Einführung in Umweltanalysen mit R

Leitprogramm
Klemens Rosin

Inhalt:
R ist ein sehr leistungsfähiges Programm, mit dem Daten analysiert und dargestellt werden können. Es ist beabsichtigt, dass sich die Lernenden mit diesem Leitprogramm ein fundiertes Grundwissen bezüglich R aneignen, mit dem sie konkrete Analysen durchführen können.

Unterrichtsmethode: Leitprogramm

Fachliches und fachdidaktisches Review:
Gianpietro Cerletti, Kantonsschule Zug

Publiziert auf EducETH:
8. April 2011

Einführung in Umweltanalysen mit R

Leitprogramm

K. Rosin

Institution:	ETH Zürich, Institut für Verhaltenswissenschaften
Unterrichtsform:	Leitprogramm
Thema:	Einführung in Umweltanalysen mit R
Fachgebiet:	Umweltwissenschaften
Unterrichtsstufe:	Hochschule, Fachhochschule, Gymnasium
Vorkenntnisse:	Grundlagen in Statistik und Umweltwissenschaften
Bearbeitungsdauer:	Sechs Lektionen (je etwa drei Stunden)
Autor:	K. Rosin, D-BAUG, rosin@ifu.baug.ethz.ch
Betreuung:	Dr. G. Cerletti
Kolektorat:	B. Bründler, D. Rosin
Fassung:	Oktober 2008

Inhaltsverzeichnis

1. Arbeitsanleitung

Wie funktioniert ein Leitprogramm?

Ein Leitprogramm ist eine schriftlich formulierte Anleitung, mit der Sie sich den vorgesehenen Lernstoff in Ihrem Tempo erarbeiten. Es besteht aus mehreren Kapiteln, die jeweils gleich aufgebaut sind: Zuerst werden Sie mit einem kurzen Text in das Thema eingeführt. Danach werden die Lernziele genannt, welche Sie am Schluss des Kapitels erreichen sollen. Anschliessend beginnt der Hauptteil des Kapitels, der aus Texten, Arbeitsaufträgen und Übungsaufgaben besteht. Befassen Sie sich ruhig und gründlich damit. Am Ende jedes Kapitels befinden sich Lernkontroll-Aufgaben. Damit stellen Sie selbstständig fest, ob Sie die Lernziele eines Kapitels erreicht haben. Bevor Sie sich mit dem nächsten Kapitel befassen, müssen Sie einen Kapiteltest bestehen. Wenn Sie sich genügend sicher fühlen, melden Sie sich bei der Lehrperson, um den Kapiteltest zu schreiben. Erst wenn Sie diesen bestehen, dürfen Sie mit dem nächsten Kapitel beginnen. Falls Sie den Kapiteltest nicht bestehen, repetieren Sie das entsprechende Stoffgebiet, bevor Sie sich zu einem weiteren Test melden. Beim Kapiteltest dürfen Sie das abgegebene Merkblatt verwenden, auf dem Sie im Notizen-Bereich eigene Kommentare anbringen dürfen.

Symbole

In diesem Leitprogramm werden unterschiedliche Teile mit folgenden Symbolen gekennzeichnet:

Symbol	*Erklärung*
	Ergänzende Informationen (nicht prüfungsrelevant).
	Lernziele.
	Arbeitsauftrag (grauer Hintergrund).
	Übungsaufgabe.
	Merkpunkte.
	Zusatzinformation/Zusatzaufgaben.
	Lernkontrolle.
	Kapiteltest.

Inhaltliche Übersicht

R ist ein sehr leistungsfähiges Programm, mit dem Sie Daten analysieren und darstellen können. Es lohnt sich deshalb, dass Sie zuerst den Aufbau von R kennen lernen (Kapitel 2 und 3). Wenn Sie später R anwenden, zahlt es sich aus, dass Sie die Struktur von R kennen! In Kapitel 4 lernen Sie mit R Datensätze zu ordnen, sortieren und vereinigen. Im Gegensatz zu Microsoft Excel können Sie mit R grosse Datensätze wesentlich effizienter bearbeiten. In R ist eine Vielzahl mathematischer Funktionen erhältlich. Einige davon lernen Sie in Kapitel 5 kennen. Dieses Kapitel enthält auch eine Einführung in Programmieren mit R. In Kapitel 6 eignen Sie sich Fertigkeiten an, wie Sie mit R Daten analysieren und beschreiben können. Ein Schwerpunkt dieses Kapitels ist das Erstellen von Grafiken mit R. Im Kapitel 7 lernen Sie wie mit R statistische Modelle erzeugt und getestet werden. Im Zentrum dieses Kapitels stehen lineare Regressionsmodelle dar. Es ist beabsichtigt, dass Sie sich mit diesem Leitprogramm ein fundiertes Grundwissen bezüglich R aneignen, mit dem Sie konkrete Analysen durchführen können. Zudem sollten Sie durch diese Einführung ermuntert werden, ihr R-Wissen zu vertiefen!

Schreibweisen

Englische Wörter, Programm-Befehle, R-Objekte und zusammenhängende Begriffe werden *kursiv* geschrieben. Für Programmteile wird die Schrift Tahoma verwendet. Falls nur bestimmte Teile eines bestehenden Programmiercodes geändert werden, sind die Änderungen in dunkelgrau deutlich hervorgehoben.

Betriebssysteme

Dieses Leitprogramm wurde für Windows-Betriebssysteme erstellt. Falls Ihr Computer über ein anderes Betriebssystem verfügt, sollten Sie die Lehrperson kontaktieren.

Vorausgesetztes Wissen

Bei diesem Leitprogramm wird ein Grundlagenwissen bezüglich Umweltwissenschaften und Statistik vorausgesetzt. Umweltsysteme wie beispielsweise der Wasserkreislauf und dazugehörende Wasserinhaltsstoffe sollten Ihnen grundsätzlich bekannt sein. Zudem wird vorausgesetzt, dass Sie über das Wissen einer einführenden Statistikveranstaltung verfügen. Beispielsweise werden grundlegende Begriffe wie Mittelwert oder Standardabweichung in diesem Leitprogramm als bekannt vorausgesetzt.

2. Erste Schritte mit R und TINN-R

Übersicht

In diesem Kapitel lernen Sie die Grundfunktionen der für Umweltanalysen sehr hilfreichen Programme R und TINN-R kennen. Dazu gehört, wie die Programme geöffnet, miteinander verbunden sowie korrekt beendet werden. Zudem werden einige Quellen aufgezeigt, wo Sie bei Problemen Hilfe erhalten. Zusammenfassend werden in diesem Kapitel Grundlagen vermittelt, damit Sie später mit R und TINN-R umweltrelevante Aufgaben anpacken können.

Lernziele

Nachdem Sie dieses Kapitel bearbeitet haben, sollen Sie

- R und TINN-R korrekt aufstarten und beenden können (K2).
- das Laden und Installieren von Zusatzpaketen beherrschen (K3).
- drei Möglichkeiten aufzählen, wo Sie Informationen finden, um sich bei Problemen selbst helfen zu können (K1).

2.1. Informationen zu R und TINN-R

Geschichte von R

In den 70er Jahren ist die Programmiersprache S entstanden, welche in den letzten Jahrzehnten weiterentwickelt wurde. Unter anderem ist daraus die kommerzielle Software S-PLUS hervorgegangen. 1992 beginnen Ross Ihaka und Robert Gentleman in Neuseeland ihr R-Projekt. R ist eine kostenlose Implementation von S, basiert auf dessen Grundideen, verwendet aber nicht die gleiche Codierung. R entwickelte sich rasant: 1997 schliessen sich 17 Forscher und Wirtschaftsvertreter zum *R Development Core Team* zusammen. 1998 wird das *Comprehensive R Archive Network* (CRAN) gegründet. Heute ist R eine weit verbreitete Programmiersprache für Statistikanwendungen.

Nutzen von R

R ist eine kostenlose, vielseitig einsetzbare Programmier-Software. Der Quellcode von R ist frei zugänglich (*open source*). R kann unter anderem für Folgendes verwendet werden:

- Datenauswertung.
- Statistische Analysen.
- Grafische Darstellungen.

TINN-R

TINN-R ist ein auf R abgestimmter, kostenloser Editor. TINN-R erleichtert das Arbeiten mir R erheblich, kann aber nur mit Windows-Betriebssystemen verwendet werden.

Namen

Es wird vermutet, dass R nach den Anfangsbuchstaben der Vornamen von Ross Ihaka und Robert Gentleman in Anlehnung an S benannt worden ist. Es ist aber auch möglich, dass R für *Reduced version of S* steht. Der Name TINN-R ist von der rekursiven Schreibweise *Tinn Is Not Notepad* abgeleitet, die darauf hinweist, dass TINN-R mehr als nur ein einfaches Editor-Programm ist.

CRAN

Das *Comprehensive R Archive Network* (*CRAN*) ist ein weltweites Server-Netzwerk auf dem R-Dokumente verschiedener Art gespeichert sind. Unter anderem sind dort der R-Quellcode und Zusatzpakete von R abgelegt. Falls Sie später Daten von *CRAN* runterladen, sollten Sie dies von einem möglichst nahe gelegenen Server tun. Die Download-Server zu *CRAN* werden *Mirror* genannt.

R-Homepage

Unter www.r-project.org befindet sich die Homepage von R. Dort sind Links zu Dokumentationen oder zum CRAN zu finden. Die R-Homepage dient zudem als Einstieg, um R herunter zu laden.

- Schauen Sie sich während etwa zwei Minuten auf der R-Homepage (www.r-project.org) um.

Falls R und TINN-R auf Ihrem Computer noch nicht installiert sind, führen Sie die Installation nun durch (Anleitung siehe Anhang B). Das vorliegende Leitprogramm basiert auf den Versionen R-2.11.1 und TINN-R-1.17.2.4. Falls auf Ihrem Computer ältere Versionen installiert sind, wenden Sie sich an die Lehrperson, um zu entscheiden, ob die Programme neu installiert werden müssen.

2.2. R starten

Ordner-Struktur

Nun werden Sie R gleich starten. Es lohnt sich jedoch vorher auf Ihrem Computer eine sinnvolle Ordnerstruktur anzulegen. In Rahmen dieses Leitprogramms ist von der Lehrperson auf dem D-Laufwerk ihres Computers bereits ein Ordner angelegt worden (*E:\R\temp*). Wenn Sie in Zukunft selbständig mir R arbeiten, wird empfohlen, dass Sie für Ihre Projekte eine sinnvolle Ordnerstruktur anlegen, und die Ordnernamen auf Inhalte beziehen (beispielsweise: *C:\R\Schnee_Freiburg* und nicht *C:\R\Projekt4*).

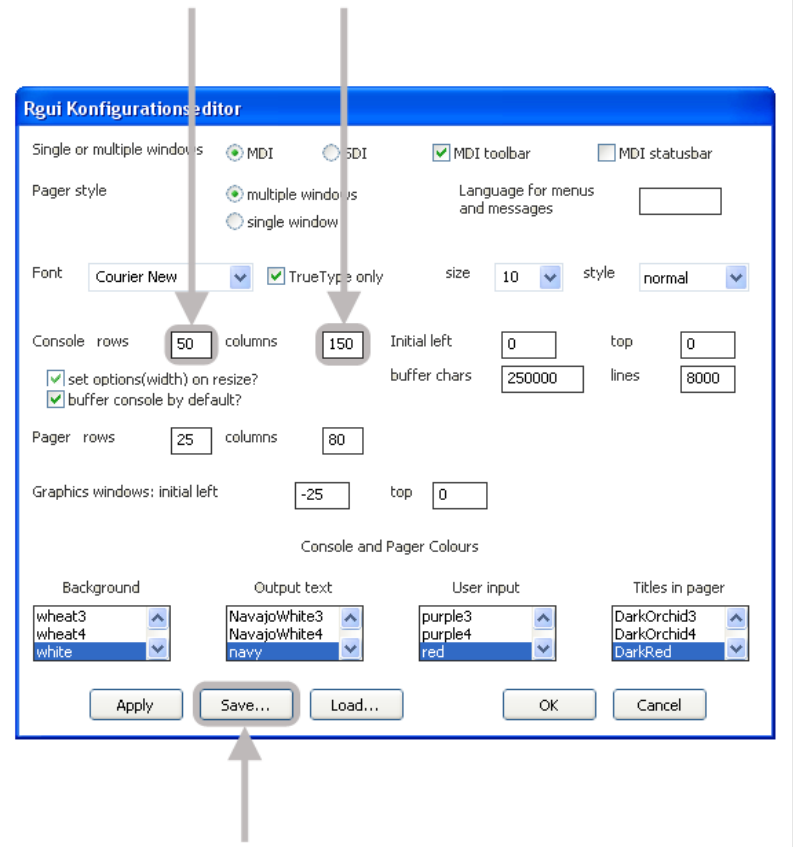
R starten

Sie können R nun starten!

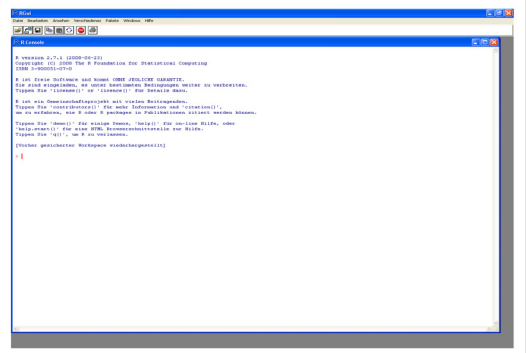
- Gehen Sie auf *Start/Programme/R*, und klicken Sie auf *R 2.8.1* um R zu öffnen. Das Eingabefenster von R, die so genannte R-Konsole, öffnet sich.
- Klicken Sie im Menüpunkt *Bearbeiten* auf *GUI Einstellungen…* um die Grösse der R-Konsole zu wählen.

R-Konsole

- Stellen Sie die Grösser der R-Konsole auf 50 Zielen (*rows*) und 150 Spalten (*columns*) ein, und speichern Sie diese Einstellungen. Übernehmen Sie den angegebenen Speicherpfad.



- Die R-Konsole sieht nun wie folgt aus:



Aufforderungs-Zeichen

Das *Grösser-Zeichen* (>) wird von R als Eingabeaufforderung verwendet. Es ist die Art und Weise von R zu fragen: Was folgt nun? Das *Plus-Zeichen* (+) verwendet R als *Weiterfahren*-Aufforderung, wenn ein eingegebener Befehl nicht komplett ist. Nachdem Sie einen Befehl eingegeben haben, müssen Sie jeweils die Enter-Taste betätigen, um diesen Befehl in R laufen zu lassen.

Arbeitsplatz und Arbeitsverzeichnis

Schon bald werden Sie intensiv mit R arbeiten. Beim Arbeiten mit Daten und Funktionen werden in R so genannte Objekte erstellt. Diese Objekte werden am jeweiligen Arbeitsplatz (Englisch *workspace*) abgelegt. Sie können Ihren Arbeitsplatz mit dem Befehl *save.image()* speichern. Dieser wird damit in einer so genannten *.Rdata* Datei gespeichert. Sie können einen gespeicherten Arbeitsplatz mit *load()* zu dem aktuellen Arbeitsplatz dazu laden. Den Arbeitsplatz in R können Sie mit der realen Welt vergleichen, bei der sich Objekte an ihrem Arbeitsplatz befinden. Wie in Realität, können Sie in R ebenfalls Daten von anderen Orten zu ihrem Arbeitsplatz holen, oder von Ihrem Arbeitsplatz an einem anderen Ort ablegen. Wo Daten geholt oder abgelegt werden, wird mit der Wahl des Arbeitsverzeichnisses (Englisch: *working directory*) festgelegt.

Arbeitsverzeichnis anschauen und ändern

Es kann vorkommen, dass das Arbeitsverzeichnis geändert werden muss, um beispielsweise von einem anderen Verzeichnis Daten zu Ihrem Arbeitsplatz zu importieren. Das kann direkt im R-Eingabefenster gemacht werden. Dazu wird aber zuerst abgeklärt, welches Arbeitsverzeichnis gerade verwendet wird.

- Geben Sie *getwd()* in das Eingabefenster ein, und klicken Sie auf die *Enter* Taste. Der Befehl *getwd* steht für *Get Working Directory* (Deutsch: Zeige das Arbeitsverzeichnis an). R gibt das aktuelle Arbeitsverzeichnis aus.
- Um das Arbeitsverzeichnis beispielsweise auf den Ordner mit den Daten zu ändern, geben Sie nun *setwd("E:/R")* in das Eingabefenster ein, und betätigen die *Enter* Taste. R hat nun das neue Arbeitsverzeichnis übernommen.
- Wichtig: *Forward-Slash* (/) und nicht *Back-Slash* (\) verwenden.

Befehl wiederholen

Mit der Pfeiltaste (Pfeil nach oben), können zuvor eingegebene Befehle wieder aufgerufen werden.

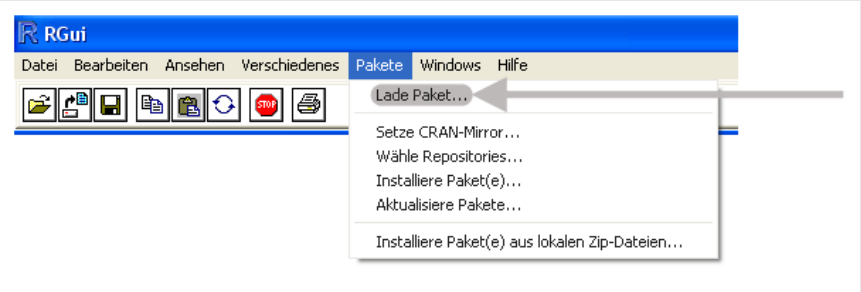
- Drücken Sie die Pfeiltaste (Pfeil nach oben).
- Ergänzen Sie die Eingabe zu *setwd("E:/R/temp")*.

2.3. Pakete installieren und laden

In R sind bereits viele Funktionen vordefiniert. Sie können aber noch weitere Prozeduren dazuladen. Diese sind in so genannten Paketen (Englisch: *Packages*) verpackt. Einige Pakete sind durch das Installieren von R bereits auf Ihrem Computer vorhanden. Sie müssen aber noch geladen werden. Andere Pakete müssen zuerst über einen CRAN-Mirror bezogen, und auf Ihrem Computer installiert werden, bevor sie geladen werden können.

Laden eines bereits auf dem Computer vorhandenen Pakets

- Geben Sie *library()* in R ein. In einem neuen Fenster wird Ihnen die „Bibliothek“ der bereits auf Ihrem Computer vorhandenen Pakete angezeigt. Dazu gehören beispielsweise die Pakete *stats* und *graphics*. Diese Pakete werden Sie gleich laden. Schliessen Sie das Fenster mit der Paket-Liste, indem Sie auf die obere rechte Ecke klicken.
- Sie können ein Paket über die Menüleiste laden. Dazu wählen Sie *Lade Paket…* im Menü *Pakete.*



- Wählen Sie das Paket *stats* aus, und klicken Sie *OK*. Das Paket *stats* ist nun geladen.
- Sie können ein Paket aber auch mit einem Befehl laden. Geben Sie *require(graphics)* in das Eingabefenster ein, und drücken Sie auf die *Enter* Taste. Damit ist das Paket *graphics* ebenfalls geladen.

Hinweis: Nachdem das Paket geladen worden ist, gibt R manchmal eine falsche Meldung aus („lade nötiges Paket“). Diese hat keine Bedeutung.

Installieren eines Paketes

Falls Sie ein zusätzliches, auf Ihrem Computer nicht vorhandenes Paket benützen wollen, können Sie dieses über das Internet von einen *CRAN-mirror* herunterladen, und auf Ihrem Computer installieren. Ein Paket kann entweder über die Menüleiste oder mit Befehlen installiert werden. In beiden Fällen müssen Sie aber einen *CRAN-mirror* auswählen. Solange R geöffnet bleibt, müssen Sie den *CRAN-mirror* nur einmal auswählen.

- Gehen Sie online (VPN client „*Internet Einwahl*“ auf dem Desktop).
- Wählen Sie *Installiere Paket(e)…* im Menü *Pakete.*
- Wählen Sie im neuen Fenster den CRAN mirror *Switzerland* aus und klicken Sie *OK*. Es erscheint nun eine Liste aller vorhandenen Pakete.
- Suchen Sie das Paket *RSAGA*, wählen Sie es an, und klicken Sie *OK*.
- Falls Sie auf ihrem Rechner nicht über Installationsrechte für die allgemeine Bibliothek (*C:\Programme\R\R-2.11.1\library*) verfügen, kann es sein, dass Sie gefragt werden, ob unter *C:\Dokumente und Einstellungen\student\Eigene Dateien\R\win-library\2.8* eine persönliche Bibliothek erstellen werden soll. Klicken Sie auf *Ja*.
- Benützen Sie den vorher kennen gelernten *library*-Befehl, um zu überprüfen, ob das Paket *RSAGA* jetzt auf ihrem Computer verfügbar ist. Sie sehen aber auch, dass nun zwei Bibliotheken vorhanden sind.
- Laden Sie das Paket mit dem Befehl *require(RSAGA).*

Sie können ein Paket auch mit einem R-Befehl installieren. Mit *install.packages()* erhalten Sie einen Überblick über alle Pakete; falls Sie ein bestimmtes Paket wie beispielsweise *gstat* installieren wollen, können Sie dieses mit *install.packages("zoo")* tun. Beachten Sie, dass der Paketname dabei in Anführungszeichen stehen muss.

Übungsaufgabe 2.1

Mit den folgenden drei Aufgaben üben Sie das Installieren und Laden von Pakteten. Die Lösungen zu den Übungsaufgaben finden Sie anschliessend an die Aufgabenstellung.

1) Überprüfen Sie, ob die Pakete *Kendall* und *cluster* bereits auf Ihrem Computer vorhanden sind.
2) Falls die Pakete auf Ihrem Computer nicht vorhanden sind, installieren Sie diese.
3) Laden Sie beide Pakete.

Lösungen zu Übungsaufgabe 2.1

1) Eingabe: *library().* Die Liste im aufgehenden Fenster weist darauf hin, dass das Paket *cluster* bereits auf dem Computer vorhanden ist, das Paket *Kendall* hingegen nicht.
2) *install.packages("Kendall")*
3) *require(Kendall); require(cluster)*

2.4. Hilfe

Falls bei Ihrer zukünftigen Arbeit mit R Probleme auftreten, können Sie diese oft selbständig lösen. Denn es besteht ein grosses Angebot an Hilfestellungen. In diesem Kapitel werden einige Möglichkeiten erläutert.

Online-Hilfe: Häufige Fragen

Auf der R-Webseite (www.r-project.org) sind Antworten zu häufig gestellten Fragen (Englisch: *Frequently Asked Questions*, *FAQs*) aufgeführt.

- Öffnen Sie die R-Homepage (www.r-project.org); auf der linken Seite klicken Sie unter *Documentation* auf *FAQs*.
- Dort klicken Sie auf *R FAQ*, um Antworten zu allgemeinen Fragen zu erhalten.
- Finden Sie zwei Fragen, die für Sie als relevant betrachten.

Online-Hilfe: Dokumente

Über die R-Webseite können Sie zu unzähligen Dokumentationen gelangen.

- Öffnen Sie die R-Homepage (www.r-project.org); auf der linken Seite klicken Sie unter *Documentation* auf *Manuals*. Sie finden dort einige R-Dokumente.
- Zuunterst klicken Sie auf *contributed documentation*, um zu Dokumenten zu gelangen, die von Leuten eingereicht wurden, die nicht zum engsten R-Entwicklerkreis gehören. Diese Webseite ist nach Sprachen und Umfang der Dokumente (mehr oder weniger als hundert Seiten) gegliedert.
- Schauen Sie sich kurz zwei Dokumente an. Versuchen Sie zu beurteilen, ob sie für Sie hilfreich sein könnten (verständlicher Schreibstil, interessante Themen, anschauliche Beispiele).

Online-Hilfe: Suchen

Falls Sie Funktionen oder Pakete suchen, deren Namen Sie nicht kennen, kann die Suchfunktion auf der R-Homepage weiterhelfen.

- Öffnen Sie die R-Homepage (www.r-project.org); auf der linken Seite klicken Sie unter *R Project* auf *Search*.
- Geben Sie *cluster* in das Suchfenster ein, und klicken Sie auf *Google Search*. Schauen Sie sich die Suchergebnisse kurz an.

Online-Hilfe: Pakete

Von der R-Homepage aus gelangen Sie über einen Mirror zu einer CRAN-Webseite. Auf dieser sind unter anderem sämtliche Pakete beschrieben.

- Öffnen Sie die R-Homepage (www.r-project.org); auf der linken Seite klicken Sie unter *Download* auf *CRAN*.
- Um die Datenübertragung zu optimieren, wählen Sie einen möglichst nahegelegenen Mirror, und klicken unter *Switzerland* auf den Mirror *http://cran.ch.r-project.org/*. Nun sind Sie auf einer CRAN-Webseite.
- Klicken Sie auf der linken Seite unter *Software* auf *Packages*.
- Klicken Sie auf den Paketnamen *zoo*, um eine Übersichts des Paktes zu erhalten.
- Klicken Sie in der drittuntersten Zeile auf *zoo.pdf*. Damit öffnen Sie eine pdf-Datei, welche das Paket detailliert beschreibt. Schauen Sie diese Datei kurz an, und schliessen Sie die *pdf-Datei* anschliessend wieder.

Online Hilfe Forum

Es bestehen verschiedene Online Foren, in denen R Probleme diskutiert werden.

- http://r-forge.r-project.org
- http://wiki.r-project.org
- http://search.r-project.org
- http://blog.revolution-computing.com

Das r-forge Forum ist zurzeit das umfassendste Forum.

- Schauen Sie sich in zwei der oben genannten Foren um.

Informationsquellen

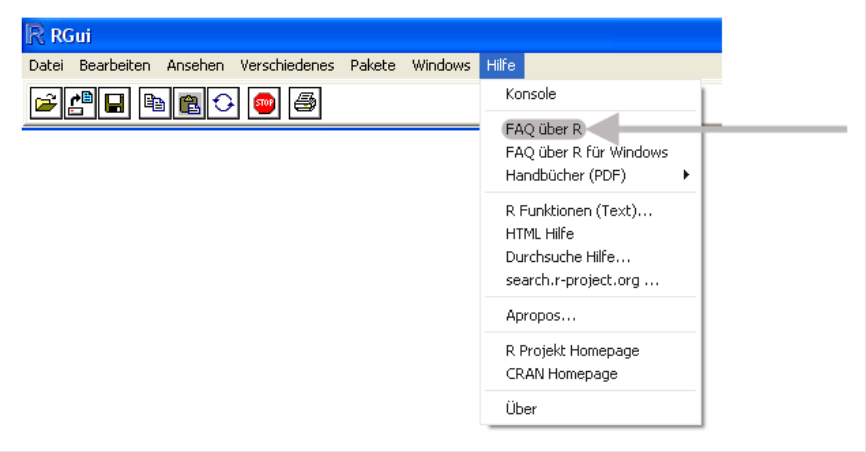
Um sich über R-Neuerungen zu informieren, werden die Rnews (http://cran.r-project.org/doc/Rnews/) oder entsprechende Mailing Listen (http://www.r-project.org/mail.html) empfohlen. Für ausgewählte Themen werden in sogenannten Task Views relevante Funktionen und Methoden erläutert (http://cran.r-project.org/web/views/).

- Schauen Sie sich die Task Vies Webseite an: http://cran.r-project.org/web/views/.
- Betrachten Sie insbesondere die Task Views über räumlich Daten (*Spatial*) und Zeitreihen (*TimeSeries*).

Hilfestellung über die R-Menüleiste

Die Hilfestellungen, die Sie vorher über die R-Homepage betrachtet haben, können Sie auch direkt von R aus aufrufen. Beispielsweise können Sie über R zu den FAQ Informationen gelangen. Der Vorteil dabei ist, dass Sie nicht online sein müssen.

- In dem geöffneten Programm R klicken Sie in der Menuleiste unter *Hilfe* auf *FAQ über R*.



- Sie können feststellen, dass hier die gleichen Inhalte vorhanden sind, die Sie vorher auf der R-Webseite gefunden haben. Nun können Sie das aufgegangene Fenster wieder schliessen.

Hilfestellung: R-Befehle

Es besteht auch die Möglichkeit, Hilfe direkt mit einem R-Befehl aufzurufen. Das ist dann sinnvoll, wenn Sie Hilfe für eine Ihnen bekannte Funktion benötigen.

- Geben Sie *?setwd* in die R-Konsole ein. Das aufgehende Fenster enthält Informationen zur *setwd*-Funktion.
- Schauen Sie sich bei dieser Hilfestellung um, indem Sie auf die Register *Contents*, *Index*, und *Search* klicken. Schliessen Sie danach das Hilfefenster wieder.

Falls Ihnen der Name einer Funktion nicht genau bekannt ist, helfen eventuell die Befehle *help.search* oder *apropos* weiter, wobei der gesuchte Name jeweils in Anführungszeigen angegeben werden muss. Beispielsweise schreiben Sie *help.search("getwd")*.

Manchmal ist es hilfreich, dass Sie sich ein Beispiel anzeigen zu lassen:

- Geben Sie *example(setwd)* in der R-Konsole ein. In der Konsole wird ein Beispiel der *setwd* Funktion angezeigt.
- Für einige Themen sind auch Demonstrationen vorhanden. Geben Sie *demo(graphics)* ein, und klicken Sie solange *Enter*, bis die Demonstration beendet ist. Schliessen Sie danach das aufgegangene Grafikfenster.

Sie haben gesehen, dass die Hilfestellungen in R sehr vielseitig sind. Damit Sie die Übersicht behalten, sind in der folgenden Tabelle die wichtigsten Hilfeleistungs-Möglichkeiten zusammengefasst:

Online-Hilfe (Einstieg über www.r-project.org)	Häufige Fragen (FAQ) Dokumente Suche Foren, Newsletters, Mailinglisten Pakete (auf einer CRAN-Webseite)
In R (Menüleiste)	Vorteil: Sie müssen nicht online sein Nachteil: Nicht sehr ausführlich
In R (Befehle)	*?*-Funktion *example()*-Funktion

Übungsaufgabe 2.2

1) Finden Sie ein R-Dokument in Deutscher Sprache. Wie heisst es?
2) Suchen Sie über die Online-Hilfe Informationen zu Regressions-Analysen.
3) Finden und öffnen Sie die pdf-Datei zum *GRASS*-Paket.
4) Finden Sie mit einem R-Befehl heraus, was die *plot*-Funktion bewirkt.
5) Rufen Sie Beispiele zur *plot*-Funktion ab.

Lösungen zu Übungsaufgabe 2.2

1) "Einführung in R" von Günther Sawitzki ist das einzige deutsche Dokument. Es kann wie folgt gefunden werden: Unter www.r-project.org klicken Sie auf der linken Seite unter *Documentation* auf *Manuals*, und weiter auf *contributed documentation*.
2) Lösungsweg: Geben Sie unter *R Project/Search* das Wort *Regression* ein.
3) Lösungsweg: Die Dokumentation zum *GRASS*-Paket wird über die CRAN Webseite gefunden.
4) Eingabe des folgenden Befehls: *?plot*
5) Eingabe des folgenden Befehls: *example(plot)*

Schliessen Sie jetzt R, indem Sie auf die rechte obere Ecke klicken, und die Frage *Save workspace image?* mit *No* beantworten. Damit werden die Änderungen Ihres Arbeitsplatzes nicht gespeichert. Sie werden im Kapitel 2.6 gleich lernen, wie R am Besten beendet wird. Zuerst werden Sie sich aber mit dem Programm TINN-R befassen.

2.5. Mit TINN-R arbeiten

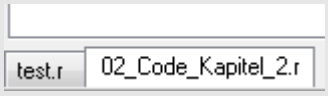
Bis jetzt haben Sie gesehen, wie R von der R-Konsole bedient wird. R lässt sich jedoch mit TINN-R etwas eleganter nutzen. TINN-R verfügt über typische Windowsmenüs wie *File*, *Edit*, *Format*, *Options*, *Tools*, *Search*, *View* und *Help*. Es lohnt sich, dass Sie einige dieser Menüs kennen.

Tinn-R starten, abspeichern

- Klicken Sie unter *Start/Programme/TINN-R* auf *TINN-R,* um das Programm zu öffnen.
- Unter *File*, wählen Sie *Save as*. Navigieren Sie zu Ihrem Arbeitsordner *(E:\R\temp)*, und speichern Sie Ihre TINN-R Datei unter dem Namen *test.r* ab.

.R Dateien öffnen

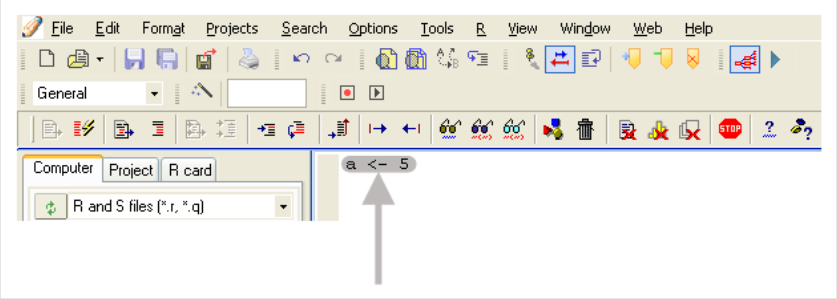
- Öffnen Sie in TINN-R mit *File/Open* die Datei *Code_Kapitel_2.r*, welche sich im Ordner E:\R\6_Code befindet.
- In der linken unteren Ecke von TINN-R können Sie zwischen den beiden Dateien hin und her welchseln.



Sie können wählen, ob Sie in diesem Leitprogramm die R-Befehle von Hand anhand des Skripts in die *test.r* Datei eingeben, oder ob Sie mit den vorgegebenen .r Dateien der einzelnen Kapitel arbeiten. Es wird empfohlen, dass Sie die bereits erstellten .r Dateien verwenden, damit Sie Zeit sparen, und sich keine Kopierfehler einschleichen.

Wiederherstellen

- Schreiben Sie `a <- 5` in das Eingabefenster von TINN-R (siehe Pfeil). Damit definieren Sie das Objekt *a*, und ordnen ihm den Wert fünf zu.



- Wählen Sie *Edit/Undo*, und beobachten Sie, was passiert. Klicken Sie danach unter *Edit* auf *Redo*.

Gross-, Kleinschreibung

- Markieren Sie nur den Buchstaben a.
- a <- 5
- Klicken Sie auf *Format/Selection/Uppercase*; was hat sich verändert?
- Wählen Sie das (nun gross geschriebene) A, und klicken Sie auf *Format/-Selection/Invert case.* Was hat das bewirkt?

Wichtig: R unterscheidet zwischen Gross- und Kleinschreibung! Das heisst, dass Sie beispielsweise zwei unterschiedliche Objekte mit den Namen *a* und *A* definieren können.

Zeilen-, Spaltenmodus

- Schreiben Sie Folgendes in das Eingabefenster (es wird empfohlen, dass Sie die Texte des Leitprogramms jeweils kopieren und in TINN-R einfügen):

```
a <- 5
b <- 6
c <- 7
1
2
3
```

  Damit haben Sie drei Objekte a, b und c definiert sowie drei Zahlen 1, 2 und 3 geschrieben.
- Klicken Sie auf *Format/Selection Mode/Column*.
- Wählen Sie die Zahlen 1, 2 und 3 aus.

```
a <- 5
b <- 6
c <- 7
1
2
3
```

- Kopieren Sie diese drei Zahlen (mit den Tasten *ctrl* und *c*).
- Wählen Sie die Zahlen 5, 6 und 7 aus.

```
a <- 5
b <- 6
c <- 7
1
2
3
```

- Fügen Sie die Zahlen 1, 2 und 3 ein (Tasten *ctrl* und *v*). Dieser so genannte Spaltenmodus kann beim Programmieren mit R hilfreich sein!
- Wechseln Sie nun in den normalen Modus zurück, indem Sie auf *Format/Selection Mode/Normal* klicken.

Texteinzug

- Wählen Sie die ersten drei Zeilen aus.

```
a <- 5
b <- 6
c <- 7
```

- Klicken Sie auf *Format/Block/Indent*, und beobachten Sie, was passiert. Texteinzüge verbessern die Leserlichkeit eines langen Programms sehr!
- Machen Sie mit *Format/Block/Unindent* den Texteinzug rückgängig.

Auskommentieren

- Wählen Sie erneut die ersten drei Zeilen aus.
- Klicken Sie auf *Format/Block/Comment*.
- Bei der ersten Anwendung, müssen Sie das Kommentar-Zeichen # als Kommentarbeginn wählen (zweit-unterste Zeile im aufgegangenen Fenster).
- Das #-Zeichen bewirkt, dass die drei Zeilen von R nur als Kommentar gelesen werden, was Auskommentieren bezeichnet wird. Auskommentierter Text wir in TINN-R mit grüner Farbe dargestellt.
- Machen Sie das Auskommentieren mit *Format/Block/Uncomment first occurence* den Texteinzug rückgängig.

Bemerkung: Manchmal lohnt es sich, grössere Codebereiche auszukommentieren, welche ihrerseits aus Code und Kommentaren bestehen. Beim Rückgängig machen des Auskommentierens ist es hilfreich, dass TINN-R zwischen *Format/Block/Uncomment first occurence* (das erste # Zeichen pro Zeile wird entfernt) und *Format/Block/Uncomment all occurence* (alle # Zeichen der angewählten Zeilen werden entfernt) unterscheidet.

Klammern

- Schreiben Sie die folgenden Klammern in das Eingabefenster: (( )
  Dabei gibt R an, welche Klammern zusammen gehören.
- Fügen Sie eine schliessende Klammer hinzu. Was stellen Sie fest? Löschen Sie danach alle Klammern wieder.

Editor Eigenschaften

Mit *Options/Main/Editor* können Sie Eigenschaften des Editor Fensters wie beispielsweise dessen Grösse anpassen.

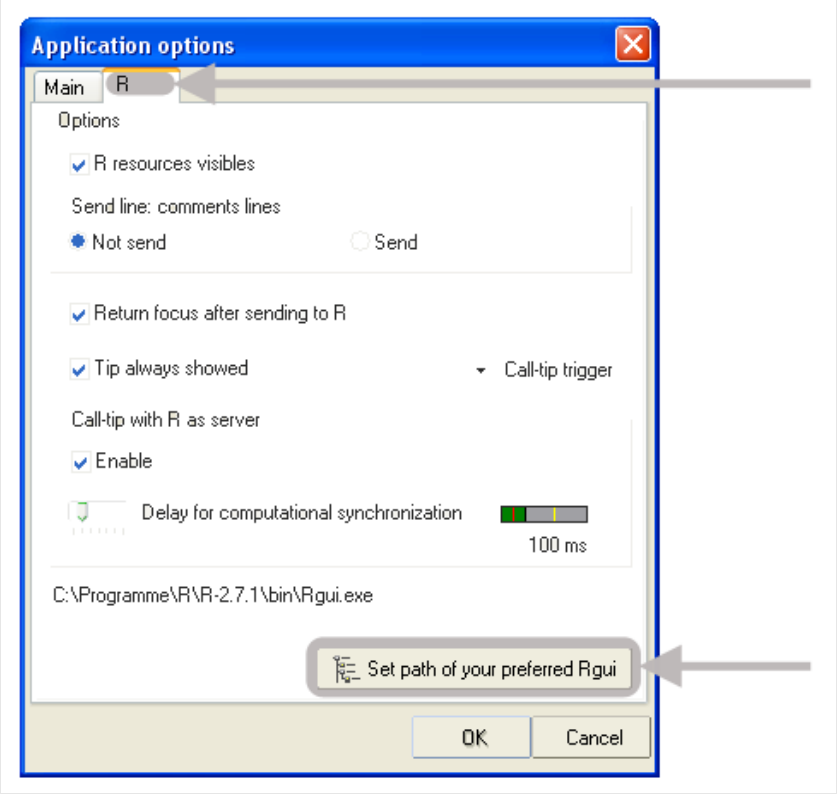
Ansicht

Unter *View* können Sie die Ansicht des TINN-R Fensters verändern. Besonders bei einem langen R Code sind Zeilennummern hilfreich.

- Klicken Sie auf *View/Line numbers* um Zeilennummern anzeigen zu lassen.
- Es ist Ihnen überlassen ob Sie die Zeilennummern wieder deaktivieren.

Pfad zu *Rgui.exe*

- Klicken Sie auf *Options/Main/Application*
- Wechseln Sie in das Tabellenblatt *R*.



- Klicken Sie auf *Set path to your preferred Rgui*.
- Suchen Sie die *Rgui.exe*-Datei. Sie befindet sich oft unter *C:\Programme\R\R-2.11.1\bin*. Wählen Sie die *Rgui.exe*-Datei an, und klicken Sie auf *Open*, und danach auf *OK*. Damit kann R nun mit TINN-R gestartet werden. Dieser Pfad zu Rgui.exe muss nur einmal, und nicht bei jedem Aufstarten von R angegeben werden.

Übungsaufgabe 2.3

1) Definieren Sie drei Objekte A, B und C, denen wie den Objekten a, b und c die Werte 5, 6 und 7 zugeordnet werden. Tipp: Verwenden Sie den bereits in TINN-R eingegebenen Text sowie den Spaltenmodus.

Lösung zu Übungsaufgabe 2.3

1) Spaltenmodus: *Format/Selection Mode/Column*

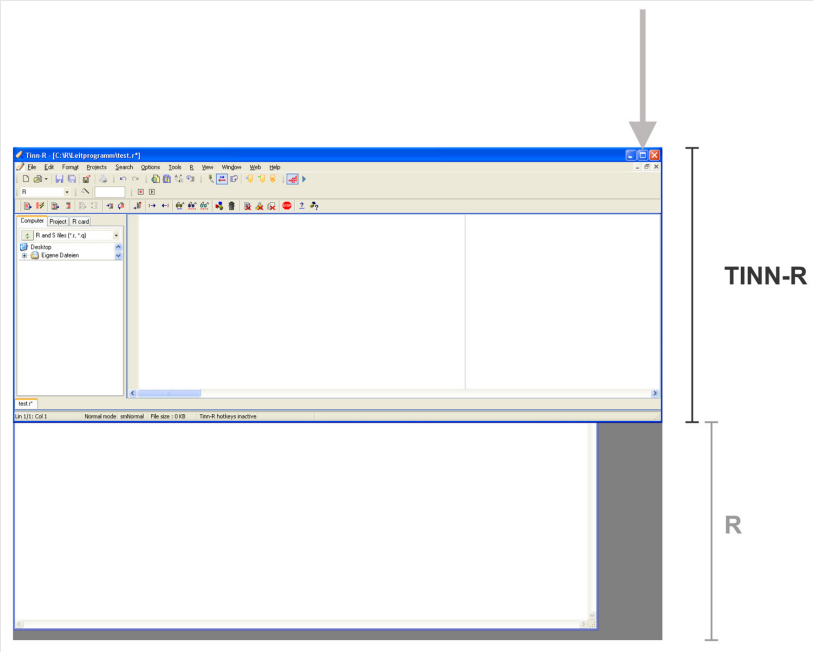
a, b und c auswählen

```
a <- 5
b <- 6
c <- 7
```

Format/Selection/Invert case

Interaktion von TINN-R zu R

- Klicken Sie auf *R/Rgui/Initiate preferred* um R zu starten.
- Es lohnt sich das Fenster von TINN-R zu verkleinern. Die Ausmasse von TINN-R werden beim nächsten Aufstarten wieder übernommen. Die folgende Anordnung ermöglicht Ihnen sowohl die Eingaben in TINN-R als auch die Resultate in R zu sehen.



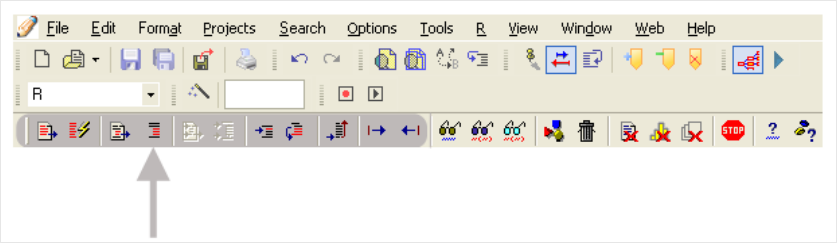
- Schreiben Sie nun Folgendes in das Eingabefenster von TINN-R, und wählen Sie nur die zweite Zeile an:

  library()

  require(boot)
- Klicken Sie auf *R/Send to R/Selection*. Was ist passiert?
- Klicken Sie auf *R/Send to R/All* um den ganzen Inhalt des Eingabefensters an R zu senden.

Shortcut-Buttons

- Sie können für die Interaktion von TINN-R mit R auch die Shortcut-Buttons verwenden. Es sind die gleichen Symbole wie unter *R/Send to R/*



2.6. R und TINN-R schliessen

„Housekeeping“

Bevor Sie R und TINN-R schliessen, muss noch aufgeräumt werden.

- Geben Sie den Befehl *objects()* ein, um zu überprüfen, welche Objekte Sie bei der aktuellen R-Session verwendet haben.
- Mit der Funktion *rm()* können Sie bestimmte Objekte entfernen (Englisch: entfernen = remove). Geben Sie *rm(a,b)* in TINN-R ein, und senden Sie diesen Befehl an R, um die Objekte *a* und *b* zu entfernen.
- Klicken Sie in TINN-R unter *R/Controlling R* auf *Remove all objects*. Damit können Sie alle Objekte gleichzeitig löschen.

R und TINN-R schliessen

- Tippen Sie *q()* in das Eingabefenster von TINN-R, und senden Sie diesen Befehl an R. Beantworten Sie die Frage *Save workspace image?* mit *No*. Damit werden die Änderungen Ihres Arbeitsplatzes nicht gespeichert, und es werden bei der nächsten R-Session nicht ungewollt Objekte der jetzigen Session verwendet.

Hinweis: Falls Sie bestimmte Objekte an Ihrem Arbeitsplatz behalten wollen, speichern Sie den Arbeitsplatz entweder mit *save.image()*, oder beantworten die Frage *Save workspace image?* mit *Yes.* Das sollten Sie aber nur tun, nachdem Sie mit *objects()* überprüft haben, welche Objekte sich zu diesem Zeitpunkt auf Ihrem Arbeitsplatz befinden. R legt diese in der Arbeitsplatz-Datei (*.Rdata*) ab. Zudem wird eine *.Rhistory*-Datei mit den letzten Befehlen erstellt.

- Speichern Sie die Änderungen in der Datei *test.r* (entweder unter *File/Save* oder mit dem entsprechenden Shortcut-Button).
- Schliessen Sie TINN-R indem Sie auf die obere rechte Ecke klicken.

2.7. Merkpunkte

Sie haben nun die ersten Schritte mit R und TINN-R unternommen. Repetieren Sie die wichtigsten Merkpunkte dieses Kapitels:

R-Homepage

- R-Homepage: www.r-project.org

Arbeitsverzeichnis

- *getwd()*: Befehl, um das aktuelle Arbeitsverzeichnis zu sehen.
- *setwd("C:/R")*: Befehl, um das Arbeitsverzeichnis zu einem Ordner R zu wechseln, welcher sich auf dem C-Laufwerk befindet.

Pakete installieren und laden

- *library()*: Befehl, um eine Liste aller auf dem Computer vorhander Pakete zu erhalten.
- *install.packages("RSAGA"):* Installieren des Pakets *RSAGA*.
- *require(stats)*: Befehl, um das Paket *stats* zu laden.

Hilfestellung

- R-Homepage: Häufige Fragen, Dokumente, Such-Funktion.
- CRAN-Webseite: Pakete.
- Foren, RNews, Mailing-Listen.
- Menuleiste in R: Offline-Hilfestellungen.
- Befehle in R: *?*-Funktion, *example*-Funktion.

TINN-R

- R von TINN-R aus starten: *R/Rgui/Initiate preferred*.
- Wahl des Spaltenmodus: *Format/Selection Mode/Column*.
- Texteinzug: *Format/Block/Indent*.
- Auskommentieren: *Format/Block/Comment*.
- Zeilennummern: *View/Line numbers*.
- Auswahl an R senden: .
- Alles an R senden: .

R beenden

- Liste aller Objekte einer bestimmten Umgebung: *objects()*.
- Alle Objekte löschen: *R/Controlling R/Remove all objects*.
- *rm()*: Objekte löschen
- *q()*: R beenden.

2.8. Lernkontrolle

Die Lernkontrolle hilft Ihnen festzustellen, ob Sie die Inhalte des zweiten Kapitels verstanden haben, und ob Sie für den Kapiteltest bereit sind. Lösungen zu den Aufgaben finden Sie auf der nächsten Seite. Verwenden Sie diese aber nur zum Korrigieren Ihrer Resultate. Schauen Sie bei Problemen nicht zu schnell bei den Lösungen nach. Falls Ihnen eine Aufgabe Schwierigkeiten bereitet, wird empfohlen, dass Sie den entsprechenden Abschnitt nochmals anschauen.

Fragen

1) Öffnen Sie R und TINN-R. Welche Befehle werden benutzt, um a) das Arbeitsverzeichnis zu *C:/Programme* zu wechseln, b) zu überprüfen, ob das aktuelle Arbeitsverzeichnis jetzt tatsächlich *C:/Programme* ist, und c) in das Arbeitsverzeichnis *C:/R* zu wechseln. Achtung: Achten Sie darauf, ob die Befehle mit oder ohne Anführungszeichen, und mit einem Back-Slash (\) oder Forward-Slash (/) geschrieben werden.
2) Welchen Befehl verwenden Sie, um a) heraus zu finden, ob das Paket *DAAG* bereits auf Ihrem Computer vorhanden ist, b) das Paket *DAAG* (falls nicht vorhanden) zu installieren, und c) das Paket *DAAG* zu laden.
3) Enthält die Dokumentation "Fitting Distributions with R" von Vito Ricci auch ein Kapitel über grafische Darstellungen? Tipp: Das Dokument ist in englischer Sprache, und hat weniger als hundert Seiten.
4) Finden Sie die Dokumentation (pdf-Datei) des *trip* Pakets. Beschreiben Sie kurz wozu dieses Paket verwendet werden kann.
5) Nennen Sie zwei R-Befehle mit denen Sie etwas über die *match*-Funktion erfahren können.
6) Beschreiben Sie, über welche Menüpunkte Sie in TINN-R a) in den Spaltenmodus, b) zurück zum normalen Modus kommen, und c) wie Sie Texteinzüge erzeugen.
7) Wie entfernen Sie mit TINN-R alle Objekte?

Die Lösungen zur Lernkontrolle finden Sie im Kapitel 8.

2.9. Kapiteltest

Falls Sie sich bei den vorher behandelten Themen sicher fühlen, melden Sie sich bei der Lehrperson zum Kapiteltest. Den Kapiteltest absolvieren Sie an einem dafür vorgesehenen Computer. Dabei dürfen Sie nur das Merkblatt, nicht aber das Leitprogramm verwenden. Das Merkblatt ist auf den Kapiteltest-Computern auch in digitaler Form vorhanden. Selbstverständlich dürfen Sie während dem Kapiteltest auch die R-Online-Hilfe verwenden. Hingegen ist der Gebrauch von Kommunikationsprogrammen sowie des Leitprogramms untersagt. Ihre Lösungen drucken Sie anschliessend aus, und lassen Sie von der Lehrperson korrigieren. Für den Kapiteltest stehen Ihnen maximal 20 Minuten zur Verfügung.

2.10. Zusatzaufgaben

Falls Sie dieses Kapitel wesentlich früher als andere Studierende durch gearbeitet haben, können Sie sich mit den folgenden Zusatzaufgaben befassen, bevor Sie sich dem Kapitel 3 zuwenden. Oder nutzen Sie die Zeit, in der die Lehrperson Ihren Kapiteltest korrigiert, um Zusatzaufgaben zu bearbeiten!

1) Eignen Sie sich möglichst viele Kurzbefehle in TINN-R an (beispielsweise *shift+ctrl+c* um in den Spaltenmodus zu gelangen, oder *alt + c* zum Auskommetieren).
2) Finden Sie auf der R-Homepage eine Dokumentation, die eventuell später für Sie hilfreich sein könnte.
3) Finden Sie heraus, was bei der *install.packages*-Funktion das Argument *repos* bewirkt.

3. Datenformate

Übersicht

In diesem Kapitel lernen Sie, in welcher Form Daten in R gespeichert werden. In Kapitel 2 wurde aufgezeigt, dass Daten bestimmten Objekten zugewiesen werden. Je nach Form der Daten oder Zweck der Analysen werden dabei verschiedene Datentypen und Datenstrukturen verwendet. Sinnvolle Datenstrukturen erleichtern Ihre Analysen in R beträchtlich.

Lernziele

Nachdem Sie dieses Kapitel bearbeitet haben,

- können Sie anhand von Beispielen drei grundlegende Datentypen sowie vier Datenstrukturen erklären (K2).
- können Sie in eigenen Worten beschreiben, was passiert, wenn arithmetische Operatoren auf Matrizen angewendet werden (K4).
- sind Sie in der Lage, Daten in R zu importieren und exportieren (K3).

3.1. R als Taschenrechner

Schon bald werden Sie mit R ausführliche Analysen durchführen. Jedoch verfügt R auch über die einfachen Funktionen eines Taschenrechners. In R werden die folgenden Operatoren verwendet:

Operatoren

Symbol in R	*Operation*
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
^	Potenzieren

- Starten Sie TINN-R und R wie Sie es in Kapitel 2 gelernt haben.
- Geben Sie Folgendes ein:

```
3 * 4 - 2
12 / 6
2 ^ 4
```

In R gelten die normalen Rechenregeln der Mathematik, so dass Potenzieren vor Multiplizieren/Dividieren sowie wiederum vor Addieren/Subtrahieren ausgeführt wird. Ansonsten müssen Sie Klammern setzen.

- Geben Sie Folgendes ein:

```
3 * 4 – 2
3 * (4 – 2)
```

Übersichtlichkeit

Damit der programmierte Text übersichtlich bleibt, werden oft Leerschläge verwendet. Beispielsweise wird nicht 3*4-2, sondern 3 * 4 - 2 geschrieben. Es wird ebenfalls empfohlen, vor und nach dem Zuweisungszeichen Leerschläge zu verwenden (a <- 5 anstatt a<-5).

Übungsaufgabe 3.1

1) Berechnen Sie in R $\frac{(8 \cdot 14)^3}{256}$

2) Und berechnen Sie $\frac{7 \cdot (181 - 37)^{0.5}}{6}$

Lösung zu Übungsaufgabe 3.1

1) Eingabe: (8 * 14) ^3 / 256
Resultat: 5488

2) Eingabe: (7 * (181 – 37) ^ 0.5) / 6
Bemerkung: Die äussere Klammer wäre nicht notwendig, verbessert aber die Leserlichkeit.
Resultat: 14

Ausblick

Im Kapitel 3 werden Sie noch weitere Funktionen wie beispielsweise die Wurzelfunktion kennen lernen. Damit Sie mit R Umweltdaten analysieren können, reichen zusätzliche Funktionen allein noch nicht aus. Sie müssen auch die von R verwendeten Formate kennen. Das lernen Sie in den folgenden Abschnitten über Objekte, Datentypen und Datenstrukturen.

3.2. Objekte

Objekte in R

Grundsätzlich werden in R alle Daten und Funktionen als Objekte bezeichnet! Im folgenden Kapitel definieren Sie Objekte, indem Sie ihnen Werte oder Daten zuordnen.

Objektnamen

Die Namen der Objekte müssen immer mit einem Buchstaben beginnen, dürfen aber Zahlen und die gebräuchlichsten Sonderzeichen beinhalten.

Objekte definieren

In Kapitel 2 haben Sie folgende Eingabe in R gemacht:

```
a <- 5
```

Dadurch haben Sie ein Objekt *a* definiert, indem Sie ihm den Wert 5 zugeordnet haben.

Bemerkung: In den folgenden Kapiteln wird davon ausgegangen, dass Sie die Grundlagen von TINN-R und R kennen. Unter „Befehl eingeben“ ist gemeint, dass Sie einen Befehl in TINN-R eingeben und diesen in R laufen lassen.

- Geben Sie Folgendes ein:

```
a <- 5
a <- 7
A <- 1
a
A
```

- Sie können feststellen, dass dem Objekt *a* zuerst der Wert 5 zugeordnet wird. Dieser wird aber anschliessend mit dem Wert 7 überschrieben. Hingegen wird das Objekt *a* nicht überschrieben, wenn ein Objekt *A* definiert wird. Wie in Kapitel 2 bereits erwähnt, unterscheidet R Gross- und Kleinschreibung.

3.3. Datentypen und Speichermodus

Atomare Datentypen

In R gibt es fünf grundlegende, so genannnte atomare Datentypen:

Rang	*Beschreibung des Datentyps*	*Bezeichnung des Datentyps in R*	*Beispiele*
-	Leere Menge	NULL	NULL
1	Logische Werte	logical	TRUE, FALSE
2	Ganze und reelle Zahlen	numeric	5.79
3	Komplexe Zahlen	complex	4.5+8i
4	Buchstaben und Zeichen	character	"Wasser"

Die Datentypen sind hierarchisch gegliedert. Das bedeutet, dass Datentypen eines niedrigen Rangs (beispielsweise logische Werte) durch einen Datentyp höheren Rangs (beispielsweise Buchstaben) ausgedrückt werden können, nicht aber umgekehrt. Die Leere Menge ist insofern ein Spezialfall, als dass sie von R ausgegeben wird, falls eine Berechnung kein Ergebnis aufweist. Der Datentyp *complex* wird in diesem Leitprogramm nicht behandelt.

Datentyp testen

Um zu erfahren, welchen Datentyp ein bestimmtes Objekt hat, verwenden Sie die *mode*-Funktion. Um auf einen bestimmten Datentyp zu testen, wird *.is* dem Datentyp-Namen vorangestellt. Beispielsweise wird *is.numeric(a)* verwendet, um zu testen, ob das Objekt *a* vom Datentyp *numeric* ist.

- Überprüfen Sie den Datentyp des Objekts *a*:

  ```
  mode(a)
  ```

- Überprüfen Sie, ob der Datentyp des Objekts *a* vom Datentyp *character* ist:

  ```
  is.character(a)
  ```

- Weisen Sie das Resultat der vorherigen Überprüfung einem neuen Objekt *b* zu, und betrachten Sie das Objekt *b*.

  ```
  b <- is.character(a)
  b
  ```

- Was vermuten Sie, welchen Datentyp das Objekt b hat? Überprüfen Sie dies!

  ```
  mode(b)
  ```

  Das Objekt *b* ist vom Datentyp *logical*.

Datentyp umwandeln

Der Datentyp kann mit einer *as.*-Funktion (beispielsweise *as.character*) umgewandelt werden. Denken Sie daran, dass Datentypen niedrigen Rangs in Datentypen höheren Rangs umgewandelt werden können, aber nicht umgekehrt.

- Wandeln Sie das Objekt *b*, welches momentan vom Datentyp *logical* ist, in den Datentyp *numeric* um:

  ```
  b <- as.numeric(b)
  b
  ```

Wenn der Datentyp *logical* in *numeric* umgewandelt wird, verwendet R für *TRUE* (Deutsch: wahr) die Zahl 1, für *FALSE* (Deutsch: falsch) die Zahl 0. Sie können anstatt *TRUE* und *FALSE* auch die Abkürzungen *T* und *F* benützen.

- Versuchen Sie, das Objekt *b*, welches nun vom Datentyp *numeric* ist, in den Datentyp *logical* umzuwandeln:

  ```
  b <- as.logical(b)
  b
  ```

Hierbei besteht die Ausnahme, dass Objekte höheren Rangs (numerische Werte) in Objekte niedrigeren Rangs (logischer Datentyp) umgewandelt werden können. Werten gleich Null wird *FALSE* zugewiesen, allen anderen *TRUE*.

Datentyp Faktor

Es gibt noch einen weiteren Datentyp, die Faktoren. Er kann jedoch auf den Datentyp *numeric* zurückgeführt werden, und gehört daher nicht zu den atomaren Datentypen. Mit der Funktion *factor()* wird der Datentyp einem Objekt zugeordnet.

- Erstellen Sie das folgende Objekt a vom Datentyp Faktor:

  ```
  a <- factor( c("Kupfer", "Mangan", "Kupfer","Kupfer") )
  a
  ```

- Überprüfen Sie den atomaren Datentyp von *a*:

  ```
  mode(a)
  ```

  Der atomare Datentyp von *a* ist *numeric*.

Speichermodus

Der Speichermodus legt fest, wieviel Speicherplatz R intern verwendet, um ein Objekt zu speichern. Für Zahlen bestehen die Speichermodi *integer* (für ganze Zahlen) sowie *double* für reelle Zahlen. Mit der Funktion *typeof()* kann der Speichermodus ermittelt, mit *as.integer(),* respektive *as.double()* der Speichermodus festgelegt werden. Der Speichermodus ist nicht mit den Datentypen zu verwechseln!

Übungsaufgabe 3.2

1) Finden Sie heraus, welchen Dateityp die folgenden drei Objekte aufweisen:

   ```
   a <- 2.71828
   b <- a < 2
   c <- "Wasser"
   ```

2) Mit welchem Befehl können Sie den Datentyp von Objekt *b* in *numeric* umwandeln? Welches Ergebnis erwarten Sie?

Lösungen zu Übungsaufgabe 3.2

1) Es wird die *mode()* Funktion verwendet, beispielsweise *mode(a)*.
   Die Objekte *a*, *b* und *c* haben die Datentypen *numeric*, *logical* und *character*.
2) Umwandlung: `b <- as.numeric(b)`. Es wird die Zahl 0 als Ergebnis erwartet, weil TRUE dem Wert 1, und in diesem Beispiel FALSE dem Wert 0 entspricht.

3.4. Datenstrukturen

Bis jetzt haben Sie verschiedenen Datentypen kennen gelernt, welche das Format der Objektinhalte beschreiben. Daten können aber verschieden angeordnet werden. Die Anordnung der Daten wird durch Datenstrukturen vorgegeben. In R werden fünf Datenstrukturen unterschieden:

Datenstruktur	*Anzahl Dimensionen*	*Anzahl Datentypen pro Objekt*	*Datensätze gleicher Länge?*
Vektoren	1	1	-
Matrizen	2	1	-
Arrays	> 2	1	-
Dataframes	≥ 1	≥ 1	Ja
Listen	≥ 1	≥ 1	Nein

In den folgenden Abschnitten werden Sie die einzelnen Datenstrukturen anhand von Beispielen kennen lernen. Dataframes sind die wichtigste Datenstruktur in R. Aus diesem Grund werden Dataframes in diesem Leitprogramm ausführlicher als andere Datenstrukturen behandelt. Zudem sind Arrays für Umweltanalysen wenig relevant. Deshalb wird in diesem Leitprogramm nur kurz auf Arrays eingegangen.

3.4.1 Vektoren

Vektoren erstellen

Vektoren weisen eine eindimensionale Datenstruktur auf. Beispielsweise kann ein Vektor aus einer Zahlenreihe bestehen. Vektoren können von jedem Datentyp (auch *logical* oder *character*) bestehen. Jedoch darf pro Vektor nur ein bestimmter Datentyp verkommen. Ein Vektor wird am einfachsten mit der *c*-Funktion (Englisch: *concatenate* = verketten) erstellt.

- Erstellen Sie je einen Vektor mit Datentyp *numeric*, *character* und *logical* indem Sie Folgendes eingeben:

```
Abfluss <- c(15.1, 7.8, 5.1, 3.3)
Monat <- c("April", "Mai", "Juni", "Juli")
Hochwasser <- c(T, F, F, F)
```

- Überprüfen Sie für jeden Vektor den Datentyp mit der *mode()* Funktion.

Es kann hilfreich sein, Vektoren mit der *rep()* oder *seq()* Funktion zu erstellen, falls Sie Replikationen (=Wiederholungen) respektive Sequenzen benötigen.

Replikationen

- Erstellen Sie je einen Vektor vom Datentyp *logical*, *numeric* und *character* indem Sie Folgendes eingeben:

```
a <- rep(2, 5)
b <- rep("Sonntag", 5)
c <- rep(TRUE, 3)
```

Sequenzen

Geben Sie die untenstehenden Befehle ein, um Vektoren mit Sequenzen zu erstellen.

- Sequenzen können nur für Vektoren des Datentyps *numeric* verwendet werden. Eine Sequenz kann durch Anfangs- und Endzahl definiert werden. Die Schrittgrösse ist – falls nicht anders vorgegeben – gleich eins.

```
d <- seq(6, 11)
```

- Für Schrittgrösse eins kann ebenfalls der folgende Befehl mit gleicher Bedeutung verwendet werden:

```
d <- 6 : 11
```

- Sie können die Schrittgrösse (*by*) oder die Länge (*length*) wählen:

```
e <- seq(6, 24, by = 3)
f <- seq(6, 24, length = 10)
```

- Sequenzen: Die Länge eines Vektors kann (bei gegebener Schrittgrösse oder Endzahl) der Länge eines bereits vorhandenen Vektors angepasst werden.

```
g <- seq(66, by=4, along = b)
i <- seq(66, 78, along = b)
```

- Mit `j <- rev(i)` werden die Elemente des Vektors i in umgekehrter Reihenfolge angeordnet.

Vektoren sind eine zentrale Datenstruktur in R. Die nachfolgenden Datenstrukturen bauen auf Vektoren auf. Im Prinzip werden in R einzelne Werte wie Vektoren der Länge eins behandelt. Im nächsten Abschnitt lernen Sie, wie mit Vektoren des Datentyps *numeric* gerechnet wird.

Rechnen mit Vektoren

Beim Rechnen mit Vektoren werden die Elemente der Vektoren einzeln berechnet. Das wird mit den folgenden Beispielen veranschaulicht.

- Betrachten Sie den Vektor d, und führen Sie die folgenden Operationen aus:

```
d
d1 <- d + d
d2 <- 3 * d
d3 <- d1 * d2
```

Bemerkung: Wenn bei einer Berechnung mehrere Vektoren verwendet werden, müssen diese gleich viele Elemente haben.

Übungsaufgabe 3.3

1) Überlegen Sie sich, was die unten stehenden Eingaben bewirken. Falls Sie zu einer Lösung gekommen sind, geben Sie die Befehle in R ein, um Ihre Überlegungen zu überprüfen.

```
test1 <- c(1, 8, 17, 25)
test2 <- rep(1, 5)
test3 <- seq(4, 10, by = 2)
alle <- c(test1, test3, test2)
```

2) Wie sieht der untenstehende Vektor *test4* aus? Von welchem Datentyp ist er? Überprüfen Sie Ihre Überlegungen mit R.

```
test4 <- c( seq(7,5) , rep("Phosphor", 3) )
```

3) Mit welchem Befehl können Sie (auf einfache Art und Weise) den folgenden Vektor *test5* erstellen?

```
9 7 5 3 9 7 5 3 9 7 5 3 9 7 5 3
```

Lösungen zu Übungsaufgabe 3.3

1) *test1* ist ein Vektor zu dem die vier Zahlen 1, 8, 17 und 25 verkettet (Englisch *concatenate*) wurden.

```
1  8  17  25
```

test2 ist ein Vektor, bestehend aus der fünffachen Replikation der Zahl 1.

```
1  1  1  1  1
```

test3 ist ein Vektor, der aus einer Sequenz von 4 bis 10 mit einer Schrittgrösse von 2 besteht.

```
4  6  8  10
```

alle ist ein Vektor bestehend aus der Verkettung der Vektoren *test1*, *test2* und *test3*.

2) `"7"  "6"  "5"  "Phosphor" "Phosphor" "Phosphor"`

Der Befehl *mode(test4)* ergibt, dass dieser Vektor vom Datentyp character ist. Begründung: Wenn zwei Vektoren unterschiedlichen Datentyps zusammen geführt werden, wird der Datentyp höheren Rangs übernommen.

3) `rep( seq(9, 3, by = -2), 4)`

3.4.2 Matrizen

Matrizen erstellen

Matrizen sind eine Erweiterung von Vektoren: Eine Matrix besteht nicht nur aus einer Zeile, sondern aus Zeilen und Spalten. Matrizen können auf verschiedene Arten erstellt werden. Beispielsweise können Vektoren mit der Funktion *matrix ()* in Matrizen umgewandelt werden. Diese Funktion verfügt über die Argumente *data* (welchen Daten werden verwendet?), *nrow* (Anzahl Zeilen der Matrix), *ncol* (Anzahl Spalten der Matrix) sowie *byrow* (wird die Matrix zeilen- oder spaltenweise aufgebaut). Die Argumentbezeichnungen beziehen sich auf die Englischen Namen von Zeile (*row*) und Spalte (*column*).

- Wandeln Sie den Vektor I in eine Matrix mit fünf Zeilen und zwei Spalten um, indem Sie Folgendes eingeben:

  ```
  I <- seq(4, 22, by = 2)
  J1 <- matrix (data = I, nrow = 5, ncol = 2, byrow = TRUE)
  ```

  Beachten Sie, dass die Elemente des Vektors in der Matrix zeilenweise angeordnet sind!
- Finden Sie heraus, inwiefern sich die *J1* Matrix von der folgenden *J2* Matrix unterscheidet, wenn *byrow = FALSE* gewählt wird:

  ```
  J2 <- matrix(data = I, nrow = 5, ncol = 2, byrow = FALSE)
  ```

Grundsätzlich müssen die Argumentbeschreibungen (*data*, *nrow*, *ncol*, *byrow*) nicht geschrieben werden. Um die Verständlichkeit eines Programmiercodes zu verbessern, wird jedoch empfohlen, die Argumente dennoch hinzuschreiben. Falls *byrow* komplett weggelassen wird, wird dieses Argument gemäss Vorgabe (Englisch *default*) auf *FALSE* besetzt. Die Vorgaben können Sie unter der Funktions-Hilfestellung (*?matrix*) anschauen. Sie sehen die Vorgaben auch, wenn Sie die Funktion in TINN-R bearbeiten. Eine Matrix ist vollständig definiert, wenn die Daten sowie entweder die Anzahl Zeilen oder die Anzahl Spalten vorgegeben werden.

- Geben Sie Folgendes ein:

  ```
  J3 <- matrix(I, nrow= 5, byrow= TRUE)
  ```

  J3 ist die gleiche Matrix wie J1.
- Sie können die Matrix direkt definieren:

  ```
  J4 <- matrix( seq(4, 22, by = 2), nrow= 5, byrow= TRUE)
  ```

  J4 ist die wiederum die gleiche Matrix wie J1.

Zeilen und Spalten anhängen

An bestehende Matrizen können Zeilen oder Spalten angehängt werden. Dazu werden die Funktionen *rbind* (r für *row*; Deutsch: Zeile) und cbind (c für *column*, Deutsch: Spalte) verwendet.

- ```
  K1 <- rbind(J4 , c(24, 26) )
  K2 <- cbind(J4 , rep(0, 5) )
  ```

Datentyp

Wie bereits erwähnt, können Matrizen nur einen Datentyp gleichzeitig enthalten. Falls eine Matrix mit Daten verschiedenen Datentyps erzeugt wird, übernimmt die Matrix den Datentyp mit dem höchsten Rang.

- ```
  L1 <- matrix( c("Ammonium","Nitrat","Nitrit","Phosphat") ,nrow = 2)
  L2 <- cbind(L1 , c(20, 37) )
  ```
- Überprüfen Sie, welche Datentypen die beiden Matritzen *L1* und *L2* haben.

Rechnen mit Matrizen

Da Matrizen eine Erweiterung von Vektoren sind, geschieht das Rechnen mit Matrizen ebenfalls elementweise.

- ```
  M1 <- matrix(seq(3, 25, length = 12), nrow = 3)
  M2 <- 2 * M1 - 1
  ```

Übungsaufgabe 3.4

1) Schreiben Sie auf, wie die Matrix *N1*, welche mit dem untenstehenden Befehl erzeugt wird, aussieht. Kontrollieren Sie anschliessend mit R Ihre Antwort.

   ```
   N1 <- matrix( seq(3, by = 2, length = 9), nrow = 3, byrow = TRUE)
   ```

2) Nennen Sie einen Befehl, mit dem Sie die folgende Matrix *N2* erstellen können:

   ```
   1   1   1   1
   2   2   2   2
   3   3   3   3
   ```

3) Erstellen Sie eine *N3* Matrix indem Sie bei der *N2*-Matrix eine Zeile mit der Ziffer 4 anhängen.
   ```

Lösungen zu Übungsaufgabe 3.4

1) Die Matrix N1 sieht wie folgt aus:

```
3   5   7
9   11  13
15  17  19
```

2) Folgenden Befehle sind möglich:

```
N2 <- matrix( rep(1:3, 4), nrow = 3)
N2 <- matrix( rep(1:3, 4), nrow = 3, byrow = FALSE)
N2 <- matrix( rep( c(1, 2, 3), 4), nrow = 3)
N2 <- matrix( rep( seq(1, 3), 4), nrow = 3)
```

3) Sie können beispielsweise die folgenden Befehle verwenden:

```
N3 <- rbind( N2, rep(4, 4) )
N3 <- rbind( N2, c(4, 4, 4, 4) )
```

3.4.3 Arrays

Sie haben in den vorangehenden Kapiteln erfahren, dass Vektoren eine, und Matrizen zwei Dimensionen (Zeilen und Spalten) aufweisen. Arrays können beliebig viele Dimensionen aufweisen. Arrays werden mit der *array()* Funktion erzeugt. In der Praxis zeigt es sich aber, dass Arrays mit mehr als drei Dimensionen wenig sinnvoll sind, da ihre Struktur nicht mehr durchschaubar ist.

3.4.4 Dataframes

Dataframes erstellen

Dataframes können entweder aus Vektoren oder Matrizen, oder direkt erstellt werden. Im Gegensatz zu den vorher besprochenen Datenstrukturen können verschiedene Spalten eines Dataframes unterschiedliche Datentypen enthalten. Deshalb ist das Dataframe eine sehr nützliche Datenstruktur!

- Geben Sie Folgendes ein, um ein Dataframe mit dem Name *Meteo* und den Spalten *Mittlere Temperatur* und *Hauptwindrichtung* zu erstellen.

```
Meteo <- data.frame ( Mittlere_Temperatur = c (8.4, 9.3, 12.4, 14.1),
                      Hauptwindrichtung = c("W", "WSW", "W", "NW") )
```

- Die Spaltennamen werden mit *names()*, die Zeilennamen mit *row.names()*, benannt oder umbenannt.

```
names(Meteo) <- c("TEMP", "WIND")
row.names(Meteo) <- c("Januar", "Februar", "Maerz", "April")
```

- Die Namen eines Dataframes können mit *dimnames()* betrachtet werden.

```
dimnames(Meteo)
```

Spalten von Dataframes werden auch *Komponenten* genannt. Um Verwechslungen zu vermeiden, sollten Sie keine Komponente gleich wie das Dataframe benennen! Zudem wird empfohlen, dass Sie häufig vorkommende Komponenten-Namen wie beispielsweise *"Datum"* mit einem Dataframe-spezifischen Zusatz (beispielsweise *"Meteo_Datum"*) versehen.

Übungsaufgabe 3.5

1) Erstellen Sie ein Dataframe mit dem Namen *Bach*. Es soll die zwei Spalten *Ba_Abluss* (mit den Werten 2.1, 2.8, 2.9) und *Ba_Wochentag* (Inhalte: Mo, Di, Mi) enthalten.

Lösungen zu Übungsaufgabe 3.5

1)
```
Bach <- data.frame (Ba_Abfluss = c (2.1, 2.8, 2.9),
                    Ba_Wochentag = c("Mo", "Di", "Mi"))
```

3.4.5 Listen

Listen erstellen

Wie Dataframes können auch Listen mehrere Datentypen enthalten, denn Dataframes sind ein Spezialfall der Listen! Während bei Dataframes alle Spalten gleich viele Elemente beinhalten, kann eine Liste aus Komponenten unterschiedlicher Länge bestehen.

- Erstellen Sie eine Liste mit folgendem Befehl:
```
Stationen <- list( Totalisatoren = c("Total1", "Total2", "Total3"),
                   Jahresniederschlag = c(1050, 1102, 1274),
                   Klimatuerme =c ("Klima1", "Klima2"),
                   Globalstrahlung = c(851, 930)  )
```
- Um eine bestimmte Komponente der Liste zu betrachten, kann das $-Zeichen verwendet werden. Geben Sie Folgendes ein:
```
Stationen$Globalstrahlung
```
- Die Namen der Komponenten werden mit *names()* ausgegeben:
```
names(Stationen)
```

3.5. Datenimport und -export

CSV-Dateien

Um möglichst wenig Speicherplatz zu belegen, lohnt es sich, grosse Datensätze als CSV-Datei (Comma Separated Values) abzuspeichern. In CSV-Dateien werden die einzelnen Werte durch Kommas, in Deutschland jedoch durch Strichpunkte, getrennt; eventuelle Formatierungen werden aber nicht übernommen. In diesem Leitprogramm wird als Trennungszeichen ein Strichpunkt und als Dezimalzeichen jeweils in Punkt verwendet (beispielsweise 3.14).

- Öffnen Sie in Excel die Datei *Messgroessen.xls*. Sie ist im Ordner *E:/R/3_Daten* abgelegt, und enthält Messparameter der Station Weil am Rhein in der Nähe von Basel.
- Speichern Sie diese Datei in ihrem Arbeitsordner (*E:/R/temp)* mit *speichern als* und wählen unter Dateityp *.csv* aus. Fragen bezüglich dem Speichern beantworten Sie mit *Ja*, damit die Datei gespeichert wird. Schliessen Sie die Datei nun. Beantworten Sie eventuelle Fragen mit *Nein*, da die Datei bereits gespeichert ist.
- Gehen Sie in den Ordner *E:/R/temp*, und rechtsklicken Sie auf die von Ihnen erstellte Datei *Messgroessen.csv*. Wählen Sie *öffnen/Editor*, damit Sie die erstellte Datei in einem neutralen Format anschauen können. Sie können sehen, dass die einzelnen Werte durch Strichpunkte und Dezimalziffern durch ein Komma getrennt sind. Schliessen Sie die Datei wieder.

Bemerkung: CSV-Dateien werden automatisch von Excel erkannt. Ihr Computer ist so eingestellt, dass beim Doppelklicken auf CSV-Dateien, diese automatisch mit Excel geöffnet werden. Das kann durchaus hilfreich sein; aber um die Struktur der Dateien oder die Trennungszeichen zu betrachten, verwenden Sie hingegen besser einen Editor.

Daten importieren

Die Funktion *read.table()* ist ein allgemeiner Befehl um tabellenförmige Datensätze in R zu importieren. Zusätzlich sind spezialisierte Import-Funktionen vorhanden. Beispielsweise existiert für Komma-getrennte Werte die Funktion *read.csv(),* respektive für Strichpunkt-getrennte Daten *read.csv2()*.

- Legen Sie zuerst ein Import Pfad fest indem Sie folgendes eingeben:
  ```
  MESS_PFAD <- "E:/R/temp/Messgroessen.csv"
  ```
- Geben Sie Folgendes ein:
  ```
  a <- read.table(MESS_PFAD, sep = ";", dec = ",", header = TRUE)
  ```
- Falls Sie "E:/R/temp" als Arbeitsverzeichnis festgelegt haben, können Sie die Datei auch mit dem folgenden Befehl importieren.
  ```
  a <- read.table("Messgroessen.csv", sep = ";", dec = ",", header = TRUE)
  ```

Mit dem Argument *sep* wird das Trennungszeichen zwischen den Werten, mit *dec* das Dezimalzeichen (Beispiel: 0.3 oder 0,3) gewählt. Falls das Argument *header* als *TRUE* gesetzt wird, wird die erste Zeile des Datensatzes als Bezeichnung der jeweiligen Spalten übernommen (Englisch: *header* = Überschrift).

- Analog dazu können Sie den folgenden Befehl verwenden:
  a <- read.csv2(MESS_PFAD, dec = ",", header = TRUE)
  Hier muss bloss das Trennungszeichen nicht mehr spezifiziert werden.
- Schauen Sie sich das Objekt *a* kurz an.

Daten exportieren

Entsprechend dem Daten-Import, können Daten mit *write.table(),* respektive *write.csv()* und *write.csv2()* exportiert werden. Wählen Sie dabei das Argument *row.names* als FALSE, damit die Zeilennamen nicht exportiert werden.

- Wählen Sie den Exportpfad:
  MESS_NEU <- "E:/R/temp/Messgroessen_Neu.csv"
- Exportieren Sie das Objekt *a* indem Sie Folgendes eingeben:
  write.table(a, file = MESS_NEU, sep = ";", dec = ",", row.names= FALSE)
- Das gleiche Resultat wird mit folgender Eingabe erzielt:
  write.csv2(a, file = MESS_NEU, dec = ",", row.names= FALSE)
- Öffnen Sie die neu erstellte Datei mit den exportierten Daten im Ordner *E:/R/temp* einmal mit Excel und einmal mit dem Editor. Schauen Sie sich die Unterschiede kurz an, und schliessen Sie Excel und den Editor anschliessend wieder.

Übungsaufgabe 3.6

1) Unter *E:/R/3_Daten* finden Sie die Datei *Magnesium.xls*. Sie enthält die Magnesiumkonzentration in mg/l des Rheins bei Weil am Rhein im Jahr 2004 (14-tägige Werte; ingesamt 26 Werte, da ein Jahr 52 Wochen hat). Erstellen Sie daraus eine CSV-Datei.
2) Importieren Sie die CSV-Datei in R. Ordnen Sie die Werte einem Objekt *Konz1* zu.
3) Erstellen Sie ein Objekt *Konz2* welches die Magnesium-Konzentration in der Einheit mol/l enthält. Hinweis: Das Molekulargewicht von Magnesium beträgt 24.305 g/mol, was 24305 mg/mol entspricht. Daher müssen Sie *Konz1* durch 24305 dividieren, um die *Konz2* in mol/l zu erhalten.
4) Exportieren Sie Konz2 in eine Datei mit dem Namen *Magnesium_Mol.csv* in den Ordner *E:/R/temp.*

Lösungen zu Übungsaufgabe 3.6

1) Datei öffnen, *speichern unter*, dabei den Dateityp *.csv* wählen. Die Datei schliessen; weitere Fragen mit *Nein* beantworten.

2)
```
KONZ_PFAD <- "E:/R/temp/Magnesium.csv"
Konz1 <- read.table (KONZ_PFAD, sep = ";", dec = ",", header= TRUE)
```

3)
```
Konz2 <- Konz1 / 24305
```

4)
```
KONZ2_PFAD <- "E:/R/temp/Magnesium_Mol.csv"
write.table(Konz2, file = KONZ2_PFAD, sep = ";", dec = ",", row.names = F)
```

3.6. Merkpunkte

Im Kapitel 3 haben Sie wichtige Datenformate von R kennen gelernt. Repetieren Sie nochmals die wichtigsten Merkpunkte dieses Kapitels:

Operatoren
- Addition +, Subtraktion -, Multiplikation *, Division /, Potenzieren ^.

Objektnamen
- Objektnamen müssen mit einem Buchstaben beginnen.

Datentypen
- Fünf atomare Datentypen: *NULL*, *logical*, *numeric*, *complex* und *character*. Zudem ist *factor* ein nicht atomarer Datentyp.
- Datentyp abfragen: *mode()* oder *is.*-Funktion (beispielsweise *is.character*).
- Der Datentyp umwandeln: *as.*-Funktion (beispielsweise *as.numeric*).

Datenstrukturen
- Fünf Datenstrukturen: Vektoren, Matrizen, Arrays, Dataframes und Listen.
- Vektoren erstellen: Verketten (*c*), Sequenzen (*seq*) oder Replikationen (*rep*).
- Doppelpunkt = Sequenz mit Schrittlänge 1 (beispielsweise 3:7).
- Reihenfolge der Elemente eines Vektors umkehren: *rev()*
- Rechnen mit Vektoren oder Matrizen: Die Operationen werden elementweise durchgeführt.
- Matrizen erstellen: *matrix(data, nrow, ncol, byrow=TRUE).*
- Zeilen resp. Spalten anhängen: *rbind(), cbind().*
- Dataframes: Die Spalten können von unterschiedlichem Datentyp sein.
- Dataframes erstellen: *data.frame();* Spalten respektive Zeilen von umbenennen: *names(), row.names().*
- Listen erstellen: *list();* Komponenten einer Liste abfragen: Mit dem $-Zeichen.
- Komponenten von Listen müssen nicht gleich viele Elemente enthalten!

Daten importieren
- read.table(), read.csv(), read.csv2().
- Trennungs- und Dezimalzeichen spezifizieren: *sep*, *dec.*

Daten exportieren
- *write.table(), write.csv(), write.csv2().*
- Zeilennamen nicht exportieren: *row.names = FALSE.*

3.7. Lernkontrolle

Führen Sie nun die Lernkontrolle durch! Wie bei Kapitel 2 sollten Sie die Lösungen erst am Schluss verwenden, um Ihre Antworten zu korrigieren. Falls Sie Schwierigkeiten haben, schauen Sie den entsprechenden Abschnitt nochmals an.

Fragen

1) Erzeugen Sie den folgenden Vektor *a*:
   10 15 20 25 30 35 10 15 20 25 30 35
2) Geben Sie Folgendes ein:
   `b <- a > 20`
   Von welchem Datentyp ist das Objekt *b*?
3) Wandeln Sie die beiden Vektoren *a* und *b* in Matrizen *A* und *B* mit drei Spalten um. Füllen Sie dabei die Werte nach Zeilen ein.
4) Multiplizieren Sie die beiden Matrizen zu einer Matrix *M*. Beschreiben Sie, was während der Multiplikation mit den beiden Matrizen mit dem Datentyp passiert.
5) Definieren Sie eine Matrix *N*, bei der Sie zur *M*-Matrix eine zusätzliche Spalte mit den Werten 0, 40, 0, 40 anhängen. Welchen Befehl verwenden Sie?
6) Im Ordner *E:/R/3_Daten* finden Sie die Datei *pH.xls*. Erzeugen Sie daraus eine CSV-Datei. Importieren Sie die Daten in R. Hinweise: Stellen Sie sicher, dass Sie das entsprechende Arbeitsverzeichnis verwenden. Schauen Sie sich die CSV-Datei eventuell mit dem Editor an, um die richtigen Trennungs- respektive Dezimalzeichen zu verwenden.
7) Berechnen Sie aus dem pH-Wert die Protonen-Konzentration. Dabei benützen Sie folgende Formel: $Protonen = 10^{-pH}$. Welchen Befehl verwenden Sie?
8) Exportieren Sie die Protonen-Konzentrationen in eine Datei *Protonen.csv* in den Ordner *E:/R/temp*. Exportieren Sie dabei die Zeilennamen nicht.

Die Lösungen zur Lernkontrolle finden Sie im Kapitel 8.

3.8. Kapiteltest

Falls Sie sich bei den Themen dieses Kapitel sicher fühlen, melden Sie sich bei der Lehrperson zum Kapiteltest. Der Test zu diesem Kapitel verläuft gleich wie der letzte Kapiteltest. Sie drucken Ihre Lösungen aus, und lassen Sie von der Lehrperson korrigieren. Für den Kapiteltest stehen Ihnen maximal 20 Minuten zur Verfügung.

3.9. Zusatzaufgaben

Falls Sie mit diesem Kapitel wesentlich früher als andere Studierende fertig sind, können Sie sich mit den folgenden Zusatzaufgaben befassen, bevor Sie mit dem Kapitel 4 beginnen. Oder nutzen Sie die Zeit, in der die Lehrperson Ihren Kapiteltest korrigiert, um Zusatzaufgaben zu bearbeiten!

1) Bisher haben Sie die Argumente *sep* und *dec* der Funktion read.table*()* kennen gelernt. Machen Sie sich mit weiteren Argumenten vertraut.
2) Finden Sie heraus, wann die Funktion *read.fwf()* verwendet wird.
3) Lernen Sie die *append()* Funktion kennen.
4) Falls Sie mehr über Matrizen-Funktionen wissen wollen, schauen Sie in Ligges (2007) die Tabelle 2.4 auf Seite 39 an.
5) Überfliegen Sie in Crawley (2007) das Kapitel 3 (*Data Input*).

4. Datenauswahl

Übersicht

Umweltdatensätze sind oftmals sehr umfangreich. In diesem Kapitel lernen Sie, wie Sie mit R auf effiziente Art und Weise Datensätze ordnen und analysieren können. Sie lernen zudem, wie Sie mit fehlenden Werten in Datensätzen umgehen.

Lernziele

Nachdem Sie dieses Kapitel bearbeitet haben,

- können Sie in eigenen Worten das „Indizieren von Datensätzen“ erklären (K2).
- sind Sie in der Lage, einen beliebigen Datensatz nach bestimmten Argumenten effizient zu analysieren (K3).
- können Sie in Datensätzen fehlende Werte ersetzen oder eliminieren (K3).

4.1. Indizieren

Einführung

In R wird jedem Element eines Objektes ein Index zugewiesen. Beispielsweise wird dem ersten Element eines Vektores der Index 1, dem zweiten Element der Index 2, usw. zugeordnet. Sie werden bald sehen, dass es sehr hilfreich ist, bei Datenabfragen nicht die eigentlichen Werte, sondern die Indices zu verwenden.

Klammern

In R wird der Index eines Objektes in eckigen Klammern geschrieben. Damit unterscheidet sich die Klammersetzung des Indizierens von Funktionen, für welche runde Klammern verwendet werden.

4.1.1 Indizieren von Vektoren

In diesem Abschnitt lernen Sie, wie Vektoren indiziert werden. Weil andere Datenstrukturen auf Vektoren basieren, ist das Indizieren von Matrizen, Dataframes und Listen dem Indizieren von Vektoren ähnlich.

- Verwenden Sie den Vektor Abfluss aus Kapitel 3:

```
Abfluss <- c(15.1, 7.8, 5.1, 3.3)
```

Ein Element auswählen

- Mit dem folgenden Befehl wird das zweite Element des Vektors *Abfluss* ausgegeben:

```
Abfluss[2]
```

Mehrere Elemente auswählen

- Es ist auch möglich mehrere Elemente über den Index auszuwählen.

```
Abfluss[1:3]
Abfluss[c(4,1)]
```

Beachten Sie beim letzten Befehl die Reihenfolge der Elemente: Zuerst wird das vierte, danach das erste Element ausgegeben.

Ausschluss

- Mit dem Minus-Zeichen werden gewisse Elemente ausgeschlossen.
  Abfluss[-3]
- Analog der Auswahl, können auch mehrere Elemente ausgeschlossen werden:
  Abfluss[-c(1,3)]

Neue Objekte

- Weisen Sie die Abfragen einem neuen Objekt zu:
  Fruehling <- Abfluss[-4]

Logische Abfragen

- In Kapitel 3 sind Sie bereits einmal einer logischen Abfrage begegnet. Erstellen Sie ein Objekt einer logischen Abfrage, indem Sie Folgendes eingeben:
  ob_Hochwasser <- Abfluss > 7
  Betrachten Sie das Objekt *ob_Hochwasser*. Finden Sie heraus, von welchem Datentyp das Objekt ist.

Das Objekt *ob_Hochwasser* ist vom Datentyp *logical*. Es hat gleich viele Elemente wie das Objekt *Abfluss*. Die einzelnen Elemente von *ob_Hochwasser* geben an, ob für die entsprechenden Elemente mit gleichem Index des Objekts *Abfluss*, die Bedingung *Abfluss > 7* erfüllt oder nicht erfüllt ist. Fazit: Auch Abfragen werden in R elementweise durchgeführt!

- Verwenden Sie die logische Abfrage, indem Sie Folgendes eingeben:
  Hochwasser1 <- Abfluss[ob_Hochwasser]
  Das Objekt *Hochwasser1* beinhaltet alle Elemente des Objekts *Abfluss,* bei denen das entsprechende Element mit gleichem Index von *ob_Hochwasser* den Wert *TRUE* enthielt.
- Führen Sie die Abfrage direkt durch, indem Sie Folgendes eingeben:
  Hochwasser1 <- Abfluss[Abfluss > 7]

Which-Funktion

Mit der which-Funktion erhalten Sie einen Vektor, der die Indices derjenigen Werte enthält, für die eine Bedingung wahr ist.

- Geben Sie Folgendes ein:
  wo_Hochwasser <- which(Abfluss > 7)
  Betrachten Sie den Vektor *wo_Hochwasser*! Der Vektor *wo_Hochwasser* enthält die Indizes (in diesem Beispiel den Index 1 und Index 2) derjenigen Elemente des Objekts *Abfluss*, bei denen die Bedingung *Abfluss > 7* zutrifft.
- Den Vektor *wo_Hochwasser* können Sie nun als Indizierungsvektor verwenden:
  Hochwasser2 <- Abfluss[wo_Hochwasser]
- Sie können die Abfrage auch direkt durchführen:
  Hochwasser2 <- Abfluss[which(Abfluss > 7)]
  Beachten Sie dabei, dass für Indices eckige, für Funktionen runde Klammern verwendet werden!

Repetieren Sie hier nochmals die Unterschiede zwischen Abfragen mit logischen Objekten und mit der *which*-Funktion:

	Logische Objekte	*Which-Funktion*
Beispiel: Eingabe	ob_Hochwasser <- Abfluss > 7	wo_Hochwasser <- which (Abfluss > 7)
Beispiel: Resultat	TRUE TRUE FALSE FALSE	1 2
Anzahl Elemente	Gleichviele wie das abgefragte Objekt	Soviele, für welche die Abfrage wahr ist
Datentyp	logical	numeric

Werte ersetzen

Der Index eines Vektors kann auch benutzt werden, um bestimmte Werte zu ersetzen.

- Geben Sie Folgendes ein:
  Abfluss [wo_Hochwasser] <- 100

Übungsaufgabe 4.1

1) Erstellen Sie den folgenden Vektor *A*:
   16 14 12 10 8 6 4 2 0
2) Definieren Sie einen Vektor *B*, der das vierte, fünfte und sechste Element von *A* enthält.
3) Definieren Sie einen Vektor *C*, der alle Elemente von *A*, ausser dem dritten enthält.
4) Führen Sie zwei Abfragen durch, je eine mit logischen Objekten und eine mit der *which*-Funktion. Erstellen Sie damit je einen Vektor, der die Elemente von *A* enthält, die kleiner als 10 sind.
5) Ersetzen Sie die ersten drei Werte von *A* mit den Werten 26, 34 und 99.

Lösungen zu Übungsaufgabe 4.1

1) A <- seq(16, 0, by = -2)
2) B <- A[4:6] oder B <- A[c(4, 5, 6)]
3) C <- A[-3]
4) Logische Objekte:
   ob_kleiner <- A < 10
   D1 <- A[ob_kleiner]
   which-Funktion:
   wo_kleiner <- which(A < 10)
   D2 <- A[wo_kleiner]
5) A[1:3] <- c(26, 34, 99)

4.1.2 Indizieren von Matrizen

Matrizen werden ähnlich indiziert wie Vektoren. Weil Matrizen zweidimensionale Objekte sind, wird dementsprechend auch mit zwei Werten indiziert. Beispielsweise bezieht sich der Index A[2, 3] auf das Element der 2. Zeile und 3. Spalte der Matrix A. Der Zeilenindex wird vor dem Spaltenindex genannt, und wird mit einem Komma abgetrennt.

Zweidimensionale Indizierung

- Verwenden Sie die folgende Matrix:
  M <- matrix(seq(5, by = 6, length = 12), nrow = 4, byrow = TRUE)
  Betrachten Sie kurz die Matrix.
- Lassen Sie sich den Wert der vierten Zeile und der zweiten Spalte ausgeben:
  M[4,2]

Ganze Zeile

- Alle Werte der dritten Zeile erhalten Sie, indem Sie die Spaltenbezeichnung nach dem Komma leer lassen:

  ```
  M[3,]
  ```

Ganze Spalte

- Analog dazu erhalten Sie alle Werte der zweiten Spalte, indem Sie die Zeilenbezeichnung vor dem Komma leer lassen:

  ```
  M[,2]
  ```

Abfrage Regeln

Alle Abfrageregeln der Vektoren gelten auch für Matrizen. So können Sie beispielsweise Abfrageresultate neuen Objekten zugeordnen.

- Versuchen Sie zu erraten, wie die folgende Matrix *M1* aussieht. Kontrollieren Sie anschliessend in R Ihre Lösung.

  ```
  M1 <- M[1:2,-2]
  ```

Übungsaufgabe 4.2

1) Verwenden Sie die folgende Matrix:

   ```
   N <- matrix(seq(-50, by = 10, length = 9), nrow = 3, byrow = TRUE)
   ```

   Extrahieren Sie das Element der 2. Zeile und der 1. Spalte.
2) Definieren Sie eine Matrix *N1*, welche aus der 1. und 3. Spalte der Matrix *N* besteht.
3) Erstellen Sie eine Matrix *ob_negativ* vom Datentyp *logical*, welche angibt, welche Elemente von *N* kleiner als Null sind.
4) Welche Datenstruktur hat das folgende Objekt *N2*? Was beinhaltet es?

   ```
   N2 <- N[ ob_negativ ]
   ```
5) Definieren Sie die Matrix *N3*, bei der alle negativen Werte der *N*-Matrix gleich Null gesetzt werden. Tipp: Definieren Sie zuerst die Matrix *N3*, und verändern Sie erst danach deren Werte.

Lösungen zu Übungsaufgabe 4.2

1) `N [2,1]`
2) `N1 <- N[,-2]`
3) `ob_negativ <- N < 0`
4) *N2* ist ein Vektor. Er beinhaltet die negativen Werte der N-Matrix, nach Spalten (!) sortiert.
5) `N3 <- N`

   ```
   N3[ ob_negativ ] <- 0
   ```

4.1.3 Indizieren von Listen

Im Kapitel 3 wurde bereits erwähnt, dass Dataframes ein Spezialfall der Listen sind. Hingegen müssen sämtliche Komponenten (= Spalten der Dataframes) gleichviele Elemente enthalten. Deshalb werden in diesem Abschnitt zuerst der allgemeine Fall (Listen), und im nächsten Unterkapitel der Spezialfall (Dataframes) behandelt.

Komponenten

Grundsätzlich funktioniert das Indizieren von Listen ähnlich wie bei Vektoren und Matrizen. Im Gegensatz zu diesen Datenstrukturen verfügen Listen über mehrere Komponenten. Die einzelnen Komponenten können mit dem $-Zeichen oder doppelten eckigen Klammern [[]] extrahiert werden.

- Verwenden Sie die folgende Liste:

```
Stationen <- list( Totalisatoren = c("Total1", "Total2", "Total3"),
        Jahresniederschlag = c(1050, 1102, 1274),
        Klimatuerme =c ("Klima1", "Klima2"),
        Globalstrahlung = c(851, 930) )
```

- Kontrollieren Sie die Namen der Komponenten.

```
names(Stationen)
```

- Extrahieren Sie die Komponente *Globalstrahlung*:

```
Stationen[[4]]
Stationen[["Globalstrahlung"]]
Stationen$Globalstrahlung
```

Falls die Komponenten mit Namen benannt sind, wird Ihnen empfohlen, die einfachste Variante mit dem $-Zeichen zu verwenden. Ansonsten müssen Sie die Indizierung mit den doppelten eckigen Klammern benützen.

4.1.4 Indizieren von Dataframes

Dataframes sind die wohl wichtigste Datenstruktur in R, unter anderem weil Sie bei Dataframes mit geschicktem Indizieren viele Analysen durchführen können. Das Indizieren von Dataframes lernen Sie in diesem Kapitel!

Dataframes vorbereiten

Wenn Sie mit Dataframes arbeiten, wird folgendes Vorgehen empfohlen: 1) Daten importieren; 2) die Namen der Komponenten betrachten, und notfalls ändern; 3) den Datensatz (bei grossen Datenmengen bloss einige Zeilen davon) kurz betrachten.

- Importieren Sie die Daten:

```
Meteo <- data.frame(TEMP = c (8.4, 9.3, 12.4, 14.1),
                    WIND = c("W", "WSW", "W", "NW"))
```

- Betrachten Sie die Namen der Spalten:

```
names(Meteo)
```

- Ändern Sie eventuell die Namen der Spalten:

```
names(Meteo) <- c( "Temperatur", "Windrichtung" )
```

- Betrachten Sie den gesamten Datensatz, oder Teile des Datensatzes:

```
Meteo
```

Indizieren wie bei Listen

Da Dataframes ein Spezialfall der Listen sind, können Sie Dataframes auf gleiche Art und Weise wie Listen indizieren.

- Geben Sie Folgendes ein, um die drei letzten Temperaturwerte dem Objekt A zuzuordnen:

```
A <- Meteo$Temperatur[2:4]
```

Indizieren wie bei Matrizen

Die Struktur von Dataframes (alle Spalten haben die gleiche Länge) ist ähnlich wie diejenige von Matrizen. Deshalb ist es auch möglich, Dataframes wie Matrizen zu indizieren.

- Geben Sie Folgendes ein, um die Werte der zweiten bis vierten Zeile der ersten Spalte von *Meteo* dem Objekt A zuzuweisen:

```
A <- Meteo[2:4,1]
```

- Die Spalten können auch anhand ihrer Namen ausgewählt werden. Diese sind in Anführungszeichen zu schreiben.

```
A <- Meteo [2:4, "Temperatur"]
```

Bemerkung: Es ist auch möglich, dass Zeilen benannt, und dann anhand ihres Namens ausgewählt werden. Das ist aber nur selten sinnvoll.

Übungsaufgabe 4.3

1) Extrahieren Sie das letzte Element der Spalte *Windrichtung* je einmal mit Listen- und Matrizen-Indizierung.

Lösungen zu Übungsaufgabe 4.3

1)
```
Meteo$Windrichtung[4]
Meteo[4,2]
```

4.2. Datenabfragen

In diesem Kapitel lernen Sie, wie Sie auf dem Indizieren basierend mit logischen Operatoren oder Funktionen Daten analysieren können. Weil diese für verschiedene Datenstrukturen ähnlich sind, werden die Beispiele nicht mehr für alle Strukturen, sondern bloss für Dataframes aufgezeigt.

Bei Datenabfragen in R können die folgenden Operatoren verwendet werden:

Logische Operatoren

Operator	*Beschreibung*
==	gleich
!=	nicht gleich
<	kleiner als
>	grösser als
<=	kleiner gleich
>=	grösser gleich
&	und (elementweise)
|	oder (elementweise)

Die Datenabfragen werden am Beispiel der Messdaten von Weil am Rhein aufgezeigt, die Sie bereits verwendet haben. Bereiten Sie nun das Dataframe vor.

Dataframe vorbereiten

- Importieren Sie die Datei *Messgroessen.csv*:
  ```
  MESS_PFAD <- "E:/R/temp/Messgroessen.csv"
  Mess <- read.table(MESS_PFAD, sep =";", dec =",", header = T)
  ```
- Betrachten Sie die Namen der Spalten:
  ```
  names(Mess)
  ```
- Mit *head()* können Sie die ersten Zeilen des Datensatzes betrachten:
  ```
  head(Mess)
  ```

Einfache Abfragen

- Definieren Sie ein Objekt *Schwermet*, welches die Daten aller Schwermetalle enhält:
  ```
  ob_schwer <- Mess$Bezeichnung1 == "Schwermetalle"
  Schwermet <- Mess[ob_schwer,]
  ```
  Betrachten Sie das Objekt *ob_schwer*. Es ist vom Datentyp *logical*.

Bei dieser Abfrage wurde das Objekt *ob_schwer* mit Datentyp *logical* benutzt. Sie können das gleiche Resultat auch mit der *which*-Funktion erzielen.

- Geben Sie Folgendes ein:

```
wo_schwer <- which(Mess$Bezeichnung1 == "Schwermetalle")
Schwermet <- Mess[wo_schwer,]
```

Der Vorteil der *which-Funktion* ist, dass das resultierende Objekt höchstens gleich gross wie der abgefragte Datensatz, meistens aber wesentlich kleiner ist. Der Nachteil der *which-Funktion* besteht aber darin, dass die Datenstruktur verändert wird, wenn kein Element das Abfragekriterium erfüllt. Dieses Problem können Sie mit der *if*-Funktion beheben, welche Sie erst im Kapitel 5 kennen lernen. Darum werden bis dahin logische Abfragen verwendet.

- Es ist möglich, zusätzlich einzelne Spalten auszuwählen oder auszuschliessen. Geben Sie Folgendes ein:

```
Schwermet1 <- Mess[ob_schwer, -c(6, 7)]
Schwermet2 <- Mess[ob_schwer, c(2, 5, 3, 4)]
```

- Erstellen Sie ein Objekt *Nicht_Anorg* welches alle diejenigen Datensätze beinhaltet, bei denen die *Bezeichnung2* nicht *Anorganische Stoffe* ist.

```
ob_nicht_anorg <- Mess$Bezeichnung2 != "Anorganische Stoffe"
Nicht_Anorg <- Mess[ob_nicht_anorg, ]
```

Verknüpfte Abfragen

Mit den Zeichen | (*oder*) und & (*und*) können Sie Abfragen verknüpfen.

- Definieren Sie ein Objekt *NP*, bei welchem die *Bezeichnung1* entweder *Stickstoff* oder *Phosphor* ist.

```
ob_NP <- (Mess$Bezeichnung1 == "Stickstoff") | (Mess$Bezeichnung1 ==
        "Phosphor")
NP <- Mess [ob_NP,]
```

- Erstellen Sie ein Objekt *ob_NDez*, welches die Datensätze derjenigen Schwermetalle enthält, deren Konzentration im Dezember mindestens so gross ist wie im Juli:

```
ob_NDez <- (Mess$Bezeichnung1 == "Schwermetalle") &
        (Mess$Dezember_2004 >= Mess$Juli_2004)
NDez <- Mess [ob_NDez,]
```

Übungsaufgabe 4.4

1) Erstellen Sie einen Vektor *A* mit folgendem Befehl:

```
A <- seq(37, by = 84, length = 14)
```

Defineren Sie einen Vektor *B*, der alle Elemente von *A* enthält, welche grösser als 500 sind.

2) Die *Datei Phosphor_1996.csv* enthält Daten wie viele Tonnen Phosphor im Jahr in den Rhein eingetragen worden sind. Die Daten sind unterteilt nach Ländern und Eintragsquellen. Importieren Sie diese Daten in R, und bereiten Sie ein entsprechendes Dataframe mit dem Namen *P* vor.
3) Definieren Sie ein Objekt *Punktuell*, das nur die Datensätze der punktuellen Phosphoreinträge (*Eintrag* = *Punktuell*) beinhaltet.
4) Definieren Sie ein Objekt *Nicht_Punkt*, das alle Phosphoreinträge beinhaltet, welche nicht punktuell sind. Das Objekt soll nur die Spalten *Herkunft*, *Eintrag, Schweiz* und *Deutschland* in dieser Reihenfolge beinhalten.
5) Erstellen Sie ein Objekt *Eintrag1*, das diejenigen Datensätze des Objekts *Phosphor* enthält, bei denen der Eintrag entweder in der Schweiz oder in Deutschland mindestens 100 Tonnen Phosphor pro Jahr beträgt. Es sollen nur die Spalten *Herkunft*, *Schweiz* und *Deutschland* in dieser Reihenfolge enthalten sein.
6) Definieren Sie analog zu *Eintrag1* ein Objekt *Eintrag2*, das diejenigen Datensätze des Objekts *Phosphor* enthält, bei denen der Eintrag in der Schweiz und gleichzeitig in Deutschland mindestens 100 Tonnen Phosphor pro Jahr beträgt.

Lösungen zu Übungsaufgabe 4.4

1)
```
ob_groesser <- A > 500
B <- A [ob_groesser]
```
2)
```
P_PFAD <- "E:/R/3_Daten/Phosphor_1996.csv"
P <- read.table(P_PFAD,sep =";", dec=",", header=T)
        names(P)
        head(P)
```
3)
```
ob_punktuell <- P$Eintrag == "Punktuell"
Punktuell <- P[ob_punktuell,]
```
4)
```
ob_nicht_punkt <- P$Eintrag != "Punktuell"
Nicht_Punkt <- P[ob_nicht_punkt, c(1, 6, 2, 3) ]
```
5)
```
ob_Eintrag1 <- (P$Schweiz >= 100) | (P$Deutschland >= 100)
Eintrag1 <- P[ob_Eintrag1, 1:3]
```
6)
```
ob_Eintrag2 <- (P$Schweiz >= 100) & (P$Deutschland >= 100)
Eintrag2 <- P[ob_Eintrag2, 1:3]
```

4.3. Daten sortieren

Über die Indizierung können Datensätze sortiert werden. Dabei spielen die Funktionen *order()* und *rank()* eine besonders wichtige Rolle. Vektoren können Sie mit *sort()* sortieren.

sort-Funktion

- Verwenden Sie das Dataframe *Mess*. Betrachten Sie das Dataframe noch einmal kurz:

  ```
  Mess
  ```

- Sortieren die Elemente der Spalte *Abkuerzung*:

  ```
  Sort1 <- sort(Mess$Abkuerzung)
  ```

Die *sort*-Funktion kann bloss für Vektoren (wie beispielsweise eine einzelne Spalte eines Dataframes) verwendet werden. Das Ergebnis wird direkt ausgegeben; im Gegensatz dazu bestehen die Ausgaben der unten beschriebenen *order*- und *rank*-Funktionen aus Indizes respektive Rängen.

order-Funktion

- Geben Sie Folgendes ein:

  ```
  Reihe1 <- order(Mess$Bezeichnung1)
  ```

Mit dem Objekt *Reihe1* können die Elemente von *Bezeichnung1* aufsteigend geordnet werden. *Reihe1* enthält aber nicht direkt die sortierten Werte, sondern die Reihenfolge der Indizes der Elemente von Bezeichnung1.

Die Folge 1 8 5 usw. bedeutet, dass das kleinste Element (bei Buchstaben wird nach dem Alphabet sortiert) in der ersten Zeile, das zweit-kleinste in der achten Zeile, das dritt-kleinste Element in der fünften Zeile, usw. von Bezeichnung1 zu finden ist.

- Mit Hilfe von *Reihe1* kann das Dataframe *Mess* nun entsprechend sortiert werden:

  ```
  Mess1 <- Mess[Reihe1,]
  ```

Bemerkung: Die *order*-Funktion wird als *stabil* bezeichnet, was bedeutet, dass die Reihenfolge gleicher Elemente nicht verändert wird.

rank-Funktion

Mit der *rank*-Funktion werden Ränge ausgegeben.

- Erstellen Sie das folgenden Objekt:

  ```
  Rang1 <- rank(Mess$Bezeichnung1)
  ```

- Betrachten Sie die Rangzuweisung mit folgendem Dataframe:

  ```
  data.frame(Werte = Mess$Bezeichnung1, Rang1 = Rang1)
  ```

Gemäss den Standardeinstellungen werden gleichen Elementen durchschnittliche Ränge zugeordnet. Beispielsweise wird den vier ersten Elementen gleichen Inhalts je der Rang 2.5 zugeordnet.

Kombinierte Abfragen

Mit der *order*-Funktion können Abfragekriterien kombiniert werden. Bei Zahlen kann dabei das Minus-Zeichen verwendet werden, um Datensätze absteigend zu sortieren. Eine Kombination mit der *rank*-Funktion ermöglicht Absteigend-Sortieren auch für Elemente vom Datentyp *character*. Damit können Sie beliebig viele Sortier-Kriterien verketten.

- Kombinieren Sie eine Abfage:
```
Reihe2 <-order(Mess$Bezeichnung2,Mess$Bezeichnung1,Mess$Abkuerzung)
Mess2 <- Mess[Reihe2,]
```
- Mit dem Minus-Zeichen können numerische Elemente absteigend sortiert werden:
```
Reihe3 <- order(Mess$Bezeichnung2, -Mess$Juli_2004)
Mess3 <- Mess[Reihe3,]
```
- Die Kombination von *order()* und *rank()* ermöglicht Ihnen auch nach nicht-numerischen Elementen absteigend zu sortieren.
```
Reihe4 <- order(Mess$Bezeichnung2, -rank(Mess$Bezeichnung1))
Mess4 <- Mess[Reihe4,]
```

Bemerkung: Wenn *rank()* auf Elemente des Datentyps *character* angewendet wird, entstehen numerische Elemente, welche den Rang der Elemente beinhalten.

Übungsaufgabe 4.5

1) Erstellen Sie ein Objekt *Phosphor_sort* basierend auf dem Phosphor-Dataframe *P*. Sortieren Sie das Dataframe zuerst nach der Spalte *Eintrag* (absteigend), danach nach der Spalte *Herkunft* (aufsteigend).
2) Erzeugen Sie ein Objekt *Phosphor_D*, bei welchem Sie die Phospor-Einträge Deutschlands in absteigender Reihenfolge ordnen. Die Ausgabe soll nur aus den Spalten *Herkunft* und *Deutschland* bestehen.

Lösungen zu Übungsaufgabe 4.5

1)
```
ReiheP <- order(-rank(P$Eintrag), P$Herkunft)
Phosphor_sort <- P[ReiheP,]
```
2)
```
Reihe_D <- order(-P$Deutschland)
Phosphor_D <- P[Reihe_D,c(1, 3)]
```

4.4. Datensätze vereinigen

Merge-Funktion

Mit der *merge()*-Funktion können Sie zwei Datensätze vereinigen.

- Erstellen aus der Datei *Elemente.csv* ein Dataframe *Elemente*:

```
ELEM_PFAD <- "E:/R/3_Daten/Elemente.csv"
Elemente <- read.table(ELEM_PFAD, sep = ";", dec= ",", header = T)
        names(Elemente)
        head(Elemente)
```

- Die beiden Dataframes *Mess* und *Elemente* enthalten je eine Spalte mit Element-Symbolen. Vereinigen Sie die beiden Dataframes anhand der gemeinsamen Spalte. Achtung: Die Spaltennamen sind in den beiden Dataframes unterschiedlich; das können Sie jedoch über die Argumente *by.x* (Spaltenname im ersten Dataframe) und *by.y* (Spaltenname des im zweiten Dataframe) definieren.

```
Mess_neu <- merge(Mess, Elemente, by.x = "Abkuerzung",
        by.y = "Symbol", all.x = F)
```

Die Reihenfolge der Dataframes ist in der merge-Funktion wichtig: Das erste Dataframe wird mit Informationen des zweiten Dataframes ergänzt. Weil im vorherigen Beispiel das Argument *all.x* auf *FALSE* gesetzt wird, enthält das Objekt *Mess_neu* bloss diejenigen Datensätze von *Mess*, die mit Datensätzen aus *Elemente* ergänzt werden konnten.

- Geben Sie Folgendes ein, um alle Datensätze von *Mess* weiter zu verwenden:

```
Mess_neu2 <- merge(Mess, Elemente, by.x = "Abkuerzung",
        by.y = "Symbol", all.x = T)
```

In *Mess_neu2* ist nun beispielsweise auch ein Datensatz für die *Elektrische Leitfähigkeit* enthalten, welchem natürlich kein Molekulargewicht zugeordnet werden kann. Deshalb fehlt dieser Wert. Wie Sie in R mit fehlenden Werten umgehen, lernen Sie im nächsten Unterkapitel.

4.5. Fehlende Werte

In Umweltdatensätzen fehlen oft einzelne Werte, weil beispielsweise Messgeräte defekt waren. Es macht aber Sinn, fehlende Werte in einem Datensatz nicht einfach wegzulassen, sondern diese explizit zu kennzeichnen. In R wird für fehlende Werte die Abkürzung NA (Not Available; Deutsch: Nicht verfügbar) verwendet.

- Erstellen Sie ein Vektor *A*, bei welchem das dritte Element aus einem fehlenden Wert besteht:

  ```
  A <- c(3, 8, NA, 2)
  ```

- Berechnungen mit *NA* ergeben per Definition wieder *NA*! Geben Sie Folgendes ein:

  ```
  5 * A
  A == 8
  ```

Fehlende Werte identifizieren

- Fehlende Werte können nicht mit A == NA ermittelt werden, weil eine Berechnung mit NA ebenfalls NA ergibt. Richtigerweise wird dazu die Funktion *is.na()* verwendet. Probieren Sie die falsche und richtige Methode aus:

  ```
  A == NA          # falsch (# = Kommentar-Zeichen)
  is.na(A)         # richtig
  ```

Fehlende Werte entfernen

Viele Funktionen können nicht ausgeführt werden, wenn bei einem Datensatz Werte fehlen. Zum Entfernen dieser Fehlstellen stehen verschiedene Möglichkeiten zur Verfügung:

- Einige Funktionen enthalten die Option *na.rm* (Englisch: not-available-remove = nicht-verfügbar-entfernen). Diese Option kann beispielsweise bei der Funktion *mean()* zur Berechnung eines Mittelwerts gewählt werden:

  ```
  mean(A)                # Der Mittelwert kann nicht berechnet werden
  mean(A, na.rm = TRUE)  # Der fehlende Wert wird nicht verwendet
  ```

- Fehlende Werte können ausgeschlossen werden:

  ```
  B <- na.omit(A)
  ```

- Mit Hilfe der Funktion *is.na()* können fehlende Werte ersetzt werden.

  ```
  A [is.na(A)] <- 0          # Nullsetzen von NA
  ```

Symbole für fehlende Werte

Oft werden in Datensätzen anstatt NA bestimmt Symbole verwendet, wie Zahlen, die ansonsten nicht im Datensatz vormkommen (Beispiel: -9999). Um in R diese fehlenden Werte korrekt zu berücksichtigen, müssen Sie zuerst identifiziert, und danach mit NA ersetzt werden.

- Erstellen Sie den folgenden Vektor:

  ```
  B <- c(1, -9999, 7, -9999)
  ```

- Identifizieren Sie die fehlenden Werte:

  ```
  ob_NA <- B == -9999
  ```

- Den fehlenden Werten NA zuordnen

  ```
  B[ob_NA] <- NA
  ```

- Damit kann in R korrekt ein Mittelwert über die vorhandenen Daten berechnet werden:

  ```
  mean(B, na.rm = TRUE)
  ```

Fehlende Werte in Dataframes

In Dataframes werden beim Entfernen fehlender Werte ganze Datensätze ausgeschlossen.

- Laden Sie die Datei *Ionen_2004.csv*, welche den mittleren Abfluss (m^3/s) und Ionen-Konzentrationen (mg/l) der Station Weil am Rhein in zweiwöchigen Perioden des Jahres 2004 enthält. In diesem Datensatz fehlen einige Werte.

  ```
  IO_PFAD <- "E:/R/3_Daten/Ionen_2004.csv"
  Ionen <- read.table(IO_PFAD, sep = ";", dec = ",", header = T)
      names(Ionen)
      head(Ionen)
  ```

- Indentifizieren Sie die fehlenden Werte:

  ```
  ob_NA <- is.na(Ionen)
  ```

- Entfernen Sie die fehlenden Werte. Dabei werden ganze Datensätze (beispielsweise die zweite Zeile) ausgeschlossen.

  ```
  Ionen1 <- na.omit(Ionen)
  ```

Nicht definierte Werte

Für nicht definierte Werte wird in R die Abkürzung NaN (Not a Number) verwendet. Zum Identifizieren und Entfernen können Sie die gleichen Funktionen wie für fehlende Werte (NA) gebrauchen.

- Geben Sie Folgendes ein:

  ```
  D <- c ( 2, NA, 4, (-4) ^ 0.5 )
  ```

- Identifzieren Sie nicht definierte oder fehlende Werte:

  ```
  is.na (D)
  ```

- Entfernen Sie nicht definierte oder fehlende Werte:

  ```
  E <- na.omit (D)
  ```

Übungsaufgabe 4.6

1) Erstellen Sie den folgenden Vektor:

```
V <- c(10, NA, 8, 15, NA)
```

Berechnen Sie den Mittelwert von F (ohne fehlende Werte).

2) Identifizieren Sie die fehlenden Elemente.

3) Erstellen Sie einen Vektor *G*, bei welchem die fehlenden Werte gleich -9999 gesetzt sind. Hinweis: Weisen Sie zuerst den Vektor *V* einem identischen Vektor *G* zu.

4) Erstellen Sie einen Vektor *H*, welcher den Vektor *V* ohne fehlende Elemente enthält.

Lösungen zu Übungsaufgabe 4.6

1)
```
V <- c(10, NA, 8, 15, NA)
mean(V, na.rm = TRUE)
```

2)
```
is.na(V)
```

3)
```
G <- V
G[is.na(G)] <- (-9999)
```

4)
```
H <- as.numeric(na.omit(V))
```

4.6. Merkpunkte

Im Kapitel 3 haben Sie wichtige Datenformate von R kennen gelernt. Hier sind die wichtigsten Merkpunkte dieses Kapitels aufgelistet:

Indizieren

- Auswahl; beispielsweise x[c(2,4)].
- Ausschluss; beispielsweise x[-5].
- Über den Index können Werte eines Objekts ersetzt werden.
- Indizieren von Matrizen: [Zeilenindex, Spaltenindex].
- Komponenten von Listen werden mit dem $-Zeichen ausgewählt.
- Indizieren von Dataframes: Wie Matrizen oder wie Listen.

Abfragen

- Logische Abfragen; beispielsweise x > 3.
- Abfragen mit der *which*-Funktion.
- Logische Operatoren: ==, !=, <, >, <=, >=, &, |.

Daten importieren, Dataframes vorbereiten

- Importieren der Daten, beispielsweise mit *read.table()*.
- Trennungszeichen: *sep*, Dezimalzeichen: *dec*.
- Betrachten des Anfangs des Datensatzes: *head()*.

Daten sortieren

- Vektoren sortieren: *sort()*; Ausgabe: Sortierter Vektor.
- *order()*; Ausgabe: Sortierte Indizes; Daten-Reihenfolge verändert.
- *rank ()*; Ausgabe: Rang; Daten-Reihenfolge unverändert.
- Kombinierte Abfragen in einem Dataframe: *order(Spalte1, Spalte2)*.
- Absteigende Anordnung bei Zahlen: *order(-Spalte1)*.
- Absteigende Anordnung bei Buchstaben: *order (-rank(Spalte1))*.

Fehlende Werte

- Identifizieren: *is.na()*.
- Entfernen: Funktions-Argument: *na.rm = TRUE*, *na.omit()*, oder über die Identifikation der fehlenden Werte.

4.7. Lernkontrolle

Falls Sie sich mit dem Lernstoff dieses Kapitels genügend sicher fühlen, beginnen Sie nun mit der Lernkontrolle! Verwenden Sie dabei die Lösungen erst am Schluss, um Ihre Antworten zu korrigieren. Falls Sie Schwierigkeiten haben, befassen Sie sich nochmals mit dem entsprechenden Abschnitt.

Fragen

1) Importieren Sie die Datei *Schwermetalle_2003.csv*, und erstellen Sie daraus ein Dataframe *Schwer*. Bereiten Sie das Dataframe für weitere Analysen vor.
2) Identifizieren Sie mit einer logischen Abfrage fehlende Quecksilber (Hg) Werte. Erstellen Sie ein logisches Objekt *ob_Hg_NA*.
3) Ersetzen Sie eventuell fehlende Quecksilber-Werte mit 0.01.
4) Erzeugen Sie ein Dataframe *Schwer1*, das keine Datensätze mit fehlenden Werten enthält.
5) Definieren Sie aufgrund von *Schwer1* ein Dataframe *Schwer2*, das nur diejenigen Datensätze mit einer Bleikonzentration grösser gleich 0.4 µg/l und einer Kupferkonzentration grösser gleich 1.4 µg/l enthält. *Schwer2* soll nur die Spalten *Periode*, *Pb*, und *Cu* in dieser Reihenfolge enthalten.
6) Erstellen Sie ein Objekt *Schwer3*, bei dem die Datensätze von *Schwer1* zuerst nach *Cu* (absteigend) und danach nach *Q_Mittel* (aufsteigend) geordnet sind.

Die Lösungen zur Lernkontrolle finden Sie im Kapitel 8.

4.8. Kapiteltest

Falls Sie sich bei den vorher behandelten Themen sicher fühlen, melden Sie sich bei der Lehrperson zum Kapiteltest. Der Test zu diesem Kapitel verläuft gleich wie der letzte Kapiteltest. Drucken Sie Ihre Lösungen aus, und lassen Sie von der Lehrperson korrigieren. Für den Kapiteltest stehen Ihnen maximal 20 Minuten zur Verfügung.

4.9. Zusatzaufgaben

Falls Sie dieses Kapitel wesentlich früher als andere Studierende bearbeitet haben, können Sie sich mit den folgenden Zusatzaufgaben befassen, bevor Sie mit dem Kapitel 5 beginnen. Oder nutzen Sie die Zeit, in der die Lehrperson Ihren Kapiteltest korrigiert, um Zusatzaufgaben zu bearbeiten!

1) Bei Vektoren, Matrizen und Dataframes können Abfragen auch mit der *subset*-Funktion gemacht werden. Verwenden Sie das Dataframe *Mess,* das Sie in diesem Kapitel bereits benützt haben, und versuchen Sie den folgenden Befehl zu verstehen:

```
Auswahl <- subset( Mess, Bezeichnung1 %in% c("Schwermetalle", "Ionen")
& (Dezember_2004 > Juli_2004), select = c (1:4) )
```

2) Was bewirkt die *split*-Funktion?
3) Es gibt verschiedene Möglichkeiten, wie bei einer Datenabfrage einzelne Kriterien ausgeschlossen werden können. Lesen Sie dazu in Crawley (2007) die entsprechenden Abschnitte auf Seite 118.
4) Wozu können Sie die Funktionen *attach()* und *detach()* verwenden?
5) Was bewirken die *all*- und die *any*-Funktion? Führen Sie folgendes durch:

```
x <- c(3,4,3)
all(x>3); all(x>2)
any(x>3); any(x>5)
```

6) Finden Sie heraus was die xor Funktion berechnet, indem Sie folgendes eingeben:

```
x <- c(3,4,3)
xor(x>2, x==3)
```

5. Berechnungen

Übersicht

In den vorangehenden Kapiteln haben Sie Datenstrukturen kennen gelernt, und Datensätze geordnet. Basierend darauf lernen Sie in diesem Kapitel, wie Sie damit Berechnungen durchführen können. Das tun Sie einerseits mit vordefinierten Funktionen; anderseits werden Sie einfache Berechnungen selbst programmieren. Dabei ist es für Sie hilfreich zu wissen, wie Sie in R mit Datums- und Zeitformaten umgehen können.

Lernziele

Nachdem Sie dieses Kapitel bearbeitet haben,

- sind Sie in der Lage, vordefinierte Funktionen zu verwenden, und selbst einfache Funktionen zu schreiben (K2).
- können Sie Zeichenfolgen selbst erstellen und effizient bearbeiten (K2).
- können Sie für zwei Zeitformate in eigenen Worten erklären, wie Berechnungen durchgeführt sowie Daten importiert und exportiert werden (K2).

5.1. Funktionen in R

In R sind bereits zahlreiche Funktionen definiert. Funktionen können zusammenfassend sein: Wenn beispielsweise die Mittelwert-Funktion *mean()* auf einen Vektor angewendet wird, besteht das Ergebnis aus einem Wert. Aber es gibt auch Funktionen, deren Resultat die gleiche Struktur wie die Eingabe aufweist. So besteht beispielsweise die Ausgabe der Quadratwurzel-Funktion *sqrt()* aus der Wurzel der einzelnen Elemente des Inputobjekts; die Struktur des Inputobjekts bleibt erhalten. In den folgenden Tabellen sind für beide Fälle wichtige Funktionen aufgelistet.

Zusammenfassende Funktionen

Funktion	*Beschreibung*
mean()	Arithmetischer Mittelwert
max()	Maximum
min()	Minimum
sum()	Summe
prod()	Produkt
length()	Anzahl (!) Elemente eines Objekts
nrow()	Anzahl Zeilen einer Matrix oder eines Dataframes
ncol()	Anzahl Spalten einer Matrix oder eines Dataframes
dim()	Dimensionen (Zeilen/Spalten) einer Matrix oder eines Dataframes

- Wenden Sie die Funktionen auf den Vektor A an, und versuchen Sie die Funktionen dadurch zu verstehen.

```
A <- c(4, 6, 5, 9)
mean(A)
max(A)
min(A)
sum(A)
prod(A)
length(A)
```

Elementweise rechnende Funktionen

Funktion	*Beschreibung*
sqrt()	Quadratwurzel (English: *square root*)
abs ()	Betrag (Englisch: *absolute value*)
sign()	Vorzeichen (Ausgabe-Möglichkeiten: -1, 0, 1)
factorial()	Fakultät (x!)
round()	Runden
ceiling()	Aufrunden (Deutsch: *Decke*)
floor()	Abrunden (Deutsch: *Boden*)
trunc()	Ganze Zahlen (Deutsch: *abbrechen*)
exp()	Exponential Funktion
log()	Logarithmus (Voreinstellung: *Natürlicher Logarithmus*)
log10()	Zehner Logarithmus
sin()	Sinus
cos()	Cosinus
tan()	Tangens
asin()	Arcsinus
acos()	Arccosinus
atan()	Arctangens

- Lernen Sie nun elementweise rechnende Funktionen kennen, bei denen die Struktur der Ausgabe mit derjenigen der Eingabe übereinstimmt. Verwenden Sie dabei die folgenden Vektoren.

```
A1 <- c(4, 6, 5, 9)
A2 <- c(-4, 0, -5, 9)
A3 <- c(0.2, 6.5, 7.5, 4.8 )
A4 <- c(0, pi/2, pi)
A5 <- c(0.1, 0.6, -0.8)
sqrt(A1)
abs (A2)
sign (A2)
factorial(A1)
```

In R wird für die Zahl *Pi* die Abkürzung *pi* gebraucht. Für die Eulersche Zahl *e* besteht hingegen keine Abkürzung; sie wird mit exp(1) erstellt. Die Funktion *round()* erzeugt mathematisch korrektes, unverzerrtes Runden. Dabei werden 0.5 und -0.5 auf die nächste gerade Zahl gerundet. Das unterscheidet sich von kaufmännischem Runden, bei dem 0.5 auf die nächste grössere, und -0.5 auf die nächst kleinere Zahl gerundet wird. Das führt zu inkonsistenten Verschiebungen. Eine Funktion zu kaufmännischem Runden lernen Sie im Kapitel 5.2 kennen.

- round(A3)
  ceiling (A3)
  floor(A3)
  trunc(A3)
  exp(A1)
  exp(1)
  log(A1)
  log10(A1)
  log(A1, 10)
  Bemerkung: Verleichen Sie die beiden letzten Resultate!
  sin(A4)
  cos(A4)
  tan(A4)
  asin(A5)
  acos(A5)
  atan(A5)

Im Kapitel 7 werden Sie zusätzliche Statistikfunktionen von R kennen lernen.

Übungsaufgabe 5.1

Überlegen Sie sich bei den folgenden Aufgaben vor dem Ausführen der Funktion, ob diese zusammenfassend oder elementweise rechnend ist. Welche Funktionen verwenden Sie, um für den Vektor A6 <- c(0.1, -1.5, 5.3) Folgendes zu berechnen:

1) Mittelwert
2) Exponentialfunktion
3) Summe
4) Maximum
5) Aufgerundete Werte
6) Quadratwurzel des Betrags

Lösungen zu Übungsaufgabe 5.1

1) A6 <- c(0.1, -2.5, 5.3)
   Zusammenfassende Funktion
   mean(A6)
2) Elementweise rechnende Funktion
   exp(A6)
3) Zusammenfassende Funktion
   sum(A6)
4) Zusammenfassende Funktion
   max(A6)
5) Elementweise rechnende Funktion
   ceiling(A6)
6) Elementweise rechnende Funktionen
   sqrt(abs(A6))

5.2. Eigene Funktionen

Sie können auf einfache Art selbst Funktionen definieren. Der Aufbau einer Funktionsdefinition sieht folgendermassen aus:

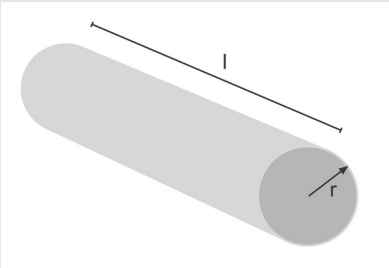
Funktionsname <- function (Argument1, Argument2, usw.) {Befehle}

- Definieren Sie eine Funktion, welche Zahlen mit 3 multipliziert und danach 15 addiert. Sie beinhaltet nur das Argument x. Die Funktion wird *NeueFunktion1* genannt.
  NeueFunktion1 <- function (x) {3*x + 15}
- Probieren Sie die neue Funktion aus:
  NeueFunktion1(7)
  A <- c (-3, 4, 18)
  NeueFunktion1(A)
- Definieren Sie eine Funktion mit dem Namen *Volumen*, welche das Volumen eines Quaders mit den Seitenlängen L, B und H berechnet.
  Volumen <- function (L, B, H) {L * B * H}
- Probieren Sie die Funktion aus!
  Volumen(7, 3, 8)
- Falls Sie die Reihenfolge der Argumente nicht einhalten, müssen Sie diese benennen:
  Volumen(H = 8, L = 7, B = 3)

Eine Funktion mit mehreren Argumenten können Sie nicht ohne weiteres auf Vektoren oder mehrdimensionale Datenstrukturen anwenden; dies werden Sie in Kapitel 5.5 lernen.

Übungsaufgabe 5.2

1) Definieren Sie eine Funktion *V_Abwasser*, die das Volumen einer Abwasserleitung (mit kreisrundem Querschnitt) in Abhängigkeit von deren Länge l und Radius r beschreibt.



2) Wenden Sie Ihre Funktion an, um das Volumen einer Abwasserleitung von 200 m Länge und 0.5 m Radius zu berechnen.

Lösungen zu Übungsaufgabe 5.2

1) `V_Abwasser <- function (l, r) { (pi*r^2) * l }`
2) `V_Abwasser(200, 0.5)`
   Resultat: ~157.1m^3

In Kapitel 5.1 wurde der Unterschied zwischen mathematischem und kaufmännischem Runden erläutert. Eine Funktion für kaufmännisches Runden kann selbst definiert werden.

- Versuchen Sie die folgende Funktions-Definition zu verstehen.
  `round1 <- function(x) {trunc(x+ sign(x)*0.5)}`
  Die Funktion *sign()* gibt das Vorzeichen eines entsprechenden Elements aus. *x+ sign(x)*0.5* bedeutet, dass von negativen Elementen von x 0.5 subtrahiert wird, Null bleibt unverändert, zu positiven Elementen wird 0.5 addiert.
  Danach rundet die *trunc*-Funktion auf ganze Zahlen.
- Schauen Sie sich die Resultate der Funktion an:
  `x1 <- seq(-3, 3, by=0.1)`
  `cbind(x1, round(x1), round1(x1))`

5.3. Zeichenfolgen

Beim Programmieren mit R ist es hilfreich, wenn Sie Zeichenfolgen trennen, analysieren, und gegebenenfalls wieder zusammenfügen können. Die folgenden Funktionen sind dabei von Nutzen:

Funktion	*Beschreibung*
toupper()	Umwandlung in Grossbuchstaben
tolower()	Umwandlung in Kleinbuchstaben
nchar()	Anzahl Zeichen
substring()	Teile der Zeichenfolge auswählen
paste()	Zusammenfügen

- Wenden Sie die Funktionen auf den Vektor *A1* an, und versuchen Sie, die Funktionen dadurch zu verstehen.

```
A1 <- c("Ammonium", "Nitrat")
toupper(A1)
tolower(A1)
nchar(A1)
```

- Neben den Daten, auf welche die Funktion angewendet wird, müssen Sie in der *substring*-Funktion die Nummer des ersten und letzten zu extrahierenden Zeichens angeben. Wird für das letzte Zeichen keine Nummer angegeben, werden alle Zeichen bis zum Schluss der Zeichenfolge ausgewählt.

```
S1 <- substring(A1, 1, 3)
S2 <- substring(A1, 2)
S3 <- substring(A1, 1, 1)
```

- Die *paste()* Funktion dient zum Zusammenfügen von Zeichen oder Zeichenfolgen. Stellen Sie in den folgenden Beispielen den Unterschied zwischen den Argumenten *sep* und *collapse* fest.

```
paste(S3, S2, sep ="_", collapse = NULL)
paste(S3, S2, sep ="_", collapse = "#")
```

Übungsaufgabe 5.3

1) Gegeben ist der Vektor *Tage1*.

```
Tage1 <- c("Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag", "Samstag")
```

Extrahieren Sie die beiden ersten Ziffern jedes Elements und weisen Sie diese einem Vektor *Tage2* zu.

2) Ändern Sie die Elemente von *Tage2* in Grossbuchstaben um; nennen Sie das Resultat *Tage3*.

3) Erstellen Sie einen Vektor *Tage4*, der die Juni-Tage (1. bis 30. Juni) in folgendem Format enthält.

```
"30.Juni"
```

4) Erzeugen Sie - basierend auf den Vektoren *Tage3* und *Tage4* - einen Vektor *Tage5*, der die Juni-Tage im folgenden Format enthält:

```
"MO 30.Juni 2008"
```

Lösungen zu Übungsaufgabe 5.3

```
1)  Tage2 <- substring(Tage1, 1, 2)
2)  Tage3 <- toupper(Tage2)
3)  Tage4 <- paste(1:30, "Juni", sep = ".", collapse = NULL)
4)  Tage5 <- paste(Tage3, Tage4, 2008, sep = " ", collapse = NULL)
```

5.4. Datum und Zeit

Für Berechnungen mit Datum und Zeit (beispielsweise Differenz zwischen zwei Zeiten) sollten Sie spezielle Formate verwenden. R stellt dazu verschiedene Format-Klassen zur Verfügung. Verwenden Sie jeweils die einfachste Format-Klasse, welche für eine bestimmte Anwendung möglich ist! Dadurch reduzieren Sie potenzielle Fehlerquellen. In diesem Leitprogramm werden zwei Formate vorgestellt: *Date* und *POSIXct*. In der folgenden Tabelle sind einige Eigenschaften dieser Formate aufgelistet:

	Date	*POSIXct*
Inhalte	Datum	Datum, Zeit, Zeitzone, usw.
Numerische Einheit	Tage seit dem 1.1.1970	Sekunden seit dem 1.1.1970
Vorteile	Einfachheit	Vielfalt
Nachteile	Kann nicht für Zeiten verwendet werden	Gefahr, dass falsche Zeitzonen Fehler verursachen

Bemerkung: Der Name *POSIX* (von *Portable Operating System Interface*) bezieht sich auf ein Standardisierungsverfahren. Dabei wurde es als notwendig betrachtet, weltweit die Zeitformate zu standardisieren.

Formate

Beim Transformieren zu Datum oder Zeit-Klassen werden Abkürzungen verwendet, damit die entsprechenden Zeichen als Datum respektive Zeit erkannt werden. Beispielsweise wird das Format von *13.05.2008 14:08* mit "%d.%m.%Y %H:%M" angegeben. Die folgende Tabelle enthält einige wichtige Abkürzungen; eine ausführliche Liste können Sie sich mit *?strptime* ansehen.

Abkürzung	*Beschreibung*
%d	Tag eines Monats (1 bis 31)
%H	Stunden (1 bis 24)
%m	Monate (1 bis 12)
%M	Minuten (0 bis 59)
%S	Sekunden (0 bis 61; Begründung: Schaltsekunden)
%Y	Jahr in vier Ziffern (Beispiel: 2008)
%y	Jahr in zwei Ziffern (Beispiel: 08)

Input (*Date*)

- Importieren Sie die Datei *Birsig.csv*, welche die Tagesmittel des Abflusses der Birsig in Binningen im April 2006 in m^3/s enthält.

```
BIRSIG_PFAD <- "E:/R/3_Daten/Birsig.csv"
Bir <- read.table(BIRSIG_PFAD, sep = ";", dec = ",", header = T)
names(Bir); dim(Bir)
head(Bir)
```

- Wandeln das Datum in ein Objekt der Format-Klasse *Date* um. Beachten Sie, dass das D von *as.Date* gross geschrieben wird.

```
Messtag1 <- as.Date(Bir$Messtag, "%d.%m.%Y")
```

- Mit der Funktion *class()* können Sie die Format-Klasse eines Objekts betrachten.

```
class(Messtag1)
```

Input (*POSIXct*)

Für Zeit-Daten verwenden Sie die Format-Klasse *POSIXct*. Beim Importieren müssen Sie beachten, dass Sie die korrekte Zeitzone verwenden. Daten sind meistens nicht in Sommer- sondern in Winterzeit vorhanden. Verwenden Sie deshalb für Daten aus der Schweiz oder Deutschland die Mitteleuropäische Winterzeit (CET).

Deutsch	*Englisch*	*Abkürzung*
Mitteleuropäische Winterzeit	Central European Time	CET
Mitteleuropäische Sommerzeit	Central European Summer Time	CEST
Osteuropäische Zeit	Eastern European Time	EET
Greenwich Mittelzeit	Greenwich Mean Time	GMT
Pazifik Standard Zeit	Pacific Standard Time	PST

- Importieren Sie die Datei *Rheinfelden_Leitf.csv*, welche die Elektrische Leitfähigkeit (µS/cm) des Rheins am 10. Juli 2008 in Rheinfelden enthält.

```
LEI_PFAD <- "E:/R/3_Daten/Rheinfelden_Leitf.csv"
Lei <- read.table(LEI_PFAD, sep = ";", dec = ",", header = T)
names(Lei); dim(Lei)
head(Lei)
```

- Fügen Sie *Datum* und *Uhrzeit* zu einem Objekt *Zeit_Lei* zusammen:

```
Zeit_Lei <- paste(Lei$Datum, Lei$Uhrzeit, sep = " ")
```

- Erstellen Sie daraus ein Objekt der Format-Klasse *POSIXct*. Geben Sie dabei Zeitzone der Daten an!

```
Zeit_Lei1 <- as.POSIXct(strptime(Zeit_Lei, format="%d.%m.%Y %H:%M",
        tz="GMT"), tz="GMT")
```

Hinweis: Bei einigen Betriebssystemen wird mit R 2.8.1 nicht die korrekte Zeitzone CET sondern CEST ausgegeben. Um diesen Fehler mit Sicherheit zu vermeiden, tun Sie so, als wären Ihre Daten in GMT. Das dürfen Sie aber nur tun, solange Sie bloss Daten einer Zeitzone verwenden.

Numerische Elemente (*Date*)

Manchmal lohnt es sich, Objekte der Format-Klasse *Date* in numerische Elemente umzuwandeln. Die Einheit des Resultats ist „Tage“.

- Erstellen Sie ein numerisches Objekt, welches die Anzahl Tage seit dem 1.1.1970 enthält:

```
Mess_Bir_num <- as.numeric(Messtag1)
```

- Wandeln Sie dieses Datum wieder zurück in die Format-Klasse *Date*:

```
Mess_Bir_dat <- as.Date("1970-01-01") + Mess_Bir_num
```

Numerische Elemente (*POSIXct*)

- Erzeugen Sie ein numerisches Objekt, welches die Anzahl Sekunden seit dem 1.1.1970 enthält:

  ```
  Zeit_Lei_num <- as.numeric(Zeit_Lei1)
  ```

- Wandeln Sie dieses wieder zurück in die Formats-Klasse *POSIXct*:

  ```
  Zeit_Lei_dat <- as.POSIXct("1970-01-01 00:00:00" , tz="GMT") +
  Zeit_Lei_num
  ```

Berechnungen (*Date*)

Mit Datums-Objekten können Sie folgende Berechnungen durchführen:

- Zu einem Datum eine Zahl addieren (resp. subtrahieren).
- Eine logische Abfrage zwischen zwei Datums-Objekten ausführen.
- Zwei Datums-Objekte voneinander subtrahieren.

Führen Sie die folgenden Berechnungen aus:

- Addition einer Zahl:

  ```
  Messtag2 <- Messtag1 + 7
  ```

- Logische Abfrage:

  ```
  Messtag1[4] > Messtag1[2]
  ```

- Subtraktion mit zwei Datums-Objekten:

  ```
  Diff_Mess_Bir1 <- Messtag1[2:25] - Messtag1[1:24]
  ```

  Sie können Datumsobjekte auch zuerst in numerische Elemente umwandeln und erst danach subtrahieren. Der Vorteil davon ist, dass das Resultat (Differenz in Tagen) bezüglich der Klasse auch ein numerisches Element ist.

  ```
  Diff_Mess_Bir2 <- as.numeric(Messtag1[2:25]) -
  as.numeric(Messtag1[1:24])
  ```

Berechnungen (*POSIXct*)

Mit Objekten des Formats *POSIXct* können Sie grundsätzlich die gleichen Berechnungen ausführen wie mit *Date*.

- Addition einer Zahl:

  ```
  Zeit_Lei2 <- Zeit_Lei1+600
  ```

- Logische Abfrage:

  ```
  Zeit_Lei1[5] > Zeit_Lei1[8]
  ```

- Subtraktion zwischen zwei *POSIXct*-Objekten:

  ```
  Diff_Zeit_Lei1 <- Zeit_Lei1[2:23] - Zeit_Lei1[1:22]
  ```

  Numerische Subtraktion (Differenz in Sekunden):

  ```
  Diff_Zeit_Lei2 <- as.numeric(Zeit_Lei1[2:23]) - as.numeric(Zeit_Lei1[1:22])
  ```

Beim Exportieren von Zeit oder Datum können Sie festlegen, in welchem Format die Ausgabe erfolgen soll.

Output (*Date*)

- Exportieren Sie das Objekt *Messtag2* in einem im deutschen Sprachraum üblichen Format:

```
Messtag_Neu <- format (Messtag2, "%d.%m.%Y")
```

Output (*POSIXct*)

- Exportieren Sie das Objekt *Zeit_Lei2* in einem im deutschen Sprachraum üblichen Format:

```
Zeit_Lei_Neu <- format(Zeit_Lei2, "%d.%m.%Y %H:%M")
```

- Trennen Sie Datum und Uhrzeit:

```
Datum_Neu <- substring(Zeit_Lei_Neu, 1, 10)
Uhrzeit_Neu <- substring(Zeit_Lei_Neu, 12)
```

Übungsaufgabe 5.4

1) Importieren Sie die Datei *DOC.csv*, welche Angaben über den gelösten organischen Kohlenstoff (mg/l) des Rheins bei Weil am Rhein im Jahr 2003 enthält. Erstellen Sie ein Dataframe *DOC*, und wandeln Sie das Datum in ein Objekt *Messtag1* mit Format *Date* um.
2) Erstellen Sie ein Objekt *Messtag2*, welches Elemente mit dem Datum jeweils zwei Tage vor den Daten von *Messtag1* enthält.
3) Erstellen Sie ein logisches Objekt *Messtag3*, welches angibt, welche Messtage von *Messtag2* sich zeitlich nach dem 1. April 2003 befinden. Hinweis: Das Format *Date* besteht aus "Jahr-Monat-Tag".

Lösungen zu Übungsaufgabe 5.4

1)
```
DOC_PFAD <- "E:/R/3_Daten/DOC.csv"
DOC <- read.table(DOC_PFAD, sep = ";", dec = ",", header = T)
names(DOC); dim(DOC)
head(DOC)

Messtag1 <- as.Date(DOC$Messtag, "%d.%m.%Y")
```
2)
```
Messtag2 <- Messtag1 - 2
```
3)
```
Messtag3 <- Messtag2 > "2003-04-01"
```

Übungsaufgabe 5.5

1) Importieren Sie die Datei *Rheinfelden_Pegel.csv*, welche den Pegelstand des Rheins am 2.Juli 2008 in Rheinfelden enthält (Einheit = Zentimeter, 15-Minuten Werte, GMT). Erstellen Sie ein Dataframe Pegel, verknüfen Sie Datum und Zeit, und wandeln Sie dieses in ein Objekt *Zeit_Peg1* mit Format *POSIXct* um.
2) Erstellen Sie ein Objekt *Zeit_Peg2*, bei welchem zu allen Elementen von *Zeit_Peg1* zwei Minuten dazugezählt werden.
3) Exportieren Sie das Objekt *Zeit_Peg2* in einem im deutschen Sprachraum üblichen Format zu einem Objekt *Zeit_Peg3*.

Lösungen zu Übungsaufgabe 5.5

1)
```
PEG_PFAD <- "E:/R/3_Daten/Rheinfelden_Pegel.csv"
Peg <- read.table(PEG_PFAD, sep = ";", dec = ",", header = T)
names(Peg); dim(Peg)
head(Peg)

Zeit_Peg <- paste(Peg$Datum, Peg$Uhrzeit, sep = " ")
Zeit_Peg1 <- as.POSIXct(strptime( Zeit_Peg,
format="%d.%m.%Y %H:%M", tz="GMT"), tz="GMT")
```
2)
```
Zeit_Peg2 <- Zeit_Peg1 + 2*60
```
3)
```
Zeit_Peg3 <- format(Zeit_Peg2, "%d.%m.%Y %H:%M")
```

Systemzeit

Mit der Funktion *Sys.time()* wird die Systemzeit Ihres Computers ausgegeben. Das können Sie verwenden, um die Dauer eines Programms von Ihnen zu berechnen.

- Die Differenz aus den Systemzeiten am Beginn und Schluss eines Befehls oder Programms ergibt die Berechnungsdauer. Weil Sie bloss eine Differenz berechnen, spielt in diesem Fall die Zeitzone der Systemzeit keine Rolle.

```
t0 <- Sys.time()
#Programm
t1 <- Sys.time()
t1 - t0
```

5.5. Grundsätze des Programmierens

In den Kapiteln 5.6 und 5.7 sind hilfreiche Programmier-Werkzeuge beschrieben. Bevor Sie jedoch einen Code programmieren, sollten Sie mit einfachen Prinzipien vertraut sein, die Sie beim Programmieren anwenden sollten.

Kommentare

Ihr Code soll auch für andere Personen nachvollziehbar sein. Deshalb sollten Sie allgemein verständliche Kommentare anbringen. Das hilft auch Ihnen, zu einem späteren Zeitpunkt Ihren Code zu lesen. Kommentare werden nach dem #-Zeichen eingefügt, damit sie von R nicht eingelesen werden. Schreiben Sie genügend oft Kommentare; meistens werden zu wenig Kommentare angebracht.

Lesbarkeit

Ein Code sollte gut lesbar sein. Schreiben Sie deshalb nicht allzu kompakte Funktionen. Diese verschlechtern nicht nur die Lesbarkeit, sondern stellen auch potenzielle Fehlerquellen dar.

Testen

Bevor Sie Ihren Code auf grosse Datensätze anwenden, sollten Sie diesen mit wenigen Daten testen. Überprüfen Sie nicht bloss den ganzen Code, sondern regelmässig die letzten neu geschriebenen Zeilen. Es lohnt sich, wenn Sie nach spätestens zehn neuen Zeilen den Code testen.

Vektorwertig programmieren

Grundsätzlich wird zwischen kompilierten und interpretierten Programmiersprachen unterschieden. Bei ersteren werden die Befehle während des Übersetzens, bei letzteren beim effektiven Durchlaufen interpretiert. R gehört zu den interpretierten Programmiersprachen. Im Kapitel 5.6 ist beschrieben, wie aus mehreren Durchgängen bestehende Schleifen programmiert werden. In R werden dabei die Zeilen der Schleife bei jedem Durchlauf immer wieder neu interpretiert. Wenn Sie hingegen vektorwertig programmieren (d.h. Operationen mit einem ganzen Vektor gleichzeitig ausführen), muss ihr Code nur einmal interpretiert werden, was sehr viel schneller sein kann. Fazit: Programmieren Sie wenn möglich vektorwertig!

Auf den Seiten 78 bis und mit 80 befinden sich Zusatzaufgaben zu Schleifen und vektorwertigem Programmieren. Falls Sie mit R programmieren werden, beinhalten diese Seiten wichtige Informationen für Sie. Auf Seite 81 finden Sie die Merkpunkte zu diesem Kapitel.

5.6. Schleifen

Auch wenn Sie grundsätzlich vektorwertig programmieren sollen, gibt es Situationen, in denen Sie nur Schleifen verwenden können. In diesem Leitprogramm werden aber nur die wichtigsten Schleifenfunktionen *for* und *if* behandelt.

for-Schleife

- Gegeben ist ein Vektor mit Niederschlagsintensitäten in mm/h.
  ```
  P <- c(1.2, 2.2, 0, 4.5, 0.5)
  ```
  Berechnen Sie daraus mit einer *for*-Schleife den kumulativen Niederschlag.
- Es ist wichtig, dass Sie bei Schleifen das resultierende Objekt vor der Schleife initialisieren:
  ```
  P_kum <- rep(NA,5)
  ```
- In diesem Fall ist das erste Element beider Objekte nicht von der Schleife betroffen.
  ```
  P_kum[1] <- P[1]
  ```
- In der untenstehenden Schleife nimmt das Objekt *i* im ersten Durchlauf den Wert 2, im zweiten Durchlauf den Wert 3, usw. und im letzten Durchlauf den Wert 5 an. Der Inhalt der Schleife steht in geschweiften Klammern. Er lässt sich am Einfachsten verstehen, wenn Sie von rechts zu lesen beginnen. Im ersten Durchlauf (*i* = 2) passiert Folgendes: Das zweite Element von *P* (*P[2]*) wird zum ersten Element von *P_kum* (*P_kum[2-1]*) addiert. Dieser Wert wird dem zweiten Element von *P_kum* (*P_kum[2]*) zugeordnet. Im nächsten Durchgang (*i* = 3) wird *P[3]* zu *P_kum[3-1]* addiert, und *P_kum[3]* zugeordnet; usw.
  ```
  for (i in 2:5) {P_kum[i] <- P_kum[i-1] + P[i]}
  ```
- Vergleichen Sie *P* und *P_kum*:
  ```
  cbind(P, P_kum)
  ```

if-Befehl

Der *if*-Befehl kann nur auf einzelne Elemente angewendet werden. Deshalb kann es sinnvoll sein, ihn in einer *for*-Schleife einzusetzen. Im folgenden Beispiel ist der *if*-Befehl mit einer *else*-Aussage ergänzt („was ist zu tun, falls *if* nicht erfüllt?“).

- Berechnen Sie mit dem Vektor *P* des vorherigen Beispiels, einen Vektor *P_cm*, der aus denjenigen Elementen besteht, die eine Niederschlagsintensität grösser als 2 mm/h aufweisen. Diese sollen neu die Einheit *cm/h* aufweisen, die anderen Elemente sollen gleich Null gesetzt werden.

- Initialisieren des resultierenden Objekts:

```
P_cm <- rep(NA, 5)
```

- Schleife:

```
for (i in 1:5)
{
        if (P[i] > 2) {P_cm[i] <- P[i]/10} else {P_cm[i] <- 0}
}
```

Übungsaufgabe 5.6

1) Gegeben sind die Koordinaten ihrer Meteo-Station und von fünf Niederschlags-Totalisatoren.

```
x_Meteo <- 550
y_Meteo <- 358
x_Total <- c(530, 517, 588, 650, 557)
y_Total <- c(320, 366, 340, 380, 420)
```

Berechnen Sie mit einer *for*-Schleife einen Vektor *Dist*, der die Distanzen der Niederschlags-Totalisatoren zur Meteo-Station enthält.

Hinweis: Die Distanz kann mit dem Satz von Pythagoras berechnet werden:

```
sqrt( (x1 –x2)^2 + (y1 – y2)^2 )
```

Lösungen zu Übungsaufgabe 5.6

1) Initialisieren:

```
Distanz <- rep(NA,5)
```

Schleife:

```
for (i in 1:5)
{
Distanz[i] <- sqrt((x_Meteo - x_Total[i])^2 + (y_Meteo - y_Total [i])^2)
}
```

5.7. Vektorwertig programmieren

Es lohnt sich sehr, vektorwertig zu programmieren! Bei Umweltanalysen werden oft grosse Datensätze verwendet. Falls Sie mit vektorwertigem Programmieren die Rechenzeit Ihres Codes beispielsweise auf 5% senken können, dauert ein langer Rechenprozess anstatt 10 Stunden nur noch 30 Minuten. Deshalb sollten Sie sich angewöhnen, auch schon bei kurzen Programmen wenn immer möglich vektorwertig zu programmieren.

Um vektorwertig zu programmieren, müssen nicht immer Vektoren gleicher Länge vorhanden sein. Beispielsweise können Sie das Ergebnis von Übungsaufgabe 5.6 auch ohne Schleife berechnen. In diesem Beispiel besteht *x_Meteo* aus einer Zahl. R wendet diese jedoch auf sämtliche Elemente von *x_Total* an.

- Berechnen Sie die Distanzen aus Übungsaufgabe 5.6 vektorwertig:
  ```
  Distanz_Vektor <- sqrt((x_Meteo - x_Total)^2 + (y_Meteo - y_Total)^2)
  ```

Neben dieser Art von vektorwertigem Programmieren, gibt es in R-Funktionen, die speziell auf vektorwertige Analysen zugeschnitten sind. In diesem Leitprogramm werden die Funktionen *apply()*, *tapply()* und *ifelse()* beschrieben.

apply-Funktion

Mit *apply* können Sie Funktionen auf Zeilen oder Spalten von Matrizen anwenden.

- Die Matrix *Sauer* beinhaltet die absolute Häufigkeit von Sauerklee an verschiedenen Standorten. Die Spalten beziehen sich auf die Lichtverhältnisse der Standorte (1. Spalte = wenig, 2. Spalte = mittel, 3. Spalte = viel); die Zeilen beschreiben die Feuchtigkeit (1. Zeile = wenig, 2. Zeile = mittel, 3. Zeile = viel).
  ```
  Sauer <- matrix(c(15, 27, 20, 9, 11, 11, 0, 4, 3), 3, 3)
  ```
- Berechnen Sie mit der *apply*-Funktion einen Vektor, der die Summe von Sauerklee-Pflanzen nach Lichtverhältnissen (=Spalten) gliedert. Sie müssen folgende Argumente festlegen: Den Datensatz, Zeilen-(1) oder Spalten-Ansatz (2) und die anzuwendende Funktion.
  ```
  Licht <- apply(Sauer, 2, sum)
  ```
- Berechnen Sie einen Vektor, der die mittlere Anzahl Sauerklee-Planzen für verschiedene Feuchtigkeits-Standorte beinhaltet.
  ```
  Feucht <- apply(Sauer, 1, mean)
  ```

tapply

Die *tapply*-Funktion (*t* steht für Tabelle) ist hilfreich, wenn Sie Datensätze nach bestimmten Faktoren analysieren wollen.

- Importieren Sie die Ihnen bereits bekannten Daten *Phosphor_1996* als Dataframe:

```
P_PFAD <- "E:/R/3_Daten/Phosphor_1996.csv"
P <- read.table(P_PFAD, sep =";", dec=",", header=T)
names(P); dim(P)
head(P)
```

- Summieren Sie die Einträge aus Deutschland nach dem Faktor *Eintrag*.

```
tapply(P$Deutschland, P$Eintrag, sum)
```

Übungsaufgabe 5.7

1) Importieren Sie die Datei *Grundwasser.csv* als Dataframe. Die Datei enthält Daten bezüglich dem mittleren monatlichen Grundwasserstand in Basel von 2000 bis 2007.
2) Berechnen Sie den mittleren Grundwasserstand pro Monat.

Lösungen zu Übungsaufgabe 5.7

1)
```
GRU_PFAD <- "E:/R/3_Daten/Grundwasser.csv"
Gru <- read.table(GRU_PFAD, sep =";", dec=",", header=T)
names(Gru); dim(Gru)
head(Gru)
```

2)
```
tapply(Gru$Grundwasserstand, Gru$Monat, mean)
```

ifelse

Sie haben in Kapitel 5.6 eine *if*-Schleife verwendet, um Starkniederschläge herauszufiltern. Diese Berechnung hätten Sie auch ohne Schleife mit Ihnen bekannten Mitteln durchführen können.

-
```
P <- c(1.2, 2.2, 0, 4.5, 0.5)
ob_P <- P > 2
P_cm <- P * 0
P_cm[ob_P] <- P[ob_P] / 10
```

Im Gegensatz zu *if*, ist die *ifelse*-Funktion vektorwertig! Damit können Sie das gleiche Problem noch effizienter lösen.

-
```
ifelse(P > 2, P_cm <- P, 0)
```

5.8. Merkpunkte

Nun haben Sie einige Möglichkeiten kennen gelernt, um mit R Berechnungen durchzuführen. Repetieren Sie die wichtigsten Merkpunkte des fünften Kapitels:

Vorgegebene Funktionen

- Zusammenfassende Funktionen: *mean()*, *max()*, *min()*, *sum()*, *prod()*.
- Elementweise rechnende Funktionen: *sqrt()*, *abs()*, *round()*
  = mathematisches Runden, *ceiling()*, *floor()*, *trunc()*, *exp()*, *log()*, *sin()*.
- Konstanten: *pi*, *exp(1)*.

Eigene Funktionen

- Funktionsname <- function (Argument1, Argument2, usw.) {Befehle}.
  Beispiel: Neu <- function (x1, x2) {x2 ^ (x1 –x2)}.

Zeichenfolgen

- *toupper()*, *tolower()*, *nchar()*, *substring()*, *paste()*.

Datum und Zeit

- *Date*: Anzahl Tage seit dem 1.1.1970, kann nur ein Datum lesen.
- *POSIXct*: Anzahl Sekunden seit dem 1.1.1970, vielfältig verwendbar.
- Erkennungs-Formate: Mit %-Abkürzung; Beispiel: %M für Minuten.
- *Date*-Input: *as.Date()*.
- *POSIXct*-Input: *as.POSIXct()*, falls notwendig vorher Datum und Uhrzeit zusammenfügen; die Zeitzone der Daten angeben! Greenwich Mean Time = GMT.
- Datum und Zeitformate können zu numerischen Formaten und zurück transformiert werden.
- Berechnungen mit Zeitformaten (oder Datumsformaten): Von einer Zeit eine Zahl subtrahieren, logische Abfragen zwischen Zeiten, Zeiten voneinander subtrahieren.
- Output: Mit der *format*-Funktion.
- Systemzeit des Computers: *Sys.time()*.

Grundlagen des Programmierens

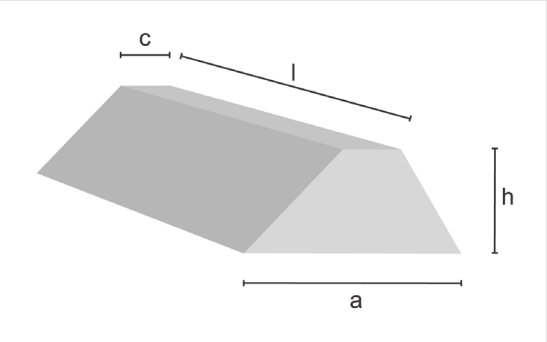
- Nachvollziehbare Kommentare, hohe Lesbarkeit, regelmässig testen
- Vektorwertig programmieren!

5.9. Lernkontrolle

Falls Sie mit dem Lernstoff dieses Kapitels genügend vertraut sind, beginnen Sie nun mit der Lernkontrolle!

Fragen

1) Gegeben ist der Vektor `A <- c(2.4, 7.8, 1.5, 5.4)`. Welchen Befehl verwenden Sie, um den Vektor A auf ganze Zahlen abzurunden? Nennen Sie zudem den Befehl, mit dem Sie das kleinste Element von A finden.
2) Definieren Sie eine Funktion *V_Damm*, die das Volumen *V* eines Damms in Abhängigkeit von dessen Länge *l*, Höhe *h* sowie den Kantenlängen *a* und *c* beschreibt. Hinweis: $V = \frac{a+c}{2} \cdot h \cdot l$



3) Wenden Sie Ihre Funktion für l = 70 m, h = 2.5 m, a = 7 m und c = 3 m an. Nennen Sie das Resultat.
4) Importieren Sie die Datei *Romanshorn.csv* als Dataframe. Leider enthält diese nur Wasserstände, aber kein Datum. Es ist aber bekannt, dass diese Daten Tagesmittelwerte vom 1. bis 30. August 2006 sind. Erstellen Sie einen Vektor *Datum_Rom* mit den entsprechenden Daten im Format 1.8.2006.
5) Erstellen Sie einen Vektor *Datum_Neu1*, welcher das neue Datum im Format *Date* enthält.
6) Importieren Sie die Datei *Rheinfelden_Temp* zu einem Dataframe. Erstellen Sie aus Datum und Uhrzeit einen Vektor vom Format *POSIXct*.
7) Berechnen Sie die Zeitdifferenz in Sekunden zwischen den ersten zehn Werten.

Die Lösungen zur Lernkontrolle finden Sie im Kapitel 8.

5.10. Kapiteltest

Falls Sie sich bei den Themen dieses Kapitels sicher fühlen, melden Sie sich bei der Lehrperson zum Kapiteltest. Der Test zu diesem Kapitel verläuft gleich wie die vorherigen Kapiteltests. Sie drucken Ihre Lösungen aus, und lassen Sie von der Lehrperson korrigieren. Für den Kapiteltest stehen Ihnen maximal 20 Minuten zur Verfügung.

5.11. Zusatzaufgaben

Falls Sie dieses Kapitel wesentlich früher als andere Studierende durch gearbeitet haben, können Sie sich mit den folgenden Zusatzaufgaben befassen, bevor Sie mit dem Kapitel 6 beginnen. Oder nutzen Sie die Zeit, in der die Lehrperson Ihren Kapiteltest korrigiert, um Zusatzaufgaben zu bearbeiten!

1) Versuchen Sie die Funktion *round2* zu verstehen, die kaufmännisch rundet, wobei mit dem Argument *deci* die Anzahl Dezimalstellen gewählt werden kann.

```
round2 <- function( x, deci ) { 1/(10^deci) * trunc( x * (10^deci) +
sign(x)*0.5 ) }
x1 <- seq( -2, 2, by=0.01 )
cbind(x1, round2(x1,1) )
```

2) Sie wollen Temperaturdaten auf 0.05°C runden. Schreiben Sie dazu zwei Funktionen *round3* (kaufmännisches) und *round4* (mathematisches Runden).
3) Machen Sie sich mit den Funktionen *cat()*, *grep()*, *sub()* und *gsub()* vertraut.
4) Lesen Sie Grothendieck und Petzoldt (2004). In dieser kurzen Publikation werden Zeit- und Datumsformate verständlich erläutert.
5) Lesen Sie in Ligges (2007) auf Seite 52, wie *while*-Schleifen verwendet werden können.
6) Aus *Markus Weiler* wird *Sukram Reliew* (wenn Sie die Buchstaben von rechts nach links lesen)! Schreiben Sie eine Funktion, die diese Umwandlung vornimmt.

   Hinweis: Verwenden Sie die *strsplit*-Funktion. Wenden Sie *lapply* und *sapply* in sinnvoller Weise auf den Output von *strsplit()* an (siehe Ligges (2007), Seite 105 bis 107).
7) Was bewirken die *diff*- und *cumsum*-Funktion?
8) Wozu können die Funktionen *pmin()* und *pmax()* verwendet werden?

6. Deskriptive Statistik

Übersicht

Einerseits wird R oft als Statistikprogramm betrachtet. Anderseits bezeichneten Ihaka und Gentleman R als eine Sprache für Datenanalysen und Grafiken! Da Sie mit R sehr effizient statistische Analysen durchführen können, und gleichzeitig die Grafikmöglichkeiten vielfältig sind, eignet sich R besonders gut für deskriptive (=beschreibende) Statistik. In diesem Kapitel lernen Sie, wie Sie Daten mit Grafiken und statistischen Auswertungen beschreiben können. Dabei wird unterschieden, ob Sie Daten einer Variablen analysieren, oder Daten von zwei Variablen vergleichen.

Lernziele

Nachdem Sie dieses Kapitel bearbeitet haben, können Sie

- in eigenen Worten formulieren, wie Daten mit Funktionen und Grafiken bezüglich Lage und Streuung beschrieben werden (K2).
- den Aufbau von Grafiken variieren, indem Sie Mehrfachdarstellungen erzeugen oder die Grafikränder verändern (K3).
- eventuelle Korrelationen zwischen zwei Datenreihen analysieren (K3).

6.1. Grafiken

Die Grafikmöglichkeiten von R sind sehr vielfältig! In R erstellen Sie eine Grafik mit einem Programmier-Code. Im Gegensatz zu Excel-Grafiken ist das Erstellen von R-Grafiken relativ aufwändig. Hingegen können Sie den Code einer Grafik sehr einfach wieder verwenden. Weil R-Grafiken mit einem Code erstellt werden, können keine einzelnen Grafik-Elemente gelöscht, sondern nur hinzugefügt werden. Falls das nicht ausreicht, können Sie eine Grafik einfach überschreiben, indem Sie Ihren Code nochmals laufen lassen. In R wird grundsätzlich unterschieden zwischen konventionellen und Trellis-Grafiken. Im vorliegenden Leitprogramm werden aber nur erstere behandelt.

In diesem Kapitel werden Sie den Datensatz *Weil_O2* verwenden, um damit die Grafik-Grundfunktionen von R kennen zu lernen. Die Datei enthält die Konzentration von gelöstem Sauerstoff (mg/l) des Rheins in Weil am Rhein. Die Beprobung fand alle 14-Tage in den Jahren 2003 und 2004 statt. Zudem enhält die Datei Temperatur (°C), Abfluss (m^3/s) sowie das Messdatum.

- Importieren Sie die Datei *Weil_O2.csv* zu einem Dataframe *O2*.

```
O2_PFAD <- "E:/R/3_Daten/Weil_O2.csv"
O2 <- read.table(O2_PFAD, sep = ";", dec = ",", header = T)
names(O2); dim(O2)
head(O2)
```

plot-Funktion

Die Funktion *plot()* ist fundamental zum Erzeugen von Grafiken in R! Sie ist eine generische Funktion, was bedeutet, dass sie sich dem Input anpasst. Aus diesem Grund können Sie die *plot*-Funktion für verschiedene Arten von Daten anwenden. Als Argumente verwendet sie unter anderem die x- und y-Koordinaten der darzustellenden Daten in dieser Reihenfolge.

Device

In R erstellte Grafiken können auf verschiedenen Ausgabegeräten, so genannten *Devices* (Englisch: Gerät), ausgegeben werden. Das *Device* wählen Sie, indem Sie mit einer Funktion das entsprechende *Device* öffnen. Bei Windows-Betriebssystemen bestehen unter anderem die folgenden Ausgabemöglichkeiten für Grafiken:

Ausgabe	*Funktion*
Bildschirm-Ausgabe	*windows()*
jpg-Datei	*jpeg()*; Vorsicht: nicht *jpg()*
bmp-Datei	*bmp()*
pdf-Datei	*pdf()*

Nachdem die Grafik ausgegeben worden ist, müssen Sie das *Device* mit dem Befehl *dev.off()* wieder schliessen. Bei Bildschirmausgaben können Sie auch auf die rechte obere Ecke der Grafik klicken.

- Gemäss Voreinstellung entsteht eine Bildschirmausgabe. Betrachten Sie mit folgendem Befehl die Voreinstellungen:

```
options("device")
```

Um eine Grafik auf dem Bildschirm anzeigen zu lassen, müssen Sie deshalb kein *Device* mehr öffnen. Erstellen Sie nun eine Grafik mit Temperatur (x-Koordinaten) und Sauerstoff-Konzentration (y-Koordinaten).

- `plot(O2$Temp, O2$O2)`
- Schliessen Sie das *Device* entweder indem Sie auf die rechte obere Ecke klicken oder mit `dev.off()`

Nun werden Sie eine Grafik im Device *jpeg* erstellen. Mit *setwd()* wählen Sie als Arbeitsverzeichnis den Ordner *E:/R/temp* aus. Das Device wird geöffnet, indem Sie in der *jpeg*-Funktion den Dateinamen der neuen Datei sowie deren Breite (*width*) und Höhe (*height*) in Anzahl Pixel angeben wird. Hinweis: Wählen Sie immer Dateinamen, die auf den Inhalt der Grafik hinweisen, damit Sie diese später wieder erkennen!

- Geben Sie folgendes ein:
  ```
  TEMP_O2_PFAD <- "E:/R/temp/Weil_Temp_O2.jpg"
  jpeg(TEMP_O2_PFAD, width = 600, height = 400)
  ```
- Platzieren Sie Ihre Grafik im geöffneten Device:
  ```
  plot(O2$Temp, O2$O2)
  ```
- Schliessen Sie das Device:
  ```
  dev.off()
  ```
- Betrachten Sie die Datei *Weil_Temp_O2.jpg* in Ihrem Arbeitsordner unter *E:/R/temp.*

Übungsaufgabe 6.1

1) Erstellen Sie eine Grafik *Weil_Abfluss_O2.bmp,* welche den Abfluss (x-Achse) und die Sauerstoffkonzentration (y-Achse) enthält. Wählen Sie die folgenden Datei-Eigenschaften: Breite = 600 Pixel, Höhe = 400 Pixel.

Lösungen zu Übungsaufgabe 6.1

1)
```
ABFLUSS_O2_PFAD <- "E:/R/temp/Weil_Abfluss_O2.bmp"
bmp(ABFLUSS_O2_PFAD, width = 600, height = 400)
        plot(O2$Abfluss, O2$O2)
dev.off()
```

Bemerkung: Die Qualität von *bmp*-Dateien ist oft höher verglichen mit *jpg*-Grafiken. Deshalb werden in diesem Leitprogramm meistens *bmp*-Dateien erstellt. In R gibt es auch ein Device *tiff*, das qualitativ ansprechende Grafiken erstellt, leider aber instabil ist. Deshalb wird dieses Device im vorliegenden Leitprogramm nicht verwendet.

plot-Parameter

In der *plot*-Funktion können Sie einige Grafik-Parameter festlegen. Sie werden in der untenstehenden Tabelle erklärt. Die Abkürzungen vieler Parameter beruhen auf Englischen Namen.

Parameter	*Bedeutung*	*Abkürzungserklärung*
col	Farbe. Mit der Funktion *palette()* erhalten Sie eine Farbauswahl, mit *colors()* eine komplette Liste. Weitere hilfreiche Farbfunktionen sind *rainbow()* oder *rgb()*.	color = Farbe
bg	Hintergrund (nur für bestimmte Zeichen; beispielsweise pch = 21, 24 oder 23).	background = Hintergrund
main	Überschrift	main = das Wichtigste
pch	Punktzeichen (als Nummer): 1 2 5 16 17 18 21 24 23 3 4 8	point character = Punktzeichen
type	Grafiktyp: "p" "l" "b" "s" "n"	type = Typ points = Punkte lines = Linien both = beides stair steps = Treppenstufen no data = keine Daten (nur Achsen und Beschriftungen)
xlab (analog: ylab)	Beschriftung der x-Achse,	x-labeling = x-Beschriftung
xlim (analog: ylim)	Grenzen der x-Achse	x-limits = x-Grenzen

Verschönern Sie nun damit ihre zuvor erstellte Grafik! Die Befehle zum Erstellen der Grafik sind unterteilt, damit sie besser kommentiert werden können. Geben Sie aber den gesamten Befehl erst am Schluss ein.

- Der Befehl besteht aus den folgenden Komponenten:

  Öffnen des *Device,* wählen der Eigenschaften der neuen Datei:

  ```
  TEMP_O2_1_PFAD <- "E:/R/temp/Weil_Temp_O2_1.bmp"
  bmp(TEMP_O2_1_PFAD, width = 500, height = 500)
  ```

  Plot-Funktion; wählen der darzustellenden Daten:

  ```
  plot(O2$Temp, O2$O2,
  ```

  Titel:

  ```
  main = "Sauerstoff und Temperatur",
  ```

Grenzen der x- und y-Achse wählen:

xlim = c(0,30), ylim = c(0,15),

Achsenbeschriftung:

xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",

Grafiktyp: Punkte. In diesem Beispiel macht es wenig Sinn, Linien oder Treppenstufen zu verwenden.

type = "p",

Punktzeichen = Dreieck, (Rand-)Farbe = rot, Hintergrundfarbe = orange.

pch = 24, col = "red", bg = "orange")

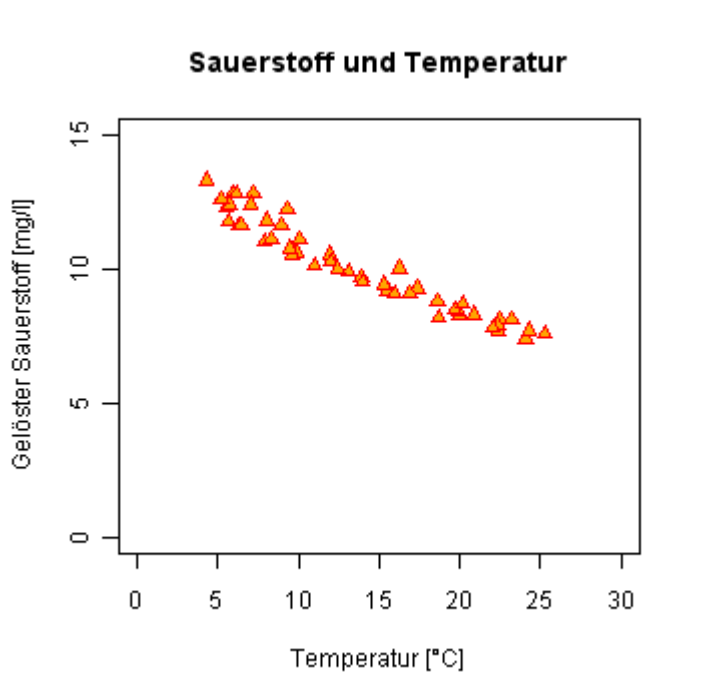
Schliessen des Devices:

dev.off()

- Geben Sie nun den gesamten Befehl ein:

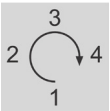
```
bmp(TEMP_O2_1_PFAD, width = 500, height = 500)
    plot(O2$Temp, O2$O2,
    main = "Sauerstoff und Temperatur",
    xlim = c(0, 30), ylim = c(0, 15),
    xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",
    type = "p", pch = 24, col = "red", bg = "orange")
dev.off()
```

- Sie erhalten damit das folgende Bild:



par-Funktion

In der *plot*-Funktion können Sie bereits einige Grafik-Parameter benützen. Die Funktion *par()* ist eine Ergänzung dazu, und stellt viele Grafikmöglichkeiten zur Verfügung. Wenn Sie mit *par()* gewisse Grafik-Parameter festgelegen, werden diese und nicht die voreingestellten Werte zum Erzeugen einer Grafik verwendet. Eine Liste der Grafik-Parameter erhalten Sie mit *?par*, Voreinstellungen können mit *par()* betrachtet werden. Die folgende Tabelle enthält die wichtigsten Parameter. Es lohnt sich, diese zu kennen! Deshalb werden Sie in diesem Kapitel des Leitprogramms Beispiele zu den unten aufgelisteten Grafik-Parametern bearbeiten.

Parameter	*Erklärung*	*Abkürzungserklärung*
cex	Streckungsfaktor (Zahl >1: vergrössern, Zahl <1: verkleinern)	character expansion= Zeichen-Streckung
font	Schriftart (1 = normal, 2 = fett, 3 = kursiv, 4 = fett und kursiv)	font = Schriftart
lty	Linientyp (1=normal, 2=gestrichelt, 3=gepunktet, 4= Striche-Punkte),	line type = Linientyp
log	Logarithmische Achsen: log = "x" log ="y" log = "xy" *xlim* und *ylim* nicht angeben, wenn log verwendet wird!	logarithmic axis = Logarithmische Achsen
lwd	Liniendicke	line width = Linienbreite
mar	Rand (in Anzahl Textlinien); Vektor in folgender Reihenfolge (Uhrzeigersinn): 	margin = Rand
mex	Streckung des Rands inklusive allen Elementen.	margin expansion = Streckung des Rands
mfrow	Anordnung von Grafiken (zeilenweise)	multiple figures (row) = mehrere Grafiken (nach Zeilen)
oma	Aussenrand (in Anzahl Textlinien); Vektor in folgender Reihenfolge (Uhrzeigersinn): 	outer margin = Aussenrand
tcl	Länge der Achsenmarkierungen	tick length= Länge der Markierungen
xaxt (analog: yaxt)	Zeichnen einer x-Achse	x-axis (true) = x-Achse (wahr)

Die Parameter *cex* und *font* können entweder für die ganze Grafik oder auf einzelne Elemente angewendet werden. Beispielsweise wird der Schriftparameter *font* wie folgt auf einzelne Elemente angewendet: *font.axis* (Achsenzahlen), *font.lab* (Achsenbeschriftung) und *font.main* (Überschrift).

Grössenanpassung

Sie können mit *cex* die Grösse aller Elemente der Grafik verändern; cex = 2 bedeutet, dass alle Elemente doppelt so gross werden. Später in diesem Kapitel werden Sie lernen, wie Sie die Grösse von Symbolen ändern können.

- Wählen Sie mit *par*-Funktion cex =2. Alle anderen Eingaben verändern Sie nicht.

```
TEMP_O2_CEX2_PFAD <- "E:/R/temp/Weil_Temp_O2_Cex2.bmp"
bmp(TEMP_O2_CEX2_PFAD, width = 500, height = 500)
   par(cex = 2)
   plot(O2$Temp, O2$O2,
       main = "Sauerstoff und Temperatur",
       xlim = c(0,30), ylim = c(0,15),
       xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",
       type = "p", pch = 24, col = "red", bg = "orange")
dev.off()
```

- Schauen Sie sich die neue Datei an! Alle Elemente (Überschrift, Punktzeichen, Achsenzahlen, Achsenbeschriftung, Abstände zwischen Achsenzahlen und -Beschriftung) sind nun doppelt so gross.
- Sie können ebenfalls bloss die Grösse einzelner Elemente verändern. Verkleinern Sie die Überschrift (*cex.main*), Achsenbeschriftung (*cex.lab*), Achsenzahlen (*cex.axis*) und die Abstände von Achsenmarkierungen, Achsenzahlen und Achsenbeschriftung von den Achsen (mex). Hinweis: Die Voreinstellung für *cex.main* ist 1.2; siehe *par()*.

```
TEMP_O2_CEX2a_PFAD <- "E:/R/temp/Weil_Temp_O2_Cex2a.bmp"
bmp(TEMP_O2_CEX2a_PFAD, width = 500, height = 500)
   par(cex = 1, cex.main = 1, cex.axis = 0.8, cex.lab = 1.2, mex = 1.1)
   plot(O2$Temp, O2$O2,
       main = "Sauerstoff und Temperatur",
       xlim = c(0,30), ylim = c(0,15),
       xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",
       type = "p", pch = 24, col = "red", bg = "orange")
dev.off()
```

Übungsaufgabe 6.2

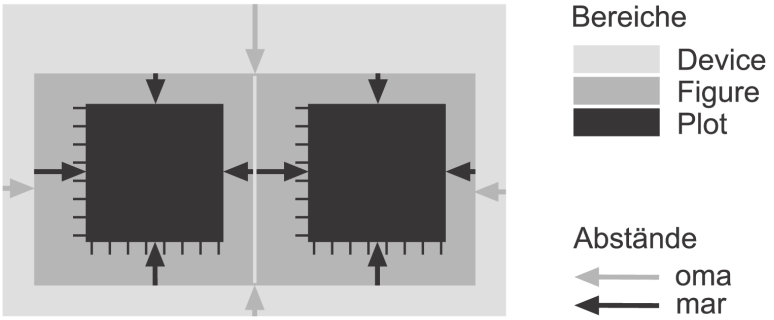
1) Verwenden Sie die zuletzt verwendete Datei (*Weil_Temp_O2_Cex2a.bmp*). Verändern Sie a) die Farbe der Punkte (Rand und Hintergrund), b) das Punktzeichen, c) Grösse und Inhalt der Achsenbeschriftung.
2) Erstellen Sie basierend auf *Weil_Temp_O2_Cex2a.bmp* eine Grafik *Weil_Temp_O2_Cex2b.bmp,* bei welcher die Überschrift um 60% grösser und fett sowie die Achsenbeschriftung um den Faktor 0.8 kleiner, fett und kursiv ist.

Lösungen zu Übungsaufgabe 6.2

1) Es sind viele Lösungen möglich!
2)
```
TEMP_O2_CEX2b_PFAD <- "E:/R/temp/Weil_Temp_O2_Cex2b.bmp"
bmp(TEMP_O2_CEX2b_PFAD, width = 500, height = 500)
   par(cex = 1, cex.main = 1.6, cex.axis = 0.8, cex.lab = 1.2*0.8,
      mex = 1.1, font.main = 2, font.lab = 4)
   plot(O2$Temp, O2$O2,
       main = "Sauerstoff und Temperatur",
       xlim = c(0,30), ylim = c(0,15),
       xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",
       type = "p", pch = 24, col = "red", bg = "orange")
dev.off()
```

Bildaufteilung

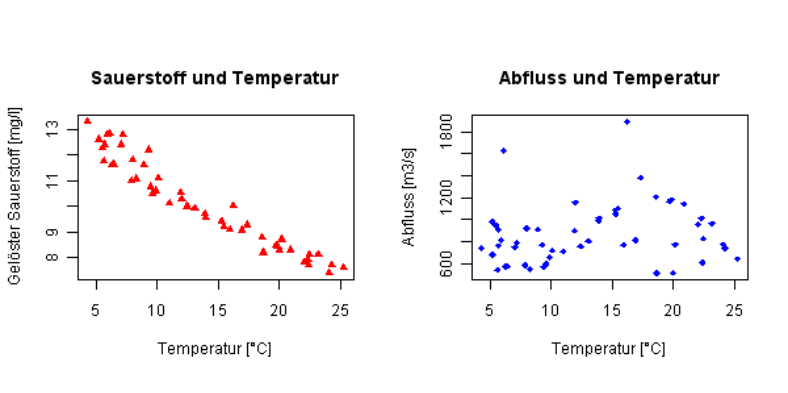
In einer Datei können Sie mehrere Grafiken anordnen. In R-Grafiken werden dazu die folgenden Bildbereiche unterschieden: *Device*-Bereich (Ausmasse der Datei), *Figure* (Fläche der Grafik) und *Plot*-Bereich (Darstellungsbereich der Daten innerhalb der Achsen). Die drei Bereiche legen Sie wie folgt fest: Zuerst bestimmen Sie die Ausmasse der Datei beim Öffnen des *Device* (siehe vorherige Beispiele: *width = 500*, *height = 500*). Mit der *par*-Funktion legen Sie die Parameter *oma* (äusserer Rand) und *mar* (innerer Rand) fest. Damit sind die drei Bereiche definiert!



- Mit *mfrow* = c(1, 2) werden die nächsten zwei ausgegebenen Grafiken in einer Zeile und zwei Spalten angeordnet. Deshalb ist es sinnvoll, die Ausmasse des Device entsprechend zu wählen (Breite = 1000 Pixel, Höhe = 500 Pixel).
- Danach bestimmen Sie die Breite der Ränder mit den Parametern *oma* und *mar*. Dabei verwenden Sie Vektoren mit vier Zahlen für den unteren, linken, oberen und rechten Rand.
- Geben Sie Folgendes ein:

```
TEMP_O2_Q_PFAD <- "E:/R/temp/Weil_Temp_O2_Abfluss.bmp"
bmp(TEMP_O2_Q_PFAD, width = 1000, height = 500)
  par(mfrow = c(1,2), oma = c(1,0,2,0), mar = c(5.1,4.1,4.1,2.1))
  plot(O2$Temp, O2$O2, main = "Sauerstoff und Temperatur",
    xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",
    pch = 17, col = "red")
  plot(O2$Temp, O2$Abfluss, main = "Abfluss und Temperatur",
    xlab = "Temperatur [°C]", ylab = "Abfluss [m³/s]",
    pch = 18, col = "blue")
dev.off()
```

- Sie erhalten die folgende Grafik:



Low-level Grafiken

Bei konventionellen R-Grafiken wird zwischen high-level und low-level Grafiken unterschieden. Bisher haben Sie high-level Grafiken erstellt: Das waren vollständige Abbildungen mit Achsen, Beschriftungen und Datendarstellungen. Low-level Grafiken sind hingegen bloss Ergänzungen zu high-level Grafiken. Beispielsweise kann damit eine Grafik mit bestimmten Elementen ergänzt werden.

points-Funktion

Die *points*-Funktion ist eine low-level Grafik-Funktion. Damit können Sie weitere Daten zu einer bestehenden Grafik hinzufügen. Entgegen ihrem Namen, können Sie mit *points()* Daten nicht nur als Punkte, sondern auch als Linien darstellen. Vorsicht: Der *Plot*-Bereich wird nur aufgrund der *plot*- und nicht der *points*-Funktion festgelegt! Wählen Sie *xlim* und *ylim* so, dass alle Daten dargestellt werden!

- Erstellen Sie eine Grafik, in der die Sauerstoff/Temperatur-Beziehung nach Jahren getrennt ist. Hinweis: Die ersten 26-Zeilen des Dataframe *O2* enthalten Daten aus dem Jahr 2003, die folgenden Zeilen aus dem Jahr 2004.

```
TEMP_O2_P_PFAD <- "E:/R/temp/Weil_T_O2_nachJahr_p.bmp"
bmp(TEMP_O2_P_PFAD, width = 500, height = 500)
   plot(O2$Temp[1:26], O2$O2[1:26],
      main = "Weil am Rhein", xlim = c(0,30), ylim = c(0,15),
      xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",
      pch = 17, col = "blue")
   points (O2$Temp[27:51], O2$O2[27:51],
      pch = 18, col = "red", cex = 2)
dev.off()
```

Grösse der Punktzeichen

In der zweituntersten Zeile der oben-stehenden Eingabe wurde die Grösse der Punktzeichen verdoppelt. Wenn Sie die Grösse der Punktzeichen einzelner Datenreihen verändern wollen, lassen Sie sich mit der *plot*-Funktion lediglich Achsen und Beschriftung ausgeben (*type* = "n"). Die Punkte fügen Sie anschliessend mit der low-level-Funktion *points()* dazu.

-

```
TEMP_O2_P1_PFAD <- "E:/R/temp/Weil_T_O2_nachJahr_p1.bmp"
bmp(TEMP_O2_P1_PFAD, width = 500, height = 500)
   plot(O2$Temp[1:26], O2$O2[1:26],
      main = "Weil am Rhein", xlim = c(0,30), ylim = c(0,15),
      xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",
      pch = 17, col = "blue", type = "n")
   points (O2$Temp[27:51], O2$O2[27:51],
      pch = 18, col = "red", cex = 2)
   points (O2$Temp[1:26], O2$O2[1:26],
      pch = 17, col = "blue", cex = 1.6)
dev.off()
```

Legende

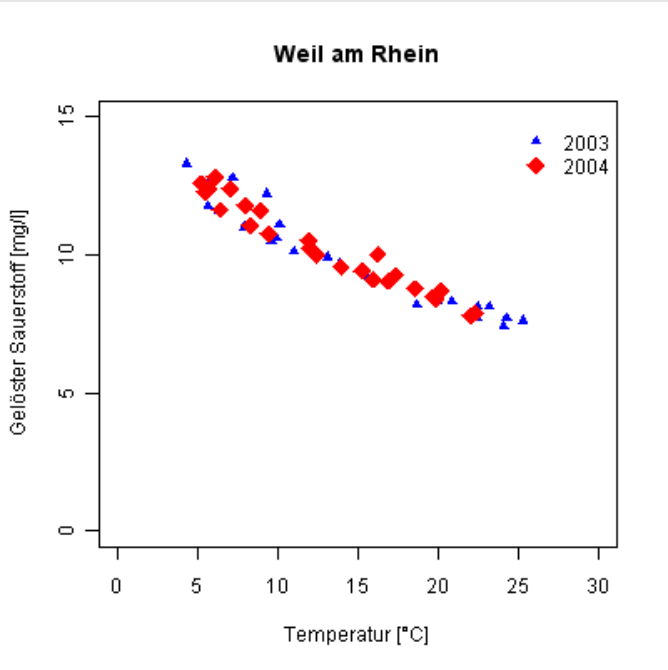
Mit der low-level-Funktion *legend()* können Sie zu Ihrer Grafik eine Legende hinzufügen. Dabei müssen Sie nicht nur die Beschriftung in der Legende, sondern auch Punktzeichen-Grösse und -Farbe nochmals angeben. Sämtliche Parameter die mehrere Elemente betreffen (beispielsweise die Zeichen der Legende) werden als Vektoren angegeben. Mit *?legend* erhalten Sie eine Liste aller Parameter. Die folgende Tabelle enthält die gebräuchlichsten Parameter:

Parameter	*Erklärung*	*Abkürzungserklärung*
bty	Legendenumrandung "f" = mit Umrandung "n" = ohne Umrandung	box type = Art der Umrandung full = voll no = keine
pt.cex	Streckungsfaktor einzelner (!) Symbole	point character expansion = Symbol-Streckung
x	Möglichkeit 1: x-Koordinate der linken oberen Ecke der Legende (in Einheiten der Daten der Grafik; siehe Achse), dabei ist das Angeben einer y-Koordinate notwendig. Möglichkeit 2: Position der Legende mit folgenden Optionen: "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center"	-
y	y-Koordinate der linken oberen Ecke der Legende (in Einheiten der Daten der Grafik; siehe Achse)	-

- Erstellen Sie nun eine Grafik mit Legende:

```
TEMP_O2_LEG1_PFAD <- "E:/R/temp/Weil_T_O2_Legende1.bmp"
bmp(TEMP_O2_LEG1_PFAD , width = 500, height = 500)
   plot(O2$Temp[1:26], O2$O2[1:26], main = "Weil am Rhein",
      xlim = c(0,30), ylim = c(0,15), pch = 17, col = "blue",
      xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]")
   points(O2$Temp[27:51], O2$O2[27:51],
      pch = 18, col = "red", cex = 2)
   legend(x = 25, y = 15, bty = "n", c("2003","2004"), col = c("blue","red"),
      pch = c(17,18), pt.cex = c(1,2))
dev.off()
```

- Die Grafik sieht wie folgt aus:



Anstatt der x- und y-Koordinaten können Sie direkt die Lage der Legende angeben:

```
TEMP_O2_LEG2_PFAD <- "E:/R/temp/Weil_T_O2_Legende2.bmp"
bmp(TEMP_O2_LEG2_PFAD , width = 500, height = 500)
   plot(O2$Temp[1:26], O2$O2[1:26], main = "Weil am Rhein",
      xlim = c(0,30), ylim = c(0,15), pch = 17, col = "blue",
      xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]")
   points(O2$Temp[27:51], O2$O2[27:51],
      pch = 18, col = "red", cex = 2)
   legend("topright", bty = "n", c("2003","2004"), col = c("blue","red"),
      pch = c(17,18), pt.cex = c(1,2))
dev.off()
```

Linien

- Mit der *points*-Funktion können Sie Daten auch als Linien hinzufügen.

```
TEMP_O2_LIN_PFAD <- "E:/R/temp/Weil_T_O2_Linien.bmp"
bmp(TEMP_O2_LIN_PFAD, width = 500, height = 500)
   plot(O2$Temp[1:26], O2$O2[1:26], main = "Weil am Rhein",
      xlim = c(0,30), ylim = c(0,15), pch = 17, col = "blue",
      xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]")
   points (O2$Temp[27:51], O2$O2[27:51],
      type = "l", lty = 2, lwd = 1.5, col = "magenta")
dev.off()
```

Übungsaufgabe 6.3

1) Erstellen Sie eine Grafik *Weil_T_Q_Jahr.bmp,* welche Temperatur (°C) und Abfluss (m^3/s) des Dataframes *O2* nach den Jahren 2003 (erste 26 Werte) und 2004 (Rest) unterteilt. Wählen Sie als Symbol für das Jahr 2003 braune Kreise (nicht ausgefüllt), für 2004 blaue X-Zeichen. Stimmen Sie die Grösse der Symbole sinnvoll aufeinander ab, und fügen Sie eine Legende hinzu.

Lösungen zu Übungsaufgabe 6.3

1)

```
TEMP_Q_JAHR_PFAD <- "E:/R/temp/Weil_T_Q_Jahr.bmp"
bmp(TEMP_Q_JAHR_PFAD, width = 500, height = 500)
   plot(O2$Temp, O2$Abfluss,
      main = "Weil am Rhein", xlim = c(0, 30), ylim = c(0, 2000),
      xlab = "Temperatur [°C]", ylab = "Abfluss [m³/s]", type = "n")
   points(O2$Temp[1:26], O2$Abfluss[1:26],
      pch = 1, col = "brown", cex = 2)
   points(O2$Temp[27:51], O2$Abfluss[27:51],
      pch = 4, col = "darkgreen", cex = 1.5)
   legend("topright", bty = "n", c("2003","2004"),
      col = c("brown","darkgreen"), pch = c(1,4), pt.cex = c(2,1.5))
dev.off()
```

Logarithmische Achsen

Wenn sich die Werte der Datenpunkte um Grössenordnungen unterscheiden, können logarithmische Darstellungen hilfreich sein. In R geben Sie dazu mit dem Parameter *log* die logarithisch darzustellende/n Achse/n an. Mit *log = "xy"* werden beide Achsen logarithmisch dargestellt. Bei logarithmischen Darstellungen können die plot-Parameter *xlim* und *ylim* nicht verwendet werden. Achten Sie deshalb darauf, dass Sie den *Plot*-Bereich so wählen, dass alle Daten dargestellt werden!

- Stellen Sie die Grafik aus Übungsaufgabe 6.3 mit logarithmischer y-Achse dar:

```
TEMP_Q_YLOG_PFAD <- "E:/R/temp/Weil_T_Q_ylog.bmp"

bmp(TEMP_Q_YLOG_PFAD, width = 500, height = 500)
   plot(O2$Temp, O2$Abfluss, log = "y", main = "Weil am Rhein",
      xlab = "Temperatur [°C]", ylab = "Abfluss [m³/s]", type = "n")
   points (O2$Temp[1:26], O2$Abfluss[1:26],
      pch = 1, col = "brown", cex = 2)
   points (O2$Temp[27:51], O2$Abfluss[27:51],
      pch = 4, col = "darkgreen", cex = 1.5)
   legend("topright", bty = "n", c("2003","2004"),
      col = c("brown","darkgreen"), pch = c(1,4), pt.cex = c(2,1.5))
dev.off()
```

6.2. Analyse einer Datenreihe

In diesem Kapitel lernen Sie, wie Sie in R eine Datenreihe mit Berechnungen oder Grafiken analysieren können. Mit den folgenden Funktionen können Sie unter anderem Lage oder Streuung der Daten berechnen:

Funktion	*Abkürzung*
mean()	Artithmetischer Mittelwert
median()	Median
sd()	Standardabweichung
var()	Varianz
min()	Minimum
max()	Maximum
range()	Spannweite (Minimum und Maximum)
quantile()	Quantile (die Quantile werden mit *probs* festgelegt)
summary()	Gibt eine Zusammenfassung der Daten aus (Minimum, 25%-Quantil, Median, Mittelwert, 75%-Quantil, Maximum)

Im folgenden Beispiel wenden Sie diese Funktionen auf maximale Jahresabflüsse an. Im deutschsprachigen Raum sind diese nicht öffentlich publiziert; zudem sind die Zeitreihen oft durch Gewässerregulierung beeinflusst. Deshalb werden in den folgenden Beispielen Daten kanadischer Flüsse verwendet.

- Importieren Sie die Datei *Nass_River.csv,* welche die maximalen Jahresabflüsse des Nass River (Kanada) enthält, zu einem Dataframe *Nass*.
- Versuchen Sie die folgenden Eingaben zu verstehen:

```
mean(Nass$Abfluss)
median(Nass$Abfluss)
sd(Nass$Abfluss)
var(Nass$Abfluss)
min(Nass$Abfluss); max(Nass$Abfluss)
range(Nass$Abfluss)
quantile(Nass$Abfluss, probs = c(0.01, 0.25, 0.5, 0.75, 0.99))
summary(Nass$Abfluss)
```

L-Momente

Hosking (1990) stellt eine Methode vor, wie Lage, Streuung, Schiefe und Kurtosis (=Gipfligkeit) mit linearen Momenten, so genannten L-Momenten beschrieben werden können. In R verwenden Sie dazu das Paket *Lmoments*.

- Importieren und laden Sie das Paket *Lmoments*.
- Um die ersten vier L-Momente zu berechnen, geben Sie Folgendes ein:

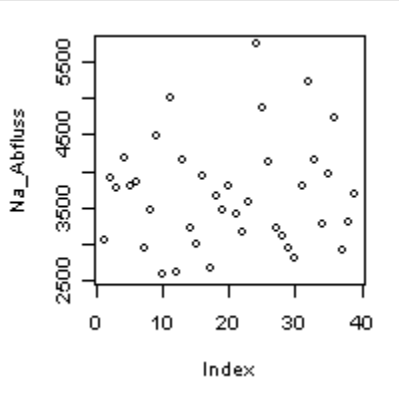
```
Lmoments(Nass$Abfluss, rmax=4)
```

Index-Grafik

Um einen ersten Eindruck der Daten zu bekommen, erstellen Sie am Besten eine Index-Grafik: Die Werte (y-Achse) werden gegen ihren Index (x-Achse) dargestellt. In einer Index-Grafik können Sie eventuelle Trends der Datenwerte anhand ihrer Reihenfolge erkennen.

- Erstellen Sie die folgende Index-Grafik:

```
INDEX_PFAD <- "E:/R/temp/Index_Grafik.bmp"
bmp(INDEX_PFAD, width = 500, height = 500)
        plot(Nass$Abfluss)
dev.off()
```



Empirische Verteilung

Bei einer empirischen Verteilung wird jedem Datenpunkt eine empirische Unterschreitungswahrscheinlichkeit F_{emp} zugeordnet. In diesem Leitprogramm wird dazu die Formel von Weibull verwendet: $F_{emp} = \frac{i}{n+1}$; i = Index des Datenelements, n = Anzahl Werte der Datenreihe.

- Berechnen Sie die Anzahl Werte der Datenreihe:

```
n <- length(Nass$Abfluss)
```

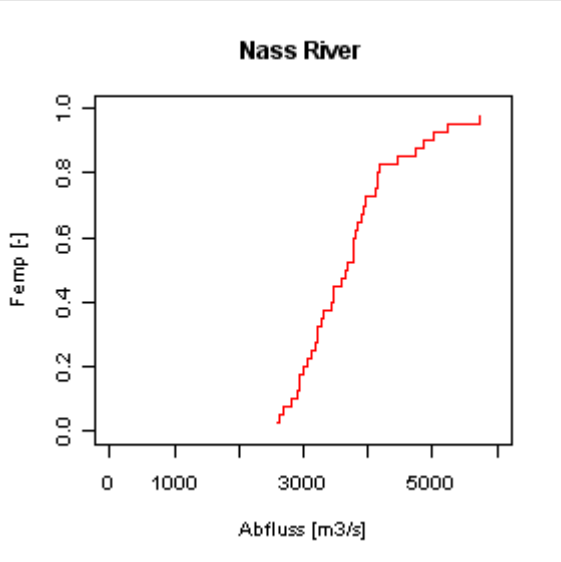
- Bestimmen Sie die empirische Unterschreitungswahrscheinlichkeit nach Weibull:

```
Femp <- 1:n/(n+1)
```

- Erstellen Sie eine Grafik der empirischen Verteilung. Dabei müssen Sie die Abflusswerte nach ihrer Grösse aufsteigend sortieren!

```
FEMP_PFAD <- "E:/R/temp/Empirische_Verteilung.bmp"
bmp(FEMP_PFAD, width = 500, height = 500)
   plot(sort(Nass$Abfluss), Femp, main = "Nass River",
      xlim = c( 0, 6000 ), ylim = c( 0, 1 ), xlab = "Abfluss [m³/s]",
      ylab = "Femp [-]", type = "s", col = "red")
dev.off()
```

- Die resultierende Grafik sieht wie folgt aus:



- Betrachten Sie die neue Grafik: Je steiler die Kurve, desto mehr Abflussdaten befinden sich in diesem Wertebereich. Schätzen Sie den Median (F_{emp} = 0.5) und das 90%-Quantil (F_{emp} = 0.9) und überprüfen Sie ihre Schätzung mit R.

Histogramm

- Erstellen Sie ein einfaches Histogramm:

```
HIST1_PFAD <- "E:/R/temp/Nass_Histogramm1.bmp"
bmp(HIST1_PFAD, width = 500, height = 500)
   hist(Nass$Abfluss)
dev.off()
```

- Verschönern Sie das Histogramm (Klassengrenzen: *breaks*).

```
HIST2_PFAD <- "E:/R/temp/Nass_Histogramm2.bmp"
bmp(HIST2_PFAD, width = 500, height = 500)
   par(mar = c(5.1,4.1,1.5,1.5))
   hist(Nass$Abfluss, main = "Nass River", breaks = seq(0,6000,500),
      xlim = c(0,6000), ylim = c(0,12), col = "red",
      xlab = "Abfluss[m³/s]", ylab = "Häufigkeit[-]")
dev.off()
```

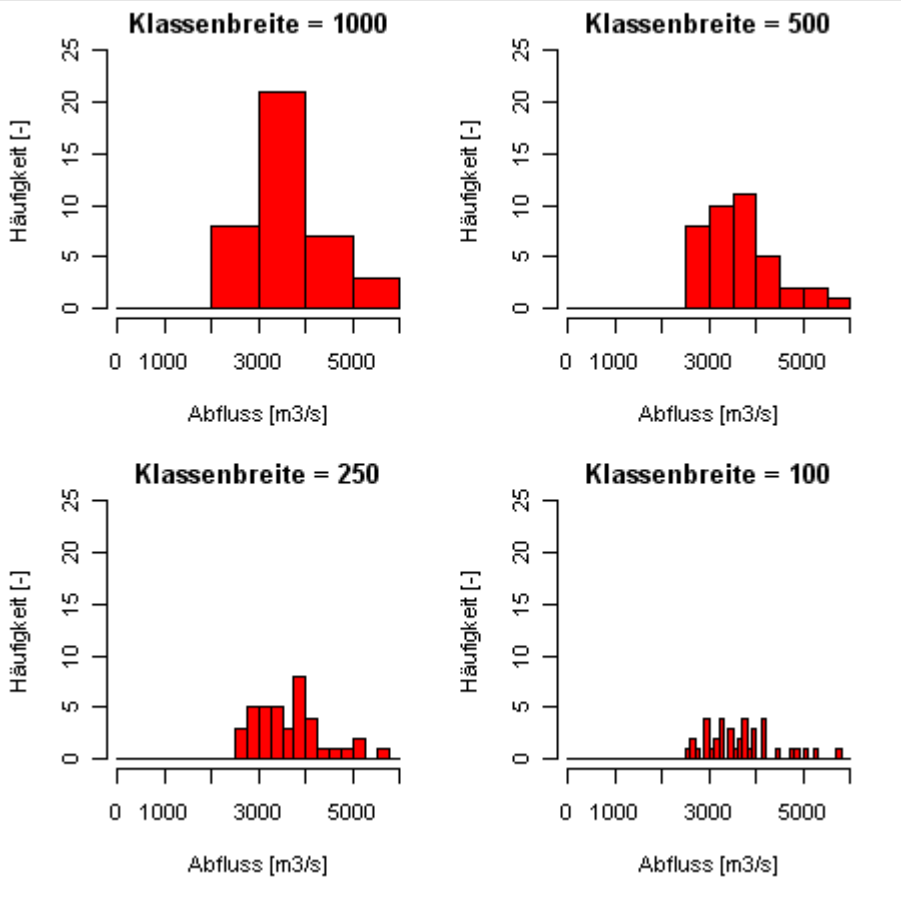
Übungsaufgabe 6.4

1) Erstellen Sie eine Grafik, welche vier Histogramme mit den Klassenbreiten 1000, 500, 250 und 100 enthält. Ordnen Sie die vier Grafiken in 2 Zeilen und 2 Spalten an. Hinweis: Verwenden Sie den bereits erstellten Code, aber passen Sie unter anderem den Parameter *ylim* an. Wählen Sie sinnvolle Grafik-Überschriften, Streckungsfaktoren und *Device*-Grösse. Lassen Sie sich nicht von einem langen Code abschrecken!

Lösungen zu Übungsaufgabe 6.4

1)

```
VIER_HIST_PFAD <- "E:/R/temp/Nass_4_Histogramme.bmp"
bmp(VIER_HIST_PFAD, width = 700, height = 700)
   par(mfrow = c(2,2), mar = c(5.1, 4.1, 1.5, 1.5), cex = 1.3)
   hist(Nass$Abfluss, main = "Klassenbreite = 1000",
      breaks = seq(0,6000,1000), xlim = c(0,6000), ylim = c(0,25),
      xlab = "Abfluss [m³/s]", ylab = "Häufigkeit [-]", col = "red")
   hist(Nass$Abfluss, main = "Klassenbreite = 500",
      breaks = seq(0,6000,500), xlim = c(0,6000), ylim = c(0,25),
      xlab = "Abfluss [m³/s]", ylab = "Häufigkeit [-]", col = "red")
   hist(Nass$Abfluss, main = "Klassenbreite = 250",
      breaks = seq(0,6000,250), xlim = c(0,6000), ylim = c(0,25),
      xlab = "Abfluss [m³/s]", ylab = "Häufigkeit [-]", col = "red")
   hist(Nass$Abfluss, main = "Klassenbreite = 100",
   breaks = seq(0,6000,100), xlim = c(0,6000), ylim = c(0,25),
   xlab = "Abfluss [m³/s]", ylab = "Häufigkeit [-]", col = "red")
dev.off()
```



Fazit: Aussage von Histogrammen hängt stark von deren Klassegrösse ab. Deshalb ist die Darstellung der empirischen Verteilung besser als ein Histogramm!

Boxplot

Boxplots sind Standard-Darstellungen, um Lage und Streuung einer Datenreihe visuell zu analysieren. In R sind die Voreinstellungen so gewählt, dass die „Schnauzhaare“ des Boxplots bis zum extremsten Datenpunkt ausserhalb der Box, aber nicht weiter als 1.5 mal den Interquartil-Abstand (75%-Quantil minus 25%-Quantil) von der Box entfernt reichen. Die vollständige Liste der Argumente der *boxplot*-Funktion erhalten Sie mit *?boxplot*.

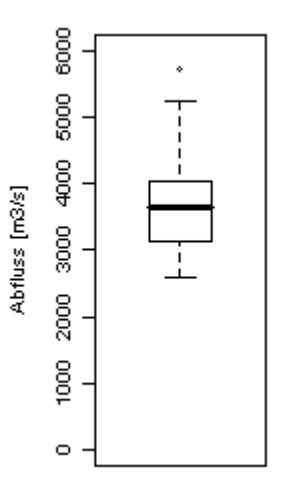
- Erstellen Sie einen einfachen Boxplot:

```
BOX1_PFAD <- "E:/R/temp/Nass_Boxplot1.bmp"
bmp(BOX1_PFAD, width = 300, height = 500)
   boxplot(Nass$Abfluss)
dev.off()
```

- Verschönern Sie nun diese Grafik!

```
BOX2_PFAD <- "E:/R/temp/Nass_Boxplot2.bmp"
bmp(BOX2_PFAD, width = 300, height = 500)
        par(mar = c(1.5, 4.1, 1.5, 1.5))
        boxplot(Nass$Abfluss, horizontal = F,
           ylim = c(0,6000), ylab = "Abfluss [m³/s]")
dev.off()
```

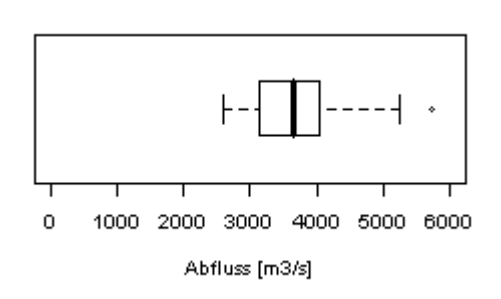
- Die Grafik sollte wie folgt aussehen:



- Vergleichen Sie die erstellte Grafik mit den vorher berechneten Werten.
- Erstellen Sie einen horizontal ausgerichteten Boxplot! Vorsicht: *xlab* aber nicht *xlim,* sondern *ylim* wird verwendet!

```
BOX3_PFAD <- "E:/R/temp/Nass_Boxplot3.bmp"
bmp(BOX3_PFAD, width = 500, height = 300)
        par(mar = c(5.1, 1.5, 1.5, 1.5))
        boxplot(Nass$Abfluss, horizontal = T,
                ylim = c(0,6000), xlab = "Abfluss [m³/s]")
dev.off()
```

- Der horizontale Boxplot sieht wie folgt aus:



6.3. Analyse einer Datenreihe mit Faktoren

Den Elementen einer Datenreihe können Faktoren zugeordnet sein. Ein Beispiel eines Faktors ist die Komponente *Eintrag* der Datei *Phosphor_1996.csv*. Im Kapitel 5.7 haben Sie bereits die *tapply*-Funktion kennen gelernt, mit der Sie nach Faktoren analysieren können.

- Importieren Sie die Datei Phosphor_1996.csv:

```
P_PFAD <- "E:/R/3_Daten/Phosphor_1996.csv"
P <- read.table(P_PFAD, sep =";", dec=",", header=T)
names(P); dim(P)
head(P)
```

- Um diese Datei nach Faktoren zu analysieren, können Sie mit der *tapply*-Funktion die Analyse-Funktionen aus Kapitel 6.2 verwenden. Je nachdem, ob die Analyse-Funktion zusammenfassend oder elementweise-rechnend ist, entsteht ein Resultat vom Datentyp *numeric* oder *list*.

```
P_CH_Summe <- tapply(P$Schweiz, P$Eintrag, sum)
P_CH_Quantile <- tapply(P$Schweiz, P$Eintrag, quantile)
```

Kuchendiagramm

- Mit der *pie*-Funktion (Englisch: Kuchen) können Sie Kuchendiagramme erstellen. Geben Sie Folgendes ein:
-

```
P_CH_KUCHEN_PFAD <- "E:/R/temp/P_CH_Kuchen.bmp"
   bmp(P_CH_KUCHEN_PFAD, width=500, height=500)
   pie(P_CH_Summe, main = "P-Eintrag der Schweiz", radius = 0.8,
      labels = names(P_CH_Summe), col = c("brown","green","orange"))
dev.off()
```

Übungsaufgabe 6.5

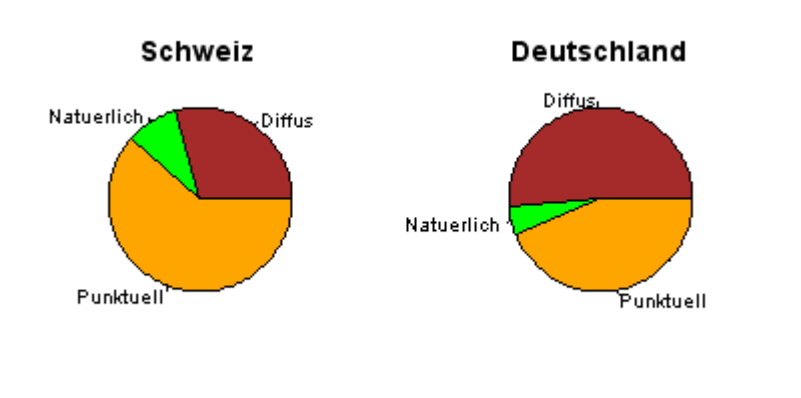
1) Vergleichen Sie mit zwei Kuchendiagrammen die Phosphoreinträge im Jahr 1996 der Schweiz (Diagramm 1) und Deutschland (Diagramm 2) anhand deren Anteile des Eintrags (*natürlich*, *punktuell*, *diffus*). Stellen Sie die beiden Diagramme in einer Mehrfachgrafik dar.

Lösungen zu Übungsaufgabe 6.5

1) Die Grafik kann mit dem folgenden Code erstellt werden:

```
P_CH_Summe <- tapply(P$Schweiz, P$Eintrag, sum)
P_D_Summe <- tapply(P$Deutschland, P$Eintrag, sum)

P_CH_D_KUCHEN_PFAD <- "E:/R/temp/P_CH_D_Kuchen.bmp"
   bmp(P_CH_D_KUCHEN_PFAD,width=600,height=300)
   par(mfrow = c(1,2), mar = c(3.5,3.5,3.5,3.5))
   pie(P_CH_Summe, main = "Schweiz", labels = names(P_CH_Summe),
      col = c("brown","green","orange"), radius = 1)
   pie(P_D_Summe,
   main = "Deutschland", labels = names(P_D_Summe),
      col = c("brown","green","orange"), radius = 1)
dev.off()
```



6.4. Vergleich zweier Datenreihen

Korrelationskoeffizient

Der Korrelationskoeffizient ist eine dimensionslose Zahl, welche beschreibt, ob ein linearer Zusammenhang zwischen den Elementen zweier Datenreihen besteht. Der Korrelationskoeffizient r ist wie folgt definiert: $r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)\cdot s_x \cdot s_y}$

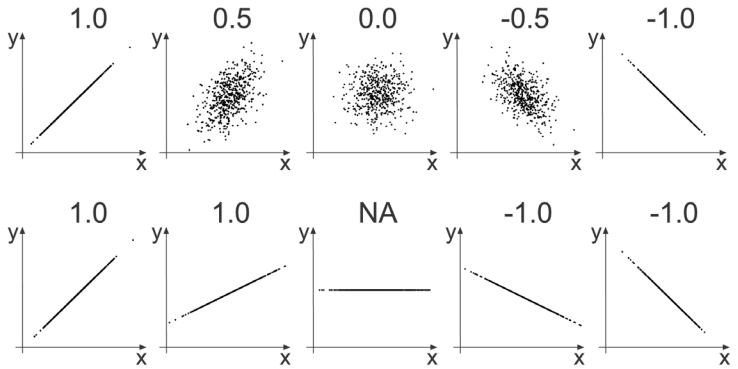
n = Anzahl Elemente pro Datenreihe

x_i und y_i = Elemente der Datenreihen

$\bar{x}$ und $\bar{y}$ = Mittelwerte der Datenreihen

s_x und s_y = Standardabweichungen der Datenreihen

Anhand dieser Formel wird ersichtlich, dass der Korrelationskoeffizient für Datenreihen mit gleichen x- oder y-Werten nicht definiert ist, weil deren Standardabweichung null ist. In der untenstehenden Abbildung sind die Korrelationskoeffizienten einiger Datenbeispiele dargestellt.



- Berechnen Sie für das Dataframe *O2* den Korrelationskoeffizienten zwischen der Sauerstoffkonzentration und der Temperatur:
  cor(O2$O2, O2$Temp)

In R können Sie nicht bloss den Korrelationskoeffizienten zwischen zwei Komponenten berechnen, sondern auf einfache Art eine Korrelationskoeffizienten-Matrix ausgeben lassen, welche die Korrelationskoeffizienten aller möglichen Vergleiche beinhaltet.

- Definieren Sie ein neues Dataframe *O2_Neu* ohne Datums- und Jahresspalten. Erstellen Sie dafür die Korrelationskoeffizienten-Matrix:
  O2_Neu <- data.frame(O2[,-(1:2)])
  cor(O2_Neu)

Um einen ersten Eindruck möglicher Korrelationen zu erhalten, lohnt es sich die zu untersuchenden Datenreihen gegeneinander darzustellen. Wie in R zwei Datenreihen gegeneinander aufgetragen werden, haben Sie im allgemeinen Kapitel über Grafiken (Kapitel 6.1) bereits kennen gelernt. Ein Spezialfall davon ist die Darstellung von Zeitreihen.

Darstellung von Zeitreihen: *Date*-Format

- Fügen Sie Datum und Jahr zusammen und erstellen Sie daraus einen Vektor im *Date*-Format:

```
W_Datum1 <- paste(O2$Datum, O2$Jahr, sep = "")
W_Datum2 <- as.Date(W_Datum1, "%d.%m.%Y")
```

- Legen Sie den Beginn und das Ende Ihrer Zeitreiche fest. Diese beiden Objekte werden zum Beschriften der x-Achse verwendet.

```
Beginn <- as.Date("1.1.2003", "%d.%m.%Y")
Ende <- as.Date("1.1.2005", "%d.%m.%Y")
```

Oder:

```
Beginn <- min (W_Datum2)
Ende <- max (W_Datum2)
```

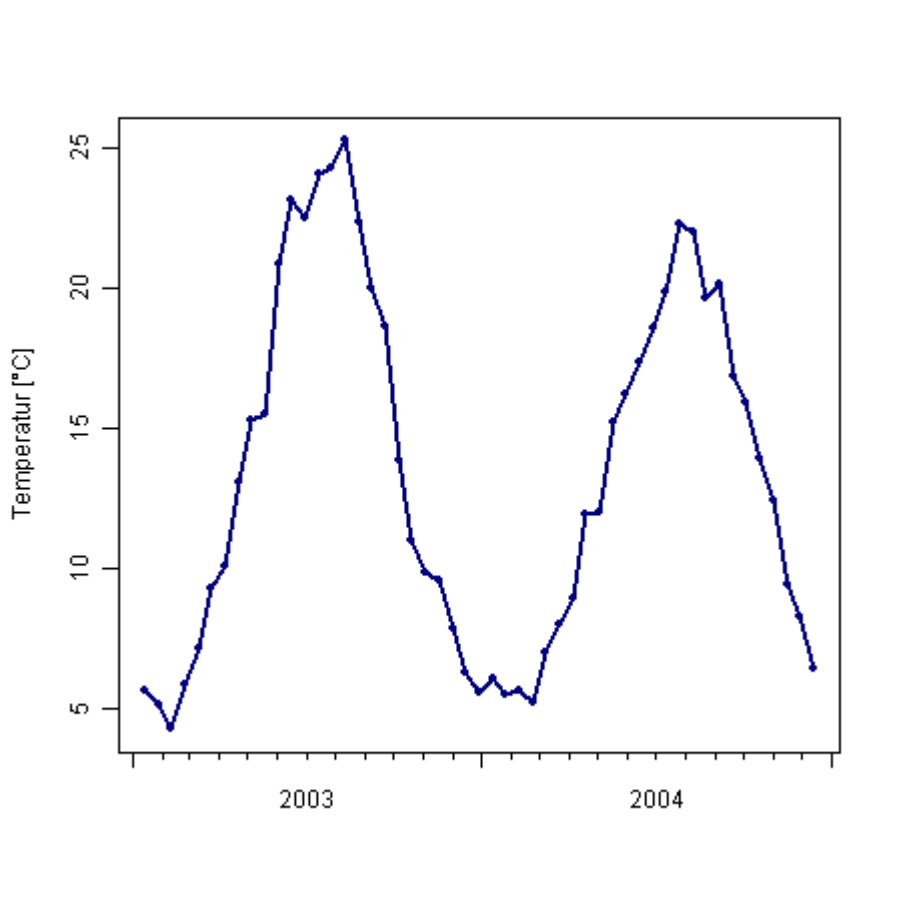
- Erstellen Sie die Zeitreihen-Grafik! Wählen Sie in der *plot*-Funktion *xaxt = "n"*, damit die x-Achse nicht erzeugt wird. Diese fügen Sie mit der low-level Grafik *axis.Date* hinzu. Mit dem Argument *at* wählen Sie den Beginn, das Ende und den Abstand zwischen den einzelnen Elementen der Achsenbeschriftung.

```
W_T_DATUM_PFAD <- "E:/R/temp/W_T_Datum.bmp"
bmp(W_T_DATUM_PFAD, width = 500, height = 500)
   plot(W_Datum2, O2$Temp, xaxt="n", type = "b")
   axis.Date(1, at = seq(Beginn, Ende, "years"), format = "%d.%m.%y")
dev.off()
```

Hinweis: Das Argument *at* muss eine Datumssequenz im Date-Format enthalten. Um den Abstand festzulegen, können Sie entweder die Ausdrücke *"years"*, *"months"*, *"days"* oder eine Zahl (=Anzahl Tage) wählen.

- Verschönern Sie Ihre Grafik mit dem folgenden Code:

```
W_T_DATUM1_PFAD <- "E:/R/temp/W_T_Datum1.bmp"
bmp(W_T_DATUM1_PFAD, width = 500, height = 500)
   plot(W_Datum2, O2$Temp, xaxt="n", type = "l", lwd = 2,
      col = "navy", ylab = "Temperatur [°C]")
   points(W_Datum2, O2$Temp, type = "p", pch = 16,
      col = "navy", cex = 0.7)
   axis.Date(1, at=seq(Beginn, Ende, "years"), tcl = -0.5, labels = F)
   axis.Date(1, at=seq(Beginn, Ende, "months"), tcl = -0.2, labels = F)
   axis.Date(1, at=seq(Beginn+365/2, Ende-366/2, "years"), tcl = 0,
      labels = T, format = "%Y")
dev.off()
```



Darstellung von Zeitreihen: *POSIXct*-Format

Zeitreihen mit *POSIXct*-Formaten werden analog dem *Date*-Format dargestellt.

- Importieren Sie die Datei *Rheinfelden_Temp2.csv* als *Dataframe*. Sie enthält die Wassertemperaturen des Rheins bei Rheinfelden vom 27. bis 30. Juli 2008.

```
RH_T_PFAD <- "E:/R/3_Daten/Rheinfelden_Temp2.csv"
Rh <- read.table(RH_T_PFAD, sep = ";", dec = ",", header = T)
names(Rh); dim(Rh)
head(Rh)
```

- Erstellen Sie einen Vektor *Rhein_Zeit2*, der Uhrzeit und Datum im *POSIXct*-Format enthält.

```
Rhein_Zeit1 <- paste(Rh$Datum, Rh$Uhrzeit, sep = " ")
Rhein_Zeit2 <- as.POSIXct(strptime( Rhein_Zeit1, format =
   "%d.%m.%Y %H:%M", tz="GMT"), tz="GMT")
```

- Wählen Sie den Beginn und das Ende der Zeitreihe. Das benötigen Sie beim Beschriften der x-Achse. Die beiden Werte werden im Format *POSIXct* benötigt.

```
Beginn <- min(Rhein_Zeit2)
Ende <- max(Rhein_Zeit2)
```

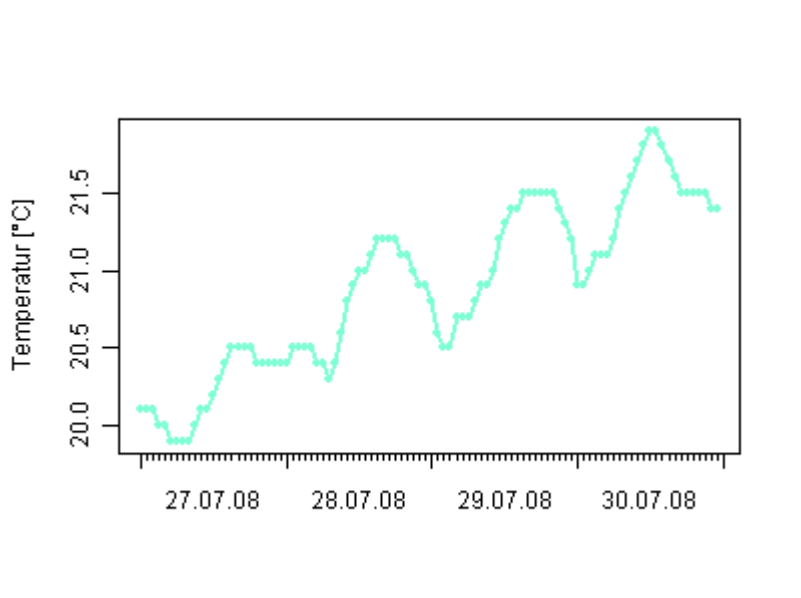
- Erstellen Sie die Grafik mit den folgenden Befehlen:

```
TEMP_ZEIT_PFAD <- "E:/R/temp/Rhein_Temp_Zeit.bmp"
bmp(TEMP_ZEIT_PFAD, width = 500, height = 500)
   plot(Rhein_Zeit2, Rh$Temp, xaxt="n", type = "b")
   axis.POSIXct(1, at = seq(Beginn, Ende, "days"),
      format = "%d.%m. %H:%M")
dev.off()
```

Hinweis: Das Argument *at* muss eine Datumssequenz im Date-Format enthalten. Als Abstand des Argumentes *at* können Sie unter anderem *"years"*, *"months"*, *"days"*, *"hours"*, *"min"*, *"sec"* oder eine Zahl (=Anzahl Sekunden) wählen.

- Verschönern Sie Ihre Grafik mit dem folgenden Code:

```
TEMP_ZEIT1_PFAD <- "E:/R/temp/Rhein_Temp_Zeit1.bmp"
bmp(TEMP_ZEIT1_PFAD, width = 600, height = 400)
   plot(Rhein_Zeit2, Rh$Temp, xaxt="n", type = "l",
      ylab = "Temperatur [°C]", lwd = 2, col = "aquamarine")
   points(Rhein_Zeit2, Rh$Temp, type = "p", pch = 16,
      col = "aquamarine", cex = 0.7)
   axis.POSIXct(1, at=seq(Beginn, Ende+3600, "days"),
      tcl = -0.5, labels = F)
   axis.POSIXct(1, at=seq(Beginn, Ende, 3600), #oder "hours"
      tcl = -0.2, labels = F)
   axis.POSIXct(1, at=seq(Beginn+(3600*24)/2, Ende+(3600*24)/2,
      "days"), tcl = 0, labels = T, format = "%d.%m.%y")
dev.off()
```



Übungsaufgabe 6.6

1) Importieren Sie die Datei Lindau_Pegel.csv, welche Daten des Pegels des Bodensees vom 27. bis 30. Juli 2008 in Stundenwerten enthält. Stellen Sie in einer Grafik den Pegelstand gegen die Zeit mit Linien (ohne Punkte) dar.

Lösungen zu Übungsaufgabe 6.6

1)
```
LI_PFAD <- "E:/R/3_Daten/Lindau_Pegel.csv"
Li <- read.table(LI_PFAD, sep = ";", dec = ",", header = T)
names(Li); dim(Li)
head(Li)

Li_Zeit1 <- paste(Li$Datum, Li$Uhrzeit, sep = " ")
Li_Zeit2  <- as.POSIXct( strptime( Li_Zeit1 , format =
   "%d.%m.%Y %H:%M", tz="GMT"),tz="GMT")

Beginn <- min(Li_Zeit2)
Ende   <- max(Li_Zeit2)

LIN_PFAD <- "E:/R/temp/Lindau_Pegel_Zeit.bmp"
bmp(LIN_PFAD, width = 500, height = 500)
   plot(Li_Zeit2, Li$Pegel, xaxt="n", type = "l")
   axis.POSIXct(1, at=seq(Beginn, Ende, "days"),
      format = "%d.%m. %H:%M")
dev.off()
```

6.5. Merkpunkte

Nun sind Sie in der Lage, Daten anhand von Grafiken oder Funktionen zu beschreiben. Repetieren Sie die wichtigsten Merkpunkte des sechsten Kapitels:

Devices	• *windows* (für Bildschirmausgabe), *bmp*, *jpeg*, *pdf*.
	• *dev.off()*.
plot-Funktion	• *plot (x, y)*. Weitere Argumente: *col* (Farbe), *bg* (Hintergrund), *main* (Überschrift), *pch* (Punktzeichen), *type* (Grafiktyp: "p", "l", "b", "s", "n"), *xlab*/*ylab* (Achsenbeschriftung), *xlim*/*ylim* (Grenzen der Achsen).
Farben	• *palette()*, *colors()*.
par-Funktion	• *cex* (Streckungsfaktor), *font* (Schriftart), *lty* (Linientyp), *log* (logarithmische Achsen), *lwd* (Liniendicke), *mar* (Rand), *mex* (Streckung des Rands), *mfrow* (Anordnung der Grafiken), *oma* (Aussenrand), *tcl* (Länge der Achsenmarkierungen), *xaxt*/*yaxt* (Zeichnen einer Achse).
Spezifische Parameter	• Für *cex* und *font*: *.axis* (Achsenzahlen), *.lab* (Achsenbeschriftung), *.main* (Überschrift).
Low-level Grafiken	• *points*-Funktion.
	• *legend*-Funktion mit den Argumenten bty (Legendenumrandung), pt.cex (Streckung einzelner Symbole), x (x-Koordinate der linken oberen Ecke, oder Lage in Worten beschreiben), y (y-Koordinate der linken oberen Ecke).
Analyse-Funktionen (eine Datenreihe)	• *mean* (Mittelwert), *median* (Median), *sd* (Standardabweichung), *var* (Varianz), *min* (Minimum), *max* (Maximum), *range* (Spannweite), *quantile* (Quantile), *summary* (Zusammenfassung).
L-Momente	• Paket *Lmoments*; lmoments (x, rmax = 4).
Grafiken	• Index Grafik: plot(x).
	• Empirische Verteilung: plot (sort(x), Femp, type = "s").
	• Histogramm: hist(x, breaks).
	• Boxplot: boxplot(x).
Vergleich von zwei Datenreihen	• Korrelationskoeffizient (ein Wert oder Matrix): cor(x).

6.6. Lernkontrolle

Falls Sie sich mit dem Lernstoff dieses Kapitels genügend vertraut fühlen, beginnen Sie nun mit der Lernkontrolle! Wie bei den vorangehenden Kapiteln, sollten Sie die Lösungen erst am Schluss verwenden, um Ihre Antworten zu korrigieren.

Fragen

1) Importieren Sie die Datei *Weil_OC.csv* zu einem Dataframe *W_OC*. Die Datei enthält die Konzentrationen an gelöstem (DOC) und gesamtem Kohlenstoff (TOC) des Rheins in Weil am Rhein in den Jahren 2001 und 2002. Die Einheit von DOC und TOC ist mg/l. Berechnen Sie den Variationskoeffizienten für DOC und TOC. Hinweis: Der Variationskoeffizient (CV) ist definiert als die Standardabweichung geteilt durch den Mittelwert. Welcher Variationskoeffizient ist grösser?
2) Lassen Sie sich für DOC und TOC eine Zusammenfassung (Mittelwert, Median, Minimum, Maximum, 25% und 75%-Quantile) ausgeben. Welche Befehle verwenden Sie?
3) Erzeugen Sie für TOC einen vertikalen Boxplot und einen Indexplot. Ordnen Sie die beiden in einer Grafik *W_TOC_Boxplot_Indexplot.bmp* wie folgt an:

Boxplot	Indexplot

   Die erstellte Grafik soll 800 Pixel breit und 400 Pixel hoch sein. Zudem soll bei beiden Darstellungen die y-Achse von 0 bis 7 mg/l gehen, und mit "TOC [mg/l]" beschriftet werden. Welche Befehle verwenden Sie?
4) Berechnen Sie den Korrelationskoeffizienten zwischen DOC und TOC. Welchen Wert erhalten Sie?

Die Lösungen zur Lernkontrolle finden Sie im Kapitel 8.

6.7. Kapiteltest

Falls Sie sich mit den Themen dieses Kapitels sicher fühlen, melden Sie sich bei der Lehrperson zum Kapiteltest. Der Test zu diesem Kapitel verläuft gleich die vorangehenden Tests. Für den Kapiteltest stehen Ihnen maximal 20 Minuten zur Verfügung.

6.8. Zusatzaufgaben

Falls Sie dieses Kapitel wesentlich früher als andere Studierende durch gearbeitet haben, können Sie sich mit den folgenden Zusatzaufgaben befassen, bevor Sie mit dem Kapitel 7 beginnen. Oder nutzen Sie die Zeit, in der die Lehrperson Ihren Kapiteltest korrigiert, um Zusatzaufgaben zu bearbeiten!

1) Schauen Sie sich weitere Grafikmöglichkeiten an! Auf den folgenden Webpages sind R-Codes der entsprechenden Grafiken aufgeführt.
   http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html
   http://addictedtor.free.fr/graphiques/thumbs.php
   http://bm2.genes.nig.ac.jp/RGM2/index.php?pageID=1
2) Mit welchem Parameter können Sie einen quadratischen Plot-Bereich bewirken?
3) Stellen Sie für die Datei *Weil_O2.csv* den Verlauf der Temperatur [°C] und der Sauerstoffkonzentration [mg/l] über die Zeit dar. Versuchen Sie, für die Sauerstoffkonzentration eine Sekundärachse einzuführen. Tipp: Verwenden Sie die *axis*-Funktion, und skalieren Sie die Sekundärachse auf die Primärachse. Auf der Webseite von Paul Murrell (erstgenannter Link von Zusatzaufgabe 1) finden Sie in Kapitel 3 ein Beispiel dazu (Grafik 3.21).
4) Weshalb kann die *locator*-Funktion beim Erstellen von Legenden hilfreich sein? Finden Sie heraus, wie die Funktion angewendet wird. Tipp: Erstellen Sie eine Grafik im Device *windows* bevor Sie die *locator*-Funktion verwenden.
5) Betrachten Sie im folgenden Code die Parameter *pch* und *col*. Was bewirkt diese Definition? Nachdem Sie eine Vermutung geäussert haben, probieren Sie den Code aus!

```
W_T_O2_JAHR_PFAD <- "E:/R/temp/Weil_Temp_O2_Jahr.bmp"
bmp(W_T_O2_JAHR_PFAD , width = 460, height = 400)
   par(mar = c( 5.1, 4.1, 2.1, 2.1))
   plot(O2$Temp, O2$O2, xlim = c(0,30), ylim = c(0,15),
      xlab = "Temperatur [°C]", ylab = "Gelöster Sauerstoff [mg/l]",
      pch = as.numeric(O2$Jahr)-2002,
      col = palette()[as.numeric(O2$Jahr)-2000])
   legend("topright", bty = "n", c("2003","2004"),
      pch = c(1,2), col = c(palette()[3], palette()[4]))
dev.off()
```

6) Wie können Sie die *barplot*-plot verwenden?

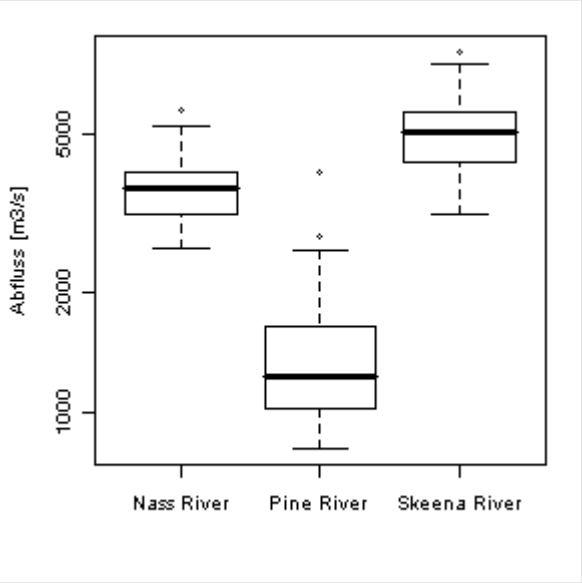
7) Installieren und laden Sie das Paket *fBasics*, und berechnen Sie damit Schiefe (Englisch: *skewness*) und *Kurtosis* von *W_Temp* (siehe vorangehendes Beispiel).

8) Versuchen Sie den folgenden Code verstehen, mit dem die untenstehende Grafik erzeugt wird:

```
NASS_PFAD <- "E:/R/3_Daten/Nass_River.csv"
PINE_PFAD <- "E:/R/3_Daten/Pine_River.csv"
SKEE_PFAD <- "E:/R/3_Daten/Skeena_River.csv"
Nass <- read.table(NASS_PFAD, sep = ";", dec = ",", header = T)
Pine <- read.table(PINE_PFAD, sep = ";", dec = ",", header = T)
Skee <- read.table(SKEE_PFAD, sep = ";", dec = ",", header = T)

Alle_Abfluss = c(Nass$Abfluss, Pine$Abfluss, Skee$Abfluss)
Alle_Namen = c(rep("Nass River",length(Nass$Abfluss)),
   rep("Pine River",length(Pine$Abfluss)),
   rep("Skeena River",length(Skee$Abfluss)))

BOX_VERGLEICH_PFAD <- "E:/R/temp/Boxplot_Vergleich.bmp"
bmp(BOX_VERGLEICH_PFAD, width = 500, height = 500)
   par(mar = c(5.1, 4.1, 1.5, 1.5))
   boxplot(Alle_Abfluss ~ Alle_Namen, log = "y", ylab = "Abfluss [m³/s]")
dev.off()
```



9) Mit der Funktion *image.plot* im Paket *fields* können Sie auf einfache Weise räumlich-verteilte Daten (Grid-Raster) darstellen. Lernen Sie diese kennen.

7. Statistische Modelle

Übersicht

Umweltanalysen werden nicht nur durchgeführt, um gemessene Daten zu beschreiben, sondern auch um Modelle zu erstellen. Modelle haben den Vorteil, dass sie auch für Prognosen verwendet werden können. Zudem helfen Modelle, Zusammenhänge zu verstehen, und komplexe Systeme auf wesentliche Beziehungen zu reduzieren. In diesem Kapitel lernen Sie zwei Modelltypen kennen: Theoretische Verteilungsfunktionen und lineare Regressionsmodelle.

Mit Verteilungsfunktionen werden Häufigkeiten berechnet. Damit können Sie beispielsweise Auftretenshäufigkeiten von Hochwasserereignissen abschätzen. In den Kapiteln 7.1 bis 7.3 lernen Sie, wie Verteilungsfunktionen erstellt und angewendet werden.

Mit linearen Regressionsmodellen versuchen Sie einen mathematischen Zusammenhang zwischen einer schwierig erhältlichen Zielgrösse und einfach verfügbaren erklärenden Variabeln herzustellen. In den Kapiteln 7.4 und 7.5 werden Sie sich zudem mit den Unsicherheiten von linearen Regressionsmodellen auseinandersetzen.

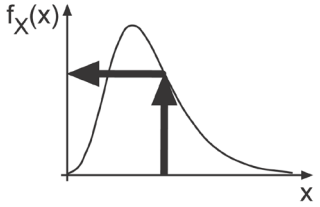
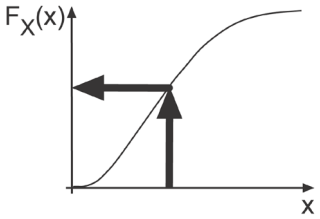
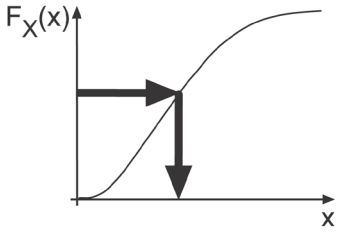
Lernziele

Nachdem Sie dieses Kapitel bearbeitet haben, können Sie

- für eine diskrete Verteilung eine praktische Anwendung nennen, und beschreiben, wie diese in R berechnet wird (K2).
- in R stetige Verteilungen an Daten anpassen, und die Verteilungen testen (K3).
- einfache lineare Regressionsmodelle erstellen (K3).
- die Aussagekraft von Regressionsanalysen beurteilen (K6).

7.1. Verteilungen

In R sind viele Verteilungsfunktionen vorprogrammiert. Dabei sind pro Verteilung jeweils vier Funktionen definiert:

Abkürzung	*Abkürzungserklärung*	*Input/Output*
d_	Resultat: density = Dichte	Input: Werte und Parameter Output: Dichte 
p_	Resultat: probability = Wahrscheinlichkeit	Input: Werte und Parameter Output: Kumulative Wahrscheinlichkeit 
q_	Resultat: quantile = Quantile	Input: Wahrscheinlichkeit und Parameter Output: Werte (= Quantile) 
r_	Resultat: random = zufällig	Input: Anzahl und Parameter Output: Zufällige Werte, die der Verteilung entsprechend verteilt sind. 

Der Funktionsname setzt sich jeweils aus der Abkürzung und der Verteilungsfunktion zusammen. Beispielsweise wird die Dichtefunktion (*d*) der Normalverteilung (*norm*) mit *dnorm* aufgerufen.

In der nächsten Tabelle sind einige Verteilungsfunktionen aufgelistet, die Sie bei Ihrer Arbeit eventuell gebrauchen können. Davon werden in diesem Leitprogramm nur fünf Funktionen verwendet. Die meisten Funktionen befinden sich im *stats*-Paket, das in R automatisch geladen ist.

Funktion	*Paket*	*Beschreibung*	*Typ*	*Leitprogramm?*
_binom()	stats	Binomialverteilung	diskret	Ja
_geom()	stats	Geometrische Verteilung	diskret	Nein
_hyper()	stats	Hypergeometrische Verteilung	diskret	Nein
_multinom()	stats	Multinomiale Verteilung	diskret	Nein
_nbinom()	stats	Negative Binomialverteilung	diskret	Nein
_pois()	stats	Poissonverteilung	diskret	Nein
_beta()	stats	Betaverteilung	stetig	Nein
_chisq()	stats	Chiquadratverteilung	stetig	Nein
_exp()	stats	Exponentialverteilung	stetig	Nein
_f()	stats	F-Verteilung	stetig	Nein
_frechet()	evd	Frechet-Verteilung	stetig	Nein
_gev()	evd	GEV-Verteilung	stetig	Nein
_gumbel()	evd	Gumbelverteilung	stetig	Ja
_lnorm()	stats	Lognormalverteilung	stetig	Ja
_norm()	stats	Normalverteilung	stetig	Ja
_t()	stats	t-Verteilung	stetig	Nein
_unif()	stats	Gleichverteilung	stetig	Ja

7.2. Diskrete Verteilungen

Diskrete Verteilungen werden für Variablen verwendet, die nur bestimmte Werte annehmen. Beispielsweise kann eine Verkehrsampel nur die Zustände *grün*, *orange*, *rot* oder *funktioniert nicht* annehmen. Die Verteilung beschreibt, wie häufig die einzelnen Zustände vorkommen. In diesem Leitprogramm wird von den diskreten Verteilungsfunktionen nur die Binomialverteilung behandelt.

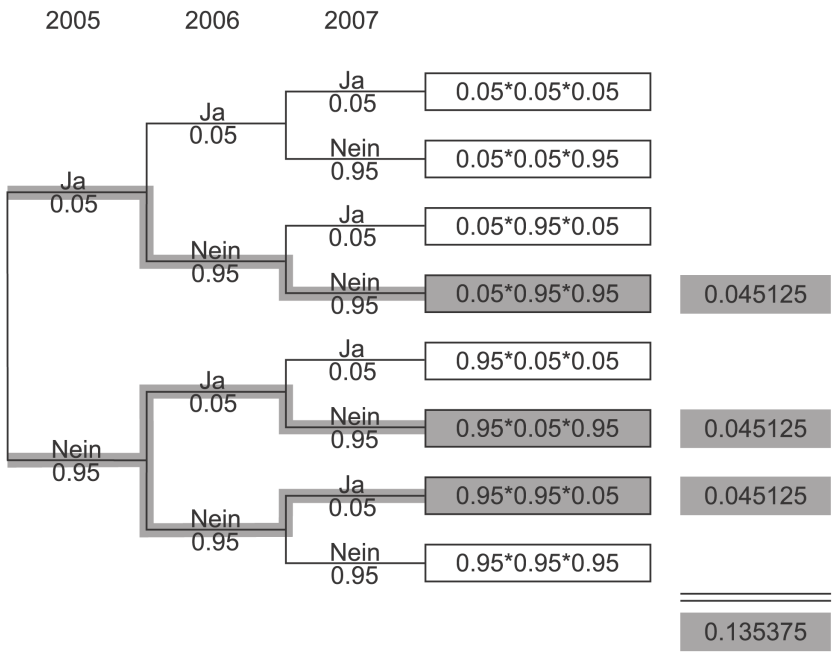
Binomialverteilung

Die Binomialverteilung beschreibt die Häufigkeit von so genannten Bernoulli-Ereignissen. Bernoulli-Ereignisse können bloss zwei Zustände annehmen. Zudem sind die Ereignisse unabhängig. Im folgenden Hochwasser-Beispiel kommen die folgenden zwei Zustände vor: 1) Der Abfluss erreicht oder überschreitet in einem bestimmten Jahr den Wert Q_0 mindestens einmal. 2) Der Abfluss ist in diesem Jahr immer kleiner als Q_0. Dabei wird davon ausgegangen, dass beispielsweise der Zustand im Jahr 2005 nicht vom Zustand des Jahres 2004 abhängt.

- Betrachten Sie als gegeben, dass ein Hochwasser mit einer Jährlichkeit von R = 20 Jahren mit einer Wahrscheinlichkeit von $p = \frac{1}{R} = 0.05$ in einem bestimmten Jahr erreicht oder überschritten wird.
- Berechnen Sie die Wahrscheinlichkeit, dass während drei Jahren (beispielsweise 2005 bis 2007) genau in einem Jahr ein 20-jährliches Hochwasser erreicht oder überschritten wird.

  dbinom(1,3,0.05)

Falls Sie mit der Binominalverteilung nicht vertraut sind, hilft Ihnen die folgende Grafik zu verstehen, was Sie soeben berechnet haben („genau 1-mal *Ja* während 3 Jahren“):



- Berechen Sie die Wahrscheinlichkeit, dass während drei Jahren in keinem Jahr ein 20-jährliches Hochwasser erreicht oder überschritten wird.

  dbinom(0,3,0.05)
- Berechen Sie die Wahrscheinlichkeit, dass während drei Jahren in weniger als zwei Jahren ein 20-jährliches Hochwasser erreicht oder überschritten wird.

  dbinom(0,3,0.05) + dbinom(1,3,0.05)

  Oder mit einer Funktion:

  pbinom(1,3,0.05)

Übungsaufgabe 7.1

1) Berechen Sie die Wahrscheinlichkeit, dass während 10 Jahren in genau einem Jahr ein 100-jährliches Hochwasser erreicht oder überschritten wird.
2) Berechnen Sie die Wahrscheinlichkeit, dass während 10 Jahren mindestens in zwei Jahren ein 100-jährliches Hochwasser erreicht oder überschritten wird.

Lösungen zu Übungsaufgabe 7.1

1) $p = \frac{1}{R} = \frac{1}{100}$

   dbinom(1,10,1/100) = 9.1%
2) In mindestens zwei Jahren

   = in 2, 3, 4, 5, 6, 7, 8, 9 oder 10 Jahren

   = nicht in 0 und 1 Jahren

   1 – pbinom(1,10,1/100) = 0.43%

7.3. Stetige Verteilungen

Sie können stetige Verteilungen als Modell für die Häufigkeitsverteilung Ihrer Daten verwenden. Falls das Modell die Häufigkeit Ihrer Daten genügend gut wiedergibt, können Sie die Verteilungsfunktion für Abschätzungen oder Prognosen benützen. Dabei gehen Sie wie folgt vor:

1) *Familie der Verteilung:* Überlegen Sie sich anhand von Grafiken (siehe deskriptive Statistik, Kapitel 6), mit welcher Verteilungsfunktion Sie die Häufigkeitsverteilung Ihrer Daten beschreiben können.
2) *Parameterschätzung:* Schätzen Sie anhand Ihrer Daten die Parameter der Verteilungsfunktion.
3) *Verteilung testen:* Testen Sie, ob die Verteilungsfunktion (mit den vorher geschätzten Parametern) die Häufigkeit Ihrer Daten genügend gut beschreibt.
4) *Anwendung:* Falls Ihr Modell beim Test nicht verworfen wird, können Sie damit Prognosen oder Abschätzungen durchführen.

Familie der Verteilung

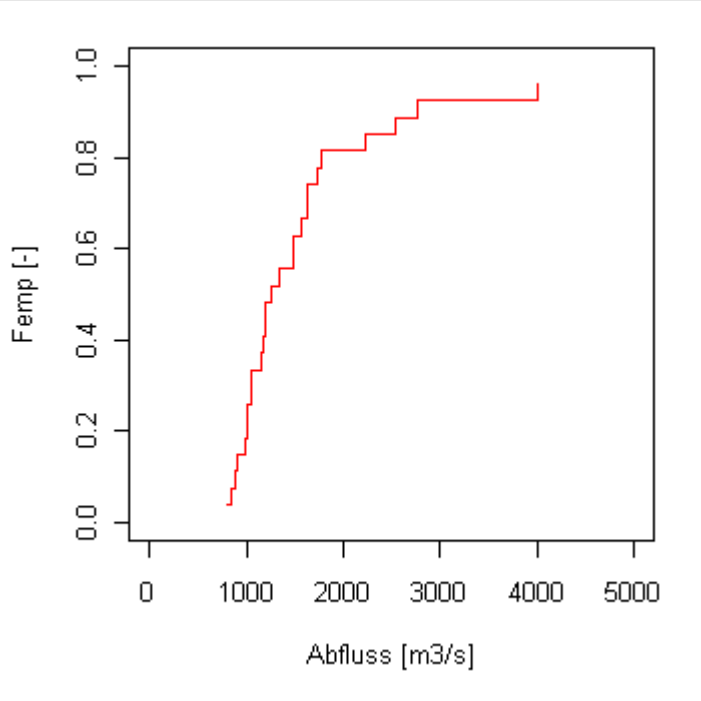
Zum Abschätzen der Verteilungs-Familie ist Erfahrung im Umgang mit Verteilungsfunktionen hilfreich. Deshalb wird in diesem Leitprogramm die Verteilungs-Familie jeweils vorgegeben. Falls Sie ausserhalb diesem Leitprogramm die Verteilungs-Familie bestimmen, sollten Sie in einem ersten Schritt abschätzen, ob die Werte gleich-verteilt, symmetrisch (z.B. normal-verteilt) oder schief (z.B. lognormal-verteilt) sind.

- Importieren Sie die Datei *Pine_River.csv,* welche die maximalen Jahresabflüssen des Pine River (Kanada) enthält, zu einem Dataframe *Pine*.

```
PINE_PFAD <- "E:/R/3_Daten/Pine_River.csv"
Pi <- read.table(PINE_PFAD, sep = ";", dec = ",", header = T)
names(Pi); dim(Pi)
Pi[1:10,]
```

- Erstellen Sie eine Grafik mit der empirischen Verteilung der Daten.

```
n <- length(Pi$Abfluss)
Femp <- 1:n/(n+1)
PI_FEMP_PFAD <- "E:/R/temp/Pine_Empirische_Verteilung.bmp"
bmp(PI_FEMP_PFAD, width = 350, height = 350)
par(cex = 1.1, mar = c(5.1, 4.1, 1.5, 1.5))
   plot(sort(Pi$Abfluss), Femp, xlim = c(0,5000), ylim = c(0,1),
      xlab = "Abfluss [m³/s]", ylab = "Femp [-]", type = "s", col = "red")
dev.off()
```



Auf der obenstehenden Grafik ist zu sehen, dass die Daten eine positive Schiefe aufweisen. Daher sollten als Verteilungs-Familien schiefe Verteilungsfunktionen (wie beispielsweise die Lognormal- oder Gumbelverteilung) verwendet werden.

Parameterschätzung

Das Paket *MASS*, welches in R standardmässig installiert (aber nicht geladen) ist, enthält unter anderem die Funktion *fitdistr* (Englisch: *fit distribution* = Kurvenanpassung). Diese Funktion schätzt die Parameter einer Verteilungsfunktion mit der Maximum Likelihood Methode (*MLE*, Englisch: *Maximum Likelihood Estimation*). Die *fitdistr*-Funktion kann jedoch nur für einige Verteilungsfunktionen verwendet werden. Eine Liste mit den verfügbaren Verteilungsfunktionen und ihrer Schreibweisen in der *fitdistr*-Funktion, finden Sie mit *?fitdistr*. Parameter dort nicht aufgeführter Verteilungsfunktionen können Sie auch mit der Momentenmethode (*MoM*, Englisch: *Method of Moments*) schätzen; die Formeln von zwei Verteilungsfunktionen finden Sie in der folgenden Tabelle.

Funktion	*Parameter*	*Verteilungsfunktion*	*Parameterschätzung (Momentenmethode)*
_unif()	*a,b*	$\frac{x-a}{b-a}$	$\hat{a} = \bar{x} - \sqrt{3} \cdot s$ $\hat{b} = \bar{x} + \sqrt{3} \cdot s$
_gumbel()	*a, b*	$\exp\left(-\exp\left(\frac{a-x}{b}\right)\right)$	$\hat{a} = \bar{x} - 0.4500535 \cdot s$ $\hat{b} = \frac{\sqrt{6}}{\pi} \cdot s$

- Laden Sie das *MASS*-Paket, das die *fitdistr*-Funktion enthält:

```
require(MASS)
```

- Schätzen Sie für *Pi$Abfluss* die Parameter der Lognormalverteilung mit der *MLE*-Methode.

```
Fit_Pi <- fitdistr(Pi$Abfluss, "lognormal")
mlog_Pi <- as.numeric(Fit_Pi$estimate[1])
slog_Pi <- as.numeric(Fit_Pi$estimate[2])
```

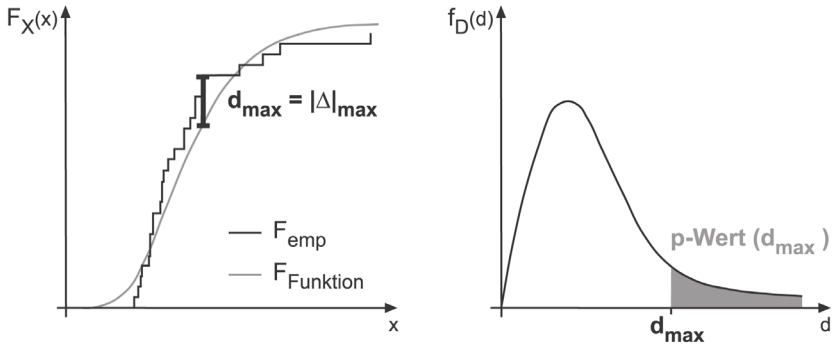
Hinweis: Die *fitdistr*-Funktion schätzt nicht nur die beiden Parameter *meanlog* und *sdlog,* sondern auch deren Standardfehler ab.

- Schätzen Sie für *Pi$Abfluss* die Parameter der Gumbelverteilung mit der *MoM*-Methode. Verwenden Sie die Formel der obenstehenden Tabelle.

```
a_Pi <- mean(Pi$Abfluss) - 0.4500535 * sd(Pi$Abfluss)
b_Pi <- sqrt(6)/pi * sd(Pi$Abfluss)
```

Verteilungen testen

In R gibt es einige statistische Tests, um zu prüfen, ob die Häufigkeiten einer Stichprobe durch eine bestimmte Verteilungsfunktion beschrieben werden können. In diesem Leitprogramm wird bloss der Kolmogorow-Smirnow-Test verwendet. Falls Sie mit dem Kolmogorow-Smirnow-Test nicht vertraut sind, hilft Ihnen die folgende Grafik, das Prinzip dieses Tests zu verstehen. Als Nullhypothese H_0 wird angenommen, dass die Verteilung der Stichprobe mit der gewählten Verteilungsfunktion beschrieben werden kann. Eine kleine Differenz d_{max} zwischen empirischer Verteilung und Verteilungsfunktion entspricht einem grossen p-Wert. Falls der p-Wert grösser als das gewählte Signifikanzniveau α (oft α=5%) ist, wird die Nullhypothese H_0 nicht verworfen.



- Laden Sie das *evd*-Paket, welches die *gumbel*-Funktion enthält.
  require(evd)
- Testen Sie, wie gut beide Modelle (Lognormal- und Gumbelverteilung) die Häufigkeiten Ihrer Stichprobe beschreiben:
  ks.test(Pi$Abfluss, "plnorm", mlog_Pi, slog_Pi)
  ks.test(Pi$Abfluss, "pgumbel", a_Pi, b_Pi)

Für beide Verteilungsfunktionen erhalten Sie p-Werte grösser 0.05. Daraus folgt, dass Sie beide Funktionen verwenden könnten um Ihre Daten zu beschreiben.

Anwendung

Wenden Sie nur die beiden Modelle an!

- Berechnen Sie mit beiden Modellen den Abfluss, der mit einer Jährlichkeit von R = 100 Jahren erreicht oder überschritten wird: Hinweise:

  Überschreitungswahrscheinlichkeit $p = \frac{1}{R} = \frac{1}{100} = 0.01$

  Unterschreitungswahrscheinlichkeit $F = 1 - p = 0.99$

  qlnorm(0.99, mlog_Pi, slog_Pi)
  qgumbel(0.99, a_Pi, b_Pi)

Mit der Lognormalverteilung erhalten Sie ein 100-jährliches Hochwasser von 3360 m^3, mit der Gumbelverteilung 3742 m^3. An dieser Stelle wird darauf hingewiesen, dass solche Hochwasserprognosen mit grossen Unsicherheiten verbunden sind.

Übungsaufgabe 7.2

1) Bei Simulationen werden oft Zufallszahlen verwendet. Erzeugen Sie einen Vektor *ZUFALL* mit 1000 gleichverteilten Zufallszahlen zwischen 5 und 8.
   Hinweis: Verwenden Sie zuerst den Befehl set.seed(5). Damit startet der Pseudo-Zufallsgenerator von R an der Stelle 5. Damit lassen sich die „Zufallszahlen“ reproduzieren, was ermöglicht, dass Sie Ihre Lösungen mit den Lösungen dieses Leitprogramms vergleichen können.
2) Doch halt! Sind die Werte überhaupt gleichverteilt? D.h. sind sie einigermassen regelmässig zwischen 5 und 8 verteilt? Sind sie nicht vielleicht sogar normalverteilt? Überprüfen Sie das!
   Hinweis: Schätzen Sie zuerst für Ihre Stichprobe *ZUFALL* die Parameter der Gleichverteilung (mit *MoM*) und Normalverteilung (mit *MLE*). Testen Sie die beiden Modelle anschliessend mit dem Kolmogorow-Smirnow-Test.

Lösungen zu Übungsaufgabe 7.2

1)
```
set.seed(5)
ZUFALL <- runif (1000, 5, 8)
```
2) Parameter der Gleichverteilung (Formeln aus der Tabelle auf Seite 119):
```
a_ZUFALL <- mean(ZUFALL) - sqrt(3) * sd(ZUFALL)
b_ZUFALL <- mean(ZUFALL) + sqrt(3) * sd(ZUFALL)
```
Parameter der Normalverteilung:
```
Fit_ZUFALL <- fitdistr(ZUFALL, "normal")
m_ZUFALL <- as.numeric(Fit_ZUFALL$estimate[1])
s_ZUFALL <- as.numeric(Fit_ZUFALL$estimate[2])

ks.test(ZUFALL, "punif", a_ZUFALL, b_ZUFALL)
ks.test(ZUFALL, "pnorm", m_ZUFALL, s_ZUFALL)
```

Auf einem Signifikanzniveau von a = 5% folgt die Stichprobe einer Gleichverteilung, nicht aber einer Normalverteilung. Glück gehabt!

7.4. Einfache Lineare Regression

Definition

Mit Einfacher Linearer Regression (ELR) können Sie den Zusammenhang zwischen einer erklärenden Variablen X und einer Zielgrösse Y beschreiben. Dabei wird die folgende mathematische Schreibweise verwendet:

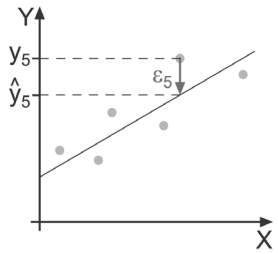
$$y_i = \beta_0 + \beta_1 \cdot x_i + \varepsilon_i$$

y_i = i-ter Wert der Zielgrösse

x_i = i-ter Wert der erklärenden Variablen

β_0, β_1 = Regressionskoeffizienten

ε_i = Zufälliger Fehler. Die Fehler werden auch Residuen genannt und sind die Differenz zwischen Datenwert y_i und Modellwert $\hat{y}_i$: $\varepsilon_i = y_i - \hat{y}_i$.



Die Bezeichnung *einfach* weist darauf hin, dass nur eine erklärende Variable im Modell vorkommt. *Linear* bedeutet, dass die Regressionsparameter in linearer Form verwendet werden; $y_i = \beta_0 + \beta_1 \cdot x_i^2 + \varepsilon_i$ ist daher auch ein einfaches lineares Regressionsmodell.

Zweck

Mit ELR versuchen Sie ein Modell zwischen einer einfach erhältlichen erklärenden Variablen X und einer schwierig verfügbaren Zielgrösse Y aufzustellen. Falls Sie ein Modell erstellt haben, können Sie damit mit X Abschätzungen für Y machen. Es lohnt sich beispielsweise ein Modell für den Zusammenhang zwischen der Wassertemperatur (X) und der Sauerstoffkonzentration (Y) zu finden, damit Sie anhand der (einfach messbaren) Wassertemperatur die Sauerstoffkonzentration abschätzen können.

Vorgehensweise

Um ein ELR-Modell zu erstellen, gehen Sie wie folgt vor:

1) Schätzen Sie die Parameter.
2) Prüfen Sie die Annahmen des Modells.
3) Falls die Annahmen nicht erfüllt sind, dürfen Sie das Modell nicht verwenden! Erstellen Sie durch Transformieren der erklärenden Variablen ein neues Modell, für das Sie wiederum die Parameter schätzen und die Annahmen prüfen.

Annahmen

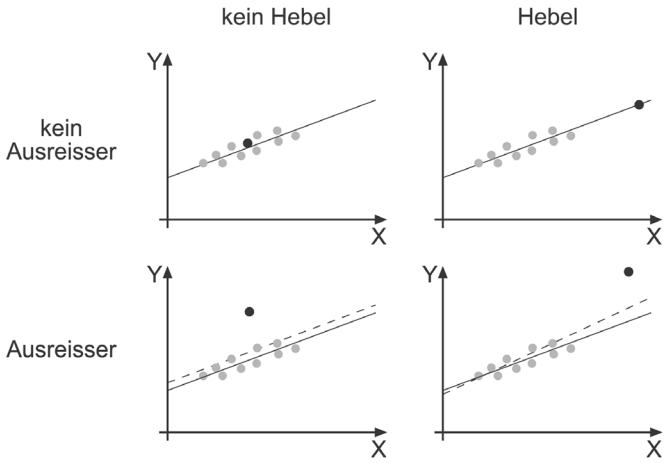
Wenn Sie ein ELR-Modell erstellen, müssen folgende Annahmen erfüllt sein:

- Der Zusammenhang zwischen der erklärenden Variablen X und der Zielgrösse Y ist ungefähr linear.
- Die Residuen ε_i haben eine konstante Varianz.
- Die Residuen ε_i sind normalverteilt.
- Die Residuen ε_i sind unabhängig.

Die Unabhängigkeit der Residuen kann grafisch überprüft werden, indem die Residuen räumlich, respektive bei Zeitreihen als Funktion der Zeit, dargestellt werden. Diese Überprüfung wird im Rahmen dieses Leitprogramms nicht durchgeführt, weil nicht bei allen Datensätzen deren räumliche oder zeitliche Struktur bekannt ist. Dennoch sollten Sie bei ihrer zukünftigen Arbeit jeweil abschätzen, ob die Residuen unabhängig sind.

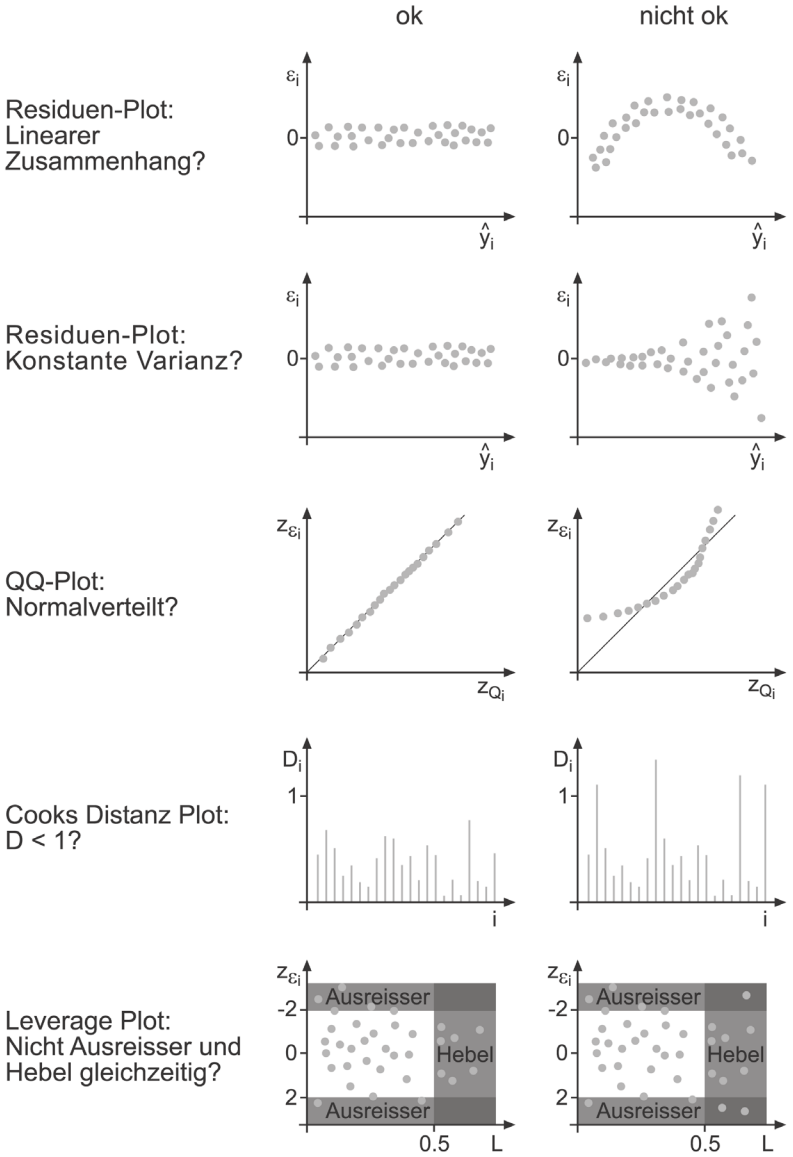
Zusätzlich zu den oben beschriebenen Annahmen sollte Ihr Regressionsmodell nicht zu stark von einzelnen Werten abhängen. Um eine eventuelle Abhängigkeit zu prüfen, verwenden Sie die so genannte *Leverage* und die *Cooks Distanz*. Letztere ist ein Mass, wie stark sich das Regressions-Modell ändert, wenn ein Datenwert weggelassen wird. Das heisst, Sie erhalten für jeden Datenpunkt eine *Cooks Distanz*. Grosse *Cooks Distanzen* bedeuten, dass sich das Regressionsmodell stark ändert, wenn dieser Wert weggelassen wird. Punkte mit *Cooks Distanzen* grösser eins sollten Sie genauer untersuchen.

Die *Leverage* (Englisch: Hebel) beschreibt die Hebelwirkung eines Datenpunkts. Insbesondere wenn ein Punkt eine grosse Hebelwirkung aufweist, und gleichzeitig ein „Ausreisser“ ist, sollten Sie dieses Regressions-Modell nicht verwenden.



Diagnostik-Plots

In diesem Leitprogramm wird nur anhand von Grafiken und nicht mit Tests entschieden, ob die Annahmen des ELR-Modells erfüllt sind. In R können Sie auf einfache Weise Diagnostik-Plots erstellen, um die Annahmen zu prüfen. Die folgende Grafik enthält je ein Beispiel, wann eine Annahme erfüllt, und wann sie nicht erfüllt ist.



Modelle in R

In R wird die Zielgrösse Y zuerst geschrieben; nach dem Tilde-Symbol (~) wird das Modell (beispielsweise die erklärende Variable X) angegeben:

y ~ Modell

y ~ x

Für lineare Modelle wird in R die *lm*-Funktion verwendet (lm = *linear model*).

Nun können Sie ein ELR-Modell erstellen und prüfen!

- Importieren Sie die Datei *Weil_O2.csv* zu einem Dataframe *O2*. Sie haben diese Daten bereits im Kapitel 6 verwendet, um die Korrelation zwischen Wassertemperatur und gelöstem Sauerstoff zu untersuchen.

```
O2_PFAD <- "E:/R/3_Daten/Weil_O2.csv"
O2 <- read.table(O2_PFAD, sep = ";", dec = ",", header = T)
names(O2); dim(O2)
head(O2)
```

- Schauen Sie sich die Daten nochmals in einer Grafik an:

```
plot(O2$Temp, O2$O2)
```

- Erstellen Sie ein ELR-Modell mit der Sauerstoff-Konzentration als Zielgrösse und der Temperatur als erklärende Variable:

```
W_Modell_1 <- lm (O2$O2 ~ O2$Temp)
```

- Lassen Sie sich eine Zusammenfassung des Modells ausgeben. Es wird bald erklärt, wie Sie diese Zusammenfassung lesen.

```
summary(W_Modell_1)
```

- Prüfen Sie, ob die Annahmen für ein ELR-Modell erfüllt sind. Wenn Sie den *plot*-Befehl auf Ihr Modell anwenden, werden Diagnostik-Plots ausgeben. Mit dem Argument *which* wählen Sie, welche Diagnostik-Plots erstellt werden; in diesem Fall werden folgende Grafiken erstellt:

Residuen-Plot	QQ-Plot
Cooks Distanz Plot	Leverage Plot

In den Diagnostik-Plots werden eventuell kritischen Punkte mit Ihrem Index gekennzeichnet. Damit können Sie diese Punkte in den Daten identifizieren.

```
ANN_MOD1_PFAD <- "E:/R/temp/Annahmen_W_Modell_1.bmp"
bmp(ANN_MOD1_PFAD, width = 600, height = 600)
   par(mfrow = c(2, 2))
   plot(W_Modell_1, which = c(1,2,4,5))
dev.off()
```

Im Residuen-Plot können Sie feststellen, dass die Varianz der Residuen eher nicht konstant ist. Deshalb versuchen Sie, mit einer Transformation der erklärenden Variablen ein besseres Modell zu erhalten:

- Transformieren Sie die erklärende Variable *O2$Temp* mit der Wurzelfunktion:

  ```
  Temp2 <- sqrt(O2$Temp)
  ```

- Vergleichen Sie das neue Objekt mit *O2$O2*:

  ```
  plot(Temp2, O2$O2)
  ```

- Erstellen Sie ein neues ELR-Modell und testen Sie dieses:

  ```
  W_Modell_2 <- lm(O2$O2 ~ Temp2)
  summary(W_Modell_2)
  ANN_MOD2_PFAD <- "E:/R/temp/Annahmen_W_Modell_2.bmp"
  bmp(ANN_MOD2_PFAD, width = 600, height = 600)
     par(mfrow = c(2, 2))
     plot(W_Modell_2, which = c(1,2,4,5))
  dev.off()
  ```

Gemäss den Diagnostik-Plots scheinen alle Annahmen erfüllt zu sein! Deshalb verwenden Sie das *Modell_2*.

summary(Modell)

In der folgenden Grafik sind diejenigen Teile der Zusammenfassung grau hinterlegt, die für Sie zu diesem Zeitpunkt interessant sind:

```
> summary(W_Modell_2)

Call:
lm(formula =  O2$O2 ~ Temp2)

Residuals:
     Min       1Q   Median       3Q      Max
-0.65693 -0.23093 -0.03462  0.22920  1.06926

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 16.94404    0.22263   76.11   <2e-16 ***
Temp2       -1.90626    0.06084  -31.33   <2e-16 ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3853 on 49 degrees of freedom
Multiple R-squared: 0.9525,     Adjusted R-squared: 0.9515
F-statistic: 981.7 on 1 and 49 DF,  p-value: < 2.2e-16
```

- Schätzung der Regressionskoeffizienten:

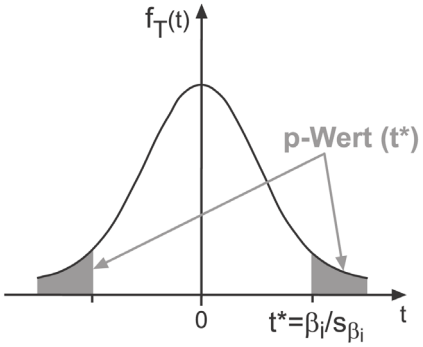
  $\hat{\beta}_0 = 16.94404$

  $\hat{\beta}_1 = -1.90626$

  Sie können sich diese auch mit coef(W_Modell_2) ausgeben lassen.

- p-Werte der Regressionskoeffizienten:
  In diesem Beispiel sind die p-Werte für beide Regressionskoeffizienten sehr klein (kleiner als 2 mal 10^{-16}). Das bedeutet, dass β_0 und β_1 beide signifikant von Null verschieden sind. Mit Stern-Symbolen (*) wird zusätzlich zum p-Wert angezeigt, auf welchem Signifikanzniveau die Regressionskoeffizienten von Null verschieden sind.

Hinweis: Es wird die Nullhypothese H_0 angenommen, dass ein Regressionskoeffizient β_i gleich Null ist. Ein kleiner p-Wert weist darauf hin, dass sich die Teststatistik t^* (= β_i standardisiert) von Null unterscheidet.



- Bestimmtheitsmass R^2:
  Das Bestimmtheitsmass sagt in diesem Fall aus, dass 95.25% der Variabilität von *W_O2* durch *W_Temp* erklärt werden. Bei der ELR entspricht das Bestimmtheitsmass dem Quadrat des Korrelationskoeffizienten (siehe Kapitel 6.4). Bei naturwissenschaftlichen Zusammenhängen gilt ein R^2 von 0.9 als hoch, bei soziologischen Studien ein R^2 von 0.4.

Modellwahl

Wenn Sie zwischen mehreren Modellen auswählen, sollten Sie grundsätzlich das Modell mit dem höchsten Bestimmtheitsmass verwenden, vorausgesetzt dieses Modell erfüllt die ELR-Annahmen! Zudem sollten Sie möglichst einfache Modelle bevorzugen.

Konfidenz- und Prognoseintervalle

Sie können Ihr Modell nun für Prognosen benutzen. Zudem können Sie damit Konfidenz- und Prognoseintervalle berechnen. Das Konfidenzintervall ist ein Mass für die Schätzgenauigkeit der Parameter. Das Prognoseintervall hingegen gibt einen Prognose-Bereich der Zufallsvariablen der Zielgrösse Y an. Falls Sie eine Aussage über die Schätzgenauigkeit Ihrer Regression machen wollen, geben Sie

das Konfidenzintervall an. Wenn Sie an Prognosen für die Zielgrösse interessiert sind, geben Sie das Prognoseintervall an.

- Stellen Sie mit der low-level Grafik *abline* die Regressionsgerade Ihres Modells dar:

```
plot(Temp2, O2$O2)
abline(W_Modell_2)
```

- Erstellen Sie mit der Funktion *predict.lm* das 95%-Konfidenz- und Prognoseintervall. Dazu definieren Sie zuerst einen Vektor *SEQ* mit Werten für *W_Temp2*, für die anschliessend Sauerstoff-Prognosen berechnet werden.

```
SEQ <- seq(2, 5, 0.1)
CONF_W <- predict.lm(W_Modell_2, data.frame(Temp2 = SEQ),
        interval="confidence", level= 0.95)
PRED_W <- predict.lm(W_Modell_2, data.frame(Temp2 = SEQ),
        interval="prediction", level= 0.95)
```

- Betrachten Sie die Objekte CONF_W und PRED_W.
- Stellen Sie die beiden Intervalle grafisch dar:

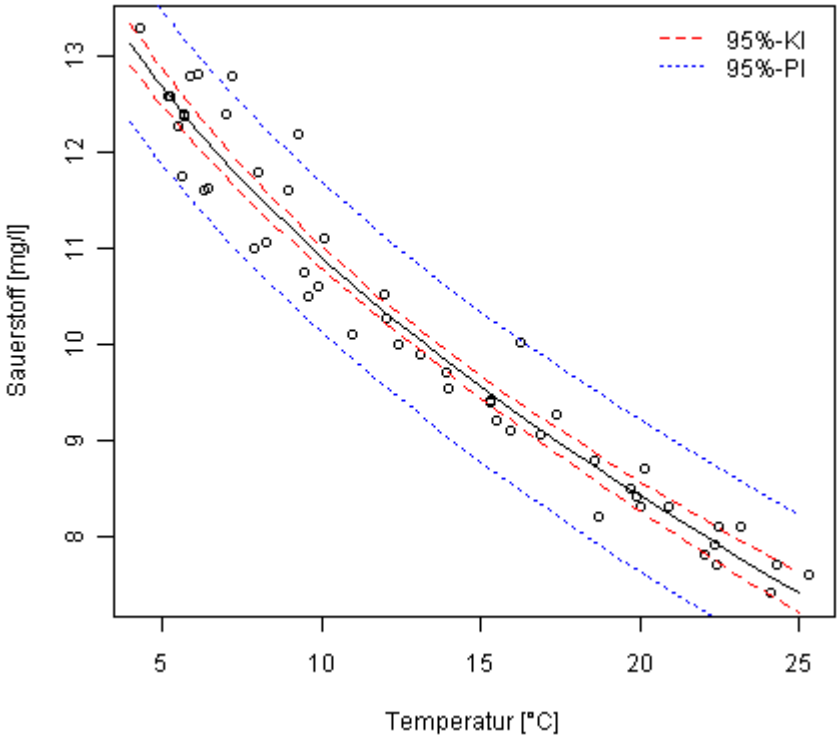
```
plot(Temp2, O2$O2)
        points(SEQ, CONF_W[,"fit"], type = "l", lty = 1)
        points(SEQ, CONF_W[,"lwr"], type = "l", lty = 2)
        points(SEQ, CONF_W[,"upr"], type = "l", lty = 2)
        points(SEQ, PRED_W[,"lwr"], type = "l", lty = 3)
        points(SEQ, PRED_W[,"upr"], type = "l", lty = 3)
```

- Erstellen Sie die gleiche Grafik für die nicht transformierte Variable *O2$Temp*. Dazu müssen Sie den Vektor *SEQ* zurücktransformieren.

```
W_KI_PI_PFAD <- "E:/R/temp/W_KI_PI.bmp"
bmp(W_KI_PI_PFAD, width = 400, height = 400)
   plot(O2$Temp, O2$O2)
      points(SEQ^2, CONF_W[,"fit"], type = "l", lty = 1)
      points(SEQ^2, CONF_W[,"lwr"], type = "l", lty = 2)
      points(SEQ^2, CONF_W[,"upr"], type = "l", lty = 2)
      points(SEQ^2, PRED_W[,"lwr"], type = "l", lty = 3)
      points(SEQ^2, PRED_W[,"upr"], type = "l", lty = 3)
dev.off()
```

Hinweis: Die Sauerstoffwerte bleiben unverändert! Sie werden bloss den zurücktransformierten Temperatur-Werten (SEQ^2) zugeordnet.

Die erstellte Grafik sieht wie folgt aus:



Übungsaufgabe 7.3

1) Importieren Sie die Datei *HydroUG.csv* zu einem Dataframe *HUG*. Sie enthält Kennzahlen von 20 hydrologischen Untersuchungsgebieten (HUG) der Schweiz. Erstellen Sie ein ELR-Modell zwischen *P06* (Gebietsniederschlag 2006 in mm) und dem natürlichen Logarithmus von *Q06* (Gebietsabfluss 2006 in mm).
2) Sind die ELR-Annahmen erfüllt? Erstellen Sie dazu vier Diagnotikplots.
3) Wie gross ist das Bestimmtheitsmass?
4) Erzeugen Sie eine Grafik mit *P06* und *Q06* inklusive der Regressionskurve sowie dem 95% Konfidenzintervall.

Lösungen zu Übungsaufgabe 7.3

1)
```
HUG_PFAD <- "E:/R/3_Daten/HydroUG.csv"
HUG <- read.table(HUG_PFAD, sep = ";", dec = ",", header = T)
names(HUG); dim(HUG)
head(HUG)

Q06A <- log(HUG$Q06)
plot(Q06A, HUG$P06)
HUG_Modell_1 <- lm (HUG$P06 ~ Q06A)
```

2)
```
ANN_HUG1_PFAD <- "E:/R/temp/Annahmen_HUG_Modell_1.bmp"
bmp(ANN_HUG1_PFAD, width = 600, height = 600)
   par(mfrow = c(2, 2))
   plot(HUG_Modell_1, which = c(1,2,4,5))
dev.off()
```

Die Annahmen sind erfüllt!

3)
```
summary(HUG_Modell_1)
```

Das Bestimmtheitsmass beträgt 0.8075

4)
```
SEQ <- seq(6.4,7.4,0.05)
CONF_HUG <- predict.lm(HUG_Modell_1, data.frame(Q06A = SEQ),
        interval="confidence", level= 0.95)

HUG_KI_PI_PFAD <- "E:/R/temp/HUG_KI_PI.bmp"
bmp(HUG_KI_PI_PFAD, width = 450, height = 400)
   plot(HUG$Q06, HUG$P06)
   points(exp(SEQ), CONF_HUG[,"fit"], type = "l", lty = 1)
   points(exp(SEQ), CONF_HUG[,"lwr"], type = "l", lty = 2)
   points(exp(SEQ), CONF_HUG[,"upr"], type = "l", lty = 2)
dev.off()
```

7.5. Multiple Lineare Regression

Bei der Multiplen Linearen Regression (MLR) wird die Zielgrösse Y durch mehrere erklärende Variabeln X1, X2,... beschrieben.

$$y_i = \beta_0 + \beta_1 \cdot x_{1i} + \beta_2 \cdot x_{2i} + ... + \varepsilon_i$$

y_i = i-ter Wert der Zielgrösse

x_{1i} = i-ter Wert der ersten erklärenden Variablen

x_{2i} = i-ter Wert der zweiten erklärenden Variablen

$\beta_0, \beta_1, \beta_2$ = Regressionskoeffizienten

ε_i = Residuen

Dabei gehen Sie grundsätzlich gleich vor, wie bei der einfachen linearen Regression! Es gibt folgende Unterschiede:

- Modellschreibweise: Y ~ X1 + X2
- Als Gütekriterium Ihres Modells verwenden Sie nicht mehr das Bestimmtsheitsmass, sondern das angepasste Bestimmheitsmass (Englisch: *adjusted*). Begründung: Je mehr erklärende Variabeln Sie verwenden, desto mehr der Varianz von Y wird erklärt. Deshalb besteht das angepasste Bestimmtsheitsmass aus R^2 multipliziert mit einem „Bestrafungsfaktor" für komplexe Modelle.
- Sie können keine Darstellungen zwischen der Zielgrösse Y und den erklärenden Variablen X1, X2,… machen. Deshalb erweisen sich die Diagnostik-Plots als hilfreich!
- In diesem Leitprogramm bestimmen Sie Ihr Modell durch Rückwärts-Eliminination. Das bedeutet, dass Sie mit relativ vielen erklärenden Variablen beginnen, und jeweils die Variable mit grösstem p-Wert entfernen, bis das Modell nur noch erklärende Variablen mit p-Werten kleiner einem Signifikanzniveau von 5% enthält.

- Erstellen Sie ein Modell für *P06* (Gebietsniederschlag). Beginnen Sie mit einem Modell bestehend aus den erklärenden Variabeln *HOEHE* (durchschnittliche Einzugsgebietshöhe), *WALD* (Waldanteil), *BEVD* (Bevölkerungsdichte) und *GEWD* (Gewässernetzdichte).

```
HUG_Modell_2 <- lm(HUG$P06 ~ HUG$HOEHE + HUG$WALD +
        HUG$BEVD + HUG$GEWD)
summary(HUG_Modell_2)
```

- Es ist nicht besonders erstaunlich, dass die Bevölkerungsdichte am wenigsten der Varianz des Niederschlags erklärt, und deshalb weg gelassen wird.

```
HUG_Modell_2 <- lm(HUG$P06 ~ HUG$HOEHE + HUG$WALD +
        HUG$GEWD)
summary(HUG_Modell_2)

HUG_Modell_2 <- lm(HUG$P06 ~ HUG$HOEHE + HUG$WALD)
summary(HUG_Modell_2)
```

Hinweis: Den Achsenabschnitt (Interzept, β_0) könnten Sie wie folgt aus Ihrem Modell entfernen: `HUG_Modell_2 <- lm (HUG$P06 ~ HUG$HOEHE + HUG$WALD - 1)`. Es gilt zu bemerken, dass die p-Werte jeweils nur die zusätzlich erklärte Varianz einer Variablen berücksichtigen. Wenn Ihr Modell zwei ähnliche Variabeln enthält, ist der p-Wert einer Variablen relativ hoch, auch wenn ihr Einfluss auf Y eigentlich beträchtlich wäre.

- Prüfen Sie die Annahmen des Modells:

```
ANN_HUG2_PFAD <- "E:/R/temp/Annahmen_HUG_Modell_2.bmp"
bmp(ANN_HUG2_PFAD, width = 600, height = 600)
   par(mfrow = c(2, 2))
   plot(HUG_Modell_2, which = c(1,2,4,5))
dev.off()
```

Die Annahmen sind (einigermassen) erfüllt!

Mit diesem Modell können Sie den Jahresniederschlag eines Einzugsgebiets ohne Niederschlagsmessstationen abschätzen. Das dürfen Sie aber nur, wenn sich dieses Einzugsgebiet ähnliche hydrologische Eigenschaften aufweist, wie die analysierten Untersuchungsgebiete.

- Wenden Sie ihr Modell an, und bestimmen Sie die Prognoseintervalle.

```
PRED_HUG_2 <- predict.lm(HUG_Modell_2,
        data.frame(HOEHE = HUG$HOEHE, WALD = HUG$WALD),
        interval="prediction", level= 0.95)
```

Übungsaufgabe 7.4

1) Erstellen Sie ein Modell für *DURCH* (mittlere Durchlässigkeit des Bodens). Beginnen Sie mit drei erklärenden Variablen ihrer Wahl. Signifikanzniveau = 5%. Eine Liste mit der vollständigen Bezeichnung der Spalten finden Sie im Anhang C dieses Leitprograms.
2) Sind die Annahmen des Modells erfüllt?

Lösungen zu Übungsaufgabe 7.4

1) Viele Lösungen sind möglich; hier ein Beispiel:

```
HUG_Modell_3 <- lm(HUG$DURCH ~ HUG$HOEHE + HUG$P06 +
        HUG$WALD)
summary(HUG_Modell_3)

HUG_Modell_3 <- lm (HUG$DURCH ~ HUG$HOEHE + HUG$P06)
summary(HUG_Modell_3)

HUG_Modell_3 <- lm (HUG$DURCH ~ HUG$HOEHE + HUG$P06 -1)
summary(HUG_Modell_3)

HUG_Modell_3 <- lm (HUG$DURCH ~ HUG$HOEHE -1)
summary(HUG_Modell_3)
```

2)

```
ANN_HUG3_PFAD <- "E:/R/temp/Annahmen_HUG_Modell_3.bmp"
   bmp(ANN_HUG3_PFAD, width = 600, height = 600)
      par(mfrow = c(2, 2))
      plot(HUG_Modell_3, which = c(1,2,4,5))
   dev.off()
```

Die Annahmen sind nicht erfüllt; das Modell darf nicht verwendet werden!

7.6. Merkpunkte

Repetieren Sie die wichtigsten Merkpunkte des siebten Kapitels:

Verteilungen

- Vier Funktionen für Verteilungen in R: d_ (density), p_ (probability), q_ (quantile), r_ (random).

Diskrete Verteilungen

- Binomialverteilung: Wahrscheinlichkeit von Bernoulli-Ereignissen.
- In 5 Jahren zwei 20-jährliche Hochwasser: dbinom (2, 5, 1/20).
- In 5 Jahren weniger als vier 20-jährliche Hochwasser: pbinom(3, 5, 1/20).

Stetige Verteilungen

- Familie der Verteilung.
- Parameterschätung:
  MoM (Momentenmethode; eigene Funktionen schreiben).
  MLE (Maximum Likelihood Methode; *fitdistr*-Funktion, im *MASS*-Paket).
  Extrahieren der Parameter mit *as.numeric(Fit$estimate[1])*.
- Verteilung testen: Kolmogorow-Smirnow-Test.
  ks.test (Daten, "Verteilung", Parameter).
- Anwendung der Verteilung.

Einfache Lineare Regression (ELR)

- Modell: $y_i = \beta_0 + \beta_1 \cdot x_i + \varepsilon_i$.
- Modell in R: *lm(y ~ x).*
- Schwierig verfügbare Zielgrösse, einfach erhältliche erklärende Variable.
- ELR-Annahmen und Diagnostik-Plots:
  Linearer Zusammenhang zwischen X und Y (Residuen-Plot).
  Konstante Varianz der Residuen (Residuen-Plot).
  Normalverteilung der Residuen (QQ-Plot).
- Abhängigkeit der Regression von einzelnen Punkten:
  Cooks Distanz (Cooks Distanz Plot).
  Hebel Wirkung (Leverage-Plot).
- Transformation der erklärenden Variable: *log(x)*, *x^2*, *sqrt(x)*, usw.
- *summary(Modell)*: Schätzungen und p-Werte der Regressionskoeffizienten, Bestimmtheitsmass R^2.

7.7. Lernkontrolle

Falls Sie sich mit dem Lernstoff dieses Kapitels genügend vertraut fühlen, beginnen Sie nun mit der Lernkontrolle! Verwenden Sie dabei die Lösungen erst am Schluss, um Ihre Antworten zu korrigieren. Falls Sie Schwierigkeiten haben, repetieren Sie den entsprechenden Abschnitt.

Fragen

1) Wie gross ist die Wahrscheinlichkeit, dass während 80 Jahren kein 200-jährliches Hochwasser auftritt?
2) Mit welchem Befehl berechnen Sie die Wahrscheinlichkeit, dass während 80 Jahren drei oder mehr 200-jährliche Hochwasser auftreten?
3) Importieren Sie die Datei *Emmenmatt.csv,* welche die Jahre enthält, in denen die Emme bei Emmenmatt mindestens einen Abfluss von 300 m^3/s erreicht hat. Zudem enthält die Datei die Anzahl Jahre zwischen diesen Hochwasserereignissen (Komponente *Dauer*). Schätzen Sie für *Dauer* den Parameter *r_DAUER* einer Exponentialverteilung mit der *MLE*-Methode.
4) Können Ihre Daten durch diese Verteilung beschrieben werden? Verwenden Sie beim Test ein Signifikanzniveau von 5%.
5) Erstellen Sie für den HUG-Datensatz ein *ELR*-Modell für *Q06* (Zielgrösse, jährlicher Gebietsabfluss 2006) und *QMAI06* (erklärende Variable, Gebietsabfluss Mai 2006). Sind die Annahmen erfüllt?

7.8. Kapiteltest

Falls Sie sich bei den vorher behandelten Themen sicher fühlen, melden Sie sich bei der Lehrperson zum Kapiteltest. Der letzte Kapiteltest dieses Leitprogramms verläuft gleich wie die vorherigen Tests. Für den Kapiteltest stehen Ihnen maximal 20 Minuten zur Verfügung.

7.9. Zusatzaufgaben

Falls Sie das Leitprogramm wesentlich früher als andere Studierende beendet haben, können Sie sich mit den folgenden Zusatzaufgaben befassen. Zudem können Sie auch Zusatzaufgaben vorangehender Kapitel lösen.

1) Wählen Sie von den nicht behandelten Funktionen auf Seite 123 dieses Leitprogramms diejenigen aus, die für Ihre Arbeit interessant sein könnten. Schauen Sie in der R-Hilfe nach, wie die Funktionen definiert sind.
2) Wozu dient die *choose*-Funktion?
3) Schätzen Sie für *Pi$Abfluss* mit der *mle*-Funktion des Pakets *stats4* die Parameter der Gumbelverteilung mit der *MLE*-Methode. Hinweis: Verwenden Sie als Startwerte die *MoM*-Parameter. Berechnen Sie das hundertjährliche Hochwasser einmal mit den *MLE*-Parametern, und einmal mit den *MoM*-Parametern der Gumbelverteilung.
4) Lesen Sie Ricci (2005), eine Dokumentation mit dem Titel *Fitting Distributions with R* (siehe R-Homepage).
5) Schauen Sie sich das Paket *nortest* an, das Funktionen enthält mit denen Sie auf Normalverteilung prüfen können.
6) Finden Sie heraus, wie Sie mit der *t.test*-Funktion auf Unterschiede zwischen zwei Mittelwerten testen können. Suchen Sie auch nach anderen hilfreichen Test-Funktionen in R.
7) Finden Sie heraus, wie Sie die *matplot*-Funktion verwenden können. Stellen Sie damit für das Modell *W_Modell_2* (Seite 134, die Annahmen sind erfüllt) die 95%-Konfidenzintervalle grafisch dar.
8) Schauen Sie sich in Ligges (2007) auf Seite 136 die Tabelle 7.3 an. Finden Sie heraus, wozu der *I()*-Operator dient.
9) Spezialaufgabe (kein Zusammenhang mit R): In diesem Leitprogramm ist oft der Zusammenhang zwischen der Auftretenswahrscheinlichkeit p und der Jährlichkeit R als $p = \frac{1}{R}$ verwendet worden. Beweisen Sie diesen Zusammenhang! Tipp: Versuchen Sie die geometrische Verteilung zu verstehen, und den Erwartungswert der geometrischen Verteilung zu beweisen.

8. Lösungen der Lernkontrollen

Kapitel 2

1) a) *setwd("C:/Programme")*, b) *getwd()*, c) *setwd("C:/Programme")*.
2) a) *library()*, b) *install.packages("DAAG")*, *require(DAAG)*.
3) Ja, das Kapitel 2 (*Graphics*).
4) Das Paket enthält Funktionen zum Analysieren von Tierspuren (räumlich und zeitlich).
5) *?match*, *example(match)*.
6) a) *Format/Selection Mode/Column, b) Format/Selection Mode/Normal c) Format/Block/Indent*.
7) *R/Controlling R/Remove all objects*.

Kapitel 3

1)
```
a <- rep (seq(10, by = 5, length = 6), 2)
a <- rep (seq(10, 35, by = 5), 2)
```
2) Prüfen mit *mode(b)* ergibt: Der Datentyp von b ist *logical*
3)
```
A <- matrix(data = a, ncol = 3, byrow =TRUE)
B <- matrix(data = b, ncol = 3, byrow =TRUE)
```
4) Damit die beiden Matrizen mit den Datentypen *numeric* und *logical* miteinander multipliziert werden können, wird der Datentyp von Matrix *B* von *logical* zu *numeric* umgewandelt.
5)
```
N <- cbind(M, c(0, 40, 0, 40) )
```
6)
```
pH_PFAD <- "E:/R/temp/pH.csv"
pH <- read.table (pH_PFAD, sep = ";", dec = ",", header= TRUE)
```
7)
```
Protonen <- 10 ^ (-pH)
```
8)
```
PRO_PFAD <- "E:/R/temp/Protonen.csv"
write.table(Protonen, file= PRO_PFAD, sep = ";", dec = ",", row.names = F)
```

Kapitel 4

1)
```
SCHWER_PFAD <- "E:/R/3_Daten/ Schwermetalle_2003.csv"
Schwer <- read.table(SCHWER_PFAD, sep = ";", dec = ",",
        header = T)
        names(Schwer)
        Schwer [1:5, ]
```
2)
```
ob_Hg_NA <- is.na(Schwer$Hg)
```
3)
```
Schwer$Hg[ob_Hg_NA] <- 0.01
```
4)
```
Schwer1 <- na.omit(Schwer)
```
5)
```
ob_Schwer2 <- (Schwer1$Pb >= 0.4) & (Schwer1$Cu >= 1.4)
Schwer2 <- Schwer1[ob_Schwer2, c("Periode", "Pb", "Cu")]
```

Oder:

```
wo_Schwer2 <- which( (Schwer1$Pb > 0.4) & (Schwer1$Cu >= 1.4) )
Schwer2 <- Schwer1[wo_Schwer2, c(2, 9, 6)]
```

6) `Reihe3 <- order (-Schwer1$Cu, Schwer1$Q_Mittel)`

```
Schwer3 <- Schwer1 [Reihe3, ]
```

Kapitel 5

1) `A <- c(2.4, 7.8, 1.5, 5.4)`

```
floor(A)
min(A)
```

2) `V_Damm <- function (l, h, a, c) { (a+c)/2 * h * l }`

3) `V_Damm(70, 2.5, 7, 3)`

875 m^3

4) Daten importieren:

```
ROM_PFAD <- "E:/R/3_Daten/Romanshorn.csv"
Rom <- read.table(ROM_PFAD, sep =";", dec=",", header=T)
names(Rom); dim(Rom)
head(Rom)
```

Datums-Vektor:

```
Datum_Rom <- paste(1:31,".","8.","2006", sep="")
```

5) `Datum_Rom1 <- as.Date(Datum_Rom, "%d.%m.%Y")`

6)
```
RHE_PFAD <- "E:/R/3_Daten/Rheinfelden_Temp.csv"
Rhe <- read.table(RHE_PFAD, sep =";", dec=",", header=T)
names(Rhe); dim(Rhe)
head(Rhe)

Zeit_Tem <- paste(Rhe$Datum, Rhe$Uhrzeit, sep = " ")
Zeit_Tem1 <- as.POSIXct(strptime(Zeit_Tem, format="%d.%m.%Y
%H:%M", tz="GMT"), tz="GMT")
```

7) `as.numeric(Zeit_Tem1[2:10] ) - as.numeric( Zeit_Tem1[1:9])`

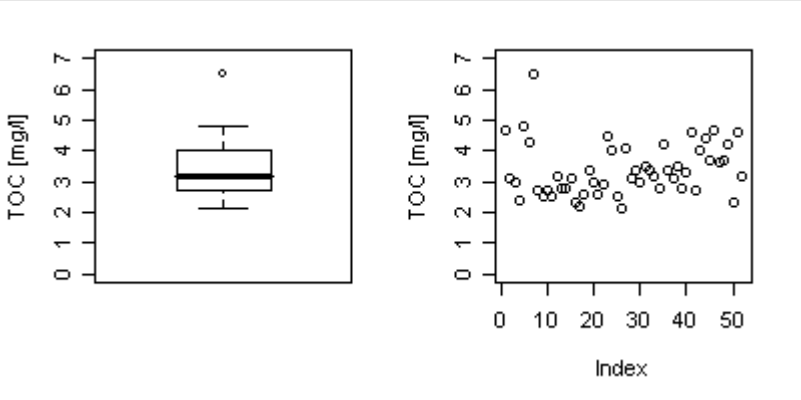
Kapitel 6

1)
```
WOC_PFAD <- "E:/R/3_Daten/Weil_OC.csv"
WOC <- read.table(WOC_PFAD, sep = ";", dec = ",", header = T)
names(WOC); dim(WOC)
head(WOC)

sd(WOC$DOC)/mean(WOC$DOC)
sd(WOC$TOC)/mean(WOC$TOC)
```
Der CV von TOC (0.255) ist grösser als der von DOC (0.221).

2)
```
summary(WOC$DOC)
summary(WOC$TOC)
```

3)
```
WOC_PLOT_PFAD <- "E:/R/temp/WOC_Indexplot.bmp"
bmp(WOC_PLOT_PFAD, width=800, height=400)
   par(mfrow=c(1,2))
   boxplot(WOC$TOC, ylim = c(0,7), ylab = "TOC [mg/l]")
   plot(WOC$TOC, ylim = c(0,7), ylab = "TOC [mg/l]")
dev.off()
```



4)
```
cor(WOC$DOC,WOC$TOC)
```
0.3989867 ~ 0.40

Bemerkung: DOC macht einen beträchtlichen Teil von TOC aus; deshalb ist es nicht besonders erstaunlich, dass eine positive Korrelation zwischen den beiden Grössen besteht.

Kapitel 7

1) 0.6696478

Berechnet mit: dbinom(0,80,1/200)

2) 1-pbinom (2,80,1/200)

Gegenwahrscheinlichkeit zu zwei oder weniger 200-jährliche Hochwasser.

3)
```
EMME_PFAD <- "E:/R/3_Daten/Emmenmatt.csv"
Emme <- read.table(EMME_PFAD, sep = ";", dec = ",", header = T)
names(Emme); dim(Emme)
head(Emme)

Fit_DAUER <- fitdistr(Emme$Dauer,"exponential")
r_DAUER <- as.numeric(Fit_DAUER$estimate[1])
```

4) ks.test(Emme$Dauer,"pexp", r_DAUER)

Die Verteilung kann verwendet werden, um die Häufigkeit der Daten zu beschreiben. Hinweis: Der Kolmogorow-Smirnow ist vor allem bei kleinen Stichproben kein "strenger" Test!

5)
```
MAI_Modell <- lm (HUG$Q06 ~ HUG$QMAI06)

ANN_MAI_PFAD <- "E:/R/temp/Annahmen_MAI_Modell.bmp"
bmp(ANN_MAI_PFAD, width = 600, height = 600)
   par(mfrow = c(2, 2))
   plot(MAI_Modell, which = c(1,2,4,5))
dev.off()
```

Die Annahmen sind (einigermassen) erfüllt.

9. Literaturverzeichnis

1 Bayerisches Landesamt für Umwelt 2008. Hochwassernachrichtendienst. http://www.hnd.bayern.de 1. August 2008.

2 Bundesanstalt für Gewässerkunde 2008. http://www.bafg.de/DE/06__Info__Service/01__Wasserstaende/wasserstaende__node.html, 1. August 2008.

3 Bundesamt für Umwelt (BAFU) 2006. Hydrologisches Jahrbuch der Schweiz. http://www.bafu.admin.ch/php/modules/shop/files/pdf/phpKNnhzt.pdf. 1. August 2009.

4 Bundesamt für Umwelt (BAFU) 2009. Hydrologische Daten von 361 Stationen. http://www.hydrodaten.admin.ch/d/index.htm?lang=de. 1. August 2009.

5 Consortium for Spatial Information (CGIAR-CSI) 2009. http://srtm.csi.cgiar.org/ 1. August 2009.

6 Crawley M. J. 2007. The R book. Wiley & Sons, New York.

7 Environment Canada 2008. Water Survey of Canada. http://www.wsc.ec.gc.ca/hydat/H2O/index_e.cfm?cname=main_e.cfm http://scitech.pyr.ec.gc.ca/climhydro/mainContent/main_e.asp?province=bc, 1. August 2009.

8 Environment Canada 2009. National Climate Data and Information Archive www.climate.weatheroffice.ec.gc.ca/climateData/dailydata_e.html. 1. August 2009.

9 Global Sea Level Observing System (GLOSS) 2009. http://www.gloss-sealevel.org/. 1. August 2009.

10 Grothendieck und Petzoldt 2004. R Help Desk, Date and Time Classes in R. Volume 4/1, June 2004, Seite 29. http://cran.r-project.org/doc/Rnews/ 1. August 2009.

11 Hosking, J. R. M. 1990. L-moments: analysis and estimation of distributions using linear combinations of order statistics. Journal of the Royal Statistical Society, Series B, 52, 105-124.

12 Internationale Kommission zum Schutz des Rheins (IKSR) 2000. Bestandsaufnahme der Phosphor- und Stickstoffeinträge 1996. Bericht Nr. 115d. Internationale Kommission zum Schutz des Rheins. Koblenz. http://www.iksr.org/index.php?id=30. 1. August 2009.

13 Internationale Kommission zum Schutz des Rheins (IKSR) 2009. Gewässergütedaten. http://www.iksr.org/index.php?id=71, 1. August 2009.

14 Kanton Basel-Landschaft 2006. Hydrographisches Jahrbuch. Oberflächengewässer. http://www.hydro-jb.bl.ch/2006/main-oberflaeche.htm, 1. August 2009.

15 Ligges U. 2007. Programmieren mit R. Springer-Verlag, Heidelberg.

16 Landesanstalt für Umwelt, Messungen und Naturschutz, Baden-Württemberg 2008. http://www2.lubw.baden-wuerttemberg.de/public/abt4/fliessgewaesser/daten/index.htm. 1. August 2008.

17 Maidment D.R. 1993. Handbook of Hydrology. McGraw-Hill. New York.

18 National Snow and Ice Data Center (NSIDC) 2009. http://nsidc.org/data/nsidc-0190.html. 1. August 2009.

19 Periodensystem der Elemente 2009. http://www.periodensystem.info/. 1. August 2009.

20 Ricci V. 2005. Fitting Distributions with R. http://www.r-project.org/ 1. August 2009.

21 Statistisches Amt des Kantons Basel-Stadt 2008. http://www.statistik-bs.ch/themen/02. 1. August 2008.

Anhang A: Beispiele aus der Hydrologie

Anhang A enthält kurz gehaltene Beschreibungen zu sechs Anwendungsbeispielen von R in der Hydrologie. Die folgenden Anmerkungen sind als Ergänzung zum in digitaler Form vorhandenen Programmiercode zu verstehen. Die Beispiele 1 bis 3 sind kurz gehalten; Beispiel 6 ist relativ umfangreich. Der Programmiercode ist in Englisch kommentiert. Es werden die folgenden sechs Beispiele vorgestellt:

- Beispiel 1: Datenaggregierung (Code: *Aggregate.R*).
- Beispiel 2: Sekundäre Achse und Niederschlag-Abfluss-Grafik (Codes: *Secondary_Axis.r* und *Precipitation_Runoff_Plot.r*).
- Beispiel 3: ASCII Dateien importieren und exportieren (Code: *ASCII_read_write.r*)
- Beispiel 4: Topographische Kenngrössen und Datenmanagement (Code: *Topography_Example.r*)
- Beispiel 5: Zeitreihen und Datenaufbereitung (Code: *Time_Series_Example.r*)
- Beispiel 6: Hydrologische Modellierung (Modellstrukturen, Modell-optimierung, lokale und regionale Sensitivitätsanalyse, Unsicherheits-analyse; Code: *Modelling_Example.r*)

Beispiel 1: Datenaggregierung

Im vorliegenden Beispiel werden Temperaturdaten der Station „Amundsen-Scott“ am Südpol für das Jahr 2001 analysiert. Die Daten werden vom National Snow and Ice Data Center (NSIDC, 2009) bezogen, und liegen als Mittelwerte über sechs Stunden vor. Aus diesen Daten werden Temperaturmittelwerte für die folgenden Zeitintervalle berechnet:

- 12 Stunden.
- Tage.
- Wochen.
- Monate.

Die Aggregierung wird mit der *aggregate*-Funktion durchgeführt. Als Alternative dazu kann auch der *tapply*-Befehl verwendet werden. Die Resultate der Aggregierung werden grafisch dargestellt, und die neuen Datensätze exportiert.

Beispiel 2: Sekundäre Achse und Niederschlag-Abfluss-Grafik

Sekundäre Achse

In einem ersten Schritt wird eine sekundäre Achse zu einer einfachen Grafik hinzugefügt. Dazu werden Daten und Achsenabschnitte der sekundären Achse auf die primäre Achse skaliert.

Niederschlag-Abfluss-Grafik

Im zweiten Schritt werden tägliche Niederschlags- und Abflussdaten eines 93.3km^2 grossen Einzugsgebiets bei Whistler (Kanada) des Jahres 2008 in einer Grafik dargestellt. Dazu werden Niederschlagsdaten der Station „Whistler Mountain High Level” (Environment Canada, 2009b) und Abflussdaten der Station „Fitzsimmons Creek below Blackcomb Creek“ (Environment Canada, 2009a) verwendet.

Die Kenntnisse über das Einfügen einer sekundäre Achse werden benötigt, um die Niederschlagsdaten in der Abflussgrafik darzustellen. Der Niederschlag wird in Form von Polygonen (vom oberen Grafikrand gegen unten) dargestellt.

Beispiel 3: ASCII Dateien importieren und exportieren

Dateien

Der Umgang mit ASCII Dateien wird mit zwei Dateien geübt:

- *testgrid.asc* (kleine Datei mit 10 Werten; selbst erstellt).
- *dem.asc* (relativ grosse Datei mit 90140 Werten; digitales Höhenmodell des Einzugsgebiets des „San Juan River“ in Kanada (CGIAR-CSI, 2009). Dieser Datensatz wird in Beispiel 4 genauer behandelt.

Code-Inhalte

Es werden die folgenden Schritte durchgeführt:

- Importieren der .asc Datei in R.
- Extrahieren der Daten (ohne fehlende Werte).
- Durchführen einfacher Berechungen.
- Einfügen der Berechnungen in ein neues Objekt (inklusive fehlenden Werten).
- Berechnungen als .asc-Datei exportieren.

Diese Schritte werden zuerst mit der kleinen Datei, anschliessend mit dem grösseren Höhenmodell durchgeführt.

Beispiel 4: Topographische Kenngrössen und Datenmanagement

Für die drei Einzugsgebiete „San Juan River“, „Shawnigan Creek“ und „South Nanaimo River“ auf Vancouver Island (Kanada) sollen topographische Kenngrössen wie Steigung, Aspekt und topographischer Index bestimmt werden. Als Vorbereitung wurden dazu die Einzugsgebiete und dazugehörigen Höhenmodelle anhand der Koordinaten der Abflussmessstationen (gemäss Environment Canada 2009a) bestimmt. Dann werden die topographischen Grössen zuerst manuell (nur für ein Einzugsgebiet) und danach mit einem R-Code (für drei Einzugsgebiete) berechnet.

SAGA GIS

SAGA (System for Automated Geoscientific Analyses) ist ein kostenloses Geographisches Informations System (GIS), das besonders für Analysen mit Rasterdaten sehr geeignet ist. Informationen zu SAGA GIS sind auf www.saga-gis.org/en/index.html zu finden. Für das vorliegende Beispiel, muss SAGA zuerst in den Ordner *E:\R\7_Beispiele\04_Beispiel_4\SAGA* herunter geladen werden. Die entsprechende zip-Datei (saga_2.0.4_bin_mswvc.zip) befindet sich unter http://sourceforge.net/projects/saga-gis/files/. SAGA muss danach nicht auf dem Computer installiert werden; das Herunterladen und Entzippen ist ausreichend. Die vorliegenden Analysen wurden mit der SAGA Version 2.0.4 durchgeführt.

Download des Höhenmodells

Die in diesem Beispiel verwendeten digitalen Höhenmodelle beruhen auf dem SRTM (Shuttle Radar Topography Mission) Höhenmodell von Consortium for Spatial Information“ (CGIAR-CSI, 2009). Es weist eine Zellgrösse von drei Bogensekunden auf, und wurde im ASCII-Format heruntergeladen (Kartenausschnitt: srtm_12_03).

Datenaufbereitung

Mit SAGA 2.0.4 wurde das Höhenmodell wie folgt aufbereitet:

SAGA Modul	*Beschreibung*
Import/Export – Grids > Import ESRI Arc/Info Grid	DHM importieren.
Grid – Tools > Cutting [interactive]	DHM auf relevanten Auschnitt verkleinern, damit die folgenden Rechnungen schneller werden.
Projection – Proj4 > Proj4 (Grid)	Projektion von ändern; Koordinatenangaben in Längen- und Breitengraden, sowie die Einheit „Bogensekunde“ sind nicht sehr anschaulich. Deshalb wird die Projektion des Höhenmodells auf UTM 10N (WGS 84) geändert, dessen Einheit „Meter“ ist. Dazu wird das Höhenmodell auf ein 80m Raster interpoliert.
Terrain Analysis – Hydrology > Parallel Processing	*Catchment Area* berechnen. Die D-Infinity Methode entspricht eher einem realen Fliessverhalten als die D8 Methode. Für die Einzugsgebietsausscheidung wird aber eine Eineindeutigkeit vorausgesetzt, weshalb hier D8 verwendet wurde.
Terrain Analysis – Preprocessing > Sink Drainage Route Detection, Sink Removal	Preprocessing; Senken füllen.
Terrain Analysis – Hydrology > Upslope Area [interactive]	Einzugsgebietsausscheidung für die drei Abflussmessstationen. Es wird der „interactive“ Modus verwendet, damit anhand der „catchment area“ der nächste Fliessweg bei der entsprechenden Station angewählt werden kann. Es wird wiederum die D8-Methode verwendet. Dieser wie die folgenden Schritte werden für die drei Einzugsgebiete durchgeführt.
Grid – Calculus > Grid Calculator	Multiplizieren des standardisierten Einzugsgebiets mit dem Höhenmodell: (a/a)*b
Grid - Tools > Crop to Data	Gridgrösse auf den Datenbereich reduzieren.
Import/Export – Grids > Export ESRI Arc/Info Grid	Als .asc Datei exportieren.

Manuelle Berechnung

Mit den folgenden Befehlen werden Steigung, Aspekt und topographischer Index manuell in SAGA berechnet. Die manuelle Berechnung wird anhand des San Juan Rivers durchgeführt. Zuerst werden Steigung und Aspekt berechnet.

SAGA Modul	*Beschreibung*
Import/Export – Grids > Import ESRI Arc/Info Grid	Digitales Höhenmodell importieren.
Terrain Analysis – Morphometry > Local Morphometry	Steigung(in Grad) und Aspekt (in Grad, 0° = Norden, im Uhrzeigersinn) berechnen
Import/Export – Grids > Export ESRI Arc/Info Grid	Steigung und Akpekt als .asc Datei exportieren.

Der topographische Index wird wie folgt ermittelt:

SAGA Modul	*Beschreibung*
Terrain Analysis – Preprocessing > Sink Drainage Route Detection, Sink Removal	Preprocessing; Senken füllen (eventuell mit dem Gridcalculator betrachten, wo Senken gefüllt worden sind).
Terrain Analysis – Morphometry > Local Morphometry	Steigung(in Grad) des korrigierten Höhenmodells berechnen (slope = create). Zuvor die zuvor erstellten Steigungs- und Aspekt-Layer umbenennen (beispielsweise Slope1 und Aspect1).
Grid – Calculus > Grid Calculator	Für Steigungen = 0 ist der topographischen Index nicht definiert. Deshalb werden zur Steigung 2 Grad addiert. Im Gridcalculator wird die Steigung in Radian umgewandelt, weshalb nicht 2 sondern 2/180*pi = 0.0349 addiert wird.
Terrain Analysis – Hydrology > Parallel Processing	"Catchment Area" berechnen. Für die Berechnung des topographischen Index werden realistische Fliessmuster benötigt, weshalb die D-Infinity Methode verwendet wird.
Grid – Calculus > Grid Calculator	Berechnung des topographischen Index mit der „catchment area" und der korrigierten Steigung. Formel: ln(a/tan(b)
Import/Export – Grids > Export ESRI Arc/Info Grid	Topographischer Index als .asc Datei exportieren.

Berechnung mit R

Analog zur manuellen Berechnung werden mit R ebenfalls Steigung, Aspekt und topographischer Index berechnet. Mit einer Schleife werden diese Dateien für die drei Einzugsgebiete berechnet. Dabei ist für jedes Einzugsgebiet ein Ordner erstellt worden; die Dateien in diesen Ordnern werden bei allen Einzugsgebieten mit gleichen Namen versehen (beispielsweise dem.asc). Dadurch kann bei allen Einzugsgebieten der gleiche Programmiercode verwendet werden, nachdem die Dateien mit der Schleife in einen Arbeitsordner kopiert worden sind.

Datenmanagement

Um Dateien effizient zu verwalten werden in R unter anderem die folgenden Befehle verwendet:

- Dateinamen eines Ordners ermitteln: list.files
- Dateien kopieren: file.copy
- Dateien entfernen: file.remove
- Ordner (mit Inhalt) entfernen: unlink
- Ordner neu erstellen: dir.create

Beispiel 5: Zeitreihen und Datenaufbereitung

Fragestellung

In diesem Beispiel soll diskutiert werden, ob der Meeresspiegel in Gan auf den Malediven seit 1987 angestiegen ist. Diese Frage ist insofern relevant, als dass der höchste Punkt der Malediven sich auf 2.4m NN befindet. Es wird ein Datensatz mit täglichen mittleren Meeresspiegelhöhen von März 1987 bis Mai 2009 verwendet (GLOSS, 2009).

Datenaufbereitung

In einem Schritt sollen die Originaldaten (Datei d109.dat) in ein für Zeitreihenanalysen geeignetes Format gebracht werden. Dazu gehören das korrekte Kennzeichnen von fehlenden Werten sowie das Erzeugen von Zeitreihenformaten der *zoo*-Klasse.

Methoden

Um eine mögliche Erhöhung des Meeresspiegels zu diskutieren, werden die folgenden Methoden verwendet:

- Graphische Methoden (einfache lineare Regression, Sen).
- Hypothesentests (einfache lineare Regression, Tests nach Mann-Kendall, Kendall, Spearmans Rangkorrelation).
- Bruchpunktanalyse nach Mann-Whitney
- Moving Windows (für Mittelwerte, Varianz, Trend-Kenngrössen)

Beispiel 6: Hydrologische Modellierung

Modell

Im folgenden Beispiel wird der tägliche Abfluss eines Einzugsgebiets anhand von täglichen Niederschlags- und Temperaturdaten abgeschätzt. Das dabei verwendete Modell besteht vor allem aus einer einfachen Schneeschmelze-Routine. Es werden dabei die folgenden Inputdaten, Parameter und Variablen verwendet:

Bezeichnung	*Abk.*	*Beschreibung*	*Einheiten*
Inputdaten	TEMP	Temperatur (Tagesdurchschitt)	[°C]
	PREC	Niederschlag (Tagessumme)	[mm/d]
Parameter	TA	Akkumulationstemperatur	[°C]
	TM	Schmelz-Temperatur	[°C]
	MF	Schmelzfaktor	[mm/°C]
	KS	Rezessionskoeffizient	[1/d]
Variabeln	ACCU	Schneeakkumulation	[mm/d]
	PACK	Schneedecke	[mm]
	MPOT	Potentielle Schneeschmelze	[mm/d]
	SINP	Input in den Bodenspeicher	[mm/d]
	STOR	Speicherinhalt	[mm]
	QSIM	Simulierter spezifischer Abfluss	[mm/d]
Daten für Modellevaluation	QOBS	Beobachter spezifischer Abfluss	[mm/d]

Einzugsgebiet

Das oben stehende Modell wird auf Mackay Creek Einzugsgebiet bei Vancouver (Kanada) angewendet. Bei einer Einzugsgebietsfläche von 3.43km² variert die Höhe von 159 bis 1108m NN. Das Einzugsgebiet ist hauptsächlich bewaldet, etwa zur Hälfte mit einem urbanan Wald. Mit dem obenstehenden Modell wird der mittlere tägliche Abfluss von der Zeit vom 1. Oktober 2006 bis 30. September 2007 simuliert. Die verwendeten Daten stammen von der Abflussmessstation „Mackay Creek“ (Environment Canada, 2009a) sowie der Klimastation „North Vancouver Grouse Mountain Resort“ (Environment Canada, 2009b). Die Klimastation liegt leicht oberhalb des Einzugsgebiets auf 1128m NN.

Methoden

Es wurden die folgenden Methoden angewendet:

- Modelloptimierung mit der *optim*- sowie der *genoud*-Funktion.
- Sensitivitätsanalyse (lokale Sensitivitätsanalyse, Kollinearitätsindizes, regionale Sensitivitätsanalyse nach Sobol).
- Unsicherheitsanalyse mit GLUE (Generalized Likelihood Uncertainty Estimation).

Anhang B: R und TINN-R installieren

R herunterladen

- Zuerst erstellen Sie auf ihrem Computer einen Ordner mit dem Namen R. Es wird empfohlen, diesen Ordner unter *C:/Programme* zu erstellen.
- Auf der linken Seite der R-Homepage (*www.r-project.org*) finden Sie den *Download*-Bereich. Dort klicken Sie auf *CRAN*.
- Um die Datenübertragung zu optimieren wählen Sie einen möglichst nahegelegenen Mirror. Deshalb klicken Sie unter *Switzerland* auf den Mirror *http://cran.ch.r-project.org/*.
- Unter *Download and Install R* wählen Sie das Betriebssystem Ihres Computers, beispielsweise *Windows*. In diesem Leitprogramm ist bloss das Vorgehen für *Windows*-Betriebssysteme beschrieben.
- Sie klicken nun auf *base* um zur Webseite zu gelangen, wo die Hauptversion von R heruntergeladen werden kann.
- Klicken Sie dort auf R-2.11.1-win32.exe und speichern Sie die Installationsdatei in Ihrem R-Ordner (beispielsweise unter *C:/Programme/R*). Das Herunterladen dauert etwa zwei Minuten.

R installieren

- Doppelklicken Sie auf die heruntergeladene *R-2.11.1-win32.exe* Datei und wählen Sie *run*. Danach folgen Sie mehrheitlich den automatisch definierten Vorgaben.
- Wählen Sie *Deutsch* als Installationssprache.
- Folgen Sie dem Installationsassistenten indem Sie zweimal *weiter* klicken.
- Suchen Sie eine geeignete Stelle für das Programm (beispielsweise unter *C:/Programme/R/R-2.11.1*).
- Es wird empfohlen, dass Sie die vorgeschlagenen Programm-Komponenten übernehmen und auf *weiter* zu klicken.
- Bei der Frage *Möchten Sie die Startoptionen anpassen* wählen Sie *nein*.
- Klicken Sie danach auf *weiter*, damit die Programmverknüpfungen im R-Ordner erstellt werden.
- Deaktivieren Sie *Desktop-Symbol erstellen*, und klicke Sie danach auf *weiter*. R wird jetzt installiert. Das dauert etwa eine Minute.
- Beenden Sie den Installationsassistenten indem Sie auf *Fertigstellen* klicken.

TINN-R herunter laden

- Gehen Sie auf die folgende Webseite: *www.sciviews.org/Tinn-R/*
- Im Downloadbereich im unteren Teil der Webseite klicken Sie auf *Setup for Tinn-R, old stable version (1.17.2.4)*.
- Klicken Sie auf auf *Save*, und speichern Sie die Installationsdatei an einem Ort, wo Sie diese schnell wieder finden (beispielsweise auf dem Desktop). Das Herunterladen dauert etwa eine Minute.

TINN-R installieren

- Doppelklicken Sie auf die Datei *Tinn-R 1.17.2.4 setup.exe* und wählen Sie *Run*.
- Klicken Sie *Next*; wählen Sie *I accept the agreement* um die Lizenzvereinbarung zu akzeptieren, und klicken Sie auf *Next*.
- Wählen Sie einen geeigneten Ort, wo das Programm installiert wird (beispielsweise *C:\Programme\Tinn-R*), und klicken Sie auf *Next*.
- Belassen Sie den Namen unter dem TINN-R im Startmenu erscheinen wird als *TINN-R*, und klicken Sie auf *Next*.
- Übernehmen Sie die Vorgaben und klicken Sie auf *Next* und danach *Install*. Die Installation dauert etwa zehn Sekunden.
- Deaktivieren Sie *Lauch TINN-R*, weil das Programm erst später gestartet werden soll, und klicken Sie auf *Finish*.
- Löschen Sie die Installationsdatei (*Tinn-R 1.17.2.4 setup.exe*), die Sie zu Beginn heruntergeladen haben. Sie befindet sich beispielsweise auf dem Desktop.

Anhang C: Datenquellen

Im vorliegenden Leitprogramm wurden die folgenden, öffentlichen Datenquellen verwendet:

Dateiname	*Beschreibung*	*Quelle*
890090_2001.dat	Temperatur in Grad Celsius (6-Stunden Mittelwerte) der Station Amundsen-Scott im Jahr 2001.	National Snow and Ice Data Center (NSIDC, 2009)
Birsig.csv	Tagesmittel des Abflusses der Birsig in Binningen im April 2006 in m^3/s.	Kanton Basel-Landschaft (2009)
d109.dat	Tägliche Meerespiegelhöhe in Gan auf den Malediven vom 1. März 1987 bis 31. Mai 2009	Global Sea Level Observing System (GLOSS, 2009)
DATA_MACKAY.csv	Tägliche mittlere Temperatur [°C] und täglicher Niederschlag [mm] für die Station „North Vancouver Grouse Mountain Resort“. Abfluss des Mackay Creek [m^3/s]. Alle Daten vom 1. Oktober 2006 bis 30. September 2007.	Environment Canada (2009a) Environment Canada (2009b)
dem.asc	Digitales Höhenmodell; Projektion GCS_WGS_1984; 3-BogensekundenRaster; erfasst durch Shuttle Radar Topography Mission (SRTM) im Jahr 2000.	Consortium for Spatial Information (CGIAR-CSI, 2009)
DOC.csv	Gelöster organischer Kohlenstoff im Rhein bei Weil am Rhein im Jahr 2003 (14-tägige Werte).	Internationale Kommission zum Schutz des Rheins (IKSR, 2008)
eng-daily-01012008-12312008.csv	Klimadaten der Station „Whistler Mountain High Level“ im Jahr 2008 (Tagesmittelwerte)	Environment Canada (2009b)
Elemente.csv	Molekulargewicht der Elemente in g/mol.	Periodensystem der Elemente (2009)
Emmenmatt.csv	Jahre in denen die Emme in Emmenmatt (DB-Nr. 1100) ein Abfluss von 200 m^3/s erreicht oder überschreitet. Zudem: Dauer in Jahre bis zum letzten Jahr mit einem Abfluss von mindestens 200 m^3/s.	Bundesamt für Umwelt (BAFU, 2009)
flowDaily__Nov-19-2009_08_01_19AM_ _ddf.csv	Abflussdaten der Station “Fitzsimmons Creek below Blackcomb Creek” (Stationsnummer: #08MG026) vom 7. Juli 1993 bis 31. Dezember 2008 (Tagesmittelwerte in m^3/s).	Environment Canada (2009a)
Grundwasser.csv	Grundwasser in „Lange Erlen“ (BS). Mittelwert des Grundwasserstandes in Zentimeter über dem Basler Nullpunkt. Monatsmittelwerte 2000 bis 2007.	Statistisches Amt des Kantons Basel-Stadt (2009)

Dateiname	*Beschreibung*	*Quelle*
HydroUG.csv	20 hydrologische Untersuchungsgebiete der Schweiz. Kenngrössen: GEW (Gewässername), HOEHE (mittlere Einzugsgebietshöhe), WALD (Waldanteil in %), GLET (Gletscheranteil in %), SEE (Seeanteil in %), DURCH (mittlere Durchlässigkeit in cm/s), GEWD (Gewässernetzdichte in km/km^2), BEVD (Bevölkerungsdichte in Personen/km^2), P06 (Gebietsniederschlag 2006 in mm), PMAI06 (Gebietsniederschlag Mai 06 in mm), Q06 (Gebietsabfluss 2006 in mm), QMAI06 (Gebietsabfluss im Mai 06 in mm).	Bundesamt für Umwelt (BAFU, 2006) Bundesamt für Umwelt (BAFU, 2009)
Ionen_2004.csv	Ionenkonzentration in mg/l und Abfluss in m^3/s des Rheins bei Weil am Rhein im Jahr 2004 (14-tägige Werte).	Internationale Kommission zum Schutz des Rheins (IKSR, 2009)
Lindau_Pegel.csv	Pegel des Bodensees in Lindau [m] vom 27. bis 30. Juli 2008 (Stundenwerte).	Bayerisches Landesamt für Umwelt (2008)
Magnesium.xls	Magnesiumkonzentration in mg/l im Rhein bei Weil am Rhein im Jahr 2004 (14-tägige Werte).	Internationale Kommission zum Schutz des Rheins (IKSR, 2009)
Messgrössen.xls	Auflistung der Messgrössen, mit welchen der Rhein bei Weil am Rhein beschrieben wird. Für zwei Stichtage (12. Juli 2004 und 13. Dezember 2004) sind Messwerte angegeben.	Internationale Kommission zum Schutz des Rheins (IKSR, 2009)
Nass_River.csv	Maximale instantane Jahresabflüsse des Nass River (Stationsnummer: 08DB001); nicht reguliertes Gewässer; 39 Werte von 1966 bis 2006.	Environment Canada (2009a)
Nationen.csv	Fläche (in km^2) und Bevölkerung (in Millionen) der Rheinanliegerstaaten (Schweiz, Deutschland, Frankreich, Luxemburg, Niederlande) im Jahr 1996.	Internationale Kommission zum Schutz des Rheins (IKSR, 2000a)
pH.xls	pH-Wert im Rhein bei Weil am Rhein im Jahr 1997 (14-tägige Werte).	Internationale Kommission zum Schutz des Rheins (IKSR, 2009)
Phoshor_1996.csv	Phosphoreintrag in den Rhein 1996, gegliedert nach Ländern und Eintragsherkunft.	Internationale Kommission zum Schutz des Rheins (IKSR, 2000a)
Pine_River.csv	Maximale instantane Jahresabflüsse des Pine River (Stationsnummer: 07FB001); nicht reguliertes Gewässer; 26 Werte von 1978 bis 2006.	Environment Canada (2009a)
Rheinfelden_Leitf.csv	Elektrische Leitfähigkeit (µS/cm) des Rheins am 10. Juli 2008 in Rheinfelden (Stundenwerte).	Landesanstalt für Umwelt, Messungen und Naturschutz, Baden-Württemberg (2008)
Rheinfelden_Pegel.csv	Pegelstand des Rheins am 2. Juli 2008 in Rheinfelden (in cm, 15-Minuten Werte).	Bundesanstalt für Gewässerkunde (2008)
Rheinfelden_Temp.csv	Temperatur (°C) des Rheins am 1. Juli 2008 in Rheinfelden (Stundenwerte).	Landesanstalt für Umwelt, Messungen und Naturschutz, Baden-Württemberg (2008)

Dateiname	*Beschreibung*	*Quelle*
Rheinfelden_Temp2.csv	Temperatur (°C) des Rheins am vom 27. bis 30. Juli 2008 in Rheinfelden (Stundenwerte).	Landesanstalt für Umwelt, Messungen und Naturschutz, Baden-Württemberg (2008)
Romanshorn.csv	Wasserstand des Bodensees (Tagesmittel) bei Romanshorn in m ü.NN vom 1. bis zum 31. August 2006.	Bundesamt für Umwelt (BAFU, 2006)
Schwermetalle_2003.csv	Schwermetallkonzentrationen in µg/l im Rhein bei Weil am Rhein im Jahr 2003 (14-tägige Werte).	Internationale Kommission zum Schutz des Rheins (IKSR, 2009)
Skeena_River.csv	Maximale instantane Jahresabflüsse des Skeena River (Stationsnummer: 08EF001); nicht reguliertes Gewässer; 51 Werte von 1953 bis 2006.	Environment Canada (2009a)
testgrid.asc	Selbst erstellte .asc-Datei mit zehn Datenpunkten sowie zwei fehlenden Werten.	-
Weil_O2.csv	Gelöster Sauerstoff (mg/l) des Rheins bei Weil am Rhein. Zudem: Temperatur (°C), Abfluss (m^3/s) sowie das Messdatum. Werte alle 14 Tage der Jahre 2003 und 2004.	Internationale Kommission zum Schutz des Rheins (IKSR, 2009)
Weil_OC.csv	Gelöster und gesamter organischer Kohlenstoff im Rhein bei Weil am Rhein in den Jahren 2001 und 2002 (14-tägige Werte).	Internationale Kommission zum Schutz des Rheins (IKSR, 2009)

Anhang D: Kleines Wörterbuch

Kleines Wörterbuch

Thema	Beschreibung	R	Matlab	IDL
Arbeitsverzeichnis	Arbeitsverzeichnis festlegen	**setwd("C:/Benutzer")**	**path('C:/user')**	*File/preferences/startup/working directory (über die Menüleiste)*
Objekte	Objekte definieren	**a <- 5**	**a = 5**	**a = 5**
	Ausgabe von Objekt a	**a** oder **print(a)**	**a** *oder* **disp(a)**	**print, a**
	Objekte löschen	**rm()**	**clear**	**delvar**
Operatoren	Arithmetische Operatoren	**+** *(Addition)* **-** *(Subtraktion)* ***** *(Multiplikation)* **/** *(Division)* ^ (Potenz)	**+** *(Addition)* **-** *(Subtraktion)* ***** *(Multiplikation)* **/** *(Division)* ^ (Potenz)	**+** *(Addition)* **-** *(Subtraktion)* ***** *(Multiplikation)* **/** *(Division)* ^ (Potenz)
	Logische Operatoren	**==** *(gleich)* **!=** *(ungleich)* **<** *(kleiner)* **>** *(grösser)* **<=** *(kleiner-gleich)* **>=** *(grösser-gleich)* **&** *(UND)* | *(UND/ODER)*	**==** *(gleich)* **~=** *(ungleich)* **<** *(kleiner)* **>** *(grösser)* **<=** *(kleiner-gleich)* **>=** *(grösser-gleich)* **&** oder && *(UND)* | oder || *(ODER)*	**eq** (gleich) **ne** (ungleich) **lt** (kleiner) **gt** (grösser) **le** (kleiner-gleich) **ge** *(grösser-gleich)* **and** *(UND)* **or** *(UND/ODER)*
Vektoren	Verketten	**c(3, 5, 7)**	**c=[3,5,7]** *oder* **c=[3;5;7]**	**a[3, 5, 7]**
	Sequenz	**1:10** *oder* **seq()**	**1:10**	**1:10**
	Replikation	**rep(3, 10)**	**10*ones(3)**	**replicate(10,3,3)**
	Länge eines Vektors	**length()**	**length()**	**n_elements()**
Matrizen	Definition	**matrix()**	**A = [1,2;3,4]**	**array = [1,2,3]**
	Matrix transponieren	**t(a)**	**A'**	**transpose(a)**
	Zeilen/Spalten anhängen	**rbind(), cbind()**	**A=[A c]; c=[5;6]**	**rebin()**
Indizieren	Vektor-Index	**a[1]**	**c(1)**	**a[1]**
	Matrix-Index: Erste Zeile	**a[1,]**	**A(1,)**	**a[1,*]**
	Matrix-Index: Erste Spalte	**a[, 1]**	**A(,1)**	**a[*,1]**
Datenanalyse	Logische Abfragen	**a > 3** *(Resultat: TRUE, FALSE)*	**a > 3** *(Resultat: 1, 0)*	**a > 1** *(Resultat 1,0)*
	Index-Abfrage	**which(a > 3)**	**find(a > 3)**	**where (a > 3)**
	Daten sortieren	**sort(), order(), rank()**	**sort()**	**sort()**
Datenimport/-export	Datenimport	**read.table()**	**load**	**readf**
	Datenexport	**write.table()**	**dlmwrite("filename", A)**	**printf**

Thema	Beschreibung	R	Matlab	IDL
Zeichenfolgen	Verketten	**paste()**	**[char1 char2]**	**AB = `A` + `B`**
	Teil einer Zeichenfolge	**substring()**	**char(:)**	**strput()**
	Anzahl Zeichen	**nchar**	**length(char)**	**strlen()**
Konstanten	pi	**pi()**	**pi()**	**!pi**
	e	**exp(1)**	**exp(1)**	**exp(1)**
	Unendlich	**inf**	**inf**	**inf**
	Fehlende Werte	**NA**	**NaN**	**NaN**
Funktionen	Wurzel	**sqrt()**	**sqrt()**	**sqrt()**
	Mittelwert	**mean()**	**mean()**	**mean()**
	Median	**median()**	**median()**	**median()**
	Standardabweichung	**sd()**	**std( )**	**stddev()**
	Varianz	**var()**	**var()**	**variance()**
	Minimum, Maximum	**min(), max()**	**min(), max()**	**min(), max()**
	Summe	**sum()**	**sum()**	**total()**
	Produkt	**prod()**	**prod()**	**a*b**
	Betrag	**abs()**	**abs()**	**abs()**
	Runden, wie wird gerundet?	**round()**, *rundet mathematisch*	**round()**, *rundet kaufmännisch*	**round()**, *rundet kaufmännisch*
	Aufrunden	**ceiling()**	**ceil()**	**ceil()**
	Abrunden	**floor()**	**floor()**	**floor()**
	Exponentialfunktion	**exp()**	**exp()**	**exp()**
	Natürlicher Logarithmus	**log()**	**log()**	**alog()**
	Zehner-Logarithmus	**log10()**	**log10()**	**alog10()**
	Sinus, Arcsinus	**sin(), asin()**	**sin(), asin()**	**sin(), asin()**
	Cosinus, Arccosinus	**cos(), acos()**	**cos(), acos()**	**cos(), acos()**
	Tangens, Arctangens	**tan(), atan()**	**tan(), atan()**	**tan(), atan()**
Zeit/Datum	Zeit/Datum-Formate	*Date, POSIXct*	*date*	*Julian date, Gregorian date*
	Referenz-Zeitpunkt	*1. Januar 1970*	*1.Januar 0000*	*1. Januar 1970*
	Zeiteinheit	*Tage (im Date-Format)* *Sekunden (im POSIXct-Format)*	*Tage*	*Tage*
Schleifen	for-Schleife	**for (i in 1:5) {x[i]}**	**for i=1:5**	**for k=1,10 do …**
	if/else	**if (x > 5) {y <- 3} else y <-2**	**if x > 5; y=-3; else y=-2; end**	**If k eq 10 then … else …**
Zeichnen	Plot-Funktion	**plot(x, y)**	**plot(x, y)**	**plot, x, y**
Hilfe	Hilfe (z.B. Runden-Funktion)	**?round**	**help round**	**? round**
„case sensitivity“	Gross- und Kleinschreibung?	*ja*	*ja*	*nein*
Kommentare	Zeichen vor Kommentaren	**#**	**%**	**;**