Erste Schritte mit MATLAB

P1 — Praktikum

1. Februar 2019

Motivation und Ziele

MATLAB ist ein Programm, welches ursprünglich zur Matrizenrechnung entwickelt wurde. Inzwischen hat sich MATLAB zu einer leistungsfähigen Software für mathematische Berechnungen entwickelt und findet insbesondere auch in vielen physikalischen Arbeitsgruppen Anwendung.

In diesem kurzen Skript können nicht alle Anwendungen die MATLAB bietet dargestellt werden, sondern das Augenmerk liegt auf einigen Grundlagen, die bereits in den physikalischen Praktika gewinnbringend eingesetzt werden können. Insbesondere wird erklärt, wie man ausgehend von experimentell erhaltenen Datensätzen statistische Verteilungen charakterisieren und eine Funktion an Daten anpassen kann. Ein Schwerpunkt wird auch die graphische Darstellung der Ergebnisse sein. Für eine vertiefte Auseinandersetzung mit dem Programm sei auf zahlreiche Bücher und im Internet vorhandene Skripte verwiesen. Die in MATLAB integrierte Hilfe hält ebenfalls sehr gute Erklärungen und Beispiele zu den Befehlen und Anwendungen bereit.

In MATLAB definierte Befehle können weitgehend auch in dem Open-Source-Programm *Octave* ausgeführt werden. Dieses ist im Internet kostenlos verfügbar, ist jedoch nicht so komfortabel wie MATLAB zu bedienen.

Mittlerweile wird auch die Programmiersprache Python zusammen mit seinen Modulen Numpy, Scipy, Matplotlib und Pandas als MATLAB-Ersatz in vielen Bereichen eingesetzt. Viele Befehle sind auch dort ähnlich oder identisch mit denjenigen aus MATLAB. Während in der Industrie oft noch MATLAB eingesetzt wird, kommt in der universitären Forschung immer häufiger auch Python zum Einsatz. Nutzen Sie daher die Gelegenheit, mit MATLAB das computergestützte Rechnen mit Vektoren und Matrizen kennenzulernen.

Inhaltsverzeichnis

1 Einleitung

Im Versuch STV werden nicht nur statistische Verteilungen betrachtet, sondern es wird auch ein Einblick in die Möglichkeit des rechnergestützten Auswertens von Messdaten gegeben. Als Hilfsmittel wird das Programm MATLAB verwendet. In diesem Skript werden einige Grundlagen des Programms dargestellt, die teilweise über die für den Versuch STV notwendigen Befehle hinaus gehen. Die sehr lesenswerten, aber für diesen Versuch nicht notwendigen Kapitel werden mit „*“ gekennzeichnet. Sie sollten dieses Skript, bevor Sie den Versuch STV durchführen, gut durcharbeiten. Es ist auch sehr empfehlenswert, die vorgestellten Beispiele entweder mit Octave am privaten Rechner oder mit MATLAB in Raum 310 des Praktikums selbst auszuprobieren. Die Version für Studierende ist im Internet zu erwerben. Dazu sei auf die Homepage des Herstellers *www.mathworks.de* bzw. *www.octave.org* verwiesen.

Zuerst werden nun einige allgemeine Dinge zu dem Programm dargelegt, um anschließend von mathematischen zu den physikalischen und praktikumsspezifischen Anwendungen zu kommen.

2 Benutzeroberfläche

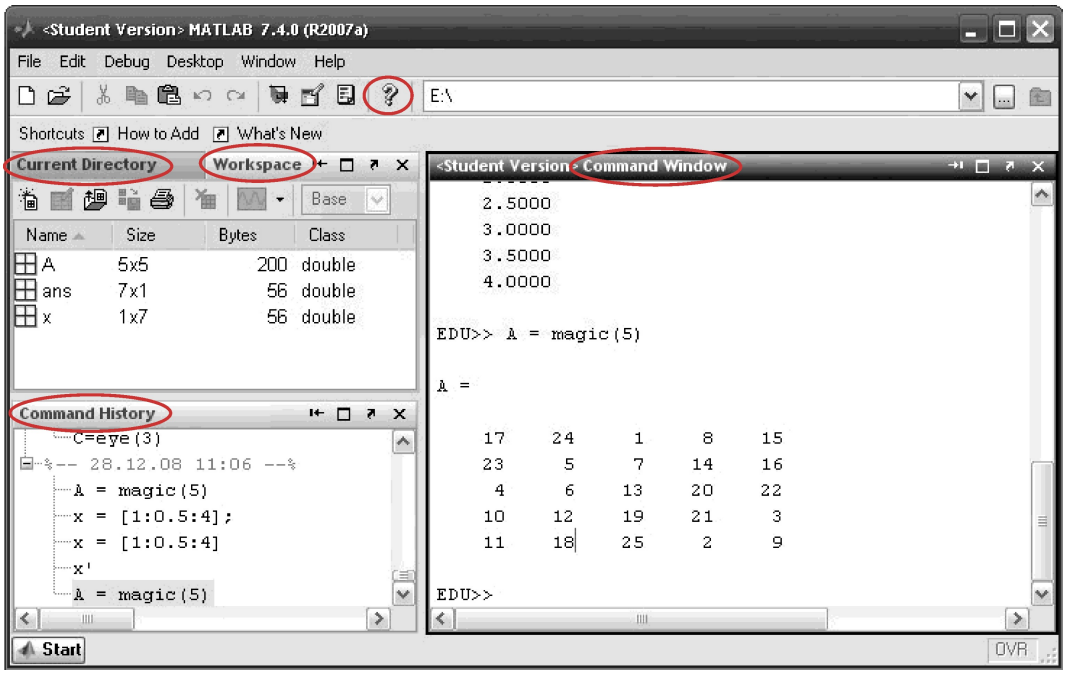

Abbildung 1: Benutzeroberfläche von MATLAB. Rot eingekreist sind die wichtigsten Komponenten. Hinter dem Fragezeichen verbirgt sich die MATLAB-Hilfe.

Nach dem Starten von MATLAB, erscheint die in Abbildung 1 dargestellte Standard-Benutzeroberfläche, die in drei Fenster aufgeteilt ist. Das rechte Fenster ist das „Command Window“. Dort können hinter dem Prompt „>>“ (in der Version für Studierende „EDU>>“) die einzelnen

MATLAB-Befehle eingegeben werden. Im oberen linken Fenster mit dem Namen „Current Directory“ kann man auf alle Dateien im aktuellen Verzeichnis zugreifen. Direkt daneben befindet sich der Reiter „Workspace“. Dort werden alle aktuellen Variablen mit Größe und Typ angezeigt, die bisher erzeugt oder geladen wurden. Im Fenster „Command History“ werden die letzten Befehle angezeigt, die im Command Window eingegeben worden sind. Durch Doppelklicken auf die Befehle, können diese sofort erneut ausgeführt werden.

3 MATLAB-Hilfe

In den folgenden Kapiteln werden zwar viele Befehle erläutert, es wird jedoch häufiger vorkommen, dass man die genaue Eingabesyntax einiger Befehle vergessen hat. Für diesen Fall kann man die MATLAB-Hilfe aufrufen und den Befehlsnamen eingeben. Die Hilfe erläutert noch einmal, was man mit dem Befehl machen kann und in welcher Art und Weise die Syntax einzugeben ist. Ein Beispiel dafür ist in Abbildung 2 zu sehen. Am Ende dieser Hilfe-Dokumentation steht meist ein sehr hilfreiches Beispiel.

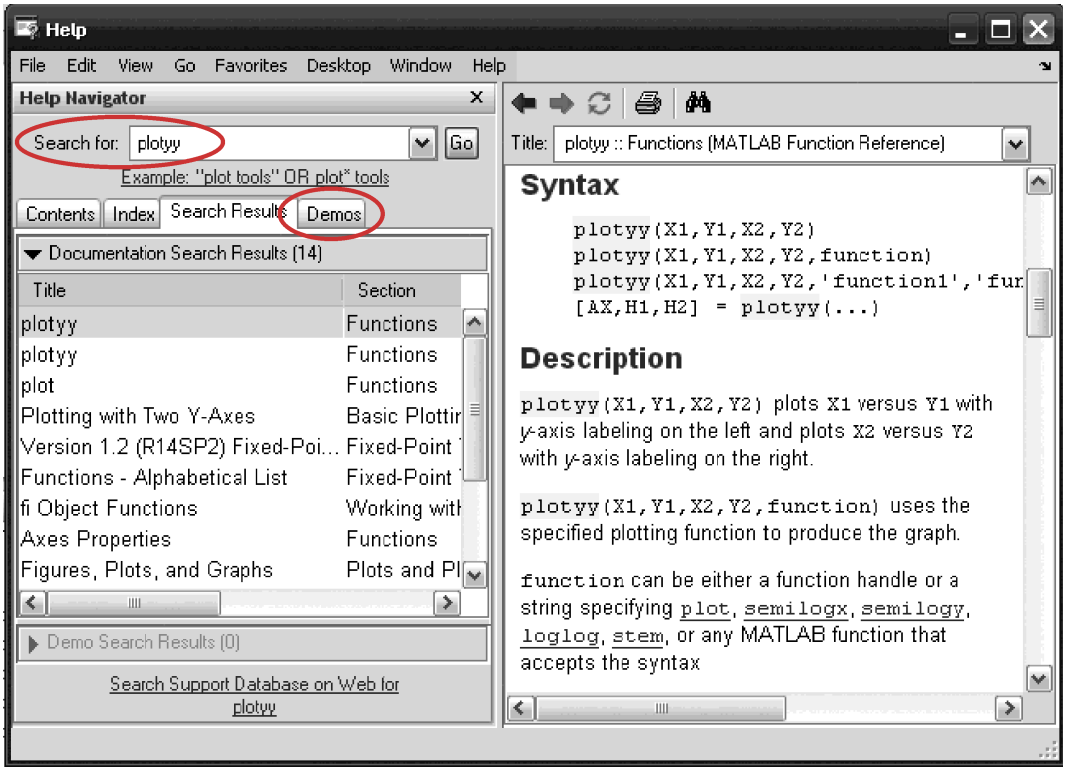

Abbildung 2: Hier ist das Fenster der integrierten MATLAB-Hilfe abgebildet. Hinter dem Reiter „Demo“ sind weitere Beispiele und Erklärungen zu bestimmten Anwendungen hinterlegt.

Wenn man einen Befehl ausführen möchte, aber nicht genau weiß, ob und wie er definiert ist, empfiehlt es sich den englischen Begriff dafür in die MATLAB-Hilfe einzugeben. Damit kommt man bei einfachen Anwendungen schon recht weit.

Desweiteren sind auf der Homepage des Herstellers einige Demo-Videos hinterlegt, die man sich auch einmal anschauen sollte. Dazu muss man im Abschnitt „Forschung und Lehre“ den Link „Für Studierende“ und anschließend den Abschnitt „Tutorials“ auswählen. Das erste unter „Watch Introductory Videos“ aufgeführte Video „Getting Started with MATLAB“ passt sehr gut zu den in Abschnitt 4 behandelten Themen. Es lässt sich direkt unter folgender URL abrufen: `https://de.mathworks.com/videos/getting-started-with-matlab-68985.html?s_cid=learn_vid`.

4 Erstellen von Variablen

Dieser Abschnitt beschäftigt sich mit den zentralen Operationen, Variablen zu erstellen und auf die in ihnen enthaltenen Daten zuzugreifen. Wie eingangs erwähnt, wurde MATLAB zur Matrizenrechnung entwickelt. Daher müssen die Daten (Messwerte) auch in Form von Matrizen vorliegen. Variablen sind durch Größe (Dimension der Matrix) und Typ (Art der Einträge) definiert.

4.1 Daten importieren

Daten kann man über Menü `File - Import Data` importieren. Diese erscheinen anschließend im Workspace. Es werden viele gängige Dateiformate unterstützt, wie zum Beispiel auch Excel Dateien. Welche Formate außerdem unterstützt werden, kann man mit dem Befehl `help fileformats` abfragen oder indem man „fileformats“ in die Hilfe eingibt.

Bei Formaten, die MATLAB nicht ohne weiteres importieren kann, kommt der Befehl `dlmread` zum Einsatz. Dieser Befehl kann aus einer Datei die gewünschten Daten auslesen.[1] Im Versuch STV untersuchen Sie natürliche Radioaktivität. Die Messung erfolgt mithilfe eines Steuerrechners. Die zugehörige Software heißt *MAESTRO* und gibt nach einer Konvertierung der Rohdaten statistische Daten mit der Dateiendung *.stat* aus. Da beispielsweise die ersten 5 Zeilen dieser Datei im Beispiel „Listing 1“ der Anleitung zu STV einen Text mit genaueren Informationen zur Messung enthalten, aber noch keine Messwerte, muss man diese Zeilen „abschneiden“. Der Befehl

```
EDU>> A = dlmread ('090108-70.stat',' ',5,0)
```

schneidet aus der Datei *090108-70.stat* die ersten 5 Zeilen aber keine Spalte ab und speichert den Rest unter der Variable *A*. Durch `' '` wird die Art, wie die einzelnen Zahlen voneinander getrennt sind, definiert. Bei der vorhandenen Datei ist dies durch einfache Leerzeichen umgesetzt, dementsprechend wurde im Befehl `dlmread` ein Leerzeichen zwischen den Strichen verwendet.

[1] Die Datei muss hierzu im ASCII-Format gespeichert sein. Dieses Format wird manchmal auch als „reiner Text“ bezeichnet und hat häufig die Endung .txt oder .dat.

4.2 Daten manuell eingeben

Daten können auch manuell eingegeben werden. Mit dem folgenden Befehl weist man einer Variable mit Namen x den Wert 12 zu:

```
EDU>> x = 12

x =

    12
```

Beendet man die Eingabe mit *Enter*, so gibt MATLAB sofort das Ergebnis aus. Aus diesem Grund steht unter der Eingabezeile nochmals der Wert $x = 12$. Oft ist es jedoch unübersichtlich, nach jeder Befehlseingabe einen Wert ausgegeben zu bekommen. Daher unterdrückt man zum Beispiel bei einer Reihe von aufeinanderfolgenden Befehlen die Ausgabe mit einem Semikolon „;“ hinter der Eingabezeile. Der Befehl

```
EDU>> x = 12;
```

speichert unter der Variable x ebenfalls den Wert 12 ab, unterdrückt aber die Ausgabe.

Typ und Größe der so angelegten Variable kann man mit dem Befehl `whos` auslesen:

```
EDU>> whos x
  Name      Size             Bytes  Class

  x         1x1                  8  double
```

4.3 Datentypen (*)

Datentypen, die in MATLAB verwendet werden, sind:

- `int32` – ganze (integer) Zahlen, Länge 4 Bytes = 32 Bit
- `double` – reelle Zahlen, Länge 8 Bytes, doppelte Genauigkeit
- `char` – Textzeichen (charakter), in MATLAB 2 Byte
- `logical` – Wahrheitswert (boolean) - $true$ oder $false$, 1 Byte

Gibt man den Datentyp nicht explizit an, wählt MATLAB automatisch einen passenden Typ.

4.4 Erstellen von Vektoren und Matrizen

Bei vielen physikalischen Versuchen nimmt man mehrere Messwerte in einer vordefinierten Schrittweite auf. Untersucht man zum Beispiel den Zerfall von Barium, misst man eine gewisse Zeit lang in regelmäßigen Abständen die zerfallenden Kerne. Um den Zerfall grafisch auszuwerten, muss man die gezählten Ereignisse gegen die Zeit auftragen. Anstatt die regelmäßigen Messzeitpunkte einzeln eintippen zu müssen, ist es effektiver, zunächst einen Vektor mit dieser definierten Schrittweite zu erstellen. Solch einen Vektor mit gleichmäßigen Abständen nennt man in MATLAB einen gleichförmigen Vektor. Mit diesem Vektor kann MATLAB weiterarbeiten, da ein Vektor eine Matrix mit Spaltendimension 1 ist. Die Messwerte der zerfallenen Kerne bilden einen weiteren Vektor. Nun kann man die beiden Vektoren gegeneinander auftragen.[2]

Gleichförmige Vektoren kann man auf zwei Arten erstellen.

Wenn der Abstand dx zwischen den einzelnen Messpunkten bekannt ist verwendet man den Befehl

```
EDU>> x = [a:dx:b];
```

um einen Vektor x mit Anfangswert a und Endwert b mit Abstand dx zu erstellen. Mit „;“ wird wieder die Ausgabe des Vektors im Command Window unterdrückt. Wenn die Schrittweite zwischen den Punkten 1 ist, kann man dx weglassen und den Vektor einfacher mit $x = [a : b]$ erstellen.

Ist der Abstand zwischen Start- und Endwert nicht bekannt, dafür aber die Anzahl n der Zwischenpunkte, verwendet man die Syntax:

```
EDU>> x = linspace(a,b,n)
```

Es werden n äquidistante Punkte zwischen den Werten a und b erstellt (inklusive den Werten a und b selbst).

Auf diese Weise erhält man einen Zeilenvektor, den man mit „′“ transponieren kann:

```
EDU>> x = x'
```

Hierbei wird die alte Variable x (also der Zeilenvektor) durch den neuen Spaltenvektor ersetzt. Möchte man den Zeilenvektor weiterhin verwenden, müsste man für den Spaltenvektor einen neuen Namen, z. B. y, definieren.

Natürlich kann man beliebige Vektoren auch ganz normal eingeben. Der Zeilenvektor, der aus den Zahlen 1, 2 und 3 besteht, wird in der Mathematik allgemein üblich mit runden Klammern geschrieben.

$x = (1,2,3)$

In MATLAB werden Vektoren (und Matrizen) mit eckigen Klammern eingegeben. Zur Erzeugung des obigen Vektors ist in MATLAB folgende Syntax notwendig:

[2] Näheres dazu finden Sie in Abschnitt 6.1.1 dieses Skriptes.

```
EDU>> x = [1,2,3]

x =

     1     2     3
```

Eingelesene oder von Hand eingegebene Daten können in MATLAB auf zwei Arten in einer Datei gespeichert werden. Entweder man markiert die Variablen im Workspace und speichert sie mit dem Disketten-Symbol unterhalb des Reiters „Workspace“, oder man speichert sie über den Befehl save.

Bei Octave kann man die Variablen (und die darin enthaltenen Daten) ausschließlich über den Befehl speichern. Die Befehlseingabe ist bei MATLAB und Octave identisch.

Die Syntax

```
octave-3.0.3.exe:4> save test6 C D
```

speichert die zuvor in Octave erstellten Variablen C und D in die Datei „test6“ ab. Diese Datei ist, wenn nichts anderes eingestellt wurde, unter dem Pfad zu finden, unter dem Octave installiert wurde.[3] Nachdem das Programm beendet und wieder neu gestartet wurde, sind die Variablen C und D zunächst unbekannt. Durch den Befehl

```
octave-3.0.3.exe:5> load test6
```

können sie geladen und weiter verwendet werden.

5 Matrizen

Bisher ist bekannt, dass Messdaten von MATLAB in Form von Matrizen gespeichert werden können. MATLAB bietet die Möglichkeit auf diese Matrizen auf vielfältige Weise zuzugreifen.

5.1 Erstellen von Matrizen

Die direkte Eingabe von Matrizen funktioniert fast genauso wie bei Vektoren, nur werden die Zeilen zusätzlich durch „;“ getrennt. Ein Beispiel:

```
EDU>> A = [1,2,3;4,5,6;7,8,9]

A =

     1     2     3
     4     5     6
     7     8     9
```

[3] Das Verzeichnis können Sie mit dem bekannten Befehl cd wechseln.

Man kann die Kommas auch weglassen, allerdings kann es bei großen Matrizen, besonders mit Dezimalzahlen als Einträgen, schnell unübersichtlich werden.

Einige spezielle Matrizen sind in MATLAB schon vorprogrammiert. Dies ist praktisch, um die im Abschnitt 5.3 definierten Rechenoperationen ausprobieren zu können.

Zu den vordefinierten Matrizen zählen:

- `zeros(n)` – erzeugt eine quadratische Nullmatrix der Dimension n.
- `ones(n)` – erzeugt eine quadratische Matrix, deren Einträge alle den Wert 1 haben.
- `eye(n)` – erzeugt die Einheitsmatrix.
- `diag(n)` – erzeugt eine Diagonalmatrix aus einem vorgegebenen Vektor.
- `magic(n)` – erzeugt ein magisches Quadrat, also ein Quadrat, bei dem die Summe der Zeilen, Spalten und Diagonalen gleich ist.

Um z. B. das magische Quadrat zu erzeugen, wird der entsprechende Befehl im Command Window eingegeben:

```
EDU>> A = magic(5)

A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Wie auch in der Mathematik üblich, definiert MATLAB ein Element der Matrix aus seiner Zeile und Spalte. Die Bezeichnung $A(m,n)$ meint den Eintrag in der m-ten Zeile und n-ten Spalte der Matrix A.

5.2 Teile von Matrizen auslesen oder überschreiben

In MATLAB ist ein Zugriff auf die gespeicherten Einträge einer Matrix möglich. Anhand der Matrix `magic(5)` soll nun erklärt werden, wie man einen Eintrag der Matrix ändern und bestimmte Bereiche einer Matrix auslesen kann.

Wenn man auf einen bestimmten Eintrag zugreifen möchte, weist man einer Variablen p einen bestimmten Eintrag zu:

```
EDU>> p = A(1,2)

p =

    24
```

Somit ist unter dem Namen p der Eintrag der ersten Zeile und zweiten Spalte, bei $magic(5)$ ist das 24, gespeichert. Diese neue Variable erscheint mit Typ und Größe im Workspace.

Im oberen Beispiel wird der Wert rechts vom Gleichheitszeichen ausgelesen und in die Variable links vom Gleichheitszeichen geschrieben. Durch Vertauschen der Seiten kann ein Eintrag der Matrix überschrieben werden.

```
EDU>> A(1,2) = 1

A =

    17     1     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Der Wert auf der rechten Seite vom Gleichheitszeichen wird auf den Eintrag in der ersten Zeile und zweiten Spalte der Matrix geschrieben. Die 24 wurde also durch eine 1 ersetzt.

Mithilfe des Doppelpunkt-Operators „:“ kann man einen Unterbereich einer Matrix auswählen und z. B. einer neuen Variablen zuweisen:

```
EDU>> B = A(1:5,2:3)

B =

     1     1
     5     7
     6    13
    12    19
    18    25
```

Der ausgewählte Bereich sind hier die Zeilen 1 bis 5 und die Spalten 2 und 3 der Matrix A. Die neue Matrix B besteht aus der zweiten und dritten Spalte der Ausgangsmatrix A. Möchte man entlang einer Dimension alles auswählen, reicht als Eingabe „:“ ohne Grenzen. So würde die folgende Syntax dasselbe Ergebnis liefern:

```
EDU>> B = A(:,2:3)
```

Man kann auch zwei Matrizen (oder Vektoren) zu einer zusammenfügen. Die Eingabe ist dieselbe wie beim Erstellen von Matrizen, nur dass die Einträge keine einzelnen Ziffern sind, sondern ebenfalls Matrizen. Im folgenden Beispiel wird eine Matrix mit 5 Zeilen und 10 Spalten erstellt, in der die Matrix A zweimal nebeneinander steht.

```
EDU>> B = [A,A]

B =

    17     1     1     8    15    17     1     1     8    15
    23     5     7    14    16    23     5     7    14    16
     4     6    13    20    22     4     6    13    20    22
    10    12    19    21     3    10    12    19    21     3
    11    18    25     2     9    11    18    25     2     9
```

Ersetzt man in der Syntax „,“ durch „;“ erhält man eine Matrix mit 10 Zeilen und 5 Spalten, in der die Matrix A zweimal untereinander steht.

```
EDU>> B = [A;A]

B =

    17     1     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
    17     1     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

5.3 Rechnen mit Matrizen

Häufig muss man Daten, die in Vektoren oder Matrizen vorliegen, bearbeiten. Es liegt also Nahe, dass mit Matrizen gerechnet werden kann. Die Rechnenoperationen +, -, *, / und ^ sind teilweise anders als in der Mathematik definiert. So kann man zwei Matrizen zum Beispiel auch komponentenweise multiplizieren, was nicht mit der Matrixmultiplikation in der Mathematik verwechselt werden darf. Aus diesem Grund muss man sich beim Rechnen mit Matrizen genau überlegen, welche Rechenoperation man verwendet.

5.3.1 Operationen + und -

Die Operationen + und - erfolgen komponentenweise und sind nur für Matrizen derselben Dimension definiert. Zu jedem Element der Matrix A wird das Element an derselben Position in Matrix B addiert. In der folgenden Befehlsfolge werden erst zwei Matrizen A und B definiert und anschließend deren Summe berechnet und unter der Variable C abgespeichert. Die verwendeten Matrizen sind:
$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ und $B = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$

```
EDU>> A = [1,2;3,4];
EDU>> B = [1,1;0,1];
EDU>> C = A+B

C =

     2     3
     3     5
```

Desweiteren kann man zu einer Matrix in MATLAB auch nur eine Zahl addieren. Dies ist in der Mathematik nicht definiert. In MATLAB wird in diesem Fall die Zahl zu jedem Element der Matrix addiert. Als Beispiel wird zur Einheitsmatrix der Dimension 3 die Zahl 1 addiert:

```
EDU>> C = eye(3)+1

C =

     2     1     1
     1     2     1
     1     1     2
```

5.3.2 Matrixmultiplikation, Operationen * und .*

Analog zur Addition einer Zahl zu einer Matrix kann man auch eine Matrix mit einer Zahl multiplizieren. Dabei wird jeder Eintrag der Matrix mit dieser Zahl multipliziert. Wird die Matrix A mit der Zahl 2 multipliziert, folgt:

```
EDU>> C = A*2

C =

     2     4
     6     8
```

Multiplikation ist natürlich auch zwischen Matrizen (Vektoren) wie in der Linearen Algebra definiert. Dabei muss man beachten, dass die Spaltendimension der ersten Matrix mit der Zeilendimension der zweiten Matrix übereinstimmt. Andernfalls käme eine Fehlermeldung. Probieren Sie es aus!

Möchte man die Matrizen komponentenweise multiplizieren, erreicht man das durch „.“ vor dem Multiplikationsoperator. Dieses komponentenweise Rechnen funktioniert ebenfalls bei den Operatoren / und ^.

```
EDU>> C = A*B

C =

     1     3
     3     7

EDU>> C = A.*B

C =

     1     2
     0     4
```

5.3.3 Die Operatoren \ und /

Um ein Gleichungssystem der Form $A * X = B$ zu lösen, müsste man B von links mit der zu A inversen Matrix A^{-1} multiplizieren. Das kann in MATLAB durch die Division durch A umgesetzt werden. Je nach dem, wie das Gleichungssystem aufgebaut ist, gibt es zwei Divisionsoperatoren:

- Der \-Operator bezeichnet die Links-Division von Matrizen und
- der /-Operator bezeichnet die Rechts-Division von Matrizen.

Für die Gleichung $A * X = B$ mit A und B wie oben folgt also:

```
EDU>> X = A\B

X =

   -2.0000   -1.0000
    1.5000    1.0000
```

Die Probe bestätigt dieses Ergebnis.

5.3.4 Beispiel: Skalarmultiplikation

Als Beispiel zum Rechnen mit MATLAB eignet sich die Skalarmultiplikation zweier Vektoren. Im folgenden Code werden zunächst zwei 4-dimensionale Spaltenvektoren auf unterschiedliche Art und Weise erzeugt:

```
EDU>> x = [1,2,7,-2]';
EDU>> y = [2:5]';
```

Vektor x entsteht durch Eintragen beliebiger Werte in die Syntax und Vektor y als gleichförmiger Vektor zwischen den Werten 2 und 5 mit Schrittweite 1. Mit „'“ werden die definierten Zeilenvektoren zu Spaltenvektoren. Das Semikolon unterdrückt die Ausgabe der Vektoren im Command Window. Nun möchte man auf die entstandenen Vektoren das Skalarprodukt

anwenden. Dies funktioniert bekanntermaßen indem man jeweils die Einträge an derselben Position der Vektoren multipliziert und dann die erhaltenen Podukte addiert. Im MATLAB wird das mit dem Befehl für komponentenweise Multiplikation `.*` und anschließendes Aufsummieren durch den Befehl `sum` umgesetzt. Im folgender Syntax wird das Skalarprodukt der Variable „prod“ übergeben:

```
EDU>> prod = sum(x.*y)

prod =

    26
```

MATLAB hat natürlich auch einen integrierten Befehl für das Skalarprodukt. Auf Englisch heißt das Skalarprodukt „dot product“, also lautet der entsprechende Befehl auch `dot(x,y)`.

```
EDU>> prod = dot(x,y)

prod =

    26
```

Im Versuch „STV - Statistische Verteilungen“ müssen Sie die Häufigkeiten bestimmter Ereignisse von vielen Einzeldurchläufen zu einer Gesamthäufigkeit der Ereignisse zusammenfassen. Die Daten der Einzeldurchläufe werden dabei in eine Matrix gespeichert. Durch Summieren der einzelnen Spalten ist es möglich, die Gesamthäufigkeit der einzelnen Ereignisse zu bestimmen. Der Befehl für das spaltenweise Aufsummieren ist `sum(data)`.

Um ein Beispiel für die Verwendung dieses Befehls zu geben, ohne auf die Durchführung des Versuchs zu statistischen Verteilungen vorzugreifen, sei auf das in Abschnitt 5.1 definierte magische Quadrat verwiesen. Mit dem Befehl `sum` kann man anschaulich überprüfen, ob die Summe der einzelnen Spalten tatsächlich gleich ist:

```
EDU>> B = sum(A)

B =

    65    65    65    65    65
```

Durch Aufsummieren von A' auf dieselbe Art und Weise, kann man verdeutlichen, dass auch die Summen der Zeilen übereinstimmen.

6 Grafische Darstellung und Analyse

Im Sinne von „Ein Bild sagt mehr als 1000 Worte“ will man Ergebnisse von Messungen und Berechnungen anschaulich darstellen. MATLAB kennt viele Möglichkeiten, Daten zu visualisieren. Welche man verwendet, hängt von der Art der Daten und dem angestrebten Ergebnis ab. Hier wird lediglich die zweidimensionale Darstellung behandelt. Es ist sehr empfehlenswert, mit den hier vorgestellten Funktionen und darüber hinaus auch selbstständig zu experimentieren.

6.1 Plotten und Visualisieren

6.1.1 Ausgabe eines Datensatzes

Zur grafischen Ausgabe eines Datensatzes verwendet man den Befehl `plot`.

Es werden zum Beispiel bei einem Experiment die Temperaturwerte alle 10 Sekunden aufgenommen. Um die Temperatur gegen die Zeit aufzutragen, definiert man sich zunächst einen Vektor x mit äquidistanten Zeitwerten (einen gleichförmigen Vektor). Als nächstes definiert man einen Vektor y, der die zugehörigen Funktionswerte enthält. Die Funktion `plot` trägt die Vektoren gegeneinander auf und verbindet die Punkte durch gerade Linien, wie in Abbildung 3 zu sehen ist.[4]

```
EDU>> x = [0:10:60];
EDU>> y = [0,1,1.5,3,3.5,3,4.5];
EDU>> plot (x,y)
```

6.1.2 Darstellung mathematischer Funktionen

In MATLAB sind viele Standardfunktionen wie $\sin(x)$, $\cos(x)$ und $\exp(x)$ bereits vordefiniert. Es ist einfach, solche Funktionen und Kombinationen aus diesen darzustellen.

Zur grafischen Darstellung von Funktionen einer Veränderlichen, gibt es in MATLAB die Grafikfunktion `fplot`. Dabei muss man den Namen der Funktion und den Definitionsbereich übergeben. Die Syntax

```
EDU>> fplot ('x^2',[-3,3])
```

zeichnet die Funktion $f(x) = x^2$ im Bereich $-3 \leq x \leq 3$ in einem separaten Fenster, dem sogenannten „figure-Window“.

[4] Häufig sollte man die Messpunkte nicht durch gerade Linien verbinden. In Abschnitt 6.1.3 ist aufgeführt, wie man das verhindern kann.

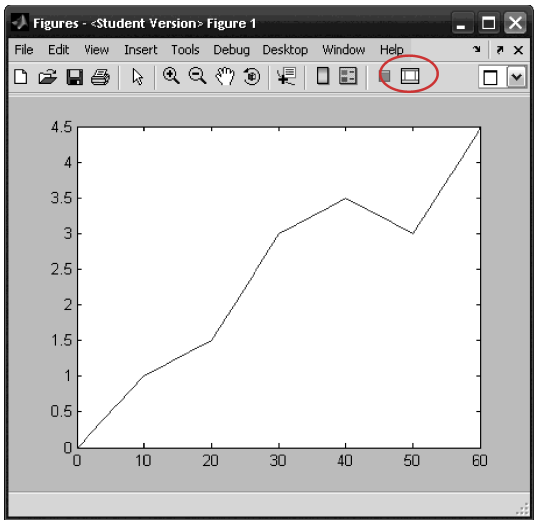

Abbildung 3: Zu sehen ist hier das figure-Window, welches sich nach dem plot-Befehl öffnet. Rot eingekreist ist die Schaltfläche, die man aktivieren muss um Linientyp, -farbe und Marker für die Messwerte verändern zu können.

6.1.3 Einstellen der grafischen Ausgabe

Durch Aktivieren der Schaltfläche „Show Plot Tools and Dock Figure“ (in Abbildung 3 rot eingekreist) wird das figure-Window durch den sogenannten „Property Editor“ erweitert. Dort kann man Linientypen, -farben und Marker für die Messwerte verändern. Hier trennen sich die Wege von Octave und MATLAB. Während bei MATLAB sowohl eine leicht zu bedienende grafische Benutzeroberfläche (GUI - Graphical User Interface) als auch vordefinierte Befehle zur Verfügung stehen, um die Art und Weise der grafischen Darstellung festzulegen, kann man dies bei Octave nur durch die Eingabe von Befehlen verwirklichen.[5] Bei Definition der Darstellung durch Befehle, erweitert man den `plot`-Befehl folgendermaßen:

```
EDU>> plot(x,y,'color_style_marker')
```

Für Linenfarbe, -typ und Symboltyp werden in den Befehl bestimmte Codes eingeben. Welche Codes es gibt, kann man der folgenden Tabelle entnehmen:

Linienfarbe		Linentyp		Symbol	
c	Cyan	-	durchgezogen	o	o-Symbol
m	Magenta	–	gestrichelt	*	*-Symbol
r	Red	:	gepunktet	x	x-Symbol
g	Green	-.	Strich-Punkt-Linie	^	^-Symbol
b	Blue			p	Fünfeck (Pentagramm)
k	BlacK			h	Sechseck (Hexagramm)

[5] Es gibt Open-Source-Projekte, auch für Octave eine GUI zur Verfügung zu stellen. Diese sind jedoch noch in der Entwicklung. Hier können Sie selbst experimentieren, wie Ihnen diese zusagen.

Mit dem folgenden Befehl erhält man beispielsweise eine rote Strich-Punkt Linie bei der die Messwerte durch x-Symbole verdeutlicht werden:

```
EDU>> plot(x,y,'r-.x')
```

In Abschnitt 6.1.1 wurde erwähnt, dass es in der Regel nicht sinnvoll ist, Messwerte durch gerade Linien stückweise zu verbinden. Wenn man im obigen Befehl das Symbol für den Linientyp weglässt, erhält man in der Ausgabe nur die farbigen Symbole für die Messergebnisse, siehe Abbildung 4.

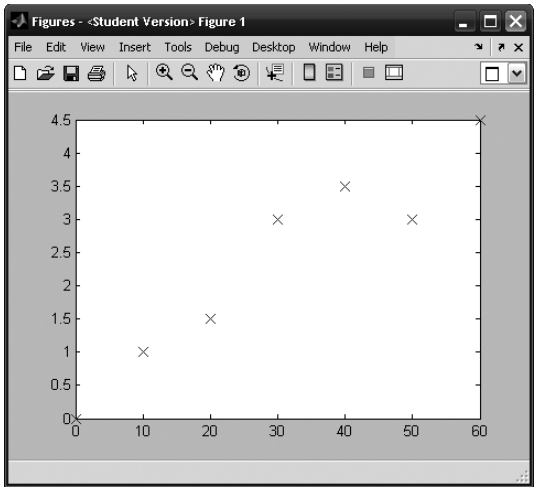

Abbildung 4: Zu sehen ist hier die Ausgabe eines Datensatzes, bei dem die Messpunkte durch Symbole gekennzeichnet, aber nicht durch gerade Linien verbunden sind.

Zu einer korrekten grafischen Darstellung gehören neben Linienart und Kennzeichnung der Messpunkte ebenfalls Titel, Achsenbeschriftung und gegebenenfalls eine Legende. Dies kann man entweder über den „Property Editor“ oder durch Befehlseingabe im Command Window umsetzen. So würde man im obigen Beispiel nach der Eingabe des `plot`-Befehls (und anschließender Ausgabe der Grafik im figure-Window) mit folgendem Befehl fortfahren um dem Diagramm einen Titel zu geben:

```
EDU>> title('Auftragen der Temperatur gegen die Zeit')
```

Ebenso kann mit den Befehlen `xlabel`, `ylabel` und `legend` die Achsenbeschriftung bzw. eine Legende erzeugt werden.

6.1.4 Diagrammtypen

Wie erwähnt verbindet der Befehl `plot` die Messpunkte durch gerade Linien. Dies hebt jedoch häufig nicht die gewünschten Charakteristika hervor. Aus diesem Grund stehen für Vektordaten weitere Diagrammtypen (siehe Abbildung 5) zur Verfügung:

- `bar` – Balkendiagramm
- `stem` – Diskretes-Folgen-Diagramm

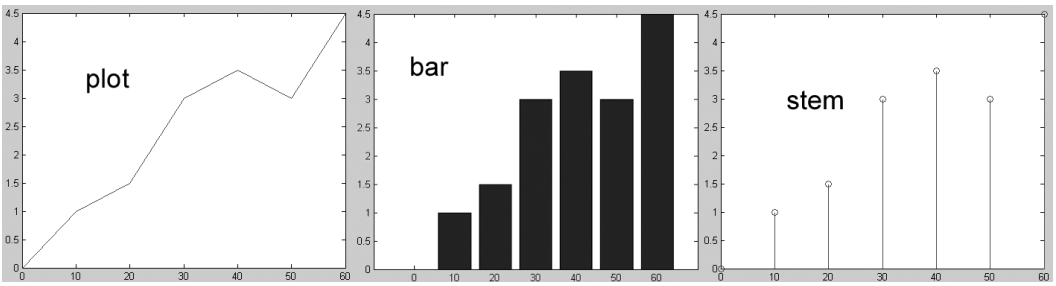

Abbildung 5: Zu sehen sind hier verschiedene Diagrammtypen, in denen die auf Seite 16 definierten Vektoren x und y aufgetragen sind.

Bei dem hier gemachten Beispiel für x und y, kann man einen linearen Zusammenhang der Messwerte erahnen. In diesem Fall kann es sehr hilfreich sein, sich eine Ausgleichsgerade in das Diagramm legen zu lassen und eventuell die Steigung bestimmen zu lassen. Das Analysieren von Daten mithilfe von Ausgleichskurven wird im folgenden Abschnitt behandelt.

6.2 Ausgleichskurven

MATLAB ist ein sehr vielschichtiges Progamm, aufgebaut aus mehreren Toolboxen für unterschiedliche Anwendungsbereiche. Mit der Standard-Version von MATLAB kann man Ausgleichskurven durch den Befehl `polyfit` an die Messwerte anpassen. In der Version für Studierende ist die zusätzliche Erweiterung „Statistics Toolbox“ enthalten, welche das Anpassen der Messwerte an Polynome (maximal 10-ten Grades) vereinfacht. Mithilfe dieser Toolbox wird nun das Anpassen einer bestimmten Funktion an die Messwerte, das sogenannte „fitten“, erläutert.

Wenn man im figure-Window das Menü `Tools` auswählt (siehe Abbildung 6), findet man ganz unten zwei sehr nützliche Mittel zur Datenanalyse. Mithilfe der Funktion „Basic Fitting“ kann man verschiedene Funktionen an die Messwerte anpassen und bei „Data Statistics“ können statistische Eigenschaften der Messreihe, wie Mittelwert und Standardabweichung, ausgegeben werden.

Wie am Ende von Abschnitt 6.1.4 schon erwähnt, kann man beim grafischen Auftragen des Vektors x gegen y einen linearen Zusammenhang vermuten. Um diesen zu bestätigen, setzt man nach Aufrufen der Basic Fitting Funktion den Haken bei „linear“ und klickt außerdem noch auf „Show equations“ um die Vorschau zu sehen. Wenn man nun auf den schwarzen Pfeil am unteren rechten Rand des Fensters (in Abbildung 7 rot eingekreist) klickt, kann man die Koeffizienten des angepassten Polynoms ersten Grades ablesen. Ein Vorteil der Statistics Toolbox ist, dass man die Ausgleichskurve direkt in das Diagramm einfügen kann.

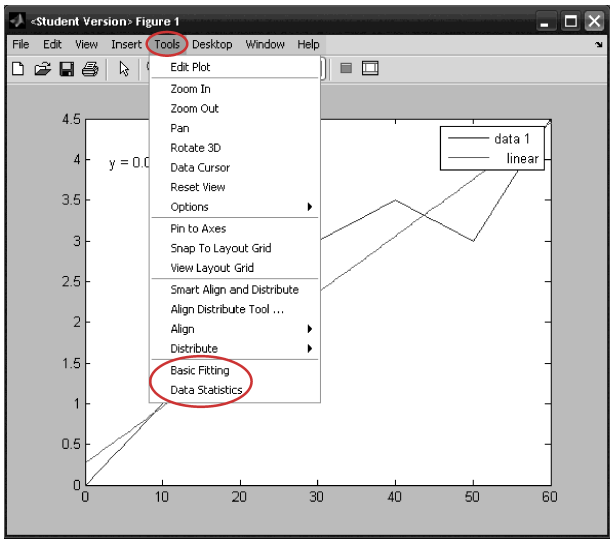

Abbildung 6: Hier ist das figure-Window mit ausgeklapptem Menü `Tools` zu sehen. Hinter dem Menü verborgen, kann man die Ausgleichsgerade erahnen.

Bei Octave und der Standard-Version von MATLAB (ohne Statistics Toolbox) kann man die Anpassung der Kurve mit dem Befehl `polyfit` erreichen. Mit der Syntax

```
EDU>> p = polyfit (x,y,n)
```

kann man die Koeffizienten eines optimalen Polynoms n-ten Grades durch die Messpunkte (x,y) heraus finden. Das Resultat p ist ein Zeilenvektor der Länge $n+1$. Der erste Eintrag des Vektors p ist der Koeffizient des Summanden mit der höchsten Potenz.

Um den linearen Zusammenhang der Vektoren x und y aus Abschnitt 6.1 zu verdeutlichen, kann man folgenden Befehl verwenden:

```
EDU>> p = polyfit (x,y,1)

p =

    0.0696    0.2679
```

Wenn man sich die Koeffizienten der Ausgleichsgerade durch Klicken auf den schwarzen Pfeil in Abbildung 7 anzeigen lassen würde, käme man auf dasselbe Resultat. Um die Ausgleichsgerade auch optisch mit dem Messpunkten vergleichen zu können, müsste man die Geradengleichung mit den zugehörigen Koeffizienten in dasselbe figure-Window zeichen lassen, in dem die Messwerte aufgetragen sind.

In MATLAB ist es auch möglich, Messwerte an eine beliebige Funktion anzupassen und so einige Parameter zu bestimmen. So kann man bei einem Versuch zur Beugung von Licht zum Beispiel aus der Lage der Maxima auf die Spaltbreite schließen. Dafür benötigt man die *Curve Fitting Toolbox*, die allerdings weder in der Standard-Version von MATLAB noch in der Version für Studierende enthalten ist.[6]

[6] Im PC-Pool des Praktikums ist die Curve Fitting Toolbox vorhanden. Nutzen Sie die Gelegenheit, damit zu

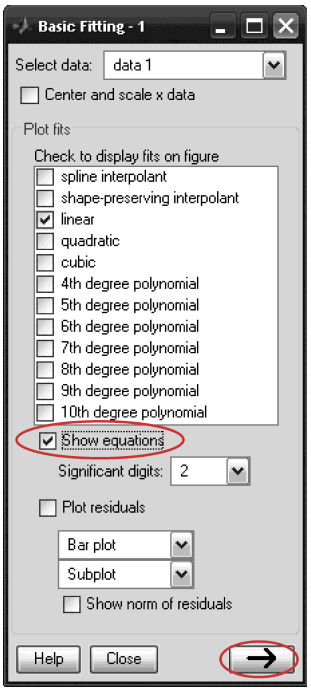

Abbildung 7: Zu sehen ist hier das Fenster, welches sich nach dem Aufrufen der „Basic Fitting“ Funktion im Menü `Tools` öffnet.

6.3 Statistik von Messungen

Im Praktikum werden Sie einen elektronischen Würfel kennen lernen (Zusatzversuch im P2). Durch Drücken einer Taste zeigt der Würfel eine Augenzahl an. Am Beispiel des elektronischen Würfels kann man die Funktionsweise von „Data Statistics“ erläutern (siehe Abbildung 6). Angenommen man betätigt die Taste des Würfels 60 mal, um Zufallszahlen zu erzeugen und notiert die Augenzahlen, wobei die Reihenfolge nicht berücksichtigt werden muss. Als Ergebnis erhält man beispielsweise:

Augenzahl	1	2	3	4	5	6
absolute Häufigkeit	10	11	9	13	10	7

Diese Daten kann man als Vektoren speichern und grafisch gegeneinander auftragen. Wählt man im figure-Window das Menü `Tools - Data Statistics` aus (vgl. Abbildung 6), so werden für beide Vektoren statistische Daten angezeigt, die durch Setzen des Häkchens wie in Abbildung 8 direkt in das Diagramm eingezeichnet werden können.

Die im Fenster „Data Statistics“ angegebenen Abkürzungen sind:

- min - Minimum der Stichprobe
- max - Maximum der Stichprobe
- mean - Mittelwert der Stichprobe

experimentieren!

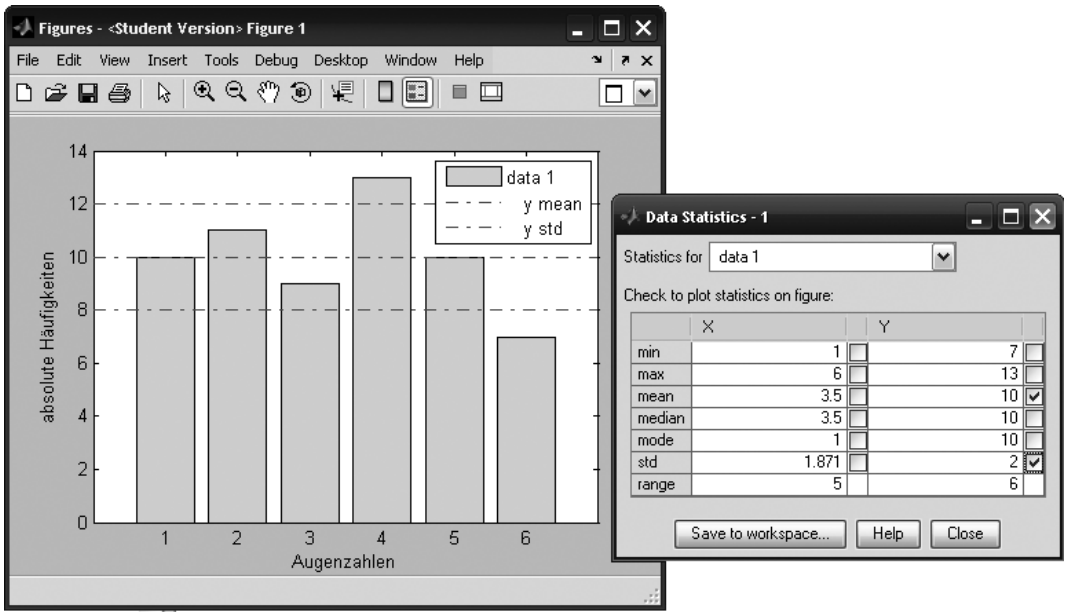

Abbildung 8: Zu sehen ist hier das figure-Window und die errechneten statistischen Werte.

- median - Median der Stichprobe (Grenze zwischen den 2 Hälften einer geordneten Stichprobe)
- mode - in der Stichprobe am häufigsten vorkommende Wert
- std - Standardabweichung der Stichprobe
- range - Spannweite der Stichprobe

In diesem Beispiel wurden der Mittelwert und die Standardabweichung für die y-Werte, also für die absoluten Häufigkeiten der Augenzahlen, eingezeichnet. Mit dem Button „Save to Workspace“ kann man die ausgewählten Werte als Variablen im Workspace speichern.

Die statistischen Daten kann man mit Octave oder der Standardversion von Matlab ebenfalls berechnen. Die Befehlseingabe erfolgt mit den Abkürzungen für die statistischen Daten. Um z. B. den Mittelwert der absoluten Häufigkeiten auszurechnen, würde man folgendes eingeben:

```
EDU>> x = [1:6];
EDU>> y = [10,11,9,13,10,7];
EDU>> my = mean (y)

my =

    10
```

Das Ergebnis stimmt mit dem Wert aus Abbildung 8 überein.

Die anderen Daten kann man äquivalent mit den entsprechenden Befehlen ausrechnen.

6.4 Mehrere Plots in ein Diagramm

Um qualitative Aussagen zu treffen, reicht es häufig, Daten optisch miteinander zu vergleichen. Dazu kann man sich mehrere Messreihen in ein und dasselbe Diagramm zeichnen lassen. Dafür gibt es zwei Befehle: `hold on` und `plotyy`.

Hat man beispielsweise drei Durchgänge des elektronischen Würfels mit je 60 Messungen aufgenommen, so könnte man folgende Daten erhalten:

Augenzahl	1	2	3	4	5	6
absolute Häufigkeit 1. Durchgang:	10	11	9	13	10	7
absolute Häufigkeit 2. Durchgang:	8	10	11	9	14	8
absolute Häufigkeit 3. Durchgang:	12	7	10	10	9	12

Diese Ergebnisse können nun mithilfe von MATLAB grafisch veranschaulicht werden. Dazu gibt man die Werte zuerst als Matrix ein und stellt sie anschließend mit dem Befehl `bar` dar.

```
EDU>> D = [10,11,9,13,10,7;8,10,11,9,14,8;12,7,10,10,9,12]

D =

    10    11     9    13    10     7
     8    10    11     9    14     8
    12     7    10    10     9    12

EDU>> bar([1:6],D)
??? Error using ==> bar at 53
The length of X must match the number of rows of Y.

EDU>> bar([1:6],D')
```

Zu sehen ist im obigen Beispiel explizit die Fehlermeldung, die daraus resultiert, dass die Matrix D erst transponiert werden muss, um auf die richtigen Daten zuzugreifen. Fehlermeldungen dieser Art erhält man beim Erlernen von MATLAB des Öfteren, lassen Sie sich davon jedoch nicht abschrecken!

Nach der richtigen Eingabe öffnet sich das figure-Window.

Nun könnte man beabsichtigen, zusätzlich den Verlauf der mittleren Häufigkeit der jeweiligen Augenzahlen einzeichnen zu lassen. Da sich jedoch nach der Eingabe des erneuten Zeichenbefehls das Diagramm im selben Fenster öffen und das alte Diagramm überschreiben würde, wäre zwischen beiden ein optischer Vergleich nicht möglich. Hier kommt der Befehl `hold on` zum Tragen. Nach dem ersten Zeichenbefehl führt man im Command Window den Befehl `hold on` aus. Alle weiteren Zeichenbefehle werden nun in demselben figure-Window ausgeführt bis man dieses „Festhalten“ mit dem Befehl `hold off` wieder auflöst.

Mit folgenden Befehlen wird nun der Mittelwert der Häufigkeit über die drei Versuchsdurchläufe in Abhängigkeit der Augenzahl in dasselbe Diagramm gezeichnet (vgl. Abbildung 9):

```
EDU>> hold on
EDU>> plot([1:6],mean(D))
```

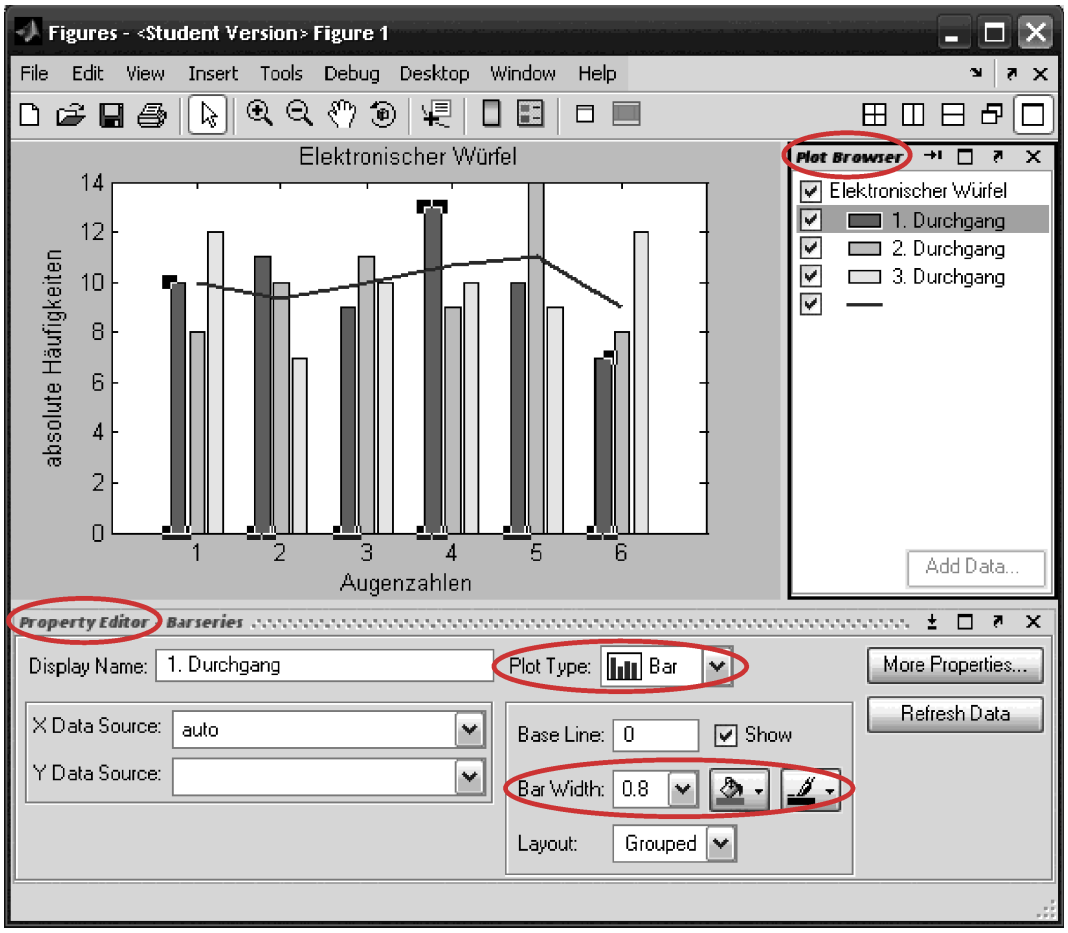

Abbildung 9: Grafische Veranschaulichung der drei Durchgänge des Experiments mit dem elektronischen Würfel.

Im Abbildung 9 sind einige zentrale Bedienelemente zur Einstellung der grafischen Ausgabe durch rote Ellipsen hervorgehoben. Durch einfaches Klicken auf einen Balken eines Durchgangs oder durch Klicken auf den Durchgang rechts im „Plot Browser“, öffnet sich der zugehörige „Property Editor“ indem man Farbe und Breite der Balken verändern und den Namen der Messreihe eingeben kann. Bei „Plot Type“ kann man nun nachträglich die Art der Darstellung der Messwerte ändern.

Im Versuch STV lernen Sie das Galton-Brett kennen. Die Theorie zum Galton-Brett besagt, dass die dabei untersuchte Zufallsgröße einer Binomialverteilung folgt. Diese Verteilung sollte bei dem im Versuch verwendeten Galton-Brett einer Gaußschen Glockenkurve mit Erwartungswert 5 und Standardabweichung 1,8 sehr ähnlich sein.[7] Um dies zu überprüfen, kann man die Verteilung der Ereignisse und den Verlauf der Gaußschen Glockenkurve in dieselbe Grafik eingezeichnet werden.

Das Maximum der Normalverteilung mit Erwartungswert 5 und Standardabweichung 1,8

[7] Mehr dazu finden Sie in den mathematischen Grundlagen der Versuchsanleitung zu STV.

beträgt etwa 0,2216. Der Wert der beobachteten Häufigkeit bestimmter Ereignisse hängt jedoch von der Gesamtanzahl der Durchläufe ab und ist im Versuch maximal etwa 60. Deshalb würde man beim Einzeichnen der Glockenkurve mithilfe des Befehls `hold on` diese kaum sehen, da sie viel zu flach wäre. Man könnte nun die Glockenkurve mit einem passenden Faktor skalieren, doch den passenden Faktor zu finden ist umständlich. Einfacher ist es, für beide Plots unterschiedliche Skalierungen der y-Achse zu wählen. Dies kann durch den Befehl `plotyy` verwirklicht werden.

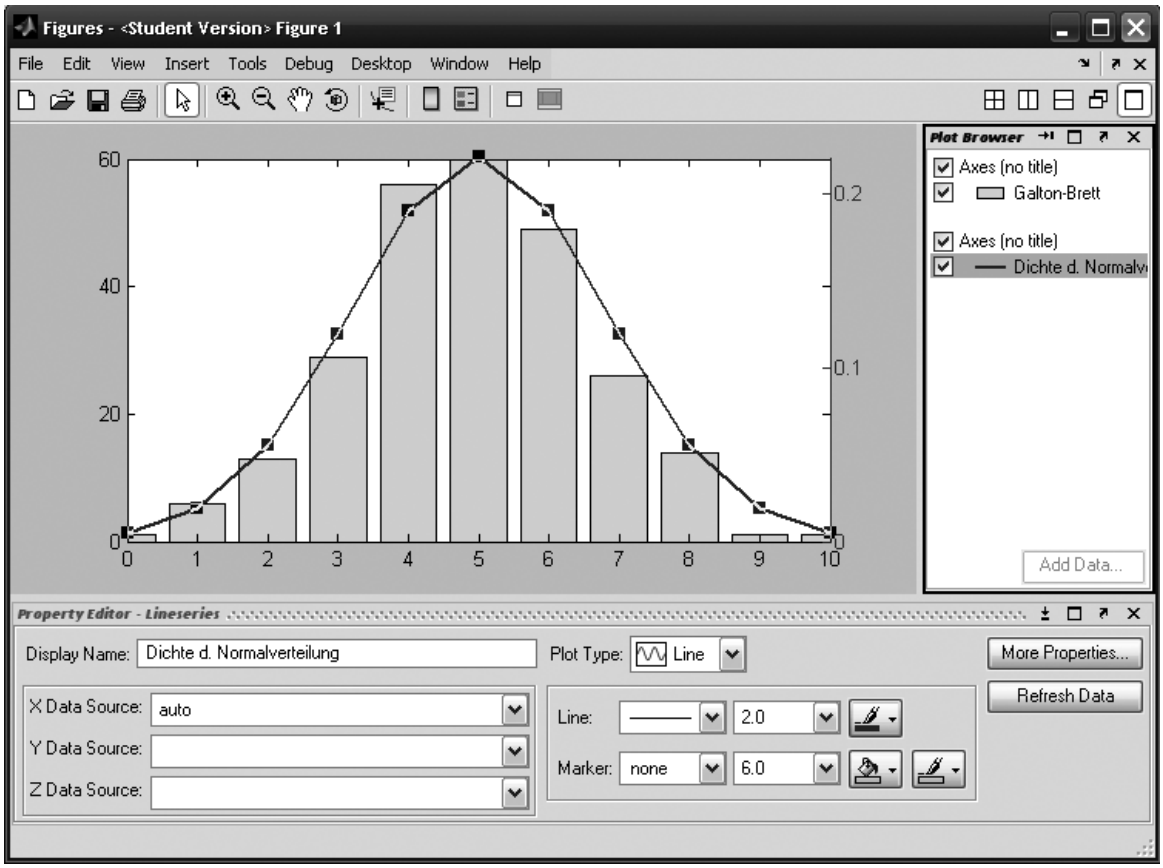

Abbildung 10: Ergebnisse eines Durchgangs mit dem Galton-Brett zusammen mit der Normalverteilung. Zu beachten sind die unterschiedlichen Ordinateneinteilungen links und rechts des Diagramms.

Zuerst werden die beobachteten Häufigkeiten am Galton-Brett (über die Kanäle 0 bis 10) als Vektoren in MATLAB generiert:

```
EDU>> x = [0:10];
EDU>> G = [1,6,13,29,56,60,49,26,14,1,1];
```

Da die Funktion, die die Gaußsche Glockenkurve darstellt, bei MATLAB bereits vorhanden ist, kann man sich ein separates Definieren dieser sparen. Die Bezeichnung der Normalverteilung in MATLAB lautet `normpdf(x,`μ`,`σ`)`. Die Ergebnisse G des Galton-Bretts und die Normalverteilung *normpdf(x,5,1.8)* lässt man sich wie folgt zeichnen:

```
EDU>> plotyy(x,G,x,normpdf(x,5,1.8),'bar','plot')
```

Die Eingabe von bar bzw. plot bezieht sich auf die Art und Weise des Diagramms des ersten bzw. zweiten Wertepaars. Zur allgemeinen Eingabe des Befehls plotyy beachten Sie bitte den Inhalt des in Abbildung 2 dargestellten Hilfe-Fensters.

Bei diesem Beispiel ist die Übereinstimmung von Histogramm und Verteilung augenscheinlich gut. Es gibt quantitative statistische Tests, die die Güte der Übereinstimmung genauer analysieren, doch dies soll hier nicht thematisiert werden.

7 Skripte und Funktionen (*)

Bisher wurden nur einfache Befehle von MATLAB und deren Anwendung vorgestellt. Das Programm kann jedoch noch viel mehr. Mithilfe von MATLAB-Skripten kann man eine Folge von MATLAB-Anweisungen erstellen und ausführen. Dies ist von Vorteil, wenn man häufig dieselben Befehle ausführen muss, wenn man zum Beispiel mehrfach dasselbe Polynom in derselben grafischen Darstellung verwenden muss. Darüber hinaus können in solchen Skripten auch z. B. Schleifen und bedingte Sprünge realisiert werden. Ein Beispiel ist in Abbildung 11 zu sehen.

Im Abschnitt 6.4 hat man zum Beispiel das Ergebnis eines Durchgangs mit dem Galton-Brett mit der Normalverteilung verglichen. Um daraus ein Skript zu erstellen, muss man lediglich die benötigten MATLAB-Befehle in eine Datei *Dateiname*.m[8] speichern. Man kann dies komfortabel machen, indem man in der Command History die nötigen Befehle (Definition der Vektoren und plot-Befehl) auswählt, mit der rechten Maustaste darauf klickt und anschließend aus dem Kontextmenü den Befehl `Create M-file` auswählt.

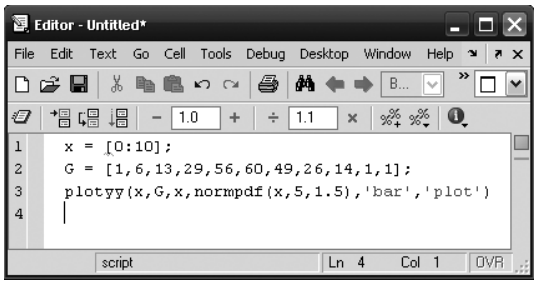

Abbildung 11: Hier ist das Editor-Fenster abgebildet, welches sich nach dem Befehl „Create M-File“ öffnet.

Nachdem diese Befehlsfolge abgespeichert wurde, kann man sie jederzeit durch Eingabe des Dateinamens in das Command Window (ohne die Endung der Datei „.m“) erneut ausführen.

Es ist auch möglich, eine M-Datei herzustellen, indem man den Editor über das Menü `File - New - M-File` aufruft und die nötigen Befehle direkt dort eingibt und in einer Datei speichert. Einen Editor kann man in Octave (und in MATLAB) auch mit dem Befehl `edit` öffnen.

Mithilfe von M-Dateien ist es außerdem möglich, MATLAB um eigene Funktionen zu erweitern. Man kann zum Beispiel eine Funktion programmieren, die zu einem bestimmten Radius den Umfang des Kreises ausrechnet. Dazu ruft man ein neues Editor-Fenster auf und gibt die Funktion wie in Abbildung 12 ein.

Der Text hinter dem Prozentzeichen % wird von MATLAB ignoriert. Er dient lediglich zur Erklärung der Funktion für weitere Nutzer. Wichtig bei der Programmierung einer Funktion ist, dass ganz oben der Name der Funktion, hier `umfang` definiert wird. Man muss eine M-Datei, die eine neue Funktion definiert unter deren Namen abspeichern, andernfalls kann es beim Aufrufen der Funktion zu Fehlern kommen. Die Funktion im obigen Beispiel muss also unter dem Namen `umfang.m` abgespeichert werden. Durch Eingabe des Befehls `help` gefolgt

[8] Aufgrund der Endung der Datei „.m“ spricht man auch von M-Dateien oder M-Files.

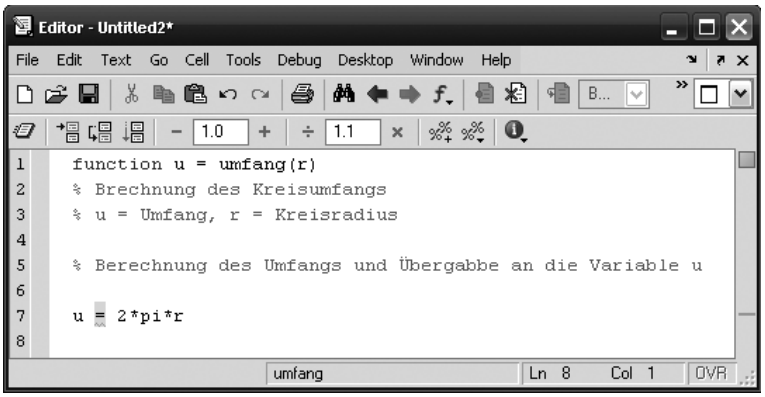

Abbildung 12: Editor mit programmierter Funktion.

von dem Namen der Funktion (hier `umfang`) gibt MATLAB den erklärenden Text zu dieser Funktion im Command Window aus.

Zur Berechung des Umfangs eines Kreises mit Radius 2,5 kann man nun folgende Syntax ins Command Window eingeben:

```
EDU>> r1 = 2.5;
EDU>> umf = Umfang(r1)

umf =

    15.7080
```

In Octave funktioniert die Programmierung der Funktion ebenso wie in MATLAB. Die Befehlseingabe im Editor ist identisch. Allerdings muss beim Speichern unbedingt die Endung „.m“ an den Funktionsnamen angehängt werden, sonst kann Octave auf die Funktion nicht zugreifen. MATLAB kann die Funktion auch benutzen, ohne dass sie explizit mit der Endung „.m“ versehen wird. Die Benutzung der Funktion in Octave ist wieder identisch mit MATLAB:

```
octave-3.0.3.exe:29> r1 = 2.5;
octave-3.0.3.exe:30> umf = umfang(r1)
umf =  15.708
```

8 Analogie-Befehlsübersicht für Python (*)

Für Studierende, die bereits erste Erfahrungen mit der Programmiersprache Python gemacht haben, stellen wir hier eine Übersicht zusammen, welche der in diesem MATLAB-Skript vorgestellten Befehle in analoger oder ähnlicher Weise auch in Python verfügbar sind.

Ein gutes Tutorial zum Selbststudium ist „Numerisches Programmieren mit Python“, abrufbar unter `https://www.python-kurs.eu/numerisches_programmieren_in_Python.php`. Dort finden sich auch Informationen, inwieweit Python eine Alternative zu MATLAB darstellt.

Ein ausführlicher Vergleich zwischen Befehlen für MATLAB und Python findet sich unter `http://mathesaurus.sourceforge.net/matlab-numpy.html`.

Einbinden von Modulen	
`import numpy as np`	Modul NumPy für Vektorberechnungen importieren
`import matplotlib.pyplot as plt`	Modul matplotlib für Diagrammerstellung importieren
`from scipy.stats import norm`	Befehl norm von Modul scipy importieren
`from scipy.misc import factorial`	Befehl factorial von Modul scipy importieren

Analogie-Befehlsübersicht für Python (*)

Daten importieren	
`f = np.loadtxt('Dateiname',`	Daten aus ASCII-formatierten Dateien einlesen,
`delimiter='',`	''=Trennzeichen zwischen den Spalten,
`comments='#')`	#=1. Zeichen von #Headerzeilen,

Erstellen von Vektoren/Matrizen	
`x = [1,2,3]`	Vektor
`x = np.array([1,2,3])`	Vektor
`x = np.linspace(a,b,n)`	gleichförmiger Vektor von a bis b mit n äquidistanten Punkten
`A = np.matrix('1,2;3,4')`	Matrix
`d = A[n,m]`	Eintrag einer Matrix auslesen (Zählung ab 0)
`d = A(0:4,1:2)`	Bereich einer Matrix auslesen
`a1 = A[:,0]`	Spalte 1 einer Matrix auslesen
`A[n,m] = 2`	Eintrag einer Matrix überschreiben
`+ - *`	mathematische Operationen
`np.multiply(A,B)`	komponentenweise Multiplikation
`np.divide(A,B)`	komponentenweise Division
`np.power(A,B)`	komponentenweise Potenzierung
`s = np.dot(a1,a2)`	Skalarprodukt zweier Vektoren a1 und a2
`s = np.sum(a1)`	Summe der Komponenten des Vektors a1

Plotten und Visualisieren	
`plt.plot(x,y,label="Legende")`	Daten in einem xy-Diagramm darstellen
`plt.bar(x,y,label="Legende")`	Daten als Histogramm in einem Diagramm darstellen
`plt.stem(xs,ys,label="Legende")`	Daten als Diskretes-Folgen-Diagramm darstellen
`plt.title("Titel")`	Titel und Legende
`plt.xlabel("Label")`	Achsenbeschriftung x-Achse (analog y-Achse)
`plt.legend()`	Legende anzeigen
`p = np.polyfit(x,y,n)`	Werte (x,y) an ein Polynom n-ten Grades anpassen (fitten)

Statistische Funktionen	
`norm.pdf(x,erwartungswert,stabw)`	Berechnen der Normalverteilung
`factorial(n)`	Berechnen der Fakultät von n

Sonstiges	
`print(a1)`	Ausgabe des Vektors a1
`clear`	Console bereinigen (nur in Console verwendbar)
`#`	Kommentar

9 Befehlsübersicht für MATLAB

Daten importieren	Seite	
`dlmread('Dateiname','',a,b)`	5	Daten aus ASCII-formatierten Dateien einlesen, ''=Trennzeichen zwischen den Spalten, a=#Headerzeilen, b=Spalten, die nicht beachtet werden sollen
Erstellen von Vektoren/Matrizen		
`x = [1,2,3]`	8	Vektor
`x = [a:dx:b]`	7	gleichförmiger Vektor von a bis b mit Schrittweite dx
`x = linspace(a,b,n)`	7	gleichförmiger Vektor von a bis b mit n äquidistanten Punkten
`A = [1,2;3,4]`	8	Matrix
`d = A(n,m)`	9	Eintrag einer Matrix auslesen
`d = A(1:5,2:3)`	10	Bereich einer Matrix auslesen
`a1 = A(:,1)`	10	Spalte 1 einer Matrix auslesen
`A(n,m) = 2`	10	Eintrag einer Matrix überschreiben
`+ - * /`	11	mathematische Operationen
`.* ./ .^`	12	komponentenweise Operationen
`s = dot(a1,a2)`	14	Skalarprodukt zweier Vektoren a1 und a2
`s = sum(a1)`	14	Summe der Komponenten des Vektors a1
Plotten und Visualisieren		
`fplot('x^2',[-3,3])`	15	plotten einer Normalparabel im Intervall -3 bis 3
`plot,bar` oder `stem(x,y)`	16	Daten in einem Diagramm darstellen
`title,xlabel,ylabel,legend`	17	Titel, Achsenbeschriftung und Legende eines Diagramms einstellen
`p = polyfit(x,y,n)`	19	Werte (x,y) an ein Polynom n-ten Grades anpassen (fitten)
`hold on`	22	Modus Hinzufügen zu aktuellem Diagramm aktivieren
`plotyy(`x_1`,`y_1`,`x_2`,`y_2`,'bar','plot')`	24	Diagramm mit zwei unterschiedlich skalierten y-Achsen
Sonstiges		
`;`		Ausgabe im Command Window unterdrücken
`clc`		Command Window bereinigen
`clear`		Workspace bereinigen
`%`		Kommentar