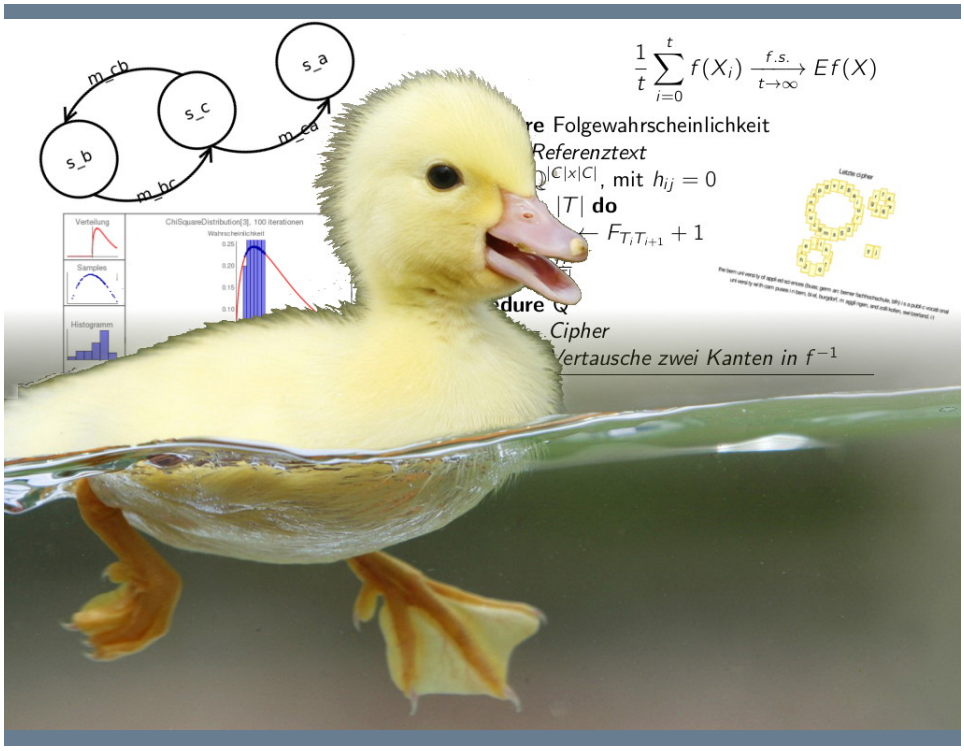


Metropolisalgorithmus mit simulated annealing

Theorie, Beweise, Anwendungen

Seminararbeit

Dies ist ein Begleitdokument für das im Rahmen des Seminar erarbeiteten Thema: Metropolisalgorithmus mit simulated annealing. Das Dokument führt die für das Verständnis des Metropolisalgorithmus wichtigen Markow-Ketten ein, diskutiert Eigenschaften solcher Ketten, stellt den Metropolisalgorithmus vor und beweist diesen. Zum Schluss wird die Heuristik simulated annealing durch den Metropolisalgorithmus motiviert. Das Zielpublikum dieses Dokumentes sind Bachelorstudenten. Für das Verständnis wird Vorwissen in linearer Algebra, Statistik und Stochastik vorausgesetzt.

Studiengang:	BSc Informatik
Autoren:	Ente
Betreuer:	XXXXXXX
Auftraggeber:	XXXXXXX
Experten:	XXXXXXX, XXXXXXX, XXXXXXX, XXXXXXX
Datum:	23.05.2016

Versionen

Version	Datum	Status	Bemerkungen
0.1	28.3.2016	Entwurf	Dokumentinitialisierung
0.2	5.4.2016	Entwurf	Markow-Ketten
0.3	30.4.2016	Entwurf	Ergodizität
0.4	8.5.2016	Entwurf	Metropolisalgorithmus
0.5	8.5.2016	Entwurf	Metropolis-Hastings-Algorithmus
0.6	15.5.2016	Entwurf	Beispiel1 Metropolisalgorithmus
0.7	16.5.2016	Entwurf	Beweis Metropolisalgorithmus
0.8	19.5.2016	Entwurf	Beispiel2 Metropolisalgorithmus
0.9	20.5.2016	Entwurf	Simulated annealing
0.10	21.5.2016	Entwurf	Einleitung, Titel, Schluss
0.11	22.5.2016	Entwurf	Korrekturen
1.0	23.5.2016	Abgabe	Finalisierung

Inhaltsverzeichnis

1. Markow-Ketten

Dieses Kapitel soll die Grundlagen für das Verständnis des Metropolisalgorithmus einführen. Hierbei werden Markow-Ketten motiviert damit im Kapitel 2 Eigenschaften, die zum ergodischen Theorem für Markow-Ketten führen, besprochen werden können.

1.1. Definition

Zeit- und Raumdiskrete Markow-Ketten bilden Prozesse ab die eine Reihe von Zufallsvariablen X generieren die alle aus einem höchst abzählbaren Zustandsraum

$$\forall_t X_t \in \{s_1, s_2, ..., s_n\} = S \tag{1.1}$$

stammen. Wobei die Wahrscheinlichkeit für das Eintreffen eines Ereignis immer nur vom letzten Ereignis abhängt.

$$P(X_t = s_j | X_{t-1} = s_i) = P(X_t = s_j | X_{t-1} = s_i, X_{t-2} = s_k, ..., X_0 = s_0) \tag{1.2}$$

1.2. Darstellung

Wollen wir solche Ketten darstellen, können wir uns die Gleichung 1.1 zunutze machen. Wir wissen, dass es nur eine abzählbare Anzahl unterscheidbaren Zuständen geben kann. Also können wir die Zustände als Knoten eines Graphen interpretieren.

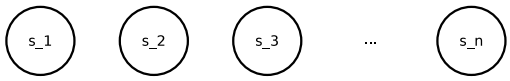


Abbildung 1.1.: Zustände einer Markow-kette

Da die Wahrscheinlichkeit einen Zustand zu erreichen nur vom vorherigen Zustand abhängt, können wir diese sogenannten Übergangswahrscheinlichkeiten in den Graph einzeichnen:

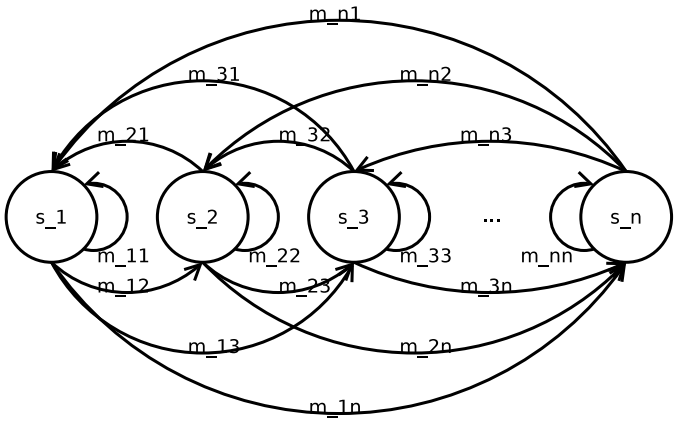


Abbildung 1.2.: Zustände einer Markow-Kette mit Übergangswahrscheinlichkeiten

oder in einer quadratischen Matrix abbilden.

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & \dots & m_{1n} \\ m_{21} & m_{22} & m_{23} & \dots & m_{2n} \\ m_{31} & m_{32} & m_{33} & \dots & m_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & m_{n3} & \dots & m_{nn} \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.3}$$

Die Übergangswahrscheinlichkeiten werden hierbei zu Zellen der Matrix, wobei gilt

$$\forall_i \forall_j P(s_j|s_i) = m_{ij} \geq 0 \tag{1.4}$$

Wir beobachten weiter, dass die einzelnen Zeilenvektoren der Matrix

$$\mu_j = (m_1, m_2, \dots, m_n) \tag{1.5}$$

alle Teilwahrscheinlichkeiten des gesamten Zustandsraumes beinhalten und somit folgendes gelten muss

$$\forall_i (\sum_{j=0}^{n} m_{ij} = 1) \tag{1.6}$$

Matrizen die den Gleichungen 1.4 und 1.6 genügen werden wir fortan Übergangsmatrizen nennen. Übergangsmatrizen sind von zentraler Bedeutung wenn wir Markow-Ketten beschreiben.

1.3. Beispiel: Wetten auf Münzwurf

Inspiriert von [12, Kapitel 2.1]: Wir spielen ein Spiel. Das Ziel des Spieles soll es sein sechs Jetons zu erreichen. Das Spiel ist verloren wenn wir keine Jetons mehr haben. In jeder Spielrunde werfen wir eine Münze. Zeigt die Münze Kopf, erhalten wir einen Jeton. Zeigt die Münze Zahl, müssen wir einen Jeton abgeben. Die Zustände sind folglich alle möglichen Höhen unseres Jetonstapels.

$$S = \{0, 1, 2, 3, 4, 5, 6\} \tag{1.7}$$

Die Wahrscheinlichkeit für Kopf sei

$$P(\text{„Kopf“}) = p \tag{1.8}$$

Wobei wir mit einer fairen Münze spielen

$$p = 1 - p = \frac{1}{2} \tag{1.9}$$

Die Übergangswahrscheinlichkeiten können nun wie folgt beschreiben werden:

$$P(X_t = j|X_{t-1} = i) = \begin{cases} p, & \text{für } i \in \{1, 2, 3, 4, 5\} \text{ und } j = i + 1 \text{ "Kopf"} \\ 1 - p, & \text{für } i \in \{1, 2, 3, 4, 5\} \text{ und } j = i - 1 \text{ "Zahl"} \\ 1, & \text{für } i \in \{0, 6\} \text{ und } j = i \text{ "Endzustand"} \\ 0, & \text{sonst "Mehr als einen Jeton bekommen oder verlieren"} \end{cases} \tag{1.10}$$

oder als dazu äquivalente Übergangsmatrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1-p & 0 & p & 0 & 0 & 0 & 0 \\ 0 & 1-p & 0 & p & 0 & 0 & 0 \\ 0 & 0 & 1-p & 0 & p & 0 & 0 \\ 0 & 0 & 0 & 1-p & 0 & p & 0 \\ 0 & 0 & 0 & 0 & 1-p & 0 & p \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{7x7} \tag{1.11}$$

Wir können nun zeitabhängige Zustandswahrscheinlichkeiten definieren

$$\pi_j(t) = P(X_t = j) \tag{1.12}$$

Wobei $\pi_j(t)$ die Wahrscheinlichkeit ist, nach t Wiederholungen im Zustand j zu sein. Zu Beginn starten wir mit vier Punkten. Die Wahrscheinlichkeit zu $t = 0$ im vierten $j = 4$ Zustand zu sein ist also 1.

$$\pi_0(0) = 0, \pi_1(0) = 0, \pi_2(0) = 0, \pi_3(0) = 0, \pi_4(0) = 1, \pi_5(0) = 0, \pi_6(0) = 0 \iff \pi(0) = (0, 0, 0, 0, 1, 0, 0) \tag{1.13}$$

Wollen wir nun die Zustandswahrscheinlichkeit nach t-Schritten berechnen, können wir dies durch t-maliges multiplizieren der Startzustandswahrscheinlichkeiten mit den Übergangswahrscheinlichkeiten erreichen

$$\pi(t) = \pi(0)M^t \tag{1.14}$$

In unserem Beispiel resultieren folgende Zustandswahrscheinlichkeiten:

j	$\pi_j(0)$	$\pi_j(1)$	$\pi_j(2)$	$\pi_j(3)$	$\pi_j(4)$	$\pi_j(5)$	$\pi_j(10)$	$\pi_j(20)$	$\pi_j(30)$	$\pi_j(40)$	$\pi_j(50)$
0	0.	0.	0.	0.	0.06	0.06	0.21	0.31	0.33	0.33	0.33
1	0.	0.	0.	0.12	0.	0.12	0.	0.	0.	0.	0.
2	0.	0.	0.25	0.	0.25	0.	0.12	0.03	0.01	0.	0.
3	0.	0.5	0.	0.38	0.	0.28	0.	0.	0.	0.	0.
4	1.	0.	0.5	0.	0.31	0.	0.12	0.03	0.01	0.	0.
5	0.	0.5	0.	0.25	0.	0.16	0.	0.	0.	0.	0.
6	0.	0.	0.25	0.25	0.38	0.38	0.55	0.64	0.66	0.67	0.67

Tabelle 1.1.: Berechnung von $\pi(t)$ für $t = 0, \ldots, 50$

Wir sehen, dass die Wahrscheinlichkeit nach 50 Runden zu gewinnen bei ~ 0.67 und zu verlieren bei ~ 0.33 liegt. Weiter können wir beobachten, dass die Zustandswahrscheinlichkeiten $\pi(t)$ mit grösser werdendem t gegen den Vektor $(\frac{1}{3}, 0, 0, 0, 0, 0, \frac{2}{3})$ zu konvergieren scheint. Wenn wir uns im Kapitel 2 genauer mit den Eigenschaften von Übergangsmatrizen auseinandersetzen, werden wir diese Beobachtung verstehen.

2. Eigenschaften von Markow-Ketten

Übergangsmatrizen können gewisse Eigenschaften zukommen. Damit Markow-Ketten Monte Carlo Algorithmen funktionieren, zu denen der Metropolisalgorithmus gehört, muss die Übergangsmatrix ergodisch sein.

Definition 1. *Ergodische Übergangsmatrizen sind zeithomogen, irreduzierbar, aperiodisch und haben eine stationäre Verteilung.*

Das folgende Kapitel definiert alle diese Eigenschaften einzeln und stellt danach das ergodische Theorem für Markow-Ketten vor. Dieses Theorem spielt eine zentrale Rolle in Kapitel 3.

2.1. Zeithomogen

Zeithomogenität sagt aus, dass sich die Übergangsmatrix mit fortschreitender Zeit nicht verändert. Ausgehend vom Zustand s_a darf sich die Wahrscheinlichkeit im Zustand s_b zu landen nie verändern [10] [7] [8].

Definition 2. *Eine Markow-Kette mit Gliedern X ist zeithomogen wenn gilt*

$$\forall_t \forall_{a,b} P(X_t = s_b | X_{t-1} = s_a) = P(X_{t-1} = s_b | X_{t-2} = s_a) \tag{2.1}$$

Dies ist in Gleichung 1.14 vorgestellten Berechnungsschema nur schwierig vorstellbar, jedoch durchaus möglich. Die Übergansmatrix in 2.2 ist zeitinhomogen da sie sich mit t verändert.

$$M = \begin{bmatrix} \frac{1}{t+1} & 1 - \frac{1}{t+1} \\ 1 - \frac{1}{t+1} & \frac{1}{t+1} \end{bmatrix} \in \mathbb{R}^{2x2} \tag{2.2}$$

2.2. Irreduzierbar

Irreduzierbarkeit sagt aus, dass von jedem Zustand s_a jeder andere Zustand s_b in beliebig vielen Schritten, mit positiver Wahrscheinlichkeit, erreicht werden kann [10] [7] [8].

Definition 3. *Eine Markow-Kette mit Gliedern X ist irreduzierbar wenn gilt*

$$\exists_{t \geq 0} \forall_{a,b} P(X_t = s_b | X_0 = s_a) > 0 \tag{2.3}$$

Die Markow-Kette in der Abbildung 2.1 ist nicht irreduzierbar da der Zustand s_a nicht mehr verlassen werden kann.

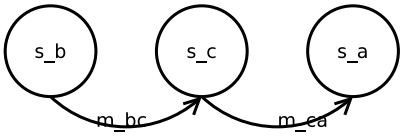


Abbildung 2.1.: Nicht irreduzierbare Markow-Kette

Durch hinzufügen einer weiteren Wahrscheinlichkeit, können wir diese Markow-Kette irreduzierbar machen.

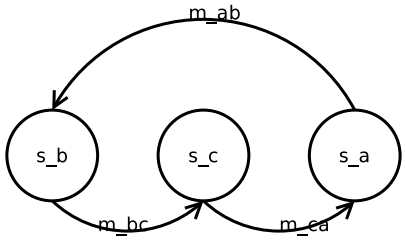


Abbildung 2.2.: Irreduzierbare Markow-Kette

2.3. Aperiodisch

Aperiodizität sagt aus, dass eine Markow-Kette keine Zyklen enthalten darf. Dies bedeutet, dass zu einem Startzustand s_a nur in nicht regelmässigen Abständen zurückgekehrt werden kann [10] [7] [8].

Definition 4. *Eine Markow-Kette mit Gliedern X ist aperiodisch wenn gilt*

$$\forall_a \gcd\{t|P(X_t = s_a|X_0 = s_a) > 0\} = 1 \tag{2.4}$$

Ein Beispiel soll dies verdeutlichen: Definieren wir die Menge R_x als die Menge all der Zeiten t die es erlauben von einem Startzustand s_x zum selbigen mit positiver Wahrscheinlichkeit zurückzukehren.

$$R_x = \{t|P(X_t = s_x|X_0 = s_x) > 0\} \tag{2.5}$$

Nun können wir für die Markow-Kette in Abbildung 2.2 folgende Tabelle erstellen.

	$t \in R_x$										
x	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	...	R_x	$\gcd R_x$
a	x			x			x			$\{0,3,6,\dots\}$	3
b	x			x			x		...	$\{0,3,6,\dots\}$	3
c	x			x			x			$\{0,3,6,\dots\}$	3

Tabelle 2.1.: Analyse von Periodizität der Markow-Kette in Abbildung 2.2

Da der gcd der einzelnen R_x gleich 3 ist, sehen wir, dass diese Markow-Kette periodisch ist. Wollen wir aus dieser Kette eine aperiodische machen, können wir sie wie folgt erweitern:

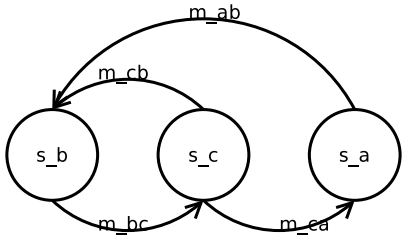


Abbildung 2.3.: Aperiodische Markow-Kette

	$t \in R_x$										
x	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	...	R_x	$\gcd R_x$
a	x			x		x	x	x		$\{0,3,5,6,7,\dots\}$	1
b	x		x	x	x	x	x	x	...	$\{0,2,3,4,5,6,7,\dots\}$	1
c	x		x	x	x	x	x	x		$\{0,2,3,4,5,6,7,\dots\}$	1

Tabelle 2.2.: Analyse von Periodizität der Markow-Kette in Abbildung 2.3

2.4. Stationäre Verteilung

Eine stationäre Verteilung ist die Zustandswahrscheinlichkeit einer unendlich langen Markow-Kette.

Definition 5. *Eine Markow-Kette mit Übergansmatrix M hat eine stationäre Verteilung π wenn gilt*

$$\pi M = \pi \tag{2.6}$$

Dies bedeutet: Jede Spalte b der Übergansmatrix M summiert mit dem Vektor π, muss wieder den Vektor π ergeben.

$$\forall_b \sum_a \pi_a M_{ab} = \pi_b \tag{2.7}$$

Eine stationäre Verteilung π ist ein Eigenvektor zum Eigenwert 1 von M.

2.5. Detailiertes Gleichgewicht

Detailiertes Gleichgewicht sagt aus, dass die Wahrscheinlichkeitsmasse die jedem Zustand abfliesst gleich der zufliessenden Wahrscheinlichkeitsmasse ist.

Definition 6. *Eine Wahrscheinlichkeitsfunktion π auf X erfüllt detailliertes Gleichgewicht hinsichtlich M wenn gilt*

$$\forall_{a,b} \pi_a M_{ab} = \pi_b M_{ba} \tag{2.8}$$

Diese Eigenschaft ist für den Metropolisalgorithmus besonders interessant, denn das Theorem 1 verbindet Übergangsmatrix und stationärer Verteilung. Wir werden diese Eigenschaft im Abschnitt 3.3 verwenden, um die Korrektheit des Algorithmus zu verifizieren.

Theorem 1. *Wenn eine Wahrscheinlichkeitsfunktion π mit einer Matrix M detailliertes Gleichgewicht erfüllt und M Übergangsmatrix einer Markow-Kette ist, ist π die stationäre Verteilung der Kette.*

$$\forall_{a,b} \pi_a M_{ab} = \pi_b M_{ba} \implies \pi M = \pi \tag{2.9}$$

Beweis. $\implies$: Stationäre Verteilung gemäss Definition 5 bedeutet

$$\sum_a \pi_a M_{ab} = \pi_b \tag{2.10}$$

durch einsetzen von Definition 6 erhalten wir

$$\sum_a \pi_b M_{ba} = \pi_b \tag{2.11}$$

und mit Assoziativität der Summe

$$\pi_b \sum_a M_{ba} = \pi_b \tag{2.12}$$

und der Zeileneigenschaft von Übergasmatrizen, Gleichung 1.6

$$\pi_b 1 = \pi_b \tag{2.13}$$

□

Markow-Ketten die dem detaillierten Gleichgewicht genügen, werden in der Literatur oft als reversibel bezeichnet.

2.6. Ergodisches Theorem für Markow-Ketten

Das für das Verständnis des Metropolisalgorithmus wichtige Theorem können wir in zwei Teile aufspalten.

Theorem 2. *X seien Glieder einer irreduzierbaren und zeithomogenen Markow-Kette mit stationärer Verteilung π, dann*

$$\frac{1}{t}\sum_{i=0}^{t} f(X_i) \xrightarrow[t\to\infty]{f.s.} Ef(X) \tag{2.14}$$

wobei X gemäss π verteilt ist, für jede beschränkte Funktion $f : X \to \mathbb{R}$

Theorem 2 besagt, dass das arithmetische Mittel einer unendlich langen, irreduzierbaren und zeithomogenen Markow-Kette fast sicher gegen den Erwartungswert einer beliebig verteilten Zufallsvariable $Ef(X)$ strebt. Bedingt, dass die stationäre Verteilung der Kette π gleich der Verteilung der Zufallsvariable X ist.

Theorem 3. *X seien Glieder einer ergodischen Markow-Kette mit stationärer Verteilung π, dann*

$$P(X_t = s_x | X_0 = s_0) \xrightarrow[t\to\infty]{f.s.} \pi_x \tag{2.15}$$

Beweis. Der Beweis würde den Rahmen dieser Arbeit sprengen, kann aber in [1] nachgeschlagen werden. □

Theorem 3 besagt, dass ein Sampel x von einer unendlich langen ergodischen Markow-Kette, gemäss π verteilt ist. Wichtig hierbei ist, dass die Wahl des Ausgangszustandes x_0 keine Auswirkung auf das Ergebnis hat. Einen ersten Hinweis für dieses Theorem, konnten wir in Kapitel 1.3 sehen.

Nun kennen wir die wichtigsten Eigenschaften und Definitionen von Markow-Ketten und können uns im Kapitel 3 dem Metropolisalgorithmus zuwenden.

3. Metropolisalgorithmus

Der Algorithmus wurde erstmals in einem Paper von Nicholas Metropolis et al. 1953 [11] als modifizierter Monte Carlo Algorithmus eingeführt:

> "So the method we employ is actually a modified Monte Carlo scheme, where, instead of choosing configurations randomly, then weighting them with exp (- E/ kT), we choose configurations with a probability exp (- E/ kT) and weight them evenly." [11, S. 1]

In diesem Kapitel wird der Metropolisalgorighmus beschrieben, eine Beispielimplementation vorgestellt und bewiesen weshalb der Algorithmus funktioniert.

3.1. Der Algorithmus

Ausgangslage sei ein System mit Zuständen **S**. Wir betrachten alle möglichen Zustände S gemäss Wahrscheinlichkeitsfunktion π

$$S = \{s \in \mathbf{S} | \pi_s > 0\} \tag{3.1}$$

und einer Häufigkeitsfunktion $f : S \to \mathbb{R}$ mit $S \propto \pi$. Dabei kann von π nur sehr schwierig gesampelt werden.

$$P(X = s_x) = \pi_x \tag{3.2}$$

Es ist also schwierig für ein gegebenes π_x den dazugehörigen Zustand s_x zu finden. Der Metropolisalgorithmus erzeugt eine Markow-Kette, deren Glieder gemäss π verteilt sind, ohne aber direkt von dieser Verteilung samplen zu müssen. Dazu verwendet der Algorithmus die Wahrscheinlichkeitsverteilung $\tilde{\pi}$ die lediglich proportional zu π sein muss:

$$\pi = \frac{\tilde{\pi}}{z}, \text{ mit } (z > 0) \tag{3.3}$$

Mit der Hilfe dieser Kette können wir danach annähernd von der Verteilung samplen oder aber den Erwartungswert approximieren [9]. Als Pseudocode sieht der Metropolisalgorithmus wie folgt aus:

Algorithm 1 Metropolisalgorithmus

```
1:  procedure Metropolis
2:      X_0 ← Startzustand ∈ S
3:      Q ← Vorschlagsmatrix, mit Q_ij = p(s_j|s_i) = p(s_i|s_j) = Q_ji
4:      n ← Iterationen
5:      t ← 0
6:      for t → n do
7:          φ ← sample von Q_{X_t}
8:          if rand(0,1) ≤ min(1, π̃_φ / π̃_{X_t}) then
9:              X_{t+1} ← φ
10:         else
11:             X_{t+1} ← X_t
```

In der Initialisierungsphase wird ein Startzustand X_0 gewählt. Dieser Zustand ist das erste Glied der Markow-Kette. Wie bereits im Kapitel 2.6 gesehen, kann der Startzustand beliebig gewählt werden. Jedoch kann ein günstiger Startzustand die Konvergenz des Algorithmus beschleunigen. Die Vorschlagsmatrix Q weist jedem Zustand s_i eine Wahrscheinlichkeit zu in jeden anderen Zustand s_j zu gelangen. Dabei muss Q symmetrisch sein und die Markow-Kette, die mit Q gebildet werden kann, ergodisch sein.

$$Q_{ij} = p(s_j|s_i) = p(s_i|s_j) = Q_{ji} \tag{3.4}$$

Wichtig zu beachten gilt, dass die Vorschlagsmatrix Q unter diesen Bedingungen frei gewählt werden kann. Q ist ein sogenannter Random Walk [5]. Weiter gibt n die Anzahl zu berechnenden Glieder der Markow-Kette vor. t ist eine Hilfsvariable die mit den Iterationen hochgezählt wird.

Nach der Initialisierungsphase wählt der Algorithmus in Abhängigkeit der Verteilung Q_{X_t} ein Vorschlagszustand φ. Danach wird das Verhältnis der Wahrscheinlichkeiten des Vorschlages $\tilde{\pi}_\varphi$ und des momentanen Zustandes $\tilde{\pi}_{X_t}$ bestimmt. Ist der Vorschlagszustand wahrscheinlicher, gilt $\frac{\tilde{\pi}_\varphi}{\tilde{\pi}_{X_t}} > 1$ und die Bedingung ist immer wahr. Ist der Vorschlagszustand weniger wahrscheinlich $\frac{\tilde{\pi}_\varphi}{\tilde{\pi}_{X_t}} < 1$, kann die Bedingung wahr werden wenn eine Zufallszahl kleiner ist als dieses Verhältnis. Dieser Vorgang kann mit dem Werfen einer unfairen Münze verglichen werden, wobei die Chance die Gewinnerseite zu sehen $\frac{\tilde{\pi}_\varphi}{\tilde{\pi}_{X_t}}$ ist. Im Falle einer wahren Bedingung ist der Vorschlagszustand das neue Glied der Markow-Kette, sonst wird der bisherige Zustand zum neuen Glied.

$$p(\text{accept } \varphi|\varphi, X_t) = min(1, \frac{\tilde{\pi}_\varphi}{\tilde{\pi}_{X_t}}) \tag{3.5}$$

Gleichung 3.5 gibt eben besprochene Akzeptanzwahrscheinlichkeit an. Dieser Vorgang wird solange wiederholt bis die Markow-Kette genau n Glieder lang ist.

Im nächsten Abschnitt soll nun eine einfache Anwendung des Algorithmus vorgestellt werden.

3.2. Beispiel: Einfache Verteilung sampeln

Ziel dieses Beispieles ist es exemplarisch zu zeigen, dass der Metropolisalgorithmus eine Markow-Kette erstellen kann deren Glieder gemäss einer vordefinierten Verteilung verteilt sind. Eine Konkrete Anwendung des Theorems 3. Wir wählen eine einfache Verteilung: die Normalverteilung mit Mittelwert 0 und Standardabweichung 1. Der Einfachheit halber ist die Vorschlagwahrscheinlichkeit $\tilde{\pi}$ gleich der gewünschten Verteilung.

$$\pi = \tilde{\pi} = NormalDistribution(0, 1) \tag{3.6}$$

Die komplette verwendete Mathematica-Implementation kann im Anhang B nachgeschlagen werden. Der Kern des Algorithmus ist in der Funktion sample implementiert.

```
sample[n_,dist_,x0_,d0_]:=
Module[{samples={},x=x0,x1=x0,d=d0},
Do[
(*Move randomly*)
x1=x+d*RandomReal[{-1,1}];
(*Flip the coin*)
If[
RandomReal[]<Min[1,PDF[dist,x1]/PDF[dist,x]] ,
x=x1,    (*Accept move*)
x=x      (*Discard move*)
];
(*Append to samples*)
AppendTo[samples,x];
,{i,n}];
Return[samples];
];
```

Abbildung 3.1.: Metropolisalgorithmus in Mathematica

Zeile 5 implementiert die Vorschlagsmatrix Q. Damit der Metropolisalgorithmus funktioniert muss Q die Gleichung 3.4 erfüllen.

Theorem 4. *s_i sei Startzustand, s_j Zielzustand und d die maximale Sprungweite. Die Vorschlagsmatrix*

$$Q_{ij} = \begin{cases} \alpha, & |i-j| < d \\ 0, & \textit{sonst} \end{cases}, \textit{ mit } \alpha > 0 \tag{3.7}$$

ist symmetrisch.

Beweis. Die Absolutwertfunktion ist eine gerade Funktion. Es gilt

$$|i-j| = |j-i| \tag{3.8}$$

Weiter ist d ist nicht von i und j abhängig. □

Somit kann die gewählte Vorschlagsmatrix Q für den Metropolisalgorithmus eingesetzt werden.

> Bemerkung: Die Gleichung 3.7 ist für eine überabzählbare Menge von Zuständen nicht korrekt. In diesem Fall gilt $\forall_x P(X = x) = 0$, was die Darstellung als Matrix verunmöglicht. Praktisch gesehen hat aber x eine maximale Genauigkeit und lässt somit nur eine endliche Zustandsmenge zu. Die Gleichung 3.7 kann also verwendet werden, auch wenn die positiven Wahrscheinlichkeiten α nicht präzise angegeben werden können.

Zeile 8 implementiert den nicht fairen Münzwurf gemäss $\tilde{\pi}$. Bei Erfolg wird in Zeile 9 der Vorschlagszustand übernommen oder bei Misserfolg in Zeile 10 verworfen. In der Abbildung 3.7 sehen wir wie die Markow-Kette durch den Metropolisalgorithmus berechnet wird. Die rote Linie ist gleich der Wahrscheinlichkeitsfunktion der gewählten Verteilung. Die blauen Punkte sind die einzelnen Glieder der Markow-Kette. Die blauen Balken sind die relativen Häufigkeiten der einzelnen Zustände als Histogramm dargestellt.

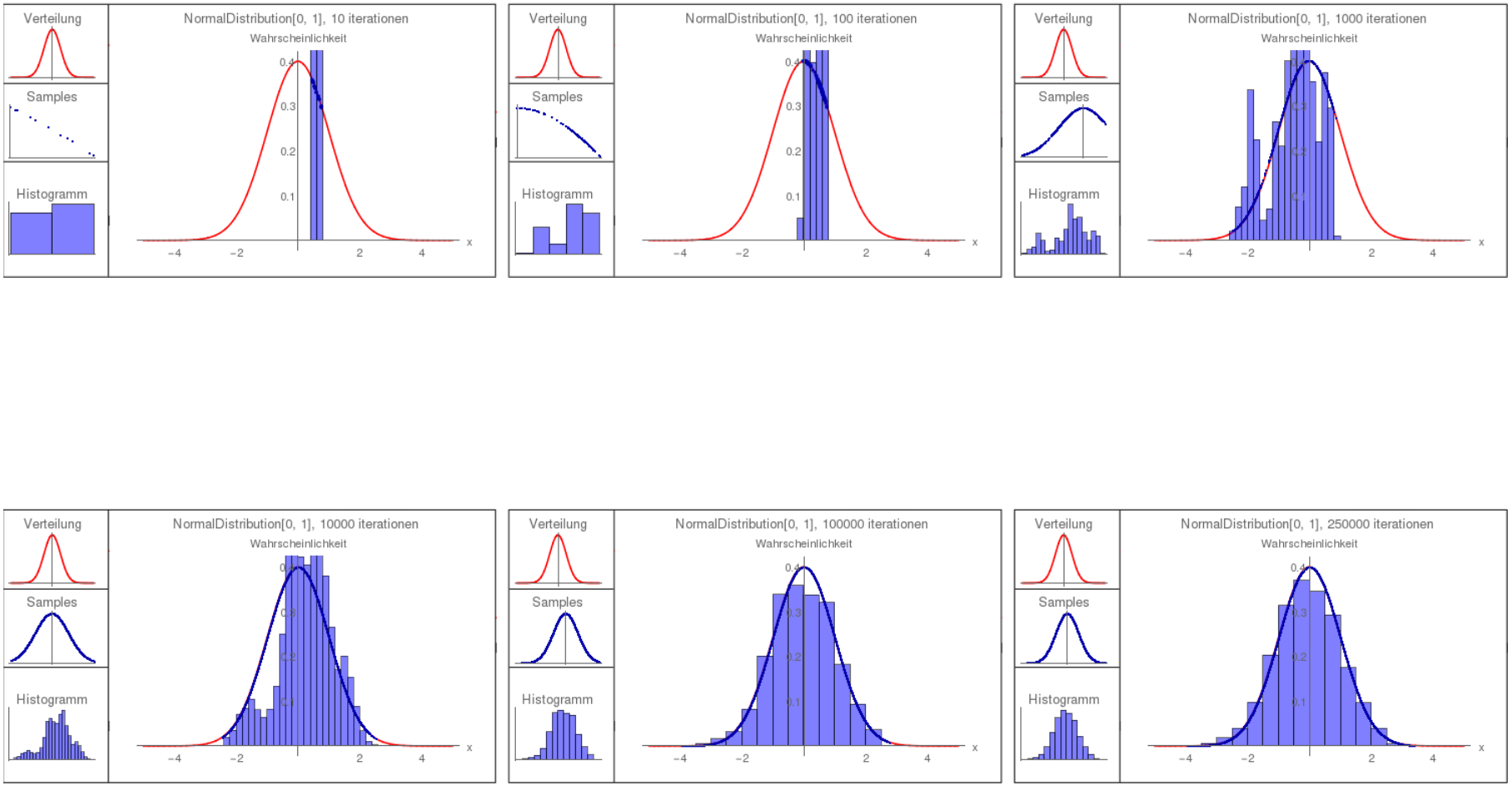


Abbildung 3.2.: Normalverteilung mit Metropolisalgorithmus approximiert

Wir sehen, dass sich das Histogramm langsam der Wahrscheinlichkeitsfunktion der zugrundeliegenden Verteilung annähert. Diese Implementation funktioniert für beliebige Verteilungen. Weitere Beispiele sind im Anhang C aufgelistet. Der Algorithmus scheint also zu funktionieren. Das nächste Kapitel beweist dies.

3.3. Korrektheit des Algorithmus

Nach [6]: Der Metropolisalgorithmus berechnet Glieder X einer Markow-Kette. Um zu zeigen, dass diese gemäss π verteilt sind, müssen wir laut Theorem 3 die Ergodizität der Markow-Kette beweisen. Als Vorarbeit zum Theorem 5 wollen wir aber erst die Übergangsmatrix M der Markow-Kette genauer definieren. Wir unterscheiden dazu zwei Fälle. Die diagonale der Matrix $i = j$ und alle anderen Zellen $i \neq j$.

$$M_{ij} = \begin{cases} 1 - \sum_{k \neq i} M_{ik}, & i = j \\ P(X_{t+1} = s_j | X_t = s_i), & \text{sonst} \end{cases} \tag{3.9}$$

Der Fall $i = j$ nutzt hierbei die Summeneigenschaft von Übergangsmatrizen aus, Gleichung 1.6. Betrachten wir $i \neq j$ so steht dort die Wahrscheinlichkeit $P(X_{t+1} = s_j | X_t = s_i)$ welche für den Metropolisalgorithmus gleich der Wahrscheinlichkeit für die Wahl eines Vorschlagzustandes $p(s_j|s_i)$ multipliziert mit der Akzeptanzwahrscheinlichkeit $p(\text{accept } \varphi|\varphi, X_t)$ ist. Wobei gilt $\varphi = s_j$ und $X_t = s_i$.

$$P(X_{t+1} = s_j | X_t = s_i) = p(s_j|s_i) p(\text{accept } s_j|s_j, s_i) \tag{3.10}$$

Durch einsetzen der Gleichung 3.4 und 3.5 in 3.10 erhält man für 3.9 die neue Gleichung 3.11.

$$M_{ij} = \begin{cases} 1 - \sum_{k \neq i} M_{ik}, & i = j \\ Q_{ij} min(1, \frac{\tilde{\pi}_j}{\tilde{\pi}_i}), & \text{sonst} \end{cases} \tag{3.11}$$

Mit diesem Verständnis für die Übergangsmatrix können wir uns dem Theorem 5 zuwenden.

Theorem 5. *Eine durch den Metropolisalgorithmus erzeugte Markow-Kette ist ergodisch wenn die Vorschlagsmatrix Q ergodisch ist.*

Beweis. Ergodizität bedingt gemäss Definition 1 verschiedene Eigenschaften. Diese werden nachfolgend bewiesen.

Teil 1. *Stationäre Verteilung: Um zu zeigen, dass die Markow-Kette stationäre Verteilung π hat, zeigen wir, dass π mit Übergangsmatrix M detailliertes Gleichgewicht erfüllt. Laut Theorem 1 müssen wir also folgendes zeigen:*

$$\pi_i M_{ij} \stackrel{?}{=} \pi_j M_{ji} \tag{3.12}$$

Wir unterscheiden wieder die Fälle $i = j$ und $i \neq j$. Für $i = j$ gilt

$$\pi_i M_{ii} = \pi_i M_{ii} \tag{3.13}$$

und ist wahr. Für $i \neq j$ ersetzen wir M_{ij} gemäss Gleichung 3.11 und erhalten

$$\pi_i M_{ij} = \pi_i Q_{ij} min(1, \frac{\tilde{\pi}_j}{\tilde{\pi}_i}) \tag{3.14}$$

Weiter ersetzen wir mit Hilfe der Gleichung 3.3 die Verteilung $\tilde{\pi}$ mit π und stellen dabei fest, dass sich der Proportionalitätsfaktor z kürzt.

$$\pi_i M_{ij} = \pi_i Q_{ij} min(1, \frac{\pi_j}{\pi_i}) \tag{3.15}$$

π_i ist eine Wahrscheinlichkeitsverteilung, also gilt $\forall_i \pi_i \geq 0$. So können wir π_i in $min()$ hinein multiplizieren.

$$\pi_i M_{ij} = Q_{ij} min(\pi_i, \pi_j) \tag{3.16}$$

Gemäss Bedingung 3.4 ist die Vorschlagsmatrix Q_{ij} symmetrisch. Weiter ist $min()$ kommutativ. Somit gilt

$$\pi_i M_{ij} = Q_{ij} min(\pi_i, \pi_j) = Q_{ji} min(\pi_j, \pi_i) = \pi_j M_{ij} \tag{3.17}$$

Teil 2. *Zeithomogen, irreduzierbar, aperiodisch: Gemäss Bedingung 3.1 könne alle Zustände s_x auftreten.*

$$\forall_x \pi_x > 0 \tag{3.18}$$

Laut Gleichung 3.3 gilt $\pi \propto \tilde{\pi}$. Damit folgt für die Gleichung 3.11,

$$\forall_{i,j} min(1, \frac{\tilde{\pi}_j}{\tilde{\pi}_i}) > 0 \tag{3.19}$$

Aus den Definitionen 2, 3 und 4 sehen wir, dass diese Eigenschaften bezüglich der Übergangsmatrix nur die Fälle $m_{ij} = 0$ und $m_{ij} \neq 0$ unterscheiden. Mit 3.19 gilt:

$$\begin{aligned} m_{ij} \neq 0 &\implies m_{ij} min(1, \frac{\tilde{\pi}_j}{\tilde{\pi}_i}) \neq 0 \\ m_{ij} = 0 &\implies m_{ij} min(1, \frac{\tilde{\pi}_j}{\tilde{\pi}_i}) = 0 \end{aligned} \tag{3.20}$$

□

Wir haben nun gesehen, dass der Metropolisalgorithmus funktioniert. Im Kapitel 3.2 wurde zudem bereits ein einführendes Beispiel besprochen. Im nächsten Teil soll nun ein etwas umfassenderes Beispiel besprochen werden.

3.4. Beispiel: Monoalphabetische Substitution

Inspiriert von [3]: In diesem Beispiel wollen wir ein Problem angehen, dass für den Metropolisalgorithmus besser geeignet ist als die bislang besprochenen. Eine Ciphertext c, welcher mittels einer monoalphabetischen Substitution [4] erstellt wurde, soll invertiert werden. Eine monoalphabetische Substitution ist ein einfaches Verschlüsselungsverfahren welches Zeichen aus dem Codespace C substituiert. Die Verschlüsselungsfunktion $f : C \to C$ kann als gerichteter Graph interpretiert werden, wobei jeder Vertex ein Element aus C ist. Die entsprechende Entschlüsselungsfunktion $f^{-1} : C \to C$, ist ein ähnlicher Graph mit invertierten Kanten.

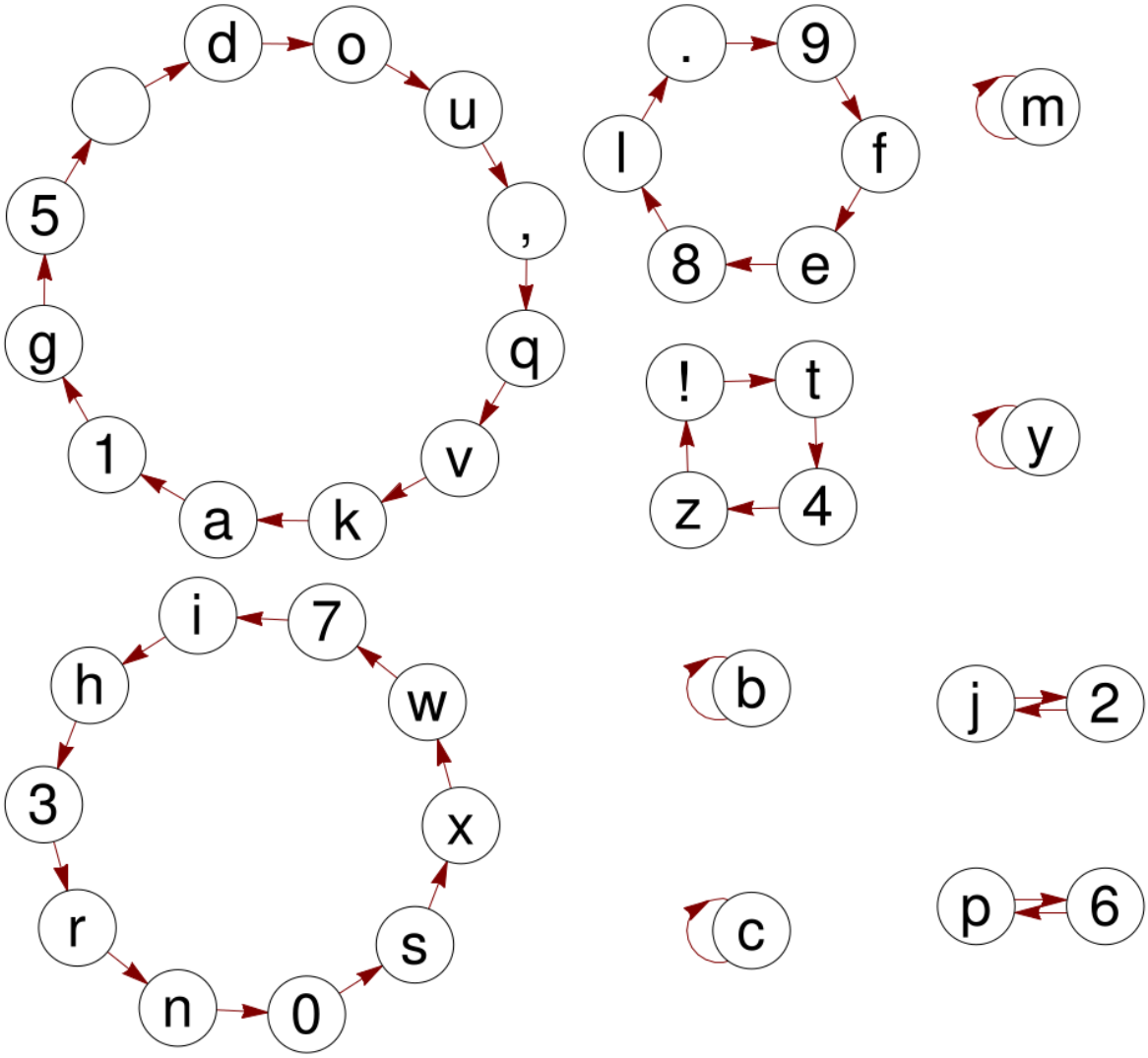


Abbildung 3.3.: Verschlüsselungsfunktion

Die gewählte Verschlüsselungsfunktion in diesem Beispiel ist in der Abbildung 3.3 zu sehen.

3.4.1. Kryptoanalyse

Ausgehend von einer zufällig gewählten Cipher f_0^{-1} erzeugen wir eine Markow-Kette mit stationärer Verteilung $\tilde{\pi}$. Die Wahrscheinlichkeit $\tilde{\pi}$ ist die natürliche Folgewahrscheinlichkeit F zweier Buchstaben in einem Text entschlüsselt mit Cipher f^{-1}. Diese Folgewahrscheinlichkeit F wird exemplarisch aus einer genügend grossen Stichprobe T abgeleitet. Der verwendete Ansatz kann durch folgende Pseudocode repräsentiert werden:

Algorithm 2 Kryptoanalyse

```
1: procedure Kryptoanalyse
2:     f_0^{-1} ← Zufällige Cipher
3:     n ← Iterationen
4:     X ← Metropolis(f_0^{-1}, Q, n)
5: procedure π̃
6:     f^{-1} ← Cipher
7:     c ← Verschlüsselter Text
8:     F ← Folgewahrscheinlichkeit()
9:     α ← 1
10:    for i → |c| do
11:        α ← α · F_{f^{-1}(c_i) f^{-1}(c_{i+1})}
12:    return α
13: procedure Folgewahrscheinlichkeit
14:    T ← Referenztext
15:    F ← ℚ^{|C|x|C|}, mit F_ij = 0
16:    for i → |T| do
17:        F_{T_i T_{i+1}} ← F_{T_i T_{i+1}} + 1
18:    return F / |T|
19: procedure Q
20:    f^{-1} ← Cipher
21:    return Vertausche zwei Kanten in f^{-1}
```

Schauen wir uns die einzelnen Prozeduren genauer an, wird klar wie der Algorithmus funktioniert.

Kryptoanalyse

Dies ist der Einstiegspunkt des Algorithmus. Es wird eine f^{-1} zufällig gewählt, eine Anzahl Iterationen bestimmt und der Metropolisalgorithmus gestartet.

$\tilde{\pi}$

Für eine gegebene Cipher f^{-1} kann die Wahrscheinlichkeit $\tilde{\pi}$ als wiederholtes Zufallsexperiment über den gesamten Ciphertext c intepretiert werden. Wobei die Wahrscheinlichkeit für Erfolg gemäss der Folgewahrscheinlichkeit F verteilt ist.

$$\tilde{\pi}_{f^{-1}} = \prod_i F_{f^{-1}(c_i) f^{-1}(c_{i+1})} \tag{3.21}$$

Wir sehen auch, dass die Menge der möglichen Ciphern f^{-1} in Relation mit der Grösse des Codespaces C steht.

$$|\{f_0, \ldots, f_n\}| = |C|! \tag{3.22}$$

Die Fakultät ist eine sehr schnell wachsende Funktion und somit wird die Berechnung des korrekten Normirungsfaktors z, gemäss Gleichung 3.23, mit grösser werdendem Codespace schnell sehr schwierig.

$$z = \sum_f \prod_i F_{f^{-1}(c_i) f^{-1}(c_{i+1})} \tag{3.23}$$

Mit der wahren Verteilung $\pi = \frac{\tilde{\pi}}{z}$ zu arbeiten ist also schnell sehr schwierig. Wie aber in Kapitel 3.3 gezeigt, funktioniert der Metropolisalgorithmus auch mit $\tilde{\pi}$.

Folgewahrscheinlichkeit

Die Prozedur zählt alle Buchstabenfolgen in einem Referenztext T und normiert diese Häufigkeiten.

Q

Die Vorschlagsmatrix ist durch ein einfaches Vertauschen zweier Kanten implementiert. Wie im Kapitel 3.3 gesehen, muss die Vorschlagsmatrix symmetrisch sein. In diesem einfachen Fall vertrauen wir der Intuition und lassen den Beweis aus.

Viermaliges Ausführen der Implementation in Anhang D mit $n = 1000$ führt zum Ergebnis in Abbildung 3.4. Die erste Spalte zeigt die Anzahl korrekter Kanten von f^{-1} für jede Iteration. Die zweite Spalte ist ein Histogramm der relativen Häufigkeiten dieser richtigen Kanten. Die dritte und vierte Spalte zeigt jeweils den Graph der letzten sowie häufigsten Cipher. Der dazugehörige Text ist der entsprechend entschlüsselte c.

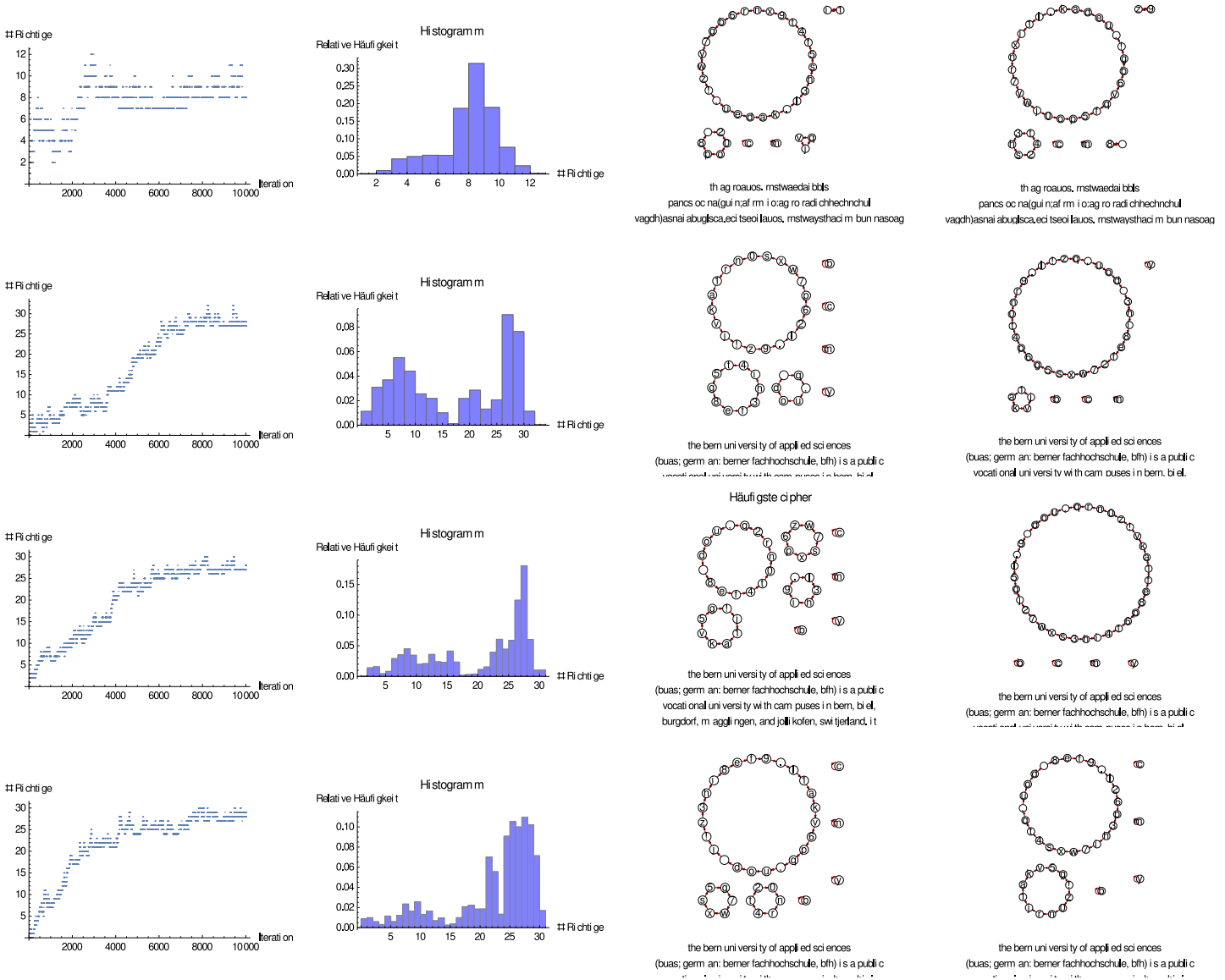


Abbildung 3.4.: Entschlüsselungsfunktion

Wir sehen, dass der erste Durchlauf keine brauchbare Entschlüsselungsfunktion gefunden hat. Da wir kein Mass der Konvergenz angegeben haben, ist dies nicht weiter erstaunlich. Die Kette hätte wohl mit grösserem n eine bessere Entschlüsselungsfunktion geliefert. Die restlichen drei Durchläufe sind besser konvergiert.

Die Symmetriebedingung an Q führt oftmals dazu, dass wir eine Übergangsmatrix wählen müssen welche wie in Durchlauf eins oder vier nur langsam konvergiert. Abhilfe bietet der Metropolis-Hastings-Algorithmus.

3.5. Metropolis-Hastings-Algorithmus

Der Metropolis-Hastings-Algorithmus erweitert den Metropolisalgorithmus um nicht symmetrische Vorschlagsmatrizen. Diese bedeutet, dass die Bedingung in Gleichung 3.4 nicht gelten muss. Somit kann eine beliebige Vorschlagsmatrix gewählt werden. Die Konvergenz des erweiterten Algorithmus wurde von W. K. Hastings [2] bewiesen. Der Metropolisalgorithmus muss dahingehen erweitert werden, dass die Akzeptanzwahrscheinlichkeit, gemäss Gleichung 3.5, neu gewichtet wird.

$$p(\text{accept } \varphi|\varphi, X_t) = min(1, \frac{\tilde{\pi}_{\varphi} p(X_t|\varphi)}{\tilde{\pi}_{X_t} p(\varphi|X_t)}) \tag{3.24}$$

Wollen wir den Metropolisalgorithmus stabiler konvergieren lassen, können wir ihn mit einer leichten Veränderung zur simulated annealing Heuristik umbauen.

4. Simulated annealing

Die simulated annealing Heuristik, motiviert durch Abkühlungsprozesse, soll die Konvergenz des Metropolisalgorithmus verbessern. Idee dahinter ist, mit fortschreitender Iterationszahl die Akzeptanzwahrscheinlichkeit für den ungünstigen Fall $\frac{\tilde{\pi}_\varphi}{\tilde{\pi}_{X_t}} < 0$ immer kleiner werden zu lassen. Dazu wählen wir eine monoton fallende Temperaturzerfallsfunktion τ

$$\tau : \mathbb{N}_0 \to \mathbb{R}, \text{ mit } \forall_{x<y} \tau(y) \leq \tau(x) \tag{4.1}$$

und passen entsprechen die Akzeptanzwahrscheinlichkeit an.

$$p(\text{accept } \varphi | \varphi, X_t) = min(1, (\frac{\tilde{\pi}_\varphi}{\tilde{\pi}_{X_t}})^{\frac{1}{\tau(t)}}) \tag{4.2}$$

Abbildung 4.1 stellt Akzeptanzwahrscheinlichkeiten für verschiedene Temperaturen $\tau(t)$ dar.

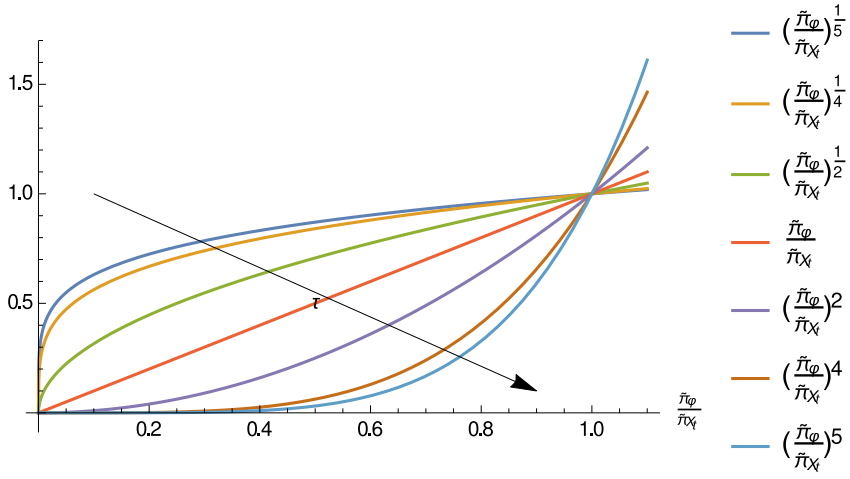


Abbildung 4.1.: Akzeptanzwahrscheinlichkeit für verschiedene Temperaturen

Studieren wir die veränderten Akzeptanzwahrscheinlichkeit fällt folgendes auf:

- Eine Verbesserung $\frac{\tilde{\pi}_\varphi}{\tilde{\pi}_{X_t}} > 0$ hat immer noch Wahrscheinlichkeiten in $[1, \infty)$ zur Folge und wird somit immer akzeptiert.
- Für eine Verschlechterung $\frac{\tilde{\pi}_\varphi}{\tilde{\pi}_{X_t}} < 0$ gilt:

$$p(\text{accept } \varphi | \varphi, X_t) \begin{cases} \text{wird grösser} & \tau(t) > 1 \\ \text{gleich wie Metropolisalgorithmus} & \tau(t) = 1 \\ \text{wird kleiner} & \tau(t) < 1 \end{cases} \tag{4.3}$$

- Die Akzeptanzwahrscheinlichkeit ändert sich mit laufendem t. Somit verändert sich die Übergansmatrix M mit t. Damit ist die Markow-Kette gemäss Definition 2 nicht mehr zeithomogen und weiter gemäss Definition 1 nicht mehr ergodisch. Da Theorem 2 und 3 nur für ergodische Ketten gelten, kann es sein, dass die Kette nicht mehr konvergiert.

Ein Beispiel kann im Anhang E gefunden werden.

5. Schluss

Mit dem Metropolisalgorithmus und der simulated annealing Heuristik haben wir zwei Algorithmen gefunden, die für ein breites Spektrum an Problemen eingesetzt werden können. Die in diesem Dokument vorgestellten Beispiele dienen der Illustration. Sie können durch andere Algorithmen effizienter gelöst werden. Die beiden vorgestellten Algorithmen eignen sich aber speziell für hochdimensionale Problemstellungen, für die herkömmliche Algorithmen meist versagen.

Literaturverzeichnis

[1] C. Casarotto, "Markov chains and the ergodic theorem," *University of Chicago*, 8 2007.

[2] P. Diaconis, "Monte carlo sampling methods using markov chains and their applications," *University of Toronto*, 1970.

[3] ——, "The markov chain monte carlo revolution," *Bulletin of the American Mathematical Society*, vol. 46, no. 2, pp. 179–205, 2009.

[4] diverse, "Monoalphabetische substitution," Mai 2016. [Online]. Available: https://de.wikipedia.org/wiki/Monoalphabetische_Substitution

[5] ——, "Random walk," Mai 2016. [Online]. Available: https://de.wikipedia.org/wiki/Random_Walk

[6] mathematicalmonk, "Correctness of the metropolis algorithm," July 2011. [Online]. Available: https://www.youtube.com/watch?v=0j-Pq1OU4LY

[7] ——, "Examples of markov chains with various properties (part 1)," July 2011. [Online]. Available: https://www.youtube.com/watch?v=Pce7KKeUf5w

[8] ——, "Examples of markov chains with various properties (part 2)," July 2011. [Online]. Available: https://www.youtube.com/watch?v=daY4lgEyEPc

[9] ——, "Metropolis algorithm for mcmc," July 2011. [Online]. Available: https://www.youtube.com/watch?v=gxHe9wAWuGQ

[10] ——, "Stationary distributions, irreducibility, and aperiodicity," July 2011. [Online]. Available: https://www.youtube.com/watch?v=tByUQbJdt14

[11] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, and A. H. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical Physics*, 3 1953.

[12] K.-H. Waldmann and U. M. Stocker, *Stochastische Modelle*. Springer-Verlag Berlin Heidelberg, 2013.

Abbildungsverzeichnis

Tabellenverzeichnis

A. Münzenspiel

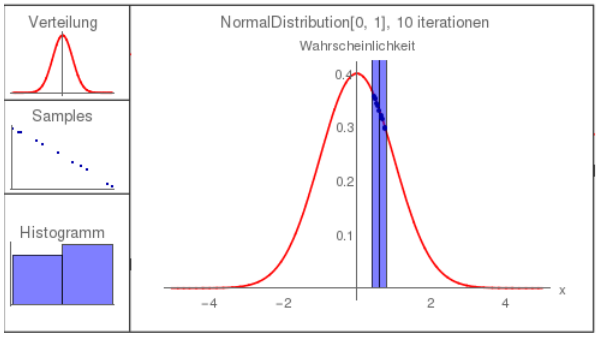
```
ClearAll["Global`*"]
M={
{1,0,0,0,0,0,0},
{1-p,0,p,0,0,0,0},
{0,1-p,0,p,0,0,0},
{0,0,1-p,0,p,0,0},
{0,0,0,1-p,0,p,0},
{0,0,0,0,1-p,0,p},
{0,0,0,0,0,0,1}
}
M // MatrixForm
p= 1/2
pi0 = {{0,0,0,0,1,0,0}};
pi0 // MatrixForm
pi[t_]:= pi0.MatrixPower[M,t]
PI=
Join[
{{0,1,2,3,4,5,6}},
pi0,
pi[1],
pi[2],
pi[3],
pi[4],
pi[5],
pi[10],
pi[20],
pi[30],
pi[40],
pi[50]
] // MatrixForm
TeXForm[
TableForm[
Round[
N[PI],0.01
]]]
```

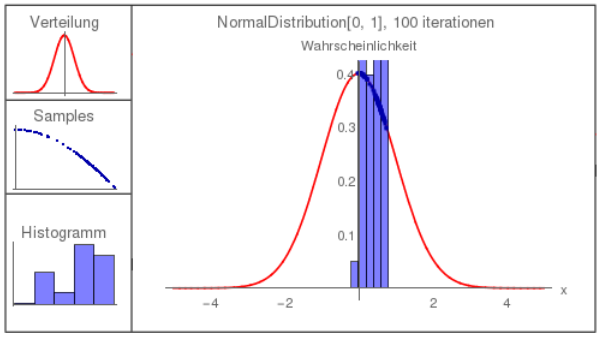
B. Verteilung sampeln

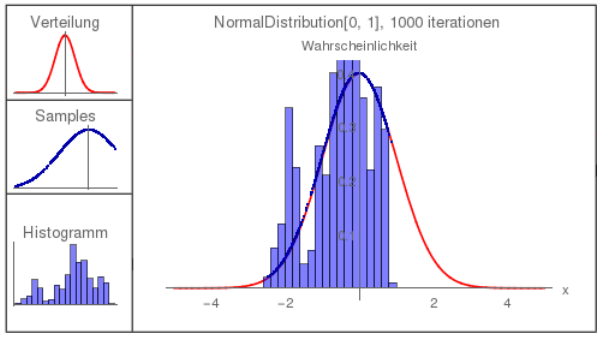
```
In[673]:= ClearAll["Global`*"];

sample[n_, dist_, x0_, d0_]:=
Module[{samples={},x=x0,x1=x0,d=d0},
Do[
(*Move randomly*)
x1=x+d*RandomReal[{-1,1}];
(*Flip the coin*)
If[
RandomReal[]<Min[1,PDF[dist,x1]/PDF[dist,x]] ,
x=x1,    (*Accept move*)
x=x      (*Discard move*)
];
(*Append to samples*)
AppendTo[samples,x];
,{i,n}];
Return[samples];
];

plot[samples_, dist_, min_, max_, n_]:=
Module[{d,p,h},
d=Plot[
PDF[dist,x],{x,min,max},
PlotStyle->Red,
PlotLegends->{"Ï€(x)"},
PlotLabel->"Verteilung"
];
p=ListPlot[
Function[x,{x,PDF[dist,x]}]/@samples,
PlotStyle->{Darker[Blue]},
PlotLegends->{"Subscript[X, t]"},
PlotLabel->"Samples"
];
h=Histogram[
samples,
Automatic,
"PDF",
ChartStyle->Opacity[.5,Blue],
ChartLegends->{"X"},
PlotLabel->"Histogramm"
];
Grid[{
{
Show[d,Ticks->None,ImageSize->Full],
Show[
d,h,p,
PlotLabel->StringForm["``, `` iterationen ",dist,n],
AxesLabel->{"x","Wahrscheinlichkeit"},
ImagePadding->25,
ImageSize->Full
]
},{
Show[p,Ticks->None,ImageSize->Full],SpanFromAbove
},{
Show[h,Ticks->None,ImageSize->Full],SpanFromAbove
}
},
Frame->All,
ItemSize->{{Scaled[.2],Scaled[.8]}},
Alignment->{Left,Center}
]
];
```

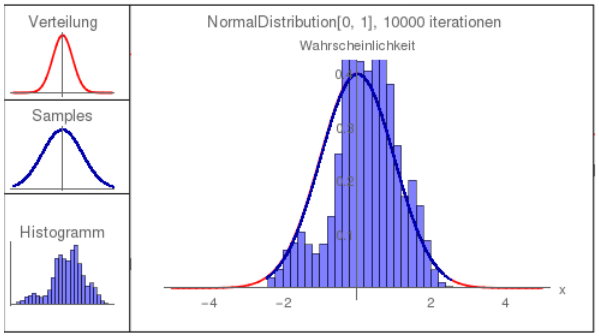
```
63
64 run[steps_,dists_,x0_,d0_,min_,max_]:=
65 Module[{dir=CreateDirectory[],plotsPerLine=3,imageSize=1500},
66 Do[
67 Module[{samples={x0},prevN=0,plots={}, grid,f=FileNameJoin[{dir,StringJoin[ToString[dist],".png"]}]},
68 Do[
69 samples=Join[samples,sample[n-prevN,dist,Last[samples],d0]];
70 plots=Append[plots,plot[samples,dist,min,max,n]];
71 prevN = n;
72 ,{n,steps}];
73 grid=GraphicsGrid[
74 Partition[plots,plotsPerLine,plotsPerLine,{1,1},{}],
75 Spacings->10,
76 ImageSize->imageSize
77 ];
78 Print[StringForm["Writing to: ``",f]];
79 Export[f,grid];
80 Print[grid];
81 ];
82 ,{dist,dists}];
83 ];
84
85 run[{10,100,1000,10000,100000,250000},{
86 NormalDistribution[],
87 HalfNormalDistribution[1],
88 LogNormalDistribution[0,1],
89 InverseGaussianDistribution[1,1],
90 ChiSquareDistribution[1],
91 ChiSquareDistribution[2],
92 ChiSquareDistribution[3],
93 StudentTDistribution[1],
94 StudentTDistribution[2],
95 StudentTDistribution[3],
96 TriangularDistribution[{-1,1}],
97 UniformDistribution[{-1,1}],
98 MaxwellDistribution[1]
99 },0.5,0.1,-5,5];
```

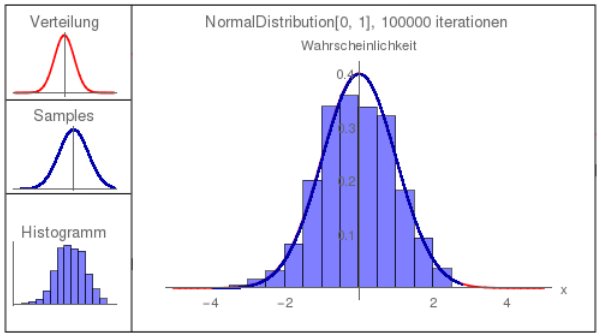
C. Diverse approximierte Verteilungen

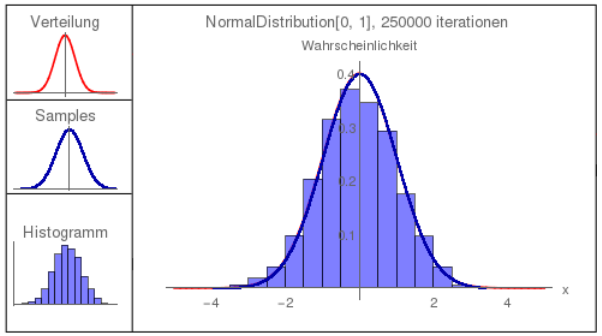

Verteilung
Samples
Histogramm
NormalDistribution[0, 1], 10 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

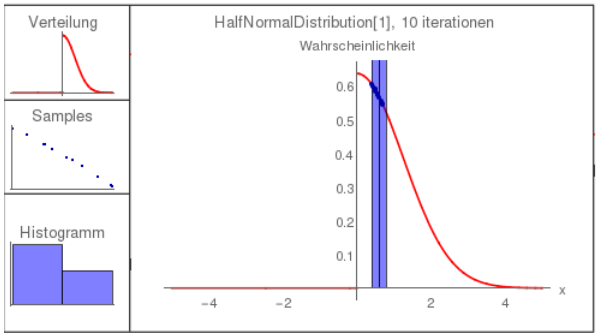

Verteilung
Samples
Histogramm
NormalDistribution[0, 1], 100 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

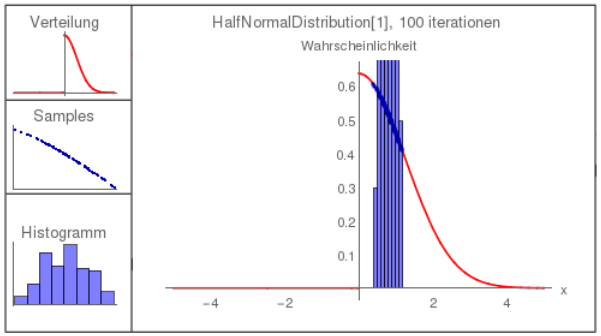

Verteilung
Samples
Histogramm
NormalDistribution[0, 1], 1000 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

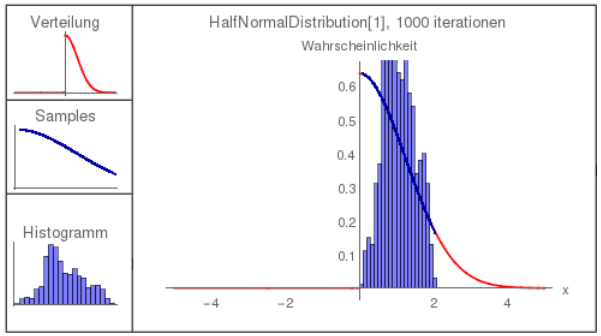

Verteilung
Samples
Histogramm
NormalDistribution[0, 1], 10000 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

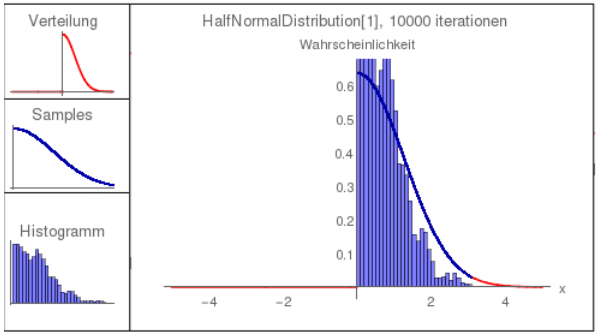

Verteilung
Samples
Histogramm
NormalDistribution[0, 1], 100000 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

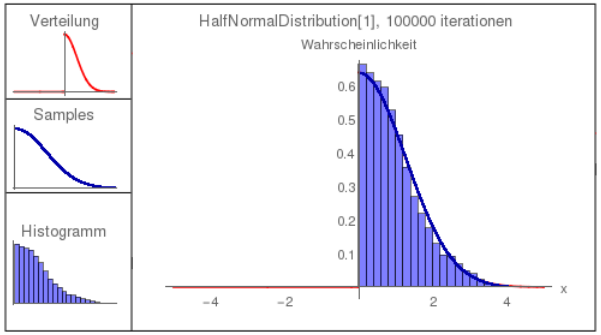

Verteilung
Samples
Histogramm
NormalDistribution[0, 1], 250000 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

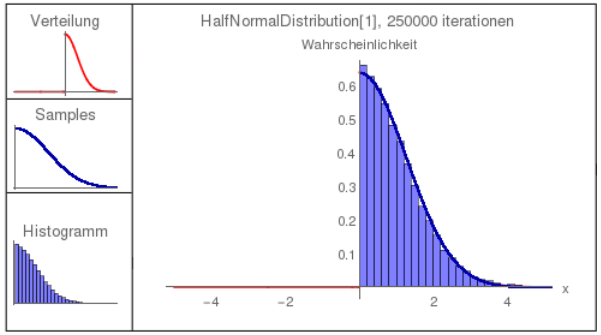

Verteilung
Samples
Histogramm
HalfNormalDistribution[1], 10 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

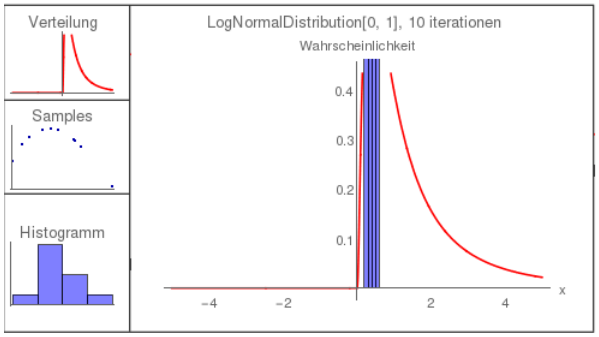

Verteilung
Samples
Histogramm
HalfNormalDistribution[1], 100 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

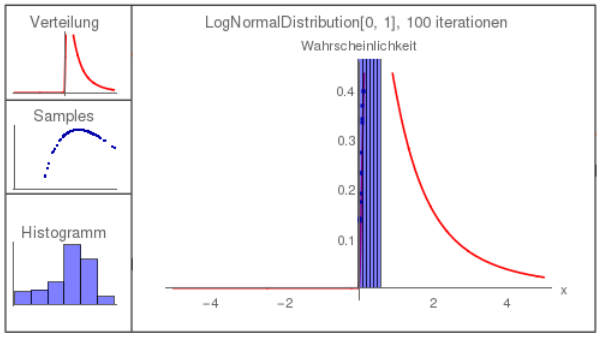

Verteilung
Samples
Histogramm
HalfNormalDistribution[1], 1000 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

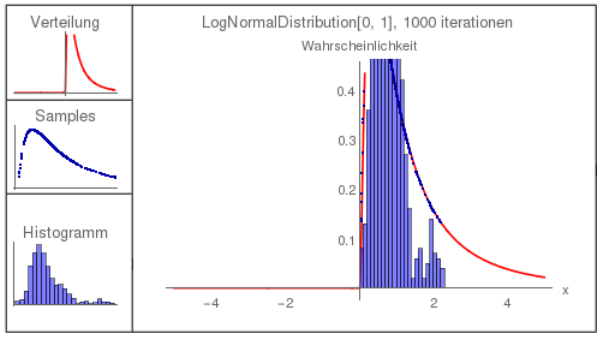

Verteilung
Samples
Histogramm
HalfNormalDistribution[1], 10000 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

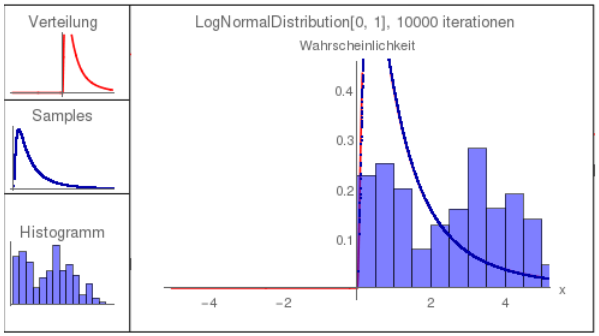

Verteilung
Samples
Histogramm
HalfNormalDistribution[1], 100000 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

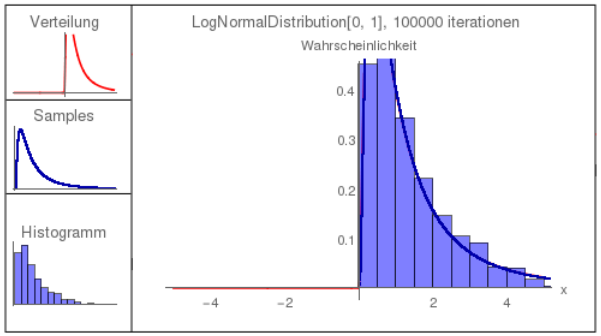

Verteilung
Samples
Histogramm
HalfNormalDistribution[1], 250000 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

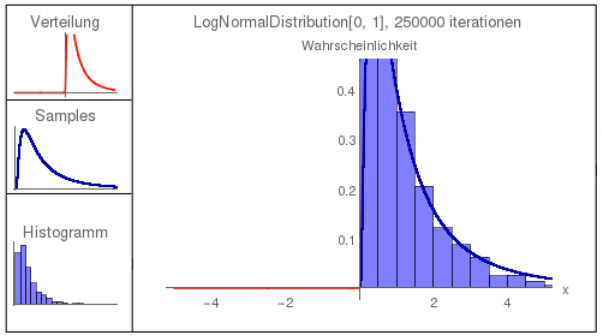

Verteilung
Samples
Histogramm
LogNormalDistribution[0, 1], 10 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

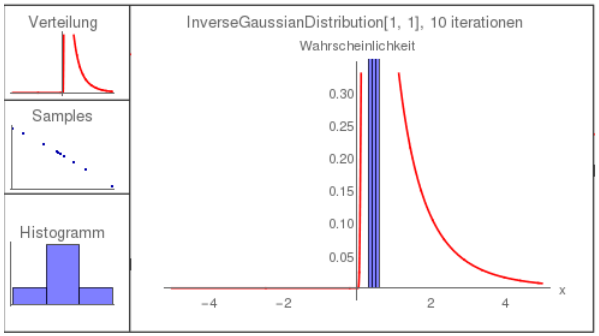

Verteilung
Samples
Histogramm
LogNormalDistribution[0, 1], 100 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

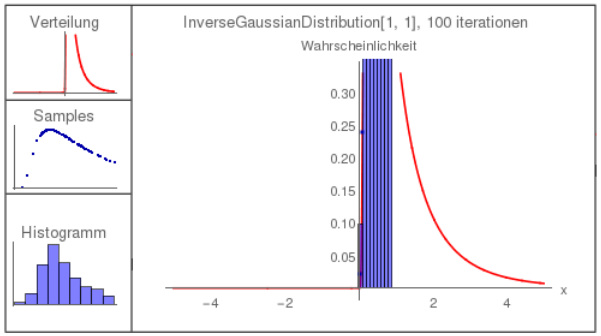

Verteilung
Samples
Histogramm
LogNormalDistribution[0, 1], 1000 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

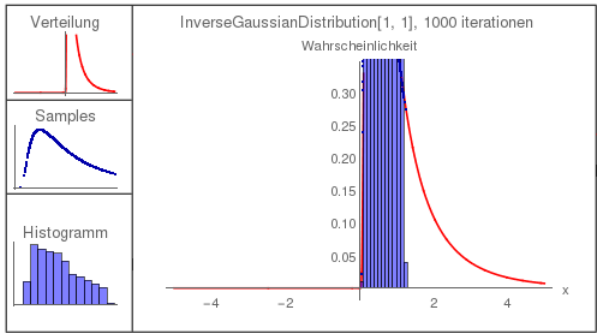

Verteilung
Samples
Histogramm
LogNormalDistribution[0, 1], 10000 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

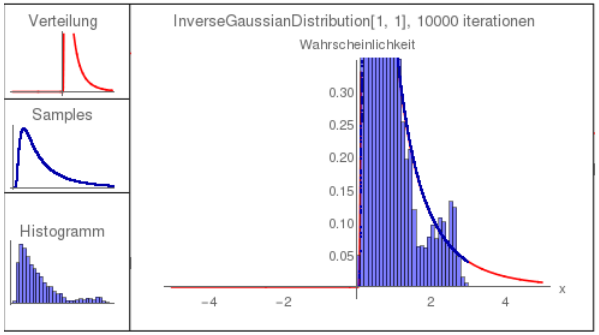

Verteilung
Samples
Histogramm
LogNormalDistribution[0, 1], 100000 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

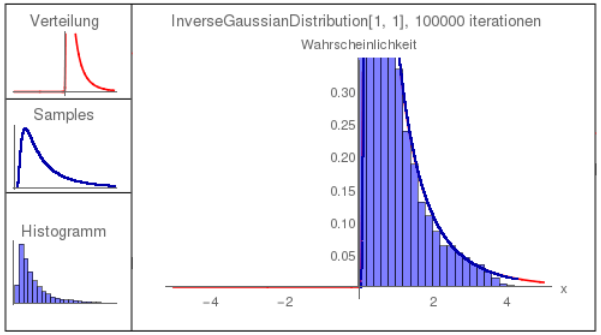

Verteilung
Samples
Histogramm
LogNormalDistribution[0, 1], 250000 iterationen
Wahrscheinlichkeit
0.4
0.3
0.2
0.1
−4
−2
2
4
x

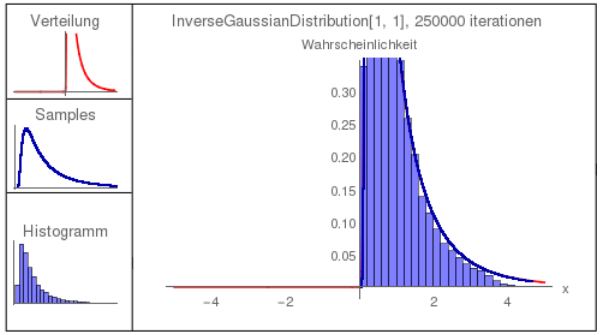

Verteilung
Samples
Histogramm
InverseGaussianDistribution[1, 1], 10 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

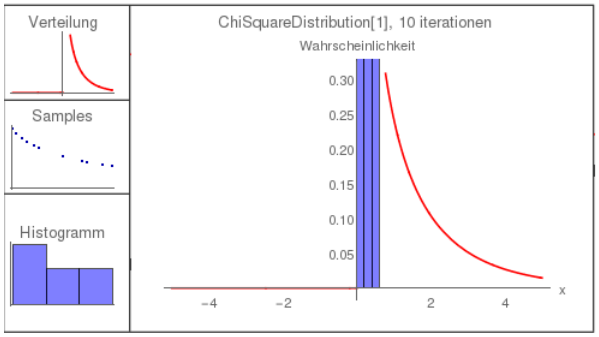

Verteilung
Samples
Histogramm
InverseGaussianDistribution[1, 1], 100 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

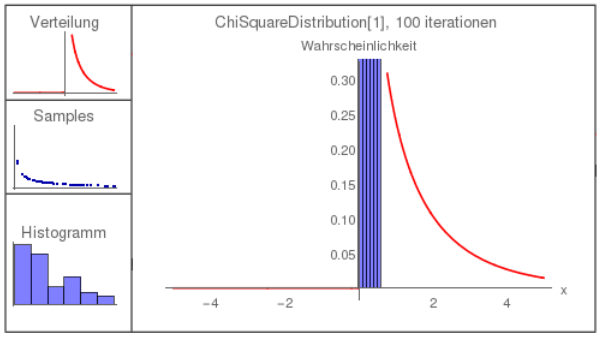

Verteilung
Samples
Histogramm
InverseGaussianDistribution[1, 1], 1000 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

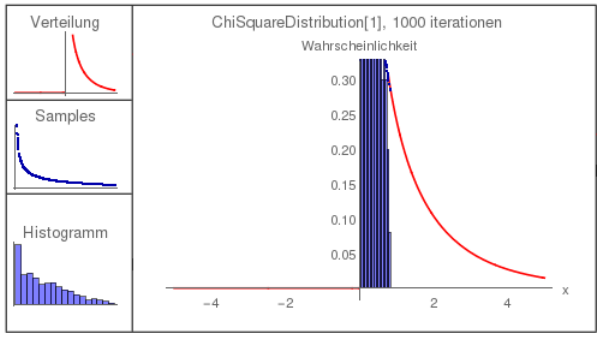

Verteilung
Samples
Histogramm
InverseGaussianDistribution[1, 1], 10000 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

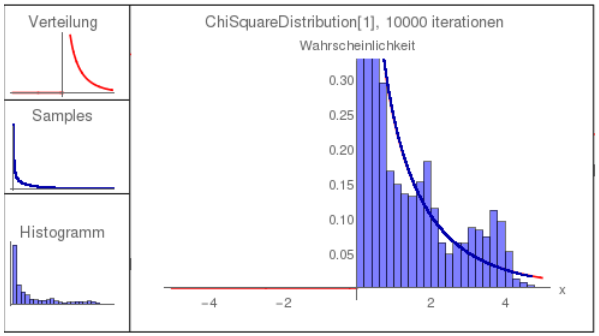

Verteilung
Samples
Histogramm
InverseGaussianDistribution[1, 1], 100000 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

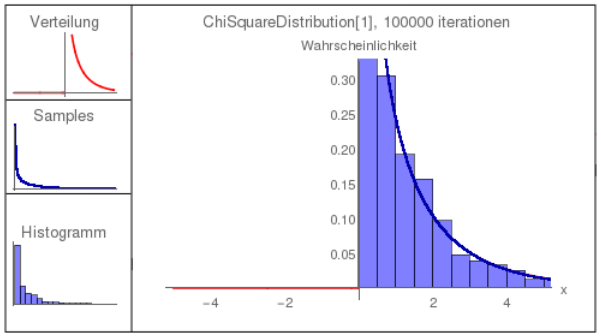

Verteilung
Samples
Histogramm
InverseGaussianDistribution[1, 1], 250000 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

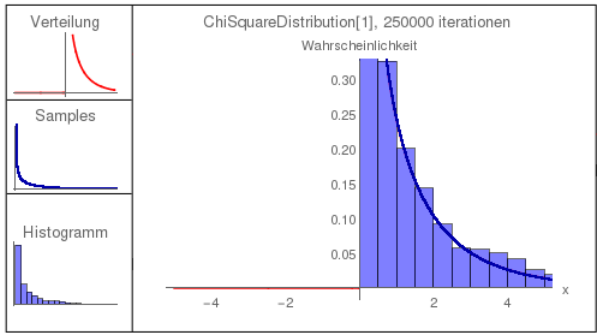

Verteilung
Samples
Histogramm
ChiSquareDistribution[1], 10 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

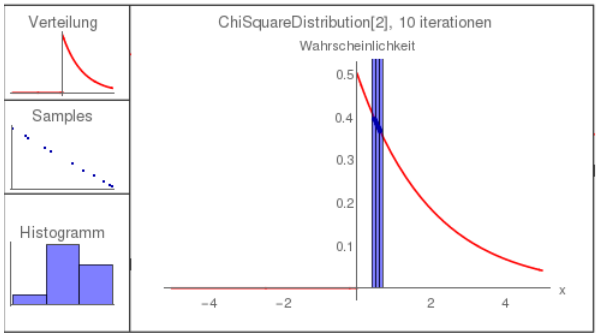

Verteilung
Samples
Histogramm
ChiSquareDistribution[1], 100 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

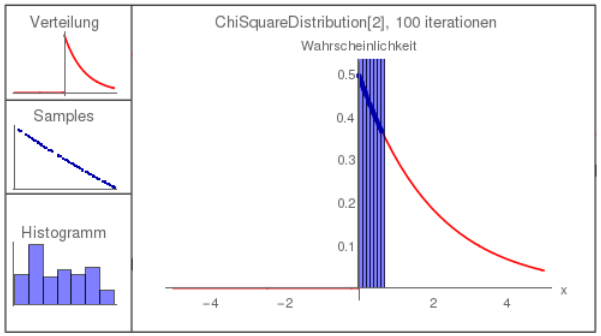

Verteilung
Samples
Histogramm
ChiSquareDistribution[1], 1000 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

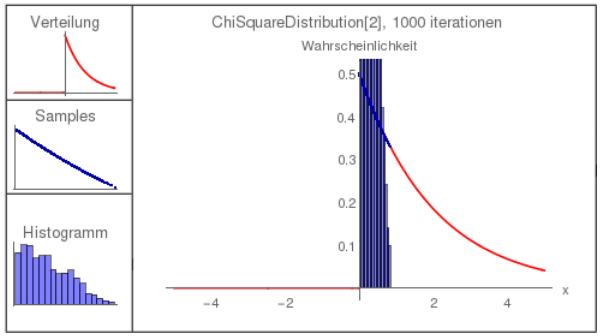

Verteilung
Samples
Histogramm
ChiSquareDistribution[1], 10000 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

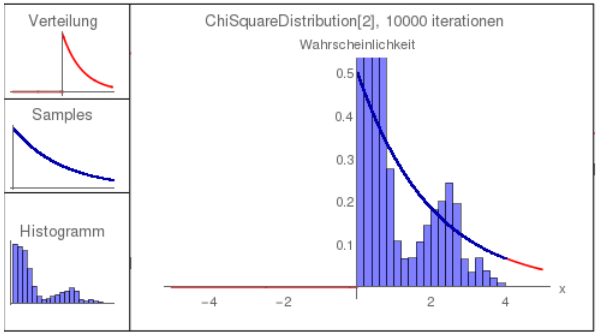

Verteilung
Samples
Histogramm
ChiSquareDistribution[1], 100000 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

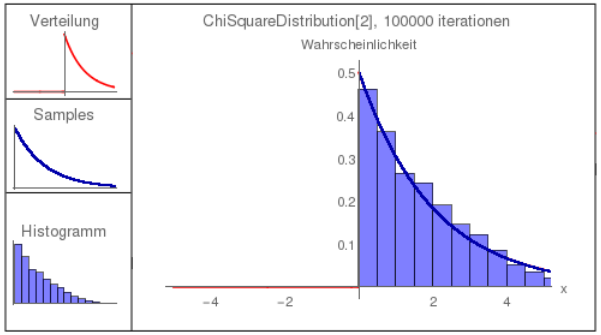

Verteilung
Samples
Histogramm
ChiSquareDistribution[1], 250000 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

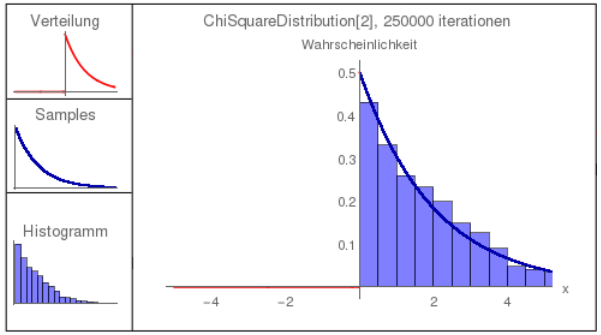

Verteilung
Samples
Histogramm
ChiSquareDistribution[2], 10 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

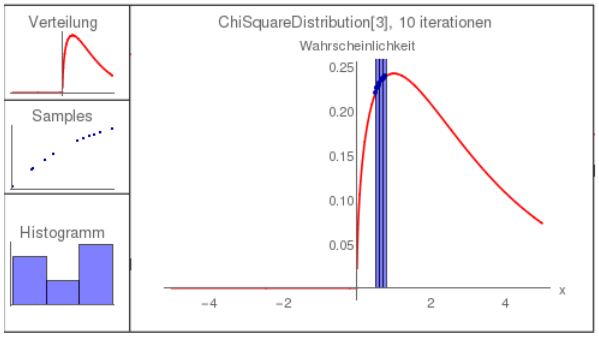

Verteilung
Samples
Histogramm
ChiSquareDistribution[2], 100 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

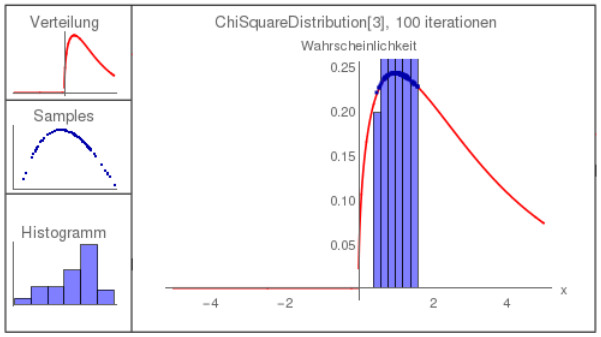

Verteilung
Samples
Histogramm
ChiSquareDistribution[2], 1000 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

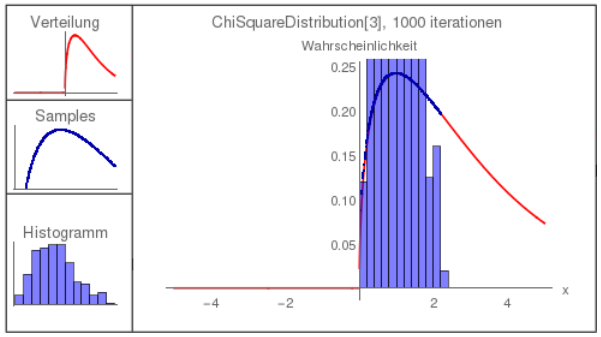

Verteilung
Samples
Histogramm
ChiSquareDistribution[2], 10000 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

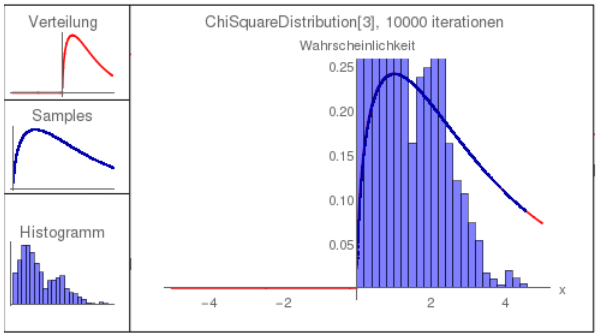

Verteilung
Samples
Histogramm
ChiSquareDistribution[2], 100000 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

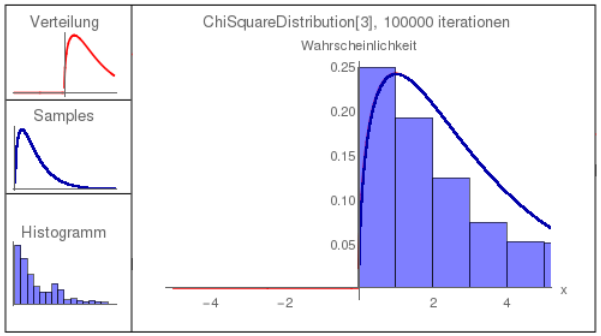

Verteilung
Samples
Histogramm
ChiSquareDistribution[2], 250000 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

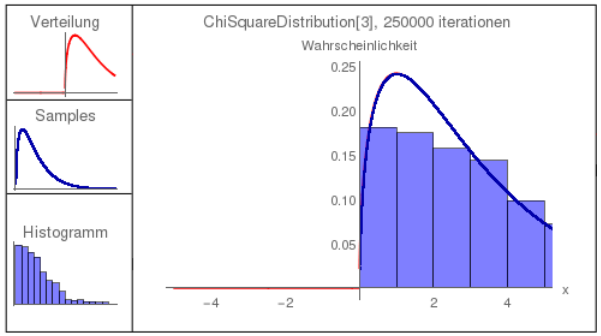

Verteilung
Samples
Histogramm
ChiSquareDistribution[3], 10 iterationen
Wahrscheinlichkeit
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

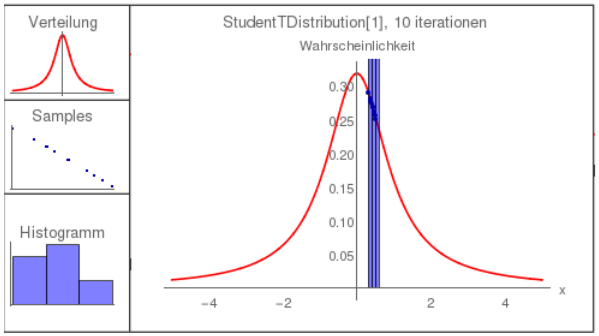

Verteilung
Samples
Histogramm
ChiSquareDistribution[3], 100 iterationen
Wahrscheinlichkeit
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

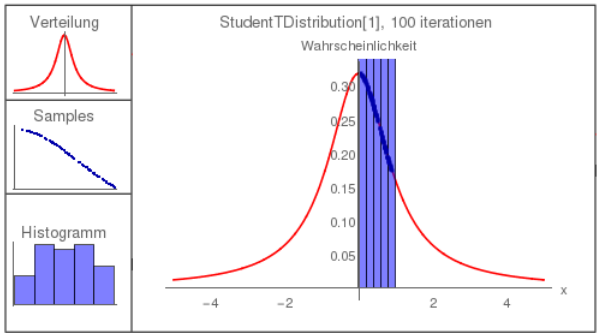

Verteilung
Samples
Histogramm
ChiSquareDistribution[3], 1000 iterationen
Wahrscheinlichkeit
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

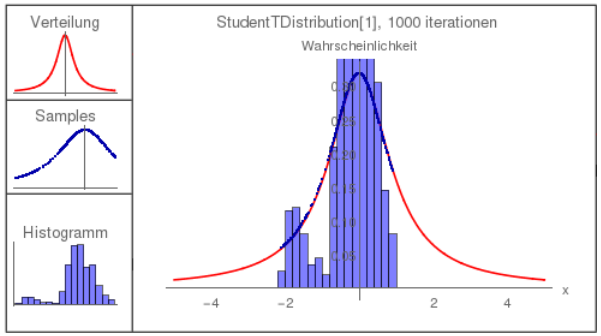

Verteilung
Samples
Histogramm
ChiSquareDistribution[3], 10000 iterationen
Wahrscheinlichkeit
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

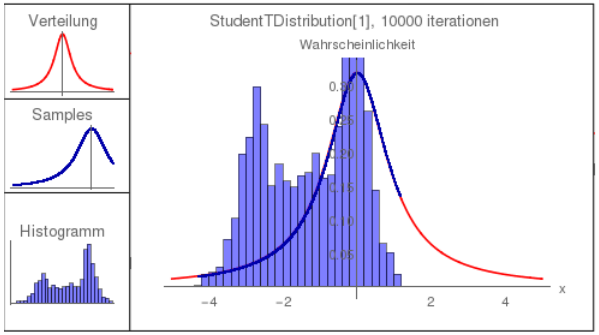

Verteilung
Samples
Histogramm
ChiSquareDistribution[3], 100000 iterationen
Wahrscheinlichkeit
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

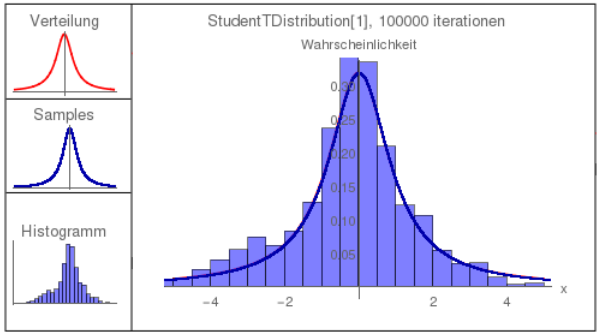

Verteilung
Samples
Histogramm
ChiSquareDistribution[3], 250000 iterationen
Wahrscheinlichkeit
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

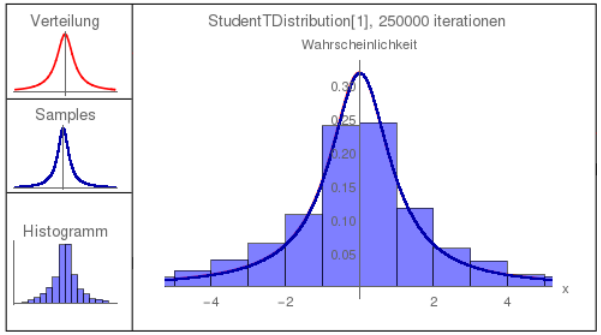

Verteilung
Samples
Histogramm
StudentTDistribution[1], 10 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

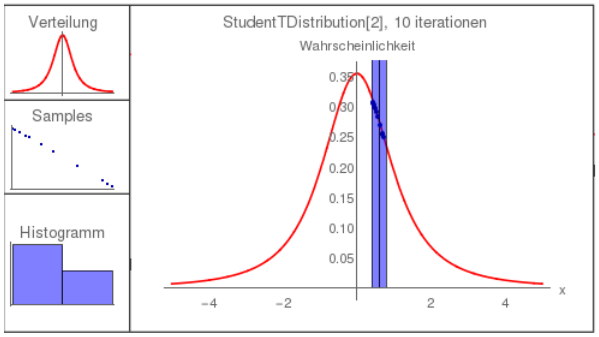

Verteilung
Samples
Histogramm
StudentTDistribution[1], 100 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

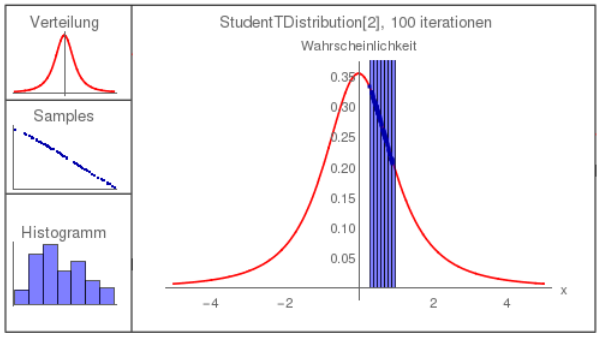

Verteilung
Samples
Histogramm
StudentTDistribution[1], 1000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

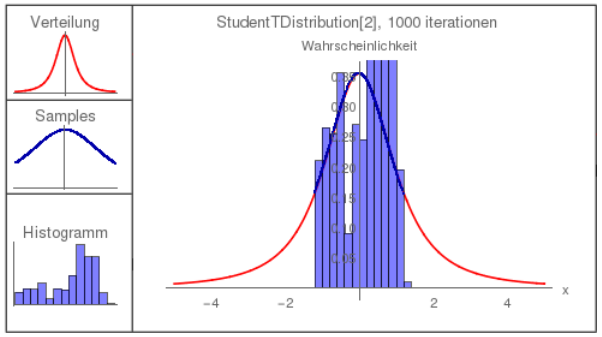

Verteilung
Samples
Histogramm
StudentTDistribution[1], 10000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

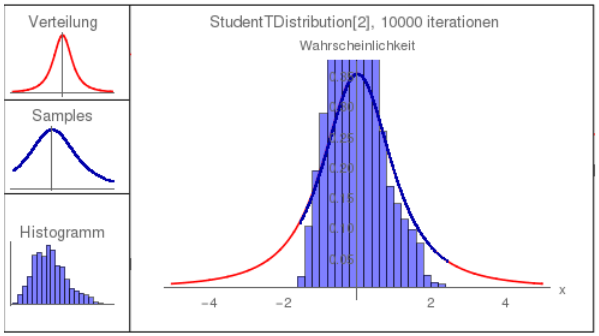

Verteilung
Samples
Histogramm
StudentTDistribution[1], 100000 iterationen
Wahrscheinlichkeit
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

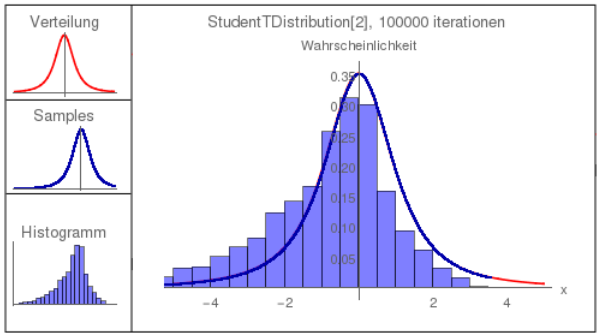

Verteilung
Samples
Histogramm
StudentTDistribution[1], 250000 iterationen
Wahrscheinlichkeit
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

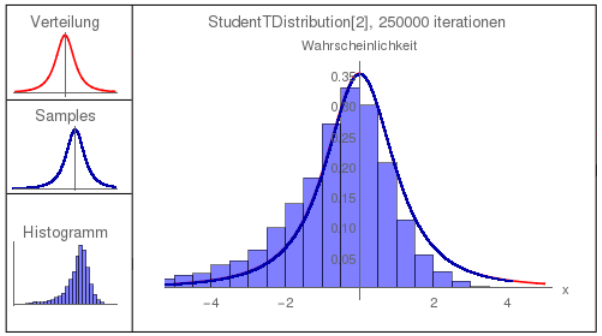

Verteilung
Samples
Histogramm
StudentTDistribution[2], 10 iterationen
Wahrscheinlichkeit
0.35
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

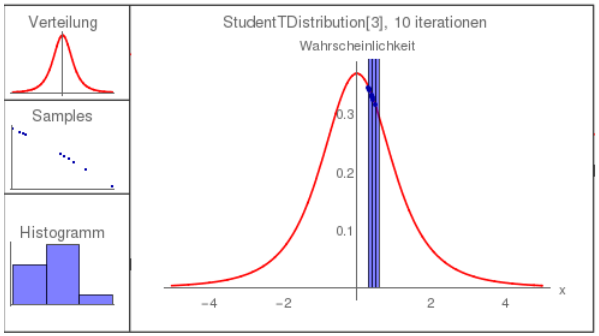

Verteilung
Samples
Histogramm
StudentTDistribution[2], 100 iterationen
Wahrscheinlichkeit
0.35
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

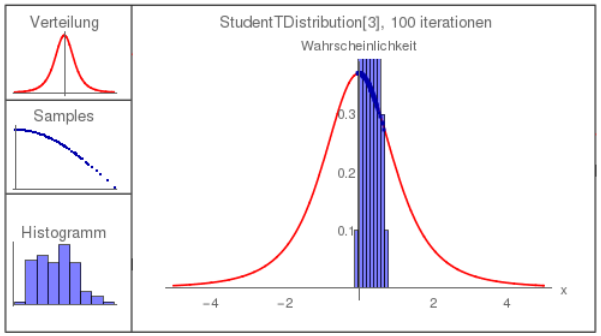

Verteilung
Samples
Histogramm
StudentTDistribution[2], 1000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

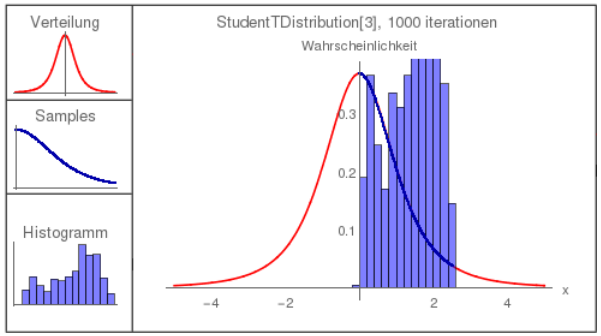

Verteilung
Samples
Histogramm
StudentTDistribution[2], 10000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

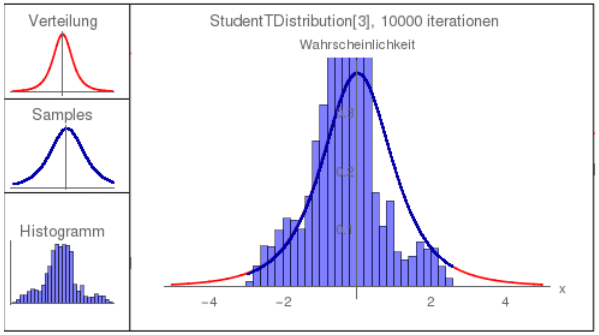

Verteilung
Samples
Histogramm
StudentTDistribution[2], 100000 iterationen
Wahrscheinlichkeit
0.35
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

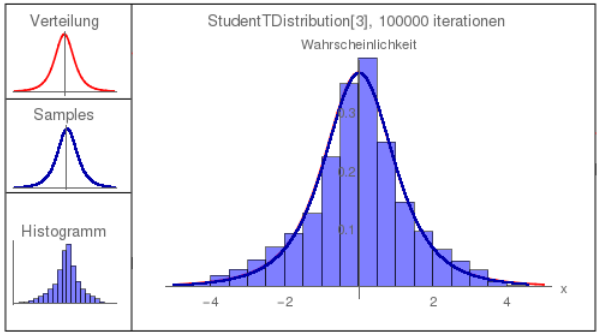

Verteilung
Samples
Histogramm
StudentTDistribution[2], 250000 iterationen
Wahrscheinlichkeit
0.35
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

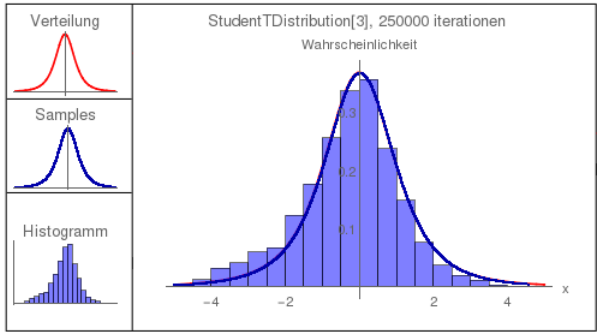

Verteilung
Samples
Histogramm
StudentTDistribution[3], 10 iterationen
Wahrscheinlichkeit
0.3
0.2
0.1
−4
−2
2
4
x

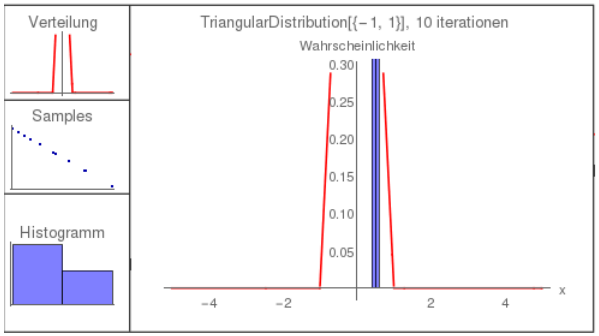

Verteilung
Samples
Histogramm
StudentTDistribution[3], 100 iterationen
Wahrscheinlichkeit
0.3
0.2
0.1
−4
−2
2
4
x

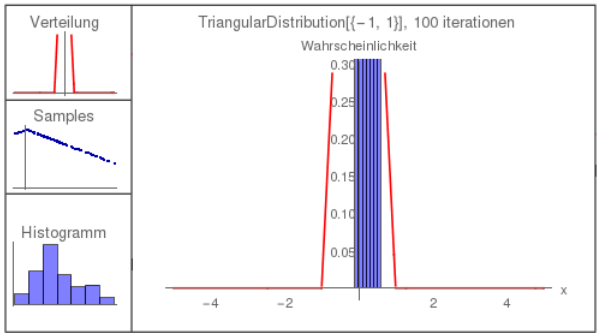

Verteilung
Samples
Histogramm
StudentTDistribution[3], 1000 iterationen
Wahrscheinlichkeit
0.3
0.2
0.1
−4
−2
2
4
x

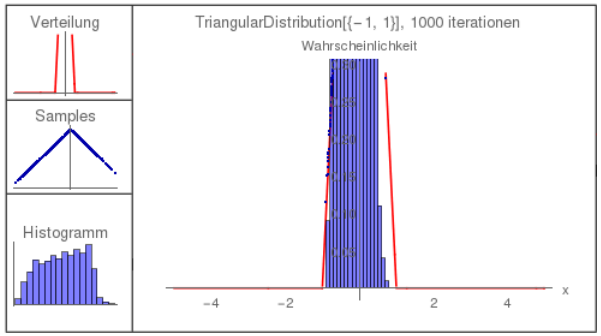

Verteilung
Samples
Histogramm
StudentTDistribution[3], 10000 iterationen
Wahrscheinlichkeit
0.3
0.2
0.1
−4
−2
2
4
x

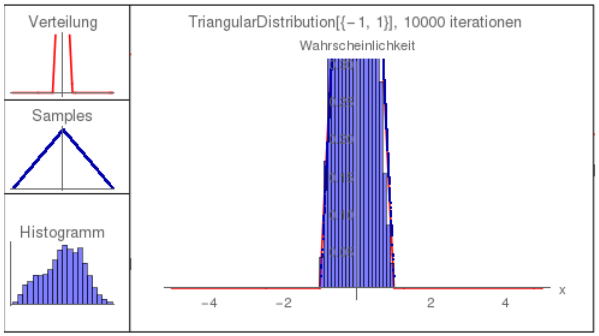

Verteilung
Samples
Histogramm
StudentTDistribution[3], 100000 iterationen
Wahrscheinlichkeit
0.3
0.2
0.1
−4
−2
2
4
x

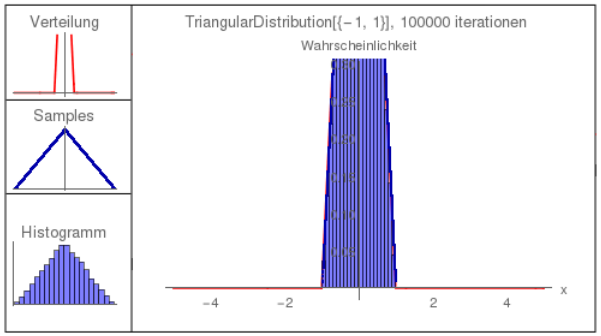

Verteilung
Samples
Histogramm
StudentTDistribution[3], 250000 iterationen
Wahrscheinlichkeit
0.3
0.2
0.1
−4
−2
2
4
x

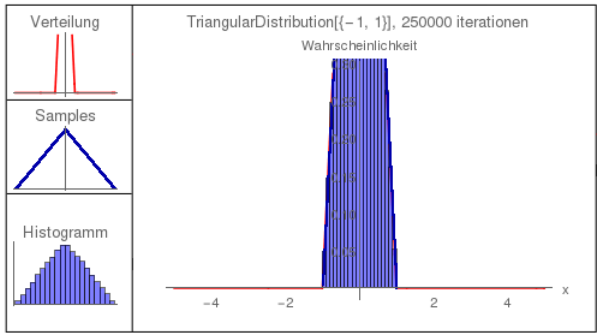

Verteilung
Samples
Histogramm
TriangularDistribution[{−1, 1}], 10 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

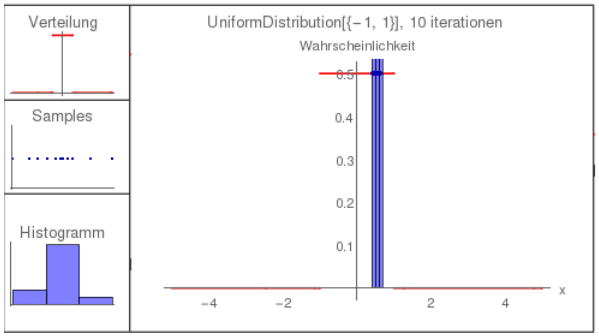

Verteilung
Samples
Histogramm
TriangularDistribution[{−1, 1}], 100 iterationen
Wahrscheinlichkeit
0.30
0.25
0.20
0.15
0.10
0.05
−4
−2
2
4
x

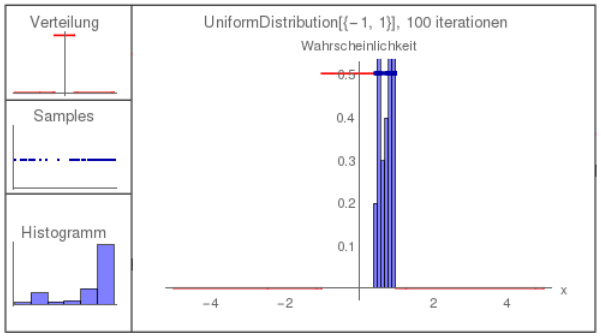

Verteilung
Samples
Histogramm
TriangularDistribution[{−1, 1}], 1000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

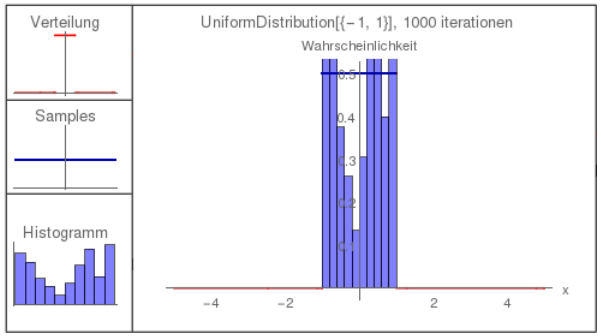

Verteilung
Samples
Histogramm
TriangularDistribution[{−1, 1}], 10000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

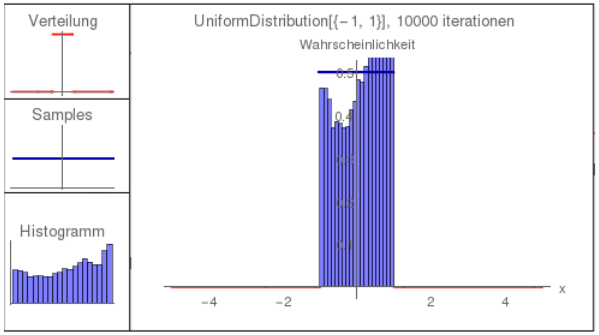

Verteilung
Samples
Histogramm
TriangularDistribution[{−1, 1}], 100000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

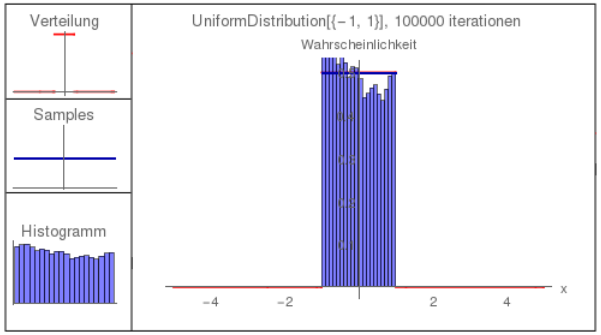

Verteilung
Samples
Histogramm
TriangularDistribution[{−1, 1}], 250000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

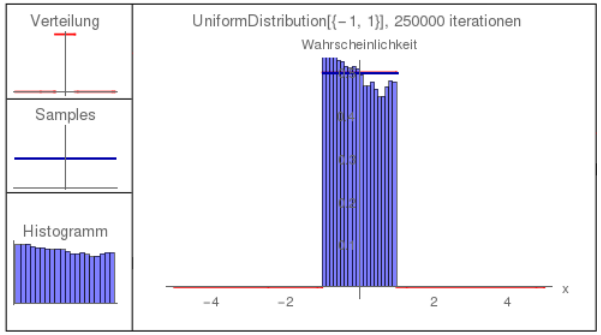

Verteilung
Samples
Histogramm
UniformDistribution[{−1, 1}], 10 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

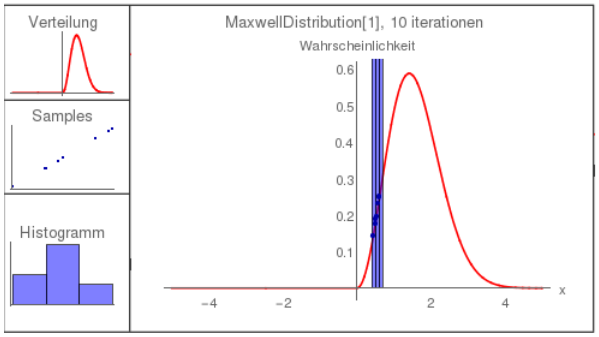

Verteilung
Samples
Histogramm
UniformDistribution[{−1, 1}], 100 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

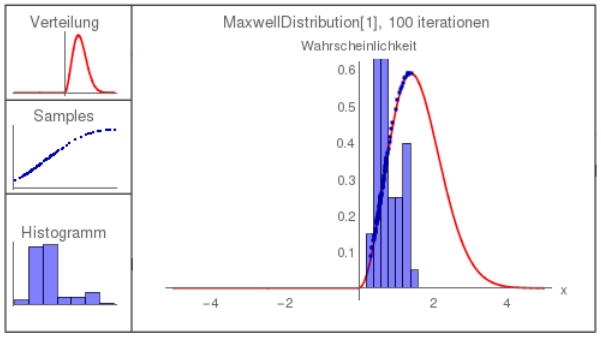

Verteilung
Samples
Histogramm
UniformDistribution[{−1, 1}], 1000 iterationen
Wahrscheinlichkeit
0.5
0.4
0.3
−4
−2
2
4
x

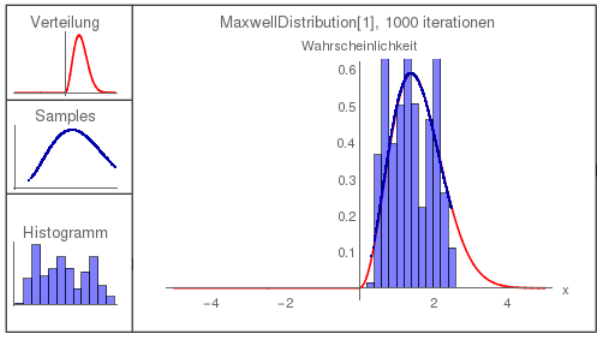

Verteilung
Samples
Histogramm
UniformDistribution[{−1, 1}], 10000 iterationen
Wahrscheinlichkeit
0.5
−4
−2
2
4
x

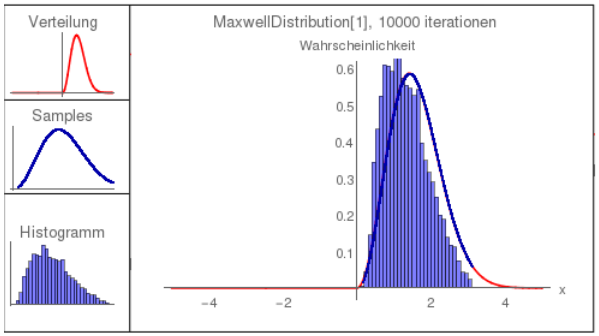

Verteilung
Samples
Histogramm
UniformDistribution[{−1, 1}], 100000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

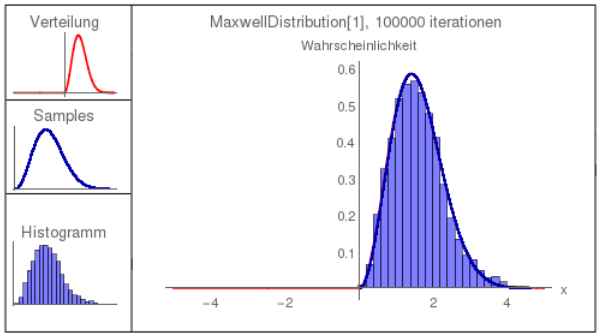

Verteilung
Samples
Histogramm
UniformDistribution[{−1, 1}], 250000 iterationen
Wahrscheinlichkeit
−4
−2
2
4
x

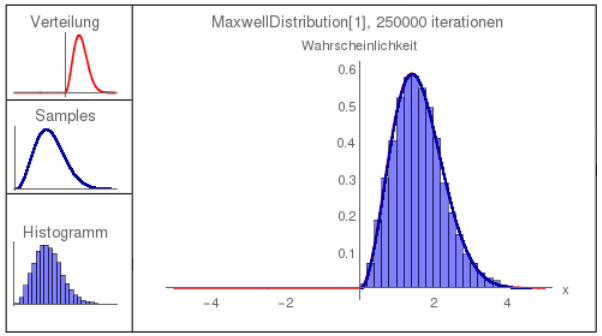

Verteilung
Samples
Histogramm
MaxwellDistribution[1], 10 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x


Verteilung
Samples
Histogramm
MaxwellDistribution[1], 100 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x


Verteilung
Samples
Histogramm
MaxwellDistribution[1], 1000 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x


Verteilung
Samples
Histogramm
MaxwellDistribution[1], 10000 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x


Verteilung
Samples
Histogramm
MaxwellDistribution[1], 100000 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x


Verteilung
Samples
Histogramm
MaxwellDistribution[1], 250000 iterationen
Wahrscheinlichkeit
0.6
0.5
0.4
0.3
0.2
0.1
−4
−2
2
4
x

D. Kryptoanalyse

```
ClearAll["Global`*"]

EncDec[m_,cipher_]:=StringReplace[m,KeyValueMap[#1->#2&,cipher]];

RandomCipher[chars_List]:=
Apply[
Association,
(#[[1]]->#[[2]])&/@
Partition[
Riffle[
chars,
RandomSample[chars,Length[chars]]
],
2
]
];

InverseCipher[x_]:=AssociationThread[Values[x],Keys[x]];

GenRef[charCount_Integer,chars_List]:=Module[
{
m=AssociationThread[Tuples[chars,2]->ConstantArray[1,Length[chars]^2]],
c=Length[chars]^2,
articleLinks=WikipediaData["Main Page","LinksList"]
},
(*Collect data*)
Print[ProgressIndicator[Dynamic[c/charCount]]];
While[c< charCount,
Block[
{article= RandomChoice[articleLinks]},
(*Get more links*)
articleLinks=Union[articleLinks,WikipediaData[article,"LinksList"]];
(*Analyze article*)
Map[
Function[
If[KeyExistsQ[m,#],
AssociateTo[m,#->1+m[#]];
c++;
];
If[c>=charCount,Break[]];
],
Partition[
Characters[ToLowerCase[WikipediaData[article]]],
2,
1
]
]
];
Return[m/c];
];

P[dist_,f_,c_]:=Times@@(
(
dist[{f[#[[1]]],f[#[[2]]]}]&/@
Partition[
Characters[c],
2,
1
]
)//DeleteMissing
);
```

```
63
64 Metropolis[n_,dist_,x0_,c_]:=
65 Module[{samples={},x=x0,x1=x0,i=0},
66 Print[ProgressIndicator[Dynamic[i/n]]];
67 While[i<n,
68 Block[
69 {
70 (*Get two random characters*)
71 s=RandomChoice[Keys[x]],
72 s1=RandomChoice[Keys[x]],
73 swap=0,
74 p=0,
75 p1=0
76 },
77 (*Swap in x1*)
78 swap=x1[s];
79 AssociateTo[x1,s->x1[s1]];
80 AssociateTo[x1,s1->swap];
81 (*Get probabilities*)
82 p=P[dist,x,c];
83 p1=P[dist,x1,c];
84 (*Print[StringForm[" `` <-> ``, p1/p: ``",s,s1,N[p1/p]]];*)
85 If[
86 RandomReal[]<Min[1,p1/p] ,
87 x=x1,
88 x1=x
89 ];
90 AppendTo[samples,x];
91 i++;
92 ]];
93 Return[samples];
94 ];
95
96 RateCipher[decCipher_,encCipher_,chars_]:=
97 Count[Partition[
98 Riffle[
99 chars,
100 EncDec[EncDec[chars,encCipher],decCipher]
101 ],
102 2],
103 x_/;x[[1]]==x[[2]]
104 ];
105
106 CiphersPlot[ciphers_,cipher_,chars_]:=Module[
107 {data,d,h,g1,g2,t1,t2},
108
109 data=RateCipher[#,cipher,chars]&/@ciphers;
110
111 d=ListPlot[data,
112 PlotLabel->"",
113 AxesLabel->{"Iteration","# Richtige"}
114 ];
115
116 h=Histogram[data,Automatic,
117 "PDF",
118 ChartStyle->Opacity[.5,Blue],
119 PlotLabel->"Histogramm",
120 AxesLabel->{"# Richtige","Relative HÃ¤ufigkeit"}
121 ];
122
123 g1 = GraphPlot[
124 KeyValueMap[#1->#2&, ciphers//Commonest //First],
125 DirectedEdges->True,
126 VertexLabeling->True,
127 PlotLabel->"HÃ¤ufigste cipher",
128 VertexRenderingFunction->({White,EdgeForm[Black],Disk[#,.3],Black,Text[#2,#1]}&)
129 ];
130
131 t1=StringTake[
132 EncDec[c,ciphers//Commonest//First],
133 200
134 ];
```

```
135
136 g2=GraphPlot[
137 KeyValueMap[#1->#2&, ciphers//Last],
138 DirectedEdges->True,
139 VertexLabeling->True,
140 PlotLabel->"Letzte cipher",
141 VertexRenderingFunction->({White,EdgeForm[Black],Disk[#,.3],Black,Text[#2,#1]}&)
142 ];
143
144 t2=StringTake[
145 EncDec[c,ciphers//Last],
146 200
147 ];
148
149 Return[{
150 d,
151 h,
152 Grid[{{g1},{t1}}],
153 Grid[{{g2},{t2}}]
154 }];
155 ];
156
157 "Set up char range"
158 chars=Characters[" .,!0123456789abcdefghijklmnopqrstuvwxyz"];
159 If[OddQ[Length[chars]],Print["Chars range must be even"];Abort[]];
160
161 "Set up message"
162 m=ToLowerCase[WikipediaData["Bern University of Applied Sciences"]];
163
164 "Set up cipher"
165 cipher =RandomCipher[chars];
166 c=EncDec[m, cipher];
167
168 "Gen ref"
169 ref=GenRef[5000000,chars];
170
171 "Cipher"
172 GraphPlot[
173 KeyValueMap[#1->#2&, cipher],
174 DirectedEdges->True,
175 VertexLabeling->True,
176 PlotLabel->"Cipher",
177 ImageSize->Full,
178 VertexRenderingFunction->({White,EdgeForm[Black],Disk[#,.3],Black,Style[Text[#2,#1],FontSize->Large]}&
179 ]
180
181 "Metropolis"
182 GraphicsGrid[MapThread[
183 CiphersPlot[
184 Metropolis[10000,ref,#,c] ,
185 cipher,
186 chars
187 ]&,
188 {{RandomCipher[chars],RandomCipher[chars],RandomCipher[chars],RandomCipher[chars]}}
189 ],
190 ImageSize->Full
191 ]
```

E. Simulated Annealing

E.1. Cipher

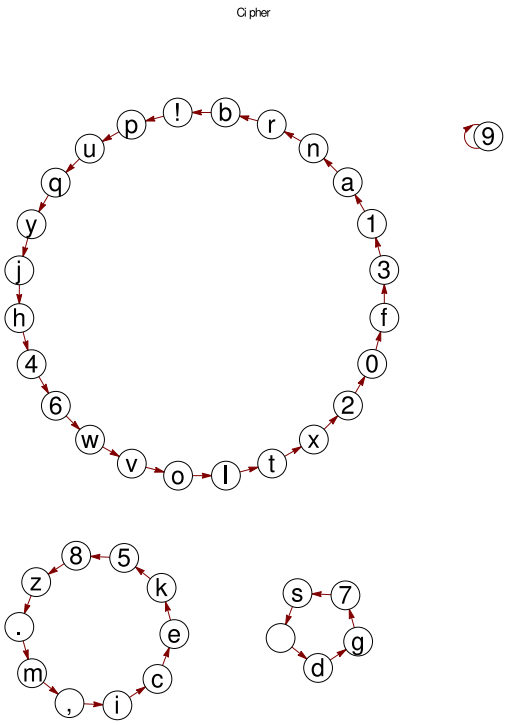


Abbildung E.1.: Verschlüsselungsfunktion, simulated annealing

E.2. Temperaturfunktion

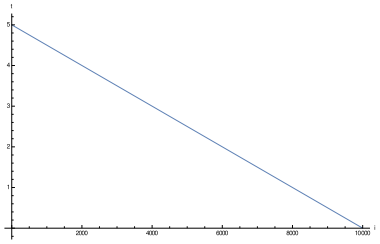


Abbildung E.2.: Temperaturzerfall

E.3. Resultat

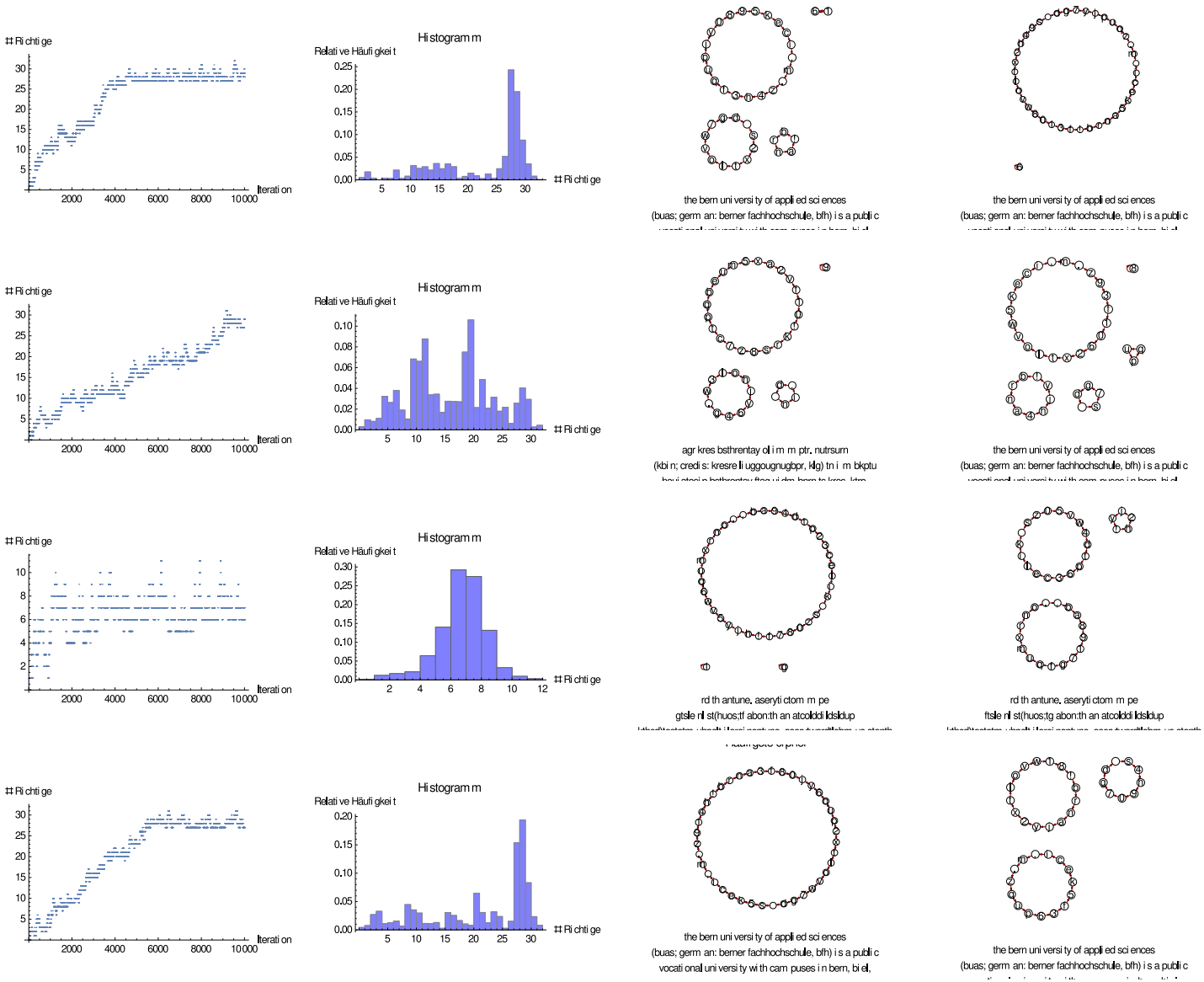


Abbildung E.3.: Entschlüsselungsfunktion, simulated annealing

E.4. Implementation

```
In[381]:= ClearAll["Global`*"]

EncDec[m_, cipher_]:=StringReplace[m, KeyValueMap[#1->#2&, cipher]];

RandomCipher[chars_List]:=
Apply[
Association,
(#[[1]]->#[[2]])&/@
Partition[
Riffle[
chars,
RandomSample[chars, Length[chars]]
],
2
]
];

InverseCipher[x_]:=AssociationThread[Values[x], Keys[x]];

GenRef[charCount_Integer, chars_List]:=Module[
{
m=AssociationThread[Tuples[chars,2]->ConstantArray[1, Length[chars]^2]],
c=Length[chars]^2,
articleLinks=WikipediaData["Main Page","LinksList"]
},
(*Collect data*)
Print[ProgressIndicator[Dynamic[c/charCount]]];
While[c< charCount,
Block[
{article= RandomChoice[articleLinks]},
(*Get more links*)
articleLinks=Union[articleLinks, WikipediaData[article,"LinksList"]];
(*Analyze article*)
Map[
Function[
If[KeyExistsQ[m,#],
AssociateTo[m,#->1+m[#]];
c++;
];
If[c>=charCount, Break[]];
],
Partition[
Characters[ToLowerCase[WikipediaData[article]]],
2,
1
]
]
];
Return[m/c];
];

P[dist_, f_, c_]:=Times@@(
(
dist[{f[#[[1]]], f[#[[2]]]}]&/@
Partition[
Characters[c],
2,
1
]
)//DeleteMissing
);

t0=5;
Temperature[i_, n_]:=Return[-t0/(n+1)*i+t0];

Metropolis[n_, dist_, x0_, c_]:=
Module[{samples={},x=x0, x1=x0, i=0},
Print[ProgressIndicator[Dynamic[i/n]]];
```

```
While[i<n,
Block[
{
(*Get two random characters*)
s=RandomChoice[Keys[x]],
s1=RandomChoice[Keys[x]],
swap=0,
p=0,
p1=0
},
(*Swap in x1*)
swap=x1[s];
AssociateTo[x1,s->x1[s1]];
AssociateTo[x1,s1->swap];
(*Get probabilities*)
p=P[dist,x,c];
p1=P[dist,x1,c];
If[
RandomReal[]<Min[1,(p1/p)^(1/Temperature[i,n])]  ,
x=x1,
x1=x
];
AppendTo[samples,x];
i++;
]];
Return[samples];
];

RateCipher[decCipher_,encCipher_,chars_]:=
Count[Partition[
Riffle[
chars,
EncDec[EncDec[chars,encCipher],decCipher]
],
2],
x_/;x[[1]]==x[[2]]
];

CiphersPlot[ciphers_,cipher_,chars_]:=Module[
{data,d,h,g1,g2,t1,t2},

data=RateCipher[#,cipher,chars]&/@ciphers;

d=ListPlot[data,
PlotLabel->"",
AxesLabel->{"Iteration","# Richtige"}
];

h=Histogram[data,Automatic,
"PDF",
ChartStyle->Opacity[.5,Blue],
PlotLabel->"Histogramm",
AxesLabel->{"# Richtige","Relative HÃ¤ufigkeit"}
];

g1 = GraphPlot[
KeyValueMap[#1->#2&, ciphers//Commonest //First],
DirectedEdges->True,
VertexLabeling->True,
PlotLabel->"HÃ¤ufigste cipher",
VertexRenderingFunction->({White,EdgeForm[Black],Disk[#,.3],Black,Text[#2,#1]}&)
];

t1=StringTake[
EncDec[c,ciphers//Commonest//First],
200
];

g2=GraphPlot[
KeyValueMap[#1->#2&, ciphers//Last],
DirectedEdges->True,
VertexLabeling->True,
```

```
142 PlotLabel->"Letzte cipher",
143 VertexRenderingFunction->({White,EdgeForm[Black],Disk[#,.3],Black,Text[#2,#1]}&)
144 ];
145
146 t2=StringTake[
147 EncDec[c,ciphers//Last],
148 200
149 ];
150
151 Return[{
152 d,
153 h,
154 Grid[{{g1},{t1}}],
155 Grid[{{g2},{t2}}]
156 }];
157 ];
158
159 "Set up char range"
160 chars=Characters[" .,!0123456789abcdefghijklmnopqrstuvwxyz"];
161 If[OddQ[Length[chars]],Print["Chars range must be even"];Abort[]];
162
163 "Set up message"
164 m=ToLowerCase[WikipediaData["Bern University of Applied Sciences"]];
165
166 "Set up cipher"
167 cipher =RandomCipher[chars];
168 c=EncDec[m,  cipher];
169
170 "Gen ref"
171 ref=GenRef[5000000,chars];
172
173 "Cipher"
174 GraphPlot[
175 KeyValueMap[#1->#2&, cipher],
176 DirectedEdges->True,
177 VertexLabeling->True,
178 PlotLabel->"Cipher",
179 ImageSize->Full,
180 VertexRenderingFunction->({White,EdgeForm[Black],Disk[#,.3],Black,Style[Text[#2,#1],FontSize->Large]}&
181 ]
182
183 "Metropolis"
184 iterations=10000;
185 repetitions=4;
186 GraphicsGrid[MapThread[
187 CiphersPlot[
188 Metropolis[iterations,ref,#,c] ,
189 cipher,
190 chars
191 ]&,
192 {Table[RandomCipher[chars],{repetitions}]}
193 ],
194 ImageSize->Full
195 ]
196
197 "Temperatur"
198 Plot[Temperature[i,iterations],
199 {i,0,iterations},
200 AxesLabel->{"i","t"}
201 ]
202 Plot[
203 {x^(1/5),x^(1/4),x^(1/2),x,x^2,x^4,x^5},
204 {x,0,1.1},
205 PlotLegends->{"(Subscript[Overscript[Ï€, ~], Ï†]/Subscript[Overscript[Ï€, ~], Subscript[X, t]])^(1/5)"
206 AxesLabel->{"Subscript[Overscript[Ï€, ~], Ï†]/Subscript[Overscript[Ï€, ~], Subscript[X, t]]"},
207 Epilog->{Text["Ï„",{0.5,0.5},{0,0}],Arrow[{{0.1,1},{0.9,0.1}}]}
208 ]
```