

Projektaufgaben Block 1

Carlo Michaelis, 573479; Lukas Ruff, 572521

15 November 2016

1 Infinite-Monkey-Theorem

1.1 Formulierung und Beweis des Infinite-Monkey-Theorems

Laut dem Infinite-Monkey-Theorem wird ein Affe, der unendlich lange zufällig auf einer Schreibmaschine tippt, fast sicher jede beliebige Zeichenkette unendlich oft schreiben. Diese bildhafte Interpretation des mathematischen Satzes soll der gedanklichen Einordnung von Unendlichkeit dienen. Das Infinite-Monkey-Theorem ist ein Beispiel für die Anwendung des Lemmas von Borel-Cantelli.

Satz 1 (Lemma von Borel-Cantelli). *Sei (Ω, \mathcal{F}, P) ein Wahrscheinlichkeitsraum.*

a) *Ist $(A_n)_{n \in \mathbb{N}}$ eine Folge von Ereignissen mit $\sum_{n \in \mathbb{N}} P(A_n) < \infty$, so gilt*

$$P\left(\limsup_{n \rightarrow \infty} A_n\right) = 0.$$

b) *Ist $(A_n)_{n \in \mathbb{N}}$ eine Folge von unabhängigen Ereignissen mit $\sum_{n \in \mathbb{N}} P(A_n) = +\infty$, so gilt*

$$P\left(\limsup_{n \rightarrow \infty} A_n\right) = 1.$$

Um das Infinite-Monkey-Theorem formulieren zu können, modellieren wir zunächst einen geeigneten Wahrscheinlichkeitsraum. Definiere dazu ein endliches Alphabet $\Omega := \{a, b, c, \dots\}$, d.h. Ω ist eine Menge von Zeichen (Buchstaben, Satzzeichen, Zahlen, etc.) mit $|\Omega| = n$ für ein $n \in \mathbb{N}$. Setze weiter die zugehörige σ -Algebra als Potenzmenge $\mathcal{F} := \mathcal{P}(\Omega)$ und definiere das Wahrscheinlichkeitsmaß P als diskrete Gleichverteilung, d.h. $P(A) = \frac{|A|}{|\Omega|} = \frac{|A|}{n}$ für $A \in \mathcal{F}$. Somit modelliert (Ω, \mathcal{F}, P) das Ziehen von Zeichen aus einem Alphabet mit gleicher Wahrscheinlichkeit. Den Übergang zu Zeichenketten (Folgen von Zeichen aus dem Alphabet Ω) können wir nun mittels des (stets existierenden und eindeutigen) Produktraumes $(\Omega^{\mathbb{N}}, \mathcal{F}^{\mathbb{N}}, \mathbb{P})$ mit $\mathbb{P} := P^{\mathbb{N}}$ bewerkstelligen. Hiermit können wir das Infinite-Monkey-Theorem wie folgt formulieren und beweisen:

Satz 2 (Infinite-Monkey-Theorem). *Betrachte den oben definierten Produktraum $(\Omega^{\mathbb{N}}, \mathcal{F}^{\mathbb{N}}, \mathbb{P})$ von (unendlich langen) Zeichenketten und sei $s = (s_1, \dots, s_m)^{\top} \in \Omega^m$ eine beliebige Zeichenkette der Länge $m \in \mathbb{N}$ (String). Definiere zu $k \in \mathbb{N}$ das Ereignis*

$$A_k := \left\{ \omega = (\omega_j)_{j \in \mathbb{N}} \in \Omega^{\mathbb{N}} : \omega_k = s_1, \dots, \omega_{k+m-1} = s_m \right\},$$

d.h. A_k ist das Ereignis, dass String s der Länge m an der Stelle k in einer Zeichenkette beginnt. Dann ist $(A_{jm+1})_{j \in \mathbb{N}_0}$ eine Folge unabhängiger Ereignisse und es gilt

$$\mathbb{P}\left(\limsup_{j \rightarrow \infty} A_{jm+1}\right) = 1.$$

Beweis. Es sei $\pi_i : \Omega^{\mathbb{N}} \rightarrow \Omega, (\omega_j)_{j \in \mathbb{N}} \mapsto \omega_i$ die i -te Koordinatenprojektion auf dem kartesischen Produkt $\Omega^{\mathbb{N}}$. Weiter sei $(I_j)_{j \in \mathbb{N}_0}$ eine Zerlegung von \mathbb{N} in disjunkte Blöcke der Länge m , d.h.

$$\mathbb{N} = \bigcup_{j \in \mathbb{N}_0} I_j \quad \text{mit} \quad I_j := \bigcup_{k=1}^m \{jm + k\}, \quad j \in \mathbb{N}_0.$$

Dann folgt für $j \in \mathbb{N}_0$

$$\begin{aligned}\mathbb{P}(A_{jm+1}) &= \mathbb{P}(\{\omega : \omega_{jm+1} = s_1, \dots, \omega_{(j+1)m} = s_m\}) \\ &= \mathbb{P}\left(\bigcap_{k=1}^m \pi_{jm+k}^{-1}(\{s_k\})\right) \\ &= \prod_{k=1}^m P(\{s_k\}) = \left(\frac{1}{n}\right)^m,\end{aligned}$$

da $\mathbb{P} = P^{\mathbb{N}}$ Produktmaß ist. Weiter folgt für $j_1, j_2 \in \mathbb{N}_0, j_1 \neq j_2$:

$$\begin{aligned}\mathbb{P}(A_{j_1m+1} \cap A_{j_2m+1}) &= \mathbb{P}(\{\omega : \omega_{j_1m+1} = s_1, \dots, \omega_{(j_1+1)m} = s_m\} \cap \{\omega : \omega_{j_2m+1} = s_1, \dots, \omega_{(j_2+1)m} = s_m\}) \\ &= \mathbb{P}\left(\left(\bigcap_{k=1}^m \pi_{j_1m+k}^{-1}(\{s_k\})\right) \cap \left(\bigcap_{k=1}^m \pi_{j_2m+k}^{-1}(\{s_k\})\right)\right) \\ &= \left(\prod_{k=1}^m P(\{s_k\})\right)^2 = \left(\frac{1}{n}\right)^m \left(\frac{1}{n}\right)^m = \mathbb{P}(A_{j_1m+1}) \cdot \mathbb{P}(A_{j_2m+1}),\end{aligned}$$

da $I_{j_1} \cap I_{j_2} = \emptyset$ für $j_1 \neq j_2$. Somit ist $(A_{jm+1})_{j \in \mathbb{N}_0}$ eine Folge unabhängiger Ereignisse mit

$$\sum_{j \in \mathbb{N}_0} P(A_{jm+1}) = \sum_{j \in \mathbb{N}_0} \left(\frac{1}{n}\right)^m = +\infty.$$

D.h. mit Teil b) des Lemma von Borel-Cantelli folgt die Behauptung. □

Es gilt also

$$\begin{aligned}\mathbb{P}\left(\limsup_{j \rightarrow \infty} A_{jm+1}\right) &= \mathbb{P}\left(\bigcap_{j \in \mathbb{N}_0} \bigcup_{k \geq j} A_{km+1}\right) \\ &= \mathbb{P}(\{\omega \in \Omega^{\mathbb{N}} : \forall j \in \mathbb{N}_0 \exists k \geq j : \omega \in A_{km+1}\}) \\ &= \mathbb{P}(\{\omega \in \Omega^{\mathbb{N}} : \omega \in A_{jm+1} \text{ für unendlich viele } j \in \mathbb{N}_0\}) = 1,\end{aligned}$$

d.h. ein beliebiger String s der Länge m erscheint fast sicher unendlich oft.

1.2 Simulation eines Infinite-Monkey Experiments

In diesem Abschnitt wollen wir das Infinite-Monkey-Theorem experimentell simulieren. Wir schreiben hierzu eine R-Funktion die solange zufällig Zeichen aus dem Alphabet $\Omega = \{a, b, \dots, y, z\}$ mit gleicher Wahrscheinlichkeit (d.h. $\frac{1}{26}$) auswählt, bis eine vorgegebene Zeichenkette $s = (s_1, \dots, s_m)^{\top} \in \Omega^m, m \in \mathbb{N}$, vollständig erschienen ist. Die R-Funktion gibt dann die Anzahl der bis zum Erscheinen der Zeichenkette s gezogenen Zeichen zurück. Damit simulieren wir die Zufallsvariable $X : \Omega^{\mathbb{N}} \rightarrow \mathbb{N}$ mit

$$X(\omega) = \min\{k + m - 1 : \omega_k = s_1, \dots, \omega_{k+m-1} = s_m \text{ für } k \in \mathbb{N}\}$$

Wir haben die Funktion wie folgt in R implementiert:¹

¹Eine effizientere Implementierung wäre je Schleifendurchlauf eine größere Anzahl an Zeichen zu generieren und den resultierenden Zeichen-Block mit einem Fenster nach `strTarget` zu durchsuchen. Dabei muss beachtet werden, dass `strTarget` auch in der Überlappung zweier Blöcke erscheinen könnte. Für unsere Untersuchung ist eine vereinfachte (ineffiziente) Implementierung jedoch ausreichend. Für große Samples oder einen langen String `strTarget` sollte jedoch auf jeden Fall eine effizientere Implementierung herangezogen werden.

```

fnInfiniteMonkey <- function(strTarget) {
  # This function is a simulation of the Infinite-Monkey-Theorem. It generates a
  # random sequence of letters until a given target string appears.
  #
  # Args:
  #   strTarget: The target string which should be matched
  #
  # Returns:
  #   The number of generated letters until the target string appeared

  # Split target string to vector of chars
  vecCharTarget <- strsplit(strTarget, "")[[1]]
  # Get the number of letters in target string
  nTarget <- length(vecCharTarget)

  # Set counting variable (at least nTarget letters needed)
  nCounter <- nTarget

  # Switch on the monkey (i.e. sample the first nTarget letters)
  vecLetterSeqTail <- sample(letters, nTarget, replace = TRUE)

  # Let the monkey type until target string was written
  while(!identical(vecCharTarget, vecLetterSeqTail)) {

    # Let the monkey hit another key (sample next letter) and store in
    # vecLetterSeqTail (first in, first out)
    if (nTarget == 1) {
      vecLetterSeqTail <- sample(letters, 1)
    } else {
      vecLetterSeqTail <- c(vecLetterSeqTail[2:nTarget],
                           sample(letters, 1, replace = TRUE))
    }

    # Count
    nCounter <- nCounter + 1
  }

  # Return the length of the generated letter sequence
  return(nCounter)
}

```

In Figure 1 haben wir ein Histogramm der simulierten Anzahlen von Zeichen und die durchschnittliche Anzahl von Zeichen, bis die Zeichenkette 'ab' vollständig erschienen ist, für 10^4 Samples geplottet. Wir können sehen, dass die Anzahl von Zeichen X einer rechtsschiefen Verteilung folgt. Die durch das starke Gesetz der großen Zahlen motivierte Monte-Carlo-Approximation von $\mathbb{E}[X]$ liegt bei 681.68.

Zur Simulation von Infinite-Monkey Experimenten wollen wir abschließend noch folgende Überlegung durchführen: würden wir anstatt einer fortlaufenden Zeichenfolge, bei welcher je Schleife nur ein weiterer Letter generiert wird, je Schleifendurchlauf einen Zeichen-Block der Länge m generieren und mit s vergleichen, so entspräche jeder Schleifendurchlauf einem Bernoulli-Experiment mit Erfolgswahrscheinlichkeit $p = \left(\frac{1}{n}\right)^m$. In diesem Fall wäre die Anzahl der Blöcke, die notwendig sind um eine Übereinstimmung mit s zu erhalten, gerade geometrisch-verteilt mit Erfolgswahrscheinlichkeit $p = \left(\frac{1}{n}\right)^m$. Diese Zerlegung in Blöcke der Länge m entspricht genau der Folge unabhängiger Ereignisse aus dem Beweis des Theorems.

Target string: 'ab'; Mean length of letter sequence: 681.68

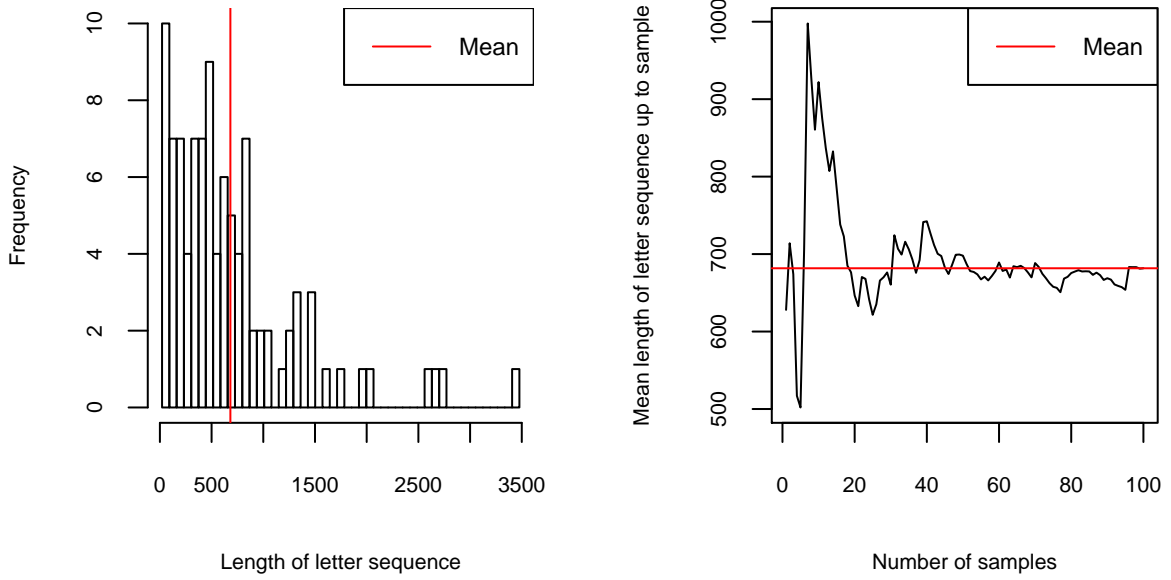


Figure 1: Histogram of letter sequence lengths and mean letter sequence length by number of samples.

2 Monte-Carlo-Approximationen

2.1 Näherung für π

Um eine Näherung für π zu erreichen, können in einem Quadrat zufällige Punkte erzeugt werden. Derjenige Anteil der Punkte, der innerhalb des Kreises liegt, ergibt π . Zunächst definieren wir die Zufallsvariablen, aus denen die Koordinaten gezogen werden:

$$X \sim \text{Uni}[-1, 1] \quad , \text{ d.h. } X_1 \sim \text{Uni}[-1, 1] \quad \text{und} \quad X_2 \sim \text{Uni}[-1, 1]$$

Dabei sind X_1 und X_2 unabhängig. Die gemeinsame Dichtefunktion ergibt sich aus den einzelnen Dichten und kann wie folgt beschrieben werden:

$$\begin{aligned} f(x_1, x_2) &= \frac{1}{4} \mathbb{1}_{[-1, 1]}(x_1) \mathbb{1}_{[-1, 1]}(x_2) \\ &= \frac{1}{2} \mathbb{1}_{[-1, 1]}(x_1) \frac{1}{2} \mathbb{1}_{[-1, 1]}(x_2) \\ &= f(x_1) f(x_2) \end{aligned}$$

π kann allgemein definiert werden als:

$$\pi = \int_{-1}^1 \int_{-1}^1 \mathbb{1}_S(x_1, x_2) dx_1 dx_2 \quad (1)$$

wobei $S = \{x_1, x_2 \mid x_1^2 + x_2^2 \leq 1\}$. Stochastisch ergibt sich π wiederum als Erwartungswert der Indikatorfunktion von der gleichverteilten Zufallsvariablen X auf der Menge der Kreispunkte:

$$\begin{aligned}
\mathbb{E}[\mathbb{1}_S(X)] &= \mathbb{E}[\mathbb{1}_S(X_1, X_2)] \\
&= \int_{\mathbb{R}^2} \mathbb{1}_S(x_1, x_2) f(x_1, x_2) d\lambda(x_1, x_2) \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbb{1}_S(x_1, x_2) \cdot \frac{1}{4} \mathbb{1}_{[-1,1]}(x_1) \mathbb{1}_{[-1,1]}(x_2) dx_1 dx_2 \quad (\text{Fubini}) \\
&= \frac{1}{4} \int_{-1}^1 \int_{-1}^1 \mathbb{1}_S(x_1, x_2) dx_1 dx_2 \\
&\stackrel{(1)}{=} \frac{1}{4} \pi
\end{aligned}$$

Durch Umstellung folgt damit direkt:

$$\pi = 4 \mathbb{E}[\mathbb{1}_S(X)] = 4 \mathbb{E}[\mathbb{1}_S(X_1, X_2)]$$

Im Folgenden Code ist die Berechnung mittels R dargestellt. Die Funktion `fnErrorEstimation` ist hier auf Grund ihrer Länge nicht explizit aufgeführt, sie ist in der Datei `2.R` zu finden.

```
fnPiRandom <- function(n) {
  # Generates random uniform values for x (x1) and y (x2) axis
  #
  # Args:
  #   n: Number of samples
  #
  # Returns:
  #   A list of x1 and x2 random values with size of n
  return(list(x1 = runif(n, min = -1, max = 1),
             x2 = runif(n, min = -1, max = 1)))
}

fnPiIndicator <- function(x1, x2) {
  # Indicate which values are inside of the circle with radius 1
  # and which are outside
  #
  # Args:
  #   x1: Random values of x-axis
  #   x2: Random values of y-axis
  #
  # Returns:
  #   A vector containing values 0, if point is outside of circle, and 4,
  #   if point is inside of circle with radius 1
  return(((x1^2 + x2^2) <= 1)*4)
}

fnPiPlotCircle <- function(x1, x2) {
  # Plots all random points.
  # Points which are inside are red, points outside blue and some circle-points
  # are calculated, to mark the border in black
  #
  # Args:
  #   x1: Random values of x-axis
  #   x2: Random values of y-axis
  #
}
```

```

# Returns:
#   Nothing, just prints the plot

inside <- as.logical(fnPiIndicator(x1, x2))
outside <- !inside

# Plot points inside and outside
plot(x1[inside], x2[inside], pch = 20, cex = 0.5, col = "red",
     xlab = "x", ylab = "y", xlim = c(-1,1), ylim = c(-1,1))
points(x1[outside], x2[outside], pch = 20, cex = 0.5, col = "blue")

# Plot circle points to mark the border
points(sin(1:10000), cos(1:10000), pch = 20, cex = 0.2, col = "black")
}

# Split plot panel
par(mfrow = c(1,2), ps = 9, cex.axis = 0.9, cex.lab = 0.9)

# Create random values
X <- fnPiRandom(2000)

# Plot random values and circle
fnPiPlotCircle(X$x1, X$x2)

# Run error estimation
fnErrorEstimation(X, fnPiRandom, fnValueCalculation = fnPiIndicator)

# Set title for plots
title(paste("Monte-Carlo simulation with sample size of ", length(X$x1),
           " results in pi value of ", mean(fnPiIndicator(X$x1, X$x2)), sep = ""),
      outer = TRUE, line = -2)

```

Die Ergebnisse sind in Darstellung 2 gezeigt. Auf der linken Seite sind die generierten Zufallswerte dargestellt. Die Werte, welche im inneren des Kreises (schwarz) gelandet sind, wurden rot markiert, Werte außerhalb blau. Auf der rechten Seite ist der kumulierte Mittelwert der Samples schwarz dargestellt. Die rote Linie markiert den Mittelwert über alle Samples und die blaue und grüne Linie markieren die zwei möglichen Methoden der Fehlerabschätzung (mittels Zentralem Grenzwertsatz und Simulation).

2.2 Approximation einer Wahrscheinlichkeit

Gegeben sei eine Zufallsvariable $X \sim N(0,1)$. Die Wahrscheinlichkeit $\mathbb{P}(X > 20)$ soll per Monte-Carlo-Simulation approximiert werden. Zunächst formulieren wir die Wahrscheinlichkeit:

$$\begin{aligned}
 P(X > 20) &= 1 - F(20) = \mathbb{E}[\mathbb{1}_{(X>20)}] \\
 &= \int_{-\infty}^{\infty} \mathbb{1}_{(X>20)} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx \\
 &= \int_{20}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx
 \end{aligned}$$

wobei F die Verteilungsfunktion der Standardnormalverteilung bezeichne.

Monte-Carlo simulation with sample size of 2000 results in pi value of 3.152

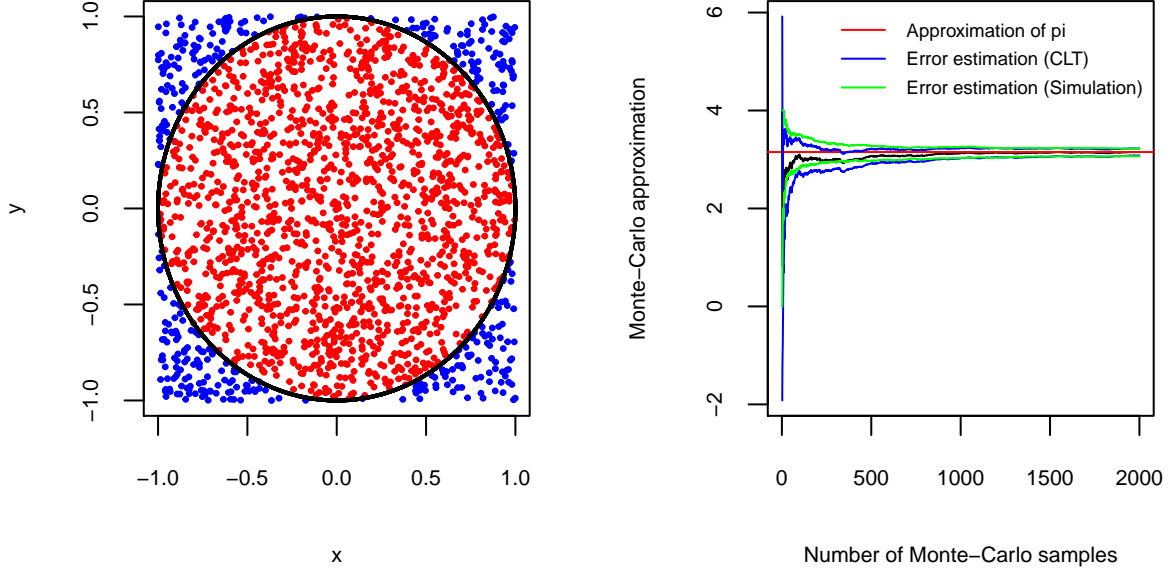


Figure 2: Simulation result (left) and error estimation (right).

Um Zufallszahlen generieren zu können, müssen die Grenzen endlich sein, daher verwenden wir die Transformation $Y = \frac{1}{X} \Leftrightarrow X = \frac{1}{Y}$ und damit $\frac{dx}{dy} = -\frac{1}{y^2}$. Damit folgt im Weiteren:

$$\begin{aligned} P(X > 20) &= \int_{20}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx \\ &= \int_0^{\frac{1}{20}} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2y^2}\right) \frac{1}{y^2} dy \\ &= \int_0^{\frac{1}{20}} \tilde{g}(y) dy = \frac{1}{20} \mathbb{E}[\tilde{g}(Y)] \end{aligned}$$

Dabei ist $Y \sim \text{Uni}[0, \frac{1}{20}]$ mit Dichtefunktion $f_Y(y) = 20 \mathbb{1}_{[0, \frac{1}{20}]}(y)$, wodurch auch der Vorfaktor $\frac{1}{20}$ vor dem Erwartungswert begründet ist. Um das deutlich zu machen, kann folgende Rück-Umformung bzgl. der Gleichverteilung betrachtet werden:

$$\begin{aligned} \frac{1}{20} \mathbb{E}[\tilde{g}(Y)] &= \frac{1}{20} \int_{-\infty}^{\infty} \tilde{g}(y) \cdot 20 f(y) dy \\ &= \int_{-\infty}^{\infty} \tilde{g}(y) \cdot \mathbb{1}_{[0, \frac{1}{20}]}(y) dy \\ &= \int_0^{\frac{1}{20}} \tilde{g}(y) dy \end{aligned}$$

Definieren wir uns nun eine Funktion $g(y) = \frac{1}{20} \tilde{g}(y)$, dann gilt:

$$P(X > 20) = \frac{1}{20} \mathbb{E}[\tilde{g}(Y)] = \mathbb{E}[g(Y)] \approx \frac{1}{n} \sum_{i=1}^n g(y_i)$$

Der letzte Schritt entspricht der Monte-Carlo-Approximation und wird durch das Gesetz der großen Zahlen bzw. den Satz von Glivenko-Cantelli ermöglicht.

Zunächst soll versucht werden die Wahrscheinlichkeit mittels der R-Funktion `pnorm` zu bestimmen.

```
# Using pnorm
1 - pnorm(20)
```

```
## [1] 0
```

Die von R zur Verfügung gestellte Funktion `pnorm` kann die an der Stelle sehr kleine Wahrscheinlichkeit nicht mehr exakt angeben und rundet auf 0. Eine Möglichkeit trotzdem eine Lösung zu erhalten, wäre zunächst eine naive Monte-Carlo-Implementierung, die einfach eine hohe Zahl an standardnormalverteilten Zufallswerten erzeugt und prüft wie häufig diese Werte außerhalb von 20 vor kommen.

```
# Naive Monte-Carlo approximation
n <- 1000000
x <- rnorm(n)
indicator <- (x > 20) * 1
mean(indicator)
```

```
## [1] 0
```

Auch hier ist das Ergebnis Null, da die Wahrscheinlichkeit so klein ist, dass selbst — wie im obigen Fall — 1.000.000 Zufallszahlen nicht ausreichen, um ein solch unwahrscheinliches Ereignis zu erhalten.

Im nächsten Versuch soll die oben beschriebene Transformation verwendet werden.

```
fnProbRandom <- function(n) {
  # Generate random uniform values between 0 and 1/20
  #
  # Args:
  #   n: Size of sample
  #
  # Returns:
  #   Random vector of size n with values between 0 and 1/20
  return(runif(n, min = 0, max = 1/20))
}

fnProbG <- function(u) {
  # Calculate values of function g
  #
  # Args:
  #   u: Random values
  #
  # Returns:
  #   Result of function g, using random values u as argument
  return((1/20) * (1/sqrt(2*pi)) * exp(-1/(2*(u^2))) * (1/(u^2)))
}

# Set plot panel to one plot
par(mfrow = c(1,1), ps = 9, cex.axis = 0.9, cex.lab = 0.9)

# Generate random values
u <- fnProbRandom(10000)

# Calculate values with function g
g <- fnProbG(u)
```



```

# Run error estimation
fnErrorEstimation(u, fnProbRandom, fnValueCalculation = fnProbG)

# Set title for plot
title(paste("Monte-Carlo simulation with sample size of ", length(u),
           " results in P value of ", format(mean(g), digits = 3), sep = ""),
      outer = TRUE, line = -2)

```

Monte-Carlo simulation with sample size of 10000 results in P value of 2.51e-89

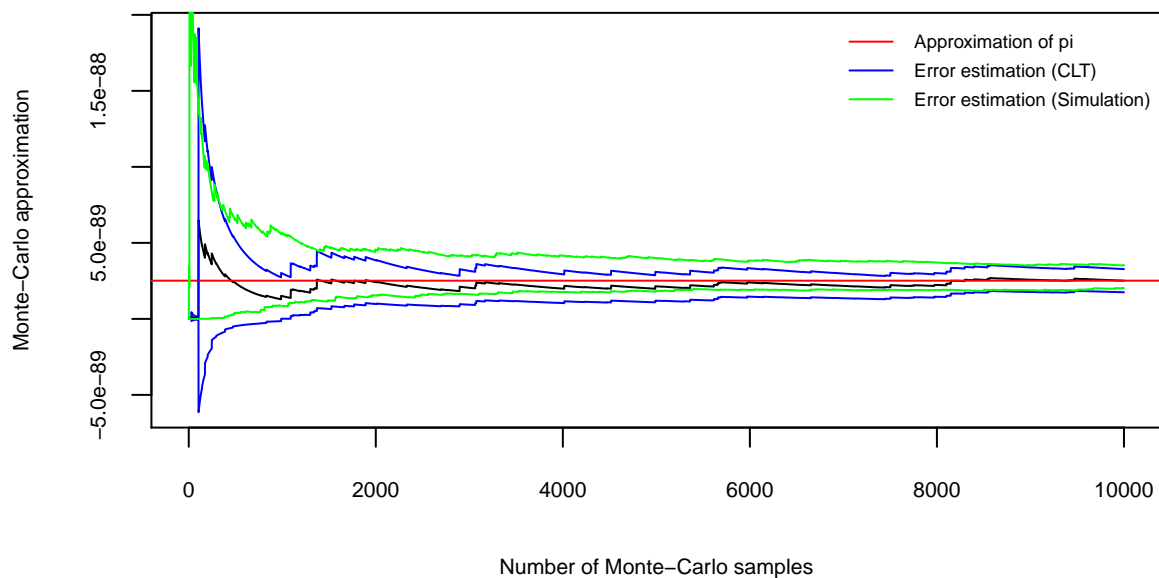


Figure 3: Results and error estimation of probability approximation.

Im Gegensatz zu den obigen naiven Ansätzen ist mittels der Transformation in gleichverteilte Zufallsvariablen eine Lösung bereits mit einer relativ kleinen Anzahl von Zufallswerten möglich. Die Ergebnisse sind in Abbildung 3 dargestellt.

2.3 Approximation eines Integrals

Die Integrale in den Grenzen zwischen 0 und 1 zweier Funktionen $h_1(x)$ und $h_2(x)$ sollen mit Hilfe einer Monte-Carlo Approximation bestimmt werden.

$$h_1(x) = (\cos(50x) + \sin(20x))^2 \quad h_2(x) = \sin\left(\frac{1}{x}\right)$$

Die Implementierung in R erfolgt über zwei Funktionen:

```

fnIntH1 <- function(x) {
  # Calculate values of function h1
  #
  # Args:
  #   x: Random values
  #
  # Returns:

```

```

# Result of function h1, using random values x as argument
return((cos(50*x)+sin(20*x))^2)
}

fnIntH2 <- function(x) {
  # Calculate values of function h2
  #
  # Args:
  #   x: Random values
  #
  # Returns:
  #   Result of function h2, using random values x as argument
  return(sin(1/x))
}

```

Wie bei der Bestimmung der Wahrscheinlichkeit oben, kann auch hier zunächst mit einem naiven Ansatz begonnen werden. Mit Hilfe der `integrate`-Funktion von R, soll eine Lösung der Integrale erfolgen.

```

# Using R's integrate function to integrate h1
h1Int <- integrate(fnIntH1, 0, 1)
h1Int

```

```
## 0.9652009 with absolute error < 1.9e-10
```

Für die Funktion $h_1(x)$ funktioniert die Bestimmung des Integrals ohne Probleme. Für die zweite Funktion $h_2(x)$, kommt es jedoch zu einem Fehler bei der Berechnung.

```

# Using R's integrate function to integrate h2
tryCatch({
  integrate(fnIntH2, 0, 1)
}, error = function(e) {
  print(e)
})

```

```
## <simpleError in integrate(fnIntH2, 0, 1): maximum number of subdivisions reached>
```

Die Monte-Carlo-Approximation kann beide Integrale annähern. Das Vorgehen ist weitestgehend analog zur Bestimmung der Wahrscheinlichkeit in der vorigen Aufgabe.

```

fnIntRandom <- function(n) {
  # Generate random uniform values
  #
  # Args:
  #   n: Size of sample
  #
  # Returns:
  #   Random vector of size n with values between 0 and 1
  return(runif(n))
}

# Set plot panel to two plots
par(mfrow = c(1,2), ps = 9, cex.axis = 0.9, cex.lab = 0.9)

# Using Monte-Carlo integration
u <- fnIntRandom(10000)
h1 <- fnIntH1(u)
h2 <- fnIntH2(u)

```

```
# Run error estimation
fnErrorEstimation(u, fnIntRandom, fnValueCalculation = fnIntH1)
title(paste("h1: ", length(u), " samples, result: ",
           round(mean(h1), 3), sep = ""))
fnErrorEstimation(u, fnIntRandom, fnValueCalculation = fnIntH2)
title(paste("h2: ", length(u), " samples, result ",
           round(mean(h2), 3), sep = ""))
```

h1: 10000 samples, result: 0.984

h2: 10000 samples, result 0.504

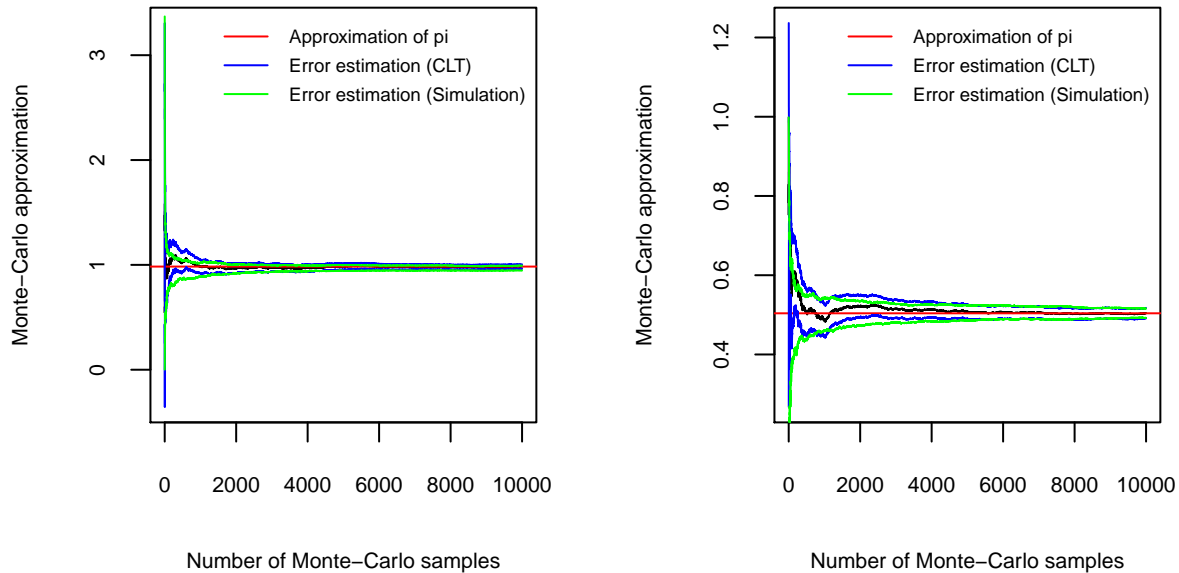


Figure 4: Results and error estimation of integral approximation.

In Abbildung 4 sind die Ergebnisse und Fehlerauswertungen der Integral-Approximationen mittels Monte-Carlo dargestellt. Ein Vergleich der Ergebnisse ist nur für $h_1(x)$ möglich, da nur hier die `integrate`-Funktion von R ein Ergebnis liefert.

```
# Calculate percentage error
round(abs(h1Int$value - mean(h1))/h1Int$value, 5)
```

```
## [1] 0.01924
```

Es ergibt sich als eine Abweichung von $\sim 1,924\%$ zwischen der `integrate`-Funktion und der Monte-Carlo-Approximation.

2.4 Erwartungswert der Fläche eines zufälligen Dreiecks

Zur einfachen Bestimmung des Erwartungswertes eines zufälligen Dreiecks kann genutzt werden, dass die Fläche des Dreiecks durch folgende Formel gegeben ist:

$$\frac{1}{2} |\text{Det } A|, \quad \text{wobei } A = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_2 & y_2 & 1 \end{pmatrix}$$

Die Monte-Carlo-Approximation wurde wie folgt implementiert:

```

fnAreaRandom <- function(n) {
  # Generate three random uniform values between 0 and 1 for x and y
  # respectively. The function directly calculates the area of triangle
  # between these three random points.
  #
  # Args:
  #   n: Size of sample
  #
  # Returns:
  #   Random vector of size n containing area values of random triangles

  areas <- NULL
  for (i in 1:n) {
    # Generate random values for x- and y-axis
    x <- runif(3)
    y <- runif(3)
    # Calculate area of triangle and add to vector
    areas[i] <- abs(0.5 * det(matrix(c(x, y, c(1, 1, 1)), nrow = 3, ncol = 3)))
  }

  # Return vector of random triangle areas
  return(areas)
}

# Set plot panel to one plot
par(mfrow = c(1,1), ps = 9, cex.axis = 0.9, cex.lab = 0.9)

# Generate random triangle area values
A <- fnAreaRandom(1000)

# Run error estimation
fnErrorEstimation(A, fnAreaRandom)
title(paste("Monte-Carlo simulation with sample size of ", length(A),
            " results in area of ", round(mean(A), 4), sep = ""))

```

Die Ergebniss und die Fehlerauswertung sind in Abbildung 5 dargestellt.

Monte-Carlo simulation with sample size of 1000 results in area of 0.0744

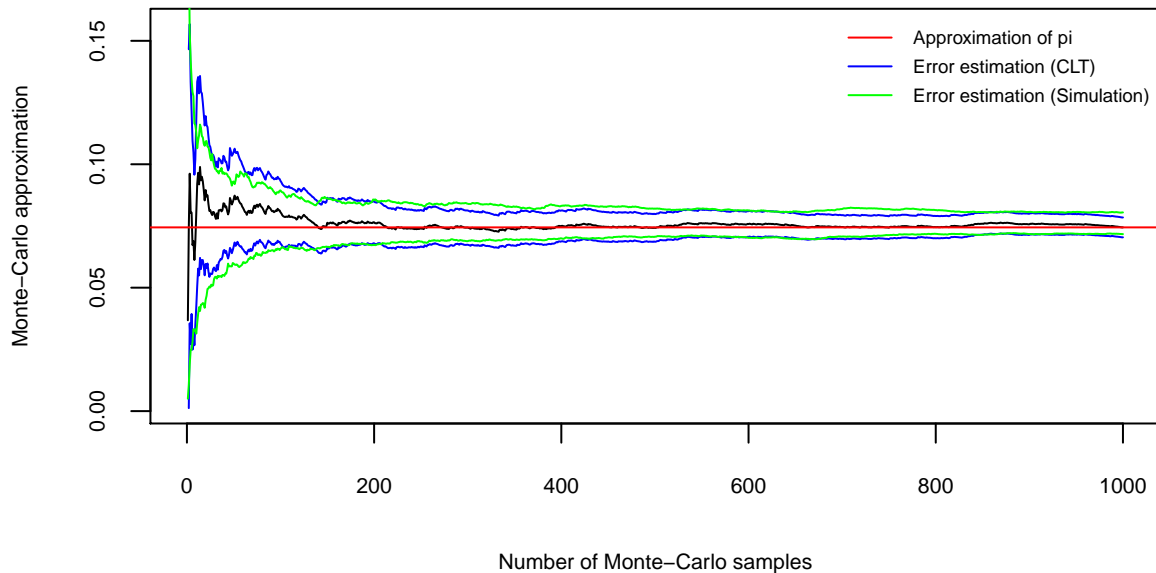


Figure 5: Results and error estimation of random triangle area.

3 Eine (naive) Datenanalyse

Für die naive Datenanalyse am Beispieldatensatz `faithful` wurden verschiedene Diagramme erstellt, die eine intuitive Auswertung der Daten ermöglichen können. Der Datensatz bezieht sich auf erhobene Daten der Ausstoßzeiten und der Ausstoßdauer des Geysirs “Old Faithful Geyser” in den USA.

3.1 Histogramm

Ein Histogramm stellt die Häufigkeit von Daten dar. Dabei werden Daten in Klassen (bins) eingeteilt und die jeweilige Häufigkeit von auftretenden Elementen innerhalb dieser Klasse auf der y-Achse aufgetragen. Ein Histogramm ermöglicht damit eine erste grobe Einschätzung der möglichen Verteilung der Daten.

Wird die gesamte Variable `eruptions` (d.h. Dauer des Ausstoßes) ausgewertet, ergibt sich ein Histogramm, welches in Abbildung ?? dargestellt ist. Hierbei wird bereits deutlich, dass es sich unter Umständen um zwei normalverteilte Teilmengen handelt. Es könnte also z.B. sein, dass der Ausstoß des Geysirs nach zwei unterschiedlichen Prozessen abläuft. Ein Blick auf getrennte Daten (Eruptionszeit größer bzw. kleiner als 3 Minuten), dargestellt in Abbildung 7, scheint diese Vermutung zu unterstützen. Ob es sich jedoch wirklich um normalverteilte Daten handelt kann erst mit dem QQ-Plot (siehe unten) genauer evaluiert werden und ist hier im besten Fall zu erraten.

3.2 Boxplot

Auch ein Boxplot soll dabei helfen die Verteilung der Daten einschätzen zu können. Der Boxplot besteht dabei aus drei oder vier Elementen. Es wird eine Box ausgegeben, deren Kanten den Bereich des unteren und oberen Quartils abdecken. Innerhalb der Box wird der Median als Linie eingezeichnet. Zusätzlich werden so genannte Whisker (ebenfalls oben und unten) eingezeichnet, die sich aus dem 1,5-fachen des Interquartilsabstands ergeben. Optional können extreme Ausreißer als Punkte mit dargestellt werden. Der Boxplot gibt damit eine optische Einschätzung der empirischen Verteilung der Daten.

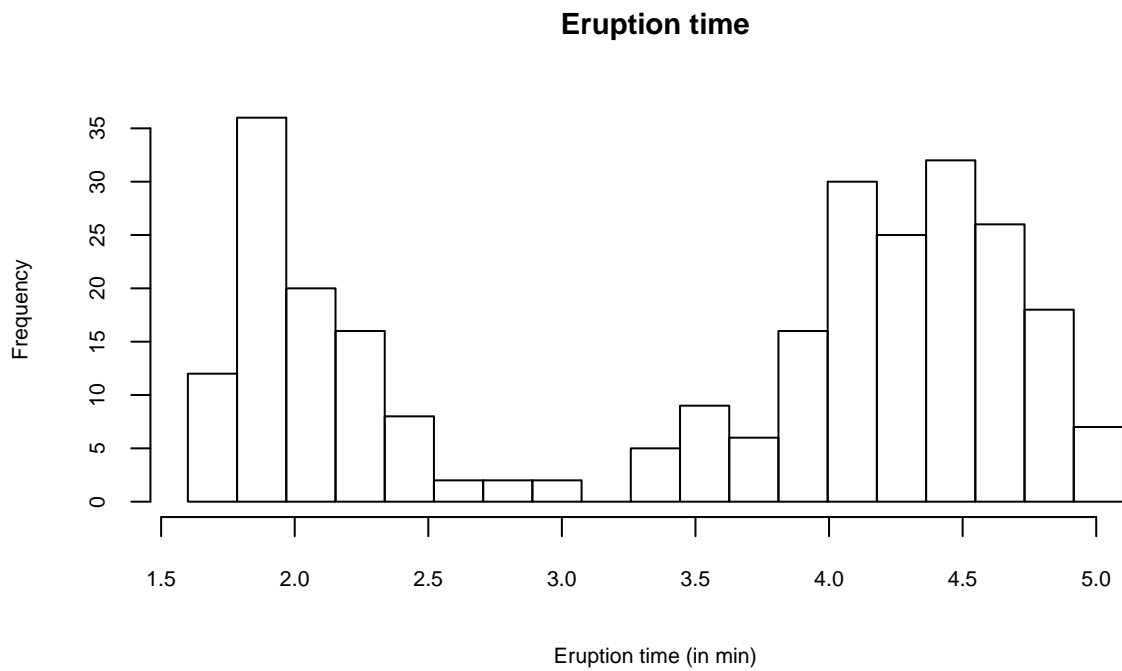


Figure 6: Histogram with all eruption data.

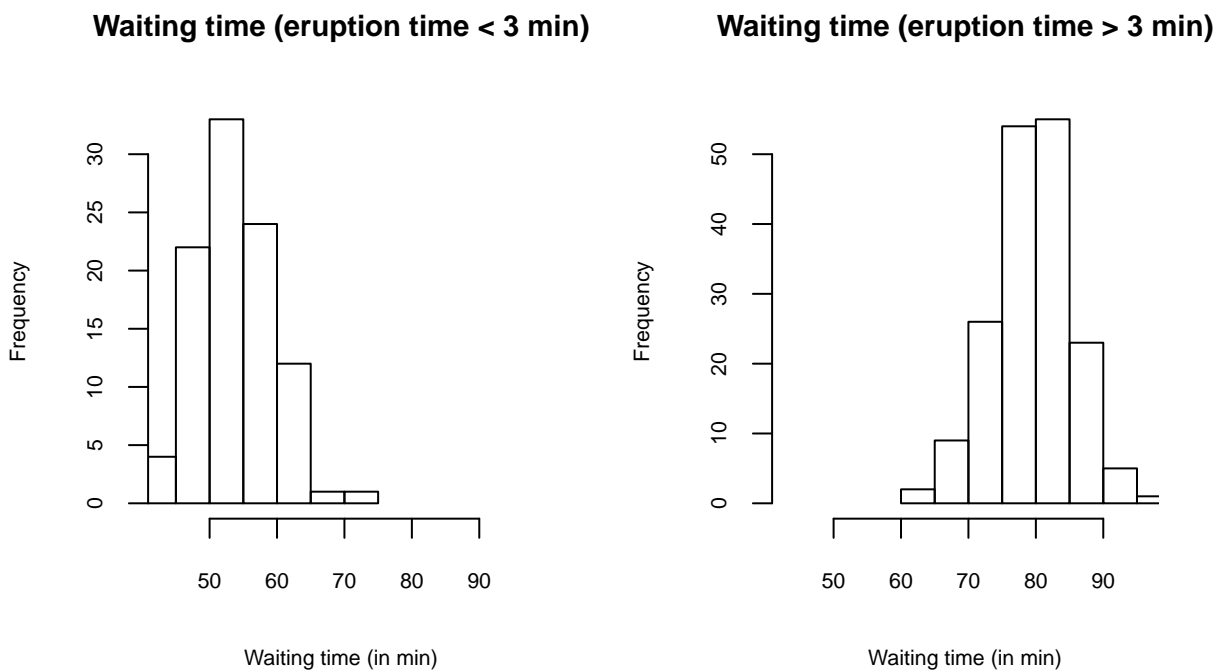


Figure 7: Histogram with eruption data, separated in two categories.

In Abbildung 8 sind die Wartezeiten für die zwei Untermengen mit Eruptionszeiten über und unter 3 Minuten dargestellt.

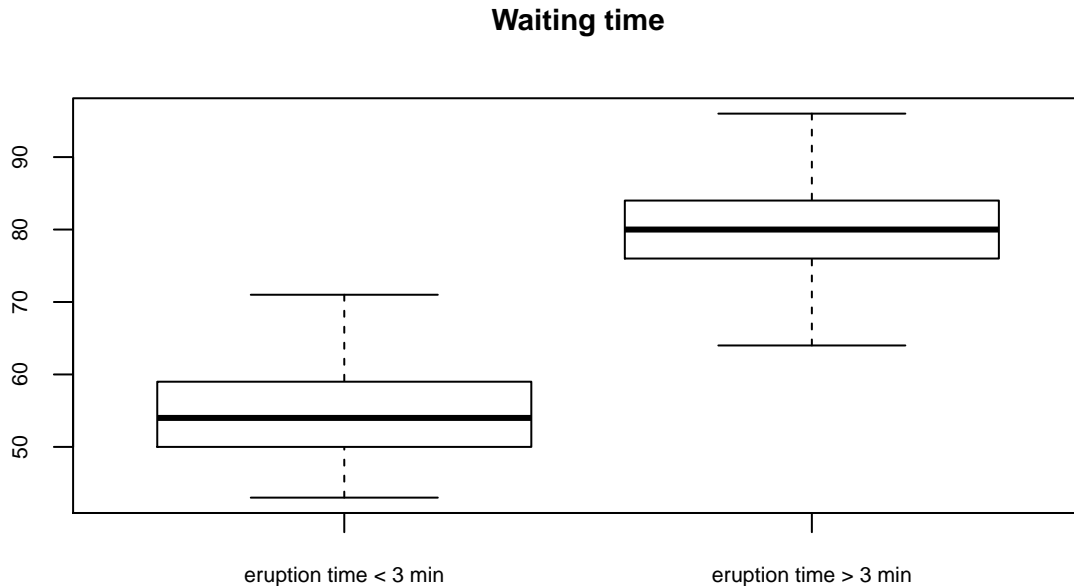


Figure 8: Boxplot with waiting data, separated in two categories.

3.3 QQ-Plot

Ein QQ-Plot vergleicht die empirischen Quantile (eine Achse) mit den theoretischen Quantilen (andere Achse). Wenn die Daten einer theoretischen Verteilung exakt folgen, liegen alle Punkte somit auf einer Linie. Mit Hilfe eines QQ-Plots ist es bereits sehr gut möglich eine Einschätzung darüber zu bekommen, welche konkrete Verteilung den Daten vermutlich zu Grunde liegt.

In Abbildung 9 sind alle Varianten dargestellt, d.h. Eruptions- und Wartezeiten, sowohl für den gesamten Datensatz, als auch getrennt für solche mit langen Eruptionszeiten (größer 3 Minuten) und kurzen Eruptionszeiten (kleiner 3 Minuten). Verglichen wird jeweils mit den theoretischen Quantilen der Standardnormalverteilung. Dabei wird deutlich, dass die getrennten Daten jeweils vermutlich einzeln normalverteilt sind, die nicht getrennten Daten ergeben aber kein solides Bild und sind gemeinsam vermutlich nicht normalverteilt. Dies spricht sehr dafür, dass die Eruptionen des Gaysirs nach zwei unterschiedlichen Prozessabläufen.

3.4 Scatter Plot and Regression

Ein Scatter-Plot stellt alle Punkte zweier (oder mehrerer Variablen) dar. Dabei ist das Ziel Abhängigkeiten zwischen den Variablen einschätzen können.

In Abbildung 10 wurden Scatter Plots für alle Variablen-Kombinationen (analog zu den QQ-Plots) erstellt. Dabei wird erneut deutlich, dass es zwei Gruppen on Daten gibt. Durch die Regressionsgeraden wird zusätzlich deutlich, dass es in allen Fällen signifikante Zusammenhänge (auf einem Niveau von $\alpha = 5\%$) gibt. Die Auswertung der Regressionsanalysen ist hier aus Platzgründen nicht ausgegeben und ist der Datei 3.R zu entnehmen.

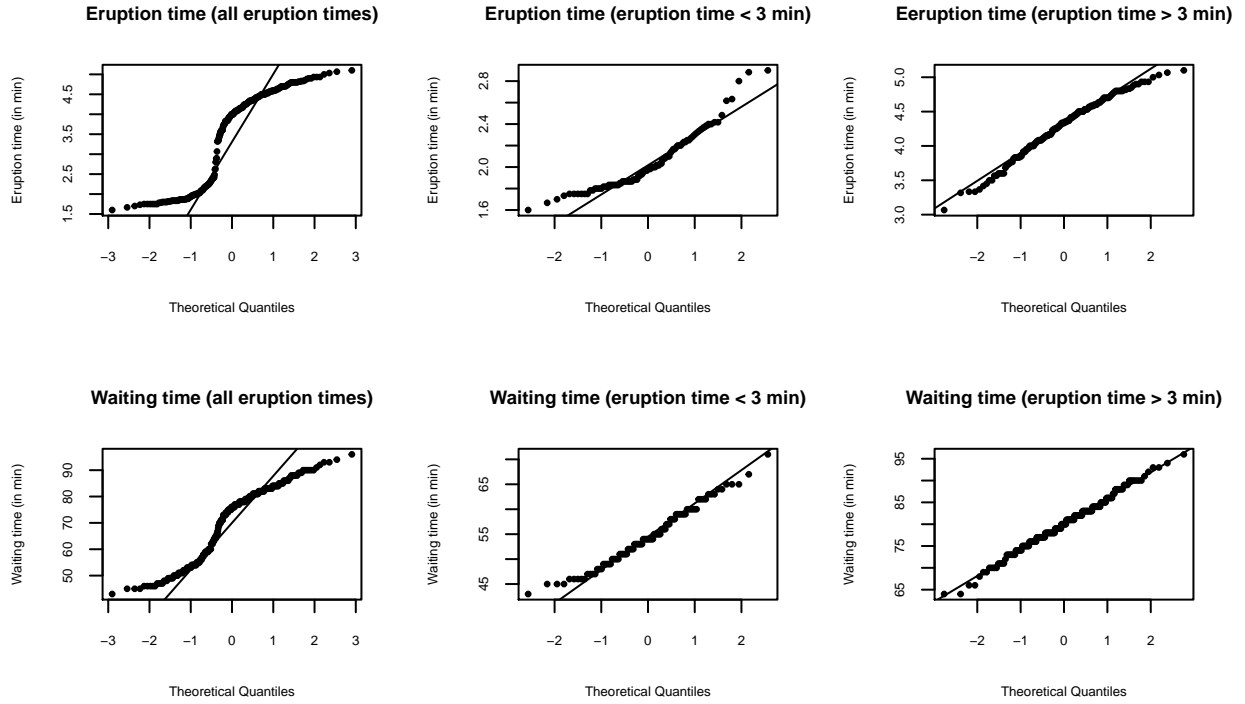


Figure 9: QQ-Plots in different data-combination.

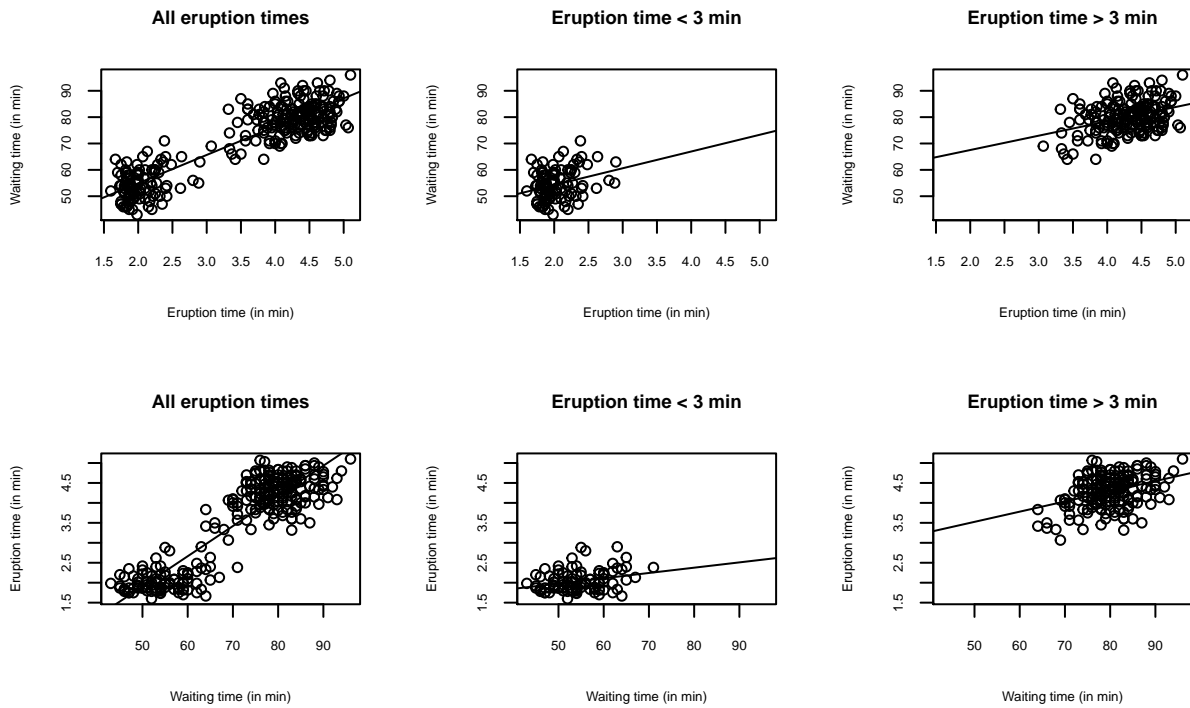


Figure 10: Scatter plots and regression lines in different data combinations.

3.5 Schätzung

Zuletzt kann eine Schätzung des erwarteten nächsten Ausbruchs erfolgen. Als Schätzer des Erwartungswerts wird einfachheitshalber der Mittelwert verwendet, welcher sich im Falle der Normalverteilung aus dem Maximum-Likelihood-Ansatz und auch aus dem Kleinste-Quadrate-Ansatz ergibt. Zusätzlich wird der Standard-Fehler bestimmt, also die Varianz des Schätzers.

Zunächst wird der nächste zu erwartende Ausbruch (Wartezeit) für alle Daten geschätzt.

```
# for all eruption times
paste("mean of waiting time:", round(mean(faithful$waiting), 3), "min")

## [1] "mean of waiting time: 70.897 min"

paste("standard error of estimator:",
      round(sqrt(var(faithful$waiting)/length(faithful$waiting)), 3), "min")

## [1] "standard error of estimator: 0.824 min"
```

Im nächsten Schritt wird der nächste erwartete Ausbruch für Ausbrüche welche weniger als 3 Minuten dauern geschätzt.

```
# for low eruption times
paste("mean of waiting time:", round(mean(faithLow$waiting), 3), "min")

## [1] "mean of waiting time: 54.495 min"

paste("standard error of estimator:",
      round(sqrt(var(faithLow$waiting)/length(faithLow$waiting)), 3), "min")

## [1] "standard error of estimator: 0.593 min"
```

Zuletzt wird der nächste erwartete Ausbruch für Ausbrüche mit mehr als 3 Minuten Wartezeit geschätzt.

```
# for high eruption times
paste("mean of waiting time:", round(mean(faithHigh$waiting), 3), "min")

## [1] "mean of waiting time: 79.989 min"

paste("standard error of estimator:",
      round(sqrt(var(faithHigh$waiting)/length(faithHigh$waiting)), 3), "min")

## [1] "standard error of estimator: 0.453 min"
```

Die Schätzer streuen weniger, wenn die Fälle einzeln betrachtet werden (lange/kurze Ausbrüche). Auch das liefert einen weiteren Hinweis für die These, dass die Ausbrüche getrennt betrachtet werden sollten.

4 Normalverteilung

In diesem Abschnitt wollen wir verschiedene Zufallsgeneratoren, d.h. deterministische Algorithmen zur Erzeugung von pseudo-zufälligen Zahlen, vergleichen. Unser Ziel ist die Erzeugung von unabhängigen standard-normalverteilten (Pseudo-)Zufallszahlen $Z \sim N(0,1)$.

Um die „Güte“ der erzeugten Zufallszahlen zu überprüfen verwenden wir Histogramme, QQ-Norm-Plots und zwei statistische Tests auf Normalverteilung (Kolmogorov-Smirnov-Test² und Shapiro-Wilk-Test³). Wir haben diese Überprüfungen in einer R-Funktion zusammengefasst:

```
fnCheckNormality <- function(vecSample) {  
  # This function helps to check a sample of generated random numbers for  
  # normality. It helps comparing different random number generators by using  
  # histograms, qq-norm-plots, Kolmogorov-Smirnov-tests, and Shapiro-Wilk-test.  
  #  
  # Args:  
  #   vecSample: A sample of generated random numbers  
  #  
  # Returns:  
  #   -  
  
  # Split plot panel  
  par(mfrow = c(1,2), ps = 9,  
      cex.axis = 0.9,  
      cex.lab = 0.9)  
  
  # Plot histogram  
  hist(vecSample,  
      breaks = seq(min(vecSample), max(vecSample), length = 25),  
      probability = TRUE,  
      xlim = c(-5, 5),  
      ylim = c(0, 0.5),  
      xlab = "Samples from random number generator",  
      main = "Histogram of random samples")  
  
  # Add density of standard normal to the plot  
  vecNorm <- seq(-5, 5, by = 0.1)  
  vecNormDensity <- dnorm(vecNorm)  
  lines(vecNorm, vecNormDensity, col = "blue")  
  legend("topright", legend = "Density of standard normal",  
      lty = 1, col = "blue", cex = 0.75)  
  
  # QQ-norm-plot  
  qqnorm(vecSample, ylim = c(-5, 5), cex = 0.8)  
  qqline(vecSample)  
  
  # Perform Kolmogorov-Smirnov test  
  print(ks.test(vecSample, "pnorm"))  
  
  # Perform Shapiro-Wilk test for normality  
  print(shapiro.test(vecSample))  
}
```

²https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test

³https://en.wikipedia.org/wiki/Shapiro%E2%80%93Wilk_test

Als Benchmark überprüfen wir zunächst die in R mit der Funktion `rnorm` standardmäßig implementierte Methode zur Erzeugung standard-normalverteilter Zufallsvariablen:

```
# Test with rnorm-function
set.seed(42)
fnCheckNormality(rnorm(5000))
```

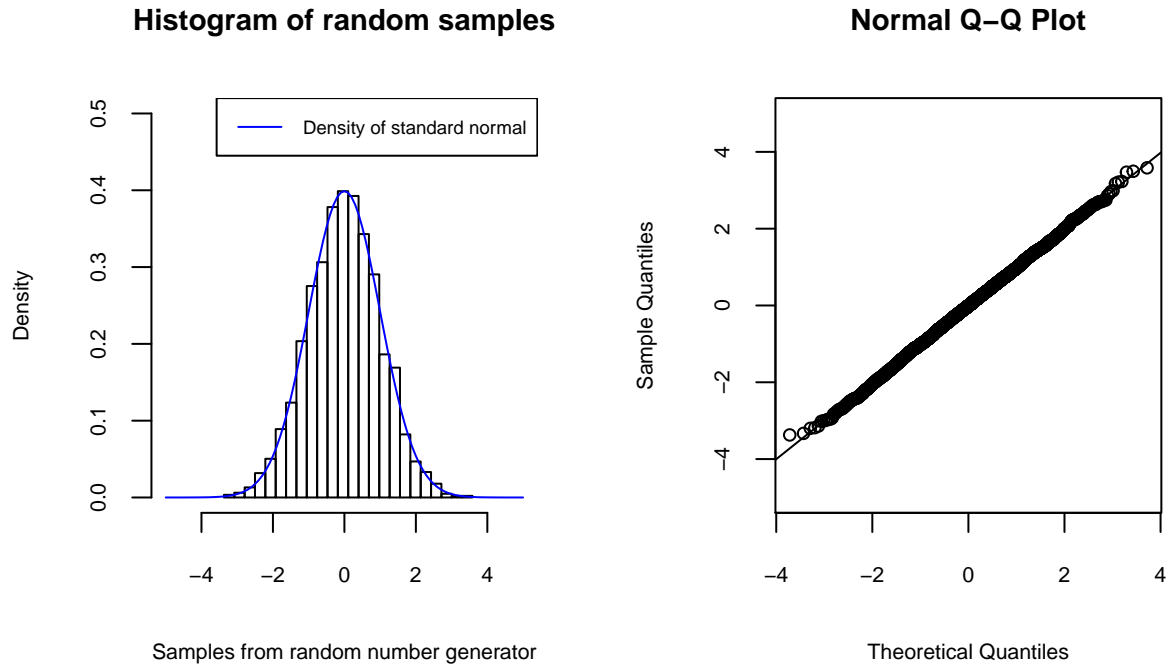


Figure 11: Histogram and QQ-Norm-plot for random numbers generated with `rnorm`.

```
##
## One-sample Kolmogorov-Smirnov test
##
## data:  vecSample
## D = 0.0089805, p-value = 0.8148
## alternative hypothesis: two-sided
##
##
## Shapiro-Wilk normality test
##
## data:  vecSample
## W = 0.99971, p-value = 0.744
```

Wir können sehen, dass Histogramm und Dichtefunktion der Standard-Normalverteilung sowie die theoretischen mit den empirischen Quantile sehr gut übereinstimmen. Dieser optische Eindruck wird durch die beiden statistischen Tests unterstützt: in beiden Tests liegen die p-Werte deutlich über den gewöhnlichen Signifikanzniveaus α von 0.05 oder 0.01. D.h. in beiden Tests kann die Nullhypothese, dass die Zufallszahlen normalverteilt sind, nicht abgelehnt werden.

Im Folgenden wollen wir drei verschiedene Methoden zur Erzeugung von standard-normalverteilten Zufallsvariablen vergleichen. Alle drei Methoden setzen die Erzeugung von unabhängigen $U([0, 1])$ -verteilten Zufallsvariablen voraus, wovon wir annehmen, dass diese zuverlässig in R erzeugt werden können.⁴

⁴Die Methoden zur Erzeugung von Zufallszahlen in R finden sich in der Dokumentation unter „RNG“ (Random Number Generation), welche auch unter <https://stat.ethz.ch/R-manual/R-devel/library/base/html/Random.html> eingesehen werden kann.

4.1 Box-Muller-Methode

Die Box-Muller-Methode nutzt aus, dass für $U_1, U_2 \stackrel{iid}{\sim} U([0,1])$ die Transformationen $Z = Z_1 := \sqrt{-2\log U_1} \cos(2\pi U_2), Z_2 := \sqrt{-2\log U_1} \sin(2\pi U_2)$ unabhängige standard-normalverteilte Zufallsvariablen sind.⁵ Wir überprüfen $Z = Z_1$:

```
# Generate random numbers from the standard normal using the Box-Muller-Method
set.seed(42)
nSamples <- 5000

vecU1 <- runif(nSamples)
vecU2 <- runif(nSamples)
vecZ1 <- sqrt((-2)*log(vecU1)) * cos(2*pi*vecU2)

# Test for normality
fnCheckNormality(vecZ1)
```

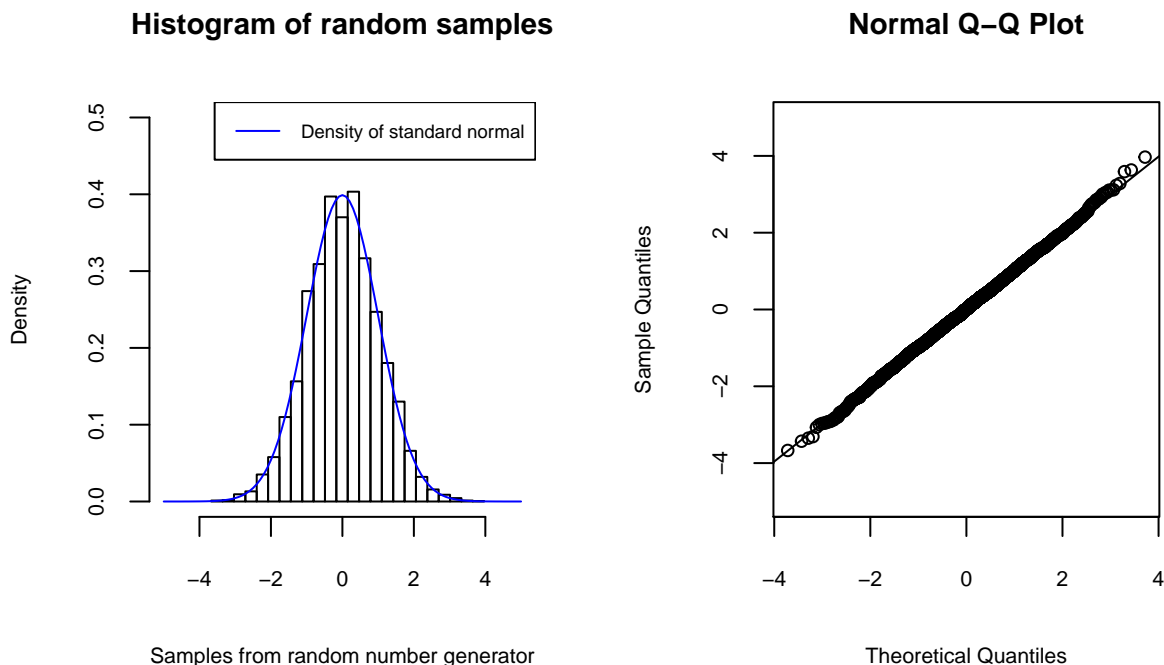


Figure 12: Histogram and QQ-Norm-plot for random numbers generated with the Box-Muller method.

```
##
## One-sample Kolmogorov-Smirnov test
##
## data: vecSample
## D = 0.009996, p-value = 0.6998
## alternative hypothesis: two-sided
##
##
## Shapiro-Wilk normality test
##
## data: vecSample
## W = 0.99974, p-value = 0.8267
```

⁵Es ist $(Z_1, Z_2) = (R \cos(2\pi U_2), R \sin(2\pi U_2))$ die Polarkoordinatentransformation mit Radius $R := \sqrt{-2\log U_1}$. Die Aussage lässt sich dann mit Hilfe des Dichtetransformationssatzes nachweisen.

Wiederum stimmen Histogramm und Dichtefunktion sowie die theoretischen mit den empirischen Quantilen sehr gut überein. Auch die Tests können zum Signifikanzniveau $\alpha = 0.05$ oder $\alpha = 0.01$ nicht abgelehnt werden, da die p-Werte deutlich größer sind.

4.2 Summe stetig gleichverteilter Zufallsvariablen

Als zweite Methode wollen wir gleichverteilte Zufallszahlen $U_1, \dots, U_m \stackrel{iid}{\sim} U([-1/2, 1/2])$ für ein $m \in \mathbb{N}$ erzeugen und $Z = \sum_{i=1}^m U_i$ setzen. Es ist $\mathbb{E}[U_1] = 0$ und $\text{Var}(U_1) = 1/12$. Damit gilt nach dem zentralen Grenzwertsatz, dass die standardisierte Summe in Verteilung gegen die Standard-Normalverteilung konvergiert:

$$\sqrt{\frac{12}{m}} \sum_{i=1}^m U_i \xrightarrow{d} N(0, 1).$$

Damit stimmt die standardisierte Summe mit der Summe $Z = \sum_{i=1}^m U_i$ für $m = 12$ überein. Wir werden nun überprüfen, ob $m = 12$ bereits hinreichend groß ist, um normalverteilte Zufallszahlen durch Z zu generieren⁶:

```
# Generate random numbers from the standard normal as sum of i.i.d.
# U([-0.5, 0.5])-distributed random variables

set.seed(42)
nSamples <- 5000
m <- 12 # number of i.i.d. random variables

matUnif <- matrix(runif(nSamples*m, -0.5, 0.5), ncol = m)
vecZ2 <- apply(matUnif, 1, sum)

# Test for normality
fnCheckNormality(vecZ2)
```

```
##
## One-sample Kolmogorov-Smirnov test
##
## data: vecSample
## D = 0.011851, p-value = 0.4837
## alternative hypothesis: two-sided
##
##
## Shapiro-Wilk normality test
##
## data: vecSample
## W = 0.9995, p-value = 0.2187
```

Das Histogramm und Dichtefunktion sowie die theoretischen mit den empirischen Quantilen stimmen wiederum gut überein. Auch die Tests können zum Signifikanzniveau $\alpha = 0.05$ oder $\alpha = 0.01$ nicht abgelehnt werden, da die p-Werte größer sind. Im Vergleich zur Box-Muller-Methode sind die p-Werte der Tests jedoch kleiner. Der QQ-Norm-Plot lässt des Weiteren vermuten, dass die Tails nicht so gut übereinstimmen.

4.3 Accept-Reject-Algorithmus mit Laplace-Kandidatendichte

Als letzte Methode wollen wir den Accept-Reject-Algorithmus mit der Dichte der Laplace-Verteilung

⁶Ein bekanntes Resultat über die Güte der Konvergenz im zentralen Grenzwertsatz ist der Satz von Berry-Esseen. Dieser besagt (in einfacher Form), dass wenn das dritte absolute Moment existiert (was in unserem Fall zutrifft), die Konvergenzrate der Ordnung $n^{-\frac{1}{2}}$ ist. (https://en.wikipedia.org/wiki/Berry%E2%80%93Esseen_theorem)

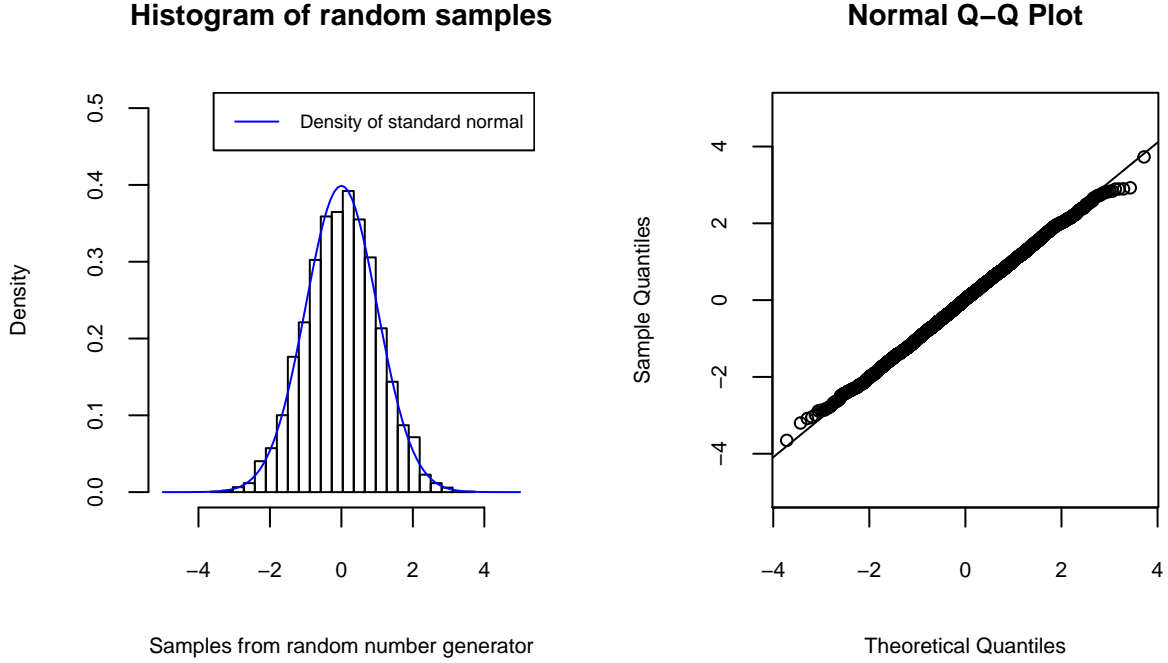


Figure 13: Histogram and QQ-Norm-plot for random numbers generated from the sum of $m = 12$ uniform random variables.

$$g_\alpha(x) = \frac{\alpha}{2} \exp(-\alpha|x|), \quad \alpha > 0,$$

als Kandidatendichte implementieren. Um Zufallszahlen der Laplace-Verteilung zu generieren, wenden wir die Inversionsmethode an. Für die Verteilungsfunktion F_α der Laplace-Verteilung gilt:

$$F_\alpha(x) = \int_{-\infty}^x \frac{\alpha}{2} \exp(-\alpha|y|) dy = \begin{cases} \int_{-\infty}^x \frac{\alpha}{2} \exp(\alpha y) dy = \frac{1}{2} \exp(\alpha x) & , x < 0, \\ \frac{1}{2} \exp(0) + \int_0^x \frac{\alpha}{2} \exp(-\alpha y) dy = 1 - \frac{1}{2} \exp(-\alpha x) & , x \geq 0. \end{cases}$$

Somit folgt für die Inverse F_α^{-1} :

$$F_\alpha^{-1}(u) = \begin{cases} \frac{1}{\alpha} \log(2u) & , u < \frac{1}{2}, \\ -\frac{1}{\alpha} \log(2(1-u)) & , u \geq \frac{1}{2}. \end{cases}$$

Um Zufallszahlen einer Zielverteilung mit Dichte f zu erzeugen, muss für die Kandidatendichte g_α und eine Konstante M gelten, dass $\frac{f(x)}{g_\alpha(x)} \leq M$ für alle $x \in \mathbb{R}$. Da die Akzeptanzwahrscheinlichkeit im Accept-Reject-Sampling durch $\frac{1}{M}$ gegeben ist, macht es Sinn M möglichst „scharf“ (d.h. klein) zu wählen, sodass weniger Samples abgelehnt werden. Definiere

$$h_\alpha(x) := \frac{f(x)}{g_\alpha(x)} = \sqrt{\frac{2}{\pi}} \cdot \frac{1}{\alpha} \cdot \exp\left(\alpha|x| - \frac{x^2}{2}\right), \quad x \in \mathbb{R},$$

da f in unserer Anwendung die Dichte der Standard-Normalverteilung ist. Dann ist h_α für $|x| = \alpha$ maximal, denn

$$\begin{aligned} \frac{\partial}{\partial x} h_\alpha(x) &= h_\alpha(x) \cdot (\alpha \mathbb{1}_{(0,\infty)}(x) - \alpha \mathbb{1}_{(-\infty,0)}(x) - x) \stackrel{!}{=} 0 \\ &\iff x = \alpha \mathbb{1}_{(0,\infty)}(x) - \alpha \mathbb{1}_{(-\infty,0)}(x), \end{aligned}$$

da $h_\alpha(x) > 0$ für alle $x \in \mathbb{R}$.

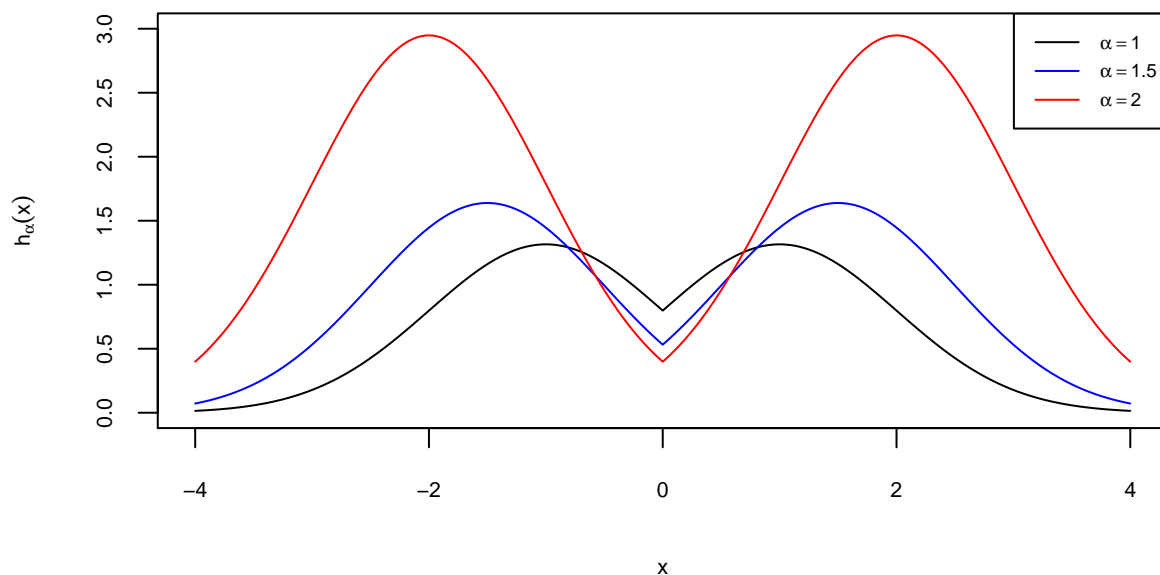


Figure 14: The function h_α for different values of α .

Somit ist h_α uniform durch $h_\alpha(\alpha) = \sqrt{\frac{2}{\pi}} \cdot \frac{1}{\alpha} \cdot \exp\left(\frac{\alpha^2}{2}\right)$ beschränkt. Weiter können wir diese Schranke in $\alpha > 0$ minimieren:

$$\frac{\partial}{\partial \alpha} \left(\sqrt{\frac{2}{\pi}} \cdot \frac{1}{\alpha} \cdot \exp\left(\frac{\alpha^2}{2}\right) \right) = \sqrt{\frac{2}{\pi}} \left(1 - \frac{1}{\alpha^2} \right) \exp\left(\frac{\alpha^2}{2}\right) \stackrel{!}{=} 0$$

$$\iff \alpha = 1.$$

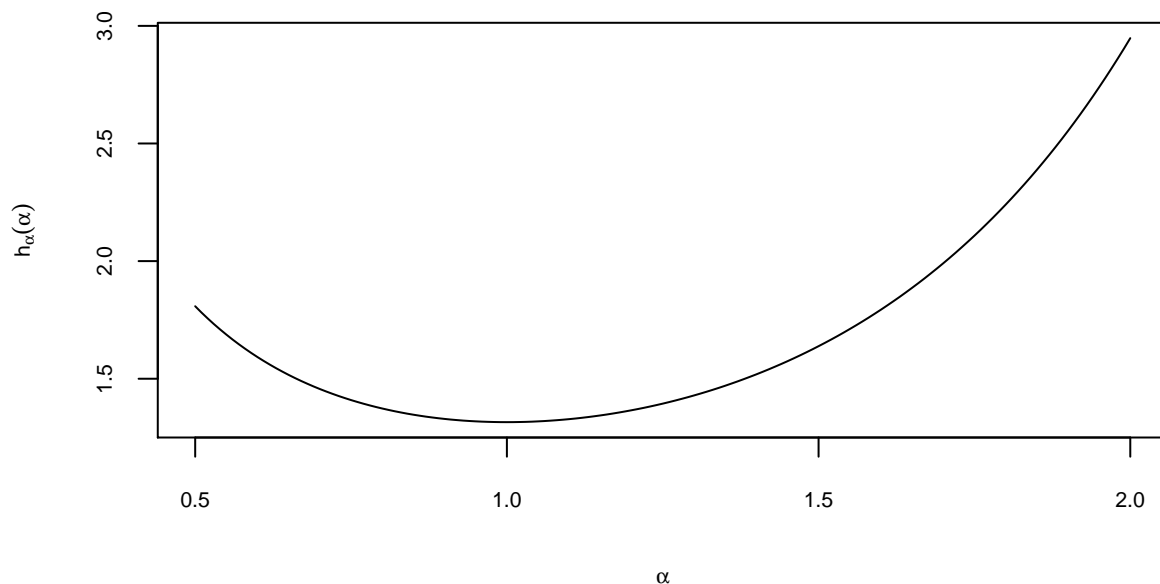


Figure 15: $h_\alpha(\alpha)$ as a function in α .

Somit wählen wir die scharfe Schranke $M = \sqrt{\frac{2}{\pi}} \exp(0.5) \approx 1.3155$ und können den Accept-Reject-Algorithmus wie folgt implementieren:

```

# Generate random numbers from the standard normal using the
# accept-reject-algorithm with Laplace proposal density

# Set seed and number of samples
set.seed(42)
nSamples <- 5000

# Define Laplace density
g <- function(x, alpha = 1) {
  return((alpha/2) * exp((-alpha)* abs(x)))
}

# Define bound M efficiently
M <- sqrt(2/pi) * exp(0.5)

# Sample from U([0,1])
vecU1 <- runif(nSamples)

# Sample from the Laplace distribution using the inversion-method
alpha <- 1
vecU2 <- runif(nSamples)
vecLaplace <- rep(0, nSamples)
vecLaplace[vecU2 < 0.5] <- (1/alpha) * log(2*vecU2[vecU2 < 0.5])
vecLaplace[vecU2 >= 0.5] <- (-1/alpha) * log(2*(1-vecU2[vecU2 >= 0.5]))

# Accept or reject?
vecZ3 <- vecLaplace[vecU1 <= dnorm(vecLaplace)/(M * g(vecLaplace))]

# Acceptance ratio
fAcceptRatio <- length(vecZ3) / nSamples

# Test for normality
fnCheckNormality(vecZ3)

```

```

##
## One-sample Kolmogorov-Smirnov test
##
## data:  vecSample
## D = 0.015448, p-value = 0.3289
## alternative hypothesis: two-sided
##
##
## Shapiro-Wilk normality test
##
## data:  vecSample
## W = 0.99968, p-value = 0.8549

```

Die Akzeptanzrate liegt bei 0.7546. D.h. es müssen im Schnitt etwa vier Zufallszahlen bzgl. g_1 erzeugt werden um drei standard-normalverteilte Zufallszahlen zu erhalten.

Wir sehen wiederum, dass Histogramm und Dichtefunktion sowie die theoretischen und empirischen Quantile sehr gut übereinstimmen. Auch die Tests können zum Signifikanzniveau $\alpha = 0.05$ oder $\alpha = 0.01$ nicht abgelehnt werden, da die p-Werte größer sind.

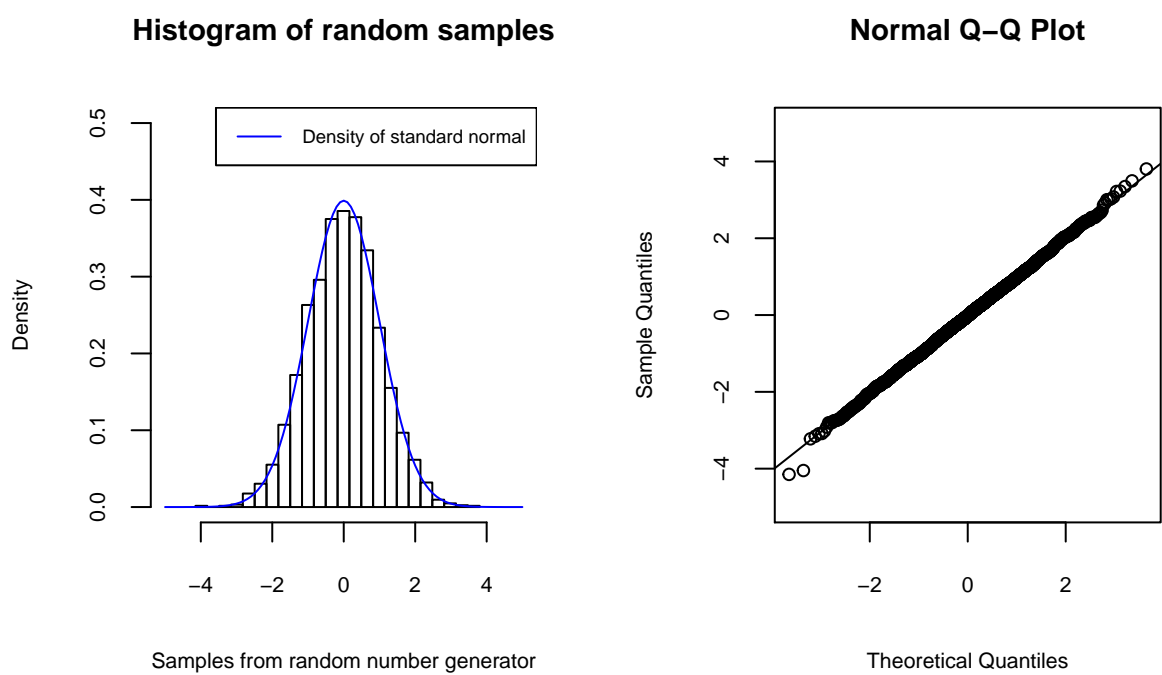


Figure 16: Histogram and QQ-Norm-plot for random numbers generated using Accept-Reject-Sampling with Laplace proposal density with $\alpha = 1$.