2. Simulation of Random Variables

Why Monte-Carlo simulation?

See the section 'Why Monte-Carlo simulation?' in sampling_ex.pdf.

Often we are interested in the distribution of a random variable X which is complicated, but which can none-the-less be built up from simple components such as independent rv's with known distributions.

Monte-Carlo simulation is an excellent tool for such problem: we seek to generate a random sample from the distribution of X, which we can use to estimate its mean, median, mode, percentiles, etc.

The starting point for any simulation is the generation of r.v.s with known distributions (binomial, poisson, exponential, normal, etc.), which are the building blocks for more complicated distributions. It turns out that all random variables can be generated by manipulating $U(0,1)$ rv's.

Seeding

If the random number generator is initialised (called the seed) before you start generating pseudo-random numbers, then you can reproduce the whole sequence exactly. This is a very good idea from a scientific point of view; being able to repeat an experiment means that your results are verifiable.

If the random number generator is not initialised, then R initialises it using a value taken from the system clock.

To generate n pseudo-random numbers in R, use `runif(n)`. For a given value of `seed` (assumed integer), the command `set.seed(seed)` always puts you at the same point on the cycle of pseudo-random numbers.

Seeding

```
> set.seed(42)
> runif(2)
[1] 0.9148060 0.9370754
> runif(2)
[1] 0.2861395 0.8304476
> set.seed(42)
> runif(4)
[1] 0.9148060 0.9370754 0.2861395 0.8304476
```

Simulating discrete random variables

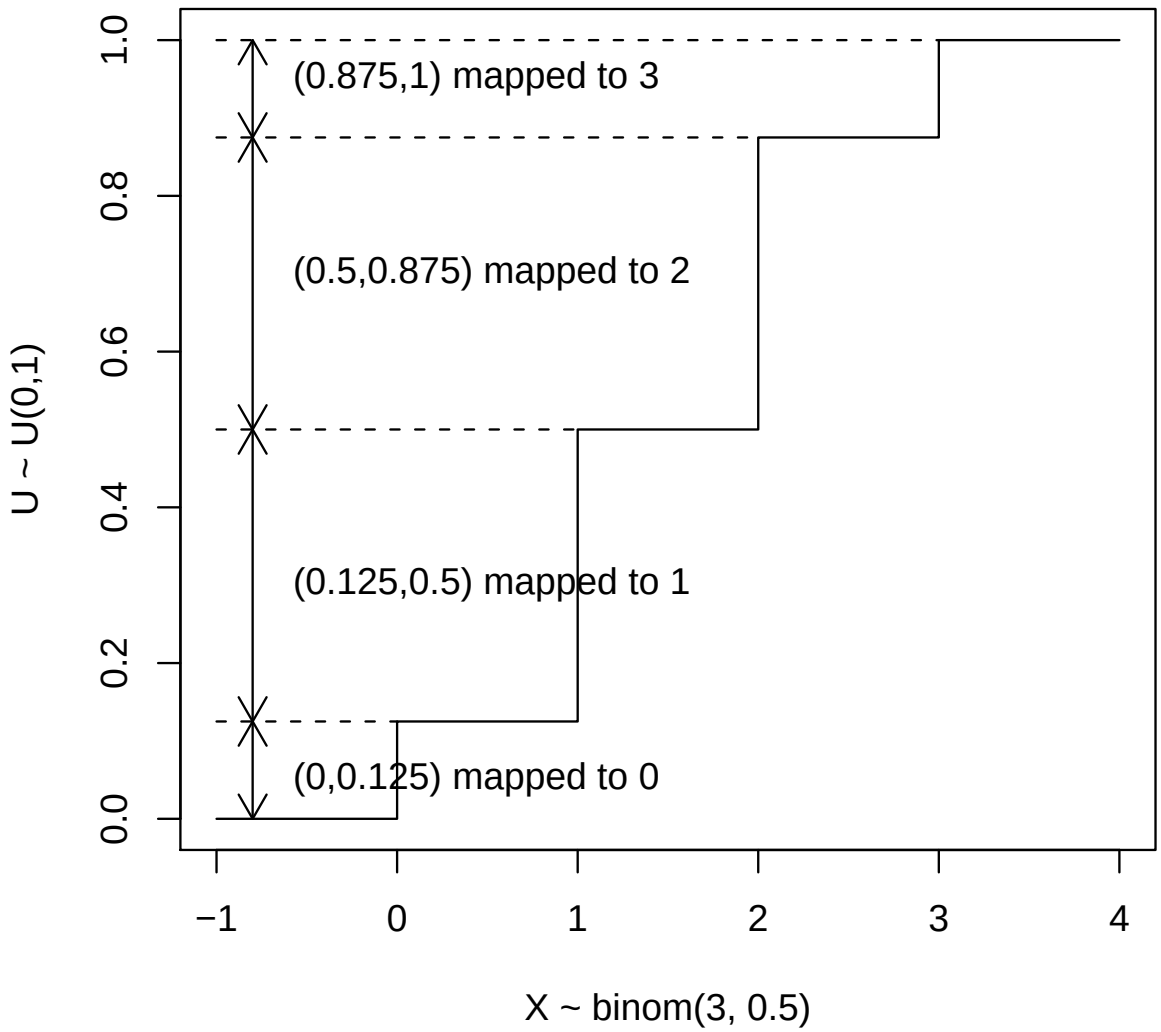
Simulating discrete random variables

Let X be a discrete random variable taking values in the set $\{0, 1, \ldots\}$ with cdf F and pmf p. The following snippet of code takes a uniform random variable U and returns a discrete random variable X with cdf F.

```
# given U ~ U(0,1)
X <- 0
while (F(X) < U) {
    X <- X + 1
}
```

When the algorithm terminates we have $F(X) \geq U$ and $F(X-1) < U$, that is $U \in (F(X-1), F(X)]$. That is,

$$\mathbb{P}(X = x) = \mathbb{P}(U \in (F(x-1), F(x)]) = F(x) - F(x-1) = p(x).$$


simulating from a binom(3, 0.5) c.d.f.
(0.875,1) mapped to 3
(0.5,0.875) mapped to 2
(0.125,0.5) mapped to 1
(0,0.125) mapped to 0
U ~ U(0,1)
0.0
0.2
0.4
0.6
0.8
1.0
−1
0
1
2
3
4
X ~ binom(3, 0.5)

To simulate binomial, geometric, negative-binomial or Poisson rv's in R, use `rbinom`, `rgeom`, `rnbinom` or `rpois`.

For simulating other (finite) discrete rv's R provides
`sample(x, size, replace = FALSE, prob = NULL)`.
The inputs are

`x`	A vector giving the possible values the rv can take;
`size`	How many rv's to simulate;
`replace`	Set this to `TRUE` to generate an iid sample, otherwise the rv's will be conditioned to be different from each other;
`prob`	A vector giving the probabilities of the values in `x`. If omitted then the values in `x` are assumed to be equally likely.

See the section 'Simulating other (finite) discrete rv using sample function' in sampling_ex.pdf.

Simulating continuous random variables

Simulating continuous random variables

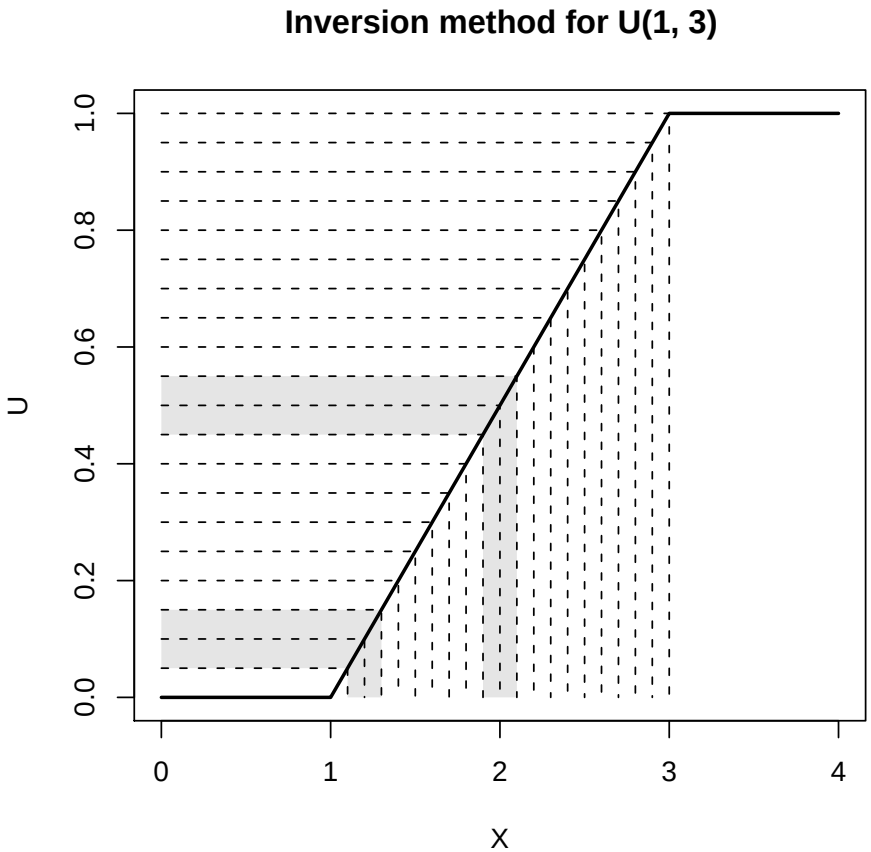
Suppose that we are given $U \sim U(0,1)$ and want to simulate a continuous rv X with cdf F_X.
Put $Y = F_X^{-1}(U)$ then we have

$$F_Y(y) = \mathbb{P}(Y \leq y) = \mathbb{P}(F_X^{-1}(U) \leq y) = \mathbb{P}(U \leq F_X(y)) = F_X(y).$$

That is, Y has the same distribution as X.

Thus, if we can simulate a $U(0,1)$ rv, then we can simulate any continuous rv X for which we know F_X^{-1}. This is called the *inverse transformation method* or simply the *inversion method*.

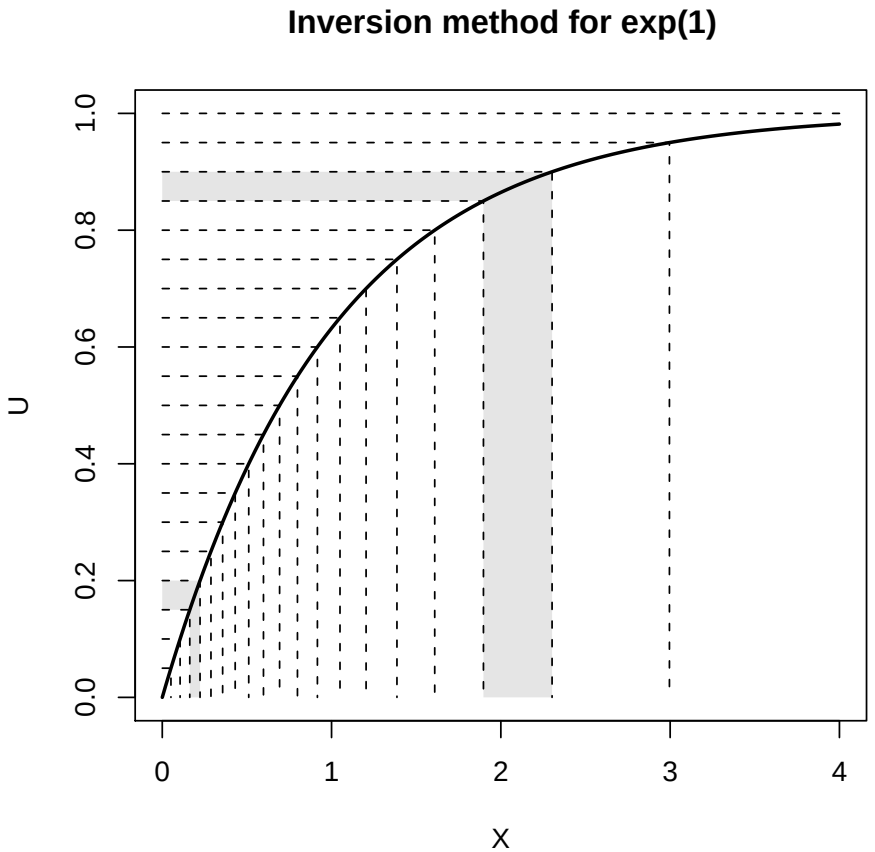
Simulating $X \sim U(1,3)$



If $X \sim U(1,3)$ then $F_X(x) = (x-1)/2$ for $x \in (1,3)$ and thus $F_X^{-1}(y) = 2y+1$ for $y \in (0,1)$.

Example: see the section 'Simulating X $\sim$ U(1,3)' in sampling_ex.pdf.

Simulating $X \sim \exp(\lambda)$



If $X \sim \exp(\lambda)$ then $F_X(x) = 1 - e^{-\lambda x}$ for $x \geq 0$ and thus $F_X^{-1}(y) = -\frac{1}{\lambda}\log(1-y)$.

Example: see the section 'Simulating X $\sim$ exp(1)' in sampling_ex.pdf.

Random variable simulators in R

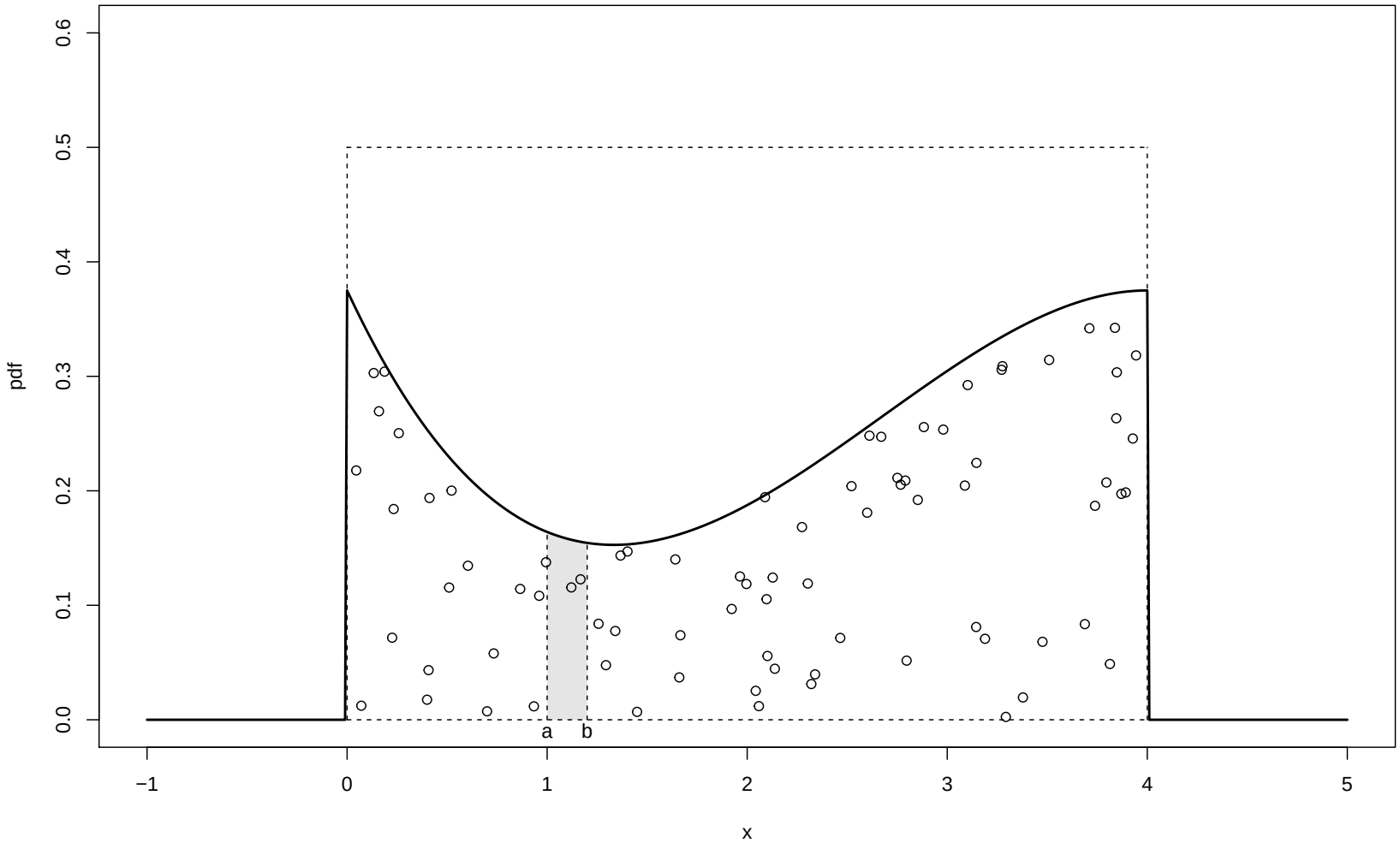
Distribution	R command
binomial	`rbinom`
Poisson	`rpoisson`
geometric	`rgeom`
negative binomial	`rnbinom`
uniform	`runif`
exponential	`rexp`
normal	`rnorm`
gamma	`rgamma`
beta	`rbeta`
student t	`rt`
F	`rf`
chi-squared	`rchisq`
Weibull	`rweibull`

The rejection method

The rejection method

The inversion method works well if we can find F^{-1} analytically. If not, one method in this situation, which is often faster, is the rejection method.

We start with an example. Suppose that we have a continuous random variable X with pdf f_X concentrated on the interval $(0, 4)$. We imagine 'sprinkling' points $P_1, P_2, \ldots$, uniformly at random under the density function, and consider the distribution of X_1, the x coordinate of P_1.


pdf
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−1
0
1
2
3
4
5
x
a
b

Let R be the shaded region under f_X between a and b, then

$$\begin{aligned}
\mathbb{P}(a < X_1 < b) &= \mathbb{P}(P_1 \text{ hits R}) \\
&= \frac{\text{Area of R}}{\text{Area under density}} \\
&= \frac{\int_a^b f_X(x)\,dx}{1} \\
&= \int_a^b f_X(x)\,dx.
\end{aligned}$$

So X_1 has the same distribution as X.

But how do we generate the points P_i uniformly under f_X? The answer is to generate points at random in the rectangle $[0, 4] \times [0, 0.5]$, and then *reject* those that fall above the pdf.

Rejection method (uniform envelope) Suppose that f_X is non-zero only on $[a, b]$, and $f_X \leq k$.

1. Generate $X \sim U(a, b)$ and $Y \sim U(0, k)$ independent of X (so $P = (X, Y)$ is uniformly distributed over the rectangle $[a, b] \times [0, k]$).
2. If $Y < f_X(X)$ then return X, otherwise go back to step 1.

Example: consider the triangular pdf f_X defined as

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

We apply the rejection method as follows: see rejecttriangle.pdf

General rejection method

Our rejection method uses a rectangular envelope to cover the target density f_X. What to do if f_X is unbounded?

Let X have pdf h and let $Y \sim U(0, kh(X))$, then (X, Y) is uniformly distributed under the curve kh:

$$\begin{aligned}
&\mathbb{P}((X, Y) \in (x, x+dx) \times (y, y+dy)) \\
&\quad = \quad \mathbb{P}(Y \in (y, y+dy) \,|\, X \in (x, x+dx)) \mathbb{P}(X \in (x, x+dx)) \\
&\quad = \quad \frac{dy}{kh(x)} h(x)\, dx \;=\; \frac{1}{k}\, dx dy.
\end{aligned}$$

Suppose we wish to simulate from the density f_X. Let h be a density we can simulate from, and choose k such that

$$k \geq k^* = \sup_x \frac{f_X(x)}{h(x)}.$$

Then kh forms an envelope for f_X, and we can generate points uniformly within this envelope. By accepting points below the curve f_X, we get the general rejection method:

General rejection method

To simulate from the density f_X, we assume that we have envelope density h from which you can simulate, and that we have some $k < \infty$ such that $\sup_x f_X(x)/h(x) \leq k$.

1. Simulate X from h.
2. Generate $Y \sim U(0, kh(X))$.
3. If $Y < f_X(X)$ then return X, otherwise go back to step 1.

Suppose that $f_X^* \propto f_X$, $h^* \propto h$, and $f_X^*(x)/h^*(x) \leq 1$, then the general rejection method is equivalent to:

1. Simulate X from h.
2. Generate $Y \sim U(0,1)$.
3. If $Y < f_X^*(X)/h^*(X)$ then return X, otherwise go back to step 1.

This equivalence is useful when it is hard to get an exact form for f_X (e.g., it is hard to get a normalisation factor in the posterior distribution).

Efficiency

The efficiency of the rejection method is measured by the expected number of times you have to generate a candidate point (X, Y).

The area under the curve kh is k and the area under the curve f_X is 1, so the probability of accepting a candidate is $1/k$.

Thus the number of times N we have to generate a candidate point has distribution geom$(1/k)$, with mean

$$\mathbb{E}N = k.$$

So, the closer h is to f_X, the smaller we can choose k, and the more efficient the algorithm.

Example: gamma

For $m, \lambda > 0$ the $\Gamma(\lambda, m)$ density is

$$f(x) = \lambda^m x^{m-1} e^{-\lambda x} / \Gamma(m), \text{ for } x > 0,$$

There is no explicit formula for the cdf F or its inverse, so we will use the rejection method to simulate from f.

We will use an exponential envelope $h(x) = \mu e^{-\mu x}$, for $x > 0$. Using the inversion method we can easily simulate from h using $-\log(U)/\mu$, where $U \sim U(0, 1)$.

To envelop f we need to find

$$k^* = \sup_{x>0} \frac{f(x)}{h(x)} = \sup_{x>0} \frac{\lambda^m x^{m-1} e^{(\mu-\lambda)x}}{\mu\Gamma(m)}.$$

Clearly k^* will be infinite if $m < 1$ or $\lambda \leq \mu$. For $m = 1$ the gamma is just an exponential. Thus we will assume $m > 1$ and choose $\mu < \lambda$.

For $m \in (0, 1)$ the rejection method can still be used, but a different envelope is required.

To find k^* we take the derivative of the right-hand side above and set it to zero, to find the point where the maximum occurs. You can check that this is at the point $x = (m-1)/(\lambda - \mu)$, which gives

$$k^* = \frac{\lambda^m (m-1)^{m-1} e^{-(m-1)}}{\mu(\lambda-\mu)^{m-1}\Gamma(m)}.$$

To improve efficiency we would like to choose our envelope to make k^* as small as possible. Looking at the formula for k^* this means choosing μ to make $\mu(\lambda - \mu)^{m-1}$ as large as possible. Setting the derivative with respect to μ to zero, we see that the maximum occurs when $\mu = \lambda/m$. Plugging this back in we get $k^* = m^m e^{-(m-1)}/\Gamma(m)$.

We can now code up our rejection algorithm: see gamma_sim.pdf

The Gibbs sampler

Notes by Guoqi Qian

Gibbs sampler

The Gibbs sampler is a random vector generation method which does not require the complete information of the target multivariate probability distribution. Instead, it only requires the information of a set of the associated conditional distributions. Gibbs sampler will be computationally feasible if the conditional distributions are ready to be simulated.

Gibbs sampler

- Suppose we want to generate a vector $\mathbf{u} = (u_1, \cdots, u_K)$ from the random vector $\mathbf{U} = (U_1, \cdots, U_K)$ which has a joint pdf $f(\mathbf{u})$.
- Suppose the pdf $f(\mathbf{u})$ has a very complicated form. But for each $k = 1, \cdots, K$, the conditional pdf of U_k given $\mathbf{U}_{-k} = (U_1, \cdots, U_{k-1}, U_{k+1}, \cdots, U_K)$ is known and relatively easy to simulate. We use

$$f(u_k|\mathbf{u}_{-k}) \equiv f(u_k|u_1, \cdots, u_{k-1}, u_{k+1}, \cdots, u_K)$$

  to denote such a conditional pdf.
- Then the Gibbs sampler used to generate one observation of $\mathbf{U}$ can be described as following:

Gibbs sampler

Algorithm A.1 [*Gibbs sampler*]

1° Arbitrarily generate/assign an initial vector $\mathbf{u}^{(0)} = (u_1^{(0)}, \cdots, u_K^{(0)})$ from the support of $f(\mathbf{u})$.

2° Generate a value $u_1^{(1)}$ from $f(u_1 | u_2 = u_2^{(0)}, \cdots, u_K = u_K^{(0)})$;
then generate a value $u_2^{(1)}$ from
$f(u_2 | u_1 = u_1^{(1)}, u_3 = u_3^{(0)}, \cdots, u_K = u_K^{(0)})$;
continue until generate a value $u_{K-1}^{(1)}$ from
$f(u_{K-1} | u_1 = u_1^{(1)}, \cdots, u_{K-2} = u_{K-2}^{(1)}, u_K = u_K^{(0)})$,
and generate a value $u_K^{(1)}$ from
$f(u_K | u_1 = u_1^{(1)}, \cdots, u_{K-1} = u_{K-1}^{(1)})$.
The generated values are delivered as $\mathbf{u}^{(1)} = (u_1^{(1)}, \cdots, u_K^{(1)})$.

Gibbs sampler

Algorithm A.1 [*Gibbs sampler (continued)*]

3° For $j = 1, 2, \cdots$, do the following:
Generate a value $u_1^{(j)}$ from
$f(u_1 | u_2 = u_2^{(j-1)}, \cdots, u_K = u_K^{(j-1)})$; then generate a value $u_2^{(j)}$ from $f(u_2 | u_1 = u_1^{(j)}, u_3 = u_3^{(j-1)}, \cdots, u_K = u_K^{(j-1)})$;
continue until generate a value $u_{K-1}^{(j)}$ from
$f(u_{K-1} | u_1 = u_1^{(j)}, \cdots, u_{K-2} = u_{K-2}^{(j)}, u_K = u_K^{(j-1)})$,
and generate a value $u_K^{(j)}$ from
$f(u_K | u_1 = u_1^{(j)}, \cdots, u_{K-1} = u_{K-1}^{(j)})$.
The generated values are delivered as $\mathbf{u}^{(j)} = (u_1^{(j)}, \cdots, u_K^{(j)})$.

Note the conditioning is always based on the latest values of $(u_1, \cdots, u_K)$.

Remarks on Gibbs sampler

- Using theory of Markov chain, it can be shown that

$$\mathbf{u}^{(j)} \xrightarrow{d} f(\mathbf{u}) \quad \text{as } j \to \infty$$

  under fairly general conditions. Details are not pursued here.
- This implies that $\mathbf{u}^{(j)}$ can be roughly regarded as an observation of $\mathbf{U}$ from pdf $f(\mathbf{u})$ when j is sufficiently large. Empirical methods are available (details not pursued here) to determine how large j should be.
- In practice, we usually generate a long sequence $\mathbf{u}^{(1)}, \cdots, \mathbf{u}^{(m)}, \mathbf{u}^{(m+1)}, \cdots, \mathbf{u}^{(m+J)}$ using the Gibbs sampler; then ignore the **burn-in** sequence $\mathbf{u}^{(1)}, \cdots, \mathbf{u}^{(m)}$ and use only the part $\mathbf{u}^{(m+1)}, \cdots, \mathbf{u}^{(m+J)}$ as a random sample from $f(\mathbf{u})$.
- $\mathbf{u}^{(m+1)}, \cdots, \mathbf{u}^{(m+J)}$ are not independent of each other but constitute a Markov chain.

Example

Eggs of certain species of insects hatch under appropriate conditions. Suppose there are N eggs of this species of insects in a particular area, and X eggs of them will hatch. Also let p be the probability that such an egg will hatch.

1. A Bayesian approach may be used to model (X, p, N), in which one can reasonably assume $N \overset{\mathrm{d}}{=} \text{Poisson}(\lambda)$, $p \overset{\mathrm{d}}{=} \text{Beta}(a, b)$ and $(X|p, N) \overset{\mathrm{d}}{=} \text{binomial}(N, p)$. Suppose $N \overset{\mathrm{d}}{=} \text{Poi}(16)$ and $p \overset{\mathrm{d}}{=} \text{Beta}(2, 4)$ based on a previous study on these insects. Find the joint pdf of (X, p, N) and the marginal pdf of X.
2. Derive a Gibbs sampler for generating (X, p, N) and X.
3. Implement the algorithm in R and check its performance.

Example

When $N \overset{\mathrm{d}}{=} \text{Poi}(16)$ and $p \overset{\mathrm{d}}{=} \text{Beta}(2,4)$, the joint pdf of (X,p,N) is

$$\begin{aligned} f(x,p,N) &= f(x|p,N)\cdot f(p)\cdot f(N) \\ &= \binom{N}{x} p^x(1-p)^{N-x}\cdot\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}p^{a-1}(1-p)^{b-1}\cdot\frac{\lambda^N}{N!}e^{-\lambda} \\ &= 20e^{-16}\binom{N}{x}\frac{16^N}{N!}p^{x+1}(1-p)^{N-x+3}; \\ & \quad x=0,1,\cdots,N;\ 0\le p\le 1;\ N=0,1,2,\cdots. \end{aligned}$$

The marginal pdf of X is

$$f(x)=\sum_{N=x}^{\infty}\int_0^1 f(x,p,N)\,dp=\sum_{N=x}^{\infty}\frac{20e^{-16}(x+1)(N-x+3)(N-x+2)(N-x+1)16^N}{(N+5)!}$$

which is difficult to be further simplified.

Example

But it is not difficult to find the full conditional pdf's:

$$(X|p,N) \stackrel{\mathrm{d}}{=} \mathsf{bin}(N,p), \qquad (p|x,N) \stackrel{\mathrm{d}}{=} \mathsf{Beta}(x+2, N-x+4),$$
$$\text{and} \qquad (N-x|x,p) \stackrel{\mathrm{d}}{=} \mathsf{Poi}(16(1-p)).$$

Proof:

$$\begin{aligned} f(p|x,N) &= \frac{f(x,p,N)}{f(x,N)} = \frac{20e^{-16}\binom{N}{x}\frac{16^N}{N!}p^{x+1}(1-p)^{N-x+3}}{\int_0^1 20e^{-16}\binom{N}{x}\frac{16^N}{N!}p^{x+1}(1-p)^{N-x+3}\,dp} \\ &= \frac{p^{x+1}(1-p)^{N-x+3}}{\int_0^1 p^{x+1}(1-p)^{N-x+3}\,dp} = \frac{\Gamma(N+6)}{\Gamma(x+2)\Gamma(N-x+4)}p^{x+1}(1-p)^{N-x+3} \end{aligned}$$

which is a pdf of $\mathsf{Beta}(x+2, N-x+4)$.

Example

$$
\begin{aligned}
f(N|x,p) &= \frac{f(x,p,N)}{f(x,p)} = \frac{20e^{-16}\binom{N}{x}\frac{16^N}{N!}p^{x+1}(1-p)^{N-x+3}}{\sum_{N=x}^{\infty}20e^{-16}\binom{N}{x}\frac{16^N}{N!}p^{x+1}(1-p)^{N-x+3}} \\
&= \frac{\binom{N}{x}[16(1-p)]^N\frac{1}{N!}}{\sum_{\tilde{N}=x}^{\infty}\binom{\tilde{N}}{x}\frac{16^{\tilde{N}}}{\tilde{N}!}(1-p)^{\tilde{N}}} = \frac{\frac{1}{(N-x)!}[16(1-p)]^N}{\sum_{\tilde{N}=x}^{\infty}\frac{1}{(\tilde{N}-x)!}[16(1-p)]^{\tilde{N}}} \\
&= \frac{[16(1-p)]^{N-x}}{(N-x)!}e^{-16(1-p)}; \quad N = x, x+1, x+2, \cdots .
\end{aligned}
$$

This implies that $(N-x|x,p) \stackrel{\mathrm{d}}{=} \text{Poisson}(16(1-p))$. □

Now a sample of (X,p,N) can be generated using the following Gibbs sampler algorithm

Example

Algorithm A.2 [*Gibbs sampler algorithm for* (X, p, N)]

1° Arbitrarily choose an initial vector, e.g. $(x^{(0)}, p^{(0)}, N^{(0)}) = (8, 0.5, 16)$.

2° $(x^{(1)}, p^{(1)}, N^{(1)})$ is obtained by:
generating $x^{(1)}$ from Bin$(N^{(0)}, p^{(0)})$;
generating $p^{(1)}$ from Beta$(x^{(1)} + 2, N^{(0)} - x^{(1)} + 4)$; and
generating an $N^{(1)}$ from Poi$(16(1 - p^{(1)})) + x^{(1)}$.

3° $(x^{(j)}, p^{(j)}, N^{(j)})$, $j = 1, 2, \cdots$, is obtained by:
generating $x^{(j)}$ from Bin$(N^{(j-1)}, p^{(j-1)})$;
generating $p^{(j)}$ from Beta$(x^{(j)} + 2, N^{(j-1)} - x^{(j)} + 4)$; and
generating an $N^{(j)}$ from Poi$(16(1 - p^{(j)})) + x^{(j)}$.

Once a sample of (X, p, N) is obtained, a sample of X can be read off from it.

Example

The above algorithm was implemented in R. Then a sample of 1000 (X, p, N) samples are generated, and the simulated marginal pdf's of X, p and N are obtained. The results are shown in simulation_Gibbs.pdf.