Freie Universität Berlin

MATLAB - Grundlagen und Anwendungen (Teil II)

Mentoring WiSe 2018/19

Maren Fanke, Alexandra Wesolek, Alexander Korzec
Freie Universität Berlin

18/10/2018

Outline

Inhalt

Standardfunktionen in MATLAB

- MATLAB besitzt bereits einige integrierte Standardfunktionen.
- Bei diesen *Standardfunktionen* handelt es sich um Unterprogramme, die Funktionalitäten übernehmen, welche über einfache Elementaroperationen hinausgehen (bspw. `cos` oder `mod`).
- Mathematische Standardfunktionen können nicht nur auf Skalare, sondern auch auf Vektoren und Martizen angewendet werden. Die Anwendung der Funktion geschieht dabei elementeweise auf die einzelnen Einträge der Matrix.
- Neben mathematischen Standardfunktionen gibt es noch andere Arten von Funktionen wie bspw. die Ausgabefunktion `display` oder die Matrixfunktion `ones`.

Funktionen

Beispiel:

```
a = 7;
b = 3;
v = [−2 6 3];
A = [5 3 2; −1 6 2; 3 −5 9];


display('|v| =');            % Setzt einzelne Vektoreinträge in Betragsfunktion.
display(abs(v));

display('mod(a,b) =');       % Gibt Rest nach Division von a durch b an.
display(mod(a,b));

display('sqrt(a) =');        % Wurzelfunktion (Äquivalent zu ^0.5 bzw.
display(sqrt(a));            % .^0.5 für Vektoren und Matrizen.

display('sin(A), cos(A)');   % Setzt einzelne Matrixeinträge in Sinus- bzw.
display(sin(A));             % Cosinusfunktion.
display(cos(A));

display('exp(A)');           % Exponentialfunktion.
display(exp(A))
```

Funktionen

Neben den Standardfunktionen, bietet MATLAB auch die Möglichkeit eigene Funktionen zu schreiben.

Mathematik

- Eine Funktion f ist eine Abbildung, die jedem Element x einer Definitionsmenge M ein Element y einer Zielmenge N zuordnet.

$$f : M \to N,\ x \mapsto y$$

Funktionen

MATLAB

- Selbst erstellte Funktionen in MATLAB werden **Funktion-Dateien** (Function-Files) genannt.
- Funktion-Dateien haben in MATLAB folgenden Aufbau:

```
function [Rueckgabewert] = funktionsname (Eingabeparameter)
<Anweisung 1>     % Anweisungsblock
<Anweisung 2>
...
end
```

- Kennzeichnung der Datei als Funktion durch Funktionskopf:

```
function [Rueckgabewert] = funktionsname (Eingabeparameter)
```

- Aufruf und Ausführung der Funktion in Command Window:

ohne Variablenzuweisung

```
funktionsname (Eingabeparameter)
```

mit Variablenzuweisung

```
var = funktionsname (Eingabeparameter)
```

Erstellen einer Funktion-Datei

1. Erstellen Sie eine **Funktion-Datei**, indem Sie an einer leeren Stelle im Datei-Explorer (`Current Folder`) einen Rechtsklick ausführen und anschließend die entsprechende Dialogoption auswählen.

   Benennen Sie die Datei: `flaeche.m`

   Die Datei wird im aktiven Verzeichnis angelegt, und befindet sich somit direkt am richtigen Ort.
2. Doppelklick auf die neu erstellte Datei, um diese zu öffnen.
3. Die Datei `flaeche.m` sollte nun wie folgt aussehen:

```
function [ output_args ] = flaeche( input_args )
%FLAECHE Summary of this function goes here
%   Detailed explanation goes here


end
```

Beispiel 1:
Es soll eine Funktion aufgestellt werden, die durch Angabe zweier Seitenlängen *a* und *b* sowohl die Fläche eines Rechtecks als auch die Fläche eines Quadrates berechnet.

```
% Die Funktion berechnet die Fläche von Rechtecken und Quadraten mit
% unterschiedlichen Seitenlängen.
function [A] = flaeche(a,b)      % Beginn Funktion
                                 % Funktionsname: flaeche
                                 % Eingabeparameter: a, b
                                 % Ausgabeparameter: A
          A = a*b;               % Funktion soll mit Eingabeparameter a, b
                                 % die Fläche von Rechtecken und Quadraten
                                 % berechnen.
end                              % Ende der Funktion
```

Funktionsaufruf

```
>> rechteck = flaeche(3,4)
rechteck =
     12
```

Freie Universität Berlin

Beispiel 2:

Es soll nun eine Funktion aufgestellt werden, die zu gegebenen Zahlen *x, y* das Produkt und die Summe berechnet.

```
% Die Funktion berechnet das Produkt und die Summe zweier Zahlen.

function [produkt,summe]=prodsum(x,y) % Beginn Funktion
                                      % Funktionsname: prodsum
                                      % Eingabeparameter: x, y
                                      % Ausgabeparameter: produkt, summe

        produkt=x*y;                  % Funktion soll mit Eingabeparameter
                                      % a und b das Produkt und
        summe=x+y;                    % die Summe aus a und b berechnen

end                                   % Ende der Funktion
```

Funktionsaufruf

```
>> [p,s]=prodsum(5,3)
p =
   15
s =
   8
```

Bemerkung

- ***Wichtig! Funktionsname und Dateiname müssen immer übereinstimmen.***
- *Namenskonflikte mit bereits vorhandenen (Standard-) Funktionen in MATLAB sollten vermieden werden. Dies gilt auch für das Speichern von Skript-Dateien.*
  Bspw. sollte eine erstellte Funktion- oder Skript-Datei nicht den Namen `cos.m` *oder* `plot.m` *bekommen, da sich diese durch Doppelbelegung nicht mehr ausführen lassen!*
- *Funktionsnamen werden klein geschrieben.*
- *Namen von Funktionen beginnen, wie Variablennamen, mit einem Buchstaben gefolgt von einer beliebigen Anzahl an Buchstaben, Zahlen oder Unterstrichen.*
- *Funktionen können mehrere Eingabe- und Ausgabeparameter haben.*

Funkionen-Handles

- Weiterer Variablentyp.
- In ihnen können Funktionen abgespeichert werden ohne extra eine Funktion-Datei anlegen zu müssen.
  - **Einfach strukturierte Funktionen** können somit direkt im Code definiert werden.
- Neben Variablen ist es möglich einer Funktion eine andere Funktion als Parameter zu übergeben.

Beispiel Funktionen-Handle:

```
% Funktionen-Handle auf eine existierende Funktion.
f = @cos;          % Cosinus.

% Funktionen-Handle auf eine neue Funktion mit einer Eingabevariable.
g = @(x) x.^2;     % Quadrat.

% Funktion, die eine andere Funktion als Parameter nimmt.
F = @f(1);         % Werte f an 1 aus.
```

Funktionsaufruf

```
display(f(pi));

display(g(1:5));

display(F(g));
```

Inhalt

Bedingungen

Mathematik

In der Mathematik können Bedingungen unterschiedliche Formen haben:

- **Wenn-Dann-Aussagen** beschreiben, ob aus einer Aussage eine andere Aussage folgt.Hierbei kann zwischen *notwendiger* und *hinreichender Bedingung* unterschieden werden.

  Beispiel:

  A: „Es hat geregnet“, B: „Die Straße ist nass“

  A ⇒ B: „Wenn es geregnet hat, dann ist die Straße nass.“

Informatik

- Bedingungen stellen in der Informatik eine Art Fallunterscheidung dar, bei der ein bestimmter Programmabschnitt nur unter einer bestimmten Bedingung ausgeführt wird. Sie werden mit der `if-else`-Anweisung beschrieben.

Bedingungen: if-Anweisung

MATLAB

- Mit der if-else-Anweisung können unterschiedliche Fallunterscheidungen durchgeführt werden.

Syntax

```
if <Bedingung 1>
  <Anweisung 1>  % Wenn Bedingung 1 wahr ist, wird Anweisung 1 ausgeführt.
elseif <Bedingung 2>
  <Anweisung 2>  % Wenn Bedingung 1 falsch ist, wird Anweisung 2 ausgeführt.
else
  <Anweisung 3>  % Wenn Bedingung 1 und 2 falsch sind, wird Anweisung 3
end              % ausgeführt.
```

Bedingungen: if-Anweisung

Bemerkung

- *Die **Bedingungen** werden Prädikate genannt und haben einen booleschen Wert (0 für `false` und 1 für `true`).*
- *Mit Hilfe von **`elseif`** lassen sich beliebig viele Fälle erstellen. Seine Benutzung hängt von der Art des Programms ab.*
- *Die Abarbeitungsreihenfolge ist von oben nach unten. Trifft ein Fall zu, wird nur seine Anweisung befolgt.*

Beispiel:

Es soll die bedingte Funktion:

$$f(x) \quad := \quad \begin{cases} -1 & \textit{falls} \quad x < 0 \\ 0 & \textit{falls} \quad x = 0 \\ +1 & \textit{falls} \quad x > 0 \end{cases}$$

programmiert werden.

Bedingungen: if-Anweisung

Beispiel:

$$f(x) \quad := \quad \begin{cases} -1 & \textit{falls} \quad x < 0 \\ 0 & \textit{falls} \quad x = 0 \\ +1 & \textit{falls} \quad x > 0 \end{cases}$$

Bemerkung

In deutscher Sprache:
Falls x kleiner ist als Null, dann ist $f(x)$ *gleich* -1*. Falls x größer ist als Null, dann ist* $f(x)$ *gleich* 1*. Ansonsten ist* $f(x)$ *gleich* 0*.*

In englischer Sprache:
If x is smaller than zero $f(x)$ *is equal to* -1*. If x is bigger than zero* $f(x)$ *is equal to* 1*. Otherwise* $f(x)$ *is equal to* 0*.*

Bedingungen: if-Anweisung

Beispiel: Initialisierung Variable

```
x = 2;                  %   Initialisiere x mit 2.

if x<0
     y =−1;
     display('y = -1');
  elseif x>0
     y = 1;
     display('y = 1');
  else
     y = 0;
     display('y = 0');
  end
```

Bedingungen: if-Anweisung

Beispiel: Funktion

```
function [ y ] = stueckfkt( x )
if x<0
      y =−1;
   elseif x>0
      y = 1;
   else
      y = 0;
   end
end
```

Funktionsaufruf

```
>> f = stueckfkt(2)

f =

    1
```

Inhalt

Informatik

Schleifen sind ein **Strukturelement**, um wiederkehrenden Programmcode abzukürzen und Anweisungen wiederholt auszuführen.

Schleifen: for-Schleife

MATLAB

for-Schleife führt eine bestimmte Folge von Anweisungen *n*-mal hintereinander aus.

Syntax

```
for <Variable>=<Matrix>
   <Anweisung 1>        % Anweisungsblock
   <Anweisung 2>
   ...
end
```

In der **for-Schleife** werden der Variable nacheinander die Spalten der Matrix zugewiesen und die Anweisungen ausgeführt.

Bemerkung

for-Schleifen *finden meist Verwendung bei Problemen, wo bereits zuvor bekannt ist, wie oft die Schleife durchlaufen werden soll.*

Beispiel:

Die folgende Summe soll mit Hilfe einer for-Schleife programmiert werden:

$$summe = \sum_{i=0}^{5} i = 0 + 1 + 2 + \cdots + 5$$

Schleifen: for-Schleife

Beispiel:

```
% Das Programm berechnet die Summe der ersten 5 natürlichen Zahlen.

summe=0;          % Der Variablen "summe" wird der Wert 0 zugewiesen.
                  % Dies ist nötig, damit mit der Variablen "summe"
                  % im Schleifenkörper gerechnet werden kann.

for i=1:5         % Schleifenkopf &
                  % Inkrement: Gibt an wie oft Schleife durchlaufen wird.
                  % Hier wird in Einerschritten von 1 bis 5 gegangen.

summe=summe+i; % Berechnet die Summe der ersten 5 natürlichen Zahlen

end               % Schleifenende
```

Der Schleifendurchlauf kann mit **break** vorzeitig beendet werden.

Beispiel:

Anschaulich sieht der Durchlauf der for-Schleife wie folgt aus:

Startwert: `summe = 0`

Beginn: `for – Schleife`

i	summe	Berechnung
1	0	summe = 0 + 1
2	1	summe = 1 + 2
3	3	summe = 3 + 3
4	6	summe = 6 + 4
5	10	summe = 10 + 5

Schleifenende: `summe = 15`

Schleifen: while-Schleife

Syntax

```
while <Eintrittsbedingung>
    <Anweisung 1>       % Anweisungsblock
    <Anweisung 2>
    ...
end
```

- **while-Schleife** ist an Eintrittsbedingung gekoppelt.
- Die **Eintrittsbedingung** kann eine Verknüpfung komplizierter Abfragen sein, so z.B. `mod(n,2)==0 && n<100`
- Die **while-Schleife** prüft vor jedem Eintritt in den Schleifenkörper, ob die Eintrittsbedingung wahr ist:
  - **wahr:** Schleifenkörper wird durchlaufen
  - **falsch:** while-Schleife wird verlassen (oder gar nicht erst betreten), Anweisungen nach while-Schleife werden ausgeführt
- Schleifendurchlauf kann vorzeitig mit **break** beendet werden.
  - Meist sinnvoll bei Anwendungen, die mit Suchen oder Zählen zu tun haben!

Schleifen: while-Schleife

Beispiel:

Eine Zahl soll solange verdoppelt werden, bis eine Obergrenze kleiner 50 erreicht wird. Dabei wird mit der Zahl 1 begonnen.

```
% Das Programm verdoppelt eine Zahl (beginnend bei 1), bis eine Obergrenze
% kleiner 50 erreicht ist.
i=1;        % Der Variablen i wird der Startwert 1 zugewiesen,
            % damit mit der Variablen im Schleifenkörper gerechnet
            % werden kann (Verdopplung soll bei 1 beginnen!)
while i<50  % Schleifenkopf & logischer Ausdruck:
            % solange i<50 ist, soll die Schleife die Berechnung ausführen
  i = i*2   % Die Variable i wird bei jedem Schleifendurchlauf mit 2
            % multipliziert.
            % Dadurch ändert sich mit jedem Schleifendurchlauf der Wert von i.
end         % Schleifenende
```

Schleifen: While-Schleife

Anschaulich sieht der Durchlauf der while-Schleife wie folgt aus:

Startwert: $\mathtt{i} = 1$

Beginn: $\mathtt{while-Schleife}$

i (neu)	Berechnung
1	$\mathtt{i}_{neu} = 1 \cdot 2$
2	$\mathtt{i}_{neu} = 2 \cdot 2$
4	$\mathtt{i}_{neu} = 4 \cdot 2$
8	$\mathtt{i}_{neu} = 8 \cdot 2$
16	$\mathtt{i}_{neu} = 16 \cdot 2$
32	$\mathtt{i}_{neu} = 32 \cdot 2$
64	Abbruchbedingung

Schleifenaustritt: $\mathtt{i} = 64$

Da $\mathtt{i} = 64$ über der vorgegebenen Schranke liegt, wird die while-Schleife an dieser Stelle beendet!

Schleifen: while-Schleife

Negativbeispiel:

Es wird nun absichtlich eine *Endlosschleife* erstellt. Eine Zahl soll dabei für positive Werte immer um Eins addiert werden. Es wird mit der Zahl 1 begonnen.

```
% Das Programm erstellt eine Endlosschleife
n=1;          % Der Variablen n wird der Startwert 1 zugewiesen,
              % damit mit ihr im Schleifenkörper gerechnet werden kann
while n>0     % Schleifenkopf & logischer Ausdruck:
              % solange n>0 ist, soll die Schleife die Berechnungen ausführen.
   n = n+1;   % Die Variable n wird bei jedem Schleifendurchlauf
              % mit 1 addiert. Dadurch wird der Zustand, dass n>0 ist erhalten
              % und es entsteht eine Endlosschleife
end           % Schleifenende
```

Bemerkung

Mit der Tastenkombination `Strg` + C (englisch: CTRL + C) kann man die aktuelle Berechnung einer Schleife abbrechen!

Inhalt

Mathematik

- Eine rekursive Funktion ist eine Funktion, die sich selbst wieder aufruft.
- Eine rekursive Funktion hat
  - einen Rekursionsanfang und
  - einen Rekursionsschritt.

Beispiel:

Rekursive Funktionsvorschrift für $n \in \mathbb{N}$:

$$f(n) = \begin{cases} 1 & \text{, falls } n = 0 \\ n \cdot f(n-1) & \text{, sonst} \end{cases}$$

Freie Universität Berlin

Informatik

- Eine rekursive Funktion ist eine Funktion, die sich selbst wieder aufruft.
- Eine rekursive Funktion hat
  - einen Rekursionsanfang und
  - einen Rekursionsschritt.
- Eine Rekursion kann mit Hilfe der Kontrollstrukturen (if-Anweisung oder Schleifen) gelöst werden.

Beispiel:

Die Fakultät soll rekursiv mit Hilfe einer Funktion berechnet werden.
(Beispiel Fakultät: $5! = 1 \cdot 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$)

Die rekursive Funktionsvorschrift für die Fakultät kann auf zwei Arten dargestellt werden:

1. Variante

$$f(n) = \begin{cases} 1 & \text{, falls } n = 0 \\ n \cdot f(n-1) & \text{, sonst} \end{cases}$$

2. Variante

$$f(n) = \begin{cases} 1 & \text{, falls } n = 0 \\ \prod_{i=1}^{n} i & \text{, sonst} \end{cases}$$

$$\begin{aligned} \prod_{i=1}^{n} i &= 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \\ &= 2 \cdot 3 \cdot \ldots \cdot n \\ &= 6 \cdot \ldots \cdot n \\ &= \prod_{i=1}^{n-1} i \cdot n \end{aligned}$$

Beispiel: 1. Variante

```
% Die Funktion berechnet die Fakultät rekursiv.
function y = fak(n)      % Beginn Funktion
                         % Funktionsname: fak
                         % Eingabeparameter: n
                         % Ausgabeparameter: y
    if (n<=1)            % Beginn if-else-Anweisung:
      y=1;               % Für n<=1 soll y=1 sein.
    else                 % Wenn n>1 ist, dann soll die Fakultät
      y = n*(fak(n-1));  % berechnet werden, d.h. 1*2*3*4*...
    end                  % Ende if-Anweisung
end                      % Ende der Funktion
```

Funktionsaufruf

```
>> f = fak(4)
f =
   24
```

Beispiel: 2. Variante

```
% Die Funktion berechnet die Fakultät iterativ rekursiv.
function [produkt] = pfak(k)
produkt=1;          % Der Variablen "produkt" wird der Wert 1 zugewiesen.
                    % Dies ist nötig, damit mit der Variablen "produkt"
                    % im Schleifenkörper gerechnet werden kann.

for i=1:k           % Schleifenkopf &
                    % Inkrement: Gibt an wie oft Schleife durchlaufen wird.
                    % Hier wird in Einerschritten von 1 bis k>0 gegangen.

produkt=produkt*i; % Berechnet das Produkt der ersten k natürlichen Zahlen

end                 % Schleifenende
end
```

Funktionsaufruf

```
>> p = pfak(4)
p =
   24
```

Inhalt

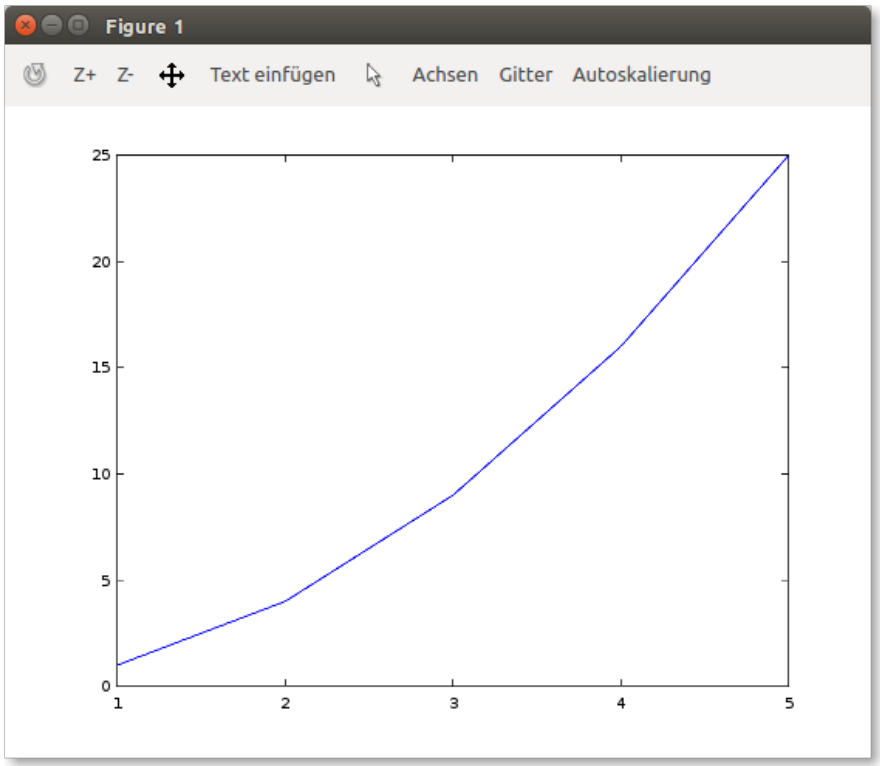
Die Ausgabefunktion (plot)

- Anwendung zum Darstellen von Funktionsgraphen oder anderen 2D-Daten.
- Syntax: `plot(x, y);`
- x ist dabei ein Vektor mit *x*-Koordinaten, *y* ist der Datenvektor.

Beispiel:

```
x = 1:5;
y = x.^2;
plot(x,y)
```

Und so sieht es aus:



Farben, Linien, Marker

Die `plot`-Funktion kann um einige Einstellungen erweitert werden. So kann z.B. die Farbe, die Linienart und die Markerform mit folgenden Kürzeln geändert werden:

Farbe	Kürzel	Linie	Kürzel	Marker	Kürzel
Blau (Standard)	b	durchgezogen (Standard)	–	Punkt (Standard)	.
Rot	r	gepunktet	:	Kreis	o
Grün	g	strich-gepunktet	–.	Kreuz	×
Geld	y	gestrichelt	–	Plus	+
Schwarz	k			Stern	*
Weiß	w			Quadrat	s
Cyan	c			Raute	d
Magenta	m			Dreieck	v,<,>,^
				Fünfeck	p
				Sechseck	h

```
% gepunktete rote Linie aus Quadraten

  plot(x,y,'r:s')
```

Einstellungen für das Plotfenster

- **`hold on/hold off`:** Um mehrere Plots in dasselbe Fenster zu zeichnen wird die Funktion `hold on` verwendet. Mit `hold off` wird die Option wieder ausgeschaltet.

- **`subplot`:** Um mehrere kleine Plotfenster nebeneinander zu plotten wird der Befehl `subplot` verwendet.
  `subplot` benötigt drei Argumente: Die Anzahl der Zeilen und der Spalten in die das Plotfenster aufgeteilt werden soll, sowie die Nummer des zu aktivierenden Feldes.

  `subplot(m, n, k)`

- **axis:** Zum festlegen der x- und y-Achseneinteilung des aktiven Plots.
  - kommt direkt nach dem `plot`—Befehl.
  - Syntaxbeispiel: `axis([ 0 5 −1 10 ]);`
  - Die Zahlen bedeuten in dieser Reihenfolge die Angabe der unteren und oberen Grenze der *x*-Achse und die untere und obere Grenze der *y*-Achse.

- **title:** Ordnet einem Plot einen Titel zu.
  - Syntaxbeispiel: `title('Dies ist der Titel!');`

- **xlabel/ylabel:** Beschriftung der x- und y-Achsen.
  - Syntaxbeispiel:
```
xlabel('Dies ist die x-Achse!')
ylabel('Dies ist die y-Achse!')
```

Plotten - Das Plotfenster

- **legend:** Legende zum Plot (hilfreich, wenn mehrere Plots in einem Fenster sind!).
  - Syntaxbeispiel:
    `legend('Name erster Graph','Name zweiter Graph','...');`
  - Die Namen der Graphen werden in der gleichen Reihenfolge zugeordnet, in der sie geplottet werden.

- **text:** Einfügen von Text im Plot.
  - Syntaxbeispiel:
    `text(2.7,3.5,'Hier Text!','Color','r','FontSize',15);`
  - Die ersten beiden Zahlen bezeichnen die Koordinaten des ersten Buchstabens, dann folgt der Text und weitere Optionen wie die Schriftfarbe (hier: rot) und die Schriftgröße (hier: 15).

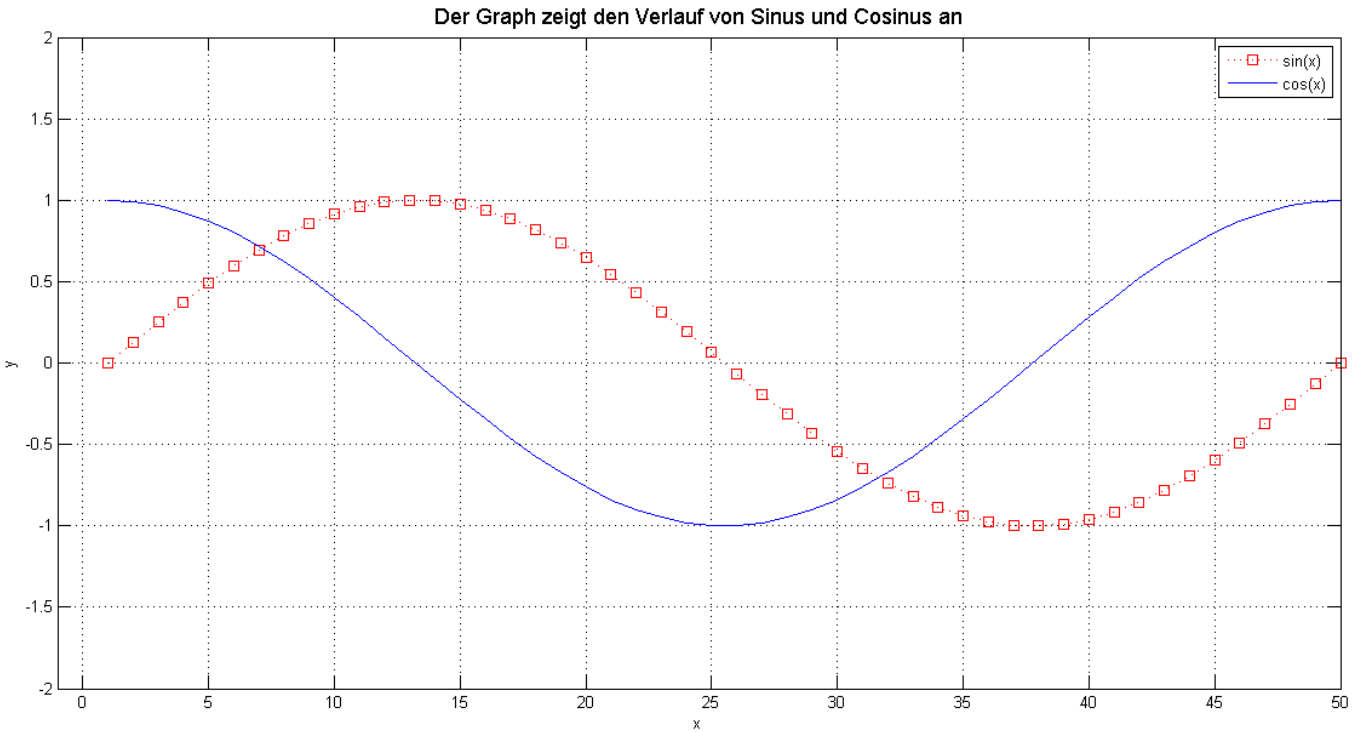
- **grid on/grid off: grid on** erstellt ein Gitter im Plotfenster, **grid off** blendet es wieder aus

Beispiel: plot

```
clear figure

x=linspace(0,2*pi,50);
y=linspace(0,2*pi,50);

plot(sin(x),'r:s');
hold on
plot(cos(x));
axis([-1 50 -2 2]);
grid on
legend('sin(x)','cos(x)');
title('Der Graph zeigt den Verlauf von Sinus und Cosinus an','FontSize',13);
xlabel('x');
ylabel('y');
```

Freie Universität Berlin

Plot

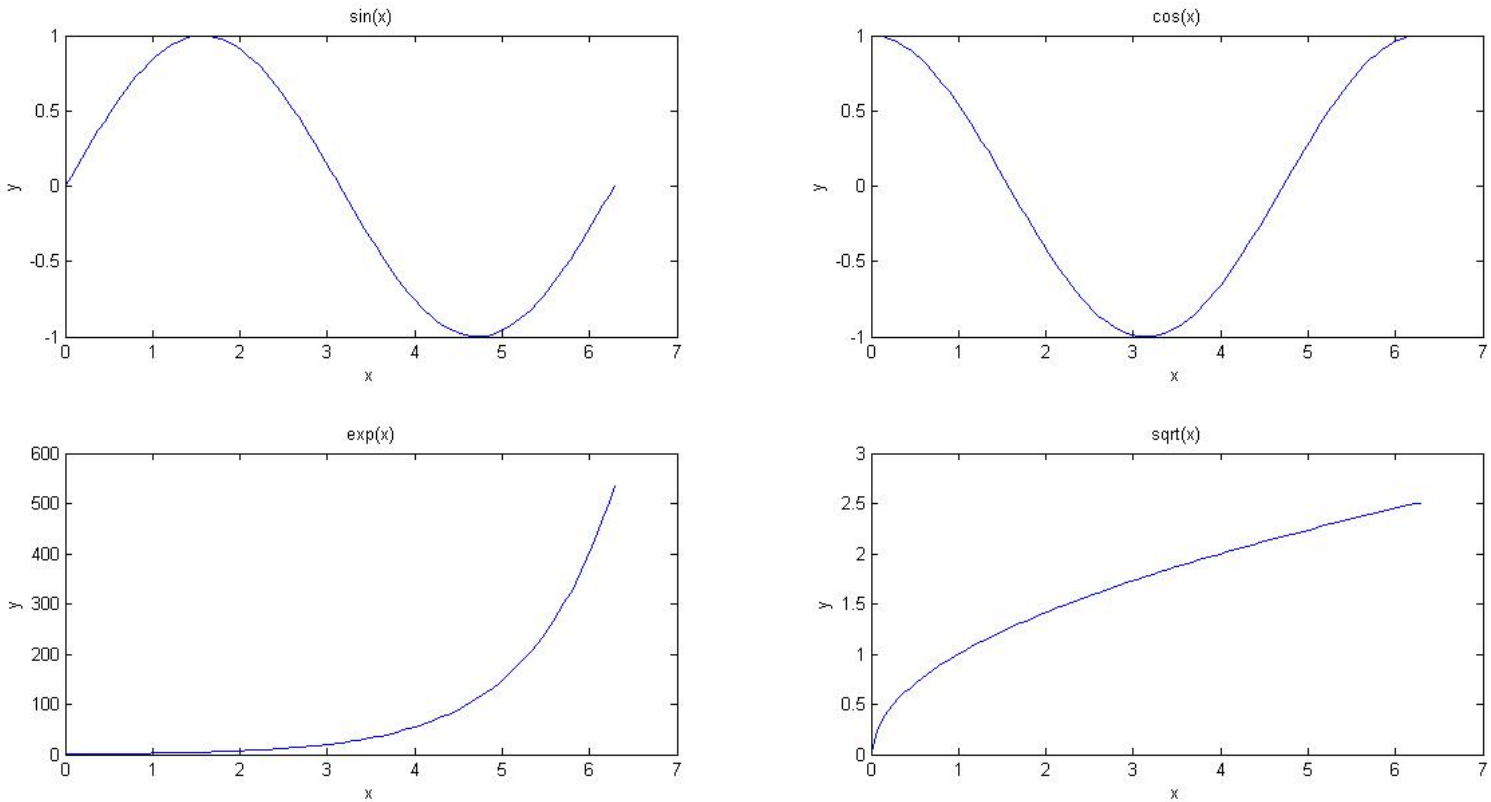


Beispiel: subplot

```
x=linspace(0,2*pi,100);

subplot(2,2,1);
plot(x,sin(x));
xlabel('x');
ylabel('y');
title('sin(x)');

subplot(2,2,2);
plot(x,cos(x));
xlabel('x');
ylabel('y');
title('cos(x)');
```

```
subplot(2,2,3);
plot(x,exp(x));
xlabel('x');
ylabel('y');
title('exp(x)');

subplot(2,2,4);
plot(x,sqrt(x));
xlabel('x');
ylabel('y');
title('sqrt(x)');
```

Subplot



Inhalt

Allgemeine Hinweise

- Neben MATLAB, kann auch das kostenfreie Octave zum Erstellen von MATLAB-Programmen bzw. -Dateien verwendet werden:
  https://www.gnu.org/software/octave/#install

  Achtung: In Octave gibt es einige integrierte Funktionen, die es in MATLAB nicht gibt (bspw. `idivide` in Octave heißt in MATLAB `floor`). Daher sollten in Octave angefertigte Programme noch einmal unter MATLAB vor etwaiger Abgabe getestet werden.

- Für weitere Informationen und Aufgaben rund um MATLAB ist das Skript „MATLAB Mini Tutorials“ von Tobias Kies zu empfehlen (Stand 22.12.2017): http://numerik.mi.fu-berlin.de/wiki/WS_2017/CoMaI_Dokumente/MATLAB_MiniTutorials.pdf

- Zum Nachschlagen einiger Befehle und bereits in MATLAB vorhandener Funktionen empfiehlt sich die Online-Dokumentation von MATLAB unter: https://mathworks.com/help/matlab/index.html

Referenzen

- Menzel, Christoph: Einführung in Matlab. Zuletzt bearbeitet: Jianis Baumgardt (Juni 2013), Projektgruppe Praktische Mathematik (TU Berlin)
- https://www.cs.uni-potsdam.de/ml/teaching/ws14/ida/Einfuehrung_in_MATLAB.pdf
- http://numerik.mi.fu-berlin.de/wiki/WS_2017/CoMaI_Dokumente/MATLAB_MiniTutorials.pdf
- https://www.uni-ulm.de/fileadmin/website_uni_ulm/mawi.inst.070/ws12_13/Numerik1/Uebung1.pdf
- https://www.uni-ulm.de/fileadmin/website_uni_ulm/mawi.inst.070/ws12_13/Numerik1/Uebung1.pdf
- http://num.math.uni-goettingen.de/plonka/Numeriksig1/matlab.pdf
- http://www.math.uni-rostock.de/~peters/MATLAB/matlab_kurz.pdf
- http://wwwmath.uni-muenster.de/num/Vorlesungen/MATLAB-Kurs_WS08/Script/matlab-einfuehrung.pdf
- https://de.wikipedia.org/wiki/Funktion_(Mathematik)#Definition
- https://de.serlo.org/mathe/sonstiges/mengenlehre-und-logik/logik/notwendige-und-hinreichende-bedingungen
- https://de.wikipedia.org/wiki/Bedingte_Anweisung_und_Verzweigung