Bi7740: Scientific computing

Introduction to Monte Carlo methods

Vlad Popovici
`popovici@iba.muni.cz`

Institute of Biostatistics and Analyses
Masaryk University, Brno

Supplemental bibliography

- Gentle, J.E.: *Random number generation and Monte Carlo methods*. 2003. Springer. 2nd Ed.
- Jones O., Maillardet R., Robinson, A. *Scientific programming and simulation using R*. 2009., CRC Press.

Outline

Numerical experiments: simulations

General approach:

1. identify the random variable of interest X
2. identify/postulate its distributional properties
3. generate one or several *large* samples *identical and independely distributed* $X_1, \ldots, X_n$ from the distribution of X
4. estimate the quantity of interest (e.g. estimate $\mathbb{E}X$ using sample average) and assess its accuracy (e.g. via confidence intervals)

Random number generators (RNGs)

- all random variables can be generated by transforming a *uniformly distributed* random variable $X \in U(0,1)$
- there is no algorithmic (deterministic) way of generating infinitely long sequences of true random numbers
- computers generate *pseudorandom numbers*
- there exist devices to generate (believed to be) random sequences: e.g. radioactive decay: the time elapsed between emission of two consecutive particles (α, β, γ). See: `http://www.fourmilab.ch/hotbits`

RNGs, cont’d

- two aspects:
  1. generate *good* pseudorandom numbers in $U(0, 1)$: independent and uniformly distributed
  2. find proper trasformations to the desired distribution
- you cannot prove that an RNG is truly random
- there are a batteries of tests that an RNG must pass to be *acceptable*
- for any RNG, one can find a statistical test that will reject it as a good generator

RNGs, cont'd

Formalism:

- an RNG is a structure (S, μ, f, U, g) where
  - S is a finite set of *states*
  - μ is a probability distribution on S used to select the initial *seed (state)* s_0
  - $f : S \to S$ is a *transition function*. The state of the RNG evolves according to the recurrence $s_i = f(s_{i-1})$ for $i \geq 1$
  - U is the *output space*. Usually $U = (0, 1)$
  - $g : S \to U$ is the *output function*. The numbers $u_i = g(s_i)$ are called *random numbers* produced by the RNG

RNGs, cont'd

- S is finite $\Rightarrow \exists l \geq 0, j > 0$ finite such that $s_{l+j} = s_l$
- this implies that $\forall i \geq l$, $u_{i+j} = u_i$ since both f and g are deterministic
- the smallest positive j for which this happens is called *period lenght* of the RNG and is denoted by ρ
- obviously, $\rho \leq |S|$
- ex.: if the state is represented on k bits, then $\rho \leq 2^k$

RNGs, cont'd

Quality criteria:

- extremly long period ρ
- efficient implementation
- repeatability
- portability
- availability of jump-ahead property: quickly compute the s_{i+v} given s_i, so you can partition a long sequence in subsequences to be used in parallel
- *randomness*

RNGs, cont'd

Coverage:

- let $\Psi_t = \{(u_0, \ldots, u_t) | s_0 \in S\}$
- is Ψ_t uniformly covering the hypercube $(0, 1)^t$?
- tests of *discrepancy* between the empirical distribution of Ψ_t and the uniform distribution
- *figure of merit*: a measure of the coverage quality

RNGs, cont'd

Randomness and *i.i.d*:

- statistical tests: try to detect empirical evidence against H_0: "u_i are realizations of i.i.d $U(0, 1)$". Example: diehard tests (Marsaglia, 1995)
- passing more tests improves the confidence in RNG, but cannot *prove* the RNG is foolproof for all cases
- *good* RNG passes a set of simple tests
- *polynomial time perfect* RNG: there is no polynomial-time algorithm the can predict any given bit of u_i with a probability of success $\geq 1/2 + 2^{-k\epsilon}$, for some $\epsilon > 0$, after observing $u_0, \dots, u_{i-1}$
- the usual RNGs are not polynomial time perfect

RNGs, cont'd

Multiple Recursive Generator has a general recurrence

$$x_i = (a_1 x_{i-1} + \cdots + a_k x_{i-k}) \bmod m$$

where m (modulus) and k (order) are integers carefully selected, and coefficients $a_1, \ldots, a_k \in \mathbb{Z}_m$.
The state is $s_i = (x_{i-k+1}, \ldots, x_i)^T$.
When m is prime, it is possible to select a_i such that the period length $\rho = m^k - 1$.

RNGs, cont'd

Example (historical, not in serious use anymore): MLCG (Lehmer, 1948): multiplicative linear congruential generator:

$$s_{i+1} = (a_1 s_i + a_0) \bmod m$$

This generates integers that are converted to $(0, 1)$ by division with m. Weakness: (Marsaglia, 1968): if $(s_i, \ldots, s_{i+d})$ represent some points in a d–dimensional space, they have a lattice structure: they lie in a number of specific hyperplanes.

Famous multipliers ($a_0 = 0$):

- $a_1 = 23, m = 10^8 + 1$: original version, has higher order correlations
- $a_1 = 65539, m = 2^{29}$: infamous RANDU generator (IBM 360 series, in the 1970s): catastrophic higher order correlations
- $a_1 = 69069, m = 2^{32}$ (Marsaglia, 1972): good properties and converage up to 6 dimensions

RNGs, cont'd

Exercise:

- write a function

  ```
  rng.mlcg = function(n, a1=20, a0=0, m=53, s0=21)
  ```

  which implements the procedure MLCG (with some default parameters), and returns a sequence of *n* numbers.
- generate a sequence and plot u_{i+1} vs u_i

  ```
  > u = rng.mlcg(200)
  > plot(u[2:200],u[1:199])
  ```

- discuss!

RNGs, cont'd

Exercise:

- let $n = 20000$
- execute

```
> u = rng.mlcg(n, a1=65539, a0=0, m=2^31, s0=10)
> z = (u-0.5)/(2^31-1)    # map to (0,1)
> hist(z)                 # is it reasonably uniform?
> z1 = z[1:(n-2)]; z2 = z[2:(n-1)]; z3 = z[3:n]
> plot(z1, z2, pch=19, xlim=c(0,1), ylim=c(0,1))
> x11(); plot(z1[z3 < 0.01], z2[z3 < 0.01], ...
    pch=19, xlim=c(0,1), ylim=c(0,1))
```

- discuss!

RNGs, cont'd

In R: don't let the RNG to be "randomly" selected!

- for serious work, always set the seed, check the RNG, etc: they might be version-dependent; also you want other to be able to reproduce your results
- read the help for `RNG`
- uniform random numbers are generated with **`runif`**`()` function
- check also `{d, p, q}unif()` functions
- read the help for `.Random.seed()`

Outline

Non-uniform r.v. generation (NRNG)

Requirements:

- correctness: a good approximation of the theoretical distribution
- robustness: RNG should work well on a large range of parameters
- efficiency

NRNG: inversion method

- best choice, when feasible
- to generate X with distribution function F, starting from a uniform variate $U \in (0,1)$, apply the inverse F^{-1} to U:

$$X = F^{-1}(U) := \min\{x | F(x) \geq U\}$$

- easy to see that the distribution of X is as required:

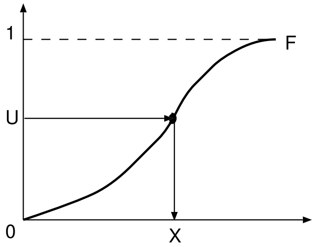
$$P[X \leq x] = P[F^{-1}(U) \leq x] = P[U \leq F(x)] = F(x)$$

- for some distributions, F^{-1} can be obtained analytically. Ex.: Weibull distribution $F(x) = 1 - \exp(-(x/\beta)^{\alpha})$, with $\alpha, \beta > 0$; has the inverse $F^{-1}(U) = \beta[-\ln(1-U)]^{1/\alpha}$
- other distributions do not have a close form inverse: e.g. normal, χ^2,... $\Rightarrow$ approximations

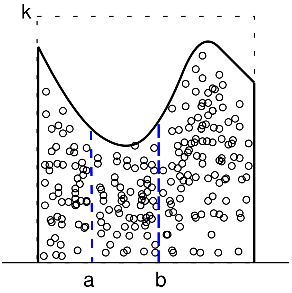
NRNG: inversion method, cont'd

Example (principle of inversion):

```
# return X with cdf F, for a
# uniform r.v. 0 < U < 1
# (look-up table method)
X = 0
while (F(X) < U) X = X + 1
return (X)
```



NRNG: Rejection method



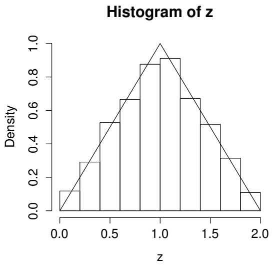
- consider F with a compact support and bounded $F(x) \leq k$
- consider a series of points (X_i, Y_i) uniformly distributed under the density function
- the distribution of X_i is the same as the distribution of X (F): $P[a < X_i < b] =$ probability of a point falling in the region $= \int_a^b F(x)dx$
- procedure:
  1. generate $X \sim U[a,b]$ and $Y \sim U[0,1]$ independently
  2. if $Y < F(X)$ return X, otherwise repeat

NRNG: Rejection method

Exercise: Implement the rejection method for generating random variates from the pdf

$$F(x) = \begin{cases} x & \text{if } 0 < x < 1 \\ 2 - x & \text{if } 1 \leq x < 2 \\ 0 & \text{otherwise} \end{cases}$$

Generate $n = 5000$ r.v., plot their histogram (use
`hist(..., freq=FALSE, ylim=c(0,1,01))`
and the original pdf.



Generating normally distributed r.v.

- you can use the rejection method
- alternative: Box-Muller algorithm: based on the observation that the coordinates of points in a 2D Cartesian system described by 2 independent normal distributions correspond to polar coordinates that are realizations of 2 independent uniform distributions
- Box-Muller transform: if U_1, U_2 are independent uniformly distributed on (0,1), then

$$Z_1 = r\cos\theta = \sqrt{-2\ln U_1}\cos(2\pi U_2)$$
$$Z_2 = r\sin\theta = \sqrt{-2\ln U_1}\sin(2\pi U_2)$$

Improved Box-Muller algorithm, with rejection step:

1. generate $U_1, U_2 \sim U(-1, 1)$
2. accept $S^2 = U_1^2 + U_2^2$ if $S^2 < 1$, else go to step 1
3. set $W = \sqrt{-2\frac{\ln S^2}{S^2}}$
4. return $X = U_1 W$ and $Y = U_2 W$

Exercise: Implement the procedure above in R!

Other methods for NRNG

- kernel density estimation: approximate the inverse using a kernels for which efficient generators exist
- composition: consider F to be a convex combination of several distributions F_j:

$$F(x) = \sum_j p_j F_j(x)$$

  To generate from F, one generates J with probability p_j and then generates X from F_j
- convolution: if $X = Y_1 + \cdots + Y_n$, with Y_j independent with specified distributions, then generate the Y_j's and sum them
- etc etc

Efficient implementations exist in R for:

- normal distribution: **`rnorm`**; log-normal: **`dlnorm`**
- binomial distribution: **`rbinom`**
- Poisson distribution: **`rpois`**
- ...

Outline

MC methods for inference

General approach:

1. identify the random variable of interest X
2. identify/postulate its distributional properties
3. generate one or several *large* samples *identical and independently distributed* $X_1, \ldots, X_n$ from the distribution of X
4. estimate the quantity of interest (e.g. estimate $\mathbb{E}X$ using sample average) and assess its accuracy (e.g. via confidence intervals)

Outline

MC inference about the mean

Reminder:

- problem: compute $z = \mathbb{E}Z$ when x is not available analytically, but Z can be simulated
- consider n replicates $Z_1, \ldots, Z_n$ of Z and estimate z by the empirical mean $\hat{z} = \sum_i Z_i/n$
- denote $\sigma^2 = Var\{Z\} < \infty$
- central limit theorem:

$$\sqrt{n}(\hat{z} - z) \to \mathcal{N}(0, \sigma^2), \text{ as } n \to \infty$$

- from this, an $1 - \alpha$ confidence interval can be obtained as

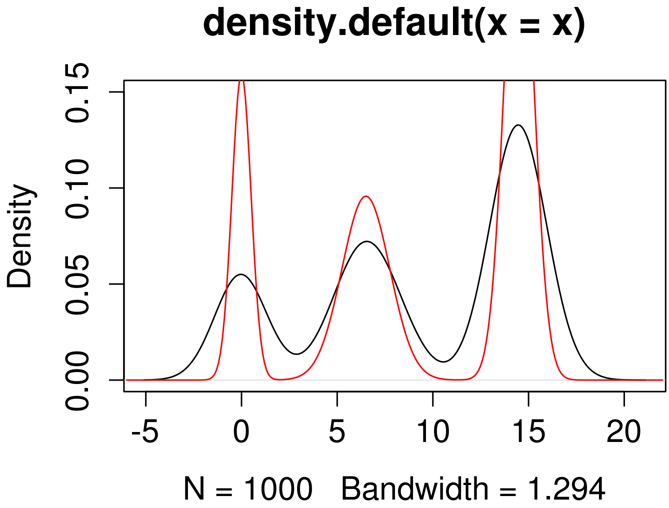
$$\left(\hat{z} - z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}, \hat{z} - z_{\alpha/2}\frac{\sigma}{\sqrt{n}}\right)$$

where z_α denotes the α–quantile of the normal distribution $(\Phi(z_\alpha) = \alpha)$

MC for inference about the mean

Implement the following procedure:

- write the R function `pdf1(n)` to generate $n = 1000$ r.v. drawn from
$$f(X) = 0.2N_1(X) + 0.3N_2(X) + 0.5N_3(X)$$
where N_i are Gaussians with parameters $\mu_1 = 0, \sigma_1 = 0.5$, $\mu_2 = 6.5, \sigma_2 = 1.25, \mu_3 = 14.5, \sigma_3 = 0.75$. Do not use **`for`** loops or any function from the various nonstandard packages!
- plot the density of the sample drawn and compare it with the theoretical plot of the mixture density
- repeat the procedure for $n = 10000$ and $n = 100000$. what do you see?


density.default(x = x)
Density
0.00
0.05
0.10
0.15
-5
0
5
10
15
20
N = 1000 Bandwidth = 1.294

- generate $p = 1000$ samples of $n = 1000$ r.v.: $X[p \times n]$
- compute $\hat{x}_i$ as the sample average for each of the p samples and the grand average $\hat{X}$
- what is the true mean of this mixture of Gaussians?
- test the normality of the distribution of $\hat{x}_i$ (e.g. `shapiro.test()`)
- estimate the 95% empirical confidence interval (using quantiles of the distribution of $\hat{x}_i$) and compare it with the theoretical one (using sample variance for σ^2) obtained from a single sample (say, `X[1,]`)