INTRODUCTION

Interactive Simulation Language (ISL) can be used by scientists and engineers without prior programming experience for the solution of nonlinear algebraic and differential equations. ISL permits hands-on interactive operation in which the user can monitor the course of a computation (via scope, printers, etc.) and change parameter values during execution (via knobs, switches, keyboard). ISL functions well in either an all digital system or in a system utilizing an analog interface. Since ISL is modular computing functions can be added or deleted for tailoring the language to many applications in data collection, data analysis, curve fitting, hybrid computer simulation, etc.

ISL processes simulations quite rapidly by using a single word mantissa and a single word exponent (minipoint). In addition extremely large problems can be solved on a small computer. ISL is currently operational on PDP 8/12, PDP 7/9/15, PDP 11, NOVA, EAI 640/PACER 100, DDP/PRIME, and many hybrid computers. This paper shows application of ISL to neurobiology by:

- presenting an overview of ISL programming
- illustrating the techniques for the simulation of a repetitively firing nerve cell
- presenting comparative results for ISL and SNAX

ISL PROGRAMMING OVERVIEW

It is easiest to consider the ISL system as providing a large number of operational elements such as found on an analog computer. These elements include integrator, add, multiply, and many others (see Table 1). An ISL program is formed by "interconnecting" these elements to satisfy the equation being solved.

Drawing the Block Diagram

To illustrate the programming technique consider the equation for adaptation (A) of a nerve cell:

\[
\frac{dA}{dt} + 0.6(A - A_{INF}) = 0 \quad A(0) = 1.0
\]

The first step in programming is to equate the highest derivative to the lower order terms. For example:

\[
\frac{dA}{dt} = -0.6(A - A_{INF})
\]

The simulation diagram (using symbols from Table 1) is:

Next by adding an integrator to the diagram the function (A) is available:

Notice the initial condition is shown as an auxiliary input to the integrator. To complete the simulation diagram it is necessary to assign numbers to each operational element. The rule for assigning numbers is analogous to building the simulation diagram. That is, to form the derivatives and then integrate the highest derivative first. Thus:

The final step in preparing the simulation diagram is to form operational blocks that:

- Label headings (MHE statement)
- Identify printing or plotting variables (PRI)
- Allow the program to iterate and determine when computation should stop (FIN statement)

To aid in the interpretation of tabulated output data, provision exists for specifying headers above each column. These headings can be the names given to the variables identified by the operational equivalent in the print (PRI) statement. Up to five names (having six characters can be included in one heading statement (MDH). The MDH automatically labels the first value as TIME before using the defined labels. Accordingly the print statement gives the value of time. A header and print statement for the example could be:

\[
\text{MDH A}
\]

A finish (FIN) statement, as shown below, would be incorporated into each ISL program. This statement allows iteration and also provides the user with a means of manually terminating a run with a keyboard interrupt. Its diagram would be:

When the value of the first input (time) is less than the second input, then for a new iteration, the FIN statement transfers control to statement N. When the first input is greater than or equal to the second input the next numerical statement is executed. In this example we will use an END statement to terminate the run.

Preparing the Listing

The program is entered into the computer by using the instructions shown in Table 2. The user area of memory is first erased with the "K" command. Then by typing "A" (for APPEND) the teletypewriter responds by typing (N) for the first line number. The user may then type in the program:

\[
\begin{align*}
0 & \text{ CON = 1.0E-1} \\
1 & \text{ ADD = -0.6, 0} \\
2 & \text{ POT = 1.0E-1} \\
3 & \text{ CON = 1.0E0} \\
4 & \text{ INT 3, 2}
\end{align*}
\]
S = 1.000E-2    T = 10

TIME        RANGE = 1.000E 0
0.000E 0     0.000E 0
9.999E-2     9.999E-2
2.000E-1     2.000E-1
3.000E-1     3.000E-1
4.000E-1     4.000E-1
5.000E-1     5.000E-1
6.000E-1     6.000E-1
7.000E-1     7.000E-1
8.000E-1     8.000E-1
9.000E-1     9.000E-1
1.000E 0     1.000E 0
1.000E 0     1.000E 0
1.000E 0     1.000E 0
1.000E 0     1.000E 0
1.000E 0     1.000E 0

a. Tabular output

b. Teletype Plot

Figure 1. ISL Output for Adaptation Example

5 TME 5
6 HDR A
7 PRI 4
8 CON = 1.5E 0
9 FIN 5, 8
10 END 11

For each line of code the teletype prints a line number, following which the user types a three-character function code, followed by its arguments (separated by commas). Arguments are either "inputs" to the elements (see integrators) or numerical values assigned to them (floating point notation for P01 and CONS).

If a spelling error is detected while entering the program the printer will respond by typing a question mark (?) and retype the statement number. It then waits for the correct information. The computer will not catch "patching errors." For example if a7 were typed for a4 in statement 1, the computer would not recognize the error. However the user can correct any numerical error at any time prior to terminating the entry with the "slash" (/). This signals the computer to cancel the entry and accept a new one. Any statement can be modified or re-entered with the MODIFY (M) command shown in Table 2.

Running the Program

The program is run by selecting the integration step size and print interval with the "I" command. For the current example a step size of 5 = 1.OE-2 and a print interval of 10 will allow tabular output as shown in Figure 1. By using the X command of Table 2 the information is plotted. Notice the range for 50 spaces is set to 1.0 as defined by the RANGE = 1.000E 0 command.

Of much more interest is output to the scope and repetitively run the simulation while varying A1W with an analog pot. This may be done by modifying the following lines with the "W" command:

0 ADC 0 (use pot O to change A1W)
6 DIS 5, 8, 4, 7 (plot A versus Time on scope)
7 CON = 2.0E 0 (normalized value of A)
10 CHM 10 (allows repetitive operation)
11 END 11

With scope output and parameter variation through an analog pot (through an ADC channel) better interactive communication is established than with just a TTY printer or a line printer. With this form of interaction one can begin to appreciate the speed and power of a simulation using ISL. Solutions will appear about 250 times faster than possible with interactive languages like BASIC. In addition an BK computer equipped with ISL can solve up to 100 non-linear differential equations. Over 1000 equations can be solved with 32K of memory.

SIMULATING A REPETITIVELY FIRING NERVE CELL

This section illustrates the use of ISL on the more complex problem of a repetitively firing nerve cell. The model is summarized with the simulation diagram of Figure 2 and the listing of Figure 3. Notice that the ISL simulation diagram combines both the mathematical and logical flow diagrams for the nerve cell. As such it is an aid (ie. mathematical analog) in visualizing the operation of the cell. The model is composed of 6 non-linear differential equations that account for:

- Conductance associated with each spike (gs)
- Current injected with a microelectrode (I)
- Resting potential adaptation (Ar)
- Short and long term active response adaptation (A1, A2)
- Active pacemaker response (P)
- Threshold (Q)
- Perturbation variable (W)

The basic equation for each term is similar to the equation for adaptation used for describing ISL programming in the last section. However there are additional complexities as can be seen by close inspection of Figure 2. A cell will fire (spike) when the membrane potential (V) becomes equal to or greater than the threshold (Q). When this happens new initial conditions are automatically calculated and imposed for the terms gs, Ar, A1, A2, W, and A. The form of the IC up-date equation is given for the adaptations:

\[ A_{\text{new}} = A_{\text{old}} + A_{\text{old}}(A_{\text{max}} - A_{\text{old}}) = A_{\text{old}}(1 - \frac{A_{\text{old}}}{A_{\text{max}}}) + A_{\text{max}} \]

The new value is a fixed fraction of the distance from the old value to a maximum value. The ISL implementation for this equation is:

The initial condition calculations appear after the FIN statement (no. 51) so that all are calculated when V = 0 and t = time of the spike. Since ISL includes a
Figure 2. ISL Simulation Diagram for Repetitively Firing Nerve Cell

Figure 3. ISL Listing for a Repetitively Firing Nerve Cell
second IC iteration the calculation must not be up-dated during the second pass. The solution to this is the INC block which allows correction during the compute pass but not the second IC pass through the program. Since the INC block adds the new input to the old output, the INC output must be subtracted from the input to satisfy the above equation.

The Change Mode (CHM) instruction (block 81) allows simulation control for repetitive operation. ISL has two modes: "compute" in which normal program execution occurs and "initial conditions" (IC), in which initial conditions are set as shown in the preceding example. When the CHM is executed and ISL is in IC, the mode is changed to compute. Control is then transferred to block 0 (zero). Conversely when CHM is executed and ISL is in compute mode the mode is changed to IC for an initial condition pass. The IFM statement then allows for the next instruction to be executed when in IC mode. This allows the CHM block to again change the mode back to compute from IC mode.

Notice that blocks 0, 81, and 1 are used for generating time. Each time there is a spike (V = 0), time as generated by the TM block (T) is automatically reset to 0. Therefore by using the INC (block 81) the value of time (when V = 0) is added to the last INC value and remembered. Thus by adding the output of TM and INC the true accumulative time is made available.

Notice also that the input relays (IRL) as shown in blocks 28 and 30 are used to select values for $r_p$ and $P_{m0}$ for the active response equation (P) which are dependent on condition requirements on certain variables.

Statements 45, 46, and 47 show the ISL commands for outputting results to digital to analog converters which are in turn used to drive a scope (see Figure 4).

Comparision of ISL and SNAX

The SNAX language was developed for the PDP 11/45 computer and is currently used for math model development and testing of the nerve cell simulation. The purpose of the ISL implementation is to provide comparative information for the two languages.

Approximately 4 hours were required to study the math model and draw the preliminary ISL flow diagram. This experience generated several questions that required consultation for correction. An additional seven hours were needed to enter, debug, test, and compare the model with the SNAX version. The same PDP 11/45 computer was used. ISL required 15 seconds of computer time to simulate the 14 firings shown in Figure 4. The ISL program required about 520 memory locations to hold the 85 lines of code. Although provision exists for using hardware multiply/divide within ISL, this feature was not used for the comparison run. ISL could have processed the information substantially faster if this additional hardware would have been used.

The SNAX simulation required 230 lines of code. SNAX processed the simulation in 14 seconds by using the floating point processor available on the PDP 11/45. The results for ISL and SNAX were found to be identical as can be seen by comparing Figure 4. To verify the comparison of ISL and SNAX a run was made where SNAX results were written over the ISL results (saved on a memory scope) and no appreciable difference could be noticed.

Figure 3A and 3B show two different time scales for a simulation of an abrupt depolarization to the cell which is responded to by an initially high but exponentially declining firing rate. For each simulation the upper line represents threshold (Q) and the lower line membrane potential (V). Impulse time-course is not simulated, but impulses occur at the discontinuities of Q and V, representing resets of these quantities by the spike.

Conclusions

ISL is an efficient general purpose language suitable for simulation of analog computer functions on minicomputers. Its block diagram representation scheme aids in visualizing the processes being simulated. For an on-line interactive language it is rapid and conservative of memory. Its availability on a number of different minicomputers makes ISL programs "exportable" for many of the computer systems currently employed in neurobiology research.

Acknowledgements

Support by Interactive Mini Systems, Inc. and grants from NSF (#GJ-43177), the Alfred P. Sloan Foundation, and the University of California Academic Senate to DKN.

References


2. Hartline, D. K., "SNAX: A language for interactive neuronal modeling and data processing", Department of Biology, University of California, San Diego, La Jolla, California 92037
### Table 1: Mathematical Instructions

<table>
<thead>
<tr>
<th>Function</th>
<th>Instruction</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute value</td>
<td>ABS N1</td>
<td>R =</td>
</tr>
<tr>
<td>Add/Subtract</td>
<td>ADD aN1, aN2, ... aN9</td>
<td>R = aN1 + aN2 + ... + aN9</td>
</tr>
<tr>
<td>Constant</td>
<td>CON Nk</td>
<td>R = Nk</td>
</tr>
<tr>
<td>Divide</td>
<td>DIV N1, N2</td>
<td>R = N1/N2</td>
</tr>
<tr>
<td>Equate</td>
<td>EQT N1, N2</td>
<td>R = 0 : N2 = N1</td>
</tr>
<tr>
<td>Multiply</td>
<td>MUL N1, N2</td>
<td>R = N1 x N2</td>
</tr>
<tr>
<td>Pot</td>
<td>POT N1, y</td>
<td>R = N1 x y</td>
</tr>
</tbody>
</table>

### Table 2: Transcendental Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Instruction</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sine</td>
<td>SIN N1</td>
<td>R = sin N1 radians</td>
</tr>
<tr>
<td>Cosine</td>
<td>COS N1</td>
<td>R = cos N1 radians</td>
</tr>
<tr>
<td>Logarithm</td>
<td>LOG N1</td>
<td>R = log10 N1</td>
</tr>
<tr>
<td>Exponential</td>
<td>EXP N1</td>
<td>R = 10^N1</td>
</tr>
</tbody>
</table>

### Table 3: Subroutine

<table>
<thead>
<tr>
<th>Function</th>
<th>Instruction</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subroutine</td>
<td>SBR N1, N2, ... N8</td>
<td>N1 = block transfer no, N2, ... N8 are inputs</td>
</tr>
</tbody>
</table>

### Table 4: Special Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Instruction</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input/output</td>
<td>HDR T1, ... T5</td>
<td>label TTYL/P columns with up to 6 chrs. each</td>
</tr>
<tr>
<td>Print/plot</td>
<td>PRI N1, ... N5</td>
<td>print/plot to TTYP/P</td>
</tr>
<tr>
<td>Display*</td>
<td>DIS N1, N2, N3, N4</td>
<td>X = N1/N2 Y = N3/N4</td>
</tr>
<tr>
<td>Digital* to Analog Conv</td>
<td>DAC N1, N2, K</td>
<td>R = N1/N2 K = channel no.</td>
</tr>
<tr>
<td>Analog* to Digital Conv</td>
<td>ADC K</td>
<td>R = ADC output K = channel no.</td>
</tr>
</tbody>
</table>

### Table 5: Control

<table>
<thead>
<tr>
<th>Function</th>
<th>Instruction</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change mode</td>
<td>CHM K</td>
<td>reverses mode IC to C or C to IC</td>
</tr>
<tr>
<td>Count</td>
<td>CNT K</td>
<td>when iteration count = K skip next instr and reset count to K</td>
</tr>
<tr>
<td>Finish</td>
<td>FIN N1, N2</td>
<td>if N1 = N2 take next instr otherwise go to block O and go to block O</td>
</tr>
<tr>
<td>Goto</td>
<td>GTO Nk</td>
<td>if N1 = N2 go to N3</td>
</tr>
</tbody>
</table>

### Table 6: Logical

<table>
<thead>
<tr>
<th>Function</th>
<th>Instruction</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increment</td>
<td>INC N1, N2</td>
<td>N1 = N1 + N2</td>
</tr>
<tr>
<td>Integral</td>
<td>INT N1, N2, ... N9</td>
<td>R = (\int_{N2}^{N9} dt )</td>
</tr>
</tbody>
</table>

### Table 7: Input/Output

<table>
<thead>
<tr>
<th>Function</th>
<th>Instruction</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>THE</td>
<td>each iteration advances time by S</td>
</tr>
</tbody>
</table>

### Table 8: ISL Monitor Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Append program</td>
</tr>
<tr>
<td>C</td>
<td>Enter program from external device</td>
</tr>
<tr>
<td>E</td>
<td>Enter job (program) title</td>
</tr>
<tr>
<td>L</td>
<td>List current program</td>
</tr>
<tr>
<td>M</td>
<td>Modify statements and through</td>
</tr>
<tr>
<td>S</td>
<td>Select plotting routine and set range</td>
</tr>
<tr>
<td>T</td>
<td>Select printing frequency</td>
</tr>
<tr>
<td>U</td>
<td>Select output unit</td>
</tr>
<tr>
<td>V</td>
<td>Select printing destination</td>
</tr>
<tr>
<td>X</td>
<td>Select printing format and set range</td>
</tr>
</tbody>
</table>

### Note
Numerical entries are terminated with a space, comma, or carriage return (CR).