Analyse von Eyetracking-Daten mit R: Aufgaben

1. Psycholinguistischer Methodenworkshop

an der Humboldt-Universität zu Berlin

Jochen Laubrock
Universität Potsdam

08. April 2010

Die Aufgaben setzen voraus, dass unser R-Paket `eyetrackR` installiert ist. Das Paket ist noch in einer frühen Entwicklungsphase und deshalb noch nicht auf CRAN verfügbar. Sie müssen es stattdessen aus einem lokalen Archiv installieren. Das geht am leichtesten wie folgt:

1. eyetrackR.tar.gz herunterladen
2. In R mit der Paketverwaltung aus lokalem Source Package installieren
3. Fertig

Vorverarbeitung

Mit dem Skript `preprocess` lassen sich Eyelink® Eyetracker-Daten vom proprietären .edf-Binärformat in standardisiertes ASCII-Textformat wandeln. Das Skript ruft das Programm `edf2asc` der Herstellers SR Research auf, welches auf den Rechnern im Pool nicht installiert ist.[1]

Sie finden jedoch den durch Anwendung von `preprocess` auf die Datendatei 02.edf produzierten Output in den Dateien 02.msg und 02.dat.

[1] `edf2asc` ist nach vorheriger Registrierung von der Webseite des Herstellers zu beziehen.

Normalerweise verarbeitet `preprocess(edfdir = '.', edf2ascpath = '~/bin/')` alle edf-Dateien im durch `edfdir` spezifizierten Verzeichnis und verschiebt sie dann in das Unterverzeichnis processed. Die Daten werden gefiltert in Rohdaten (.dat-Dateien im Unterverzeichnis dat) und informative Botschaften (.msg-Dateien im Unterverzeichnis msg). Die Rohdaten sind einfach Zeitreihen der Messungen, meist bestehend aus Zeitstempel, x- und y-Koordinate des Blickes und Pupillengröße; bei binokulärer Messung für jedes Auge eine Spalte. Die .msg-Dateien enthalten größtenteils mit einem Zeitstempel versehene Nachrichten, die der Programmierer der Experimentalsteuerung hat aufzeichnen lassen, aber auch Informationen über die Qualität der Kalibrierung, über online detektierte Fixationen etc.

Aufgaben

1. Betrachten Sie die Datei 02.msg (z.B. mit WordPad®)
2. Lesen Sie die Datei 02.msg in R ein und speichern Sie sie in der Variablen `msg'. Nutzen Sie dazu den Befehl `readLines`. Lassen Sie sich einige Zeilen von msg anzeigen.

Aufbereiten der Experimentalnachrichten (message file parsing) und Textverarbeitung

message file parsing

Die Aufgabe beim Aufbereiten der .msg-Dateien besteht darin, die relevanten Informationen zu extrahieren und in ein leichter zu verarbeitendes Format umzuformen, meistens eine Tabelle.

Im folgenden Beispiel sind einige Zeilen aus der Datei 02.msg aufgelistet:

```
# Dateikopf
MSG 1219952 DISPLAY_COORDS 0 0 1023 767
MSG 1219952 RETRACE_INTERVAL  8.32998624833

# pro Satz
MSG 1370977 TRIALID 0
START 1371033  RIGHT EVENTS
# entweder (Trialstart nicht erfogreich)
MSG 314789 -7 trialstart fixation trigger
MSG 318798 0 trialstart fixation not detected, timer
# oder  (Trialstart erfolgreich)
MSG 568813 -7 trialstart fixation trigger
MSG 569186 Start Recording
MSG 569193 -2 sentence displayed
MSG 569193 -2 !V DRAW_LIST ../../runtime/dataviewer/01/graphics/VC_1.vcl
```

```
MSG 569194 -1 !V IAREA FILE ../../runtime/dataviewer/01/aoi/IA_1.ias
MSG 569527 0 sound triggered
MSG 569539 sound presentation started
MSG 570994 0 trialend fixation trigger
MSG 570995 End Recording
END 571013  EVENTS RES   28.28   27.94
MSG 1373804 !V TRIAL_VAR trial_pre 1
[...]
# MSG $TIMESTAMP !V TRIAL_VAR ... (siehe unten)
[...]
MSG 571109 !V TRIAL_VAR rolle f
MSG 571110 TRIAL_RESULT 0
# Botschaften nach TRIAL_VAR
# trial_pre 1
# sentence_pre Sie: "Wie findest du mein Kleid?"
# boundary_pre 5
# sound_pre 1-F-Wie.wav
# rolle_pre f
# calibrationRequested False
# playsound 1
# sprecher m
# group 1
# doIncludePretest 1
# trial 1
# sentence Sie ruft "Hermann..." durch die offene Küchentür.
# boundary 9
# sound 1-M-Herm.wav
# rolle f
```

Die meisten interessanten Botschaften haben das Format `MSG Zeitstempel Botschaft`, wobei `Botschaft` wiederum aus mehreren Feldern bestehen kann.

Außerdem enthält die .msg-Datei noch einige Botschaften über Sakkaden- und Fixations-Events, generiert durch SR Research Online-Parser. Hier bedeuten SFIX bzw. EFIX Start bzw. Ende einer Fixation; SSACC/ESACC: Start/Ende einer Sakkade; in der Offline-Analyse interessieren normalerweise nur die mit `E' beginnenden Botschaften:

```
# Format:
# EFIX R t0 t1 dauer xpos   ypos   pupillenDurchmesser
# ESACC R t0 t1 dauer x0pos y0pos x1pos y1pos amplitude spitzenGeschwindigkeit
SFIX R   1374086
EFIX R   1374086 1374513 428   931.6   685.8    1224
SSACC R  1374514
ESACC R  1374514 1374615 102   931.5   687.0   213.6   409.9   26.06     436
```

Aufgaben

1. Extrahieren Sie aus der (in der vorigen Aufgabe eingelesenen) Variablen msg die Zeilen mit …

   a) `TRIALID`-Botschaften und speichern sie im Vektor `alltrialidMsg`.

   b) `sound triggered`-Botschaften und speichern sie im Vektor `allsoundtriggeredMsg`.

   c) `sound presentation started`-Botschaften und speichern sie im Vektor `allsoundstartMsg`.

   d) `sentence displayed`-Botschaften und speichern sie im Vektor `alltextonMsg`.

   e) `TRIAL_VAR playsound`-Botschaften und speichern sie im Vektor `allExpPlaysoundMsg`.

   Hinweis: Nutzen Sie den Befehl `grep` und beachten Sie dessen Parameter `value`. Falls Warnungen der Form ``input string is invalid in this locale'' auftreten, setzen Sie `useBytes=TRUE`.

2. Schreiben Sie eine Funktion `` `extractMsgTimeStamp' ``, die den Zeitstempel aus den MSG-Zeilen extrahiert. Wenden Sie die Funktion auf `allsoundstartMsg` an. Hinweis: Sie müssen den R-Befehl `function` nutzen; hilfreich sind außerdem die R-Befehle `[`, `sapply` und `strsplit`.

3. Extrahieren Sie das Feld hinter `` `playsound' `` aus den Botschaften in `allExpPlaysoundMsg` und speichern es im Vektor `expPlaysound`.

4. Studieren Sie die Funktion `parse_msg_file_AAA` (Im Unterverzeichnis demo von eyetrackR).

Ein Resultat des message file parsing ist eine Tabelle mit Informationen über die Trials, mit einer Zeile pro Trial bzw. Satz. Das Beispielskript `generate_tables_from_msg_files.R` zeigt, wie sich diese (.trialinfo) und andere Tabellen herausschreiben lassen. Das darauf aufbauende Beispiel `generate_list_of_wordinfo.R` illustriert, wie man die Zeilen aus den .trialinfo-Dateien `` `ausrollen' `` kann zu einer Zeile pro Wort. Dieses Format bietet sich an, um Corpusinformation oder (z.B. aus praat-Kodierungen erhaltene) Information über die Aussprechdauer von Wörtern hinzuzufügen.

Textverarbeitung: Eigenschaften des Satzcorpus

Oft ist in einer gesonderten Datei Satzcorpusinformationen über die einzelnen gelesenen Wörter gespeichert. Diese enthält üblicherweise für jedes gelesene Wort die Satznummer, laufende Nummer des Wortes im Satz und zusätzliche Informationen wie Wortlänge, Wortfrequenz in der Sprache (aus Datenbankabfragen, z.B. CELEX, dlexdb) etc. Die folgende Aufgabe illustriert die Berechnung der Wortlänge und der Anzahl der Wörter im Satz.

```
load(sent)
head(sent)

# [1] Sie: "Wie findest du mein Kleid?"               Er: "Welches?"
# [3] Sie: "Das ich anhabe."                          Er: "Besonders hübsch."
# [5] Sie: "Oder findest du das grüne schöner?"       Er: "Das grüne?"
# 86 Levels: Aber er antwortet: "Ich möchte jetzt nicht lesen.", und wird langsam unruhig. ...

sent <- as.character(sent)
```

Aufgaben

Gegeben die Variable `sent`, berechnen Sie

1. einen Vektor, der für jeden Satz die Länge in Buchstaben enthält.
2. einen Vektor, der für jeden Satz die Anzahl Wörter enthält.
3. eine Liste, die für jeden Satz einen Vektor mit den Wortlängen der einzelnen Wörter enthält,
   a) inklusive Punktuation.
   b) ohne Punktuation.

   Für welche Wörter unterscheiden sich (3a) und (3b) besonders deutlich?

Hinweis: die Funktionen `nchar`, `strsplit`, `sapply` und `unlist` sind hilfreich, beachten Sie auch die Hilfeseite zu regular expressions.

Konstruktion von Fixationssequenzen

Matching

Eine immer wiederkehrende Aufgabe betrifft das Zusammenfügen von Datensätzen. Diese sind oft unterschiedlich lang, haben aber korrespondieren-

de Elemente. Ein Beispiel ist das hinzufügen von Wortfrequenzen aus der Corpusdatei (nWords Zeilen) zu der alltrialinfo-Datei (nWords $\times$ nSubjects Zeilen). Ein weniger triviales Beispiel ist z.B. das Hinzufügen von praat-Kodierungen für ausgesprochene Wörter zu einer Liste von Wörtern, von denen nur einige ausgesprochen wurden. Für diese Art von Aufgabe bietet R die sehr komfortablen Operatoren `match` und `%in%`. Zusammenfügen ist eine fehleranfällige Operation, die Sicherheitschecks verdient.

Im folgenden Beispiel wird anhand simulierter Daten illustriert, wie der match-Befehl funktioniert:

```
id <- 3:5      # simulierte Vp
word <- 11:14  # simulierte Wörter

   # unsinnige Zufallszahlen als simulierte Wortfrequenzen
simFrq <- 2*10^(5.2*runif(length(word)))
   # "Corpus": Wort-ID und Wortfrequenz
wordTab <- data.frame(word, simFrq)

# Erstellen einer simulierten Trial-Tabelle, "Experiment 1"
idInTrialTab <- rep(id, each=length(word))  # Vp-ID
wordInTrialTab <- rep(word, length(id))     # Wort-ID
     # "Trial-Tabelle": Vp-ID und Wort-ID
trialTab <- data.frame(idInTrialTab, wordInTrialTab)
names(trialTab) <- c("id", "word")

# Illustration des Match-Befehls
ixWordTabIntoTrialTab <- match(trialTab$word, wordTab$word)
trialTab$frq <- wordTab[ixWordTabIntoTrialTab, "simFrq"]

str(trialTab)
# 'data.frame': 12 obs. of  3 variables:
#  $ id  : int  3 3 3 3 4 4 4 4 5 5 ...
#  $ word: int  11 12 13 14 11 12 13 14 11 12 ...
#  $ frq : num  290264.4 31.6 215.7 57859.6 290264.4 ...

# "Experiment 2": verschärfte Version: randomisierte Präsentation
wordInTrialTabR <- NULL
for (i in 1:length(id)) {
    # sample (Ziehen ohne Zurücklegen) simuliert
    # randomisierte Präsentationsreihenfolge
    wordInTrialTabR <- c(wordInTrialTabR, sample(word))
}
trialTab$wordR <- wordInTrialTabR

# match-Befehl hilft auch hier
ixWordTabIntoTrialTabR <- match(trialTab$wordR, wordTab$word)
trialTab$frqR <- wordTab[ixWordTabIntoTrialTabR, "simFrq"]

str(trialTab)
# 'data.frame': 12 obs. of  5 variables:
#  $ id   : int  3 3 3 3 4 4 4 4 5 5 ...
#  $ word : int  11 12 13 14 11 12 13 14 11 12 ...
#  $ frq  : num  290264.4 31.6 215.7 57859.6 290264.4 ...
#  $ wordR: int  14 12 11 13 12 11 13 14 14 11 ...
#  $ frqR : num  57859.6 31.6 290264.4 215.7 31.6 ...
```

Aufgaben

Hinzufügen von Korpusinformation zu Trial-Informationen

```
frqfile <- "dlexdb-beta-v0_01.csv"
wrdfile <- "wordinfo.rda"
frq <- read.table(frqfile, header=TRUE)
load(wrdfile)
# ...
```

1. Fügen Sie die Korpusfrequenzen aus der Datei `dlexdb-beta-v0_01.csv` (1 Zeile pro Wort) zu der Tabelle `wrdinfo` (1 Zeile pro Vp $\times$ Wort) aus der Datei `wordinfo.rda`) hinzu und nutzen Sie dabei die Funktion `match`. Hinweis: Die Korpusfrequenzen lassen sich mit `read.csv` einlesen.

2. Ein Wort fehlt in der Datenbank: welches?

3. Stellen Sie durch Tests mit `stopifnot` sicher, dass (a) in `wrdinfo` tatsächlich für jeden Probanden 1 Zeile pro Wort enthält, und (2) dass tatsächlich die passenden Wörter aus `frq` und `wrdfile` zusammengefügt wurden. Hinweis: Nutzen Sie für (2) `gsub('[[:punct:]]', '', ...)` und `tolower`.

Hinzufügen von Sakkadeninformation zu Fixationsinformation

1. Die Dateien 02.fix und 02.sac enthalten Informationen über die mit dem SR Research Online-Parser detektierten Fixationen bzw. Sakkaden. Erstellen Sie einen `data.frame` namens `seq`, der an alle Fixationen die Eigenschaften der *nachfolgenden* Sakkade anfügt. Hinweis: Nutzen Sie die Funktion `match` und beachten Sie, dass der SR Parser den Beginn einer Sakkade stets mit dem auf das Ende der vorhergehenden Fixation folgenden Zeitstempel markiert.

2. Einige Fixationen in `seq` werden nicht von Sakkaden gefolgt. Was zeichnet diese Fixationen aus?

Sakkadendetektion

Für Sakkaden gilt eine angenähert lineare Beziehung zwischen Amplitude und Spitzengeschwindigkeit über einen großen Bereich, die deshalb meist in log-log-Koordinaten dargestellt wird (Zuber, Stark & Cook, 1965). Diese Beziehung Sakkaden nennt man in Anlehnung an den astronomischen

Begriff auch `main sequence' (Bahill et al., 1975). Die mit `microsacc` detektierten Ereignisse folgen ebenfalls der `main sequence', siehe Abbildung 1.

```
datadir <- '~/Projects/work/eyelinkTest/AAA/daten/Experiment'
seqfile <- file.path(datadir, '/fix_seqs/AAA_allsac_all_RIGHT.seq')
dat <- read.table(file=seqfile, header=TRUE)
# a 3-panel plot
par(mfcol=c(1,3))
# histogram of fixation durations
ix1 <- dat$fdur < 1000
hist(dat[ix1,"fdur"], 100, xlab = "fixation duration [ms]",
   main = "fixation duration")
# histogram of saccade durations
ix2 <- dat$sacdur < 100
hist(dat[ix2,"sacdur"], 100, xlab = "saccade duration [ms]",
   main = "saccade duration")
# main sequence plot
ix <- dat$peakvel < 2000
logampl <- log10(sqrt(dat[ix,]$xampl^2+ dat[ix,]$yampl^2))
logvel <- log10(dat[ix,]$peakvel)
plot(logampl, logvel, pch='.', xlab="log amplitude",
   ylab="log velocity", main="main sequence")
# regressionskoeffizienten
lmva <- lm(logvel ~ 1 + logampl)
coef(lmva)
summary(lmva)
abline(reg=lmva)
```

Sakkaden beim Lesen lassen sich als extreme Ausreißer hinsichtlich der Geschwindigkeitsverteilung des Auges charakterisieren. Die meiste Zeit ruht das Auge annähernd während der Fixationen, nur während der relativ kurzen Sakkaden bewegt es sich ruckartig sehr schell. Deshalb eignen sich bei hinreichend hoher zeitlicher Auflösung der Messung geschwindigkeitsbasierte Algorithmen gut zur Detektion von Sakkaden und Fixationen. Mit `vecvel` und `microacc` enthält das Paket `eyetrackR` Routinen, die eine solche geschwindigkeitsbasierte Detektion erlauben. Durch `removeGlissades` können Überschwing- und Korrektursakkaden oder -artefakte bereinigt werden.

Aufgabe

`data(trial4)`

Die Daten für `trial4` stammen aus einem Lifespan-Leseexperiment und sind mit einer Abtastrate von 500 Hz aufgenommen. Für die Geschwindigkeitsberechung empfiehlt sich stärkere Glättung (`TYPE=2` im Aufruf von `vecvel`).

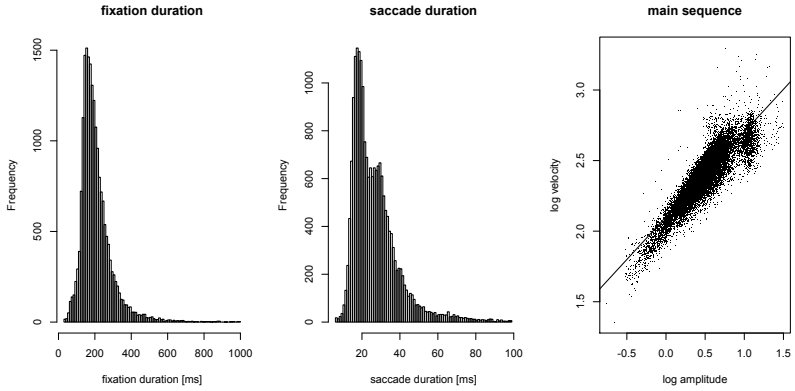

Abbildung 1: Main Sequence Plot für Potsdamer Sakkaden

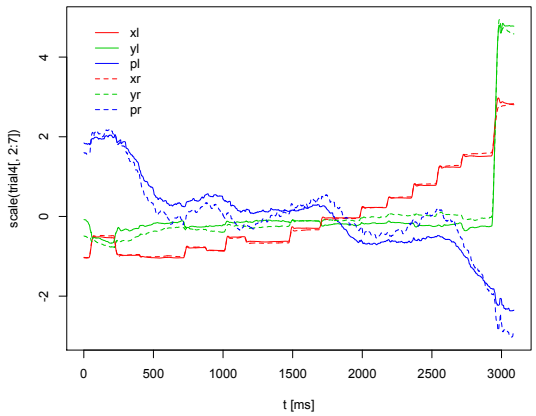

Abbildung 2: Beispiellösung Aufgabe 1

1. Lassen Sie sich grafisch die Rohdaten für einen Trial (`trial4`) anzeigen. Nutzen Sie dazu `matplot`, um den Verlauf der Variablen in den Spalten 2-7 über die Zeit (Spalte 1) darzustellen, siehe Abbildung 2.

2. Berechnen Sie mit `vecvel` die x-Geschwindigkeit und stellen Sie x-Position und -Geschwindigkeit grafisch dar. Nutzen Sie dazu zwei übereinander angeordnete Panels innerhalb eines Grafikfensters, siehe Abbildung 3. Beachten Sie, dass die Funktion `vecvel` immer die Zeitreihen für x- und y-Position benötigt.

3. Detektieren Sie Sakkaden mit der Funktion `microsacc`. Vergleichen

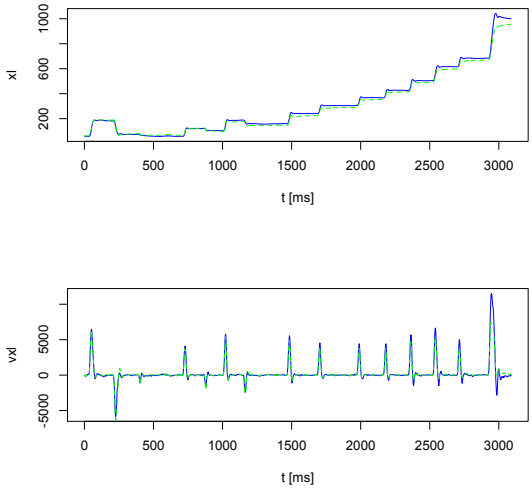

Abbildung 3: Beispiellösung Aufgabe 2

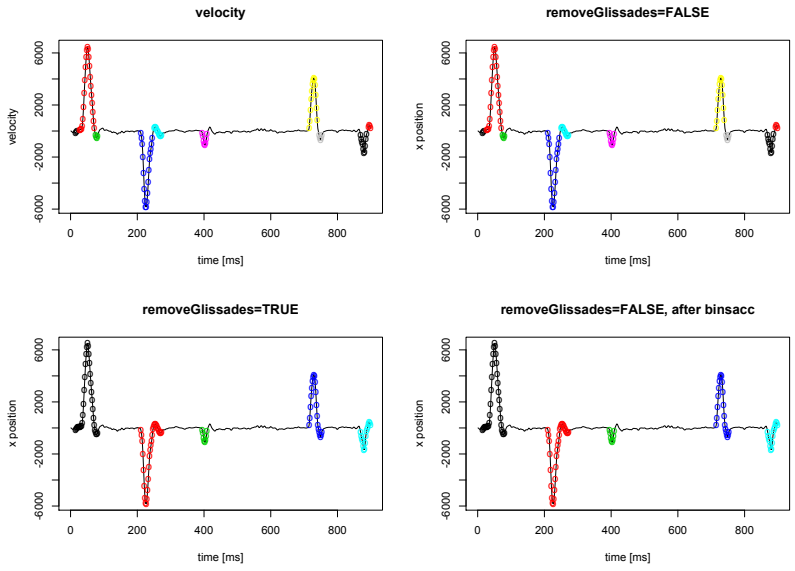

Abbildung 4: Beispiellösung Aufgabe 3

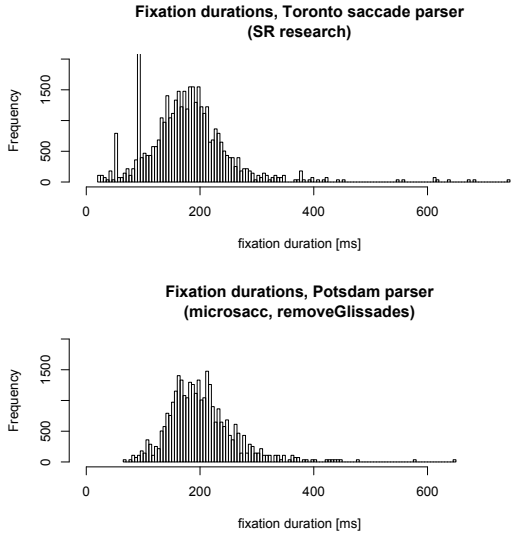

Abbildung 5: Vergleich Sakkadendetektion microsacc/removeGlissades vs. SR online parser

Sie die Sakkadendetektion (a) mit und (b) ohne Entfernen von Glissaden und stellen Sie das Ergebnis grafisch dar, siehe Abbildung 4.

4. Vergleichen Sie die mit `microsacc` und `removeGlissades` detektierten Sakkaden mit dem vom Online-Parser des Herstellers (SR research) detektierten, siehe Abbildung 5. Nutzen Sie dazu die Datensatz `allSeqSR.rda` und `allSeqPotsdam.rda` (Die Daten stammen aus einem Experiment mit blickkontingenter akustischer Sprachpräsentation). Zum Nachvollziehen: Die Dateien sind mit den R-Skripten `beispiel_sequenz_aus_fix_und_sac_SR.R` bzw. `beispiel_sequenz_aus_fix_und_sac_Potsdam.R` erzeugt worden.

Berechnung aggregierter Maße aus Fixationssequenzen

Meist werden in einem weiteren Schritt der Vorverabeitung die Fixationen aus den Fixationssequenzen gemäß bestimmter Eigenschaften klassifiziert und weiter aggregiert. In der Leseforschung ist insbesondere die Unterscheidung zwischen Firstpass-Fixationen und Fixationen im zweiten und in weiteren Durchgängen wichtig. Beispielsweise ist ein häufig zur Beschreibung kognitiver Verarbeitung beim Lesen verwendetes aggregiertes Maß die so-

genannte *gaze duration*, definiert als Summe aller Firstpass-Fixationsdauern auf einem Wort.

Aufgaben

Klassifizieren von Firstpass-Fixationen

```
data(a)
str(a)
# 'data.frame': 4749 obs. of  91 variables:
# ...
ashrt <- a[order(a$id, a$sn), c("id", "sn", "wn", "nw")]
str(ashrt)
# 'data.frame': 4749 obs. of  4 variables:
#  $ id: int  1 1 1 1 1 1 1 1 1 1 ...
#  $ sn: int  1 1 1 1 1 1 1 1 1 1 ...
#  $ wn: int  1 2 3 4 5 4 6 7 9 10 ...
#  $ nw: int  10 10 10 10 10 10 10 10 10 10 ...
```

Der Datensatz in `a` enthält die Daten einiger Probanden eines Leseexperimentes im `distributed processing'-Format. Berechnen Sie durch Nutzung der Funktion `firstpass` die Variable `fpf` (bzw. `fpf1`), die angibt, ob eine Fixation zum ersten oder zu einem der weiteren Durchgängen gehört. Sie müssen dazu `firstpass` pro Vp und Satz anwenden.

- Lange Version: Berechnen Sie `fpf` durch eine Schleife über `unique` Werte von `a$id` und `a$sn`.
- Kurze Version: Berechnen Sie `fpf1` mit Hilfe von `tapply` und `unlist`, indem Sie zunächst einen geeigneten Index erstellen.
- Stellen Sie mit `identical` oder `all.equal` sicher, dass fpf1 und fpf identisch sind

Berechnung der Blickdauer (`gaze duration') als Summe der Firstpass-Fixationsdauern durations, Hinzufügen zu `a`

```
data(a)
ashrt <- a[order(a$id, a$sn), c("id", "sn", "wn", "nw", "dur")]
```

- Gegeben die Daten in `ashrt` und die oben berechnete Indikatorvariable `fpf`, berechnen Sie die Variable `gd`, die genauso lang ist wie `ashrt$dur`. Der Vektor `gd` soll jeweils für die erste Fixation im first pass eines Wortes die Blickdauer (also die Summe aller firstpass-Fixationsdauern)

enthalten und für alle anderen Fixationen auf NA gesetzt sein. Bei Refixationen wird hier also die Blickdauer der ersten Fixation der Sequenz zugewiesen. Hinweis: Nutzen Sie die Funktion `parseq`.

- Berechnen Sie für jeden validen Eintrag in `gd` die Anzahl der Fixationen im gaze und fügen Sie die Variable `ngz` zu `ashrt` hinzu.

Analyse im linear mixed model Ansatz

Wenn endlich alles vorbereitet ist, kann der Spaß beginnen. Linear mixed models haben im Vergleich zur klassischen ANOVA, in der Subjekt- und Itemanalyse (F1 und F2) zu konfligierenden Ergebnissen kommen können, u.a. den Vorteil eindeutiger interpretierbarer Aussagen. Mit dem R-Paket `lme4` lassen sich solche Analysen komfortabel durchführen. Gute Einführungen bieten Baayen (2008) oder Kliegl (2007), siehe auch

`http://supp.apa.org/psycarticles/supplemental/xge_136_3_530/xge_136_3_530_supp.html`.

Es folgt ein Beispiel (für die Bedeutung der Variablen im data frame `a` rufen Sie in R durch `?a` die entsprechende Hilfeseite auf):

```
library(eyetrackR)
data(a)
library(lme4)

a <- a[order(a$id, a$sn),]
key <- 1e5*a$id + a$sn
a$fpf <- unlist(tapply(a$wn, key, firstpass))

a$id <- as.factor(a$id)
a$wordId <- as.factor(100*a$sn + a$wn)
a$cwfw <- as.factor(a$cwfw)

idx <- a$wn2.x>a$wn & a$wn>a$wn1.x & a$fpf

varsToScale <- c("l", "f", "p", "ao1", "ao", "o", "l1", "l2", "p1", "p2", "f1", "f2")
a[,varsToScale] <- scale(a[,varsToScale])
a$l <- scale(a$l)
a$f <- scale(a$f)
a$p <- scale(a$p)

m1 <- lmer(dur ~ cwfw + l + f + p + ao1 + ao + ao2 + o + I(o^2) + (1|id) + (1|wordId), ...
   data=a, subset=idx)
m2 <- update(m1, ~ . + l1 + f1 + p1 + l2 + f2 + p2)
m3 <- update(m3, ~ . + f*l + p2*l)

anova(m1, m2, m3)
summary(m2)
```

Aufgaben

- Führen Sie die oben stehenden Befehle aus und interpretieren Sie die Ausgabe.
- Untersuchen Sie, ob das Überspringen des vorigen Wortes einen Effekt auf die Fixationsdauer hat (im Vergleich zum Modell `m2`). Berechnen Sie dazu einen Faktor, der als Indikatorvariable kodiert, ob das vorige Wort übersprungen wurde oder nicht. Beeinflusst Überspringen des vorigen Wortes interaktiv mit Worttyp (`a$cwfw`) oder Frequenz des vorigen Wortes (`a$f1`) die Fixationsdauer?

Echte Daten

Das R-Paket eyetrackR enthält einige Hilfsroutinen, die bei der Vorverarbeitung von eyetracker-Daten nützlich sind sowie eine Reihe von Demos, die als Vorlage für eigene Analysen genutzt werden können (demos liegen nach Installation von eyetrackR in `file.path(.libPaths()[1], "eyetrackR", "demo")`). Die folgenden Schritte fallen üblicherweise bei der Vorverarbeitung der Daten an:

1. Hilfsroutinen
    - `preprocess` oder edf2asc: Umwandeln der Eyelink-Binärdateien in Text (ASCII)
    - `readSubjects`: Lesen einer Liste von Vp-Codes aus einer Datei und Test auf Vorhandensein der korrespondierenden Datensätze
    - `parseq`: allgemeine Hilfsroutine, Detektion von Start, Ende und Länge von Sequenzen
    - `firstpass`: Klassifikation von first pass vs. Rest
    - `vecvel`: Geschwindigkeit des Auges berechnen
    - `microsacc`: Sakkadendetektion
    - `removeGlissades`: Entfernung von Glissaden (over-/undershoot)
    - `binsacc`: implementiert binokuläres Sakkadenkriterium, detektiert und kombiniert außerdem Sakkadencluster
    - `saccpar`: berechnet und kombiniert Parameter für binokuläre Sakkaden
    - `wordLetterPos`: konvertiert Fixationsposition [Pixel] in Wort- und Buchstabenposition (nimmt z.Zt. Font mit fester Laufweite und bekannter Breite in Pixel an)

- `probdur1`: Berechnung aggregierter Maße aus Fixationssequenz für einen Satz

2. Demos

   - `xallsac_EXP`: Daten-Management, ruft preprocessing für jede Vp auf und akkumuliert Ergebnisse; Output: Fixationssequenz
   - `parse_msg_file_EXP`: parse eyetracker message files
   - `xSetup1_EXP`: Fixationssequenzen aufbereiten zur Analyse im verteilten Format, d.h. unter Analyse von Maßen des fixierten Wortes unter Berücksichtigung der Eigenschaften benachbarter oder auch unmittelbar vorher bzw. anschließend fixierter Wörter
   - `xIndex_fix_EXP_logical.R`: Erstellen einiger nützlicher Indizes, z.B. für valide first-pass Fixationen
   - `xSetup2_EXP`: Zentrieren und Transformieren von Prädiktoren; Fallauswahl für weitere Analyse (z.B. Beschränkung auf valide Firstpass-Fixationen, auf (n-1, n, n+1)-Fixationstriplets etc.)
   - `xProbdur_EXP`: Erzeugen deskriptiver Statistiken über Wahrscheinlichkeiten, Fixationsdauern und -Positionen für first-pass und second-pass Lesen

Die übliche Reihenfolge bei der Analyse ist

1. `preprocess`
2. `xallsac_EXP`, `parse_msg_file_EXP`, Anpassen der Parameterdatei
3. `xSetup1_EXP`
4. `xIndex_fix_EXP_logical`
5. `xSetup2_EXP`, ggf. auch `xProbdur_EXP`
6. Analyse mit linear mixed models