Simulation

Henning Schulzrinne
GMD Fokus, Berlin
hgs@fokus.gmd.de

June/July, 1994

Overview

- Definition
- Motivation
- Types of Simulation
- Discrete Event Simulation
- Languages and Tools
- Random Numbers
- Statistical Methods
- References

Definition

Simulation means driving a model of a system with suitable inputs and observing the corresponding outputs. [BFS87]

Applications:

- systems that cannot be built, tested and fail (air craft, communication networks, ...)
- training (flight simulator)
- systems that are beyond analysis
- "verification" of analysis

Applications of Simulation

- physical systems: climate, chemical reactions, …
- manufacturing systems
- traffic flow
- computer systems
- computer networks
- Monte-Carlo simulation (numerical integration)

Disadvantages of Simulation

- hard to check for correctness
- cannot prove properties (“proof by example”)
- excessive resources for low-probability events ($< 10^{-6}$); need ≥ 100 events
- hard to generate lots of cases
- lots of numbers $\rightarrow$ false security
- random number generators

Simulation in Performance Evaluation

Some systems are not (usually) analyzable; you are in trouble with …

- non-FIFO resource access
- dependencies (between arrivals, service, …)
- buffer overflow systems
- non-Markovian input or service
- queueing networks with non-probabilistic routing
- media access protocols

Types of Simulation

static: no notion of time → Monte-Carlo simulation

continuous time: differential equations (chemical processes)

discrete event (DES): networks, computer systems, …

asynchronous: time is real variable

+ most common/general form of simulation
– can be slow if many events pending
– can't do *wait-until condition* easily

synchronous: time is integer variable

+ slotted systems (ATM networks)
+ clocked systems (digital logic simulation)
+ faster (no event list)
– less general than asynchronous

- difficulty: multiple clocks (e.g., different ATM link speeds)
- difficulty: simultaneous events

→ will only treat asynchronous DES

finite horizon: natural ending time, e.g., work day, product life, …

steady state: fiction; statistical measures independent of starting condition; most real systems are not stationary; ending time?

Keynes: "In the long run we are all dead."

Components of DES

"random inputs" $\rightarrow$ events $\rightarrow$ samples $\rightarrow$ statistical measures

random number generator: source of (usually uniformly distributed) random variates

"true": white noise (resistor), nuclear decay, …$\rightarrow$ not reproducible

pseudo: mathematically generated sequence of numbers that have properties like "true" random numbers

event list: maintains time-ordered list of future events; events advance simulation clock

samples: interarrival times, delays, queue occupancy, …

statistical measures: describe behavior

Components: Statistical Measures

- mean
- higher-order moments (variance, coefficient of variation, …); higher order $\rightarrow$ more samples
- distribution (histogram)
- auto-correlation
- confidence intervals

Checking Simulations

- "back-of-envelope" calculations
- extreme cases (buffer size $= \infty$, Poisson arrivals, …)
- consistency checks:
  - range of random variables
  - statistics of source random variables
  - "lost" events, tokens, resources, …

Programming Simulations

- simulation can be programmed in Fortran, C, Pascal, Lisp, …
- repeat lots of common code: random number generation, statistics, event list handling, …
- every university their own simulation system
- → simulation toolkits and languages
  - memory allocation/deallocation (no explicit `malloc`)
  - run-time library for statistics, random number generation
  - optimized event list handling
  - error detection
- "faster to program, slower to run"

- but: more carefully tuned common parts
- rough classification:

GPL extensions: extensions of general-purpose languages (C, Pascal) through libraries; example: SIMUL

task-specific: LANs (LANSF), communication systems (Opnet, CLASS): provide high-level functionality (Ethernet module), hard to extend

simulation languages: special compiler, possibly incorporates GPL

Simulation Languages

event oriented: routine directly dispatches on events; creates new events
→ LANSF, SIMUL

process oriented: communicating sequential processes → CSIM

transaction oriented: transactions and actions → SES/Workbench

SIMUL: Event Oriented

- variable `clock` tracks simulated time
- `schedule(func, e, dt)` creates new event `dt` time units from current time, executes `func()`
- `ex()` generates exponential r.v.
- `create_()/dispose()` creates new customer record
- `insert()` appends to queue
- `fremove(lfr())` removes head (first) of queue
- messy with several queues; easy to lose customers

SIMUL: Event Oriented

```
/*
* SIMUL M/M/1 example
*/

/*
* Arrival handler.
*/
void arrival(entity e)
{
  e = create_();
  schedule(arrival, e, ex(1.0));
  insert(q, e);
  /* only one in queue -> create departure after service */
  if (NQ(q) == 1) schedule(departure, e, ex(0.5));
} /* arrival */
```

```
void departure(entity e)
{
  e = fremove(lfr(q));
  /* schedule next departure from queue */
  if (NQ(q) > 0) schedule(departure, ex(0.5));
  dispose(e);
} /* departure */


main(int argc, char *argv[])
{
  queue_t q;      /* queue; first customer is in service */

  q = queue_create();
  schedule(arrival, 0, 0.0);
}
```

CSIM: Process Oriented

- C with additional functions
- “function call” starts new process (different semantics!)
- like a simulated parallel processor
- main routine `sim()`

CSIM

Process can …

- execute function as process: `create`
- wait for simulated time to elapse: `hold`
- wait for event to occur: `wait`
- cause an event to occur: `set`
- exclusive resource use: `reserve`, `release`
- request shared resource use: `use`

CSIM Example

```
/*
* simulate an M/M/1 queue
*/

#include <csim.h>

#define SVTM  1.0    /* mean of service time */
#define IATM  2.0    /* mean of interarrival time */
#define NARS  5000   /* number of arrivals to be simulated */

FACILITY f;       /* pointer for facility */
EVENT done;       /* pointer for counter */
TABLE tbl;        /* pointer for table */
QTABLE qtbl;      /* pointer for qhistogram */
int cnt;          /* number of active tasks */
```

```
/* generate customers */
sim()              /* 1st process - must be named sim */
{
  int i;

  set_model_name("M/M/1 Queue");
  create("sim");                 /* required create statement */

  f = facility("facility");             /* init. facility */
  done = event("done");                 /* init. event */
  tbl = table("resp tms");              /* init. table */
  qtbl = qhistogram("num in sys", 10); /* init. qhistogram */

  cnt = NARS;                    /* initialize count */
  for (i = 1; i <= NARS; i++) {
    hold(expntl(IATM));          /* hold interarrival */
    cust();                      /* initiate process cust */
  }
```

```
  wait(done);                        /* wait until all done */
  report();                          /* print report */
  mdlstat();
}

cust()          /* process arriving customer */
{
  float t1;

  create("cust");                  /* required create statement */

  t1 = clock;                      /* time of request */
  note_entry(qtbl);                /* note arrival */
  reserve(f);                      /* reserve facility f */
  hold(expntl(SVTM));              /* hold service time */
  release(f);                      /* release facility f */
  record(clock-t1, tbl);           /* record response time*/
  note_exit(qtbl);                 /* note departure */
```

```
  cnt--;                         /* decrement cnt */
  if (cnt == 0)
    set(done);                   /* if last arrival, signal */
}
```

SES Workbench: Transaction Oriented

- hierarchy of extended directed graphs
- transactions flow
- graphical definition, with extensions through C++
- no built-in support for confidence intervals

Statistical Methods

- random-number generation
- transient period
- confidence intervals
- optimization
- variance reduction
- experimental design
- rare-event simulation

Pseudo-Random Numbers

- look like random numbers, act like random numbers, but repeatable
- cyclical: should cycle through (close to) all 2^N possibilities
- most common operation $\rightarrow$ need to be fast
- low bits may not be random
- critical: good choice of initial seed to avoid correlated streams (do *not* pick clock; pick separated list)
- random numbers for encryption are different

Random Number Generators

- generate random integer uniformly distributed between 1 and m
- translate to interval $(0, 1)$ (approximation!)
- use one or more to generate desired distribution
- types of generators:
  - additive congruential
  - linear congruential $\in$ multiplicative
  - Tausworthe
  - composite

Additive Congruential Generator

$$X_i = (X_{i-1} + X_{i-2}) \bmod m$$

Bad:

- $X_{i-1} < X_{i+1} < X_i$ and $X_i < X_{i+1} < X_{i-1}$ never appear (should have probability 1/6)!!
- serial correlation: $m = 1000, X_0 = 1, X_1 = 1 \rightarrow 2, 3, 5, 8, 13, 21,$ $34, \ldots$

Linear Congruential Generators

$$X_i = (aX_{i-1} + c) \bmod m$$

- $c = 0$: multiplicative congruential generator
- $c = 0, m = 2^{31} - 1, a = 16807 = 7^5$ or $a = 630360016$
- $(4096x_{n-1} + 150889) \bmod 714025$ [PFTV86]
- $(314,159,269x_{n-1} + 453,806,245) \bmod 2^{31}$

Full-cycle (all numbers between 0 and $m - 1$) iff:

1. c is relative prime to m
2. If q is a prime or 4 that divides m, then q divides $a - 1$

LCG Implementation

On a 32-bit machine (need to avoid arithmetic overflow):

$$
\begin{aligned}
a &= 16807 \\
b &= 127773 \\
c &= 2836 \\
P &= 2^{31} - 1 \\
k &= \lfloor x_{n-1}/b \rfloor \\
x_n &= a(x_{n-1} \bmod b) - kc \text{(subtract and check)} \\
u_n &= x_n \times 1/P
\end{aligned}
$$

(Bratley, p. 214)

Tausworthe Generator

generate bits b_i recursively:

$$b_i = (b_{i-r} + b_{i-q}) \bmod 2$$

Example: $b_1 = \ldots = b_5 = 1$; r=3; q=5
$\rightarrow$ 111110001101110101000010010110011111000110 $\rightarrow$ string together bits, possibly skipping some

- long periods
- cryptographically secure
- but not completely understood

Composite Generators

- combine several generators $\stackrel{?}{\rightarrow}$ better properties
- certainly more expensive
- examples:
  - use LCG to shuffle other LCG $\rightarrow$ extend period, lower autocorrelation
  - bit-shift and exclusive-or
- generally improve upon poor LCG, but not (much) on good ones

Tests for Random Numbers

- mean, variance, higher moments, …
- Chi-square test for distribution fit (histogram)
- spread as N-dimensional vector
- runs up-and-down (normally distributed)
- runs above and below the median
- permutation tests
- autocorrelation

χ^2 Test for Goodness of Fit

- compute histogram with k cells
- o_i : observed frequency, e_i : expected frequency

$$D = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i}$$

- assure $e_i > 5$, equiprobable cells
- if exact fit, $D = 0$
- D has χ^2 distribution with $k - 1$ degrees of freedom
- degrees of freedom = cell count $-$ independent linear restrictions on cell counts (e.g., total sum, mean estimation)

- "The null hypothesis that the observations come from the specified distribution cannot be rejected at a level of significance α if the computed D is less than the $\chi^2_{1-\alpha;k-1}$ value" (from table)

Generating RVs: Inversion Method

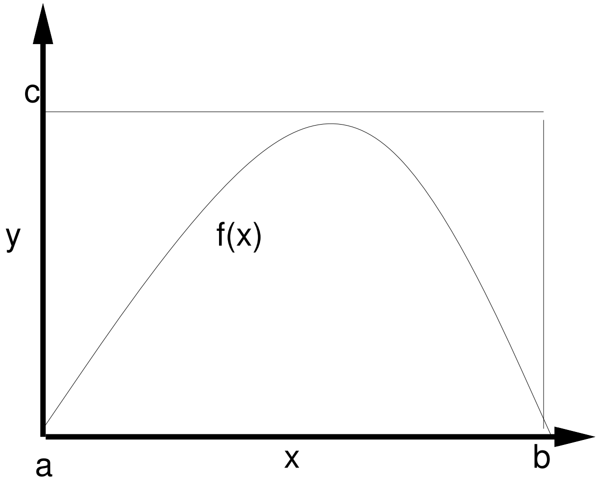
For continuous random variables with invertible distribution function:

1. Generate uniform variate $U \sim U(0,1)$
2. Return $X = F^{-1}(U)$

Proof: $P\{X \leq x\} = P\{F^{-1}(U) \leq x\} = P\{U \leq F(x)\} = F(x)$ (apply invertible $F(x)$ to both sides; definition of uniform distribution)

Generating RVs: Acceptance-Rejection Method

If no closed-form cdf is known, but pdf f bounded and non-zero on $[a, b]$:



1. compute $c = \max(f(x) | a \leq x \leq b)$
2. generate X uniform on (a, b)
3. generate Y uniform on $(0, c) \rightarrow$ point (X, Y)
4. if $Y \leq f(X)$, then output X; otherwise go to step 2 $\rightarrow$ if point below curve, accept

Works well only if rejection ratio is small!

Generating RVs: Generalized Rejection

Compute majorizing density g

1. generate X with the pdf g
2. generate Y uniform on $[0, ag(X)]$
3. if $Y \leq f(X)$, then output X; otherwise go to step 1

or

1. generate X with the pdf g
2. generate U uniform on $(0, 1)$
3. accept X if $f(X)/ag(X) \geq U$

Proof: Law/Kelton, p. 273

Aside: Independent vs. Uncorrelated

independent: random variables X, Y or events A, B are independent if

- $P(X = x, Y = y) = P(X = x)P(Y = y)$
- $f(x, y) = f_1(x)f_2(y)$
- $F(x, y) = F_1(x)F_2(y)$.
- $P(A|B) = P(A)$

uncorrelated: weaker:

$$\rho = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{E(X - \mu_X)^2 E(Y - \mu_Y)^2}$$

- independent $\rightarrow$ uncorrelated; uncorrelated $\not\rightarrow$ independent
- normal RVs: independent $\leftrightarrow$ uncorrelated [Pap84, p. 153]
- correlated $\not\rightarrow$ causality

Aside: Memoryless

Definition 1 *A distribution is memoryless if remembering the time since the last event does not help in predicting the time until the next event. The future is independent of the past.*

- exponential distribution is only memoryless continuous distribution

No arrival seen until $t = t_0$. Time t remaining until arrival:

$$\begin{aligned}
P[\tau \leq t + t_0 | \tau > t_0] &= \frac{P[t_0 < \tau \leq t_0 + t]}{P[\tau > t_0]} \\
&= \frac{P[\tau \leq t + t_0] - P[\tau \leq t_0]}{P[\tau > t_0]} \\
&= \frac{1 - e^{-\lambda(t+t_0)} - (1 - e^{-\lambda t_0})}{1 - (1 - e^{-\lambda t_0})} \\
&= 1 - e^{-\lambda t}
\end{aligned}$$

$\rightarrow$ same as unconditional density

Continuous: Exponential Distribution

density: $f(x) = \lambda e^{-\lambda x} \forall x \geq 0$

expectation: $E(x) = \frac{1}{\lambda}$

variance: $\mathrm{Var}(x) = \frac{1}{\lambda^2}$

Coefficient of variation: 1

application: memoryless; Poisson sources

generation: inversion: $X = -\log(1-U)/\lambda$

Continuous: Hyperexponential Distribution

$\equiv$ random selection of one of n exponential variates

density: $f(x) = \sum_{i=1}^{n} p_i \lambda_i e^{-\lambda_i x} \forall x \geq 0$

expectation: $E(x) = \sum_{i=1}^{n} \frac{p_i}{\lambda_i}$

variance: $\mathrm{Var}(x) = \sum_{i=1}^{n} \frac{p_i}{\lambda_i^2}$

Coefficient of variation: > 1

application: high-variance events

generation: generate U_1 and find k such that

$$\sum_{i=1}^{k-1} p_i < U_1 < \sum_{i=1}^{k} p_i$$

Output

$$X = -\log(U_2)/\lambda_k$$

Continuous: Erlang Distribution

Sum of k iid exponential variates (Gamma: k non-integer); $k \to \infty$: uniform

density: $f(x) = \frac{\lambda e^{-\lambda x}(\lambda x)^{k-1}}{(k-1)!} \forall x \geq 0$

expectation: $E(x) = k/\lambda$

variance: $\mathrm{Var}(x) = k/\lambda^2$

Coefficient of variation: < 1

application: low-variance events

generation:

$$X = -\sum_{i=1}^{k} \log(U_i)/\lambda_i$$

D: Binomial Distribution

Number of successes in n trials, with p per-trial success probability

density: $f(x) = \frac{n!}{(n-x)!x!} p^x (1-p)^{n-x}$

expectation: $E(x) = np$

variance: $\mathrm{Var}(x) = np(1-p)$

generation: for small n:

1. Set X to 0.
2. Do the following n times: Generate U uniform on $(0, 1)$; if $U \leq p$, increment X.

D: Geometric Distribution

discrete analog of exponential (memoryless)

density: $f(x) = (1-p)^x p$

expectation: $E(x) = (1-p)/p$

variance: $\mathrm{Var}(x) = (1-p)/p^2$

generation: for larger p:

1. Set X to -1.
2. Repeat until $U \leq p$: Increment X. Generate U uniform on $(0, 1)$.

or $X = \lfloor \log(1-U)/\log(1-p) \rfloor$

Statistical Estimation

N samples, $x_1 \ldots x_N$

- mean

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

- variance

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} [x_i - \bar{x}]^2$$

Warning: estimate *biased* if x_i correlated.

- squared coefficient of variation

$$c = \frac{s^2}{\bar{x}^2}$$

Bias: $E(\text{estimate}) \neq$ true value

Estimation of Transient Period

- aka: initial bias, warm-up period
- recall: not constant value, but constant expectation
- option: include or exclude initial set of measurements

Methods [Jai91, p. 425]

long runs: hope for the best… → may be reasonable if rare events drive simulation length

proper initialization: start in close to expected steady state (but often: most likely state (0 for M/M/1) $\neq$ average state)

truncation: remove initial samples (higher variance); cut out $x_1 \ldots x_l$ observations until x_{l+1} is neither minimum nor maximum of $x_{l+1} \ldots x_n$. Need to store whole sample trace!

Estimation of Transient Period, cont'd.

initial data deletion: Perform m replications of n samples. Get trajectory and plot relative change.

moving average: Perform m replications of n samples. Plot moving average of trajectory.

batch means: Divide long simulation run into *batches*; plot batch mean variance as function of batch size; pick point where variance starts decreasing ($\leftarrow$ only first batch is different)

CI Explanation

Assume:

$$\cdots$$

The variable

$$T = \frac{Z}{\sqrt{Y/(n-1)}}$$

is t-distributed with $n-1$ degrees of freedom.

CI: Example

Symbols:

s	sample estimate of standard deviation (root of variance)
$\bar{x}$, $\hat{\mu}$	sample estimate of mean
μ	population mean
σ	population s.d.
c	$\equiv t$ statistic

100 samples from normally distributed population. Get 99% confidence interval on the sample mean μ. Estimates: $\bar{x} = 1.944$, $s^2 = 0.00282$, $s = 0.053$.

Table for very large n

γ	0.90	0.95	0.99	0.999
c	1.645	1.960	2.576	3.291

Table for smaller n:

n	$c/\sqrt{n}$	
	$\gamma = 0.95$	$\gamma = 0.99$
5	1.240	2.060
10	0.715	1.030
20	0.468	0.640
50	0.284	0.379
100	0.198	$\boxed{0.263}$
200	0.139	0.184
500	0.088	0.116
1000	0.062	0.081

1. $\gamma = 0.99$
2. Since $n = 100$, $c/\sqrt{n} = 0.263$

3. width = $s \cdot c/\sqrt{n} = 0.053 \cdot 0.263 = 0.014$

4. CI: $1.930 \leq \mu \leq 1.958$

Note similarity to typical survey statistics…

Statistical Tests

Questions:

- "The distribution has the mean 17.9"?
- "Algorithm A (mean: 17) is faster than algorithm B (mean 17.1)"?

classical application: "statistical quality control"

Statistical Tests

Example (Mendenhall): "Jones claims that he will win more than 50% of the votes. 15 voters are sampled."

Null hypothesis: H_0, hypothesis to be tested ("Jones wins" $\rightarrow$
$H_0 : p = 0.5$ – minimum needed)

Alternative hypothesis: H_a, hypothesis to be accepted if H_0 is rejected
$\rightarrow H_0 : p < 0.5$

Test statistic: function of the sample measurements

Rejection/critical region: where H_0 is rejected

Statistical Tests: Errors

Type I error: rejection of H_0 when H_0 is true. Probability: α; example: "Jones will lose", but he wins.

Type II error: acceptance of H_0 when H_a is true. Probility: β; example: "Jones will win", but he looses.

Choose $C = 2$; binomial distribution.

$$\alpha = P(Y \in C; p = 0.5; n = 15) = 0.004$$

H_a is true:

$$\beta = P(Y \in \bar{C}; p = 0.3) = P(Y > 2; p = 0.30) = 0.873$$

Even if $p = 0.3$, we will declare Jones a winner.

Statistical Tests: Test of Mean

Hypotheses:

$$H_0 : \qquad \mu = \mu_0 \tag{1}$$

$$H_a : \qquad \mu > \mu_0 \tag{2}$$

$$\tag{3}$$

Test statistic (how many standard deviations away):

$$Z = \frac{\hat{\mu} - \hat{\mu_0}}{\sigma_\mu} \approx N(0,1)$$

Rejection region: $z > z_\alpha$, for fixed α

Statistical Tests: Large-Sample Example

$H_0 : \mu = 15$

$H_a : \mu > 15$

$z > 1.645$ for $\alpha = 0.05$

$n = 36$, $\hat{\sigma^2} = 9$

$\hat{\mu} = 17, \hat{\sigma} = 3/\sqrt{36} \rightarrow z = 4 \rightarrow z$ in rejection region

Statistical Tests: Two Distributions

$n_i, i \in 1, 2$ samples, μ_1 mean
Hypothesis: $\mu_1 = \mu_2$
Alternative: $\mu_1 > \mu_2$

1. choose α
2. compute rejection region z: $P(T \leq z) = 1 - \alpha$, with $n_1 + n_2 - 2$ degrees of freedom
3. compute $\hat{\mu}_i$
4. compute

$$t_0 = \sqrt{\frac{n_1 n_2 (n_1 + n_2 - 2)}{n_1 + n_2}} \frac{\hat{\mu}_1 - \hat{\mu}_2}{\sqrt{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}}$$

References

[BFS87] Paul Bratley, Bennett L. Fox, and Linus E. Schrage. *A Guide to Simulation*. Springer-Verlag, New York, New York, 2nd edition, 1987.

[Jai91] Raj Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley, New York, 1991.

[Kle87] Jack P. C. Kleijnen. *Statistical Tools for Simulation Practitioners*. Marcel Dekker, New York, New York, 1987.

[Kre68] E. Kreyszig. *Statistische Methoden und ihre Anwendungen*. Vandenhoek und Ruprecht, G ttingen, 1968.

[Lav83] Stephen S. Lavenberg, editor. *Computer Performance Modeling Handbook*. Notes and Reports in Computer Science and Applied Mathematics. Academic Press, New York, New York, 1983.

[LK82] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, New York, 1982.

[MS73] William Mendenhall and Richard L. Scheaffer. *Mathematical statistics with applications*. Duxbury Press, North Scituate, Massachusetts, 1973.

[Pap84] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill Book Company, New York, New York, 2nd edition, 1984.

[PFTV86] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes - The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, 1986.

[SC81] Charles H. Sauer and K. Mani Chandy. *Computer systems performance modeling*. Prentice-Hall, Princeton, New Jersey, 1981.