# Systemrprüfung mit Hilfe der Statistik Software R

$$y \quad = \quad a \cdot x^3 + b \cdot x^2 + c$$

$s_{(t)} \bullet\!\!-\!\!\circ\; S_{(f)}$

$S_{(f)} \circ\!\!-\!\!\bullet\; s_{(t)}$

Urs von Ballmoos

11. Dezember 2020

Geschrieben mit dem
Satzsystem von LaTeX

PDF Erstellung: 14. Dezember 2020

# Inhaltsverzeichnis

# Vorwort

Für die Erhaltung meines CISA Zertifikates werden jedes Jahr eine gewisse Anzahl an Weiterbildungsstunden verlangt. Normalerweise nutze ich dafür neben Online-Veranstaltungen vorwiegend Präszenz-Veranstaltungen, die kosten zwar meist etwas mehr, dafür sind sie interaktiver und ich kann mehr davon profitieren. Das Jahr 2020 ist von Covid-19 und somit auch von Absagen diverser Weiterbildungsveranstaltungen geprägt. Da reines Selbststudium nicht als Weiterbildung anerkannt wird, mache ich aus der Not eine Tugend und veröffentliche meine Erkenntnisse aus Theorie und Praxis. Damit können alle gewinnen: Ich lerne etwas und Sie liebe Leserschaft können hoffentlich von meinen Erfahrungen profitieren.

In meiner Tätigkeit als CISA bin ich immer wieder mit Systemen konfrontiert, die mit erträglichem Aufwand nicht mit den Testmöglichkeiten aus meiner CISA Ausbildung geprüft werden können. Dazu später mehr in einem eigenen Kapitel.

Zu meiner Person. Ich bin im Jahre 1969 in Zürich geborgen, habe ab Mitte der 80er Jahre eine Lehre als El. Mech. gemacht und anschliessend in Rapperswil Elektrotechnik studiert. Im Laufe der Zeit sind noch Nachdiplomstudien in BWL, Information Management und Business Engineering dazu kommen. Nachdem Studium war ich während 12 Jahren in der Industrie in verschiedenen Positionen im der Schweiz und Singapur tätig, seit rund 8 Jahren bei einem grossen Telco Provider in der Schweiz und in Folge von Reorganisationen in verschiedenen Funktionen angestellt, aktuell als Business Analyst tätig.

# Einleitung und Voraussetzungen

## 1. Einleitung

Wie bereits im Vorwort erwähnt kommt der Autor in seiner Tätigkeit in der Prüfung von Systemen immer wieder in Situationen, wo das klassische Testing mit *substantive Testing* und *Blackbox Testing* nicht oder nur unzureichend funktioniert. Typische Fälle sind Situationen, wo sich die Systeme nicht linear verhalten, die Algorithmen unbekannt sind oder mehrere Inputs zu einem Output führen. Neben Maschien kommen noch Menschen hinzu, z.B. Produktmanager die aufgrund von Gefühlen und Marktbeobachtungen einen wesentlichen Teil der Capex beeinflussen können.

Bereits an dieser Stelle soll darauf aufmerksam gemacht werden, dass daraus nicht der Umkehrschluss möglich ist, dass mit Anwendung diese Systeme einfach austauschbar wären. Es geht um die Überprüfung von Korrelationen und Plausibilitäten und nicht um das Abfangen von Einzelfällen.

## 2. Voraussetzungen

Um von diesem Manuskript optimal profitiert zu können, sind viele einfache Beispiele mit allen Daten vorhanden. Damit alle Beispiele selber nachgerechnet werden können, sind diese meist als komplettes Beispiel in diesem Manuskript integriert. Zudem findet sich im Anhang alle R-Beispiele zusammengefasst.

## 3. Literatur

Da die meisten Beispiele und Erklärungen aus der täglichen Arbeit des Authors stammen, gibt es nur wenig referenzierte Literatur. Selbstverständlich findet sich aber sehr viel Literatur zu R und Statistik um Lösungen zu individuellen Problemen zu finden. Besonders zu empfehlen ist selbstredend die Helpfunktion von R.

# Teil I.

# Einführung und Theorie

# 1. R Software

Viele Beispiele in diesem Manuskript könnten auch mit Hilfe von klassischen Tabellenkalkulationsprogrammen aus Office Paketen durchgeführt werden. Dabei ist aber zu beachten, dass die Beispiele möglichst einfach gehalten sind und sich die Office Programme nur sehr beschränkt für grosse Datenmengen und unterschiedliche Datenquellen eignen. Das bedeutet nicht, dass in Einzelfällen der Einsatz von Tabellenkalkulationsprogrammen auch einen geeigneten Weg darstellen.

Ein weiterer Vorteil von R gegenüber den Office Paketen besteht in der Möglichkeit R auf einem Server laufen zu lassen oder aus der Cloud zu beziehen und damit vieles zu automatisieren. Weitere Details dazu im Kapitel R in der Cloud.

## 1.1. Grundpaket R

R ist Opensource und kann somit uneingeschränkt im privaten als auch geschäftlichen Umfeld genutzt werden. Die Hauptseite von R findet sich hier: `https://www.r-project.org/`

Unter `https://cran.r-project.org/` können für die gängigen Betriebssysteme wie Max OS X, Windows oder Linux bereits kompilierte und installierbare R Pakete runtergeladen werden. Bei Linux besteht häufig die Möglichkeit dies direkt über die Paketverwaltung durchzuführen. Details dazu finden sich bei den verschiedenen Distributionen. Zum Zeitpunkt bei der Erstellung dieses Manuskriptes ist die Version 4.03 aktuell.

R besteht aus einem Grundpaket und je nach Bedarf können weitere Pakete geladen werden. Siehe Kapitel Grundstruktur der Befehle in R.

## 1.2. R Studio

R selber läuft in einer Standard Kommandozeilenumgebung. Deshalb ist es hilfreich diese Umgebung für besseren Arbeitskomfort um eine Integrated Development Environment (IDE) zu ergänzen. Dazu bietet sich R Studio `https://rstudio.com/products/rstudio/` an, das sowohl in einer Opensource als auch einer kommerziellen Version zur Verfügung steht. Für unsere Bedürfnisse reicht die Opensource Lizenzierung vollauf.

### 1.2.1. R Studio Server

Zusätzlich zur normalen IDE gibt es noch R Studio Server `https://rstudio.com/products/rstudio/#rstudio-server`. Diese Version steht unter einer Opensource Lizenzierung zur Verfügung und kann hilfreich sein, wenn aufgrund von Einschränkungen auf dem Arbeitsplatzcomputer nichts installiert werden darf. R Studio Server läuft als Webapplikation. Auch hier weitere Informationen im Kapitel R in der Cloud.

## 1.3. Objektorientiert

R ist durchgängig objektorientiert. Das bedeutet, dass jede Variable, jede Formel etc. ein Objekt ist, das aus einer Klasse heraus entstanden ist. In unserer Umgangssprache entspricht eine Klasse einer Definition und das Objekt einer Instanz.

Bsp. Velokatalog: Im Katalog sind die verschiedenen Velos mit ihren Eigenschaften definiert. Wenn wir ein Velo kaufen, dann wurde eine Instanz gebildet die alle, oder auch nur einen Teil der Eigenschaften hat, die in der Klasse (Katalog) beschrieben sind, abhängig von den durch uns gewählten Optionen. Dabei gibt es auch Eigenschaften, die sich gegenseitig ausschliessen. So kann nicht gleichzeitig einen Mountainbike-Lenker und einen Rennvelo-Lenker haben montiert sein.

# 2. Grundstruktur der Befehle in R

## 2.1. R starten

Beim Start von R (nicht R Studio), bekommt man eine einfache Konsole mit wenigen Menupunkten. In der Abbildung 2.1 ist ein solches Beispiel, das beim Start zusätzlich noch ein paar Systeminformationen preisgibt.

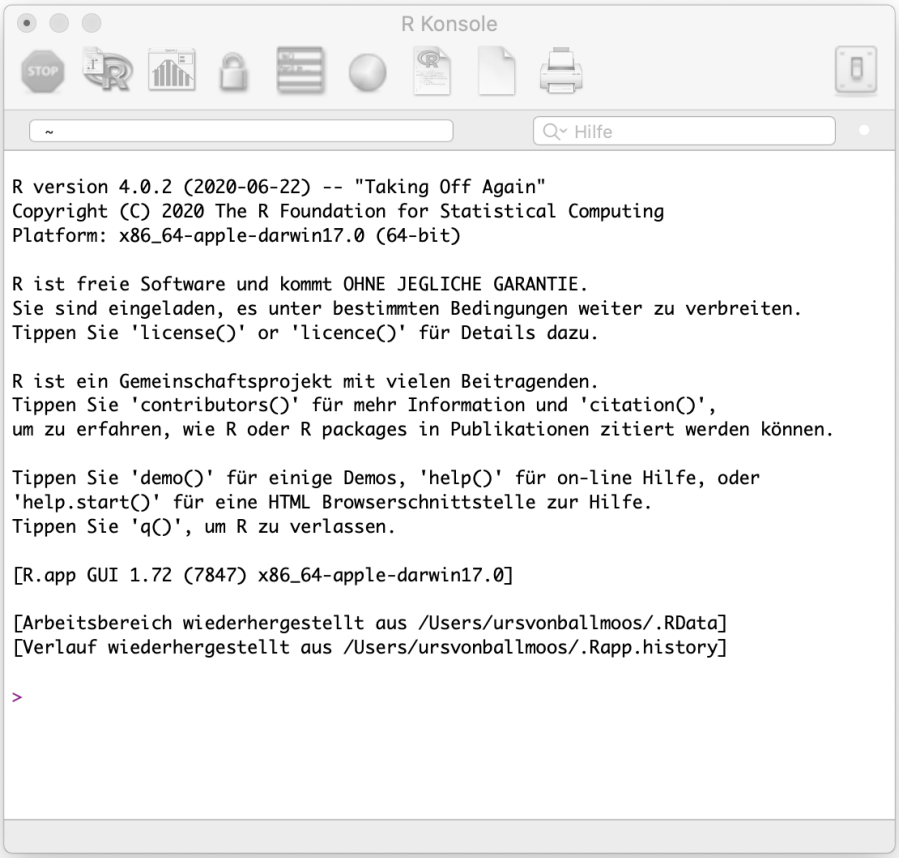


Abbildung 2.1.: R Konsole

### 2.1.1. Erste Berechnung in R

Auch wenn hier im Gegensatz zum unten beschriebenen R Studio weniger Unterstützung vorhanden ist, R kann hier bereits als einfacher Taschenrechner herhalten. Wenn z.B. die Summe von $3+8$ gewünscht ist, dann wird einfach `3+8` eingetippt und im Anschluss mit der Eingabetaste abgeschlossen und 11 angezeigt.

**Einfache Summe in R Command Line Interface (CLI)**

```
> 3+8
[1] 11
>
```

Das Zeichen '>' weist auf die Eingabe hin und die Zahl in eckigen Klammern `[1]` bedeutet, dass es sich um das erste Resultat handelt.

## 2.2. Syntax

Die folgenden Beschreibungen bleiben an der Oberfläche und sollen nur die Grundzüge vermitteln, damit die Syntax der Beispiele nachvollziehbar wird.

### 2.2.1. Variablen

Da es unpraktisch wäre grosse Datenmengen in Formeln darzustellen, können alle Objekte einer Variablen zugewiesen werden. Dies geschieht mit dem Operator $VarName <- Variable$. Im folgenden Beispiel wird $a$ der Wert 5 und $b$ den Wert 8 zugewiesen. Inzwischen kann zwar R auch mit dem $=$ Zeichen für Zuweisungen umgehen, aber aus Kompatibilitätsgründen zu älteren Versionen sollten die Benutzer bei $<-$ verbleiben. Es könnte schliesslich auch in einer aktuellen R Installation ein älteres Paket aufgerufen werden, das mit der neuen Zuweisung nicht zurecht kommt.

**Zuweisung von Variablen in R**

```
> a <- 5
> b <- 8
```

Und selbstverständlich lässt sich auch damit rechnen:

**Rechnen Variablen in R**

```
> a*b
[1] 40
```

### 2.2.2. Listen, Matrizen und Data Frames

In der Statistik warten normalerweise die Werte verschiedener Beobachtungen auf. Dies können am einfachsten in einer Liste (auch Vektoren genannt) zusammengefasst werden. Dazu kommt die Funktion `c`, die für *Combine* steht zum Einsatz.

**Listen oder Vektoren**

```
> myList <- c(1, 3, 7.7, 32)
>
> myList
[1]  1.0  3.0  7.7 32.0
```

Natürlich können auch mehrere Listen ineinander verschachtelt werden.

**Listen kombinieren**

```
> myList <- c(1, 3, 7.7, 32)
> myList
[1]  1.0  3.0  7.7 32.0
> myNewList <- c(myList, 345)
> myNewList
[1]   1.0   3.0   7.7  32.0 345.0
```

Und dann gibt es noch den Fall, dass pro Beobachtung mehrere Werte bestehen, also mehrere Werte auf einer Zeile in verschiedenen Spalten. Mit "cbind"werden verschiedene Listen in Spalten zusammengeführt.

**cbind**

```
> farbenListe <- c("blau", "grün", "gelb", "rot")
> modellListe <- c("ModelA", "ModelB", "ModelC", "ModelD")
> alterListe <- c (12, 33, 27, 12)
>
> matrize <- cbind(farbenListe, modellListe, alterListe)
> matrize
farbenListe modellListe alterListe
[1,] "blau"       "ModelA"    "12"
[2,] "grün"       "ModelB"    "33"
[3,] "gelb"       "ModelC"    "27"
[4,] "rot"        "ModelD"    "12"
>
```

oder Zuweisung in Zeilen

**rbind**

```
> matrize <- rbind(farbenListe, modellListe, alterListe)
> matrize
[,1]     [,2]     [,3]     [,4]
farbenListe "blau"   "grün"   "gelb"   "rot"
modellListe "ModelA" "ModelB" "ModelC" "ModelD"
alterListe  "12"     "33"     "27"     "12"
>
```

Aufmerksame Lesende aus der Programmierwelt erkennen diese Matrizen ähnlich von Arrays. Der wesentlichste Unterschied liegt allerdings darin, dass der Ursprung nicht `[0/0]` sondern `[1/1]` ist. Dafür fühlen sich Leute aus der Excel-Welt hier wohler.

Am flexibelsten einzusetzen sind Data Frames. Diese sind auch am ähnlichsten mit Arrays aus Programmiersprachen wie Java, C++, PHP etc. Gleich den Matrizen wird auch der Data Frame aus Listen gebildet.

**Data Frame**

```
namen <- c('Peter','Fritz','Daniel')
hobby <- c('Rennradfahren', 'Schwimmen', 'Klettern')
alter <- c(51, 46, 53)
df.freunde <- data.frame(freunde, hobby, alter)
```

### 2.2.3. Befehle

R kennt fast beliebig viele Befehle und diese können mit nochmals unzähligen vielen Bibliotheken erweitert werden. Meist ist die Syntax jeweils ähnlich und diese besteht aus dem Namen der Funktion und den Argumenten in runden Klammern.

**Funktionsaufruf am Beispiel der Sinus-Funktion**

```
> sin(1/2 * pi)
[1] 1
```

### 2.2.4. Graphiken

Ohne zusätzliche Bibliotheken können in R schon eine Vielzahl von Graphiken erstellt werden. Die dazu notwendige Grundfunktion heisst "plot". Im folgenden Beispiel erstellen wir ein simplex x-y-Diagramm.

**x-y Diagramm**

```
x <- c(1,2,3,4)
y <- c(1,2,9,16)
plot(x,y)
```

Das Resultat ist in Abbildung 2.3 ersichtlich. In diesem Beispiel wurden zwei Vektoren mit je vier Zahlen erstellt, einmal für die x- und einmal für y-Achse. Selbsverständlich lassen sich diese Plots mit Optionen den eigenen Bedürfnissen anpassen (und verschönern). Weitere Informationen dazu können direkt in R mit der Funktion "helpäufgerufen werden.

**Hilfefunktion**

```
help("plot")
```

Direkt in der Konsole von R-Studio ist es noch einfacher: Es genügt den Befehl einzugeben, warten bis die Kurzfassung des Help kommt und dann F1 gedrückt wird. Siehe Beispiel in Abbildung 2.2. Im rechten Teil des Fensters ist der komplette Hilfetext ersichtlich. Weitere Informationen zu R Studio im Kapitel 3.

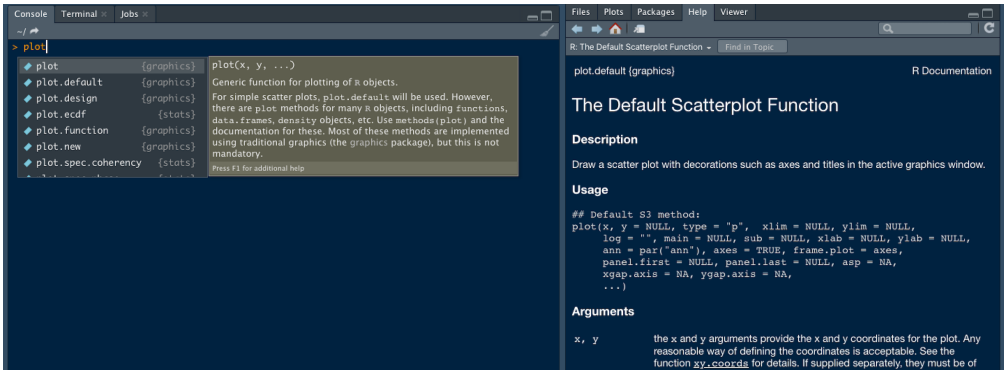


Abbildung 2.2.: Beispiel Hilfe-Funktion

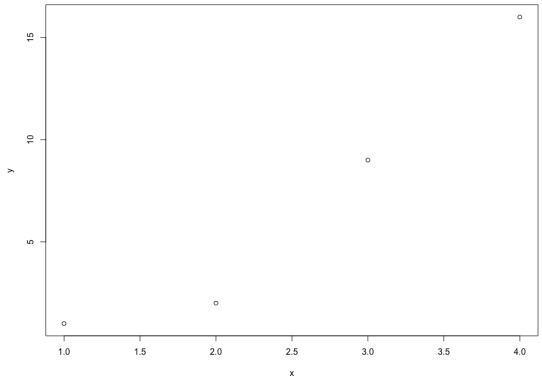


Abbildung 2.3.: x-y Diagramm Beispiel

# 3. RStudio

## 3.1. RStudio

Nachdem die ersten Versuche in der R Konsole erfolgreich waren, macht ein Wechsel ins RStudio Sinn,wo deutlich mehr Komfort und auch mehr Möglichkeiten zur Verfügung stehen. Es ist hier nicht das Ziel eine ausführliche Dokumentation zu RStudio zu schreiben, dazu gibt es genügend Hilfen im Netz und Büchern, sondern ein Überblick zu verschaffen. RStudio bietet im Gegensatz zur Konsole folgende wesentlichen Vorteile:

- gleichzeitiges offenhalten mehrerer Notizbücher
- Bildschirm in mehreren Blöcken:
  - Arbeitsbereich
  - Konsole
  - Umgebung mit aktuellen Variablen etc.
  - Files, Plots etc.
  - Updates, Laden von Bibliotheken etc. via GUI

In der Bedienung unterscheidet sich ein R Script dadurch, dass die Betätigung der Eingabetaste einen Zeilenumbruch einfügt und nicht wie erwartet, die Eingabe abschliesst. Um die Eingabe abzuschliessen muss in Windows Alt + Enter gedrückt werden, auf dem Mac ist es CMD + Enter. In allen folgenden Beispielen erfolgt die Eingabe in einem R Script. Diese Scripte können gespeichert und wiederverwendet werden. Falls RStudio auf einem Server läuft, dann findet die Bedienung in einem Webbrowser statt (siehe auch 9.1), ansonsten ist die Handhabung und Darstellung vergleichbar. Siehe Abbildung 3.1.

Die Fensteraufteilungen innerhalb von RStudio lassen sich in RStudio frei definieren. Die Standardeinstellung ist in Abbildung 3.2 ersichtlich.

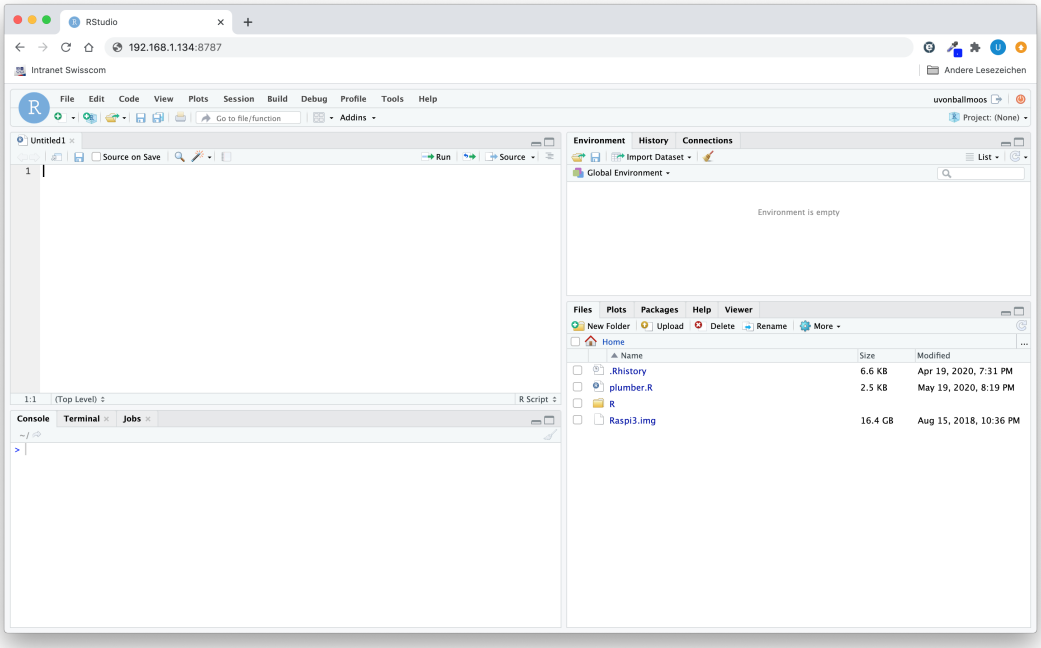


Abbildung 3.1.: Beispiel RStudio Server

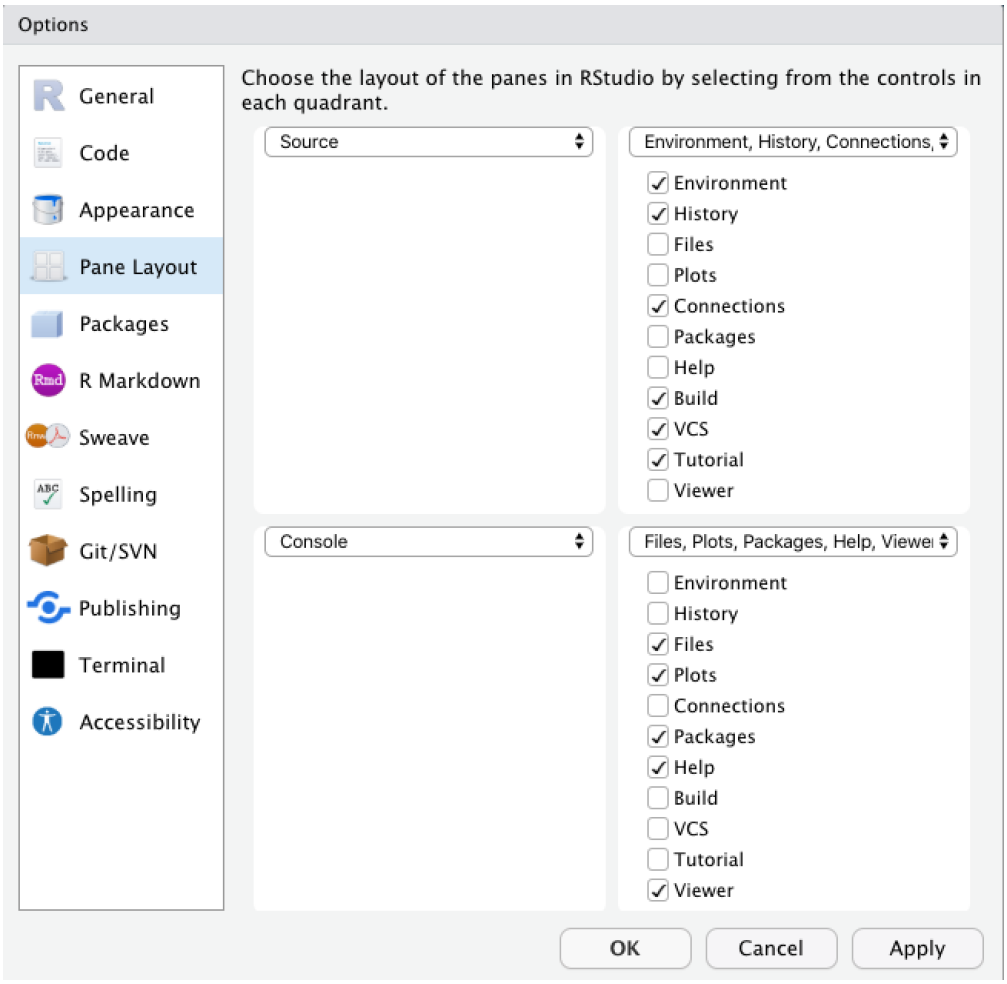


Abbildung 3.2.: Standard Fenstereinteilung in RStudio

# 4. Zeitreihengrundlagen

## 4.1. Um was geht es

Beobachtungen über die Zeit können in einer Tabelle aufgeschrieben und mit deren Hilfe verschiedene Berechnungen durchgeführt werden. Diese Tabelle nennt sich *Zeitreihe*. Beispiele dazu sind:

- Anstieg von Lizenzen nach dem Product Launch
- Heizenergieverbrauch über mehrere Jahre

Aus Sicht des Controlling kommt hinzu, dass nicht nur die Vergangenheit zu betrachten ist, sondern aus dessen Verlauf eine Extrapolation mit allfälligen Prognosen in die Zukunft verglichen werden kann.

Die einfachste Variante dazu wäre, über die verschiedenen Punkte der Zeitachse eine Gerade zu legen und diese in die Zukunft zeigen zu lassen. Nur, wie wird nun diese Gerade gelegt und reicht eine Gerade überhaupt? Die Antworten dazu erhalten wir später im Kapitel 7.

## 4.2. Was ist eine Zeitreihe

Basis für unser «Problem» ist immer eine *Zeitreihe*, also eine Ansammlung von Punkten mit dem Zeitpunkt der Beobachtung, die z.B. einen Messwert, mit ihrem dazugehörigen Zeitstempel repräsentiert.

| **Datum** | **Wert** |
|---|---|
| 2019-10-31 | 7 |
| 2019-11-30 | 9 |
| 2019-12-31 | 11 |

Tabelle 4.1.: Muster zu Datum/Wert

In der Abbildung 4.1 sind die Punkte mit ihren Werten auf der y-Achse und dem Datum auf der x-Achse eingetragen.

## 4.3. Wie geht man mit einer Zeitreihe um?

Leider taugen Datums- und Zeitwerte schlecht als Referenzwert auf der x-Achse oder als Input in einer Formel (x-Achse). Zudem in den meisten Fällen ist der Verlauf nicht absolut, sondern nur relativ von der Zeit abhängig. So lautet z.B. die Frage, wie sieht es am 2. oder 3. Tag aus. Ob der Start dazu am 22. oder 23. November war, ist hingegen

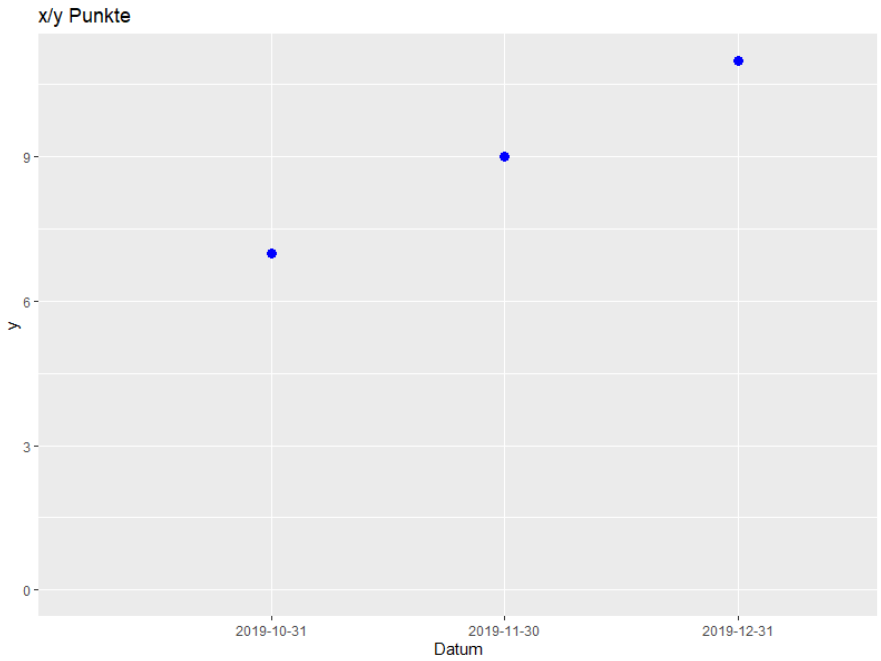


Abbildung 4.1.: Datum/y Punkte

unerheblich. Natürlich kann es von Belang sein, ob der 22. November nun ein Samstag oder Sonntag ist, aber dann ist die Konstellation relevant und nicht das absolute Datum. Das Gleiche gilt natürlich auch bei Feiertagen, beim 25. Dezember wird in aller Regel relevant sein, dass Weihnachten ist und nicht, dass es der 25. Tag im Dezember ist. Es ist somit wichtig zu hinterfragen, ob der Wert tatsächlich vom Datum und nicht von der Verbindung eines Datum mit einem (willkürlichen) Ereignis verbunden ist. Selbst in den Formeln für Sonnenenergieberechnungen wird ein relatives Mass an Anzahl Tagen genommen, wobei hier häufig die Anzahl Tage ab längstem oder kürzestem Tag des Jahres in die Rechnung einfliessen.

Typischerweise sieht eine Formel auch in einer Zeitreihe so aus: $y = f_{(x)}$[1] Das heisst, wir müssen unsere Tabelle um einen x-Wert erweitern, indem wir das Datum um eine fortlaufende Zahl (Index) erweitern. Dabei es am Einfachsten mit einem Start bei 0 (x-Achse, $x = 0$) und überspringen fehlende Werte zwingend auch bei der Indexierung. Natürlich könnte auch jeder andere Startwert genommen werden. In der Tabelle 4.2 und der Graphik 4.2 können wir diese Ergänzung sehen.

| **Nr.(Index)** | **Datum** | **Wert** |
|---:|---|---:|
| 0 | 2019-10-31 | 7 |
| 1 | 2019-11-30 | 9 |
| 3 | 2020-01-31 | 11 |

[1] $f_{(x)}$ bedeutet Funktion von x

Tabelle 4.2.: Einfaches Beispiel einer Zeitreihe. Da der Dezember fehlt, ist auch der Index auszulassen.

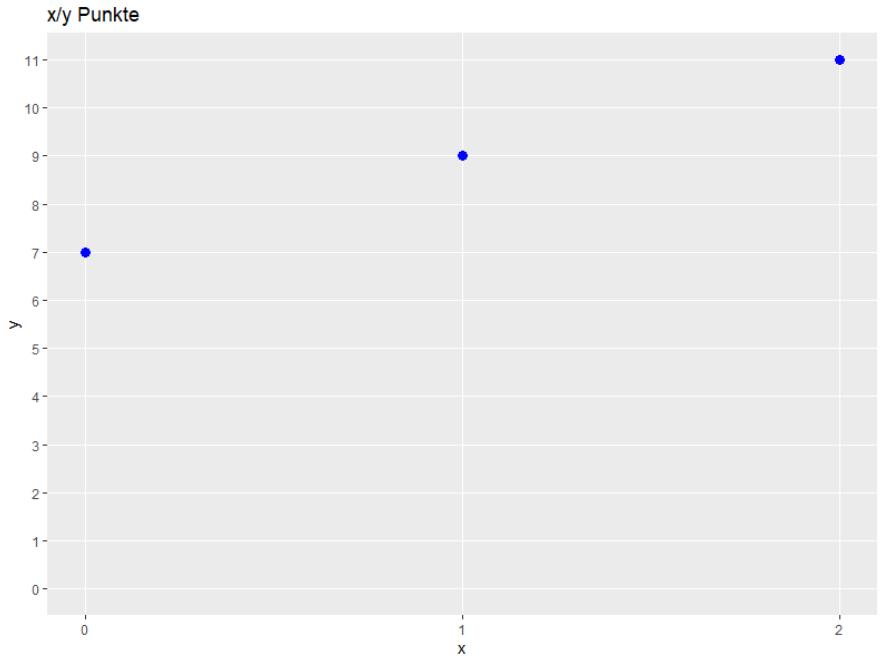


Abbildung 4.2.: x/y Punkte, wobei x jeweils der Index in einer Zeitreihe ist und somit in einer $f_{(x)}$ Formel direkt als unabhängige Variable einfliesst.

Zusammenfassend muss aus diesem Kapitel mitgenommen, dass Datumswerte in einer Zeitreihe Indizes zu erweitern sind. Diese Indizes werden in der Formel $f_{(x)}$ als *unabhängige* Variable eingesetzt.

# 5. Rechengrundlagen für Polynome

## 5.1. Ziel

Sobald Zeitreihen vorhanden sind und weder Blackbox- noch Substantive-Testing möglich sind, braucht es eine Möglichkeit um aus bekannten und verifizierten Zahlentupels (also In- und Output) ein Modell zu bauen, wo mit einer gewissen Annäherung eine Aussage getroffen werden kann, ob ein neuer Input auch zum gewünschten Resultat (Output) führt. Dabei kann der Verlauf linear (also eine Gerade) sein oder er ist nicht linear und muss durch ein höherwertiges Polynom, e-Funktion, log-Funktion etc[1]. abgebildet werden. Das heisst, es stehen eine (möglichst grosse) Anzahl von Inputwerten mit den dazu gehörigen und als korrekt bestätigen Outputwerten zur Verfügung.

Ein typischer Usecase könnte sein: Inzwischen wurden am zu untersuchenden System Änderungen vorgenommen und mangels Testsystem können wir keinen direkten Regressionstest durchführen. In diesem Fall berechnen wir aufgrund der Vergangenheitswerte ein Polynom. Im Anschluss nehmen wir neue Inputs, berechnen mit dem vorhin erhaltenen Polynom einen erwarteten Output und vergleichen ihn mit dem tatsächlichen Output.

Die Methode kann selbstverständlich auch bei Systemen angewendet werden, die sich alles andere als linear und rational verhalten, nämlich bei Menschen. Gerade in der Budgetphase wird eine Prognose für die Zukunft erstellt und mit Hilfe der Polynome erhalten die Prüfenden ein probates Mittel um die Budgetaussagen zu kontrollieren.

### Beispiel

Bei 1000 Kunden fallen CHF 10'000 Lizenzkosten an. Bei 2000 Kunden sind es CHF 19'000 in Folge eines Mengenrabattes. Die Prognose des Verkaufs weist für das Folgejahr 3000 Kunden aus. Mit Hilfe von Polynomen kann nun ein Erwartungswert für die Lizenzen für die 3000 Kunden berechnet und mit dem Budget des Einkaufs vergleichen werden.

## 5.2. Einfache Gerade

Um in einem einfachen x-y-System eine Gerade einzeichnen zu können, wird eine Formel benötigt, um aus den gegeben x-Werten die gewünschten y-Werte zu berechnen. Die generische Formel lautet:

$$y = a \cdot x + b \tag{5.1}$$

In der Formel 5.1 entspricht y dem Resultat, also dem Wert den wir schon haben, a der Steigung der Kurve, b der Verschiebung der Kurve auf der y-Achse und x der Variablen.

[1] Da es in diesem Manuskript um die Methoden und nicht mathematische Herleitungen geht, beschränkt sich der weitere Inhalt auf polynominale Funktionen

Werden nun in der Formel 5.1 die Zahlen 0, 1 und 2 für $x$, $a = 2$ und $b = 3$ eingesetzt, dann sieht das wie folgt aus:

$$3 = 2 \cdot 0 + 3 \tag{5.2}$$
$$5 = 2 \cdot 1 + 3 \tag{5.3}$$
$$7 = 2 \cdot 2 + 3 \tag{5.4}$$

| x | y |
|---|---|
| 0 | 3 |
| 1 | 5 |
| 2 | 7 |

Tabelle 5.1.: Resulate aus $y = ax + b$

Jetzt können die Punkte aus der Tabelle 5.1 entnommen und in einer Graphik dargestellt werden.

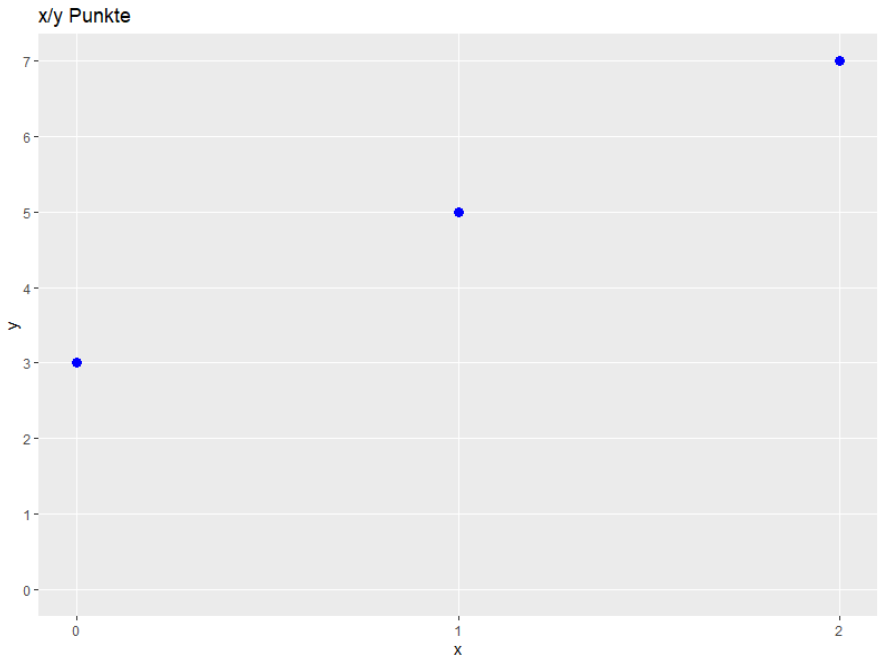


Abbildung 5.1.: Zahlenbeispiel aus $y = ax + b$

Unschwer lässt ich in der Abbildung 5.1 erkennen, dass hier eine einfache Gerade durch die Punkte gelegt wird. (Abbildung 5.2).

Für eine Gerade würden grundsätzlich zwei Punkte reichen.

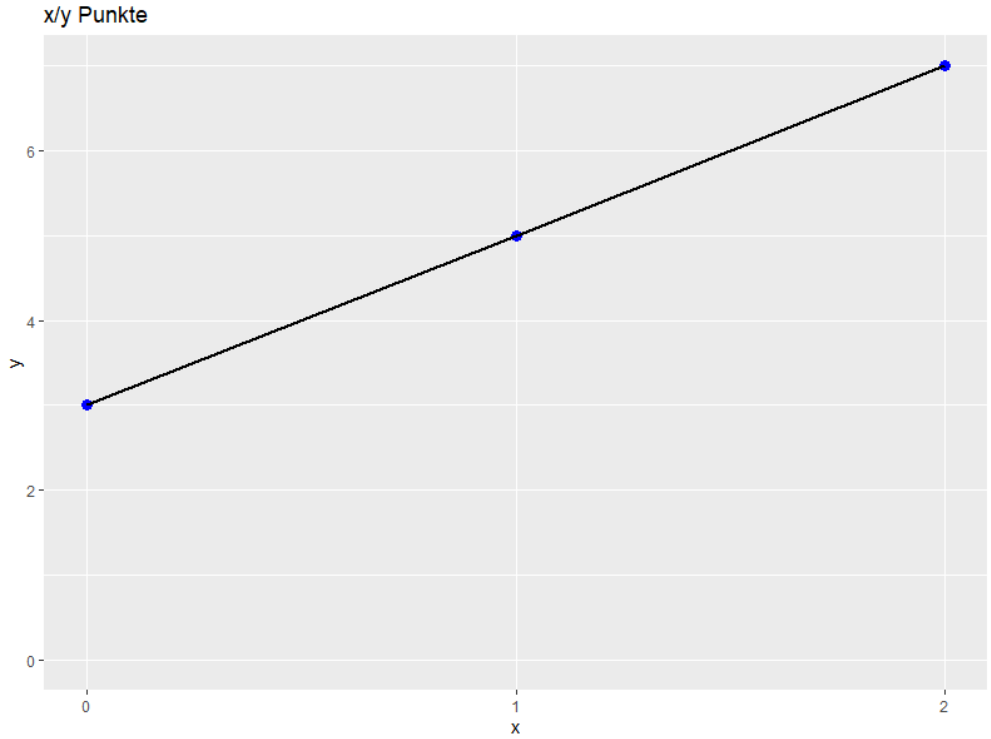


Abbildung 5.2.: Zahlenbeispiel aus $y = ax + b$ und Linie durch die Punkte

## 5.3. Wenn eine Gerade nicht mehr reicht

Leider ist selten alles linear und die Punkte könnten so verteilt sein, dass keine Gerade mehr gelegt werden kann die alle Punkte schneidet. Deshalb muss auf kompliziertere Funktionen zurückgegriffen werden, die graphisch dargestellt als *Polynome* bezeichnet werden (theoretisch auch e, log und weitere Funktionen).

## 5.4. Polynome

Polynome sind die graphische Darstellung von Funktionen die mit der Funktion gemäss der Formel 5.5 berechnet werden.

$$f_{(x)} = a \cdot x^n + b \cdot x^{n-1} + \cdots + c \tag{5.5}$$

Das einfachste nichtlineare Polynom ist die quadratische Gleichung. Hier ist $x^2$ als nichtlineares Element enthalten. Dieses sorgt dafür, dass sich eine Kurve bildet.

$$y = a \cdot x^2 + b \cdot x + c \tag{5.6}$$

Werden die Zahlen 0 bis 4 für x, a = 1, b = 0 und c = 0 eingesetzt, dann kann $y$ gemäss Tabelle 5.2 berechnet und als Graphik in Abbildung 5.3 dargestellt werden.

| x | y |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
| 5 | 25 |

Tabelle 5.2.: Resulate aus $y = ax^2 + bx + c$

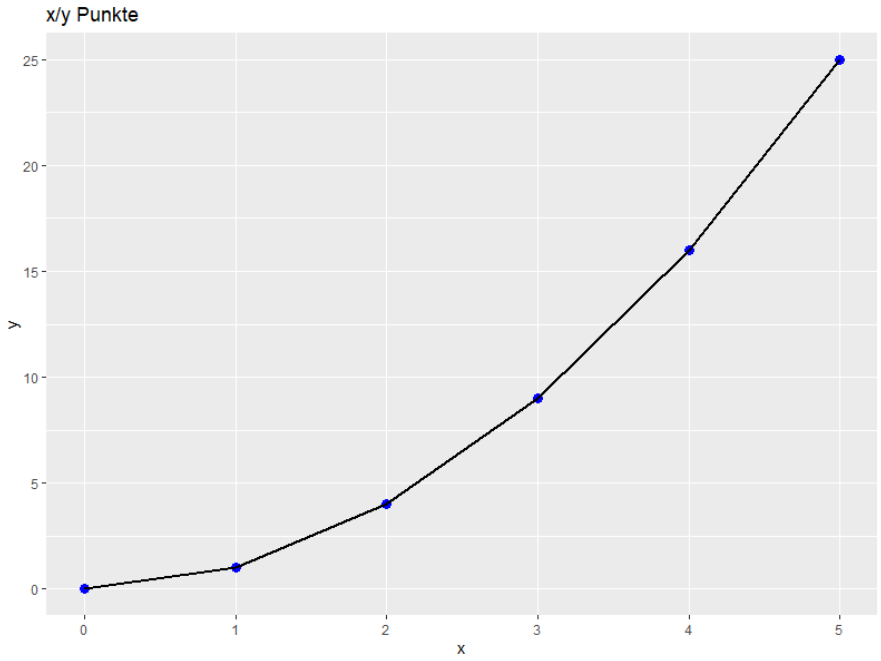


Abbildung 5.3.: Zahlenbeispiel aus $y = ax^2$ und Linie durch die Punkte

Mit den Koeffizienten a, b und c wird die Steilheit und Position der Kurve im Koordinatensystem beeinflusst.

## 5.5. Welchen Polynomgrad benötigen wir?

In Abbildung 5.2 ist zu sehen, dass zwei Koeffizienten benötigt werden um zwei Punkte miteinander zu verbinden. Für drei Punkte sind folglich 3 Koeffizienten und so weiter bis zum n-ten Grad. Dabei wird der höchste auftretende Exponent als Grad des Polynoms bezeichnet. Im Beispiel in der Abbildung 5.4 wird ein Polynom 5 Grades gezeigt. Falls

von Interesse um die Graphik z.B. im Excel nachzubauen, die Formel befindet sich in der Bildunterschrift.

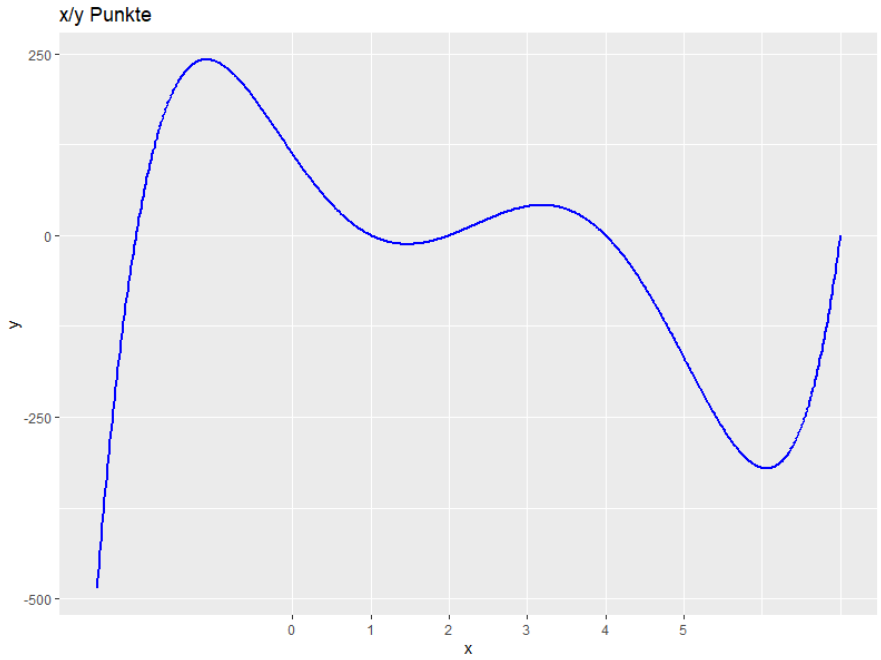


Abbildung 5.4.: Zahlenbeispiel aus $y = x^5 - 12 \cdot x^4 + 35 \cdot x^3 + 20 \cdot x^2 - 156 \cdot x + 112$

## 5.6. Polynome in R darstellen

Dieses Kapitel setzt den Schwerpunkt, wie Polynome in R dargestellt werden können. Wie diese berechnet werden, folgt später.

### 5.6.1. Plot Funktion

Die Standardfunktion in R um Graphiken zu erzeugen ist Plot. Diese Funktion zeichnet Punkte in ein x-y-Diagramm, um eine Linie/Kurve zu erhalten, werden demnach viele Punkte benötigt. Das bedeutet also, wir müssen mit der Funktion auch eine Handvoll Punkte berechnen um diese darstellen zu können. Typischerweise wollen wir für verschiedene x-Werte die y-Werte berechnen und diese darstellen. Dafür benötigen wir als Erstes einen Vektor mit den x-Werten, für die wir y berechnen wollen. [2]

Am einfachsten wird dazu eine Sequenz von x-Werten generiert um im Anschluss diese in der Formel eingesetzt

[2] Hinweis: Gemäss dem Abtasttheorem von Shannon muss die Anzahl Punkte mindestens doppelt so gross sein, wie die höchst vorkommende Frequenz

**Sequenz und Plot Funktion**

```
#Polynome in R darstellen
#Squenz von 0 bis 100 in Schritten von 1 erstellen.
x <- seq(0,100,1)

#y für jedes x berechnen
y <- 0.2 * x^2 - 0.5 * x + 10

#und in Graphik darstellen
plot(x,y)
```

und daraus resultiert die Abbildung 5.5. Dieser Plot sagt zwar etwas aus, besonders schön ist er aber nicht. Mit verschiedenen Optionen lässt sich das Ganze etwas schöner gestalten.

**Resultat plot-Funktion**

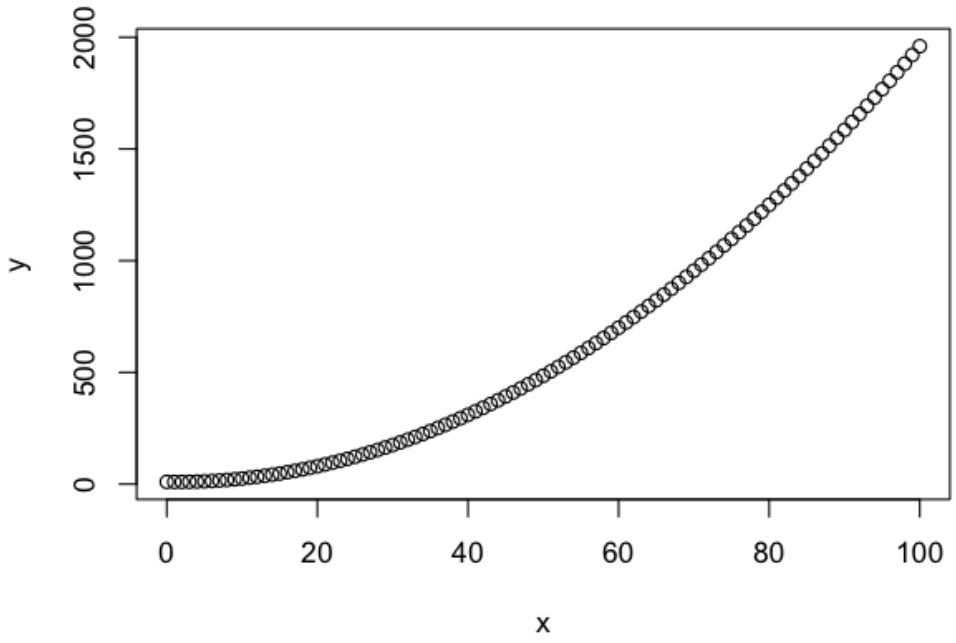


Abbildung 5.5.: Beispiel der Plot Funktion ohne Optionen

Mit ein paar Optionen sieht das Ganze schon viel Schöner aus. Siehe Abbildung **??**

**Plot mit Optionen**

```
#Mit Optionen gestalten
plot(x,y,
type="l",
main="Plot Funktion",
xlab="x",
ylab="y= 0.2 * x^2 - 0.5 * x + 10",
col="blue"
)
```

**Resultat plot-Funktion**

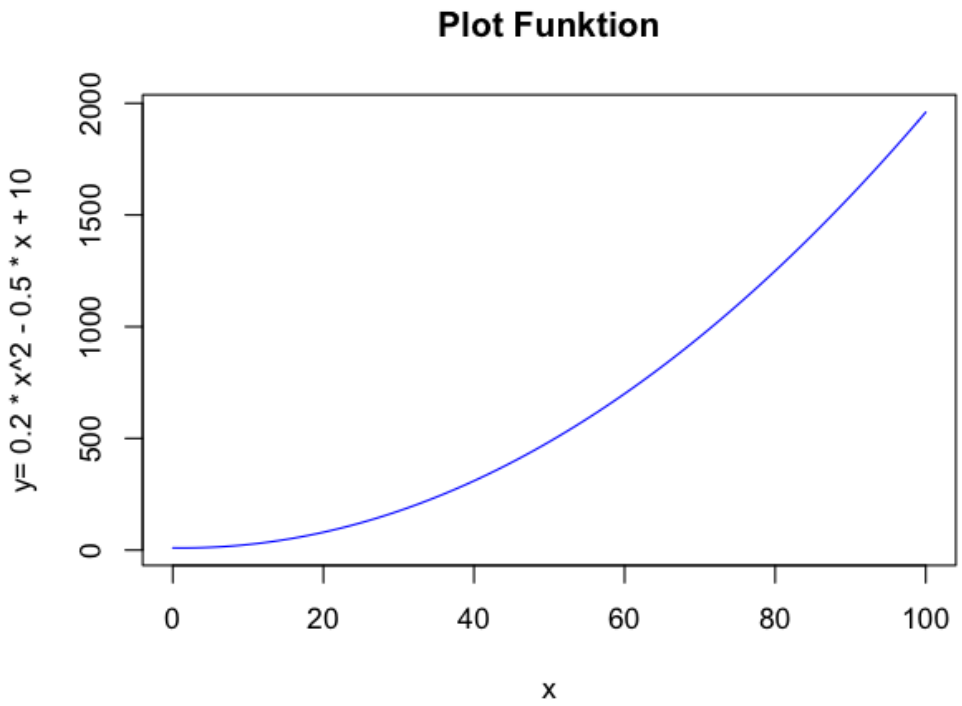


Abbildung 5.6.: Einfacher Plot Optionen

### 5.6.2. ggplot Funktion

Die Standardplotfunktion von R ist zwar einfach in der Bedienung, hat aber viele Einschränkungen was die Gestaltung anbelangt. Für schöne Graphiken bietet sich `ggplot`,

das sehr viel mehr an Freiheiten in der Gestaltung zulässt. Es reicht allerdings nicht, einfach den Befehl "plot"durch "ggplotßu ersetzen. Um ggplot nutzen zu können muss das Paket "ggplot2"geladen werden. Falls das Paket noch nicht installiert ist, muss es entweder via GUI von RStudio oder via Befehl installiert werden.

**Installation "ggplot2"**

```
install.packages("ggplot2")
```

Wenn der Computer mit dem Internet verbunden ist, dann holt sich R das Paket und installiert es auch gleich mit allen weiteren abhängigen Paketen. R kümmerst sich um die Paketverwaltung bei Nutzung der Standardeinstellungen.

Von jetzt an, ist es ausreichend das Paket bei Bedarf zu laden, da es bereits installiert ist.

**Paket "ggplot2"laden**

```
library("ggplot2")
```

Jetzt ist R bereit um die Funktionen aus diesem Paket einzusetzen. Die Funktion "ggplotërwartet die darzustellenden Daten als Data.Frame. Wir erinnern uns, Data.Frames können direkt aus Listen oder Vektoren erzeugt werden. Für die Zahlen aus dem Plot Beispiel von oben wird dazu folgende Befehlssequenz benötigt.

**Date.Frame erstellen**

```
#Data.Frame erstellen und dem Objekt df1 zuweisen.
df1 <- data.frame(x,y)

#Mit head wird der Kopf des Data.Frame
#und die ersten paar Zeilen ausgegeben
head(df1)
```

**Resultat**

```
  x    y
1 0 10.0
2 1  9.7
3 2  9.8
4 3 10.3
5 4 11.2
6 5 12.5
```

Die ggplot Funktion sieht somit wie folgt aus und die Optionen bedeuten:

*data* Zuweisung des gewünschten Dataframes

*aes* Aesthetic Mapping: Auswahl der Spalten aus dem Dataframe und Zuordung zur x- bzw. y-Aches

*geom_point* Geometic Objects: Auswahl der gewünschten Darstellung und den dazugehörigen Optionen in der Klammer

*labs* Labels: Titel, Untertitel etc.

**ggPlot für die Funktion y**

```
ggplot(data = df1, aes(x=x, y=y)) +
geom_point(color = "blue", size = 1) +
labs(title="y= 0.2 * x^2 - 0.5 * x + 10")
```

Das entsprechende Resultat sieht somit so aus:

**Resultat ggplot-Funktion**

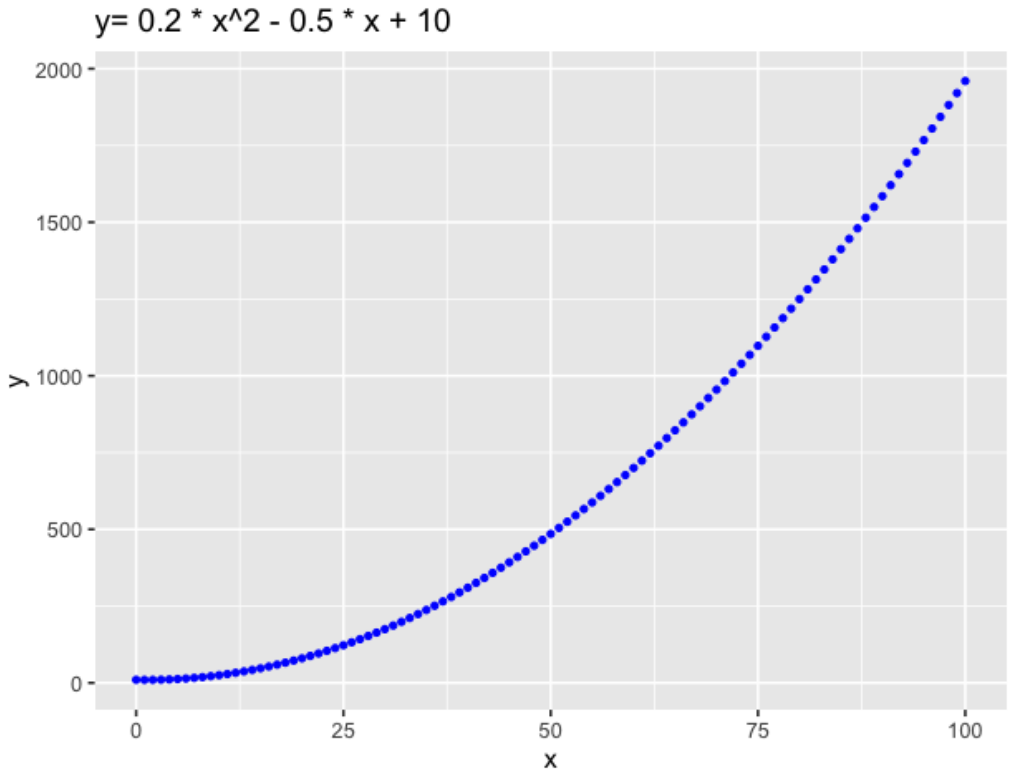


Abbildung 5.7.: Beispiel der ggPlot Funktion mit minimalen Optionen

# 6. Polynom Anwendung

## 6.1. Einleitung

Nachdem im vorangehenden Kapitel erklärt wurde, was Polynome sind geht es nun darum Nutzen daraus zu ziehen. Detaillierte Beispiele folgen später, insbesondere im Teil Fallbeispiele.

## 6.2. Stetige Funktion

Polynome sidn überall dort angebracht, wo ein stetiges Systemverhalten vorhanden ist. Konkret bedeutet dies, dass das System mit einer Funktion $y = f_{(x)}$ zu beschreiben ist, also keine Sprünge hat. Sichtbar ist das immer dann, wenn ein einem x-y-Diagramm eine durchgängige Linie erstellt werden kann, ohne Unterbrechungen. Es ist somit möglich für ein System mit genau einer Formel für jeden x-Wert den dazugehörigen y-Wert zu berechnen.

### 6.2.1. Beispiel

Der Preis ist direkt von der Menge abhängig und es gibt keine Rabattstufen. Wenn 10 Äpfel 10 Franken kosten, dann kosten 100 Äpfel entsprechend 100 Franken und für 3674 Äpfel sind somit 3674 zu bezahlen.

## 6.3. Unstetige Funktion

Das Gegenteil wäre demnach eine unstetige Funktion, das ist immer dann der Fall, wenn die Linie im x-y-Diagramm entweder Lücken oder Sprünge enthält.

### 6.3.1. Beispiel

Der Preis ist von der Menge und der Rabattstufe abhängig, wobei die Rabattstufe in Sprüngen bewertet wird. Bis 10 Äpfel kostet ein Apfel 1 Franken. Von 11 bis 100 Äpfel gibt es einen Rabatt von 5% und ab 101 Äpfel gibt es 10% Rabatt. Würden wir nun dies in einem x-y-Diagramm aufzeichnen, dann sehen wir bei 10 auf 11 Äpfel un beim Wechsel von 100 auf 101 Äpfel einen Sprung. Somit kann diese Funktion nicht mehr mit einer Formel beschrieben werden, sondern es werden drei Formeln benötigt.

**Unstetige Funktion**

```
#Unstige Funtion
#Sequenzen erstellen
x1 <- seq(0,10,1)
x2 <- seq(11,100,1)
x3 <- seq(101, 200, 1)

#y für jedes x berechnen
y1 <- x1
y2 <- x2 * 0.95
y3 <- x3 * 0.9

#Data.Frame erstellen und dem Objekt df1 zuweisen.
df1 <- data.frame(x1,y1)
df2 <- data.frame(x2,y2)
df3 <- data.frame(x3,y3)

#Graphik erzeugen
ggplot() +
geom_line(data = df1, aes(x=x1, y=y1),color = "blue", size = 1) +
geom_line(data = df2, aes(x=x2, y=y2),color = "blue", size = 1) +
geom_line(data = df3, aes(x=x3, y=y3),color = "blue", size = 1) +
labs(title="Unstetige Funktion", x="x", y="y")
```

**Unstetiges Beispiel**

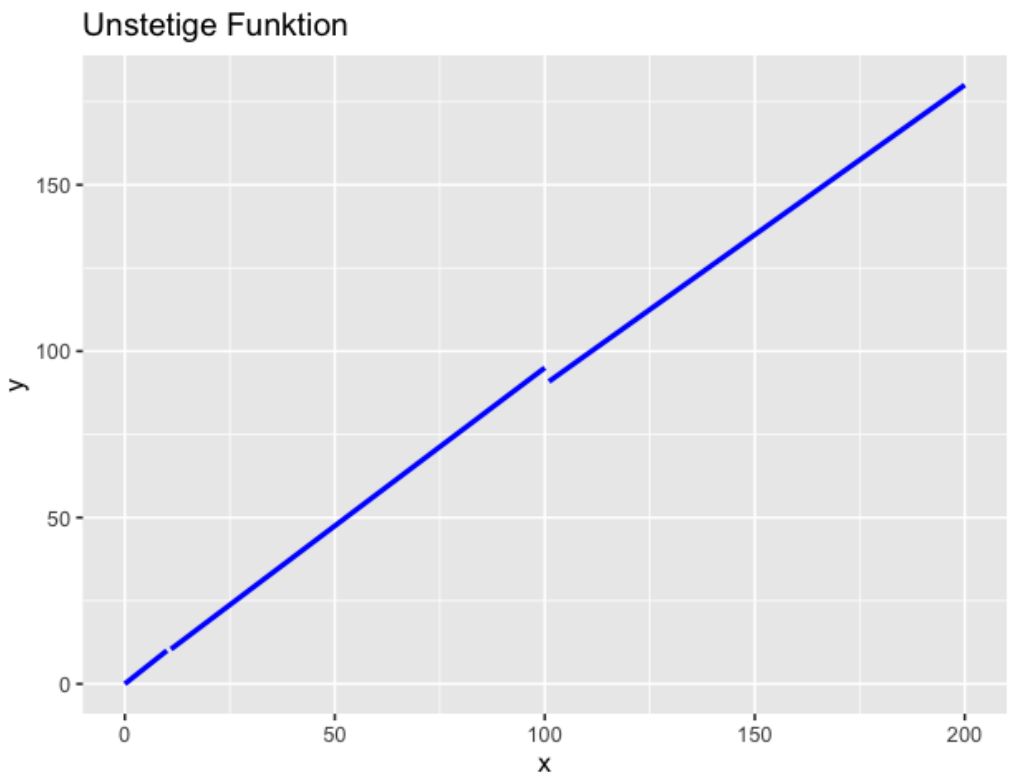


Abbildung 6.1.: Darstellung einer unstetigen Funktion. Sichtbar sind die Sprünge bei x=10 und x=100

# 7. Zeitreihenanalyse

## 7.1. Um was geht es?

Bis jetzt wurden Polynome angeschaut und erklärt, was eine Zeitreihe ist. Im folgenden Kapitel wird die Zeitreihe vertieft angeschaut und insbesondere erläutert, wie damit zu arbeiten ist.

## 7.2. Wie setzen ich die Messpunkte zusammen

Man stelle sich viele Messwerte über einen längeren Zeitraum, am besten über mehrere Jahre und täglichen Messwerten, vor. Jeder einzelne dieser Punkte setzt sich mathematisch wie folgt zusammen:

- Rauschen (stochastisches Verhalten zwischen den einzelnen Messwerten und nur sichtbar, wenn in kleinen Zeitabständen gemessen wird)
- Saisonale Schwankungen
- Konjunkturelle Schwankungen (dazu sind Messdaten über viele Jahre nötig)
- Trend

Das heisst nichts anderes, als dass jeder einzelne Punkt Bestandteil folgender Formel ist. Manchmal wird noch ein weiterer Faktor addiert, die irregulären Komponenten. Das können z.B. Störfaktoren sein, die auch als *unsystematische Komponenten* bezeichnet werden.

$$\text{Messwerte} = \text{Rauschen} + \text{Saisoneffekte} + \text{Konjunktureffekt} + \text{Trend} \tag{7.1}$$

Für die Langzeitprognose ist nur noch der Trend relevant, alles andere muss mit mathematischen Methoden entfernt werden. Möglichkeiten dazu werden weiter unten kurz diskutiert.

## 7.3. Trendlinien

### 7.3.1. Gerade als linearer Trend

Die Gerade als Trendlinie darf in ihrer Wichtigkeit nicht unterschätzt werden. Sie sieht zwar im Vergleich zu höhergradigen Polynomen etc. unspektakulär aus, ist aber trotzdem eine wichtige Trendfunktion, weil sie meistens für eine Prognose in die nahe Zukunft genügend genau ist und die wichtigste Aussage liefert, nämlich ob es auf- oder abwärts geht oder stagniert. Eine Gerade als Trendlinie ist ein linearer Trend.

### 7.3.2. Polynom als nicht linearer Trend

Polynome helfen enorm um eine gut passende Kurvenfunktion für die vorhandenen Werte in einer Zeitreihe zu finden. In den meisten Fällen wird ein solches Polynom sogar deutlich besser passen als eine Gerade. Aber das Polynom birgt auch gewaltige Gefahren für die Extrapolation. Je höher der Grad des Polynome ist, desto mehr wird die Extrapolation in der Zukunft abdriften und jenseits jeder vernünftiger Erwartung liegen. Gut sichtbar ist das im Beispiel der Abbildung 5.4 wo es außerhalb von ca. 0 bis 4 schon enorme Steilheiten der Kurve gibt.

## 7.4. Wie kommt man zu einem Trend

Bevor wir im Kapitel 8 zu Berechnung kommen müssen wir uns zuerst überlegen, wie aus vielen Punkten ein Trend sichtbar wird. In der Formel 7.1 haben wir gesehen, wie sich die Messpunkte zusammensetzen und jetzt geht es der Frage nach, wie aus diesem Punktehaufen zur Trendline resultiert.

### 7.4.1. Auftrennung von Messpunkten

Die folgende Erklärung bezieht sich auf ein Mustervorgehen, wie man mit Messwerten vereinfacht umgehen kann folgt im Anschluss.

Die meisten von uns mögen sich an die Zeiten der grossen und teuren Stereoanlagen erinnern. Auf den besseren Verstärken gab es nicht nur den Lautstärkeregler, sondern auch einen *Equalizer*. (Auf den professionellen Mischpulten gibt es diese weiterhin.) Was macht der Equalizer? Auf dem Equalizer sind für die verschiedenen Frequenzbereiche Schieberegler vorhanden, mit denen das gewünschte Frequenzband verstärkt oder abgeschwächt werden kann. Wenn jeder Messpunkt mit dem Nächsten verbunden wird, dann ergibt sich eine Linie, die die verschiedenen Frequenzen beinhaltet.

Beim Auftrennen der Messpunkte wird nicht anderes durchgeführt, als mit den Schiebereglern auf dem Equalizer. Wir schwächen das hochfrequente Rauschen so stark ab, dass es verschwindet. Falls wir Tageswerte haben, dann wäre hier die Frequenz:

$$f = \frac{365 \text{ Tage}}{1 \text{ Jahr}} \tag{7.2}$$

Man beachte, dass damit nicht die Tageswerte gelöscht, sondern nur die Tagesausschläge geglättet werden. Was bleibt übrig? Der Verlauf der noch zusammengesetzt ist aus

$$\text{Messwerte ohne Rauschen} \quad = \quad \text{Saisoneffekte} + \text{Konjunktureffekt} + \text{Trend} \tag{7.3}$$

In der Realität wird dieses hochfrequente Rauschen meist so entfernt, dass aus Werten die häufiger als täglich vorhanden sind, z.B. 5 Minuten Werte (Standard bei Unix-Systemen), durch ein arithmetisches Mittel oder einer ähnlichen Funktion zu einem Tageswert zusammengefasst werden.

### 7.4.2. Filtermethoden

Leider gibt es für Messwerte nicht so einfache Filter, mit ein paar Spulen und Kondensatoren, wie im analogen Equalizer. Somit muss gerechnet werden. Grob sieht das dann so aus. Der Mathematiker Jean Baptiste Joseph Fourier, konnte nachweisen, dass jede beliebige Kurve sich aus verschiedenen Sinus- und Cosinus-Kurven zusammensetzen lässt. Im Beispiel 7.1, alle Teilbilder decken den Bereich $0 < x < 2 \cdot \pi$[1], zeigt Teilbild A einen einfachen Sinus. Im Teilbild B wurde $y = \frac{Sin(3x)}{3}$ berechnet; also die dreifache Frequenz von Teilbild A, dafür wurde die Amplitude[2] auf $\frac{1}{3}$ reduziert.

Im Teilbild C werden die beiden Kurven aus den Teilbildern A und B addiert und es wird sichtbar, der Sinus hat bereits eine gewisse Ähnlichkeit mit einem Rechteck. Zum Schluss wurden in Teilbild D die Berechnung aus Teilbild C bis $y = \frac{Sin(15x)}{15}$ weitergeführt und hier ist die Kurve nun schon deutlich näher bei einem Rechtecksignal als beim Sinus.

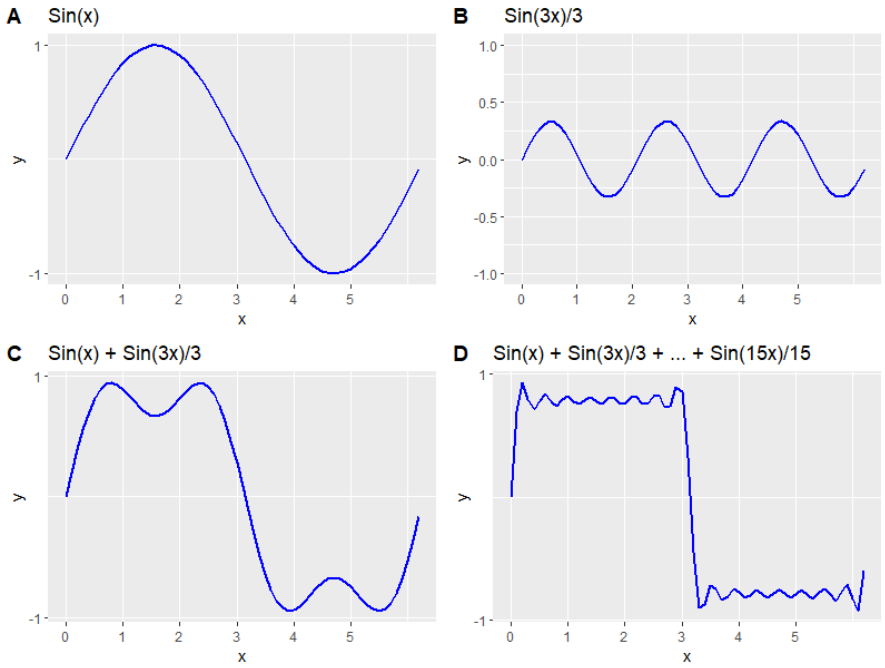


Abbildung 7.1.: Vom Sinus zum Rechteck

Mit Hilfe einer Fourier-Transformation oder einer Laplace-Transformation (unterschiedliche mathematische Wege) wird geschaut, welche Frequenzen in welcher Amplitude vorhanden sind. Graphisch aufbereitet ist in Abbildung 7.2 ersichtlich, mit der Quelle Teilbild D in Abbildung 7.1, die vorkommenden Frequenz mit ihren Amplituden. Hier wird wieder das Spektrum ersichtlich, wie es die ganz teuren Equalizer in einem Display anzeigen. Zum Filtern braucht es jetzt nur noch definieren, welche Frequenzen weg müssen,

[1] $0 < x < 360°$

[2] Amplitude = grösster Ausschlag einer Schwingung oder eines Pendels aus der Mittellage; Schwingungsweite

setzen die Amplitude also auf 0 und transformieren das ganze wieder zurück in eine Kurve. Der erste Peak setzt somit bei der Hauptfrequenz an. Dies wird gemäss 7.4 wie folgt berechnet:

$$f = \frac{1}{\text{Periodenlänge}} = \frac{1}{2 \cdot \pi} = 0.159154 \tag{7.4}$$

Beim Blick auf den zweiten Peak, dann müsste dieser in der Graphik 7.2 gemäss der oben berechneten Fourierreihe eine Amplitude von $\frac{1}{3}$ haben, also 0.3333, was auch der Fall ist.

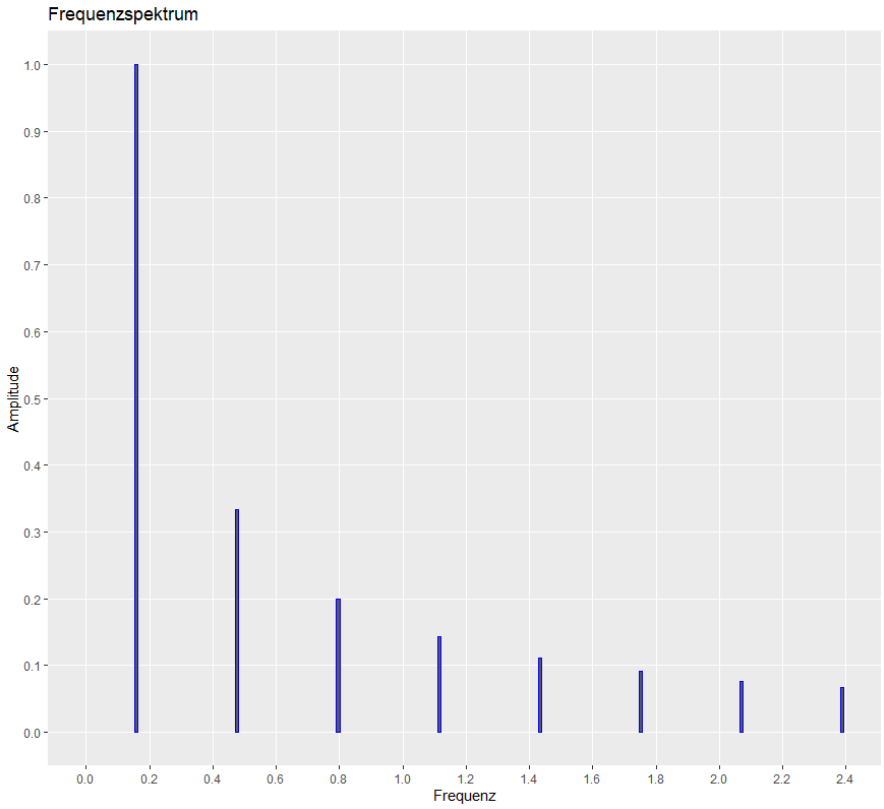


Abbildung 7.2.: Frequnzspektrum zu Teilbild D in Abbildung 7.1

## 7.5. Umsetzung Frequenzanalyse

Im folgenden Beispiel wird eine komplexe Kurvenform berechnet. Sie ist aus einer Cosinus- und zwei unterschiedlichen Sinusfunktionen zusammengesetzt.

- $x1 = \cos(2 \cdot \pi \cdot t)$
- $x2 = 0.75 \cdot \sin(2 \cdot \pi \cdot 4 \cdot t)$

- $x3 = 2 \cdot \sin(2 \cdot \pi \cdot 7 \cdot t)$

In diesem Beispiel ist höchst vorkommende Frequenz in der Kurve x3 7 Hz. Der geniale Mathematiker Shannon hat bewiesen, dass die Abtastfrequenz mindestens der doppelten höchsten vorkommenden Frequenz entsprechen muss. Benannt ist die Frequenz allerdings als *Nyquist-Frequenz*. Für dieses Beispiel bedeutet dies demnach, dass die Samplingfrequenz mindestens 14 Hz betragen muss. Umgemünzt auf eine Zeitreihe wie, sie häufig aus Sicht Revision vorkommt, also einer Jahresbetrachtung, können bei monatlichen Messungen somit keine Effekte berechnet werden, die häufiger als alle zwei Monate vorkommen. $f = \frac{12 \text{ Monate}}{2} = 12$ Monate.

$$f_{Nyquist} = \frac{f_{\text{Abtast}})}{2} \tag{7.5}$$

In den ersten Schrittfolgen werden die drei Kurven berechnet und dann addiert.

**Musterkurve zusammenstellen**

```
ts <- 0.005  #Samplingperiode
fs <- 1/ts #Samplingfrequenz
t <- seq(0,100,by=ts) #Sequenz erstellen
x1 <- cos(2*pi*t) #Kurve 1
x2 <- 0.75*sin(2*pi*4*t) #Kurve 2
x3 <- 2*sin(2*pi*7*t) #Kurve 3

x = x1 + x2 + x3 #Kurve zusammensetzen
```

in den zweiten Schrittfolgen werden die drei Kurven einzeln und in der Summe graphisch dargestellt.

**Musterkurven anzeigen**

```
par(mfrow = c(2, 2)) #2 Spalten und 2 Zeilen für Bilder
#Bilder mit horizontaler 0-Linen
plot(x1, xlim=c(0,500), ylim=c(-2,2),
type="l", main = "cos(2*pi*t)")
abline(h=0, lty=3)

plot(x2, xlim=c(0,500), ylim=c(-2,2),
type="l", main = "0.75*sin(2*pi*4*t) ")
abline(h=0, lty=3)

plot(x3, xlim=c(0,500), ylim=c(-2,2),
type="l", main = "2*sin(2*pi*7*t)")
abline(h=0, lty=3)

plot(x, xlim=c(0,500), ylim=c(-4,4),
type="l", main = "Zusammengesetzte Kurve\nx = x1 + x2 + x3")
abline(h=0, lty=3)
```

In den dritten Schrittfolgen erfolgt die Berechnung des Spektrums und die Anpassung der normalisierten Ausgabe an die Frequenzen und der Amplitude.

**Spektrum berechnen**

```
#Spectrum berechnen, keine log-Skala, 5 Spikes im Kern, kein Plot
x.spec <- spectrum(x,log="no",span=5,plot=FALSE)
#Ausgabe mit Samplingfrequenz multiplizieren.

#Da die Standardausgabe in Anzahl Zyklen pro Sampling Interval ist.
spx <- x.spec$freq * fs

#Es macht Sinn diespektrale Dichte mit 2 zu multiplizieren,
#damit die Fläche unter dem Periodogramm auch der
#Varianz der Zeitreihe entspricht.
spy <- 2*x.spec$spec
```

In der vierten und letzten Schrittfolge wird das Frequenzspektrum angezeigt.

**Frequenzspektrum anzeigen**

```
#Umstellen auf eine Graphik pro Seite und Frequenz
#von 0 bis 10 einschränken
par(mfrow = c(1, 1))

plot(spy~spx,xlab="Frequenz",ylab="Spektrum",type="l", xlim=c(0,10))
```

## 7.6. Weitergehende Zeitreihenmethoden

Neben den hier gezeigten Grundprinzipien zu Zeitreihen gibt es noch viele andere Möglichkeiten, insbesondere das *AutoRegressive Integrated Moving Average (ARIMA)* Modell und die *Box-Jenkins-Methode*. Für beide Methoden wurden Bibliotheken für die Anwendung in R entwickelt. Für die Anwendung des ARIMA-Modells verweist der Author auf das Buch «Angewandte Zeitreihenanalyse mit R» von Schlittgen (2015).

# 8. Regression

## 8.1. Was ist eine Regression

Bei einer Regression geht es darum eine Beziehung zwischen einer abhängigen und einer oder mehreren unabhängigen Variablen herzustellen. Die abhängige Variable ist die Zielgrösse, also der Untersuchungsgegenstand. Als unabhängige Variablen werden die beeinflussenden Werte genannt, in unserem Fall z.B. die Messwerte, die am Eingang des Systems gemessen werden. Als allgemeingültiges Beispiel könnte man die Anzahl Regentage und das Gewicht von Äpfeln in Relation setzen. Die Anzahl der Regentage wäre dann die unabhängige Variable und das Apfelgewicht die abhängige Variable. Es wird somit versucht anhand der Regentage das Gewicht der Äpfel zu erklären.

Mit einer Regression wird mit Hilfe von unabhängigen Eingangsgrössen versucht den Verlauf der abhängigen Variablen zu erklären.

Mit anderen Worten, mit einer einfachen Regressionsanalyse können folgende Fragestellungen untersucht werden:

*Ursache* Gibt es einen Zusammenhang, zwischen der abhängigen und unabhängigen Variablen

*Wirkung* Wie verändert sich die abhängige Variable, wenn sich die unabhängige verändert (z.B. wenn sich die Systemeingangsgrösse ändert)

*Prognose/Vorhersage* Können die Messwerte der abhängigen Variable durch die Werte der unabhängigen Variable vorausgesagt werden?

*Qualität* Mit $r^2$ (siehe 8.2.2) können wir eine Aussage der Abhängigkeit in % machen.

### 8.1.1. Definition

Nach Gross (2010, S. 191) gilt:

> Man spricht von einem Modell der linearen Einfachregression, wenn folgende Annahmen als gültig angesehen werden:
>
> (a) Die Beobachtungen $y_i, i = 1, \ldots, n.$ von $Y$ können als voneinander unabhängige Realisationen aus einer Normalverteilung mit unbekannten Parametern $\mu_i$ und $\sigma$ angesehen werden.
>
> (b) Die Parameter $\mu_i$ hängen linear von den Beobachtungen $x_i$ der Variablen X ab, d. h. es gibt unbekannte Parameter $\beta_0$ und $\beta_1$ mit
>
> $$\mu_i \quad = \quad \beta_0 + \beta_1 \cdot x_i$$
>
> für $i = 1, \cdots, n$.

Mit anderen Worten, wir können jeden Punkt in der Form

$$y = a \cdot x + b \tag{8.1}$$

darstellen. Also genau so, wie es im Kapitel der Polynome schon beschrieben ist. Weiter ist es möglich eine nicht lineare Regression mit höherwertigen Polynomen darstellen. In Abbildung 8.1 ist der Weg von $x_i$ zu $y_i$ mit jeweils drei Punkten dargestellt. das $f_{(x)}$ symbolisiert die zu suchende Regressionsformel.

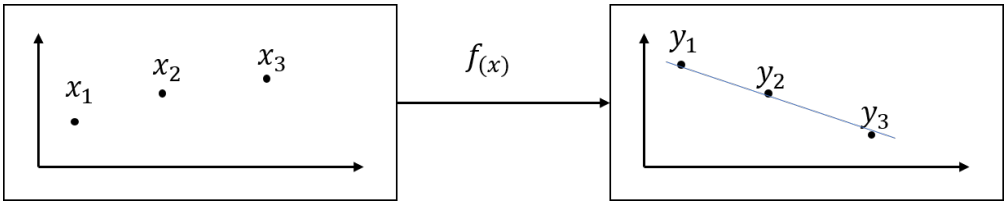


Abbildung 8.1.: Beispiel einer einfachen linearen Regression

## 8.2. Wie wird eine Regression berechnet

### 8.2.1. Methode der kleinsten Quadrate

Grundsätzlich wird versucht eine Kurve so zu berechnen, dass sie sich optimal an die gegebenen Punkte schmiegt. Damit der Aufwand mit negativen Werten wegfällt, wird ganz simple jede Abweichung zwischen dem gegeben Punkt und dem Berechneten mit sich selbst multipliziert, das bedeutet auch bei einer negativen Abweichung ergibt sich ein positives Resultat (weil $-3 \cdot -3 = 3 \cdot 3 = 9$ ist) (Kreyszig, 1968, 258ff). Würde man statt der Quadrate nur die Linien messen, dann müsste ein Wert gesucht werden, der möglichst nach bei 0 ist, was das ganze verkomplizieren, da sowohl positive als auch negative Werte resultieren können. So ist z.B. $-0.3$ näher bei 0 als 0.4.

Im Beispiel der Abbildung 8.2 sind blau die Messpunkte in der Reihe (3, 6, 7) eingetragen. Anschliessend werden folgende Schritte durchgeführt:

1. Es wird eine Linie geschätzt und eingezeichnet
2. Zwischen jedem Punkt und der Linie wird ein Quadrat eingezeichnet, dessen Kantenlänge exakt dem Abstand zwischen dem Punkt und der Linie entspricht
3. Summenbildung über alle Quadratflächen
4. Ändern der Linienneigung und horizontale Verschiebung bis die Summe der Quadratflächen am kleinsten ist. Dies ist dann die Regression.

Natürlich gibt es hier bessere und schlechtere Methoden um die Linie möglichst schnell optimal zu platzieren, aber diese Aufgabe überlassen wir Computern und Programmen wie R (oder Excel) etc. Auf jeden Fall ist jetzt bekannt, wie die Regression zu Stande kommt. Die gleiche Methode wird auch bei komplexeren Kurvenformen, also Polynomen beliebigen Grades, $e^x$, $log(x)$ etc. angewendet.

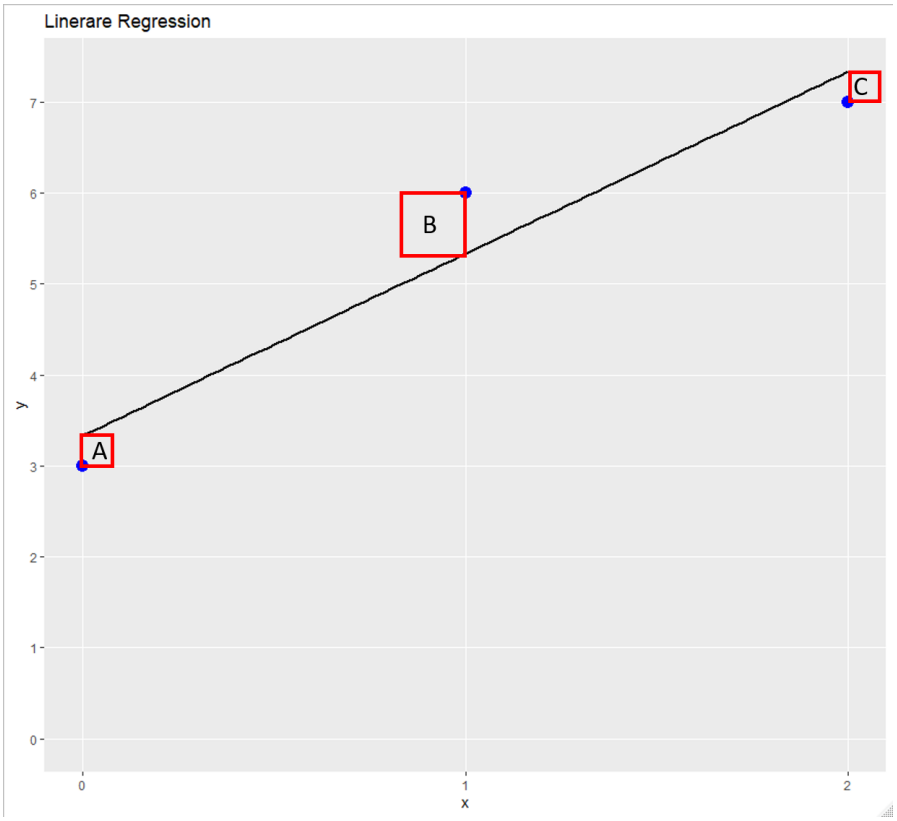


Abbildung 8.2.: Beispiel einer linearen Regression mit Methode der kleinsten Quadrate

### 8.2.2. Qualität einer Regression

Eine schöne Kurve ohne Information zur Qualität nützt herzlich wenig. Deshalb existiert das *Bestimmtheitsmass* $r^2$ oder auch *Güte* genannt und je näher dieser Wert bei 1 liegt, desto besser ist die Qualität der Regression (*Multiple Regressionsanalyse* 2018, Kapitel 3.7). 0 bedeutet demnach, dass die Regression überhaupt nicht passt und 1 dass sie perfekt passt. Die Formel zur Berechnung lautet gemäss 8.2 mit den Variablen:

- $y$ als Messwert
- $\hat{y}$ geschätzter Wert (also der Regressionslinie)
- $\bar{y}$ Mittelwert über alle Messwerte $\bar{y} = 5.33 = \frac{3+6+7}{3}$

$$\begin{aligned} R^2 &= \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \\ &= \frac{(3.33 - 5.33)^2 + (5.33 - 5.33)^2 + (7.33 - 5.33)^2}{(3 - 5.33)^2 + (6 - 5.33)^2 + (7 - 5.33)^2} \\ &= 0.92307 \end{aligned} \tag{8.2}$$

Somit passt das Modell (also die Regressionsgerade) und ist zu rund 92% zu erklären. Das heisst aber auch, dass rund 8% von anderen Einflüssen abhängig sind, also nicht durch diese Abhängigkeit erklärbar sind. Wichtig ist dabei, dass aufgrund eines Bestimmtheitsmasses keine Aussage über die Qualität der Kurve mit Blick in die Bereiche ausserhalb der Messpunkte möglich ist. Das Bestimmtheitsmass sagt in diesem Beispiel nichts über die Qualität in der Zeit vor der ersten Messung oder in der Zukunft nach der letzten Messung aus.

Das Bestimmtheitsmass sagt nichts über die Qualität der Regression ausserhalb des Messzeitfensters aus. Je weiter weg wir uns vom Messzeitbereich entfernen, desto grösser kann der Fehler werden, insbesondere bei höherwertigen Polynomen.

## 8.3. Regressionskurven für Prognose

Jetzt wird geklärt, wie dies für den Blick in die Zukunft behilflich ist. Das ist so einfach, dass wir gleich mit einem Beispiel starten können. Angenommen es bestehen Werte aus 3 Messungen und für die Regressionsberechnung die Werte $0 < x < 3$ zugewiesen. Daraus entsteht eine Formel und aus der Resultate für $x \geq 3$ berechnet werden.

Die Berechnung sieht demnach so aus. Als Beispiel haben wir für die Zeitspanne Januar bis April:

| **Monat** | **x** | **y** |
|---|---|---|
| Januar | 0 | 100 |
| Febraur | 1 | 120 |
| März | 2 | 144 |
| April | 3 | 172 |

Tabelle 8.1.: Muster für Regression Demand Actual Daten

und es ist eine Prognose für Mai bis Juli zu berechnen. Mit Hilfe von R (oder Excel etc.) finden wir heraus, dass dir Formel für obige Tabelle wie folgt sein könnte:

$$y = 24 \cdot x + 98 \tag{8.3}$$

Mit der Formel 8.2 ergibt sich ein Bestimmtheitsmass von 0.9945, also ein recht guter Wert. Aber wie sieht es aus, wenn statt einer linearen Regression ein Polynom 2. Grades zum Zug kommt, also $y = a \cdot x + b$?

Wieder mit Hilfe von R resultiert die die Formel:

$$y = 2 \cdot x^2 + 18 \cdot x + 100 \tag{8.4}$$

Hier ergibt sich wieder mit der Formel 8.2 ein Bestimmtheitsmass von 1. Passt also noch besser, genau genommen sogar perfekt, passender geht es gar nicht! Wie bereits oben erwähnt, kann aber aufgrund des Bestimmtheitsmasses kein Entscheid getroffen werden , welches Modell für den Blick in die Zukunft besser ist. Besser geeignet ist dazu ein Blick auf die Visualisierung 8.3.

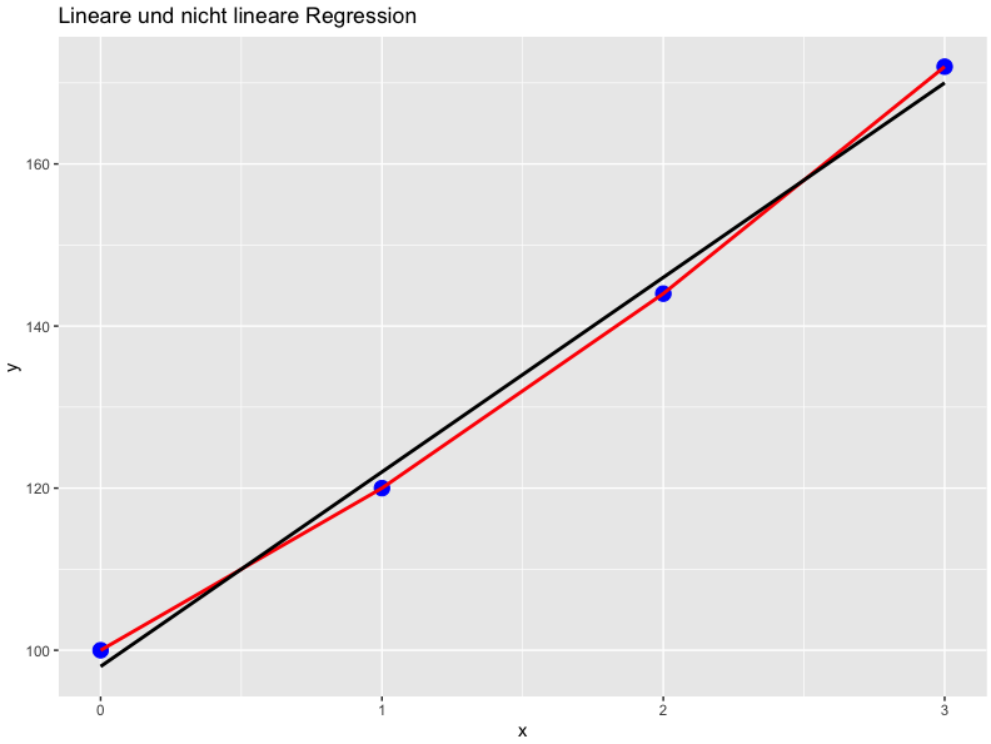


Abbildung 8.3.: Blau: Messpunkte, schwarz lineare Regression, rot Regression mit Polynom 2ten Grades.

Aber welche der beiden Kurven dürfte jetzt besser für den Blick in die Zukunft sein? An dieser Stelle ist zu überlegen, woher diese Messpunkte überhaupt kommen. Eine typische Frage könnte sein, weshalb gibt es eine Steigerung und was ist der Auslöser und wir bedienen uns an zwei Beispielüberlegungen:

1. Es findet eine Migration statt und wegen verschiedener Anzahl Arbeitstage pro Monat gibt es Verschiebungen → Für die nächsten Monate dürfte die lineare Steigung passend sein.

2. Es wurde ein neues Produkt lanciert und daher ist Steigerung pro Monat 20% → Eine quadratische Regression passt besser.

Jetzt können wir die Tabelle 8.2 für die nächsten drei Monate ergänzen:

| Monat | x | y | linear | quadratisch |
|---|---|---|---|---|
| Januar | 0 | 100 | | |
| Febraur | 1 | 120 | | |
| März | 2 | 144 | | |
| April | 3 | 172 | | |
| Mai | 4 | | 194 | 204 |
| Juni | 5 | | 218 | 240 |
| Juli | 6 | | 242 | 280 |

Tabelle 8.2.: Muster für Regression Demand Actual Daten

Damit wurde erläutert, wie mit Hilfe von Regressionen ein Blick in die Zukunft gewagt werden kann, aber auch wo auzupassen ist um die Wahrscheinlichkeit einer Fehlprognose zu reduzieren.

## 8.4. Multiple Regression

Mit der einfachen Regression besteht die Limite, dass nur genau eine unabhängige Variable vorkommen darf, was im folgenden Beispiel betrachtet wird:

Beispiel mit Äpfeln:

| Jahr | Regentage | ∅ Apfelgewicht (g) |
|---|---|---|
| 2010 | 134 | 178 |
| 2011 | 141 | 176 |
| 2012 | 127 | 181 |
| 2013 | 130 | 179 |

Tabelle 8.3.: Vergleich Regentage und durchschnittliches Apfelgewicht

Mit Hilfe einer 3-D Darstellung bekommen wir die Möglichkeit sowohl die Zeit, Anzahl Regentage und Apfelgewicht gleichzeitig in einer Graphik zu zeigen, aber es bedarf eines gewissen räumlichen Verständnisses um daraus etwas interpretieren zu können. Die obigen Zahlen sind in Abbildung 8.4 in einer 3-D Darstellung ersichtlich.

Ganz am Anfang dieses Kapitels steht, dass es sich um eine oder mehrere unabhängige Variablen geht. Das einfache Apfelbeispiel zeigt dies bereits eindrücklich. Oder ob wir uns wohlfühlen ist nicht nur von der Temperatur sondern auch vom Wetter, Hunger, Durst etc. abhängig. Wir haben also mehrere von einander unabhängige Faktoren die das Resultat beeinflussen. Siehe auch *Multiple Regressionsanalyse* (2018).

Bevor mit einer multiplen Regression gestartet wird muss die Unabhängigkeit der verschiedenen erklärenden Variablen überprüft werden.

### 8.4.1. Definition

Nach Gross (2010, S. 205) gilt:

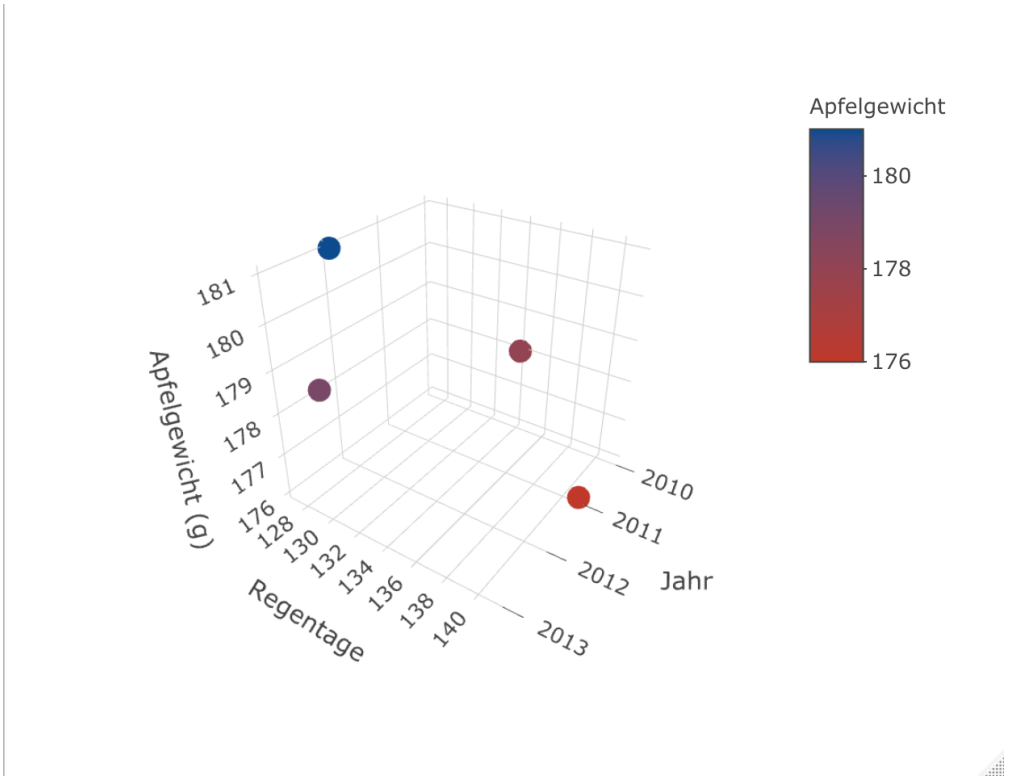


Abbildung 8.4.:

Eine Variable $Y$ erfülle als Zielgrösse dieselben Voraussetzungen wie im Modell der linearen Einfachregression (siehe 8.1.1). Dann spricht man von einem multiplen Regressionsmodell, wenn

$$\mu_i \quad = \quad \beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_p \cdot x_{ip}$$

$i = 1, \ldots, n.$ unterstellt wird. Dabei bezeichnen $x_{ij}$ die Beobachtungen der Variable $X_j, j = 1, \ldots, p.$ Die unbekannten Parameter $\beta_0, \beta_1, \ldots, \beta_p$ heissen Regressionskoeffizienten.

Zusammengefasst bedeutet dies, dass eine Formel benötigt wird, die alle Einflussgrössen einbezieht um damit die Zielgrösse zu bestimmen. Auch hier wird nicht auf die Berechnung eingegangen, sondern nur auf die Ausgabe und die Interpretation. Da wir hier für jeden zu beschreibenden Punkt mehr als eine Variable haben können, wird die Formel einfach komplizierter, aber es bleibt ein Polynom. Das was Gross (ebd., S. 205) in der obigen Definition kompliziert beschreibt ist in Abbildung 8.5 dargestellt. Damit ist auch ersichtlich, dass eine multiple Regression eine Zusammensetzung mehrerer einfacher Regressionen ist. Auch hier bedeutet das $f_{(x)}$ die Formel um aus den Punkten $x_{11}, \ldots, x_{(ip)}$ zu den Punkten $y_1, \ldots, y_i$ zu kommen. Auch hier gilt, dass statt einer Geraden auch ein höherwertiges Polynom zum Einsatz kommen kann.

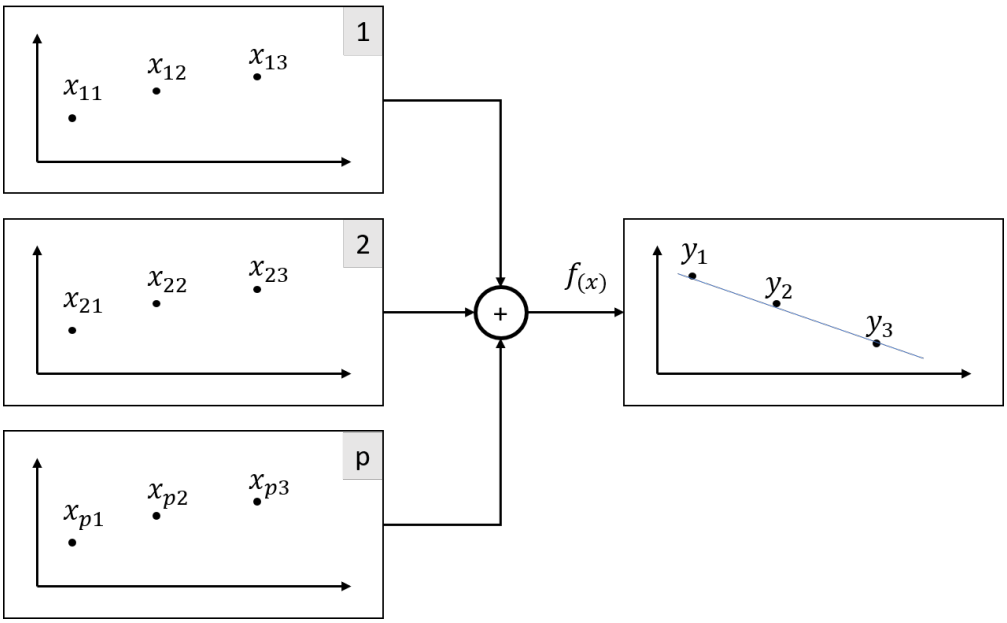


Abbildung 8.5.: Beispiel einer multiplen linearen Regression

### 8.4.2. Unterschiede zwischen der einfachen und der multiplen Regression

#### Signifikanz

Obwohl eine multiple Regression wie eine Zusammensetzung aus mehreren einfachen Regressionen betrachtet werden kann, gibt es doch einen Punkt der im Verständnis extrem wichtig ist. Es geht um die Gewichtung der verschiedenen Einflussgrössen. Beispiel: Wir möchten eine Aussage über den Bandbreitenverlauf eines Netzwerkgerätes[1] durchführen. Als Einflussgrössen stehen zur Verfügung:

- Datenmenge der Benutzer
- Datenmenge für das Management des Router

und wir möchten nun eine Aussage über die Abhängigkeit der Datenmenge am Uplink-Port machen. Wenn nun die Summe der Benutzer mehrere Terabyte pro Monat sind und das Management des Router nur wenige Megabyte sind, dann können wir zwar eine multiple Regression anwenden, der Einfluss des Managements wird aber sehr klein sein. Deshalb wird bei der Berechnung der Koeffizienten immer auch der Signifikanzwert berechnet. Je näher dieser bei 0 ist, desto signifikanter ist er, hat also einen wesentlichen Einfluss auf das Resultat. In unserem Beispiel mit dem Netzwerkgerät ist schon durch das Gefühl anzunehmen, dass die Datenmenge der Benutzer eine grosse Signifikanz hat, die Datenmenge des Managements eher klein. Grundsätzlich kann gesagt werden, wenn der

[1] z.B. Switch, Router, Firewall etc.

Signifikanzwert kleiner 0.05 ist, dann ist der Koeffizient (also diese unabhängige Variable) signifikant.

### Bestimmtheitsmass

Für die einfache Regression wird das Bestimmtheitsmass mit der Formel 8.2 berechnet. Gut erkennbar ist, dass diese Formel auch für eine multiple Regression angewendet werden kann. Allerdings gibt es dabei ein Problem. Je mehr Einflussfaktoren vorhanden sind, desto grösser wird hier $R^2$ auch ohne, dass tatsächlicher Qualitätsverbesserung. Deshalb sollte in einer multiplen Regression mit einem adjustierten bzw. korrigiertem Bestimmtheitsmass gerechnet werden. Gestartet wird mit $R^2$ und anschliessend eine Korrektur folgt. Nach Gross (2010, S. 208): „Fügt man in einem bestehenden multiplen Regressionssatz weitere erklärende Einflussgrössen hinzu, so kann $R^2$ nicht kleiner als zuvor werden, in der Regel wird $R^2$ sogar grösser."Mit der Formel 8.5 und dabei $k$ für die Anzahl Koeffizienten, $p$ für die Anzahl Einflussfaktoren (unabhängige Variablen) und $n$ die Anzahl Beobachtungen stehen, wird der Wert entsprechend korrigiert. In der Literatur gehen die Meinungen auseinander, ob nun mit $R^2$ oder $\overline{R}^2$ gerechnet werden soll. Da meistens mit Werkzeugen gerechnet wird und diese jeweils beide Werte ausgeben, wird eine weitere Diskussion in diesem Bereich unnötig.

$$\begin{aligned} \overline{R}^2 &= 1 - (1 - R^2) \cdot \frac{n - (k - p)}{n - k} \\ &= 1 - (1 - R^2) \cdot \frac{n - 1}{n - p - 1} \end{aligned} \tag{8.5}$$

# 9. R in der Cloud

Es gibt verschiedene Möglichkeiten um R als Server Applikation oder in der Cloud zu betreiben. Hier werden die bekanntesten Methoden kurz beschrieben, ohne Anspruch auf Vollständigkeit zu erhalten. Bei der Wahl optimalen Lösung sind die Bedürfnisse zu beachten. Die untenstehenden Werkzeuge wurden vom Author getestet und insbesondere R Plumber mit einer PHP Applikation in einer Cloud Foundry Umgebung.

Diverse dieser Lösungen haben kein eigenes Berechtigungssystem, aus Sicht der Security muss dazu immer in der Dokumentation nachgeschaut werden, welche Möglichkeiten zur Verfügung stehen.

## 9.1. R Studio Server

R Studio Server gibt es in verschiedenen Lizenzierungen, wobei alle ein Ziel verfolgen: Den Betrieb und die Nutzung der R Applikationen in einem Webbrowser. Die Ansicht wurde bereits am Anfang in der Abbildung 3.1 gezeigt. R Studio Server läuft auf Linux und für viele Distributionen sind bereits fertige Pakete vorhanden, die einfach zu installieren sind. Die Authentisierung der Benutzer erfolgt über in Linux bereits vorhandenen Mechanismen. Die Nutzung selber verlangt nur einen Webbrowser und ermöglicht somit die Arbeit unabhängig von bestimmten Betriebssystemen. Webseite und Downloadmöglichkeiten: `https://rstudio.com/products/rstudio/#rstudio-server`

Unter Umständen ist R Studio Server gut geeignet um Applikationen mit R Plumber aufzubauen (Siehe 9.2), denn so kann R Plumber einfach über eine Weboberfläche auf dem Server in Betrieb genommen werden.

## 9.2. R Plumber

Speziell für das zur Verfügung stellen von R Funktionalitäten bietet sich R Plumber `https://www.rplumber.io/` an. Plumber bietet eine Standard REST API an, was speziell für Machine to Machine (M2M) geeignet ist.

Mit den richtigen Paketen kann R Plumber zu Testzwecken direkt in R Studio gestartet werden. Auf der Webseite sind fertige Beispiele vorhanden.

Wenn keine weiteren Optionen für die Serialisierung angegeben werden, dann arbeitet R Plumber mit Standard JavaScript Object Notation (JSON) In- und Outputs. Allfällige Graphik Outputs werden in den gängigen Bildformaten zurückgegeben.

Ganz wichtig: R Plumber muss innerhalb von R gestartet werden.

### 9.2.1. Plumber als Docker

R Plumber steht als fertiges Docker Image zur Verfügung und es ist bestens dokumentiert, wie damit gearbeitet werden kann.

### 9.2.2. Security

Wichtig: R Plumber stellt keine Benutzer Authentisierung zur Verfügung. Daher sollte R Plumber nie direkt aus dem Internet erreichbar installiert werden. Es wird empfohlen R Plumber hinter einem Reverse Proxy zu starten und die Authentisierung über den Reverse Proxy Server durchzuführen. Auch dazu ist eine ausführliche Dokumentation auf der Webseite verfügbar.

## 9.3. Shiny from RStudio

Speziell für Webapplikationen die mit Graphiken aus R angereicht werden sollen, steht als Erweiterung für das R Studio Shiny zur Verfügung. `https://shiny.rstudio.com/`

Shiny benötigt zusätzliche Bibliotheken, wobei als Hilfestellung zur Benutzung von Shiny sowohl Text als auch längere Einführungsvideo zur Verfügung stehen.

## 9.4. Rserve

Über eine Standard TCP/IP Verbindung kann Rserver genutzt werden. `https://www.rforge.net/Rserve/`

Für Rserve stehen verschiedene Clients zur Verfügung, die alle auf der Webseite über einen Link abgerufen werden können. Damit werden die wichtigsten Programmiersprachen wie Java, C++, Phyton, .NET/CLI, C# und Ruby unterstützt. Vermutlich findet man im Internet noch weitere Clients, falls nicht, Rserve kann direkt über TCP/IP und einen zu definierenden Port angesprochen werden. Getestet kann Rserve direkt über Telnet.

Leider findet der Author dieses Manuskriptes die Dokumentation eher schwach ausgeprägt und je nach Interpretation der Versionenliste scheint die Weiterentwicklung eher gemächlich abzulaufen. Aufgrund des einfachen Ansprechens des Servers dürfte Rserve aber insbesondere für kleines Projekte die optimale Wahl sein.

# Teil II.

# Fallbeispiele

# 10. Beispiel mit linearer Regression

## 10.1. Speicherbedarf (Storage) für einen Fileserver

Die IT-Leitung beantragt für das nächste Jahr den SSD-Speicher um 50% zu erhöhen. Argumentiert wird mit 20% mehr Benutzern und einem erhöhten Volumen von 20% pro Benutzer, Rest ist Reserve. Die Aufgabe der Revision besteht aus zwei Aufgaben:

1. Schauen ob der in der Vergangenheit angegebene Speicherbedarf in etwa realistisch ist

2. Die Prognose nachvollziehbar ist

| **Jahr** | **Benutzer** | **Storagebedarf (GB)** |
|---:|---:|---:|
| 2015 | 778 | 2957 |
| 2016 | 843 | 3288 |
| 2017 | 870 | 3654 |
| 2018 | 937 | 4685 |
| 2019 | 980 | 5684 |
| 2020 | 1023 | 6343 |
| **Prognosen** | | |
| 2021 | 1250 | 7800 |

Tabelle 10.1.: Nutzungsdaten Speicher auf Fileserver.

### 10.1.1. x-y-Diagramm

Als Erstes wird eine Graphik mit den Werten aus der Vergangenheit und zusätzlichen Punkten für die Prognose erzeugt. Dazu wird in einem x-y Diagramm vergleichen, ob eine Korrelation zwischen Anzahl Benutzer und Storage besteht. Der Faktor Zeit wird in dieser Betrachtung bewusst weggelassen, da zuerst einmal nur das Verhältnis zwischen Anzahl Benutzer und Speicherbedarf interessiert. Eine andere Möglichkeit wäre das Verhältnis direkt auszurechnen und dieses auf der Zeitachse abzubilden. Wichtig ist in dieser Betrachtung eine Normalisierung der Anzahl Benutzer, da es primär um den Speicherbedarf pro Benutzer geht. Denn wenn bereits jetzt der Faktor Zeit mitberücksichtigt wird, könnte dies zu einer Falschinterpretation des Resultates führen, denn es könnte ja sein, dass die Anzahl Benutzer überhaupt nicht linear ansteigt oder in einem Jahr kurzfristig der doppelte Speicherbedarf während einer Systemmigration nötig war und dann aber auch während der Migration die theoretische Benutzerzahl grösser war, weil die Benutzer auf beiden System zugreifen mussten.

**Schritt 1 x-y-Diagramm zwischen Anzahl Benutzern und Speicherbedarf**

```
id <- c(0, 1, 2, 3, 4, 5)
jahre <- c(2015, 2016, 2017, 2018, 2019, 2020)
benutzer <- c(778, 843, 870, 937, 980, 1023)
storage  <- c(2957, 3288, 3654, 4685, 5684, 6343)

#Da der rote Punkte ausserhalb der Werte der Vergangenheit ist,
#muss mit xlim, und ylim die Graphik so angepasst werden,
#dass der rote Punkte angezeigt wird.
plot(benutzer, storage, main = "Vergleich Benutzer und Storage",
        xlim=c(700, 1300), ylim=c(3000,8000))
#Prognosepunkt in rot einzeichnen.
points(1250,7800,col="red", pch = 19)
```

In Abbildung 10.1 ist das Resultat ersichtlich und es scheint eine Korrelation zwischen Anzahl Benutzern und Speicherbedarf zu geben. Dass es keine Gerade zu sein scheint ist dürfte dem Umstand geschuldet sein, dass mit der Zeit und der Anzahl Benutzer durch mehr Datenaustausch der Speicherbedarf pro Benutzer ansteigt. Der Prognosepunkt ist in roter Farbe hinzugefügt. Irgendwie passt der rote Punkt nicht auf die gedachte Linie der schwarzen Kreise. Somit deutet diese Abweichung schon einmal darauf hin, dass dies näher zu untersuchen ist.

x-y Diagramm

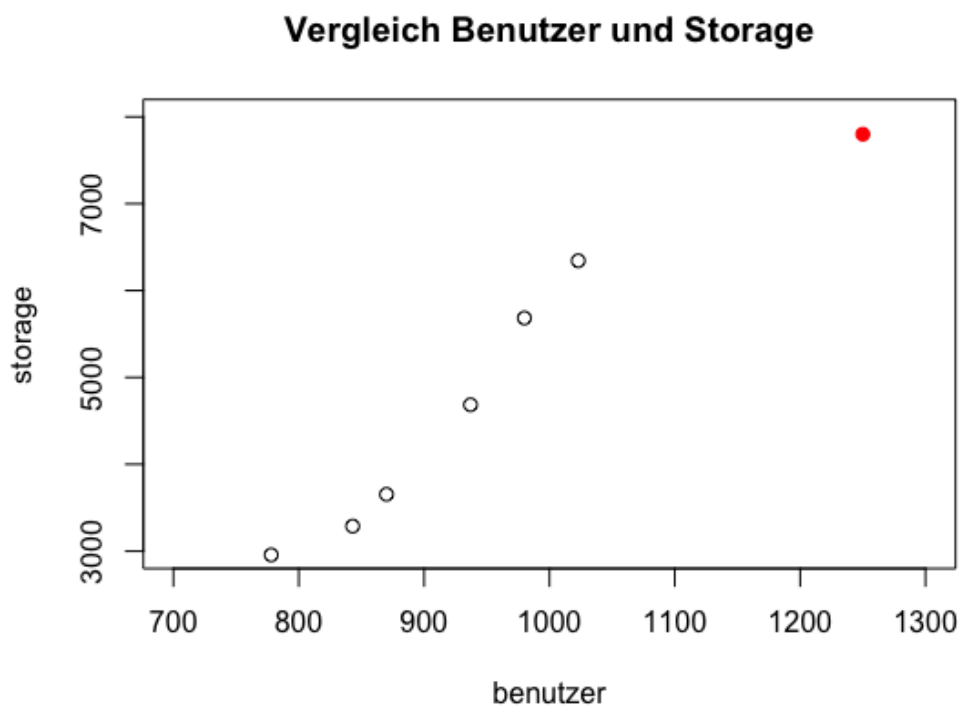


Abbildung 10.1.: x-y Diagramm zwischen Anzahl Benutzern und Speicherbedarf.

### 10.1.2. Verhältnis Speicher pro Benutzer über die Zeit

Im nächsten Schritt vergleichen wir den Verlauf des Verhältnisses Speicherbedarf pro Benutzer. Zusätzlich wird auch eine Trendformel berechnet. Die Interpretation dazu folgt weiter unten in einem weiteren Beispiel.

**Verlauf Verhältnis Speicher pro Benutzer**

```
#Ratio anschauen
ratio = storage/benutzer

#Trendformel berechnen
lm.ratio = lm(ratio~id)
#Zusammenfassung zur Trendformelberechnung.
summary(lm.ratio)

#Trendpunkte berechnen für die Vergangenheit
pred.ratio.vergangenheit<- predict(lm.ratio, newdata = data.frame(id))
pred.ratio.vergangenheit
#Trendpunkte berechnen für die Zukunft
pred.ratio.2021 <- predict(lm.ratio, newdata=data.frame(id=6))
pred.ratio.2021

#Graphik anzeigen
plot(jahre, ratio,
        main = "Verlauf des Verhältnisses Speicher pro Benutzer",
        xlim=c(2015, 2021),
        ylim=c(min(pred.ratio.vergangenheit), max(pred.ratio.2021)))

#Prognosepunkt in rot einzeichnen
points(2021, 7800/1250,col="red", pch = 19)

#Mit einer blauen Linie den Trend zeichnen
lines(jahre, pred.ratio.vergangenheit, col="blue")
#und wieder als blauer Punkt den berechneten Trend
points(2021, pred.ratio.2021, col="blue", pch = 19)
```

In den Jahren 2015 bis 2020 ist der Speicherbedarf kontinuierlich gestiegen. Entgegen der Aussage der IT-Leitung scheint das Verhältnis gemäss Trendextrapolation für das Jahr 2021 etwa gleich, wie es für das Jahre 2020 (aktuelles Jahr) zu erwarten ist.

**Verhältnis-Zeit Diagramm**

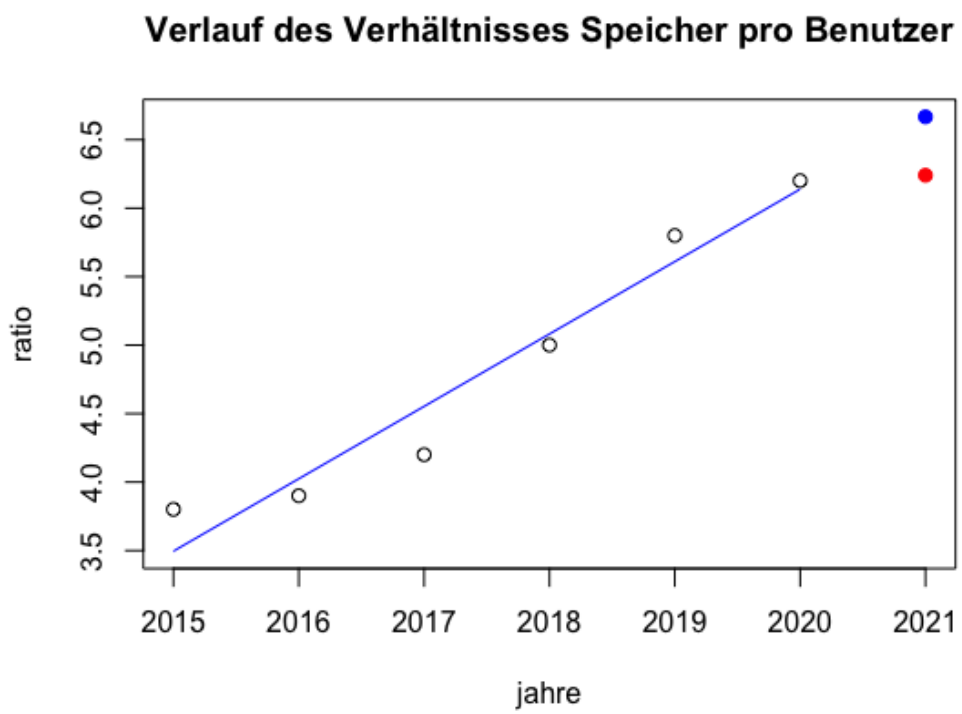


Abbildung 10.2.: Verlauf des Verhältnisses zwischen Speicher pro Benutzer

### 10.1.3. Trend Anzahl Benutzer

Die Quelle der Prognose für die Anzahl Benutzer ist unbekannt, sie kann von der IT-Leitung stammen, aber auch vom HR falls es sich um interne Benutzer handelt oder vom Produktmanagement, falls es sich um externe Kunden geht. Trotzdem gibt eine Trendbetrachtung der Anzahl Benutzer aus der Vergangenheit und einer Extrapolation in die Zukunft der Prognose einen Vergleich für das Jahr 2021 anzustellen. Dazu nehmen wir die R-Funktion "lm"für Linear Model. Mit der Funktion Summary erhalten wir die notwendigen Angaben.

**Lineare Modell Anzahl Benutzer**

```
#Lineares Modell für Anzahl Benutzer über die Zeit berechnen.
lm.benutzer <- summary(lm(benutzer~id))
lm.benutzer
```

In den Kapiteln der Polynome (Kapitel 5 und Kapitel 6) und Regression (Kapitel 8) wurde primär auf folgende Elemente in einer Regression eingegangen.

- Koeffizienten
- $r^2$
- Signifikanz

Die lm-Ausgabe aus von R enthält alle Informationen. Unter den Koeffizienten sind wie erwartet zwei Zeilen: Der Intercept (Konstanter Anteil) und die id. Der Intercept ist der Achsenabschnitt, also dem y-Wert wenn x=0 ist. Da in dieser linearen Regression die x-Achse der id entspricht, wird auch die id ausgegeben.

**Resulate Lineares Modell Anzahl Benutzer**

```
Call:
lm(formula = benutzer ~ id)

Residuals:
      1       2       3       4       5       6
 -5.524  10.819 -10.838   7.505   1.848  -3.810

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  783.524      6.666   117.5 3.14e-08 ***
id            48.657      2.202    22.1 2.48e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9.21 on 4 degrees of freedom
Multiple R-squared:  0.9919,        Adjusted R-squared:  0.9898
F-statistic: 488.5 on 1 and 4 DF,  p-value: 2.481e-05
```

Somit wird die Formel 10.1 gebildet, aus der Trendwerte für die Zukunft berechnet werden.

$$\text{Anz. Benutzer} \quad = \quad 48.657 \cdot id + 783.524 \tag{10.1}$$

Sowohl für den Intercept als auch die id wurden Signifikanzwerte nahe bei 0 errechnet und zusammen mit dem $r^2$ bei fast 1 scheint das Modell sehr gut zu sein.

Glücklicherweise muss weder die Rückrechnung noch die Extrapolation nicht manuell gerechnet werden, mit *predict* stellt R die notwendige Funktion bereits zur Verfügung.

**Lineares Modell anwenden und Extrapolation berechnen**

```
#Intercept in Variable schrieben, damit dieser anschliessend in
#der Graphik als ylim obere Grenze benutzt werden kann
intercept <- coefficients(lm.benutzer)["(Intercept)"]

#Aus dem lm Modell für die gebenen x-Werte (id) die y-Werte (benutzer)
#berechnen, um diese mit den unsprünglichen Werten
#vergleichen zu können.
pred.vergangenheit <- predict(lm.benutzer, newdata = data.frame(id))
pred.vergangenheit
#Und nun den Trend rechnen.
pred.2021 <- predict(lm.benutzer, newdata=data.frame(id=6))
pred.2021

#In Graphik die effekten Werte einzeichnen
plot (jahre, benutzer, xlim=c(2015,2021), ylim=c(intercept, 1250),
main="Trend Benutzer")
#Mit einer blauen Linie den Trend zeichnen
lines(jahre, pred.vergangenheit, col="blue")
#und als roter Punkte die Prognose der IT
points(2021, 1250, col="red", pch = 19)

#und wieder als blauer Punkt den berechneten Trend
points(2021, pred.2021, col="blue", pch = 19)
```

Dargestellt ist das Resultat in der Abbildung 10.3 und zeigt deutlich, dass der rote Punkt, also die Prognose der IT-Abteilung sehr hoch gegenüber dem Trend ist.

Trend Benutzer

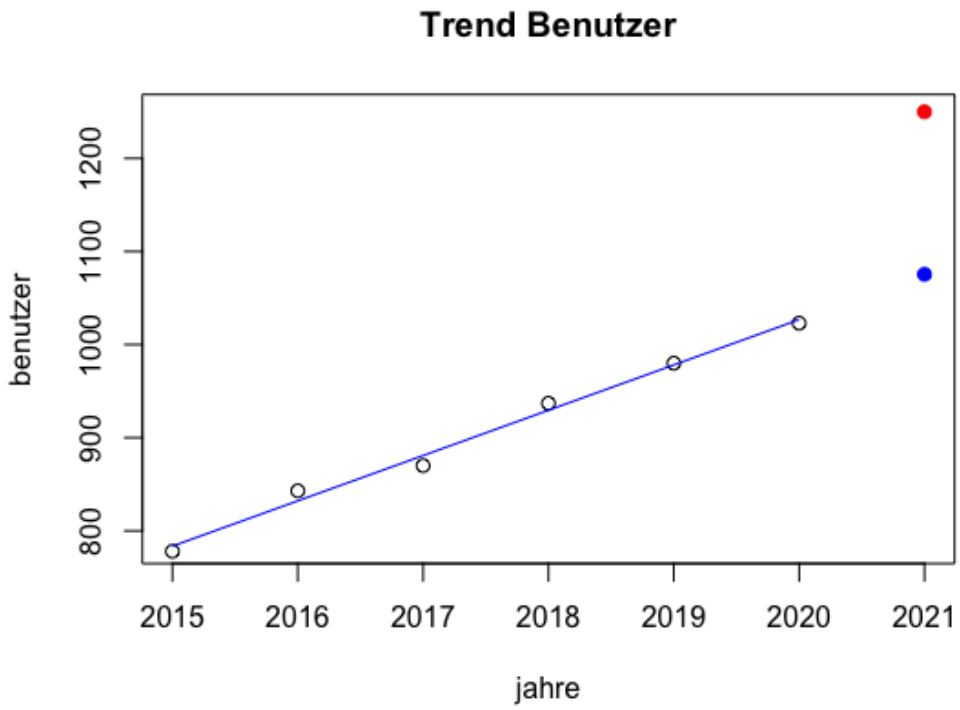


Abbildung 10.3.: Trend und Extrapolation der Anzahl Benutzer

### 10.1.4. Mögliche Aussagen zu Speicherbedarf und Anzahl Benutzer aufgrund der Auswertung

#### Interpretation Speicher pro Benutzer

Obwohl die IT-Abteilung mit vermehrtem Speicherbedarf pro Benutzer argumentiert, kann dies gemäss der Trendberechnung nicht nachvollzogen werden. Dies ist sowohl im x-y-Diagramm (siehe Abbildung 10.1) noch im direkten Verlauf des Verhältnisses gemäss Abbildung 10.2 ersichtlich. Der Verlauf ist so offensichtlich, dass die Trendberechnung auch hätte eingespart werden. Somit könnte eine Empfehlung sein, dass die IT-Abteilung ihre Argumente besser ausführen oder gegebenenfalls die Prognose anpassen muss.

#### Interpretation Anzahl Benutzer

Der Prognose der Anzahl Benutzer kann aufgrund des zeitlichen Verlaufs aus der Vergangenheit nicht nachvollzogen werden. Jedenfalls liegt die Prognose mit dem roten Punkt

in der Abbildung 10.3 deutlich über dem extrapolierten Trend. Je nach Situation ist aber die Anzahl Benutzer nur beschränkt aus der Vergangenheit berechenbar. Mögliche Gründe für den plötzlichen Anstieg könnte eine grössere Akquisition oder eine Fusion sein. Im Falle eines IT-Dienstleisters einen oder mehrere zusätzliche Kunden. Auf jeden Fall dürfte es sinnvoll sein, dem nachzugehen. Vermutlich dürfte hier die IT-Abteilung kaum der richtige Ansprechpartner sein, im Falle einer Akquisition oder Fusion wäre es wohl die Geschäftsleitung oder bei einem IT-Dienstleister der Verkauf bzw. das Produktmanagement.

# 11. Beispiel mit multipler linearer Regression

Meist haben Systeme mehr als einen Parameter, die Einflüsse auf das Resultat haben, weshalb dazu ein Beispiel konstruiert wird.

## 11.1. Gebrauchtwagenpreise

Ein Auto-Occasionshändler stellt mit Hilfe einer Software sicher, dass seine Verkaufspreisschätzungen richtig sind um einerseits Ladenhüter zu verhindern und andererseits aus Unachtsamkeit falsche Preise anzuschreiben, also ein typisches IKS-System.

Leider kennt niemand die Formeln hinter dieser Software und auch der Sourcecode ist unauffindbar. Da die Applikation auf eine neue Plattform verschoben wurde, muss die Revision nun überprüfen, ob noch alles richtig läuft (Regressionstest). Aus Zeiten des Systemtests stehen die Referenzdaten für die untere Mittelklasse gemäss Tabelle 11.1 zur Verfügung. Es handelt sich also um ein typisches Beispiel von Substantive Testing. Siehe Abbildung 11.1.

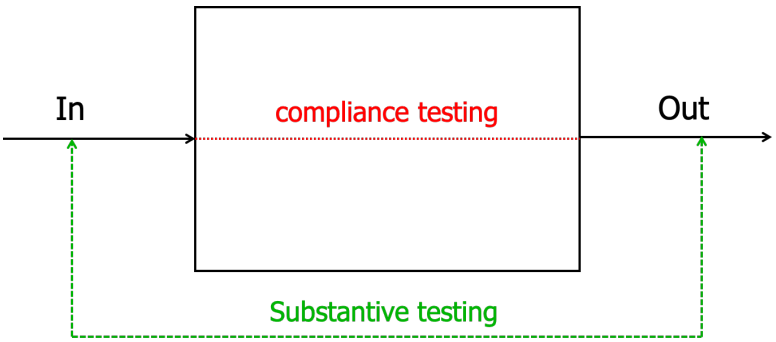


Abbildung 11.1.: Substantive- und Compliance-Testing

| **Alter** | **km** | **PS** | **Preis** |
|---|---|---|---|
| 13 | 123000 | 143 | 7900 |
| 8 | 107852 | 136 | 20390 |
| 0 | 10 | 150 | 41350 |
| 5 | 160000 | 326 | 19999 |
| 5 | 48000 | 180 | 21900 |
| 2 | 23000 | 272 | 44900 |
| 16 | 76000 | 140 | 4900 |
| 2 | 19900 | 184 | 33500 |

Tabelle 11.1.: Referenzautos

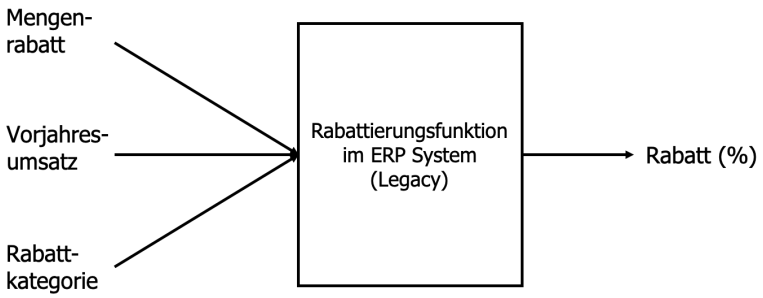


Abbildung 11.2.: In- und Output des Legacy-Rabattsystems

Die Schritte zur Bearbeitung in R sind wie folgt:

**Lineares Modell aus Alter, km, PS und Preis erstellen**

```
#Vektoren für Alter, km, ps und Preis erstellen
alter <- c(13,8, 0, 5, 5, 2, 16, 2)
km <- c(123000, 107852, 10, 160000, 48000, 23000, 76000, 19900)
ps <- c(143, 136, 150, 326, 180, 272, 140, 184)
preis <- c(7900, 20390, 41350, 19999, 21900, 44900, 4900, 33500)
#Vektoren in einen data.frame überführen
df.occassion = data.frame(alter, km, ps, preis)
#schauen, ob alles wie gewünscht in der Tabelle ist
head(df.occassion)

#lineares Modell erstellen
lm.auto <- lm(preis~., data=df.occassion)
#und Zusammenfassung ausgeben
summary(lm.auto)
```

Daraus resultiert die *lm* Zusammenfassung und gut ersichtlich ist, dass die Anzahl km und die Leistung in PS nur einen marginalen Einfluss auf den Preis haben. relevant ist primär das Alter, erkennbar am einen *. Das Modell selber ist nicht schlecht, es passt mit einem $r^2$ von 0.866 ziemlich gut.

**Resultat aus dem linearen Modell**

```
Call:
lm(formula = preis ~ ., data = df.occassion)

Residuals:
      1       2       3       4       5       6       7       8
 1489.2  4357.2   788.1 -2111.1 -6533.9  5948.3 -1072.4 -2865.4

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.563e+04  8.525e+03   4.179   0.0139 *
alter       -1.670e+03  5.884e+02  -2.837   0.0470 *
km          -9.934e-02  5.506e-02  -1.804   0.1455
ps           3.289e+01  4.034e+01   0.815   0.4607
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5332 on 4 degrees of freedom
Multiple R-squared:  0.9234,        Adjusted R-squared:  0.866
F-statistic: 16.07 on 3 and 4 DF,  p-value: 0.01072
```

Mit der Ausgabe aus *lm* kann die Formel 11.1 gebildet werden.

$$\begin{aligned} \text{Preis} \quad = \quad & \text{alter} \cdot -1670 \\ & +\text{km} \cdot -0.09934 \\ & +\text{ps} \cdot 32.89 \\ & +35630 \end{aligned} \tag{11.1}$$

Auf den ersten Blick ist erkennbar, dass ein neues Auto mit 0 PS (gibt es natürlich nicht), 0 km einen Preis von 35630 Franken hätte. Das dürfte für die untere Mittelklasse etwa passen. Für jedes zusätzliche PS würde gemäss dieser Regressionsrechnung ein Preis von 32.89 fällig. Setzen wir eine realistische Leistung von 130 PS ein, dann resultiert ein Preis von CHF 39’905.70. Auch das scheint in etwa zu passen.

# 12. Beispiel Grenzkostenberechnung

In der Betriebswirtschaft ist die Grenzkostenberechnung wichtig, wenn es um die Preisgestaltung geht. Denn nur wenn der Umsatz pro Stück grösser als die Grenzkosten ist, haben wir einen positiven Deckungsbeitrag.

## 12.1. Wie setzen sich die Kosten zusammen

Insbesondere in der Produktion von Gütern, aber auch im Dienstleistungssektor sind die Kosten der Erzeugnisse nicht nur von der Produktionsmenge abhängig, sondern es fallen auch Fixkosten an, die nur begrenzt von der produzierten Menge abhängig sind. Beispiel sind: Bürokosten, Abschreibungen auf Produktionsmaschinen etc. In der Grenzkostenberechnung geht es nun darum, auszurechnen was ein zusätzliches produziertes Stück kostet, unter der Annahme, dass noch freie Produktionskapazität vorhanden ist und durch dieses zusätzliche Stück keine neuen (Sprung-) Fixkosten entstehen. Beispiel: Die Produktion ist zu 80% ausgelastet und produziert pro Tag 1000 Stück. In der Grenzkostenbetrachtung wird jetzt berechnet, was es zusätzlich kostet, wenn nun 1001 Stück produziert werden. Somit entsprechen die Grenzkosten den variablen Kosten für die betrachtete Zusatzmenge.

### 12.1.1. Lineares Modell

Als Formel (siehe 12.1 ) setzen sich die Kosten (K) aus den Variablen Kosten (VK), der Anzahl produzierte Einheiten (x) und den Fixkosten (FK) zusammen.

$$K_{(x)} = VK \cdot x + FK \tag{12.1}$$

Die Grenzkosten entsprechen der ersten Ableitung.

$$K'_{(x)} = VK \tag{12.2}$$

### 12.1.2. Generisches Modell

Ist der Kostenverlauf zwar stetig, aber nicht linear, dann sieht es wie folgt aus:

$$K_{(x)} = f_{(x)} + FK \tag{12.3}$$

Die Grenzkosten werden auch hier über die erste Ableitung berechnet.

$$K'_{(x)} = f'_{(x)} \tag{12.4}$$

## 12.2. Problemstellung

Eine kleine Bäckerei weiss, dass ihre Gipfeliproduktion auf etwa 100 Gipfeli pro Tag ausgerichtet ist (neben der Produktion aller anderer Produkte), produziert aber inzwischen mehr als 100 Gipfeli pro Tag. Aus Sicht der Revision ist nun zu überprüfen, bis zu welcher Gipfelianzahl pro Tag die Produktion überhaupt kostendeckend ist, um beurteilen zu können, ob eine Investition für grössere Gipfelimengen sinnvoll erscheint.

### 12.2.1. Stück versus Kosten

Die Geschäftsführerin hat einmal aufgrund der Produktionszeiten, Materialaufwände etc. folgende Tabelle erstellt.

| **#Gipfeli** | **Kosten (CHF)** |
|---|---|
| 10 | 5 |
| 20 | 7.5 |
| 30 | 8.5 |
| 40 | 9 |
| 50 | 9.35 |
| 60 | 9.7 |
| 70 | 10 |
| 80 | 10.7 |
| 90 | 12 |
| 100 | 13 |
| 110 | 15 |
| 120 | 20 |

Tabelle 12.1.: Produktionskosten Gipfeli

Als ersten Schritt müssen die Zahlen in R übernommen werden.

**Dataframe für Anzahl Gipfeli und Kosten**

```
#Grenzkosten Gipfeli
AnzahlGipfeli <- c(10,20,30,40,50,60,70,80,90,100,110,120)
Kosten <- c(5, 7.5,8.5,9,9.35,9.7, 10, 10.7,12,13,15,20)
#DataFrame erstellen
df.GipfeliKosten <- data.frame(AnzahlGipfeli, Kosten)
```

Als Nächstes wird ein x-y Plot erzeugt. Dank der Situation, dass bereits ein Dataframe vorhanden ist, werden die Vectornamen direkt als den Achsen zugeordnet.

**Verlauf Gipfeli versus Kosten**

```
#Verlauf Anzahl Gipfeli und Kosten als Plot
plot(df.GipfeliKosten)
```

und erhalten folgende Graphik, in der bereits gut sichtbar ist, dass die Kosten ab ca. 100 Stück Gipfeli schnell ansteigen. Gemäss Aussage der Bäckerei ist das darauf zurückzuführen, dass die Stellflächen zu klein sind, die Kapazität des Backofens an der Grenze ist und deshalb ab ca. 100 Stück viel zusätzliche Arbeitsschritte entstehen.

**Anz. Gipfeli vs. Kosten**

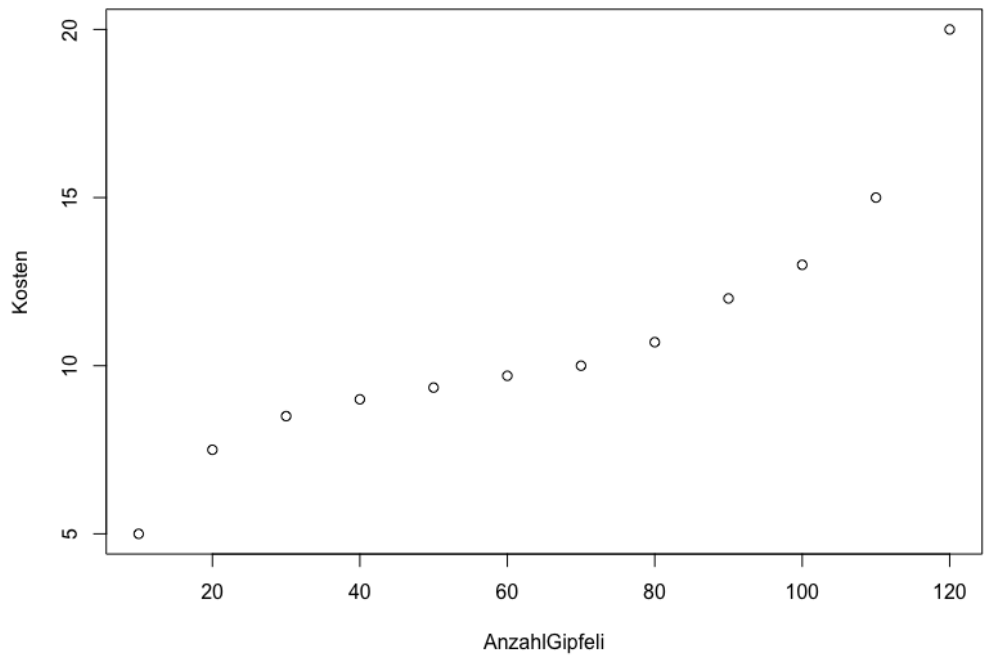


Abbildung 12.1.: Verlauf der Kosten in Relation zur Anzahl produzierter Gipfeli

## 12.3. Vorgehen zur Berechnung der Grenzkosten

### 12.3.1. Kostenfunktion erhalten

Oben wurde bereits berichtet, dass die Grenzkostenfunktion die erste Ableitung der Kostenfunktion ist. Somit muss zuerst die Kostenfunktion bekannt sein, bevor die Grenzkostenfunktion berechenbar ist. Mit Hilfe der R Funktion *lm* und dem bereits in R abgelegten

Dataframe, gilt es nun diese Funktion zu erhalten.

In den bisherigen Beispielen war die Art der Funktion bereits irgendwie vorgegeben, während in diesem Beispiel zuerst die Überlegung nötig ist, was der Kostenverlauf überhaupt für eine Funktion sein könnte. Aufgrund, dass es zwei zusammengesetzte Kurven sein könnten, die beide nicht linear sind, liegt eine Funktion $f_{(x)} = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$ nahe. Nun muss probiert werden, ob diese Annahme passen könnte.

**Lineares Gipfelimodell mit einem Polynom 3. Grades**

```
#Lineares Modell für Polynom 3. Grades
lm.gipfeli <- lm(Kosten~AnzahlGipfeli + I(AnzahlGipfeli^2) +
        I(AnzahlGipfeli^3), data=df.GipfeliKosten)
summary(lm.gipfeli)
```

Und R gibt zurück:

**Resultat für Gipfelimodell**

```
        Call:
lm(formula = Kosten ~ AnzahlGipfeli + I(AnzahlGipfeli^2) +
        I(AnzahlGipfeli^3),
    data = df.GipfeliKosten)

Residuals:
     Min       1Q   Median       3Q      Max
-0.79940 -0.18687 -0.01106  0.23649  0.51587

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)         2.015e+00  6.790e-01   2.968   0.0179 *
AnzahlGipfeli       3.676e-01  4.342e-02   8.468 2.89e-05 ***
I(AnzahlGipfeli^2) -6.124e-03  7.604e-04  -8.053 4.16e-05 ***
I(AnzahlGipfeli^3)  3.565e-05  3.856e-06   9.244 1.52e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.415 on 8 degrees of freedom
Multiple R-squared:  0.9917,        Adjusted R-squared:  0.9886
F-statistic: 317.8 on 3 and 8 DF,  p-value: 1.176e-08
```

Die Annahme, dass ein Polynom 3. Grades gut passen könnte wird durch ein $R^2$ von nahezu 1 bestätigt. Jetzt kann mit dem gleichen Input der x-Achse und der erhaltenen Formel das Modell berechnet werden

**Modell aus Formel zurückrechnen**

```
#Modellzahlen aus Formel berechnen.
pred.lm.gipfeli <- predict(lm.gipfeli,
        newdata = data.frame(AnzahlGipfeli))

pred.lm.gipfeli
```

und daraus folgt das Resultat:

**Resultat beim Zurückrechnen**

```
       1        2        3        4        5
5.114835 7.203596 8.495305 9.203830 9.543040
       6        7        8        9       10
9.726807 9.968998 10.483483 11.484133 13.184815
      11       12
15.799401 19.541758
```

Dieses Resultat wird nun zum Vergleich wieder in der Graphik dargestellt, aber zur Unterscheidung mit einer roten Linie.

**Predict Linie**

```
#Verlauf Anzahl Gipfeli und Kosten als Plot
plot(df.GipfeliKosten)
lines(AnzahlGipfeli, pred.lm.gipfeli, col="red", pch = 19)
```

**Anz. Gipfeli vs. Kosten Predict**

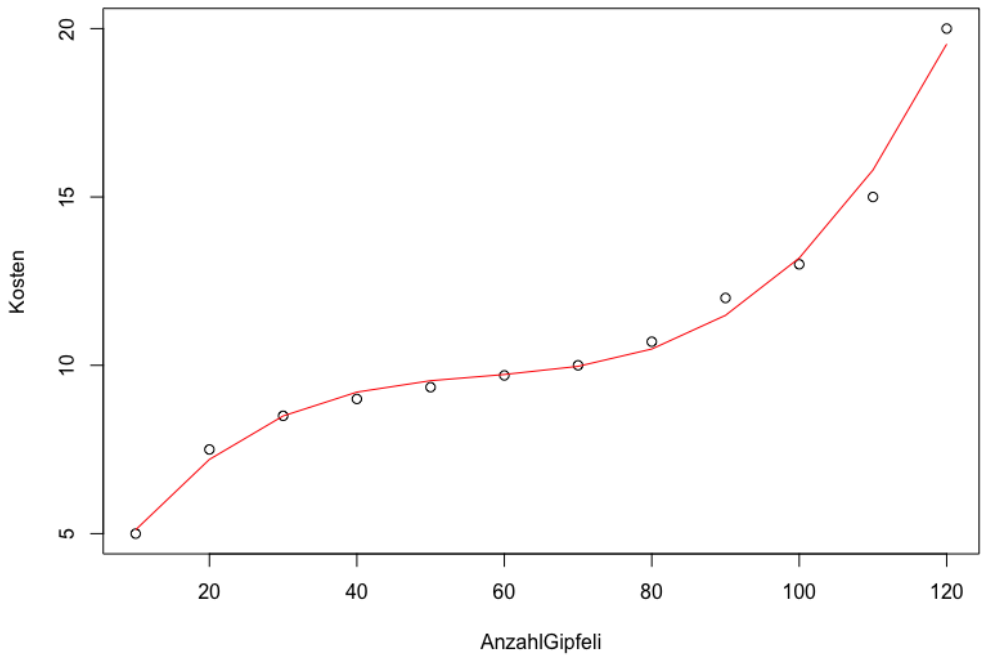


Abbildung 12.2.: Verlauf der Kosten in Relation zur Anzahl produzierter Gipfeli und Kurve auf Basis des Polynomes 3. Grades

### 12.3.2. Grenzkostenfunktion berechnen

Aus dem linearen Modell ergeben sich folgende Koeffizienten (siehe auch oben):

```
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)         2.015e+00  6.790e-01   2.968   0.0179 *
AnzahlGipfeli       3.676e-01  4.342e-02   8.468 2.89e-05 ***
I(AnzahlGipfeli^2) -6.124e-03  7.604e-04  -8.053 4.16e-05 ***
I(AnzahlGipfeli^3)  3.565e-05  3.856e-06   9.244 1.52e-05 ***
```

und somit kann die Formel zwischen Kosten und Anz. Gipfeli zusammengestellt werden. (Siehe 12.5)

$$\begin{aligned} \text{Kosten} &= 3.565 \cdot 10^{-5} \cdot \text{Anz. Gipfeli}^3 \\ &- 6.124 \cdot 10^{-3} \cdot \text{Anz. Gipfeli}^2 \\ &+ 3.676 \cdot 10^{-1} \cdot \text{Anz. Gipfeli} \\ &+ 2.015e+00 \end{aligned} \tag{12.5}$$

Bedauerlicherweise konnte der Autor keine R Bibliothek finden, mit der ein Output aus lm eine Ableitungsfunktion berechnen kann. Auf der anderen Seite ist die Ableitung von Polynomen relativ einfach. Immerhin stellt uns R die Ableitungsfunktion aus einer Formel zur Verfügung. Dazu muss die Ausgabe aus *lm* in eine Formel geschrieben werden.

**1. Ableitung Gipfelikosten**

```
#Ableitung des Resultates aus LM
# Expression or formula
f = expression(3.565e-05 * AnzahlGipfeli^3
-6.124e-03 * AnzahlGipfeli^2
+ 3.676e-01 * AnzahlGipfeli + 2.015e+00)

D.Gipfeli <- D(f, 'AnzahlGipfeli')

#Grenzkostenfunktion ausgeben
D.Gipfeli

#Ableitung mit den Anzahl Gipfeli berechnen
D.Gipfeli.Eval <- eval(D.Gipfeli)
D.Gipfeli.Eval

#Grenzkostenverlauf anzeigen
plot(AnzahlGipfeli, D.Gipfeli.Eval)
lines(AnzahlGipfeli, D.Gipfeli.Eval, col="blue", pch = 19)
```

Die erste Ableitung der Funktion 12.6 ergibt:

$$\begin{aligned} \text{Kosten} &= 3 \cdot 3.565 \cdot 10^{-5} \cdot \text{Anz. Gipfeli}^2 \\ &- 2 \cdot 6.124 \cdot 10^{-3} \cdot \text{Anz. Gipfeli} \\ &+ 3.676 \cdot 10^{-1} \end{aligned} \tag{12.6}$$

In der Abbildung 12.3 ist bestens sichtbar, dass die Kosten bis ca. 60 Gipfel sinken um anschliessend wieder anzusteigen. Das ist zwar auch direkt in der Kurve von Abbildung 12.2 ersichtlich, aber nicht in dieser Deutlichkeit! Zudem lassen sich erst in der Grenzkostenkurve ablesen, wie gross der variable Anteil der Kosten (pro Gipfeli) überhaupt ist.

**1. Ableitung**

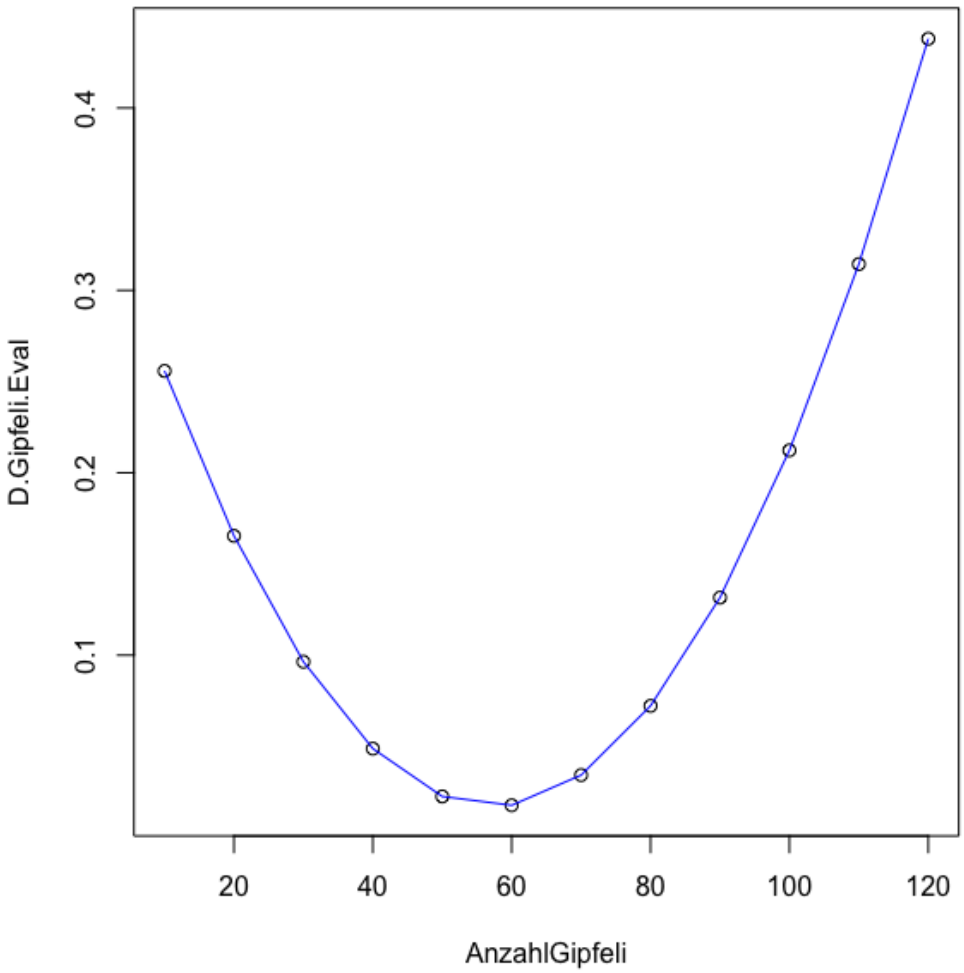


Abbildung 12.3.: Verlauf der Grenzkostenfunktion aus der ersten Ableitung der Kostenfunktion.

Wenn jetzt von einem Gipfelipreis von CHF 1 ausgegangen wird und die reinen Verkaufskosten 30 Rappen betragen, also für Ladenmiete, Löhne des Verkaufspersonals etc, dann darf die Anzahl an Gipfeli pro Tag soweit ansteigen, bis die variablen Kosten 70 Rappen erreichen. Darüber hinaus beginnt ein Verlustgeschäft.

**Grenzkostenfunktion 0 bis 140 Gipfeli**

```
#Das Ganze nochmals, aber dieses Mal mit einer max. Anzahl
#Gipfeli von 150.

AnzahlGipfeli <- c(0,10,20,30,40,50,60,70,80,90,100,110,120,130,140)
#Ableitung mit den Anzahl Gipfeli berechnen
D.Gipfeli.Eval <- eval(D.Gipfeli)
D.Gipfeli.Eval

#Grenzkostenverlauf anzeigen
plot(AnzahlGipfeli, D.Gipfeli.Eval)
lines(AnzahlGipfeli, D.Gipfeli.Eval, col="blue", pch = 19)
abline(h = 0.7, col = "red") #rote Linie bei 0.7
grid (NULL,NULL, lwd = 2)  #Gridlines
```

In der Abbildung 12.4 ist ersichtlich, dass bei knapp 140 Gipfeli pro Tag die variablen Kosten bei 70 Rappen liegen. Würde jetzt die Produktion weiter ausgeweitet, dann würde der Deckungsbeitrag negativ und somit zu einem Verlustgeschäft.

**Grenzkosten bis 140 Gipfeli**

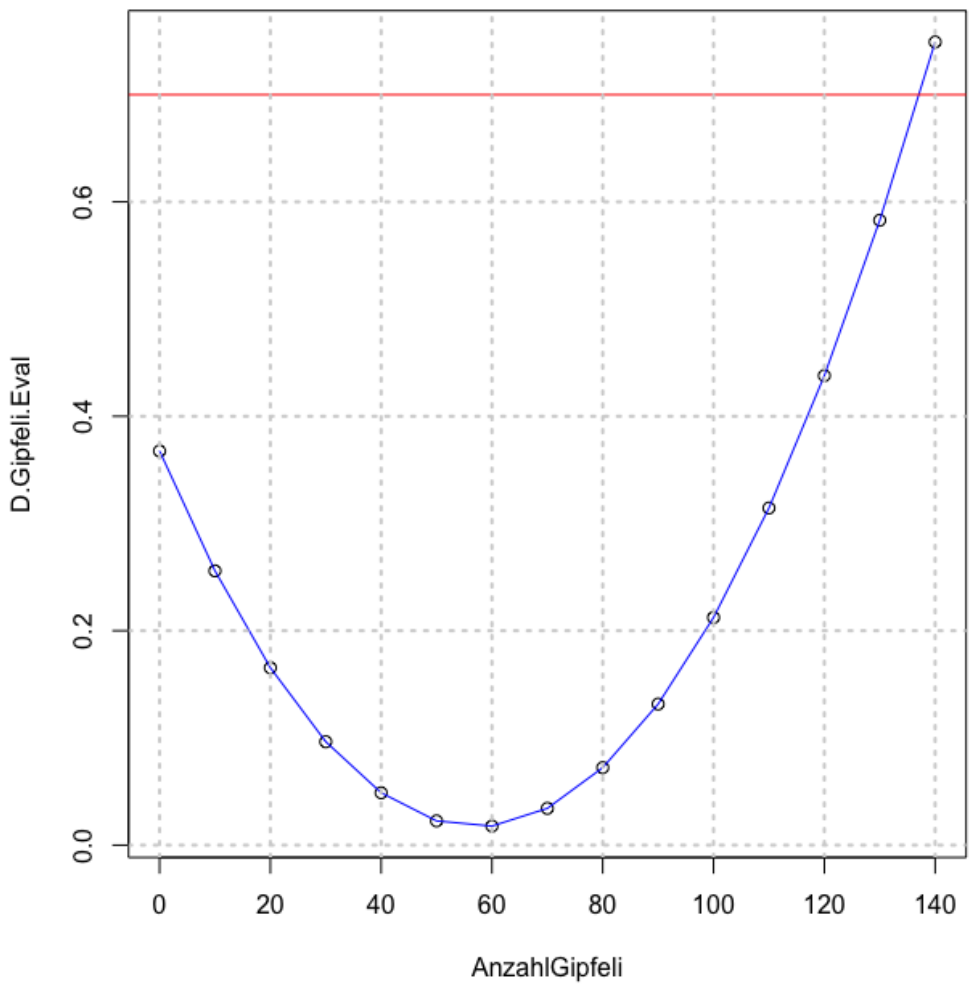


Abbildung 12.4.: Rot der max. Produktionspreis = Verkaufspreis minus Verkaufskosten.

Spannend in dieser Betrachtung ist, dass die Produktion von 1. Gipfel einen höheren Deckungsbeitrag ergibt, als wenn 120 Stück produziert werden. Dies deutet darauf hin, dass die Bäckerei eher wie Manufaktur als eine Fabrik arbeitet.

### 12.3.3. Ab wann Investition?

Zum Schluss muss die Frage aus dem Kapitelanfang beantwortet werden. Wie bereits oben beschrieben endet gemäss Abbildung 12.4 die Rentabilität. Somit sollte eine Produktionsumstellung oder Investition ins Auge gefasst werden, wenn regelmässig mehr als 140 Gipfeli pro Tag produziert werden.

# A. Anhang

## A.1. R File mit allen Beispielen

Bitte beachten, es handelt sich hier um ein R-Script. Die Ausgaben erfolgen bei der Nutzung von R-Studio im Ausgabefenster, weshalb diese in dieser Darstellung nicht sichtbar sind.

```
#Einfache Summe bilden für 3+8
3+8

#Vektoren anlegen
namen <- c('Peter','Fritz','Daniel')
hobby <- c('Rennradfahren', 'Schwimmen', 'Klettern')
alter <- c(51, 46, 53)

#Data Frame aus Vektoren bilden
df.freunde <- data.frame(freunde, hobby, alter)

#Struktur des Data Frames ausgeben
str(df.freunde)

#Zwei Vektoren für x und y-Achsen bilden
x <- c(1,2,3,4)
y <- c(1,2,9,16)
#und Graphik anzeigen
plot(x,y)

#Polynome in R darstellen
#Squenz von 0 bis 100 in Schritten von 1 erstellen.
x <- seq(0,100,1)

#y für jedes x berechnen
y <- 0.2 * x^2 - 0.5 * x + 10

#und in Graphik darstellen
plot(x,y)


#Mit Optionen gestalten
plot(x,y,
     type="l",
     main="Plot Funktion",
     xlab="x",
     ylab="y= 0.2 * x^2 - 0.5 * x + 10",
     col="blue"
```

```
    )


install.packages("ggplot2")

library("ggplot2")

#Data.Frame erstellen und dem Objekt df1 zuweisen.
df <- data.frame(x,y)

#Mit head wird der Kopf des Data.Frame
#und die ersten paar Zeilen ausgegeben
head(df1)

ggplot(data = df1, aes(x=x, y=y)) +
  geom_point(color = "blue", size = 1) +
  labs(title="y= 0.2 * x^2 - 0.5 * x + 10")



#Unstige Funtion
#Sequenzen erstellen
x1 <- seq(0,10,1)
x2 <- seq(11,100,1)
x3 <- seq(101, 200, 1)

#y für jedes x berechnen
y1 <- x1
y2 <- x2 * 0.95
y3 <- x3 * 0.9

#Data.Frame erstellen und dem Objekt df1 zuweisen.
df1 <- data.frame(x1,y1)
df2 <- data.frame(x2,y2)
df3 <- data.frame(x3,y3)

#Graphik erzeugen
ggplot() +
  geom_line(data = df1, aes(x=x1, y=y1),color = "blue", size = 1) +
  geom_line(data = df2, aes(x=x2, y=y2),color = "blue", size = 1) +
  geom_line(data = df3, aes(x=x3, y=y3),color = "blue", size = 1) +
  labs(title="Unstetige Funktion", x="x", y="y")
```

```
########################################
#Beispiel Spectrum
ts <- 0.005  #Samplingperiode
fs <- 1/ts #Samplingfrequenz
t <- seq(0,100,by=ts) #Sequenz erstellen
x1 <- cos(2*pi*t) #Kurve 1
x2 <- 0.75*sin(2*pi*4*t) #Kurve 2
x3 <- 2*sin(2*pi*7*t) #Kurve 3

x = x1 + x2 + x3 #Kurve zusammensetzen

par(mfrow = c(2, 2)) #2 Spalten und 2 Zeilen für Bilder
#Bilder mit horizontaler 0-Linen
plot(x1, xlim=c(0,500), ylim=c(-2,2),
     type="l", main = "cos(2*pi*t)")
abline(h=0, lty=3)

plot(x2, xlim=c(0,500), ylim=c(-2,2),
     type="l", main = "0.75*sin(2*pi*4*t) ")
abline(h=0, lty=3)

plot(x3, xlim=c(0,500), ylim=c(-2,2),
     type="l", main = "2*sin(2*pi*7*t)")
abline(h=0, lty=3)



plot(x, xlim=c(0,500), ylim=c(-4,4),
     type="l", main = "Zusammengesetzte Kurve\nx = x1 + x2 + x3")
abline(h=0, lty=3)

#Spectrum berechnen, keine log-Skala, 5 Spikes im Kern, kein Plot
x.spec <- spectrum(x,log="no",span=5,plot=FALSE)
#Ausgabe mit Samplingfrequenz multiplizieren.

#Da die Standardausgabe in Anzahl Zyklen pro Sampling Interval ist.
spx <- x.spec$freq * fs

#Es macht Sinn diespektrale Dichte mit 2 zu multiplizieren,
#damit die Fläche unter dem Periodogramm auch der
#Varianz der Zeitreihe entspricht.
spy <- 2*x.spec$spec
```

```
#Umstellen auf eine Graphik pro Seite und Frequenz von 0 bis 10 einschränken
par(mfrow = c(1, 1))

plot(spy~spx,xlab="Frequenz",ylab="Spektrum",type="l", xlim=c(0,10))


##########################################################3
#Beispiel Speicherbedarf
id <- c(0, 1, 2, 3, 4, 5)
jahre <- c(2015, 2016, 2017, 2018, 2019, 2020)
benutzer <- c(778, 843, 870, 937, 980, 1023)
storage  <- c(2957, 3288, 3654, 4685, 5684, 6343)
#Da der rote Punkte ausserhalb der Werte der Vergangenheit ist,
#muss mit xlim, und ylim die Graphik so angepasst werden,
#dass der rote Punkte angezeigt wird.
plot(benutzer, storage, main = "Vergleich Benutzer und Storage",
     xlim=c(700, 1300), ylim=c(3000,8000))
#Prognosepunkt in rot einzeichnen.
points(1250,7800,col="red", pch = 19)

#Ratio anschauen
ratio = storage/benutzer

#Trendformel berechnen
lm.ratio = lm(ratio~id)
summary(lm.ratio)

#Trendpunkte berechnen für die Vergangenheit
pred.ratio.vergangenheit<- predict(lm.ratio, newdata = data.frame(id))
pred.ratio.vergangenheit

#Trendpunkte berechnen für die Zukunft
pred.ratio.2021 <- predict(lm.ratio, newdata=data.frame(id=6))
pred.ratio.2021

#Graphik anzeigen
plot(jahre, ratio, main = "Verlauf des Verhältnisses Speicher pro Benutzer",
     xlim=c(2015, 2021), ylim=c(min(pred.ratio.vergangenheit), max(pred.ratio.2021)))
#Prognosepunkt in rot einzeichnen
points(2021, 7800/1250,col="red", pch = 19)

#Mit einer blauen Linie den Trend zeichnen
lines(jahre, pred.ratio.vergangenheit, col="blue")
#und wieder als blauer Punkt den berechneten Trend
```

```
points(2021, pred.ratio.2021, col="blue", pch = 19)
```

```
#Lineares Modell für Anzahl Benutzer über die Zeit berechnen.
lm.benutzer <- lm(benutzer~id)
#Zusammenfassung aus lm ausgeben
summary(lm.benutzer)
#Intercept in Variable schrieben, damit dieser anschliessend in
#der Graphik als ylim obere Grenze benutzt werden kann
intercept <- coefficients(lm.benutzer)["(Intercept)"]

#Aus dem lm Modell für die gebenen x-Werte (id) die y-Werte (benutzer)
#berechnen, um diese mit den unsprünglichen Werten vergleichen zu können.
pred.vergangenheit <- predict(lm.benutzer, newdata = data.frame(id))
pred.vergangenheit
#Und nun den Trend rechnen.
pred.2021 <- predict(lm.benutzer, newdata=data.frame(id=6))
pred.2021

#In Graphik die effekten Werte einzeichnen
plot (jahre, benutzer, xlim=c(2015,2021), ylim=c(intercept, 1250),
      main="Trend Benutzer")
#Mit einer blauen Linie den Trend zeichnen
lines(jahre, pred.vergangenheit, col="blue")
#und als roter Punkte die Prognose der IT
points(2021, 1250, col="red", pch = 19)

#und wieder als blauer Punkt den berechneten Trend
points(2021, pred.2021, col="blue", pch = 19)


#################################################################
#Bsp. Occasions Autos
#Vektoren für Alter, km, ps und Preis erstellen
alter <- c(13,8, 0, 5, 5, 2, 16, 2)
km <- c(123000, 107852, 10, 160000, 48000, 23000, 76000, 19900)
ps <- c(143, 136, 150, 326, 180, 272, 140, 184)
preis <- c(7900, 20390, 41350, 19999, 21900, 44900, 4900, 33500)
```

```
#Vektoren in einen data.frame überführen
df.occassion = data.frame(alter, km, ps, preis)
#schauen, ob alles wie gewünscht in der Tabelle ist
head(df.occassion)

#lineares Modell erstellen
lm.auto <- lm(preis~., data=df.occassion)
#und Zusammenfassung ausgeben
summary(lm.auto)




######################################################################
#Grenzkosten Gipfeli

AnzahlGipfeli <- c(10,20,30,40,50,60,70,80,90,100,110,120)
Kosten <- c(5, 7.5,8.5,9,9.35,9.7, 10, 10.7,12,13,15,20)
#DataFrame erstellen
df.GipfeliKosten <- data.frame(AnzahlGipfeli, Kosten)

#Verlauf Anzahl Gipfeli und Kosten als Plot
plot(df.GipfeliKosten)

#Lineares Modell für Polynom 3. Grades
lm.gipfeli <- lm(Kosten~AnzahlGipfeli + I(AnzahlGipfeli^2) +
                   I(AnzahlGipfeli^3), data=df.GipfeliKosten)
summary(lm.gipfeli)




#Modellzahlen aus Formel berechnen.
data.frame(AnzahlGipfeli)
pred.lm.gipfeli <- predict(lm.gipfeli, newdata = data.frame(AnzahlGipfeli))
pred.lm.gipfeli



#Verlauf Anzahl Gipfeli und Kosten als Plot
plot(df.GipfeliKosten)
lines(AnzahlGipfeli, pred.lm.gipfeli, col="red", pch = 19)
```

```
#Ableitung des Resultates aus LM
# Expression or formula
f = expression(3.565e-05 * AnzahlGipfeli^3
               -6.124e-03 * AnzahlGipfeli^2
               + 3.676e-01 * AnzahlGipfeli + 2.015e+00)

D.Gipfeli <- D(f, 'AnzahlGipfeli')

#Grenzkostenfunktion ausgeben
D.Gipfeli

#Ableitung mit den Anzahl Gipfeli berechnen
D.Gipfeli.Eval <- eval(D.Gipfeli)
D.Gipfeli.Eval

#Grenzkostenverlauf anzeigen
plot(AnzahlGipfeli, D.Gipfeli.Eval)
lines(AnzahlGipfeli, D.Gipfeli.Eval, col="blue", pch = 19)


#Verlauf Anzahl Gipfeli, Kosten und Grenzkostenals Plot
plot(df.GipfeliKosten, ylim =c(0, 20))
lines(AnzahlGipfeli, pred.lm.gipfeli, col="red", pch = 19)
lines(AnzahlGipfeli, D.Gipfeli.Eval, col="blue", pch = 19)


#Das Ganze nochmals, aber dieses Mal mit einer max. Anzahl
#Gipfeli von 150.

AnzahlGipfeli <- c(0,10,20,30,40,50,60,70,80,90,100,110,120,130,140)
#Ableitung mit den Anzahl Gipfeli berechnen
D.Gipfeli.Eval <- eval(D.Gipfeli)
D.Gipfeli.Eval

#Grenzkostenverlauf anzeigen
plot(AnzahlGipfeli, D.Gipfeli.Eval)
lines(AnzahlGipfeli, D.Gipfeli.Eval, col="blue", pch = 19)
abline(h = 0.7, col = "red") #rote Linie bei 0.7
grid (NULL,NULL, lwd = 2)  #Gridlines
```

# Abbildungsverzeichnis

# Tabellenverzeichnis

## Literatur

Gross, Jürgen (2010). *Grundlegende Statistik mit R*. Springer.

Kreyszig, Erwin (1968). *Statistische Methoden und ihre Anwendungen*. Bd. 6. Vandenhoeck & Ruprecht Göttingen.

*Multiple Regressionsanalyse* (13. Aug. 2018). URL: `https://www.methodenberatung.uzh.ch/de/datenanalyse_spss/zusammenhaenge/mreg.html` (besucht am 05. 09. 2020).

Schlittgen, Rainer (2015). *Angewandte Zeitreihenanalyse mit R*. Walter de Gruyter GmbH & Co KG.

## Glossar

*A*

*ARIMA* AutoRegressive Integrated Moving Average. 39

*C*

*CLI* Command Line Interface. 6

*I*

*IDE* Integrated Development Environment. 3

*J*

*JSON* JavaScript Object Notation. 51

*M*

*M2M* Machine to Machine. 51