Montecode—an interpretive program for Monte Carlo simulations

By D. H. Kelley* and J. N. Buxton†

MONTECODE is an interpretive program that has been written for the Ferranti Pegasus 1 computer and is in many respects similar to the normal Pegasus Autocode. It was designed in an attempt to speed up the time-consuming and complicated task of preparing Monte Carlo simulation programs.

MONTECODE has been in regular use now for over two years and has enabled Monte Carlo simulations to be written in one-tenth of the time previously taken. This advantage is partially offset by an increase in computer running time but an attractive net gain has usually been experienced in practice.

A worked example is given in the Appendix.

Introduction

During Operational Research studies it is frequently necessary to represent real life situations in the form of mathematical models or analogies. Such models, if sufficiently realistic, can be used to predict the effects of changes in the real life situation without making any, possibly expensive, experiments, even if such experiments are possible. Some such models can be satisfactorily explored by ordinary analytical methods, but in many cases the models need to be very complex in order to be realistic, and they defy solution in their analytical form. It is in cases like this that the technique of Monte Carlo simulation is resorted to (Neate and Dacey, 1959).

The essence of a simulation is that the state of each element of the model is examined at a given point in time, and any interaction between the elements resolved. Time is then advanced by a specified amount and the model is re-examined at the new point in time. This process is continued until the desired period of time has been simulated.

Simulations can be manually operated on the Gantt chart principle; a row on the chart would be allocated to each element of the model, e.g. a crane or a stock area, and using a horizontal time axis the state of each element at various times would be recorded. The chart then represents the complete operation of the model during the simulated period. Fig. 1 shows a simple example of this technique; the model represents a single-pump petrol filling station at which cars arrive at no set time interval. It has been assumed that it takes five minutes to service each and every car. This model enables the queue characteristics to be studied. In practice, this example could be examined by Queueing Theory unless the car arrival pattern had some peculiarities.

The method of determining car inter-arrival times and the service time, which would not be a constant in a realistic model, is a distinguishing feature of Monte Carlo simulations. It is not realistic to take mean or extreme values for these times, and a probability distribution of times must be determined for each independent variable. Once these distributions have been obtained a histogram approximation is usually constructed of, for example, 100 elements, see Fig. 2. Note that times with a higher probability, e.g. 3·5 minutes, are represented by a higher number of elements. Whenever a time is required during the simulation, an element will be selected from the histogram by random sampling, and the time it represents will be used.

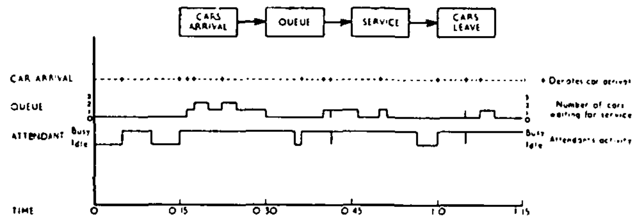


Fig. 1.—Gantt chart simulation of a petrol service station

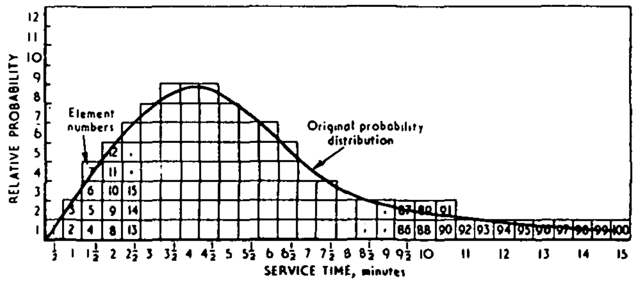


Fig. 2.—Service time histogram

A Monte Carlo simulation performed using this sampling technique may not represent any actually observed period of time, but it would represent a period that might arise at any instant and can therefore be described as typical.

The advantages of Monte Carlo simulations are

(i) that complex situations, that otherwise defy analysis, can be studied,

* Head of Systems Evaluation Section, Operational Research Department, British Iron and Steel Research Association.
† Formerly of the British Iron and Steel Research Association, now Applied Science Department. I.B.M. (United Kingdom) Ltd.

(ii) that changes to operating rules can be made quite easily,

(iii) that, consequently, predictions can be made without actually altering the real life situation, and the predictions can cover many years in advance of real time.

The effort required to complete a simulation study can be prohibitive if graphical methods are employed, and within recent years such analyses have been carried out on digital computers. However, the advantages of using a computer, i.e. its meticulous operation of complex models and its great speed, can only be realized once a program has been written and this task, although smaller than the task of manually operating a simulation, is not without its difficulties. It is not uncommon to find a simulation taking many programmer-months to program. Consequently, it has become worth while to consider interpretive and compiler-type programs written especially to assist in the preparation of Monte Carlo simulation programs. MONTECODE is one such program of the interpretive class.

Computer Simulation Programs

In its simplest form a computer simulation is identical to the Gantt chart form except that each element or row is represented by one or more computer storage locations, and the model within the machine only represents a single instant in time. The rules of operation are provided in the form of a computer program, and the results of the simulation are either printed out after each time interval has been completed or accumulated in locations allocated for that purpose. In the simple example shown in Fig. 1, a histogram of the number of cars queueing could be formed.

Computer simulations can be written in two basic styles, the *unit time scan* style or the *event to event* style. In the first style, the simulation is scanned or examined every unit of time, e.g. every minute of simulated time, whether or not any action, such as the arrival of a car, is due to occur. In the second style, time is advanced by the amount necessary to cause the next possible event to take place. This latter method saves computer running time when the simulation is static for periods of simulated time. Montecode has been designed primarily for event to event simulations, although unit-scan simulations can be accommodated.

The Design of MONTECODE

The aim of Montecode is to facilitate programming of a simulation, and this has been achieved in two main ways. In the first place, in addition to the normal input, output, arithmetic, and logical facilities expected in an autocode, Montecode provides several facilities frequently required during simulations, for example, random sampling from distributions, the maintenance of queues, and the formation of results in histogram form. Secondly, the organization of event to event scanning is automatically provided. The basic language used in Montecode is very similar to that used in the Pegasus Autocode, to facilitate learning and to avoid confusion since most Montecode programmers will be familiar with the Autocode.

Table 1

MONTECODE Storage Facilities

SINGLE LOCATIONS

Each location can store -2^{25} to $+2^{25}-1$

$\nu 0$, $\nu 1$, $\nu 2$, $\nu 3$, .	Usually allocated to action times
. . . . $\nu 239$	Normal single number storage

INDICES OR COUNTERS

$\eta 0$, $\eta 1$, $\eta 2$, ... $\eta 29$	Each index can store -8192 to $+8191$

PROGRAM STEPS

Maximum of 250 instructions.

DISTRIBUTIONS

D1, D2, . . . Dn	Maximum number depends on size

STORES OR QUEUES

S1, S2, ... Sn	Maximum number depends on size

HISTOGRAMS

H1, H2, . . . Hn	Maximum of 7 histograms

N.B.—The maximum number of storage locations, program steps, etc., can be increased at the expense of other facilities.

The controlling elements in a simulation can be called "action times"; they indicate how long, in simulated time, it will be before one of the important events occurs—e.g. a car arriving at the petrol station. Time in a simulation is measured relative to the present instant; i.e. a time zero means "now." When an action becomes zero, an "event" occurs. There is an event associated with each action time.

In the concept of Montecode a number of locations $\nu 1$ onwards are set aside for action times; there are a maximum of 240 such locations, and those not required for action times may be utilized as normal locations. Action times must, however, be stored consecutively. All locations will store only integers. Montecode also provides storage facilities for distributions, stores or queues, and histograms; these storage areas are referred to simply as D1, D2. . . ., S1, S2, . . ., H1, H2, ... H7 (see Table 1).

The model is set up in the computer by instructions such as the following.

READ D1, 500	which reads in from paper tape and stores a distribution of 500 elements.
PREPARE H1, 1, 9	which prepares the storage for a histogram of cell width 9 and lower limit 1.
STORAGE S1, 20	which prepares for the storage of 20 locations capacity.
CLEAR $\nu 1$, 10	which sets equal to zero ten locations starting at $\nu 1$.
FILL $\nu 20$, 5	which sets five locations starting at $\nu 20$ equal to the largest possible number +33554431.
$\left.\begin{array}{l}\nu 300 = +\ 197 \\ \eta 3 = -\ \ 10\end{array}\right\}$	which sets specified locations to given valves.

The actual running of a Montecode program, once the model has been defined, takes place in two phases. In phase 1 the order

SCAN 10

causes ten action times starting at $\nu 1$ to be scanned, and the simulation time advanced so that the next possible event will take place. In phase 2 the events corresponding to action times that are now zero or negative will be examined by programmed subroutines. The subroutines are labelled 1), 2), etc., corresponding to $\nu 1$, $\nu 2$, etc. When a subroutine has been completed a return is automatically made from the subroutine to the scanning mechanism. The process is then repeated until the desired simulated time has elapsed.

Examples of the types of instruction employed in the subroutines are as follows.

$\nu 10$ = SAMPLE D1, 5	which causes a value to be randomly selected from distribution 1, using random-number sequence 5.
TEXT T1	which prints preset text number 1.
COMPILE $\nu 27$, H3	which adds 1 to a cell in histogram 3 according to the contents of location 27.
REMOVE $\nu 24$, S2, *e*	which removes an item from store 2 and places it in location 24. The value of *e* determines whether the maximum value, minimum value, last in value or first in value is withdrawn from the store.

In addition instructions are available such as

$\nu 10 = \nu 20/\nu 30$	Division.
PRINT $\nu 100$	Print value of location 100.
$n1 = n1 - 2$	Subtraction.
$\rightarrow 10,\ n3 \neq 0$	Test contents of index 3.
$\nu 30$ = TAPE	Read an integer from tape.
HSTEST 10, 3	Test Handswitch 3.

Experience in the use of MONTECODE

The main measure of success for the Montecode program is its frequency of use; since it became available over two years ago all the Monte Carlo simulations that have arisen at BISRA have been programmed using Montecode. Furthermore, although facilities exist that enable machine-language interludes to be included they have rarely been used; this suggests that the facilities embodied in Montecode adequately cover simulation programming requirements.

It has proved to be extremely easy to teach people to program in Montecode, and non-programmers have managed to prepare short simulation programs on the first day of a Montecode course. The fact that non-specialist programmers can directly use Montecode is particularly useful since Monte Carlo simulations can be extremely complex and cannot be defined as precisely, in mathematical terms, as most other forms of numerical analysis. A major source of errors, i.e. communication of the requirements between analyst and programmer, is elminated when the analyst can prepare his own programs. Interpretation of results has been found to be much simpler with the Montecode facilities, which enable the result to be produced in the form really required by the analyst.

There have been minor complaints by Montecode users, about the specific nature of some of the instructions; for example, an output histogram must have 14 cells and on occasions more or less would have been useful.

The designer of a program of this nature must compromise between complete generality and undue complexity. However, there is often a way out and in this case, for example, two histograms can be used to give effectively a 28-cell histogram.

A Montecode simulation can be prepared in less than 10% of the time formerly taken to prepare a machine-language simulation. The speed of operation, however, is only about a third of that for a machine-language program, mainly due to the repeated necessity of transfers between the two levels of storage. However, as program development time is now virtually eliminated, the overall machine running time is only increased by a factor of two or so. This increase is more than adequately compensated for by the fact that results can be obtained much more quickly and with much less effort in preparation time.

Montecode is often used as an alternative to the Pegasus Autocode, owing to the greater speed, when the trigonometric functions are not required and when the calculations only involve integers. One frequent use of Montecode is in the analysis of data when simple arithmetic and histogram facilities are all that are required.

Conclusions

The Montecode interpretive program has been successfully used to reduce the time between the specification of a Monte Carlo simulation and the production of results. The initial effort required to produce the Montecode program has been repaid manyfold during the past two years. Other Pegasus owners and users have also used the program profitably.

In view of the loss of computer running speed some of the larger simulations tend to take up a lot of computer time to simulate a satisfactory length of simulated time. The Operational Research Department of the United Steel Companies Ltd. have tackled the problems that initiated the design of Montecode in a different manner, and a compiler-type program, the General Simulation Program (GSP) has been developed (Tocher and Owen, 1960). Although the GSP system is not so easy to learn, and the simulation preparation time takes a little longer than for Montecode, the program running speed is far superior. Consequently, the GSP is to be preferred for large simulations.

Appendix

Worked Example—A Doctor's Waiting-room

The situation simulated in this example is the operation of an appointment system in a doctor's waiting-room. Patients who wish to see the doctor are assigned appointments at 5-minute intervals throughout the surgery hours, with the exception that after every *n*th patient there is a ten-minute interval. The time the doctor spends with each patient varies from 1 to 20 minutes, and the mean consulting time is about 6 minutes.

The aim of the simulation is to establish the value of *n* which gives an acceptably small average idle time during the surgery for the doctor, while avoiding unnecessarily long queues in the waiting-room.

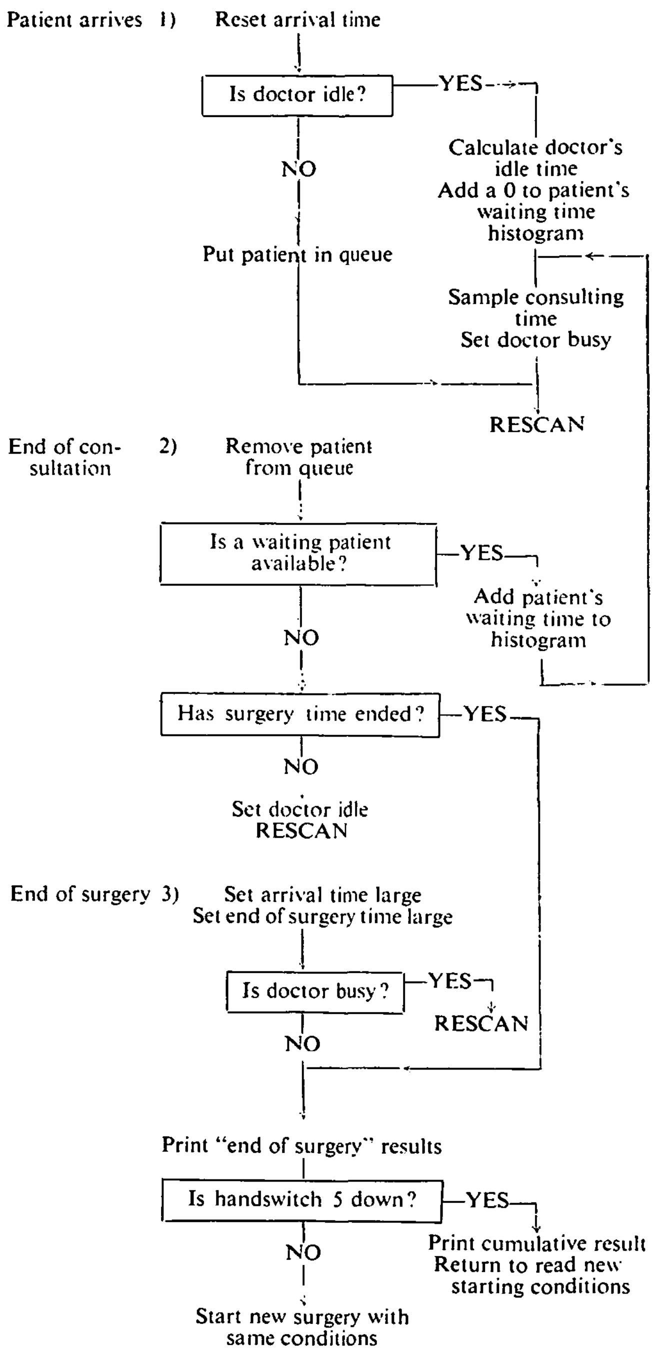
The flow diagram, program instructions and explanatory notes are given in Fig. 3 and Table 2. A specimen set of results is given in Table 3.

The program took about two hours to write and a further half hour to test. Simulation of one surgery takes about 40 seconds, not including optional printing.

It will be seen that there are three action times in this simulation; arrival of patient, end of consultation, and end of surgery. Some interplay takes place between the orders in each subroutine; for example, when a consultation ends, the store used as a queue is searched to find if a patient is available and, if this condition is satisfied, a consulting time is sampled. If nothing is in the store, then the remove order extracts a zero. This section of the subroutine may also be entered directly from the patients' arrival subroutine if the doctor is idle; i.e. the patient does not queue but is seen immediately.

Different values of *n* were inserted into the simulation, and $n = 4$ was found to give the best compromise between doctor's idle time and patients' queueing time, for the consulting time distribution obtained from the doctor studied in this example.

In this example, after runs of 20 surgeries the average



Patient arrives 1) Reset arrival time
Is doctor idle?
YES
NO
Calculate doctor's idle time
Add a 0 to patient's waiting time histogram
Put patient in queue
Sample consulting time
Set doctor busy
RESCAN
End of consultation 2) Remove patient from queue
Is a waiting patient available?
YES
Add patient's waiting time to histogram
NO
Has surgery time ended?
YES
NO
Set doctor idle
RESCAN
End of surgery 3) Set arrival time large
Set end of surgery time large
Is doctor busy?
YES
RESCAN
NO
Print "end of surgery" results
Is handswitch 5 down?
YES
Print cumulative result
Return to read new starting conditions
NO
Start new surgery with same conditions

Fig. 3.—Doctor's waiting-room flow diagram of action subroutines

queueing time per patient was about 10 minutes, and the doctor's idle time average was about 7 minutes per complete surgery.

This is, in fact, an example which is based on a practical case; the appointment method suggested by the simulation is now being used in general practice and shows results which confirm those obtained from the simulation.

Table 2

Doctor's Waiting-room Program

v1	= arrival time	v8	= working space
v2	= end consultation	v9	= cumulative waiting time
v3	= end surgery	v10	= number of surgeries
v4	= clock time	v11	= cumulative overtime
v5	= doctor busy marker		
v6	= doctor's waiting time	n1	= No. of 5 min. appointments
v7	= 0	n2	= 5 min. appointment counter

```
N
DOCTOR'S WAITING-ROOM

J1.0
READ D0, 996, 1.
TEXT IN T0
TEXT IN T1
TEXT IN T2              Initial input instructions
TEXT IN T3
TEXT IN T4
TEXT IN T5
STORAGE S0, 10        }
PREPARE H0, 1, 3      }
PREPARE H1, 1, 3      } Initial preparation instructions
PREPARE H2, 1, 3      }

                        Main Program
12) CLEAR v9, 3         Re-entry point; clear cumulative results
n1 = HSREAD.            Read new value for n1
11) n2 = n1             Set n2
CLEAR v1, 8.            Clear surgery results
v10 = v10 + 1           Increase number of surgeries
v3 = 121                Set initial conditions
v1 = 1
v5 = -1
SCAN 3, 1
UPDATE S0
RESCAN

                        Patient arrives
1) v1 = 5             }
n2 = n2 - 1           }
→14, n2 ≠ 0           } Set new arrival time to 5 or 10 minutes
n2 = n1               }
v1 = 10               }
14) → 4, v5 ≠ 0         Jump to 4 if doctor idle
ASTORE v7, S0           Store patient if doctor busy and exit
→6
4) v8 = v5 - v4         Calculate and accumulate doctor's
v6 = v6 + v8              waiting time
COMPILE v7, H0          Add a patient who has waited time zero
                          to H0
5)v2 = SAMPLE D0        Sample consulting time
OPRINTX v2, 2
v5 = 0                  Set doctor busy
6) RESCAN

                        End of consultation
2) AREMOVE v8, S0, 1    Remove patient from queue
OPRINTX v8, 6
→7, v8 = 0              Jump to 7 if queue empty
COMPILE v8, H0          Otherwise, add patient's waiting time to H0
→5                      and jump to 5
7)→8, v3 > 10000        Jump to 8 if surgery time exceeded
v5 = v4                 Set doctor waiting
RESCAN

                        End of surgery
3) v1 = 10000           Prevent further arrivals
v3 = 100000             Set v3 large to use as marker
→6, v5 = 0              Rescan if doctor busy, or run on to print
                          results
```

```
8) v9 = v9 + v6         Print Results
TEXT T0
PRINT v6, 2             Print doctor's waiting time
v8 = - 121 - v4         Calculate and print overtime
TEXT T5
PRINT v8, 2
v11 = v11 + v8
HSTEST 13, 3
→9
13) TEXT T1             Print queueing time histogram
HPRINT H0, 4              for this surgery if H.S.3 down
9) TOTAL H0, H1
HSTEST 10, 5
→11                     Restart next surgery if H.S.5 up
10) TEXT T2             Print cumulative results if
PRINT n1                H.S.5 down
TEXT T3
TEXT T0
TEXT T4
PRINT v9, v10, 1
TEXT T5
TEXT T4
PRINT v11, v10, 1
TEXT T1
HPRINT H1
TOTAL H1, H2
STOP                    Stop, and return to read new value for n1
→12
*→0                     Order to start operation of program at 0

DOCTOR'S WAITING TIME
PATIENTS' QUEUEING TIMES          Texts

CUMULATIVE RESULTS, EVERYTH APPT. SET TO 10 MINS
(AVERAGE) DOCTOR'S OVERTIME

n = 996                 Consulting Time Distribution
 93 × 1
 97 × 2                   (tape in reader 1)
113 × 3
115 × 4
108 × 5
 96 × 6
 81 × 7
 68 × 8
 49 × 9
 47 × 10
 33 × 11
 25 × 12
 20 × 13
 14 × 14
 10 × 15
  8 × 16
  5 × 17
  4 × 18
  4 × 19
  2 × 20
  4 × 22
*
```

Table 3

Doctor's Waiting-room Results

```
DOCTOR'S WAITING TIME  + 0  }  Results of first surgery
DOCTOR'S OVERTIME   − 19    }
1                              Optional printing for second surgery
0 − 826                        H.S.0 up, H.S.1 and H.S.2 down
 + 2
2
      + 0
1                              Label print
0 + 714                        Random number (stream 0)
 + 3                           Sample consulting time
2
      + 0                      Patients  queueing time
1
0 + 233                        H.S.2 up
      + 8
1
2
      + 3
 + 9
1
2                              H.S.1 up
      + 7
 + 3
      + 0
 + 9
      + 4
 + 6                           H.S.0 down; all optional printing suppressed
DOCTOR'S WAITING TIME  + 9  }  Second surgery results
DOCTOR'S OVERTIME  + 6      }
DOCTOR'S WAITING TIME  + 23 }  Third surgery results
DOCTOR'S OVERTIME  + 9      }
DOCTOR'S WAITING TIME  + 15 }  Fourth surgery results
DOCTOR'S OVERTIME  + 3      }
CUMULATIVE RESULTS, EVERY  + 5TH APPT. SET TO 10 MINS.
DOCTOR'S WAITING TIME (AVERAGE)   + 11·7
DOCTOR'S OVERTIME (AVERAGE)   + 9·2
PATIENTS' QUEUEING TIMES

 80 TERMS
ACTUAL MEAN  + 6·400   VARIANCE    + 45·51
HISTO. MEAN  + 9·556   VARIANCE    + 39·91
 CORR. −0·75
  RANGE        FREQ.   0         10        20        30        40   0/0
   →       0    26     1---------1---------1---------1---------1--------   *
  1→       3     9     1---------1-------*
  4→       6    16     1---------1---------1---------*
  7→       9     4     1------*
 10→      12     9     1---------1-------*
 13→      15     4     1------*
 16→      18     6     1---------1*
 19→      21     3     1-----*
 22→      24     3     1-----*
 25→      27     0     *
 28→      30     0     *
 31→      33     0     *
 34→      36     0     *
 37→      39     0     *
 40→      42     0     *
  ≥       43     0     *
```

References

Neate, R., and Dacey, W. J. (1959). "A Simulation of Melting Shop Operations," *The Computer Journal*, Vol. 2, p. 59.

Tocher, K. D., and Owen, D. G. (1960). "The Automatic Programming of Simulations," *Proceedings of the Second International Conference on Operational Research*, p. 50.