ELLIOTT 900 SERIES SIMULATOR

ELLIOTT SIMULATION PACKAGE (ESP).

Elliott provided ALGOL users with a source code library of
procedures for event-based simulation.

ESP is a source code library consisting of a partial program
terminated by a halt code and a relocatable binary tape
containing a random generator. The programmer should load the
ESP source (ESP1(AJH).900 in the ESP directory) for
translation. When the ESP source code has been read, the
programmer should read in his own program which should be
followed by 2 * end statements to close the blocks opened
within the ESP source. (On a real 903 the programmer would
likely first prepare a copy of ESP containing just the
routines needed in their program, to economize on memory use).
The tape containing the random number generator should be
loaded as an additional relocatable code tape to the ALGOL
interpreter.

Note (1) there are two intermediate tapes, ESP2(ISS1).RLB and
ESP2(ISS2A).RLB: ESP2(ISS1) is for versions of ALGOL up to and
including issue 5, ESP2(ISS2A) is for versions from issue 6
onwards.

Note (2) ESP1(AJH).900 is an edited version of ESP1(ISS2).900,
which is a copy of the tape distributed by Elliotts. The
changes include some tidying up of how printing is handled and
the removal of intermediate halt codes.

When the program is run it first prints to the teletype
showing the amount of free store available. The user is then
invited to input the amount of memory to be allocated to an
event list data structure internal to the simulator. The
input should be a positive integer. 0 selects the default
allocation of 500 elements (each requiring two words of
memory).

Before using any other facilities of the simulator the user's
program should call

prepare;

This initializes the ESP data structures and seeds the random
number generator.

ELLIOTT 900 SERIES SIMULATOR

The contents of the library are made up of three sets of
procedures: the event simulator, statistical sampling
procedures and histogram procedures.

Event simulator.

Introduction.

The programming model for ESP assumes there are a series of
event classes that lead to actions, and actions may schedule
further events. Actions are represented as integers and
action 1 is assumed to be a housekeeping event that checks the
state of the simulation following other actions and if
necessary schedules further actions based on state changes
made by preceding actions. Events can be parameterized with
additional information, which is then made available to the
corresponding action when scheduled. Actions may schedule
further events by using the procedure "call". "call (hap, t)"
schedules an occurrence of event class "hap" at time "t".

The event simulation should be structured as a series of
blocks of code, one for each event class, including one for
the housekeeping event class. Each block should be labelled
and a switch statement constructed with each label in the
corresponding position to the integer by which the action is
coded: for example:
     switch act := [housekeeping, enter, exit, ..];

Each simulation block should finish with a statement such as
     goto act[next]
where "next" is a procedure that computes the next action to
occur, based on the queue of pending events set up by use of
"call". If there are no pending actions for the current time
step, "next" returns 1 to indicate housekeeping should be
done.

Finally, note that before "call" or "next" are used, the
procedure "prepare" should be called to initialize the
simulation system.

Detailed description.

procedure prepare;

Set up the simulation package ready for use. Should be called early on in the application to initialize internal data structures.

procedure call (hap, t); value hap, t; integer hap, t;

Schedule the event hap to occur t time steps in the future.

hap should be a positive integer, greater than the value 1, the interpretation of which is left to the user's program. Typically it encodes the class of event being scheduled. If it is desired to associate further information with the event this can be done by writing integers into the array send [1], send [2], ..., send [10]. The number of elements of send that contain useful data should be written in send [0]. This data is then copied into the event time line to be returned when the scheduled event is returned by the next procedure (q.v.).

integer time;

The variable called time holds current time step in the simulation. This variable is initialized to 0 by "prepare". The variable "time" is advanced by the procedure "next", which retrieves the next scheduled event at or after the current time step in the simulation. The user program must not update the variable "time".

integer procedure next;

Retrieve the next scheduled event at or after the current time step.

If there are no further events queued for the current time step, a first call of next returns the value 1. This is an indication that the user's program should review the state of the simulation following the sequence of preceding events and if necessary use the procedure "call" to set up future events triggered by changes in the simulation state. When next is then called again, time is advanced to the time of the next event in the event queue.

The action of next when there are scheduled events at the
current time step is for one event to be selected and removed
from the event list. The procedure returns the value (plus 1)
of the hap parameter used at the time the event was scheduled
using call (q.v.).

If additional parameters were associated with the event these
are available in the array get [1], get [2], ..., get [10],
with the number of elements used recorded in get [0].
If there are no associated parameters, get [0] will contain 0.

The user must keep track of how many pending events are in the
simulation: calling next when there are no events scheduled
results in an error message.

Random Sampling Procedures.

The application should ensure the "prepare" procedure
described above is called before using any of the random
sampling procedures.

procedure initialize (a, b, c);
real a, b, c;

Compute initial parameters for the random number generator.
Not intended for use by applications. (Implemented as a code
procedure. Called by the "prepare" procedure).

integer procedure x base random (rand, mult1, mult2);
real rand, mult1, mult2;

The core random number routine (implemented as a code
procedure). This is not intended for use by applications.

real procedure set random;

Computes a seed random number for a new independent sequence
of random numbers.

integer procedure random (a, b, x);
value a, b; integer a, b; real x;

Using the random sequence x, sample the uniform distribution
in the interval (a, b).

<u>real</u> <u>procedure</u> geometric (p, u);
<u>value</u> p; <u>real</u> p, u;

The geometric probability distribution function represents the
number of trials required before a success is observed for a
given probability of success (p) for each trial.

Using u as a seed, sample a geometric distribution with
parameter p. p must be in the interval (0.0, 1.0).

<u>real</u> <u>procedure</u> log normal (mean, sigma, u);
<u>value</u> mean, sigma; <u>real</u> mean, sigma, u;

The log-normal distribution is a probability distribution of a
random variable whose logarithm is normally distributed with a
specified mean and variance (sigma).

Using the random sequence u, sample a logarithmic normal
distribution with parameters mean and sigma.

<u>real</u> <u>procedure</u> neg exp (mean, u);
<u>value</u> mean; <u>real</u> mean, u;

The negative exponential distribution describes the time
between events in a Poisson process, i.e., a process in which
events occur continuously and independently at a constant
average rate (mean).

Using the random sequence u, sample the negative exponential
distribution with the rate specified by the parameter mean.

<u>real</u> <u>procedure</u> normal (mean, sigma, u);
<u>value</u> mean, sigma; <u>real</u> mean, sigma, u;

Using the random sequence u, sample the normal distribution
with specified mean and variance sigma.

<u>real</u> <u>procedure</u> fast normal (mean, sigma, u);
<u>value</u> mean, sigma; <u>real</u> mean, sigma, u;

A faster, but less accurate, version of normal.

<u>real</u> <u>procedure</u> poisson (mean, u);
<u>value</u> lambda; <u>real</u> lambda, u;

The Poisson distribution describes the probability of the number
of events occurring in a fixed period of time for a Poisson
process with average rate mean.

Using the random sequence u, sample the Poisson distribution
with the rate specified by the parameter mean.

```
integer procedure binomial (n, p, u);
value n, p; integer n; real p, u.
```

The binomial distribution describes the expected distribution of
the number of successes in a sequence of n independent yes/no
experiments, each of which succeeds with probability p.

Using the random sequence u, sample the binomial distribution
for n trials each with probability p.

```
integer pascal (m, p, u);
value m, p; integer m; real p, u;
```

The Pascal (negative binomial) distribution describes the
expected number of trials in a sequence of Bernoulli trials,
each with probability p, before m failures occurs.

Using the random sequence u, sample the Pascal distribution
for m failures of a series of trials each with probability p.

```
integer erlang (a, k, u);
value a, k; integer k; real a, u;
```

The Erlang distribution describes the waiting times between k
occurrences of a Poisson event occurring at rate a.

Using the random sequence u, sample the Erlang distribution
with rate a and shape k.

```
integer hypergeom (tn, ns, p, u);
value ns; integer tn, ns; real p, u;
```

The hypergeometric distribution describes the number of
successes in a sequence of ns draws from a finite population
of size tn without replacement where the ratio of successes to
failures in the population is p.

Using the random sequence u, sample the hypergeometric
distribution for a population of size tn, with ratio p and ns
trials.

The parameter tn will be set to the size of the remaining
population after sampling (i.e., the initial value of tn less

ns) and p to the expected ratio of successes to failures in
that population.

Histogram procedures.

Histograms are represented by <u>integer</u> <u>arrays</u>. A histogram
array has dimension [-7:N] with the elements used as follows:

     Histogram [-7] is the number of buckets to be used (N)
     histogram [-6] is the lower bound of the first bucket(L)
     histogram [-5] is the width of each bucket(W)
     histogram [-4] is the count of values entered
     histogram [-3] is the sum of the values
     histogram [-2] is the sum of the squares of the values
     histogram [-1] is a count of underflow entries – i.e.,
                     entries < L
     histogram [0]   is the count of entries in the first
bucket
     ...
     histogram [N-1] the count of entries in the N-th bucket
     histogram [N]   is a count of overflow entries – i.e.,
                     entries > (L+N-1)*W.

<u>procedure</u> hst set (a, n, l, w); <u>value</u> n, l, w; <u>integer</u> n, l,
w;
<u>integer</u> <u>array</u> a;

Initialize the array a as a histogram with n buckets of width
w, ranging from l to l+(n-1)*w.

<u>procedure</u> hst put (a, x); <u>value</u> x; <u>integer</u> x; <u>integer</u> <u>array</u> a;

Add the value x to the histogram a.

<u>procedure</u> hst true (a, m, v); <u>integer</u> <u>array</u> a; <u>real</u> m, v;

Set m to the mean value of the histogram a, and v to its
variance, including underflow and overflow values, i.e., the
mean and variance of all elements put into the histogram.

<u>procedure</u> hst group (a, m, v); <u>integer</u> <u>array</u> a; <u>real</u> m, v;

As for hst true, but not including underflow or overflow
elements. However, if such elements are present, a variance
of -1.0 and a mean of 10^{78} is returned.

procedure hst print (a, devout); <u>value</u> devout; <u>integer</u> devout;
<u>integer</u> <u>array</u> a;

Output the histogram a to device devout. The output shows a
line for each bucket (including under and overflows) showing:
   1. The lower and upper bounds of the bucket
   2. The count of values in the bucket
   3. The fraction of values in the bucket (i.e., bucket count
      divided by total across all buckets)
   4. The cumulative fraction of values up to an including the
      bucket.

procedure hst add (a, b, c); <u>integer</u> <u>array</u> a, b, c;

Set the histogram c to be the sum of histograms a and b (i.e.,
a+b). An error message (ERR4) is printed if a and b have
different values for N, L and W.

procedure hst sub (a, b, c); <u>integer</u> <u>array</u> a, b, c;

Set the histogram c to be the difference of histograms a and b
(i.e., a-b). An error message (ERR4) is printed if a and b
have different values for N, L and W.
procedure hst mult (a, k, c); value k; integer <u>array</u> a, c;
<u>integer</u> k;

Set the histogram c to be the result of multiplying histogram
a by the integer scalar k. An error message (ERR5) is printed
if a and b have different values for N, L and W.

procedure hst in (a); <u>integer</u> <u>array</u> a;

Read in a histogram from paper tape. The input should be at
least integer values for each of N, L, W, count of values
entered, sum of values entered, sum of squares of values
entered, count of underflow entries, N successive count of
entries for each bucket, count of overflow entries. (If the
count of values entered, sum of values entered and sum of
squares of values entered are ALL input as zeros, their values
are recalculated. Otherwise a check is made that the total of
the values counted in each bucket (including under and over
flow buckets) adds up to the input count of values entered.
If this check fails an error message (ERR6) is printed.
procedure hst out (a); <u>integer</u> <u>array</u> a;

Output a histogram to paper tape as a line of integers
suitable for input by hst in.
<u>integer</u> <u>procedure</u> hst sample (a, u); <u>real</u> u; <u>integer</u> <u>array</u> a;

Sample a random value from the histogram a with seed u.

<u>Error messages.</u>

If an error is encountered in the simulation, a short message
is output to the teletype and the program stops.

ERR1: The internal event list is full.

ERR2: Attempt to extract an event when the internal event list
is empty.

ERR3: Problem with use of the call procedure: specifically,
hap <= 0, t < 0 or t + time > 131071, or send [0] < 0 or > 10.

ERR4: A, N, L or W values mismatch in histogram parameters to
hst add or hst sub.

ERR5: A, N, L or W values mismatch in histogram parameters to
hst mult.

ERR6: Input values for hst in are inconsistent (count of
elements in buckets does not match count elements in
histogram).

<u>Notes on the event list.</u>

Events are stored at the top end of this list in the form of a
heap. Each event is a pair, the first element is the time of
the event, the second either a (non-negative) to a parameter
record or a negative event type (-hap). When an event is
scheduled it is removed from the vector and any associated
parameter record marked as free. Parameter records are stored
at the bottom end. Each record starts with a three word
header: first a flag where 0 means "in use" and -1 means
"free", second a length of the data part, third an event type
code (hap). The header is then followed by the required
number of data words. Whenever more space is needed in the

event list, free parameter records are removed and allocated
records above them are shifted down to create a gap between
records and events in the list.

Examples.

DEMO6.DAT and DEMO7.DAT in the 903ALGOL and MASDALGOL
directories demonstrate the use of ESP, including manipulating
histograms, sampling distributions and a simple event based
simulation of a queuing system. Note that both load the ESP
code procedures as a library.