Freie Universität Berlin
Institut für Informatik
Takustraße 9
14195 Berlin

SS 2010

Proseminar Technische Informatik

Random-Number Generation for Testbeds and Simulation

Phillip Berndt

14. Juni 2010

Betreut durch Philipp Reinecke und Matthias Wählisch

Inhaltsverzeichnis

I. Zufall und Simulation

Eine wichtige Aufgabe jeder Naturwissenschaft ist die Analyse komplexer Systeme. Da die notwendigen Versuchsreihen oft mit viel Aufwand und hohen Kosten verbunden sind, bedient man sich gerne der Simulation: Anstatt am „echten“ System zu arbeiten, erstellt man ein mathematisches Modell und untersucht sein Verhalten am Computer.

Dabei ist die Simulation umso besser, je ausgereifter das mathematische Modell ist. Auf der anderen Seite ist mit dem Erstellen eines Modells selbst Analyseaufwand verbunden, sodass man zwischen exakter Abbildung und vertretbarem Aufwand abwägen muss. Durch den Verzicht auf Genauigkeit führt man in die Simulation ein Zufallselement ein, dass man kontrollieren möchte: Als Beispiel betrachte die Simulation eines Stromnetzes, von dem man weiß, dass es in 20 Prozent aller Fälle ausfällt, aber nicht weiß, warum. Anstatt die Ursache genau zu erforschen, kann man das System einfach so modellieren, dass es ebenfalls in 20 Prozent aller Fälle einen Ausfall simuliert.

Selbst mit hohem Aufwand kann man auf die Simulation von Zufall oft nicht verzichten. Als Extrembeispiel, in dem bereits die Theorie ein exaktes Modell verbietet, sei die Quantenmechanik angeführt: Werner Heisenberg propagierte 1927 seine bekannte Unschärferelation, nach der im Mikrokosmos keine exakte Naturbeobachtung *möglich* ist. Physiker behelfen sich an dieser Stelle, indem sie die Vorgänge statistisch analysieren – und bei Simulationen *Zufall* verwenden.

Es muss also eine Möglichkeit geschaffen werden, Zufall zu simulieren. Wie genau das geschieht, wird im Folgenden beschrieben.

II. Mathematische Grundlagen

Um Zufall selbst zu modellieren, bedient sich der Mathematiker der Stochastik (von $\sigma\tau\acute{o}\chi o\varsigma$, griechisch für Mutmaßung [4, S. 1]). Diese Disziplin beinhaltet die *Wahrscheinlichkeitstheorie*, die sich der Beschreibung zufälliger Vorgänge widmet.

Während die Theorie an dieser Stelle Ereignismengen, σ-Algebren und Wahrscheinlichkeitsräume einführt, wollen wir uns auf die für unsere Zwecke wichtigen Definitionen beschränken und diese zum Teil vereinfacht darstellen. Für ein genaueres Studium der Wahrscheinlichkeitstheorie verweisen wir auf [4]. Diesem Buch folgen wir in diesem Abschnitt.

Wir beginnen damit, die Abbildung, die zwischen den möglichen Ausgängen eines Experimentes und ihrer Wahrscheinlichkeit abbildet, formal zu definieren:

Definition II.1. *Sei Ω die abzählbare Menge von möglichen Ergebnissen eines Experimentes. Eine Funktion*

$$\begin{aligned} P : 2^{\Omega} &\to [0,1] \\ X &\mapsto P(X) \end{aligned}$$

heißt Wahrscheinlichkeitsmaß*, wenn gilt:*

1. $P(\Omega) = 1$
2. $A, B \in 2^\Omega, A \cap B = \emptyset \implies P(A \cup B) = P(A) + P(B)$

Ein Wahrscheinlichkeitsmaß weist jedem möglichen Ausgang eines Experimentes eine Zahl zwischen 0 und 1 zu, die als Wahrscheinlichkeit für das eintreten dieses Ausgangs zu verstehen ist.

Die Abzählbarkeit von Ω ist in dieser Definition wesentlich, aber nicht wünschenswert: Sie verbietet uns, reellen Größen wie Orte und Zeiten ein Wahrscheinlichkeitsmaß zuzuordnen. Der Übergang zu *kontinuierlichen Wahrscheinlichkeitsmaßen* ist formal sehr kompliziert, anschaulich jedoch leicht zu fassen:

Anstatt jedem Punkt in $\mathbb{R}$ eine Wahrscheinlichkeit zuzuordnen, diskretisiert man und betrachtet nur die Wahrscheinlichkeit, ein Messergebnis im Intervall $[x, x + \Delta x]$ zu erhalten. Offenbar ist Ω nun abzählbar. Mit $\Delta x \to 0$ geht man zu einem infitesimalen Intervall $[x, x + dx]$ über. Das resultierende Wahrscheinlichkeitsmaß wird auch *Dichtefunktion* genannt. Da ihm infinitesimal kleine Intervalle zugrunde liegen, muss man die Dichtefunktion über I integrieren, um aus ihr zurück zu einer Wahrscheinlichkeit für ein Messergebnis in einem Intervall I zu gelangen.

Es kann vorkommen, dass einem bereits Messdaten aus einem Experiment zur Verfügung stehen, die mehr Information enthalten, als man braucht: Will man z.B. wissen, ob ein Teilchen sich innerhalb eines Kastens aufhält, benötigt man nicht seinen genauen Standort – kann aber aus letzterem die gewünschte Größe bestimmen. Im mathematischen Kontext hilft einem das Konzept der *Zufallsvariable* dabei, die unbenötigten Informationen zu „vergessen“ und danach nur noch auf einem kleineren Ω' rechnen. Das kann Rechnungen zum Teil erheblich vereinfachen:

Definition II.2. *Eine Funktion*

$$\begin{aligned} \mathcal{F} : (2^\Omega, P) &\to (2^{\Omega'}, P') \\ \Omega \supset X &\mapsto \mathcal{F}(X) \end{aligned}$$

heißt Zufallsvariable*, falls gilt:*

$$A \in 2^{\Omega'} \implies \mathcal{F}^{-1} A \in 2^\Omega$$

Zur Veranschaulichung betrache den Wurf eines Würfels. Als Zufallsvariable betrachten wir, ob der Würfel eine sechs gewürfelt hat. Dann gilt mit der Nomenklatur von oben:

1. $\Omega = \{1, 2, 3, 4, 5, 6\}$
2. $P(X) = \frac{1}{6} \cdot |X|$
3. $\Omega' = \{0, 1\}$

4. $P'(X) = \begin{cases} 1 & X = \{0,1\} \\ \frac{1}{6} & X = \{1\} \\ \frac{5}{6} & X = \{0\} \\ 0 & X = \emptyset \end{cases}$

5. $\mathcal{F}(X) = \begin{cases} \{1,0\} & 6 \in X \wedge X \setminus \{6\} \neq \emptyset \\ \{1\} & X = \{6\} \\ \emptyset & X = \emptyset \\ \{0\} & 6 \notin X \wedge X \neq \emptyset \end{cases}$

Anstatt Teilmengen von Ω explizit anzugeben, schreibt man oft

$$[X\odot] := \{X \in 2^\Omega | X\odot\}$$

$\odot$ ist dabei so zu ersetzen, dass $X\odot$ zu einem Wahrheitswert auflösbar ist. Dabei werden Mengen-Klammern häufig weggelassen, wenn X einelementig ist. z.B. verwenden wir $[X = 1]$ um für den Würfel „eine Eins wurde geworfen“ zu beschreiben.

Die auch außerhalb der Wahrscheinlichkeitstheorie bekannten Begriffe des *Mittelwert*es und der *Abweichung* werden in unserem Kontext zu:

Definition II.3. *μ ist der* Erwartungswert *einer Verteilung, je nach Fall (diskret oder kontinuierlich) definiert über*

$$\mu = \int_{\mathbb{R}} x f(x) dx = \sum_{n=0}^{\infty} n P(\{n\}) \ .$$

Er gibt an, welcher Wert im Mittel über viele Wiederholungen des Experimentes angenommen wird. σ ist die Standardabweichung

$$\sigma = \left(\int_{\mathbb{R}} x f(x)^2 dx - \left(\int_{\mathbb{R}} x f(x) \right)^2 \right)^{\frac{1}{2}} = \left(\sum_{n=0}^{\infty} n P(\{n\})^2 - \left(\sum_{n=0}^{\infty} n P(\{n\}) \right)^2 \right)^{\frac{1}{2}} ,$$

die Auskunft über die zu erwartende Abweichung vom Mittelwert gibt.

II.1. Zufallsverteilungen

Statt an Einzelexperimenten ist man häufig an *Verteilungen* interessiert. Bei einer Verteilung betrachtet man statt der Wahrscheinlichkeit $P([X = x])$ die kumulierte Wahrscheinlichkeit $P([X \leq x])$. Im kontinuierlichen Fall ist die Verteilungsfunktion gegeben durch

$$V(x) := \int_{-\infty}^{x} f(\xi) d\xi$$

Wir wollen den Begriff am Beispiel der *Gleichverteilung* verdeutlichen.

II.1.1. Gleichverteilung

Man spricht von einer Gleichverteilung, wenn alle möglichen Ereignisse $x \in \Omega$ gleichwahrscheinlich sind, also $P(\{x\})$ für alle $x \in \Omega$ denselben Wert annimmt. Diese Art der Verteilung ist uns bereits vom Würfel bekannt, bei dem die Wahrscheinlichkeit bei jeder Seite $\frac{1}{6}$ beträgt.

Wir wollen den allgemeinen Fall kontinuierlich einführen. Betrache für reelle $a < b$ die Menge $\Omega = [a, b)$. Dann gilt:

1. $\rho(x) = \begin{cases} \frac{1}{b-a} & x \in [a, b) \\ 0 & \text{sonst} \end{cases}$
2. $P(X) := \int_X \rho(x)$
3. $P([X \leq m]) = \int_{-\infty}^{m} \rho(x)dx$
4. $\sigma = \frac{1}{6}\sqrt{3}(b-a)$
5. $\mu = \frac{b+a}{2}$

Ein Beispiel für einen Fall, in dem man an einer solchen Verteilung interessiert sein kann, ist das zufällige drehen einer Rouletteschеibe, wobei man den Winkel in dem Intervall $[0, 2\pi)$ misst, oder aber auch die Messung von Zeitpunkten.

Wir werden später sehen, dass in Zufallszahlengeneratoren meist diese Verteilung erzeugen und dann mithilfe von Zufallsvariablen aus der Gleichverteilung andere Verteilungen errechnen.

Um häufige Wiederholungen von Experimenten mit nur zwei möglichen Ausgängen simulieren zu können, benutzt man die Binomialverteilung. Sie wird vor allem verwendet, um zufällige Abweichungen von bekannten Größen, wie Werkstückvorgaben oder Messdaten, zu beschreiben.

II.1.2. Binomialverteilung

Wir betrachten $\Omega = \{1, 0\}$ und eine Wahrscheinlichkeitsverteilung, die $\{1\}$ die Wahrscheinlichkeit $p \in [0, 1]$ zuweist. Einen solchen Versuchsaufbau nennt man *Bernoulli-experiment* und wir sind ihm schon im Beispiel des Würfels begegnet. Statt diesem Experiment interessiert uns nun aber, wie oft $\{1\}$ auftaucht, wenn wir das Experiment unabhängig von den vorherigen Versuchen n mal wiederholen.

Die resultierende Wahrscheinlichkeitsverteilung ist offensichtlich für Werte in $[0, n]$ definiert. Sie nennt sich *Binomialverteilung*. Für sie gilt:

1. $P([X = k]) := \binom{n}{k} p^k (1-p)^{n-k}$
2. $P([X \leq k]) := \sum_{j=0}^{k} \binom{n}{j} p^j (1-p)^{n-j}$
3. $\sigma = \sqrt{np(1-p)}$

4. $\mu = np$

Mit der Binomialverteilung modelliert man alle Vorgänge, die sich auf einzelne „Ja-/Nein“-Experimente herunterbrechen lassen. Ein einfaches Beispiel dafür ist die Frage, wie hoch die Wahrscheinlichkeit ist, dass von n Personen k an einem 12. April Geburtstag haben.

II.1.3. Normalverteilung

Für große Werte von n kann die Berechnung der Binomialverteilung am PC sehr langwierig werden. Man approximiert die Binomialverteilung daher durch die kontinuierliche *Normalverteilung*:

$$\begin{aligned} f(x) := \mathbb{R} &\to \mathbb{R}^+ \\ x &\mapsto \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \end{aligned}$$

Um die Binomialverteilung zu approximieren, setzt man für Erwartungswert und Standardabweichung die entsprechenden Werte der Verteilung ein. Die Approximationseigenschaft wird durch den zentralen Grenzwertsatz der Wahrscheinlichkeitstheorie ([4, S. 133]) gesichert, nach dem die Summe von n unabhängigen, gleich verteilten Zufallsvariablen (in diesem Fall der einzelnen, der Binomialverteilung zugrundeliegenden, Experimente), für $n \to \infty$ wie ein Normalverteilung verteilt ist.

Erwartungswert und Standardabweichung entsprechen bei dieser Verteilung den eingesetzten Parametern.

II.1.4. Exponentialverteilung

Um die Lebensdauer z.B. von radioaktiven Materialien, aber auch von Produkten, zu modellieren, verwendet man die *Exponentialverteilung*

$$\begin{aligned} f(x) := \mathbb{R}^+ &\to \mathbb{R}^+ \\ x &\mapsto \lambda e^{-\lambda x} \end{aligned}$$

λ ist dabei ein frei wählbarer, positiver Parameter, der die Geschwindigkeit des Abfalls der Wahrscheinlichkeit bestimmt. Für die Exponentialverteilung gilt:

1. $P([X \leq x]) = 1 - e^{-\lambda x}$
2. $\mu = \frac{1}{\lambda}$
3. $\sigma = \frac{1}{\lambda}$

III. Pseudozufall

In Simulationen ist es nicht immer wünschenswert, echte Zufallszahlen zu verwenden. Deren zufällige Natur macht es unmöglich, eine Simulation exakt zu wiederholen, was zum Auffinden von Modellfehlern notwendig ist. Außerdem steht nicht in jedem Fall eine Quelle für Zufallszahlen zu Verfügung. Es muss also ein deterministischer Algorithmus entwickelt werden, der möglichst reproduzierbar und gleichzeitig möglichst unvorhersagbar Zufallszahlen generiert. In diesem Abschnitt werden Anforderungen an einen solchen *Pseudo-Random-Numbers-Generator* (PRNG) zusammengestellt und exemplarisch ein Algorithmus vorgeführt. Wir wollen dabei zunächst auf gleichverteilte Zufallszahlen eingehen.

III.1. Anforderungen an einen PRNG

Wir wollen reproduzierbare Zufallszahlen erzeugen. Um Reproduzierbarkeit zu gewährleisten, muss der verwendete Algorithmus einen Zustand verwalten. Wir gehen hier davon aus, dass es nur abzählbar viele Zustände gibt. Da Computer nur über begrenzt viel Speicher verfügen, ist das sicherlich gerechtfertigt. Über einen vom Anwender vorgegebenen Startzustand (auch *Seed*) wird die erzeugte Zufallszahlenreihe kontrolliert. Diese Überlegung führt zu einer möglichen Darstellung eines Algorithmus über die beiden Funktionen

$$\zeta : \Lambda \to \Lambda,\ \Lambda \subset \mathbb{N}$$
$$\psi : \Lambda \to \Omega$$

Dabei ist ζ die Übergangsfunktion zwischen den Zuständen in Λ und ψ eine Funktion, die einem Zustand eine Zufallszahl aus der Menge Ω zuordnet. (Siehe [1, S. 25])

Den Algorithmus können wir dann schreiben als

```
1 def random():
2         static random_seed = 1; # Oder eine andere Zahl
3         random_seed = zeta(random_seed);
4         return psi(random_seed);
```

Es ergeben sich auf natürliche Weise Anforderungen an beide Funktionen:

1. ζ sollte injektiv sein, denn wenn verschiedene Wege auf denselben Zustand führen, werden ab dort die erzeugten Zufallszahlen übereinstimmen.
2. $\psi(\zeta(\Lambda))$ muss notwendigerweise ganz Ω sein, denn sonst sind die erzeugten Zufallszahlen nicht gleichverteilt.
3. Falls ζ periodisch ist (ist Λ endlich, so ist das immer der Fall), sollte die Periode möglichst groß sein, damit sich die Zufallszahlen nicht zu schnell wiederholen. Auf jeden Fall sollte die Periode größer sein als die Menge benötigter Zufallszahlen.

4. Die Berechnung von ζ und ψ sollte möglichst einfach sein, damit der Algorithmus am Computer schnell ausgewertet werden kann.

Außerdem sollte natürlich die Beziehung zwischen $\psi(\zeta(n))$ und $\psi((\zeta)^2(n))$ möglichst „zufällig“ aussehen. Wie genau die Zufälligkeit gemessen werden kann, wird im Folgenden erläutert.

III.2. Das Messen von Zufälligkeit

Laut [1, S. 25], [5, S. 4] ist ein gutes Kriterium für die Bestimmung der Qualität eines PRNG, dass für beliebige Sequenzlängen $n \in \mathbb{N}$ die sequenziell durch n generierte Zufallszahlen gebildeten Vektoren gleichverteilt im n-dimensionalen Einheitswürfel $[0,1]^n$ sind.

Da die Zustandsmenge des Zufallszahlengenerators in der Regel durch die Hardwarevorgaben beschränkt ist und der Generator daher eine endliche Periodenlänge haben muss, kann das Kriterium formell nicht erfüllt werden. Daher misst man nur, ob die t-Vektoren gleichmäßig verteilt sind. Eine Methode zum Nachvollziehen, ob eine gegebene Menge Zufallszahlen gleichverteilt ist, ist der χ^2-Test. Da eine Erläuterung den Rahmen dieser Arbeit sprengen würde, verweisen wir an dieser Stelle wiederum auf [4].

Je nach Anwendungsgebiet kommen weitere Qualitätsanforderungen hinzu. In der Kryptographie ist besonders die *Unvorhersagbarkeit* wichtig, die wir hier jedoch ebenfalls nicht näher betrachten wollen.

III.2.1. Softwarebasierte Qualitätstests

In den neunziger Jahren haben sich softwarebasierte Tests als unterstützende Werkzeuge bei der Analyse von Zufallszahlengeneratoren herausgebildet. Nennenswert sind hier DIEHARD ([6]) sowie TestU01 ([5]). Beide führen eine Reihe gegebener statistischer Tests für einen Eingabealgorithmus aus. Der Benutzer kann die Abweichung von den mathematisch bestimmten Idealergebnissen der Tests auswerten, um Aussagen zur Qualität des Algorithmus zu treffen.

III.3. Methoden zur Erzeugung von Zufallszahlen

Zur Erzeugung gleichverteilter Zufallszahlen stehen mehrere Techniken zur Verfügung. Im Rahmen dieser Arbeit werden die grundlegendsten kurz umrissen und ein vollständiger Algorithmus näher analysiert werden. Dabei folgen wir [1].

Lineare Kongruenzgeneratoren sind von der Form

$$\begin{aligned} \psi(x) &:= \frac{\zeta(x)}{M} \\ \zeta(x) &:= (a \cdot x + c) \mod M \end{aligned}$$

Bei passender Wahl der Konstanten a und c lässt sich eine Periode von M erziehlen. (Für einen Beweis und nähere Details siehe [1, S. 27].) Wird statt eines Polynoms erster Ordnung für die Definition von ζ eine beliebige Funktion modulo M gewählt, spricht man von *Nichtlinearen Kongruenzgeneratoren*. Lineare Generatoren erzeugen allerdings eine Gitterstruktur bei einer Abbildung in den $[0,1]^n$, die unerwünscht sein kann. ([1, S. 30])

Schieberegister-Generatoren verwenden zur Zustandsberechnung eine binären Folge, die die Rekusion $b_n = b_{n-p} + b_{n-q} \mod 2$ erfüllt. Die Zufallszahlen werden erzeugt, indem jeweils L Glieder dieser Folge zusammen als Dualbruch aufgefasst werden. Vorteil der Basis 2 ist leichte Berechenbarkeit, da sich die Operationen im Wesentlichen auf XOR reduzieren. Verwendet man hier eine andere Basis, spricht man von einem *Lagged Fibonacci*-Generator.

Multiplikation mit Übertrag verwendet als inneren Zustand die Folge c_i. Man definiert

$$c_i = \left[\frac{ax_{i-1} + c_{i-1}}{M}\right]$$
$$x_i = (ax_i + c_{i-1} \mod M$$

Die x_i sind dabei die generierten Zufallszahlen. Auf diese Weise erzeugte Zufallszahlen haben bei geschickter Wahl der Parameter eine Periodenlänge von $\frac{aM-2}{2}$. Die Berechnung am Computer ist bei dieser Methode sehr einfach.

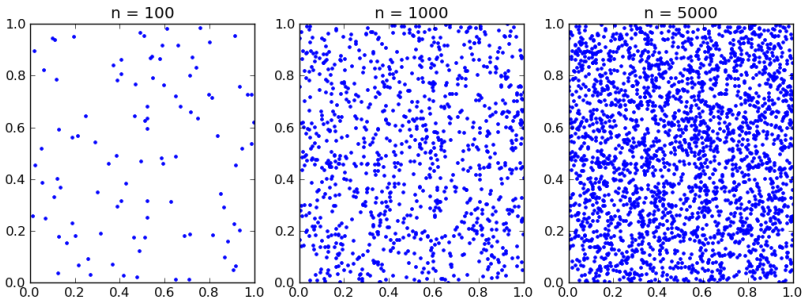
Kombinationen dieser grundlegenden Typen führen zu höheren Periodenlängen und besserer Verteilung der Zufallszahlen.

III.4. Ein PRNG-Algorithmus

Wir werden nun beispielhaft einen Algorithmus näher analysieren. Die Wahl ist dabei auf den Algorithmus von Wichmann-Hill ([3]) gefallen, der in der Standardbibliothek der Programmiersprache Python als Standardquelle für gleichverteilte Zufallszahlen verwendet wird. Er verwendet drei gekoppelte, lineare Kongruenzgeneratoren und ist damit einfach nachvollziehbar. In der Datei `random.py` [10] werden Zufallszahlen so erzeugt: (Kommentare und für unsere Zwecke überflüssige Zeilen wurden hier gekürzt)

```
1  class WichmannHill(Random):
2      def seed(self, a):
3          a, x = divmod(a, 30268)
4          a, y = divmod(a, 30306)
5          a, z = divmod(a, 30322)
6          self._seed = int(x)+1, int(y)+1, int(z)+1
7      def random(self):
8          x, y, z = self._seed
9          x = (171 * x) % 30269
10         y = (172 * y) % 30307
```

Abbildung 1: Verteilung nach Wichmann-Hill im $[0,1]^2$




```
11          z = (170 * z) % 30323
12          self._seed = x, y, z
13          return (x/30269.0 + y/30307.0 + z/30323.0) % 1.0
```

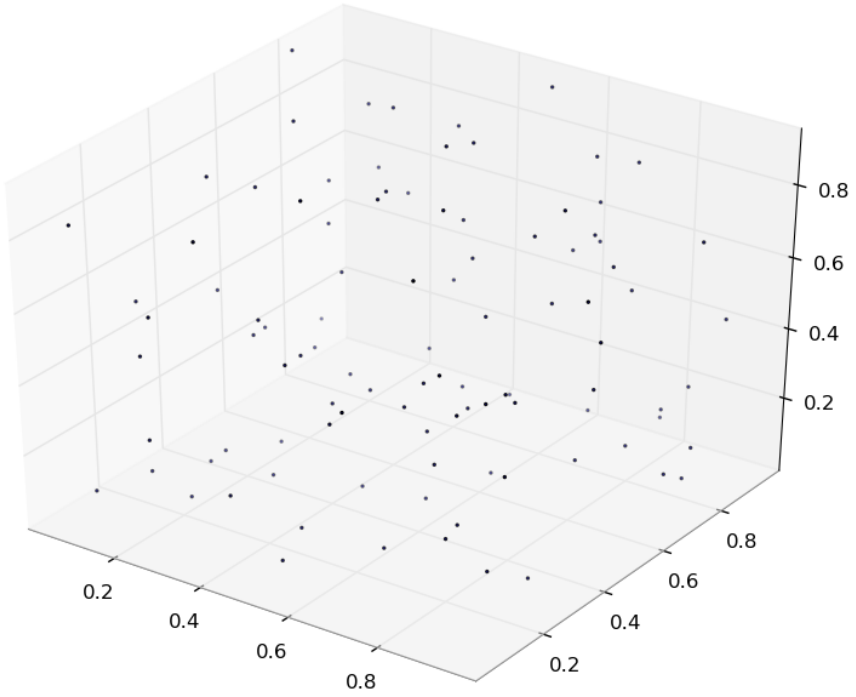
Die Initialisierung erfolgt über die Entropiequelle des Systems oder eine Berechnung anhand der Systemzeit.

Auf den ersten Blick zu erkennen ist, dass der Algorithmus seinen Zustand in einem Tripel kodiert (was aufgrund der Abzählbarkeit von $\mathbb{N}$ nach wie vor eine Darstellung als ζ zulässt), dass es sich um eine Kombination linearer Kongruenzgeneratoren handelt und dass $\psi(x,y,z) = \frac{x}{30269} + \frac{y}{30307} + \frac{z}{30323} \mod 1$. Die Periodenlängen 30269, 30307 und 30323 sind allesamt prim, sodass die Division mit Rest in der `seed`-Methode bei gegebenem `a` unterschiedliche Werte für `x`, `y` und `z` erzeugt. Die Wahl der Multiplikatoren a sichert zudem, dass jede der Zustandsvariablen eine maximale Periode durchläuft. (Siehe [1, S. 27]) Die Eigenschaft des Algorithmus, die Einheitswürfel möglichst gut auszufüllen, konnte im Rahmen dieser Arbeit nur empirisch überprüft werden. Siehe dazu Abbildung 1 und Abbildung 2.

Es stellt sich die Frage, warum drei lineare Generatoren verwendet werden. Künsch erläutert in Lemma 2.1 [1, S. 32], dass die Kombination der Generatoren die Periode sicher nicht verschlechtert, sondern im besten Fall sogar auf das kleinste gemeinsame Vielfache der Perioden erhöht. Es ist davon auszugehen, dass die Wahl hier entsprechend getroffen wurde, dies wurde aber nicht nachvollzogen.

Abschließend betrachten wir noch die aus Sicht der Informatik interessanten Eigenschaften des Algorithmus: Laut eines Kommentars in der Python-Implementierung wird der von `random` zurückgegebene Wert auf Plattformen mit IEEE-754 Double Arithmetik nie null, es wird also eine Verteilung auf dem offenen Intervall $(0,1]$ erzeugt. Das kann in Simulationen unerwünscht sein, erlaubt auf der anderen Seite aber das gefahrlose Rechnen mit Zufallszahlen (insbesondere ist es für die im letzten Abschnitt vorgestellte Methode zur Erzeugung nicht-gleichverteilter Zufallszahlen wichtig, da dort durch eine Zufallszahl geteilt wird). Die Modulooperation gestaltet sich einfach, da die Module nahe an 2^{15} liegen ([1, S. 29]). Der Aufwand zur Berechnung einer Zufallszahl sind 10 (Fließkomma–)Punktoperationen sowie 2 Additionen. Sieht man von der unnötigen

Abbildung 2: Verteilung nach Wichmann-Hill im $[0, 1]^3$



Kapselung des Zustandes in einem höheren Datentyp ab, hat der Algorithmus einen Speicherbedarf von 3 Integervariablen, auf denen er in-place arbeitet und benötigt für jede Ausführung (cirka) 22 CPU-Anweisungen (in der Annahme, dass der Zustand im Hauptspeicher verwaltet wird).

III.5. PRNGs in Simulationswerkzeugen

Im direkten Vergleich dazu betrachten wir den von NS-2, einem Werkzeug zur Simulation von Netzwerken, verwendeten Zufallszahlengenerator. Das Programm verwendet seit der Version 2.1b9 den Generator MRG32k3a von Pierre L'Ecuyer ([9]). Dieser verwendet zur Berechnung des Zustandes x_n die letzten k Zufallszahlen $x_{n-k} \dots x_{n-1}$, anstatt nur von einer auszugehen und erreicht damit laut seines Autors sowohl eine wesentlich höhere Periode (In der Größenordnung von 2^{100}) als auch bessere strukturelle Eigenschaften als ein einfacher linearer Kongruenzgenerator. ([8]) In der praktischen Anwendung werden diese Eigenschaften jedoch oft durch eine falsche Initialisierung seitens der Benutzer wieder zunichte gemacht ([9])

III.6. Erzeugung nicht-gleichverteilter Zufallszahlen

Im letzten Abschnitt wollen wir die Erzeugung nicht-gleichverteilter Zufallszahlen diskutieren. Hierzu bieten sich verschiedene Methoden an.

Das naheliegendste Verfahren ist die sogenannte *Quantiltransformation*, auch *Inversionsmethode*, bei der die Verteilungsfunktion der gewünschten Verteilung invertiert wird, um gleichverteilte Daten in die Zielverteilung umzurechnen. Als Beispiel diene ([1, S. 36]) die Exponentialverteilung mit Verteilungsfunktion

$$F(x) = 1 - e^{-\lambda x}$$

Deren Umkehrfunktion lautet

$$F^{-1}(u) = -\frac{1}{\lambda}\ln(1-u)$$

und damit ist $F^{-1}(\psi(\zeta(\cdots)))$ exponentialverteilt. (Für einen Beweis dieser Behauptung siehe [4, Satz 1.30]) Damit sich die Auswertung billiger gestaltet, kann man in der Implementierung $1-u$ durch u ersetzen.

Ein weiteres interessantes Vorgehen ist die Bildung von Quotienten gleichverteilter Zufallsvariablen auf bestimmten, zur gewünschten Verteilung passend gewählten, Ausschnitten des $\mathbb{R}^2$. Dazu generiert man für eine gegebene kontinuierliche Dichte $f(x)$ zwei gleichverteilte Zufallsvariablen (U, V) auf

$$\left\{(u,v) \in \mathbb{R}^2 : 0 \le u \le f^{\frac{1}{2}}\left(\frac{v}{u}\right)\right\} \ .$$

$\psi(u,v) = \frac{V}{U}$ ist dann entsprechend f verteilt. Für einen Beweis dieser Behauptung siehe [2, S. 2] oder auch [1, S. 40ff].

Als Beispiel betrachte die Implementierung der Normalverteilung im `random`-Modul von Python:

```
1  NV_MAGICCONST = 4 * _exp(-0.5)/_sqrt(2.0)
2  def normalvariate(self, mu, sigma):
3      random = self.random
4      while 1:
5          u1 = random()
6          u2 = 1.0 - random()
7          z = NV_MAGICCONST*(u1-0.5)/u2
8          zz = z*z/4.0
9          if zz <= -_log(u2):
10             break
11     return mu + z*sigma
```

In der `while`-Schleife werden zwei gleichverteilte Zufallszahlen in $[0,1]$ generiert. Das ψ aus unserer Betrachtung heißt hier z, wobei `u1` (unser u) erst an dieser Stelle auf

die richtige Höhe des $\mathbb{R}^2$-Ausschnittes umgerechnet wird. Um eine Gleichverteilung auf dem ganzen Ausschnitt zu erziehlen, wird ein *Acceptance-Rejection*-Verfahren verwendet: Die Zufallszahlen werden auf $[0, 1]^2$ erzeugt und Punkte, die außerhalb der gewünschten Teilmenge liegen, werden verworfen (in Zeile 9). Der Rückgabewert wird schlussendlich auf die gewünschten Parameter μ bzw. σ umgerechnet.

Abhängig von der Implementierung der `random()` Funktion und den übergebenen Parametern kann diese Methode allerdings sehr viele Schleifendurchläufe benötigen, bis die Abbruchbedingung erreicht wird. Eine Abschätzung für den hier gewählten Wichmann-Hill-Algorithmus kann hier leider nicht angegeben werden.

In [2, S. 3] wird dieses Verfahren zur Erzeugung nicht-gleichverteilter Daten als überlegen gegenüber anderen Ansätzen angegeben: Kein Algorithmus war im Test der Autoren sowohl schneller in der Ausführung als auch kürzer in der Implementierung.

IV. Schlussbemerkung

In den vorherigen Abschnitten konnte nur ein kleiner Ausschnitt von Verteilungen und Simulationsmethoden vorgestellt werden. Gerade im Vergleich der letzten beiden vorgestellten Verfahren, der Quantiltransformation und dem Quotientenverfahren, sollte klargeworden sein, dass stochastische Simulation viele mathematische Überlegungen neben dem reinen Programmieren erfordert.

Zum weitergehenden Studium der Simulation von Zufallszahlen sei dem Leser vor allem [1] ans Herz gelegt.

Literatur

[1] Hansruedi Künsch, *Stochastische Simulation* (Vorlesungsskript), ETH Zürich, 2006, ftp://ftp.stat.math.ethz.ch/U/Kuensch/skript-sim.ps [abgerufen am 12. Juni 2010]. 8, 9, 10, 11, 13, 14

[2] A. J. Kinderman, J. F. Monahan, *Computer Generation of Random Variables Using the Ratio of Uniform Deviates*, Volume 3, Issue 3, Pages: 257 - 260, Association for Computing Machinery, New York, 1977, DOI 10.1145/355744.355750 13, 14

[3] B. A. Wichmann, *Generating good pseudo-random numbers*, Computational Statistics and Data Analysis, Volume 51, Issue 3, Pages: 1614-1622, National Physical Laboratory, Teddington, UK, 2006, DOI 10.1016/j.csda.2006.05.019 10

[4] Hans-Otto Georgii, *Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik*, 3. Auflage, de Gruyter, 2007. 3, 7, 9, 13

[5] Pierre l'Ecuyer und Richard Simard, *TestU01: A C Library for Empirical Testing of Random Number Generators*, Universite de Montreal, Montreal, 2007, http://www.iro.umontreal.ca/~lecuyer/myftp/papers/testu01.pdf [abgerufen am 12. Juni 2010]. 9

[6] George Marsaglia, *The marsaglia random number cdrom including the diehard battery of tests of randomness*, FloridaState University, 1995, http://www.stat.fsu.edu/pub/diehard/ [Abgerufen am 14. Juni 2010], Insb. `keynote.ps` auf der CD-Rom. 9

[7] Philip Bevington, *Data Reduction and Error Analysis for the Physical Sciences*, 3. Auflage, McGraw-Hill Science/Engineering/Math, 2002.

[8] Pierre l'Ecuyer, *Good parameters and implementations for combined multiple recursive random number generators*, Universite de Montreal, Montreal, 1998, http://www.iro.umontreal.ca/~lecuyer/myftp/papers/combmrg2.ps[abgerufen am 21. Mai 2010]. 12

[9] Martina Umlauft, Peter Reichl, *Experiences with the NS-2 network simulator*, Vienna University of Technology und Telecommunications Research Center, Wien, 2007, http://userver.ftw.at/~reichl/publications/WTS07.pdf [abgerufen am 21. Mai 2010]. 12

[10] Python Standarddistribution, `random.py`, Python 2.6.5, http://www.python.org/ftp/python/2.6.5/Python-2.6.5.tar.bz2 [Abgerufen am 14. Juni 2010]. 10