.

The relays operate when the output of a specified component exceeds that of a second specified component. When the relay operates, it will set the output of a particular pot or integrator equal to the current output of still another component. An example of relay patching is:

\[ R01 = A01A02P08P09. \]

Relay R01 will set pot P09 equal in value to pot P08 when the output of adder A01 exceeds that of A02. Electronic analog computer relays are usually used to alter the connections between analog computing components. Even though DYSAC relays operate considerably differently than their electronic analog counterparts, they may be made to perform the same function by setting a coefficient pot to zero to “open” a circuit or by changing a zero pot to some non-zero value to “close” a circuit.

Four of the potentiometers (P01 to P04) have special functions during the solution of a problem. Pot P01 is automatically increased during the solution so that it always contains the current value of time (independent variable). P02 contains the value of the time increment used in the numerical integration algorithm. P03 contains the time value at which the solution is terminated. P04 determines the number of iterations performed between points at which the output of specified components are listed. All four of the special pots are subject to alteration by relays during the course of the solution.

When a series of problems is run, it often happens that only a few parameters are changed from the preceding problem. In order to save card punching effort it is possible to retain any of the main data sections (as given in Table II) from the previous problem. These possible variations in the operation of the program are regulated by 7 control cards, one preceding each of the 7 main data sections. Table III lists the various options available for the control cards.

The provision for inclusion of machine language level instructions as part of a problem
may seem to be strange since the purpose of DYSAC is presumably to avoid having to program in machine language. Actually this option was provided originally as a “last resort” method for implementing calculations and special function generation impossible with the normal DYSAC components; however, it has proven valuable for making experimental changes and modifications in the DYSAC program itself. This option is also quite convenient in the simulation of sampled-data systems where the discrete and digital functions may be realized directly by the proper machine language instructions, as is illustrated later in Example 3.

Magnitude and time scaling which are often bothersome necessities of electronic analog problem preparation are, for all practical purposes, not required when using the DYSAC simulator. Floating point arithmetic is used throughout, which provides a 36 bit fraction, a ten bit exponent, and a sign bit for each number (48 bit wordlength). Thus a number range from $10^{-308}$ to $10^{308}$ is covered and about 11 decimal places of significance is provided.

Generation of Special Functions

The DYSAC integrators and adders do not duplicate their analog counterparts in all respects. In particular, nothing in DYSAC corresponds to the summing junction of a conventional operational amplifier. Thus DYSAC cannot duplicate directly some of the standard analog computer techniques for realizing special transfer functions such as the appropriate connection of series or parallel combinations of capacitors, resistors, diodes, etc., to the summing junction. However, such transfer functions can usually be generated easily in other ways by a suitable combination of DYSAC components.

Some simulations may call for functions that are inconvenient, or impossible, to implement with available DYSAC components. Three alternatives are available:

1. Rewrite DYSAC to include the new function generator as an additional component every time the need for a new function arises. This procedure would require the services of a skilled CDC 1604 programmer, familiar with the DYSAC program, to alter extensively the basic structure of the patchword conversion routines, and would necessitate a reassembly of the program after every modification. This procedure has not been used.

2. A simple machine language program using octal addresses and instructions can be written to realize the desired function. This program is then punched on IBM cards as supplemental machine language instructions and read into the 1604 with the patching and data cards for the problem. Differentiation, using the fundamental relation,

$$\frac{dx}{dt} = \frac{x(t) - x(t - \Delta t)}{\Delta t}$$

implemented in this manner requires the execution of 10 1604 instructions. A more accurate differentiation algorithm obviously can be implemented in this manner, if desired. (A differentiator can also be constructed to realize $dx/dt$ by using a time delay unit to generate $x(t-\Delta t)$, a divider, and relays to set up initial conditions. However, the octal machine language differentiator program is executed considerably faster than the processing time for the differentiator constructed by patching the DYSAC components.) Octal machine language is the most commonly used and least costly method of generating special functions.

3. Use a previously written symbolic language program generating the required function. The original symbolic language DYSAC program is then reassembled with any new function programs as subroutines. From the new assembly, the octal addresses of these additional subroutines are noted for reference. If one of these special functions is required it is only necessary to use a few octal machine language instructions to enter and exit from the subroutine. This procedure does not require a time-consuming and costly modification of the DYSAC program logic; however, reassembly of the DYSAC symbolic program is necessary at a cost of approximately 3 minutes of 1604 time whenever new functions are added. These new functions then become permanent additions to DYSAC. (The additional circular functions of tangent, arctangent, arcsine and arccosine have been added to the original DYSAC program to facilitate coordinate conversions required in the solution of such problems as aircraft and missile dynamics.)
Accuracy Considerations

The operation of the simulator program is centered around the fourth order Runge-Kutta method of numerical integration. The choice of integration algorithm is necessarily a compromise between the desirable but conflicting requirements of accuracy, minimum computer running time, ease of starting a solution, and the ease with which the increment in independent variable may be changed during the solution. Although in comparison with Runge-Kutta methods, algorithms like the Adams-Moulton and other predictor techniques permit the use of larger increments in the independent variable $\Delta t$ as well as simplify error estimation during solution, a fourth-order Runge-Kutta method has been used in DYSAC primarily because of the ease both in starting solutions and in making changes in independent variable increment during solutions.

The value of $\Delta t$ must be sufficiently small that the Runge-Kutta process not only is stable, but gives acceptable accuracy. In general, the smaller the time increment, the greater the accuracy, but also the more computer time required. Cumulative round-off error due to very small increments is in general not a problem since the floating point number system used by the 1604 provides about 11 decimal places of significance. If the time increment is too large, the numerical integration method becomes unstable and solutions "blow up".

A certain amount of experimentation with the value of $\Delta t$ is often necessary when beginning unfamiliar simulations. As in selecting the time scale in conventional analog computer studies, prior knowledge of the approximate system natural frequencies and loop gains can be extremely useful in selecting a trial value of $\Delta t$.

The mathematical literature contains methods for approximating the error in the use of several integration algorithms, and suggestions for automatically changing the increment in independent variable to keep the error during each step within prescribed bounds. However, selecting a suitable error tolerance can be quite difficult, particularly for simulations involving many variables, since during portions of a solution relatively small errors may propagate to produce significant errors at later stages of the solution. Conversely, relatively large errors in some variables during certain portions of the solution may be inconsequential in their later effect on the solution. Different methods of numerical integration and possible ways for the DYSAC program to automatically adjust $\Delta t$ during solution to improve accuracy and minimize computer time are the subject of current study.

DYSAC Outputs

The output from a DYSAC digital simulation is: a) a printed tabular listing with seven output variables listed per page, 50 values per variable per page, with a maximum of 56 output variables, but with the number of output values for the variables limited only by number of iterations necessary to obtain the solution to the problem; b) graphical plots of the output variables cross plotted in any desired manner obtained by use of a digital plotter and auxiliary plotter programs.

Two output tapes are written by the 1604 during every DYSAC solution of one or a series of problems. A binary output tape contains only the raw numerical output quantities, with end-of-file marks separating data of different problems. The second, or printer tape, has the numerical results in BCD format with descriptive column headings and additional alphanumeric information as the title assigned to the problem, a complete printout of the patching, a listing of the settings of all pots and function table values. This printer tape is then printed off-line through a small CDC 160 computer using an Anleex 1000 line/minute printer.

The binary output tape is the input to another program, DYPLOT, which rearranges the numerical output, and generates the necessary commands to prepare curves on a CALCOMP Model 570 plotting system. The output of DYPLOT is a third tape, the plotter tape, which is read by the CALCOMP 570 tape unit and the curves then plotted on the CALCOMP 560R incremental digital plotter. The DYPLOT program requires certain additional information, such as the title of the graph, axis designations, scaling, etc., be supplied on punched cards by the user.

Illustrative example:

A block diagram of a simple control system is shown in Figure 2. Its purpose is to energize an inductive load with currents up to 10,000 am-
peres in response to input signals in the range up to 20 volts. The characteristics of the elements are given below.

<table>
<thead>
<tr>
<th>Component</th>
<th>Time Constant</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>amplifier</td>
<td>0</td>
<td>40 volts/volt</td>
</tr>
<tr>
<td>exciter</td>
<td>.4 sec.</td>
<td>1 volt/volt</td>
</tr>
<tr>
<td>generator</td>
<td>2 sec.</td>
<td>25 volts/volt</td>
</tr>
<tr>
<td>load coil</td>
<td>1 sec.</td>
<td>25 amps/volt</td>
</tr>
<tr>
<td>sensing element</td>
<td>0</td>
<td>.002 volt/amp</td>
</tr>
</tbody>
</table>

The DYSAC solution is to show the response of the system to a 20.4 volt step input over a 7 second interval. The time increment selected for the numerical integration is 0.005 second and printout is desired every .02 seconds (every 4th iteration). Figure 3 shows one of the possible DYSAC configurations which will simulate the system of Figure 2. The complete DYSAC input data for the sample problem is reproduced from IBM cards in Figure 4. Figures 5 and 6 illustrate the two modes of output possible with DYSAC, the printed listing and the graphical output provided by the CALCOMP digital plotter.

This simulation involving 1400 iterations on 4 adders and 4 integrators required 40 seconds of CDC 1604 computer time.

Examples of Simulations

Three broad classes of system simulations that are representative of the applications of DYSAC are:

1. Aircraft and missile problems.
2. Electromechanical transients in electric machines.
3. Chemical processes.
Example 1: An Aircraft Simulation

Aircraft and missile simulations are often characterized by the large number of analog computer components necessary to represent the airframe, autopilot, power plant characteristics, aerodynamics, coupling between control axes, and coordinate transformations required by the analysis. The drift and noise free character of digital simulations and the freedom from the usual scaling problems are distinctly of value in these simulations, which are often of linear systems or of systems with only one or two small nonlinearities. Large nonlinearities, when they do occur, can also be handled easily using DYSAC.

In a recent series of studies Grzelak has made extensive use of the DYSAC program in studies of automatic landing systems for aircraft. In these studies the aircraft was presumed to fly a straight line extension of the glide slope of an existing ILS (Instrument Landing System) down to a point at or near the approach end of the landing runway, but at a height of 40 ft. above the runway. A flare computer continuously generated commands to the autopilot which caused the aircraft to follow closely a predetermined flare path and to land about 1500 feet down the runway, with a rate-of-descent at touchdown of 2 feet/second or less.

Of primary interest in these studies was the dispersion of the point-of-touchdown and of the rate-of-descent at touchdown as influenced by the choice of flare path controller and wind gusts. Comparative studies were made of the system performance with many types of flare paths, including exponential, circular, segmented straight-line, and terminal control, and some combinations of these flare paths. The studies were conducted using a simplified pitch-axis representation for a F94C aircraft with an E-10 autopilot because the necessary aircraft and autopilot data and some check solutions from conventional analog computer studies were available.

The block diagram of the system studied is shown in Figure 7. The system is completely linear for some types of flare paths, such as exponential, when the airspeed is assumed to be constant. Air speed computation and some types of flare path controllers resulted in nonlinearities that required the use of dividers and multipliers in the DYSAC representations of these situations. However, the problem was basically that of a linear system and the DYSAC representation for each combination of autopilot-aircraft and flare computer studied employed an average of 13 integrators and 15 adders.

Although the distance to the point of touchdown and the rate of descent at touchdown were of primary concern, additional information on the aircraft performance as a function of time during flare-out was also desired and the following quantities were included in the printout: time (from initiation of the flare), elevator deflection, pitch rate, pitch attitude, change in airspeed, angle of attack, glide angle, height rate (rate of descent), altitude, and distance (from beginning of flare to touchdown).

Approximately 8 seconds of flight time elapses from the beginning of the F94C aircraft flare to the point of touchdown, varying slightly with the type of flare controller and with the nature of the gust disturbances (which were introduced as step changes in the angle of attack or airspeed). The actual solution time on the 1604 computer for a complete flareout solution averaged about 10 seconds, using a value of time increment \( \Delta t = 0.05 \text{ seconds} \), with excellent accuracy in the results. This approximated real time operation. A time increment of 0.1 second could have been used with acceptable accuracy in the results, with even less computer time required per solution.

With such a short computer time required for a complete solution (approximately real time), many combinations of flare controller, airspeed and gust conditions were economically studied, at a computer cost of about $1 per case.

Example 2: Electric Machine Transients

The study of electromechanical transients in electrical machines is complicated by the wide
disparity in some of the electrical and mechanical time constants. In many studies it is convenient, and quite accurate, to assume the machine to be in a steady state condition either electrically or mechanically, thereby greatly simplifying the computer study. However, there are also many problems of interest where it is necessary to represent both electrical and mechanical transients. Digital simulation using DYSAC has been utilized by several University of Wisconsin staff members and graduate students in studies of induction and synchronous machines with non-sinusoidal applied voltages and with complex control circuits, including discontinuous controllers and silicon controlled-rectifiers. The digital simulations have been convenient, and in some cases, possible only because DYSAC handles nonlinearities so easily.

It is common practice to transform the transient electrical equations describing machines to reference frames convenient to the problem under study. Commonly used reference frames include the stator, the rotor, and a synchronously rotating reference frame.

With both electrical and mechanical transients considered, and neglecting saturation, the dynamic equations describing a two-phase, two pole, uniform air gap, wound rotor induction motor, referred to rotor reference axes $\alpha$ and $\beta$ coinciding with the stator phase axes $a$ and $b$, may be written as:

\[
\begin{align*}
(R_{sa} + L_{sa})i_a &= (MP)i_b \\
(R_{sb} + L_{sb})i_b &= (MP)i_a + (MI_{ia})\theta_m + (R_{sa} + L_{sa})i_a + (L_{sa}i_b)\theta_m \\
-\frac{d\theta_m}{dt} &= (MI_{ia})\theta_m + (MP)i_b - (L_{sa}i_a)\theta_m + (R_{sa} + L_{sa})i_a \\
J\frac{d^2\theta_m}{dt^2} + T_L &= T_f = M(i_{ia} - i_{ib})
\end{align*}
\]

where $R_{sa}$, $L_{sa}$, $R_{sb}$, $L_{sb}$, and $M$ are winding constants; $i_a$, $i_b$, $i_{sa}$, and $i_{sb}$ the current variables; $\theta_m$ the angular position of the rotor; $P\theta_m$ the instantaneous speed; and $T_L$ the load torque (a function of speed).

Flux linkage equations are probably more commonly used in machine analysis than equations (1) - (5), which are limited to those applications where saturation can be neglected (as is often reasonable). Both the flux-linkage form of the equations and equations (1) - (4) have been used in digital simulations.

Both equations (1) and (3) contain both $P_{ia}$ and $P_{ia}$; similarly, equations (2) and (4) contain both $P_{ib}$ and $P_{ib}$. Thus two of the same highest-order derivatives appear in each of two pairs of equations, a condition known to lead to "algebraic loops" (loops without integrators), and instability in analog computer setups. Because the variables are thus implicitly defined in terms of themselves, the Runge-Kutta integration method of DYSAC will usually also be unstable if equations (1) - (4) are programmed directly.

The undesirable implicit relations between the variables can be eliminated by eliminating either $P_{ia}$ or $P_{ib}$ between (1) and (3); and eliminating $P_{ib}$ or $P_{ia}$ between (2) and (4). The two resulting new equations are then programmed for solution together with a proper choice of two of the original equations (1) - (4), plus the torque equation (5). There are 16 different combinations of equations that can be derived in this manner.

Studies of electromechanical transients in both relatively high rotor resistance, small inertia servomotors and studies of multiple-horsepower, large inertia machines have shown that some of the methods of elimination give distinctly more accurate results for a specified $\Delta t$ than is obtainable with the other methods. We conclude that attention should be given to the method of elimination used in eliminating algebraic loops in any system of equations and that some experimentation with different methods may be worthwhile.

The nonlinear differential equations (1) - (5) give rise to variable loop gains with the appearance of speed dependent terms such as $(MI_{ia}) (P\theta_m)$, because both $i_a$ and $\theta_m$ are variables that vary with time. Values of $\Delta t$ that result in stable Runge-Kutta numerical integrations for small values of speed ($\omega_m = P\theta_m$), can result in instability and obviously incorrect results at larger values of speed because of the larger loop gains.
In a series of studies of electromechanical transients during the starting of a 5-HP induction machine a value of $\Delta t = 0.001$ seconds gave excellent accuracy, while $\Delta t = 0.01$ seconds gave an unstable solution. It is characteristic of the Runge-Kutta process in unstable situations that the solution appears to be proceeding smoothly, and then diverges suddenly as the speed, and thus a loop gain, exceeds a critical value.

In a series of studies on the starting transients in servomotors, with some of the nonlinearities in the machine represented, the simulations have used 8 integrators, 14 adders, 1 sine generator, 1 cosine generator, 4 function generators, 1 divider and 1 relay. The phenomena of interest lasted about 0.07 seconds and a $\Delta t$ of 0.0001 seconds was used to get acceptable accuracy. The DYSAC solutions for each case averaged about 50 seconds of CDC 1604 computer time.

**Example 3: Sampled Data Systems**

Chemical processes often have inherent transport time delays and DYSAC can be used readily for such simulations. Due to the ease with which nonlinearities may be handled, DYSAC is particularly valuable in conducting simulations of chemical reacting systems, where nonlinearities are nearly always associated with reaction kinetics. A simulation of an isothermal catalytic reactor which was approximated by 5 ideally mixed stages and involving only 3 chemical species required 20 multiplications and 5 divisions. Had the reactor temperature sensitivity been included, an additional 4 function generators would have been required. Turbulent fluid flow problems often require generation of nonintegral powers of flow rate in order to determine pressure drops. The logarithm and exponential generators of DYSAC handle these situations easily and accurately.

The inclusion of relays and transport time delay units in the DYSAC program (as well as the possibility of including discrete logical instructions using the machine language option) has resulted in a programming system well-adapted as a tool for the analysis and simulation of sampled-data or pulsed data systems in addition to continuous systems.

To indicate both the feasibility and ease of data preparation for such applications, a very simple sampled-data system and a digital controller was simulated via DYSAC. The process as shown in Figure 8 consists of a heater with a liquid flowing through it at a constant rate. On leaving the heater, the fluid flows through a long pipe exhibiting a 10 second transport delay. The long pipe empties into a perfectly mixed vessel having a 10 second retention time. The temperature is measured at the outlet of the vessel and is to be controlled by varying the heat rate to the heater. The controller is to be a sampled-data device which compares the desired and actual temperatures every 5 seconds, and makes an adjustment on the heating rate every 5 seconds. The controller is to operate so that the temperature error due to a step change in set-point is reduced to zero in a minimum number of sampling periods. The signal flow diagram for the sampled-data system is shown in Figure 9.

![Figure 8. Sketch of transport process.](Image)

**Figure 8. Sketch of transport process.**

**Figure 9. Signal flow diagram of sampled data system.**
The Laplace transform of the process function is:

\[ G_p(S) = \frac{0.02 e^{-10s}}{1 + 10s} \]  

Using the standard techniques of sampled-data system design, the z-transform of a controller exhibiting the specified characteristics is found to be:

\[ D(Z) = \frac{127.07(1 - 0.6065Z^{-1})}{1 - Z^{-3}} \]

After the form of the desired controller was developed by analytic means, the entire system was simulated by DYSAC. The first step was the formulation of a basic analog computer diagram as shown in Figure 10. Four relays and an integrator can be arranged to create the sampling switch.

As mentioned previously, the possibility of simulating the controller function via machine language instructions exists. This is in fact an easy and efficient course of action to follow since in essence the digital control function is performed by the 1604 digital computer directly instead of being simulated by DYSAC. Changing equation (7) to a time domain recursion formula where \( c \) is the controller output and \( r \) its input:

\[ c_n = c_{n-3} + 127.07r_n - 77.07r_{n-1} \]  

To use this algorithm, the output values \( (c's) \) must be saved for 3 sampling periods and the input values \( (r's) \) for one sampling period.

A slightly different algorithm which closely parallels the analog flow diagram for realization of \( D(Z) \) as shown in Figure 10 may be developed by arbitrarily defining a quantity \( e_n \) as follows:

\[ e_n = \frac{1}{127.07} (c_n + 77.07e_{n-1}) \]  

and thus

\[ c_n = 127.07e_n - 77.07e_{n-1} \]  

substituting for \( c_{n-3} \) in the right member of equation (8)

\[ c_n = 127.07(e_{n-1} + r_n) - 77.07(e_{n-4} + r_{n-1}) \]  

In comparing equations (10) and (11) it is obvious that

\[ e_n = r_n + e_{n-1} \]  

Thus a method of computing \( e_n \) from known quantities has been derived so that it fulfills the arbitrary definition of equation (9). Note that \( e_n \) corresponds to the output of the third adder in Figure 10. The method of realizing the digital controller involves saving only one quantity for 3 sample periods, a saving of 1 storage cell over the method of equation (8). Furthermore, only 19 CDC 1604 instructions are required to implement the controller and the sampling switch action, one less than required for the first method.

The response of this sampled data system to a 50 degree change in set point is illustrated in Figure 11.

**CRITIQUE OF DIGITAL SIMULATION**

**Advantages**

1. The easy-to-learn coding structure of DYSAC enables individuals with no prior knowledge of either analog computing
techniques or digital computer programming to very quickly solve analog computer type problems.

2. It is not necessary for the analog computer type of connection diagram to be drawn for all problems. The DYSAC patching can be written down directly from the differential equation in many instances.

3. Retention of the block diagram representation of systems and components, while not necessary, is a desirable feature to many users. Conventional digital solution of system problems often results in some loss of identification of the system.

4. Elimination of the necessity for number scaling with DYSAC is a substantial advantage over conventional analog computer studies, particularly in the case of complex, nonlinear systems.

5. The problems of noise and drift in dc levels so familiar in analog computers do not exist with digital simulation.

6. Nonlinear operations such as squaring, cubing, forming sine, cosine, exponential, logarithm functions, and multiplication present no special problems in digital simulation with DYSAC.

7. The availability of true representation for time variable transport delays is a real advantage over most of the analog computer techniques that use truncated expansions of various functions to approximate, and often poorly, transport time delays.

8. The patching and data for a problem can be completely and inexpensively checked before going on the computer. While punching and checking program and data cards may take an appreciable time for a large simulation, the digital computer is not required.

9. Individual case studies can be run with practically no setup time once the program patching deck is available. (This compares favorably with the use of separate patch panels, tape set pots, etc., in large analog computer installations.)

10. Studies of many different problems can proceed almost simultaneously, unlike with an analog computer installation where one complex simulation may tie up the computer for several weeks.

11. The floating point arithmetic used in DYSAC can be used to achieve accuracy in problem results that is completely unattainable with an analog computer.

12. Graphical outputs from digital simulations are easily prepared, are accurate, and are readily reproducible when using a digital plotter such as a California Computer Products Model 570 off-line magnetic tape plotting system.

13. The ability to do large scale analog-computer type simulations on a digital computer is attractive to many users having good digital computing facilities and little or no analog computation capability.

Disadvantages

1. A DYSAC solution to a problem will require more computing time than would be required for a conventional digital computer program written by a skilled programmer to run on the same computing machine. (However, the time required to program DYSAC will be much less than the time necessary to write a conventional program to solve the same problem.)

2. Some flexibility is lacking in DYSAC, as in similar programs, because the user is usually limited to only those components available in the program. The provision in DYSAC for special machine language instructions substantially improves the flexibility of the program.

3. In its present form, neither DYSAC nor similar programs have provision for the introduction of real-time data from external hardware, as is done in some analog computer simulations.

4. The choice of the value of $\Delta t$ to be used in a simulation is a compromise; a small value of $\Delta t$ is desirable for accuracy, while a large value minimizes computer rental cost. While rule-of-thumb guides can be used to determine an initial value of $\Delta t$ to be used for a given problem, no procedure exists for manually or automatically selecting a value of $\Delta t$ to give a prescribed accuracy to all output quantities of interest at all times during a simulation. Some experimentation with the value of $\Delta t$ is often necessary when studying a new system of equations.
5. Because of the expense of computer time, digital simulation does not permit the type of experimentation with pot settings and immediate observation of the results that is possible with analog computers, particularly for repetitive type computers.

6. To avoid unnecessary computer rental, decision making and data changes while actually running problems must be discouraged.

7. The computer running time and thus the cost of digital simulations varies almost directly with the number of components in the simulation, in contrast with analog studies where computer running time is not a function of the number of components. (However, rental costs for an analog facility will increase with the size of the facility.)

8. As presently written, DYSAC is not readily adaptable to random process studies, although Hurley\textsuperscript{10} has made a number of simulations of an adaptive process controller using a machine language random noise generator.

CONCLUSIONS

Digital Simulation should be considered as an extremely valuable supplement for an analog computer, but not as a complete replacement. It will prove very useful where inadequate analog computer facilities exist to solve a particular problem, whenever more accuracy is desired than can be obtained from an analog computer, or when it is necessary to obtain results in less elapsed time than would be required for either a conventional analog or a digital computer solution. Although Digital Simulation is usually excessively time consuming and costly on small digital computers, it is economically attractive on computers in the CDC 1604 and IBM 7090 class.

The DYSAC program is receiving wide usage at the University of Wisconsin and research is continuing both in the development of improved digital simulation techniques and in applications in many areas of engineering and science.

ACKNOWLEDGEMENTS

The DYSAC program was developed as one phase of a continuing project on digital simulation, under the direction of Professors J. J. Skiles and V. C. Rideout, Department of Electrical Engineering, University of Wisconsin, with National Science Foundation support under grant G19886.

Early phases of the \textquoteleft \textquoteleft Digital Simulation\textquoteright \textquoteright project were supported in part by the Research Committee of the Graduate School from funds supplied by the Wisconsin Alumni Research Foundation.

We wish to acknowledge the many helpful suggestions and comments from Professor V. C. Rideout, who is participating in the Digital Simulation studies.

REFERENCES