Der virtuelle Geigerzähler und die Monte-Carlo Methode als wissenschaftliches Überbleibsel des Manhatten-Projekts

Bernd Laquai, 2. Oktober 2015

Es kommt einem schon mächtig der Grusel, wenn man liest mit welchem Hype und mit welcher Begeisterung internationale Forscher gegen Ende des zweiten Weltkriegs an den ersten Atomwaffen "gebastelt" haben. Noch heute gibt es viele Seiten im Internet, die diese Begeisterung widerspiegeln und das Ganze als eine heroische Tat darstellen. Man kann sehr deutlich erkennen, dass die damals größte amerikanische Geheim-Mission mit dem Tarnnamen "Manhattan-Projekt" ganz ähnlich lief, wie etliche Jahre später die Apollo-Mission, welche die Amerikaner auf den Mond bringen sollte. Es wurden die besten Forscher aller Länder angeheuert, es wurde unheimlich viel Geld bereitgestellt und jedes Hindernis, das sich auf dem Weg zum Ziel befand mit allen Mitteln aus dem Weg geschafft. Forschung und Entwicklung um jeden Preis quasi oder anders ausgedrückt der Zweck heiligt die Mittel. Wie bei der späteren Apollo-Mission, so entstanden auch beim Manhattan-Projekt unzählige, heute noch viel benutzte Forschungsergebnisse praktisch als „Abfallprodukte". Sie wurden dann wegen ihrer später erlangten Berühmtheit auch teilweise dazu benutzt um solche doch sehr fragwürdigen, als "Meilensteine der Menschheit" gefeierten, aber meist militärisch begründeten Großprojekte der Supermächte zu legitimieren.

Eines dieser „Abfallprodukte" der Kernwaffen-Forschung und Entwicklung ist die heute aus der Welt der Wissenschaft kaum noch weg zu denkende Monte-Carlo Methode. Sie wird zur Berechnung komplexer Vorgänge, die mit „normalen" analytischen Mitteln schwer zu erreichen sind eingesetzt. Der amerikanische Physiker Nicholas Metropolis und der polnische Mathematiker Stanislaw Ulam arbeiteten zusammen in einem Team um den italienischen Kernphysiker Enrico Fermi und dem ungarischen Kernphysiker Edward Teller um die selbsterhaltende Kernspaltung als Kettenreaktion militärisch nutzbar zu machen. Dabei galt es einiges zu berechnen, wo die damaligen Rechner, die seinerseits teilweise noch analog arbeiteten, schnell überfordert waren. Da Ulams Onkel öfters das Spielcasino in Monte-Carlo besuchte und dort einiges an Geld verzockte und man zudem beim Manhattan-Projekt für alles einen Code-Namen verwenden musste, kamen Metropolis und Ulam auf die Idee, ihre neue Rechenmethode, die mit Zufallszahlen arbeitete, die „Monte-Carlo" Methode zu nennen. Mit dieser auf Zufallsprinzipien basierenden Methode waren die beiden richtig erfolgreich und da man sie auch später noch für vieles andere mehr einsetzen konnte, wurden die Methode trotz ihres sehr unrühmlichen Ersteinsatzes doch ziemlich berühmt.

Die Monte-Carlo Methode lässt sich zum Beispiel ganz vorteilhaft dazu einsetzen, einen virtuellen Geiger-Zähler zu bauen, der Zählraten mit der richtigen Statistik als Anzeigewerte produziert. In anderen Worten: mit der Monte-Carlo-Methode kann man Zufallsprozesse mit beliebiger Verteilungsfunktion zu simulieren. Viele Rechenprogramme stellen einen Zufallsgenerator bereit, der Zahlen zwischen 0 und 1 gleichverteilt erzeugen kann. Wenn nun aber ein Prozess, eine ganz bestimmte Verteilung erfordert, so wie beispielsweise der

radioaktive Zerfallsprozess einer Poisson-Verteilung gehorchen soll, dann hat man erstmal das Problem, dass die mit gleichverteilten Zufallszahlen erzeugten Clicks nicht so schön ticken wie ein echter Geiger-Zähler. So ist beispielsweise der Mittelwert eines solchen gleichverteilten Zufallsgenerators immer 0.5 und die Streuung $1/\sqrt{(12)}$. Das ist beim Poisson-Prozess oder negativ exponentiell verteilten Zeitintervallen zwischen den Clicks nicht der Fall. Wie kommt man nun per Simulation auf das echte Ticken oder eine realitätsgerechte Zählraten-Anzeige eines Geigerzählers, was man in der Fachsprache auch einen Poisson-Prozess nennt? Die Monte-Carlo Methode bietet eine Lösungsmöglichkeit dafür. Man füttert einfach die inverse Verteilungsfunktion des gewünschten Prozesses mit gleichverteilten Zufallszahlen. Und schon entstehen als Ergebnis dieser Rechnung Zufallszahlen mit der gewünschten Statistik. Für einen virtuellen Geiger-Zähler mit Zählraten-Anzeige braucht man also nur die inverse Verteilungsfunktion der Poisson-Verteilung. Wenn man die analytisch nicht finden kann, dann kann man die Inversion mit dem Rechner per Tabelle erreichen. Was dabei hilft ist, dass es sich eh um eine diskrete Verteilungsfunktion handelt.

Veranschaulichen kann man sich das wie folgt: Eine Verteilungsfunktion F(x) hat als Ergebnis immer Werte zwischen 0 und 1. Die Eingangsgröße bewegt sich dagegen immer im Wertebereich der Zufallsvariablen x, welche durch die Verteilungsfunktion beschreiben wird. Invertiert man nun die Verteilungsfunktion, hat man als Eingangsgröße die Wahrscheinlichkeit 0 bis 1 und als Ergebnis den zugehörigen Wert der Zufallsgröße x. Erzeugt man nun als Eingangsgrößen gleichverteilte Zufallszahlen, dann sind logischerweise die zugehörigen Ergebnisgrößen Zahlen, die nach der benutzten Verteilungsfunktion verteilt sind. Und wenn das eben die Verteilungsfunktion die eines Poisson-Prozesses ist, dann sind die entstehenden Zufallszahlen Poisson-verteilte Zählraten, wie bei einem echten Geiger-Zähler. Selbst das typische Ticken eines Geigerzählers kann man so nachbilden. Da die Zeitabstände zwischen den Clicks negativ exponentiell verteilt sind muss man eben diese Verteilung invertieren und mit gleichverteilten Zufallszahlen zwischen 0 und 1 in die entstandene Funktion reingehen.

So ist die Verteilungsfunktion der Exponentialverteilung gegeben durch:

$F(x) = 1-\exp(-\lambda x)$ für $x >= 0$, und 0 sonst

Dabei ist der Mittelwert $\mu = 1/\lambda$ (die mittlere zeit zwischen den Clicks), wenn λ die mittlere Zählrate ist.

Die invertierte Funktion dazu ist:

$x = -\log_n(1-F)/\lambda$

Bildhaft gesprochen macht man die Monte-Carlo Simulation nun so, dass man mit den zwischen 0 und 1 gleichverteilten Zufallszahlen auf der Ordinate (y-Achse) der Verteilungsfunktion reingeht und das Ergebnis an der Abszisse (x-Achse) abliest. Das entspricht dann der Abbildung an der inversen Verteilungsfunktion. Um nun die Sache in einem Rechenprogramm durchzuführen, ist es bei der negativ exponentiellen Verteilungsfunktion

natürlich so, dass man die inverse Funktion leicht angeben kann und damit dann auch das Ergebnis „in einem Schuss“ erhält. Wenn man also für F der Reihe nach zwischen 0 und 1 verteilte Zufallszahlen einsetzt, dann ergibt $-\log_n(1-F)/\lambda$ auch der Reihe nach negativ exponentiell verteilte Zufallszahlen für die Zeitintervalle zwischen den Clicks eines virtuellen Geiger-Zählers und $1/\lambda$ ist die der Aktivität im Mittel proportionale Zählrate.

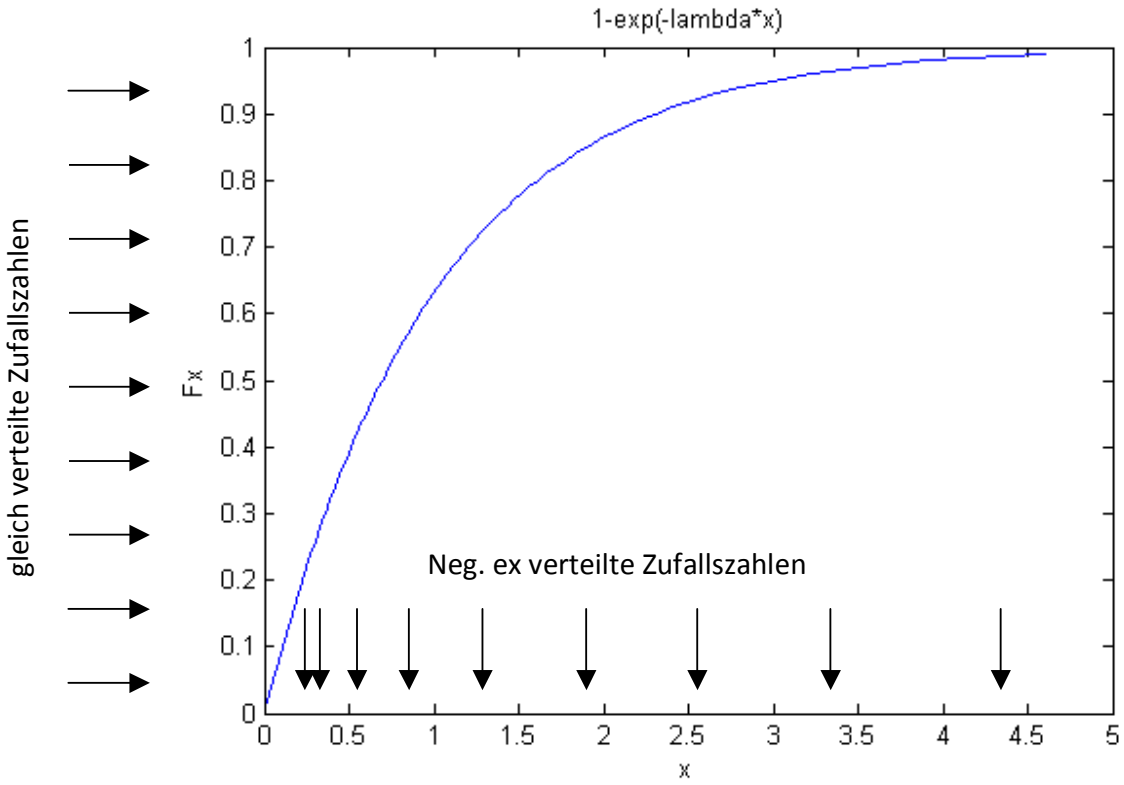

Abb. 1: Graphische Darstellung der Erzeugung beliebig verteilter Zufallszahlen aus gleichverteilten Zufallszahlen bei der negativ exponentiellen Verteilung des Radioaktiven Zerfalls

In Matlab beispielsweise würde so eine Simulation wie folgt aussehen, wenn man 10000 Zeitintervalle zwischen den Clicks eines Geigerzähler bei einer Rate von 1 pro Zeiteinheit simulieren wollte:

```
clear;
nSamp=100000;
lambda=1;
Fx=rand(1,nSamp);
x=-log(1-Fx)/lambda;

nClass=1000;
```

```
[Hx, xout] = hist(x,nClass);
dx=xout(2)-xout(1);
plot(xout, Hx/dx/nSamp, '-');
hold on;
x=0:0.1:10;
plot(x, lambda*exp(-lambda*x),'r');
hold off;
```

Die Funktion rand() liefert die 100000 gleichverteilten Zufallszahlen als eindimensionale Matrix Fx, mit der man einfach in die inverse Verteilungsfunktion reingeht. Das Ergebnis x ist dann eine Matrix mit entsprechend vielen neg. exp. verteilten Zeitintervallen zwischen den Clicks. Der Rest dient der Anzeige der relativen Häufigkeit der ermittelten Zufallszahlen in einem Histogramm (blau) und dem Vergleich mit der theoretischen Verteilungsfunktion (rot) zur Kontrolle. Man kann nun die Übereinstimmung der Monte-Carlo Rechnung mit der statistischen Theorie direkt erkennen.

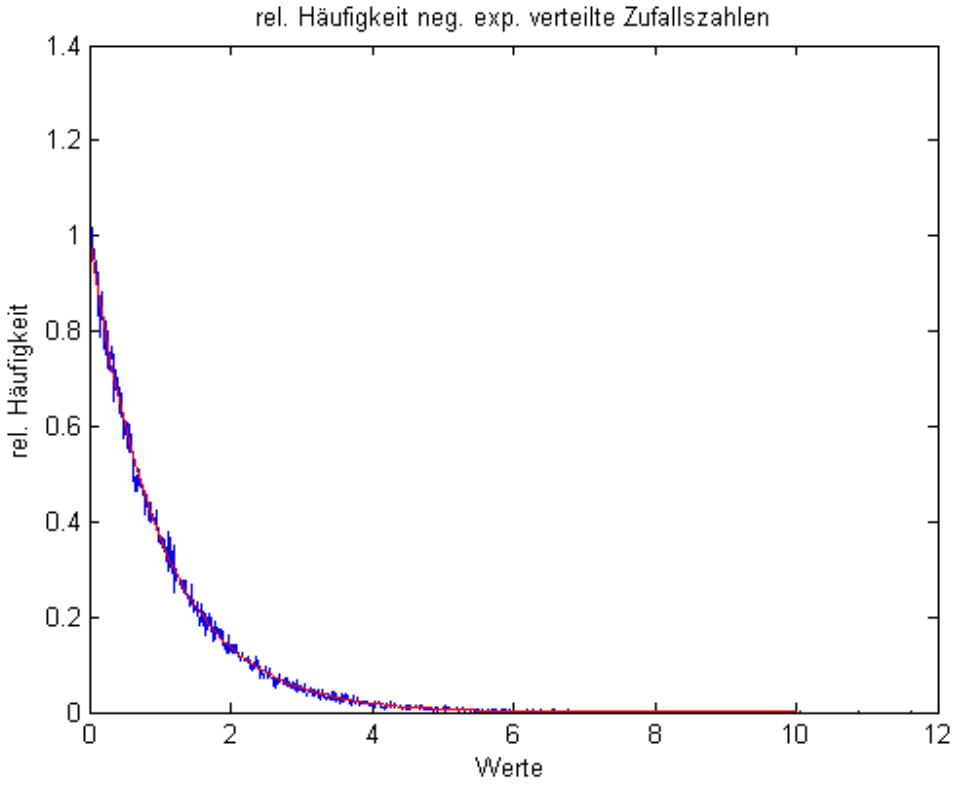

Abb. 2: Vergleich der mit der Monte-Carlo Methode erzeugten negativ exponentiell verteilten Zufallszahlen mit der statistischen Theorie

Die Simulation von Zählraten eines virtuellen Geigerzählers ist dagegen ein klein wenig aufwändiger, da die inverse Verteilungsfunktion der Poisson-Verteilung so einfach nicht analytisch darstellbar ist. Man kann sich aber damit helfen, dass man die normale

Verteilungsfunktion der Poisson-Verteilungsfunktion für die zu erwartenden Zählratenwerte berechnet und dann mit einer gleichverteilten Zufallszahl sucht, welche Zählrate eine Wahrscheinlichkeit hat, die der gleichverteilten Zufallszahl entspricht. Wenn die Verteilungsfunktionswerte in einer Matrix abgespeichert sind, dann stellen viele Rechenprogramme schnelle Suchfunktionen zur Verfügung, welche den Index der Matrix zurückliefern. Wenn man nun die zugehörigen Zählraten ebenfalls in einer gleich indizierten Matrix stehen hat, dann kann man mit dem gewonnen Index nun die zugehörige Zählrate ermitteln. Das folgende Matlab Programm zeigt diese Vorgehensweise. Zunächst wird die Poisson-Verteilungsdichte-Funktion erzeugt. Die normale Darstellung mit:

$P(k, \lambda) = \lambda^k/k!*\exp(-\lambda)$

macht gelegentlich bei der Berechnung großer k Probleme, deswegen wurde hier ein rekursiver Ansatz gewählt. Es gilt nämlich:

$P(k, \lambda) = \lambda/k * P(k-1, \lambda)$ mit $P(0, \lambda) = \exp(-\lambda)$

Sind die einzelnen Wahrscheinlichkeiten $P(k, \lambda)$ für das Auftreten für k Impulse in der Zeit T bei einer mittleren Anzahl von λ Pulsen im Zeitraum T, dann kann man die Verteilungsfunktion fK durch Summation dieser Einzel-Wahrscheinlichkeiten $pK = P(k, \lambda)$ berechnen. Das macht die Funktion cumsum() welche die kumulative Summe berechnet. Mit bar() lässt sich die korrekte Form der Verteilungsfunktion graphisch überprüfen.

Als nächstes werden die gleichverteilten Zufallszahlen berechnet und der eindimensionalen Matrix y zugeordnet. Danach wird in der for-Schleife der Index in der Matrix der Verteilungsfunktion gesucht, welcher den Wert fK als Wahrscheinlichkeit hat. Dieser Index wird in der Matrix über die Zählimpulse benutzt um K zu ermitteln. Im Prinzip hätte man hier auch vom ermittelten Index einfach 1 abziehen können, da bei der diskreten Verteilungsfunktion K einfach nur die Ganzzahlen von 0 bis Kmax sind. Jetzt kann man sich das Histogram anzeigen lassen und zur Kontrolle Mittelwert und Standardabweichung der Stichprobe berechnen. Da muss natürlich $m = \lambda$ und $s = \sqrt{(\lambda)}$ rauskommen, wenn es wirklich eine Poisson-Verteilung ist, und das tut es auch.

```
clear;
lambda = 9;
Kmax=30;
nSamp=100000;
K=0:Kmax;
pK=zeros(1,Kmax+1);
pK(1) = exp(-lambda);
for n=2:Kmax+1
    pK(n) = lambda/K(n)*pK(n-1);
```

```
end;
fK = cumsum(pK);
bar(K,pK);
y=rand(1,nSamp);
x=zeros(1,nSamp);
for i=1:nSamp
    idx=find(y(i)<fK,1,'first');
    x(i)=K(idx);
end
figure;hist(x,Kmax*5);
mean(x)
std(x)
```

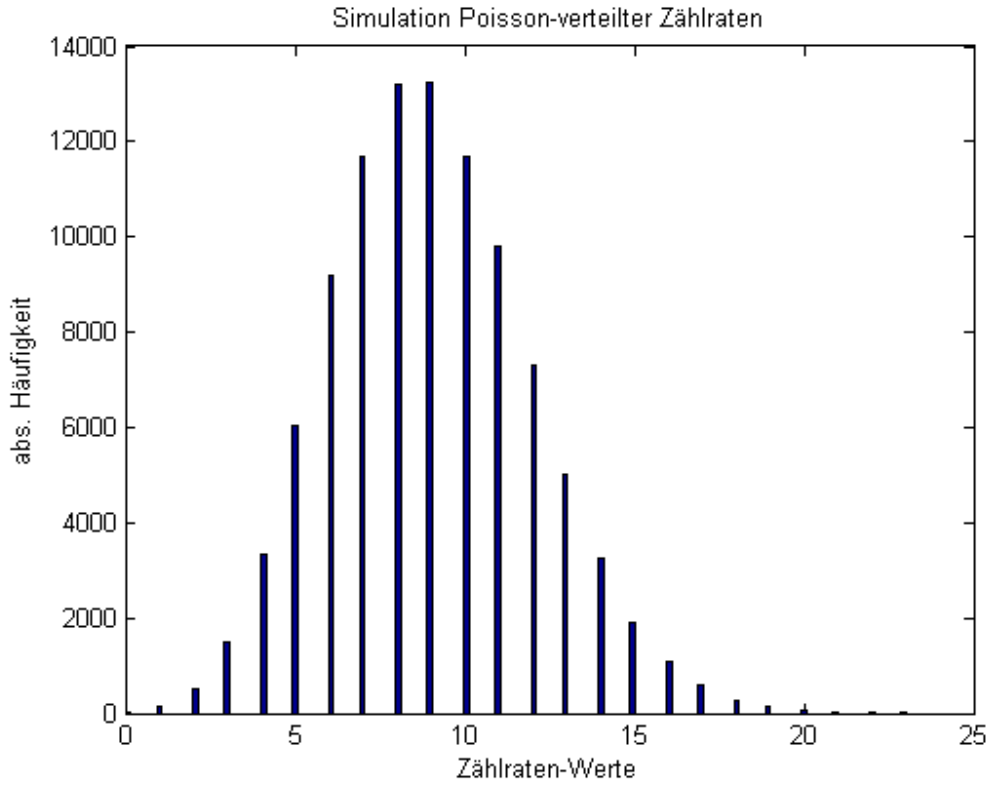

Abb. 3: Absolute Häufigkeiten von simulierten Zählraten für den Mittelwert 9

So und nun ist man praktisch am Ziel für eine virtuellen Geigerzähler angelangt. Wenn man sich nämlich das Zeitintervall mit einer Minute denkt, dann ist λ direkt die Zahlrate in Pulse pro Minute (counts per minute) oder cpm. Damit ist der Inhalt der eindimensionalen Matrix die Anzeige des virtuellen Geigerzählers in cpm in einer Abfolge von Messwerten bei konstanter Aktivität. Graphisch kann man das in Matlab mit der Funktion stem() anzeigen lassen. Und wenn man die Simulation gleich mit zwei unterschiedlichen Zählraten macht, dann sieht man diesen Zählraten-Sprung auch sehr deutlich bzw. statistisch signifikant, trotz der statistischen Unsicherheit wenn die Zählraten ausreichend auseinander liegen. Und was ganz deutlich ist, auch die Streuung ändert sich sichtbar. Die höhere Zählrate hat eben die höhere Streuung ganz gemäß dem Grundssatz der Poisson-Verteilung, dass gilt $\sigma = \sqrt{(\lambda)}$.

```
clear;
%Rate 1
lambda = 3;
Kmax=30;
nSamp=100;
K=0:Kmax;
pK=zeros(1,Kmax+1);
pK(1) = exp(-lambda);
for n=2:Kmax+1
    pK(n) = lambda/K(n)*pK(n-1);
end;
fK = cumsum(pK);
y1=rand(1,nSamp);
x1=zeros(1,nSamp);
for i=1:nSamp
    idx=find(y1(i)<fK,1,'first');
    x1(i)=K(idx);
end
%
%Rate 2
lambda = 36;
Kmax=90;
nSamp=100;
K=0:Kmax;
pK=zeros(1,Kmax+1);
pK(1) = exp(-lambda);
for n=2:Kmax+1
    pK(n) = lambda/K(n)*pK(n-1);
end;
fK = cumsum(pK);
bar(K,pK);
y2=rand(1,nSamp);
x2=zeros(1,nSamp);
for i=1:nSamp
    idx=find(y2(i)<fK,1,'first');
    x2(i)=K(idx);
end
x=[x1 x2];
stem(x);
```

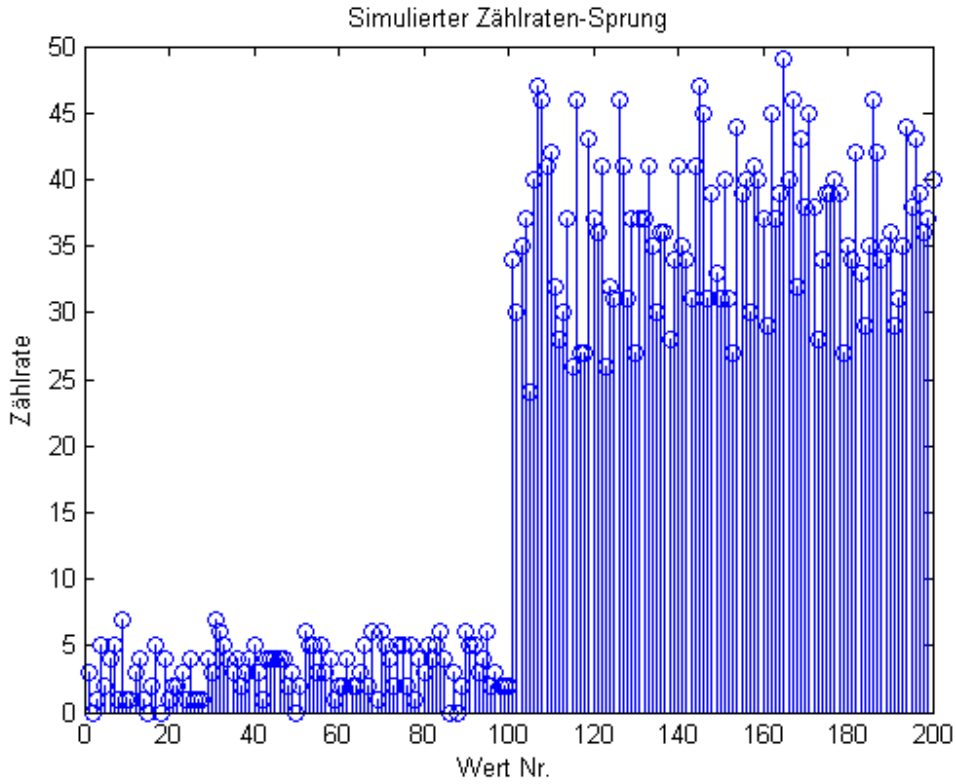

Abb. 4: Simulierter Zählratensprung von λ = 3 auf λ = 36,. Man erkennt deutlich wie auch die Streuung und damit die Unsicherheit mit der Zählrate zunimmt.

Im Anhang ist eine Simulation der Zählraten eines RD2007 Teviso Moduls im Strahlungsfeld einer radioaktiven Quelle mit 1uSv/h gegeben. Laut Spezifikation liefert es dann im Mittel eine Rate von λ = 3.4cpm. Gibt man diesen Wert in die Simulation ein erhält man beispielsweise eine solche Zählratenfolge für Messintervalle der Länge 1 Minute. Diese Folge ist dann auch echt Poisson verteilt.

Bei großen Zählraten > 50 kann man ohne viel Ungenauigkeit zu riskieren die Poisson-Verteilung durch eine Gauß-Verteilung ersetzen. Für die Gauß-Verteilung gibt es in vielen Rechenprogrammen wieder eine inverse Verteilungsfunktion, so dass das Programm wieder einfacher wird und die Simulation meist auch schneller abläuft. In Matlab kann man die inverse Verteilungsfunktion aus der Fehlerfunktion erf() ableiten. In Excel heißt die inverse Verteilungsfunktion der Gauss-Verteilung Norminv(). Als Parameter benutzt man jetzt $\mu=\lambda$ und berechnet σ zu $\sqrt{(\lambda)}$. Die genaue Lösung für den Gauß'schen Fall soll dem Leser zu Übung überlassen bleiben.

Literatur

Computing and the Manhattan Project
http://www.atomicheritage.org/history/computing-and-manhattan-project

Nicholas Metropolis; S. Ulam
The Monte Carlo Method
Journal of the American Statistical Association, Vol. 44, No. 247. (Sep., 1949), pp. 335-341.
www.rpi.edu/~angel/MULTISCALE/metropolis_Ulam_1949.pdf

Anhang

Simulation von 100 Poisson-verteilten Zählraten eines Teviso RD2007 Moduls bei 1uSv/h Dosis in cpm.

4	4	6	3
5	4	2	2
0	1	4	5
7	4	3	2
5	1	6	5
3	5	3	2
3	8	3	5
1	2	6	3
3	2	4	2
1	1	6	3
3	5	4	6
3	3	2	4
4	9	2	2
0	1	4	4
4	0	6	5
6	5	4	4
4	3	6	5
5	6	1	6
4	5	2	3
2	3	6	5
4	1	2	3
8	2	7	1
2	2	4	
2	1	1	
4	1	2	
2	1	3	