Simulation und Prognose

FT 2010

Dr. Detlef Steuer
Tel. 2819, steuer@hsuhh.de

20. April 2010

Orgakram

- Vorlesungszeiten: Überschneidungen mit Ökonometrie II.
- Raum: PC-Labor Maschinenbau H1 R. 1392.
- Prüfungsorganisation: Bisher ist der Plan, die Klausur am 30.6.2010 durchzuführen.
- Sprechstunde: Jederzeit, wenn der voraussichtliche Zeitbedarf unter 10 Minuten liegt, sonst bitte telefonisch oder per mail Termin absprechen!

Ablauf der Veranstaltung

- Vorlesung mit integrierter *hands-on* Übung in R (2+2 - 3+1 nach Bedarf).
- Skript wird vorlesungsbegleitend bereit gestellt.
- Programmieren von Simulationen in R.
- Voraussetzungen: Begrifflichkeiten aus Statistik I + II.
- Eigeninitiative um etwas R zu lernen. Es ist mehr Programmierung erforderlich als in Datenanalyse I + II, insbesondere ein Verständnis von Schleifen bzw. wiederholter Anwendung von Programmteilen auf Vektoren oder Listen. Im Übungsteil der Veranstaltung werden die nötigen Teile von R allerdings alle geübt.
- Bitte Rückmeldung geben, wenn zu schnell, zu langsam etc.!

Begriffsklärung

- Was ist Gegenstand der Vorlesung?
- Begriffsklärung Simulation.
- Begriffsklärung Prognose.
- Die Programmiersprache `R`: Das Werkzeug.

Was ist Simulation?

- Simulation ist ein *Vorgehen* zur Analyse von Systemem, die einer theoretischen oder analytischen Analyse nicht zugänglich sind.
- Häufig gilt dies für dynamische oder rückgekoppelte System.
- Simulation ist hier Computersimulation!
- Es gibt auch eine Simulation durch Modelle o.ä.
- Simulation ist hier stochastische Simulation, keine Simulation durch deterministische Modellierung (z.B. FEM, Differentialgleichungen)

- Simulation im Rahmen dieser Vorlesung steht immer für eine stochastische Simulation, bei der man versucht die analytischen Schwierigkeiten bei der Behandlung eines Systems durch vielfache, "verrauschte" Simulation eines möglicherweise vereinfachten Modells zu umgehen.

- Ein einfaches Beispiel: Warteschlangenproblem. Wie viel Wechselgeld benötigt man zur Ladenöffnung in n Kassen bei m Schlangen, damit es in weniger als 0.1 % der Fälle zu Problemen kommt?

- Klar ist der Zusammenhang zur Modellbildung. Eine Simulation kann nur so gut sein, wie das zugrundeliegende Modell!

- Wichtiges Problem: Woher kommt eigentlich der Zufall? Der Computer arbeitet doch hoffentlich deterministisch?

Was bedeutet Prognose?

- Simulationen werden in der Regel durchgeführt, um Eigenschaften von Systemen vorherzusagen, die entweder nicht analytisch zugänglich sind oder die nicht beobachtbar sind, weil sie beispielsweise in der Zukunft liegen. Bekanntestes Beispiel: Meteorologie.

- Aber auch Validierung statistischer Modelle durch Vorhersage der Ergebnisse von Experimenten bei unbekannten Parametereinstellungen.

- Prognose beinhaltet hier immer auch die Prognose eines Unsicherheitsniveaus (Schätzung mit $\hat{\sigma}$)!

- Bekannte Begriffe: Prognoseintervall, Konfidenzintervall.

- Die Prognose ist also das natürliche Ziel einer Simulation!

Die Programmiersprache R: Das Werkzeug

- R ist eine (Interpreter-)Sprache und eine Arbeitsumgebung für statistische Grafik und Analyse.

- R liefert in der Standardinstallation bereits eine große Zahl von statistischen und grafischen Verfahren der Datenanalyse und ist darüber hinaus entworfen, um leicht erweiterbar zu sein. Es gibt über 2000 Erweiterungspakete für alle Aspekte der Datananalyse.

- Eine große Stärke von R liegt in der leichten Anfertigung von veröffentlichungsfähigen Plots, inklusive mathematischer Annotationen.

- Das R-Core Team nennt R eine Umgebung für statistische Berechnugen und Grafik.

Die Programmiersprache R

- In dieser Vorlesung: Nicht nur Beschränkung auf die bereits implementierten Teile.

- `R` ist eine Art statistischer (Hochleistungs-)Taschenrechner und in dieser Veranstaltung ein geduldiger Zufallsgenerator.

- Der typische Ablauf wird die Konstruktion einer einzelnen Simulation in `R` sein, deren Eigenschaften dann durch vielfache Wiederholung untersucht werden.

Warum R?

- R ist *Freie Software* (kostenlos und open source).
- R ist plattformunabhängig, d.h. Sie nutzen weiter den Rechner und das Betriebssystem, das sie gewohnt sind, sei es Windows, MacOs oder Unix.
- Hervorragende Fähigkeiten: Immer mehr Firmen nutzen R, also bekommen Sie ein Werkzeug an die Hand, das Sie fast sicher im beruflichen Umfeld wieder sehen werden. R entwickelt sich im universitären Bereich zur Standardsoftware, ebenso, wenn auch verzögert im industriellen Bereich.
- Am 7.1.2009 sogar eine Titelseitengeschichte der NYT!

Warum R?

- Hervorragende eingebaute Hilfefunktion!
- Lokalisiert in etlichen Sprachen.
- Professioneller (oder besser) Support über Mailinglisten!
- Professionelle (oder besser) Qualitätskontrolle der Software ('make check'). Validierung der Software und der Rechenergebnisse während der ganzen Entwicklung.
- Sehr gute Handbücher werden mitinstalliert (Reference Manual $>$ 1800 Seiten).
- Für Bachelor-, Master- oder Doktorarbeiten: sehr gute Integration mit LaTeXund OpenOffice. (MS Office ist auch ok.)

Benutzerinterfaces für R

- Eingebaut ist nur ein CLI. R ist ein Interpreter mit *read-eval-loop*!
- Empfehlenswert: Interface zu einem externen Editor (emacs (!), winedt, etc.).
- Es gibt GUIs: gehören nicht zur Standardinstallation und werden in der Vorlesung nicht behandelt. Windows hat ein rudimentäres Mausinterface, aktuell scheint Tinn-R das einfachste zu sein.
- Batch mode (skriptgesteuert).
- etwas ausgefallener: R als Modul des Webservers.
- oder: R als shared library aus anderen Programmiersprachen aufrufen (python, perl).

Eine erste R-Sitzung

```
steuer@gaia:~> R
R version 2.10.1 Patched (2010-01-08 r50953)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()',
um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder
'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.
>
```

Zuerst:

```
> Sys.setenv(http_proxy="http://backspace.unibw-hamburg.de:3128")

### Setzt den Uni-Proxy. Nicht nötig außerhalb des Uni-Netzes!

> contributors()

### Liste der Entwickler

> citation()

### Zitierung von R als Literaturstelle.
### R hat eine ISBN!
```

Erste Schritte: interaktive Nutzung von R

- R als Taschenrechner

```
> 3 + 4
[1] 7

> log(0)
[1] -Inf

> log(-1)
[1] NaN
Warning message:
NaNs were generated in: log(x)
```

Erste Schritte: interaktive Nutzung von R

```
> pi  #es kommt auf Groß- oder Kleinschreibung an
[1] 3.141593
> x <- .Last.value
# x = .Last.value geht "neuerdings" auch
> ls()
[1] x
> rm(x)
> seq(1,10)^2
# R ist eine vektororientierte Sprache
> q()
```

Eingebautes Hilfesystem

- http://cran.r-project.org/doc/contrib/Short-refcard.pdf
  Das cheat-sheet für `R`!

- `help` oder "?": äquivalent zu RTFM: versuchen Sie `help(plot)` oder `?plot`.

- Wenn man das genaue Kommando nicht weiß oder `help` nicht hilft, dann kann man `apropos()`, `find()` oder `help.search()` versuchen.

- Versteht man die Hilfeseite nicht, dann kann man mit `example(`*`command`*`)` oder `demo(`*`command`*`)` versuchen, den Befehl und seine Nutzung am Beispiel zu lernen.

Eingebautes Hilfesystem

- `help.start()` zeigt die Dokumentation im Standard-Webbrowser an.
- Die meisten von Nutzern hinzugefügten Pakete enthalten eine sog. Vignette, eine kurzes Handbuch im PDF Format. Mit dem Kommando `vignette()` kann man sich dieses anzeigen lassen.

Externe Pakete

- In erheblichem Umfang zusätzliche Funktionalität in externen *packages* (oder *views*)
  `available.packages()` gibt eine Liste der aktuell vorhandenen Pakete.

- Einfaches Einfügen in eine bestehende R Installation:

```
install.packages("scatterplot3d")
```

- Laden in eine laufende R-Sitzung mit `library(scatterplot3d)` or `require(scatterplot3d)`

- Entfernen aus einer laufenden Sitzung `detach(package:scatterplot3d)`

Mathematische Operatoren

Symbol	Funktion
^ oder **	Potenz
*, /, +, -	Multiplikation, Division, Addition, Subtraktion
%/%, %%	ganzzahlige bzw. modulo Division
%*%	Matrixmultiplikation

Natürlich gibt es alle üblichen mathematischen Operationen: `round()`, `sin()`, `abs()`, `sqrt()` etc.

Wichtig für das Konzeptverständnis: Alle diese Operatoren sind gewöhnliche R-Funktionen:

```
> "+"(3,4)
[1] 7
```

Mathematische Operatoren

- Wichtig sind die Bezeichner für die speziellen Zahlen:
  - NaN : Not a Number,
  - Inf, -Inf : plus resp. minus unendlich,
  - NULL : nichts, leer,
  - TRUE, FALSE : Wahr oder falsch,
  - NA : not available, fehlender Wert, *missing value*.

- Achtung: `R` implementiert IEEE Arithmetik! Internationaler Standard.

```
> round(1.5) ; round(0.5)
[1] 2
[1] 0
```

- Achtung: `pi` ist nicht `PI`! `R` beachtet Groß- und Kleinschreibung!

Logische Operatoren

- == : beide Objekt sind **identisch**,
- all.equal() testet auf numerische Gleichheit bis auf eine festgelegte Abweichung,
- != : ungleich,
- <, > , <=, >= kleiner als, größer als (oder gleich),
- &, |, ! : (logisch) AND, OR, NOT .

Kleine Fallstricke

```
> a <- 3
> b <- 2.1/0.7
> a == b
[1] FALSE
```

Was passiert hier?

Kleine Fallstricke

```
> a <- 3
> b <- 2.1/0.7
> a == b
[1] FALSE
```

Was passiert hier?

Lösung in R: es gibt die Funktion `all.equal()`

```
> all.equal(a, b)
[1] TRUE
> ?all.equal
```

`all.equal()` überprüft die numerische Gleichheit bis auf ein ϵ
Standard: `sqrt( .Machine.double.eps )`

Kleine Fallstricke

Naiver Weise vermutet man, dass das Folgende funktioniert:

```
> a <- NA
> a == NA
```

oder

```
> a <- NaN
> a == NaN
```

Kleine Fallstricke

Naiver Weise vermutet man, dass das Folgende funktioniert:

```
> a <- NA
> a == NA

[1] NA
> a <- NaN
> a == NaN
[1] NA
```

Macht es aber nicht!
Für diese Fälle stellt R Folgendes zur Verfügung:

```
> a <- NA ; is.na(a)
> a <- NaN ; is.nan(a)
```

Erzeugung von Vektoren

- Da Simulationen oft das Ausprobieren einer Reihe von Parametern oder das Zwischenspeichern von vielen Ergebnissen erfordert, ist der Umgang mit Listen und Vektoren essentiell.

- R operiert standardmäßig auf Vektoren!

```
> (x <- c(1, 3, 4.5))
[1] 1.0 3.0 4.5
> (x <- c(1, x, 3))
[1] 1.0 1.0 3.0 4.5 3.0
> length(x) ### length of vector
[1] 3
> t(x)
```

Indizierung von Vektoren (Teilmengen)

```
> (x <- c(2, 3, 5, 7, 11, 13, 17, 19, 23))
[1]  2  3  5  7 11 13 17 19 23
> x[1]           ### Zugriff über Index
[1] 2
> x[2:4]         ### mehrere Elemente
[1] 3 5 7
> x[-(2:4)]      ### Ausschluss einiger Elemente
[1]  2 11 13 17 19 23
> x[-c(1, 7, 9)] ### wahlfreier Zugriff
[1]  3  5  7 11 13 19
> x[]            ### ist x
[1]  2  3  5  7 11 13 17 19 23
```

Indizierung über Eigenschaften der Einträge

```
> which(x < 10) ### Indizes, die eine Bedingung erfüllen
[1] 1 2 3 4
> x
[1]  2  3  5  7 11 13 17 19 23

> x > 10
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE

> sum(x>10) # Anzahl der Einträge, die eine Bedingung erfüllen
5

> x [ x > 10 ] # Zugriff über boole'schen Vektor
[1] 11 13 17 19 23
```

Erzeugung spezieller Vektoren `seq()`, `rep()`

```
> 1:5; 5:1     ### Abstand 1
[1] 1 2 3 4 5
[1] 5 4 3 2 1
> seq(1,5,2/3) ### Abstand != 1
[1] 1.000000 1.666667 2.333333 3.000000 3.666667 4.333333 5.000000
> seq(along=x) ### Nummerierung von x
[1] 1 2 3 4 5 6 7 8 9
> rep(TRUE, 5)
[1] TRUE TRUE TRUE TRUE TRUE
> rep(c("red","blue"),c(4,7))
 [1] "red"  "red"  "red"  "red"  "blue" "blue" "blue" "blue"
 [9] "blue" "blue" "blue"
```

Rechnen mit R-Vektoren

```
> (x <- seq(3,7))
[1] 3 4 5 6 7
> 1+x ### Recycling der 1
[1] 4 5 6 7 8
> 2*x ### elementweises Rechnen
[1]  6  8 10 12 14
> x*x ### auch Multiplkation
[1]  9 16 25 36 49
> x%*%x  ### Skalarproduct, implizite Transposition!
     [,1]
[1,]  135
> 6:1 - 1:3 ### kürzerer Vektor wird recycled
[1]  5  3  1  2  0 -2
```

Funktionen von Vektoren

- In R wirken Funktionen in der Regel auf ganzen Vektoren:

```
> sin(seq(0,2+pi,pi/16))
[1]  0.000000e+00  8.660254e-01  8.660254e-01  1.224606e-16
     -8.660254e-01
```

- Selbstdefinerte Funktionen in R sind sehr einfach:

```
> ownodd <- function(x) {if (x%%2) cat("ungerade\n")
                                 else cat("gerade\n")}
> ownodd(1) ; ownodd(2) # mehrere Befehle in einer Zeile
ungerade
gerade
```

- Oft sind selbstgeschriebenen Funktionen einfacher zunächst für skalare Argumente zu formulieren.

```
> ownodd(1:2) # if funktioniert nicht auf Vektoren
Warnmeldung:
In if (x%%2) cat("ungerade\n") else cat("gerade\n") :
Bedingung hat Länge > 1 und nur das erste Element wird benutzt
```

- Die Funktion `Vectorize()` hilft dann:

```
> Vectorize(ownodd)(1:2)
ungerade
gerade
```

- Oft sehr nützlich bei Simulationen!

Matrizen

- Eine Matrix kann mit `matrix()` erzeugt werden.
  Bsp.: `matrix(1:9,nrow=3, byrow=TRUE)`

- Wichtige Funktionen: `%*%`, `crossprod()`, `solve()`, `diag()`, `eigen()`, `qr()`, `svd()`, `t()`

- Vorsicht: Bloß nicht aus Versehen Dimensionen verlieren!

```
> x <- matrix(1:9,nrow=3)
> x[1,1]
[1] 1
> x[1,1, drop=FALSE]
     [,1]
[1,]    1
```

Indizierung von Matrizen

- Zwei Indizes beschreiben die Position eines Elements in einer Matrix. x[i,j] ist in der i. Zeile und j. Spalte. Ansonsten pro Dimension Zugriff wie bei Vektoren!
  Ex.: x[1,] , x[2:4, 5:7]

Listen

- `R` hat Wurzeln in LISP, weshalb die Liste ein grundlegender Datentyp ist. Im Gegensatz zu Vektoren können Listen Einträge verschiedener Datentypen enthalten.

- Erzeugt werden Listen mit dem Befehl `list()`, wobei den einzelnen Elementen Namen zugewiesen werden können.

- Der Zugriff auf Listenelement mit Index, z.B. `LL`$[[i]]$, oder dem Namen des Elements, z.B. `LL$ab`.

```
> LL <- list(ab="hi", numbers=c(2, 4, 5),
                                  xx=matrix(1:4,2))
```

```
> LL
$ab
[1] "hi"
$zahlen
[1] 2 4 5
$xx
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> LL[[2]]
zahlen
[1] 2 4 5
> LL$ab
[1] "hi"
```

Elementare Statistik (Statistik I)

Vielzahl eingebauter Funktionen!

- `mean()`, `var()`, `sd()`, `cor()` etc.
- `runif()`, `rnorm()` etc. Zufallszahlenerzeugung
- `fivenum()`, `range()`, `summary()`, `stem()` Tukey's numbers, Spannweite, Stem-and-leaf plot
- `boxplot()`, `pie()`, `hist()` grundlegende grafische Darstellungen
- `lm()`, `t.test()` lineare Regression, t-Test

Datenhaltung

- `ls()` zeigt in einer R-Sitzung alle aktuell exisitierenden Objekte an. Für etwas mehr Auskünfte über die Objekte kann man `ls.str` versuchen. Alle Objekte der aktuellen Sitzung werden in der Datei `.Rdata` im aktuellen Arbeitsverzeichnis gespeichert, wenn man mit 'y' auf `'q()'` antwortet.

- Das aktuelle Arbeitsverzeichis liest und schreibt man mit `getwd()` und `setwd()`. Unter Windows kann das ARbeitsverzeichnis über das Menü gesetzt werden.

- Pro: Automatische Datensicherung!

- Kontra: Es handelt sich um ein Binärformat! Man sollte dies nicht als Hauptsicherung wichtiger Daten nutzen!

- Siehe auch `save()` und `save.image()`.

- Es bietet sich eine Arbeitsweise an, die pro Projekt ein Verzeichnis vorsieht.

- Den Verlauf der letzten eingegebenen Befehle findet man in der Datei `.Rhistory`.

Verteilungsbezogene Funktionen in R

- Für viele Verteilungen gibt es Funktionen wie `pnorm, qnorm, dnorm, rnorm`, die das Ablesen aus Tabellen ersetzen bzw. der Verteilung entsprechende Zufallszahlen erzeugen. Die Verteilungsparameter können jeweils als Funktionsparameter angegeben werden. Hier am Beispiel für die Normalverteilung:

- `pnorm(q, mean=0, sd=1, ...)` Verteilungsfunktion (*probability*-),
  `dnorm(x, mean=0, sd=1, ...)` Dichtefunktion (*density*-),
  `qnorm(p, mean=0, sd=1, ...)` Quantilsfunktion (*quantile*-) und
  `rnorm(n, mean=0, sd=1)` Zufallszahlenerzeugung (*randomnumber*-).

- Alle üblichen Verteilungen liegen in R bereits derart vor.

- `sample(x, size, replace = FALSE, prob = NULL)` Ziehen mit und ohne Zurücklegen

Ablaufkontrolle in einem R-Programm

- Um im Ablauf eines Programms Bedingungsabhängig zu verzweigen gibt es die Befehle `if, ifelse` und `switch`. Im Einzelnen:

- `if ()` Syntax: `if` (*Bedingung*) { *Befehlsblock*} [`else` { *Befehlsblock* }]
  - geschweifte Klammern nur nötig, wenn der Befehlsblock mehr als einen Befehl enthält
  - `else` ist optional
  - `if` operiert nicht auf Vektoren!

- `ifelse()` Syntax: `ifelse(`*Bedingung*`, `*TRUE - Ergebnisse*`, `*FALSE - Ergebnisse*`)`
  - operiert auf Vektoren!

- `switch(EXPR, list of commands , ...)`
  - sehr starke Verallgemeinerung für Programmablauf - für Spezialfälle.

Beispiele `if/ifelse`

```
> x <- c( 7, 9)
> if (x > 8) {cat (x)} # Nur das erste Element wird genutzt
> x <- 5
> if ( x == 5) {x <- x + 1 ; y<-3} else y <- 7
> x ; y
> ifelse (x ==  c(5, 6), c("A1", "A2"), c("A3", "A4"))
```

Schleifen in R

- `while`
  Syntax: `while`(*Bedingung*) { *Befehlsblock* }
  - Läuft weiter und startet nur, wenn *Bedingung* erfüllt.

- `repeat`
  Syntax: `repeat`{ *Befehlsblock* }
  - wiederholen, bis die Schleife explizit abgebrochen wird.

- `for`
  Syntax: `for (i in M)`{ *Befehlsblock* }
  - Wiederholt den Befehlsblock für alle Werte i in M.

- `next` springt unmittelbar an den Anfang der Schleife.

- `break` beendet Schleife sofort.

Übungsaufgabe zur Programmierung

Übung 1: Berechnen Sie die ersten 20 Fibonacci-Zahlen.
Erinnerung: Für die Fibonacci Zahlen $F_i, i \in \mathbb{N}$ gilt:

$$F_i = F_{i-1} + F_{i-2} \text{ mit } F_1 = 1, F_2 = 1.$$

Nutzen Sie eine Schleifenkonstruktion Ihrer Wahl.

Tipp: Für Teilnehmer ohne Programmiererfahrung ist es vermutlich sinnvoll zunächst einen Vektor festzulegen, der die ersten 20 Zahlen enthalten soll. Dann kann man von Hand die ersten Werte berechnen und schliesslich versuchen den Ansatz in einer Schleife zu automatisieren.