Fachhochschule Nordwestschweiz
Hochschule für Technik

Einführung in Matlab

E.Gutknecht / Th. Heim

HS 2018

Inhaltsverzeichnis

1 Was ist Matlab ?

MATLAB ist ein interaktives Software-System der Firma *Mathworks* für technisch wissenschaftliche Berechnungen und Visualisierungen. Es eignet sich für alle mathematischen Aufgaben von einfachen Berechnungen als Ersatz für einen Taschenrechner bis zu professionellen Anwendungen (2D- und 3D-Graphik, Signalverarbeitung, Simulationen, Statistik, usw.)

Der Name MATLAB steht für *Matrix Laboratory*.

Technisch gesehen ist Matlab ein Interpreter einer vollwertigen mathematisch orientierten Programmiersprache. Es verwendet konsequent den Ansatz von Programmiersprachen, basierend auf *Variabeln* und *Ausdrücken:*

- Variabeln für Zahlen, Vektoren und Matrizen mit beliebigen Namen

  ```
  g = 9.81;
  rErde = 6370;
  noten = [ 4.5, 5.2, 4.8, 3.5, 5.0 ];
  ```

  Die Variable `noten` ist ein Vektor, bestehend aus mehreren Werten (Werteliste).

- Zusammengesetze Ausdrücke mit Konstanten, Variabeln, Operatoren, Klammern und Funktionen.

  ```
  x1 = (- b + sqrt(b*b-4*a*c) / (2*a);
  ```

Vorteile dieses Ansatzes

- Mathematische Lösungswege können direkt übersetzt werden
- Einfache Dokumentation (Text statt Tastenfolgen)
- Skripts
- Nahtloser Übergang zur Programmierung

Verfügbarkeit von Matlab

Matlab steht für Windows, MacOS und Linux zur Verfügung. Für Tablets und Smartphones (Android, iOS) können Matlab-Apps installiert werden, die jedoch eine Internet-Verbindung zum Matlab-Server benötigen.

Alternative Systeme

Für die Verwendung von Matlab sind Matlab-Lizenzen erforderlich, die relativ teuer sind. Es existieren jedoch alternative Programme und Apps für diverse Betriebssysteme, die weitgehend äquivalent zu Matlab sind und gratis oder zu günstigen Preisen zur Verfügung stehen:

Programm/App	Betriebssystem
Octave	Windows, Linux
Mathmatiz	Android (Tablets, Smartphones)
SIMO	iOS (iPad, iPhone)

Details dazu, siehe Anhang, Seite 36. Octave steht auch für Android zur Verfügung, die Installation und Bedienung sind jedoch relativ aufwendig.

2 Grundlagen

2.1 Einfache Berechnungen

Nach dem Start von Matlab erscheint das Hauptbild, mit dem *Command-Window* für die Eingabe von Daten und Befehlen. Das Prompt-Zeichen '>>' (Eingabeaufforderung) markiert die Eingabeposition:

```
>>
```

Die einfachste Anwendung von Matlab ist die Auswertung von algebraischen Ausdrücken, bestehend aus Konstanten, Variabeln, Operatoren, Klammern und Funktionsaufrufen, welche gemäss dem Prinzip von Programmiersprachen (C++, Java, Python) geschrieben werden.

Beispiel: arithmetisches Mittel zweier Zahlen

```
>> mittel = (4.5 + 5.2) / 2
```

Die Eingabe wird mit der Return-Taste abgeschlossen, wodurch sie von Matlab verarbeitet wird. Der Ausdruck auf der rechten Seite wird ausgewertet, und das Resultat wird in der Variabeln mit dem (beliebig wählbaren) Namen 'mittel' gespeichert und ausgegeben:

```
mittel =
    4.8500
```

Ein Ausdruck kann auch ohne Zuweisung zu einer Variabeln eingegeben werden. Das Resultat wird dann der System-Variabeln ans (Answer) zugeordnet:

```
>> (4.5 + 5.2) / 2

ans =
    4.8500
```

Algebraische Operatoren

Operator	Bedeutung	Stufe
+ , −	Addition, Subtraktion	1
* , /	Multiplikation, Division	2
^	Potenz	3

Auswertungsreihenfolge

Die Ausdrücke werden nach den gewohnten Regeln der Mathematik ausgewertet:

- Die Operationen zweiter Stufe werden vor den Operationen erster Stufe ausgeführt.
- Operationen dritter Stufe werden vor den Operationen der zweiten und ersten Stufe ausgeführt.

– Operationen gleicher Stufe werden von links nach rechts ausgeführt.

– Für Abweichungen von dieser Reihenfolge werden runde Klammern gesetzt.

Beispiele:

Matlab-Ausdruck	Bedeutung	Wert
(4.5+5.2)/2	$\frac{4.5+5.2}{2}$	4.85
4.5+5.2/2	$4.5 + \frac{5.2}{2}$	7.1
12 / (4 * 2.5)	$\frac{12}{4 \cdot 2.5}$	1.2
12 / 4 * 2.5	$\frac{12}{4} \cdot 2.5$	7.5

Im letzten Beispiel werden die Division und die Multiplikation von links nach rechts ausgeführt, da beide von derselben Stufe 2 sind.

Wiederholung von Eingaben (Command-History)

Mit den Pfeiltasten 'up' und 'down' können die eingegebenen Zeilen (History) wieder auf die Prompt-Zeile geholt werden und nach eventueller Modifikation wieder verarbeitet werden (Return-Taste). Mit der up-Taste geht es eine Zeile in der History zurück und mit down wieder vorwärts.

2.2 Variabeln

Eine *Variable* ist ein Name für einen Platz im Hauptspeicher (RAM) des Computers, in dem ein Wert gespeichert werden kann. Die Namen werden vom Benutzer gewählt und sollten möglichst aussagekräftig sein.

In Matlab wird eine Variable einfach durch eine *Wertzuweisung* definiert: *Wertzuweisung*

```
>> note = 4.5
```

Dadurch wird eine Variable mit dem Namen 'note' definiert und mit dem Wert 4.5 versehen. Wenn die Variable schon definiert wurde, wird ihr bisheriger Wert durch den neuen ersetzt.

Der Name einer Variabeln muss mit einem Buchstaben beginnen, anschliessend sind Buchstaben, Ziffern und das Unterstreichungszeichen '_' erlaubt. Nicht erlaubt: Blanks, Umlaute, Sonderzeichen. Gross- und Kleinschrift spielt bei den Namen eine Rolle: note und Note sind zwei verschiedene Variabeln.

Allgemeines Format einer Wertzuweisung

variable = wert

Dabei ist *'wert'* ein beliebiger Ausdruck. Dieser wird ausgewertet und das Resultat wird in die Variable auf der linken Seite abgespeichert.

▶ *Merke:*

Das Zeichen '=' hat in Matlab (im Gegensatz zur Mathematik) *nicht* die Bedeutung eines Vergleichsoperators, sondern stellt den *Zuweisungsoperator* dar, welcher von rechts nach links funktioniert: Der Ausdruck auf der rechten Seite wird ausgewertet und in die Variable auf der linken Seite abgespeichert.

Beispiele:

1. `>> note1 = 4.5`
   `>> note2 = 5.2`
   `>> mittel = (note1 + note2) / 2`

2. `>> x = 2.4`
   `>> y = 4*x^2 + 2*x - 2`
   `>> x = -5.5`

   Die letzte Zuweisung hat keinen Einfluss auf den Wert von `y`, da bei einer Zuweisung nur der Wert der Variabeln zu diesem Zeitpunkt massgebend ist. Nach einer Zuweisung bleibt der Wert der Variabeln unverändert, bis eine neue Zuweisung für die Variable erfolgt.

3. Inhalt einer Variabeln erhöhen

Matlab-Ausdruck	resultierender Wert von `x`
`>> x = 4`	4
`>> x = x + 1`	5

   Gemäss der Funktionsweise des Zuweisungsoperators '=' von rechts nach links, macht auch der zweite Ausdruck in Matlab (im Gegensatz zur Mathematik) einen Sinn: der Ausdruck auf der rechten Seite wird berechnet, dann wird das Resultat (5) in `x` gespeichert.

Abfrage des Wertes einer Variabeln

Der Wert einer Variabeln kann jederzeit angezeigt werden, indem einfach der Name der Variabeln eingegeben wird:

```
>> note1
```

Abfrage der definierten Variabeln

```
>> whos
```

Der Befehl zeigt die momentan definierten Variabeln an, den sog. *Workspace*.

2.3 Funktionen und Konstanten

In algebraischen Ausdrücken stehen weiter die bekannten mathematischen Funktionen und Konstanten zur Verfügung:

pi	Zahl $\pi = 3.14159265$
Inf	Infinity (Unendlich), z.B. Resultat von $\frac{1}{0}$
sin(x), cos(x), tan(x)	trigonometrische Funktionen, Winkel in rad
sind(x), cosd(x), tand(x)	trigonometrische Funktionen, Winkel in Grad
asin(x), acos(x), atan(x)	Umkehrfunktionen, Resultat in rad
asind(x), acosd(x), atand(x)	Umkehrfunktionen, Resultat in Grad
atan2(y,x)	Vier-Quadranten Arcus-Tangens, $-\pi \leq$ atan2(y,x) $\leq \pi$
exp(x)	Exponentialfunktion e^x
log(x)	Logarithmus zur Basis e (Umkehrfunktion von exp)
log10(x), log2(x)	Logarithmen zur Basis 10 bzw. 2
abs(x)	absoluter Betrag
sqrt(x)	Quadratwurzel (square root)
nthroot(x,n)	n-te Wurzel
sign(x)	Vorzeichenfunktion
fix(x)	Dezimalstellen abschneiden
round(x)	Runden auf nächste ganze Zahl
floor(x)	Abrunden auf nächstliegende ganze Zahl
ceil(x)	Aufrunden auf nächste ganze Zahl
mod(x,y)	Rest der Division x/y (modulo)
rand	Zufallszahl im Intervall $[0, 1]$

Online-Dokumentation:

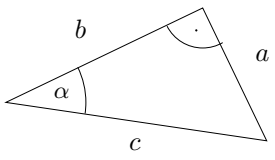
```
>> help elfun
```

Details des online Help-Systems, siehe unten.

Beispiele:

1. Rechtwinkliges Dreiecks (Pythagoras)

   ```
   >> a = 4
   >> b = 6
   >> c = sqrt(a^2 + b^2)
   >> alpha = atand(a/b)
   ```

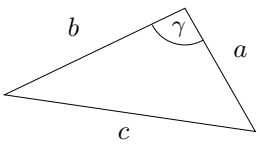


2. Allgemeines Dreieck

   Gegeben: $a = 3.4$, $b = 2.8$, $\gamma = 20^o$

   Berechnen Sie die Seite c mit dem Cosinus-Satz:

   $$c = \sqrt{a^2 + b^2 - 2ab\cos(\gamma)}$$



   Achtung: Matlab-Funktion 'cosd' (degree) verwenden. [$c = 1.23$]

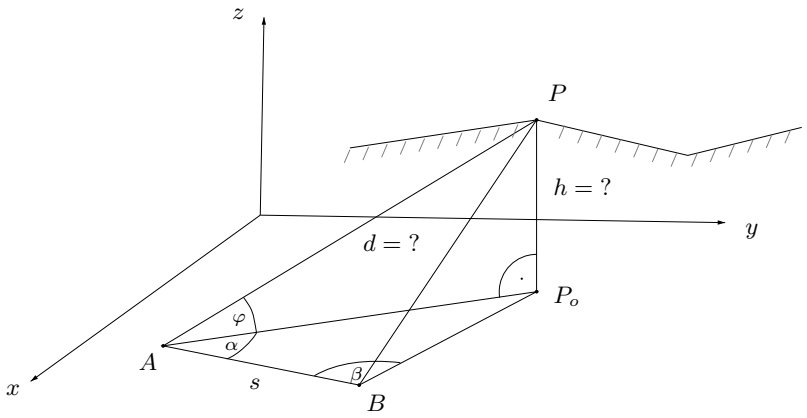
3. **Vorwärtseinschneiden (Vermessung)**

   Gegeben ist eine Referenzstrecke $s = \overline{AB}$ in der Ebene. Gesucht sind

   (a) die Entfernung d der Bergspitze P vom Punkt A und

   (b) die Höhe h von P über der Ebene.

   Dazu werden die Winkel α, β und φ benötigt, welche mit einem Theodoliten gemessen werden können.



Gegeben: $s = 800\,m$, $\alpha = 88^o$, $\beta = 75^o$, $\varphi = 24^o$

Lösungsweg:

1. Dreieck ABP_o:

Der Winkel beim Punkt P_o sei γ, d.h.

$$\gamma = 180^o - \alpha - \beta$$

Nach dem Sinus-Satz gilt für die
Strecke $b = \overline{AP_o}$:

$$\frac{b}{\sin\beta} = \frac{s}{\sin\gamma}, \qquad b = s \cdot \frac{\sin\beta}{\sin\gamma}$$

2. Dreieck AP_oP:

$$h = b \cdot \tan(\varphi), \quad d = \frac{b}{\cos\varphi}$$

Hier zeigt sich der Vorteil der Verwendung von Variabeln, weil dadurch die Lösung schrittweise berechnet werden kann.

[Resultate: $b = 2643.01\,m$, $h = 1176.74\,m$, $d = 2893.13\,m$]

2.4 Die Online-Dokumentation (Help-System)

Matlab enthält eine vollständige Online-Dokumentation, welche mit dem Befehl 'help' abgfragt werden kann.

help	Help-System starten

Help-Befehl mit Zusatzangaben:

help elfun	Dokumentation der elementaren Funktionen
help *befehl*	Dokumentation für einen Matlab-Befehl, z.B. help atan2

Ersetzt man hier 'help' durch 'doc' so wird ein erweitertes Dokumentationssystem gestartet, dessen Ausgaben nicht in das Copmmand-Window, sondern in ein spezielles Window geschrieben werden. *doc*

2.5 Beenden von Matlab

Zum Beenden von Matlab wir der Befehl 'exit' oder 'quit' eingegeben:

>> quit

2.6 Ergänzungen und Nachträge

1. Ausgabeformat festlegen

Die Anzahl ausgegebener Dezimalstellen kann mit dem Befehl 'format' beeinflusst werden:

format long	15 Dezimalstellen
format short	4 Dezimalstellen
format rat	Brüche (rational)

Bei dem Format rat werden für irrationale Zahlen (z.B. sqrt(2)) Bruchnäherungen angezeigt.

2. Textausgabe

Für Textausgaben steht der Befehl disp (display) zur Verfügung:

```
disp('Hello')
```

3. Unterdrückung von Ausgaben

Wenn man bei der Definition einer Variabeln die Ausgabe unterdrücken möchte, kann die Eingabe mit einem Strichpunkt beendet werden:

>> note1 = 4.5;

4. Fortsetzungszeilen für Eingaben

Für längere Eingaben kann die Eingabe mit '...' und Return unterbrochen werden. Dadurch wird der Ausdruck noch nicht ausgewertet, sondern es erscheint eine Leerzeile für die Fortsetzung der Eingabe.

```
>> mittel = (4.5 + 5.25 + 3.5 + 4 + ...
 3.75 + 5) / 6
```

5. Funktionen für den Workspace

Der *Workspace* von Matlab besteht aus der Gesamtheit der Variabeln.

Liste aller Variabeln anzeigen

Die momentan im Workspace definierten Variabeln werden mit dem Befehl 'who' oder mit 'whos' (Detailinformationen) aufgelistet:

```
>> whos
```

Die Werte der Variabeln werden dabei nicht angezeigt.

Variabeln in ein File abspeichern und wieder einlesen

Mit den Befehlen save und load können Variabeln vom Workspace in ein File abgespeichert und wieder in den Workspace eingelesen werden: *save, load*

`save 'c:\mydata\d.mat' a b`	Variabeln a und b in File speichern
`load 'c:\mydata\d.mat'`	Variabeln vom File einlesen

Wenn beim save-Befehl keine Namen von Variabeln angegeben werden, werden alle Variabeln des Workspace gespeichert.

Variabeln löschen

Mit dem Befehl clear können Variabeln gelöscht, d.h. aus dem Workspace entfernt werden. Dadurch wird Speicher freigegeben und die Variabeln können nicht fälschlicherweise verwendet werden. *clear*

`clear a`	Variable a löschen
`clear all`	(oder clear) alle Variabeln des Workspace
`clear`	wie clear all

6. Protokollierung der Eingaben

Eine Matlab-Session kann mit dem Befehl diary in einem Text-File protokolliert werden: *diary*

```
diary 'c:\mydata\protokoll.txt'
diary off      Protokoll beenden
```

3 Vektoren

Ein *Vektor* ist eine Variable, die mehrere Werte, eine *Werteliste*, umfasst. Die Werte eines Vektors werden in eckigen Klammern, getrennt durch Kommas, eingegeben:

```
>> noten = [4.2, 3.8, 5.1, 4.4, 3.0, 5.2, 3.5, 4.5, 4.8, 6.0, 3.5, 4.4, 5.2, 5.5, 4.9];
```

In der Programmierung nennt man Vektoren *Arrays* (Felder). Die Werte eines Vektors heissen *Komponenten*. Sie werden mit einem Index (Nummer) in *runden* Klammern angesprochen. Die Indizes beginnen immer bei 1 (erstes Element).

`noten(1)`	erstes Element (Wert 4.2)
`noten(3) = 5.0`	Element mit neuem Wert versehen

Algebraische Operationen

Die folgenden algebraischen Oprerationen sind komponentenweise definiert, d.h. die betreffende Operation wird auf die entsprechenden Komponenten der Vektoren angewandt.

Matlab	Bedeutung
`x + y`	Addition
`x - y`	Subtraktion
`t * x`	Multiplikation mit einer reellen Zahl t
`x * t`	”
`x / t`	Division durch eine reelle Zahl t

(Skalar- und Vektorprodukt siehe S. 23.)

Dabei sind `x` und `y` Vektoren mit der gleichen Anzahl Komponenten. Mit diesen Operatoren können auch zusammengesetzte Ausdrücke wie mit reellen Zahlen gebildet werden.

Beispiele:

1. `v = 3.4 * x - 5.5 * (y - x)`
2. `v = (x + y) / 2`

Die folgenden Operatoren, welche ebenfalls elementweise definiert sind, sind in der Mathematik wenig gebräuchlich. Sie sind für die Berechnung von Vektoren jedoch nützlich. Sie werden mit einem Punkt vor dem Operationszeichen geschrieben:

Matlab	Bedeutung
`x .* y`	komponentenweise Multiplikation
`x ./ y`	komponentenweise Division
`x .^ t`	komponentenweise Potenz mit einer reellen Zahl t

Beispiele:

1. `v = x .* y`
2. `v = x .^ 2`

Bemerkung:

Das spezielle Operationszeichen '.*' ist erforderlich zur Unterscheidung vom Matrix-Produkt '*' der Linearen Algebra für Matrizen und Vektoren.

Definition von Vektoren mit dem Bereichsoperator ':'

Mit dem Operator ':' können Vektoren definiert werden, deren Komponenten eine Zahlenfolge mit einen konstanten Zuwachs bilden.

Beispiel:

```
x = 0:2:10    ist äquivalent zu    x = [0, 2, 4, 6, 8, 10]
```

Allgemein:

```
x = a:d:e
```

Dabei sind `a` der Anfangswert, `d` das Inkrement und `e` der Endwert. Der so definierte Vektor besteht aus den Elementen

```
a, a+d, a+2*d, ...
```

bis zum letzten Wert vor der Überschreitung des Endwertes `e`.

Wenn das Inkrement 1 ist, kann es weggelassen werden.

Beispiele:

```
x = 10:15        entspricht  x = [ 10, 11, 12, 13, 14, 15]
x = 0:0.1:0.5    entspricht  x = [ 0, 0.1, 0.2, 0.3, 0.4, 0.5]
x = 10:-0.3:9    entspricht  x = [ 10, 9.7, 9.4, 9.1]
```

Vektoren als Argumente von Funktionen

Vektoren können als Argumente der mathematischen Funktionen verwendet werden. Die Funktion wird dabei auf jede Komponente des Vektors angewandt. Das Resultat ist der Vektor mit den Funktionswerten als Komponenten.

Beispiele:

```
>> x = 0:0.1:1;
>> y = sin(x)
>> y = exp(-x.^2)
```

Dies ergibt die Vektoren mit den Komponenten

$y_i = \sin(x_i)$ bzw. $y_i = e^{-(x_i)^2}$

Teilvektoren extrahieren

Aus einem Vektor `x` können Teilvektoren extrahiert werden. Dazu werden im Ausdruck `x( )` anstelle eines einzelnen Index `i` die Indizes der gewünschten Elemente in der Form eines Vektors angegeben. Gegeben sei der Vektor

```
x = [1.5, 2.8, 4.2, 1.4, 0.1]
```

Teilvektoren:

Ausdruck	Bedeutung	Resultat
x(3:5)	Elemente 3 bis 5	[4.2 1.4 0.1]
x([1, 3, 4])	Elemente 1, 3 und 4	[1.5 4.2 1.4]

Sortierung eines Vektors

Die Komponenten eines Vektor können auf- oder absteigend sortiert werden:

```
xs = sort(x)              aufsteigend
xs = sort(x,'descend')    absteigend
```

Anzahl Werte eines Vektors abfragen

```
n = length(x)
```

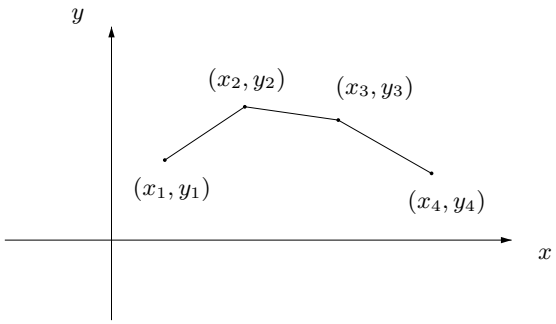
4 Graphische Darstellungen

Der Grundbefehl für graphische Darstellungen ist der plot-Befehl zum Zeichnen eines Streckenzuges, gegeben durch endlich viele Punkte *plot*

$(x_1, y_1),\ (x_2, y_2),\ \ldots$

Die Koordinaten der Punkte werden als Vektoren x und y übergeben:

```
x = [ 2, 5, 8.5, 12 ]
y = [ 3, 5, 4.5, 2.5 ]
plot(x,y)
```

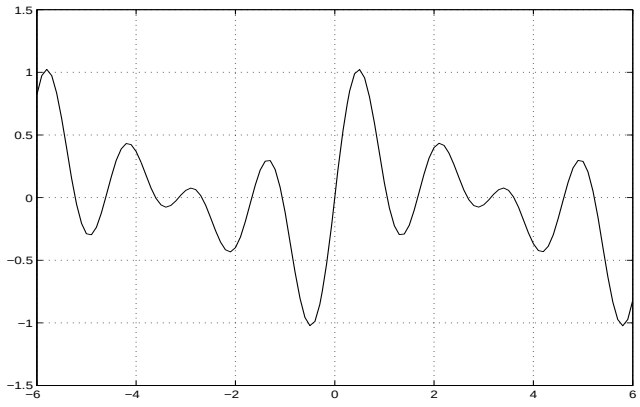


Die Ausgabe erfolgt in einem neuen Fenster (Figure-Object). Der plot-Befehl eignet sich insbesondere für die Darstellung von Funktionen:

Beispiel:

Summe von Sinus-Funktionen (Fourier-Reihe)

```
a=0.26; b=0.16; c=0.4; d=0.4;
x = −6:0.1:6;
y = a*sin(x) + b*sin(2*x) + c*sin(3*x) + d*sin(4*x);
plot(x,y)
grid on
```



Wenn das Inkrement der x-Werte erhöht wird, sieht man, dass tatsächlich ein Streckenzug durch die Punkte gezeichnet wird.

Koordinatenbereich festlegen

Das Rechteck der xy-Ebene, welches im Output-Window sichtbar ist, wird mit dem Befehl axis spezifiziert. Die Grenzen des Rechtecks werden als Vektor [xmin, xmax, ymin, ymax] übergeben:

```
axis([xmin, xmax, ymin, ymax])
```

axis

Wenn das Seitenverhältnis des Output-Windows auf dem Bildschirm nicht dem des Rechtecks entspricht, entstehen dabei Verzerrungen (Kreise werden zu Ellipsen, Quadrate zu Rechtecken, ...). Zur Vermeidung von Verzerrungen dient der Befehl

```
axis equal
```

Dieser bewirkt, dass die Koordinateneinheiten der x- und y-Achse auf dem Bildschirm gleich gross erscheinen. Dazu wird der angegebene x- oder y-Koordinatenbereich von Matlab entsprechend angepasst.

Weitere Befehle

- Gitterlinien ein-/ausschalten

  ```
  grid on
  grid off
  ```

- Zeichenfarbe setzen

  plot(x,y,'Color',[r,g,b])

  Der Befehl setzt das Attribut 'Color', welches die Farbe für Zeichenoperationen festlegt. Die Farbe wird als Vektor mit den RGB-Werten r, g, b angegeben. Dies sind Werte im Intervall $[0, 1]$. Alternativ kann die Farbe mit einem Character spezifiziert werden, z.B. 'b' für Blau:

  plot(x,y,'b')

  'b'=blue 'g'=green 'r'=red 'm'=magenta
  'c'=cyan 'y'=yellow 'k' =black 'w'=white

- Hintergrundfarbe des Graphik-Windows setzen

  set(gca, 'Color', [r,g,b])

  Dabei steht 'gca' für 'get current axis handle'. Der Befehl setzt das Attribut 'Color' des Objektes *axis*.

- Plot mit Darstellung von Punkten statt Strecken

  plot(x,y,'.')

  Optionale Zusatzparameter (Farbe Grün, Punktgrösse 1 Pixel):

  plot(x,y,'g.','MarkerSize',1)

  Andere Markierungszeichen für die Punkte, siehe 'help plot'.

- Mehrfachausgaben auf gleiches Bild

  Normalerweise wird das Graphik-Fenster bei jeder Graphik-Ausgabe gelöscht und neu aufgebaut. Wenn mehrere Ausgaben auf dasselbe Bild kommen sollen, muss die Graphik mit hold on festgehalten werden:

  hold on

  Mit hold off wird die Graphik wieder freigegeben.

- Graphik-Fenster löschen (clear figure)

  clf

- Beschriftung einer Graphik

  title('Summe von Sinus−Funktionen')
  xlabel('x')
  ylabel('y')

- Graphik ausdrucken

  print

  Ausgabe des aktuellen Graphik-Fensters auf den Standard-Drucker.

- Graphik als encapsulated Postscript-File abspeichern

  print −deps 'c:\bilder\bild.eps'

  Mit dem Parameter '-d' wird der gewünschte Device-Driver angegeben:

-dps Postscript
-dpsc Color Postscript
-deps encapsulated Postscript
-depsc encapsulated Color Postscript

Anschliessend folgt der Name des Output-Files. Weitere Devices, siehe Online-Dokumentation.

Polygone (Vielecke)

Polygone werden mit der Funktion fill gezeichnet. Sie werden mit einer Füllfarbe ausgemalt.

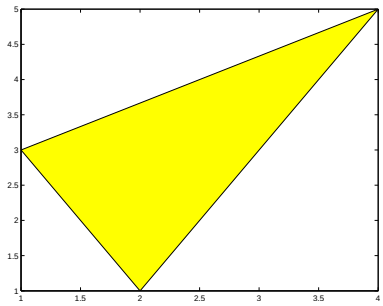
fill(x,y, [r,g,b]) gefülltes Polygon

x Zeilenvektor mit den x-Koordinaten der Eckpunkte
y y-Koordinaten der Eckpunkte
[r,g,b] Vektor mit den RGB-Werte der Füllfarbe $(0 \leq r, g, b \leq 1)$

Dreieck ABC:

$A(2,1),\ B(4,5),\ C(1,3)$

```
x=[2 4 1];
y=[1 5 3];
fill(x,y,[1 1 0])
```



Farbe der Randlinien:

```
fill(x,y, [r,g,b], 'EdgeColor', [r1, g1, b1])
```

ohne Randlinien:

```
fill(x,y, [r,g,b], 'EdgeColor', 'none')
```

5 Datenanalyse

Vektoren eignen sich für die Erfassung und Auswertung von Messwerten, z.B. für statistische Untersuchngen. Matlab bietet dazu viele Möglichkeiten.

5.1 Boxplots

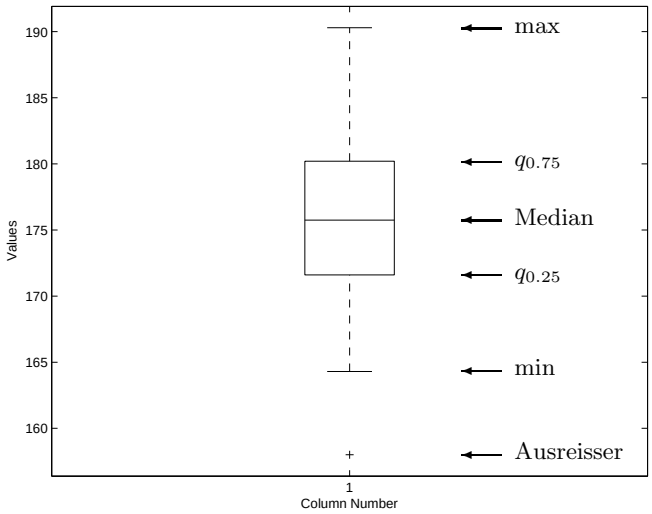
Beispiel: Körpergrösse von Personen

Gegeben sei ein Datensatz mit Körpergrössen einer Stichprobe von $n = 50$ erwachsenen Männern. Die Messwerte x_i $[cm]$ werden in einen Vektor x gespeichert :

```
>> x = [ 190.3, 176.4, 175.9, 174.8, 170.8, ...
         179.2, 175.6, 180.0, 182.1, 168.3, ...
         182.1, 171.2, 171.6, 171.6, 174.4, ...
         166.6, 181.8, 172.5, 182.7, 189.3, ...
         178.4, 173.9, 180.2, 169.0, 176.8, ...
         176.5, 175.4, 172.0, 170.0, 176.1, ...
         181.1, 174.6, 167.2, 179.5, 183.5, ...
         169.7, 158.0, 182.9, 176.9, 164.3, ...
         172.5, 184.1, 172.7, 174.7, 176.5, ...
         176.2, 173.2, 168.9, 181.1, 181.2];
```

Das Boxplot-Diagramm gibt einen Eindruck in welchem Bereich die Daten liegen und wie sie sich über diesen Bereich verteilen.

```
>> boxplot(x)
```



Values Die vertikale Achse repräsentiert die x-Werte (Körpergrössen).

Median Der Median ist das 50% Quantil, d.h. 50% der Werte x_i liegen unterhalb und 50% oberhalb des Medians.

$q_{0.25}$ unteres Quartil. Dies ist das 25% Quantil, d.h. 25% der Werte x_i liegen unterhalb $q_{0.25}$.

$q_{0.75}$ oberes Quartil, 75% Quantil

Die Box (Rechteck) unterteilt also die gegebenen Werte so, dass 50% der Werte innerhalb der Box liegen, 25% darunter und 25% darüber.

Die Differenz der beiden Werte $q_{0.75}$ und $q_{0.25}$ heisst Interquartilsabstand IQR (Interquartile Range).

min, max kleinster bzw. grösster Wert der gegebenen Daten, wobei Ausreisser (siehe unten) nicht berücksichtigt sind.

Ausreisser Als Ausreisser bezeichnet man Werte, die kleiner als

$$q_{0.25} - 1.5 \cdot IQR$$

oder grösser als

$$q_{0.75} + 1.5 \cdot IQR$$

sind.

Ausreisser können auch von Messfehlern verursacht sein. Sie werden einzeln mit dem Zeichen '+' dargestellt.

5.2 Lage- und Streuungsmasse

Für die statistische Auswertung eines Datensatzes x (z.B. die Körpergrössen) stehen die folgenden Matlab-Funktionen zur Verfügung:

Funktion	Bedeutung
min(x)	kleinster Wert (158.0 cm)
max(x)	grösster Wert (190.3 cm)
mean(x)	Mittelwert (175.69 cm)
median(x)	Median (175.75 cm)
quantile(x,p)	Quantil zum Wert p, z.B. $p = 0.25$ (unteres Quartil, 171.6 cm)
var(x)	Varianz (mittlere quadratische Abweichung vom Mittelwert) normiert mit $n-1$ für Stichproben (38.03 cm^2)
var(x,1)	Varianz normiert mit n für Gesamtheit (37.27 cm^2)
std(x)	Standardabweichung normiert mit $n-1$ für Stichproben (6.17 cm)
std(x,1)	Standardabweichung normiert mit n für Gesamtheit (6.11 cm)
sum(x)	Summe aller Werte
prod(x)	Produkt aller Werte

Der Median und die Quantile sind interpolierte Werte, Algorithmus siehe 'help quantile'.

5.3 Zufallszahlen

Zufallszahlen sind ein wichtiges Hilfsmittel für Simulationen in verschiedenen Bereichen. Matlab stellt zwei Funktionen zur Verfügung für die Erzeugung von gleich- und normalverteilten Zufallszahlen:

```
x = rand(1,1000);
y = randn(1,1000);
```

Der erste Befehl erzeugt einen Zeilenvektor mit 1000 gleichverteilten reellen Zufallszahlen im Intervall $[0, 1]$. Der erste Parameter '1' gibt die Anzahl Zeilen des Resultates an, bei Werten grösser als 1 ist das Resultat eine Matrix.

Der zweite Befehl erzeugt Zufallszahlen gemäss der Standard-Normalverteilung (siehe unten).

5.4 Verteilungen

Für graphische Darstellungen von Daten, z.B. in der Form von Balkendiagrammen, wird die *Verteilung* der Daten benötigt. Darunter versteht man die Tabelle mit den vorkommenden Werten und den zugehörigen Häufigkeiten mit denen sie vorkommen.

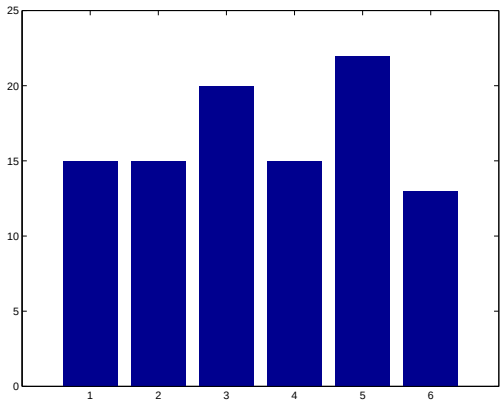
Beispiel:

Wir lassen einen Würfel $n = 100$ Mal fallen und zählen die Anzahl Vorkommen der einzelnen Augenzahlen. Beispiel eines möglichen Resultates:

Augenzahl:	1	2	3	4	5	6
Häufigkeit:	15	15	20	15	22	13

Balkendiagramm der Verteilung

Zu jeder Augenzahl 1 .. 6 wird ein Balken mit der Häufigkeit der Augenzahl als Höhe dargestellt:



Matlab stellt für Balkendiagrammme den Befehl `bar(x,y)` zur Verfügung: *bar(x,y)*

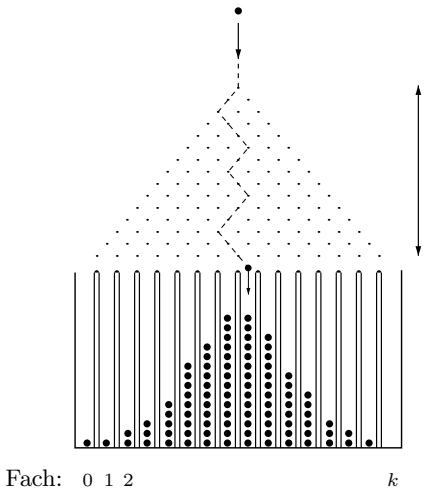
```
>> x = 1 : 1 : 6;
>> y = [15 , 15 , 20 , 15 , 22 , 13];
>> bar(x,y)
```

Das Galtonsche Brett

Bei dem Versuch von F. Galton werden Kugeln zufällig auf Fächer verteilt, und die Verteilung ist direkt sichtbar. Die Kugeln fallen auf einem vertikal aufgestellen Brett mit Nägeln hinunter.

Der Weg einer Kugel wird bestimmt durch die zufälligen Ablenkungen an den Nägeln. Auf jeder Stufe des Gitters trifft die Kugel auf einen Nagel und wird dabei um eine Position nach links oder rechts abgelenkt.

Nach der untersten Stufe fallen die Kugeln in eines der Fächer.

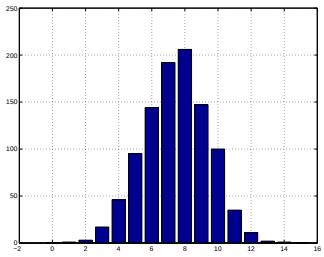


Wir lassen n Kugeln hinunterfallen. Die Verteilung der Kugeln auf die Fächer ist die Tabelle, welche zu jeder Fach-Nummer die Anzahl Kugeln des Faches enthält, z.B.

Fach-Nr:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Anzahl:	0	1	3	17	46	95	144	192	206	147	100	35	11	2	1	0

Balkendiagramm

```
>> x = 0 : 1 : 15;
>> y = [0 , 1 , 3 , 17 , 46 , 95 , 144 , 192 , 206 , 147 , 100 , 35 , 11 , 2 , 1 , 0];
>> bar(x,y)
```



Das Galton'sche Brett kann leicht mit dem Computer simuliert werden. Ein zufälliger Weg ist eine Folge von k zufälligen Nullen und Einsen, z.B. für $k = 15$

011011100101011

wobei '0' eine Linksablenkung und '1' eine Rechtsablenkung sei (vom Betrachter aus gesehen). Bei lauter Nullen fällt die Kugel in das Fach 0 und jede Rechtsablenkung erhöht die Fach-Nr. um 1. Also ergibt die Summme der Ziffern die resultierende Fach-Nr.

Eine Simulation mit einem Matlab-Programm und Zufallszahlen für die Links-/Rechtsablenkungen hat für $N = 1000$ Kugeln die oben dargestellte Verteilung ergeben.

Berechnung der Kenngrössen einer Verteilung

Gegeben sei eine Verteilung, z.B. die Häufigkeiten der Augenzahlen bei 100 Würfen eines Würfels.

x mögliche Werte (Augenzahlen 1 .. 6)

y zugehörige Häufigkeiten

Totale Anzahl Werte:	`n = sum(y)`
Mittelwert:	`xm = sum(x .* y) / n`
Varianz:	`sum((x−xm).^2 .* y / (n−1)`
Standardabweichung:	`sqrt( sum( (x−xm).^2 .* y) / (n−1))`

Erklärungen

1. Mittelwert (mittlere Augenzahl)

   Ein Mittelwert x_m ist die Summe aller Werte dividiert durch die Anzahl Werte. Da jeder Wert x_i mit der Häufigkeit y_i vorkommt, müssen bei der Berechnung der Summe die x_i mit den Häufigkeiten y_i multipliziert werden. Dazu eignet sich der Matlab-Operator '.*' für die elementweise Multiplikation der Vektoren.

2. Varianz

   Auch bei der Varianz müssen die quadratischen Abweichungen vom Mittelwert $(x_i - x_m)^2$ mit den zugehörigen Häufigkeiten y_i multipliziert werden. Im Term `x−xm` ist `x` ein Vektor und `xm` eine Zahl. Matlab erlaubt dies und subtrahiert die Zahl von jeder Komponente des Vektors, was in unserem Fall gewollt ist.

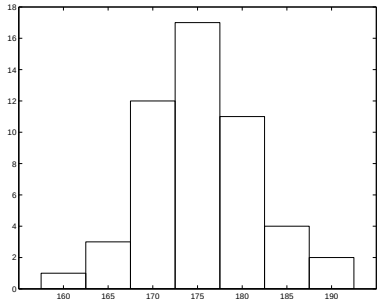
Verteilung von klassierten Daten

Bei Stichproben mit kontinuierlichen Werten wie die Körpergrössen ist für eine graphische Darstellung der Verteilung eine Einteilung der Werte in Klassen erforderlich. Dazu dient der Matlab-Befehl hist (Histogramm). Er erwartet die Werte x und die Klassenmitten in einem weiteren Vektor c : *hist*

```
>> c = [ 160, 165, 170, 175, 180, 185, 190 ];
>> hist(x,c)
```

Das Histogramm stellt die Häufigkeiten der Klassen dar. Die Werte (Körpergrössen) sind auf der horizontalen Achse dargestellt und die Häufigkeiten sind die Höhen der Rechtecke.

Verteilung der Körpergrössen:



Die erste Klasse umfasst alle Werte $x \leq 162.5$, die zweite die Werte im Intervall $162.5 < x \leq 167.5$, usw. und die letzte Klasse alle Werte $x > 187.5$.

Bemerkungen

1. Die Variante histc(x,c) anstelle von hist(x,c) interpretiert die Werte von c als Klassengrenzen anstelle von Klassenmitten (siehe 'help histc').

2. Die Funktionen hist und histc können auch als Funktionen mit Resultat aufgerufen werden:

   H = hist(x,c)

   In diesem Fall werden die Häufigkeiten der Klassen als Resultat (Vektor) zurückgegeben und die graphische Darstellung entfällt. Sie kann in einem nachfolgenden Schritt mit dem Matlab-Befehl bar(c,H) erzeugt werden.

3. Die Berechnung des Mittelwertes und der Standardabweichung aus der Verteilung H ist hier noch näherungsweise möglich, wenn man für jede Klasse die Klassenmitte als Näherungswert für die Klassse nimmt:

   Anzahl Werte: n = sum(H)

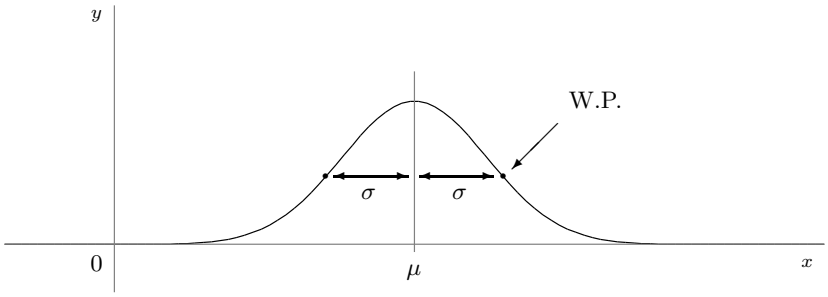
   Mittelwert: xm = sum(c .* H) / n

   Varianz: sum((c−xm).^2 .* H) / (n−1)

   Standardabweichung: sqrt (sum((c−xm).^2 .* H) / (n−1))

Die Normalverteilung

Viele Verteilungen in praktischen Beispielen haben eine glockenförmige Gestalt. Die *Normalverteilung* ist die Idealisierung für diese Verteilungen. Sie ist mathematisch durch die Gauss'sche Glockenkurve gegeben. Die Lage des Maximums der Kurve und ihre Breite bis zu den Wendepunkten sind durch die Parameter μ und σ festgelegt. Diese Parameter sind in den Anwendungen durch den Mittelwert und die Standardabweichung der Werte gegeben.



Beispiel:

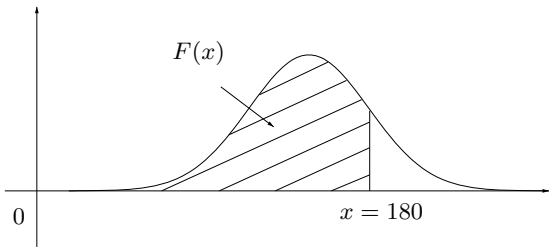
Körpergrösse x von erwachsenen Personen: $\mu = 176.5\,cm$, $\sigma = 6.8\,cm$

Berechnung von relativen Häufigkeiten

Flächen unter der Glockenkurve entsprechen theoretischen relativen Häufigkeiten der x-Werte. Die ganze Fläche ist 1, d.h. 100 % und die Fläche über einem Intervall $[a, b]$ ist die relative Häufigkeit der x-Werte in diesem Intervall.

Beispiel:

Die relative Häufigkeit der Personen mit Körpergrösse $\leq 180\,cm$ ist gleich der Fläche unter der Glockenkurve von $-\infty$ bis zu $x = 180$. Diese Fläche wird mit $F(x)$ bezeichnet.



Die Funktion $F(x)$ heisst kumulative *Verteilungsfunktion* der Normalverteilung. In Matlab heisst sie normcdf (normal cumulative distribution function) *normcdf*

$$F(x) \;=\; \mathsf{normcdf}(x, mu, sigma) \;= 0.6966 = 69.66\%$$

Dabei ist x=180, mu=176.5, sigma=6.8.

Mit der Verteilungsfunktion $F(x)$ können auch Flächen über einem Intervall $[a, b]$ als Differenzen $F(b) - F(a)$ berechnet werden:

Relative Häufigkeit der Personen mit einer Grösse $175\,cm \leq x \leq 180\,cm$

$$\mathsf{normcdf}(180, mu, sigma) - \mathsf{normcdf}(175, mu, sigma) \;=\; 28.39\,\%$$

Umgekehrte Fragestellung:

Welche Körpergrösse x wird nur von 10 % aller Personen übertroffen ?

Gesucht ist x mit

$$F(x) = 0.9$$

Dazu wird die Umkehrfunktion der Verteilungsfunktion $F(x)$ benötigt, welche in Matlab norminv heisst: *norminv*

$$x = \mathsf{norminv}(0.9, mu, sigma) \;=\; 185.21\,cm$$

6 Vektor-Geometrie

Vektoren eignen sich auch für geometrische Anwendungen in der Ebene und v.a. im Raum.

6.1 Zeilen- und Spaltenvektoren

Für Anwendungen in der Geometrie und Physik werden die Vektoren häufig als Spaltenvektoren (statt Zeilenvektoren) geschrieben, was logisch auf dasselbe hinauskommt. Für die Definition eines Spaltenvektors in Matlab werden die Werte durch *Strichpunkte* (statt Kommas) getrennt:

```
>>  x = [ 2.5 ; 1.4; 9.8 ];
```

Matlab-Antwort:

```
x =
    2.5000
    1.4000
    9.8000
```

Umwandlung Spaltenvektor in Zeilenvektor und umgekehrt (Transposition):

`y = x'` Transposition

In den algebraischen Operationen '+' und '−' müssen beide Operatoren entweder Zeilen- oder Spaltenvektoren sein (nicht gemischt).

6.2 Skalar- und Vektorprodukt

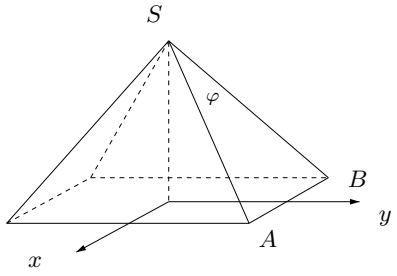
Für geometrische Anwendungen sind das Skalar- und das Vektorprodukt, sowie die Länge (Norm) eines Vektors wichtig:

Matlab	Bedeutung
dot(a,b)	Skalarprodukt (Dotproduct)
cross(a,b)	Vektorprodukt (Crossproduct) für dreidimensionale Vektoren
norm(a)	Länge (Norm)

Beispiel:

Cheops-Pyramide

Die Cheops-Pyramide hat eine Höhe $h = 146\,m$ und eine quadratische Grundfläche mit Kantenlänge $230\,m$. Berechnen Sie den Winkel φ zwischen zwei Seitenkanten bei der Spitze.



$h = 146, \quad a = 230/2 = 115$

$A(a, a, 0), \quad B(-a, a, 0), \quad S(0, 0, h)$

$\vec{u} = \overrightarrow{SA} = A - S, \qquad \vec{v} = \overrightarrow{SB} = B - S$

$$\varphi = \arccos\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|}\right)$$

Matlab:

```
h = 146;
a = 115;
A = [a,a,0];
B = [-a,a,0];
S = [0,0,h];
u = A - S;
v = B - S;
phi = acosd(dot(u,v) / (norm(u)*norm(v))
```

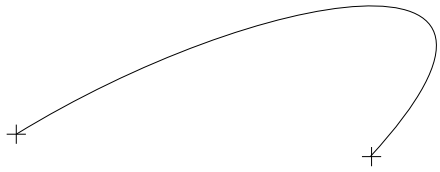
Der Winkel ist 63.5^o, d.h. die Seitendreiecke sind nicht gleichseitig.

6.3 Bézier-Kurven

Die Bézier-Kurven wurden von Pierre Etienne Bézier, einem Automobil-Designer bei der Firma Renault, um 1960 eingeführt. Kurz zuvor hatte sie schon Paul de Faget de Casteljau bei Citroën entdeckt, jedoch aus Gründen der Geheimhaltung nicht veröffentlicht.

Man unterscheidet quadratische und kubische Bézier-Kurven. Sie werden mit Teilungspunkten von Strecken definiert. Wir beschränken uns auf Kurven in der Ebene.

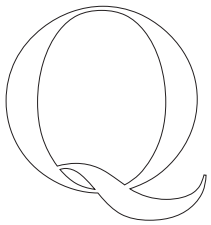
Beispiel einer kubischen Bézier-Kurve



Einsatzbeispiele:

- Zeichnungsprogramme (CorelDraw)
- Computer Aided Geometric Design (CAGD)
- Schriften

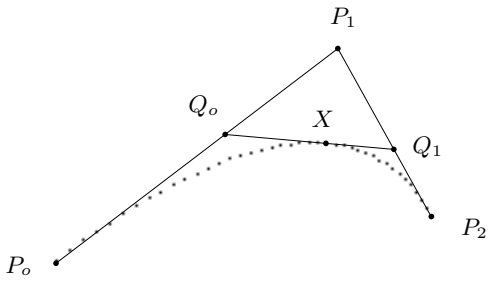
  Die Umrisskurven der Buchstaben von Postscript-Schriften sind durch mehrere aneinandergefügte kubische Bézier-Kurven definiert.



Quadratische Bézier-Kurven

Gegeben sind drei Punkte P_o, P_1, P_2 in der Ebene (Kontrollpunkte).

Zu einem Parameter t im Intervall $[0, 1]$ bilden wir die folgenden Teilungspunkte Q_o, Q_1 und X:



$$\begin{aligned} Q_o &= P_o + t \cdot \overrightarrow{P_o P_1} \\ Q_1 &= P_1 + t \cdot \overrightarrow{P_1 P_2} \\ X &= Q_o + t \cdot \overrightarrow{Q_o Q_1} \end{aligned}$$

Der so definierte Punkt X wird mit $X(t)$ bezeichnet. Lässt man t von 0 bis 1 laufen, so überstreichen die Punkte $X(t)$ eine Kurve. Dies ist die von den drei Punkten P_o, P_1 und P_2 bestimmte *Bézier-Kurve*.

Aus der Definition ist ersichtlich, dass die Kurve im Punkt P_o tangential an die Strecke P_oP_1 und im Punkt P_3 tangential an P_2P_3 ist. Weiter verläuft sie im Viereck $P_oP_1P_2P_3$, sofern die Punkte in dieser Reihenfolge ein konvexes Viereck bilden.

Der Punkt $X(t)$ kann als Linearkombinationn von P_o, P_1 und P_2 dargestellt werden:

$$\boxed{X(t) = (1-t)^2 \cdot P_o + 2t(1-t) \cdot P_1 + t^2 P_2}$$

Da die Koeffizienten Polynome zweiten Grades sind, heisst die Kurve 'quadratische' Bézier-Kurve.

Herleitung:

$$\begin{aligned} Q_o &= P_o + t \cdot (P_1 - P_o) &&= (1-t) \cdot P_o + t \cdot P_1 \\ Q_1 &= P_1 + t \cdot (P_2 - P_1) &&= (1-t) \cdot P_1 + t \cdot P_2 \\ X &= Q_o + t \cdot (Q_1 - Q_o) &&= (1-t) \cdot Q_o + t \cdot Q_1 \end{aligned}$$

Durch Einsetzen folgt die Behauptung.

Kubische Bézier-Kurven

Eine kubische Bézier-Kurve ist gegeben durch vier Kontrollpunkte

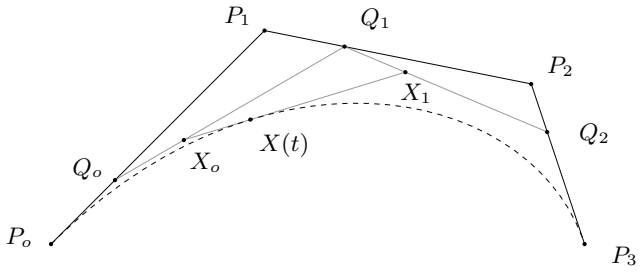
$$P_o, \quad P_1, \quad P_2, \quad P_3$$

in der Ebene. Der Punkt $X(t)$, $0 \leq t \leq 1$, der kubischen Bézier-Kurve ist definiert durch

$$X(t) \; = \; X_o + t \cdot \overrightarrow{X_o X_1}$$

mit:

- X_o Punkt der quadratischen Bézier-Kurve $P_0 P_1 P_2$ zum Parameter t
- X_1 Punkt der quadratischen Bézier-Kurve $P_1 P_2 P_3$ zum Parameter t



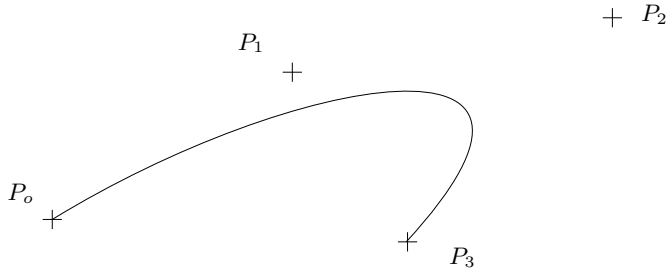
Die Bézier-Kurve besteht aus den Punkten $X(t)$ für $0 \leq t \leq 1$.

Darstellung von $X(t)$ als Linearkombination von P_o bis P_3:

$$\boxed{X(t) = (1-t)^3 \cdot P_o + 3t(1-t)^2 \cdot P_1 + 3t^2(1-t)P_2 + t^3 P_3}$$

Darstellung der Kurve mit Matlab

```
p0 = [34;111];
p1 = [109;157];
p2 = [209;174];
p3 = [145;104];
t = 0:0.01:1;
x = (1−t).^3 * p0(1) + 3*t.*(1−t).^2 * p1(1) + 3*t.^2.*(1−t) * p2(1) + t.^3 * p3(1);
y = (1−t).^3 * p0(2) + 3*t.*(1−t).^2 * p1(2) + 3*t.^2.*(1−t) * p2(2) + t.^3 * p3(2);
plot(x,y);
axis equal
```



7 Matrizen und Gleichungssysteme

Eine *Matrix* ist ein rechteckförmiges Schema von reellen Zahlen. Matrizen werden normalerweise mit grossen Buchstaben bezeichnet.

Matrix mit 2 Zeilen und 3 Spalten (2×3-Matrix):

$$A = \begin{pmatrix} 2 & 5 & -3 \\ 8 & 4 & -1 \end{pmatrix}$$

Anstelle der runden Klammern verwendet man häufig auch eckige Klammern. Für Beispiele verwenden wir zur Vereinfachung der Notation meistens ganze Zahlen.

Matlab-Definition:

```
>> A = [ 2, 5, -3; 8, 4, -1]
```

Die Elemente einer Zeile werden mit Kommas oder Blanks getrennt, ein Strichpunkt bedeutet das Ende einer Zeile. Die Werte einer Matrix werden wie folgt angesprochen:

`A(2, 3)`	Element in Zeile 2, Spalte 3
`A(1, :)`	erste Zeile (Zeilenvektor)
`A(:, 3)`	dritte Spalte (Spaltenvektor)

Spezielle Matrizen

Matrix	Matlab	Bezeichnung
$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	`zeros(3,3)`	Nullmatrix
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	`eye(3)`	Einheitsmatrix
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	`ones(3,3)`	–
$\begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix}$	`diag([5,3,4])`	Diagonalmatrix

Der Befehl `diag` erhält als Parameter einen Vektor [...] mit den Diagonalelementen der Matrix.

Abfrage der Zeilen- und Spaltenzahl einer Matrix

`size(A,1)`	Anzahl Zeilen von `A`
`size(A,2)`	Anzahl Spalten von `A`

Lineare Gleichungssysteme

Matrizen eignen sich für die Darstellung und Bearbeitung von linearen Gleichungssystemen.

System von drei linearen Gleichungen mit drei Unbekannten:

$$\begin{aligned} 2x &- 3y + 4z = 5 \\ 4x &+ 2y - 8z = 8 \\ 3x &+ 6y - 5z = 1 \end{aligned}$$

Die Koeffizienten der Unbekannten und die Konstanten der rechten Seiten werden in eine 3×4 Matrix eingetragen:

$$A = \begin{pmatrix} 2 & -3 & 4 & 5 \\ 4 & 2 & -8 & 8 \\ 3 & 6 & -5 & 1 \end{pmatrix}$$

Dies ist die *Matrix* des Systems.

Auflösung eines Gleichungssystems

Ein lineares Gleichungssystem kann mit dem *Gauss'schen Algorithmus* gelöst werden. Dies ist ein Eliminationsverfahren, bei dem die Unbekannten systematisch eliminiert werden, bis die erste Gleichung nur noch die erste Unbekannte enthält, die zweite nur die zweite usw. *Gauss-Algorithmus*

Dies ist die sogenannte *Diagonalform*, in der die Lösungen des Systems direkt abgelesen werden können. Alle Schritte des Gauss'schen-Algorithmus sind *Äquivalenzumformungen,* d.h. sie verändern die Lösungsmenge des Systems nicht. Somit ist die Lösung des Diagonalsystems auch die des ursprünglichen.

Der Matlab-Befehl für den Gauss'schen Algorithmus ist `rref` (row reduced echelon form). *rref*

Beispiel:

$$\begin{aligned} 2x &- 3y + 4z = 5 \\ 4x &+ 2y - 8z = 8 \\ 3x &+ 6y - 5z = 1 \end{aligned} \qquad A = \begin{pmatrix} 2 & -3 & 4 & 5 \\ 4 & 2 & -8 & 8 \\ 3 & 6 & -5 & 1 \end{pmatrix}$$

Matlab:

```
>> B = rref(A)
```

Resultat:

$$B = \begin{pmatrix} 1 & 0 & 0 & 1.7375 \\ 0 & 1 & 0 & -1.0250 \\ 0 & 0 & 1 & -0.3875 \end{pmatrix}$$

Zugehöriges Gleichungssystem (Diagonalsystem):

$$\begin{aligned} x &= 1.7375 \\ y &= -1.0250 \\ z &= -0.3875 \end{aligned}$$

Da dieses System äquivalent zum gegebenen ist, ist dies auch die Lösung des ursprünglichen Systems.

8 M-Files

Ein *M-File* ist ein Text-File, welches Matlab-Befehle enthält. Der Name des Files muss die Endung '.m' haben. Die Files heissen daher 'M-Files'.

Man unterscheidet zwei Arten von M-Files, *Scripts* und *Functions*. Die letzteren haben wie die mathematischen Funktionen Parameter und ein Resultat.

M-Files werden in einem beliebigen Directory abgespeichert, und das Directory wird mit dem Menu 'File/SetPath' zum Matlab-Path hinzufügt.

Octave/Windows:

```
>> addpath("c:/myScripts")
>> savepath
```

Achtung: Pfad in Unix-Notation, d.h. Slash '/' nicht Backslash '\'

Mit `rmpath("c:/myScripts")` kann das Directory wieder aus dem Pfad entfernt werden (remove path). Weitere Informationen, siehe Help-System (z.B. `help addpath`).

Octave/Android:

Beim Starten des Apps wird bei den Startmeldungen angegeben, in welchem Directory das M-File gespeichert werden muss.

Ausführung eines M-Files

Das M-File wird mit dem Namen des Files ohne '.m' als Befehl im Command-Window von Matlab, oder in einem anderen M-File ausgeführt.

Achtung:

Der Filename eines M-Files darf weder Sonderzeichen noch Umlaute noch Blanks enthalten und muss mit einem Buchstaben A-Z, a-z (nicht mit einer Ziffer) beginnen. Sonst gibt es Probleme beim Aufruf des Scripts.

Kommentare

Ein M-File kann neben Matlab-Befehlen Erklärungen für den Leser, sogenannte Kommentare, enthalten. Diese werden von Matlab nicht verarbeitet.

Ein Kommentar wird mit dem Kommentarzeichen '`%`' eingeleitet. Alles was hinter diesem Zeichen bis zum Zeilenende steht wird von Matlab nicht verarbeitet.

```
n = 10                  % Anzahl Werte
```

8.1 Scripts

Ein Script ist einfach ein File, welches eine Folge von Matlab-Befehlen enthält. Bei der Ausführung des Scripts werden die Befehle der Reihe nach ausgeführt, wie wenn sie direkt im Command-Window eingegeben würden.

Wir betrachten ein typisches Beispiel.

Umlaufzeit eines Satelliten

Ein Satellit bewege sich auf einer Kreisbahn um die Erde

Gegeben: $h = 36'000\,km$ Höhe über der Erdoberfläche (Fernseh-Satelliten)

Gesucht: T Umlaufzeit

Formel (Physik):

$$T = \frac{2\,\pi}{r_{Erde}} \cdot \sqrt{\frac{r^3}{g}}$$

$g = 9.81\,m/s^2$ Erdbeschleunigung auf der Erdoberfläche
$r = r_{Erde} + h$ Bahnradius
$r_{Erde} = 6372 \cdot 10^3\,m$ Erdradius

Matlab-Script

```
g = 9.81;                          % Erdbeschleunigung [m/s^2]
rErde = 6372e3;                    % Erdradius [m]
h = 36000e3;                       % Hoehe ueber Erdoberflaeche [m]
r = rErde + h;                     % Bahnradius [m]
T = 2*pi*sqrt(r^3/g) / rErde;      % Umlaufzeit [sek]
T = T / 3600                       % Umlaufzeit [h]
```

Das File wird mit dem Namen 'sat.m' abgespeichert und mit dem Befehl

```
>> sat
```

aufgerufen.

Resultat: 24.1205

Die Umlaufzeit beträgt also einen Tag, wie die der Erde. Wenn sich der Satellit in der Äquatorebene der Erde bewegt, rotiert er synchron mit der Erde und erscheint daher für einen Beobachter auf der Erde immer am selben Ort, wie es für einen Fernsehsatelliten sein muss (geostationärer Satellit).

Benutzer-Eingaben

Mit dem Befehl `input` werden Daten vom Benutzer eingelesen. Der Befehl erhält als Parameter einen Textstring, der als Anweisung für den Benutzer ausgegeben wird. Die eingelesenen Werte werden als Resultat (Matrix) zurückgegeben. *input*

```
x = input('Vektor eingeben: ');
```

Eine bessere Alternative ist die Übergabe von Daten als Parameter. Dazu führen wie *Function-Files* ein.

8.2 Functions

Eine *Function*, definiert in einem File, wird wie eine mathematische Funktion mit Parametern (Argumenten) aufgerufen und kann ein Resultat zurückgeben. Das Kennzeichen einer Function ist das Schlüsselwort function am Anfang des Files. Dadurch wird die Definition einer Funktion mit Parametern und Resultat eingeleitet. *function*

Beispiel: Funktion für die Umlaufszeit eines Satelliten

```
function T = sat(h)
  g = 9.81;                    % Erdbeschleunigung
  rErde = 6372e3;              % Erdradius [m]
  r = rErde + h;               % Bahnradius
  T = 2*pi*sqrt(r^3/g)/rErde;  % Umlaufzeit in sek
  T = T / 3600;                % Umlaufzeit in h
end
```

Aufruf der Funktion

Die Funktion wird wie eine Funktion von Matlab mit Übergabe der Parameter und der Speicherung des Resultates in einer Variabeln aufgerufen:

```
>> umlaufzeit = sat(36000e3)
```

Allgemeines Format eines Function-Files

Ein Function-File muss mit dem Schlüsselwort function beginnen, gefolgt von der sogenannten *Signatur* oder *Spezifikation* der Funktion. Diese besteht aus dem Namen der Funktion, der Parameterliste und einem Namen für das Resultat.

```
function resultat = name(parameter1, parameter2, ...)
  ...
end
```

Name der Funktion

Dies ist ein beliebiger Name für die Funktion, mit dem sie aufgerufen wird. Dieser muss mit dem Filenamen (ohne '.m') entsprechen.

Parameterliste

In der Parameterliste werden die Input-Parameter (Argumente) der Funktion definiert. Dazu wird für jeden gewünschten Parameter ein Name angegeben:

```
(parameter1, parameter2, ...)
```

Diese Namen werden innerhalb der Funktion verwendet, um die Parameter anzusprechen. Wenn mehrere Parameter definiert werden, sind sie mit Kommas zu trennen.

Technisch gesehen, wird für jeden Parameter eine sogenannte *lokale Variable* mit dem angegebenen Namen definiert, welche nur innerhalb der Funktion sichtbar ist. *lokale Variabeln*

Beim Aufruf der Funktion muss für jeden Parameter ein Wert übergeben werden. Diese Werte werden in die lokalen Variabeln der Parameter kopiert. Dabei wird der erste Wert in den ersten Parameter kopiert, der zweite in den zweiten usw. Die übergebenen Werte werden mit Kommas getrennt (wie in der Definition der Parameter).

Wenn die Funktion keine Parameter hat, entfällt die Parameterliste.

Resultat

Für das Resultat (Rückgabewert) der Funktion wird vor dem Namen der Funktion ein Name gewählt. Zu diesem wird wie für die Parameter eine lokale Variable definiert. In der Funktion wird der Wert des Resultates in diese Variable gespeichert.

Am Ende der Funktion (end-Statement) wird der Inhalt der lokalen Variabeln *resultat* als Rückgabewert an das aufrufende Programm zurückgegeben.

Wenn die Funktion keinen Rückgabewert hat, entfällt die Resultat-Definition. Die Funktion wird dann ohne Zuweisung des Resultates aufgerufen.
Beispiele: Graphik-Funktionen

Lokale Variabeln

In der Funktion können weitere Variabeln definiert werden. Diese sind wie die Parameter und das Resultat *lokale Variabeln* der Funktion, d.h. sie sind nur *in* der Funktion bekannt. Sie werden (wie die Parameter und das Resultat) bei jedem Aufruf der Funktion im Speicher angelegt und am Ende der Funktion gelöscht.

Ergänzungen zu Funktionen

1. Viele Matlab-Funktionen, z.B. cross (Vektorprodukt) sind als solche Function-Files implementiert (im Directory 'Toolbox').

2. Mehrfachaufrufe einer Funktion

   Wenn eine Funktion das erste Mal aufgerufen wird, wird sie interpretiert und in den Speicher geladen. Sie bleibt im Speicher, sodass nachfolgende Aufrufe weniger aufwendig sind.

   Die lokalen Variabeln der Funktion werden jedoch bei jedem Aufruf neu angelegt, sodass keine Werte von einem Aufruf zum nächsten darin gespeichert werden können.

3. Funktionen mit mehreren Resultaten

   Funktionen können auch mehrere Resultate zurückgeben. Diese werden in eckigen Klammern definiert:

   ```
   function [y1, y2] = f(x1, x2, x3)
     ...
   end
   ```

   Aufruf der Funktion:

   ```
   [a1, a2] = f(b1,b2,b3)
   ```

4. Mehrere Funktionen in einem File

   In einem M-File, welches eine Funktion definiert, können nach der Hauptfunktion weitere Funktionen definiert werden, welche in der Hauptfunktion aufgerufen werden. Diese Unterfunktionen sind nur innerhalb des betreffenden Files bekannt.

```
function y =f(x)
end

function u = g(v)
end
```

   Dies geht in einem Script-File nicht. Das betreffende Script-File kann jedoch für diesen Zweck mit einem function Statement in eine Funktion ohne Parameter und ohne Resultat umgewandelt werden.

Anwendungsbeispiel: Eigene Sinus-Funktion

Wie berechnet ein Taschenrechner die Funktionen Sinus, Cosinus, Logarithmus usw.? Antwort: Die Funktionen können mit Polynomen der Form

$$p(x) = a_o + a_1\,x + a_2\,x^2 + \cdots + a_n\,x^n$$

approximiert werden. Diese benötigen zur Berechnung nur die Grundoperationen Addition und Multiplikation.

Für Winkel x im Intervall $-\frac{\pi}{6} \le x \le \frac{\pi}{6}$ (x in rad) gilt mit einer Genauigkeit von vier Dezimalstellen:

$$\sin(x) \approx x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} \tag{1}$$

Dies ist das Taylor-Polynom 7. Grades für die Sinus-Funktion im Nullpunkt.

Sei jetzt x ein Winkel von 0 bis $\frac{\pi}{2}$ (90^o).

Mit (1) kann der Sinus von $x_1 = x/3$ berechnet werden, und damit erhält man mit der nachfolgenden trigonometrischen Formel (2) für den dreifachen Winkel den Sinus von x.

$$\sin(3x) = 3\sin(x) - 4\sin^3(x) \tag{2}$$

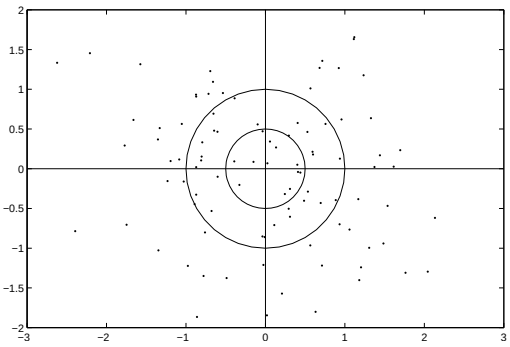
Matlab-Function:

```
function y = mySin(x)
  x1 = x/3;
  y1 = x1 - x1^3/6 + x1^5/120 - x1^7/5040;
  y = 3 * y1 - 4 * y1^3;
end
```

Filename: mySin.m
Aufruf: y = mySin(pi/6)

9 Workshop

Simulation von Schüssen (Normalverteilung)



Lösung:

Koordinatenachsen

x-Achse: $A(-3,0)$, $B(3,0)$
y-Achse: $C(0,-3)$, $D(0,3)$

Erzeugung der Schüsse

Die x- und die y-Koordinaten sind normalverteilt. Erzeugen Sie zwei Vektoren x und y mit je 100 normalverteilten Zufallszahlen mit der Funktion `randn`. Dies ergibt normalverteilte Werte mit Erwartungswert $\mu = 0$ und Standardabweichung $\sigma = 1$.

Zielscheibe

Die Kreise (Radien 1 bzw. 0.5) werden als regelmässige n-Ecke gezeichnet. Eckpunkt-Koordinaten:

```
ex = r * cosd(phi)
ey = r * sind(phi)
```

Dabei ist `phi` ein Vektor mit den Winkeln 0^o, 10^o, 20^o, ... , 360^o :

```
phi = 0:10:360
```

Streuung

Variieren Sie die Streuung durch Multiplikation der Zufallszahlen mit einem Wert σ (Standardabweichung der Normalverteilung).

Wahrscheinlichkeiten

Die theoretischen Wahrscheinlichkeiten der Normalverteilung können mit der Verteilungsfunktion `normcdf` (normal cumulative distribution function) der Normalverteilung berechnet werden.

Wahrscheinlichkeit p, dass die x-Koordinate eines Schusses kleiner oder gleich einem Wert x ist:

```
p = normcdf(x,mu,sigma)
```

mu	Erwartungswert μ (in unserem Fall 0)
sigma	Standardabweichung σ

Beispiele ($\mu = 0$ und $\sigma = 1$) :

1. Wahrscheinlichkeit, dass $x \leq 1.5$:

   p = normcdf(1.5, 0, 1) [93.3 %]

2. Wahrscheinlichkeit p, dass ein Schuss in einem Rechteck $[a, b] \times [c, d]$ liegt:

   p1 = normcdf(b, 0, 1)−normcdf(a, 0, 1)

   p2 = normcdf(d, 0, 1)−normcdf(c, 0, 1)

   p = p1 * p2

10 Anhang: Matlab-Alternativen

Die folgende Aufzählung erhebt keinen Anspruch auf Vollständigkeit.

Windows

- *Octave*

  Das Softwareprodukt *Octave* ist weitgehend äquivalent zu Matlab und steht kostenlos zur Verfügung.

  Website: http://gnu.org/software/octave/download.html

  32-Bit Windows: octave−4.2.1−w32−installer.exe (170 MB)

  64-Bit Windows: octave−4.2.1−w64−installer.exe

  Weitere Informationen, siehe

  http://wiki.octave.org/Octave_for_Microsoft_Windows

Linux

- *Octave*

  Wie Windows-Version. Für weitere Informationen verweisen wir auf die oben aufgeführte Website.

Android (Tablets/Smartphones)

- Mathmatiz

  Installation: Standardinstallation von Google Play Store

  Die Installation und Verwendung des Apps sind problemlos.

  Tutorials:

  - Basic Operations
    https://www.youtube.com/watch?v=9B4wCIChDGs
  - Script Program Demo
    https://www.youtube.com/watch?v=j-AaTkLAskY
  - Matrix Operations
    https://www.youtube.com/watch?v=WyWAEnh27JA
  - Function Program Demo
    https://www.youtube.com/watch?v=qnOsihC2hnQ
  - Plotting Demo
    https://www.youtube.com/watch?v=IdnNdU_3Xhg

  Command-History:

  Zur Wiederholung eines vorangegangenen Befehls kann dieser in der Liste angetippt werden, worauf ein Menu erscheint. Auf diesem ‘copy expression’ wählen.

– GNURoot Octave

  Dies ist die Portierung von Octave auf Android. Leider ist die Installation relativ aufwendig. Zuerst muss das App *GNURoot Debian* installiert werden. Dies ist ein Linux-System, welches denselben Linux-Kernel wie das Android-System verwendet.

  Anschliessend kann die App *GNURoot Octave* installiert werden. Diese läuft dann in dem Linux-System GNURoot Debian, wovon der Anwender wenig merkt. Sehr empfehlenswert dazu ist die (kostenlose) App *Hacker's Keyboard*, welche eine von Windows gewohnte Tastatur mit den Pfeiltasten ermöglicht (für Wiederholung und Modifikation von Befehlen). Diese Tastatur ist auch für andere Situationen sehr günstig.

  Zur Aktivierung des Keyboards die App starten und die Anweisungen befolgen (Enable keyboard, Set input method).

  Das Keyboard kann jederzeit mit 'Settings/Language & input' inaktiviert und wieder aktiviert werden.

iOS (iPads/iPhones)

– SIMO

  Die App kostet 10 Fr. Sie verwendet vier Windows. Die ersten beiden sind für Scripts, das dritte ist das Command-Window und das vierte das Graphik-Window.