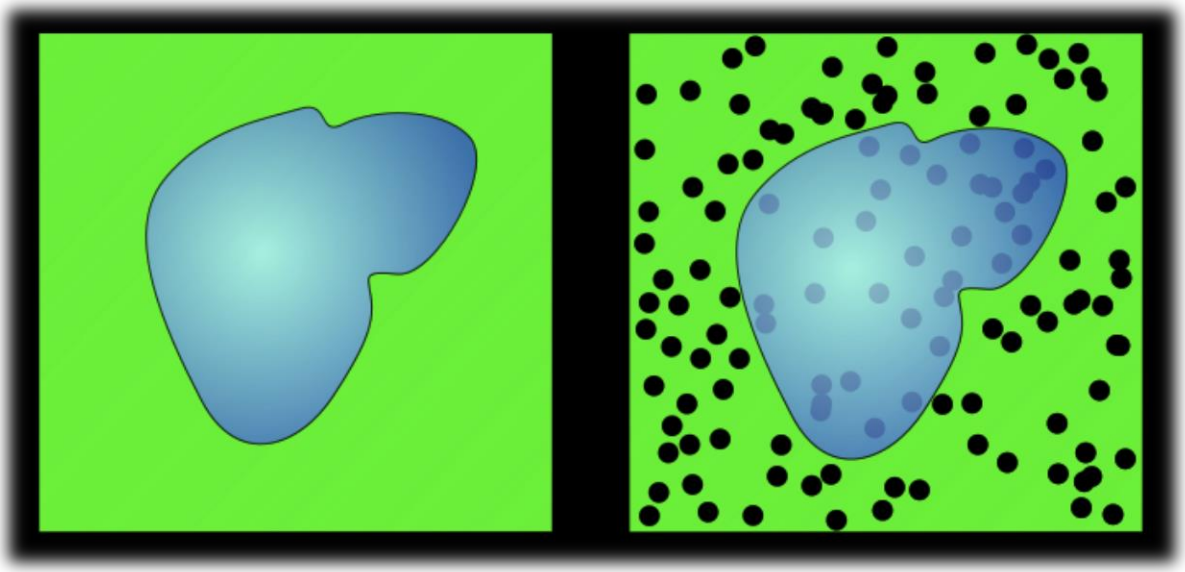
AUSGEWÄHLTE STATISTISCHE METHODEN



Randomisierte Algorithmen & Monte Carlo Methoden

Spezialgebiet im Fach Mathematik

Von Micha Birklbauer

I. Einführung

Was kann ich mir unter Monte Carlo Simulation überhaupt vorstellen? Die Idee hinter Monte Carlo Simulationen ist die, ein Problem, das sich schwierig bzw. unmöglich direkt mit wahrscheinlichkeitstheoretischen Methoden berechnen lässt, als Zufallsexperiment zu konstruieren und dieses anschließend viele Male von einem Computer durchführen zu lassen. Indem man über die Ergebnisse mittelt erhält man näherungsweise den Erwartungswert.

Doch bevor wir uns näher mit der Monte Carlo Simulation beschäftigen, sollten wir zuerst einige grundlegende Dinge aus der Wahrscheinlichkeitstheorie definieren.

Die Definition der Wahrscheinlichkeit

Eine Abbildung P welche jedem Ereignis A aus dem Ereignisraum Ω (A ϵ Ω) eine Zahl P(A) ϵ R zuordnet. Wobei die folgenden Kriterien gelten müssen:

- P(Ω) = 1;
  Oder in Worten: Die Wahrscheinlichkeit, dass irgendein Ereignis aus dem Ereignisraum eintritt, muss 1 betragen.
- ∀ A ϵ Ω : P(A) ≥ 0;
  Für alle Ereignisse aus dem Ereignisraum muss gelten, die Wahrscheinlichkeit, dass dieses Ereignis eintritt ist größer als 0.
- $\Omega = \sum_{i=1}^{n} A_i \mid A_i bis\ A_n = disjunkt;$
  Der Ereignisraum setzt sich zusammen aus allen möglichen Ereignissen, wobei diese sich nicht überschneiden.
- $P(A_1 \cup A_2 \cup \dots) = P(A_1) + P(A_2) + \dots;$
  Die Wahrscheinlichkeit, dass Ereignis A_1 oder Ereignis A_2 eintritt entspricht der Summe der Wahrscheinlichkeit, dass Ereignis A_1 eintritt + der Wahrscheinlichkeit, dass Ereignis A_2 eintritt.

Spezielle Ereignisse sind die leere Menge (das unmögliche Ereignis) und der ganze Ereignisraum (das sichere Ereignis).

Die Definition der Zufallsvariablen

Eine Zufallsvariable X ordnet jedem Ausfall eines Zufallsversuchs eine reelle Zahl zu. Zum Beispiel: Würfel mit Augenzahlen 1 bis 6. Zufallsvariable = Augenzahl. Wert(e) der Zufallsvariable = {1, 2, 3, 4, 5, 6}

Die Definition der Wahrscheinlichkeitsverteilung

Wird jedem möglichen Ereignis A (Wert) der Zufallsvariable X eine Wahrscheinlichkeit P(X = A) zugeordnet, nennt man das Wahrscheinlichkeitsverteilung von X. Zum Beispiel:

Wahrscheinlichkeitsverteilung eines Laplace-Würfels:						
X	1	2	3	4	5	6
P(X)	1/6	1/6	1/6	1/6	1/6	1/6

Die Definition des Erwartungswertes

Der Erwartungswert entspricht näherungsweise dem Mittelwert aller Zufallsausfälle bei einer hohen Anzahl von Versuchsausführungen.

$$\mu = E(X) = \sum_{x \in X} x \,.\, P(\{X = x\})$$

Nach diesen Begriffen aus der Wahrscheinlichkeitstheorie sollten wir auch den Begriff des ***Algorithmus*** definieren, da ich ihn im weiteren Verlauf des Öfteren verwenden werde.

Als Algorithmus bezeichnet man im Grunde genommen eine genaue Vorschrift zum Lösen eines Problems oder einer ganzen Klasse von Problemen. Da Algorithmen aus sehr vielen genauestens definierten Einzelschritten bestehen, eignen sie sich hervorragend für die Implementierung am Computer mittels zur Verfügung stehender Programmiersprachen. Vereinfacht kann man das Schema eines Algorithmus als „Input -> Problemlösung -> Output“ darstellen.

II. Randomisierte Algorithmen

Randomisierte Algorithmen (auch stochastische oder probabilistische Algorithmen genannt) versuchen durch den Einsatz von Zufallszahlen auf ein annähernd korrektes Ergebnis zu kommen. Sie bilden das Gegenstück zu deterministischen Algorithmen, da sie nur zum Teil rekonstruierbar sind. Vorteile der randomisierten Algorithmen sind vor allem, dass sie meist einfacher zu verstehen sind, einfacher in Programme implementierbar sind und in den meisten Fällen auch effizienter als deterministische Algorithmen für ein und dasselbe Problem.

Monte Carlo Methoden sind also randomisierte Algorithmen, die Zufallszahlen benutzen. Neben den üblichen Befehlen (die Grundrechenarten, Vergleich von Zahlen, Berechnung von Funktionswerten) müssen also zusätzlich Befehle der Art

- „Wähle x Є [0; 1] zufällig“ oder
- „Wähle x Є {0, …, N – 1} zufällig“

definiert werden. Hier stellt sich das nächste Problem: Wie erzeugt man künstlich zufällige Zahlen?

Der Pseudozufall (Definition)

Als pseudozufällig wird alles bezeichnet, was aus der Perspektive des Betrachters nicht von wirklicher Zufälligkeit unterschieden werden kann. Das Ergebnis eines Münzwurfs wird beispielsweise generell als zufällig angesehen. Befindet sich die Münze bereits in der Luft, ist es jedoch theoretisch möglich, anhand ihrer Rotation, Geschwindigkeit und anderer Messdaten das Ergebnis vorherzusagen. Jemandem, dem entsprechende Messgeräte (und Rechenkapazität) nicht zur Verfügung stehen, erscheint der Wurf aber immer noch zufällig; der Wurf mit der Münze in der Luft ist für ihn pseudozufällig. Generell definiert als pseudozufällig, was durch effiziente Algorithmen nicht vorhergesagt werden kann. Pseudozufälligkeit ist aber immer noch berechenbar (man kann sie effizient erzeugen), nur nicht vorhersagbar.

III. Erzeugung von Zufallszahlen

Sowohl zur Simulation von Zufallsprozessen im eigentlichen Sinn als auch zur Lösung vieler anderer mathematischen Aufgaben ist es wünschenswert, am Computer Stichproben von Zufallsvariablen erzeugen zu können. Im Prinzip könnte man dazu natürlich „echte" Zufallsprozesse wie den Zerfall einer radioaktiven Quelle oder thermisches Rauschen verwenden. Abgesehen von instrumentellen Problemen (und wer hätte schon gern eine Strahlungsquelle auf dem Schreibtisch), würden sich auf diese Weise aber kaum Zufallszahlen in hinreichender Menge und mit der Geschwindigkeit erzeugen lassen, wie sie von vielen Anwendungen verbraucht werden. Außerdem muss es für Testzwecke möglich sein, ein und dieselbe Folge von Zufallszahlen beliebig oft zu reproduzieren. Aus diesem Grund verwendet man in Simulationen fast ausschließlich Pseudozufallszahlen, also Folgen von Zahlen, die zwar durch einen streng deterministischen Algorithmus erzeugt werden, aber „zufällig aussehen", d.h. gewisse statistische Tests auf Zufälligkeit erfüllen. (Daraus folgt auch, dass es für jeden Pseudo-Zufallszahlengenerator eine Anwendung geben kann, bei der er sich als nicht zufällig genug erweist). In den meisten Compilern, Interpretern und manchen Script-Sprachen ist zumindest ein (Pseudo)-Zufallszahlengenerator implementiert, der gleichverteilte Zufallszahlen aus dem Intervall [0, 1) liefert. Häufig ist das ein sogenannter linearer Kongruenzgenerator (Modulo-Generator), der im einfachsten Fall nach dem folgenden Prinzip arbeitet:

Es werden ganze Zahlen a, c und m sowie ein Startwert („Seed") $i_0 < m$ vorgegeben und rekursiv die Folgen

- $i_{n+1} = (a * i_n + c) \bmod m$
- $\xi_{n+1} = \frac{i_{n+1}}{m}$

gebildet. Die Zahlen a, c und m sind fest und bestimmen die Eigenschaften des Zufallszahlengenerators. Nach Vorgabe eines (mit gewissen Einschränkungen) beliebigen Startwerts i_0 erhält man durch sukzessives Anwenden der obigen Vorschrift zunächst eine streng deterministische Folge von ganzen Zahlen $i_0, i_1, i_2, \ldots$ mit $0 \leq i_n \leq m$. Die eigentlichen „Funktionswerte" des Generators sind die ξ_n für die nach Konstruktion $0 \leq \xi_n \leq 1$ gilt. Auf Grund der Formeln ist klar, dass die i_n höchstens m verschiedene Werte annehmen können; dann muss sich die Folge wiederholen. Unter bestimmten

Voraussetzungen an die Parameter a, c und m kann man erreichen, dass die Periode des Generators den Maximalwert m erreicht. Dann sind die ξ_n in ihrer Gesamtheit auf jeden Fall gleichverteilt auf dem Intervall [0, 1). Je nach Wahl von a, c und m erfüllen sie auch gewisse Tests auf Zufälligkeit (statistische Unabhängigkeit). Auf keinen Fall dürfen jedoch für die Parameter des Generators x-beliebige Zahlen eingesetzt werden.

Die folgende Tabelle enthält die Parameter für einige einfache Generatoren:

a	c	m	i_0
16807	0	$2^{31}-1$	ungerade
65539	0	2^{31}	ungerade
69069	1	2^{32}	
1664525	0	2^{32}	ungerade
25214903917	11	2^{48}	

Der Eintrag in der zweiten Zeile („IBM-Generator") war seinerzeit auf Großrechnern weit verbreitet, ist aber eigentlich ein Gegenbeispiel eines guten Zufallszahlengenerators, da die resultierenden Zufallszahlen stark korreliert sind. Fasst man nämlich Tripel von aufeinanderfolgenden Funktionswerten zu Koordinaten $\{\xi_n, \xi_{n+1}, \xi_{n+2}\}$ im dreidimensionalen Einheitswürfel zusammen, so kommen alle diese Punkte auf einer geringen Anzahl von Ebenen zu liegen. Solche Korrelationen treten zwar prinzipiell immer auf, bei guten Generatoren aber erst in viel höheren Dimensionen.

Mersenne-Twister

Der Mersenne-Twister ist einer der heute am meisten verwendetsten Zufallszahlengeneratoren, er kommt unter anderem in der Software Mathematica zum Einsatz. Er wurde im Jahre 1997 von Makoto Matsumoto und Takuji Nishimura entwickelt und wurde populär, da er viele Probleme älterer Algorithmen überwand. Seine Eigenschaften sind die extrem lange Periode von $2^{19937-1}$ von der auch sein Name stammt. Die Periodenlänge ist eine Mersenne-Primzahl von der sich viele Eigenschaften des Generators herleiten lassen. Die Ausgabe ist hochgradig gleichverteilt wodurch nur eine sehr geringe Korrelation zwischen aufeinanderfolgenden Werten entsteht. Der Algorithmus zeichnet sich außerdem durch Schnelligkeit aus.

IV. Monte Carlo Simulationen/Monte Carlo Methoden

Während im deutschen Sprachgebrauch noch manchmal zwischen MC Simulation und MC Methoden unterschieden wird, existiert im Englischen der Begriff der MC Simulation gar nicht. Im Deutschen spricht man allgemein von einer Monte Carlo Simulation wenn eine Monte Carlo Methode oft ausgeführt wird und anschließend der Mittelwert der Ereignisse als Ergebnis dient, wobei sich hier MC Simulation und MC Methode schon überschneiden, da dies auch der Grundsatz vieler MC Methoden ist.

Als Monte Carlo Methode wird allgemein definiert (nach Wolfram MathWorld):

“Any method which solves a problem by generating suitable random numbers and observing that fraction of the numbers obeying some property or properties. The method is useful for obtaining numerical solutions to problems which are too complicated to solve analytically.”

Geschichte & Namensgebung

Im Jahre 1934 war Enrico Fermi der erste der die Monte-Carlo-Simulation nutzte, bevor sie einen Namen hatte. Er nutze sie für die Simulation einer Neutronenstreuung. Im Jahre 1946 entdeckte dann ein Mann namens Stan Ulam diese Methode für sich. Ulam arbeitete im Jahre 1946 in einem geheimen Projekt Namens „Manhattan Project“. Als Ulam aufgrund einer Krankheit im Bett lag, spielte er dabei gerne Canfield Solitär. Dabei stellte er sich die Frage mit welcher Wahrscheinlichkeit er in Canfield Solitär mit 52 Karten gewinnen kann. Am Anfang versuchte er alle mögliche Kombinationen durchzugehen. Dabei fiel ihm auf, dass es sehr zeitaufwendig ist alle Kombinationen durchzugehen. Als nächstes versuchte er 100 zufällige Spiele zu spielen, dabei zählte Ulam die gewonnen Spiele. Zum Schluss dividierte er die Anzahl der gewonnen Spiele durch 100. Dabei fiel ihm auf, dass er durch eine Annäherung ein schwieriges Problem in ein einfacheres Problem umgewandelt hat. Dies berichtete er John von Neumann, der auch im „Manhattan Project“ tätig war. Von Neumann erkannte sofort das Potential dieser Methode, gemeinsam entwickelten sie auf dieser Basis mehrere Algorithmen und lösten damit viele Probleme in der Wissenschaft. Den Namen erhielt die Methode von John von Neumann. Er kommt von einer Anspielung auf Ulam’s Onkel, der sich im gleichnamigen Casino gerne „verzockte“.

Beispiele für Monte Carlo Methoden:

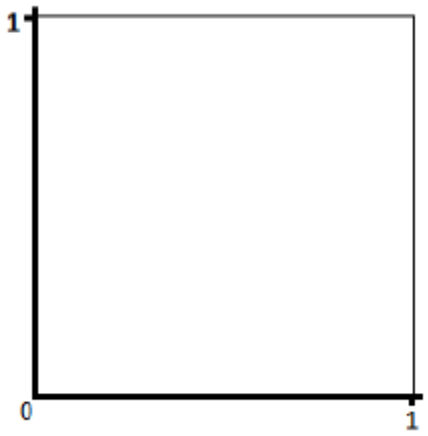
Die Monte Carlo Integration – Lösung von Integralen mittels MC

Wohl das bekannteste Beispiel für die Lösung von Integralen durch MC ist die Berechnung der Zahl π. Dieses Beispiel demonstriert sehr gut die Methode für die Berechnung von Integralen und wie sie verallgemeinert werden kann. Man betrachtet ein Quadrat dem ein Viertelkreis eingeschrieben wird. Man „schießt" nun blindlings auf dieses Quadrat, indem man zwei voneinander unabhängige, gleichverteilte Zufallszahlen im Intervall [0; 1] generiert welche wie in untenstehender Abbildung die Koordinaten des Schusses darstellen. Die Wahrscheinlichkeit, dass der Treffer innerhalb des Kreises liegt verhält sich zur Wahrscheinlichkeit eines Fehlschusses, wie die Kreisfläche zur Quadratfläche. Bei jedem Treffer innerhalb des Kreises erhöht man einen Zähler c um 1, und gleichzeitig wird für jeden Schuss der Nenner N um 1 erhöht. Aus diesem einfachen Ansatz erhält man nun:

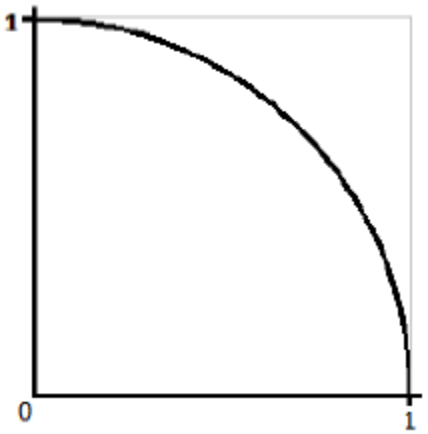
$$\pi(N) = 4\frac{c}{N}$$

Und für den entstehenden Fehler gilt:

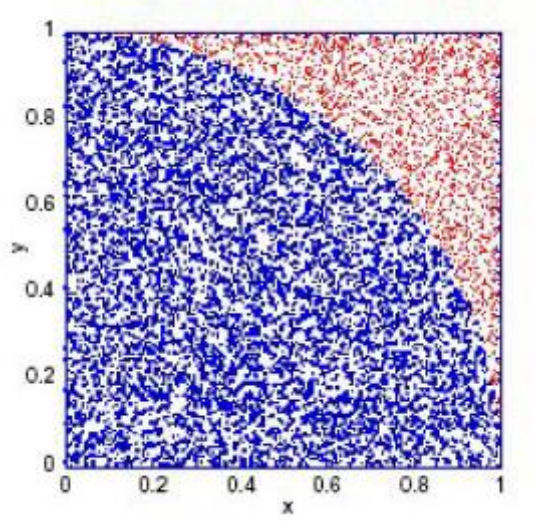
$$\varepsilon = |\pi - \pi(N)|$$



Ein Quadrat mit Seitenlänge 1 wird erstellt.



Ein Kreis mit Radius 1 wird eingeschrieben



In dem Quadrat werden zufällig Punkte generiert.

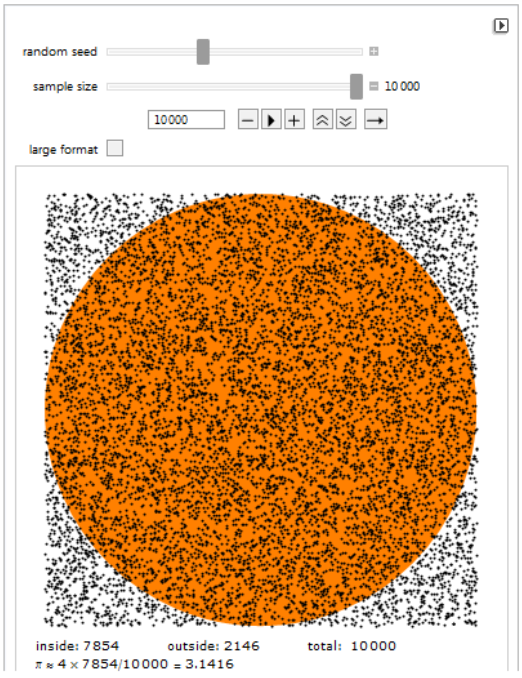
$$\pi = 4 * \frac{Anzahl\ der\ Punkte\ im\ Viertelkreis}{Anzahl\ aller\ generierten\ Punkte}$$

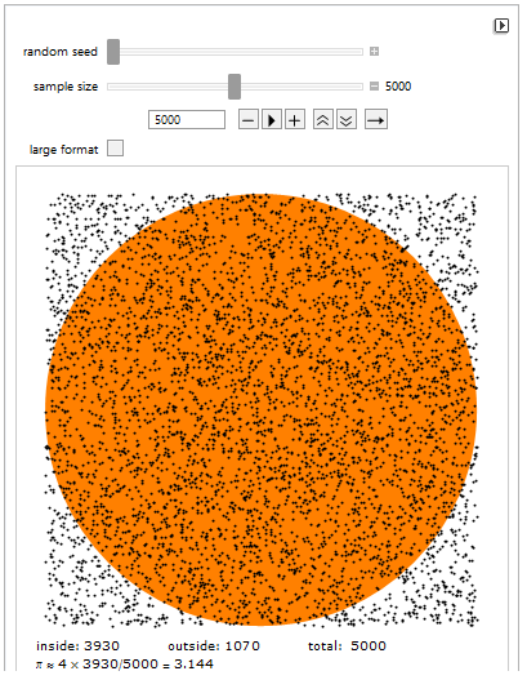
$$A = 1^2 * \frac{Anzahl\ der\ Punkte\ im\ Viertelkreis}{Anzahl\ aller\ generierten\ Punkte}$$

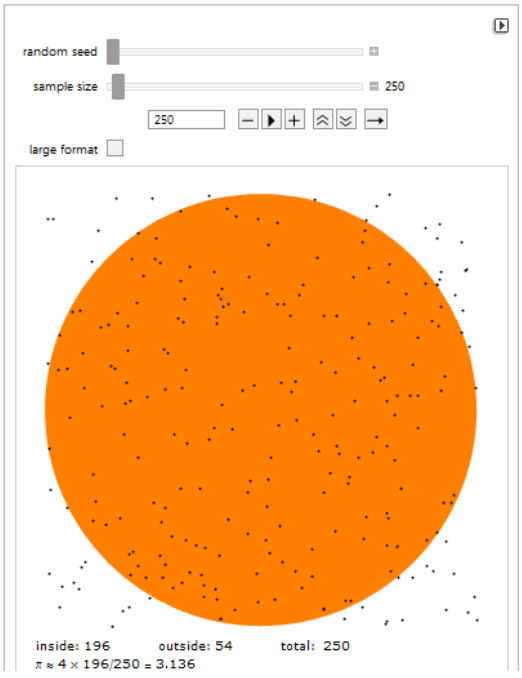
Für Monte Carlo Simulationen gilt die Faustregel: Um den entstehenden Fehler zu halbieren muss die Anzahl der Simulationen quadriert werden.

Bei 100 000 Simulationen ist das Ergebnis etwa auf 2 Nachkommastellen genau.

Dass die Genauigkeit mit der Anzahl der Simulation (bzw. Anzahl der generierten Punkte) steigt, sieht man anhand dieser Grafiken:







1) 10 000 generierte Punkte. Fehler: ~0,000 007
2) 5000 generierte Punkte. Fehler: ~0,002 407
3) 250 generierte Punkte. Fehler: ~0,005 592

Lösung konkreter Fragestellungen mittels MC (in Mathematica)

Zum Beispiel: Ein Spieler spielt ein faires Spiel mit 5€ Startkapital. Er gewinnt mit 50% WS und verliert mit 50% WS. Wenn er gewinnt, bekommt er 1€, wenn er verliert, zahlt er 1€. Das Spiel geht solange bis der Spieler pleite ist.

Fragestellung:

- Wie groß ist die Wahrscheinlichkeit, dass der Spieler bis zum A-ten Spiel pleite ist?
- Wie groß ist die Wahrscheinlichkeit, dass der Spieler bis zum A-ten Spiel genau das Kapital d hat?
- Wie groß ist die Wahrscheinlichkeit, dass der Spieler bis zum A-ten Spiel mehr als das Kapital d hat?

Wir lösen das Problem indem wir mittels Mathematica mehrere tausend Spiele simulieren und dann über das Ergebnis mitteln.

Mittels des Befehls *RandomInteger[]* erzeugen wir eine Zufallszahl, die entweder 0 oder 1 ist (beide treten mit WS von ½ ein). Unsere erste Zeile wird sein:

If[RandomInteger[] == 1, k++, k = k - 1];

Welche so viel bedeutet wie, wenn die Zufallszahl 1 ist, füge dem Kapital k 1 hinzu, wenn die Zufallszahl 0 ist, ziehe dem Kapital 1 ab. Im nächsten Schritt fügen wir die Bedingung, dass das Spiel bei k = 0 zu Ende ist, hinzu.

While[k > 0 && length > a, If[RandomInteger[] == 1, k++, k = k - 1]; AppendTo[l, k]; a++];

Solange k größer als 0 ist und mehr als a Runden gespielt wurden, wird obiges ausgeführt, wobei a als Laufvariable dient. Das Kapital k wird anschließend der Liste l angefügt. Diese Zeile wäre eine erste Simulation, da wir jedoch mehrere tausende Simulationen möchten, müssen wir dies auch noch mithilfe einer „for“-Schleife definieren.

```
For[i = 0, i <= n, i++,

    While[k > 0 && length > a, If[RandomInteger[] == 1, k++, k = k - 1];

        AppendTo[l, k]; a++];

        AppendTo[m, l];        (*Alle Versuchsausführungen*)

        AppendTo[e, Last[l]];

    If[Last[l] == d, j++];          (*Endkapital gleich d*)

    If[Last[l] > d, h++];           (*Endkapital größer d*)

    c = j/n;

k = 5;                              (*Kapitalrücksetzung*)

l = {5};                            (*Listenrücksetzung*)

a = 0;

]
```

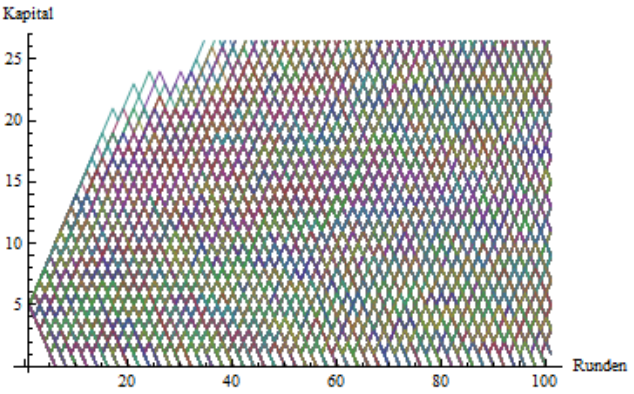
Die vorige Simulation wird mittels des Befehls *For[i = 0, i <= n, i++, …]* n-male ausgeführt. Nach jeder Simulation wird die Liste l einer größeren Liste m hinzugefügt. Letzten Endes beinhaltet die Liste m insgesamt n-viele Listen l. Mit den Befehlen *If[Last[l] == d, j++];* bzw. *If[Last[l] > d, h++];* fügen wir der Zählvariable j 1 hinzu, wenn der letzte Wert der Liste l gleich d ist, wir fügen der Zählvariable h 1 hinzu, wenn der letzte Wert der Liste l größer als d ist. Die Variable c ist die Wahrscheinlichkeit, dass wir nach „length“-vielen Runden das Kapital d haben. Sie ist definiert als Wert der Zählvariablen j durch die Anzahl der Simulationen n. Die Wahrscheinlichkeit, dass wir nach „length“-vielen Runden mehr als das Kapital d haben lässt sich gleichermaßen mit h/n berechnen. Da jede Simulation den Variablen k, l und a neue Werte zuweist, müssen diese am Ende jeder Simulation wieder neu ihren ursprünglichen Werten zugewiesen werden.

In[37]:=
```
l = {5}; (*Liste der Versuchsausfälle*)
k = 5; (*Kapital*)
n = 10000; (*Anzahl der Simulationen*)
m = {}; (*Liste aller Versuchsausführungen*)
length = 100; (*Versuchslänge*)
a = 0; (*Laufvariable*)
b = {}; (*Leere List*)
d = 0; (*Endkapital*)
e = {}; (*kleine Liste für Balkendiagramm*)
g = {}; (*große Liste für Balkendiagramm*)
j = 0; (*Zählvariable für Endkapital gleich d*)
h = 0; (*Zählvariable für Endkapital größer d*)
For[i = 0, i ≤ n, i++,
 While[k > 0 && length > a, If[RandomInteger[] = 1, k++, k = k - 1]; AppendTo[l, k]; a++];
 AppendTo[m, l]; (*Alle Versuchsausführungen*)
 AppendTo[e, Last[l]];
 If[Last[l] = d, j++]; (*Endkapital gleich d*)
 If[Last[l] > d, h++]; (*Endkapital größer d*)
 c = j / n;
 k = 5; (*Kapitalrücksetzung - Ändern falls k geändert wird*)
 l = {5}; (*Listenrücksetzung - siehe Kapitalrücksetzung*)
 a = 0;
]
For[f = 0, f ≤ length + k, f++, AppendTo[g, Count[e, f]]]; (*WS aller Versuchsausfälle*)
c // N (*WS das Endkapital gleich d*)
h / n // N (*WS das Endkapital größer d*)
ListLinePlot[m, AxesLabel → {Runden, Kapital}, AxesOrigin → {1, 0}] (*Grafische Darstellung zu m*)
BarChart[g, ChartStyle → "DarkRainbow"]
(*Balkendiagramm zu WS aller Versuchsausfälle*)
```

Out[38]= 0.612

Out[39]= 0.3881

Out[40]=



Komplette Implementierung für 10 000 Spiele und einer Spiellänge von 100 Runden. Wobei das zu erreichende Kapital d in diesem Fall 0 betrug. Die WS bis zum hundertsten Spiel Pleite zu gehen beträgt etwa 61%.

Heutige Anwendung von MC Methoden

- Mathematik: Berechnung hochdimensionaler Integrale, Berechnung von Differentialgleichungen unter komplexen Randbedingungen
- Physik: Simulation von brownschen Bewegungen, Rauschen und Diffusion
- Chemie/Biologie: Simulation von Bakterienkulturen
- Finanzmathematik: „Vorhersagen“ von Aktienkursen

V. Las Vegas Algorithmen

Las Vegas Algorithmen sind Varianten der Monte Carlo Algorithmen (=Methoden), im Gegensatz zu einem Monte Carlo Algorithmus darf ein Las Vegas Algorithmus nie ein falsches Ergebnis liefern. Im Allgemeinen werden sämtliche randomisierte Algorithmen die nie ein falsches Ergebnis liefern, als Las Vegas Algorithmen bezeichnet. Dabei wird zwischen zwei Typen von LV Algorithmen unterschieden:

- Algorithmen die nicht versagen, daher nicht „aufgeben“ dürfen. Die Rechenzeit um auf ein Ergebnis zu kommen kann in diesem Fall oft sehr lange sein.
- Algorithmen die „aufgeben“ dürfen. Die Qualität dieser Algorithmen wird in ihrer Versagenswahrscheinlichkeit bemessen.

Verbindet man einen MC Algorithmus mit einem Algorithmus der die Lösung verifiziert, erhält man eine Las Vegas Algorithmus.

Inhaltsverzeichnis

Quellenverzeichnis

Projektwoche der angewandten Mathematik: Kurs Monte Carlo Simulation unter der Leitung von Harald Hinterleitner

Müller-Gronbach, Thomas/Novak, Erich/Ritter, Klaus: Monte Carlo-Algorithmen. Heidelberg: Springer Verlag, 2012

Kroese, Dirk/Taimre, Thomas/Botev, Zdravko: Handbook of Monte Carlo Methods. Hoboken: John Wiley & Sons, 2011

Theis, Christian/Kernbichler, Winfried: Grundlagen der Monte Carlo Methoden. Graz: TU, 2002

Schlundt, Johannes: Schriftliche Ausarbeitung zum Vortrag Monte-Carlo-Simulation. Hamburg: Universität Hamburg, 2013

Von der Linden, Wolfgang/Prüll, Alexander: Wahrscheinlichkeitstheorie, Statistik und Datenanalyse. Graz: TU, 2002

Karch, Rudolf/Neumann, Martin: Monte Carlo-Simulation. Wien: Med. Uni. Wien, 2008

McLoone, Jon: Monte Carlo Esimate for Pi. In: URL:
http://demonstrations.wolfram.com/MonteCarloEstimateForPi/
[Stand 18.05.2014]

O.V.: Randomisierter Algorithmus. In: URL:
http://de.wikipedia.org/wiki/Randomisierter_Algorithmus
[Stand 18.05.2014]

O.V.: Monte Carlo Method. In: URL:
http://mathworld.wolfram.com/MonteCarloMethod.html
[Stand 18.05.2014]