Hurst, Knuth, Mandelbrot, Marsaglia, Pareto, Poisson, Tausworthe, Weibull und Zipf, cache, chaos, file system, internet, self-similarity, server und traffic, Analysen, Experimente, Modelle, R, Simulationen, Wahrscheinlichkeiten und Zufall

—

eine Sammlung spröder Gedanken zur Statistik für Betriebsinformatik 2002/03

H. P. Wolf, pwolf@wiwi.uni-bielefeld.de

formatiert: February 13, 2009, file: stabi.rev
http://www.wiwi.uni-bielefeld.de/statistik/mitarbeiter/wolf/

Übersicht

1 Vorweg

Der umfangreiche Titel kündigt ein Sammelsurium an Blicken hinter verschiedene Kulissen an. Letztendlich bestand das Ziel darin, Interessantes aus dem Schnittbereich von Statistik und Informatik zu diskutieren und sich mit einer Vielzahl von Fragestellungen, Techniken, Analyse- und Modellierungsansätzen auseinanderzusetzen. Wie schon zu Beginn zu erwarten war, führte dieses Eintauchen nicht zu einer völlig runden Geschichte, jedoch wurden die Hörer hoffentlich durch eine Reihe für sie unbekannter, spannender Phänomene aus den berührten Gebieten entschädigt. Weiterhin enthält dieses Werk über 50 R-Code-Chunks, die zum großen Teil auch zu Hause wiederbelebbar sein müßten und zu weiteren Experimenten anregen sollen. Zur Technik siehe: `http://www.wiwi.uni-bielefeld.de/StatCompSci/software/revweb/revweb.html`

2 Problemfeld Internet-Traffic

2.1 Orientierungspunkte und der gewählte Weg

- Welche Historie besitzt die Veranstaltung *Statistik für BI?*

- Was heißt Studieren? Vorlesungen ←→ eigenes Tun
- Welche Bedeutung hat Statistik / Informatik für die Lösung betrieblicher Probleme?
- Welche Gliederungsprinzipien bieten sich an?
  - statistische Methoden
  - Prozeß: Problem → blackbox mit Hardware/Software → Lösung
  - Einzelproblemen
- Welche Gegenstände / Problemkreise lassen sich behandeln? Zuverlässigkeitsfragen, Übertragungsstörungen, Warteschlangenprobleme, Simulation, Zufallszahlen, Experimente, Stichprobenziehung, Implementierung komplexer Formeln, Optimierungsverfahren, Bedienung von Software, ...
- Entscheidung:

Vision: Diskussion anhand eines tragfähigen Problemkreises →

Es soll ein Zwischenspeicher (Cache) für Internetdokumente entwickelt werden. Hierdurch sollen die Ladezeiten gesenkt werden. Für die Entscheidung über Speichergröße, Geschwindigkeit und den einzusetzenden Algorithmus sind

- Beobachtungen über den Netzbetrieb zu sammeln und auszuwerten,
- Modelle über Zugriffs-, Zeitverhalten, Datenmengen und Zuverlässigkeiten zu konstruieren,
- mit denen sich dann Simulationsprogramme zu erstellen sind,
- welche Simulations-Experimente erlauben,
- deren Auswertung für den Problembereich Einsichten bringt.
- Für solche Simulationen sind Überlegungen zum Thema Stichprobenziehung
- und Experimental Design anzustellen.
- Zufallszahlengeneratoren und
- numerische Verfahren gehören zum Grundwissen.

2.2 Interessensgruppen

Verschiedene Personen stellen aufgrund ihrer Sicht unterschiedliche Fragen:

2.2.1 Dokumentersteller

- Wie häufig wird auf meine Seite zugegriffen?
- Wer greift auf meine Seite zu?
- Wie muß ich die Seite erstellen, damit sie oft gefunden wird?

Betrachten wir zum Beispiel die `www`-Page zur Vorlesung Statistik II. Bei dieser wurden in drei Monaten folgende Zugriffe beobachtet:

Monat	Jahr	Zugriffe
Juli	2002:	114
September	2002:	84
Oktober	2002:	45

Was sagen diese Zahlen? Es lassen sich sofort eine Reihe von Anschlußfragen stellen: Wie viele davon waren aus der Zielgruppe? Wie viele von netzabsuchenden Prozessen? Wie viele von Zufallsbesuchern? Wie viele durch das Lehrpersonal?

Auch ist es im Rahmen dieser Vorlesung interessant zu überlegen, was eigentlich zu tun ist, um solche Ergebnisse zu erhalten?

Tätigkeit	Tool
Protokolle definieren und erheben	
relevante Datensätze aus Protokollen extrahieren	`grep`
relevante Infos aus Datensätzen herausschälen	`awk`
Einbringung in statistische Umgebung	`R: x<-scan(...)`
Datenbereinigung / Ausreißer entfernen?	`boxplot(...)`
Datenanalyse	
Modellbildung: Verteilungsanpassung	
Interpretation	
Bericht	

2.2.2 Serververwalter

Aus Sicht des Serververwalters können folgende Fragen aufkommen:

- Wie viele Zugriffe gibt es pro Zeiteinheit?
- Wie sieht es aus mit der Zeit zwischen Zugriffen?
- Welche Mengen werden transportiert?
- Gibt es Strukturen in den Anfragen?
- Deutet etwas auf Fehlnutzungen hin?
- Welche Entwicklungen zeichnen sich ab?

Wir wollen als Beispiel Serverdaten von 1997 betrachten. Zur Extraktion von Zeiten, Mengen und Zwischenzeiten waren dazu eine Reihe von Schritten notwendig.

Damit Schritte des Vorgehens plastisch werden, soll an einem konkreten Beispiel eine Liste von Operationen gezeigt werden:

Tätigkeit	Umsetzung mit `R`
Rechner starten	
R starten	`library(rtrevive); r()`
Datensatz anzeigen:	`print(x<-zeitpunkte.03.02.1997)`
Datensatz plotten:	`plot(cbind(zeitpunkte.03.02.1997,1))`
Histogramm zeichnen:	`hist(zeitpunkte.03.02.1997)`
Modellanpassung: x	`xn<-seq(from=1340000,to=1480000,length=100)`
y	`yn<-dnorm(xn,mean(x),var(x)^0.5))`
Darstellung	`lines(xn,yn)`
Bericht speichern	
Bericht weiterverarbeiten	

Zusammen führt dieses zu der wiederbelebbaren Sequenz:

1 ⟨ * 1⟩ ≡

```
if((h<-exists("zeitpunkte.03.02.97"))){
  print(x<-zeitpunkte.03.02.97)
} else x <- cumsum(rexp(100,0.001))
par(mfrow=1:2)
plot(cbind(x,1))
title(if(h)"zeitpunkte.03.02.97"else"Zufallsdaten")
hist(x)
xn<-seq(from=1340000,to=1480000,length=100)
yn<-dnorm(xn,mean(x),var(x)^0.5)
lines(xn,yn)
par(mfrow=c(1,1))
```

2.2.3 Cache Designer

Natürlich steht der Kunde im Vordergrund. Wenn dieser eine Seite aus dem Internet haben möchte, sollte diese möglichst schnell zur Verfügung stehen. Die verlangte Seite sollte, wenn schon nicht mit Sicherheit, mit großer Wahrscheinlichkeit aus einem schnellen Zwischenspeicher (Cache) übermittelt werden. Für einen Cache-Designer folgt hieraus die zentrale Frage:

Nach welchem Prinzip sollen Seiten in den Speicher aufgenommen werden?

Nichts leichter als das! Wenn eine Seite nachgefragt wird, wird sie herangeholt und im Speicher abgelegt. Leider muß von einer maximalen Speichergröße ausgegangen werden. Deshalb dürfen nur ausgewählte Seiten gespeichert werden oder bzw. wird es erforderlich, gespeicherte Seiten zu entfernen.

Bewahre immer die zuletzt angeforderten Seiten!

Dieser Algorithmus (LRU — Least-Recently-Used replacement policy) bietet sich als einfach zu implementierende Idee an. Sicher werden durch dieses Vorgehen mit hoher Sicherheit die am häufigsten nachgefragten Seiten im Speicher festgehalten, aber sind damit alle Probleme gelöst? Genauere Überlegungen zeigen, daß die Entscheidung für einen konkreten Algorithmus durch verschiedene Punkte erschwert wird:

- Die Nachfrage ändert sich permanent. *Hot Pages* von heute müssen morgen nicht auch noch stark nachgefragt sein. Auch dürfte die Menge der Nachfrager im Tagesablauf variieren.
- Das Angebot ändert sich. Wenn Originalseiten verändert werden, ist die Version im Speicher nicht mehr aktuell. Von der Vorstellung her ist es naheliegend, daß zum Beispiel Nachrichtenseiten über allgemein interessierende Ereignisse aus Aktualitätsgründen auch eine hohe Änderungsgeschwindigkeit aufweisen.
- Es gibt sehr kleine und sehr große Seiten. Soll man lieber viele kleine Seiten festhalten oder eher weniger große Seiten?
- Die Geschwindigkeit des Netzes ist über den Tag nicht konstant. In Zeiten mit geringer Auslastung kann ein Transport von der Originalquelle durchaus akzeptabel sein als zu kritischen Zeiten.
- Es gibt verschiedene Möglichkeiten – Architekturen –, das Problem Zwischenspeicherung anzugehen.
- Neben Geschwindigkeitsfragen können Sicherheits- und Bedeutungsüberlegungen für die Art der Speicherung eine große Rolle spielen.

2.3 Einige griffige Fragen

Aus diesem ganzen Problemfeld wollen wir für die weiteren Betrachtungen ein paar griffige Fragen formulieren.

- Wie läßt sich die Nachfrage nach Seiten modellieren?
- Wie hängt die Trefferquote von der Anzahl der gespeicherten Seiten ab?
- Was läßt sich über die Zeit zwischen Seitenanforderungen sagen?
- Was läßt sich über verschiedene Politiken sagen?

3 On the Implications of Zipf's Law for Web Caching

Mit genau dieser Überschrift ist im Internet ein Artikel von Breslau, Cao, Fan, Phillips und Shenker zu finden, [3], [4]. Da er hier so schön hineinpaßt, ist der Titel als Kapitelüberschrift übernommen worden. Natürlich gehen auch viele der vorgestellten Gedanken auf die aufgelisteten Autoren zurück.

3.1 Einleitung

Wie üblich wird in dem Aufsatz zunächst das Problemfeld beschrieben:

- *Web proxy caching wird mit allgemeiner Traffic Zunahme interessanter.*

  Bevor wir mit der Aufzählung wichtiger Punkte fortfahren, wollen wir recherchieren, was man unter proxy *versteht:*

  EXKURS
  `What is a Web Proxy Server?`

  `A Web proxy server is a specialized HTTP server. The primary use of a proxy server is to allow internal clients access to the Internet from behind a firewall. Anyone behind a firewall can now have full Web access past the firewall host with minimum effort and without compromising security.`

  `The proxy server listens for requests from clients within the firewall and forwards these requests to remote internet servers outside the firewall. The proxy server reads responses from the external servers and then sends them to internal clients.`

  `In the usual case, all the clients within a given subnet use the same proxy server. This makes it possible for the proxy to cache documents efficiently that are requested by a number of clients.`

  `People using a proxy server should feel as if they are getting responses directly from remote servers.`

  (Quelle: [24])
  EXKURS-Ende

- *Fragen:*

  Wie sieht die Beziehung aus zwischen Trefferquote und Cache-Größe?

  Wie sieht die Beziehung aus zwischen Trefferquote und Anzahl von Anfragen und kurzfristige Ortsabhängigkeiten?

- *verschienene Studien zeigen bisher nicht erklärte empirische Beobachtungen:*

- *unendlicher Cache* $\Rightarrow$
  Trefferquote $\sim \log$*(Clientpopulation), Trefferquote* $\sim \log$*(Anzahl Anfragen)*
- *Trefferquote* $\sim \log$*(Cache-Größe)*
- *Wahrscheinlichkeit für erneute Dokument-Anfrage beim* k*-ten Versuch* $\sim 1/k$.

Relevant für: Caching Algorithmen Design, Caching Server Konfiguration.

- *zentrale Message:*

  Unter der Annahme, daß einzelne Anfragen voneinander unabhängig sind und sich die angefragten Seiten mit Zipf's Law beschreiben lassen, folgen verschiedene der beobachteten Eigenschaften. Es lassen sich also typische beobachtete Strukturen mit einem einfachen Modell rekonstruieren.

3.2 Trace Characteristics

Was genau unter Zipf's Law zu verstehen ist, wird im nächsten Unterkapitel beschrieben. Dieses Gesetz besitzt deshalb im vorliegenden Kontext eine Relevanz, weil verschiedene Studien aufzeigen, daß Zipf's Law zur Modellierung von Web-Anfragen geeignet ist.

Bei solchen Studien sind in den Modelldichten $f(x) = c/x^{\alpha}$ *als Exponenten* α *1, 0.97 oder auch etwas kleinere Werte (0.67) identifiziert worden, zur Modelldichte selbst* $\rightarrow$ *siehe unten.*

Geeignete Verteilungen für Web-Anfragen: $\sim 1/i$ *bzw.* $1/i^{0.97}$ *und* $1/i^{0.67}$.

Proxy Server Untersuchungen:

- *Web-Zugriffe über Proxy Server gemäß Zipf's Law mit* $\alpha \in [0.57, 0.67]$. *Je größer die User-Anzahl, umso kleiner* α.
- *Web-Zugriffe protokolliert von parent caches* $\alpha \approx 0.57$.
- *Keine Korrelation von Anfragehäufigkeit und Dokumentgröße.*
- *Einige heiße Dokumente werden oft geändert.*
- *Zur Verteilung heißer Seiten über die Landschaft der Server: Heiße Dokumente sind über das Web verteilt. Betrachten wir die ersten* x *heißen Dokumente, auf wie viele Servern* y *werden sich diese dann verteilen?* $\Rightarrow y = x^3/4$. *D.h., es gibt nicht den Web-Server für heiße Dokumente.*

Implikationen:

Strategie ersetze immer große Dokumente *ist nicht unbedingt brauchbar.*
Strategie behalte sehr populäre Dokumente *ist nicht unbedingt brauchbar.*
Im Web dominiert nicht ein einziger Server.

3.3 Zipf's Law

... ist jetzt schon mehrfach als Kandidat zur Modellierung erwähnt worden und muß erst einmal vorgestellt werden.

3.3.1 Historie

Der Soziologe George Kingsley Zipf hat Untersuchungen über Worthäufigkeiten in Texten angestellt, z.B.: James Joyce's Ulysses. [35]. Dabei stellte er fest, daß das Produkt aus Häufigkeit und Rang in der nach Häufigkeiten geordneten Reihe der Wörter konstant (C) ist. Es folgt das Zipfsche Gesetz:

Häufigkeit = C / Rang ⇒

$$P(i) = c/i \quad \text{mit} \quad c = (\sum_{1}^{n} 1/i)^{-1}.$$

Hierbei ist n die Anzahl der verschiedenen Wörter und c die normierende Konstante.

Die Häufigkeiten liegen also auf einer Hyperbel. Durch Logarithmieren der beiden Achsen entsteht eine Gerade. Denn:

$$f(x) = c/x \quad \wedge \quad y := \ln f(x) \quad \Rightarrow y = \ln f(x) = \ln(c/x) = -\ln x + \ln c$$

Tragen wir also y gegen $x^* = \ln x$ ab, muß sich eine Gerade mit Steigung -1 und Achsenabschntt $c^* = \ln c$ ergeben. Eine Verallgemeinerung ergibt sich durch den Einsatz eines zusätzlichen Parameters α, mit dem wir die schon oben angesprochene Form erhalten:

$$f(x) = c/x^{\alpha}$$

Als Erweiterung des Zipfschen Gesetzes hat Mandelbrot die Formel

$$P(i) = \frac{c}{(i+b)^a}$$

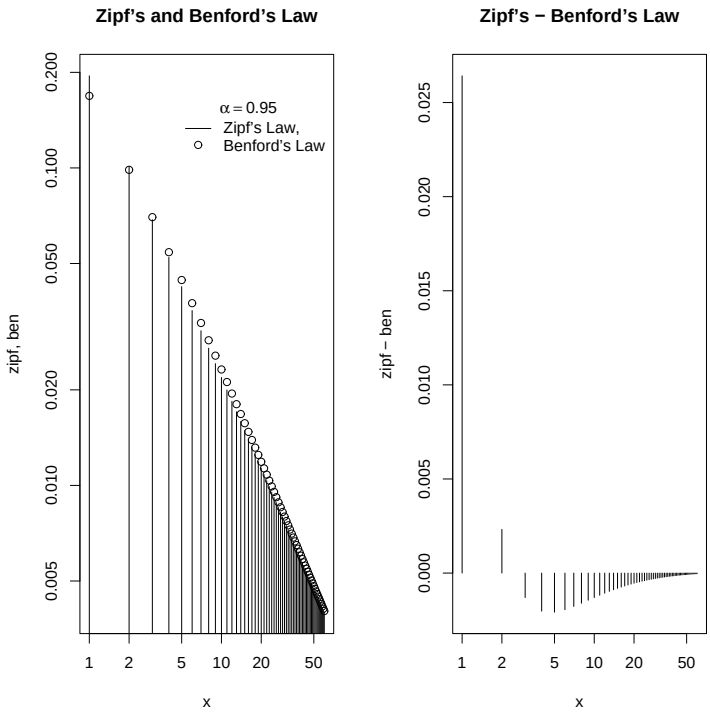
vorgeschlagen. Für $b = 0$ und $a = 1$ folgt die einfache Version.

Im weiteren Sinne vergleichbar ist das Zipfsche Gesetz mit Benford's Law [2]. Dieses beschreibt die Häufigkeit der ersten Ziffer von durch Zählprozesse ermittelte Zahlen, Preise u.ä. Dieses Gesetz geht auf Frank Benford zurück (1938), obwohl das Phänomen bereits 1881 von Simon Newcomb formuliert wurde. In diesem Zusammenhang hört man die berühmte Geschichte über verschmutzte Seiten von Logarithmentafeln, die jüngere Leser wahrscheinlich nur noch aus Erzählungen kennen. Interessant ist insbesondere die große Ähnlichkeit für höhere Ränge:

3.3.2 Experiment: Zipf's Law und Benford's Law

2 ⟨ * 1⟩+ ≡

```
x<-1:60; a<-alpha<-.95
zipf<-1/(x^alpha); zipf<-zipf/sum(zipf)
ben<-log(1+1/x,base=10); ben<-ben/sum(ben)
par(mfrow=1:2)
plot(x,zipf,type="h",log="xy",ylab="zipf, ben")
points(x,ben)
text(16,.155, substitute(alpha==a,list(a=a)))
legend(5,0.15, c("Zipf's Law,","Benford's Law"),
               lty=1:0, pch=c(26,1),bty="n")
title("Zipf's and Benford's Law")
plot(x,zipf-ben,type="h",log="x")
title("Zipf's - Benford's Law")
par(mfrow=c(1,1))
#man<-1/(x+c)^alpha
```



Übrigens kennt die Welt weitere Gesetze, auf die wir hier nicht eingehen wollen: Lotka's Law, Bradford's Law.

3.4 Zipf's Law zur Modellierung von Seitenanfragen.

3.4.1 Das Modell

Das Gesetz von Zipf wird auf die Fragestellung von Seitenanfragen übertragen.

Annahmen:

- *unabhängige Zugriffe,*
- $N :=$ *Anzahl der Dokumente im Universum,*
- *Dokumente sind gemäß Zugriffshäufigkeit von 1 bis N durchnumeriert.*

Dann folgt für die Wahrscheinlichkeit, daß das i-t populärste Dokument bei der nächsten Anfrage verlangt wird:

$$P_N(i) = \frac{\Omega}{i}$$

mit der Konstanten

$$\Omega = \left(\sum_{i=1}^{N} \frac{1}{i} \right)^{-1}$$

3.4.2 Experiment: Zipf's Law zur Modellierung von Seitenanfragen.

Wir wollen in diesem Abschnitt prüfen, ob unsere Serveranfragen sich an das Gesetz von Herrn Zipf halten.

Zunächst benötigen wir einen Strom von Serveranfragen. In einer Logdatei könnte zum Beispiel auftauchen:

```
xxx.hrz.uni-bielefeld.de - - [02/Mar/1999:20:23:25 +0100]
  "GET /~naeve/lehre/material_allgemein/latex7/sieben1.tex HTTP/1.0"
  200 13232
```

Die Einträge bedeuten:

`xxx.hrz.uni-bielefeld.de`	anfordernder Rechner
`02/Mar/1999:20:23:25 +0100`	Zeit der Anfrage
`GET`	Operation: hole Datei
`/~naeve/.../sieben1.tex`	Zieldatei
`HTTP/1.0`	Protokoll
`200`	Fehlercode: hier ok
`13232`	übermittelte Bytezahlen

Wie kommen wir an die harte Information? Die angeforderten html-Seiten in der Anforderungsreihenfolge erhalten wir mittels:

```
cat log.datei | gawk '{print $7; next}' |grep "tml$" > pages
```

Die Menge der angeforderten Seiten liefert:

```
cat pages | uniq > pagesuniq
```

Nun können wir mit R einen Vektor der angeforderten Seiten erstellen, Indizes gemäß ihrer Häufigkeit vergeben und graphisch die Angemessenheit von Zipf's Law überprüfen.

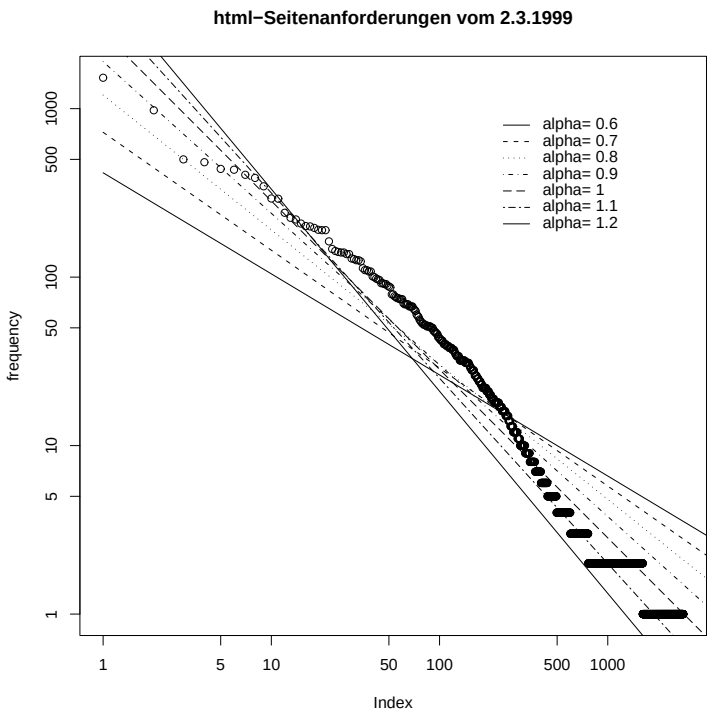
Vor der Verarbeitung wurden zur Anonymisierung die angeforderten Seiten gegen Zahlen ausgetauscht:

3 ⟨*anonymisiere angeforderte Seiten* 3⟩ ≡

```
pages    <- scan("pages","")
N.pages  <- length(pages)
pagesuniq<- scan("pagesuniq","")
N        <- length(pagesuniq)
NO<-match ( pages, pagesuniq)
cat(NO,file="pages",sep="\n")
cat(1:N,file="pagesuniq",sep="\n")
```

4 ⟨* 1⟩+ ≡

```
pages    <- scan("pages","")
N.pages  <- length(pages)
pagesuniq<- scan("pagesuniq","")
N        <- length(pagesuniq)
freq <- rev(sort(table(match(pages , pagesuniq))))
plot(1:N,freq,log="xy",ylab="frequency")
alpha.set<- ((6:12)*.1)
for(i in 1:length(alpha.set)){
  alpha<-alpha.set[i]
  omega<-sum(  (1:length(index))^(-alpha) ) ^(-1)
  lines(1:N.pages, N.pages*omega/(1:N.pages)^alpha,lty=i )
}
legend(200,1000, paste("alpha=",alpha.set),
lty=1:length(alpha.set), bty="n")
title("html-Seitenanforderungen vom 2.3.1999")
cat("Seitenanzahl:",N,"\nangeforderte Seiten:",N.pages,"\n")
```



Wir erhalten eine erstaunlich gute Anpassung.

Für die persönliche Vorstellung ist ein Experiment interessant, an dem man sehen kann, daß bei ganz anderen Nachfrageerzeugungsprozessen nicht unbedingt etwas mit dem Zipfschen Gesetz Vergleichbares herauskommt. Deshalb folgender Chunk als Randbemerkung, in dem aus einer zu wählenden Grundgesamtheit Zufallszahlen gezogen, gerundet und als Dokumentnamen interpretiert werden. Zu diesen ausgewählten Namen werden dann die Häufigkeiten absteigend sortiert in einer Graphik mit logarithmischen Achsen dargestellt.

5 ⟨ * 1⟩+ ≡

```
N.pages <- 1000
choice <- menu(c("Normal","Exponential","beta"))
switch(choice,
  gg<-"rnorm(N.pages,mean=5000,sd=10)",
  gg<-"rexp(N.pages,.5)",
  gg<-"200*rbeta(N.pages,2,2)"
)
pages <- as.character(floor(eval(parse(text=gg))))
pagesuniq<- names(table(pages))
N<-  length(pagesuniq)
freq <- rev(sort(table(match(pages , pagesuniq))))
plot(1:N,freq,log="xy",ylab="frequency")
omega<-sum(  (1:length(index))^(-1) ) ^(-1);omega
lines(1:N.pages, N.pages*omega / (1:N.pages) )
title(paste(gg,", N.pages:",N.pages))
```

Wir sehen, daß sich fallende konvexe Kurven einstellen und keine linearen Verläufe.

3.5 Trefferquote bei unendlicher Cache-Größe

3.5.1 Modellfall: Unendlicher Cache, endlicher Nachfragestrom

Diese Annahme ist zwar unrealistisch, jedoch läßt sich so eine Vorstellung über die Abhängigkeit der Trefferquote von der Anzahl der gespeicherten Seiten unter Modellbedingungen entwickeln.

Annahmen:

- *unendlicher Cache*
- *endlich viele Anfragen, bisher R*
- *$N :=$ Anzahl der Webseiten*
- *$P_N(i) :=$ Wahrscheinlichkeit, daß als nächste Seite i verlangt wird*
- *$H(R) :=$ Wahrscheinlichkeit, daß nächste Seite im Cache.*

Dann folgt für die Wahrscheinlichkeit $H(R)$, daß die nächste Anfrage aus dem Cache befriedigt werden kann:

$$H(R) = \sum_{i=1}^{N} P_N(i) \left(1 - (1 - P_N(i))^R\right)$$

Warum?

$$\begin{aligned}
P_N(i) &= P(\text{Seite } i \text{ gewünscht}) \\
(1 - P_N(i)) &= P(\text{Seite } i \text{ nicht gewünscht}) \\
(1 - P_N(i))^R &= P(\text{Seite } i \text{ bei } R \text{ Zugriffen nicht gewünscht}) \\
(1 - (1 - P_N(i))^R) &= P(\text{Seite } i \text{ bei } R \text{ Zugriffen} \\
&\qquad \text{mindestens einmal gewünscht}) \\
P_N(i)(1 - (1 - P_N(i))^R) &= P(\text{Seite } i \text{ nach } R \text{ Zugriffen im Cache} \\
&\qquad \text{sowie } i \text{ beim nächsten Versuch gewünscht}) \\
\textstyle\sum_{i=1}^{N} P_N(i) \left(1 - (1 - P_N(i))^R\right) &= P(\text{neu angeforderte Seite im Cache })
\end{aligned}$$

Es folgt, falls $P_N(i)$ sich durch das Zipfsche Gesetz beschreiben läßt:

$$H(R) = \sum_{i=1}^{N} \Omega/i \left(1 - (1 - \Omega/i)^R\right)$$

Dieses soll sich durch $\Omega \ln(R\Omega)$ approximieren lassen sowie noch besser durch $\bar{H}(R) = \Omega \ln(R)$.

Betrachten wir $f(i) = (1 - (1 - \Omega/i)^R)$, dann gilt für $1 \ll R \ll N$ und $i = R\Omega$:

$$f(R\Omega) = (1 - (1 - \Omega/R\Omega)^R) = (1 - (1 - 1/R)^R) \to e^{-1}$$

Für kleine i links von $R\Omega$ wird $f(.)$ groß werden, darf aber nicht über 1 wachsen. Für große i wird $(1 - \Omega/i)$ groß, jedoch nicht größer als 1, so daß sich als Grenze für $f(.)$ der Wert 0 einstellt:

$$f(i) \approx \begin{cases} 1, & i \leq R\Omega \\ 0, & otherwise \end{cases}$$

Wenn dieses nicht zu grob ist, folgt:

$$H(R) = \sum_{i=i}^{N} \frac{\Omega}{i} f(i) \approx \sum_{i=i}^{R\Omega} \frac{\Omega}{i} \approx \Omega \ln R\Omega$$

Verbesserungsvorschlag: $\bar{H}(R) = \Omega \ln R$.

Optisch scheint dagegen ein anderer Faktor besser zu passen.

3.5.2 Experiment: Approximation von $H(R)$

In diesem Experiment wollen wir optisch überprüfen, wie nahe die Approximationen für $H(R)$ der Funktion $H(R)$ kommen, wenn $P_N(i)$ durch das Zipfsche Gesetz beschrieben wird. Zuerst berechnen wir die Auswirkungen auf die Trefferrate:

6 ⟨* 1⟩+ ≡

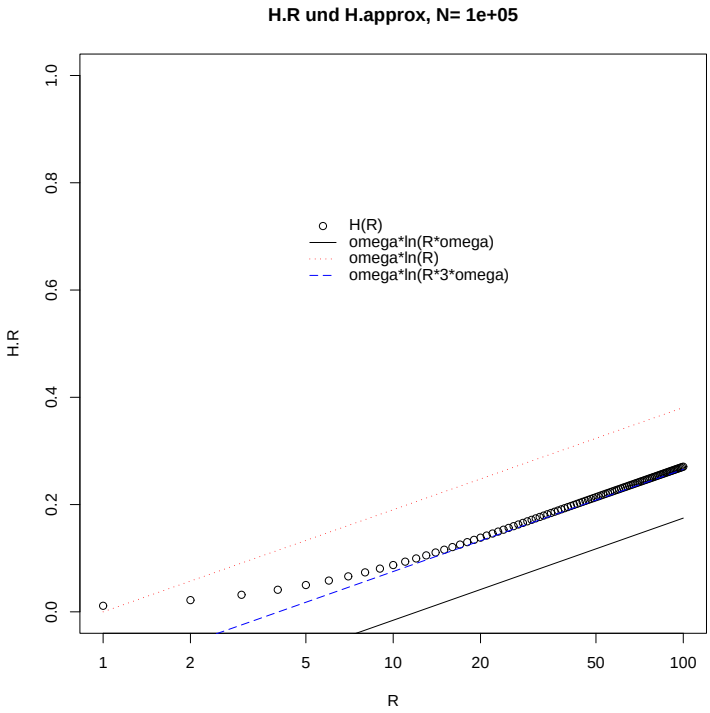
```
N<-100000
R<-(1:100)
i<-1:N
omega<-sum(1/i)^(-1)
p.n<-omega/i
fi<-(1-outer(1-p.n,R,"^"))
H.R<-apply(matrix(p.n,N,length(R)) * fi , 2, sum)
"Berechnung von H(R) beendet"
```

Dann läßt sich schnell eine graphische Veranschaulichung erzielen, indem wir $H(R)$ wie auch die Aprroximationen zeichnen.

7 ⟨* 1⟩+ ≡

```
plot(R,H.R,log="x",xlim=c(1,100),ylim=c(0,1))
lines(R,omega*log(R*omega),lty=1)
lines(R,omega*log(R),col="red",lty=3)
lines(R,omega*log(R*3*omega),col="blue",lty=5)
legend(5,0.75, c("H(R)","omega*ln(R*omega)","omega*ln(R)",
                 "omega*ln(R*3*omega)", ),
       col=c("black", "black","red","blue"),
       lty=c(0,1,3,5), pch=c(1,30,30,30), bty="n")
```

```
title(paste("H.R und H.approx, N=",N))
```

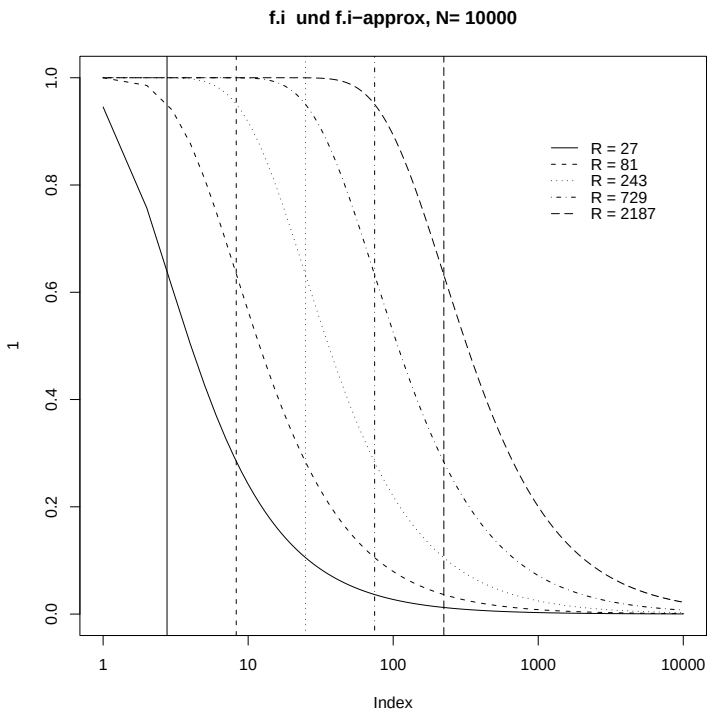


3.5.3 Experiment: Approximation von $f(i)$

Wir wollen die Approximation noch näher untersuchen. Dazu wird die Approximation von $f(i) = (1 - (1 - \Omega/i)^R)$ wie beschrieben durch An-Aus-Funktionen veranschaulicht:

8 $\langle * 1 \rangle + \equiv$

```
N<-10000; base<-3
R<-base^(3:floor(log(N/base,base=base)))
i<-1:N
omega<-sum(1/i)^(-1)
p.n<-omega/i
fi<-(1-outer(1-p.n,R,"^"))
plot(1,type="n",log="x",xlim=c(1,N),ylim=c(0,1))
for(j in 1:ncol(fi)){
  points(i,fi[,j],type="l",pch=j,lty=j)
}
abline(v=R*omega,lty=1:30)
legend(N/10,.9, paste("R =", R),lty=1:5, bty="n")
title(paste("f.i  und f.i-approx, N=",N))
```

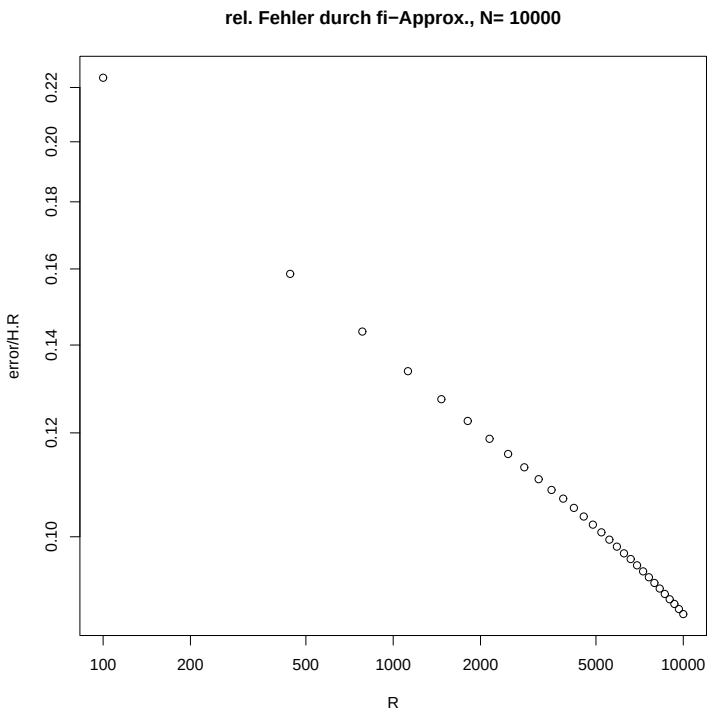


Immerhin liegen die Stellen, an denen die An-Aus-Funktion von 1 auf 0 wechselt immer in derselben Höhe der entsprechenden Funktion $f(i)$.

Betrachten wir außerdem noch den relativen Fehler:

9 ⟨ * 1⟩+ ≡

```
N<-10000
R<-seq(from=N/100,to=N,length=30)
i<-1:N
omega<-sum(1/i)^(-1)
p.n<-omega/i
fi<-(1-outer(1-p.n,R,"^"))
pn.fi<-matrix(p.n,N,length(R)) * fi
H.R   <- apply(pn.fi , 2, sum)
error <- apply(pn.fi*outer(i,R*omega,">" ), 2, sum)
plot(R,error/H.R ,log="xy")
title(paste("rel. Fehler durch fi-Approx., N=",N))
```



3.5.4 Experiment: Unendlicher Cache, endlicher Nachfragestrom

Unser Modell lieferte eine Formel für die Trefferhäufigkeit. Mit Hilfe der Ersetzung von Wahrscheinlichkeiten durch beobachtete Häufigkeiten wollen wir $H(R)$ in einer zweiten Experimentreihe schätzen.

Vorbereitung:

10 ⟨* 1⟩+ ≡

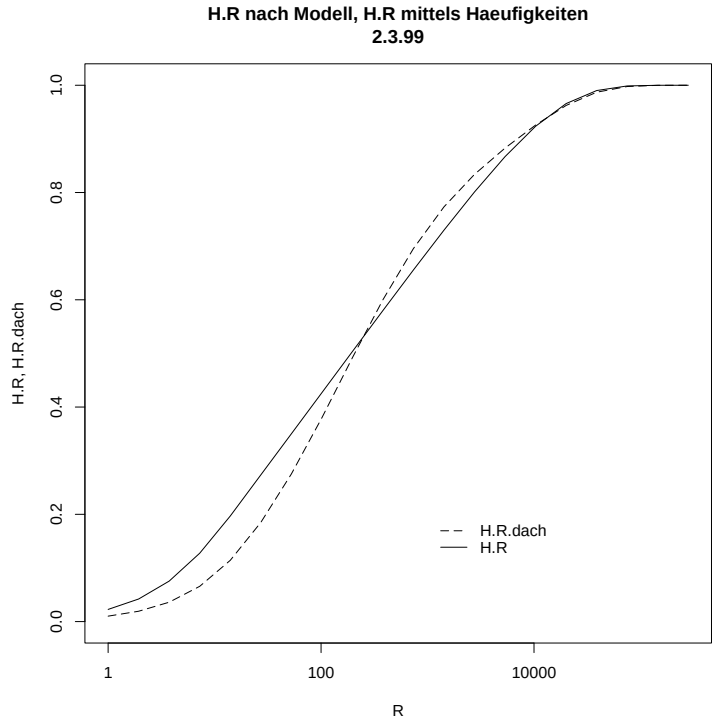
```
pages    <- scan("pages","")
N.pages  <- length(pages)
pagesuniq<- scan("pagesuniq","")
N        <- length(pagesuniq)
```

Umsetzung und Graphik:

11 ⟨* 1⟩+ ≡

```
# empirischer Teil
freq <- rev(sort(table(match(pages , pagesuniq))))
P.N.dach <- freq/sum(freq)
R <- exp(seq(from=0,to=log(100*N),length=20))
pn<-matrix(P.N.dach,N,length(R))
summanden<-pn*(1-outer(1-P.N.dach,R,"^"))
H.R.dach <- apply(summanden,2,sum)
plot(R,H.R.dach,type="l",lty=5,ylim=c(0,1),log="x",
     ylab="H.R, H.R.dach")
# theoretischer Teil
```

```
i<-1:N; omega<-sum(1/i)^(-1); p.n<-omega/i
fi<-(1-outer(1-p.n,R,"^"))
H.R<-apply(matrix(p.n,N,length(R)) * fi , 2, sum)
lines(R,H.R,lty=1)
# Komplettierung
legend(1000,.2,c("H.R.dach","H.R"),lty=c(5,1), bty="n")
title("H.R nach Modell, H.R mittels Haeufigkeiten\n2.3.99")
```



Jetzt noch die Überlegung, die Trefferquoten $H(R)$ aus den Daten heraus zu bestimmen. Diese Frage gehen wir in zwei Schritten an. Zuerst unterstellen wir eine Zugriffsverteilung, die mit Zipf's Law übereinstimmt.

Wir setzen die Zahl der Dokumente `N` fest und generieren eine Grundgesamtheit `gg`, in der die Seite i grob mit den relativen Häufigkeiten $1/i$ vertreten ist. Für den Plot wollen wir die Menge `R.set` der zu untersuchenden `R`-Werte betrachten. Die geschätzten Wahrscheinlichkeiten werden auf `H.R.dach` gesammelt. Für jedes `R` sollen `anz.wd` Wiederholungsexperimente durchgeführt werden. In jedem dieser Experimente wird eine Realisation eines Cache mit `R` Elementen gewählt. Dann wird für `anz.anfragen` Anfragen die relative Trefferhäufigkeit berechnet. Der Mittelwert der Trefferhäufigkeit wird auf `H.R.dach` abgelegt.

12 ⟨ * 1⟩+ ≡

```
# Vorbereitung
N<-100; size<-1000
```

```
gg<-rep(1:N,floor(size*(1/(1:N))))
R.set<-unique(floor(exp(seq(from=0,to=log(size),length=20))))
H.R.dach<-NULL; anz.anfragen<-50; anz.wd<-40

# Umsetzung
for(R in R.set){
  result<-NULL
  for(j in 1:anz.wd){
    dok.cache<-sample(gg,size=R,           replace=T)
    anfragen <-sample(gg,size=anz.anfragen,replace=T)
    result <-c(result,sum(anfragen %in% dok.cache)/anz.anfragen)
  }
  H.R.dach<-c(H.R.dach,mean(result))
}

# Plot
plot(R.set,H.R.dach,type="b",log="x",xlab="R")
title("Simulation von H.R, Zipf-verteilte GG")
```

Nun können wir als letzten Schritt dieser Versuchsreihe statt der künstlichen die beobachtete Situation vom 2.3.99 verwenden. Im Prinzip brauchen wir im Code nur die künstliche gegen die konkrete Grundgesamtheit austauschen.

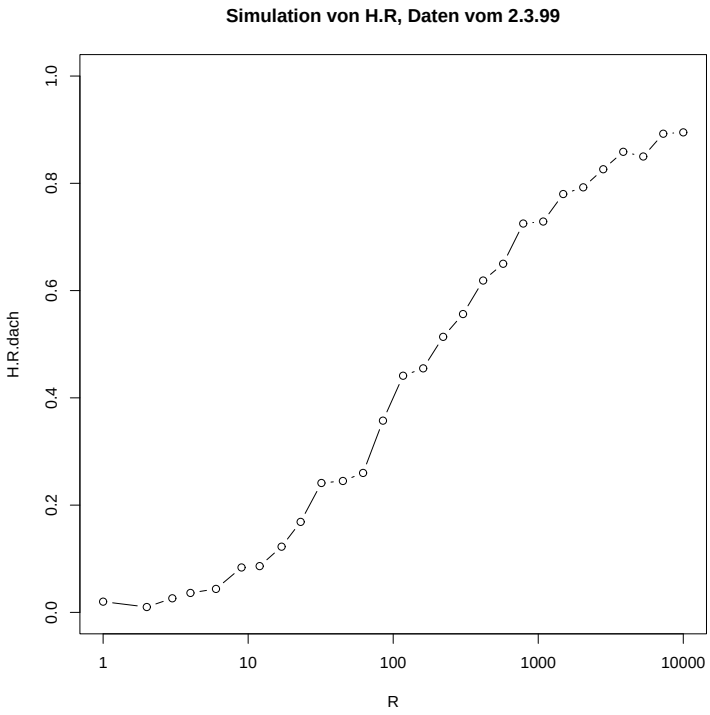
Notwendige Verbesserung hier und an verschiedenen anderen Stellen: Explizite Festlegung der ZZ-Generatorparameter!

13 ⟨ * 1⟩+ ≡

```
# Vorbereitung
gg   <- scan("pages","")
size <-5000
R.set<-unique(floor(exp(seq(from=0,to=log(size),length=20))))
H.R.dach<-NULL; anz.anfragen<-20; anz.wd<-40

# Umsetzung
for(R in R.set){
  result<-NULL
  for(j in 1:anz.wd){
    dok.cache<-sample(gg,size=R,           replace=T)
    anfragen <-sample(gg,size=anz.anfragen,replace=T)
    result <-c(result,sum(anfragen %in% dok.cache)/anz.anfragen)
  }
  H.R.dach<-c(H.R.dach,mean(result))
}

# Plot
plot(R.set,H.R.dach,type="b",ylim=0:1,log="x",xlab="R")
title("Simulation von H.R, Daten vom 2.3.99")
```



Damit sind die Experimente zur Trefferquote abgeschlossen.

3.6 Trefferquote bei endlicher Cache-Größe

Finite Cache, Infinite Request Stream. *Einsichtiger ist ein endlicher Speicher. Über die Zeit läßt sich ein unendlicher Strom von Abfragen vorstellen.*

Frage: In welcher Weise hängt dann die Trefferquote von der Cache-Größe ab?

Annahmen:

- $C :=$ *Cache-Kapazität*
- *Anfragestrom: unendlich*
- *Cache enthält die* C *am häufigsten nachgefragten Seiten*
- *Die Nummer einer Seite im Cache entspricht der Nachfrage — perfekter LFU Seitenaustausch ist damit angenommen (least frequently update).*

Gesucht: $H(C)$

Es gilt:

$$H(C) = \sum_{i=1}^{C} P_N(i) \approx \Omega \ln C$$

Denn mit dem Zipf's Law folgt:

$$H(C) = \sum_{i=1}^{C} \frac{\Omega}{i} = \Omega \sum_{1}^{C} \frac{1}{i} \approx \Omega \int_1^C t^{-1} dt = \Omega \ln t|_1^C = \Omega \ln C$$

Da die Darstellung von $H(C)$ uns nur ein Bild der log-Funktion zeigen würde, verzichten wir auf eine Darstellung.

3.7 Wahrscheinlichkeiten für Anfragewiederholungen

3.7.1 Modellfall: erneute Anfrage nach $k-1$ anderen

Zur Betrachtung der Zwischenzeiten wollen wir von folgenden Annahmen ausgehen:

- *unendlicher Strom eintreffender Anfragen*
- *$d(k)$:= Wahrscheinlichkeit, daß die nächste Anfrage nach der gerade angeforderten Seite nach k Versuchen stattfindet*
- *Anfragen unabhängig*
- *Anfragestruktur zeitunabhängig*
- *$P_N(i)$:= Wahrscheinlichkeit, daß bei einer Anfrage Seite i verlangt wird*

Dann folgt:

$$d(k) = \sum_{i=1}^{N} (P_N(i))^2 (1 - P_N(i))^{k-1}$$

Für $\alpha = 1$ erhalten wir approximativ:

$$d(k) \approx \frac{1}{k \ln N} \left(\left(1 - \frac{1}{N \ln N} \right)^k - \left(1 - \frac{1}{\ln N} \right)^k \right)$$

und weiterhin:

$$N \ln N \gg k \gg lnN \Rightarrow d(k) \approx \frac{1}{k \ln N}$$

Wieso? Für $P_N(i) = \Omega/i$ folgt:

$$\begin{aligned} d(k) &= \sum_1^N \left(\tfrac{\Omega}{i}\right)^2 \left(1 - \tfrac{\Omega}{i}\right)^{k-1} \\ &\approx \int_1^N \left(\tfrac{\Omega}{i}\right)^2 \left(1 - \tfrac{\Omega}{i}\right)^{k-1} di \end{aligned}$$

Mit $u := 1 - \Omega/i$ folgt $i = \Omega/(1-u)$ und $di = \Omega/(1-u)^2 du$. Hiermit erhalten wir:

$$\begin{aligned} d(k) &\approx \int_{1-\Omega}^{1-\Omega/N} (1-u)^2 u^{k-1} \tfrac{\Omega}{(1-u)^2}\, du \\ &= \int_{1-\Omega}^{1-\Omega/N} u^{k-1}\Omega\, du \\ &= \left.\tfrac{\Omega u^k}{k}\right|_{1-\Omega}^{1-\Omega/N} \\ &= \left(\left(1-\tfrac{\Omega}{N}\right)^k - (1-\Omega)^k\right)\tfrac{\Omega}{k} \\ &\approx \tfrac{1}{k\ln N}\left(\left(1-\tfrac{1}{N\ln N}\right)^k - \left(1-\tfrac{1}{\ln N}\right)^k\right) \end{aligned}$$

Jetzt läßt sich $N \ln N \gg k$ betrachten. Dann folgt für $\varepsilon = k/(N \ln N)$:

$$\left(1-\frac{1}{N\ln N}\right)^k = 1 - \binom{k}{1}\frac{1}{N\ln N} + \binom{k}{2}\left(\frac{1}{N\ln N}\right)^2 - \ldots \approx 1 - \varepsilon + 0.5\varepsilon^2 - \ldots \approx 1$$

Weiterhin gilt für $k \gg \ln N$:

$$\frac{1}{\ln N} \gg \frac{1}{k} \Rightarrow \left(1-\frac{1}{\ln N}\right) \ll \left(1-\frac{1}{k}\right) \Rightarrow \left(1-\frac{1}{\ln N}\right)^k \ll \left(1-\frac{1}{k}\right)^k \approx e^{-1}$$

Zusammen folgt:

$$N \ln N \gg k \gg \ln N \qquad \Rightarrow \qquad d(k) \approx \frac{\Omega}{k}$$

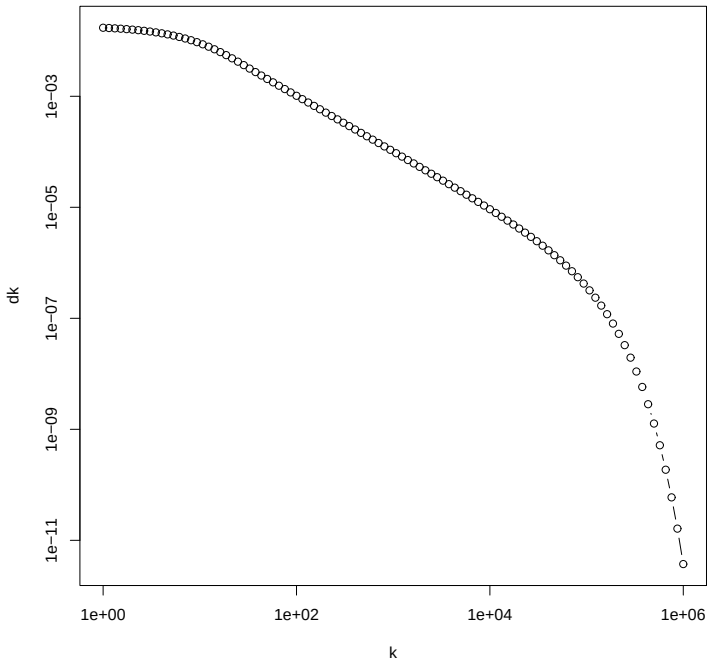
Soviel dazu.

3.7.2 Experiment: Darstellung von $d(k)$

In dieser Experimentreihe wollen wir die Graphiken aus dem hier zugrundeliegenden Artikel rekonstruieren. Zuerst interessiert uns die Funktion $d(k)$. Wie schon oben ist besonders der mittlere Bereich interessant. Auf diesen wollen wir bei der nun folgenden Zeichnung besonders achten, in der $d(k)$ der Anzahl der Anfragen k gegenübergestellt wird.

14 ⟨ * 1⟩+ ≡

```
N<-10000
i<-1:N
omega<-sum(1/i)^(-1)
p.n<-omega/i  # ^0.8
k<- exp(seq(from=0,to=log(100*N),length=100))
pnq<-matrix(p.n^2,N,length(k))
summanden<-pnq*outer(1-p.n,k-1,"^")
dk<-apply(summanden,2,sum)
plot(k,dk,type="b",log="xy")
```



3.7.3 Experiment: $d(k)$ geschätzt — WS für erneute Anfragen

In dieser Serie von Experimenten wollen wir $d(k)$ durch relative Häufigkeiten und direkt abschätzen.

Wie läßt sich die für $d(k)$ abgeleitet Formel absichern? Wir wollen dieses mit vier Experimenten tun. Erstens sollen für eine Modellgrundgesamtheit die Wahrscheinlichkeiten durch beobachtete Häufigkeiten ersetzt werden. Zweitens sollen Häufigkeiten aus den Seitenanfragen vom 2.3.99 ermittelt und eingesetzt werden, drittens soll $d(k)$ direkt aus einer Modellgrundgesamtheit und viertens direkt aus den Daten bestimmt werden.

Experiment 1: Es wird eine Grundgesamtheit definiert, eine Stichprobe gezogen und `P.N.dach` bestimmt. Dann kann die Formel umgesetzt werden. Da nicht alle Seiten unbedingt erwischt werden, wird Grundgesamtheit und `N` ggf. noch einmal korrigiert. Am Anfang werden die Größenverhältnisse denen der Daten vom 2.3.99 angepaßt.

15 $\langle * 1 \rangle +\equiv$

```
N<-2811; N.pages<-24130
gg<-rep(1:N,floor(size*(1/(1:N))))
stpr<-sample(gg,size=N.pages, replace=T)
gg<-unique(stpr); N<-length(gg)
freq <- rev(sort(table(match(stpr, gg))))
P.N.dach <- freq/sum(freq)
k.exp1<- exp(seq(from=0,to=log(100*N),length=40))
```

```
pnq<-matrix(P.N.dach^2,N,length(k.exp1))
summanden<-pnq*outer(1-P.N.dach,k.exp1-1,"^")
dk.exp1<-apply(summanden,2,sum)
plot(k.exp1,dk.exp1,type="b",log="xy")
```

Experiment 2: Im Prinzip wie oben mit realer Grundgesamtheit.

16 ⟨ * 1⟩+ ≡

```
stpr<-pages
gg<-unique(stpr); N<-length(gg)
freq <- rev(sort(table(match(stpr, gg))))
P.N.dach <- freq/sum(freq)
k.exp2 <- exp(seq(from=0,to=log(100*N),length=40))
pnq<-matrix(P.N.dach^2,N,length(k.exp2))
summanden<-pnq*outer(1-P.N.dach,k.exp2-1,"^")
dk.exp2<-apply(summanden,2,sum)
plot(k.exp2,dk.exp2,type="b",log="xy")
```

Experiment 3: Für den Fall, daß eine Stichprobe `stpr` vorliegt, zählen folgende Anweisungen, wie sich das Warten auf die Wiederkehr eines Elementes gemessen in Versuchen verteilt.

17 ⟨ * 1⟩+ ≡

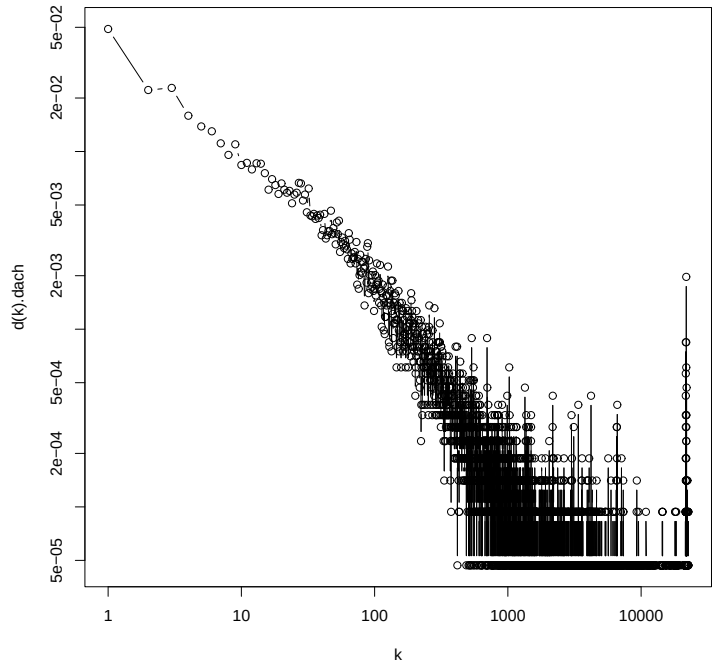
```
stpr<-sample(101:110,size=100,replace=T)
f.wd<-table(unlist(lapply(unique(stpr),function(x,y)diff(seq(y)[y==x]),stpr)))
f.wd<-f.wd/sum(f.wd)
plot(as.numeric(names(f.wd)), f.wd,
     log="xy", type="b", xlab="k", ylab="d(k).dach")
f.wd.exp3<-f.wd
```

Experiment 4: Im Unterschied zu Experiment 3 muß im Experiment 4 nur die Stichprobe durch das beobachtete Datenmaterial ausgetauscht werden. Für `stpr` muß man also nur `pages` einsetzen und schon ergibt sich die empirische Verteilung. Leider ist die Berechnung etwas länglich.

18 ⟨ * 1⟩+ ≡

```
stpr<-pages
f.wd<-table(unlist(lapply(unique(stpr),function(x,y)diff(seq(y)[y==x]),stpr)))
dk.exp4 <- f.wd/sum(f.wd)
k.exp4<-as.numeric(names(f.wd))
```

```
plot(k.exp4, dk.exp4, log="xy", type="b", xlab="k", ylab="d(k).dach")
```

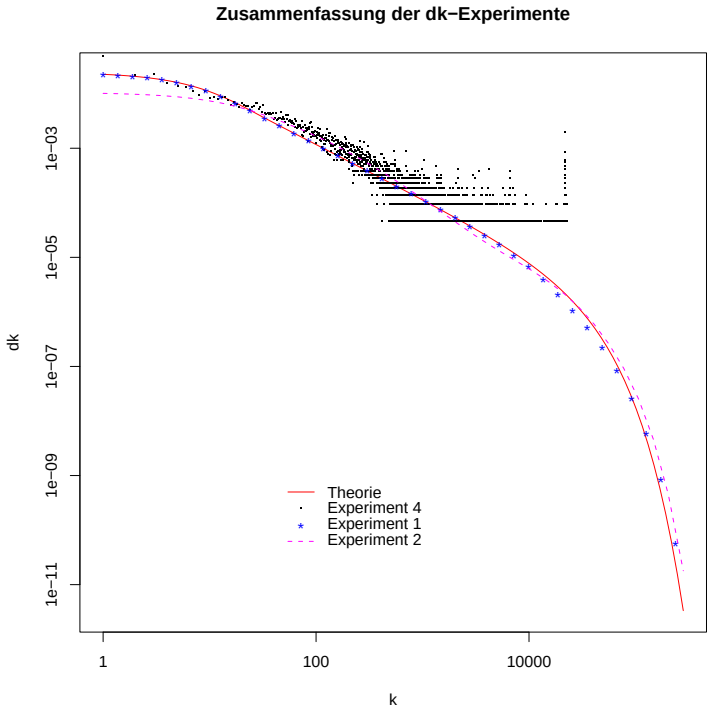


Zusammenfassung: In einem zusammenfassenden Bild werden die verschiedenen Berechnungen zusammengefaßt: `dk.exp4, dk.exp2, dk.exp1` und `dk.theo`.

19 ⟨ * 1⟩+ ≡

```
# theoretisch
N<-2811; N.pages<-24130
i<-1:N; omega<-sum(1/i)^(-1); p.n<-omega/i
k.theo <- exp(seq(from=0,to=log(100*N),length=100))
pnq<-matrix(p.n^2,N,length(k.theo))
summanden<-pnq*outer(1-p.n,k.theo-1,"^")
dk.theo<-apply(summanden,2,sum)
# Experiment theo
plot(k.theo,dk.theo,col="red",
     type="l",log="xy",xlab="k",ylab="dk")
# Experiment 4
points(k.exp4,dk.exp4,col="black",pch=".")
# Experiment 1
points(k.exp1,dk.exp1,col="blue",pch="*")
# Experiment 2
lines(k.exp2,dk.exp2,lty=2,col="magenta")
legend(50,1e-9,bty="n",
```

```
    c("Theorie","Experiment 4","Experiment 1","Experiment 2"),
 lty=c(1,        0,             0,              2),
 pch=c(" ",      ".",           "*",            " "),
 col=c("red",    "black",       "blue",         "magenta"))
title("Zusammenfassung der dk-Experimente")
```



Es ist doch erstaunlich, wie Welt und Wirklichkeit zusammenpassen — oder?

3.8 Fazit

- Nachfrage von Seiten hält sich an das Zipfsche Gesetz
- Trefferquote $\sim \ln(\text{Cache-Größe})$
- Im Zentrum gilt: Wiederanfragewahrscheinlichkeiten $\sim 1/k$
- Chunks zur empirischen Berechnung von $d(k), H(C), H(R)$ stehen für andere Serverprotokolle bereit.

4 Zwischenzeiten zwischen Anfragen

Wenden wir uns nun den Zwischenzeiten zwischen Anfragen zu. Je kürzer die Intervalle zwischen Anfragen, umso höher ist die Netzlast. Zur Frage der groben

Netzlast kommen wir sicher mit Mittelwerten aus. Für weitergehende Fragen, zum Beispiel nach der Häufigkeit von kurzfristigen Überlasten, benötigen wir geeignete Modelle. Deshalb macht es Sinn, nach passenden Ausschau zu halten.

4.1 Der Poisson-Prozeß als erstes Modell

Eine der einfachsten Modellierungen von Ereignissen im Zeitablauf liefert der Poisson-Prozeß. Mit der unmittelbar folgenden Poisson-Verteilung lassen sich verschiedene Situationen beschreiben: die Anzahl von Autos vor einer roten Ampel, die Anzahl der Rosinen in Kuchenstücken, die Anzahl der Druckfehler auf einer Seite. Weiter führt der Poisson-Prozeß zu einer einfachen Verteilung der Wartezeit bis zum nächsten Ereignis. Treffen in einem Zeitintervall Autos gemäß eines Poisson-Prozesses ein, so werden die Zwischenankunftszeiten durch eine Exponentialverteilung beschrieben.

So ist es auch nicht verwunderlich, daß die Exponentialverteilung zur Modellierung von Zwischenankunftszeiten ins Spiel kommt. Beispielsweise lesen wir bei Wenguang Wang:

> *Inter-arrival Time:*
>
> *Figure 1 shows the Probability Density Function inter-arrival time. The y-axis is in log-scale. Because the inter-arrival time in the trace is in microseconds (which is too small to be coped with), all arrivals within milliseconds* $[a - 0.5ms, a + 0.5ms)$ *are counted and assumed to occur at millisecond a. Then, the Probability Density Function is computed based on this reduced data set (which is in the unit of milliseconds).*
>
> *The resulted PDF is very close to a straight line with negative slope, which indicates that it follows an* Exponential distribution. *This is because the PDF of Exponential distribution is* $f(x) = \frac{1}{a}e^{-\frac{x}{a}}$. *Its equation in log-log scale is* $\log f(x) = -\frac{\log e}{a}x + (-\log a)$, *which is a straight line with negativ slope. To verify this hypothesis, the statistics of inter-arrival time are computed and listed in Table 4. The coefficient of variance (C.O.V.) is 1.18, which is close to the C.O.V of the Exponential distribution (which is 1).*

Mean	*Stdv*	*C.O.V*
15471	*18300*	*1.18*

Table 4: Characteristics of Interval arrival Time

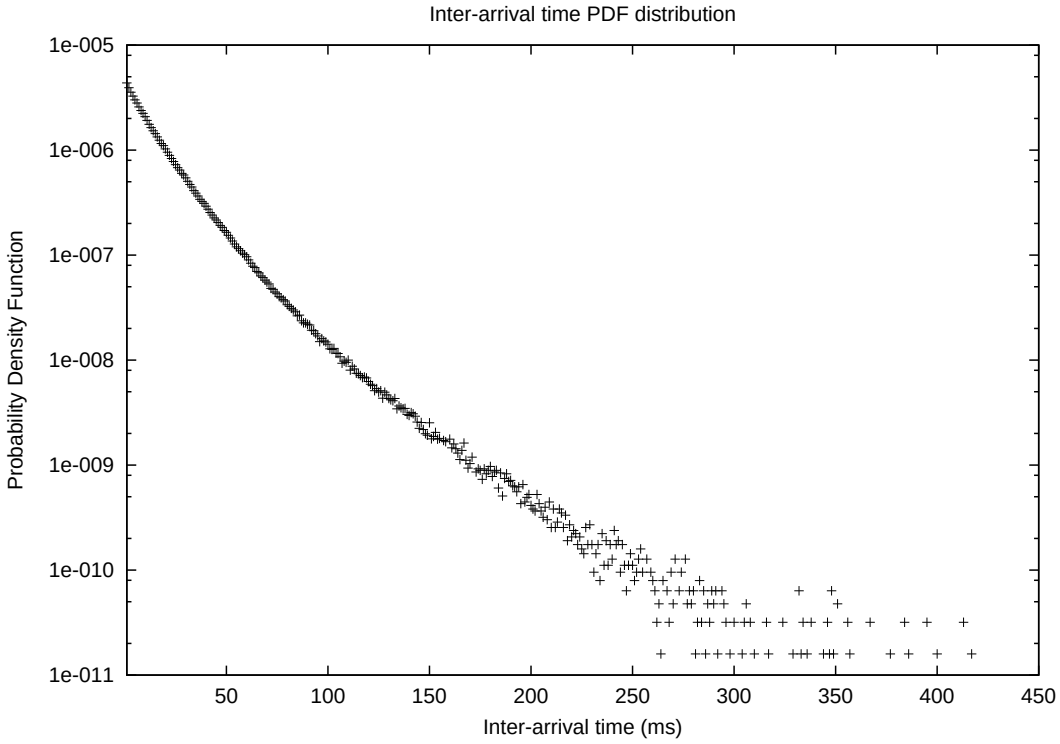


Figure 1: Probability Density Function of Inter-arrival Time

> *The exponentially distributed inter-arrival time means that there is little dependence between the arrivals of client requests. The reason for this is many clients send requests to the Web server independently.*
>
> [32]

Verstehen wir die Bemerkungen? Sind sie schlüssig? Und allgemein anwendbar? Um hierzu zu Antworten zu gelangen, wollen wir uns ein wenig mit dem Poisson-Prozeß auseinandersetzen. Zunächst gehen wir darauf ein, wie man überhaupt zur Poisson-Verteilung gelangt und welcher Zusammenhang zwischen Poisson-Verteilung und Exponentialverteilung existiert.

4.1.1 Von einigen Annahmen zur Poisson-Verteilung

Zum Nachschauen sei hingewiesen auf die Bücher von P. Naeve [19] und von Fisz [13] sowie auf die Internet-Seiten [14] und [29].

Postulate und Definition. Gelten für einen Prozeß, der im Zeitablauf Ereignisse hervorbringt, folgende vier Bedingungen:

1. Prozeßstruktur ist zeitinvariant

2. das Auftreten von Ereignissen ist in sich nicht überschneidenen Intervallen von einander unabhängig

3. für kleine (Zeit-) Intervalle ist die Wahrscheinlichkeit für genau ein Ereignis proportional zur Intervall-Länge

4. für kleine (Zeit-) Intervalle ist die Wahrscheinlichkeit für zwei oder mehr Ereignisse verschwindend klein,

dann ist die Wahrscheinlichkeit $P_n(t)$ für n Ereignisse im Intervall $[0, t]$ gegeben durch:

$$P_n(t) = \frac{(\lambda t)^n \cdot e^{-\lambda t}}{n!}$$

und der Prozeß heißt *Poisson-Prozeß*.

Formalisierung. Notwendig ist die Formalisierung der Bedingungen 3 und 4:

zu 3. $P_1(\Delta t) \approx \lambda \Delta t \Rightarrow P_1(\Delta t) = \lambda \Delta t + o(\Delta t)$.
zu 4. $\sum_{i>1} P_i(\Delta t) \approx 0 \Rightarrow \sum_{i>1} P_i(\Delta t) = o(\Delta t)$.

Mit einigen Fingerübungen folgen die Differentialgleichungen:

$$\begin{array}{lll} n = 0 & P_0'(t) = & -\lambda P_0(t) \\ n > 0 & P_n'(t) = & -\lambda P_n(t) + \lambda P_{n-1}(t) \end{array}$$

deren Lösung sich als Poisson-Wahrscheinlichkeitsfunktion mit dem Parameter λt entpuppt.

Probe. Zur Probe läßt sich überprüfen, ob die letzten beiden Bedingungen erfüllt sind.

Mit $\Delta t \to 0$ geht auch $e^{-\lambda \Delta t} \to 1$, und es folgt:

$$P_1(\Delta t) = \frac{(\lambda \Delta t)^1 \cdot e^{-\lambda \Delta t}}{1!} \approx \lambda \Delta t$$

sowie:

$$\sum_{i=2} P_i(\Delta t) = \sum_{i=2} \frac{(\lambda \Delta t)^i \cdot e^{-\lambda \Delta t}}{i!} = c \sum_{i=2} \frac{1}{i!} (\lambda \Delta t)^i$$

Der kleinste Polynomgrad dieser Summe ist 2, so daß sie für sehr kleine Δt viel schneller als linear gegen 0 geht.

Zu der Bedingung *Zeitinvarianz* läßt sich überlegen:

$$P_n(\Delta t) = P_n(n \text{ Ereignisse in } (t, t + \Delta t])$$

Zu der Bedingung der Unabhängigkeit überlegen wir: Falls diese Bedingung erfüllt ist, muß sich einerseits (A) die Wahrscheinlichkeit für n Ereignisse in

einem Intervall der Länge $(t+s)$ berechnen lassen mit der Poisson-Verteilung mit Parameter $\lambda(t+s)$. Andererseits muß sich dieselbe Wahrscheinlichkeit einstellen, wenn diese nach einer Zerlegung des Zeitintervalls über die Wahrscheinlichkeiten für Ereignisse in den beiden Intervallen berechnet wird. Bei dieser Sicht müssen sich bei n Erfolgen in $(0, t+s]$ ein Teil (i) in dem Teilintervall bis t und der Rest $(n-i)$ im verbleibenden Teilintervall realisieren.

Die Berechnung von A liefert:

$$P(n \text{ Ereignisse} \in (0, t+s]) = \frac{((s+t)\lambda)^n e^{-\lambda(s+t)}}{n!}$$

Die Berechnung von B liefert:

$$\begin{aligned}
&P(n \text{ Ereignisse} \in (0, t+s]) = \\
&= \textstyle\sum_{i=0}^{n} P(i \text{ in } (0,t] \text{ und } n-i \text{ Ereignisse} \in (t, t+s]) \\
&= \textstyle\sum_{i=0}^{n} P(i \text{ in } (0,t]) \cdot P(n-i \text{ in } (t, t+s]) \\
&= \textstyle\sum_{i=0}^{n} \frac{(t\lambda)^i e^{-\lambda t}}{i!} \cdot \frac{(s\lambda)^{n-i} e^{-\lambda s}}{(n-i)!} \\
&= \lambda^n e^{-\lambda(t+s)} \cdot \textstyle\sum \frac{t^i s^{n-i}}{i!(n-i)!} \\
&= \lambda^n e^{-\lambda(t+s)} \cdot \textstyle\sum \binom{n}{i} t^i s^{n-i} \cdot \frac{1}{n!} \\
&= \lambda^n e^{-\lambda(t+s)} \cdot (t+s)^n \cdot \tfrac{1}{n!} \\
&= \tfrac{((s+t)\lambda)^n e^{-\lambda(s+t)}}{n!}
\end{aligned}$$

Übrigens läßt sich auch zeigen, daß die Differenzialgleichungen erfüllt sind:

$$P_0'(t) = \left(e^{-\lambda t}\right)' = -\lambda e^{-\lambda t} = -\lambda \frac{(\lambda t)^0 \cdot e^{-\lambda t}}{0!} = -\lambda P_0(t)$$

und für $n > 0$ folgt:

$$\begin{aligned}
P_n'(t) &= \left(\tfrac{(\lambda t)^n e^{-\lambda t}}{n!}\right)' \\
&= \tfrac{1}{n!}\left(n(\lambda t)^{n-1}\lambda e^{-\lambda t} + (\lambda t)^n(-\lambda)e^{-\lambda t}\right) \\
&= \lambda P_{n-1}(t) - \lambda P_n(t)
\end{aligned}$$

4.1.2 Der Schritt zur Exponentialverteilung

Gegeben sei ein Poisson-Prozeß mit dem Parameter λ. Bezeichnet X_t die Anzahl der Vorkommnisse im Intervall $(0, t]$, dann ist X_t Poisson-verteilt mit dem Parameter λt. Wie ist unter diesen Bedingungen die Wartezeit T auf das nächste Ereignis verteilt? Formal formuliert ist $F_T(t) = P_T(T \leq t)$ gesucht.

Es gilt:

$$P_T(T \leq t) = 1 - P_T(T > t)$$

Der zweite Term läßt sich mit Hilfe der Zählvariable X_t des Poisson-Prozesses ausdrücken:

$$P_T(T > t) = P_{X_t}(0 \text{ Ereignisse in } (0,t]) = \frac{(\lambda t)^0 e^{-\lambda t}}{0!} = e^{-\lambda t}$$

Dieses führt zu der gesuchten Verteilungsfunktion:

$$P_T(T \leq t) = 1 - e^{-\lambda t} = F_T(t)$$

und durch Ableitung zu der Dichte

$$f_T(t) = \lambda e^{-\lambda t}.$$

Wenn es also so ist, daß Seiten gemäß eines Poisson-Prozesses angefragt werden, lassen sich die Zwischenankunftszeiten durch eine Exponentialverteilung beschreiben.

4.1.3 Eigenschaften der Exponentialverteilung

Momente: Es sei erinnert, daß der Erwartungswert der Exponentialverteilung durch $\mathrm{E}(X) = 1/\lambda$ gegeben ist:

$$\mathrm{E}(T) = \int_0^\infty t f_T(t)\, dt = \int_0^\infty t\lambda e^{-\lambda t}\, dt = \lambda \int_0^\infty t e^{-\lambda t}\, dt$$

Mit $(uv)' = u'v + uv'$ gilt $uv = \int u'v + \int uv'$ bzw. $\int uv' = uv - \int u'v$, so daß für $u = t$ und $v' = e^{-\lambda t}$ folgt $v = -e^{-\lambda t}/\lambda$ sowie

$$\mathrm{E}(T) = \lambda \left[t(-e^{-\lambda t}/\lambda)\big|_0^\infty - \int_0^\infty 1 \cdot (-e^{-\lambda t}/\lambda)\, dt \right] = \lambda \left[(0-0) - \frac{1}{\lambda^2} e^{-\lambda t}\Big|_0^\infty \right] = \frac{1}{\lambda}$$

Dieses Ergebnis ist plausibel, denn je größer die Rate der auftretenden Ereignisse pro Zeiteinheit λ ist, umso kürzer müssen die Wartezeiten ausfallen. Wird die Rate verdoppelt, müssen sich im Schnitt die Wartezeiten halbieren. Dieser einsichtige Zusammenhang spiegelt sich in der reziproken Beziehung von $\mathrm{E}(T)$ und $\mathrm{E}(X_1)$ wider. Ähnlich läßt sich die Varianz berechnen:
$\mathrm{Var}(T) = 1/\lambda^2$.

Gedächtnislosigkeit: Als weitere Eigenschaft sollen die *Gedächtnislosigkeit* der Exponentialverteilung hervorgehoben werden. Verbal beschrieben ist die Verteilung der restlichen Wartezeit gleich der ursprünglichen Wartezeitverteilung, auch wenn bekannt wird, daß bis zu einem Zeitpunkt t_0 kein Ereignis eingetreten ist.

Für $t > t_0$ ergibt sich:

$$\begin{aligned} P(T \leq t \mid T > t_0) &= 1 - P(T > t \mid T > t_0) \\ &= 1 - \tfrac{P(T>t)}{P(T>t_0)} \\ &= 1 - \tfrac{e^{-\lambda t}}{e^{-\lambda t_0}} \\ &= 1 - e^{-\lambda(t-t_0)} \\ &= F(t - t_0) \\ &= P(T \leq t - t_0) \end{aligned}$$

4.1.4 Identifikation der Exponentialverteilung

Bevor wir uns auf Daten oder andere Analysen stürzen, wollen wir noch kurz überlegen, wie wir prüfen können, ob für vorliegendes Datenmaterial die Exponentialverteilung ein geeignetes Modell darstellt. Folgende Vorschläge lassen sich verfolgen:

- Berechne Mittelwert und Standardabweichung aus der Stichprobe. Beide Werte müssen ungefähr gleich groß sein.
- Erstelle eine QQ-Plot, in dem die geordneten Daten gegen Quantile einer Exponentialverteilung geplottet werden. Die Punkte müssen auf einer Geraden liegen. Denn: $F(t) = 1 - exp(-\lambda t) \Rightarrow t = -\log(1 - F(t))/\lambda$. Die Prozentpunkte verschiedener Exponentialverteilungen erhält man also durch Umskalierung. Noch eleganter können wir wegen $-\log(1 - F(t)) = \lambda t$ gleich $-\log(1 - \hat{F}(t))$ gegen t abtragen und aus der Steigungsinformation λ schätzen.

Diese Ideen wollen wie gleich implementieren. Da eventuell Ausreißer – hier große Werte – stören können, definieren wir für die Funktion `identify.exponential` einen cut-off-Parameter, der $(1-$`cut.off`$) \cdot 100$ % der größten Werte aus den Daten entfernt. Als Default wird `cut.off = 4/5` gesetzt.

20 ⟨*definiere* `identify.exponential` 20⟩ ≡ ⊂ 79

```
identify.exponential<-function (x, cut.off = 4/5)
{
    n <- length(x)
    x <- sort(x)
    y <- -log(1 - ((1:n) - 0.5)/n)
    plot(x, y, xlab = "x", ylab = "-log(1-F.dach(x))")
    h <- floor(n * cut.off)
    r <- c(lsfit(x[1:h], y[1:h], intercept = F)$coef)
    if (length(r) == 1) r <- c(0, r)
    abline(r)
    if (length(r) == 2) r <- r[2]
    return(c(mean=mean(x),std.abw=var(x)^0.5,lambda.dach=r))
}
```

Mit diesen Überlegungen sind wir gut gerüstet, um weitere empirische Befunde zu erarbeiten.

4.1.5 Empirie: Ist die Exponentialverteilung als Modell geeignet?

Wie steht es mit den Serverdaten? Stellt dort die Exponentialverteilung ein geeignetes Modell dar?

Für die Daten von unserem Server vom 3.2.97 ergibt sich folgendes Bild:

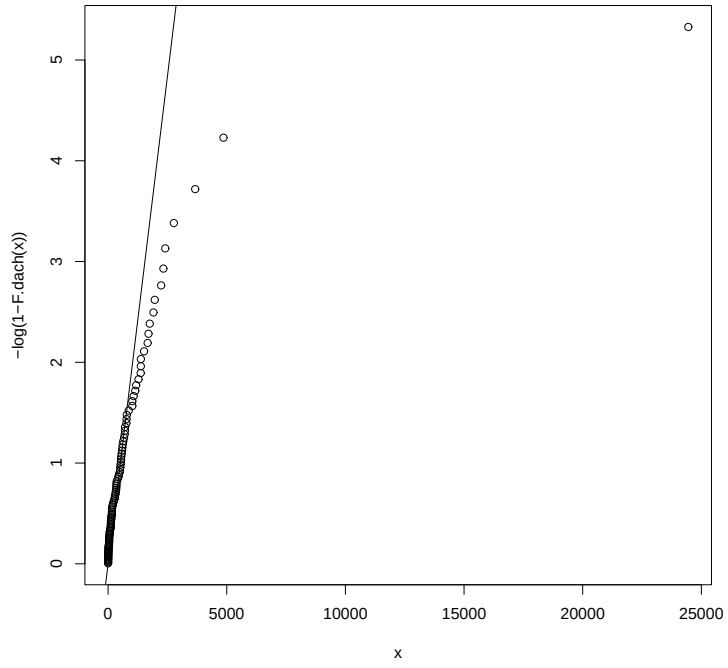
21 ⟨ * 1⟩+ ≡

```
cat("x <- dzeitpunkte.03.02.97\n")
x<-dzeitpunkte.03.02.97
identify.exponential(x)
```

Als Kenngrößen erhalten wir:

```
Fri Nov 15 18:56:40 2002
         mean       std.abw lambda.dach.X
 8.237961e+02  2.487414e+03  1.934472e-03
```

Offensichtlich sind Mittel und Standardabweichung sehr unterschiedlich. Die Diskrepqnz erzeugt verschiedene Bedenken. Hier ist die Graphik:



Ist die Graphik Ihrer Meinung nach überzeugend? Nicht so ganz. Denn erstens weichen eine Reihe von Punkten von der eingezeichneten Gerade ab und weisen auf einige besonders große Zwischenzeiten hin. Zum anderen ist bei der Schätzung der Geraden ein Teil der Daten ausgeblendet worden.

Bei Jin ist zu lesen:

> *General inter-arrival times can be generated by distributing the requests over the spanning time of the synthetic workload. If the requests are distributed uniformly, then general inter-arrival times roughly follows the exponential distribution. However, several studies have shown that streaming accesses exhibit diurnal patterns.*

> *... We call such phenomena* seasonal patterns, *i.e., there are, hourly, daily, and weekly patterns. Users are more likely to request streaming objects during particular periods, making a uniform distribution of requests over the spanning time of the synthetic workload unrealistic.*
>
> [16]

Im Prinzip läßt sich also die Exponentialverteilung als erstes Modell verwenden, jedoch sind verschieden intensive *Zeitphasen* zu erwarten. Dieses entspricht unserer Erwartung.

> *The conventional exponential model, established for telephone conversations, fails to accurately capture the packet level activity observed in these traces, e.g., the heavy-tail distributions of the talkspurt and silence periods.*
>
> [6]

Aha! Obwohl das Exponential-Modell eine gewisse Bewährung bei der Modellierung von Telefongesprächen erfahren hat, ist dieses auf der Paketebene nicht ganz der Fall. Dort werden Verteilungen mit dickeren Schwänzen benötigt, zum Beispiel Weibull.

> *We find that user-initiated TCP session arrivals, such as remote-login and file-transfer, are well-modeled as Poisson processes with fixed hourly rates, but that other connection arrivals deviate considerably from Poisson; that modeling TELNET packet interarrivals as exponential grievously underestimates the burstiness of TELNET traffic, but using the empirical Tcplib [Danzig et al, 1992] interarrivals preserves burstiness over many time scales; and that FTP data connection arrivals within FTP sessions come bunched into "connection bursts", the largest of which are so large that they completely dominate FTP data traffic.*
>
> [22]

Wir sehen, der Poisson-Prozeß ist für manche Dinge ganz gut geeignet, für andere zu einfach. Auf jeden Fall handelt es sich um einen verstehbaren Ausgangspunkt. Besonders auch wegen des Titels des zuletzt zitierten Artikels *Wide-Area Traffic: The Failure of Poisson Modeling* wollen wir ihn uns noch etwas näher anschauen bzw. ausschlachten.

4.2 Wide-Area Traffic: The Failure of Poisson Modeling

In diesem Aufsatz befassen sich Paxson und Floyd mit unterschiedlichen Detailfragen und Protokollen des Netzverkehrs. Positiv für die Freunde des Poisson-Prozesses ist:

> *We show that for TELNET connection arrivals and for FTP session arrivals, within one-hour intervals the arrival process can be well-modeled by a homogeneous Poisson process; each of these arrivals reflects an individual user starting a new session.*
>
> [22]

... dagegen ernüchtert wiederum:

> *Over one hour intervals, no other protocol's connection arrivals are well-modeled by a Poisson process. Even if we restrict ourselves to ten-minute intervals, only FTP session and TELNET connection arrivals are statistically consistent with Poisson arrivals, though the arrival of SMTP connections and of FTPDATA "bursts" (discussed later in §6) during ten-minute intervals are not terribly far from what a Poisson process would generate. The arrivals of NNTP, FTPDATA, and WWW (World Wide Web) connections on the other hand, are decidedly not Poisson processes.*
>
> [22]

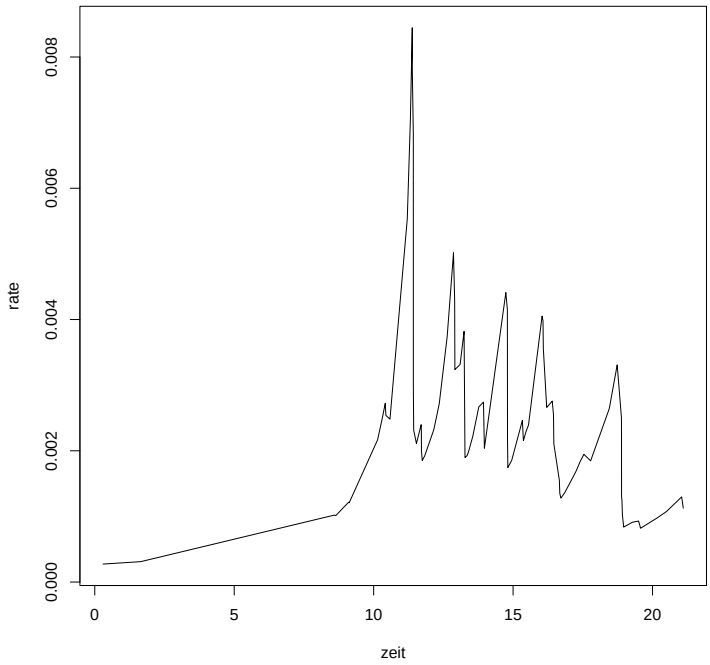
4.2.1 Traffic-Rate im Zeitablauf

Folgen wir den beiden Autoren, müßten wir uns anschauen, wie sich die Traffic-Rate im Zeitablauf entwickelt. Schätzen wir λ mittels $1/\bar{X}$, dann können wir im ersten Versuch einen folgenden Plot erstellen: x-Achse: reale Zeit, y-Achse: Kehrwert der jeweiligen Wartezeit auf das nächste Ereignis.

22 ⟨ * 1⟩+ ≡
```
plot(zeitpunkte.03.02.97, dzeitpunkte.03.02.97,type="l")
```

Leider wird der Eindruck durch das große Loch mitten in der Nacht gestört. Außerdem scheint eine gewisse Glättung angeraten zu sein.

23 ⟨ * 1⟩+ ≡
```
width<-10
n<-length(dzeitpunkte.03.02.97)
cumdz<-cumsum(dzeitpunkte.03.02.97)
y<-cumdz[(width+1):n]-cumdz[1:(n-width)]
rate<-(y/width)^-1
zeit<- (zeitpunkte.03.02.97[1:(n-width)] %% (24*3600)) / 3600
plot(zeit, rate,type="l",xlab="zeit")
```



Jetzt sehen wir, daß – wie gut auch immer die Schätzung ist – die Intensität der Zugriffe im Tagesablauf beträchtlich variiert. Hier ein homogenes Modell über den ganzen Tag anzunehmen grenzt an Unsinn.

> *Figure 1 shows enough daily variation that we cannot reasonably hope to model connection arrivals using simple homogeneous Poisson processes, which require constant rates. The next simple model is to postulate that during fixed-length intervals (say, one hour long) the arrival rate is constant and the arrivals within each interval might be well modeled by a homogeneous (fixed-rate) Poisson process. Telephone traffic, for example, is fairly well modeled during one-hour intervals using homogeneous Poisson arrival processes [FL91].*
>
> [22]

4.2.2 Ist Poisson-Prozeß als Modell geeignet?

Diese Frage ist so schön, weil man an ihr demonstrieren kann, wie man Schritt für Schritt in die Tiefen der statistischen Technik herabsteigen kann. Dabei muß man sich an verschiedene Konzepte der statistischen Grundausbildung wieder erinnern.

> *To evaluate these Poisson models, we developed a simple statisticaal methodology (Appendix A) for testing whether arrivals during a given one-hour or ten-minute interval are Poisson with a fixed rate.*

> *We test two aspects of each protocol's interarrivals: whether they are consistent with exponentially distributed interarrivals, and whether they are consistent with independent interarrivals.*
>
> [22]

Testen — wesentliche Aspekte. Die Zwischenankunftszeiten werden also auf Exponentialverteilung *getestet*. Testen — wie ging das noch?

1. Testen heißt, aufgrund von Beobachtungen über eine Hypothese zu entscheiden.
2. Die vorliegende Frage *Paßt die Exponentialverteilung?* führt uns zu Anpassungstests.
3. Der bekannteste ist der χ^2-Anpassungstest und würde von den meisten Studierenden bei einer entsprechenden Frage vorgeschlagen.
4. Mit dem Hinweis, daß die Klassenbildung zu Verlusten führt, wird als zweite Antwort *Kolmogorov-Smirnov* vorgeschlagen werden.
5. Paxson und Floyd verwenden jedoch einen *Anderson-Darling-Test*, der als Weiterentwicklung des Kolmogorov-Smirnov-Tests angesehen werden kann.
6. Besser heißt auch mehr *Power*. Die Power- oder Gütefunktion zeigt die Wahrscheinlichkeit, eine Beobachtung im kritischen Gebiet zu bekommen, also die Nullhypothese abzulehnen. Haben wir die Tests A und B mit gleichem α vorliegen, wobei Test A eine größere Power als Test B hat, dann gilt:
   - die Wahrscheinlichkeit ist bei beiden Tests gerade α, daß die Nullhypothese verworfen wird, falls die Nullhypothese korrekt ist.
   - Für den Fall, daß die Nullhypothese falsch ist, ist die Verwerfungswahrscheinlichkeit für A größer als für Test B. Der Test A erkennt also mit höherer Wahrscheinlichkeit, daß die Nullhypothese zurecht nicht zutrifft.

Wichtiger als die Powerfunktion dürfte die Frage sein, ob ein Test gerade die Blickrichtung umsetzt, die für die vorliegende Problematik wichtig ist, ob er also auf uns störende Phänomene reagiert. Als kritisch werden hier Abweichungen in dem langen Schwanz der Exponentialverteilung angesehen. Wir wollen mal sehen, wie der Anderson-Darling-Test damit umgeht.

4.2.3 Der Anderson-Darling-Test als G.o.f-Test-Baustein

Die Idee. Die folgenden Ausführungen sind bei D'Agostino und Stephens nachzulesen [8] Die Test-Statistik des Kolmogrov-Smirnov-Test ist gegeben durch:

$$D = \sup_x |F_n(x) - F(x)| = \max(D^+, D^-)$$

Hierbei sind: $F_n(x)$ empirische Verteilungsfunktion, $F(x)$ fragliche Verteilungsfunktion, D^+ maximale positive Abweichungen $F_n(x) - F(x)$, D^- maximale negative Abweichungen $F_n(x) - F(x)$. D zeigt also, wie stark empirische und Modellverteilungsfunktion maximal voneinander entfernt sind. Zwar wird nur eine einzige Stelle verwertet, jedoch verkörpert diese Stelle durch den Kumulations- oder Zählprozeß Eigenschaften der Gesamtverteilung.

Diese Statistik ist, wie in der Wissenschaft üblich, verallgemeinert und verfeinert worden. Zum Beispiel führt das Integral

$$Q = n \int_{-\infty}^{\infty} (F_n(x) - F(x))^2 \psi(x) f(x)\, dx$$

zu einer ganzen Klasse von Tests. Für $\psi(x) = 1$ ergibt sich die Cramér-von Mises-Statistik und für

$$\psi(x) = \frac{1}{F(x)(1 - F(x))}$$

die Anderson-Darling-Statistik:

$$A^2 = n \int_{-\infty}^{\infty} \frac{(F_n(x) - F(x))^2}{F(x)(1 - F(x))} f(x)\, dx$$

Offensichtlich zeigt uns A^2/n den Erwartungswert von $(F_n(x) - F(x))^2/(F(x)(1 - F(x))$, der groß ausfällt, wenn F als Modell unpassend ist. Da sich Verteilungsfunktionen zwischen 0 und 1 bewegen, ist $F(x)(1 - F(x))$ eine Funktion, die an der Stelle $F(x) = 0.5$ ihr Maximum besitzt. Der Kehrwert hat also dort sein Minimum, so daß diese Gewichtsfunktion Abweichungen in den Randbereichen einer Verteilung viel stärker bewertet als den mittleren Bereich.

Der Test. Die Operationalisierung der Statistik für einen Test erfordert einerseits eine (approximative) Berechnung von A^2, andererseits für die Entscheidung einen Vergleichs- oder Tabellenwert. Weiter kommt erschwerend hinzu, daß in der Regel Modellparameter geschätzt werden müssen. Das erste Problem wird durch Transformation der Beobachtungen mittels $Z = F(X)$ angegangen. Falls F das richtige Modell ist, wird Z gleichverteilt sein. Hierdurch läßt sich F aus der Formel entfernen und man erhält:

$$A^2 = -n - (1/n) \sum_i (2i - 1)(\ln Z_{(i)} + \ln(1 - Z_{(n+1-i)}))$$

sowie:

$$A^2 = -n - (1/n)\sum_i((2i-1)\ln Z_{(i)} + (2n+1-2i)\ln(1-Z_{(i)}))$$

Das Tabellenwertproblem wird gelöst mit einer Tabelle von D'Agostino und Stephens. Mit dem Hinweis, daß die Exponentialverteilung eine Weibull-Verteilung mit Shape-Parameter 1 ist, ist die Tabelle von Seite 146 brauchbar. Durch Zugriff auf *Case 2: scale parameter β unknown, but α known* ist auch das dritte Problem gelöst. Hier ist der Tabellenausschnitt, der für ein modifiziertes A^2 kritische Werte liefert:

Case 2: $A^* = A^2(1.0 + 0.6/n)$.

$\alpha =$	0.25	0.20	0.15	0.10	0.05	0.025	0.01	0.005	0.0025
A^*_α	.736	.816	.916	1.062	1.321	1.591	1.959	2.244	2.534

Was ist also zu tun?

Algorithmus:

1. wähle α, zum Beispiel $\alpha = 0.05$
2. schätze aus Beobachtungen den Parameter der Exponentialverteilung: der ML-Schätzer $\hat{\lambda} = 1/\overline{X}$ maximiert die Likelihoodfunktion:

   $$L(\lambda) = \prod_i f(x_i;\lambda) = \prod_i \lambda e^{-\lambda x},$$

   also: schätze λ mittels dem Kehrwert der durchschnittlichen Wartezeit
3. sortiere Beobachtungen: $x_{(1)} \le x_{(2)} \le \cdots \le x_{(n)}$
4. berechne $z_{(i)}$-Werte mittels der Transformation:

   $$z_{(i)} = F(x_{(i)};\hat{\lambda}) = 1 - e^{-\hat{\lambda}x_{(i)}}$$

5. berechne Realisation von A^2 nach:

   $$A^2 = -n - (1/n)\sum_i((2i-1)\ln Z_{(i)} + (2n+1-2i)\ln(1-Z_{(i)}))$$

6. berechne Modifizierte Statistik:

   $$A^* = A^2(1 + 0.6/n)$$

7. vergleiche A^* mit Tabellenwert, zum Beispiel mit 1.321
8. liefere Ergebnis ab.

Eine R-Umsetzung. Wir wollen mutig die beschriebene Prozedur umsetzen.

24 ⟨*definiere* ad.exp 24⟩ ≡ ⊂ 79

```
ad.exp<-function(x, alpha=0.05){
   # Anderson-Darling-Test fuer Exponential-Verteilung
   # Vgl. Ralph B. D'Agostino, Michael A. Stephens (1986):
   # Goodness-of-Fit Techniques, Marcel Dekker, New York, pp.100-101, 134
   # pw 11/02

   lambda<-1/mean(x); z<-1-exp(-lambda*sort(x))
   i<-1:(n<-length(z))
   Aq<- -n - 1/n * sum( (2*i - 1)       * log(    z)
                      +(2*n + 1 - 2*i) * log(1 - z) )
   Aq.mod <- Aq * (1+0.6/n)
   alpha.tab<- c(.25,  .20,  .15,  .10,  .05,  .025, .01,  .005, .0025,  0)
   A.tab <-    c(.736, .816, .916,1.062,1.321,1.591,1.959,2.244,2.534, Inf)
   A.krit   <- A.tab[length(A.tab)+1-sum(alpha <= alpha.tab)]
   return(c(Entscheidung= if(A.krit < Aq.mod) "Verwerfe Nullhypothese"
                     else "Nullhypothese wird nicht verworfen",
            lambda=lambda, Aq.mod=Aq.mod, alpha=alpha))
}
```

Zum Test können wir auf das Beispiel von Seite 137 zurückgreifen.

25 ⟨* 1⟩+ ≡

```
# D'Agostino/Stephens, p. 137
x<-c(12,21,26,27,29,29,48,57,59,70,74,153,326,386,502)
ad.exp(x)
```

```
Tue Nov 26 16:08:55 2002
                        Entscheidung                                lambda
"Nullhypothese wird nicht verworfen"      "0.00824628916987356"
                              Aq.mod                                 alpha
                  "1.20980231210351"                             "0.05"
```

4.2.4 G.o.f.-Test-Strategie mit Anderson-Darling-Test

Die Strategie. Nachdem wir unser Grundwissen aufgefrischt haben, wollen wir uns anschauen, wie Vern Paxson und Sally Floyd den Anderson-Darling-Test als Baustein einsetzen. Zunächst testen sie mit dem Anderson-Darling-Test, ob die Beobachtungen aus einem inhomogenen Poisson-Prozeß stammen können. Hier ihr Vorgehen in algorithmischer Form.

1. Bestimme eine Zeitintervall-Länge I, für das angenommen werden kann, daß die Ankunftsrate konstant ist.

2. Zerlege den zu betrachtenden Zeitraum T in N Intervalle der Länge I: $N = T/I$.

3. Führe in jedem Intervall wie oben beschrieben einen Anderson-Darling-Test zum Beispiel zum Niveau $\alpha = 0.05$ durch.

4. Falls die Daten einem Poisson-Prozeß entsprungen sind, wird der Test mit einer Wahrscheinlichkeit von 95 % zu keiner Ablehnung der Nullhypothese kommen. Genauer ist dann die Anzahl der Akzeptanzen binomialverteilt mit den Parametern $n = N$ und $p = 0.95$.

5. K sei die Anzahl der Zeitintervalle, die nicht zur Ablehnung geführt haben. Wenn diese Anzahl zu klein ist, werden wir stutzig. Oder: Ist die Wahrscheinlichkeit für die realisierte Anzahl oder aber kleinere Anzahlen von Nicht-Verwerfungen sehr klein, dann glauben wir nicht weiter an den Poisson-Prozeß. Ein Poisson-Prozeß wird als Modell dann verworfen, wenn nach dem Modell $Binom(n = N, p = 0.95)$ eine Realisation von K oder weniger Erfolgen eine Wahrscheinlichkeit von $\alpha = 0.05$ oder weniger besitzt.

R-Umsetzung. Auch dieses soll umgesetzt werden.

26 ⟨*definiere* `arrival.poisson.gof` 26⟩ ≡ ⊂ 79

```
arrival.poisson.gof<-function(x,diff=T,N,alpha=0.05){
  if(diff) x<-cumsum(x)
  I <- (T<-(rev(x)[1]-x[1]))/N
  K <- 0
  for(i in 1:N){
    T.I <- x[ ( (i-1)*I ) < x & x <= (i*I) ]
    diff.T.I <- diff(T.I)
    h<-ad.exp(diff.T.I)
    K<-K+(h["Entscheidung"]=="Nullhypothese wird nicht verworfen")
  }
  return(if(pbinom(K,size=N,p=0.95) <= 0.05)
           "Nullhypothese wird verworfen"
         else
           "Poisson-Prozess wird als Modell nicht abgelehnt")
}
arrival.poisson.gof(c(rexp(100,0.2),rexp(200,2.0)),N=10)
# for(i in 1:100) print(arrival.poisson.gof(c(rexp(99,0.2),rexp(99,2.0)),N=10)
```

27 ⟨* 1⟩+ ≡

```
x<-dzeitpunkte.17.02.97
arrival.poisson.gof(x,N=4)
```

liefert:

```
Tue Nov 26 16:34:50 2002
[1] "Nullhypothese wird verworfen"
```

4.2.5 Unabhängigkeits-Test-Strategie mittels acf

Methodology for testing for Poisson arrivals – Independence. Selbst wenn der Anpassungstest einen Erfolg meldet, können Abhängigkeiten zwischen den Zwischenankunftszeiten die Modellierung empfindlich stören. Deshalb schließen Paxon und Floyd diesbezüglich weitere Tests an.

Erstens berechnen sie für jedes Zeitintervall die Autokorrelation zum Lag 1. Für einen white noise process gilt, daß sich mit einer Wahrscheinlichkeit von 95% die Autokorrelationen in dem Intervall $[-1.96/\sqrt{n}, +1.96/\sqrt{n}]$ bewegen. Liegt die Beobachtung innerhalb des Intervall, hat das Zeitintervall den Teiltest erfolgreich passiert. Unter der Hypothese der Unabhängigkeit sollten circa 95% erfolgreiche Testergebnisse resultieren. Genauer müßten die Anzahl der erfolgreichen Testergebnisse wieder binomialverteilt sein mit n =Anzahl der Zeitintervalle und $p = 0.95$. Viel zu wenige erfolgreiche Ergebnisse führen wiederum zur Ablehnung der Unabhängigkeitshypothese. Der kritische Wert der Anzahlen wird über den 0.05-Prozentpunkt die Binomialverteilung ermittelt.

Zweitens werden die Korrelationen zum Lag $\tau = 1$ betrachtet. Unter Unabhängigkeit müßten diese ein positives Vorzeichen mit $p = 0.5$ haben. Weicht die Anzahl der positiven Vorzeichen deutlich von der Erwartung np ab, spricht dieses gegen die Unabhängigkeitshypothese. Die Entscheidung wird für die Gegenhypothese gefällt, wenn als Anzahl der positiven Vorzeichen eine Zahl außerhalb des Prozentpunkt-Intervalls $(x_{\alpha/2}, x_{1-\alpha/2})$ beobachtet wird.

R-Umsetzung. Dieses läßt sich schnell umsetzen.

28 ⟨*definiere* `arrival.poisson.ind` 28⟩ ≡ ⊂ 79

```
arrival.poisson.ind<-function(x,diff=T,N,alpha=0.05){
  if(diff) x<-cumsum(x)
  I <- (T<-(rev(x)[1]-x[1]))/N
  Kor.ok <- 0; Vz.pos<-0
  for(i in 1:N){
    T.I <- x[ ( (i-1)*I ) < x & x <= (i*I) ]
    diff.T.I <- diff(T.I)
    r1<-cor(diff.T.I[-1],diff.T.I[-length(diff.T.I)])
    Vz.pos <- Vz.pos + r1 > 0
    Kor.ok <- Kor.ok + (1.96 / sqrt(length(diff.T.I))) > abs(r1)
  }
  result.Kor <- if(pbinom(Kor.ok,size=N,p=0.95) <= 0.05)
                   "Unabhaengigkeit wegen zu grosser cor abgelehnt"
                else
                   "Unabhaengigkeit wegen zu grosser cor nicht abgelehnt"
  result.Vz  <- if(abs(pbinom(Vz.pos,size=N,p=0.50)-.5) >= 0.45)
```

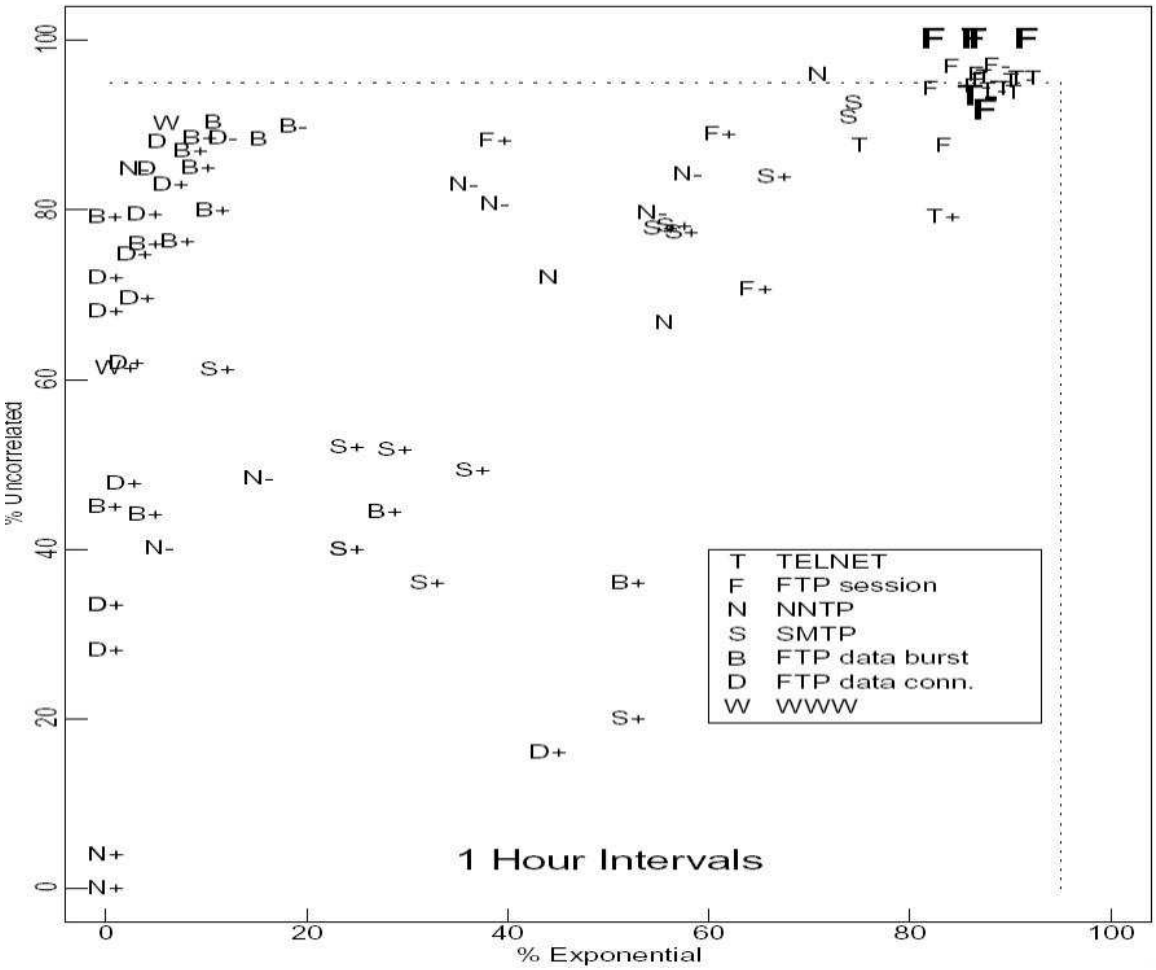
```
                    "Unabhaengigkeit wegen Vorzeichen von cor abgelehnt"
                else
                    "Unabhaengigkeit wegen Vorzeichen von cor nicht abgelehnt"
  return(list(c(result.Kor, result.Vz), c(Kor.ok=Kor.ok,Vz.pos=Vz.pos)))
}
```

Ein Testeinsatz – ist noch zu überprüfen.

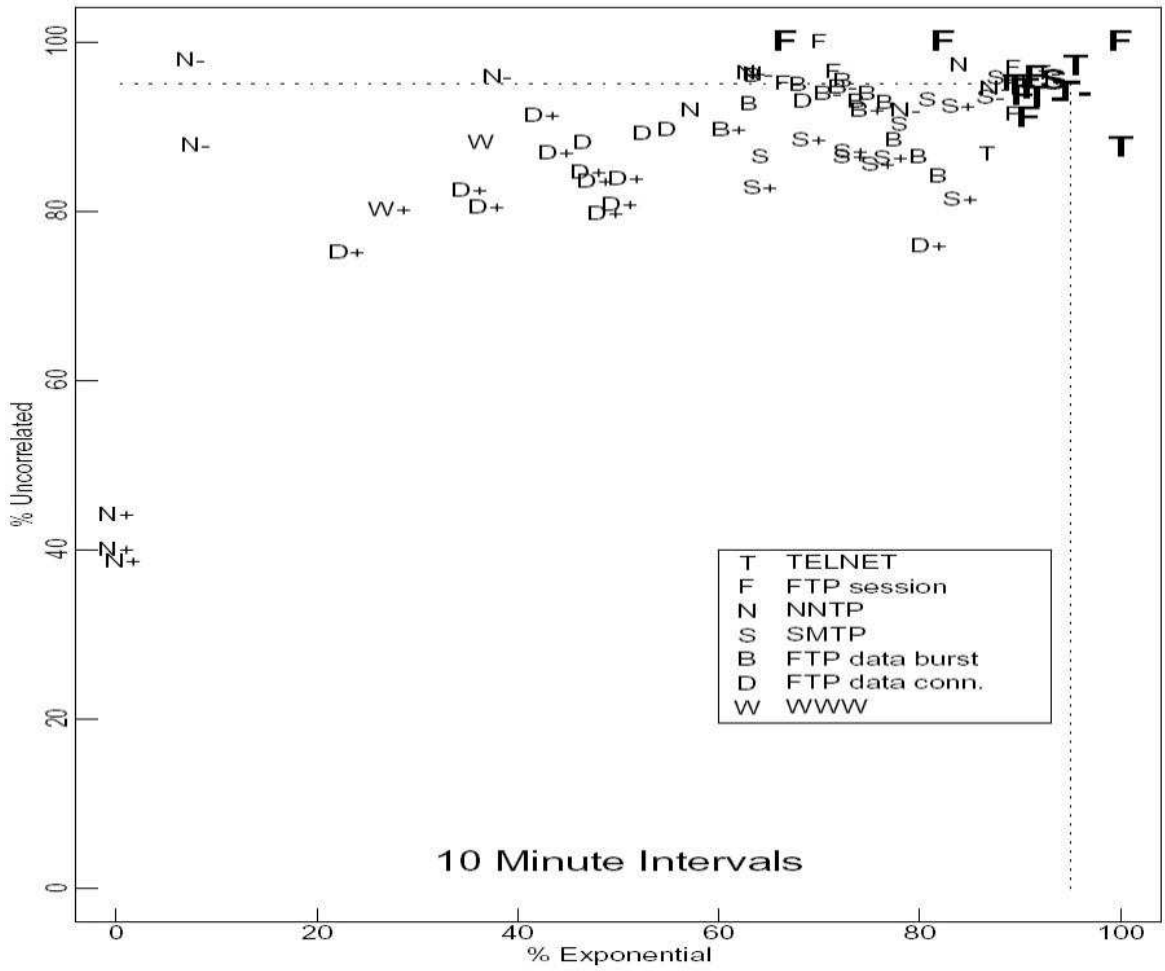
29 ⟨ * 1⟩+ ≡

```
arrival.poisson.ind(c(rexp(100,0.2),rexp(200,2.0)),N=10)
```

Die Ergebnisse aus den beiden Tests lassen sich gegeneinander abtragen. Diese zweidimensionale Darstellung ist für den menschlichen Betrachter natürlich sehr viel anschaulicher als reines Zahlenmaterial. Hier ist ein kurzer Blick auf die Ergebnisse von Paxson und Floyd:



Die Interpretation dieser und des nächsten Bildes sei dem Leser überlassen. Eventuelle Unklarheiten sind im Aufsatz von Paxson und Floyd zu recherchieren: [22].



4.3 Fazit

> *We see immediately that TELNET connection arrivals and FTP session arrivals are very well modeled as Poisson, both for 1-hour and 10-minute fixed rates. No other protocol's arrivals are well modeled as Poisson with fixed hourly rates. If we require fixed rates only over 10-minute intervals, then SMTP and FTPDATA burst arrivals are not terribly far from Poisson, though neither is statistically consistent with Poisson arrivals, and consecutive SMTP interarrival times show consistent positive correlation. NNTP, FTPDATA, and WWW arrivals, on the other hand are clearly not Poisson.*
>
> [22]

Wir haben verstanden, wie die Überprüfung des Poisson-Prozesses unternommen werden könnte. Erfreulicherweise gab es Situationen, in denen der Poisson-Prozeß geeignet erscheint. Leider waren WWW arrivals nicht dabei. Zur Ergänzung sei angefügt: SMTP=Simple Mail Transfer Protocol, NNTP=Network News Transport Protocol, FTP=File Transport Protocol.

Das Fazit lautet also: Poisson paßt nicht immer!

Wie die Beobachtungen zeigen, kommen Anfragen schubweise, d.h., es wechseln Phasen höherer Intensität mit Phasen niedrigerer Intensität ab. Als Folge werden Verteilungen mit einer höheren Wahrscheinlichkeitsmasse im rechten Schwanz benötigt. Außerdem können mit der Exponentialverteilung aufgrund ihrer Gedächtnislosigkeit keine Ballungen von Anfragen modelliert werden. Wir müssen uns also nach anderen Modellen oder Prozessen umschauen. Zunächst wollen wir nach Verteilungen mit größerer Wahrscheinlichkeitsmasse im rechten Schwanz Ausschau halten.

4.4 Weibull — ein Modell mit dickem Ende

Angeregt durch verschiedene Hinweise in den schon vorgestellten Artikeln wollen wir uns die Weibull-Verteilung als besseres Wahrscheinlichkeitsmodell anschauen.

4.4.1 Wichtige Eigenschaften der Weibull-Verteilung

Ihre Verteilungsfunktion ist gegeben durch:

$$F(x) = 1 - \exp\left[-(x/b)^c\right] \qquad 0 \leq x$$

Offensichtlich besitzt die Weibull-Verteilung die beiden Parameter b und c. Für $c = 1$ ergibt sich die Verteilungsfunktion der Exponentialverteilung mit $\lambda = 1/b$. b ist wie bei der Exponentialverteilung ein scale Parameter, der Kehrwert der Traffic-Rate. c ist ein shape oder Gestalt-Parameter. Der Erwartungswert ist gegeben durch

$$E(X) = b\Gamma[(c+1)/c] \qquad \Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx$$

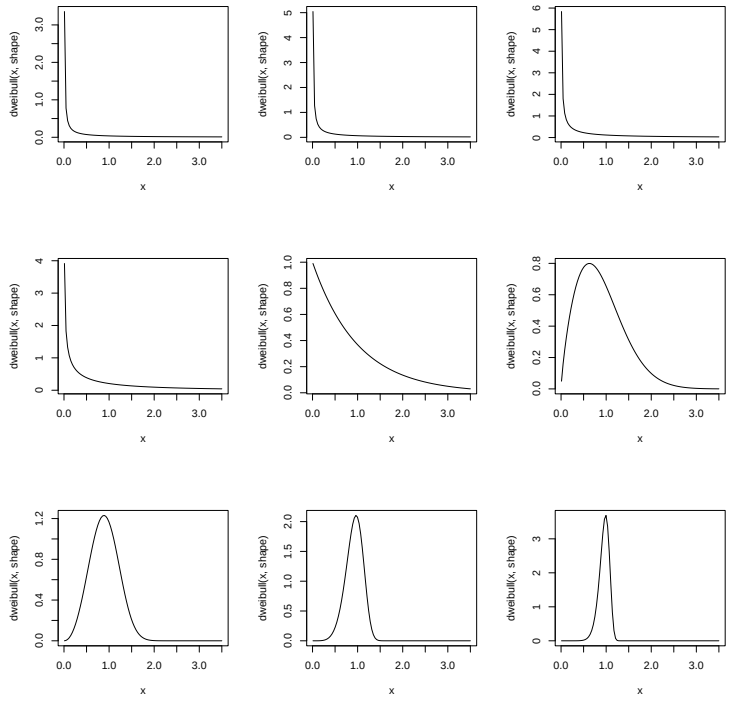
und die Varianz durch

$$\mathrm{Var}(X) = b^2\Gamma[(c+2)/c] - E(X)^2$$

Je kleiner c ist, umso mehr Wahrscheinlichkeitsmasse befindet sich im rechten Schwanz der Verteilung. Entsprechend steigen mit fallendem c Erwartungswert und Varianz an. Für die Vorstellung mag die Darstellung der Dichtefunktion helfen:

30 ⟨ * 1⟩+ ≡

```
x <- seq(.01,3.5,length=100)
par(mfrow=c(3,3))
for(shape in exp(seq(log(.1),log(10),length=9)))
  plot(x,dweibull(x,shape),type="l")
par(mfrow=c(1,1))
```



4.4.2 Parameterschätzung

Die Maximierung der Likelihood-Funktion der Weibull-Verteilung führt nach Hastings [12] zu folgenden beiden Gleichungen. Deren Lösung definieren den Maximum-Likelihood-Schätzer für (b, c).

$$\hat{b} = \left[\left(\frac{1}{n} \right) \sum_{i=1}^{n} x_i^{\hat{c}} \right]^{1/\hat{c}}$$

und

$$\hat{c} = \frac{n}{(1/\hat{b})^{\hat{c}} \sum_{i=1}^{n} x_i^{\hat{c}} \log x_i - \sum_{i=1}^{n} \log x_i}$$

Eine numerische Lösung. Ein numerisches Ergebnis erhalten wir mit folgender Funktion `weibull.est`. Diese sucht c in einem Intervall von .2 bis 5 iterativ in 9 Schritten einzugrenzen. In jedem Schritt werden 10 Werte für c probiert. Mit den c-Werten werden die b-Werte bestimmt, um mit diesen gemäß der zweiten Formel $\hat{c}$ zu berechnen. Das Intervall der beiden c-Werte mit den geringsten Differenzen zu den berechneten c-Werten wird als Start-Intervall für die nächste Iteration verwendet. Ausgegeben wird die Mitte des zuletzt berechneten Intervalls. Außerdem wird die Entwicklung der Differenzen graphisch dargestellt.

31 ⟨ *definiere* `weibull.est` 31⟩ ≡ ⊂ 79

```
weibull.est<-function(x){
```

```
  par(mfrow=c(3,3))
  n <- length(x); n.c=10
  log.x <- log(x)
  log.x.mat <- matrix(log.x,length(x),n.c)
  c.min <- .2; c.max <- 5
  for(tiefe in 1:9){
    c.dach<-exp(seq(from=log(c.min), to=log(c.max), length=n.c))
    x.hoch.c  <- outer(x,c.dach,"^")
    b.dach.hoch.c <- apply(x.hoch.c,2,mean)
    delta <- c.dach - n/(
                          ( 1/b.dach.hoch.c ) * apply(x.hoch.c*log.x.mat,2,sum)
                          - sum(log.x)
                        )
    plot(c.dach,delta,type="l"); abline(h=0)
    c.min<-max(c.dach[delta<0]); c.max<-min(c.dach[delta>0])
    if(length(c.min)==0 || length(c.max)==0) break
  }
  par(mfrow=c(1,1))
  c.dach<-0.5*(c.min+c.max)
  return(c(c.dach=c.dach,b.dach=mean(x^c.dach)^(1/c.dach)))
}
```

Ein Testeinsatz liefert für exponentialverteilte Zufallszahlen:

32 ⟨* 1⟩+ ≡

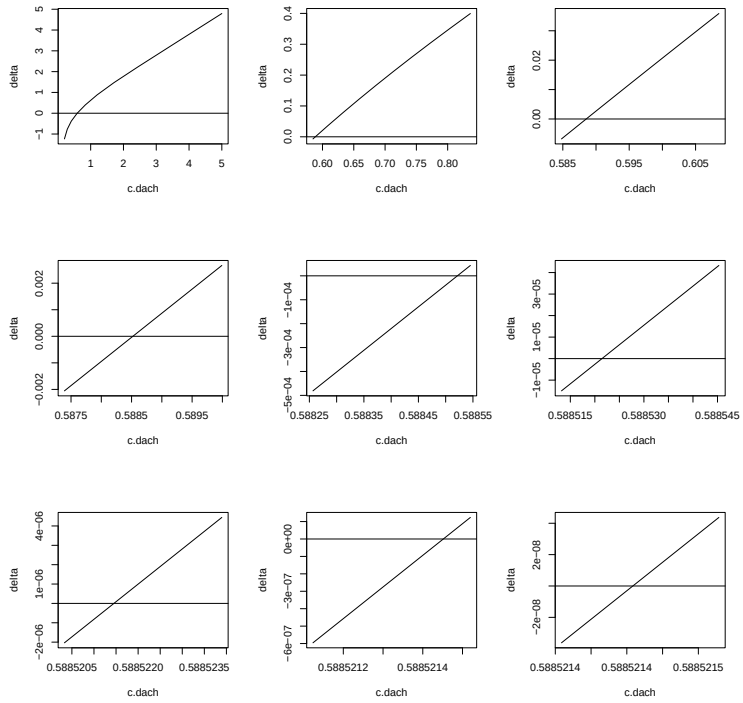
```
weibull.est(rexp(100))
```

Für unsere Zeitdifferenzen aus dem Jahr 1997 liefert die Schätzung:

33 ⟨* 1⟩+ ≡

```
weibull.est(dzeitpunkte.03.02.97)
```

```
Tue Dec  3 12:10:38 2002
     c.dach      b.dach
  0.5885215 490.1179108
```



Die Newton-Methode. Natürlich geht es auch vornehmer. So finden wir im Internet mittels Google unter den Stichwörtern "Parameter Estimation", "Maximum Likelihood", "Weibull" zum Beispiel ein Kapitel über "Iterative Methods for Parameter Estimation" (`stat.tamu.edu/~jnewton/604/chap4.pdf`), das wiederum einen Abschnitt zur Maximum-Likelihood-Schätzung mittels Newton-Raphson enthält.

Newton-Raphson? Hier ein kurzer Exkurs nach [28]

> Die Newton-Raphson-Methode ist ein Nullstellenverfahren, das ausgehend von der Taylorschen Reihenentwicklung auf folgendem Ansatz basiert:
>
> $$0 = f(s) = f(x_i) + (s - x_i)f'(x_i) + \frac{(s - x_i)^2}{2} f''(x^*)$$
>
> Die Nullstelle s läßt sich hiermit approximieren durch
>
> $$s \approx x_i - \frac{f(x_i)}{f'(x_i)}$$
>
> Iterativ können wir uns unter günstigen Bedingungen (Funktion stetig differenzierbar, x_i in der Nähe der Nullstelle s) der Nullstelle annähern:
>
> $x_0 :=$ initial value
> **for** $i := 0$ **to** ∞ **until** convergence **do**
> $\quad x_{i+1} := x_i - f(x_i)/f'(x_i)$.

Falls keine Ableitung vorliegt, läßt sich diese durch einen Differenzenquotienten annähern. Diese Modifikation heißt Sekanten Methode.

Zur Optimierung läßt sich die Newton-Methode einsetzen, indem man die Nullstelle der ersten Ableitung sucht. Für das Verfahren ist dann auch die zweite Ableitung erforderlich. Beide Ableitungen findet man in dem erwähnten Kapitel. Dadurch daß zwei Parameter geschätzt werden müssen, kann man zum Beispiel das Newton-Verfahren abwechselnd auf die beiden Dimensionen ansetzen. Man kann natürlich auch gleich mit einem Gradientenverfahren das Maximum der Likelihood-Funktion bestimmen.

4.4.3 Ein Erkennungsplot für die Weibull-Verteilung

Entsprechend dem Erkennungsplot zur Exponentialverteilung wollen wir schauen, ob es gelingt, für die Weibull-Verteilung einen entsprechenden Plot anzustellen. Betrachten wir noch einmal die Verteilungsfunktion:

$$F(x) = 1 - \exp\left[-(x/b)^c\right]$$

Dann läßt sich diese Gleichung durch Logarithmierung umformen zu:

$$\ln(1 - F(x)) = -(x/b)^c$$

und eine weitere Logarithmierung erwirtschaftet eine lineare Beziehung:

$$\ln(-\ln(1 - F(x))) = c\,(\ln(x) - \ln(b)) \stackrel{!}{=} a_1 x^* + a_0$$

Werden die Werte $\ln(-\ln(1 - \hat{F}(x)))$ gegen $\ln(x)$ abgetragen, können wir durch den entstandenen Scatterplot eine Gerade legen und aus den Geradenparametern $\hat{a}_0$ und $\hat{a}_1$ Schätzungen für die gesuchten Parameter b und c ermitteln:

$$\hat{a}_1 = \hat{c}, \qquad \hat{a}_0 = \hat{c}\ln(\hat{b})$$

Es folgt:

$$\hat{c} = \hat{a}_1, \qquad \hat{b} = \exp\left(\frac{\hat{a}_0}{\hat{c}}\right)$$

Vorteil dieses Ansatzes ist natürlich wiederum, daß wir per Auge entscheiden können, ob gravierende Ungereimtheiten vorliegen. Dieser Ansatz läßt sich auch schnell per Funktion umsetzen:

34 ⟨*definiere* `identify.weibull` 34⟩ ≡ ⊂ 79

```
identify.weibull<-function(x){
  x    <- sort(x); n<-length(x)
  ln.x <- log(x)
  F.x  <- (1:n)/n - 1/(2*n)
  y    <- log(-log(1-F.x))
  plot(ln.x, y, type="b", xlab="log(x)",
```

```
        ylab="ln(-ln(1-F.dach(x)))")
  coef<-lsfit(ln.x,y)$coef; names(coef)<-NULL
  abline(coef)
  shape<-coef[2]; scale<-exp(coef[1]/coef[2])
  title(paste("Weibull: scale=",signif(scale,3),
                  "- shape=",signif(shape,2)))
  return(c(shape=shape, scale=scale))
}
```

Hier eine Testanwendung:

35 ⟨ * 1⟩+ ≡

```
identify.weibull(rexp(100,10))
```

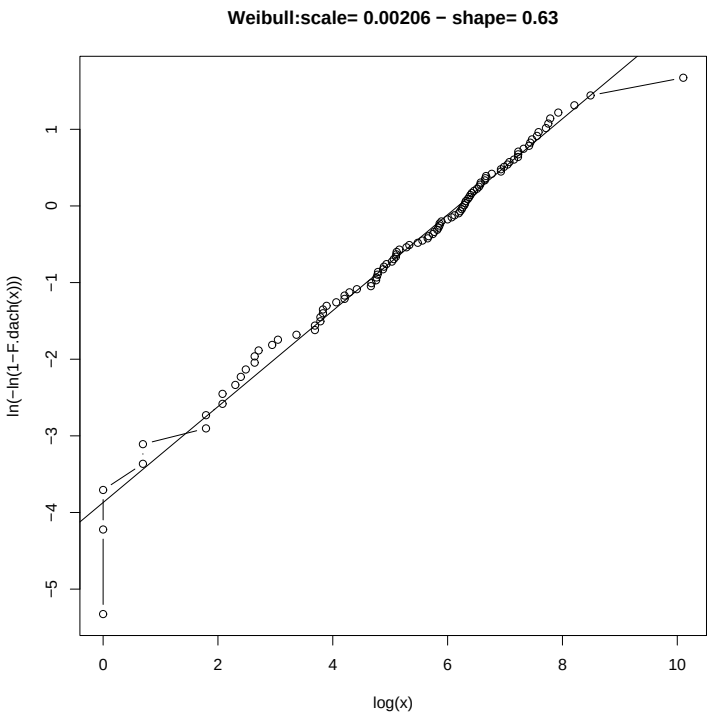
```
Tue Dec  3 14:24:56 2002
   shape     scale
1.032544 9.952769
```

Für weitere Bemerkungen zur Parameterschätzung sei an dieser Stelle auch hingewiesen auf [20].

36 ⟨ * 1⟩+ ≡

```
identify.weibull(dzeitpunkte.03.02.97)
```

```
Tue Dec  3 14:25:24 2002
      shape        scale
0.625982580 0.002064576
```



4.5 Die Pareto-Verteilung

4.5.1 Allgemeines

Als zweite Verteilung taucht in verschiedenen Artikeln die Pareto-Verteilung auf:

$$F(x) = 1 - (a/x)^c \qquad x \geq a$$

Durch Ableitung ergibt sich die Dichte:

$$f(x) = ca^c/x^{c+1} \cdot I_{[a,\infty)}(x)$$

Der Parameter a ist ein Lageparameter, er definiert die linke Grenze für die Realisationen. c ist ein Gestaltparameter. Für den Erwartungswert finden wir:

$$E(X) = ca/(c-1) \qquad \text{jedoch nur, falls } c > 1 \text{ gilt.}$$

Die Varianz berechnet sich nach der Formel:

$$\mathrm{Var}(X) = \frac{ca^2}{(c-1)^2(c-2)} \qquad \text{jedoch nur, falls } c > 2 \text{ gilt.}$$

Formal sind die angegebenen Einschränkungen verständlich. Was steckt jedoch inhaltlich dahinter? Falls der Parameter c kleiner als 2 wird, existiert die Varianz nicht mehr. Wird c sogar kleiner als 1, ist es nicht mehr möglich, einen Mittelwert anzugeben. Wir können uns das so vorstellen, daß es immer mal wieder eine so große Realisation gibt, daß diese sämliche Berechnungen sprengt.

Verständlicher dürfte vielleicht ein Vergleich von Exponentialverteilung und Pareto-Verteilung sein: Die *Überlebenswahrscheinlichkeit*, also die Wahrscheinlichkeit für eine Realisation größer als x_0, ist bei der Exponentialverteilung gegeben durch

$$1 - F_E(x_0) = \exp(-\lambda x_0).$$

Bei der Pareto-Verteilung ergibt sich entsprechend:

$$1 - F_P(x_0) = (a/x_0)^c.$$

Erstere sinkt exponentiell und damit viel schneller als die zweite, deren Sinken durch den Kehrwert des Polynoms x_0^c beschrieben wird. Bei der Pareto-Verteilung mit kleinem Parameter c stellen sich also viel höhere Überlebenswahrscheinlichkeiten ein.

Funktionen zur Pareto-Verteilung:

37 ⟨*definiere Funktionen zur Pareto-Verteilung* 37⟩ ≡ ⊂ 79

```
dpareto<-function(x,a=.001,c=1){
  # density of pareto distribution
  ifelse(x>=a,c*a^c / (x^(c+1)),0)
}
ppareto<-function(x,a=.001,c=1){
  # c.d.f. of pareto distribution
  ifelse(x>=a,1-(a/x)^c,0)
}
qpareto<-function(p,a=.001,c=1){
  # quantiles of pareto distribution
  a/(1-p)^(1/c)
}
rpareto<-function(n,a=.001,c=1){
  # random numbers of pareto distribution
  u <- runif(1:n)
  a/u^(1/c)
}
```

Ein kurzer Blick auf die Dichten der Pareto-Verteilung:

38 ⟨* 1⟩+ ≡

```
x<-seq(1,5,length=50)
plot(1,type="n", xlab="x", ylab="f(x)",
                 xlim=c(0.5,3), ylim=c(0,2))
c.set <- c(.1, .3, .7, 1, 2, 5)
for(i in seq(c.set))lines(x, dpareto(x,a=1,c=c.set[i]),lty=i)
legend(2,1,bty="n",paste("c=",c.set),lty=seq(c.set))
title("Pareto -- densities, a=1")
```

4.5.2 Parameterschätzung

Der Maximum-Likelihood-Schätzer für a ist durch die kleinste Beobachtung gegeben:

$$\hat{a} = \min x_i$$

Für den Kehrwert von c liefert

$$\hat{c} = \left[\frac{1}{n} \sum_{i=1}^{n} \log(x_i/\hat{a}) \right]^{-1}$$

den Maximum-Likelihood-Schätzer.

Läßt sich auch hier ein Erkennungsplot erstellen? Die Verteilungsfunktion

$$F(x) = 1 - (a/x)^c \qquad x \geq a$$

liefert:

$$1 - F(x) = (a/x)^c \Rightarrow \log(1 - F(x)) = c\log(a) - c\log(x)$$

Also kann $\log(1 - F(x))$ gegen $\log(x)$ abgetragen werden. Dann lassen sich die Ergebnisse aus der Geradenschätzung auf die Verteilungsparameter zu übertragen.

39 ⟨*definiere* `identify.pareto` 39⟩ ≡ ⊂ 79

```
identify.pareto<-function(x){
  x    <- sort(x); n<-length(x)
  ln.x <- log(x)
  F.x  <- (1:n)/n - 1/(2*n)
  y    <- log(1-F.x)
  plot(ln.x, y, type="b", xlab="ln(x)", ylab="ln(1-F.dach(x))")
  coef<-lsfit(ln.x,y)$coef; names(coef)<-NULL
  abline(coef)
  c.dach <- -coef[2]
  a.dach <- exp(coef[1] / c.dach)
  title(paste("Pareto: a.dach=",signif(a.dach,3),
                    "- c.dach=",signif(c.dach,3)))
  return(c(a.dach=a.dach, c.dach=c.dach,
           a.ml=min(x),c.ml=mean(log(x/min(x))) ))
}
```

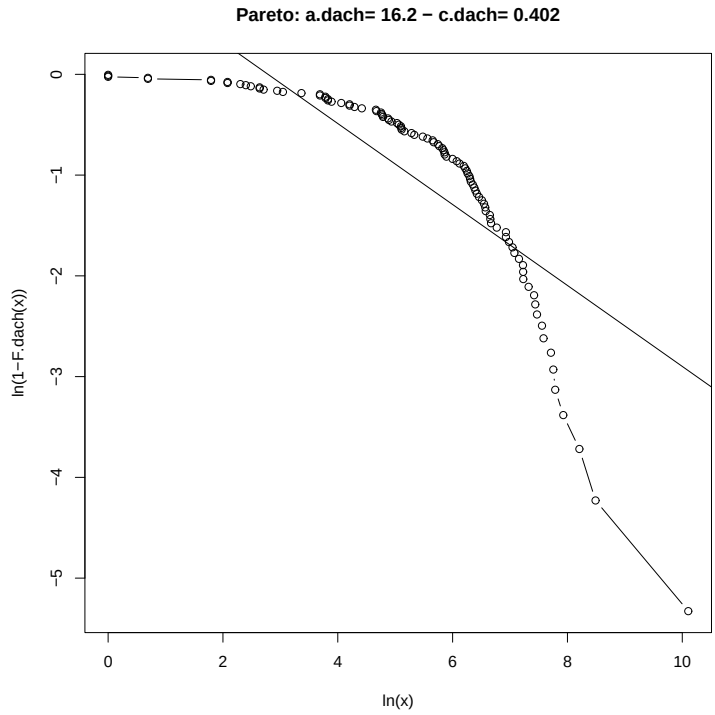
Könnte für unsere Serverdaten eine Pareto-Verteilung geeignet sein? Hier ein kurzer Test:

40 ⟨* 1⟩+ ≡

```
x<-dzeitpunkte.03.02.97
identify.pareto(x)
```

```
Tue Dec  3 16:56:01 2002
    a.dach     c.dach       a.ml       c.ml
16.1962469  0.4017833  1.0000000  5.2653182
```

Schon aus der Unterschiedlichkeit der Schätzungen können wir eine Pareto-Verteilung als geeignetes Modell ausschließen. Dieses wird durch die Inspektion des von `identify.pareto` erstellten Plots noch untermauert:



Damit wollen wir die Diskussion von Verteilungen mit langen rechten Schwänzen abschließen und uns wieder der Frage zuwenden, wie man die *burstiness* der Anfragen in den Griff kriegen kann.

4.6 Selbstähnliche Modelle

Die Ausführungen dieses Unterkapitels stützen sich wesentlich auf: [7], [18], [21], [33].

4.6.1 Ausgangspunkt

Wo stehen wir? Fassen wir noch mal zusammen:

- Bei den bisherigen Untersuchungen haben wir festgestellt, daß eine Exponentialverteilung zur Modellierung der Zwischenankunftszeiten eher nicht geeignet ist. Mit einer Weibull-Verteilung gelingt uns dagegen eine viel bessere Anpassung. Würden wir die mehrfachen einzelnen Zugriffe, die für die komplette Konstruktion von `www`-Seiten erforderlich sind, mit berücksichtigen oder gar auf Paketebene hinabsteigen, dann würde sich – wie aus verschiedenen Literaturstellen hervorgeht – eine Pareto-Verteilung

anbieten. Pareto-Verteilungen zeichnen sich durch noch dickere Schwänze aus, besonders wenn der `shape`-Parameter sehr klein wird. Für $c < 2$ ist Varianz, für $c < 1$ ist Erwartungswert nicht mehr existent.

- Weiterhin hat sich gezeigt und ist schon aus Sach- und inhaltlichen Gründen verständlich, daß die Unabhängigkeitsannahme des Poisson-Prozesse besonders auf den tieferen Betrachtungsebenen (Paketebene) nicht erfüllt ist.

- Als dritter Punkt soll folgende Graphik aus dem Papier von Willinger, Taqqu, Sherman und Wilson, [33], zur Einstimmung dienen. Diese liefert sowohl eine Begründung für die Überschrift des Kapitels, wie auch einen Ansatzpunkt für weitere Überlegungen.

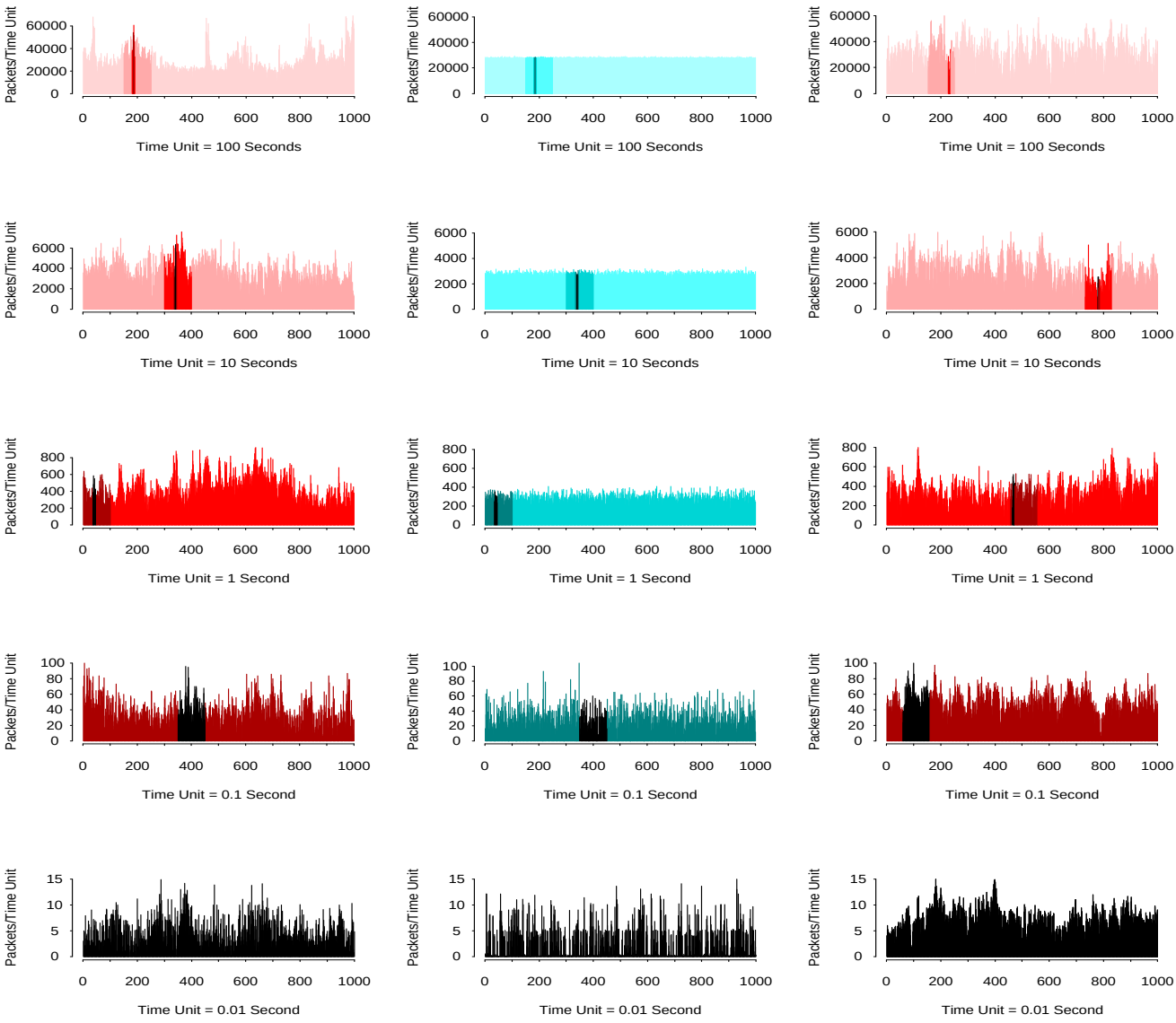


Figure 1: Actual Ethernet traffic (left column), synthetic trace generated from an appropriately chosen traditional traffic model (middle column), and synthetic trace generated from an appropriately chosen self-similar traffic model with a single parameter (right column) – on five different time scales. Different gray levels indicate the same segments of traffic on the different time scales.

Was zeigt uns die Graphik? In der mittleren Spalte sind die Verhältnisse dargestellt, wie sie im Falle eines Poisson-Prozesses zu erwarten sind: Werden kurze Zeitintervalle der Länge t betrachtet (siehe Graphik unten), dann wird die

Anzahl der Ereignisse Y_t in diesen Intervallen Poisson-verteilt sein mit einer Varianz von λt. Durch Aggregation von m Intervallen müssen wir für die Anzahl der Ereignisse $Y_t^{(m)}$ in dem entstandenen Intervall der Länge mt zu der Poisson-Verteilung mit der Varianz λmt übergehen. Die Intervall-Länge eines Bildes unterscheidet sich um den Faktor 10 von der Intervall-Länge des direkt darunter befindlichen. Durch Mittelung lassen sich die Eigenschaften der kurzen Intervalle mit dem Aggregat vergleichen:

$$A_t^{(m)} = Y_t^{(m)}/m = \sum_{i=1}^{m} Y_i/m$$

Dann gilt für $A_t^{(m)}$:

$$E(A_t^{(m)}) = E(Y_t^{(m)}/m) = E\left(\sum_{i=1}^{m} Y_i/m\right) = \lambda mt/m = \lambda t$$

und

$$\mathrm{Var}(A_t^{(m)}) = \mathrm{Var}(Y_t^{(m)}/m) = \mathrm{Var}\left(\sum_{i=1}^{m} Y_i/m\right) = \lambda mt/m^2 = \lambda t/m = \mathrm{Var}(Y_1)/m$$

Es gilt natürlich auch hier das Wurzel-n-Gesetz. Dieses Gesetz führt dazu, daß die oberen Stabdiagramme der mittleren Spalte zu den größeren Intervallen glatter erscheinen als die darunter liegenden.

Erstellt man jedoch solche Bilder zu realen Daten vom Paketverkehr (siehe linke Spalte), dann läßt sich in bestimmten Größenordnungen dieses Phänomen nicht beobachten. Die Varianzen für das gemittelte Aggregat fallen größer aus und die Beziehung zu der Varianz der kurzen Intervalle könnte allenfalls gegeben sein durch:

$$\mathrm{Var}(A_t^{(m)}) > \frac{1}{m}\mathrm{Var}(Y_1)$$

Über den Gesamteindruck können wir festzustellen, daß sich die Strukturen in allen Graphiken der linken Spalte deutlich ähneln. So ist es nicht sehr verwunderlich, diese markante Eigenschaft als *Selbstähnlichkeit* zu bezeichnen. Die rechte Spalte zeigt übrigens Graphiken eines synthetischen Prozesses, mit dem es gelungen ist, die Selbstähnlichkeitseigenschaft zu reproduzieren.

Es stellen sich für diesen Abschnitt folgende Fragen:

- Was wird unter dem Begriff Selbstähnlichkeit verstanden?
- Wie läßt sich prüfen, ob bzw. inwieweit Selbstähnlichkeit vorliegt?
- Welche Test-Ergebnisse lassen sich berichten?
- Wie lassen sich Prozesse mit selbstähnlichen Strukturen erzeugen?
- Welche Ursachen führen zu selbstähnlichen Strukturen im Internet-Traffic?

4.6.2 Selbstähnlichkeit — was ist das?

Beginnen wir mit einer Definition.

Definition H-self-similar:

$X = (X_t; t = 1, 2, 3, \ldots)$ sei ein stationärer Prozeß mit Mittel 0. Durch Aggregation von sich nicht überschneidenen Abschnitten erhalten wir den m-aggregierten Prozeß $X^{(m)} = (X_t^{(m)}; t = 1, 2, 3, \ldots)$ mit

$$X_t^{(m)} = X_{tm-m+1} + \cdots + X_{tm} = \sum_{i=(t-1)m+1}^{tm} X_i \qquad t \geq 1$$

Dann heißt X H-selbstähnlich (H-self-similar), wenn für alle positiven Zahlen m die Verteilung der X_t und $m^{-H} X_t^{(m)}$ gleich ist:

$$X_t \stackrel{d}{=} \frac{1}{m^H} X_t^{(m)} \qquad \text{für alle } m \in \mathcal{N}$$

Der Parameter H heißt *Hurst*-Paramter.

∘

Wenn wir also abschnittsweise Zeitreihenwerte addieren, dann sollte die (empirische) Verteilung diese Aggregats dividiert durch m^H genauso aussehen wie die Verteilung aller Zeitreihenwerte.

Nur auf die ersten Momente bezogen findet man als Definition:

Definition second-order self-similarity:

X sei ein kovarianzstationärer Prozeß mit einer acf

$$\rho_\tau \sim \tau^{-\beta}, \qquad \tau \to \infty.$$

Dann heißt X second-order self-similar mit dem Selbstähnlichkeitsparameter $H = 1 - \beta/2$, wenn $X^{(m)}$ für alle $m = 1, 2, \ldots$ dieselbe Korrelationsstruktur hat:

$$\rho_\tau^{(m)} = \rho_\tau$$

∘

Auch asymptotisch selbstähnliche Prozesse lassen sich bedenken:

Definition asymptotisch second-order selbstähnlich:

Ein kovarianz-stationärer Prozeß heißt asymptotisch (second-order) selbstähnlich mit dem Selbstähnlichkeitsparameter $H = 1 - \beta/2$, wenn $\rho_\tau^{(m)}$ asymptotisch mit ρ_τ von X übereinstimmt:

$$\rho_\tau^{(m)} - \rho_\tau = \rho_\tau^{(m)} - c\tau^{-\beta} \to 0 \quad \text{für} \quad \tau \to \infty$$

∘

Was ergibt sich aus den Definitionen? Ist X H-selbstähnlich, dann besitzen X und $X^{(m)}$ dieselbe Autokorrelationsfunktion (acf):

$$\rho_\tau = E[(X_t - \mu)(X_{t+\tau} - \mu)]/\sigma^2$$

Über die Autokorrelationsfunktion läßt sich eine Klasse selbstähnlicher Prozesse beschreiben:

Ergebnis ohne Beweis.

Besitzt die acf die Gestalt

$$\rho_\tau \sim \tau^{-\beta} \quad \text{für} \quad 0 < \beta < 1,$$

dann entstehen Abhängigkeiten über große Lags und der Prozeß ist selbstähnlich mit dem Parameter H:

$$H = 1 - \beta/2.$$

Für selbstähnliche Prozesse mit gemäß $\tau^{-\beta}$ fallender acf folgt deshalb $0.5 < H < 1$. Für solche Prozesse fällt die acf langsamer als exponentiell.

Der Parameter β hat noch eine anschauliche Bedeutung:

Ergebnis ohne Beweis.

Die Varianz des Mittel von m aufeinander folgenden Zeitreihenzufallsvariablen $X_t, \ldots, X_{t+m-1}$ eines kovarianzstationären Prozesses mit $\rho_\tau \sim \tau^{-\beta}$ fällt mit m nicht proportional zu $1/m$, sondern es gilt:

$$\mathrm{Var}(X_t^{(m)}/m) = c \cdot m^{-\beta}$$

Die Aggregate selbstähnlicher Prozesse besitzen also die oben angesprochene Eigenschaft, daß die Mittel von n Reihenwerten nicht dem Wurzel-n-Gesetz gehorchen. Die Varianzen der Mittel verringern sich nur mit dem Faktor $m^{-\beta}$.

Im Frequenzbereich schlägt sich die hyperbolische Struktur der acf als Polstelle des Spektrums nieder. Diese Eigenschaft geht technisch auf die Nicht-Summierbarkeit der acf zurück.

Definition Spektrum:

Das Spektrum von X ist die Fouriertransformierte der Autokovarianzfunktion $\gamma_\tau = \gamma_0 \cdot \rho_\tau$ von X:

$$f(\lambda) = \sum_{\tau=-\infty}^{\infty} \gamma_\tau e^{i2\pi\lambda\tau} = \gamma_0 + 2\sum_{\tau=1}^{\infty} \gamma_\tau \cos 2\pi\lambda\tau.$$

$\circ$

Empirischer Ursprung. Hurst stellte bei einigen in der Natur beobachteten Zeitreihen das Phänomen fest, daß der Quotient aus einem Rangemaß und der Standardabweichung von jeweils m Werten proportional zu m^H mit $H \approx 0.73$ wächst. Für Reihen mit nur kurzfristigen Abhängigkeiten zeigten Mandelbrot und Van Ness, daß die Erwartung dieses Quotienten für $m \to \infty$ proportional zu Wurzel m wächst und damit weniger schnell als dieser Bruch bei den von Hurst betrachteten besonderen Zeitreihen. Diese Diskrepanz wird auch *Hurst-Effekt* genannt. Wir halten fest:

Ergebnis ohne Beweis.

Für selbstähnliche Prozesse mit dem Selbstähnlichkeitsparameter H gilt:

$$E\left(\frac{\text{range von m Beobachtungen}}{\text{Standardabweichung}}\right) \sim m^H$$

Vier Eigenschaften selbstähnlicher Prozesse, wie wir sie betrachten wollen, sind zu merken:

- Die Varianzen von Reihenmitteln sinken langsamer als proportional $1/m$.
- Der Quotient von Range und Standardabweichung ist proportional m^H.
- Die acf fällt mit τ weniger stark als $1/\tau$ und ist nicht summierbar (Josef-Effekt).
- Das Spektrum besitzt in 0 eine Polstelle und wächst gegen 0 gemäß: $f(\lambda) \sim c\lambda^{-\gamma}$ und $\gamma = 1 - \beta$.

Traditionelle Modelle (Poisson-Prozeß, Prozesse mit nur kurzer acf) weisen diese Eigenschaften nicht auf.

Es sei erwähnt, daß im Zusammenhang mit der Frage der Selbstähnlichkeit verschiedene Klassen differenziert werden. Im Besonderen können Abhängigkeiten für große Lags betrachtet (Josef-Effekt) oder aber unendlich große Varianzen (Noah-Effekt) als Idealisierung im Vordergrund stehen. Die Namen für diese beiden Effekte gehen nach Willinger et al. auf Mandelbrot zurück. Wir wollen dieser Unterscheidung hier jedoch nicht weiter nachgehen.

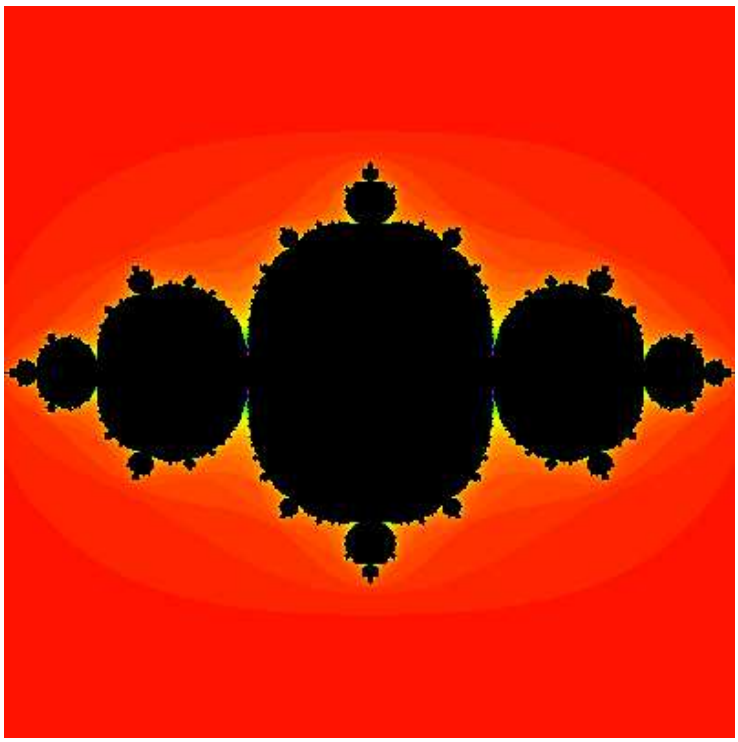
4.6.3 Apfelmännchen.

Jetzt ist in diesem Skript schon zum zweiten Male der Name Mandelbrot gefallen. Deshalb ist es wohl notwendig, einen kleinen Ausritt ins Reich der Apfelmännchen zu machen. Angefangen hat die Geschichte mit Herrn Gaston Julia (1893–1978), dem Mann, der im ersten Weltkrieg seine Nase verloren hatte. Mit nur 25 Jahren hatte Julia seine Arbeit verfertigt, die ihm seinen Platz in der Mathematik gesichert hat. Auf ihn gehen die berühmten Julia-Sets zurück, die wir zunächst beschreiben wollen. Betrachten wir dazu die unendliche Operationsfolge

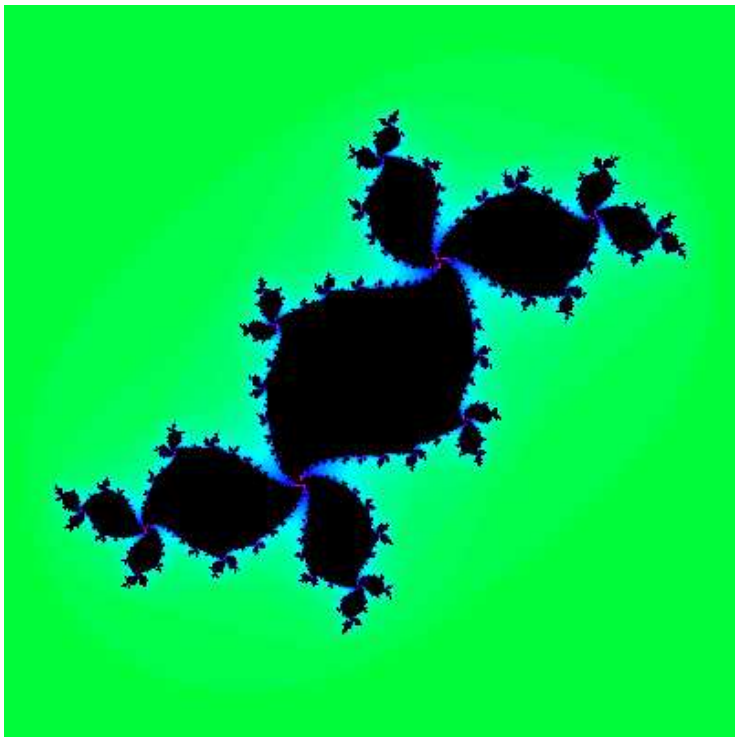
$c :=$ initial value
$x_0 :=$ initial value
for $i := 0$ **to** ∞ **do**
    $x_{i+1} := x_i^2 + c.$

Dann wird für ein vorgegebenes $c \in \mathbb{C}$ und jedes $x_0 \in \mathbb{C}$ die Folge der x_i entweder gegen 0 gehen, divergieren oder aber weder divergieren noch gegen 0 gehen. Wählen wir $c = 0 + 0i$, dann werden alle Punkte innerhalb des Einheitskreises gegen 0 streben; alle Punkte außerhalb des Einheitskreises führen mit jedem Schritt zu einem immer größerem Wert und alle Anfangspunkte x_0 auf dem Einheitskreis werden wieder auf Punkte des Einheitskreises abgebildet und bleiben damit betragsmäßig begrenzt.

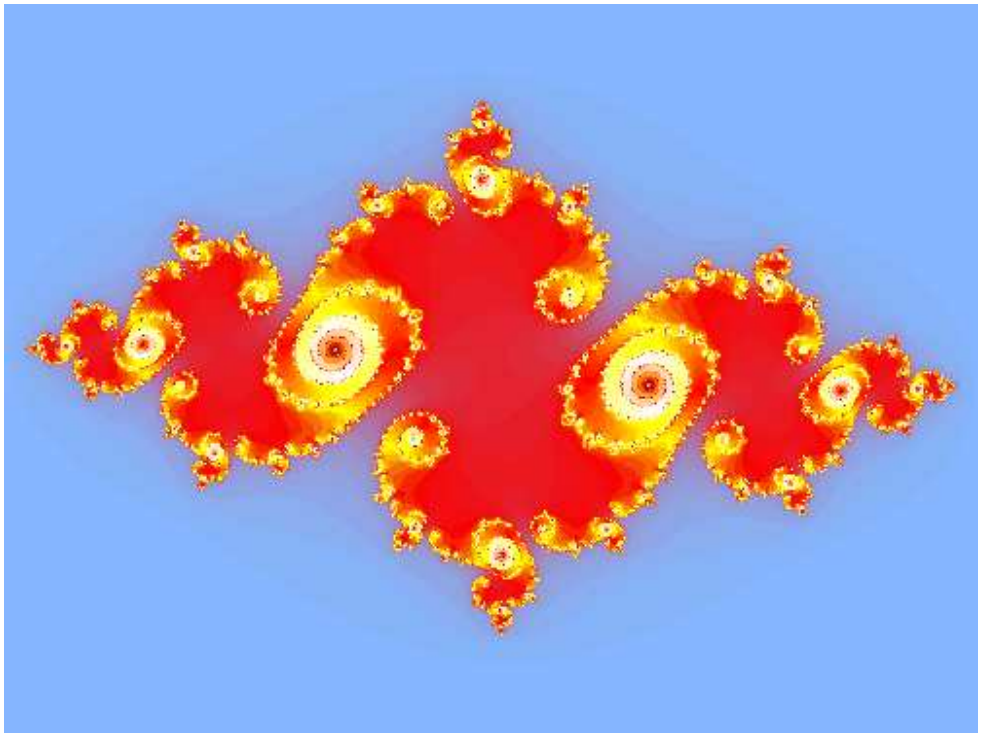
Punkte, für die die x_i über alle Grenzen wachsen, gehören zum sogenannten *escape set*, die Anfangspunkte, deren Folgeglieder begrenzt bleiben, bilden das *prisoner set*. Die letzte Menge enthält zum Beispiel alle x, für die gilt: $x = x^2 + c$ und ist somit nicht leer. Das *Julia set* ist definiert als die Grenze zwischen dem escape set und dem prisoner set. Die Darstellungen dieser Mengen sind für $c \neq 0$ ausgesprochen schön, und es lohnt sich, sie zu berechnen und zu zeichnen. Hier einige Kostproben:



[5] `http://www.geocities.com/CapeCanaveral/2854/`

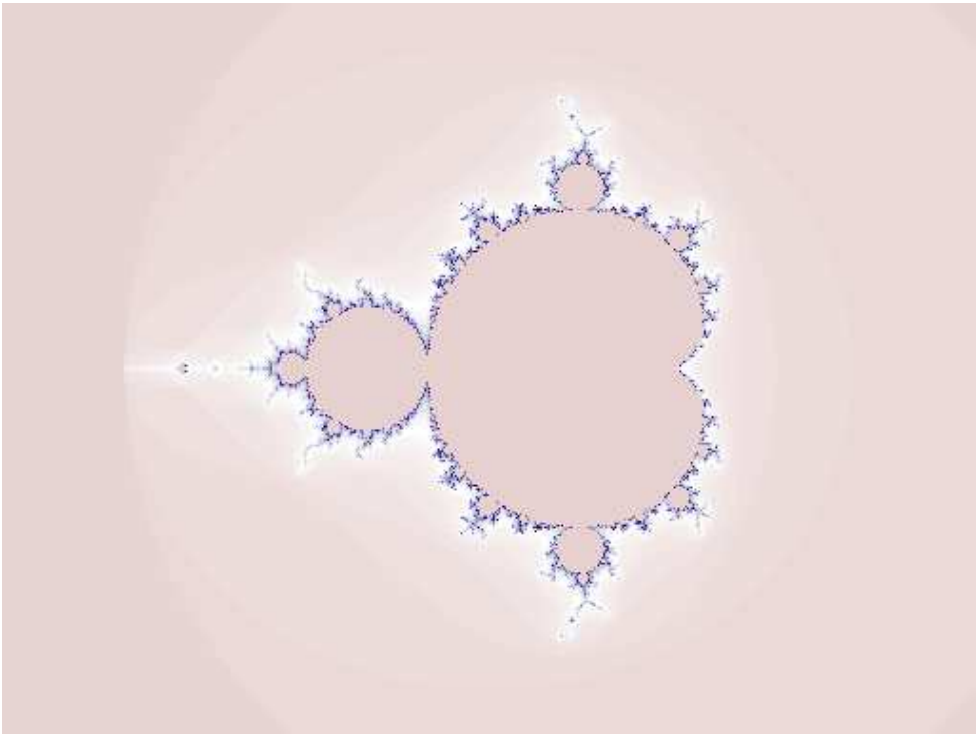


[5] http://www.geocities.com/CapeCanaveral/2854/



[5] http://www.geocities.com/CapeCanaveral/2854/

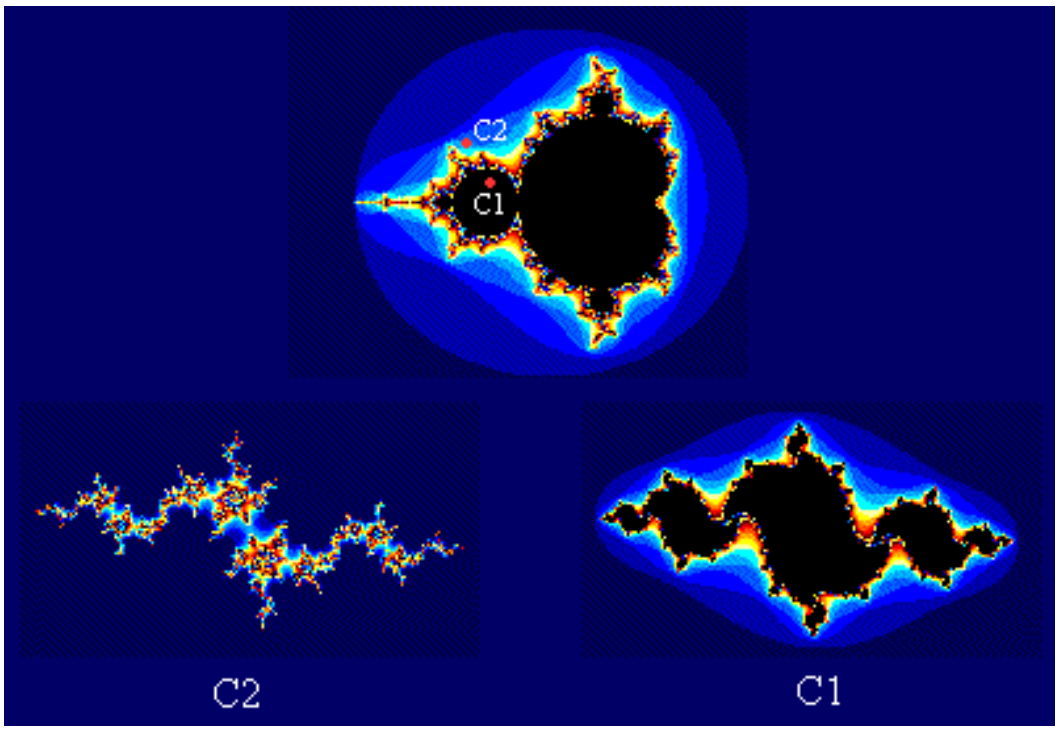
Eine der spannenden Fragen, die sofort von einem Mathematiker aufgeworfen werden, ist: Wenn für jedes c ein anderes schönes Bild entsteht, wie läßt sich diese Galerie ordnen? Einfache Frage, einfache Antwort: Plaziere in einen Raum alle Bilder, die zusammenhängende Julia sets zeigen und in einen zweiten alle die, bei denen die Julia-Menge nicht zusammenhängend ist. Mit dieser Weltsicht läßt sich die Menge $\mathbb{C}$ entsprechend in zwei Teile teilen. Die Menge aller Punkte $c \in \mathbb{C}$, für die die Julia-Menge zusammenhängend ist, wollen wir *Mandelbrotsche Menge* (M) nennen. Und diese Menge können wir wiederum darstellen und erhalten hierdurch das berühmte Apfelmännchen:



Es ist kaum zu glauben, daß man die beschriebenen Überlegungen so implementieren kann, daß sie in zumutbarer Zeit ein Ergebnis hervorbringen. Jedoch haben die Mathematiker immer ein paar Tricks auf Lager. So hat Mandelbrot als erster 1979 ein Apfelmännchen erzeugt mit Hilfe des Zusammenhangs, daß gilt:

$$M = \{c \in \mathbb{C} \mid c \to c^2 + c \to \ldots \quad \text{ist begrenzt}\}$$

Zum Abschluß noch ein Bild für den Zusammenhang von Julia-Mengen und Apfelmännchen.



[5] `http://www.geocities.com/CapeCanaveral/2854/`

Für weitere Fragen siehe [23].

Beispiel-Programm siehe [34] `http://www.th-soft.com/d_wincig.htm`.

4.6.4 Überprüfungen auf Selbstähnlichkeit – Varianz-Zeit-Plot

Der Varianz-Zeit-Plot greift die schon in der einführenden Graphik festgestellte Eigenschaft der Varianzen aggregierter Reihen auf. Wie berichtet gilt für die Varianzen der Blocksummen dividiert durch Blocklänge: Die Varianzen der Mittel von Teilsummen selbstähnlicher Zeitreihen nehmen nicht proportional zur Summandenanzahl ab. Für entsprechende Zeitreihenzufallsvariablen mit endlicher Varianz gilt stattdessen:

$$\mathrm{Var}\left(\sum_{t=1}^{m} X_t/m\right) = c \cdot m^{-\beta} \quad \Rightarrow \quad \log\left(\mathrm{Var}\left(\sum_{t=1}^{m} X_t/m\right)\right) = c^* - \beta \log(m)$$

Wenn also die logarithmierten Varianzen der Mittel von Zeitreihenwerten nicht überlappender Blöcke gegen die logarithmierte Anzahl m abgetragen werden, muß sich eine lineare Beziehung ergeben. Aus der Schätzung des Steigungsparameters kann für realen Zeitreihen der Parameter β und aus diesem über $H = 1 - \beta/2$ der Parameter H geschätzt werden.

Bevor wir uns mit Zeitreihen auseinandersetzen, wollen wir schauen, ob in einer kontrollierten Situation über das Verfahren das β erkannt wird. Nehmen wir an,

daß der Prozeß X selbstähnlich ist mit dem Parameter H sowie die acf

$$\rho_\tau = \tau^{-\beta}$$

besitzt. Dann läßt sich die Varianz von $X^{(m)}/m$ berechnen:

$$\begin{aligned}
\mathrm{Var}(X_t^{(m)}/m) &= \mathrm{Var}\left(\sum_{i=1}^m X_i/m\right) \\
&= \sum_i^m \sum_j^m \mathrm{Cov}(X_i, X_j)/m^2 \\
&= m \cdot \gamma_0/m^2 + 2/m^2 \cdot ((m-1)\gamma_1 + (m-2)\gamma_2 + \cdots + 1\gamma_{m-1}) \\
&= \gamma_0/m + 2/m^2 \cdot (m-1, m-2, \ldots, 1) \cdot (\gamma_1, \gamma_2, \ldots, \gamma_{m-1})^T
\end{aligned}$$

Wir können hiermit folgendes Experiment machen:

Algorithmus:

1. wähle `Gamma0` und `beta`
2. wähle eine Menge von m-Werten: `m.set`
3. berechne mit der obigen Formel für `m.set` die Varianzen von $X_t^{(m)}$
4. plotte die logarithmierten Varianzen gegen die logarithmierten `m`-Werte
5. passe eine Gerade an und bestimme die Steigung als Schätzung für β.

41 ⟨ * 1⟩+ ≡

```
Gamma   <- function(m,beta,c0=1) c0*1/(1:m)^beta
var.x.m <- function(m,beta) Gamma0/m   +  2/m^2*((m-1):1)%*% Gamma(m-1,beta)
Gamma0<-1; beta<-0.8;
m.set<-exp(seq(log(10),log(100000),length=30))
Var.xq<-NULL
for(k in m.set) Var.xq<-c(Var.xq,var.x.m(k,beta))
plot(log(m.set), log(Var.xq))
print(summary(Var.xq))
ind<-floor(length(m.set)/2):length(m.set)
res<-lsfit(log(m.set[ind]), log(Var.xq[ind]))
abline(res)
cat("beta=",beta,", res$coef=",res$coef)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5860  0.6571  0.7359  0.7397  0.8211  0.9091
beta= 0.1 , res$coef= 0.1492577 -0.09893
Mon Dec 09 23:16:00 2002
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.3484  0.4378  0.5483  0.5632  0.6812  0.8309
beta= 0.2 , res$coef= 0.3066673 -0.1969011
Mon Dec 09 23:16:10 2002
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2201  0.3107  0.4378  0.4700  0.6120  0.8309
beta= 0.2 , res$coef= 0.3206302 -0.1991109
Mon Dec 09 23:16:21 2002
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1389  0.2202  0.3486  0.3981  0.5484  0.8309
beta= 0.2 , res$coef= 0.3256485 -0.1997329
Mon Dec 09 23:16:34 2002
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
0.05314 0.10610 0.21100 0.28120 0.41440 0.76350
beta= 0.3 , res$coef= 0.5109181 -0.2992364
Mon Dec 09 23:16:44 2002
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.02082 0.05228 0.13040 0.21110 0.31790 0.70530
beta= 0.4 , res$coef= 0.7128116 -0.3980833
Mon Dec 09 23:16:54 2002
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.003542 0.013970 0.053920 0.136400 0.196800 0.611100
beta= 0.6 , res$coef= 1.159969 -0.5903977
Mon Dec 09 23:17:06 2002
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0007546 0.0044960 0.0254700 0.1001000 0.1312000 0.5400000
beta= 0.8 , res$coef= 1.604229 -0.762587
Mon Dec 09 23:17:19 2002
```

Nun zum `var.time.plot`.

Algorithmus:

1. wähle eine Menge `m.set` von Aggregations- oder Blocklängen `m`
2. für jede Länge `m` aus `m.set`:
   (a) summiere für jeden Block die Zeitreihenwerte und dividiere durch `m`
   (b) berechne die Varianz der Blocksummen
3. plotte die logarithmierten Anzahlen gegen die logarithmierten Varianzen der Blocksummen
4. passe an die Punkte eine Gerade an – dabei sollten die ersten Punkte übergangen werden, vielleicht auch die letzten
5. ermittle aus dem Steigungsparameter eine Schätzung für H.

Wesentliche Größen:

Variable	Bedeutung
`xt`	Zeitreihe
`n.m`	Anzahl der zu betrachtenden Blocklängen
`m.set`	betrachtete Blocklängen
`N`	Länge der Input-Reihe
`K`	Anzahl nicht-überlappender Blöcke
`m`	Blocklänge

42 ⟨*definiere* `var.time.plot` 42⟩ ≡ ⊂ 79

```
var.time.plot<-function(xt,n.m=20){
  N<-length(xt); m.set <- floor(exp(seq(log(10),log(N),length=n.m)))
  result <- NULL
  for(m in m.set){
```

```
    K <- floor(N/m)
    xt.bloecke<-matrix(xt[1:(m*K)],m,K)
    Xm <- apply(xt.bloecke,2,sum) / m
    V  <- var(Xm)
    result <- rbind(result,c(m,V))
  }
  result<-log(result)
  plot(result)
  abline(out<-lsfit(result[-(1:3),1],result[-(1:3),2]))
  paste("var.time.plot - H.dach=",signif(1+out$coef[2]/2,3),
        ", coef=",signif(out$coef[1],3),",",signif(out$coef[2],3))
}
```

4.6.5 Überprüfungen auf Selbstähnlichkeit – Poxplot

Aus der Beziehung über den Quotienten aus Range und Streuung läßt sich ein zweiter Erkennungsplot konstruieren.

Algorithmus:

1. wähle eine Menge `m.set` von Aggregations- oder Blocklängen `m`
2. für jede Länge `m` aus `m.set`:
   (a) berechne für jeden Block Mittelwert und Streuung
   (b) berechne für jeden Block den Range/Streuung
3. plotte die logarithmierten Anzahlen gegen die logarithmierten Quotienten
4. passe an die Punkte eine Gerade an – dabei sollten die ersten Punkte übergangen werden
5. ermittle aus dem Steigungsparameter eine Schätzung für H.

Auf die Berechnung des Maßes für die Spannweite muß noch kurz eingegangen werden:

$$R(m) = \max(0, W_1, W_2, \ldots, W_m) - \min(0, W_1, W_2, \ldots, W_m)$$

Hierbei ist W_k gegeben durch

$$W_k = \sum_{i=1}^{k} X_i - k \sum_{i=1}^{m} X_i/m \qquad \text{für } k = 1, 2, \ldots, m$$

Wesentliche Größen:

Variable	Bedeutung
`xt`	Zeitreihe
`n.m`	Anzahl der zu betrachtenden Blocklängen
`m.set`	betrachtete Blocklängen
`N`	Länge der Input-Reihe
`K`	Anzahl nicht-überlappender Blöcke
`m`	Blocklänge
`M`	Blockmittel
`S`	Blockstandardabweichungen
`Wk`	Spannweiten

43 ⟨*definiere* `poxplot` 43⟩ ≡ ⊂ 79

```
poxplot<-function(xt,n.m=10){
  N<-length(xt)
  m.set <- floor(exp(seq(log(10),log(N),length=n.m)))
  result <- NULL
  for(m in m.set){
    K <- floor(N/m)
    xt.bloecke<-matrix(xt,m,K)
    M <- apply(xt.bloecke,2,mean)
    S <- apply(xt.bloecke,2,function(x) var(x)^0.5)
    Wk<- apply(xt.bloecke,2,cumsum) - outer(1:m,M)
    R <- apply(Wk,2,function(x)max(x)-min(x))
    result <- rbind(result,cbind(m,R/S))
  }
  result<-log(result)
  plot(result)
  abline(out<-lsfit(result[,1],result[,2]))
  paste("poxplot       - H.dach=",signif(out$coef[2],3),
        ", coef=",signif(out$coef[1],3),",",signif(out$coef[2],3))
}
```

4.6.6 Überprüfungen auf Selbstähnlichkeit – Test-Ergebnisse

Mit den zwei definierten Instrumenten zur Erkennung von Selbstähnlichkeit können wir erste Gehversuche anstellen.

44 ⟨*definiere* `selfsim.test` 44⟩ ≡ ⊂ 79

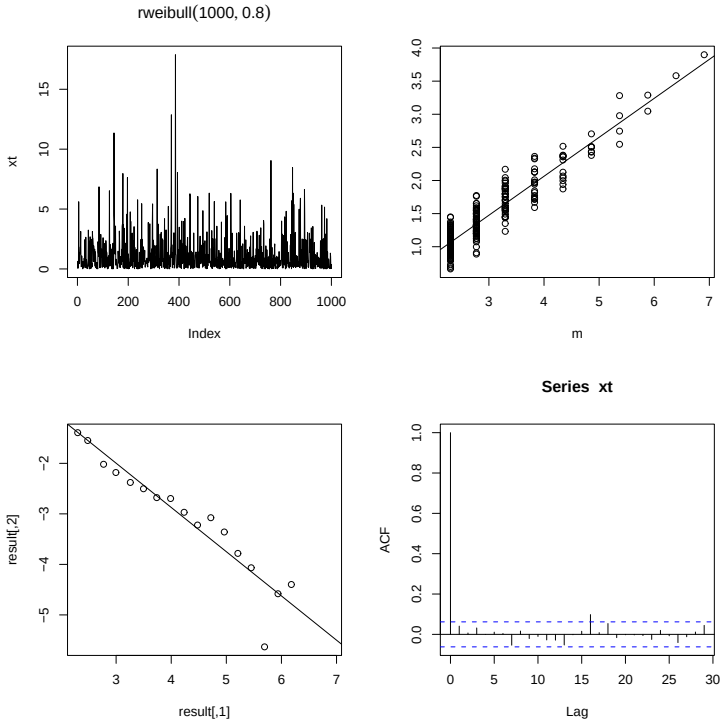
```
selfsim.test<-function(xt){
  par(mfrow=c(2,2))
  plot(xt,type="h",main=substitute(xt))
```

```
  rp<-poxplot(xt)
  rv<-var.time.plot(xt)
  if(!exists("acf")) library(ts) ; acf(xt)
  par(mfrow=c(1,1))
  cat(rp,rv,"",sep="\n")
 }
```

45 ⟨ * 1⟩+ ≡

```
 selfsim.test(rweibull(1000,shape=.8))
```

```
poxplot       - H.dach= 0.588 , coef= -0.289 , 0.588
var.time.plot - H.dach= 0.562 , coef= 0.63 , -0.876
```



46 ⟨ * 1⟩+ ≡

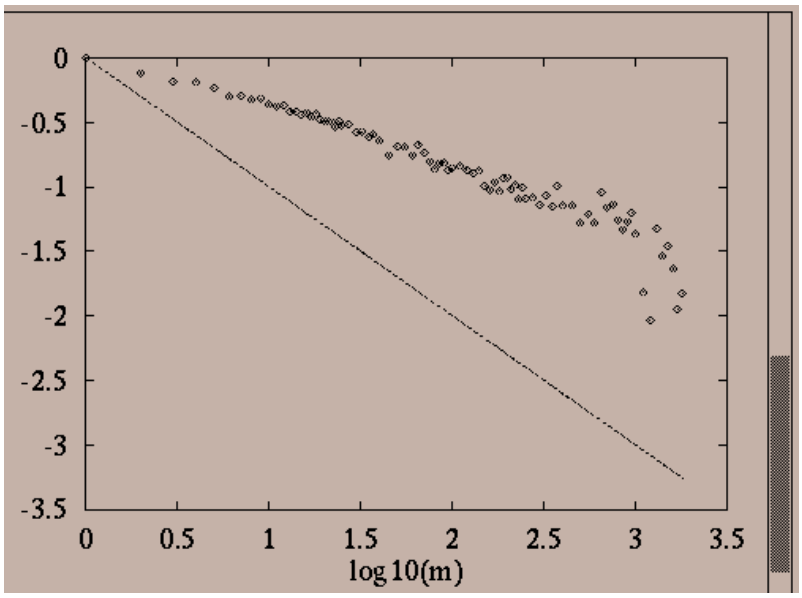
```
 selfsim.test(rpareto(1000))
```

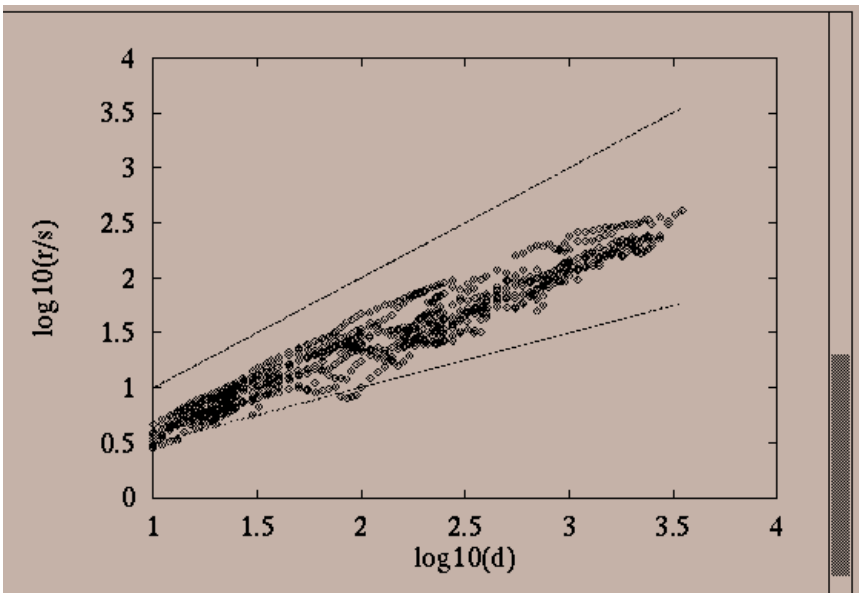
47 ⟨ * 1⟩+ ≡

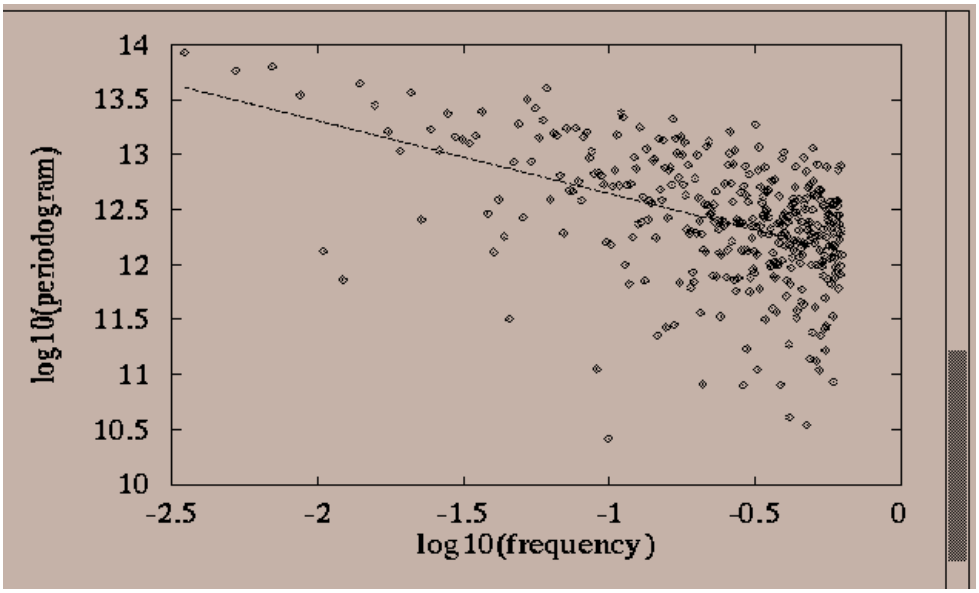
```
 xt<-rnorm(1000)
 selfsim.test(xt[-(1:2)]+0.5*xt[1:998] +.6*xt[-c(1,1000)])
```

4.6.7 Empirische Ergebnisse

Unsere alten Demonstrations-Server-Daten sind nicht als Input der entworfenen Funktionen geeignet, da einfach der Datenumfang zu gering ist. Deshalb zeigen wir einige Ergebnis-Plots von Crovella und Bestavros, [7]. Diese basieren auf einer Analyse von WWW-Traffic-Daten vom 05.02.95 von 16.00 bis 17.00, Computer Science Department, Boston University.







Aus den Graphiken folgen Schätzungen von 0.76 (Varianz-Zeit-Plot), 0.75 (R/S-Plot), 0.83 (Periodogramm-Plot). Damit steht $H = 0.5$ nicht mehr zur Diskussion. Die Kritik ist zutreffend, daß in diesem Papier das Verfahren zur Schätzung von H mittels Periodogramm, einem Zwischenergebnis auf dem Wege der Schätzung des Spektrums, nicht näher vorgestellt wird. Vielleicht kann der interessierte Leser sich dieser Frage als Übung annehmen.

Die beiden Autoren schließen ihre Ergebnisse mit der Bemerkung ab:

> *Thus the results in this section show evidence that WWW traffic at stub networks might be self-similar, when traffic demand is high enough. We expect this to be even more pronounced on backbone links, where traffic form a multitude of sources is aggregated. In addition, WWW traffic in stub networks is likely to become more self-similar as the demand for, and utilization of, the WWW increase in the future.*
>
> [21]

4.6.8 Synthese selbstähnlicher Prozesse

Zur Konstruktion selbstähnlicher Prozesse wollen wir folgendes Konstruktionsprinzip, das auf Mandelbrot zurückgeht, verwenden:

> *A self-similar process may be constructed by superimposing many simply renewal reward processes, in which the rewards are restricted to the values 0 and 1, and in which the inter-renewal times are heavy-tailed.*
>
> [7]

Als schwanzlastige Verteilungen bieten sich Pareto-Verteilungen mit einem Shape-Parameter $\alpha < 2$ an. Wir können also Prozesse betrachten, die zwischen den Zuständen Off und On wechseln und bei denen die Länge der Perioden ohne Zustandsänderung Pareto-verteilt mit $\alpha < 2$ ist. Wenn wir genügend zu äquidistanden Zeitpunkten die Anzahl der On-Zustände solche Prozesse addieren, erhalten wir eine Zeitreihe mit selbstähnlichen Strukturen. Falls für die On-Phasen der Parameter gegeben ist durch α_1 und für die Off-Phasen durch α_2, folgt: $H = (3 - \min(\alpha_1, \alpha_2))/2$ nach Willinger et al., [33].

Algorithmus:

1. realisiere `n.zz` An- und Aus- Ereignisse von `n.proz` Prozessen mit Zwischenankunftszeiten aus Pareto-Verteilungen
2. betrachte die Abfolge der Ereignisse eines Prozesses als abwechselnde An-Ab-Schaltvorgänge
3. bestimme den Ende-Zeitpunkt, bis zu dem für alle Prozesse noch Ereignisse berechnet worden sind
4. entferne alle Ereignisse, die nach dem Ende-Zeitpunkt liegen
5. wähle eine Reihe äquidistanter Zeitpunkte
6. stelle zu den äquidistanten Zeitpunkte die jeweilige Anzahl der An-Zustände fest
7. liefere die Zeitreihe der An-Zustands-Anzahlen als simulierte Reihe ab.

Variable	Bedeutung
`shape[1]`	Shape-Parameter der Pareto-Verteilung des Wartens auf nächstes On
`shape[2]`	Shape-Parameter des Wartens auf nächstes Off
`n.proz`	Anzahl der An-Aus-Prozesse
`n.zz`	Anzahl der An-Ereignisse jedes An-Aus-Prozesses
`zz`	Ereigniszeitmatrix: Ereignisnummer × Prozeß
`on.off.ind`	Indikator für An-Aus-Ereignis
`ind`	Position jedes Ereignisses in der Abfolge aller Ereignisse
`zz.vec`	Ereignis-Zeiten in ihrer Abfolge
`count`	aktuelle Zahl von Ons in jeder Ereigniszeit
`time.end`	ist der letzte Zeitpunkt, an dem alle Prozesse noch Ereignisse zeigen
`time.ok`	dient zur Reduktion der Ereignis-Zeiten und Anzahlen auf die zentrale Arbeitszeit
`N`	ist die Anzahl der zu untersuchenden Zeitpunkte
`time`	Zeitpunkte, in denen die Anzahl der Ereignisse festgestellt werden soll
`n.on.time`	Anzahl der Ereignisse zu den Zeiten `time`

48 ⟨*definiere* `sim.selfsim` 48⟩ ≡ ⊂ 79

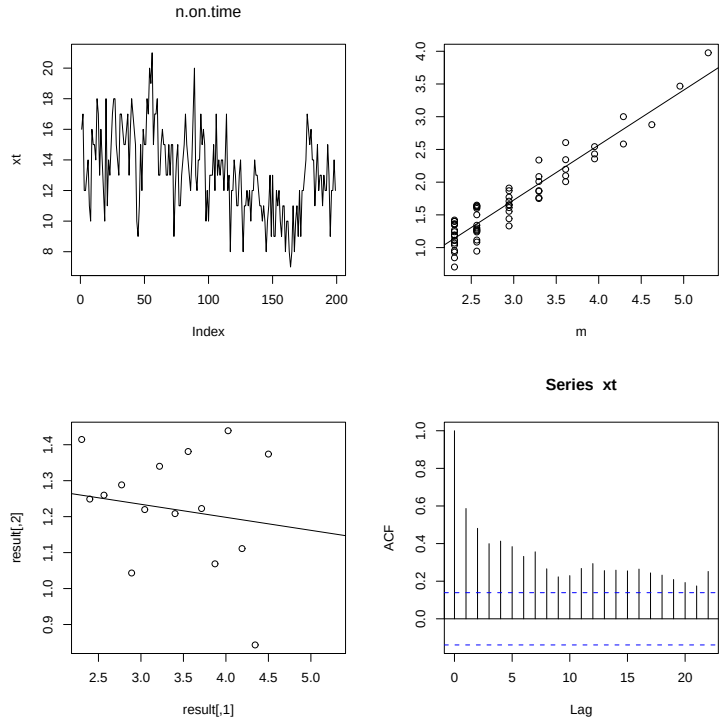
```
sim.selfsim<-function(N=200,shape=.8,n.proz=30,n.zz=1000){
  if(length(shape)==1) shape<-c(shape,shape)
  zz <- rbind(rpareto(n.zz*n.proz,c=shape[1]),rpareto(n.zz*n.proz,c=shape[2]))
  n.zz <- 2*n.zz
  zz   <- apply(matrix(zz,n.zz,n.proz),2,cumsum)
  on.off.ind <- matrix(c(1,-1),n.zz,n.proz)
  ind  <-order(zz); zz.vec <- as.vector(zz[ind])
  count<-cumsum(on.off.ind[ind])
  time.ok <- zz.vec <= (time.end<-min(zz[n.zz,]))
  zz.vec <- zz.vec[time.ok]; count <- count[time.ok]
  time <- seq(0,max(zz.vec),length=N)
  n.on.time<-count[cumsum(hist(zz.vec,breaks=time,plot=FALSE)$counts)]
  n.on.time.red <- n.on.time[n.on.time <= min(zz[n.zz,])]
  return(n.on.time.red)
}
```

Ein kurzer Test geht über alles.

49 ⟨* 1⟩+ ≡

```
n.on.time<-sim.selfsim()
selfsim.test(n.on.time)
```

```
poxplot       - H.dach= 0.84 , coef= -0.792 , 0.84
var.time.plot - H.dach= 0.982 , coef= 1.34 , -0.0362
```



Das ist doch überzeugend — oder?

4.6.9 Ursachen der Selbstähnlichkeit im Internet-Traffic

Wenn wir für die Zeiten zwischen Ereignissen langschwänzige Verteilungen annehmen, ergeben sich selbstähnliche Strukturen. Aber ist das so? Und: Warum ist das so?

Zur ersten Frage konstruieren Crovella und Bestavros einen LLCD-Plot für Übertragungszeiten für einzelne Web-Seiten. Diese Zeiten ermittelten sie durch Abgleich von Informationen der Netzwerkschicht mit semantischen Einheiten auf der Anwenderschicht. Diese Übertragungszeiten werden nun grob als On-Zeiten interpretiert. Die Abkürzung heißt: Log-Log Complementary Distribution-Plot und bedeutet, daß $\log(\hat{F}(x))$ gegen $\log(x)$ abzutragen ist. LLCD-Plot ist also gerade unserer Pareto-Erkennungsplot und auch das Bild in ihrem Aufsatz ähnelt sehr stark unserem Pareto-Erkennungsplot für unsere Zwischenzeiten vom 3.2.1997.

Auch in ihrem Bild scheint eine Pareto-Verteilung nicht brauchbar zu sein, jedoch gibt es in der Mitte einen großen Bereich, für den eine Gerade gar nicht so schlecht paßt. Es resultiert ein $\hat{\alpha}$ von ≈ 1.2. Aus der weiteren Schätzung für

H folgt ein Wert von $\hat{H} = 0.9$. Dieser liegt höher als der direkt aus den Daten geschätzte Wert von $\hat{H} = 0.7$ bis 0.8. Dies ist sicher eine Folge davon, daß die Pareto-Verteilung eben doch nicht so richtig gut paßt.

Und *wieso* besitzen die Übertragungszeiten einen dicken rechten Schwanz, so daß sie mit einer hohen bzw. idealisierterweise nicht-existierenden Varianz ausgestattet sind? Dazu kann man zunächst überlegen, daß die Übertragung proportional mit der Dateigröße wächst, und somit sich die Verteilung der Größe der Dateien auf die Verteilung von Übertragungen niederschlägt. Wenn man also nachweisen kann, daß die Dateigrößen dickschwänzige Verteilungen besitzen, sind aussagekräftige Indizien gefunden. Deshalb wollen wir uns im sogleich folgenden Abschnitt mit Dateigrößen auseinandersetzen.

5 Datenmengen

Unser Interesse für Dateigrößen wird aus zwei Richtungen gespeist: Einerseits besitzen die Größen der Dateien, die über das Netz angefordert werden, für die Planung von Zwischenspeichern – sei es aus algorithmischen oder kapazitätsmäßigen Gründen – eine direkte Relevanz. Andererseits sind wir durch die Diskussion von den Zwischenzeiten, genauer durch Phänomene der *burstiness* des Internet-Traffic, der langen Schwänze sowie geeigneter Modellierungen mit Hilfe von Verteilungen mit nicht existierenden Varianzen ebenfalls auf die Bedeutung der Dateigrößen gestoßen. In diesem Abschnitt werden wir den zweiten Gedankengang fortsetzen, jedoch uns gleichzeitig der Bedeutung des ersten bewußt sein.

Wir sind zuletzt auf die Frage der Dateigrößen gestoßen, weil notwendigerweise der Transport größerer Dateien mehr Zeit kostet. Hiermit lassen sich kurzfristige Intensitätssteigerungen erklären. Im Prinzip setzt die Funktion zur Simulation von selbstähnlichen Prozessen diesen Gedanken um. Jede Seitenanforderung führt zu einer Belastung über eine gewisse Zeitspanne. Dieses wurde mit der On-Zeit modelliert. Zwischen Anforderungen (einer Stelle) entsteht somit ein Loch, was durch die Off-Zeit abgebildet wurde. Mit diesem Ansatz konnte nachgewiesen werden, daß die Summation simulierter einzelner Prozesse (mit heavy tailed Zwischenereigniszeiten – z.B. Pareto-verteilten) genau solche Eigenschaften aufweist, wie sie im realen Netz-Verkehr beobachtet werden. Damit stellen sich, um die letzte Lücke zu schließen, folgende Fragen:

- Welche Verteilungen passen zu den Größen der Dateien, die über Netz ausgetauscht werden?
- Schlagen sich die Verteilungen in entsprechenden Verteilungen für die Transportzeiten nieder?
- Welche Bemerkungen lassen sich zu den Off-Zeiten machen?

5.1 Verteilung von Dateigrößen

Eine genaue Betrachtung zeigt, daß diese Frage auf drei Arten interpretiert werden kann:

1. Wie sind die Größen von auf Servern gespeicherten Dateien verteilt (Angebotsseite)?
2. Wie sind die Größen von übertragenen Dateien verteilt (Nachfrageseite)?
3. Wie sind die Größen der unterschiedlichen, übertragenen Dateien verteilt (Optimaler Cache)?

Die Betrachtung des dritten Punktes bietet sich an, weil diese Daten leicht zu bekommen sind und weil sie auf die Auswirkungen eines optimal funktionierenden Cache-Systems hinweisen. Denn falls jede einmal transportierte Information geeignet zwischengespeichert wird, ist ein weiterer Transport nicht mehr erforderlich. Hierdurch wird zwar ignoriert, daß Dateien von unterschiedlichen Stellen geordert werden, jedoch kann mit der dritten Frage ein Extrempunkt markiert werden. Die Unterscheidung der ersten beiden Fragen beleuchtet, ob eher die Nachfrageseite eine durchschlagende Wirkung auf Phänomene des Netzverkehr hat oder ob schon die Angebotsseite die Rolle Hauptverursacher spielt. Wenden wir uns zunächst der Angebotsseite zu.

5.1.1 Angebotsseite

Zur Untersuchung der Verteilung von Dateigrößen auf Servern wollen wir uns die Situation auf unserem Server anschauen. Für die Dateigrößen vom 03.05.2001 erstellen wir mit `identify.pareto` einen Pareto-Erkennungsplot bzw. einen LLCD-Plot.

50 ⟨*hole Größendaten vom 03.05.01* 50⟩ ≡ ⊂ 51, 52

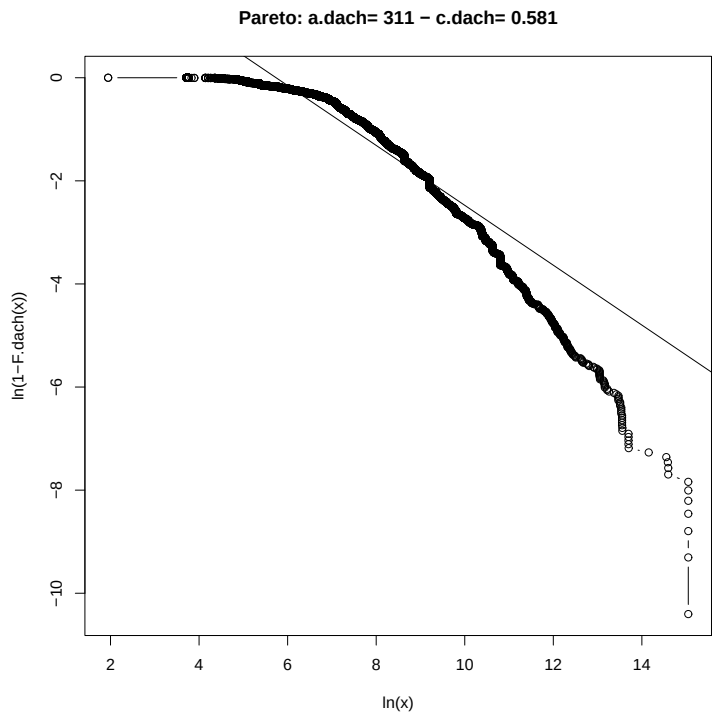
```
if(file.exists("/home/wiwi/pwolf/projekte/explizit/probe/sizes030501",0)){
   x<-scan("/home/wiwi/pwolf/projekte/explizit/probe/sizes030501",0)
}else{
   x<-rpareto(10000)
}
```

51 ⟨* 1⟩+ ≡

⟨*hole Größendaten vom 03.05.01* 50⟩

```
identify.pareto(x[x>0])
```

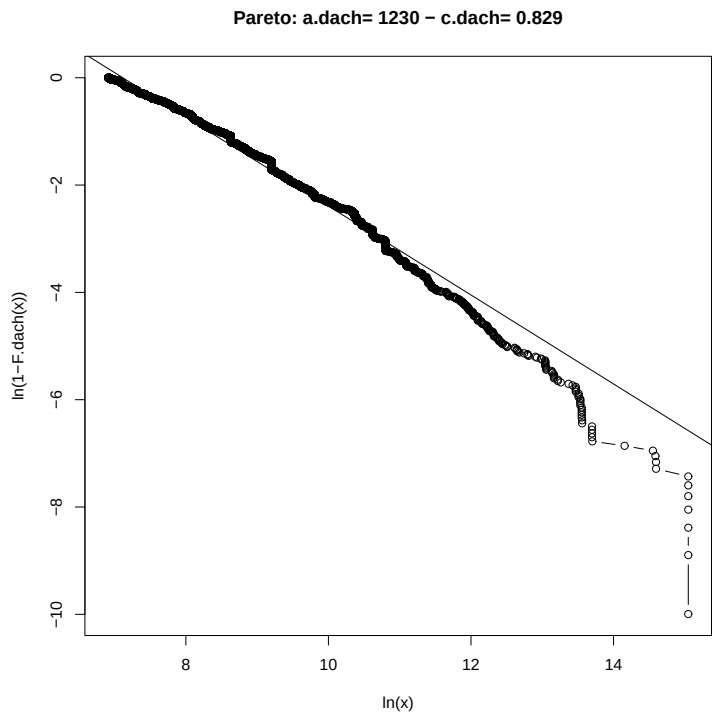
```
Fri Dec 13 17:15:55 2002
     a.dach      c.dach        a.ml        c.ml
310.8484679   0.5808206   7.0000000   5.5150613
```



Ein seltsames Ergebnis: ML-Schätzung liefert einen Wert von 5.5, wogegen das graphische Verfahren 0.58 errechnet. Die Verteilung paßt nicht besonders gut. Das ergibt sich auch aus dem Plot. Abschnittsweise sind jedoch lineare Stücke auszumachen. Wenn wir die etwas größeren Dateien unter die Lupe nehmen, erhalten wir:

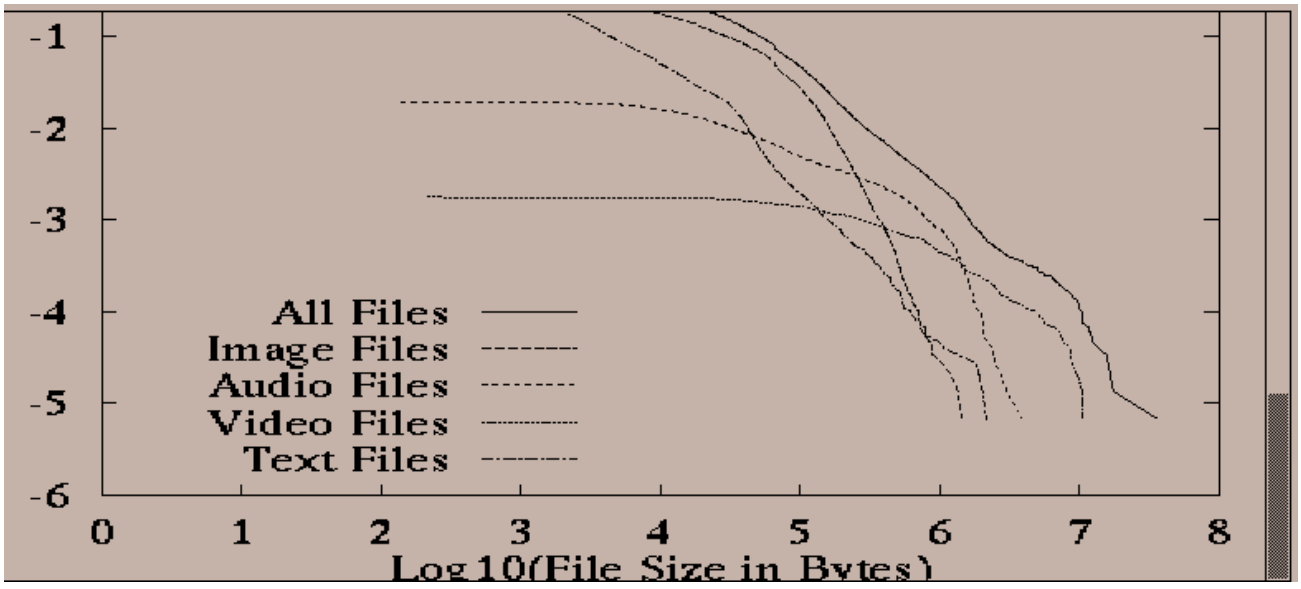
52 ⟨ * 1⟩+ ≡
⟨*hole Größendaten vom 03.05.01* 50⟩
```
identify.pareto(x[x>1000])
```

```
Fri Dec 13 17:20:51 2002
      a.dach       c.dach         a.ml         c.ml
1231.2411809    0.8289883 1002.0000000    1.4122763
```



Jetzt erhalten wir zwar immer noch unterschiedliche Schätzwerte, doch in beiden Fällen Vorschläge für c, bei denen keine Varianz existiert!

Auch Crovella und Bestavros berechneten Pareto-Parameter in der Größenordnung zwischen 1 und 1.2. Andere Autoren haben Werte zwischen 0.9 und 1.1 ermittelt. Als Erklärungen kommen normale Variabilitäten, Unterschiede in der Zeit und eventuell auch Gebiet der betrachteten Server in Betracht. Eine Einteilung der Dateien in Klassen führt zu folgendem Bild



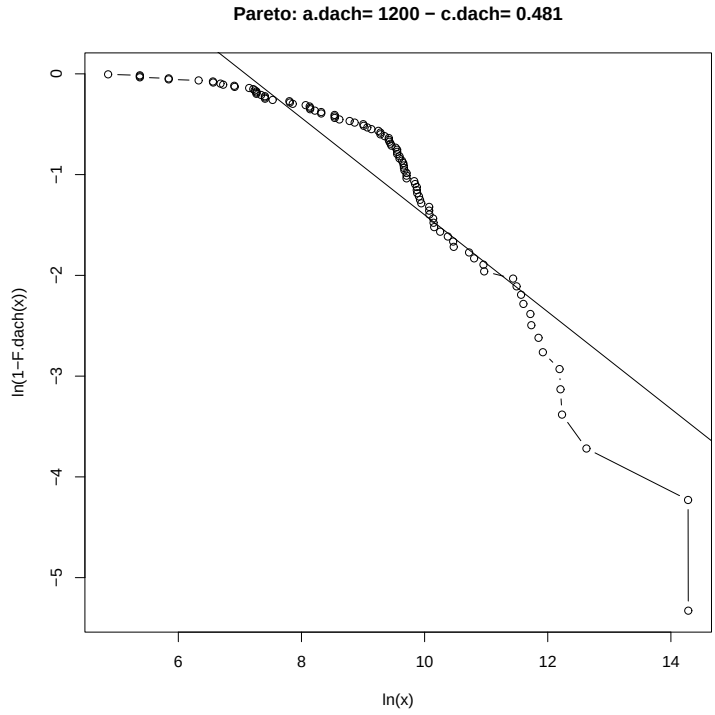
5.1.2 Nachfrageseite und optimaler Cache

Wieder beginnen wir mit Daten, die uns direkt zur Verfügung stehen, und sich mit den passenden Rechten sammeln lassen (z.B. mittels: `ll -R . | grep " 1 " | awk '{print $5} '`).

53 ⟨ * 1⟩+ ≡

```
x<-mengen.03.02.97
identify.pareto(x)
```

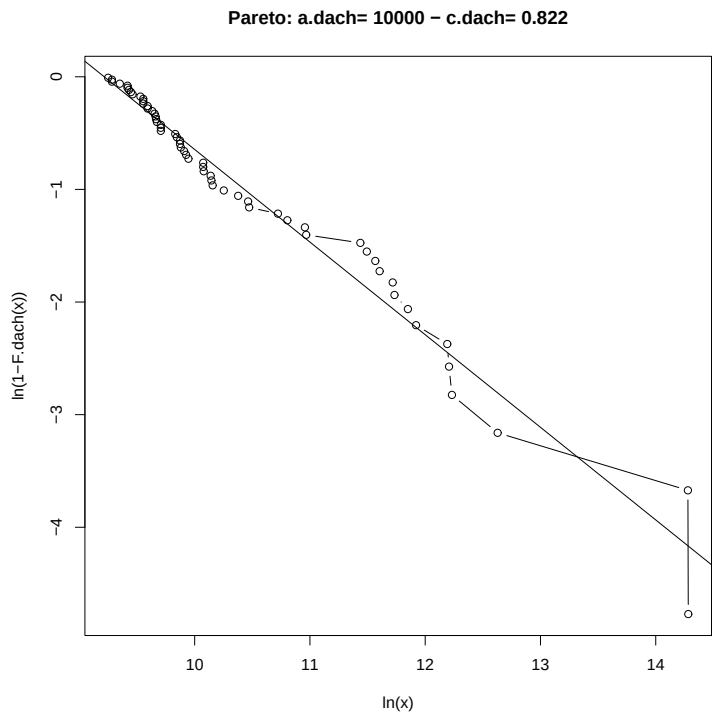
```
Sat Dec 14 11:30:34 2002
      a.dach       c.dach         a.ml         c.ml
1202.5974517    0.4812377  129.0000000    4.3034178
```



54 ⟨ * 1⟩+ ≡

```
x<-mengen.03.02.97
identify.pareto(x[x>10000])
```

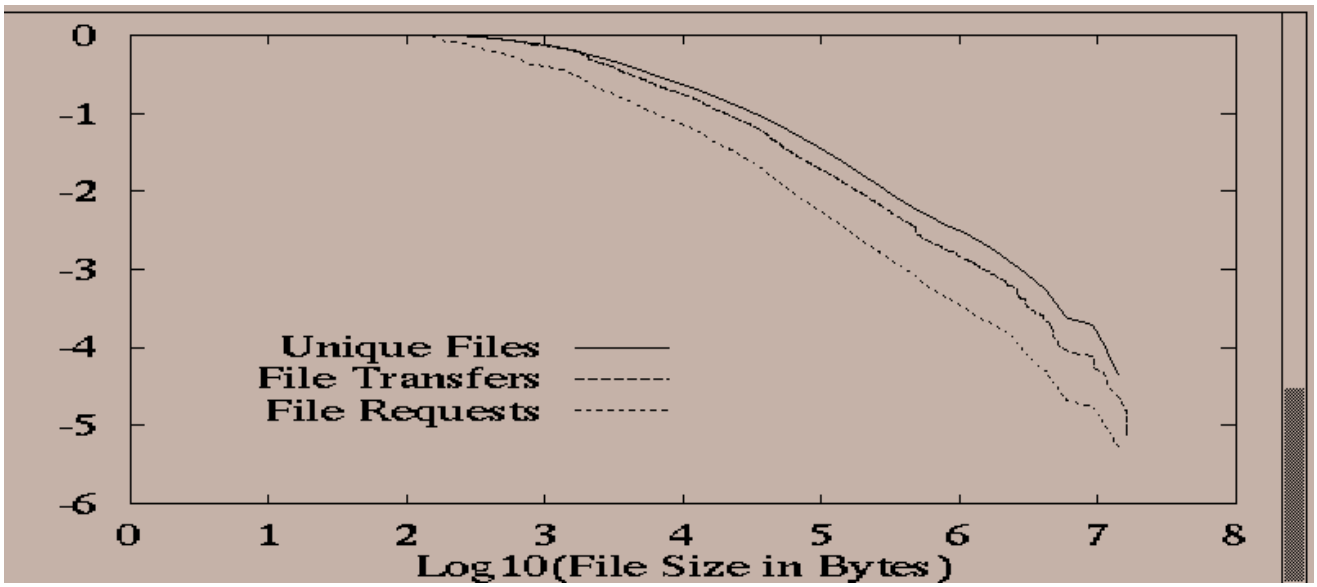
```
Sat Dec 14 11:31:43 2002
      a.dach       c.dach         a.ml         c.ml
1.004827e+04 8.223407e-01 1.039500e+04 1.174987e+00
```



Wie steht es mit der Nachfrageseite bei Crovella und Bestavros? Sie stellen als Argumentationshilfen weitere LLCD-Plots von Dateigrößen zur Nachfrageseite bereit. In folgendem Bild sind drei Situationen zusammengefaßt:

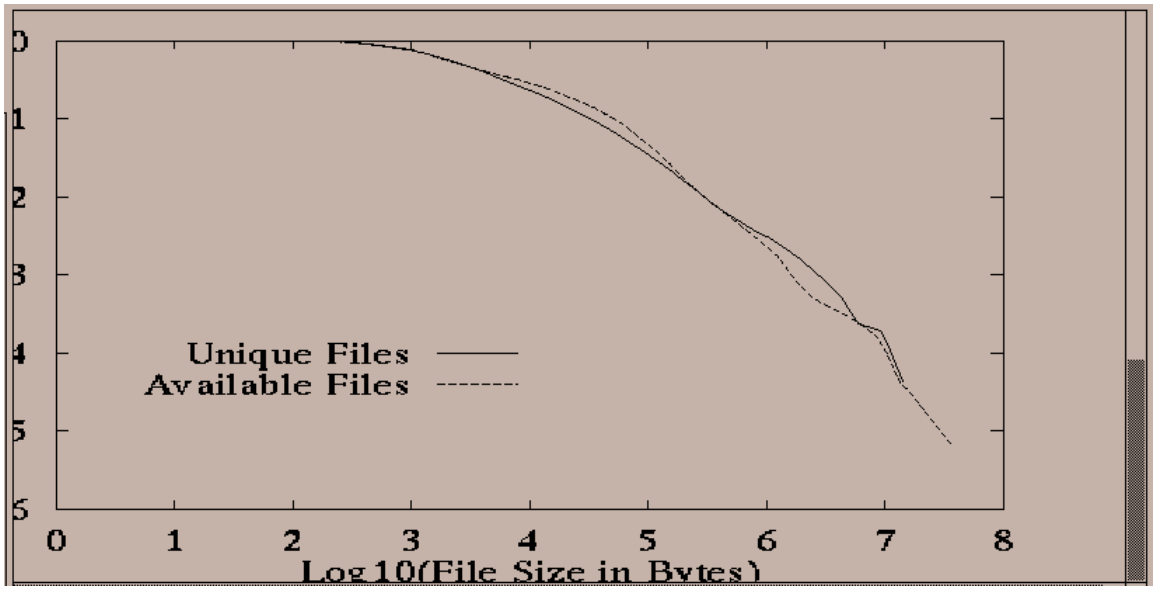
- gepunktet – LLCD-Plot aller angeforderter Dateien
- durchgezogen – LLCD-Plot der unterschiedlichen transportierten Dateien
- gestrichelt – LLCD-Plot der transportierten, nicht aus dem Cache gelieferten Dateien

Die Bilder sehen erstaunlich ähnlich aus und liefern ein α um 1.2 bzw. 1.1. Hier ist das zusammenfassende Bild: (Es sei bemerkt, daß für die Schätzung nur Dateien größer als 10000 Byte berücksichtigt wurden.)



Damit sehen wir eine große Parallelität zur Angebotsseite. Auch wenn sich die Werte ein wenig unterscheiden, liegen auch diese deutlich unter 2.

Crovella und Bestavros zeigen auch eine Gegenüberstellung der angebotenen Dateien und verschiedenen transportierten Dateien:



Sie kommen zu folgendem Resultat:

> *Thus we conclude that as long as caching is effective, the set of files available in the Web is likely to be the primary determiner of heavy-tailed characteristics of files transferred—and that the set of*

> *requests made by users is important. This suggests that available files are of primary importance in determining actual traffic composition, and that changes in user request patterns are unlikely to result in significant changes to the self-similar nature of Web traffic.*
>
> Crovella und Bestavros, [7]

Da grob in verschiedensten File-Systemen verwandte Eigenschaften festgestellt worden sind, ist auch zum Teil die Frage: *Welche Erklärungen lassen sich für die gefundenen Verteilungen angeben?* beantwortet. Wir schließen diesen Abschnitt mit einem weiteren Zitat:

> *In conclusion, these observations seem to show that heavy-tailed size distributions are not uncommon in various data storage systems. It seems that the possibility of very large file sizes may be non-neglibible in a wide range of contexts; and that in particular this effect is of central importance in understanding the genesis of self-similar traffic in the Web.*
>
> Crovella und Bestavros, [7]

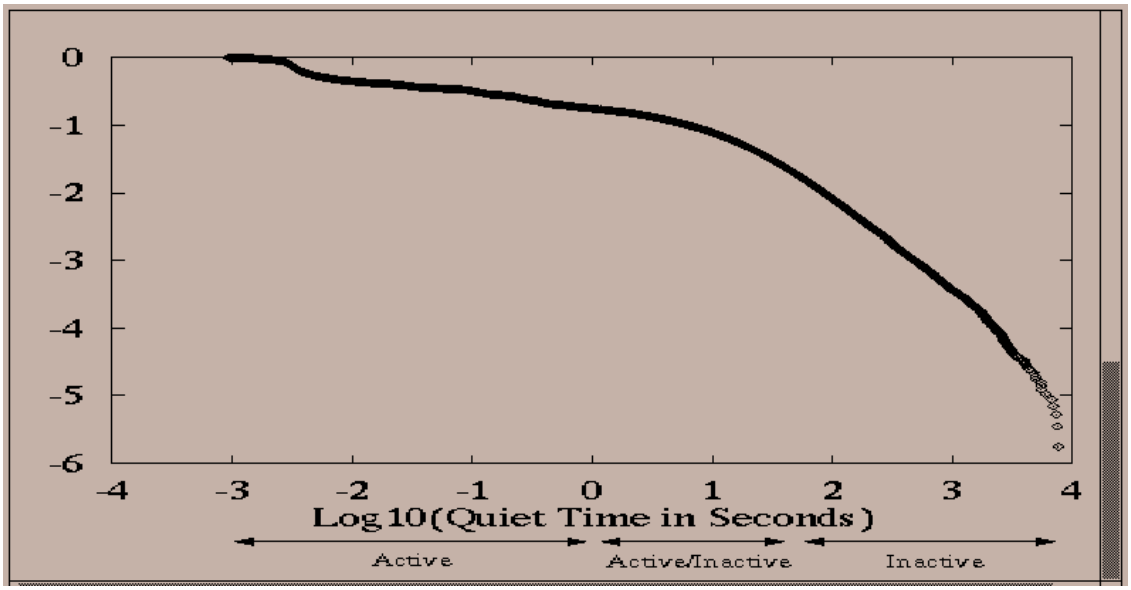
5.1.3 Wirkungen der Dateigrößen-Verteilungen

Schlagen sich die Verteilungen in entsprechenden Verteilungen für die Transportzeiten nieder? Die Antwort lautet ja sowie, daß die Zeiten in der Tat dicke Schwänze aufweisen. Wir belassen es bei diesem intuitiv einsichtigen Ergebnis.

5.2 Phasen geringer Verkehrsintensität

Nachdem die letzten Ausführungen sich mit Transporten – modelliert durch On-Phasen – beschäftigt haben, müssen wir uns abschließend noch mit den Off-Phasen auseinandersetzen. Wenn die Transportzeiten Verteilungen mit dicken Schwänzen besitzen, gilt dieses auch, wie in der oben implementierten Prozeß-Simulation unterstellt, für die Zeit zwischen zwei Anforderungen? Damit wendet sich die Diskussion der Frage zu: *Welche Bemerkungen lassen sich zu den Off-Zeiten machen?*

Crovella und Bestavros zeigen dazu einen LLCD-Plot der von ihnen ermittelten Off-Zeiten:



Er zeigt wieder einmal sehr deutlich, daß ein Pareto-Modell nicht passend ist. Andererseits lassen sich zwei Bereiche ausmachen, in denen grob eine lineare Struktur zu erkennen ist: Der Bereich von Off-Zeiten, die kürzer als eine Sekunde sind, und die übrigen. Für die längeren Ruhezeiten ist damit eine hohe Wahrscheinlichkeitsmasse im rechten Schwanzbereich belegt. Als Erklärung bieten die Autoren an, daß es aktive und passive Ruhezeiten gibt, die sich in ihrer Ursache unterscheiden.

Web-Browser, die eine Seite laden, müssen diese aus verschiedenen Elementen zusammenbauen. Dabei wechseln Anforderungsaktionen mit Aufbereitungsarbeiten ab. Da im Prinzip bis zum Seitenaufbau die Anforderungen noch nicht abgeschlossen sind, sollen die in diesen Phasen entstehenden Pausen als *Active-Off* bezeichnet werden. Ist eine Seite geladen, vergeht manchmal sehr viel Zeit, bis es zur nächsten Anforderung kommt. Diese Phasen, in denen zum Beispiel Anwender die geladene Seite studieren, sollen *Inactive-Off* heißen.

Dieses Modell vermag die beobachtete Struktur zu erklären. Aufgrund ihrer Daten kommen die Autoren zu dem Vorschlag, die Active-Off-Phasen mit Weibull-Verteilungen zu modellieren, wogegen für die Inactive-Regime besser Pareto-Verteilungen Verwendung finden können.

Zusammenfassend schreiben sie:

> *Since we saw in the previous section that ON times were heavy-tailed with* $\alpha \approx 1.0$ *to 1.3, and we see in this section that OFF times are heavy tailed with* $\alpha \approx 1.5$*, we conclude that ON times (and, consequently, the distribution of available files in the Web) are more likely responsible for the observed level of traffic self-similarity, rather than OFF times.*
>
> Crovella und Bestavros, [7]

Dieses Ergebnis, im Besonderen die Pareto-Verteilung, paßt wieder zu den ersten Feststellungen über das Zipfsche Gesetz. Vielleicht ist die (menschliche) Natur in allen möglichen Zusammenhängen so veranlagt, daß sich heavy-tailed Verteilungen einstellen, ob es sich nun um Denkzeiten, Bücherstapel, Verwendung von Vokabeln oder eben Dateigrößen und -nachfragen handelt.

Natürlich gäbe es auch in diesem Kapitel noch viel mehr zu diskutieren. Zum Beispiel können nach den gewonnenen Einblicken detailiertere Fragen gestellt, speziellere Daten erhoben und verarbeitet werden. Zum Beispiel könnte man Wang's Artikel auswerten, in dem die *Arbeitslast* modelliert wird.

6 Cache-Simulation

Beginnen wir mit zwei Zitaten ([9][10]):

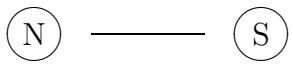
> *We present an artificial workload model of wide-area internetwork traffic. The model can be used to drive simulation experiments of communication protocols and flow and congestion control experiments.*
>
> [9]

> *This paper presents a tool called GISMO (Generator of Internet Streaming Media Objects and workloads). GISMO enables the specification of a number of streaming media access characteristics, including object popularity, temporal correlation of request, seasonal access patterns, user session durations, user interactivity times, and variable bit-rate (VBR) self-similarity and marginal distributions.*
>
> [16]

Zur Simulation von Internet-Traffic für die Untersuchung der Wirkungen von Zwischenspeichern sind verschiedene Fragen zu klären:

- Konstruktion von Modellen — S steht im Folgenden für Server, N für Nachfrager, C für Cache.

  Modell 1:

  Modell 2:

Modell 3: Durch Aufsplittung der Knoten N, S und C läßt sich eine beliebige Kompliziertheit erzielen.

- Formulierung eines Simulationsmodells:

    1. initialisiere Simulation
    2. wiederhole so oft wie verlangt:
        (a) generiere Anforderungen
        (b) suche Dokumente im Cache
        (c) falls Dokumente nicht enthalten:
            i. hole Dokumente
            ii. schaffe ggf. Platz im Speicher
            iii. merke Dokumente
            iv. aktualisiere Verwaltung
        (d) liefere Dokumente aus
        (e) aktualisiere Verwaltung
    3. erstelle Simulationsreport

- Modellierung von Daten / Parameterdefinition

    – Cachegröße
    – Nachfragermenge
    – Servermenge
    – Dokumentmenge von hot ... cold pages
    – Nachfragerauswahl aus Nachfragermenge
    – Dokumentenauswahl
    – Serverauswahl
    – Anforderungszeitpunkte
    – Dateigrößen
    – Übertragungsgeschwindigkeiten
    – Störungen

- Implementierung der Modelle, Planung von Experimenten, Definition von Protokollen Durchführung der Experimente, Verwaltung und Auswertung der Resultate.

Im Folgenden werden wir die Themen *Imitation von Zufälligkeiten* und kurz *Modellierung von Störungen* betrachten, auch wenn andere Teilaspekte ebenso eine Diskussion verdient hätten.

7 Imitation von Zufälligkeiten

Historie: Kryptologie, Student/Gosset/t-Verteilung, Bücher mit Zufallszahlen, heute: algorithmische Generation: Pseudozufallszahlen.

Problemstellung: Wie lassen sich mit Hilfe von Programmen Zahlenströme generieren, die von zufällig entstandenen Zahlenströmen nicht zu unterscheiden sind, die also (fast) die gleichen Eigenschaften besitzen wie zufällig entstandenen Zahlenströme?

Erwartung / Hoffnung: mittels Rechner lassen sich mit wenig Speicherplatz schnell viele gute Zufallszahlen erzeugen.

Bemerkung zur Sprechweise: Im Folgenden wird ab und zu in einer etwas laxen Weise über Zufallszahlen gesprochen. Der Leser möge darüber hinwegsehen, da präzisere Formulierungen viele Sätze unnötig verkomplizieren würden. Zum Beispiel müßte korrekterweise für den Ausdruck *normalverteilte Zufallszahlen* so etwas stehen wie: *Pseudozufallszahlen, deren Eigenschaften (weitgehend) mit denen einer Stichprobe aus einer normalverteilten Grundgesamtheit übereinstimmen.*

7.1 Middle Square Method

Betrachtet werden Zahlen mit $2a$ Stellen. Dann erhalten wir auf Basis einer Inputzahl eine neue nach dem Prinzip:

$$x_{n+1} \leftarrow f(x_n)$$

Die Funktion f quadriert den Input und liefert die mittleren Ziffern als Ergebniszahl ab:

$$f(x) = \left[\frac{x^2}{10^a}\right] \left[-\frac{x^2}{10^{3a}}\right] \cdot 10^{2a}$$

Algorithmus:

1. wähle Stellenanzahl $2a$
2. wähle Startwert
3. $n \leftarrow 0$
4. lege Startwert auf x_n ab
5. quadriere x_n
6. streiche vorn und hinten a Stellen des Ergebnisses weg
7. inkrementiere n
8. lege Ergebnis auf x_n ab und gib x_n aus

9. falls noch mehr Zufallszahlen erforderlich sind, gehe zu Punkt 5.

Hinweis: Zur Erzeugung 0-1-gleichverteilter Zufallszahlen müssen die x-Werte durch $2a$ dividiert werden.

Feststellungen / Eigenschaften:

- das Verfahren besitzt einen Parameter
- es benötigt einen Startwert
- $x_k = x_l \Rightarrow x_{k+1} = x_{l+1}$
- wird der Startwert wieder erreicht, wiederholt sich der Zahlenstrom
- es gibt maximal 10^{2a} verschiedene Werte
- der Zahlenstrom besitzt also eine Periode mit Länge $\leq 10^{2a}$
- es gibt sehr schlechte Startwerte: betrachte für $a = 1$ beispielsweise die Startwerte: 50, 60, 0 – vgl. [19]
- middle square method ist ziemlich ungeeignet, jedoch lassen sich an ihr typische Eigenschaften demonstrieren
- Geschwindigkeitskriterium: einfacher Algorithmus
- Speicherplatzkriterium: nur a und x_k sind zu merken.

7.2 Lineare Kongruenzgeneratoren

Kern eines linearen Kongruenzgenerators ist die Beziehung:

$$x_{n+1} = \sum_{i=0}^{j} a_i x_{n-i} + r \bmod m$$

Aufgrund des Koeffizientenvektors $(a_0, \ldots, a_j)$ sowie r (additive Konstante) und m (Modulus) wird mit Hilfe der Startwerte $x_n, \ldots, x_{n-j}$ die neue Zufallszahl ermittelt.

Spezialffälle.

- Multiplikativer Kongruenzgeneratoren:

$$x_{n+1} = a x_n \bmod m$$

- Gemischter Kongruenzgeneratoren:

$$x_{n+1} = a x_n + r \bmod m$$

- Fibonacci-Generatoren:

$$x_{n+1} = x_n + x_{n-k} \bmod m$$

Implementation. Hier ist eine direkte Implementation:

55 ⟨*definiere* `random.gkg` 55⟩ ≡ ⊂ 79

```
random.gkg<-function(n.max,m,a,r,x.0){
   x<- x.0; res<-1:n.max
   for(i in 1:n.max){
     res[i] <- x <- (a*x+r) %% m
   }
   res
}
```

Einige Experimente. 56 ⟨* 1⟩+ ≡

```
plot(random.gkg(100,m=32,a=9,r=0,x.0=1),type="b")
```

57 ⟨* 1⟩+ ≡

```
plot(random.gkg(100,m=2^20,a=4095,r=12794,x.0=253),type="b")
```

58 ⟨* 1⟩+ ≡

```
plot(random.gkg(100,m=2^32,a=1,r=1,x.0=23714),type="b")
```

59 ⟨* 1⟩+ ≡

```
plot(random.gkg(100,m=16,a=5,r=0,x.0=6),type="b")
```

60 ⟨* 1⟩+ ≡

```
res<-random.gkg(100,m=161,a=5,r=0,x.0=3)
plot(res[-length(res)],res[-1])
```

Erkenntnisse. Für lineare Kongruenzgeneratoren liegen eine Reihe theoretische Erkenntnisse vor, die sich auf Periodenlängen und Strukturen der erzeugten Zahlen beziehen. Unter Beachtung dieser lassen sich Generatoren auswählen, die sich durch eine ausreichende Periode auszeichnen, nicht zu viele Ressourcen erfordern und deren erzeugte Zufallszahlen ganz gute Eigenschaften aufweisen. Als Gegenbeispiel wird immer wieder der Generator `Randu` von IBM angeführt, dessen Zufallszahlen sich im Raum auf 15 Ebenen aufhalten. Hier die Konstruktionsvorschrift:

$$x_{n+1} = 65539 x_n \bmod 2^{31}$$

Mit `random.gkg` lassen sich leicht Realisationen von `Randu` erzeugen. Durch

Definition von Tripeln mittels drei aufeinander folgenden Zahlen erhalten wir Punkte im $\mathbb{R}^3$, deren Projektion wir uns im $\mathbb{R}^2$ anschauen können. Spannend wird das ganze, wenn wir um die Wolke herumgehen können und so nach den 15 Ebenen suchen können. Dieses ist gleichbedeutend damit, daß wir die Wolke rotieren und immer von demselben Standpunkt aus betrachten. Sei X eine $(n \times 3)$-Matrix, in der in jeder Zeile die Koordinaten eines Punktes stehen. Die Rotation um den Winkel α um die zweite Achse, die y-Achse, läßt sich durch eine Rotationsmatrix T bewerkstelligen.

$$X^* = X \times T = X \times \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ \sin\alpha & 0 & \cos\alpha \\ 0 & 1 & 0 \end{pmatrix}$$

Zum Verständnis kann man zum Beispiel überlegen, daß die Zeilenvektoren $E_1 = (1, 0)$ und $E_2 = (0, 1)$ durch folgende Operation im Uhrzeigersinn gedreht werden:

$$\begin{pmatrix} E_1 \\ E_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

Für andere Drehachsen müssen die Zeilen und Spalten der Matrix X entsprechend umsortiert werden. Durch eine Abfolge von Drehungen läßt sich folglich jede Perspektive realisieren. Nach dieser Idee wollen wir eine mit `randu` erzeugte Punktwolke inspizieren. Das spannenste Bild wird auf `tripel.save` festgehalten.

61 ⟨ * 1⟩+ ≡

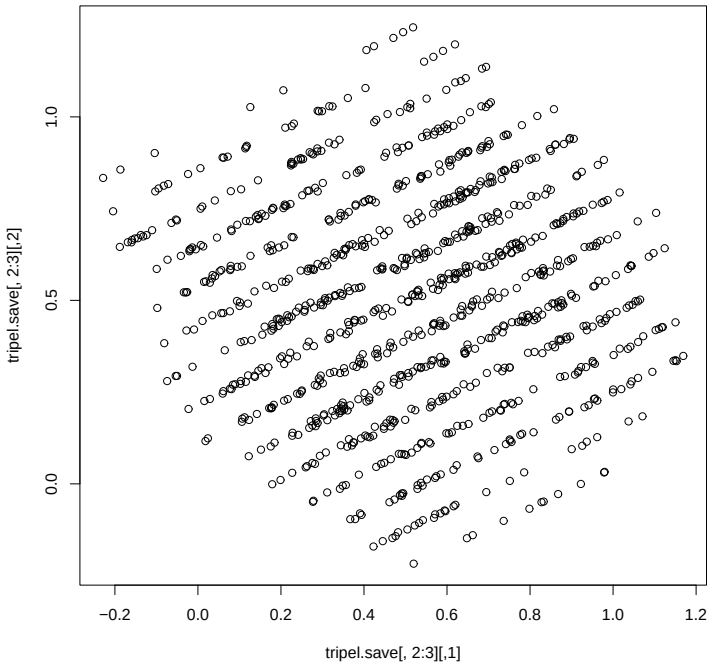
```
# Generierung von Zufallszahlen
res<-random.gkg(1000, 2^31, 65539, 0, 100000)/2^31 # randu
  # res<-runif(100)
  # res<-kg(100, 32, 5, 1, 1)/32
# Definition der Rotationsmatrix
tripel<-cbind(res[-c(length(res),length(res)-1)],
              res[-c(1,length(res))], res[-c(1:2)])
alpha<- 2*2*pi/360; ca<-cos(alpha); sa<-sin(alpha)
beta <- 2*2*pi/360; cb<-cos(beta);  sb<-sin(beta)
gamm <- 2*2*pi/360; cg<-cos(gamm);  sg<-sin(gamm)
rot1<-matrix(c(ca,sa,0,-sa,ca,0,0,0,1),3,3)
rot2<-matrix(c(cb,0,-sb,0,1,0,sb,0,cb),3,3)
rot3<-matrix(c(1, 0, 0, 0,cg,sg,0,-sg,cg,0),3,3)
rot <- rot1%*%rot2%*%rot3
# Initialplot
plot(tripel[,c(2,3)] ,xlim=c(0,1), ylim=c(0,1),type="p")
# Plot aus verschiedenen Perspektiven
wait <- F
for(i in 1:60) {
  points(tripel[,c(2,3)],col="white")
  tripel<-tripel%*%rot
```

```
  points(tripel[,c(2,3)],col="black") # ,type="b")
  Sys.sleep(.03); if(wait & i>18 & i<23) Sys.sleep(1)
  if(i==21) tripel.save <- tripel
}
```

Hier ein Blick auf die 15 Ebenen aus einer verräterischen Perspektive.

62 ⟨ * 1⟩+ ≡

```
plot(tripel.save[,2:3])
```



Noch schöner ist es, interaktiv das Gebilde zu betrachten. Auch das ist möglich:

63 ⟨ * 1⟩+ ≡

```
res<-random.gkg(1000, 2^31, 65539, 0, 100000)/2^31 # randu
tripel<-cbind(res[-c(length(res),length(res)-1)],
              res[-c(1,length(res))], res[-c(1:2)])
spin3R(tripel)
```

Die Definition von `spin3R` ist in einem eigenen Papier sowie im Anhang zu finden. Siehe zum Beispiel:

```
http://www.wiwi.uni-bielefeld.de/
... StatCompSci/wolf/spin3R/spin3R.ps
```

7.3 Generatoren von R

Die bisherigen Ausführungen zeigen einige prinzipielle Probleme bei der algorithmischen Generation von Zufallszahlen. Jetzt stellt sich die Frage, wie Software-Produkte sich aus der Affäre ziehen. Dazu finden wir zum Beispiel in der Hilfe von R:

```
The currently available RNG kinds are given below.  'kind' is
partially matched to this list.  The default is
'"Marsaglia-Multicarry"'.


'"_W_i_c_h_m_a_n_n-_H_i_l_l"'
     The seed, '.Random.seed[-1] == r[1:3]' is an
     integer vector of length 3, where each 'r[i]' is in '1:(p[i]
     - 1)', where 'p' is the length 3 vector of primes, 'p =
     (30269, 30307, 30323)'. The Wichmann-Hill generator has a
     cycle length of 6.9536e12 (= 'prod(p-1)/4', see Applied
     Statistics (1984) 33, 123 which corrects the original
     article).


'"_M_a_r_s_a_g_l_i_a-_M_u_l_t_i_c_a_r_r_y"':
     A multiply-with-carry RNG is used, as
     recommended by George Marsaglia in his post to the mailing
     list 'sci.stat.math'. It has a period of more than 2^60 and
     has passed all tests (according to Marsaglia).  The seed is
     two integers (all values allowed).


'"_S_u_p_e_r-_D_u_p_e_r"':
     Marsaglia's famous Super-Duper from the 70's.
     This is the original version which does not pass the MTUPLE
     test of the Diehard battery.  It has a period of about
     4.6*10^18 for most initial seeds.  The seed is two integers
     (all values allowed for the first seed: the second must be
     odd).

     We use the implementation by Reeds et al. (1982-84).

     The two seeds are the Tausworthe and congruence long
     integers, respectively.  A one-to-one mapping to S's
     '.Random.seed[1:12]' is possible but we will not publish one,
     not least as this generator is not exactly the same as that
     in recent versions of S-PLUS.


'"_M_e_r_s_e_n_n_e-_T_w_i_s_t_e_r":'
     From Matsumoto and Nishimura (1998). A
     twisted GFSR with period 2^19937 - 1 and equidistribution in
     623 consecutive dimensions (over the whole period).  The
     ''seed'' is a 624-dimensional set of 32-bit integers plus a
     current position in that set.


'"_K_n_u_t_h-_T_A_O_C_P":'
     From Knuth (1997), as updated in Knuth (2002). A
     GFSR using lagged Fibonacci sequences with subtraction. That
```

```
     is, the recurrence used is

            X[j] = (X[j-100] - X[j-37]) mod 2^30

     and the ``seed'' is the set of the 100 last numbers (actually
     recorded as 101 numbers, the last being a cyclic shift of the
     buffer).  The period is around 2^129.

'"_K_n_u_t_h-_T_A_O_C_P-_2_0_0_2":'
     The 2002 version which not backwards
     compatible with the earlier version: the initialization of
     the GFSR from the seed was altered. So although workspaces
     with a saved `.Random.seed' will work unchanged, the sequence
     set by `set.seed' is different in the two versions.  R did
     not allow you to choose consecutive seeds, the reported
     `weakness', and already scrambled the seeds.
```

Wir wollen uns einmal den Super-Duper anschauen, da dieser auch in S-Plus 3.4 verwendet wird. Vergleiche dazu auch [1], Seite 656.

```
Random number generation in S-PLUS is adapted from
Marsaglia (1973); see Kennedy and Gentle (1980) for
background information.
```

Die Zufallszahlen werden durch Kombination der Zahlenströme von zwei Generatoren durch eine *exclusive-or*-Operation erzielt. Den ersten Typ haben wir bereits kennengelernt. Bei dem zweiten handelt es sich um einen Tausworthe- oder Shift-Generator. Auch um noch ein zweites Konstruktionsprinzip kennenzulernen, wollen wir uns Shift-Generatoren anschauen.

7.4 Tausworthe-Generatoren

Für den Einstieg sei zunächst verwiesen auf [15]. Aus diesem Buch sind auch die folgenden Gedanken übernommen worden. Gegeben sei ein Bitfolge $b_0, \ldots, b_{n-1}$. Dann wird das nächste Bit b_n unter Einbeziehung des q-elementigen Bitvektors $c_0, \ldots, c_{q-1}$ ermittelt nach der Vorschrift:

$$b_n = c_{q-1} \wedge b_{n-1} \oplus c_{q-2} \wedge b_{n-2} \oplus \cdots \oplus c_0 \wedge b_{n-q}$$

mit $\wedge$ als logischer *und*-Operation und $\oplus$ als *exclusive-or*. Zur Vereinfachung werden wir auf das $\wedge$-Zeichen verzichten:

$$b_n = c_{q-1}b_{n-1} \oplus c_{q-2}b_{n-2} \oplus c_0 b_{n-q}$$

Durch Einführung eines Indexshiftoperators D (delay), für den gilt: $Db_n = b_{n+1}$ und $D^i b_n = DD^{i-1}b_n$, können wir schreiben:

$$b_n = D^q b_{n-q} = c_{q-1}D^{q-1}b_{n-q} \oplus \cdots \oplus c_0 D^0 b_{n-q}$$

Zur weiteren Vereinfachung der Schreibweise können wir auf die nun gleichen Terme b_{n-q} verzichten. Die *exclusive-or*-Operationen können durch eine "Addition mod 2" beschrieben werden:

A	B	$\oplus$	A+B mod 2	A−B mod 2
0	0	0	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	0	0

Hierdurch kommen wir zu der Gleichung

$$0 = D^q - \sum_{i=1}^{q} c_{q-i} D^{q-i} \bmod 2$$

Da Addition und Subtraktion in obiger Gleichung zum gleichen Resultat führen, folgt

$$0 = D^q + \sum_{i=1}^{q} c_{q-i} D^{q-i} \bmod 2$$

Der Berechnungsprozeß für das neue Bit läßt sich somit charakterisieren durch das *charakteristische Polynom*

$$p(x) = x^q + \sum_{i=1}^{q} c_{q-i} x^{q-i}$$

Erkenntnisse. Mit solchen Polynomen lassen sich Eigenschaften von Shift-Generatoren beschreiben: Die kleinste Zahl n, für die $x^n - 1$ durch $p(n)$ teilbar ist, liefert die Periodenlänge. Maximal läßt sich durch Generatoren, zu denen ein Polynom q-ten Grades gehört, eine Periode von $2^q - 1$ erzielen. Polynome mit einer solchen Periodenlänge heißen *primitive Polynome*. Wir werden jedoch nicht weiter in die Welt der primitiven Polynome vorstoßen. Vergleiche dazu: [17] – Seite 31, 439 ff – und [15] sowie [27].

Beispiel. Gegeben sei das Polynom

$$p(x) = x^7 + x^3 + 1$$

Dann gehört dazu die Beziehung

$$D^7 b_{n-7} + D^3 b_{n-7} + b_{n-7} = 0 \bmod 2$$

und wir erhalten

$$b_n + b_{n-4} + b_{n-7} = 0 \bmod 2$$

oder

$$b_n \oplus b_{n-4} \oplus b_{n-7} = 0.$$

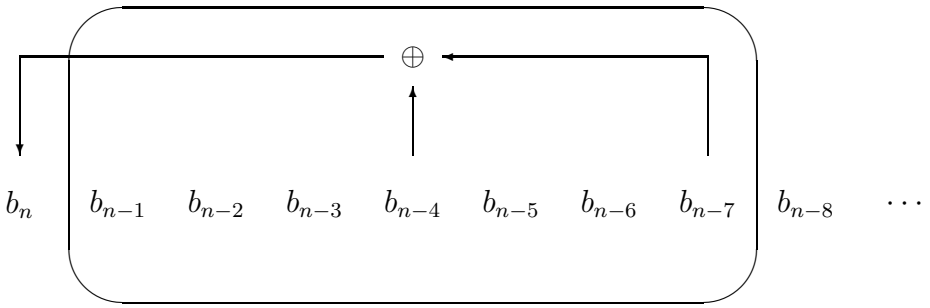
Damit ist der Generator

$$b_n = b_{n-4} \oplus b_{n-7}.$$

bzw.

$$b_n = b_{n-4} + b_{n-7} \bmod 2$$

beschrieben. Graphisch läßt sich der Erzeugungsprozeß wie folgt darstellen:



Nach dem Mechanismus ergibt sich für den Generator die folgende aus [15] kopierte Sequenz, wobei die Bits in der Reihenfolge $b_0, b_1, b_3, \ldots$ notiert sind.

```
1111111 0000111 0111100 1011001 0010000 0010001 0011000
1011101 0110110 0000110 0110101 0011100 1111011 0100001
0101011 1110100 1010001 1011100 0111111 1000011 1000000
```

Wir wollen mal schauen, ob wir diese Sequenz rekonstruieren können.

64 ⟨*definiere* `random.tg` 64⟩ ≡ ⊂ 79

```
random.tg<-function(x,p1=4,p2=7,reverse=F,Print=T){
  if(length(x)==1) x<-substring(x,1:nchar(x),1:nchar(x))
  if(reverse) x<-rev(x)
  x<-as.numeric(x)
  bi <- (x[p1] + x[p2]) %% 2
  x<-c(bi,x); if(reverse) x<-rev(x)
  x<-paste(x,collapse=""); if(Print) cat(x,"\n")
  invisible(x)
}
```

65 ⟨* 1⟩+ ≡

```
x<-rep(1,7)
```

```
for(i in 1:20) x<-random.tg(x)
"end of job"
```

```
01111111
001111111
0001111111
00001111111
100001111111
1100001111111
11100001111111
011100001111111
1011100001111111
11011100001111111
111011100001111111
1111011100001111111
01111011100001111111
001111011100001111111
1001111011100001111111
0100111011100001111111
1010011110111000011111111
1101001111011100001111111
0110100111101110000111111111
0011010011110111000011111111
Tue Jan 14 15:47:31 2003
[1] "end of job"
```

Wenn man berücksichtigt, daß die jüngsten Bits immer links stehen, erkennt man, daß die Funktion das Beispiel reproduziert.

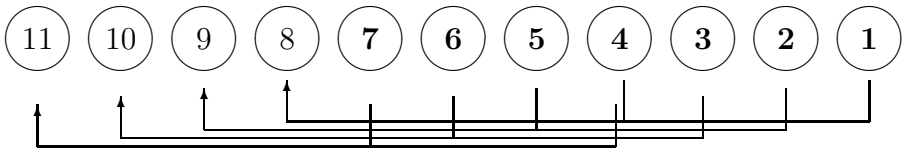
Wir wollen nun der Frage nachgehen, wie sich ein Tausworthe-Generator geschickt implementieren läßt. Betrachten wir wieder den Beispiel-Generator

$$b_n = b_{n-4} + b_{n-7} \bmod 2.$$

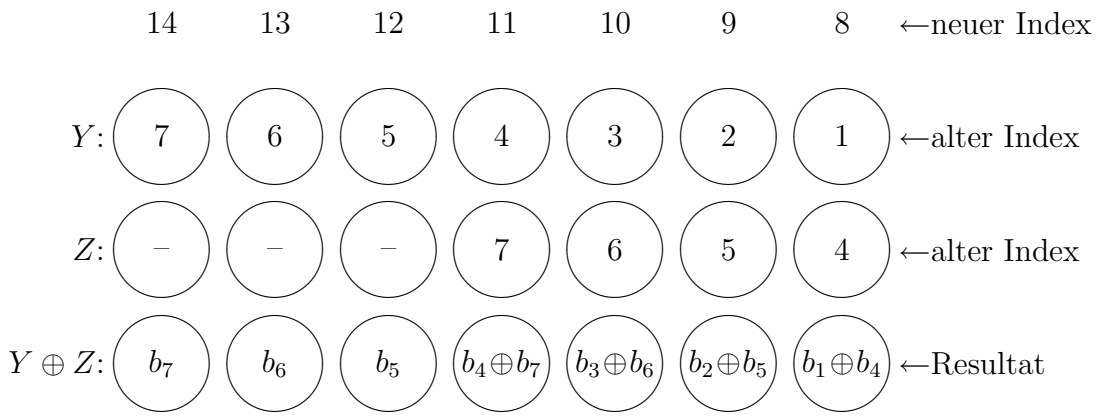
Stellen wir uns vor, daß die 7 zuletzt erzeugten Bits auf dem Vektor X stehen, das neuste Bit ganz links. Weiter wollen wir überlegen, wie es gelingt, die nächsten 7 Bits zu generieren. Hierzu sind zunächst folgende Operationen erforderlich:

$$\begin{array}{lcl} \text{Bit } 4 \oplus \text{Bit } 1 & \Rightarrow & \text{Bit } 8 \\ \text{Bit } 5 \oplus \text{Bit } 2 & \Rightarrow & \text{Bit } 9 \\ \text{Bit } 6 \oplus \text{Bit } 3 & \Rightarrow & \text{Bit } 10 \\ \text{Bit } 7 \oplus \text{Bit } 4 & \Rightarrow & \text{Bit } 11 \end{array}$$

Zusammenfassend zeigt folgende Zeichnung, daß sich die nächsten vier Bits unabhängig voneinander aus den vorliegenden 7 Bits erzeugen lassen.



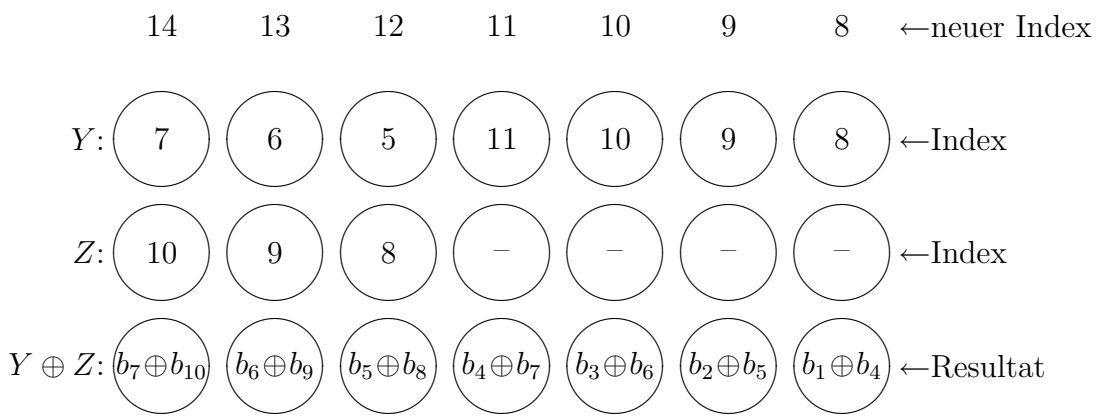
Parallele Verarbeitung. Wenn wir eine Recheneinheit mit parallelen Berechnungsfähigkeiten haben, können wir also im ersten Schritt die alten 7 Bits auf die neuen Plätze kopieren. Bezeichnen wir diese neuen 7 Plätze mit Y. Nun legen wir diesen Vektor auch auf Z ab und verschieben ihn drei Einheiten nach rechts, indem wir von links mit Nullen auffüllen. Wenn wir nun Y und Z untereinanderschreiben, stehen für die ersten vier neuen Bits die entsprechenden Elemente untereinander. Eine bitweise $\oplus$-Operation erledigt den Job:



Wir stellen fest, daß wir die letzten vier der neuen 7 Bits erfolgreich berechnet haben. Welche Operationen sind für die fehlenden drei Bits zu erledigen?

$$\begin{aligned} \text{Bit } 8 \oplus \text{Bit } 5 &\Rightarrow \text{Bit } 12 \\ \text{Bit } 9 \oplus \text{Bit } 6 &\Rightarrow \text{Bit } 13 \\ \text{Bit } 10 \oplus \text{Bit } 7 &\Rightarrow \text{Bit } 14 \end{aligned}$$

Im letzten Ergebnisvektor $Y \oplus Z$ stehen für diese Operationen die notwendigen Bits geschickterweise auf den ersten drei Plätzen. schon passende Operanden. Legen wir den Ergebnisvektor wieder unter Y ab, dann brauchen wir nur noch die Bits 8 bis 10 in Position bringen. Dazu kopieren wir Y nach Z und verschieben Z um vier nach links:



Wir sehen, daß der Ergebnisvektor die erwünschten 7 Bits enthält. Mit dieser Einsicht können wir den Prozeß formalisieren.

Für

$$b_n = b_{n-p} + b_{n-q} \bmod 2.$$

ergibt sich der

Algorithmus:

1. kopiere alten Bitvektor der Länge q nach Y
2. kopiere Y nach Z und shifte Z um $(q-p)$ Elemente nach rechts
3. bilde $Y \oplus Z$ und lege das Ergebnis auf Y ab
4. kopiere Y nach Z und shifte Z um p Einheit nach links
5. bilde $Y \oplus Z$ und liefere Ergebnis ab

Die Sprache `C` verfügt über bitweise Shiftoperationen. Mit diesen läßt sich der beschriebene Algorithmus, sofern auf `y` die jüngsten q Bits stehen, umsetzen durch:

```
z = y >> (q-p)
y = y ^ z
z = y << p
y = y ^ z
```

Die Operation `^` steht dabei für *exclusive-or*, `>> sh` für einen Rechtsshift um `sh` Elemente, `<< sh` für einen Linksshift um `sh` Elemente. Mit der alternativen Notation des Generators

$$b_n = b_{n-(q-r)} + b_{n-q} \bmod 2.$$

verändert sich der Algorithmus in kürzester Form zu:

```
y = y ^ y >> r
y = y ^ y << q-r
```

Wir wollen mal schauen, ob wir dieses im R-Code wiederfinden.

```
double unif_rand(void)
{
    double value;

    switch(RNG_kind) {

    case WICHMANN_HILL:
        I1 = I1 * 171 % 30269;
        I2 = I2 * 172 % 30307;
        I3 = I3 * 170 % 30323;
        value = I1 / 30269.0 + I2 / 30307.0 + I3 / 30323.0;
        return value - (int) value;/* in [0,1) */

    case MARSAGLIA_MULTICARRY:/* 0177777(octal) == 65535(decimal)*/
        I1= 36969*(I1 & 0177777) + (I1>>16);
        I2= 18000*(I2 & 0177777) + (I2>>16);
        return ((I1 << 16)^(I2 & 0177777)) * i2_32m1; /* in [0,1) */

    case SUPER_DUPER:
        /* This is Reeds et al (1984) implementation;
         * modified using __unsigned__  seeds instead of signed ones
         */
        I1 ^= ((I1 >> 15) & 0377777); /* Tausworthe */
        I1 ^= I1 << 17;
        I2 *= 69069;            /* Congruential */
        return (I1^I2) * i2_32m1; /* in [0,1) */

    case MERSENNE_TWISTER:
        return MT_genrand();

    case KNUTH_TAOCP:
    case KNUTH_TAOCP2:
        return KT_next() * KT;

    case USER_UNIF:
        return *((double *) User_unif_fun());

    default:/* can never happen (enum type)*/ return -1.;
    }
}
```

Quellcode R-1.6.1, `RNG.c`

Tatsächlich! Unter dem Fall *Super-Duper* finden wir als Kernoperationen einen Rechtsshift um 17 und einen Linksshift um 15, so daß offensichtlich als Generator

$$b_n = b_{n-17} + b_{n-32} \bmod 2$$

Verwendung findet. Dieses Ergebnis entspricht übrigens auch den Angaben aus [31] Seite 125:

> *The integer* `tausval` *follows a 32-bit Tausworthe generator (of period 4 292 868 097* $< 2^{32} - 1$*):*
>
> $$b_i = b_{i-p} \texttt{ xor } b_{i-(p-q)}, \qquad p = 32, \quad q = 15$$
>
> Venables und Ripley, [31]

Als Hausaufgabe sei überlegt, was die zusätzlichen Operationen des *Super-Duper*-Generators für einen Zweck haben.

Wir wollen nun einmal schauen, ob wir den Algorithmus abbilden können.

66 ⟨*definiere* `random.tg.shift` 66⟩ ≡ ⊂ 79

```
random.tg.shift<-function(x,p1=4,p2=7,reverse=F,Print=T,Show=F){
  if(length(x)==1) x<-substring(x,1:nchar(x),1:nchar(x))
  if(reverse)      x<-rev(x)
  y<-as.numeric(x[1:p2])
  if(Show) cat("Input:  ",y,"\n")

  z<-y
  z<-c(rep(0,p2-p1),z[1:p1])      # shift-right um p2-p1
  if(Show) cat("Rshift: ",z,"\n")

  z<-y<-( y + z ) %% 2            # y xor z
  if(Show) cat("y xor z:",y,"\n")

  z<-c(z[(p1+1):p2],  rep(0,1+p2-p1)) # shift-left um p1
  if(Show) cat("Lshift: ",z,"\n")

  y<-( y + z ) %% 2               # y xor z
  if(Show) cat("y xor z:",y,"\n")

  x<-c(y,x); if(reverse) x<-rev(x)
  x<-paste(x,collapse=""); if(Print) cat(x,"\n")
  invisible(x)
}
```

67 ⟨* 1⟩+ ≡

```
x<-rep(1,7)
random.tg.shift(x,Show=T)
```

```
Input:   1 1 1 1 1 1 1
```

```
Rshift:  0 0 0 1 1 1 1
y xor z: 1 1 1 0 0 0 0
Lshift:  0 0 0 0 0 0 0
y xor z: 1 1 1 0 0 0 0
11100001111111
Tue Jan 14 17:35:09 2003
[1] "11100001111111"
```

68 ⟨ * 1⟩+ ≡

```
x<-rep(1,7)
x<-random.tg.shift(x,reverse=T,Show=T)
x<-random.tg.shift(x,reverse=T,Show=T)
```

```
Input:   1 1 1 1 1 1 1
Rshift:  0 0 0 1 1 1 1
y xor z: 1 1 1 0 0 0 0
Lshift:  0 0 0 0 0 0 0
y xor z: 1 1 1 0 0 0 0
11111110000111
Input:   1 1 1 0 0 0 0
Rshift:  0 0 0 1 1 1 0
y xor z: 1 1 1 1 1 1 0
Lshift:  1 1 0 0 0 0 0
y xor z: 0 0 1 1 1 1 0
111111100001110111100
Tue Jan 14 17:36:21 2003
[1] "111111100001110111100"
```

Offensichtlich arbeitet unser Generator einwandfrei, so daß wir uns an die Rekonstruktion der oben aufgelisteten Zufallszahlen wagen können.

69 ⟨ * 1⟩+ ≡

```
x<-rep(1,7)
for(i in 1:20) x<-random.tg.shift(x,reverse=T,Print=F)
x<-substring(x,1:nchar(x),1:nchar(x))
x<-matrix(x,ncol=7,byrow=T)
cbind(apply(x,1,paste,collapse=""))
```

```
Tue Jan 14 17:46:26 2003
      [,1]
[1,] "1111111"
[2,] "0000111"
[3,] "0111100"
[4,] "1011001"
```

```
 [5,] "0010000"
 [6,] "0010001"
 [7,] "0011000"
 [8,] "1011101"
 [9,] "0110110"
[10,] "0000110"
[11,] "0110101"
[12,] "0011100"
[13,] "1111011"
[14,] "0100001"
[15,] "0101011"
[16,] "1110100"
[17,] "1010001"
[18,] "1011100"
[19,] "0111111"
[20,] "1000011"
[21,] "1011110"
```

Eine genaue Betrachtung zeigt, daß die zuletzt erzeugten 7 Bits nicht mit denen aus dem Buch von Jain übereinstimmen. Wer hat da einen Fehler gemacht?

$U(0,1)$-Zufallszahlen. Mit Hilfe von Bitfolgen gibt es übrigens verschiedene Möglichkeiten, Zufallszahlen zu generieren. Durch Auswahl von jeweils k Bits und deren Interpretation als ganze Zahlen erhalten wir Werte aus der Menge $\{0, 1, \ldots, 2^k - 1\}$. Durch Division dieser Werte bekommen wir wiederum Vorschläge aus $[0, 1)$.

7.5 Marsaglia-Multicarry

Und was meint George Marsaglia zu seinem Super-Duper-Generator?

`http://www.mathematik.uni-bielefeld.de/~sillke/ALGORITHMS/random/marsaglia-c`

```
From: George Marsaglia <geo@stat.fsu.edu>
Newsgroups: sci.math
Subject: Re: A RANDOM NUMBER GENERATOR FOR C
Date: Tue, 30 Sep 1997 05:29:35 -0700
Organization: Florida State University

I have often been asked to suggest random number generators
for use in C.  Many good ones are available through the net,
but they often require  complicated code and various
ways to use or initialize through calls to subroutines.
The following suggestion has these properties:
     Seems to pass all tests of randomness put to it.

     Has much much longer period, >2^60, than most
     system generators, <2^32,  and versions can be
```

```
     combined to get periods > 2^90, 2^120, etc.

     Exploits the feature of C that provides segments
     of in-line code through #define statements.  Thus
     random integers or reals can be put directly into
     expressions, avoiding the cost of calls to subroutines.

     Uses what I view as the most promising new method
     for random number generation: multiply-with-carry.

Here is what you do: keep the following six lines of code
somwhere in your files.

   #define znew  ((z=36969*(z&65535)+(z>>16))<<16)
   #define wnew  ((w=18000*(w&65535)+(w>>16))&65535)
   #define IUNI  (znew+wnew)
   #define UNI   (znew+wnew)*2.328306e-10
   static unsigned long z=362436069, w=521288629;
   void setseed(unsigned long i1,unsigned long i2){z=i1; w=i2;}

Whenever you need random integers or random reals in your
C program, just insert those six lines at (near?) the beginning
of the program. In every expression where you want a random real
in [0,1) use UNI, or use IUNI for a random 32-bit integer.
No need to mess with ranf() or ranf(lastI), etc, with their
requisite overheads.  Choices for replacing the two multipliers
36969 and 18000 are given below.  Thus you can tailor your own
in-line multiply-with-carry random number generator.

This section is expressed as a C comment, in case you want to
keep it filed with your essential six lines:
   /* Use of IUNI in an expression will produce a 32-bit unsigned
    random integer, while UNI will produce a random real in [0,1).
    The static variables z and w can be reassigned to i1 and i2
    by setseed(i1,i2);
   You may replace the two constants 36969 and 18000 by any
   pair of distinct constants from this list:
   18000 18030 18273 18513 18879 19074 19098 19164 19215 19584
   19599 19950 20088 20508 20544 20664 20814 20970 21153 21243
   21423 21723 21954 22125 22188 22293 22860 22938 22965 22974
   23109 23124 23163 23208 23508 23520 23553 23658 23865 24114
   24219 24660 24699 24864 24948 25023 25308 25443 26004 26088
   26154 26550 26679 26838 27183 27258 27753 27795 27810 27834
   27960 28320 28380 28689 28710 28794 28854 28959 28980 29013
   29379 29889 30135 30345 30459 30714 30903 30963 31059 31083
   (or any other 16-bit constants k for which both k*2^16-1
   and k*2^15-1 are prime)*/

I don't program in C, so the above comes from looking in manuals,
(the mathematics, of course, is my own and I am pretty sure of it,
and pretty sure of the randomness through my Diehard battery,
available in
          The Marsaglia Random Number CDROM
                        with
          The Diehard Battery of Tests of Randomness
```

```
at    http://www.cs.hku.hk
or    http://www.stat.fsu.edu
Theory behind multiply-with-carry is in the file mwc1.ps
in the postscript directory of the CDROM).

I welcome suggestions if I have misinterpreted C conventions or
standards.  But interest in random number generators seems great
enough to risk cluttering up the news group with yet another dumb
post.

--
George Marsaglia geo@stat.fsu.edu
```

... und deshalb ist die Voreinstellung bei R entsprechend.

7.6 Zufallszahlen und Zufälligkeit

Jetzt können wir auf verschiedene Weisen Pseudo-Zufallszahlen erzeugen, und zwar durchaus schnell, doch wie steht es mit der Qualität? Sind die sich ergebenden Zahlenströme zufällig? Natürlich nicht! Sollen sie ja gar nicht. Sie sollen nur so aussehen wie mit Hilfe von Zufallsprozessen erzeugte Zahlen. Also deren Charakteristika besitzen. Oder besser keine offensichtlichen Eigenschaften von Nicht-Zufallsprozessen besitzen, eben keine erkennbaren Strukturen besitzen. Dem eingesetzten Mechanismus ist nicht direkt anzusehen, inwieweit solche Wünsche in Erfüllung gehen. So lesen wir zum Beispiel bei Knuth den Hinweis zu den Shift-Generatoren:

> *Caution: Several people have been trapped into believing that this random bit-generation technique can be used to generate random whole-word fractions* $(X_0X_1 \ldots X_{k-1})_2, (X_kX_{k+1} \ldots X_{2k-1})_2, \ldots$*; but it is actually a poor source of random fractions, even though the bits are individually quite random.*
>
> [17], Seite 32

Und auch Jain weist auf Probleme hin:

> *A disadvantage of Tausworthe generators is the observation by Tootill et al. (1971) that while the sequence may produce good test results over the complete cycle, it may not have satisfactory local behavior. In particular, it performed negatively on the runs up and down test. Although the first-order serial correlation is almost zero, it is suspected that some primitive polynomials may give poor high-order correlation. Later, in Section 27.5, it is shown that not all primitive polynomials are equally good.*
>
> [15]

Als Reaktion auf solche Schwierigkeiten, bietet sich an, die Parameter sehr sorgfältig zu wählen. Weiterhin lassen sich die Ströme von mehrerer Zufallszahlengeneratoren kombinieren, um die Schwächen der einzelnen zu verdecken, wie das zum Beispiel beim Super-Duper-Generator geschehen ist. Eine kleine Übersicht, welche Generatoren Verwendung eingesetzt werden, findet man bei Jain. Dort werden auch verschiedene Mythen diskutiert, die nicht zutreffende Vorurteilte aufzeigen.

- *A complex set of operations lead to random results.*
- *A single test such as the chi-square test is sufficient to test of the goodness of a random-number generator.*
- *Random numbers are unpredictable.*
- *Some seeds are better than others.*
- *Accurate implementation is not important.*
- *Bits of successive words generated by a random-number generator are equally randomly distributed.*

[15], 26.6

Zusammenfassend läßt sich sagen, daß man sich zunächst die Rolle überlegen sollte, wofür man genau die Zufallszahlen benötigt und welche Arten von Strukturen besonders üble Konsequenzen nach sich ziehen würden. Dann kann man Generatoren auswählen, die über allgemein gute Eigenschaften verfügen. Man sollte jedoch im Zweifelsfall den verwendeten Strom von Zufallszahlen auf Brauchbarkeit untersuchen. Zumindest sollte man verschiedene Startwerte verwenden und bei großen Diskrepanzen auch die Generatoren kritisch beäugen. Hierzu können Tests auf Zufälligkeit zur Anwendung kommen.

7.6.1 Tests auf Zufälligkeit

Aus mehreren Gründen erscheint es sinnvoll, auf Zufälligkeitstests einzugehen:

- zur direkten Überprüfung eines vorliegenden Stroms von Zufallszahlen
- zum Verständnis, warum bestimmte Generatoren als brauchbar angesehen werden
- zur Demonstration, mit welcher Vielzahl von Perspektiven auf einen Gegenstand geschaut werden kann
- zur Sensibilisierung bezüglich der Frage, was eigentlich Zufälligkeit heißt und wie schwer es ist, die Vorstellungen in Worte zu fassen

Die Frage: *What is a random sequence* wird von Knuth im Kapitel 3.5 seines Werkes *The Art of Computer Programming* behandelt. Hier findet man zum Beispiel zwei Zitate, die gut zu den obigen Gedanken passen.

> *A random sequence is a vague notion embodying the idea of a sequence in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests, traditional with statisticians and depending somewhat on the uses to which the sequence is to be put.*
>
> von D.H. Lehmer übernommen aus [17]

und

> *The sequence (1) is random if it has every property that is shared by all infinite sequences of independent samples of random variables from the uniform distribution.*
>
> von J.N. Franklin übernommen aus [17]

Wir wollen uns an dieser Stelle exemplarisch eine Folge von Ziffern betrachten.

```
592653589793238462643383279 50
```

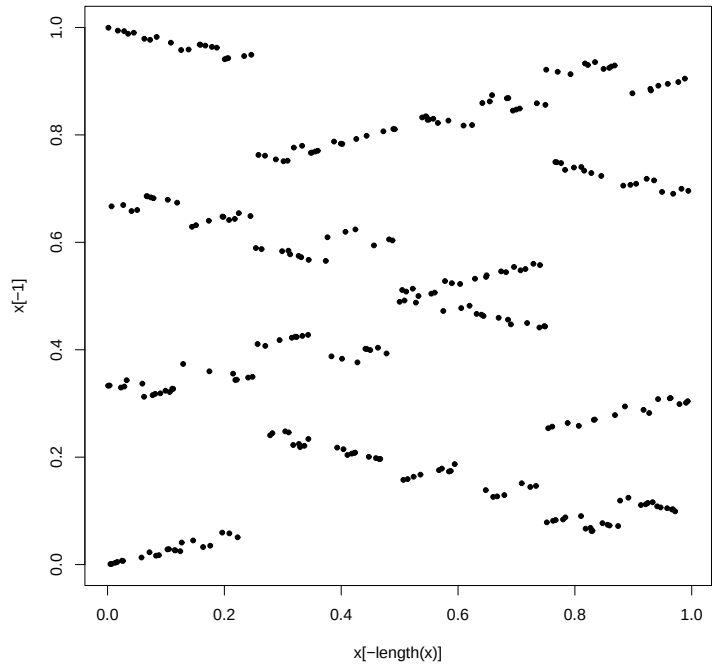
Kommt dem Leser diese Folge bekannt vor? Würde sie unseren Vorstellungen von Zufälligkeit entsprechen? Genauere Betrachtungen zeigen, daß die Folge einige Besonderheiten aufweist: Es lassen sich Paare aufeinanderfolgender Zahlen entdecken, die in dieser Folge mehrfach vorkommen. Weiterhin enthält die Folge keine einzige 1:

```
  ++-------------++
  ||             ||
592653589793238462643383279 50
        ||||||       ||||||
        ||||++------++||||
        ||++----------++||
        ++--------------++
```

Das folgendes Bild zeigt, daß sich bei Shift-Generatoren auch sehr schöne Strukturen ergeben können. Es ist nach [26] konstruiert worden.

70 ⟨ * 1⟩+ ≡

```
x<-rep(1,11)
for(i in 1:300) x<-random.tg.shift(x,p1=9,p2=11,reverse=F,Print=F)
x<-as.numeric(substring(x,1:nchar(x),1:nchar(x)))
L<-11
x<-(matrix(x,ncol=L,byrow=T)%*%2^{0:{L-1}})
x<- x / 2^L
plot(x[-length(x)],x[-1],pch=20)
```



χ^2-Test. Beginnen wir einfach. Aus statistischer Sicht läuft eine Untersuchung der Häufigkeiten von einzelnen Ziffern bzw. die Überprüfung, ob die Ziffernhäufigkeiten zu einer diskreten Gleichverteilung passen, auf einen Anpassungstest hinaus. χ^2-Test, richtig geraten. Zur Erinnerung: Die Abweichungen der Häufigkeiten werden in folgender Prüfgröße zusammengefaßt:

$$X^2 = \sum_{i=1}^{k} \frac{(n_i - np_i)^2}{np_i} = \frac{1}{n} \sum_{i=1}^{k} \frac{n_i^2}{p_i} - n$$

Hierbei steht n für Stichprobenumfang (zum Beispiel Anzahl der betrachteten Ziffern), k für Anzahl der gewählten Klassen (zum Beispiel $k = 10$ Ziffern), p_i für die Klassenwahrscheinlichkeiten (zum Beispiel $p_i = 0.1$), n_i für die beobachteten Häufigkeiten.

Die genaue Verteilung von X^2 im Falle der Zufälligkeit ergibt sich aus der Multinomialverteilung und ist approximativ χ^2-verteilt mit ν Freiheitsgraden (zum Beispiel $\nu = 10 - 1 = 9$). Für $\alpha = 0.05$ folgt die Entscheidungsregel: Falls $X^2 > \chi^2_{\nu;1-\alpha}$, verwerfe die Zufälligkeitsannahme.

Dieser Test kann direkt zur Beurteilung von Ziffernhäufigkeiten oder von Trefferhäufigkeiten für ein- und mehrdimensionale Intervalle verwendet werden, er dient daneben als zentraler Baustein verschiedenster Testprozeduren, die ganz spezielle Perspektiven auf die Zahlenströme umsetzen. Als alternativer Baustein kann aber auch der K-S-Test eingesetzt werden.

Da Tests auf Zufälligkeit an verschiedenen Stellen kompakt beschrieben werden, sollen hier nur einige zentrale Ideen aufgelistet werden. Vergleiche: [17], [15].

Folgende Liste orientiert sich an [17].

Name	Idee / Ansatz der Untersuchung
Frequency Test	Häufigkeiten von Elementen
Serial Test	Häufigkeiten von k-dim Intervallen
Gap Test	Wiederauftretenswartezeiten
Poker Test	Häufigkeiten von Mustern aufeinanderfolgender Elemente
Coupon Collector Test	Wartezeiten bis Vollständigkeit von Serien
Permutation Test	Häufigkeiten von Permutationen nach Abbildung von aufeinanderfolgenden Tupeln mit ganzen Zahlen
Run Test	Runlängenverteilungen
Maximum-of-t Test	Verteilung der Maxima von Teilmengen
Birthday Spacings Test	Verteilung der Abstände von sortierten Elemente
Serial Correlation Test	Korrelationen mit verschobenen Zahlenstrom
Tests on Subsequences	Teilmengen

Ein einzelner Test ist immer nur in der Lage, eine ganz bestimmte Perspektive umzusetzen. Erst durch Einsatz mehrerer kann man auf Nummer sicher gehen. Hierdurch wird letztendlich die ursprüngliche Idee des Testens in der Statistik völlig verändert. Man beschreibt eher bestimmte Eigenschaften der Zahlenströme, aufgrund derer der Anwender dann sein Urteil fällen muß.

Theoretische Tests. Die angesprochenen Tests haben einen großen Nachteil: sie können erst nach Realisation eingesetzt werden. Deshalb wäre es viel besser, auf theoretischem Wege Generatoren zu untersuchen. In die Klasse solcher theoretischen Tests gehört der sogenannte Spektraltest, bei dem für lineare Kongruenzgeneratoren festgestellt wird, wie groß der maximale Abstand benachbarter Ebenen ist. Ein anschauliches Beispiel befindet sich in [15].

Diehard-Test-Battery. Und was wird nun praktisch gemacht? Hierzu gibt es Vorschläge, bestimmte Mix von Tests einzusetzen. Ein prominenter ist die Diehard-Test-Battery, die auch in der R-Hilfe angesprochen wird. Das Programm ist erhältlich unter [11]:

```
http://stat.fsu.edu/~geo/diehard.html
```

Ein paar Kommentare zu der Test-Batterie erhalten wir aus dem Internet.

Diehard battery of tests

General

Prof. Georges Marsaglia from the Florida State University has devised a set of powerfull statistical tests for testing randomness of sequences of numbers, called the Diehard battery of randomness tests.

The Diehard program written by B. Narasimhan can be found on the Web, but unfortunately seems not to have been maintained for the last few years.

A renewed version of the Diehard program, including the graphical interface, is included in the PowerTest.

The Diehard test is important because it seems to be the most powerfull general test of randomness. Many software and hardware generators that we have tested, and which claim "perfect randomness" actually fail one or more sections of Diehard. There are whole classes of frequently used software pseudo-random generators which are known to fail Diehard test, such as Linear Congruental and Lagged Fibonacci generators.

In testing of hardware random number generators some other tests may be even more sensitive (efficient) to the hardware-specific problems. In hardware generators the main problems are the unavoidable bias (the probability of 1's is not the same as the probability of 0's) and bits correlation (a bit depends (statistically) on previous bits). Testing longer and longer sequences of random bits, the Diehard test will eventually detect these defects too, but specialized frequency and autocorrelation tests will do it faster thus saving the time. That is the reason why the PowerTest contains many other interesting tests besides the Diehard.

In conclusion, a generator which pass Diehard test should be considered good.

Description

The Diehard battery of tests consisits of 18 different, independent statistical tests. Results of tests are so called "p-values". In the PowerTest version of Diehard, these values are of Kolmogorov-Smirnov type, which means their values are real, between 0 and 1. For any given test, smaller p-value means better test result. An individual test is considered to be failed if p value approaches 1 closely, for example p>0.9999. See the PowerTest description for further details.

What follows is the original description of the tests as found in the original Diehard program by B. Narasimhan. ...

`http://random.com.hr/products/random/Diehard.html`, [25]

Eine Liste der Tests dieser Test-Batterie soll dem Leser hier nicht vorenthalten werden.

```
BIRTHDAY SPACINGS TEST
OVERLAPPING 5-PERMUTATION TEST
BINARY RANK TEST
BINARY RANK TEST
BINARY RANK TEST
BITSTREAM TEST
OPSO TEST
OQSO TEST
DNA TEST
```

```
The COUNT-THE-1's TEST on a stream of bytes.
The COUNT-THE-1's TEST for specific bytes.
PARKING LOT TEST
MINIMUM DISTANCE TEST
3DSPHERES TEST
SQEEZE TEST
OVERLAPPING SUMS TEST
RUNS TEST
CRAPS TEST
```

Leider zeigt die an derselben Stelle angebotenen Seite:

```
http://random.com.hr/products/powertest/PowerTest.html
```

– über die eine verbesserte Umsetzung angeboten werden soll – den Kommentar:

```
THIS PRODUCT IS NOT YET AVAILABLE
```

π. Zum Abschluß dieses Abschnitts sei das Rätsel, woher der oben aufgelistete Zahlenstrom stammt, gelichtet. Betrachte dazu:

```
3.14159265358979323846264338327950
```

In diesem Zusammenhang muß ein Papier von Veith Tiemann erwähnt werden, in dem er zum Beispiel untersucht, ob sich die Ziffern von π als Zufallszahlen verwenden lassen. Dazu berichtet er über verschiedene Episoden und entwirft eine Reihe von Experimente, von denen er einige in der Sprache S-Plus umsetzt: [30].

7.7 Beliebige Zufallszahlen

7.7.1 Die Inversionsmethode

Für die Erzeugung von Pseudozufallszahlen, die als Realisationen aus nicht-$U(0,1)$-verteilten Grundgesamtheiten durchgehen können, hilft der Satz:

Satz

Ist U gleichverteilt über dem Intervall $(0,1)$, X kontinuierliche Zufallsvariable mit der Verteilungsfunktion F_X und F_X^{-1} die Inverse von F_X, dann gilt:

$$Y = F_X^{-1}(U) \sim X.$$

Beweis:

$$\begin{aligned} F_Y(y) &= P(Y \leq y) \\ &= P(F_X^{-1}(U) \leq y) \\ &= P(F_X(F_X^{-1}(U)) \leq F_X(y)) \\ &= P(U \leq F_X(y)) \\ &= F_U(F_X(y)) \\ &= F_X(y) \end{aligned}$$

Y besitzt also dieselbe Verteilungsfunktion wie X. Damit haben die Zufallsvariablen X und Y auch dieselbe Verteilung.

Aus diesem Satz läßt sich für kontinuierliche Zufallsvariablen das folgende Konstruktionsprinzip ableiten.

Algorithmus:

1. ermittle Invervse F_X^{-1} von F_X
2. ziehe Zahl u_0 aus $U(0,1)$
3. transformiere u_0 mit Hilfe von F_X^{-1}
4. liefere Transformationsergebnis ab.

Die Exponentialverteilung bietet sich als einfaches Beispiel zur Demonstration an:

Schritt 1: invertiere F_X:

$$\begin{aligned} u &= F_X(x) && \Rightarrow \\ u &= 1 - e^{-\lambda x} && \Rightarrow \\ 1 - u &= e^{-\lambda x} && \Rightarrow \\ \log(1-u) &= -\lambda x && \Rightarrow \\ -\log(1-u)/\lambda &= x = F_X^{-1}(u) \end{aligned}$$

Schritt 2: erzeuge u_0
Schritt 3: transformiere: $-\log(1-u_0)/\lambda =: x_0$
Schritt 4: liefere x_0 ab.

Da mit U auch $1-U$ (0,1)-gleichverteilt ist, erhalten wir exponentialverteilte Zufallszahlen mit Parameter λ auch einfach durch die Transformation $-\log(u_0)/\lambda$.

In R können wir exponentialverteilte Zufallszahlen direkt oder aber mit Hilfe der beschriebenen Transformation generieren. Beide Methoden sollten zu Ergebnissen mit vergleichbaren Eigenschaften führen:

71 $\langle * 1 \rangle + \equiv$

```
n<-10000; seed<-13
set.seed(seed); y<- rexp(n)
```

```
set.seed(seed); x<- -log(runif(n))
qqplot(x,y,log="xy")
```

Man sieht, daß die Punkte gut auf einer Geraden liegen. Wenn intern nach diesem Verfahren gearbeitet würde, hätten die Punkte exakt auf einer Geraden liegen müssen. Da dieses nicht der Fall ist, ist in R ein anderer Algorithmus umgesetzt. Damit Simulationsexperimente überprüft werden können, müssen sie wiederholt werden können. Deshalb ist es empfehlenswert, den Start des Zufallszahlenstromes zu merken bzw. zu setzen. Hierzu dient in dem Beispiel die Funktion `set.seed`.

Zwei kleine Fragen. Wie bekommt man nach dieser Methode Realisationen von (a, b)-gleichverteilte Zufallsvariablen?

Nach dieser Transformationstechnik ist übrigens auch die oben vorgestellte Funktion zur Erzeugung Pareto-verteilter Zufallszahlen entworfen worden. Der Leser möge diese als ein zweites Beispiel ansehen und sich von der Korrektheit der Implementation überzeugen.

Diskrete Zufallsvariablen. Was tun, wenn diskrete Zufallsvariablen zur Diskussion stehen und die Verteilungsfunktion im spannenden Bereich nicht streng monoton und damit nicht invertierbar ist? Der Mangel läßt sich einfach heilen:

Algorithmus:

1. zeichne F_X
2. ermittle Realisation u_0
3. zeichne Gerade $y = u_0$ in den Graphen
4. liefere die x-Koordinate des Schnittpunktes der Geraden mit der Treppenfunktion als Zufallszahl ab.

Falls bei dem Algorithmus genau ein Funktionwert getroffen wird – was auf einem Rechner nur mit winziger Wahrscheinlichkeit vorkommt –, liefere den kleinsten x-Wert mit dem vorgegebenen Funktionswert u_0 ab.

7.7.2 Die Verwerfungsmethode

Zur Vorstellung der zweiten Methode sei die Existenz einer Dichte f_X unterstellt. Es soll eine Zufallszahl aus dem Intervall $(x_{\min}, x_{\max})$ generiert werden. Dann läßt sich folgende allgemein verwendbare Methode anwenden.

Algorithmus:

1. zeichne Dichte f_X
2. ziehe Zahl x aus $U(x_{\min}, x_{\max})$
3. ziehe Zahl y aus $U(0, f_{\max})$
4. falls $y < f(x)$, liefere x ab und stoppe
5. gehe zum zweiten Punkt zurück.

Für diskrete Zufallsvariablen ist das Verfahren entsprechend zu modifizieren. Im Falle der Normalverteilung könnte zur Anwendung der Inversionsmethode die Invertierung der Verteilungsfunktion ein technisches Problem darstellen. Sollen beispielsweise standardnormalverteilte Zufallszahlen aus $(0, 4)$ erzeugt werden, dann können wir die Verwerfungsmethode erfolgreich einsetzen.

Schritt 1: ziehe x aus $U(0, 4)$
Schritt 2: ziehe y aus $U(0, h)$ mit $h =$`dnorm(0)`
Schritt 3: falls $y \leq$ `dnorm(x)`, liefere x ab und stoppe
Schritt 4: gehe zum zweiten Punkt zurück.

Hier ist eine Implementation mit Überprüfung:

72 ⟨ * 1⟩+ ≡

```
n<-1000; seed<-13
set.seed(seed); z <- rnorm(3*n); z <- z[0<z & z<4][1:n]
set.seed(seed)
x <- runif(5*n, 0, 4); y <- runif(5*n, 0, dnorm(0))
x <- x[ y <= dnorm(x) ][1:n]
qqplot(x,z)
```

Wie man sieht, bringt auch dieser Ansatz zufriedenstellende Ergebnisse hervor. Jedoch werden mit hoher Wahrscheinlichkeit viele Zufallszahlen verschwendet und damit unnötig viele Berechnungen durchgeführt. Durch Modifikation des Verfahrens lassen sich diese Nachteile verringern. Natürlich gibt es zur Erzeugung von Zufallszahlen, die aus Normalverteilungen stammen, viel elegantere Methoden.

7.7.3 Spezielle Methoden

Da nach dem Zentralen Grenzwertsatz die Verteilungen von standardisierten Mitteln bei existierender Varianz mit wachsendem Stichprobenumfang gegen die Standardnormalverteilung gehen, lassen sich zur Imitation normalverteilter Zufallszahlen auch geeignet transformierte Mittel verwenden. Insofern ist es nicht verwunderlich, daß man Vorschläge findet wie:

Algorithmus:

1. ziehe 12 Zahlen aus U(0,1)
2. liefere die Summe dieser vermindert um 6 als Normal-Zufallszahl ab

Was spricht für, was gegen diesen Vorschlag?

Eher ist zum Beispiel die Box-Muller-Methode zu empfehlen, die an vielen Stellen in der Literatur beschrieben wird, siehe zum Beispiel [15], Seite 494. Bei dieser Methode werden auf Grundlage von zwei Zufallszahlen (u_1, u_2) aus einer (0,1)-Gleichverteilung zwei neue (x_1, x_2) erzeugt nach der Vorschrift:

$$x_1 = \mu + \sigma \cos(2\pi u_1)\sqrt{-2\ln(u_2)}$$

$$x_2 = \mu + \sigma \sin(2\pi u_1)\sqrt{-2\ln(u_2)}$$

So berechnete Zahlen besitzen dann Eigenschaften, wie Stichproben aus normalverteilten Grundgesamtheiten, sofern ...

Auswirkungen ungeschickter Parameter. Eine ungeschickte Wahl der Parameter zur Erzeugung der verwendeten gleichverteilten Zufallszahlen kann zu äußerst schönen aber auch unzweckmäßigen Ergebnissen führen. Zur Demonstration definieren wir zunächst `random.box.muller`, eine Funktion zur Generierung von Zufallszahlen nach der Box-Muller-Methode. Mit dem Parameter `change` läßt sich die Rolle von u_1 und u_2 vertauschen.

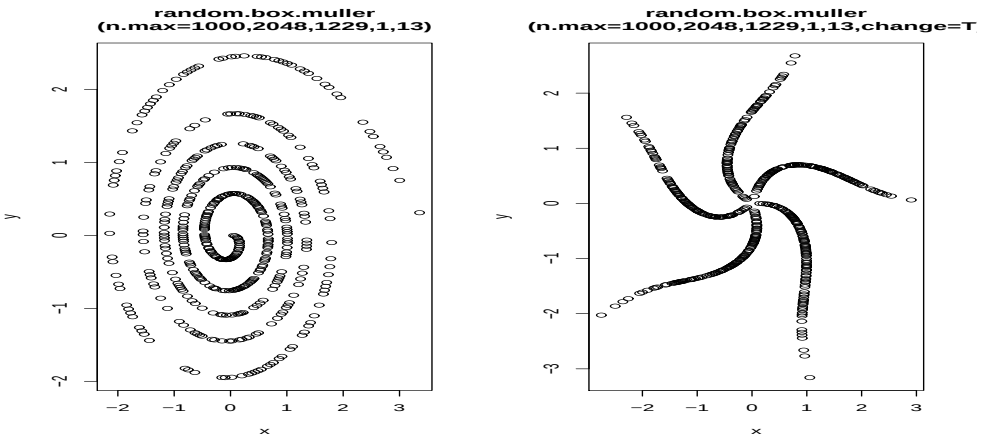
73 ⟨*definiere* `random.box.muller` 73⟩ ≡ ⊂ 79

```
random.box.muller<-function(n.max,m,a,r,x.0,change=F){
  n<-2*ceiling(n.max/2)
  u<-matrix(random.gkg(n,m,a,r,x.0)/m,nrow=2); if(change) u<-u[2:1,]
  u2<-u[1,]; u1<-u[2,]
  x<-sqrt(-2*log(u1))*cos(2*pi*u2)
  y<-sqrt(-2*log(u1))*sin(2*pi*u2)
  return(cbind(x,y))
}
```

Folgende Bilder sind erzeugt worden nach [26], Seite 57.

74 ⟨* 1⟩+ ≡

```
par(mfrow=1:2)
plot(random.box.muller(n.max=1000,2048,1229,1,13))
title("random.box.muller\n(n.max=1000,2048,1229,1,13)")
plot(random.box.muller(n.max=1000,2048,1229,1,13,change=T))
title("random.box.muller\n(n.max=1000,2048,1229,1,13,change=T)")
par(mfrow=c(1,1))
```



Fazit. Man kann also spezielle Eigenschaften einer Verteilung, besonders natürlich auch die Verwandtschaft zu anderen Verteilungen, ausnutzen, um elegant die Zufallszahlen der gewünschten Art hervorzubringen. Solche Beziehungen findet man in Fülle zum Beispiel in [12]. Jain beschreibt zur groben Orientierung ein Vorgehensmodell für die Wahl eines Generierungsverfahrens:

```
if(is CDF invertible)                          then  use inversion
if(is CDF a sum of other CDFs)                 then  use composition
if(is pdf a sum of other pdfs)                 then  use composition
if(is the variate a sum of other variates)     then  use convolution
if(is the variate related to other variates) then  use characterization
if(does a majorizing function exists)          then  use rejection
                                             else  use empirical inversion
```

nach [15], Seite 481

7.8 Ein Praxis-Bericht

Können wir die vielen Auseinandersetzungen in der heutigen Zeit als überflüssig ansehen? Betrachten wir dazu folgende Mail

Hello everyone.

If I do

```
f <- function(n){max(rnorm(n))}
plot(sapply(rep(5000,4000),f))
#[this takes my PC about 30 seconds]
```

then I get something quite unexpected: gaps in the distribution. For me, the most noticable one is at about 3.6.

Do others get this? Is it an optical illusion? It can't be right, can it? Or maybe I just don't understand the good ol' Gaussian very well.

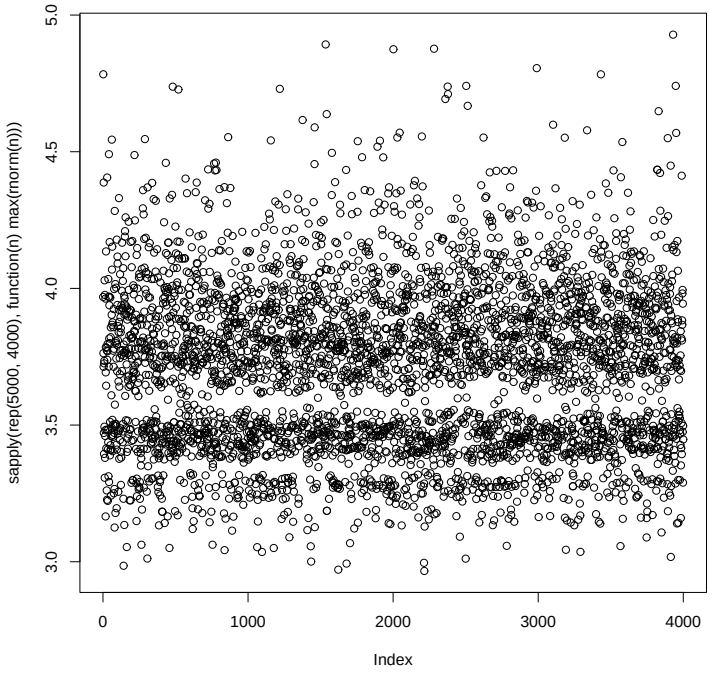
anyone got an explanation?

Robin Hankin, R-help-Mail, 26.11.02

Ob die Frage überhaupt berechtigt ist, wollen wir sofort nachprüfen:

75 ⟨ * 1⟩+ ≡

```
plot(sapply(rep(5000,4000), function(n)max(rnorm(n))))
```



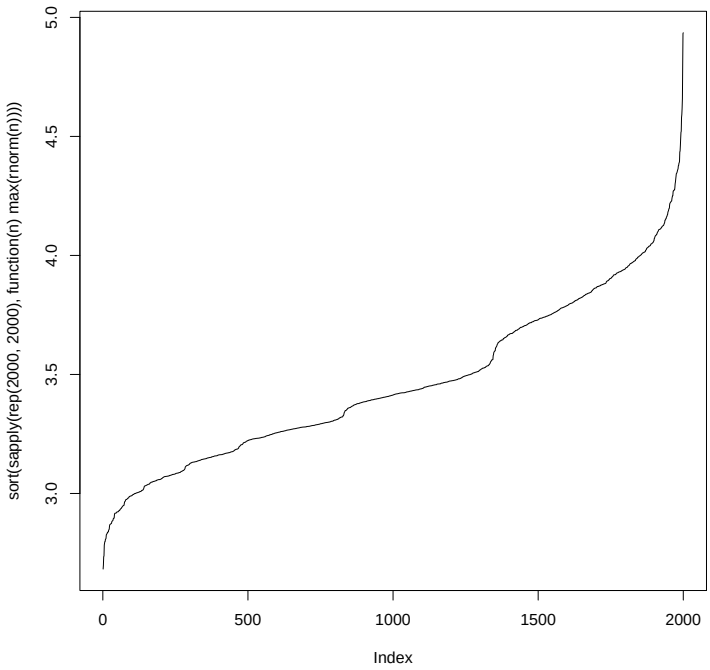
In der Tat, die Berechnung dauert recht lange. Abhilfe bringt ein Vorschlag von Dalgaard:

> *This is also illuminative:*
>
> ```
> plot(sort(sapply(rep(2000,2000),f)),type="l")
> ```
>
> *We've seen odd behaviour of max(rnorm()) with the default random generators before. Switching to RNGkind("Wichmann-Hill") made the effect disappear here. Same thing switching the normal generator with RNGkind("Marsaglia-Multicarry","Box-Muller").*
>
> Peter Dalgaard, R-help-Mail, 26.11.02

76 ⟨ * 1⟩+ ≡

```
plot(sort(sapply(rep(2000,2000),function(n)max(rnorm(n)))),type="l")
```

Hier das Ergebnis:



Auch die Antwort von Ripley soll dem Leser übermittelt werden:

> *That's the maximum of 5000 normals, right? That's pushing the accuracy of some internal calculations too hard.*
>
> *If you want to do this, you should use*
>
> *RNGkind(, "Inversion")*
>
> *That's not the default for back-compatibility reasons.*
>
> Brain D. Ripley, R-help-Mail, 26.11.02

7.9 Anwendungshinweis: Übertragungsstörung

Betrachten wir ein sehr einfaches Netz der Form

und unterstellen, daß es auf jeder der beiden Verbindung zu einem Verlust der übertragenen Information kommen kann. Die Verlustwahrscheinlichkeiten seien q_1 und q_2 und unabhängig voneinander. Dann ergibt sich die Wahrscheinlichkeit für den Verlust insgesamt durch

$$P(\text{Verlust}) = q_1 + (1 - q_1)q_2$$

und für erfolgreichen Transport durch

$$P(\text{Transport ok}) = 1 - (q_1 + (1 - q_1)q_2) = (1 - q_1)(1 - q_2) =: p_1 p_2$$

Diese Situation können wir für `n=1000` Übermittlungen unter Verwendung des standardmäßig angebotenen R-Generators entweder simulieren durch

77 ⟨ * 1⟩+ ≡

```
set.seed(13)
q1<-0.01; q2<-0.02; n<-1000
stoer1<-runif(n)<q1
stoer2<-runif(n)<q2
stoerung <- stoer1 | stoer2
table(stoerung)
```

... oder aber durch Verwendung obiger Umformungen

78 ⟨ * 1⟩+ ≡

```
set.seed(13)
q1<-0.01; q2<-0.02; n<-1000
q.system <- q1 + (1-q1)*q2
stoerung<-runif(n) <  q.system
table(stoerung)
```

Damit haben wir zwei Ansätze zur Simulation motiviert: Einerseits können wir auf der elementaren Ebene die Elemente eines Systems simulieren und dann die Ergebnisse (hier durch eine `|`-Operation) zusammenführen. Wir können aber auch auf Grund der Analyse versuchen, die Gesamtcharakteristika des Systems (hier: `q.system`) abzubilden. Für den zweiten Weg ist es hilfreich, ein paar Gedanken darüber zu verschwenden, wie man aufgrund der Systemstruktur auf das Systemverhalten schließen kann.

Das vorgestellte Beispiel entspricht einer Reihenschaltung aus der Elektronik. Bei solchen *und*-Architekturen ergibt sich die Wahrscheinlichkeit für erfolgreichen Transport durch Multiplikation der einzelnen Erfolgswahrscheinlichkeiten:

$$P_{\text{Reihensystem}}(\text{Transport ok}) = \prod_k P_{\text{Komponente } k}(\text{Transport ok})$$

Die Zuverlässigkeit des Internets basiert wesentlich darauf, daß der Transport auch dann gelingt, wenn einzelne Verbindungen nicht funktionieren. Die Möglichkeit, alternative Wege zur Auswahl zu haben, läßt sich mit einer Parallelschaltung aus der E-Technik identifizieren. Für eine solche *oder*-Architektur gilt:

$$P_{\text{Parallelsystem}}(\text{Transport ok}) = 1 - \prod_k \Big(1 - P_{\text{Komponente } k}(\text{Transport ok})\Big)$$

Mit diesen beiden Grundregeln lassen sich auf einfache Weise auch sehr komplizierte Systeme beschreiben. Notwendige Voraussetzung ist natürlich die

unterstellte Unabhängigkeit, die oftmals nicht gegeben sein dürfte. Als Einstieg soll an dieser Stelle verwiesen werden auf Naeve, Kapitel 9.2, [19].

7.10 Anwendungshinweis: Lebensdauern von Systemen

In der Zuverlässigkeitstheorie wird zunächst ohne Einbeziehung stochastischer Elemente aufgrund der beiden elementaren Architekturen Reihen- (oder auch Serien-) und Parallelschaltung die Struktur eines Systems beschrieben. Erst im zweiten Schritt werden die Erkenntnisse um Wahrscheinlichkeiten erweitert. In konkreten Anwendungen sind dann realitätsnahe Konkretisierungen zu finden. Werden zum Beispiel Lebensdauern von Komponenten – in unserem Problemfeld entweder mit Übertragungsleitungen oder Knotenrechnern zu identifizieren – betrachtet, kann jeder Komponente eine Lebensdauerverteilung zugeordnet werden.

Für Reihenschaltungen ergibt sich als Gesamtlebensdauer das Minimum der einzelnen Lebensdauern. Für die Verteilung der Systemlebensdauer T folgt mit den Lebensdauern der Komponenten T_k:

$$\begin{aligned} F_{\text{Reihensystem}}(t) &= P_{\text{Reihensystem}}(T \leq t) \\ &= 1 - P_{\text{Reihensystem}}(T > t) \\ &= 1 - P(T_1 > t \wedge \cdots \wedge T_n > t) \\ &= 1 - \textstyle\prod_k P(T_k > t) \\ &= 1 - \textstyle\prod_k (1 - F_{T_k}(t)) \end{aligned}$$

oder in Termen der sogenannten Zuverlässigkeitsfunktion $\overline{F}_{\text{Reihensystem}}(t) = 1 - F_{\text{Reihensystem}}(t)$:

$$\overline{F}_{\text{Reihensystem}}(t) = \prod_k \overline{F}_{T_k}(t)$$

Entsprechend gilt für ein Parallelsystem

$$\begin{aligned} F_{\text{Parallelsystem}}(t) &= P(T \leq t) \\ &= P(T_1 \leq t \wedge \cdots \wedge T_n \leq t) \\ &= \textstyle\prod_k P(T_k \leq t) \\ &= \textstyle\prod_k F_{T_k}(t) \end{aligned}$$

Elegant lassen sich diese Berechnungen durchführen, wenn als Verteilungsfunktion eine Exponentialverteilung angenommen werden kann. Für eine Reihenschaltung aus n verschiedenen Komponenten ergibt sich zum Beispiel:

$$F_{\text{Reihensystem}}(t) = 1 - \prod_k (1 - F_{T_k}(t)) = 1 - e^{-t\sum_k \lambda_k t}$$

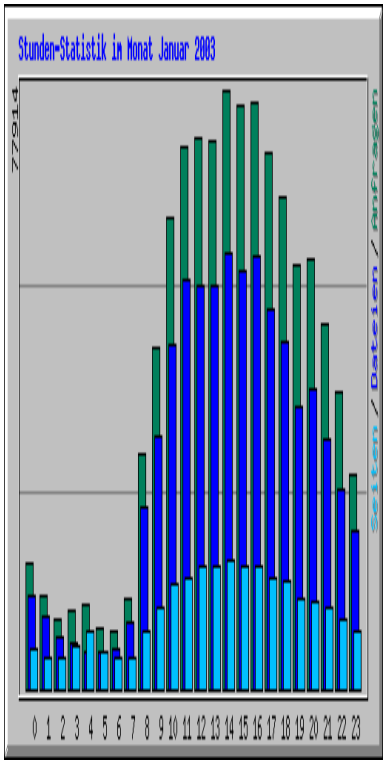
Eine Parallelschaltung n identischer Komponenten führt uns zu der Formel:

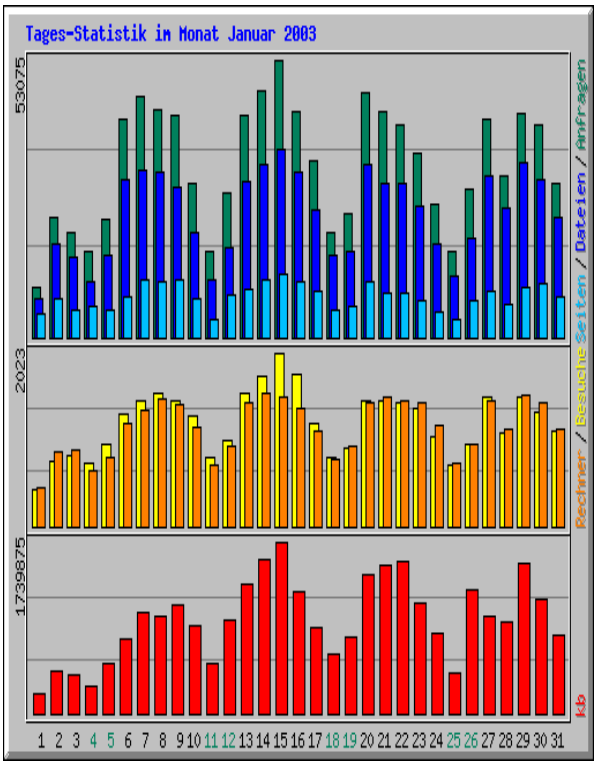
$$F_{\text{Parallelsystem}}(t) = \prod_k F_{T_k}(t) = (1 - e^{-\lambda t})^n$$

Mit Hilfe solcher Beziehungen lassen sich die Bedeutung von Ausfällen, Lebensdauern sowie Übertragungszeiten für größere Systeme studieren. Die Ergebnisse könnten dann in geeignete Simulationsmodelle einfließen, um letztendlich Planungen zu verbessern.

7.11 Schluß

Und wie sieht die Realität aus? Dem Administrator werden heutzutage eine Reihe von Tools angeboten, die den Internet-Traffic recht bunt darstellen:





Aber werden die Darstellungen vom Betrachter auch mit tiefsinnigen Gedanken angereichtert? Diese Frage können wir hier nicht beantworten. Ebenso geht es mit vielen anderen Fragen zum diskutieren Gegenstand. Denn leider ist mit diesen Bemerkungen das Ende der Vorlesung erreicht wie auch das Ende der für diese Niederschrift vorgegebene Zeit. Es besteht die Hoffnung, daß es auf Basis der zusammengetragenen Erfahrungen in naher Zukunft zu einer Wiederholung der Veranstaltung kommt und dann ein viel runderen Ergebnis resultiert. Aber ich denke, daß der Leser auch in dieser spröden Form viele Anregungen bekommen hat, um über den Gegenstand nachzudenken. Außerdem dürfte er eine Reihe von hilfreichen, statistischen wie auch informatikspezifischen Elementen kennengelernt hat, die hoffentlich an ganz anderen Stellen helfen werden.

Februar 2003 Peter Wolf

8 Anhang

8.1 Initialisierung einiger Funktionen

79 ⟨*start* 79⟩ ≡
⟨*definiere* `ad.exp` 24⟩
⟨*definiere* `arrival.poisson.gof` 26⟩
⟨*definiere* `arrival.poisson.ind` 28⟩
⟨*definiere* `identify.exponential` 20⟩
⟨*definiere* `identify.weibull` 34⟩

⟨*definiere* `weibull.est` 31⟩
⟨*definiere* `identify.pareto` 39⟩
⟨*definiere Funktionen zur Pareto-Verteilung* 37⟩
⟨*definiere* `poxplot` 43⟩
⟨*definiere* `var.time.plot` 42⟩
⟨*definiere* `sim.selfsim` 48⟩
⟨*definiere* `selfsim.test` 44⟩
⟨*definiere* `random.gkg` 55⟩
⟨*definiere* `random.tg` 64⟩
⟨*definiere* `random.tg.shift` 66⟩
⟨*definiere* `random.box.muller` 73⟩
⟨*definiere* `spin3R` 81⟩
```
print("Initialisierung von stabi beendet")
```

8.2 Serverdaten

80 ⟨*start* 79⟩+ ≡
```
"zeitpunkte.03.02.97" <-
c(173877, 178739, 203188, 203742, 203906, 205573, 205640, 209310,
209949, 210299, 210382, 210906, 213142, 213545, 213785, 213791,
213922, 213924, 213925, 213969, 214320, 214935, 214949, 214961,
214969, 215075, 215387, 216570, 217267, 218288, 219059, 219105,
219258, 219287, 219959, 220475, 220591, 220876, 220949, 220964,
221084, 221098, 221588, 222376, 222976, 223093, 225865, 226062,
226068, 226079, 226081, 226082, 226101, 226122, 226625, 227999,
228132, 228469, 228789, 230543, 230662, 230706, 231139, 231857,
232015, 232061, 232773, 232831, 232970, 233010, 233175, 233519,
234897, 235485, 235940, 236810, 239219, 240240, 240787, 240797,
240798, 240806, 240846, 240913, 241079, 242224, 243002, 243262,
244777, 245317, 246598, 248563, 248770, 250683, 253013, 253187,
253760, 255462, 255511, 255618, 255908, 256270, 257651)
"dzeitpunkte.03.02.97" <-
c(1077, 4862, 24449, 554, 164, 1667, 67, 3670, 639, 350, 83,
524, 2236, 403, 240, 6, 131, 2, 1, 44, 351, 615, 14, 12, 8, 106,
312, 1183, 697, 1021, 771, 46, 153, 29, 672, 516, 116, 285, 73,
15, 120, 14, 490, 788, 600, 117, 2772, 197, 6, 11, 2, 1, 19,
21, 503, 1374, 133, 337, 320, 1754, 119, 44, 433, 718, 158, 46,
712, 58, 139, 40, 165, 344, 1378, 588, 455, 870, 2409, 1021,
547, 10, 1, 8, 40, 67, 166, 1145, 778, 260, 1515, 540, 1281,
1965, 207, 1913, 2330, 174, 573, 1702, 49, 107, 290, 362, 1381
)
"mengen.03.02.97" <-
c(14101, 105563, 1866, 11505, 216, 19391, 35340, 10746, 205246,
```

```
8192, 124667, 25467, 23819, 49226, 2450, 5520, 1646, 2471, 12301,
3431, 1646, 15560, 9287, 5105, 7070, 32116, 1363, 3190, 1548,
12851, 1395, 1001, 16418, 3431, 20106, 35024, 5113, 10395, 1589837,
57965, 200133, 98082, 57314, 5113, 1007, 16418, 45354, 18606,
3705, 711, 16418, 1275, 708, 18868, 28354, 92723, 14101, 19533,
25741, 20445, 12671, 150325, 4107, 20867, 1434, 305118, 23719,
13756, 14101, 345, 25300, 6515, 3406, 109591, 14640, 197070,
216, 1434, 15747, 1434, 12301, 12451, 15251, 8130, 8687, 19391,
800, 2587, 23719, 345, 836, 4107, 1594950, 15871, 10746, 140075,
1650, 129, 15688, 561, 122754, 216, 14640)
"zeitpunkte.17.02.97" <-
c(1382822, 1384215, 1384346, 1384788, 1387047, 1388561, 1388822,
1394377, 1395438, 1403951, 1406572, 1407042, 1412097, 1414932,
1415151, 1415161, 1416757, 1416849, 1418675, 1419331, 1419391,
1421413, 1422235, 1422257, 1422303, 1422368, 1422608, 1422906,
1423487, 1423547, 1424076, 1424318, 1425637, 1425819, 1427253,
1427281, 1427302, 1427307, 1427326, 1427682, 1427765, 1428085,
1428687, 1429426, 1430373, 1430374, 1430375, 1430443, 1430530,
1430958, 1430977, 1430992, 1430995, 1431731, 1432460, 1432784,
1434197, 1434355, 1434378, 1434676, 1435392, 1435413, 1436708,
1439140, 1440101, 1440637, 1441413, 1441456, 1442088, 1442378,
1445136, 1447472, 1448909, 1449247, 1449544, 1452266, 1454031,
1454319, 1454503, 1456332, 1459823, 1459830, 1460204, 1461119,
1461847, 1461906, 1466870)
"dzeitpunkte.17.02.97" <-
c(422, 1393, 131, 442, 2259, 1514, 261, 5555, 1061, 8513, 2621,
470, 5055, 2835, 219, 10, 1596, 92, 1826, 656, 60, 2022, 822,
22, 46, 65, 240, 298, 581, 60, 529, 242, 1319, 182, 1434, 28,
21, 5, 19, 356, 83, 320, 602, 739, 947, 1, 1, 68, 87, 428, 19,
15, 3, 736, 729, 324, 1413, 158, 23, 298, 716, 21, 1295, 2432,
961, 536, 776, 43, 632, 290, 2758, 2336, 1437, 338, 297, 2722,
1765, 288, 184, 1829, 3491, 7, 374, 915, 728, 59, 4964)
"mengen.17.02.97" <-
c(20760, 2034, 65218, 7526, 6944, 3481, 14115, 1448, 9985, 14115,
1448, 14115, 30273, 1448, 1448, 13012, 18688, 7572, 1551, 18688,
34890, 9723, 2285, 7283, 1448, 1969, 6739, 129898, 38348, 14115,
14115, 52438, 38517, 1448, 29969, 10749, 3287, 324, 29140, 8346,
14115, 32836, 14115, 16418, 4024, 445, 10533, 6021, 41378, 1448,
2910, 4051, 29140, 19473, 8346, 345, 27785, 8564, 4425, 478920,
30533, 16418, 29083, 14115, 868, 27852, 19473, 12322, 4273, 207117,
1448, 12515, 1001, 5519, 87888, 16721, 33964, 22307, 9078, 78540,
14115, 4262, 11289, 18688, 1448, 870, 36422)
print("Datensaetze abgelegt")
print(ls())
```

8.3 Eine Funktion zur Rotation von Punkten

Die in diesen Abschnitt definierte Funktion ermöglicht dem Anwender eine Punktwolke interaktiv zu drehen und zu betrachten.

81 ⟨*definiere* spin3R 81⟩ ≡ ⊂ 79

```
spin3R <- function(x, alpha=1, delay=.015){
 ##################################################################
 # spinR3: simple spin function to rotate a 3-dim cloud of points#
 # pw 160103                                                      #
 #                                                                #
 # arguments:                                                     #
 #                                                                #
 #  x             (nx3)-matrix of points                          #
 #  alpha         arc of rotation                                 #
 #  delay         sleeping time between rotations                 #
 #                                                                #
 ##################################################################
 library(tcltk)
 ⟨generiere Steuerungsfenster 82⟩
 ⟨definiere Rotationen 84⟩
 ⟨definiere Bindungen 83⟩
 ⟨initialisiere Plot 85⟩
 ⟨starte Endlosschleife 86⟩
 ⟨entferne Steuerungsfenster 87⟩
}
```

82 ⟨*generiere Steuerungsfenster* 82⟩ ≡ ⊂ 81

```
Rot <-tclVar("relax");bw <- 4
topl<-tktoplevel();   tkwm.geometry(topl,"+100+100")
f1 <- tkframe(topl);f2 <- tkframe(topl);f3 <- tkframe(topl)
f4 <- tkframe(topl);f5 <- tkframe(topl);tkpack(f1,f2,f3,f4,f5)

b12 <- tkbutton(f1, relief="ridge",  width=bw, text="up")
b21 <- tkbutton(f2, relief="ridge",  width=bw, text="left")
b22 <- tklabel(f2,  relief="flat",   width=bw)
b23 <- tkbutton(f2, relief="ridge",  width=bw, text="right")
b32 <- tkbutton(f3, relief="ridge",  width=bw, text="down")
b41 <- tkbutton(f4, relief="ridge",  width=bw, text="clock")
b42 <- tklabel(f4,  relief="flat",   width=bw)
b43 <- tkbutton(f4, relief="ridge",  width=bw, text="cclock")
b51 <- tkbutton(f5, relief="raised", width=bw, text="reset")
b52 <- tklabel(f5,  relief="flat",   width=bw)
b53 <- tkbutton(f5, relief="raised", width=bw, text="exit")
tkpack(b12,b32)
tkpack(b21,b22,b41,b42,b51,b52,side="left")
```

```
tkpack(b23,b43,b53,side="right")
```

83 ⟨*definiere Bindungen* 83⟩ ≡ ⊂ 81

```
for(type in c("12","21","23","32","41","43")){
 b<-eval(parse(text=paste("b",type,sep="")))
 tkbind(b, "<Enter>",
     eval(parse(text=paste("function()tclvalue(Rot)<-\"",type,"\"",sep=""))))
 tkbind(b, "<Leave>",function() tclvalue(Rot) <- "relax")
}
tkconfigure(b51,command=function() tclvalue(Rot) <- "reset" )
tkconfigure(b53,command=function() tclvalue(Rot) <- "exit" )
```

Für die Rotation bezüglich zwei Achsen wird nur eine 2×2-Rotationsmatrix benötigt.

84 ⟨*definiere Rotationen* 84⟩ ≡ ⊂ 81

```
alpha<-alpha/360*2*pi; ca<-cos(alpha); sa<-sin(alpha)
rot<-matrix(c(ca,-sa,sa,ca),2,2)
```

`x` hält die Daten, `x.o` die Originaldaten, `xa` die 2-dim Projektionen. Für die Anschaulichkeit wird ein Andeutung der Achsen mitgeliefert: `A` beschreibt die Achsen, `A.o` die Originalachsen, `Aa` den darzustellenden Teil.

85 ⟨*initialisiere Plot* 85⟩ ≡ ⊂ 81

```
n <- nrow(x); x <- x - matrix(apply(x,2,min),n,3,T)
x.o<-x<-x / matrix(apply(x,2,max),n,3,T) - 0.5;                xa <- x[,2:3]
A.o<-A<-0.5*matrix(c(1,0,0, 0,0,0, 0,1,0, 0,0,0, 0,0,1),5,3,T);Aa <- A[,2:3]
plot(xa, xlim=.7*c(-1,1), ylim=.7*c(-1,1),
          pch=20, xlab="",ylab="",xaxt="n",yaxt="n")
lines(Aa)
```

86 ⟨*starte Endlosschleife* 86⟩ ≡ ⊂ 81

```
i <- 0             #  ; i.max<-100
cat("Ende durch Klick auf Exit\n")
if(delay < 0.015) delay <- 0.015
repeat{
  Sys.sleep(delay)
  choice <- tclvalue(Rot)
  if(choice=="exit"
                   #  || ((i<-i+1)>i.max)
                    ){ break }
  if(choice=="relax") next
  if(choice=="reset") {
    points(xa, pch=20, col="white"); lines(Aa, col="white")
```

```
    x <- x.o; A <- A.o; xa<-x[,2:3]; Aa<-A[,2:3]
    points(xa, pch=20, col="black"); lines(Aa, col="black")
    tclvalue(Rot)<-"relax"; next
  }
  switch(choice,
   "12" = ind<-c(1,3), "21" = ind<-c(2,1), "23" = ind<-c(1,2),
   "32" = ind<-c(3,1), "41" = ind<-c(3,2), "43" = ind<-c(2,3)
  )
  x[,ind] <- x[,ind]%*%rot; A[,ind] <- A[,ind]%*%rot
  points(xa, pch=20, col="white"); lines(Aa, col="white")
  xa<-x[,2:3]; Aa<-A[,2:3]
  points(xa, pch=20, col="black"); lines(Aa, col="black")
}
```

87 ⟨*entferne Steuerungsfenster* 87⟩ ≡ ⊂ 81

```
tkdestroy(topl)
"Rotationsfunktion beendet"
```

References

[1] Becker, Richard A., Chambers, John M., Wilks, Allen R. (1988): The new S-Language, Wadsworth & Brooks/Cole.

[2] Bogomolny, Alexander (?): Internetseite mit der Überschrift: Benford's Law and Zipf's Law, http://www.cut-the-knot.com/do_you_know/zipfLaw.shtml

[3] Breslau, Lee, Cao, Pei, Fan, Li, Phillips, Graham, Shenker, Scott (?): On the Implications of Zipf's Law for Web Caching, http://www.cs.wisc.edu/~cao/papers/zipf-implication

[4] Breslau, Lee, Cao, Pei, Fan, Li, Phillips, Graham, Shenker, Scott (1999): Web Caching and Zipf-like Distributions: Evidence and Implications, Proceedings of IEEE Infocom, March 1999

[5] Cesari, Fabio (?): Internetseite über Fraktale, Chaos, Apfelmännchen, http://www.geocities.com/CapeCanaveral/2854/

[6] Chen-Nee Chuah, Randy H. Katz (∼1999): Characterizing Packet Audio Streams from Internet Multimedia Applications, http://www.cs.berkeley.edu/~chuah/research/paper/chuah_icc02.pdf

[7] Crovella, Mark E., Azer Bestavros, Azer (1997): Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes, IEEE/ACM Transactions on Networking, Vol. 5, 6, 835–846

[8] Ralph B. D'Agostino, Ralph B., Stephens, Michael A. (1986): Goodness-of-Fit Techniques, Marcel Dekker, New York

[9] Danzig, Peter B., Jamin, Sugih, Càceres, Ramòn, Mitzel, Danny J., Estrin, Deborah (1992): An Empirical Workload Model for Driving Wide Area TCP/IP Network Simulations, Internetworking: Research and Experience, Vol. 3, 1–26, http://www.kiskeya.net/ramon/work/pubs/jinet92.pdf

[10] Danzig, Peter B., Jamin, Sugih (1991): tcplib: A Library of TCP Internetwork Traffic Characteristics, Computer Science Department, University of Southern California, http://nms.lcs.mit.edu/~jyjung/paper/danzig91tcplib.pdf

[11] random.com.hr: Internetseite mit Diehard-Test-Battery-Programm, http://stat.fsu.edu/~geo/diehard.html

[12] Evans, Merran, Hastings, Nicholas, Peacock, Brian (1993): Statistical Distributions, Wiley, New York

[13] Fisz, Marek (1989): Wahrscheinlichkeitsrechnung und mathematische Statistik, Dt. Verl. d. Wiss., Berlin

[14] Grieser, Jürgen (1998): Symbolische Dynamik, Punktprozesse und Markov-Ketten, → Poisson-Prozeß in kontinuierlicher Zeit, http://www.rz.uni-frankfurt.de/~grieser/...symbkopf/node9.html,

[15] Jain, Raj (1991): The Art of Computer Systems Performance Analysis, Wiley, New York

[16] Jin, Shudong, Bestavros, Azer (2001): GISMO — A Generator of Internet Streaming Media Objects and Workloads, http://www.cs.bu.edu/techreports/ps/2001-020-gismo.ps

[17] Knuth, Donald E (1998): The Art of Computer Programming II, 3rd Edition, Addison-Wesley Reading, Massachusetts

[18] Leland, Will E., Taqqu, Murrad, Willinger, Walter, Wilson, Daniel V. (1994): On the Self-Similar Nature of Ethernet Traffic, IEEE/ACM Trans. on Networking, Vol. 2, 1, 1–15, February, http://ejo.univ-lyon1.fr/doc/english/ethernet/Leland93.ps

[19] Naeve, Peter (1995): Stochastik für Informatik, Oldenbourg, München

[20] Niermann, Stefan, Jöhnk, M.D. (2002): Parameterschätzung für klassierte Daten. Statistical Papers, Vol. 43, 2002

[21] Park, Kihong, Kim, Gitae, Crovella Mark (?): On the Relationship between File Sizes, Transport Protocols, and Self-Similar Network Traffic

[22] Paxson, Vern, Floyd, Sally (1995): Wide-Area Traffic: The Failure of Poisson Modeling, IEEE/ACM Transactions on Networking, 3(3), 226–244, Proceedings of SIGCOMM'94

[23] Peitgen, Heinz-Otto, Jürgens, Hartmmut, Saupe, Dietmar (1992): Chaos and Fractals – New Frontiers of Science, Springer

[24] Software Corporation: Administrator's Guide – Proxy-Server, `http://vms.process.com/~help/helpproxy.html`

[25] `random.com.hr`: Internetseite zur Diehard-Test-Battery, `http://random.com.hr/products/random/Diehard.html`

[26] Ripley, B.D. (1987): Stochastic Simulation, Wiley, New York

[27] Tausworthe, R.C. (1965): Random Numbers Generated by Linear Recurrence Modulo Two. Mathematics of Computations, 19, 201–209

[28] Thisted, Ronald A. (1988): Elements of Statistical Computing, Chapman & Hall, p. 165

[29] Tiemann, Veith (1999): Simòn Denis P. und die Rosinen im Teig, `http://www.wiwi.uni-bielefeld.de/StatCompSci/...`
`...lehre/material_spezifisch/praktikum99.00/...`
`...rosinen/rosinen.html`

[30] Tiemann, Veith (1999/2000): Paradoxa in der Statistik, `http://www.wiwi.uni-bielefeld.de/StatCompSci/...`
`...lehre/material_spezifisch/praktikum99.00/...`
`...sonstiges/paradox.ps`

[31] Venables, W.N., Ripley, B.D. (1994): Modern Applied Statistics with S-Plus, Springer, New York

[32] Wang, Wenguang (2000): CMPT 815 Assignment 1: Trace Characterization and Modelling, `www.cs.usask.ca/grads/wew036/815/ass1.ps`

[33] Willinger, Walter, Taqqu, Murrad S., Sherman, Robert, Wilson, Daniel v. (1997): Self-Similarity through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level, `http://math.bu.edu/people/murad/pub/source-printed-version-posted.ps`

[34] Der Chaos-Bilder-Generator WinCIG, `http://www.th-soft.com/d_wincig.htm`

[35] Zipf, George Kingsley (1972): Human Behavior and the Principle of Least Effort: an Introduction to Human Ecology. Hafner, New York, Repr. d. Ausg. Cambridge, Mass. 1949

Code Chunk Index

Object Index

zz $\in$ 48
zz.vec $\in$ 48