ulm university universität
uulm
Institute of Stochastics

Grundlagen der Monte-Carlo-Simulation

Prof. Dr. V. Schmidt und S. Lück |
8. November 2007

Contents

Motivation für Monte-Carlo-Simulation

- Ziel:
  Analyse stochastischer Modelle, die zu komplex sind, um ihr Verhalten analytisch studieren zu können.

Motivation für Monte-Carlo-Simulation

- Ziel:
  Analyse stochastischer Modelle, die zu komplex sind, um ihr Verhalten 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

Motivation für Monte-Carlo-Simulation

- Ziel:
  Analyse stochastischer Modelle, die zu komplex sind, um ihr Verhalten 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
- $\Rightarrow$ 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.

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 Verwefungsmethoden

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 Verwefungsmethoden
- 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.

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 $m_0 > 0 \Rightarrow z_{k+j} = z_{k+m_0+j}\ \forall j \geq 0$.

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 $5, 0, 5, 0, \ldots$.

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.

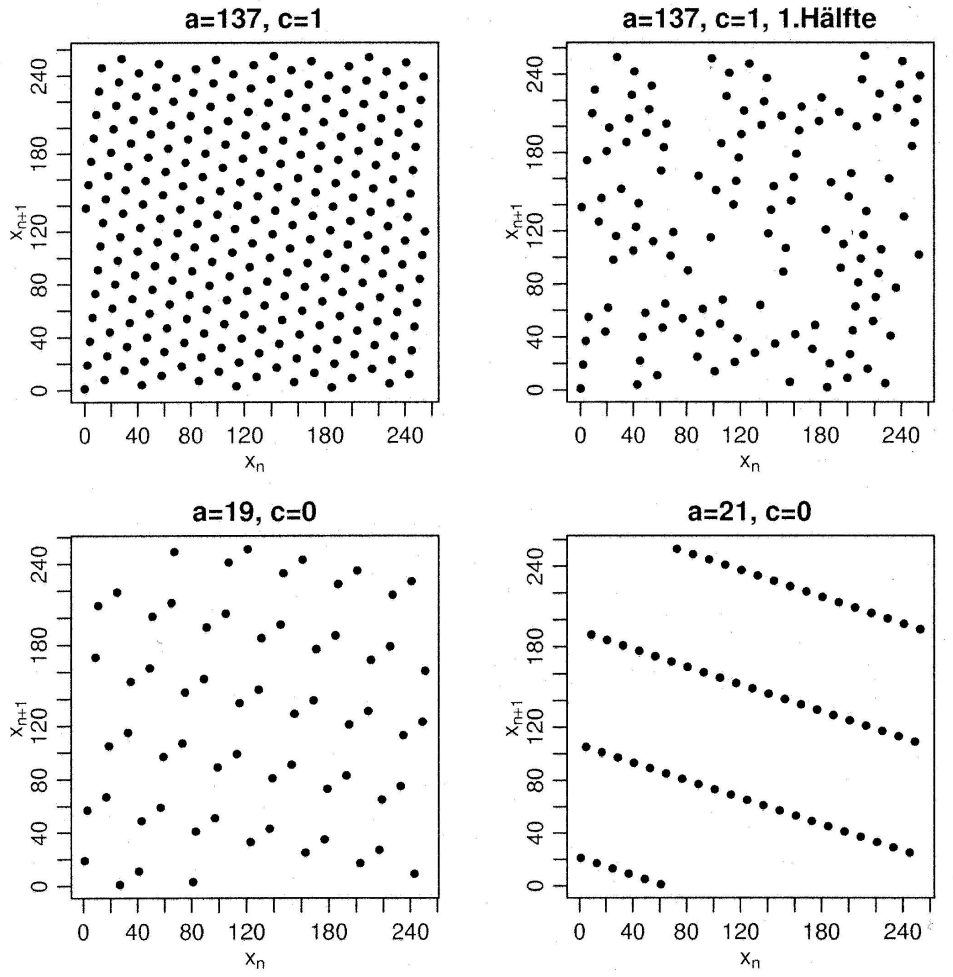
Bemerkungen

- 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.
- Statistische Tests für die Güte von SPZZ siehe Statistik I und II $\rightsquigarrow$ χ^2-Anpassungstest etc.

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.
- Statistische Tests für die Güte von SPZZ siehe Statistik I und II $\rightsquigarrow$ χ^2-Anpassungstest etc.
- weitere wünschenswerte Eigenschaft für SPZZ $u_1, \ldots, u_n$:
  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 (u_i, u_{i+1}) aufeinanderfolgender SPZZ für $m = 256$



Quelle: Vorlesungsmanuskript von H. Künsch
(ftp://stat.ethz.ch/U/Kuensch/skript-sim.ps)

Fazit

- LKGs sind einfach zu implementierende und schnelle Algorithmen

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.

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.
- andere SPZZ-Generatoren:
  - Nichtlineaere Kongruenzgeneratoren
  - Schieberegistergeneratoren
  - Fibonacci-Generatoren
  - Mersenne-Twister (schneller Algorithmus, statistisch sehr gute SPZZ, in Reinform nicht für kryptographische Zwecke geeignet)
  - Blum-Blum-Shub (für kryptographische Anwendungen)

Java

- Klasse *java.util.Random*:
  - LKG mit 48-bit Keim
  - Keim im Default-Fall: Systemzeit in Millisekunden zur Zeit der Initialisierung des Random-Objektes

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.

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.

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

R

- Standard ZZ-Generator: Mersenne-Twister
- k SPZZ ("Realisierungen" von k unabh. $U((0,1))$-verteilten ZV) erhält man mittels *runif(k)*

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)*

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.

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.

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

- **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.
- **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.

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.
- **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

- $F(x) = (1 - e^{-\lambda x})\mathbb{1}_{\{x \geq 0\}}$

Beispiele: Exponentialverteilung mit Parameter λ

- $F(x) = (1 - e^{-\lambda x})\mathbb{1}_{\{x \geq 0\}}$
- $\Rightarrow F^{-1}(u) = -\frac{\log(1-u)}{\lambda} \; \forall u \in [0, 1)$

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. $U([0, 1))$-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, $\lambda > 0$, $r \in \mathbb{N}$

Beispiele: Erlangverteilung

- Erlang-Verteilung: $\Gamma(\lambda, r)$-Verteilung, $\lambda > 0$, $r \in \mathbb{N}$
- $F(x) = \int_0^x \frac{\lambda e^{-\lambda v}(\lambda v)^{r-1}}{(r-1)!} dv \mathbb{1}_{\{x \geq 0\}}$

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}

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}
- Lösung: $X_1, \ldots, X_r$ unabh. Exp(λ)-verteilt
  $\Rightarrow \sum_{i=1}^r X_i \sim \Gamma(\lambda, r)$.

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}
- Lösung: $X_1, \ldots, X_r$ 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: Übungsaufgabe

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: Übungsaufgabe
- 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

$$y_1' = \sigma_1 + \mu, \ldots, y_2' = \sigma_n + \mu$$

    als Realisierungen unabh. $N(\mu, \sigma^2)$-verteilter 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.

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.
- Definiere $p_j = P(X_i = a_j)$.

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.
- Definiere $p_j = P(X_i = a_j)$.
- Wenn $U \sim U([0,1))$, dann gilt für

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

$P(X = a_j) = p_j$

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 $x_1, \ldots, x_n$ Realsierungen einer Folge unabh. ZV der durch $p_0, p_1 \ldots$ definierten Verteilung betrachtet werden.

Poisson-Verteilung

- **Lemma:** Seien $X_1, X_2, \ldots$ unabh. und $\mathrm{Exp}(\lambda)$-verteilte ZV. Dann gilt

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

Beweis: Übungsaufgabe

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.

Literatur

Diese Präsentation basiert auf einem Teil des Skriptes
"Markov Chains and Monte-Carlo Simulation" von Prof. Dr. Volker Schmidt, Universität Ulm, Juli 2006.