ulm university universität
uulm
Institute of Stochastics

Grundlagen der Monte-Carlo-Simulation

Dr. Sebastian Lück | 7. Februar 2012

Contents

Motivation für Monte-Carlo-Simulation

- Ziel:
  Analyse stochastischer Modelle, die zu komplex sind, um ihr Verhalten effizient analytisch studieren zu können.
- praktische Relevanz:
  - Ersatz aufwendiger und teurer Laborexperimente bzw. Erhebung von Realdaten
  - Erhebung großer Datenmengen mit wenig Aufwand
  - Analyse von Systemen, die experimentell nicht beobachtbar sind
  - Analyse von Systemen, deren Analyse mit deterministischen mathematischen Methoden zu aufwändig oder unmöglich ist
- $\Rightarrow$ Prinzipielle Aufgabenstellung:
  Entwicklung von Simulationsalgorithmen zur Erzeugung von Realisierungen stochastischer Modelle:
  - Zufallsvariablen
  - zufällige geometrische Objekte
  - stochastische Prozesse in zeitlicher und/oder räumlicher Dimension

Klassen von Algorithmen

- Grundlage der Monte-Carlo-Simulation: Zufallszahlengeneratoren
  - Erzeugung von Standard-Pseudo-Zufallszahlen (SPZZ).
  - SPZZ: Folgen von Zahlen die als Realisierungen von iid $U((0,1])$ Zufallsvariablen betrachtet werden können.
- Transformationsalgorithmen:
  - Transformiere SPZZ so, dass sie als Realisierungen einer Folge komplexer verteilter ZV betrachtet werden können.
  - wichtige Teilklasse: Akzeptanz- und Verwerfungsmethoden
- Markov-Chain-Monte-Carlo Algorithmen:
  - Konstruktion von Markovketten, deren ergodische Grenzverteilung der zu simulierenden Verteilung entspricht.
  - Grundidee: starte eine solche Markovkette und lasse sie lange genug laufen, um 'in die Nähe'der Grenzverteilung zu gelangen.
  - Anwendungen: u.a. Simulation von Punktprozessen und anderer komplexer stochastischer Objekte.

Lineare Kongruenzgeneratoren (LKG)

- LKG: Algorithmus zur Generierung von Folgen $u_1, \ldots, u_n$ von Zahlen $u_i \in ([0,1))$, die als Realisierungen der unabhängig $U([0,1))$-verteilten ZV $U_1, \ldots, U_n$ betrachtet werden können.
- Algorithmus: Definiere
  - einen Modulus $m \in \mathbb{N}$,
  - einen Keim $z_0 \in \{0, \ldots m-1\}$,
  - einen Faktor $a \in \{0, \ldots, m-1\}$
  - ein Inkrement $c \in \{0, \ldots, m-1\}$.

  Setze rekursiv

$$z_k = (az_{k-1} + c) \mathrm{mod} m \quad \forall k = 1, \ldots, n$$

  Erzeuge die SPZZ u_k vermöge

$$u_k = \frac{z_k}{m}.$$

Bemerkungen

- Ein LKG mit Modulus m generiert höchstens m verschiedene SPZZ.
  Falls $z_{k+m_0} = z_k$ für ein $m_0 > 0 \Rightarrow z_{k+j} = z_{k+m_0+j}\ \forall j \geq 0$.
- Ungünstige Parameterwahl $\Rightarrow$ kurze Periode m_0.
  Bsp.: $a = c = z_0 = 5$, $m = 10$ erzeugt die Folge $5, 0, 5, 0, \ldots$.

Theorem (Bedingung für maximale Periodenlänge)

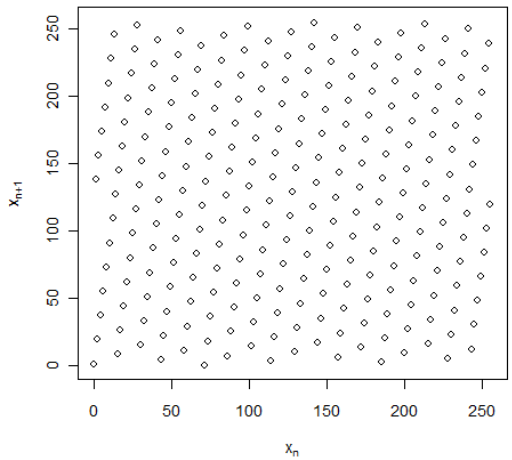
Der LKG hat genau dann volle Periode, wenn folgende drei Bedingungen gelten:

- c und m haben keine gemeinsamen Primfaktoren.
- $a - 1$ ist durch alle Primfaktoren von m teilbar.
- Falls m ein Vielfaches von 4 ist, dann ist auch $a - 1$ ein Vielfaches von 4.

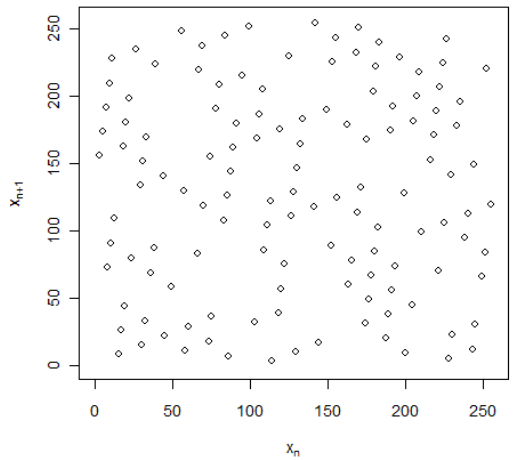
Eigenschaften guter Zufallszahlengeneratoren

- Generierte SPZZ sollten bei Tests auf statistische Unabhängigkeit und Gleichverteilung nicht abgelehnt werden.
- Lange Periode (d.h. viele verschiedene Zahlen ca. 10^50)
- Effizienz: schnell und nicht speicherintensiv (viele MC-Verfahren brauchen Milliarden an SPZZ)
- 0 und 1 sollten als Outputs nicht auftreten, um Division durch 0 zu vermeiden.
- Für kryptographische Zwecke: Unvorhersehbarkeit
- Paare aufeinanderfolgender SPZZ (u_i, u_{i+1}) sollten gleichmäßig auf $[0,1)^2$ verteilt sein ($\rightsquigarrow$ Unabhängigkeit).
  Analoges für Tripel etc.
  Das kann beim LKG ziemlich schiefgehen...

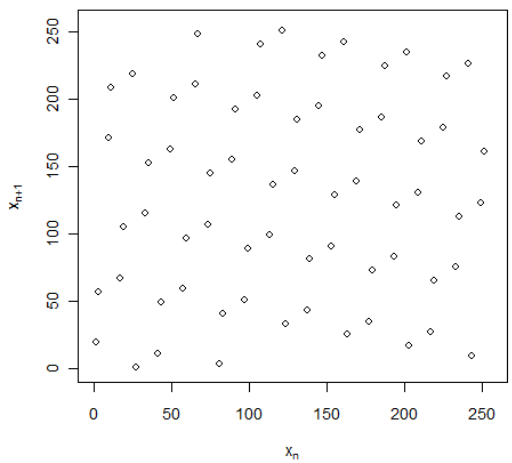
Punktmuster für Paare (x_n, x_{n+1}) aufeinanderfolgender SPZZ (vor Normierung) für $m = 256$ (nach H. Künsch; 2006)



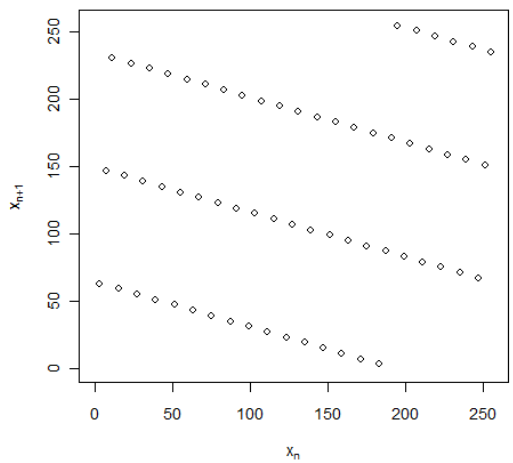
$a = 137,\ c = 1$



$a = 137,\ c = 1$ erste Hälfte



$a = 19,\ c = 0$



$a = 21,\ c = 0$

Fazit

- LKGs sind einfach zu implementierende und schnelle Algorithmen
- LKGs sind u.a. wegen ihrer linearen Abhängigkeiten nicht geeignet für
  - sensible MC-Simulationen,
  - kryptographische Zwecke.

Modulo 2 lineare Generatoren

Generelles Schema

- 1. Initialisiere Generator mit Bitwort $\mathbf{x}_0 \in \{0,1\}^k$
- 2. Definiere

$$\mathbf{x}_n = A\mathbf{x}_{n-1}$$

mit Matrix $A \in \{0,1\}^{k\times k}$, wobei Multiplikation und Addition modulo 2 definiert sind (also Rechnen in $\mathbb{F}_2$).

- 3. Definiere $\mathbf{y}_n = B\mathbf{x}_n$ mit Matrix $B \in \{0,1\}^{w\times k}$, wobei w die Darstellungslänge der Zahlen des Computers ist (z.B. 32 Bit).
- 4. Return $\mathbf{u}_n = \sum_{\ell=1}^{w} y_{n,\ell} 2^{-\ell}$.
- 5. Setze $n = n+1$ und gehe zu Schritt 2.

Beachte:

Durch Multiplikation mit A können Bitshifts und Bit-Vertauschungen (Twists) realisiert werden.

Beispiele für modulo 2 lineare Generatoren

- Linear Feedback Shift Register Generator
- Generalized Feedback Shift Register Generator
- Mersenne-Twister (häufig standardmäßig in Software verwendet (z.B. R)) siehe u.a. M. Matsumoto und Nishimura (1998)

Mersenne Twister I

- Generiert PZZ $\mathbf{x} \in \{0, 1, \ldots, 2^{w-1}\}$.
- Division durch $2^w - 1$ ergibt reelle Zahl in $[0, 1]$
- Initialisierungs-Seeds $\mathbf{x}_0, \ldots, \mathbf{x}_n$
- Rekursionsvorschrift: $\mathbf{x}_{k+n} = \mathbf{x}_{k+m} \oplus (x_{k,w-1}, \ldots, x_{k,r}, x_{k+1,r-1}, \ldots, x_{k+1,0})A$, $k = 0, 1, \ldots$, wobei
- $\oplus$ bitweise Addition
- $m, r \in \mathbb{N}$, $0 \leq r \leq w - 1$,
- $\mathbf{a} = (a_{w-1}, \ldots, a_0) \in \{0, 1\}^w$

$$A = \begin{pmatrix} 0 & 1 & & & & \\ & 0 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \ddots & \\ & & & & 0 & 1 \\ a_{w-1} & a_{w-2} & \ldots & \ldots & a_1 & a_0 \end{pmatrix}$$

Mersenne Twister II

Beachte:

$$\mathbf{x}A = \begin{cases} \mathbf{x} >> 1, & \text{wenn } x_0 = 0, \\ (x >> 1) \oplus \mathbf{a}, & \text{wenn } x_0 = 1, \end{cases}$$

wobei $\mathbf{x} >> u$ ein bitweiser Rechts-Shift, um $u \in \mathbb{N}$ Positionen ist.

- Wenn $r = 0$ und A durch die Einheitsmatrix ersetzt wird, ergibt der obige Algorithmus den sog. Generalized Feedback Shift Operator
- Die Rekursionsvorschrift des Mersenne-Twisters kann allein durch Bit-Shifts, bitweises XOR, OR und AND realisiert werden
  $\Rightarrow$ sehr schnelle Operationen
- Zur Verbesserung der Güteeigenschaften der PZZ wird der vorläufige Output $\mathbf{x}$ des Mersenne-Twisters wie folgt modifiziert:
  - $\mathbf{y} = \mathbf{x} \oplus (\mathbf{x} >> u)$,
  - $\mathbf{y} = \mathbf{y} \oplus \big((\mathbf{y} << s) \text{ AND } \mathbf{b}\big)$,
  - $\mathbf{y} = \mathbf{y} \oplus \big((\mathbf{y} << t) \text{ AND } \mathbf{c}\big)$,
  - $\mathbf{z} = \mathbf{y} \oplus (y >> \ell)$.

Mersenne Twister MT 19937

- von M. Matsumoto und Nishimura (1998) eingeführte Parameterkonstellation für den MT:
  - $(w, n, m, r) = (32, 624, 397, 31)$,
  - $(\ell, s, t, u) = (18, 7, 15, 11)$,
  - Bitmasken $\mathbf{b} = 9D2C5680_{16}$, $\mathbf{c} = EFC60000_{16}$ (als Hexadezimalzahlen).
- extrem große Periode $2^{19937} - 1$ (dies ist eine Mersenne-Primzahl),
- sehr schnell, generiert immer 624 Zustandswörter mit je 32 Bits auf einmal,
- in unmodifizierter Form nicht für kryptographische Zwecke geeignet,
- sehr gute statistische Qualität der PZZs (für Details siehe M. Matsumoto und Nishimura (1998))

Java

- Klasse *java.util.Random*:
  - LKG mit 48-bit Keim
  - Keim im Default-Fall: Systemzeit in Millisekunden zur Zeit der Initialisierung des Random-Objektes
- Beispiele für Methoden:
  nextDouble() erzeugt eine SPZZ in $(0, 1)$,
  nextGaussian() erzeugt eine $N(0, 1)$-verteilte PZZ.
- Vorsicht!
  - Bei zeitnaher Erzeugung mehrerer Objekte vom Typ Random haben diese evtl. den gleichen Keim.
  - $\Rightarrow$ Alle Objekte liefern die gleiche Folge von PZZ.
  - Lösung: Für die zeitnahe Erzeugung von PZZ nur ein Random-Objekt anlegen und aus diesem alle PZZ generieren.
- Klasse *java.security.SecureRandom*:
  - PZZ-Generator für kryptographische Anwendungen
  - ähnliche Methoden wie *java.util.Random* aber bessere PZZ

R

- Standard ZZ-Generator: Mersenne-Twister
- k SPZZ ("Realisierungen" von k unabh. $U((0,1))$-verteilten ZV) erhält man mittels *runif(k)*
- k standardnormalverteilte PZZ erhält man mittels *rnorm(k)*
- Zur Änderung des ZZ-Generators siehe Hilfe zu "RNG".

Transformation von SPZZ

- Bisher: Erzeugung von Standard-Pseudo-Zufallszahlen (SPZZ)
  $\Rightarrow$ SPZZ können als Realisierungen von unabhängig $U([0,1))$-verteilten ZV aufgefasst werden.
- Ziel: Methoden für die Transformation von SPZZ
  $\Rightarrow$ transformierte Zahlenfolge soll als Realisierungen unabhängiger ZV einer anderen Verteilung betrachtet werden können, z.B.
  - $\mathrm{Exp}(\lambda)$
  - $\mathrm{Poi}(\lambda)$
  - $\mathrm{Bin}(n,p)$
  - $\mathrm{N}(\mu,\sigma)$

Inversionsmethode

Definition

Sei $F : \mathbb{R} \to [0, 1]$ eine beliebige Verteilungsfunktion, d.h. F ist

- rechtsstetig,
- monoton wachsend,
- $\lim_{x \to -\infty} F(x) = 0$ und $\lim_{x \to \infty} F(x) = 1$.

Dann heisst $F^{-1} : (0, 1] \to \mathbb{R} \cup \{\infty\}$

$$F^{-1}(y) = \inf\{x : \ F(x) \geq y\}$$

die verallgemeinerte Inverse von F.

Im Falle der Invertierbarkeit von F ist F^{-1} die normale Inverse.

Inversionsmethode

Theorem

Seien $U_1, U_2, \ldots \sim U([0,1))$ unabhängig. Sei F eine Verteilungsfunktion. Dann sind die ZV

$$X_i = F^{-1}(U_i),\ i \in \mathbb{N}$$

unabhängig mit Verteilungsfunktion F.

Direkte Anwendung des Theorems erfordert analytische Formel für $F^{-1} \Rightarrow$ nur selten möglich.

Beispiele: Exponentialverteilung mit Parameter λ

- $F(x) = (1 - e^{-\lambda x})\mathbb{1}_{\{x \geq 0\}}$
- $\Rightarrow F^{-1}(u) = -\frac{\log(1-u)}{\lambda} \; \forall u \in [0, 1)$
- Weil $U \sim U(0, 1] \Rightarrow 1 - U \sim U[0, 1)$:
  Wenn $u_1, \ldots, u_n$ als Realisierungen von unabh. $U((0, 1])$-verteilten ZV betrachtet werden können, dann können

$$x_i = -\frac{\log u_i}{\lambda}, \; i = 1, \ldots, n$$

als Realsierungen von unabh. Exp(λ)-verteilten ZV aufgefasst werden.

Beispiele: Erlangverteilung

- Erlang-Verteilung: $\Gamma(\lambda, r)$-Verteilung, $\lambda > 0$, $r \in \mathbb{N}$
- $F(x) = \int_0^x \frac{\lambda e^{-\lambda v}(\lambda v)^{r-1}}{(r-1)!} dv \mathbb{1}_{\{x \geq 0\}}$
- $\Rightarrow$ keine analytische Formel für F^{-1}
- Alternativansatz: $X_1, \ldots, X_r$ unabh. Exp(λ)-verteilt
  $\Rightarrow \sum_{i=1}^r X_i \sim \Gamma(\lambda, r)$.

Algorithmus

- Generiere rn SPZZ $u_1 \ldots, u_{rn}$
- Definiere $y_i = -(\frac{\log(u_{r(i-1)+1})}{\lambda} + \ldots + \frac{\log(u_{ri})}{\lambda})$
- Dann können $y_1 \ldots, y_n$ als Realisierungen unabh. Erlang-verteilter ZV betrachtet werden

Simulation $N(\mu, \sigma^2)$-verteilter ZV: Box-Muller-Algorithmus

Lemma

Seien U_1, U_2 unabh. $U([0,1))$-verteilte ZV. Dann sind die ZV

$$Y_1 = \sqrt{-2\log U_1}\cos(2\pi U_2) \text{ und } Y_1 = \sqrt{-2\log U_1}\sin(2\pi U_2)$$

unabh. und $N(0,1)$-verteilt.

Beweis: siehe Skript *Markov-Chains and Monte Carlo Simulation* von Prof. V. Schmidt

Algorithmus

- Erzeuge $2n$ SPZZ $u_1, \ldots, u_{2n}$.
- Definiere

$$y_{2k-1} = \sqrt{-2\log u_{2k-1}}\cos(2\pi u_{2k}) \text{ und } y_{2k} = \sqrt{-2\log u_{2k-1}}\sin(2\pi U_{2k})$$

- Für $\mu \in \mathbb{R}$, $\sigma > 0$ können $y'_{2k-1} = \sigma y_{2k-1} + \mu$, $y'_{2k} = \sigma y_{2k} + \mu$, $k = 1, \ldots, n$ als Realisierungen von $2n$ unabh. $N(\mu, \sigma^2)$-verteilten ZV betrachtet werden.

Simulation diskreter Verteilungen: generelle Methode

- **Ziel:** Erzeuge PZZ $x_1 \ldots, x_n$, die als Realisierungen unabh. diskreter ZV $X_1, \ldots, X_n : \Omega \to \{a_0, a_1, \ldots\} \subset \mathbb{R}$ betrachtet werden können.
- Notation $p_j = P(X_i = a_j)$
- Wenn $U_i \sim U([0,1))$, dann gilt für

$$X_i = \begin{cases} a_0, \text{ wenn } U_i < p_0 \\ a_1, \text{ wenn } p_0 \leq U_i < p_0 + p_1 \\ \vdots \\ a_j, \text{ wenn } p_0 + \ldots + p_{j-1} \leq U_i < p_0 + \ldots + p_j \\ \vdots \end{cases}$$

$P(X_i = a_j) = p_j,\ i = 1, \ldots, n.$

Simulation diskreter Verteilungen: generelle Methode

Algorithmus

- Erzeuge SPZZ $u_1 \ldots, u_n$.
- Definiere

$$x_i = \sum_{j=0}^{\infty} a_j \mathbb{1}_{[\sum_{k=0}^{j-1} p_k, \sum_{k=0}^{j} p_k)}(u_i), \ i = 1, \ldots, n.$$

- Dann können $x_1, \ldots, x_n$ als Realisierungen einer Folge unabh. ZV der durch die Zähldichte $p_0, p_1 \ldots$ definierten Verteilung betrachtet werden.

Poisson-Verteilung

Lemma

Seien $X_1, X_2, \ldots$ unabh. und Exp(λ)-verteilte ZV. Dann gilt

$$Y = \max\{k \geq 0 : \ X_1 + \ldots + X_k \leq 1\} \sim \text{Poi}(\lambda).$$

Beweis: Übungsaufgabe

Algorithmus

- Seien $u_1, u_2, \ldots$ SPZZ.
- Setze $y_0 = 0$ und für $i > 0$
  $y_i = \max\{k \geq 0 : \ \frac{-\log u_1}{\lambda} + \ldots + \frac{-\log u_k}{\lambda} \leq i\} - y_{i-1}$.

Dann kann $y_1, y_2, \ldots$ als Folge von Realisierungen unabh. Poi(λ)-verteilter ZV betrachtet werden.

Akzeptanz- und Verwerfungsmethode

- eine der universellsten Methoden, um PZZ mit einer bestimmten Verteilung zu erzeugen
- für diskrete und absolutstetige Vertilungen anwendbar
- Ausgangspunkt im diskreten Fall:
  - Verteilung mit Zähldichte $\mathbf{p} = (p_0, p_1 \ldots)$, die bereits simuliert werden kann
  - Zähldichte $\mathbf{q} = (q_0, q_1 \ldots)$ der zu simulierenden Verteilung erfüllt

$$q_k \leq c\, p_k \text{ für alle } k \in \mathbb{N} \text{ und eine Konstante } \mathrm{c} \geq 1.$$

- Ausgangspunkt im absolutstetigen Fall:
  - Verteilung mit Dichte $g(x)$, die bereits simuliert werden kann
  - Dichte $f(x)$ der zu simulierenden Verteilung erfüllt

$$f(x) \leq c\, g(x) \text{ für alle } x \in \mathbb{R}^{\mathrm{m}} \text{ und eine Konstante } \mathrm{c} \geq 1.$$

Diskreter Fall

Theorem

Seien $(U_1, X_1), (U_2, X_2), \ldots$ iid Zufallsvektoren mit unabhängigen Komponenten, so dass

$$U_i \sim U((0,1]) \text{ und } X_i \sim \mathbf{p}.$$

Es gelte ferner $q_k \leq c\, p_k$ für alle $k \in \mathbb{N}$ und eine Konstante $\mathrm{c} \geq 1$.

Dann gilt $I = \min\left\{ k \geq 1 \ : \ U_k < \frac{q_{X_k}}{c p_{X_k}} \right\} \sim Geo(c^{-1})$ und $X_I \sim \mathbf{q}$.

Beweis: siehe Skript *Markov-Chains and Monte Carlo Simulation* von Prof. V. Schmidt

Algorithmus

1. Generiere eine Realisierung x einer ZV X mit Zähldichte $\mathbf{p}$.
2. Generiere eine Realisierung u einer von X unabhängigen ZV $U \sim U((0,1])$.
3. Wenn $u \leq \frac{q_{X_k}}{c p_{X_k}}$, dann ist x das Simulationsergebnis, ansonsten gehe wieder zu Schritt 1.

Absolutstetiger Fall

Theorem

Seien $(U_1, X_1), (U_2, X_2), \ldots$ iid Zufallsvektoren mit unabhängigen Komponenten, so dass

$U_i \sim U((0,1])$ und die Z-Vektoren $X_i : \Omega \to \mathbb{R}^{\mathrm{m}}$ die Dichte $\mathrm{g} : \mathbb{R}^{\mathrm{m}} \to [0, \infty)$ haben. Es gelte ferner für die Dichte $f : \mathbb{R}^{\mathrm{m}} \to [0, \infty)$, dass $f(x) \leq c\, g(x)$ für alle $x \in \mathbb{R}^{\mathrm{m}}$ und eine Konstante $\mathrm{c} \geq 1$.

Dann gilt $I = \min\left\{k \geq 1 \ : \ U_k < \frac{f(X)}{cg(X)}\right\} \sim Geo(c^{-1})$ und X_I hat die Dichte f.

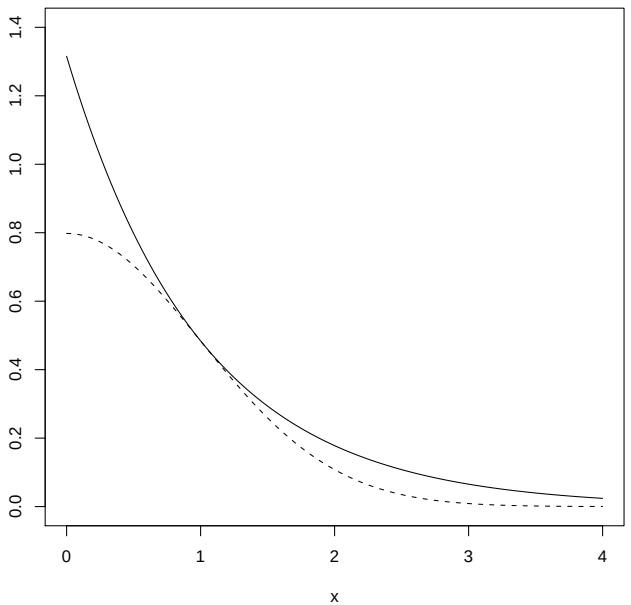
Beweis: siehe Skript *Markov-Chains and Monte Carlo Simulation* von Prof. V. Schmidt

Algorithmus

1. Generiere eine Realisierung x eines Zufallsvektors X mit der Vorschlagsdichte g.
2. Generiere eine Realisierung u einer von X unabhängigen ZV $U \sim U((0,1])$.
3. Wenn $u \leq \frac{f(x)}{c\, g(x)}$, dann ist x das Simulationsergebnis, ansonsten gehe wieder zu Schritt 1.

Beispiel: Positive Normalverteilung

- Dichte der positiven Standard-Normalverteilung: $f(x) = \sqrt{\frac{2}{\pi}} e^{-x^2/2} \mathbb{1}_{\{x \geq 0\}}$
- Vorschlagsdichte: $e^{-x} \mathbb{1}_{\{x \geq 0\}}$ (der $Exp(1)$-Verteilung)
- Man sieht leicht, dass $c = \sqrt{\frac{2e}{\pi}}$ die kleinste Konstante ist, so dass $f(x) \leq c\, g(x)$ für alle $x \geq 0$.



Dichte der positiven Normalverteilung (gestrichelte Kurve) und dominierende Funktion $\sqrt{\frac{2}{\pi}} e^{-x+\frac{1}{2}}$ (durchgezogene Linie)

Beispiel: Positive Normalverteilung

Algorithmus zur Simulation einer positiv normalverteilten ZV

1. Generiere eine Realisierung x einer ZV $X \sim Exp(1)$.
2. Generiere eine Realisierung u einer von X unabhängigen ZV $U \sim U((0,1])$.
3. Wenn $u \leq \exp\left(-\frac{x^2}{2} + x - \frac{1}{2}\right)$, dann ist x das Simulationsergebnis, ansonsten gehe wieder zu Schritt 1.

Beachte: Wenn X positiv standardnormalverteilt ist und $U \sim U((0,1])$ unabhängig von X, dann ist

$$S = X(1 - 2\mathbb{1}_{\{U \leq 1/2\}}) \text{standardnormalverteilt.}$$

Monte-Carlo-Integration

Ziel

Berechnung eines Integrals der Form $\int_B f(x)dx$, für eine Borelmenge $B \subset \mathbb{R}^d$.

- Benutze: $I = \frac{1}{|B|} \int_B f(x)dx = \mathbb{E}f(X)$, wobei $X \sim U(B)$.
- Approximiere $I = \frac{1}{|B|} \int_B f(x)dx$ durch

$$\hat{I}_n = \frac{1}{n} \sum_{i=1}^{n} f(X_i) \text{ mit } X_1, \ldots, X_n \sim U(B) \text{ unabhängig.}$$

Eigenschaften von $\hat{I}_n$

- $\hat{I}_n$ ist erwartungstreu, denn

$$\mathbb{E}\hat{I}_n = \mathbb{E}\frac{1}{n}\sum_{i=1}^{n} f(X_i) = \mathbb{E}f(X_1) = I.$$

- $\hat{I}_n$ ist stark konsistent, denn nach dem SGGZ gilt

$$\frac{1}{n}\sum_{i=1}^{n} f(X_i) \xrightarrow{f.s.} \mathbb{E}f(X_1) = I \ (n \to \infty).$$

- $\hat{I}_n$ ist asymptotisch normalverteilt, falls $\mathbb{E}f(X_1)^2 < \infty$ d.h.

$$n\frac{\hat{I}_n - I}{\sqrt{\sum_{i=1}^{n}(f(X_i) - \hat{I}_n)^2}} \xrightarrow{d} Y \sim N(0,1).$$

Das folgt aus dem Lemma von Slutsky, da $\frac{1}{n}\sum_{i=1}^{n}(f(X_i) - \hat{I}_n)^2 \xrightarrow{f.s.} \operatorname{Var} f(X_1)$ und nach dem ZGWS

$$\sqrt{n}\frac{\hat{I}_n - \mathbb{E}f(X_1)}{\sqrt{\operatorname{Var} f(X_1)}} = \sqrt{n}\frac{\hat{I}_n - I}{\sqrt{\operatorname{Var} f(X_1)}} \xrightarrow{d} Y \sim N(0,1).$$

Konfidenzintervalle für $\hat{I}_n$

Durch die asymptotische Normalverteiltheit von $\hat{I}_n$, ergibt sich für I das asymptotische Konfidenzintervall

$$\left(\hat{I}_n - z_{1-\frac{\alpha}{2}} \frac{\sqrt{\sum_{i=1}^{n} \left(f(X_i) - \hat{I}_n\right)^2}}{n}, \; \hat{I}_n + z_{1-\frac{\alpha}{2}} \frac{\sqrt{\sum_{i=1}^{n} \left(f(X_i) - \hat{I}_n\right)^2}}{n} \right)$$

zum Niveau $1 - \alpha$, wobei $\alpha \in (0, 1)$ und $z_{1-\frac{\alpha}{2}}$ das $1 - \frac{\alpha}{2}$-Quantil der Standardnormalverteilung bezeichnet.

Literatur

- D.P. Kroese, T. Taimre and Z.I. Botev *Handbook of Monte Carlo Methods*, 2011, J. Wiley & Sons
- H. Künsch *Stochastische Simulation* Vorlesungsmanuskript, ETH Zürich, April 2006. (ftp://stat.ethz.ch/U/Kuensch/skript-sim.ps)
- M. Matsumoto und T. Nishimura, *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator* ACM Transactions on Modeling and Computer Simulation 1998, 8(1): 3-30.
- V. Schmidt, *Markov Chains and Monte-Carlo Simulation* Vorlesungsmanuskript, Universität Ulm, Juli 2006.