Computer Intensive Methods in Mathematical Statistics

Johan Westerborn

Department of mathematics
KTH Royal Institute of Technology
johawes@kth.se

Lecture 2
Random number generation
23 March 2017

Last time: Principal aim

- We formulated the main problem of the course, namely to compute some expectation

$$\tau = \mathbb{E}(\phi(X)) = \int_{\mathsf{X}} \phi(x) f(x)\, dx,$$

where

  - X is a random variable taking values in $\mathsf{X} \subseteq \mathbb{R}^d$ (where $d \in \mathbb{N}^*$ may be very large),
  - $f : \mathsf{X} \to \mathbb{R}_+$ is the probability density (target density) of X, and
  - $\phi : \mathsf{X} \to \mathbb{R}$ is a function (objective function) such that the above expectation exists.

Last time: The MC method in a nutshell

- Let $X^1, X^2, \ldots, X^N$ be independent random variables with density f. Then, by the law of large numbers, as N tends to infinity,

$$\tau_N \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} \phi(X^i) \to \mathbb{E}(\phi(X)). \quad \text{(a.s.)}$$

- Inspired by this result, we formulated the basic MC sampler:

for $i = 1 \to N$ **do**
  draw $X^i \sim f$;
end
set $\tau_N \leftarrow \sum_{i=1}^{N} \phi(X^i)/N$;

Some properties of MC estimators

- We note that τ_N is unbiased in the sense that for all N,

$$\mathbb{E}(\tau_N) = \mathbb{E}\left(\frac{1}{N}\sum_{i=1}^{N}\phi(X^i)\right) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}(\phi(X^i)) = \tau.$$

- In addition, as we saw last time, the variance of τ_N is

$$\mathbb{V}(\tau_N) = \frac{1}{N}\sigma^2(\phi) \Rightarrow \mathbb{D}(\tau_N) = \frac{1}{\sqrt{N}}\sigma(\phi).$$

- Finally, the CLT implies that the normalized MC error is asymptotically normally distributed:

$$\sqrt{N}\,(\tau_N - \tau) \xrightarrow{\text{d.}} \mathrm{N}(0, \sigma^2(\phi)), \quad \text{as } N \to \infty,$$

What do we need to know?

- OK, so what do we need to master for having practical use of the MC method?
- We agreed on that, for instance, the following questions should be answered:
  - How do we generate the needed input random variables?
  - How many computer experiments should we do? What can be said about the error?
  - Can we exploit problem structure to speed up the computation?
- Today we will discuss the first two issues.

Plan of today's lecture

1. Plug-in MC estimators
2. MC output analysis
3. Generating random numbers
   - Uniform pseudo-random numbers
   - Transformation-based methods
4. Summary

Outline

Plug-in MC estimators

- For a given estimand τ one is often interested in estimating $\varphi(\tau)$ for some function $\varphi : \mathbb{R} \to \mathbb{R}$. For instance, one may be interested in the squared expectation $\varphi(\tau) = \tau^2$.
- Question: what are the properties of the plug-in estimator $\varphi(\tau_N)$ of $\varphi(\tau)$?
- The estimator $\varphi(\tau_N)$ is generally biased for finite N; indeed, under suitable assumptions on φ it holds that

$$\mathbb{E}\left(\varphi(\tau_N) - \varphi(\tau)\right) = \frac{\varphi''(\tau)}{2}\mathbb{V}(\tau_N) + O(N^{-3/2}) = \frac{\varphi''(\tau)\sigma^2(\phi)}{2N} + O(N^{-3/2}),$$

implying that $\varphi(\tau_N)$ is still consistent.

Plug-in MC estimators

- For a given estimand τ one is often interested in estimating $\varphi(\tau)$ for some function $\varphi : \mathbb{R} \to \mathbb{R}$. For instance, one may be interested in the squared expectation $\varphi(\tau) = \tau^2$.
- Question: what are the properties of the plug-in estimator $\varphi(\tau_N)$ of $\varphi(\tau)$?
- The estimator $\varphi(\tau_N)$ is generally biased for finite N; indeed, under suitable assumptions on φ it holds that

$$\begin{aligned}\mathbb{E}\left(\varphi(\tau_N) - \varphi(\tau)\right) &= \frac{\varphi''(\tau)}{2}\mathbb{V}(\tau_N) + O(N^{-3/2}) \\ &= \frac{\varphi''(\tau)\sigma^2(\phi)}{2N} + O(N^{-3/2}),\end{aligned}$$

implying that $\varphi(\tau_N)$ is still consistent.

The delta method

- In addition, one may establish the CLT

$$\sqrt{N}\,(\varphi(\tau_N) - \varphi(\tau)) \xrightarrow{\text{d.}} \mathrm{N}(0, \varphi'(\tau)^2\sigma^2(\phi)), \quad \text{as } N \to \infty,$$

  which implies that the standard deviation of the plug-in estimator is obtained by simply scaling the standard deviation of the original estimator by $\varphi'(\tau)$.
- This will be discussed further during the first exercise class.

Outline

Confidence bounds

- Using the CLT one obtains straightforwardly the two-sided confidence bound

$$\mathrm{I}_\alpha = \left(\tau_N - \lambda_{\alpha/2} \frac{\sigma(\phi)}{\sqrt{N}}, \tau_N + \lambda_{\alpha/2} \frac{\sigma(\phi)}{\sqrt{N}} \right),$$

  for τ, where λ_p denotes the p-quantile of the standard normal distribution.
- I_α covers τ with (approximate) probability $1 - \alpha$.
- A problem here is that $\sigma^2(\phi)$ is generally not known.

Confidence bounds

- Using the CLT one obtains straightforwardly the two-sided confidence bound

$$\mathsf{I}_\alpha = \left(\tau_N - \lambda_{\alpha/2} \frac{\sigma(\phi)}{\sqrt{N}}, \tau_N + \lambda_{\alpha/2} \frac{\sigma(\phi)}{\sqrt{N}} \right),$$

  for τ, where λ_p denotes the p-quantile of the standard normal distribution.
- I_α covers τ with (approximate) probability $1 - \alpha$.
- A problem here is that $\sigma^2(\phi)$ is generally not known.

Confidence bounds (cont.)

- Quick fix: the variance $\sigma^2(\phi)$ is again an expectation that can be estimated using the already generated MC sample $(X^i)_{i=1}^N$. More specifically, using the plug-in estimator,

$$\sigma^2(\phi) = \mathbb{E}(\phi^2(X)) - \mathbb{E}\left(\phi(X)\right)^2 = \mathbb{E}(\phi^2(X)) - \tau^2$$
$$\approx \frac{1}{N}\sum_{i=1}^N \phi^2(X^i) - \tau_N^2 = \frac{1}{N}\sum_{i=1}^N \left(\phi(X^i) - \tau_N\right)^2.$$

- This estimator has however bias. A bias-corrected estimator (in MATLAB: `var` alt. `std`) is

$$\sigma_N^2(\phi) \stackrel{\text{def}}{=} \frac{1}{N-1}\sum_{i=1}^N \left(\phi(X^i) - \tau_N\right)^2.$$

Confidence bounds (cont.)

- Quick fix: the variance $\sigma^2(\phi)$ is again an expectation that can be estimated using the already generated MC sample $(X^i)_{i=1}^N$. More specifically, using the plug-in estimator,

$$\begin{aligned}\sigma^2(\phi) = \mathbb{E}(\phi^2(X)) - \mathbb{E}\left(\phi(X)\right)^2 &= \mathbb{E}(\phi^2(X)) - \tau^2 \\ &\approx \frac{1}{N}\sum_{i=1}^{N} \phi^2(X^i) - \tau_N^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\phi(X^i) - \tau_N\right)^2.\end{aligned}$$

- This estimator has however bias. A bias-corrected estimator (in MATLAB: `var` alt. `std`) is

$$\sigma_N^2(\phi) \stackrel{\text{def}}{=} \frac{1}{N-1}\sum_{i=1}^{N}\left(\phi(X^i) - \tau_N\right)^2.$$

Example: Buffon's needle

- Consider a grid of parallel lines of spacing d on which we drop randomly a needle of length ℓ, with $\ell \leq d$. Let

$$\begin{cases} X = \text{distance from lower needlepoint to upper grid line,} \\ \theta = \text{angle between needle and grid normal} \in (-\pi/2, \pi/2). \end{cases}$$

- Then

$$\tau = \mathbb{P}(\text{needle intersects grid}) = \mathbb{P}(X \leq \ell \cos\theta) = \ldots = \frac{2\ell}{\pi d}$$

or, equivalently,

$$\pi = \varphi(\tau) = \frac{2\ell}{\tau d}.$$

Example: Buffon's needle

- Consider a grid of parallel lines of spacing d on which we drop randomly a needle of length ℓ, with $\ell \leq d$. Let

$$\begin{cases} X = \text{distance from lower needlepoint to upper grid line}, \\ \theta = \text{angle between needle and grid normal} \in (-\pi/2, \pi/2). \end{cases}$$

- Then

$$\tau = \mathbb{P}(\text{needle intersects grid}) = \mathbb{P}(X \leq \ell \cos \theta) = \ldots = \frac{2\ell}{\pi d}$$

or, equivalently,

$$\pi = \varphi(\tau) = \frac{2\ell}{\tau d}.$$

Example: Buffon's needle (cont.)

- Since

$$\tau = \mathbb{P}(\text{needle intersects grid}) = \mathbb{E}\left(\mathbb{1}_{\{X \le \ell \cos\theta\}}\right),$$

we may obtain an approximation of π through

```
X = d*rand(1,N);
theta = - pi/2 + pi*rand(1,N);
tau = mean(X <= L*cos(theta));
```

and then letting `pi_est = 2*L./(tau*d)`.

- In addition, a 95% confidence interval is obtained through

```
sigma = std(X <= L*cos(theta));
LB = pi_est - norminv(0.975)*2*L/(d*tau^2*sqrt(N))*sigma;
UB = pi_est + norminv(0.975)*2*L/(d*tau^2*sqrt(N))*sigma;
```

Example: Buffon's needle (cont.)

- Since

$$\tau = \mathbb{P}(\text{needle intersects grid}) = \mathbb{E}\left(\mathbb{1}_{\{X \le \ell \cos\theta\}}\right),$$

we may obtain an approximation of π through

```
X = d*rand(1,N);
theta = - pi/2 + pi*rand(1,N);
tau = mean(X <= L*cos(theta));
```

and then letting `pi_est = 2*L./(tau*d)`.

- In addition, a 95% confidence interval is obtained through

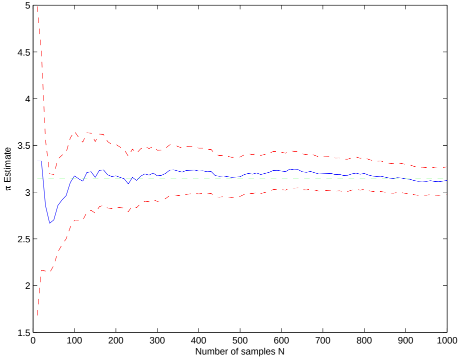
```
sigma = std(X <= L*cos(theta));
LB = pi_est - norminv(0.975)*2*L/(d*tau^2*sqrt(N))*sigma;
UB = pi_est + norminv(0.975)*2*L/(d*tau^2*sqrt(N))*sigma;
```

Example: Buffon's needle (cont.)

- Here we are actually cheating, since we require the value of π in our simulation code...
- As we will see later this lecture and during the first exercise class, it is however possible to simulate $\cos(\theta)$ directly by using only $\mathrm{U}(0, 1)$ random numbers.

Example: Buffon's needle (cont.)

- Executing this code for `N = 1:10:1000` yields the following graph (where the red-dashed lines are confidence bounds):



Outline

"Random"?! . . .

- Laplace's demon:

  "We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at a certain moment would know all forces that set nature in motion, and all positions of all items of which nature is composed, if this intellect were also vast enough to submit these data to analysis, it would embrace in a single formula the movements of the greatest bodies of the universe and those of the tiniest atom; for such an intellect nothing would be uncertain and the future just like the past would be present before its eyes."

  –P.-S. Laplace, *Essai philosophique sur les probabilités*, 1814

- Expresses determinism.
- The possibility of Laplace's demon is a fundamental question in philosophy.

Generating pseudo-random numbers

- Pseudo-random numbers $=$ numbers exhibiting statistical randomness while being generated by a deterministic process.
- We will discuss
  - how to generate pseudo-random uniform numbers,
  - transformation and inversion methods,
  - rejection sampling, and
  - conditional methods.

Good pseudo-random numbers

- "Good" pseudo-random numbers
  - appear to come from the correct distribution (also in the tails),
  - have long periodicity,
  - look "independent", and
  - are fast to generate.
- Most standard computing languages have packages or functions that generate either $U(0, 1)$ random numbers or integers on $U(0, 2^{32} - 1)$:
  - `rand` and `unifrnd` in MATLAB,
  - `rand` in C/C++,
  - `Random` in Java.

Good pseudo-random numbers

- "Good" pseudo-random numbers
  - appear to come from the correct distribution (also in the tails),
  - have long periodicity,
  - look "independent", and
  - are fast to generate.
- Most standard computing languages have packages or functions that generate either $\mathrm{U}(0,1)$ random numbers or integers on $\mathrm{U}(0,2^{32}-1)$:
  - `rand` and `unifrnd` in MATLAB,
  - `rand` in C/C++,
  - `Random` in Java.

Outline

Linear congruential generator

- The linear congruential generator is a simple, fast, and popular way of generating pseudo-random numbers:

$$X_n = (a \cdot X_{n-1} + c) \mod m,$$

where a, c, and m are integers.

- This recursion generates integer numbers (X_n) in $[0, m-1]$. These are mapped to $(0, 1)$ through division by m. It turns out that the period of the generator is m if
  - (i) c and m are relatively prime,
  - (ii) $a - 1$ is divisible by all prime factors of m, and
  - (iii) $a - 1$ is divisible by 4 if m is divisible by 4.
- As an example, MATLAB (pre v. 5) uses $m = 2^{32} - 1$, $a = 7^5 = 16807$, and $c = 0$.

Linear congruential generator

- The linear congruential generator is a simple, fast, and popular way of generating pseudo-random numbers:

$$X_n = (a \cdot X_{n-1} + c) \mod m,$$

  where a, c, and m are integers.
- This recursion generates integer numbers (X_n) in $[0, m-1]$. These are mapped to $(0, 1)$ through division by m. It turns out that the period of the generator is m if
  (i) c and m are relatively prime,
  (ii) $a - 1$ is divisible by all prime factors of m, and
  (iii) $a - 1$ is divisible by 4 if m is divisible by 4.
- As an example, MATLAB (pre v. 5) uses $m = 2^{32} - 1$, $a = 7^5 = 16807$, and $c = 0$.

Linear congruential generator

- The linear congruential generator is a simple, fast, and popular way of generating pseudo-random numbers:

$$X_n = (a \cdot X_{n-1} + c) \mod m,$$

where a, c, and m are integers.

- This recursion generates integer numbers (X_n) in $[0, m-1]$. These are mapped to $(0, 1)$ through division by m. It turns out that the period of the generator is m if
  (i) c and m are relatively prime,
  (ii) $a - 1$ is divisible by all prime factors of m, and
  (iii) $a - 1$ is divisible by 4 if m is divisible by 4.
- As an example, MATLAB (pre v. 5) uses $m = 2^{32} - 1$, $a = 7^5 = 16807$, and $c = 0$.

Outline

Bijective transformations of random variables

- Let X be a stochastic variable with density f_X on $\mathsf{X} \subseteq \mathbb{R}$ and let $g : \mathsf{X} \to \mathsf{Y} \subseteq \mathbb{R}$ be some differentiable, strictly increasing function with inverse g^{-1}.
- Define $Y = g(X)$ and let f_Y denote the density of Y. We show that

$$f_Y(y) = f_X(g^{-1}(y)) \frac{d}{dy} g^{-1}(y) \quad (y \in \mathsf{Y}).$$

- In the case where g is strictly decreasing we may argue similarly, and it holds generally that

$$f_Y(y) = f_X(g^{-1}(y)) \left| \frac{d}{dy} g^{-1}(y) \right| \quad (y \in \mathsf{Y})$$

for a strictly monotone function g.

Bijective transformations of random variables

- Let X be a stochastic variable with density f_X on $\mathsf{X} \subseteq \mathbb{R}$ and let $g : \mathsf{X} \to \mathsf{Y} \subseteq \mathbb{R}$ be some differentiable, strictly increasing function with inverse g^{-1}.
- Define $Y = g(X)$ and let f_Y denote the density of Y. We show that

$$f_Y(y) = f_X(g^{-1}(y)) \frac{d}{dy} g^{-1}(y) \quad (y \in \mathsf{Y}).$$

- In the case where g is strictly decreasing we may argue similarly, and it holds generally that

$$f_Y(y) = f_X(g^{-1}(y)) \left| \frac{d}{dy} g^{-1}(y) \right| \quad (y \in \mathsf{Y})$$

for a strictly monotone function g.

The transformation theorem

- Now, let X be n-dimensional, i.e., $\mathsf{X} \subseteq \mathbb{R}^n$, $g : \mathsf{X} \to \mathsf{Y} \subseteq \mathbb{R}^n$ a bijection, and set

$$
Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = g(X) = \begin{pmatrix} g_1(X_1, \ldots, X_n) \\ g_2(X_1, \ldots, X_n) \\ \vdots \\ g_n(X_1, \ldots, X_n) \end{pmatrix} \Leftrightarrow
$$

$$
X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{pmatrix} = g^{-1}(Y) = \begin{pmatrix} h_1(Y_1, \ldots, Y_n) \\ h_2(Y_1, \ldots, Y_n) \\ \vdots \\ h_n(Y_1, \ldots, Y_n) \end{pmatrix}.
$$

The transformation theorem (cont.)

- Generally, the following holds true.

Theorem (the transformation theorem)

The density of Y is

$$f_Y(y) = \begin{cases} f_X(h_1(y), h_2(y), \ldots, h_n(y))\,|\mathbf{J}(y)| & \text{if } y \in \mathsf{Y}, \\ 0 & \text{otherwise.} \end{cases}$$

where $\mathbf{J}(y)$ is the Jacobian matrix, i.e.,

$$\mathbf{J}(y) = \begin{pmatrix} \frac{\partial}{\partial y_1} h_1(y) & \frac{\partial}{\partial y_2} h_1(y) & \cdots & \frac{\partial}{\partial y_n} h_1(y) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial y_1} h_n(y) & \frac{\partial}{\partial y_2} h_n(y) & \cdots & \frac{\partial}{\partial y_n} h_n(y) \end{pmatrix}.$$

Example: Buffon's needle (again)

- In order to simulate $\cos(\theta)$, where $\theta \sim \mathrm{U}(-\pi/2, \pi/2)$, we may generate repeatedly independent $U_1 \sim \mathrm{U}(-1, 1)$ and $U_2 \sim \mathrm{U}(0, 1)$ until
$$U_1^2 + U_2^2 \leq 1$$
and then return
$$Y = \frac{U_1}{\sqrt{U_1^2 + U_2^2}}$$
- Using the transformation theorem, one may the show that
$$Y \mid \{U_1^2 + U_2^2 \leq 1\} \overset{\text{d.}}{=} \cos(\theta).$$

The inversion method

- The transformation theorem can be used for generating random variables being functions of easily-simulated variables.
- In particular, we now assume that we have access to a $\mathrm{U}(0,1)$ pseudo-random number generator and want to generate a random number X from some univariate distribution with a distribution function F whose inverse F^{-1} is at hand.
- For this purpose, we proceed as follows:

  draw $U \sim \mathrm{U}(0,1)$;
  set $X \leftarrow F^{-1}(U)$;

The inversion method, remarks

- Then the following is a consequence of the transformation theorem:

Theorem (the inverse method)

The output X of the algorithm above has distribution function F.

- The previous result holds can be extended to the generalized inverse

$$F^{\leftarrow}(u) \stackrel{\text{def}}{=} \inf\{x \in \mathbb{R} : F(x) \geq u\}.$$

- If F is continuous and strictly increasing, then $F^{\leftarrow} = F^{-1}$.
- The method is limited to cases where
  - we want to generate univariate random numbers and
  - the generalized inverse $F^{\leftarrow}$ is easy to evaluate (which is far from always the case).

Example: exponential distribution

- We use the inverse method for generating exponentially distributed random numbers (with unit expectation). Recall that this distribution has density

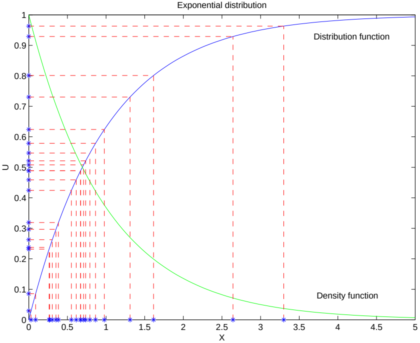
$$f(x) = e^{-x} \Rightarrow F(x) = \int_0^x f(z)\,dz = 1 - e^{-x} \quad (x \geq 0).$$

- Taking the inverse yields

$$F(x) = 1 - e^{-x} = u \Leftrightarrow x = F^{-1}(u) = -\log(1-u) \quad (u \in (0,1)).$$

Example: exponential distribution

```
F_inv = @(y) - log(1 - y);
U = rand(1,20);
X = F_inv(U);
```



Outline

Summary

Today we have

- shown that plug-in estimator $\varphi(\tau_N)$ of $\varphi(\tau)$ is asymptotically consistent,
- discussed how to construct confidence intervals of the MC estimates using the CLT,
- shown how to generate pseudo-random numbers using
  - the linear congruential generator (for uniform random numbers) and
  - the inversion method (when the general inverse $F^{\leftarrow}$ of F is obtainable).