Computerpraktikum im GP II
Einführung in *Mathematica*

Daniel Brete Michael Karcher Jens Koesling Tim Baldsiefen

Was ist *Mathematica*

Mathematica ist ein Computeralgebrasytem, d. h., dass *Mathematica* z.B. Integrale symbolisch lösen kann. Dies ist vermutlich die meist genutzte Funktion von *Mathematica* in den ersten Semestern. Für den Praktikumsversuch nutzen wir nicht diese Eigenschaft, sondern die Möglichkeit relativ einfache Rechnungen, die man auch mit Papier und Taschenrechner durchführen könnte zu automatisieren. Diese Aufgabe lässt sich mit praktisch jeder Programmiersprache bearbeiten.

Wir haben uns für *Mathematica* entschieden, weil Sie diese Software im weiteren Verlauf Ihres Studiums noch häufiger verwenden werden. Zum Beispiel für die Bearbeitung von Übungsaufgaben zu den Theorievorlesungen. Einige Dozenten stellen auch bereits in den Vorlesungen *Physik I* oder *II* Aufgaben, die mit *Mathematica* gelöst werden sollen. In diesem Fall haben sie schon erste Erfahrungen gesammelt. Vorteile bei unseren kleinen Rechenaufgaben im Praktikum gegenüber Tabellenkalkulationen sind, dass man nicht an ein starres Tabellenraster gebunden ist, Formeln und Ergebnisse gleichzeitig sichtbar sind und die Schreibweise von Funktionen der gewohnten Darstellung näher kommt.

Mathematica kann weit mehr als wir hier ausnutzen. Neben den verblüffenden Funktionen zum symbolischen Rechnen gibt es Funktionen für numerische Berechnungen, die Konstruktion komplizierter Grafiken, das Einbinden externer C-Programme und für die statistische Datenanalyse. Ausserdem gibt es einige Textverarbeitungsfunktionen und Möglichkeiten zum Formelsatz, mit denen dieses Heft erstellt wurde.

Sie können mit *Mathematica* ein komplettes Praktikumsprotokoll von den physikalischen Grundlagen bis zur Fehlerrechnung erstellen. Zumindest aber sollten Sie nach dieser Einführung in der Lage sein, grafische Auswertungen statt mit Bleistift und Millimeterpapier mit *Mathematica* auszufühen

Hinweise zur Bedienung

Starten

Geben Sie in einem Terminalfenster `mathematica` ein. Geschickt ist es, vor dem Starten in das Verzeichnis zu wechseln, in dem sich die zu bearbeitenden Dateien befinden oder erstellt werden sollen. Sie ersparen sich so das wiederholte Auswählen von Verzeichnissen in einer etwas unübersichtlichen Dialogbox. Sie können auch beim Programmaufruf eine Datei mit angeben, z.B.: »`mathematica meinedatei.nb`«. *Mathematica*- Dateien heissen Notebooks und haben die Endung »`.nb`«.

Dateien öffnen, schliessen, Programm beenden

Diese Funktionen erreichen Sie über das Menü **File**, dass sich mit der Maus in gewohnter Weise bedienen lässt. Das Beenden des Programms gelingt nur mit dem Befehl **Quit** aus diesem Menü. Ein Doppelklick auf das Fenster-schliessen-Symbol in der Titelleiste des Programms schliesst zwar das aktuelle Fenster, öffnet aber ein neues, falls es sich um das einzige *Mathematica*-Fenster handelt.

Einige Unzulänglichkeiten

In *Mathematica* funktionieren leider viele Tasten nicht so, wie man es gewohnt ist. Auf vielen Systemen löscht DEL nicht vorwärts sondern rückwärts. **Der numerische Zehnerblock kann nicht benutzt werden.** Bei gedrückter Nummlocktaste funktioniert kaum noch etwas. Glücklicherweise ist der Spuck beendet, wenn Sie sie wieder ausschalten.

Drucken

Drucken können Sie über die Druckfunktion im Menu **File**. Achten Sie unbedingt darauf, dass sie die Box **Include Mathematica Fonts** einschalten; die Sonderzeichen werden sonst nicht richtig gedruckt. Auch die Wahl des richtigen Papierformats (A4) ist entscheidend. Um auf dem Drucker im Rechenerraum auszudrucken, geben Sie im Feld **Print to** »`lpr -Ptal`« ein. Oder erzeugen Sie eine Postscriptdatei, in dem Sie den Punkt **File** markieren und einen Dateinamen eingeben. Vorsicht, Sie können gleichzeitig sowohl in eine Datei, als auch auf den Drucker drucken. Achten Sie also darauf, dass nur die gewünschte Option markiert ist. Gegenwärtig sind die von *Mathematica* erzeugten Postscript-Dateien nicht ganz DSC-Standard konform. Dies kann dazu führen, dass die Sonderzeichen nicht richtig wiedergegeben werden, wenn Sie versuchen die Dateien mit den pstools (psbook, psnup usw.) nachzubearbeiten.

Hilfe

Die Hilfe rufen Sie über den Menupunkt **Help → Documentation Center ...** auf.
Den gesuchten Begriff oder Teile davon (z.B. `ParametricPlot3D` oder auch nur `Paramet`) im Feld **SEARCH:** eingeben und mit ↵ bestätigen.

Meist ruft man aus einem von zwei Gründen die Hilfe auf. Entweder sucht man eine Befehlsbeschreibung, man weiß also schon (oder vermutet es zumindest), welcher Befehl das leistet, was man erreichen möchte; oder man will etwas bestimmtes erreichen, weiss aber nicht mit welchem Befehl.
Im ersten Fall sucht man einfach den Befehlsnamen. Die Befehlserklärungen sind meist sehr illustrativ.
Im zweiten Fall sollte man nach Begriffen suchen, die in Texten zu eben diesem Thema vorkommen könnten. Da die Hilfe auch das *Mathematica*- Buch enthält, kann einem auch hier meist geholfen werden.

■ Eingeben von Befehlen

Klicken Sie auf die weisse Arbeitsfläche oder erzeugen Sie ein neues leeres Notbook mit **File→New** und beginnen Sie zu schreiben. Eine Zelle (Eingabe) in Mathematica kann mehrere Zeilen umfassen. Der Zeilenwechsel erfolgt einfach mit ENTER. Nach der letzten Anweisung drücken Sie SHIFT ENTER, um alle Befehle in einer Zelle auf einmal auszuführen. Dieses SHIFT ENTER ruft den Kernel, den mathematischen Kern Mathematicas, auf und lässt diesen die eingegebene Zelle interpretieren und ausrechnen. Dieser Vorgang heisst Evaluieren (Evaluation). Im folgenden sind die fettgedruckten Zeilen die **`Eingabe,`** und die dünn gedruckten Zeilen die zugehörige `Ausgabe` von *Mathematica*. Probieren Sie das folgende Beispiel aus. In *Mathematica* können Sie für die Multiplikation anstelle des Sterns »*« auch ein Leerzeichen verwenden.

```
2 × 5
10
```

Möchten Sie zwischen zwei Zellen (Durch eckige Klammern am rechten Rand markiert.) eine neue Zelle einfügen, klicken Sie zwischen die Zellen. Es erscheint eine horizontale Linie. Wenn Sie zu schreiben beginnen, öffnet sich die neue Zelle.

Man kann auch das ganze Notebook auf einmal auswerten. Hierzu muss man in der Befehlsleiste **Evaluation→ Evaluate Notebook** auswählen.

Mathematica speichert alle Evaluierungen (mit allen Daten, Variablen und Funktionen), die es während einer Sitzung durchführt. So kommt es, dass *Mathematica* Funktionen, die man schon vor langem gelöscht hat, immer noch kennt und deren Funktionsnamen deshalb nicht freigibt.
In einer solchen Situation hilft es, den Kernel zu beenden und dann das Notebook erneut insgesamt zu evaluieren. Dann kann man dort, wo man ausgesetzt hatte, weiter arbeiten. Den Kernel beendet man über **Evaluation→Quit Kernel→local**.

■ Grundlagen der Syntax von Mathematica

Sämtliche in *Mathematica* vordefinierten Funktionen beginnen mit einem Großbuchstaben. Die Parameterlisten für Funktionen werden in **eckigen** Klammern angegeben. Die runden Klammern dienen nur zum Klammern von mathematischen Ausdrücken. Daher heisst es `3 (2 + 4)` aber `Sin[2 Pi]`, da nur im ersten Fall die Klammern die Funktion der Gruppierung von Termen haben, im zweiten Fall aber die Klammern angeben, dass Argumente an eine Funktion übergeben werden. Da alle vordefinierten Namen (die von Funktionen wie `Sin`, `Cos` oder `Sqrt` oder von Variablen wie `Pi`) mit einem Großbuchstaben beginnen, wird empfohlen, alle selbst definierten Namen mit einem Kleinbuchstaben anfangen zu lassen.

- Ein einfacher Ausdruck: `4/(2+3)`
- Ein Ausdruck, in dem eine Funktion benutzt wird: `3*Sin[2*(2+x)]`

■ Kommentare

Will man, dass Mathematica eine Zelle nicht auswertet, da sie nur Fließtext oder einen Kommentar enthält, so muß Mathematica gesagt werden, dass es sich nur um eine Text-Zelle handelt. Dazu muss die Zelle am rechten Rand markiert werden, dann wählt man aus dem Menu **Format→Style→Text.** Markiert man hingegen nur den Inhalt der Zelle oder Teile davon, bewirken die verschiedenen Styles nur unterschiedliche Formatierungen, ohne die Art der Auswertung der Zelle zu verändern.

Es gibt eine große Menge weiterer Style- und allgemeiner Einstellungen für Zellen, die sich unter **Format→Style**, bzw. **Format→...** befinden.

■ Ausdrücke

Ausdrücke, die *Mathematica* zur Bearbeitung gegeben werden, werden so weit wie möglich ausgewertet:

```
3 (2 + 4)
```
```
18
```
```
3 (2 + 5 a + 3)
```
```
3 (5 + 5 a)
```

Wie man sieht, werden die Zahlen so weit wie möglich zusammengefasst, aber sobald Variablen auftreten, arbeitet *Mathematica* sehr vorsichtig. Um *Mathematica* zu sagen, dass ein Ausdruck vereinfacht werden soll, verwendet man die Funktion `Simplify:`

```
Simplify[3 (2 + 5 a + 3)]
```
```
15 (1 + a)
```

Hier hat *Mathematica* jetzt von selbst den Faktor 5 ausgeklammert.

Die Funktion **`FullSimplify`** tut dasselbe, versucht jedoch bei komplizierten Ausdrücken noch mehr Umformungen und liefert so manchmal ein besseres Ergebniss.
Will man einen Ausdruck ausmultiplizieren, muss man `Expand` verwenden:

```
Expand[3 (2 + 5 a + 3)]
```
```
15 + 15 a
```

Möchte man dagegen aus einem ausmultiplizierten Term wieder ein Produkt gewinnen, kann man `Factor` benutzen, obwohl in den meisten Fällen `Simplify` auch schon ausreicht.

```
Factor[x^3 + 3 x^2 y + 3 x y^2 + y^3]
```
```
(x + y)^3
```

■ Variablen

Variablen sind vom Prinzip her sehr einfach. Man kann mit dem Gleichheitszeichen Variablennamen, die, wie schon oben empfohlen, mit einem Kleinbuchstaben beginnen sollten, einen Wert zuweisen, ohne dass eine vorherige Deklaration erforderlich ist. Sobald der Name später wieder auftaucht, setzt *Mathematica* dort den zugewiesenen Wert ein. Mit der folgenden Zeile wird `n` auf `3` gesetzt:

```
n = 3
```
```
3
```

Jetzt kann man `n` wie eine Zahl verwenden, und wenn man `n` danach einen neuen Wert zuweist, gilt ab dann dieser Wert:

```
n * 2
```
```
6
```
```
2^n
```
```
8
```
```
n = 4
```
```
4
```
```
n * 2
```
```
8
```

Wie man sieht, kommt nach der Zuweisung `n=4` bei dem gleichen Ausdruck `2*n` ein anderer Wert heraus, da sich `n` selbst geändert hat. Diese Funktion ist manchmal nützlich, sollte aber vorsichtig benutzt werden, denn es ist in *Mathematica* vorgesehen, dass man jederzeit jede Formel ändern und neuberechnen kann. Das geht natürlich nur solange gut, wie sich die Variablen seit der ersten Berechnung nicht geändert haben. Daher lautet eine Empfehlung: Variablen sollten nur einmal gesetzt werden, und zwar natürlich vor jeder Anwendung dieser Variablen.

- **Unterdrücken der Ausgabe**

Häufig möchte man das Ergebnis einer Berechnung nicht auf dem Bildschirm ausgeben. Um dies zu erreichen, kann man die Anweisung deren Ausgabe unterdrückt werden soll mit einem Semikolon abschliessen. Wir machen davon in diesem Text gelegentlich Gebrauch um Platz zu sparen. Wenn Sie die Beispiele nachvollziehen wollen, können sie die Semikolons weglassen, um die Ausgabe zu sehen.

```
5 + 3;
```

ist ohne Ausgabe sinnlos. Aber wenn Sie einer Variablen einen Wert zuweisen, kennen Sie die Ausgabe schon:

```
h = 7;
```

Ausserdem können mit dem Semikolon mehrere Anweisungen in einer Zeile eingegeben werden.

```
h = 7; l = 9;
```

- **Sonderzeichen**

Will man nun griechische Buchstaben in einem Ausdruck, Umlaute in einem Kommentar oder mathematische Symbole wie ∈ oder ∀ in einem Text verwenden, muss man diese Zeichen gesondert einfügen, wenn sie sich nicht auf der Tastatur befinden. Dies kann man über Paletten, die man nach Belieben im Menu unter **Palettes → ...** aufrufen kann. Für viele dieser Zeichen existiert aber auch ein Tastatur Shortcut. Diese Shortcuts werden in der Form ESC Shortcut ESC eingegeben.

So sind, z.B.
ESC D ESC = Δ
ESC d ESC = δ
ESC [[ESC = ⟦
ESC a " ESC = ä

- **Listen - Vektoren und Matrizen**

Die grundlegende Datenstruktur in Mathematica ist die Liste. Eine Liste ist eine Aufzählung oder eine Menge von Elementen. Hierbei kommt es, im Gegensatz zu mathematischen Mengen, jedoch auch auf die Reihenfolge der Elemente an. Die Elemente einer Liste können so gut wie alles sein, was Mathematica kennt. Also ist auch eine Liste einer Zahl, einer Funktion und einer weiteren Liste eine Liste.

Um Elemente zu einer Liste zusammenzufassen, gibt es den Befehl `List`.

```
List[1, a, ArcTan, {c, d}, "String"]
```
```
{1, a, ArcTan, {c, d}, String}
```

Abkürzend kann man aber auch einfach die Elemente in ein Paar gemeinsamer geschweifter Klammern setzen.

```
{1, a, ArcTan, {c, d}, "String"}
```
```
{1, a, ArcTan, {c, d}, String}
```

Die meisten naiven Funktionen in Mathematica sind dergestalt, dass sie auf eine Liste angewendet auf jedes Listenelement einzeln wirken. Dies ist eine Kurzschreibweise für die Funktion `Map`.

```
Sin[{0, Pi / 2, Pi}]
```
```
{0, 1, 0}
```

kurz für:

```
Map[Sin, {0, Pi / 2, Pi}]
```
```
{0, 1, 0}
```

Listen sind eine Möglichkeit, Vektoren und Matrizen darzustellen. Vektoren fasst man als Listen von Zahlen (`Real`) auf und Matrizen als Listen von Vektoren. Das Arbeiten mit Vektoren und Matrizen ist in *Mathematica* recht bequem möglich, da die einfachen Rechenoperatoren Vektoren und Matrizen elementweise verknüpfen:

```
{2, 3, 4} + {1, 5, 8}
```
```
{3, 8, 12}
```

Ist nur einer von zwei Operanden ein Vektor, der andere dagegen ein einfacher Ausdruck, so wird der Ausdruck mit jedem einzelnen Element des Vektors verknüpft. Das mag bei der Addition zwar reichlich sinnlos erscheinen, ist aber das Verhalten, das man bei einer Multiplikation erwartet:

```
{2, 3, 4} + 1
```
```
{3, 4, 5}
```
```
{2, 3, 4} * 2
```
```
{4, 6, 8}
```

Man kann natürlich auch zwei Vektoren miteinander multiplizieren, aber das führt wie bei der Addition, bei der eine Zahl zu einem Vektor addiert wird, nicht zu mathematisch sinnvollen Operationen:

```
{2, 3, 4} * {1, 1, 2}
```
```
{2, 3, 8}
```

Zum Bestimmen des Skalarproduktes verwendet man statt des Sternes einen Punkt. Dieser steht im allgemeinen für Matrixprodukt. Das Skalarprodukt zweier Vektoren ist auch nichts weiter als das Matrixprodukt eines Zeilenvektors mit einem Spaltenvektor. Da *Mathematica* zwischen Zeilen- und Spaltenvektoren sowieso nicht unterscheidet, ist das Skalarprodukt tatsächlich ein Matrixprodukt

```
{2, 3, 4}.{1, 1, 1}
```
```
9
```

Eine Matrix wird als Vektor von Vektoren gleicher Länge eingegeben:

```
{{1, 2}, {3, 4}}
```
```
{{1, 2}, {3, 4}}
```

Die Ausgabe erscheint normalerweise schöner, wenn man die Matrix in der sogenannten `MatrixForm` darstellt, indem man die Funktion `MatrixForm` auf die Matrix anwendet. *Mathematica* macht die Ausgabe in der Spezialform dadurch kenntlich, dass an die Ausgabennummer `//MatrixForm` angehängt wird. Das kommt von einer ganz allgemeinen Methode, Funktionen zu verketten, die erst etwas später erklärt wird:

```
MatrixForm[{{1, 2}, {2, 3}}]
```
$$\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$$
```
MatrixForm[{{1, 2}, {2, 3}} + {{1, 0}, {0, 1}}]
```
$$\begin{pmatrix} 2 & 2 \\ 2 & 4 \end{pmatrix}$$
```
MatrixForm[{{1, 2}, {2, 3}} * {{1, 0}, {0, 1}}]
```
$$\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$$
```
MatrixForm[{{1, 2}, {2, 3}}.{{1, 0}, {0, 1}}]
```
$$\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$$

Man beachte vor allem den Unterschied zwischen der elementweisen Multiplikation mit dem Stern und der Matrixmul tiplikation mit dem Punkt.

- **Zugreifen auf Elemente einer Liste:**

Wir definieren zu Demonstrationszwecken einen Vektor `u` und eine Matrix `v`.

```
u = {6, 9}
```
```
{6, 9}
```
```
MatrixForm[v = {{1, 2}, {3, 4}}]
```
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Häufig kommt es vor, dass man auf ein ganz bestimmtes Element eines Vektors, einer Matrix oder allgemein einer Liste zugreifen will. Diesen Zugriff ermöglicht die Funktion `Part`. Ihre Kurzformen sind viel einfacher. Alle der folgenden Formen sind Kurzschreibweisen für den mit `Part` realisierten Zugriff auf die Elemente einer Liste.

Zugreifen auf das erste Element des Vektors `u` geschieht so:

```
Part[u, 1]
u[[1]]
```
```
6
```
```
6
```

und der Zugriff auf das Element in der ersten Zeile und zweiten Spalte der Matrix v so:

```
Part[Part[v, 1], 2];
Part[v, 1, 2];
v[[1]][[2]];
v[[1, 2]];
v[[1, 2]];
v[[1, 2]];
v[[1,2]]
```
```
2
```

der Zugriff auf eine komplette Spalte kann mit Hilfe des Platzhalters "All" erreicht werden:

```
v[[All, 2]]
```
```
{2, 4}
```

- **Die Listeneingabehilfe**

Die Eingabe längerer Listen z.B. Messwertetabellen mit geschweiften Klammern ist umständlich und unübersichtlich. Wenn sie solche langen Tabellen nicht aus einer Datei einlesen möchten (siehe später), können Sie die Listeneingabehilfe verwenden. Wählen Sie hierzu aus dem Menu **Insert→Table/Matrix/Palette**. In dem sich öffnenden Dialogfeld wählen Sie, ob Sie die Liste als Matrix oder Tabelle formatieren möchten und die Zahl der Zeilen und Spalten. In der so eingegügten Vorlage können Sie mit TAB zum nächsten freien Feld springen.

- **Funktionen**

Funktionen werden genauso wie Variablen mit dem einfachen Gleichheitszeichen definiert. Dabei steht auf der linken Seite des Gleichheitszeichens ein sogenanntes Muster. Über Muster kann man sehr viel sagen, was aber den Rahmen dieses Skripts bei weitem sprengt. Daher fassen wir hier kurz zusammen, was von vornehmlicher Bedeutung ist: Schreibt man einfach `f[x]=x^2,` scheint es auf den ersten Blick auch zu funktionieren:

```
f[x] = x^2
```
$$x^2$$
```
x + f[x]
```
$$x + x^2$$

Aber sobald man versucht, ein anderes Argument als `x` zu verwenden, tut *Mathematica* plötzlich so, als ob es keine Ahnung hätte, wie `f` definiert ist:

```
y + f[y]
```
```
y + f[y]
```

Hier wurde die Funktion nicht eingesetzt. Man kann auch noch `Simplify` darauf anwenden: Es klappt einfach nicht. Der Fehler liegt in der Definition der Funktion. Wie schon erwähnt, steht auf der linken Seite des Gleichheitszeichens ein Muster. In diesem Fall ist es ein sehr einfaches Muster, nämlich die Funktion `f` mit dem Argument `x`. Daher wird »Die Funktion `f` mit dem Argument `x`« auch durch die rechte Seite ersetzt. Allerdings nicht »Die Funktion `f` mit dem Argument `y`«, da diese Eingabe nicht auf das Muster passt.
Was man benötigt, ist ein Muster, das allgemeiner gefasst ist. Dazu verwendet man den Unterstrich, häufig auch »Blank« genannt, da er für einen beliebigen Ausdruck steht, wie eine Blankostelle, die noch ausgefüllt werden muss. Daher der nächste Versuch mit einer konstanten Funktion:

```
constZwei[_] = 2
```
```
2
```
```
constZwei[5] + constZwei[x] + constZwei[x^2]
```
```
6
```

Hier hat *Mathematica* festgestellt, dass das Muster »Die Funktion `constZwei` mit einem beliebigen Argument« in der unteren Zeile dreimal vorkommt und für jedes Vorkommen `2` eingesetzt, so dass sich das Gesamtergebnis `6` ergeben hat. Das ist aber auch noch nicht ganz das, was man eigentlich braucht, da man zwar jetzt ein schön allgemeines Muster hat, aber leider nicht mehr an das Argument herankommt.
Daher gibt es in *Mathematica* die Möglichkeit, dem Blank einen Namen zu geben, indem man ihn vor den Unter-

strich schreibt. Dieser Name kann dann auf der rechten Seite wie eine normale Variable verwendet werden: Die Funktion `quadrat` wird jetzt mit dem Muster »Die Funktion `quadrat` mit einem beliebigen Argument, dass im weiteren `x` genannt wird,« definiert.

```
quadrat[x_] = x^2
```

x^2

```
quadrat[3]
```

9

Man sollte darauf achten, dass, wenn auf der rechten Seite eine Variable vorkommt, die bereits einen Wert zugewiesen bekommen hat, dieser Wert verwendet wird, egal, ob es vielleicht einen Parameter gibt, der den gleichen Namen hat:

```
n = 15
```

15

```
negiereFalsch[n_] = -n
```

−15

```
negiereRichtig[nn_] = -nn
```

−nn

```
{negiereFalsch[8], negiereRichtig[8]}
```

{−15, −8}

Die Funktion `negiereFalsch` ist ein Beispiel für dieses Problem. Man sieht es allerdings hier schon in der Antwort von *Mathematica*, denn nicht jede Zahl kann negiert -15 ergeben. Leider bemerkt man bei schwierigen Funktionen dieses Problem nicht immer sofort. Daher die Warnung: Variablen sollten nur mit Bedacht und aussagekräftigeren Namen als `n` benutzt werden.

Funktionen können mit = oder := definiert werden. Üblicherweise wird := verwendet. Dann tritt auch das oben beschriebene Problem nicht auf.

```
negieredochnichtfalsch[n_] := -n
```

```
negieredochnichtfalsch[27]
```

−27

Bei Verwendung von = wird die rechte Seite erst so weit wie möglich ausgewertet und dann als Funktionsdefinition gespeichert. Verwendet man hingegen := erfolgt die Auswertung erst beim Aufruf der Funktion, nachdem die Parameter eingesetzt wurden. Bei vielen einfachen Funktionen ist das egal. Der Unterschied wird aber bei Konstruktio nen wie im folgenden Beispiel, das wir aus der Hilfe entnommen haben, deutlich.

```
ex[x_] := Expand[(1 + x)^2]
```

```
iex[x_] = Expand[(1 + x)^2]
```

$1 + 2\,x + x^2$

```
ex[y + 2]
iex[y + 2]
```

$9 + 6\,y + y^2$

$1 + 2\,(2 + y) + (2 + y)^2$

Es gibt drei Möglichkeiten, eine Funktion auf einen Ausdruck anzuwenden. Bisher haben wir nur die Funktionsnotation kennengelernt. Darüberhinaus gibt es noch die Präfix- und die Postfixnotation. Die Syntax lautet `Funktion@Argument`, bzw. `Argument//Funktion`.

Jeweils das gleiche leisten also

```
Sqrt[4];
Sqrt@4;
4 // Sqrt
```

2

■ Symbolisch Rechnen

Was *Mathematica* wirklich besser macht als einen Taschenrechner ist nicht nur die Möglichkeit, mit selbstdefinierten Funktionen oder Vektoren zu rechnen, sondern vor allem die Möglichkeit, symbolisch zu rechnen, also die mathematische Terme zu bearbeiten. Zum Beispiel kann *Mathematica* viele Gleichungen exakt lösen, in dem es die Umformungen, die nötig sind, um nach den Variablen aufzulösen selbst vornimmt:

```
Solve[3 * x + x^2 == 10]
```

```
{{x → -5}, {x → 2}}
```

Für den Vergleich zweier Terme verwendet man das doppelte Gleichheitszeichen, das im Gegensatz zum einfachen Gleichheitszeichen keine Zuweisung, sondern einen Vergleich vornimmt. Die Funktion `Solve` dient dazu, die Lösungen einer Gleichung zu suchen. Solange nur eine Variable darin vorkommt, wird automatisch nach dieser Variablen aufgelöst. Dabei darf der Variablen vorher kein Wert zugewiesen worden sein, denn dieser Wert würde eingesetzt, bevor die Gleichung gelöst wird, und damit kann *Mathematica* nur noch feststellen, ob der aktuelle Wert der Variablen die Gleichung löst oder nicht:

```
n = 15
```

```
15
```

```
Solve[2 * n == 30]
```

```
{{}}
```

```
Solve[2 * n == 29]
```

```
{}
```

Die Ausgabe sieht etwas seltsam aus. In dem ersten Fall erhält man eine Liste, die die leere Liste enthält, im zweiten Fall einfach die leere Liste. Man erklärt, dass die einfache leere Liste bei `Solve` stets dann herauskommt, wenn eine Gleichung unlösbar ist, da es dann eben keine Lösung gibt, die in der Liste stehen könnte, und die leere Liste in einer Liste bei allgemeingültigen Gleichungen, da es eine Lösung gibt, bei der aber keine weiteren Aussagen gemacht werden können. Diese Aussage "man muss nichts weiter vorraussetzen" wird durch die innere leere Liste ausgedrückt, und ist in der Liste aller Lösungen das einzige Ergebnis.

Man kann symbolisch nicht nur Gleichungen lösen, sondern auch differenzieren und integrieren:

```
D[2 x + x^2, x]
```

```
2 + 2 x
```

```
Integrate[2 + 2 x, x]
```

$$2\,x + x^2$$

Im ersten Fall wurde der Term `2x+x^2` nach `x` differenziert, im zweiten Fall wurde über x integriert. Die Integrationskonstante lässt *Mathematica* weg, man muss sie selbst addieren, wenn man eine Integrationskonstante braucht.

■ Numerisch rechnen

Die Anweisung

```
Sin[Pi / 8]
```

$$\text{Sin}\left[\frac{\pi}{8}\right]$$

funktioniert nicht so, wie Sie es vermutlich erwartet haben. *Mathematica* wertet Ausdrücke normalerweise nur symbolisch aus. Wenn das nicht funktioniert wird der Ausdruck unverändert zurückgegeben. Nur wenn der Ausdruck Gleitkommazahlen enthält, erfolgt automatisch eine numerische Auswertung. Ganze Zahlen können als Gleitkommazahlen geschrieben werden, in dem man einen Punkt anhängt.

```
{Sin[Pi], Sin[Pi / 8], Sin[0.125 Pi], Sin[1], Sin[1.]}
```

$$\left\{0,\ \text{Sin}\left[\frac{\pi}{8}\right],\ 0.382683,\ \text{Sin}[1],\ 0.841471\right\}$$

Mit der Funktion `N` kann man eine numerische Auswertung erzwingen. Sie wird oft in der übersichtlicheren Postfixnotation verwendet.

```
N[Sin[Pi / 8]]
Sin[Pi / 8] // N
```

```
0.382683
```

```
0.382683
```

■ Wichtige Befehle

▪ ReadList[ ]

`ReadList["file",{typ1,typ2,...}]` liest Daten aus `file` ein und erstellt eine Liste mit Elementen der Form `{typ1,typ2,...}`.

```
datenliste = ReadList["sinusn.dat", {Real, Real}]
```

```
{{0., 0.}, {1., 0.0174524},
 {2., 0.0348995}, {3., 0.052336}, {4., -0.756802}}
```

`Real`, `Integer`, `Complex` sind hierbei grundlegende Zahlentypen, die Mathematica kennt. Der Punkt hinter einer Zahl in der Ausgabe bedeutet, dass es sich um eine Gleitkommazahl vom Type `Real` und nicht um eine Ganzzahl vom Typ `Integer` handelt.

▪ TableForm[ ]

Um diese Daten übersichtlicher darzustellen, kann man sich des Befehles `TableForm` bedienen.

```
TableForm[datenliste]
```

```
0.  0.
1.  0.0174524
2.  0.0348995
3.  0.052336
4.  -0.756802
```

In Postfixnotation wird auch die Befehlseingabe übersichtlicher:

```
datenliste // TableForm
```

```
0.  0.
1.  0.0174524
2.  0.0348995
3.  0.052336
4.  -0.756802
```

Der Tabelle kann man noch Spaltenüberschriften verpassen, indem man beim Funktionsaufruf von `TableForm` die Option `TableHeadings` anspricht. Die leere geschweifte Klammer steht dabei für die hier nicht benutzte Zeilenbeschriftung.

```
TableForm[datenliste, TableHeadings → {{}, {"n", "sin(n)"}}]
```

n	sin(n)
0.	0.
1.	0.0174524
2.	0.0348995
3.	0.052336
4.	-0.756802

Wie man schnell bemerkt, ist eine Postfixnotation unter Verwendung von mehreren Argumenten nicht möglich. Hier muss man sich mit einem Trick behelfen. Auch wenn das zuerst umständlicher aussieht, als in der Funktionsnotation, zahlt es sich bei längeren Ausdrücken durch eine größere Übersichtlichkeit wieder aus. Dazu definiert man eine Hilfsfunktion, die sich auf ein Argument beschränkt. Die primitive Variante sieht dann so aus:

```
tableWithHeads[table_] =
  TableForm[table, TableHeadings → {{}, {"n", "sin(n)"}}];
datenliste // tableWithHeads;
```

Das ergibt zwar das gewünschte Ergebnis, allerdings ist es etwas aufwändig, für die einmalige Benutzung eine eigene Funktion zu definieren. Daher kennt *Mathematica* das Konzept der »pure functions«, das man vielleicht mit »anonyme Funktionen« übersetzen könnte. Eine pure function hat keinen Namen, sondern wird dort definiert, wo sie benutzt wird.

Die ausführliche Variante benutzt dazu den *Mathematica*-Befehl `Function`, der eine solche Funktion erzeugt. Der

erste Parameter von `Function` ist der Name der Variablen, die als Parameter der zu definierenden anonymen Funktion dient. Möchte man in seiner Funktion mehrere Parameter benutzen wollen, so gibt man einfach statt einer Variablen eine Liste von Variaben als ersten Parameter von `Function` an. Der zweite Parameter von `Function` ist die Definition der anonymen Funktion. Also das, was auf der rechten Seite des Gleichheitszeichen in der Funktionsdefinition steht. Das Beispiel sieht jetzt so aus:

```
datenliste // Function[table,
   TableForm[table, TableHeadings → {{}, {"n", "sin(n)"}}]];
```

Die benannten Parameter, hier also `table`, machen die ganze Sache zwar etwas übersichtlicher, aber auch länger. Daher gibt es auch noch eine Syntax, bei der nicht nur die Funktion keinen Namen mehr hat, sondern auch die Parameternamen entfallen. Den ersten (und häufig einzigen) Parameter bezeichnet man mit #, die weiteren Parameter mit `#2`, `#3` und so weiter. Der Konsistenz wegen kann der erste Parameter auch mit `#1` angesprochen werden. Um diese Syntax zu benutzen, lässt man das erste Argument von `Function` einfach ganz weg:

```
datenliste //
  Function[TableForm[#, TableHeadings → {{}, {"n", "sin(n)"}}]];
```

Und schliesslich gibt es noch die Möglichkeit, auch das Wort `Function` durch den Postfixoperator & zu ersetzen, der hinter den Funktionsrumpf zu schreiben ist, wodurch man kürzere und daher übersichtlichere Ausdrücke erhält, sofern man die Möglichkeiten nicht überstrapaziert:

```
datenliste // TableForm[#, TableHeadings → {{}, {"n", "sin(n)"}}] &
```

n	sin(n)
0.	0.
1.	0.0174524
2.	0.0348995
3.	0.052336
4.	−0.756802

- **Transpose[ ]**

`Transpose[list]` bildet das Transponierte eines Vektors oder einer Matrix.

```
(v = {{1, 2, 3}}) // MatrixForm
(M = {{11, 12, 13}, {21, 22, 23}, {31, 32, 33}}) // MatrixForm
```

$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix}$$

```
Transpose[v] // MatrixForm
Transpose[M] // MatrixForm
```

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} 11 & 21 & 31 \\ 12 & 22 & 32 \\ 13 & 23 & 33 \end{pmatrix}$$

- **Replace[ ], ReplaceAll[ ] oder /.**

`ReplaceAll` benutzt Ersetzungsregeln der Form `x→2`, um Ausdrücke umzuformen, bzw. zu lösen. Die Syntax ist `ReplaceAll[Ausdruck,Ersetzungsregel(n)]` oder äquivalent dazu auch `Ausdruck /. Ersetzungsregel(n)`.

```
ReplaceAll[x + x^2, x → 2]
x + x^2 /. x → 2
```

```
6
```

```
6
```

Mehrere Ersetzungsregeln werden in Form einer Liste von Ersetzungsregeln angegeben.

```
Cos[a] + b^2 /. {a → 0, b → 1}
```

```
2
```

Im Gegensatz hierzu ersetzt der Befehl `Replace` nur ganze Ausdrücke.

```
Replace[1 + x, 1 + x → 1 + a]
Replace[1 + x, x → a]
```

```
1 + a
```

```
1 + x
```

Das Interessante an `Replace` ist nun, dass man hiermit Ersetzungen bis zu einem vorher bestimmten Level eines Ausdrucks vornehmen kann. Der Level bezeichnet hier die Mathematica-interne Schachtelung der Ausdrücke. Denn Level kann man immer mit der Funktion `FullForm` ansehen, der Ausdrücke vollständig in Funktionsnotation, so wie sie in *Mathematica* intern dargestellt werden und auch eingegeben werden könnten, ausgibt.

```
FullForm[a + b - c * d]
```

```
Plus[a, b, Times[-1, c, d]]
```

```
FullForm[x^2]
```

```
Power[x, 2]
```

```
Replace[x^2, x → a, 1]
```

$$a^2$$

Der Level von x in x^2 ist 1, wie man aus `FullForm[x^2]` zu ersehen ist :

$$\underbrace{\texttt{Power}\left[\underbrace{x}_{\text{Level 1}}, \underbrace{2}_{\text{Level 1}}\right]}_{\text{Level 0}}$$

```
FullForm[Sin[x]^2]
```

```
Power[Sin[x], 2]
```

```
Replace[Sin[x]^2, x → a, 1]
```

```
Sin[x]^2
```

`Replace` hatte hier keinen Effekt, weil sich x erst auf dem zweiten Level befindet.
Betrachten wir einmal, wie sich das Ersetzen in Listen verhält.

```
{FullForm[{x}] , Replace[{x}, x → a, 2]}
```

```
{List[x], {a}}
```

```
{FullForm[{{x}}] , Replace[{{x}}, x → a, 2]}
```

```
{List[List[x]], {{a}}}
```

```
{FullForm[{{{x}}}] , Replace[{{{x}}}, x → a, 2]}
```

```
{List[List[List[x]]], {{{x}}}}
```

Die dritte Ersetzung hatte wiederum keinen Effekt, da sich das x in der dritten Liste auf der dritten Ebene befindet.

- **Funktionen aus Packages**

Für das folgende Beispiel wird der Befehl `ErrorListPlot` benötigt, der nicht zum Standardsprachumfang gehören, sondern in einem Zusatzpaket enthalten ist, welches mit der folgenden Eingabe geladen wird. Bitte beachten Sie, dass es sich bei dem Hochkomma »`« nicht um ein Apostroph »'« sondern um einen Akzent grave handelt, der sich auf englischen Tastaturen auf der Taste links neben der »1« befindet.

```
Needs["ErrorBarPlots`"]
```

`ErrorListPlot`$\left[\left\{ ..., \left\{x_i, y_i, \Delta\, y\right\}, ...\right\}\right]$ zeichnet die Meßwerte (x, y) mit y – Fehlern Δ y

Mit `PlotStyle→{Anweisungen}` kann man grafische Eigenheiten, wie Farbe und Strichbreite eines Plots bestimmen. Hier gibt es eine große Anzahl verschiedener Möglichkeiten, die in der Mathematica-Hilfe beschrieben werden. Dies gilt ebenso für den Befehl `PointSize`.

■ Anwendungsbeispiel: Lineare Regression

Dieses Beispiel sollten Sie Schritt für Schritt nachvollziehen. Am Ende werden Sie eine Funktion erhalten, die Sie für die Auswertung Ihrer Praktikumsversuche weiter verwenden können.

- **Laden der Mathematika-Standardpakete**

```
Needs["ErrorBarPlots`"]
```

Mit dem nächsten Befehl werden die Messwerte aus einer Textdatei in eine Liste gelesen. In der Datei stehen die Werte durch Leerzeichen getrennt in drei Spalten.

```
linear = ReadList["linear.txt", {Real, Real, Real}]
```

```
{{0., 2.8, 1.}, {2., 5.2, 1.5}, {4., 6.8, 1.2},
 {6., 9.6, 0.9}, {8., 11.2, 1.4}, {10., 14., 1.4}, {12., 15.6, 1.5},
 {14., 17., 1.2}, {16., 19.8, 1.3}, {18., 21.4, 1.5}}
```

Die inneren Listen der so erzeugten Liste `linear` entsprechen den Zeilen der Datei. Der Zugriff auf die einzelnen Werte wird einfacher, wenn wir Listen erzeugen, die einer Spalte der ursprünglichen Datei entsprechen. Diese Aufgabe lässt sich mit folgender Eingabe lösen:

```
x = linear[[All, 1]]
y = linear[[All, 2]]
Δ y= linear[[All, 3]]
```

```
{0., 2., 4., 6., 8., 10., 12., 14., 16., 18.}
```

```
{2.8, 5.2, 6.8, 9.6, 11.2, 14., 15.6, 17., 19.8, 21.4}
```

```
{1., 1.5, 1.2, 0.9, 1.4, 1.4, 1.5, 1.2, 1.3, 1.5}
```

In den Formeln für die lineare Regression treten Summen über alle Datenpunkte auf. Um diese auzuwerten, brauchen wir die Zahl der Datenpunkte, d.h., die Zahl der Elemente einer Liste. `Length` liefert den gewünschten Wert.

```
n = Length[x]
```

```
10
```

Jetzt können wir die Formeln aus dem Skript verwenden, um Steigung, Achsenabschnitt und die zugehörigen Fehler der Regressionsgeraden zu ermitteln. Das Summenzeichen wurde aus einer Palette eingefügt, die sich beim Starten von Mathematica automatisch öffnet. Ist dies nicht der Fall, wählen Sie aus dem Menu **File→Palettes→BasicInput**.

$$S = \sum_{i=1}^{n} \frac{1}{(\Delta y[[i]])^2} * \sum_{i=1}^{n} \frac{(x[[i]])^2}{(\Delta y[[i]])^2} - \left(\sum_{i=1}^{n} \frac{x[[i]]}{(\Delta y[[i]])^2} \right)^2$$

```
1395.94
```

$$a0 = \frac{1}{S} * \left(\sum_{i=1}^{n} \frac{(x[[i]])^2}{(\Delta y[[i]])^2} * \sum_{i=1}^{n} \frac{y[[i]]}{(\Delta y[[i]])^2} - \sum_{i=1}^{n} \frac{x[[i]]}{(\Delta y[[i]])^2} * \sum_{i=1}^{n} \frac{x[[i]] * y[[i]]}{(\Delta y[[i]])^2} \right)$$

```
3.01566
```

$$b0 = \frac{1}{S} * \left(\sum_{i=1}^{n} \frac{1}{(\Delta y[[i]])^2} * \sum_{i=1}^{n} \frac{x[[i]] * y[[i]]}{(\Delta y[[i]])^2} - \sum_{i=1}^{n} \frac{x[[i]]}{(\Delta y[[i]])^2} * \sum_{i=1}^{n} \frac{y[[i]]}{(\Delta y[[i]])^2} \right)$$

```
1.03634
```

$$\Delta a0 = Sqrt\left[\frac{1}{S} * \sum_{i=1}^{n} \frac{(x[[i]])^2}{(\Delta y[[i]])^2} \right]$$

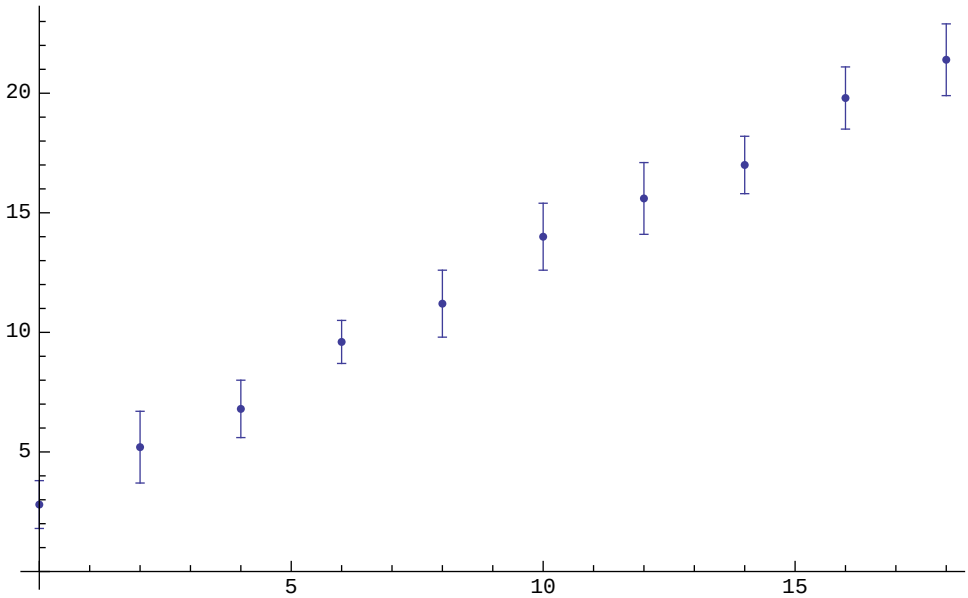
```
0.675303
```

$$\Delta b0 = Sqrt\left[\frac{1}{S} * \sum_{i=1}^{n} \frac{1}{(\Delta y[[i]])^2} \right]$$

```
0.0685982
```

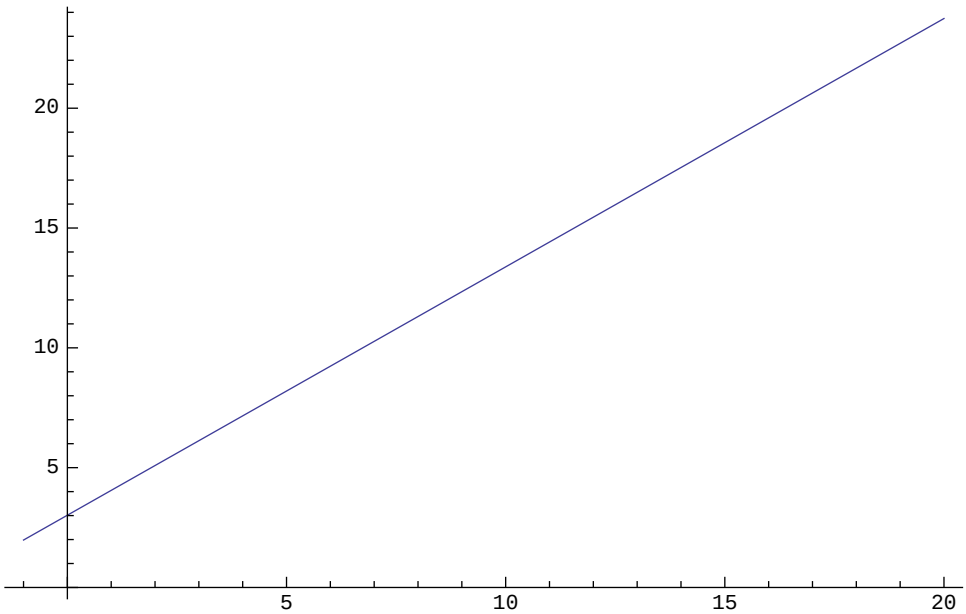
Nun sind alle Paramter berechnet und wir können das Ergebnis grafisch ausgeben. Die Messpunkte werden mit dem Befehl `ErrorListPlot` ausgegeben:

```
ErrorListPlot[linear]
```



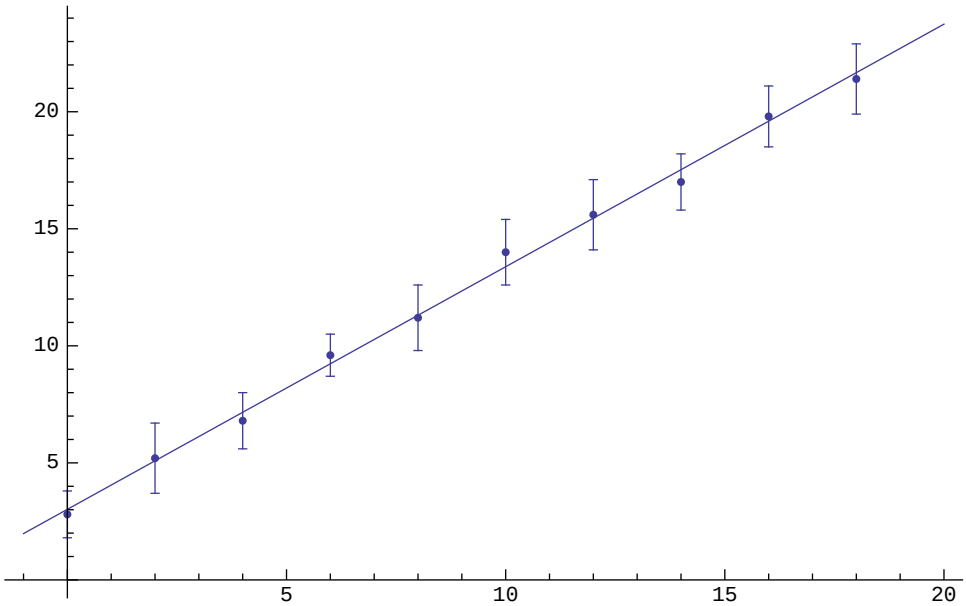
Die Ausgleichsgerade kann mit dem Befehl `Plot` gezeichnet werden. Das erste Argrument dieser Funktion ist die zu plottende Funktion, das zweite eine Liste, die Laufvariable und die Grenzen der x-Achse beinhaltet. x kann nicht als Laufvariable verwendet werden, weil wir schon eine Liste gleichen Namens definiert haben. Probieren Sie es aus!

```
Plot[a0 + b0 t, {t, -1, 20}]
```



Jetzt müssen wir nur noch beide Graphen zusammen anzeigen lassen. Hierzu verwenden wir den Befehl `Show`, der als Argrument eine Liste von Grafik-Befehlen erhält und auch dafür sorgt, dass alle Graphen gleich skaliert werden.

```
Show[{ErrorListPlot[linear], Plot[ a0 + b0 t, {t, -1, 20}]}]
```



- **Lineare Regession als wiederverwendbare Funktion**

Die Aufgabe, Lineare Regression auf einen Satz Werte anzuwenden, haben wir im vorigen Abschnitt gelöst. Wenn wir das Ganze nun auf eine andere Datei anwenden wollen, müssen wir die Schritte alle einzeln wiederholen. Oder besser das Notebook unter einem anderen Namen speichern, den Dateinamen der Messwerte ändern und das Notebook neu auswerten. Sie könnten so verfahren und diesen Abschnitt überspringen, wenn Sie nicht weiter in Mathematica eindringen wollen. Sinnvoller ist es aber, eine Funktion zu definieren, der die Werte übergeben werden und die dann alle Berechnungen durchführt.

Funktionen, die irgendwo aufgerufen werden, dürfen keine eventuell schon vorhandenen Variablen überschreiben. Deshalb ist die Verwendung von lokalen Variablen unerlässlich. Lokale Variablen sind Variablen, die nur innerhalb eines gewissen Bereiches - hier einer Funktion - gelten.

Im Gegensatz zu anderen Programmiersprachen sind innerhalb von Funktionen definierte neue Variable in Mathematica grundsätzlich global. Dies ist eine gefährliche Fehlerquelle!

Um Variablen als lokal zu deklarieren wird der Befehl `Module` verwendet. Erstes Argrument ist eine Liste der lokalen Variabeln, das zweite die Anweisungen, die ausgeführt werden sollen. Die einzelnen Anweisungen werden mit Semikolons getrennt. Das Semikolon unterdrückt gleichzeitig die Ausgabe des jeweiligen Befehles. Vergessen Sie das Semikolon, erhalten Sie eine auf den ersten Blick nicht unbedingt nachvollziehbare Fehlermeldung. Probieren Sie es aus, damit sie später nicht verzweifeln!
Beginnen wir mit einem einfachen Beispiel, die Übergabe von Parametern haben wir ja schon kennen gelernt:

```
EinfachesBeispiel[a_, b_] := Module[{summe, mittel},
      summe = a + b;
  mittel = summe / 2
 ]
```

Mit dieser Funktion kann man tatsächlich den Mittelwert zweier Zahlen berechnen:

```
EinfachesBeispiel[3, 5]
```

```
4
```

Die lokalen Variablen existieren nach diesem Aufruf nicht mehr. (Hier platzsparend als Liste ausgegeben):

```
{summe, mittel, a, b}
```

```
{summe, mittel, a, b}
```

Auch eine vor dem Funktionsaufruf definierte globale Variable wird nicht überschrieben:

```
summe = 12
```

```
12
```

```
EinfachesBeispiel[2, 6]
```

```
4
```

```
summe
```

```
12
```

Schreiten wir also zur Tat und bauen wir unser Beispiel zu einer Funktion um. Als Parameter übergeben wir unserer Funktion die aus der Datei eingelesene Liste, dann folgt die Deklaration der lokalen Variabeln, die Summenindizes sind dabei automatisch lokal. Die restlichen Zeilen können wir unverändert weiter verwenden. Dabei helfen die Funktionen Kopieren und Einfügen. Lediglich ein paar Semikolons müssen ergänzt werden. Nicht mehr benötigte Teile des Notebooks können Sie löschen, in dem Sie die Zellen am rechten Rand mit der Maus markieren und DEL drücken. Durch dieses Löschen vergisst *Mathematica* Ihre Eingaben jedoch noch nicht. Das können Sie nur durch Beenden des Kernels und erneutes Auswerten des Notebooks erreichen. Deshalb sollten Sie dies stets tun bevor Sie eine mit *Mathematica* erstellte Lösung zum Abgeben ausdrucken.

```
LinReg[daten_] := Module[{x, y, Δ y, S, Δ a0, Δ b0, a0, b0, n},
  n = Length[daten];
  x = daten[[All, 1]];
  y = daten[[All, 2]];
  Δ y= daten[[All, 3]];
  S = Sum[1/(Δ y[[i]])^2, {i, 1, n}] * Sum[(x[[i]])^2/(Δ y[[i]])^2, {i, 1, n}] - (Sum[x[[i]]/(Δ y[[i]])^2, {i, 1, n}])^2;
  a0 = 1/S * (Sum[(x[[i]])^2/(Δ y[[i]])^2, {i, 1, n}] * Sum[y[[i]]/(Δ y[[i]])^2, {i, 1, n}] - Sum[x[[i]]/(Δ y[[i]])^2, {i, 1, n}] * Sum[x[[i]] * y[[i]]/(Δ y[[i]])^2, {i, 1, n}]);
  b0 = 1/S * (Sum[1/(Δ y[[i]])^2, {i, 1, n}] * Sum[x[[i]] * y[[i]]/(Δ y[[i]])^2, {i, 1, n}] - Sum[x[[i]]/(Δ y[[i]])^2, {i, 1, n}] * Sum[y[[i]]/(Δ y[[i]])^2, {i, 1, n}]);
  Δ a0= Sqrt[1/S * Sum[(x[[i]])^2/(Δ y[[i]])^2, {i, 1, n}]];
  Δ b0= Sqrt[1/S * Sum[1/(Δ y[[i]])^2, {i, 1, n}]]
 ]
```

Wenden wir die Funktion nun auf unseren Datensatz an, erhalten wir:

```
LinReg[linear]
```

```
0.0685982
```

Das ist nicht ganz das, was wir uns vorgestellt haben. Es wird nur das Ergebnis der letzten Anweisung ausgegeben. Um mehrere Variablen in übersichtlicher Form auszugeben, unterdrücken wir zunächst auch die Ausgabe der letzten Anweisung mit einem Semikolon und erweitern die Prozedur um die folgenden Zeilen:

```
Print["Berechnung der Fehler aus den Eingangsfehlern"];
Print[TableForm[{{"", "Wert", "Fehler"},
    {"Steigung b", b0, Δ b0}, {"Achsenabschnitt a", a0, Δ a0}}]];
```

Der Befehl Print erzeugt auch dann eine Ausgabe, wenn er mit einem Semikolon abgeschlossen wird.

Als Ausgabe erhalten wir dann:

```
LinReg[linear]
```

```
Berechnung der Fehler aus den Eingangsfehlern

                 Wert     Fehler
Steigung b       1.03634  0.0685982
Achsenabschnitt a 3.01566  0.675303
```

Um auf die von unserer Funktion berechneten Werte in folgenden Rechnungen einfach zugreifen zu können, lassen wir die Funktion mit einer zusätzlichen Zeile eine Liste von Ersetzungsregeln zurückgeben. Beachten sie, dass die Anweisung, deren Ausgabe der zurückgegebene Funktionswert sein soll, nicht mit einem Semikolon abgeschlossen werden darf und folglich als letzte stehen muss.

```
{b → b0, Δ b→ Δ b0, a → a0, Δ a→ Δ a0}
```

■ Anwendung der Funktion

Um die Funktion anzuwenden und mit den berechneten Werten weiterzurechnen, wird die von der Funktion zurückgegebene Liste in einer Variabeln gespeichert.

```
regerg = LinReg[linear]
```

```
Berechnung der Fehler aus den Eingangsfehlern

                  Wert    Fehler
Steigung b        1.03634 0.0685982
Achsenabschnitt a 3.01566 0.675303
```

```
{b → 1.03634, Δ b→ 0.0685982, a → 3.01566, Δ a→ 0.675303}
```

Auf einzelne der berechneten Werte können wir jetzt zugreifen, indem wir ausnutzen, dass `regerg` eine Liste von Ersetzungsregeln ist, und daher a, Δ a,b und Δ bdurch Anwendung der Ersetzregeln mit `/. regerg` durch ihre Werte ersetzt werden.

```
a /. regerg
```

```
3.01566
```

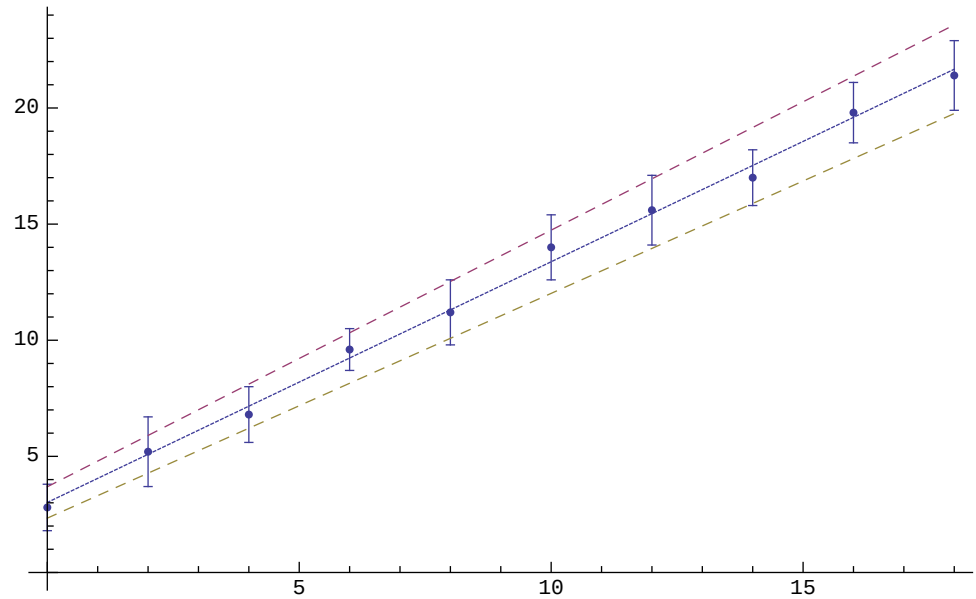
oder etwas komplizierter:

```
a + b /. regerg
```

```
4.052
```

Ausgleichs- und Grenzgeraden können dann wie folgt erzeugt werden:

```
Show[ErrorListPlot[linear],
  Plot[Evaluate[
   {b t + a, (b + Δ b) t + a + Δ a, (b - Δ b) t + a - Δ a} /. regerg], {t, 0, 18},
   PlotStyle → {Dashing[{.0}], Dashing[{0.01}], Dashing[{.01}]} ] ]
```



Der Befehl `Evaluate` ist an dieser Stelle notwendig, um zu erzwingen, dass erst die Ersetzung vorgenommen wird und dann die Stützstellen der Kurven berechnet werden. Lässt man ihn weg, erhält man eine Fehlermeldung, weil Mathematica bei Abarbeitung des Plotbefehls zunächst die zu plottende Funktion kompiliert, um die Rechnung zu beschleunigen. Dies ist mit Ersetzungsanweisungen nicht möglich.
Der Parameter `PlotStyle` dient dazu, die Geraden unterscheidbar zu machen. Details finden Sie in der Hilfe.

■ Linearisierte Exponentialfunktionen

Lesen wir zunächst eine Datei mit Messwerten, die einem exponentiellen Gesetz folgen, ein. Schön wäre es, wenn die Werte dabei gleich übersichtlich ausgegeben werden könnten. Dies gelingt mit dem schon zuvor gebrauchten Befehl `TableForm` - diesmal als nachgestellter Operator.

```
exponential = ReadList["exp.txt", {Real, Real, Real}] //
  TableForm[#, TableHeadings → {{}, {"x", "y", "Δ y"}}] &
```

x	y	Δ y
0.	82.	1.
2.	56.	1.
4.	37.	1.
6.	26.	1.
8.	18.	1.
10.	11.	1.
12.	9.	1.
14.	5.	1.
16.	3.	1.

Versucht man weiter zu arbeiten, stellt man schnell fest, dass man auf die einzelnen Listenelemente nicht mehr richtig zugreifen kann. Dies liegt daran, dass die Variable `exponential` nun die formatierte Tabelle enthält. Durch richtiges Klammern gelingt es, der Variablen die unformatierte Liste zuzuweisen und dennoch eine formatierte Ausgabe zu erzeugen:

```
(exponential = ReadList["exp.txt", {Real, Real, Real}] ) //
 TableForm[#, TableHeadings → {{}, {"x", "y", "Δ y"}}] &
```

x	y	Δ y
0.	82.	1.
2.	56.	1.
4.	37.	1.
6.	26.	1.
8.	18.	1.
10.	11.	1.
12.	9.	1.
14.	5.	1.
16.	3.	1.

Der Funktion `LinReg` werden nun nicht die Werte direkt sondern die logarithmierten Werte mit entsprechend umgerechneten Fehlern übergeben. Das dritte Argument der `Replace`-Anweisung gibt dabei an, dass nur bis zur zweiten Schachtelungsebene der Liste ersetzt werden soll.

```
(linearisiert = Replace[exponential,
    {xi_, yi_, Δ yi_} → {xi, Log[yi], (1 / yi) * Δ yi}, 2]) //
 TableForm[#, TableHeadings → {{}, {"x", "ln(y)", "Δ ln(y) = 1/y Δ y"}}] &
```

x	ln(y)	Δ ln(y) = 1/y Δ y
0.	4.40672	0.0121951
2.	4.02535	0.0178571
4.	3.61092	0.027027
6.	3.2581	0.0384615
8.	2.89037	0.0555556
10.	2.3979	0.0909091
12.	2.19722	0.111111
14.	1.60944	0.2
16.	1.09861	0.333333

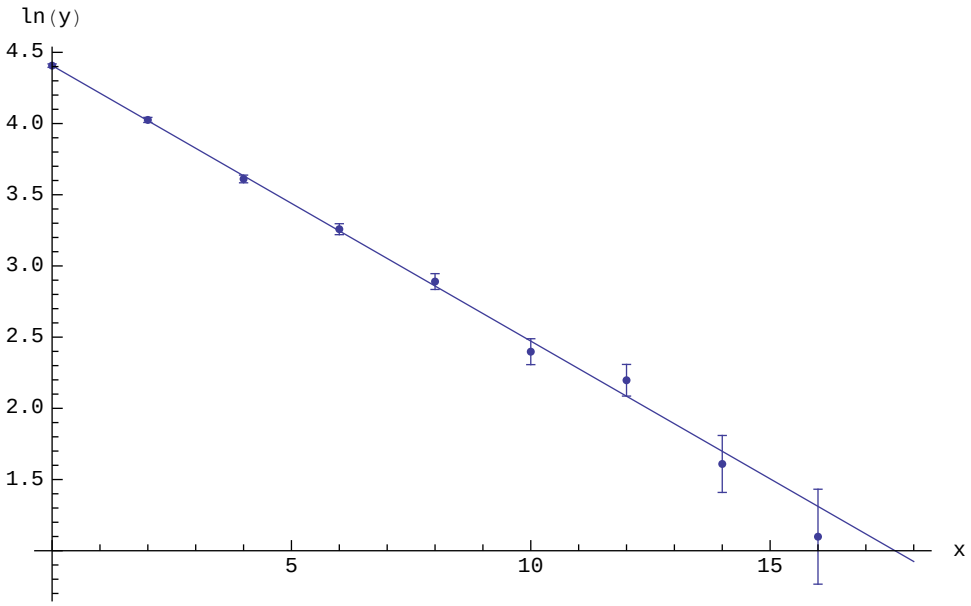
```
experg = LinReg[linearisiert]
```

```
Berechnung der Fehler aus den Eingangsfehlern

                 Wert      Fehler
Steigung b       -0.193466 0.00365475
Achsenabschnitt a 4.40686   0.010881
```

```
{b → -0.193466, Δ b→ 0.00365475, a → 4.40686, Δ a→ 0.010881}
```

```
Show[
 ErrorListPlot[linearisiert, AxesLabel → {"x", "ln(y)"}],
 Plot[Evaluate[b t + a /. experg], {t, 0, 18}]
]
```



Ergebnis

Wenn Sie an diesem Punkt angekommen sind, sollten Sie ein Notebook vor sich haben, dass etwa wie folgt aussieht:

```
Needs["ErrorBarPlots`"]

LinReg[daten_] := Module[{x, y, Δ y, S, Δ a0, Δ b0, a0, b0, n},
  n = Length[daten];
  x = daten[[All, 1]];
  y = daten[[All, 2]]; Δ y = daten[[All, 3]];
  S = Σ_{i=1}^{n} 1/(Δ y[[i]])^2 * Σ_{i=1}^{n} (x[[i]])^2/(Δ y[[i]])^2 - (Σ_{i=1}^{n} x[[i]]/(Δ y[[i]])^2)^2;

  a0 = 1/S * (Σ_{i=1}^{n} (x[[i]])^2/(Δ y[[i]])^2 * Σ_{i=1}^{n} y[[i]]/(Δ y[[i]])^2 - Σ_{i=1}^{n} x[[i]]/(Δ y[[i]])^2 * Σ_{i=1}^{n} x[[i]] * y[[i]]/(Δ y[[i]])^2);
  b0 = 1/S * (Σ_{i=1}^{n} 1/(Δ y[[i]])^2 * Σ_{i=1}^{n} x[[i]] * y[[i]]/(Δ y[[i]])^2 - Σ_{i=1}^{n} x[[i]]/(Δ y[[i]])^2 * Σ_{i=1}^{n} y[[i]]/(Δ y[[i]])^2);
  Δ a0 = Sqrt[1/S * Σ_{i=1}^{n} (x[[i]])^2/(Δ y[[i]])^2];
  Δ b0 = Sqrt[1/S * Σ_{i=1}^{n} 1/(Δ y[[i]])^2];
  Print["Berechnung der Fehler aus den Eingangsfehlern"];
  Print[TableForm[{{"", "Wert", "Fehler"},
     {"Steigung b", b0, Δ b0}, {"Achsenabschnitt a", a0, Δ a0}}];
  {b → b0, Δ b → Δ b0, a → a0, Δ a → Δ a0}
 ]
```

- **Auswertung lineare Funktion**

```
linear = ReadList["linear.txt", {Real, Real, Real}]
```

```
{{0., 2.8, 1.}, {2., 5.2, 1.5}, {4., 6.8, 1.2},
 {6., 9.6, 0.9}, {8., 11.2, 1.4}, {10., 14., 1.4}, {12., 15.6, 1.5},
 {14., 17., 1.2}, {16., 19.8, 1.3}, {18., 21.4, 1.5}}
```

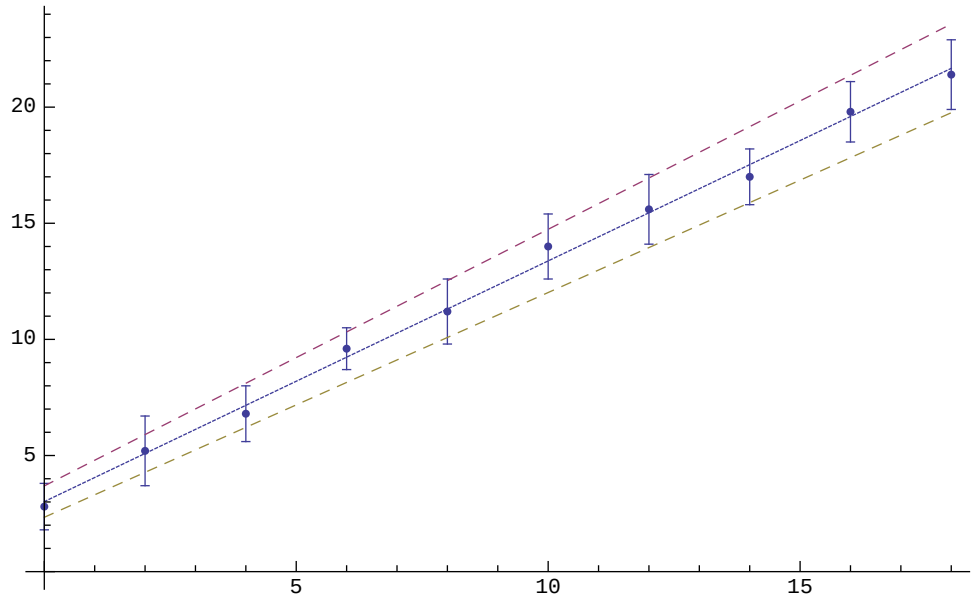
```
regerg = LinReg[linear]
```

```
Berechnung der Fehler aus den Eingangsfehlern

                  Wert    Fehler
Steigung b        1.03634 0.0685982
Achsenabschnitt a 3.01566 0.675303
```

$\{b \to 1.03634, \Delta b \to 0.0685982, a \to 3.01566, \Delta a \to 0.675303\}$

```
Show[ErrorListPlot[linear],
  Plot[Evaluate[
   {b t + a, (b + Δ b) t + a + Δ a, (b - Δ b) t + a - Δ a} /. regerg], {t, 0, 18},
  PlotStyle → {Dashing[{.0}], Dashing[{0.01}], Dashing[{.01}]} ] ]
```



■ Auswertung Exponentialfunktion

```
(exponential = ReadList["exp.txt", {Real, Real, Real}] ) //
 TableForm[#, TableHeadings → {{}, {"x", "y", "Δ y"}}] &
```

x	y	Δ y
0.	82.	1.
2.	56.	1.
4.	37.	1.
6.	26.	1.
8.	18.	1.
10.	11.	1.
12.	9.	1.
14.	5.	1.
16.	3.	1.

```
(linearisiert = Replace[exponential,
    {xi_, yi_, Δ yi_} → {xi, Log[yi], (1 / yi) * Δ yi}, 2]) //
 TableForm[#, TableHeadings → {{}, {"x", "ln(y)", "Δ ln(y) = 1/y Δ y"}}] &
```

x	ln(y)	$\Delta \ln(y) = \frac{1}{y}\Delta y$
0.	4.40672	0.0121951
2.	4.02535	0.0178571
4.	3.61092	0.027027
6.	3.2581	0.0384615
8.	2.89037	0.0555556
10.	2.3979	0.0909091
12.	2.19722	0.111111
14.	1.60944	0.2
16.	1.09861	0.333333

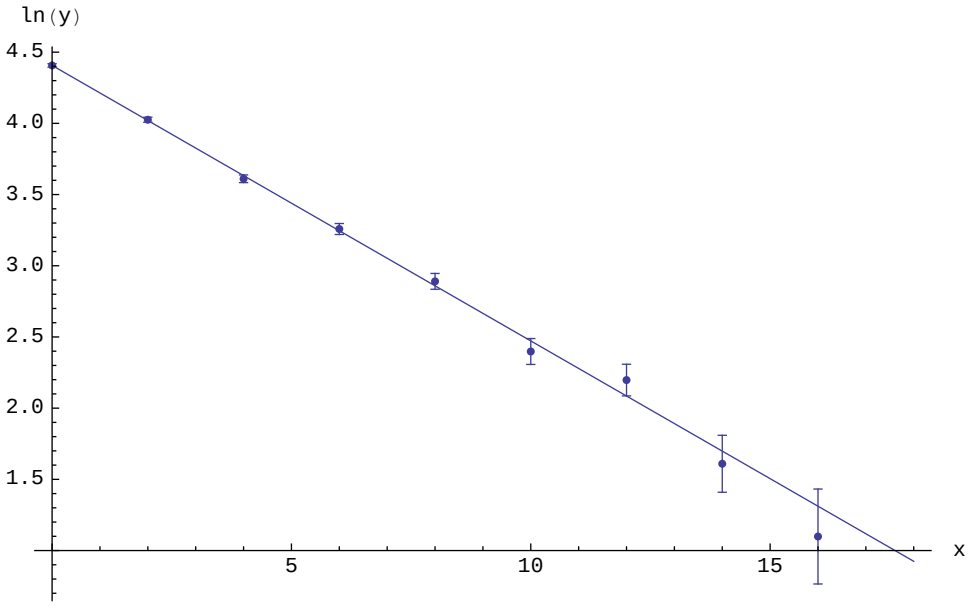
```
experg = LinReg[linearisiert]
```

```
Berechnung der Fehler aus den Eingangsfehlern

                  Wert      Fehler
Steigung b        -0.193466 0.00365475
Achsenabschnitt a 4.40686   0.010881
```

```
{b → -0.193466, △ b→ 0.00365475, a → 4.40686, △ a→ 0.010881}
```

```
Show[
 ErrorListPlot[linearisiert, AxesLabel → {"x", "ln(y)"}],
 Plot[Evaluate[b t + a /. experg], {t, 0, 18}]
]
```



Literatur:

Die beiden folgenden Titel sind bei der Auskunft in der Fachbereichsbibliothek erhältlich. Sie können im Rahmen den Kurzausleihe für einen Tag (z.B. in den Rechnerraum), über Nacht oder übers Wochenende ausgeliehen werden.

Wolfram, Stephen: *Mathematica book*. 4 Aufl. 1999
Gaylord, Richard J. et al: *Programming with Mathematica*. 2. Aufl 1996

Das *Mathematica book* enthält eine gut lesbare Einführung, die vor allem das Lösen kleinerer mathematischer Probleme ohne eigentliche Programmierung erklärt. Ausserdem enthält es eine ausführliche Beschreibung der Befehle von *Mathematica* und der meisten Standard-Packages. Das Buch ist auch im Volltext über die Hilfe-Funktion zugänglich.

Programming with Mathematica richtet sich an völlige Anfänger sowohl im Bezug auf Mathematica als auch im Hinblick auf allgemeine Programmiertechniken. Es führt gleichzeitig Schritt für Schritt in Mathematica und in elementare Programmiertechniken ein.