Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Funktionen und Formeln in R

Ausarbeitung zum Proseminar Programmierung in R

vorgelegt von Daniel Laskow
Betreuer: Eugen Betke
Hamburg, 27. Juli 2016

Inhaltsverzeichnis

1 Abstract

Formeln sind ein wichtiger Bestandteil der Programmiersprache R, ob damit normale mathematische Formeln gemeint sind oder spezielle Formeln mit denen man Daten in Abhängigkeit setzen kann. Sie werden verwendet um Werte zu berechnen, Beeinflussungen herauszufinden oder um Werte zu visualisieren.
In dieser Ausarbeitung geht es darum, wie man Formeln erstellt und wie man sie in R verwenden kann. Im ersten Kapitel geht es um arithmetische Operatoren und welche Operationen auf welche Typen angewendet werden dürfen. Dann werden einige wichtige und typische in R verfügbare Funktionen gezeigt. Im nächsten Teil wird die Syntax und der Aufbau einer Funktion erklärt. Als letztes in diesem Kapitel wird gezeigt, wie man - nachdem man eine Funktion geschrieben hat - die Ergebnisse dieser Funktion graphisch darstellen kann.
Im zweiten Kapitel geht es um eine andere Art Formeln, welche man auch häufig in R benutzt. Es geht dabei um den Typ „formula“, was man mit ihm machen kann und wie man ihn benutzt, um unterschiedliche Aufgaben wie z.B Regressionsverfahren zu erledigen.

2 Mathematische Funktionen

In diesem Teil geht es um mathematische Funktionen und wie man mathematische Formeln in R mithilfe von Funktionen implementieren kann.

2.1 Arithmetische Operatoren

Um Rechnungen in R durchführen zu können, stellt R einige arithmetische Operatoren zur Verfügung, diese sind:

Operator	**Beschreibung**	**Beispiel**	**Ausgabe**
+ - * /	Addition, Subtraktion usw.	10/3	[1] 3.333333
** oder ^	Potenzieren	2^3	[1] 8
% \%	Ganzzahlige Division	10 % \% 4	[1] 2
%%	Modulo	10% % 3	[1] 1

Man kann sie verwenden, um Rechnungen in der Konsole einzugeben und sich das Ergebnis ausgeben zu lassen oder um sie in Funktionen zu benutzen, um damit größere Rechnungen durchzuführen. Dabei kann R nicht nur mit Typen wie Integer und Double rechnen, sondern auch mit Listen in denen Zahlen enthalten sind, wie z.B. ein Vektor oder eine Matrix. Einige Beispiele sieht man hier:

```
1 >vektor=c(1,2,3,4)
2 >vektor^2
3 [1]  1  4  9 16
4 >m=matrix(vektor, nrow=2, byrow=
    ↪ TRUE)
5 >m*5
6      [,1] [,2]
7 [1,]    5   10
8 [2,]   15   20
```

Listing 1: Mit Vektoren und Matrizen rechnen

```
> y = matrix(c(1,2,3,4), nrow
   ↪ =2)
> y*y

     [,1] [,2]
[1,]    1    9
[2,]    4   16
> y^y
     [,1] [,2]
[1,]    1   27
[2,]    4  256
```

Listing 2: Matrizen berechnen

Im Listing 1 wird gezeigt, wie man einen Vektor erstellt und alle Elemente in diesem Vektor mit 2 potenziert werden. Die Matrix `m` in Beispiel 1 enthält alle Elemente aus der

Variable **vektor**. Mit ihr kann man auf die gleiche Weise Rechnungen durchführen. Es ist wie in Listing 2 zu erkennen auch möglich Matrizen miteinander zu multiplizieren und zu potenzieren, solange die beiden Matrizen die gleiche Anzahl an Elementen und Reihen haben. Der Ergebnistyp dieser Rechenoperationen ist der Typ der Liste mit der gerechnet wird. Wenn man mehrere Typen von Listen für eine Rechnung benutzt, wird das Ergebnis vom Typ der flexibleren Liste sein. Z.B. wenn man einen Vektor mit einem Array multipliziert, ist das Ergebnis vom Typ Array. Wenn man einen Vektor mit einem Dataframe multipliziert, kommt wieder ein Dataframe heraus. Allerdings sind nicht alle Listen miteinander kompatibel was Rechnungen angeht, z.B. kann man nicht Matrizen und Arrays in der gleichen Rechnung benutzen.

2.2 Wichtige Funktionen aus der Statistik

Im Folgenden sind ein paar häufig benutzte Funktionen gelistet, die bereits in R implementiert sind und mit denen man einfache Rechnungen, die man in der Statistik und in anderen Bereichen aus der Mathematik braucht, durchführen kann.

Funktion	**Beschreibung**	**Beispiel für x=c(1,2,3,4,5,10)**
mean(x)	Arithmetisches Mittel (Durchschnitt)	[1] 4.166667
median(x)	Median (das mittlere Element einer Liste)	[1] 3.5
var(x)	Varianz (durchschnittliche quadratische Abweichung vom Mittelwert)	[1] 10.16667
sd(x)	Standardabweichung (durchschnittliche Abweichung vom Mittelwert)	[1] 3.188521
cumsum(x)	Alle Elemente bis zum nten Glied summiert	[1] 1 3 6 10 15 25
cumprod(x)	Alle Elemente bis zum nten Glied multipliziert	[1] 1 2 6 24 120 1200

Alle diese Funktionen erwarten als Parameter eine Liste mit Zahlen. Darüber hinaus gibt es noch viele weitere grundlegende Funktionen in R, wie z.B. Die Logarithmus-Funktion: **log(x)**, die Betragsfunktion: **abs(x)** und die üblichen trigonometrischen Funktionen: **sin(x)**, **cos(x)**, **tan(x)**, **acos(x)** ... .

2.3 Grundlegendes zu Funktionen

Anders als in anderen gängigen Programmiersprachen haben Funktionen in R einen Typ. Dieser Typ nennt sich „function“. Er kann benutzt werden wie jeder andere Typ auch z.B. kann man Funktionen als Parameter für eine andere Funktion verwenden oder auch eine Funktion als Rückgabewert definieren. Die Syntax zu einer Funktion ist wie folgt:

```
1 myfunction <- function(arg1, arg2, ... ){
2         statements
3         return(object)
4    }
```

Listing 3: Syntax einer Funktion [1]

Wie man sehen kann, besteht eine Funktion aus einem Namen, einer Parameterliste und einem Codeblock. Um die Parameter zu definieren, wird die Methode **function()** aufgerufen und ihr genau die Parameter übergeben, die die eigene Funktion besitzen soll. Danach wird der Codeblock in geschweiften Klammern definiert. Die Funktion **return()** kann optional verwendet werden, um einen Wert zurückzugeben. Nutzt man **return()** nicht, wird implizit der zuletzt ausgerechnete Wert zurückgegeben.
R bietet weitere Besonderheiten, was Funktionen angeht, einige davon sind, dass Parameter keine vorbestimmten Typen besitzen. Man kann also bei Funktionsaufrufen eine bestimmte

Variable mit verschiedenen Typen belegen. Wenn man allerdings aktuelle Parameter mit Typen übergibt, mit denen die Funktion nicht arbeiten kann, kann es zu Fehlern kommen. Weiterhin müssen Parameter bei einem Funktionsaufruf nicht übergeben werden bspw. Ist der folgende Funktionsaufruf gültig:

```
>myfunction2 <- function(a, b){
    return(1)
 }
>myfunction2()
[0] 1
```

Listing 4: Funktionsaufruf ohne Parameterübergabe

`myfunction2()` hat die beiden Parameter a und b und macht nichts anderes außer den Wert 1 zurückzugeben. Mit der Anweisung `>myfunction2()` wird die Funktion aufgerufen und als Ausgabe erscheint der Wert 1. Es kam also zu keinem Fehler. Wenn aber in der Funktion Operationen mit a oder b durchgeführt werden würden, könnte es zu einem Fehler kommen. Deshalb kann man Werte für die Parameter festlegen für den Fall, dass für einen bestimmten formalen Parameter kein aktueller Parameter übergeben wurde. Solche Standardwerte definiert man folgendermaßen:

```
myfunction3 <- function(arg1, arg2=5, arg3=2 ){.}
```

In diesem Beispiel sieht man, dass die Standardwerte für den Parameter **`arg2`** gleich 5 und für **`arg3`** gleich 2 ist. Wenn also die letzten beiden Parameter nicht übergeben werden, sind sie standardmäßig 5 und 2. In R sind nicht nur alle Parameter optional, sondern sie sind auch benannte Argumente, man kann also direkt beim Funktionsaufruf einen beliebigen Parameter mit einem Wert belegen, indem man ihn benennt wie z.B. so: `>myfunction2(b=10)`.

2.3.1 Beispiele

In diesem Abschnitt sind Beispiele für Funktionen zu sehen, welche mathematische Formeln berechnen.

Beispiel 1

```
fakultaet <- function(n) {
    if( n == 0 )
        1
    else
        n * fakultaet( n - 1 )
  }
```

Listing 5: Fakultätsfunktion

In Listing 5 sieht man eine Funktion, die die Fakultät einer Zahl n berechnet. Aufrufen kann man sie, indem man den Namen der Funktion eingibt und in den Klammern die Parameter übergibt, so wie hier zu sehen ist:

```
> fakultaet(5)
[1] 120
```

Man kann das Ergebnis auch gleich in eine Variable speichern:

```
> y <- fakultaet(5)
```

Beispiel 2.1

```
wahrscheinlichkeit <- function(k,n,p) {
   (fakultaet(n)/(fakultaet(k)*fakultaet(n-k)))*p^(k)*
   (1-p)^(n-k)
 }
```

Listing 6: Wahrscheinlichkeitsfunktion einer Binomialverteilung

Dieses Beispiel zeigt die Implementation einer Wahrscheinlichkeitsfunktion einer Binomialverteilung. Sie rechnet also die Wahrscheinlichkeit dafür aus bei n Versuchen k Erfolge zu erzielen mit einer Wahrscheinlichkeit p pro Erfolg. Sie entspricht also dieser Formel:

$$B(k,n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

In R gibt es bereits viele Funktionen, die mathematische Probleme lösen. Auch für dieses Problem gibt es eine Funktion, die sich **dbinom(k,n,p)** nennt. Sie macht im Wesentlichen das gleiche, wie die Funktion im Listing 6. Der große Unterschied zwischen diesen beiden Funktionen ist, dass **dbinom(k,n,p)** für den Parameter k nicht nur normale Zahlen akzeptiert, sondern auch Listen wie Vektoren oder Matrizen.

```
> dbinom(seq(1:3),13,0.4)
[1] 0.01131927      0.04527707      0.11067729
```

Dieser Aufruf der Funktion **dbinom(k,n,p)** mit einem Vektor mit drei Elementen, gibt also auch einen Vektor mit drei Elementen zurück. Wenn eine selbstgeschriebene Funktion ebenfalls elementenweise Operationen mit Listen durchführen soll, muss durch die Liste iteriert werden. Im Folgenden ist gezeigt, wie die Funktion aus Listing 6 so abgeändert werden kann, dass sie genau wie **dbinom(k,n,p)** für k eine Liste akzeptiert.

Beispiel 2.2

```
wahrscheinlichkeit <- function(k,n,p) {
     x =c()
     for(i in k)
     {
        x =  append(x,(fakultaet(n)/(fakultaet(i)*
             fakultaet(n-i)))*p^(i) * (1-p)^(ni))
     }
     return(x)
}
```

Listing 7: Wahrscheinlichkeitsfunktion einer Binomialverteilung Version 2

In der ersten Zeile des Codeblocks wird ein leerer Vektor erstellt, der am Ende die Ergebnisse beinhalten soll. Dann wird in einer for-each-Schleife jedes Element in dem Vektor k durchlaufen und immer wieder die Rechnung, welche schon in Listing 6 zu sehen war durchgeführt. Das Ergebnis wird mit der Funktion **append()** an den Vektor x angehängt. Am Ende wird der Vektor zurückgegeben.

```
>u = c(1,2,5)
>wahrscheinlichkeit(u,10,0.3)
[1] 0.1210608       0.2334744         0.1029193
```

```
> wahrscheinlichkeit(5,10,0.3)
[1] 0.1029193
```

Listing 8: Aufruf der Wahrscheinlichkeitsfunktion mit einem Vektor und ohne Vektor

Wie man in diesem Funktionsaufruf sieht, werden jetzt auch Vektoren als Parameter für k akzeptiert und auch entsprechend viele Ergebnisse in Form eines Vektors zurückgegeben. Die Funktion lässt sich allerdings immer noch genauso aufrufen wie noch in Listing 6, indem man auch eine Zahl statt eine Liste übergeben kann.

Beispiel 3

```
cumWahrscheinlichkeit <- function(k,n,p){
      a=0
      for (i in 0:k){
          a = a + wahrscheinlichkeit(i,n,p)
      }
      return(a)
}
```

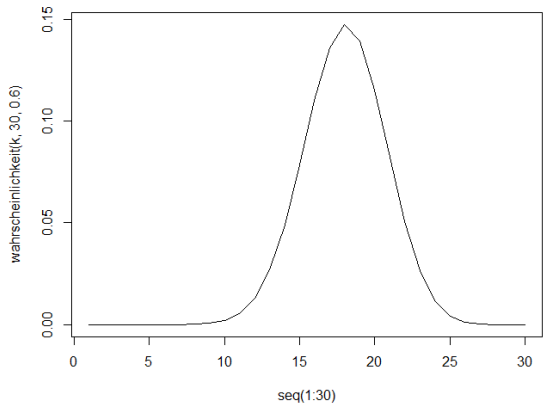
Listing 9: Kumulierte Wahrscheinlichkeitsfunktion einer Binomialverteilung

Diese Funktion entspricht der kumulierten Wahrscheinlichkeitsfunktion einer Binomialverteilung. Es wird in einer normalen for-Schleife von 0 bis k hochgezählt und für jede Zahl die Funktion **`wahrscheinlichkeit()`** aus Beispiel 2.2 aufgerufen. Die Ergebnisse werden addiert und die Summe zurückgegeben. Auch hierfür gibt es eine entsprechende Funktion in R, die sich **`pbinom(k,n,p)`** nennt.

2.4 Plotten von Ergebnissen

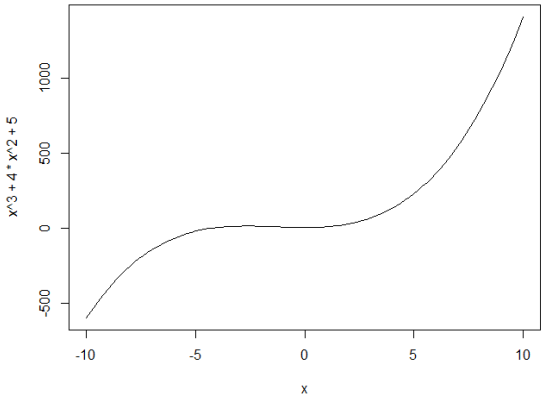
Im Folgenden sind zwei Funktionen gezeigt mit denen man Ergebnisse aus mathematischen Funktionen oder selbstgeschriebenen Funktionen einfach graphisch darstellen kann.

```
>k= seq(1:30)
>plot(k, wahrscheinlichkeit(k,
   ↪ 30, 0.6), type = "l")
```



Listing 10: Plot der Funktion **`wahrscheinlichkeit()`** mit k als Parameter

```
>curve(x^3+4*x^2+5, from=-10, to
   ↪ =10)
```



Listing 11: Plot des Polynoms $x^3 + 4x^2 + 5$ mit **`curve()`** im Intervall [-10,10]

Im Beispiel 1 sieht man den Aufruf der Funktion **`plot()`**, die als ersten Parameter einen Vektor mit den Zahlen 1 bis 30 bekommt. Dieser Vektor stellt die x-Achse dar. Der zweite

Parameter besteht aus den Ergebnissen der Funktion **`wahrscheinlichkeit()`** mit dem Vektor k als Parameter, damit am Ende ein Vektor mit 30 Ergebnisse zurückgegeben wird, die die Funktion **`plot()`** darstellen kann. Der dritte Parameter gibt die Art des Graphen an. In diesem Fall ist der Parameter type="l" , welcher einen normalen Graph angibt. Die zweite Möglichkeit, Ergebnisse einer Funktion darzustellen, ist die Funktion **`curve()`**. Dies sieht man in Listing 11 an der Funktion $x^3 + 4x^2 + 5$. Man übergibt ihr eine mathematische Funktion und ein Intervall. Der Vorteil gegenüber **`plot()`** ist, dass man keine Funktion schreiben muss, um eine mathematische Funktion grafisch darzustellen. Andererseits kann **`curve()`** nur einfache mathematische Ausdrücke und Funktionen, welche als Parameter nur die Variable x haben, darstellen.

3 Formeln

Dieses Kapitel geht um den Typ „formula“ und um die Funktionalitäten dieses Typs. Objekte dieses Typs nennt man Formeln, welche nicht zu verwechseln sind mit mathematischen Formeln. Diese Formeln bestehen hauptsächlich aus vektoriellen Variablen, die in Abhängigkeit voneinander stehen. Man benutzt diese Formeln für unterschiedliche Zwecke wie z.B. um daraus statistische Modelle zu erstellen oder um sich die Abhängigkeiten der enthaltenen Variablen graphisch darstellen zu lassen.

3.1 Zweck von Formeln

Formeln in R werden verwendet um statistische Modelle zu erstellen, Plots zu erstellen oder um sie in eigenen Funktionen für eigene Zwecke zu verwenden.
Um aus Formeln statistische Modelle zu erstellen, gibt es die Funktionen **`lm()`** (für lineare Modelle), **`glm()`** (für generalisierte lineare Modelle), **`lme()`** (für gemischte Modelle) und einige weitere.
Um Grafiken aus Daten mithilfe von Formeln zu erstellen gibt es z.B. die Funktionen **`plot()`** (für einfache Plots), **`boxplot()`** (für Boxplots), Funktionen aus dem Package lattice (für komplexere Grafiken), Funktionen aus dem Package ggplot (für komplexere Grafiken) und einige mehr.
Um Formeln für eigene Zwecke zu verwenden kann man die Daten aus einer Formel extrahieren, indem man sich die Variablen der Formel als Dataframe mit der Funktion **`model.frame(`*`Formel`*`)`** zurückgeben lässt oder sich die Designmatrix, also nur die unabhängigen Variablen mit **`model.matrix(`*`Formel`*`)`** zurückgeben lässt. Mit diesen Daten kann man weitere Operationen anstellen.

3.2 Aufbau einer Formel

Die Operatoren die man in Formeln benutzen darf, sind hier vorgestellt.

Operatoren	**Beispiel**	**Beschreibung**
+	y ~ x + z	Einfluss einer Variable
-	y ~ . -x, data=d	Variable aus der Formel löschen
:	y ~ x:z	Interaktion zweier Variablen hinzufügen
*	y ~ x*z	Direkter Einfluss und Interaktion zweier Variablen
^	y ~ (x+z+w) ^ 3	Direkter Einfluss und alle möglichen Interaktionen
-1 oder +0	y ~ x + z - 1	Kein Interzept

Eine Formel hat immer die Form: y ~ x $\circ_1$ z $\circ_2$ ..., wobei y die abhängige Variable, $\circ_i$ ein Operator ist und x, z... die unabhängigen Variablen sind. Ein Formel-Objekt erzeugt

man wie folgt `f <- formula(y ~ x` $\circ_1$ `z` $\circ_2$ `...)`.
Der ‚`+`'-Operator, gibt hierbei keine Addition an, sondern die Hinzunahme einer Variable. In dem Beispiel y ~ x + z wird also y in Abhängigkeit von x und von z gesetzt. Der ‚`-`'-Operator löscht eine Variable aus der Formel. Ihn benutzt man dann, wenn man viele Variablen in einem Datensatz hat und alle bis auf einige wenige benutzen will. Dazu kann man nach der Tilde einen Punkt setzen und den Datensatz am Ende der Formel nach einem Komma der Variable „data“ zuweisen. Dann gibt man die zu löschenden Variablen mit dem ‚`-`'-Operator an. Damit wird jede Variable aus dem gewählten Datensatz in die Formel hinzugefügt und die ausgewählten Variablen mit dem ‚`-`'-Operator wieder gelöscht.
Der ‚`:`'-Operator gibt eine Interaktion zwischen zwei Variablen an. Ihn benutzt man immer dann, wenn der Wert der einen Variable den Wert der anderen Variable beeinflussen kann.
Der ‚`*`'-Operator fügt die Variablen hinzu und die Interaktion zwischen diesen beiden Variablen. Der ‚`^`'-Operator fügt alle Variablen in den Klammern hinzu und alle möglichen Interaktion zwischen ihnen bis maximal n Variablen. Im Beispiel werden alle möglichen Interaktionen zwischen den Variablen x, z und w hinzugefügt. Die -1 oder 0 gibt an, dass die Formel keinen Interzept, also den y-Achsenabschnitt 0 haben soll.
Desweiteren ist es erlaubt in Formeln mathematische Ausdrücke und Operationen zu verwenden wie `y ~ log(x) + I(z+5)`. Hierbei werden alle Werte in der Variable x logarithmiert und jeder Wert in z wird mit Fünf addiert. Dabei wird eine Syntax verwendet, die dafür sorgt, dass arithmetische Operatoren und Funktionen nicht mit Operatoren einer Formel verwechselt werden. Um also mathematische Operatoren von den Operatoren einer Formel zu trennen, müssen die mathematischen Operatoren und Operanden in Klammern gesetzt werden und ein „I“ vorangestellt werden.
In einer Formel muss unter Umständen der Datensatz, aus dem die Variablen der Formel stammen, angegeben werden. Das ist nötig, wenn die Variablen der Formel nicht im workspace gespeichert sind oder wenn der ‚`-`'-Operator benutzt wird. Dazu schreibt man die Quelle der Variablen mit einem Komma separiert in die Formel, so wie in der zweiten Zeile der Tabelle zu sehen ist. Die Quelle muss zudem vom Typ Dataframe sein, außerdem müssen die Spalten im Dataframe, also die Einträge in der ersten Zeile, so heißen wie die Variablen, die in der Formel benutzt werden.

Im Folgenden werden die Funktionen einer Formel am Beispiel der linearen Modelle vorgestellt.

3.3 Lineare Modelle

Lineare Modelle sind Modelle, die eine reale Beobachtung beschreiben und die man mittels mathematischer Notation ausdrücken kann. Dabei wird davon ausgegangen, dass die unabhängige Variable und die abhängigen Variablen einen linearen Zusammenhang haben. Das Ziel von linearen Modellen ist es Regressionsverfahren durchzuführen, um damit Prognosen für die abhängige Variable zu erstellen oder die Stärke des Zusammenhangs von abhängigen und unabhängigen Variablen herauszufinden.

Beispiel 1
Hier sieht man ein Beispiel dafür wie ein lineares Modell in R erstellt werden kann.

```
1 >x <- rnorm(100, sd = 20)
2 >y <- x + rnorm(100, sd = 9)
3 >plot(y ~ x)
4 >model <- lm(y ~ x)
5 >abline(model)
```

Listing 12: Erstellung eines Datensatzes mit Zufallswerten, plotten der Daten und zeichnen der Regressionsgeraden

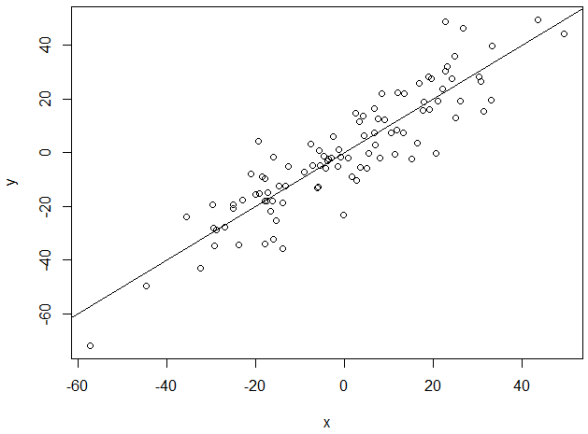


Abbildung 1: Punktewolke und Trendlinie

In diesem Beispiel werden zwei Variablen x und y definiert, wobei x aus 100 zufälligen Werten besteht und y alle Werte aus x mit 100 weiteren zufälligen Werten addiert, enthält.
Dabei soll y die abhängige Variable und x die Einflussvariable, also die unabhängige Variable darstellen. Die Formel y ~ x wird dann mit der Funktion **plot** grafisch dargestellt, wie man in Abbildung 1 sehen kann. Dabei wird jedem x-Wert ein y-Wert zugewiesen und die Punkte werden an den entsprechenden Stellen angezeigt. Mit der Funktion **abline()** zeichnet man die Trendlinien oder auch Regressionsgeraden ein, wie man ebenfalls in Abbildung 1 sehen kann. Diese Regressionsgerade ist eine Gerade, welche so angelegt ist, dass sie eine möglichst gute Approximation für die Datenpunkte liefert. Die Summe der Residuen (Abweichung der Punkte von der Regressionsgerade) ist also mit dieser Geraden am geringsten.
Mit der Funktion **lm()** wird die Formel in ein lineares Modell umgewandelt und in die Variable **model** gespeichert. In mathematischer Notation entspricht dieses lineare Modell dem Regressionsmodell $Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$, wobei Y_i der y-Wert an der Stelle X_i ist, β_0 ist der y-Achsenabschnitt, β_1 ist der Koeffizient, also die Stärke des Einflusses von X und ε_i ist das Residuum für den Punkt an der Stelle X_i. Eine weitere Vorraussetzung für ein lineares Modell ist, dass der Fehlerterm ε als vernachlässigbare Zufallsgröße angenommen wird, er also durch zufällige Störeinflüsse oder andere Ursachen herrührt und man annimt, dass sein Erwartungswert Null ist. Durch die Hinzunahme weiterer Variablen, wird die mathematische Formel entsprechend mit der Variable und dem Koeffizienten ergänzt, sodass z.B eine Interaktion wie y ~ x:z der mathematischen Notation $Y_i = \beta_0 + \beta_1 X_i Z_i + \varepsilon_i$ entspricht. Die Funktion **lm()** rechnet bei der Erstellung des linearen Modells zusätzlich den Interzept β_0, die Residuen ε_i und die Koeffizienten $\beta_1, ..., \beta_n$ der Variablen für die Regressionsgleichung aus.

Nachdem man ein lineares Modell erstellt hat, kann man dieses Modell für weitere Funktion in R benutzen, z.B. für die Funktion summary.

```
1 >summary(lm(y ~ x))
2
3 Residuals:
4      Min       1Q   Median       3Q      Max
5 -24.3014  -6.5649   0.8644   6.0194  27.8797
6
7 Coefficients:
8                    Estimate Std. Error t value Pr(>|t|)
9 (Intercept)         0.05374    0.97350   0.055    0.956
```

```
x                   0.96420    0.05117  18.843   <2e-16 ***

Residual standard error: 9.718 on 98 degrees of freedom
Multiple R-squared:  0.7837,    Adjusted R-squared:  0.7815
F-statistic: 355.1 on 1 and 98 DF,  p-value: < 2.2e-16
}
```

Listing 13: summary() für das lineare Modell **lm(y ~ x)**

Die Funktion **summary()** erstellt Zusammenfassungen von linearen Modellen, indem alle wichtigen Werte ausgerechnet und ausgegeben werden. Dazu gibt es z.B. den Block „Residuals", welcher Informationen zu den Residuen anzeigt. Weiterhin gibt es den Block „Coefficients", welcher Informationen über die Koeffizienten der Variablen gibt. Aus diesem liest man ab, dass x den Koeffizienten 0.9642 hat und dass der y-Achsenabschnitt der Regressionsgeraden 0.05374 ist. Aus diesen beiden Werten kann man die Funktion der Regressionsgerade bestimmen: Sie lautet $y = 0.9642x + 0.05374$. Dann gibt es noch die Standardabweichung, die man unter dem Namen „Residual standard error" findet. Hier liegt sie bei 9.718. Ein weiterer wichtiger Wert ist Multiple R-squared, auf Deutsch Bestimmtheitsmaß. Er gibt den Anteil der von den unabhängigen Variablen erklärten Abweichungen der Punkte von der Regressionsgerade an. Im Beispiel 1 ist das Bestimmtheitsmaß 78,37 %, also erklärt die Variable x 78,37% der Residuen.

Das folgende Listing zeigt, wie man einen Wert für einen Datensatz vorhersagt.

```
>wert = data.frame(x=120)
>predict(model, wert)
       1
115.7577
}
```

Listing 14: Vorhersage eines Wertes

Man sieht hier die Vorhersage für den Wert $x = 120$ für den Datensatz der in Listing 12 erstellt wurde. Es wird zuerst ein Dataframe erstellt, der mit einem Wert für die Variable x belegt wird, für den die Prognose durchgeführt werden soll. Danach wird die Funktion **predict()** aufgerufen und ihr das lineare Modell und das Dataframe übergeben. In diesem Fall wurde für den Wert $x = 120$ der Wert 115.7577 vorhergesagt. Eine äquivalente Methode, um eine Vorhersage zu machen, ist den Wert in die Gleichung der Regressionsgerade einzusetzen, wie hier zu sehen ist: $y = 0.9642 \cdot 120 + 0.05374 = 115.7577$. Die Vorhersage für eine Belegung von Variablen in einem Datensatz ist also lediglich der Funktionswert der Regressionsgleichung mit der Belegung der Variablen.

Beispiel 2
Im Folgenden sieht man ein weiteres Beispiel für ein lineares Modell, indem ein Datensatz benutzt wird in dem es um den Konsum von Eiscreme geht. Im Datensatz sind die Variablen **cons** (die Menge des Konsums), **temp** (die Außentemperatur), **price** (der Preis der Eiscreme) und **income** (das Einkommen der Konsumenten) enthalten. Hier wurde die Variable **cons** in Abhängigkeit der Variable **temp** gestellt.

```
>a = read.csv(file.choose(), header=TRUE, sep=",")
>model = lm(a$cons ~ a$temp)
>plot(a$cons ~ a$temp)
>abline(model)
>summary(model)
...
```

```
Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.2068621  0.0247002   8.375 4.13e-09 ***
a$temp      0.0031074  0.0004779   6.502 4.79e-07 ***
...
Residual standard error: 0.04226 on 28 degrees of freedom
Multiple R-squared:  0.6016,    Adjusted R-squared:  0.5874
F-statistic: 42.28 on 1 and 28 DF,  p-value: 4.789e-07
```

Listing 15: summary() für das lineare Modell **lm(a$cons ~ a$temp)**

In der Zusammenfassung des linearen Modells sieht man, dass der Koeffizient von **temp** 0.0031074 ist. Das heißt, dass nach der Regressionsgleichung der Konsum um 0.0031074 Konsumeinheiten steigt, wenn die Temperatur um eine Temperatureinheit steigt. Der dazugehörige Plot und die Trendlinie sehen so aus:

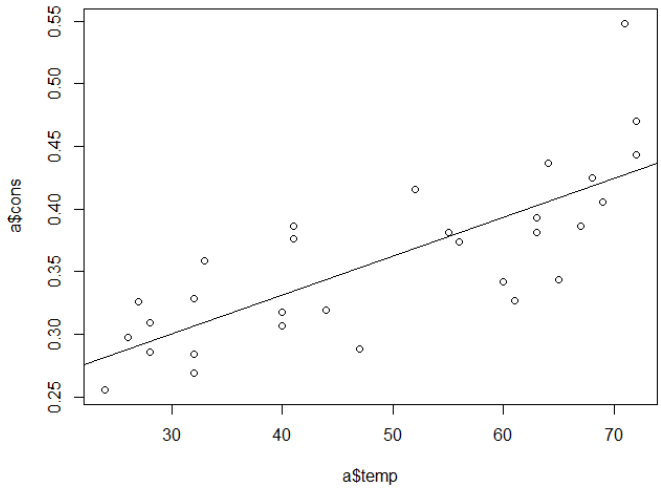


Abbildung 2: Plot und Trendlinie

Das Bestimmtheitsmaß dieses Modells ist 0.6016. Indem man weitere unabhängige Variablen in die Formel hinzufügt, kann man das Bestimmtheitsmaß erhöhen, denn mehr Variablen können mehr Variabilität erklären und damit die Güte des linearen Modells erhöhen, so wie im nächsten Beispiel zu sehen ist.

Beispiel 3

In diesem Beispiel wird ein lineares Modell erstellt, welches die Variable **cons** in Abhängigkeit von **temp**, **price** und **income** stellt.

```
>a = read.csv(file.choose(), header=TRUE, sep=",")
>model = lm(a$cons ~ a$temp + a$price + a$income)
>#plot(a$cons~a$temp+a$price+a$income)
>summary(model)
...
 Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
 (Intercept)  0.1973151  0.2702162   0.730  0.47179
 a$temp       0.0034584  0.0004455   7.762  3.1e-08 ***
 a$price     -1.0444140  0.8343573  -1.252  0.22180
 a$income     0.0033078  0.0011714   2.824  0.00899 **
 ...
 Residual standard error: 0.03683 on 26 degrees of freedom
```

```
14 Multiple R-squared:  0.719,    Adjusted R-squared:  0.6866
15 F-statistic: 22.17 on 3 and 26 DF,  p-value: 2.451e-07
```

Listing 16: Beispiel 1: summary()

Ein Plot einer Formel mit mehr als zwei Variablen kann man in R nicht optimal darstellen, daher ist die Zeile 3 auskommentiert. Denn die Funktion Plot würde mehrere Plots anzeigen, in diesem Fall würde sie drei Plots erstellen, in denen die abhängige Variable **cons** ist und die einzige unabhängige Variable jeweils eine von **temp**, **price** und **income** ist. Einen einzigen Plot kann man in diesem Fall mit **plot()** also nicht erstellen. Außerdem bräuchte man für jede Variable eine neue Dimension in dem Koordinatensystem und mit vier Variablen, wie in Listing 16 gezeigt, könnte man sich den Plot nicht sinnvoll anzeigen lassen.
Im Vergleich mit der Formel, die für das lineare Modell in Listing 15 verwendet wurde, hat dieses Lineare Modell einen etwa 11% höheren Bestimmtheitsmaß, der durch die Hinzufügung der Variablen **price** und **income** zustande gekommen ist. Durch die Hinzunahme dieser beiden Variablen wurde die Variable **cons**, also besser erklärt.

3.4 Vergleich von linearen Modellen

Das Ziel von Vergleichen ist es herauszufinden, welches lineare Modell eine bestimmte abhängige Variable am besten beschreibt und ob eine unabhängige Variable überhaupt einen Einfluss auf die abhängige Variable hat, ob diese Variable also nötig ist. Hierbei wird verglichen, ob der Unterschied des Bestimmtheitsmaßes zweier linearer Modelle signifikant ist oder nur durch Zufall bedingt ist. Im Folgenden wird gezeigt, wie man diesen Vergleich anstellen kann.

```
> model1<-lm(a$cons~a$temp+a$price)
> model2<-lm(a$cons~a$temp)
> anova(model1, model2)

Analysis of Variance Table

Model 1: a$cons~a$temp+a$price
Model 2: a$cons~a$temp
Res.Df   RSS Df Sum of Sq   F Pr(>F)
1   27 0.046
2   28 0.050 -1 -0.0039 2.296 0.1413
```

Listing 17: Beispiel 1: anova mit P-Wert>0.05

```
> model3<-lm(a$cons~a$temp)
> model4<-lm(a$cons~a$temp+a$income)
> anova(model4, model3)

Analysis of Variance Table

Model 1: a$cons~a$temp+a$income
Model 2: a$cons~a$temp
Res.Df   RSS Df Sum of Sq  F  Pr(>F)
1   27 0.037
2   28 0.050 -1 -0.01261 9.10 0.0055
```

Listing 18: Beispiel 1: anova mit P-Wert<0.05

In dem Listing 17 sieht man zwei lineare Modelle, die als Parameter für die Funktion **anova()** genutzt werden. Diese Funktion vergleicht zwei lineare Modelle miteinander und gibt unter anderem den P-Wert aus. An diesem kann man erkennen wie hoch die Wahrscheinlichkeit ist, dass der Unterschied der Bestimmtheitsmaße der beiden Modelle nur durch Zufall bedingt ist. In Listing 17 sieht man den P-Wert 0.1413, welcher größer ist als der Standardwert 0.05. Dieser Standardwert bedeutet, dass ein P-Wert unter 0.05 einen signifikanten Unterschied zwischen zwei Modellen angibt. Damit ist der Unterschied zwischen den beiden Modellen nicht signifikant und daher kann man annehmen, dass der Unterschied nur durch Zufall bedingt ist. Aus diesem Grund ist es egal welches dieser linearen Modelle man weiterhin verwendet, wenn es darum geht dasjenige lineare Modell zu finden, welches die abhängige Variable am besten erklärt. Da in der Formel von **model1** die Formel von **model2** enthalten ist und der Unterschied nicht signifikant ist, weiß man zudem auch, dass die Variable **price** keinen oder nur einen geringen Beitrag zur Erklärung

der Variable `cons` leistet.
Im Listing 18 sieht man wieder zwei unterschiedliche Modelle und das Ergebnis von `anova()`. Hier ist der P-Wert 0.0055 und damit kleiner als 0.05, was bedeutet, dass der Unterschied signifikant ist und daher kann man annehmen, dass der Unterschied nicht zufallsbedingt ist und damit `cons` durch die Variable `income` signifkant erklärt wird. Wenn es also darum geht ein Modell zu finden, welches die abhängige Variable am besten beschreibt, sollte man sich in diesem Beispiel für `model4` entscheiden.

4 Zusammenfassung

Im ersten Abschnitt wurde gezeigt, wie man mathematische Formeln in R mithilfe von Funktionen erstellt. Diese sind dank ihrer einfachen Syntax und den flexiblen Datentypen einfach zu erstellen, wodurch sich Formeln leicht als Quelltext implementieren lassen. Zudem enthält R viele mathematische und statistische Funktionen und auch komplexe Datentypen mit denen sich rechnen lassen und damit stellt R eine für Rechnungen optimierte Umgebung dar.
Im zweiten Kapitel wurde gezeigt, wie Formeln mit dem Datentyp „formula“ erstellt werden und welchen Nutzen sie haben. Sie werden häufig benutzt, um statistische Modelle zu erstellen und damit Analysen durchzuführen. Die Anwendung von Formeln wurde an dem Beispiel der linearen Modelle gezeigt, indem die linearen Modelle mithilfe der Formeln erstellt wurden und dann Funktionen an diesen Modellen angewandt wurden. Die Syntax einer Formel ist dabei angelehnt an die mathematische Notation für die Beschreibung eines statistischen Modells. Am Ende wurde gezeigt, wie man zwei lineare Modelle anhand des P-Wertes vergleicht.
Durch den großen Funktionsumfang der Formeln und statistischer Modelle fällt auf, dass R im Vergleich zu anderen Programmiersprachen eine besondere Unterstützung für beobachtete Daten und auch für statistische Modelle hat. Daher kann man mit R, ohne viel Aufwand, statistische Modelle erzeugen und an ihnen Operationen durchführen, welche die Werkzeuge in R bereitstellen.

5 Literaturverzeichnis

Literatur

[1] - Robert I. Kabacoff, *User-written Functions* ,
http://www.statmethods.net/management/userfunctions.html, 23.05.2016.

[2] - Richard Hahn, *Statistical Formula Notation in R*,
http://faculty.chicagobooth.edu/richard.hahn/teaching/formulanotation.pdf, 18.05.2016

[3] - *Using R for Linear Regression*,
http://www.montefiore.ulg.ac.be/~kvansteen/GBIO0009-1/ac20092010/Class8/Using%20R%20for%20linear%20regression.pdf, 18.05.2016

[4] - Jim Frost, *Regression Analysis*,
http://blog.minitab.com/blog/adventures-in-statistics/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit, 18.05.2016

[5] - Vincent Arel-Bundock, *Datasets*,
https://vincentarelbundock.github.io/Rdatasets/datasets.html, 18.05.2016

[6] -*Formula*,
https://stat.ethz.ch/R-manual/R-devel/library/stats/html/formula.html, 18.05.2016

[7] -*Modeling in R*,
http://www.springer.com/cda/content/document/cda_downloaddocument/9781461444749-c1.pdf?SGWID=0-0-45-1344211-p174513603, 18.05.2016

[8] - Steven Buechler, *Statistical Models in R*,
http://www3.nd.edu/~steve/Rcourse/Lecture8v1.pdf, 18.05.2016

[9] -*Lineares Modell*,
https://de.wikipedia.org/wiki/Lineares_Modell, 18.05.2016

[10] - Vincent Zoonekynd, *Programming in R: Miscellaneous*,
http://zoonek2.free.fr/UNIX/48_R/02.html, 03.07.2016

[11] - Dr Mike Allerhand, *Linear models*,
http://forums.psy.ed.ac.uk/R/P01582/essential-18/, 03.07.2016

[12] -*Regressionsanalyse*,
https://de.wikipedia.org/wiki/Regressionsanalyse, 13.07.2016

[13] -*Lineare Regression*,
https://de.wikipedia.org/wiki/Lineare_Regression, 13.07.2016

[14] - Günter Roolfs -*Regressionsgerade*,
http://nibis.ni.schule.de/~lbs-gym/Verschiedenespdf/Regressionsgerade.pdf, 13.07.2016