GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Nummerisches Programmieren

WS 21/22

11.10.2021 – 15.10.2021

Organisatorisches

- Corona Maßnahmen
- Ablauf des Blockseminars
- Lernziele

Organisatorisches

Corona Maßnahmen

- 3G-Nachweis
- Maskenpflicht
- Unterweisungsbogens

Organisatorisches

Ablauf des Blockseminars

Block 1: 09:00 – 10:30 – Vorlesung und Onlineübung

15min Pause

Block 2: 10:45 – 12:15 – Übung

1h Mittagspause

Block 3: 13:15 – 14:45 – Vorlesung und Onlineübung

15min Pause

Block 4: 14:00 – 16:30 – Übung

Organisatorisches

Lernziele

- Modulhandbuch: Die Studierenden lernen Algorithmen computergestützt zu implementieren und auf konkrete Probleme anzuwenden.

- Die Studierenden lernen MatLab kennen und lernen damit eigenständig zu arbeiten.

Inhaltsverzeichnis

Inhaltsverzeichnis

1: Einführung

Inhaltsverzeichnis

- Einleitung
- Zugang zu MatLab
- Erste Schritte
- Hilfe zu MatLab

1: Einführung

Lernziele:

- Sie wissen, was MatLab ist und was die Software kann.
- Sie wissen, wie Sie Hilfe zu MatLab bekommen.
- Sie können einfache Rechnungen mit MatLab durchführen.

1: Einführung

Inhaltsverzeichnis

1: Einführung

Einleitung

Was ist MatLab?

- MatLab ist ein Softwarepaket für numerische Berechnungen und für die Visualisierung von Daten im technisch-wissenschaftlichen Bereich.

- Eine Programmiersprache, in der die Lösungsalgorithmen implementiert werden.

1: Einführung

Einleitung

Was kann MatLab?

- MatLab liefert viele vorgefertigte Standardalgorithmen, die direkt verwendet werden können.
- Die Funktionalität von MatLab ist in Toolboxes aufgeteilt, bei der jede Toolbox vorgefertigte Algorithmen für bestimmte Aufgabenstellungen enthalten. Beispiele:

  Optimization Toolbox, Partial Differential Equations Toolbox, etc…

1: Einführung

Einleitung

Wie kann ich MatLab einsetzen?

- Bei der **interaktiven Verwendung** werden Anweisungen direkt über die Tastatur eingegeben und sofort ausgeführt.

- Für umfangreiche Probleme ist es empfehlenswert, MatLab als **Programmiersprache** einzusetzen.

1: Einführung

Inhaltsverzeichnis

- Einleitung
- Zugang zu MatLab
- Erste Schritte
- Hilfe zu MatLab
- Matrizen und Lineare Algebra

1: Einführung

Zugang zu MatLab

- MatLab Zugang und Installationsanleitung unter (Link auf Homepage):

  https://www.rz.uni-frankfurt.de/69806761/Mathworks_MATLAB

- Onlinezugang:

  https://matlab.mathworks.com/

1: Einführung

Zugang zu MatLab

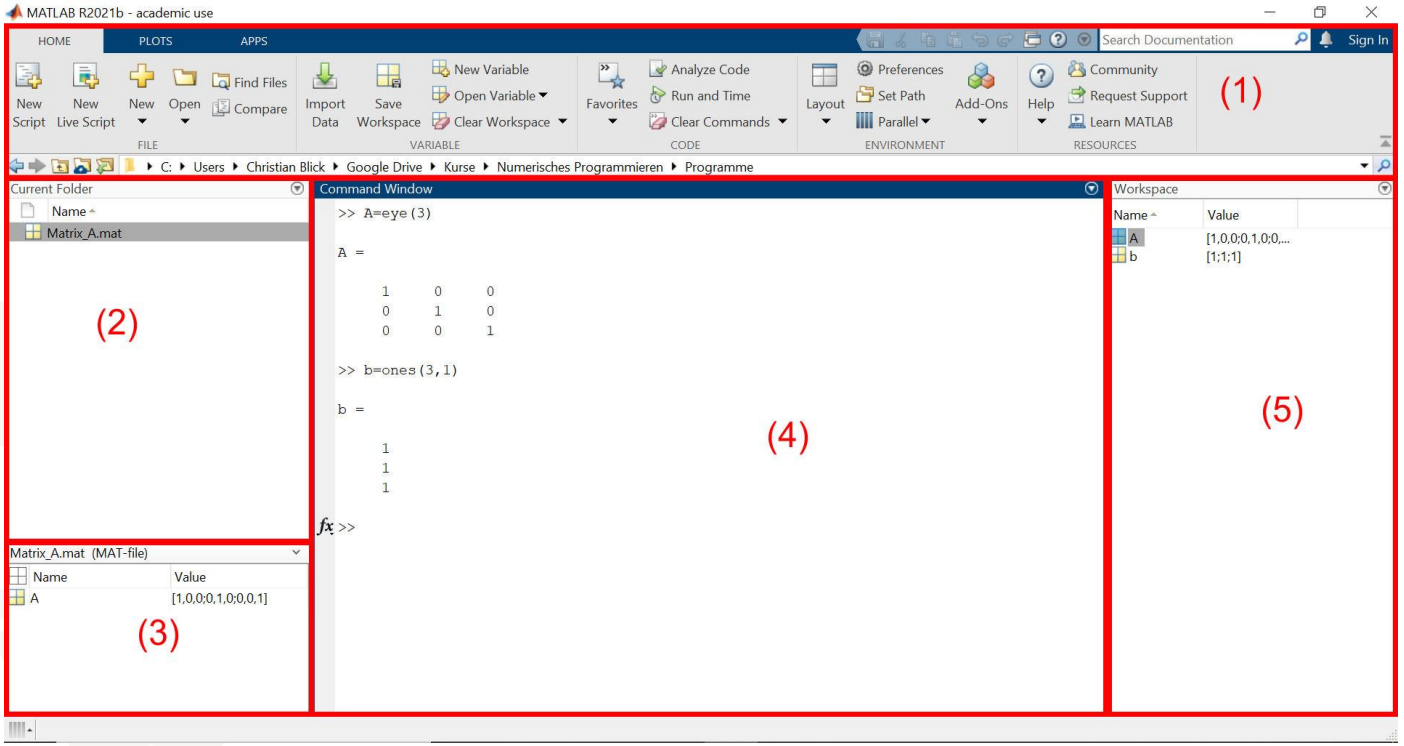
- In beiden Fällen ist eine Registrierung bei Mathworks mit ihrer offiziellen HRZ-Account E-Mailadresse.
- Sollte Ihr Mathworksaccount nicht automatisch mit der TAH Lizenz verknüpft werden, finden Sie den Lizenzcode nach einloggen unter

  https://kartenservice.uni-frankfurt.de/mitarbeitercard/login?

  unter dem Punkt „Software Tokens anzeigen“.

1: Einführung

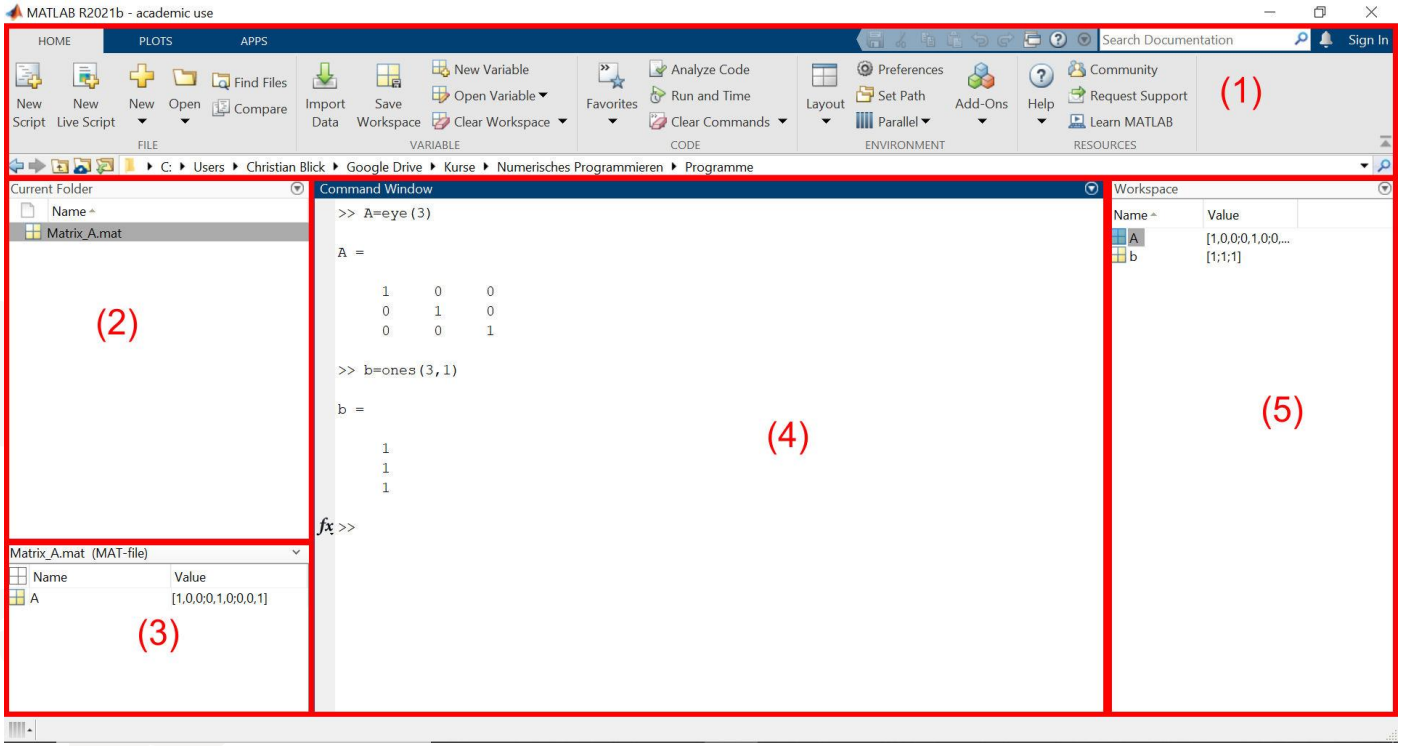
Aufbau von MatLab



(1) Enthält das Menü. Hier können Sie z. B. Einstellungen ändern und Hilfsfunktionen aufrufen (z. B. plotten).

1: Einführung

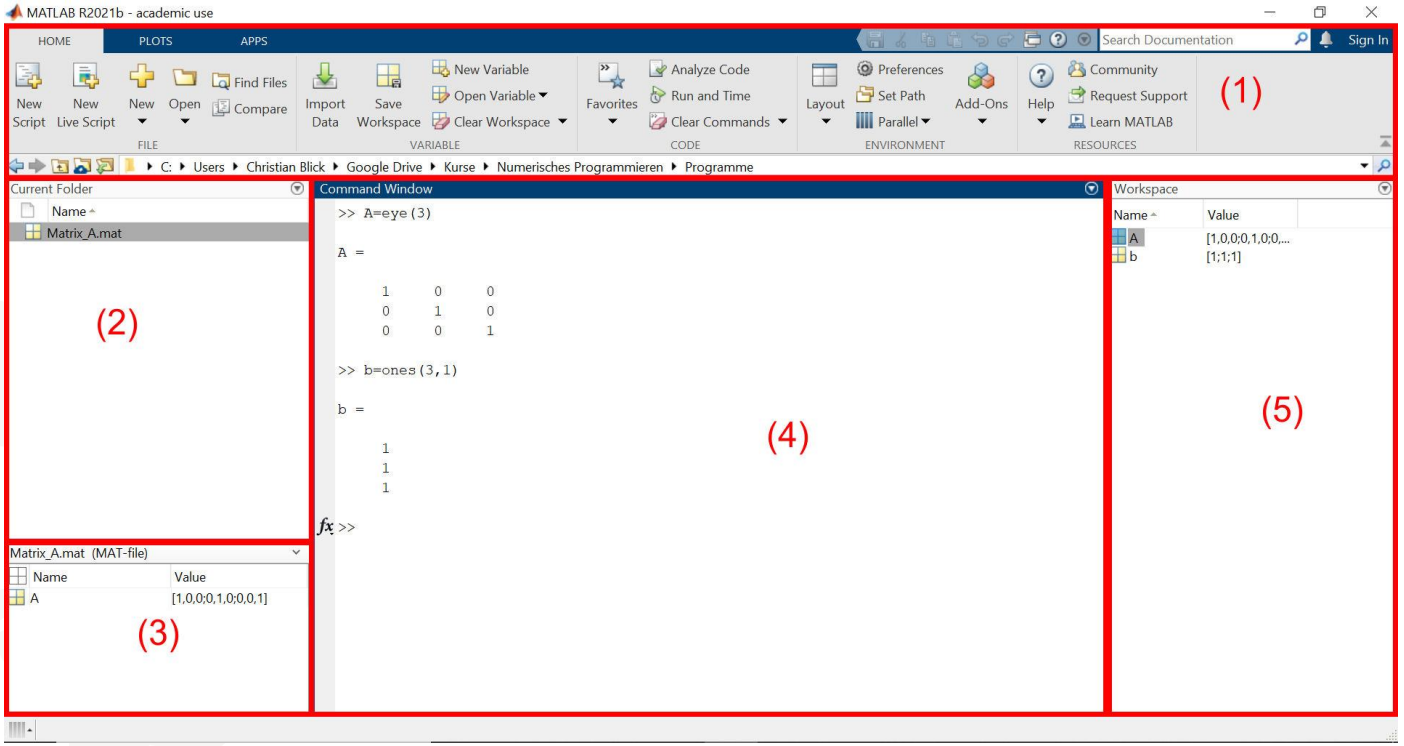
Aufbau von MatLab



(2) Zeigt die Dateien und Unterordner, die sich unter dem aktuellen Pfad befinden.

1: Einführung

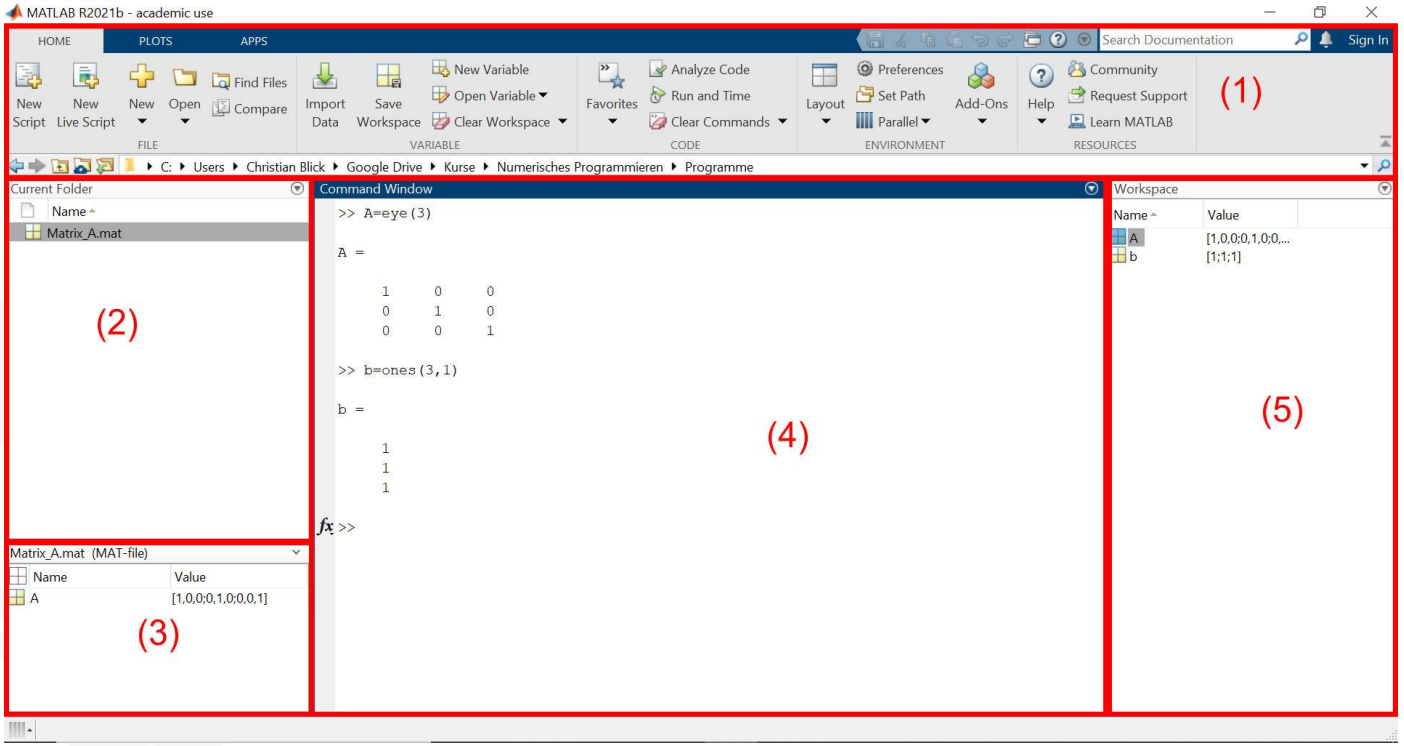
Aufbau von MatLab



(3) Zeigt eine Kurzbeschreibung der Datei, die gerade in (2) ausgewählt wurde.

1: Einführung

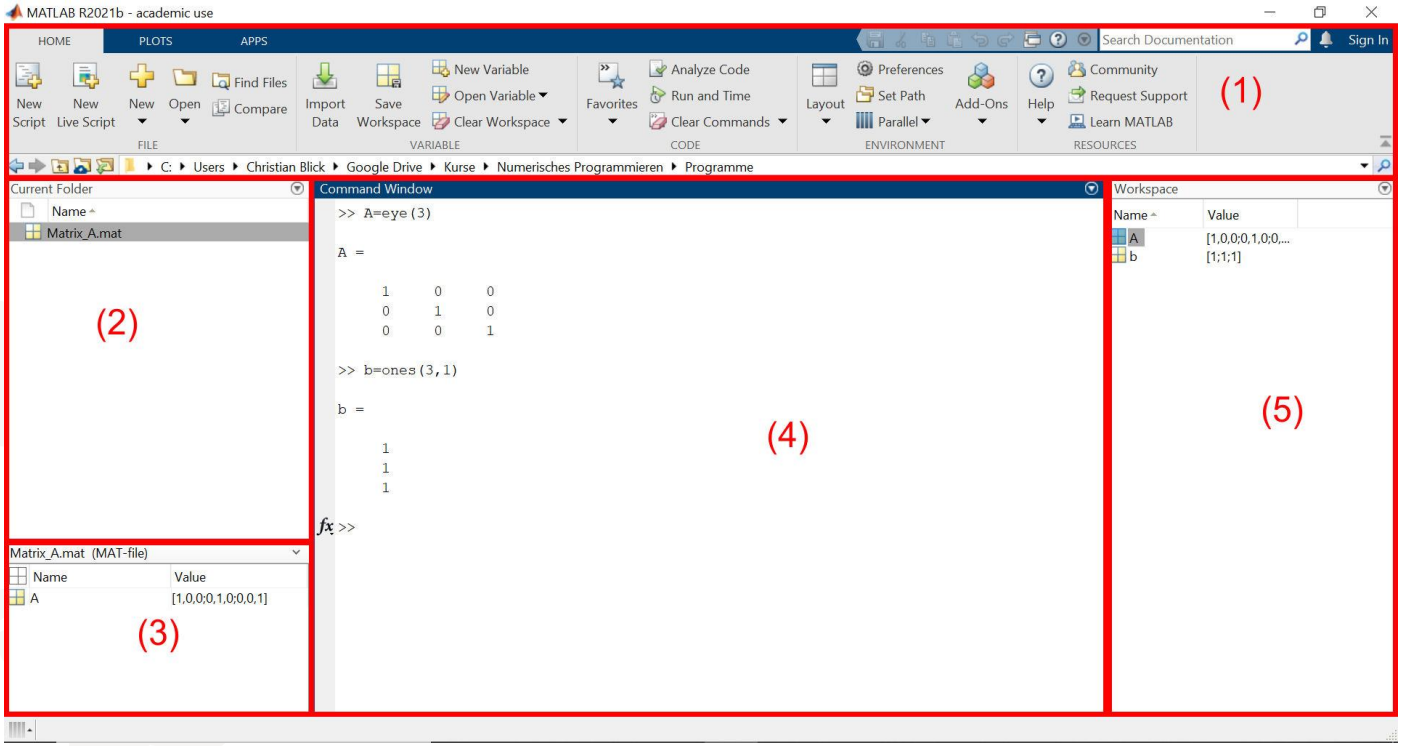
Aufbau von MatLab



(4) Zeigt das *Command Window*, in dem Befehle direkt ausgeführt werden können (Taschenrechner).

1: Einführung

Aufbau von MatLab



(5) Zeigt den *Workspace*, eine Liste mit zur Verfügung stehenden Variablen, Funktionen, Strukturen, …. .

1: Einführung

Inhaltsverzeichnis

- Einleitung
- Zugang zu MatLab
- Erste Schritte
- Hilfe zu MatLab
- Matrizen und Lineare Algebra

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

Befehle können direkt in das Command Window hinter dem Prompt-Zeichen >> eingegeben werden:

```
>> 2-4
ans =
    -2
```

`ans` ist eine Hilfsvariable mit der gerechnet werden kann.

```
>> ans+3
ans =
     1
```

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

```
>> a=5.6
a =
    5.6000
```

definiert eine Variable a mit Wert `5.6` (Dezimalpunkt statt Komma). Die Ausgabe `a=5.6000` kann durch ein Semikolon verhindert werden:

```
>> a=5.6;
```

Das Semikolon am Zeilenende unterdrückt nur die Ausgabe im Command Window, die Berechnung wird dennoch ausgeführt.

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

Variablen können wie folgt ausgegeben werden:

```
>> a
a =
    5.6000
```

`i` steht in MatLab für die Imaginäre Einheit.

```
>> a+2i
ans =
   5.6000 + 2.0000i
```

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

Das Multiplikationszeichen kann bei der Multiplikation mit `i` weggelassen werden.

```
>> a+2*i

ans =

   5.6000 + 2.0000i
```

Vorsicht! Mit

```
>> i=5;
```

wird `i` ein neuer Wert zugewiesen. Dies kann zu Fehlern im Programm führen.

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

```
>> a+2*i
ans =
    15.6000
```

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Namen von Variablen und Funktionen beginnen mit einem Buchstaben gefolgt von einer beliebigen Anzahl von Buchstaben, Zahlen oder Unterstrichen.
- MatLab unterscheidet zwischen Groß- und Kleinbuchstaben. Parallel zur Variablen `a` kann also eine Variable `A` definiert und genutzt werden.
- Das Minus-Zeichen − darf in Variablen oder Funktionsnamen nicht auftauchen.

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Welche der folgenden Variablennamen sind in MatLab zulässig?

(a) 2nach1

(b) die erste

(c) dieZweite

(d) die_dritte

(e) variable-nummer-1

(f) variable_nummer.2

(g) zins_in_%

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Welche der folgenden Variablennamen sind in MatLab zulässig?

(a) 2nach1

(b) die erste

(c) dieZweite

(d) die_dritte

(e) variable-nummer-1

(f) variable_nummer.2

(g) zins_in_%

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Welche der folgenden Variablennamen sind in MatLab zulässig?

(a) 2nach1

(b) die erste

(c) dieZweite

(d) die_dritte

(e) variable-nummer-1

(f) variable_nummer.2

(g) zins_in_%

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Welche der folgenden Variablennamen sind in MatLab zulässig?

(a) 2nach1

(b) die erste

(c) dieZweite

(d) die_dritte

(e) variable-nummer-1

(f) variable_nummer.2

(g) zins_in_%

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Welche der folgenden Variablennamen sind in MatLab zulässig?

(a) 2nach1

(b) die erste

(c) dieZweite

(d) die_dritte

(e) variable-nummer-1

(f) variable_nummer.2

(g) zins_in_%

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Welche der folgenden Variablennamen sind in MatLab zulässig?

(a) 2nach1

(b) die erste

(c) dieZweite

(d) die_dritte

(e) variable-nummer-1

(f) variable_nummer.2

(g) zins_in_%

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Welche der folgenden Variablennamen sind in MatLab zulässig?

(a) 2nach1

(b) die erste

(c) dieZweite

(d) die_dritte

(e) variable-nummer-1

(f) variable_nummer.2

(g) zins_in_%

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

- Welche der folgenden Variablennamen sind in MatLab zulässig?

(a) 2nach1

(b) die erste

(c) dieZweite

(d) die_dritte

(e) variable-nummer-1

(f) variable_nummer.2

(g) zins_in_%

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

MatLab rechnet nach dem IEEE Standard für Fließkommazahlen:

```
>> b = 1/1111101 * 1/3

b =

   3.0000e-07
```

Dabei steht

3.0000e-07 für den Wert $3.0000*10^{-07}$.

1: Einführung

Erste Schritte: MatLab als „Taschenrechner“

Achtung!

Intern rechnet MatLab standardmäßig auf 15 Stellen hinter dem Komma genau, stellt aber standardmäßig nur 4 dar. Dies kann mit dem Befehl

```
>> format long
```

geändert werden:

```
>> b = 1/1111101 * 1/3
b =
    3.000027300248433e-07
```

1: Einführung

Inhaltsverzeichnis

- Einleitung
- Zugang zu MatLab
- Erste Schritte
- Hilfe zu MatLab
- Matrizen und Lineare Algebra

1: Einführung

Hilfe zu MatLab

- MatLab verfügt über ein äußerst umfangreiches Hilfesystem.

- Sämtliche MatLab-Befehle, Operatoren und Programmierstrukturen wie Schleifen, Fallunterscheidungen etc. sind ausführlich dokumentiert.

Beispiel: Wie stelle ich das Darstellungsformat zurück?

1: Einführung

Hilfe zu MatLab

Methode 1: Benutzen Sie den Befehl `help Befehlsname`.

Dies hilft bei Unsicherheiten bei der Syntax oder Funktionsweise von bekannten Funktionen.

```
>> help format

format Set output format.

format with no inputs sets the output
format to the default appropriate for
the class of the variable. For float
variables, the default is format
SHORT. ...
```

1: Einführung

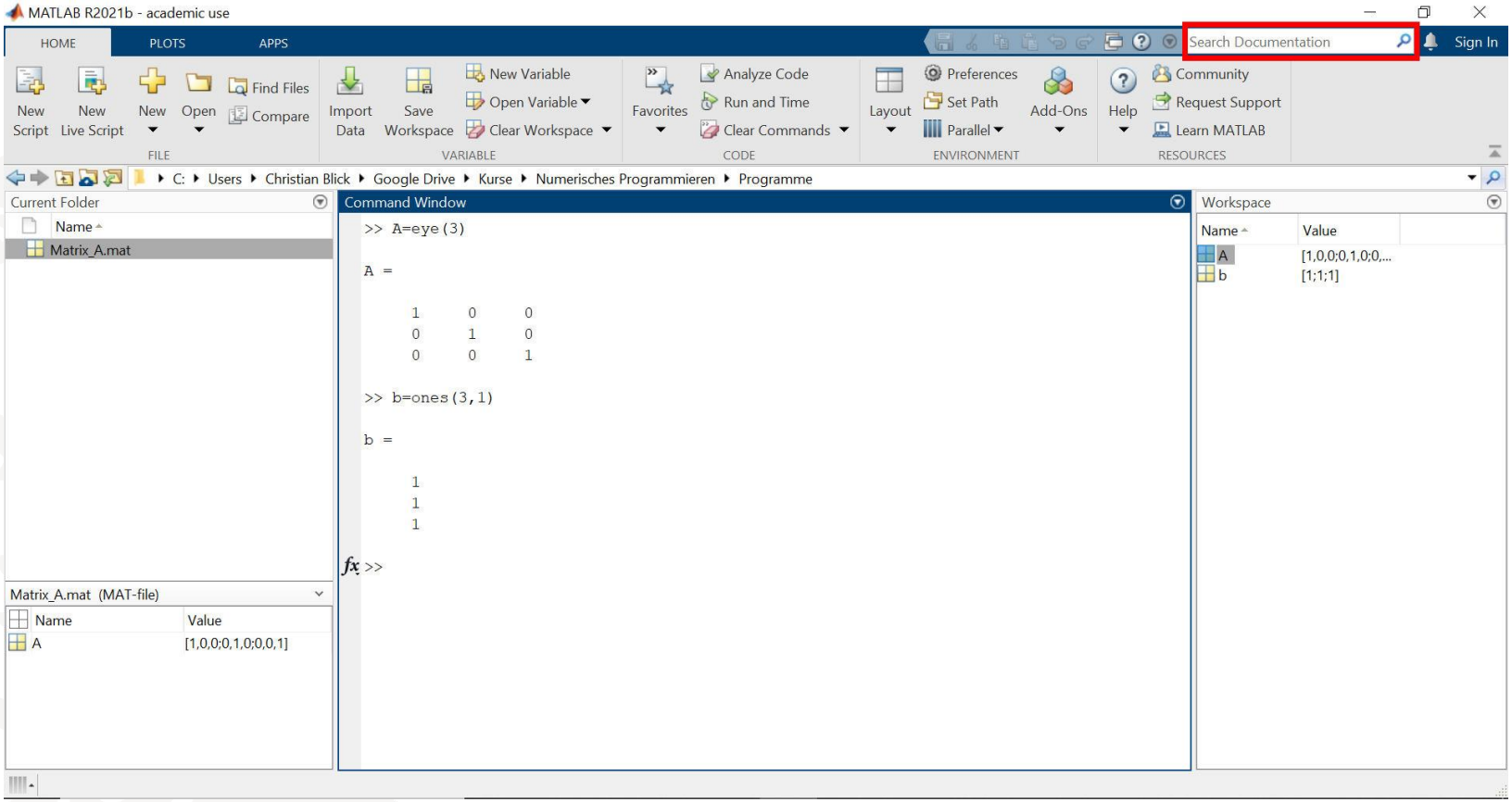
Hilfe zu MatLab

```
>> format short
>> b = 1/1111101 * 1/3
b =
   3.0000e-07
```

1: Einführung

Hilfe zu MatLab

Methode 2: Benutzen Sie die Suchmaske der Hilfe.

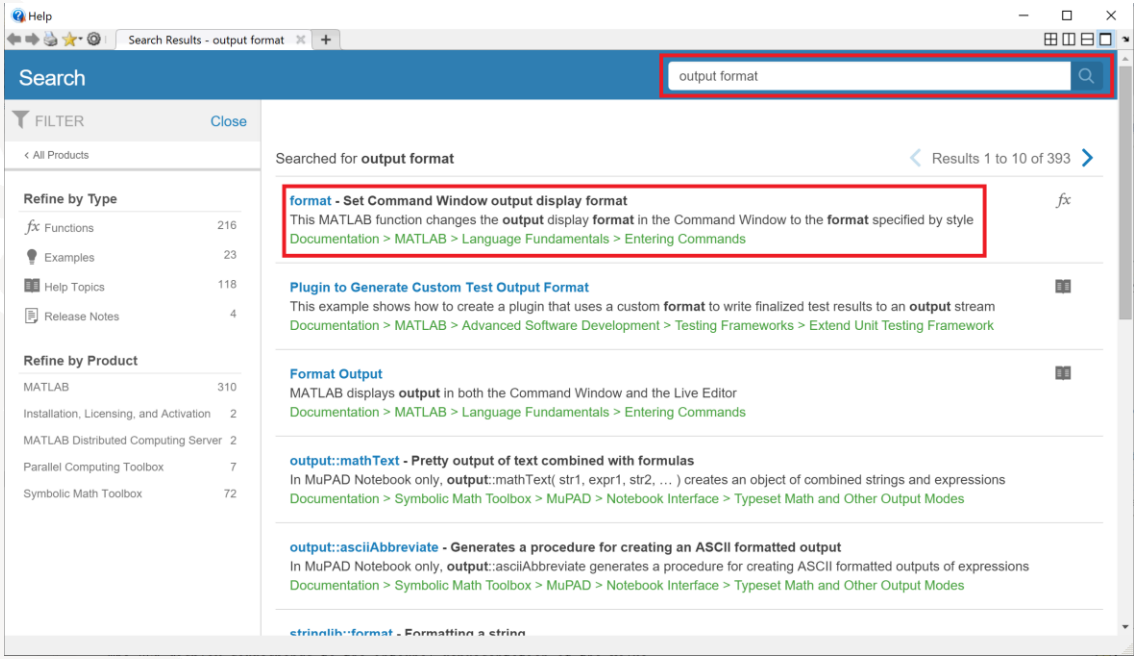


1: Einführung

Hilfe zu MatLab

Methode 2: Benutzen Sie die Suchmaske der Hilfe.

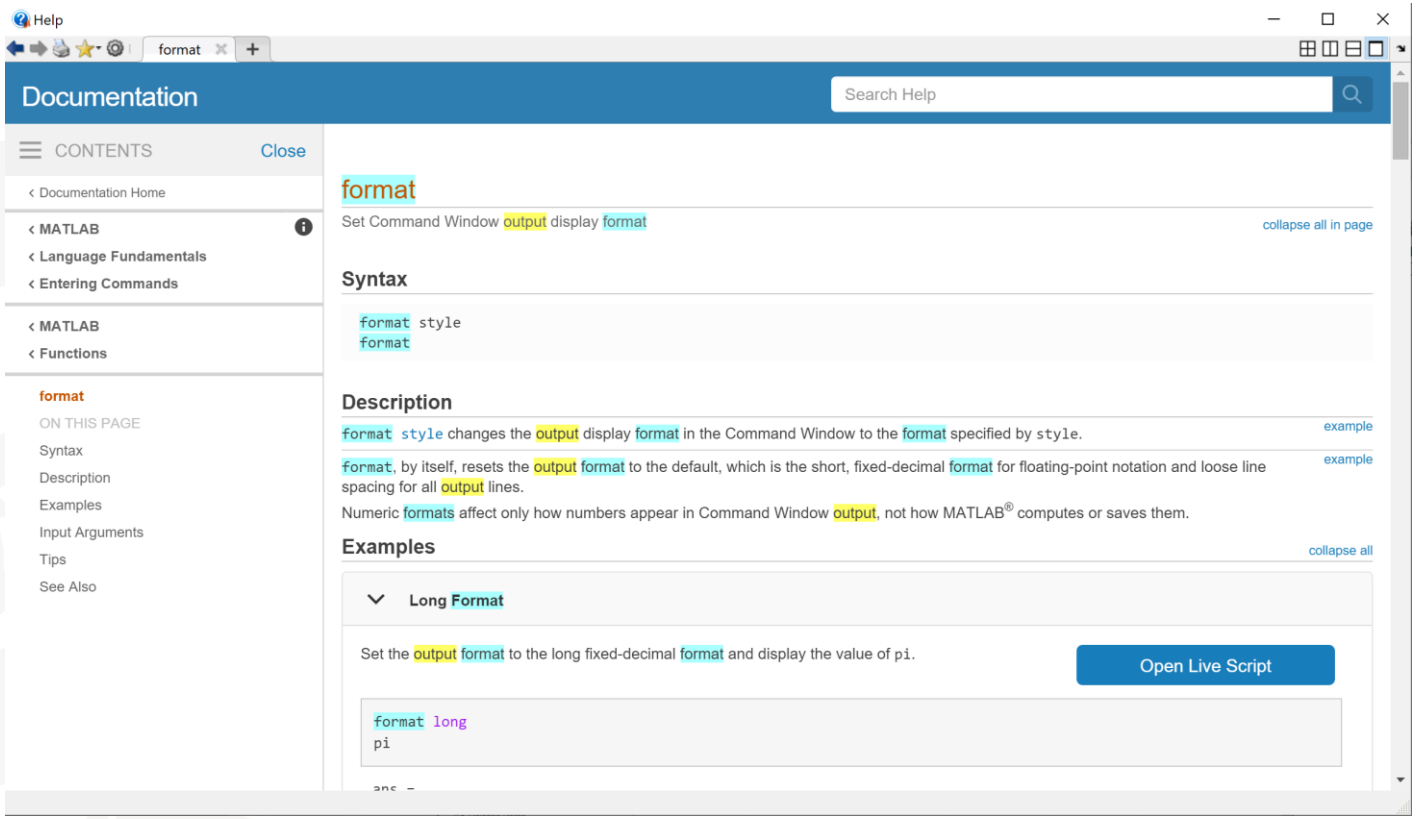
Dies hilft, wenn die Funktion nicht bekannt ist.



1: Einführung

Hilfe zu MatLab

Methode 2: Benutzen Sie die Suchmaske der Hilfe.



1: Einführung

Hilfe zu MatLab

Methode 3: Google Suche

Dies hilft,

- wenn die MatLab Suche nicht weiter hilft.
- wenn die Lösung so umfangreich ist, dass vollständige Skripte und Funktionen benötigt werden.

Beispiel: Die Mitte einer Colorbar eines Plots soll weiß sein. Google Suche: *matlab center of colorbar white*

Inhaltsverzeichnis

2: Lineare Algebra

Inhaltsverzeichnis

- Das Skript
- Erstellen von Matrizen
- Rechnen mit Skalaren, Vektoren und Matrizen
- Manipulation von Matrizen
- Spezielle Matrizen erstellen

2: Lineare Algebra

Einleitung

Lernziele:

- Sie können mit Skalaren, Vektoren und Matrizen rechnen.
- Sie können erste Skripte in MatLab schreiben.

2: Lineare Algebra

Inhaltsverzeichnis

- Das Skript

- Erstellen von Matrizen

- Rechnen mit Skalaren, Vektoren und Matrizen

- Manipulation von Matrizen

- Spezielle Matrizen erstellen

2: Lineare Algebra

Das Skript

Befehle hintereinander in das Command Window einzugeben kann unübersichtlich werden.

In MatLab können sogenannte Skripte (M-Files) erstellt werden.

In diesen können Befehle gespeichert werden, die nacheinander ausgeführt werden.

Ein neues Skript kann über *File* → *New Script* geöffnet werden.

2: Lineare Algebra

Das Skript

In das Skript werden die Befehle genau wie in das Command Window eingegeben. Anschließend wird die Datei als Dateiname.m gespeichert.

Die Befehle lassen sich über das Command Window abrufen, indem Sie den Dateinamen ohne die Dateiendung .m eingeben.

Alternativ können Sie das Skript unter Editor → ausführen.

2: Lineare Algebra

Das Skript

Das Beispiel `K2_Beispiel_Skript.m` addiert zwei Zahlen `a` und `b` und gibt die Summe im Fließtext aus.

Mit `%` werden Kommentare im Skript definiert. Alles, was in einer Zeile nach dem `%`-Zeichen im Skript steht, wird bei der Ausführung des Skripts ignoriert.

Mit `disp` wird ein vorgegebener Text oder eine Variable ausgegeben. Der Text muss dabei in `' '` stehen.

2: Lineare Algebra

Das Skript

Im Befehl `disp` ist es auch möglich, Text und Variablen zu verknüpfen.

Dazu wird die auszugebende Variable mit dem Befehl `num2str` von einer Variablen in eine Zeichenkette umgewandelt.

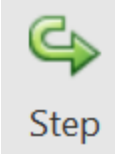
Der vorgegebene Fließtext wird mit `[ ]` mit der umgewandelten Variable verknüpft.

```
disp(['a+b ist gleich ' num2str(a+b)])
```

2: Lineare Algebra

Das Skript

Beim Programmieren schleichen sich immer Fehler ein. Um diese zu finden, liefert MatLab folgende Möglichkeiten:

- Gehen Sie den Code mit 

zeilenweise durch und beobachten Sie die Variablen im Workspace.

- Über den Button “Step In“ 

können Sie in Unterroutinen springen und diese untersuchen. Mit “Step Out“ 

verlassen Sie die Unterroutine wieder.

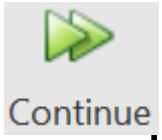
2: Lineare Algebra

Das Skript

- Klicken Sie auf die Zeilenzahl um einen Break-Point 

zu setzen. Führen Sie nun den Code mit „Run“ 

aus, stoppt der Code am gesetzten Break-Point.

- Mit dem Button “Continue“ 

läuft der Code bis zum nächsten Break-Point oder, wenn kein weiterer Break-Point existiert, bis zum Ende des Programms weiter.

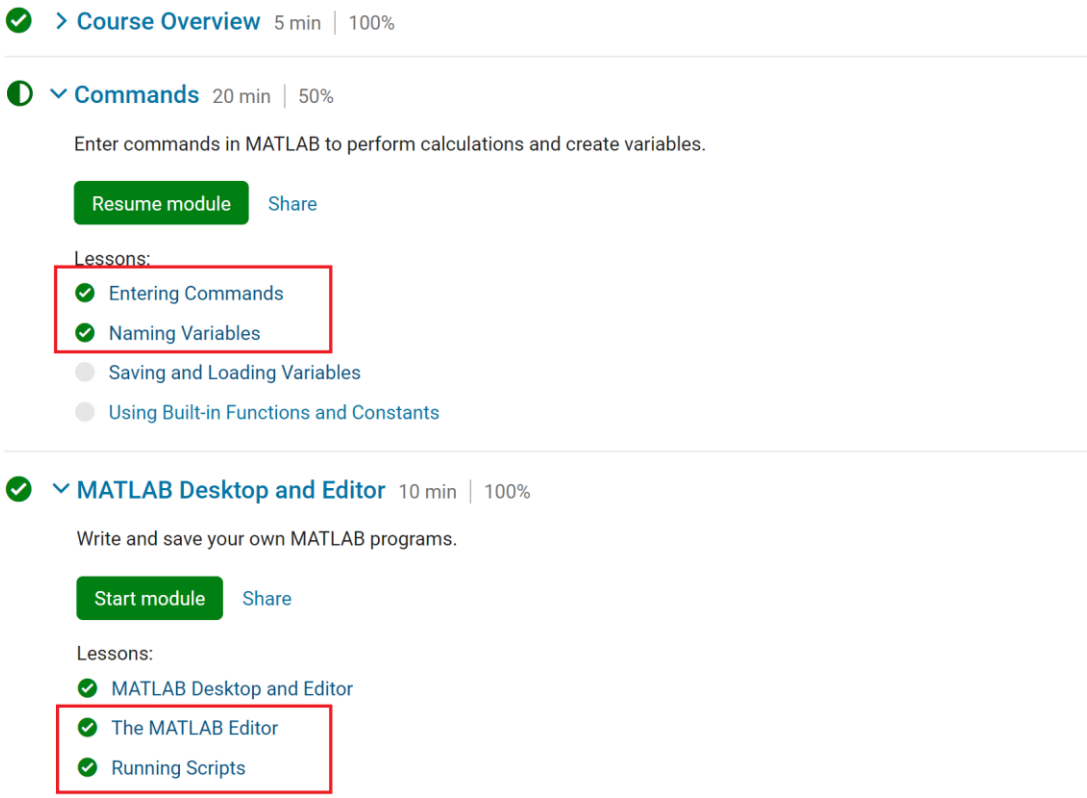
- Klicken Sie rechts neben die Zeilennummer, 

wird der Code bis zu dieser Stelle ausgeführt und hält dort an.

2: Lineare Algebra

Online-Tutorial: ca. 15-20min

https://matlabacademy.mathworks.com/details/matlab-onramp/gettingstarted



2: Lineare Algebra

Inhaltsverzeichnis

- Das Skript
- Erstellen von Matrizen
- Rechnen mit Skalaren, Vektoren und Matrizen
- Manipulation von Matrizen
- Spezielle Matrizen erstellen

2: Lineare Algebra

Erstellen von Matrizen

Werden Matrizen direkt eingegeben, ist folgendes zu beachten:

- Die einzelnen Matrixelemente werden durch Leerzeichen oder Kommas voneinander getrennt.

- Das Zeilenende in einer Matrix wird durch ein Semikolon markiert.

- Die gesamte Matrix wird von eckigen Klammern [ ] umschlossen.

2: Lineare Algebra

Erstellen von Matrizen

- Skalare Größen sind 1x1-Matrizen. Bei ihrer Eingabe sind keine eckigen Klammern nötig.

- Vektoren sind ebenfalls Matrizen. Ein Zeilenvektor ist eine 1xn-Matrix, ein Spaltenvektor eine nx1-Matrix.

2: Lineare Algebra

Erstellen von Matrizen

Beispiel: Die 2x4 Matrix

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

wird der Variable A wie folgt zugewiesen:

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

A =

     1    -3     4     2
    -5     8     2     5

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

2: Lineare Algebra

Erstellen von Matrizen

Wird Ihnen die Eingabezeile zu lang, können Sie sie durch ... umbrechen und in der nächsten Zeile fortfahren.

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

A =

     1    -3     4     2
    -5     8     2     5
```

2: Lineare Algebra

Erstellen von Matrizen

Die Dimension einer Matrix lässt sich über den Befehl `size()` bestimmen:

```
>> size(A)

ans =

     2     4
```

Dabei steht der erste Wert für die Anzahl der Zeilen, der zweite für die Anzahl der Spalten

2: Lineare Algebra

Erstellen von Matrizen

Entsprechend werden Spaltenvektoren wie folgt eingegeben:

```
>> w=[3;1;4]

w =

     3
     1
     4
```

2: Lineare Algebra

Erstellen von Matrizen

Es können auch mehrere Variablen auf einmal eingegeben werden. Dabei werden die Eingaben durch ein Komma getrennt.

Für einen Zeilenvektor und einen skalaren Wert lautet der Befehl:

```
>> v=[2 0 -1], s=7
v =
     2     0    -1
s =
     7
```

2: Lineare Algebra

Erstellen von Matrizen

Einen Überblick über die definierten Variablen verschafft der sogenannte Workspace oder der Befehl `who`:

```
>> who
Your variables are:
A   a    ans  b    s    v    w
```

Genauere Angaben liefert der Befehl `whos`

2: Lineare Algebra

Erstellen von Matrizen

```
>> whos
  Name  Size  Bytes  Class  Attributes
  A     2x4   64     double
  a     1x1    8     double
  ans   1x2   16     double
  b     1x1    8     double
  s     1x1    8     double
  v     1x3   24     double
  w     3x1   24     double
```

2: Lineare Algebra

Erstellen von Matrizen

Einzelne Variablen werden mit

```
>> clear a
```

und der gesamte Workspace mit

```
>> clear all
```

gelöscht.

2: Lineare Algebra

Erstellen von Matrizen

Der komplette Workspace kann mit

```
>> save dateiname
```

im aktuellen Verzeichnis als `dateiname.mat` gespeichert werden.

Eine alternative Schreibweise ist:

```
>> save('dateiname.mat')
```

2: Lineare Algebra

Erstellen von Matrizen

Dateien im aktuellen Verzeichnis können über

```
>> load dateiname
```

in den Workspace geladen werden.

Eine alternative Schreibweise ist:

```
>> load('dateiname.mat')
```

Eine weitere Möglichkeit ist das Laden über die Befehlsleiste via *Home* → *Import Data*

2: Lineare Algebra

Inhaltsverzeichnis

- Das Skript
- Erstellen von Matrizen
- Rechnen mit Skalaren, Vektoren und Matrizen
- Manipulation von Matrizen
- Spezielle Matrizen erstellen

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Es seien A, B Matrizen und c, d, e skalare Größen. In MatLab sind unter gewissen Dimensionsbedingungen an A, B folgende arithmetische Operationen zwischen Matrizen und Skalaren definiert:

Für $A=\left(a_{jk}\right)_{n,m}$; $j=1,...,n;$ $k=1,...m$

Symbol	Operation	MatLab Syntax	math. Syntax
+	Skalare Addition	`c+d`	c+d
+	Matrixaddition	`A+B`	A+B
+	Addition Skalar – Matrix	`c+A`	$\left(c+a_{jk}\right)_{n,m}$

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

$$a=5, \quad b=\begin{pmatrix}4\\2\\1\end{pmatrix}, \quad c=\begin{pmatrix}5\\7\\-4\end{pmatrix},$$

$$A=\begin{pmatrix}8 & 7 & 3\\2 & 5 & 1\\5 & 2 & -2\end{pmatrix}, \quad B=\begin{pmatrix}0 & -7 & 2\\3 & 2 & 6\\1 & 2\bar{i} & 0\end{pmatrix}$$

```
>> a=5; b=[4;2;1]; c=[5;7;-4];
>> A=[8 7 3;2 5 1;5 2 -2];
>> B=[0 -7 2;3 2 6;1 2i 0];
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Symbol	Operation	MatLab Syntax	math. Syntax
–	Skalare Subtraktion	`c-d`	c–d
–	Matrixsubtraktion	`A-B`	A–B
–	Subtraktion Matrix – Skalar	`A-c`	$(a_{jk} - c)_{n,m}$
*	Skalare Multiplikation	`c*d`	cd
*	Multiplikation Skalar – Matrix	`c*A`	cA
*	Matrixmultiplikation	`A*B`	AB
.*	Punktweise Multiplikation	`A.*B`	$(a_{jk} b_{jk})_{n,m}$

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Symbol	Operation	MatLab Syntax	math. Syntax
/	rechte skalare Division	`c/d`	$\frac{c}{d}$
\	linke skalare Division	`c\d`	$\frac{d}{c}$
/	rechte Division Skalar – Matrix	`A/c`	$\frac{1}{c}A$
/	rechte Matrixdivision	`A/B`	AB^{-1}
\	linke Matrixdivision	`A\B`	$A^{-1}B$
./	punktweise rechte Division	`A./B`	$\left(\frac{a_{jk}}{b_{jk}}\right)_{n,m}$
.\	punktweise linke Division	`A.\B`	$\left(\frac{b_{jk}}{a_{jk}}\right)_{n,m}$

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Symbol	Operation	MatLab Syntax	math. Syntax
^	Potenzieren	`A^c`	A^c
.^	punktweise Potenzieren	`A.^B`	$\left(a_{jk}{}^{b_{jk}}\right)_{n,m}$
.‘	transponieren	`A.'`	A^t
’	konjugiert komplex transponieren	`A'`	$\overline{A}^t$

:	Doppelpunktoperation	`c:d:e`
		`c:e`
		`(=c:1:e)`
			[c, c+d,..., c+k*d], k= $\left\lfloor\frac{e-c}{d}\right\rfloor$
			[c, c+1,..., c+k*1], k= ⌊e-c⌋

( )	Klammerausdrücke	`(A+B)*c`	(A+B)c

( )	Indizierung	`A(3,4)`
		`A(3,:)`
		`A(:,4)`
			a_{34}
			$(a_{3k})_{1,m}$
			$(a_{j4})_{n,1}$

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

$$a=5, \qquad b=\begin{pmatrix}4\\2\\1\end{pmatrix}, \qquad c=\begin{pmatrix}5\\7\\-4\end{pmatrix},$$

$$A=\begin{pmatrix}8 & 7 & 3\\2 & 5 & 1\\5 & 2 & -2\end{pmatrix}, \qquad B=\begin{pmatrix}0 & -7 & 2\\3 & 2 & 6\\1 & 2i & 0\end{pmatrix}$$

```
>> a=5; b=[4;2;1]; c=[5;7;-4];
>> A=[8 7 3;2 5 1;5 2 -2];
>> B=[0 -7 2;3 2 6;1 2i 0];
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

```
>> A*B

ans =

24.00+0.00i   -42.00+6.00i   58.00+0.00i
16.00+0.00i    -4.00+2.00i   34.00+0.00i
 4.00+0.00i   -31.00-4.00i   22.00+0.00i
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

Matrixmultiplikation

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

```
>> format shortG
>> A*B
ans =
   24 + 0i     - 42 + 6i     58 + 0i
   16 + 0i      - 4 + 2i     34 + 0i
    4 + 0i     - 31 - 4i     22 + 0i
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

punktweise Multiplikation

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

```
>> A.*B
ans =
   0 + 0i   - 49 + 0i    6 + 0i
   6 + 0i    10 + 0i    6 + 0i
   5 + 0i     0 + 4i    0 + 0i
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

Gleichungssystem Ax=b

$$A=\begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

$$b=\begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix}$$

```
>> x=A\b
x =
   0.16667
   0.29167
   0.20833
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

Vektor-Matrix Multiplikation

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix}$$

```
>> A*x
ans =
     4
     2
     1
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

konjugiert komplex

transponieren

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

```
>> B'
ans =
   0 + 0i    3 + 0i    1 + 0i
  - 7 + 0i    2 + 0i    0 - 2i
   2 + 0i    6 + 0i    0 + 0i
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

Transponieren

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

$$a = 5$$

```
>> B'+a
ans =
   5 + 0i     8 + 0i     6 + 0i
   2 + 0i     7 + 0i     5 - 2i
   7 + 0i    11 + 0i     5 + 0i
```

2: Lineare Algebra

Rechnen mit Skalaren, Vektoren und Matrizen

Beispiele:

Skalarprodukt <b,c>

$$b = \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix}$$

$$c = \begin{pmatrix} 5 \\ 7 \\ -4 \end{pmatrix}$$

```
>> b'*c
ans =
    30
```

2: Lineare Algebra

Inhaltsverzeichnis

- Das Skript
- Erstellen von Matrizen
- Rechnen mit Skalaren, Vektoren und Matrizen
- Manipulation von Matrizen
- Spezielle Matrizen erstellen

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

Einzelne Matrixelemente

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

```
>> B(2,3)
ans =
     6
```

2: Lineare Algebra

Manipulation von Matrizen

Beispiele: Matrixkombination

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

C =

   8+0i    7+0i    3+0i
   2+0i    5+0i    1+0i
   5+0i    2+0i   -2+0i
   0+0i   -7+0i    2+0i
   3+0i    2+0i    6+0i
   1+0i    0+2i    0+0i
```

$$A=\begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

$$B=\begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

Kombination von Matrizen

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

```
>> C=[A B]

C =

8+0i   7+0i   3+0i   0+0i  -7+0i   2+0i
2+0i   5+0i   1+0i   3+0i   2+0i   6+0i
5+0i   2+0i  -2+0i   1+0i   0+2i   0+0i
```

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

Kombination von Matrizen

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

```
>> D=[A;[1 0 3]]
D =
     8     7     3
     2     5     1
     5     2    -2
     1     0     3
```

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

Kombination von Matrizen

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

```
>> D=[A [1 0 3]']
D =
     8     7     3     1
     2     5     1     0
     5     2    -2     3
```

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

Lineare Indizierung;

MatLab interpretiert dabei die Matrix als einen einzigen langen Spaltenvektor

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

```
>> A(6)
ans =
     2
```

Indexnummern in diesem Beispiel:

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

`:` – Operator

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

Mit ihm kann man z. B. Teile einer Matrix extrahieren. Der folgende Befehl gibt die erste Zeile von `A` aus.

```
>> A(1,:)
ans =
     8     7     3
```

Die Nummer 1 innerhalb der Klammern bedeutet ’die erste Zeile’ und der Doppelpunkt steht für ’alle Spalten‘.

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

: – Operator

```
>> A(:,2)
ans =
     7
     5
     2
```

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

dagegen gibt zunächst alle Zeilen aus, aber nur die Elemente, die in der 2. Spalte stehen.

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

`:` – Operator

Der `:` – Operator kann auch außerhalb einer Indizierung angewandt werden.

Die vollständige Syntax lautet

Startpunkt `:` Schrittweite `:` Endpunkt

und kann bei Schrittweite=1 auf

Startpunkt `:` Endpunkt

gekürzt werden.

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

```
>> 5:0.2:6.6
ans =
5   5.2   5.4   5.6   5.8   6 6.2   6.4   6.6

>> 4:8
ans =
     4     5     6     7     8
```

2: Lineare Algebra

Manipulation von Matrizen

Beispiele:

```
>> 10:-2:0
ans =
    10     8     6     4     2     0
>> 1:0.3:2
ans =
    1.0000    1.3000    1.6000    1.9000
```

2: Lineare Algebra

Manipulation von Matrizen

Alternative: `linspace`

`linspace(a,b,n)` erzeugt `n` Punkte mit Abstand `(b-a)/(n-1)`.

Die Start- und Endpunkt sind hier immer `a` und `b`.

```
>> linspace(3,5,6)
ans =
    3    3.4    3.8    4.2    4.6    5
```

2: Lineare Algebra

Manipulation von Matrizen

Vektoren können ebenfalls zur Indizierung genutzt werden.

Beispiel:

```
>> x=[9 2 4 8 2 0 1 4 6 3 7]
x =
   9   2   4   8   2   0   1   4   6   3   7
>> x(1:2:end)
ans =
   9       4       2       1       6       7
```

‘end’ steht hierbei für den Indexwert des letzten Elements.

2: Lineare Algebra

Manipulation von Matrizen

Negative Schrittweiten bewirken eine umgekehrte Ausgabe:

```
x =
   9   2   4   8   2   0   1   4   6   3   7

>> x(end-2:-3:1)
ans =
   6       0       4
```

2: Lineare Algebra

Manipulation von Matrizen

Bei Matrizen kann der `:` – Operator auch gleichzeitig für Zeilen und Spalten verwendet werden:

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

```
>> A(1:2,2:3)
ans =
     7     3
     5     1
```

2: Lineare Algebra

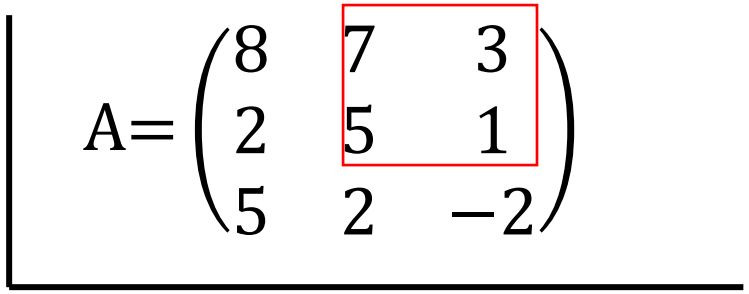
Manipulation von Matrizen

Bei Matrizen kann der

: – Operator auch gleichzeitig

für Zeilen und Spalten

verwendet werden:




```
>> A(1:2,2:3)
ans =
     7     3
     5     1
```

Hier: Zeilen 1 und 2, Spalten 2 und 3

2: Lineare Algebra

Manipulation von Matrizen

Durch absteigende Indizierung Können Zeilen oder Spalten getauscht werden:

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

```
>> A(:,[3 2 1])
ans =
     3     7     8
     1     5     2
    -2     2     5
```

vertauscht die Spalten 1 und 3

2: Lineare Algebra

Manipulation von Matrizen

Zeilen und Spalten können mit Hilfe eines leeren Vektors `[]` gelöscht werden.

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

```
>> A(:,1)=[]
A =
     7     3
     5     1
     2    -2
```

löscht die erste Spalte der Matrix A.

2: Lineare Algebra

Manipulation von Matrizen

```
>> A=[87 3;2 5 1;5 2 -2];
```

$$A=\begin{pmatrix}8 & 7 & 3\\ 2 & 5 & 1\\ 5 & 2 & -2\end{pmatrix}$$

Mit dem Befehl ‘`diag`’ lässt sich die Hauptdiagonale einer Matrix ausgeben.

```
>> diag(A)
ans =
     8
     5
    -2
```

2: Lineare Algebra

Manipulation von Matrizen

Mit zusätzlichen Argumenten `diag(A,n)` besteht Zugriff auf die Nebendiagonalen.

$$A=\begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

Positive Werte stehen für die n-te obere, negative für die n-te untere Nebendiagonale.

```
>> diag(A,1)
ans =
     7
     1
```

```
>> diag(A,-1)
ans =
     2
     2
```

2: Lineare Algebra

Manipulation von Matrizen

Der Befehl ‘`diag`’ kann auch zur Erzeugung von Matrizen eingesetzt werden.

Setzt man als erstes Element keine Matrix, sondern einen Vektor ein, wird eine Matrix erzeugt, deren n-te Nebendiagonale diesem Vektor entspricht.

Alle anderen Werte sind 0.

2: Lineare Algebra

Manipulation von Matrizen

Beispiel:

$$b = \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix}$$

```
>> diag(b)
ans =
     4     0     0
     0     2     0
     0     0     1
```

```
>> diag(b,1)
ans =
     0     4     0     0
     0     0     2     0
     0     0     0     1
     0     0     0     0
```

2: Lineare Algebra

Manipulation von Matrizen

Zum spiegeln einer Matrix werden die Befehle ‚`fliplr`‘ (links/rechts) und ‚`flipud`‘ (oben/unten) verwendet.

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

```
>> fliplr(A)
ans =
     3     7     8
     1     5     2
    -2     2     5
```

```
>> flipud(A)
ans =
     5     2    -2
     2     5     1
     8     7     3
```

2: Lineare Algebra

Manipulation von Matrizen

Auf die linke untere Dreiecksmatrix kann mit ‘`tril`’, auf die rechte obere Dreiecksmatrix mit ‘`triu`’ zugegriffen werden.

$$A = \begin{pmatrix} 8 & 7 & 3 \\ 2 & 5 & 1 \\ 5 & 2 & -2 \end{pmatrix}$$

```
>> tril(A)
ans =
     8     0     0
     2     5     0
     5     2    -2
```

```
>> triu(A)
ans =
     8     7     3
     0     5     1
     0     0    -2
```

2: Lineare Algebra

Online-Tutorial: ca. 35min

Commands 20 min | 50%

Enter commands in MATLAB to perform calculations and create variables.

Resume module Share

Lessons:

- Entering Commands
- Naming Variables
- Saving and Loading Variables
- Using Built-in Functions and Constants

MATLAB Desktop and Editor 10 min | 100%

Vectors and Matrices 15 min

Create MATLAB variables that contain multiple elements.

Start module Share

Lessons:

- Manually Entering Arrays
- Creating Evenly-Spaced Vectors
- Array Creation Functions

Indexing into and Modifying Arrays 15 min

Use indexing to extract and modify rows, columns, and elements of MATLAB arrays.

Start module Share

Lessons:

- Indexing into Arrays
- Extracting Multiple Elements
- Changing Values in Arrays

Array Calculations 5 min

Perform calculations on entire arrays at once.

Start module Share

Lessons:

- Performing Array Operations on Vectors

2: Lineare Algebra

Inhaltsverzeichnis

- Das Skript
- Erstellen von Matrizen
- Rechnen mit Skalaren, Vektoren und Matrizen
- Manipulation von Matrizen
- Spezielle Matrizen erstellen

2: Lineare Algebra

Manipulation von Matrizen

Weitere Befehle zur Erzeugung von Matrizen:

`zeros(n,m)`	Nullmatrix mit n Zeilen und m Spalten
`ones(n,m)`	Matrix mit n Zeilen und m Spalten besetzt mit Einsen
`eye(n)`	Einheitsmatrix mit n Zeilen und Spalten
`rand(n,m)`	Matrix mit n Zeilen und m Spalten besetzt mit gleichverteilten Zufallswerten aus dem Interval [0,1]
`magic(n)`	Matrix mit n Zeilen und Spalten, mit gleicher Zeilen- und Spaltensumme

2: Lineare Algebra

Manipulation von Matrizen

```
>> zeros(3,4)
ans =
     0     0     0     0
     0     0     0     0
     0     0     0     0
>> ones(2,5)
ans =
     1     1     1     1     1
     1     1     1     1     1
```

2: Lineare Algebra

Manipulation von Matrizen

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
>> rand(2,3)
ans =
    0.81472    0.12699    0.63236
    0.90579    0.91338    0.09754
```

2: Lineare Algebra

Manipulation von Matrizen

Gleichverteilte Zufallswerte im Intervall `[a,b]` können mit

`rand(n,m)*(b-a)+a` erzeugt werden. Z. B im Intervall `[2,5]`:

```
>> rand(2,3)*(5-2)+2
ans =
    2.8355    4.8725    2.4728
    3.6406    4.8947    4.9118
```

2: Lineare Algebra

Manipulation von Matrizen

Ein magisches Quadrat mit Zeilen- und Spaltensumme 15 wird mit

```
>> magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2
```

erzeugt.

Inhaltsverzeichnis

3: Funktionen & Operatoren

Inhaltsverzeichnis

3: Funktionen & Operatoren

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Lernziele:

- Sie kennen die wichtigsten vordefinierten Funktionen und Konstanten und können diese anwenden.
- Sie können Relationsoperatoren und logische Operatoren anwenden.
- Sie kennen die wichtigsten Funktionen zu Polynomen.

3: Funktionen & Operatoren

Inhaltsverzeichnis

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

Die wichtigsten vordefinierten Konstante sind:

MatLab Bezeichnung	math. Bezeichnung	Erläuterung
`pi`	π	
`eps`		Maschinengenauigkeit
`i, j`	i	Imaginäre Einheit
`Inf`	∞	
`NaN`		not a number, z. B. `inf-inf, 0/0`
`realmin`		kleinste positive Maschinenzahl
`realmax`		größte positive Maschinenzahl
`intmax`		größte ganze Zahl (int32)
`intmin`		kleinste ganze Zahl

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

```
>> format long
>> pi
ans =
   3.141592653589793

>> eps
ans =
    2.220446049250313e-16
```

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

eps gibt die Maschinengenauigkeit an, mit der MatLab rechnet.

Genauer gesagt gibt eps den Abstand von 1 zur nächsten größeren maschinell darstellbaren Zahl an.

Wenn rd(x) die Zahl x auf die Maschinenzahl rundet, die x am nächsten liegt, gilt

$$\left|\frac{x - rd(x)}{x}\right| \leq eps.$$

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

Beispiel: Die Zahl `1+1e-16` ist in MatLab nicht darstellbar und wird auf 1 gerundet.

```
>> 1+1e-16
ans =
     1
```

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

Imaginäre Zahlen können sowohl mit `i`, als auch mit `j` definiert werden.

```
>> format short
>> i
ans =
   0.0000 + 1.0000i
>> j
ans =
   0.0000 + 1.0000i
```

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

inf kann entweder direkt verwendet werden

```
>> Inf
ans =
   Inf
```

oder als Ergebnis einer Rechnung entstehen.

```
>> 1/0
ans =
   Inf
```

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

Gleiches gilt für NaN (not a number).

```
>> NaN
ans =
   NaN

>> 0/0
ans =
   NaN
```

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

Die kleinste darstellbare positive normalisierte Maschinenzahl ist

```
>> realmin
ans =
   2.2251e-308
```

Die kleinste positive nicht-normalisierte Maschinenzahl

```
>> eps*realmin
ans =
     4.94065645841246e-324
```

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

Die kleinste darstellbare positive normalisierte Maschinenzahl ist

```
>> realmin
ans =
   2.2251e-308
```

Die kleinste positive nicht-normalisierte Maschinenzahl

```
>> eps*realmin
ans =
   4.940656458412465e-324
```

```
>> 1e-324
ans =
     0
```

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

Die größte darstellbare positive Maschinenzahl ist

```
>> realmax
ans =
   1.7977e+308

>> 1.7978e+308
ans =
     Inf
```

3: Funktionen & Operatoren

Darstellung von Zahlen und Konstanten

Die größte ganze und die kleinste ganze Zahl (int32) ist

```
>> intmax
ans =
  int32
   2147483647
>> intmin
ans =
  int32
   -2147483648
```

3: Funktionen & Operatoren

Inhaltsverzeichnis

3: Funktionen & Operatoren

Wichtige Funktionen

Was ist mit der Eulerschen Zahl `e`?

Die Eulersche Zahl hat keinen eigenen definierten Befehl.

Stattdessen kann sie über die Exponentialfunktion `exp` genutzt werden.

```
>> exp(1)
ans =
    2.718281828459046
```

3: Funktionen & Operatoren

Wichtige Funktionen

`exp` gehört zur Klasse der skalaren Funktionen, auch wenn der Funktion Matrizen übergeben werden können.

Skalar in diesem Fall bedeutet, dass die Funktion auf jedes skalare Element einer Matrix einzeln angewandt wird.

Zusätzlich gibt es noch Array-Funktionen wie etwa max, die bei Übergabe eines Vektors das größte Element des Vektors ausgibt.

3: Funktionen & Operatoren

Wichtige Funktionen

Die wichtigsten skalaren Funktionen sind:

MatLab Bezeichnung	math. Syntax	Beschreibung
`exp`	exp( )	Exponentialfunktion
`log, log10`	ln( ), $\log_{10}$( )	Logarithmus
`sqrt`	$\sqrt{}$	Wurzel
`mod`	mod(,)	Modulo
`sin, cos, tan`	sin( ), cos( ), tan( )	trigonometrische Funktion
`sinh, cosh, tanh`	sinh( ), cosh( ), tanh( )	hyperbolicus Funktionen
`asin, acos, atan`	arcsin( ), arcos( ), arctan( )	trigonometrische Funktion

3: Funktionen & Operatoren

Wichtige Funktionen

Die wichtigsten skalaren Funktionen sind:

MatLab Bezeichnung	math. Syntax	Beschreibung
`abs`	$\|\ \|$	Betrag
`imag`	$\Im(\ )$	Imaginärteil
`real`	$\Re(\ )$	Realteil
`conj`		konjugieren
`sign`		Vorzeichen
`round`		Runden zur nächsten ganzen Zahl
`floor`		Runden zur nächsten kleineren ganzen Zahl
`ceil`		Runden zur nächsten größeren ganzen Zahl

3: Funktionen & Operatoren

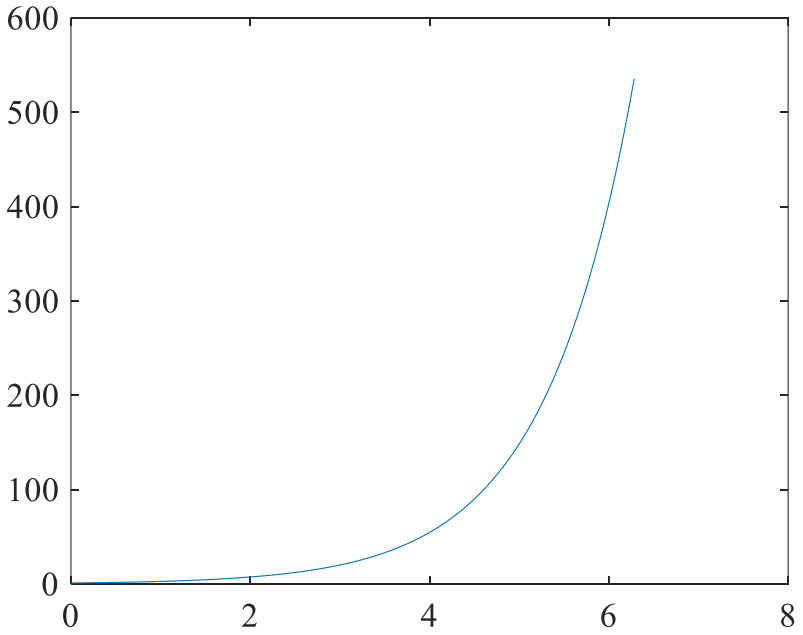
Wichtige Funktionen

```
>> format short; exp(0)
ans =
     1
>> exp(-Inf)
ans =
     0
>> exp([0 1 2])
ans =
    1.0000    2.7183    7.3891
```

3: Funktionen & Operatoren

Wichtige Funktionen

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



3: Funktionen & Operatoren

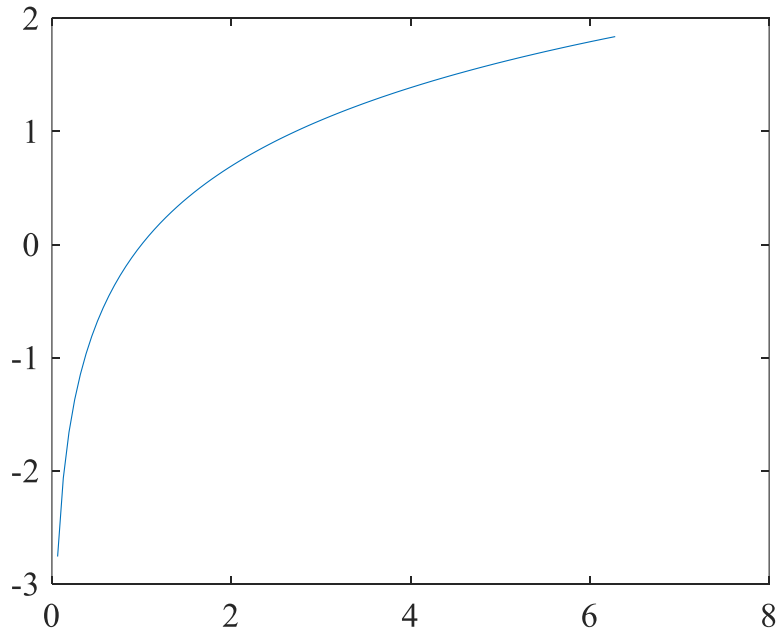
Wichtige Funktionen

```
>> log(0)
ans =
   -Inf
>> log(1)
ans =
     0
>> log([0 1 2])
ans =
      -Inf         0    0.6931
```

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> plot(x,log(x))
```



3: Funktionen & Operatoren

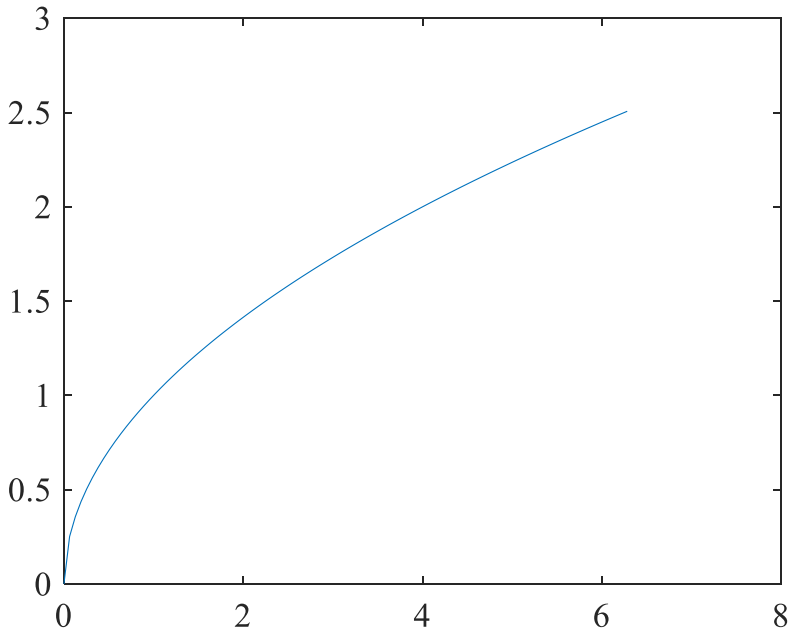
Wichtige Funktionen

```
>> sqrt(1)
ans =
     1
>> sqrt(2)
ans =
   1.4142
>> sqrt([0 1 2])
ans =
         0    1.0000    1.4142
```

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> plot(x,sqrt(x))
```



3: Funktionen & Operatoren

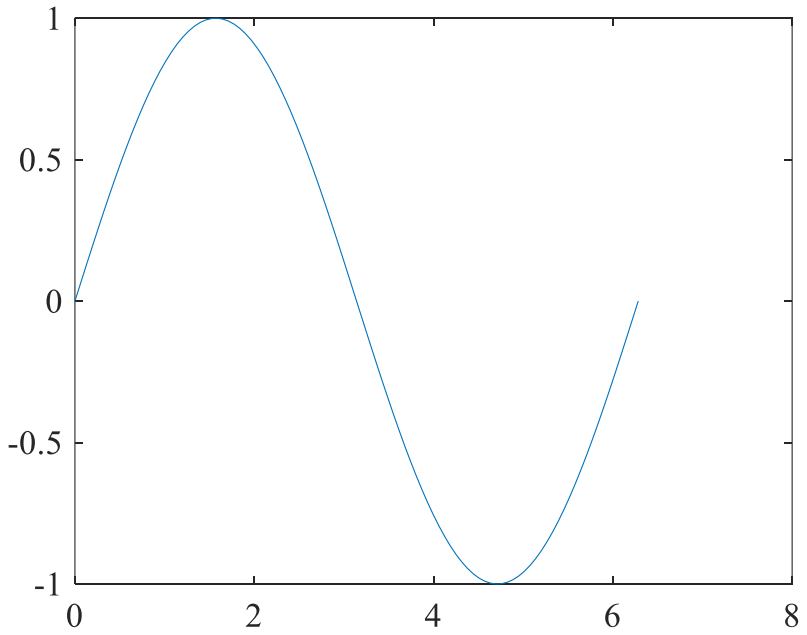
Wichtige Funktionen

```
>> sin(0)
ans =
     0
>> sin(pi/2)
ans =
     1
>> sin([0 pi/2 pi])
ans =
         0    1.0000    0.0000
```

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> plot(x,sin(x))
```



3: Funktionen & Operatoren

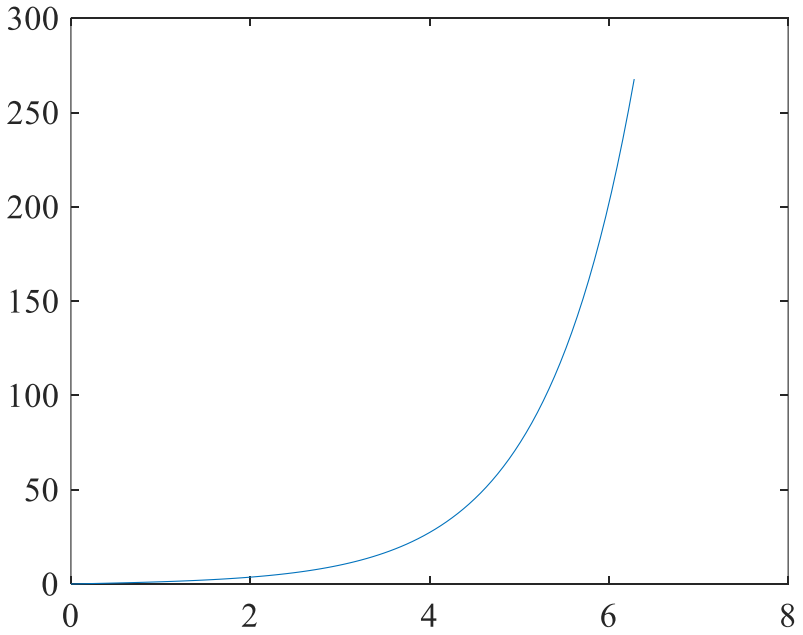
Wichtige Funktionen

```
>> sinh(0)
ans =
     0
>> sinh(pi/2)
ans =
    2.3013
>> sinh([0 pi/2 pi])
ans =
         0    2.3013   11.5487
```

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> plot(x,sinh(x))
```



3: Funktionen & Operatoren

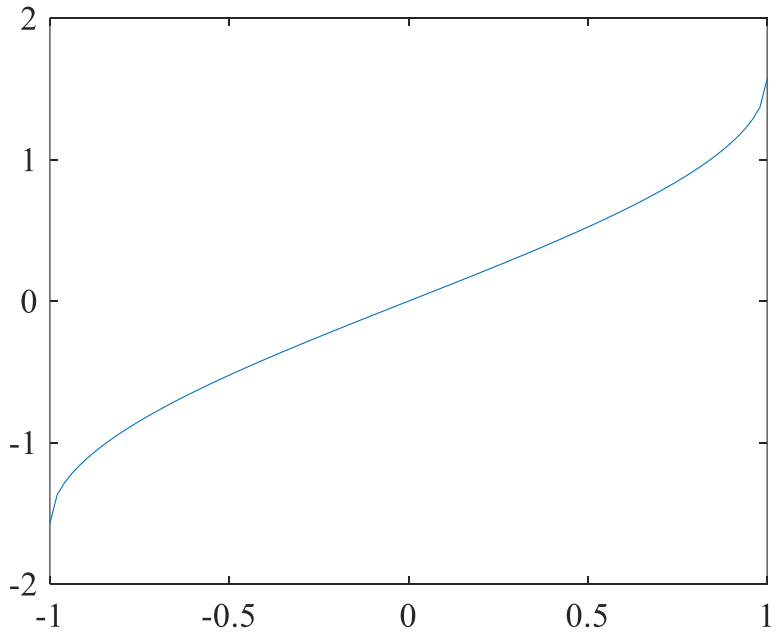
Wichtige Funktionen

```
>> asin(0)
ans =
     0
>> asin(1)
ans =
    1.5708
>> asin([-1 0 1])
ans =
   -1.5708         0    1.5708
```

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> x=linspace(-1,1,100);
>> plot(x,asin(x))
```



3: Funktionen & Operatoren

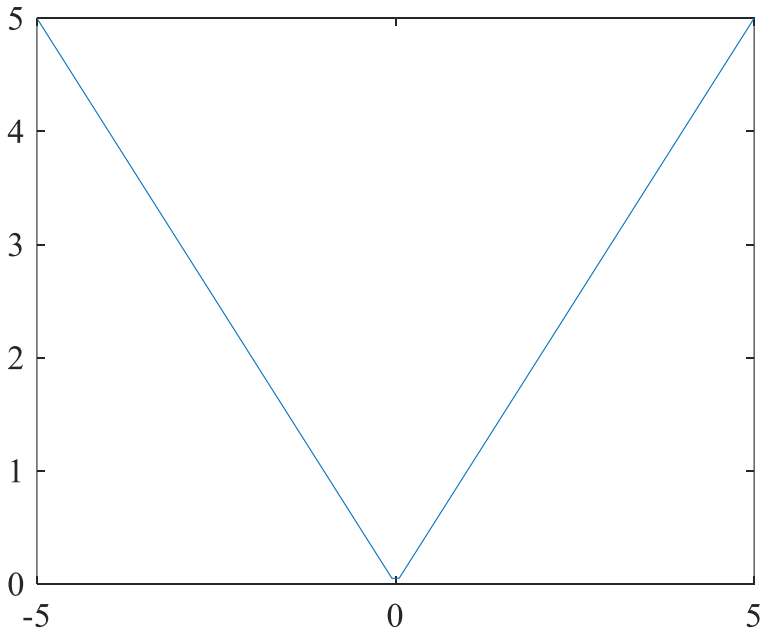
Wichtige Funktionen

```
>> abs(-1.3)
ans =
    1.3000
>> abs(4.7)
ans =
    4.7000
>> abs([-2.4 3.2 -1.5])
ans =
    2.4000    3.2000    1.5000
```

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> x=linspace(-5,5,100);
>> plot(x,abs(x))
```



3: Funktionen & Operatoren

Wichtige Funktionen

```
>> imag(1.2+0.2i)
ans =
    0.2000

>> real(1.2+0.2i)
ans =
    1.2000

>>imag([-2.1-4.7i  4.6+9.3i])
ans =
   -4.7000   9.3000

>> conj([-2.1-4.7i 4.6+9.3i])
ans =
  -2.1000 + 4.7000i   4.6000 - 9.3000i
```

3: Funktionen & Operatoren

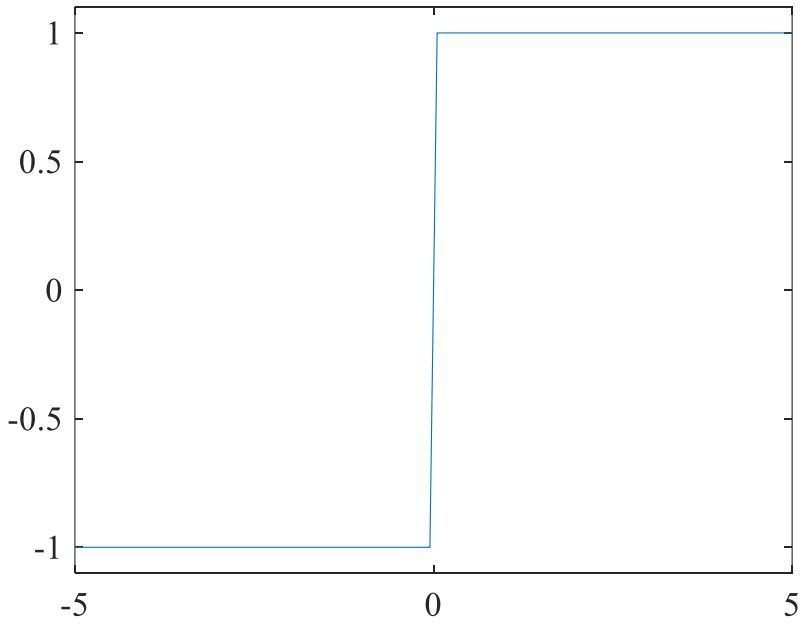
Wichtige Funktionen

```
>> sign(-1.8)
ans =
    -1
>> sign(0)
ans =
     0
>> sign(10)
ans =
     1
```

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> plot(x,sign(x))
```



3: Funktionen & Operatoren

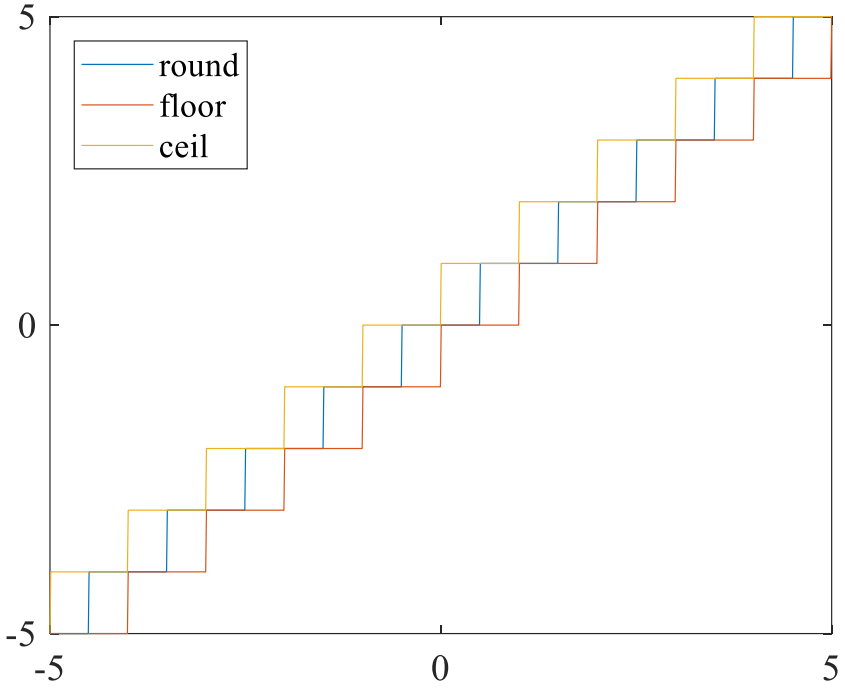
Wichtige Funktionen

```
>> round(1.3)
ans =
     1
>> floor(1.3)
ans =
     1
>> ceil(1.3)
ans =
     2
```

```
>> round(4.5)
ans =
     5
>> floor(4.5)
ans =
     4
>> ceil(4.5)
ans =
     5
```

3: Funktionen & Operatoren

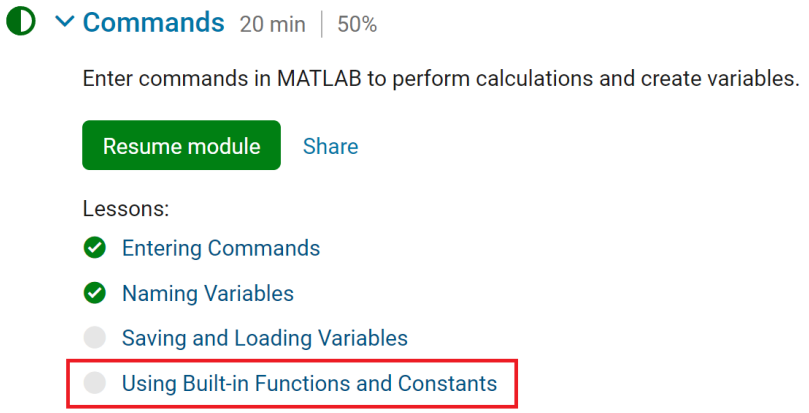
Wichtige Funktionen




```
>> plot(x,round(x))
>> hold on
>> plot(x,floor(x))
>> plot(x,ceil(x))
>> legend('round','floor','ceil',…
     'Location','northwest')
```

2: Lineare Algebra

Online-Tutorial: ca. 10min



3: Funktionen & Operatoren

Wichtige Funktionen

Eine zweite Klasse von MatLab-Funktionen sind Vektorfunktionen.

Sie können mit derselben Syntax sowohl auf Zeilen- wie auf Spaltenvektoren angewandt werden.

Solche Funktionen operieren spaltenweise, wenn sie auf Matrizen angewandt werden.

3: Funktionen & Operatoren

Wichtige Funktionen

Die wichtigsten Vektorfunktionen sind:

MatLab Bezeichnung	Beschreibung
`max`	größtes Element
`mean`	Mittelwert
`min`	kleinstes Element
`sum`	Summe aller Elemente
`prod`	Produkt aller Elemente
`sort`	Sortieren der Elemente in auf oder absteigender Reihenfolge
`sortrows`	Sortieren einer Zeile in aufsteigender Reihenfolge

3: Funktionen & Operatoren

Wichtige Funktionen

Die wichtigsten Vektorfunktionen sind:

MatLab Bezeichnung	Beschreibung
`std`	Standardabweichung
`trapz`	numerische Integration mit der Trapezregel
`transpose`	transponieren
`det`	Determinante einer Matrix
`inv`	Inverse einer Matrix
`diag`	Diagonale von Matrizen (s. Kapitel 2)
`fliplr, flipud`	Spiegelung von Matrizen (s. Kapitel 2)
`any, all`	Teste ob mindestens eine (any) oder alle (all) Komponenten einer Matrix ungleich 0 sind

3: Funktionen & Operatoren

Wichtige Funktionen

```
>>A=[0 -7 0;-2 5 0;5 0 -2];
>>b=[4;-2;1];

>>max(b)
ans =
     4
>>max(A)
ans =
     5     5     0
```

$$A=\begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b=\begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> mean(b)
ans =
     1
>> mean(A)
ans =
    1.0000   -0.6667   -0.6667
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> sum(b)
ans =
     3
>> sum(A)
ans =
     3    -2    -2
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> prod(b)
ans =
    -8
>> prod(A)
ans =
     0     0     0
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> sort(b)
ans = -2
       1
       4

>> sort(A)
ans =
     -2     -7     -2
      0      0      0
      5      5      0
```

$$A=\begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b=\begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> sort(b,'descend')
ans =    4
         1
        -2

>> sort(A,'descend')
ans =
     5     5     0
     0     0     0
    -2    -7    -2
```

$$A=\begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b=\begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> sort(b,2)
ans = 4
      -2
       1
>> sort(A,2)
ans =
      -7       0       0
      -2       0       5
      -2       0       5
```

$$A=\begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b=\begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

3: Funktionen & Operatoren

Wichtige Funktionen

`sortrows(A,column)` sortiert die Zeilen einer Matrix nach der Spaltennummer in column.

$$A=\begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

Dabei entspricht `sortrows(A)=sortrows(A,1)`.

```
>> sortrows(A)
ans =
    -2     5     0
     0    -7     0
     5     0    -2
```

```
>> sortrows(A,2)
ans =
     0    -7     0
     5     0    -2
    -2     5     0
```

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> std(b)
ans =
     3
>> std(A)
ans =
    3.6056    6.0277    1.1547
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

$$std(b) = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}|b_k - \mu|^2}, \qquad \mu = \frac{1}{n}\sum_{k=1}^{n} b_k$$

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> det(A)
ans =
     28

>> inv(A)
ans =
   -0.3571   -0.5000         0
   -0.1429         0         0
   -0.8929   -1.2500   -0.5000
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

3: Funktionen & Operatoren

Wichtige Funktionen

Aus der MatLab Hilfe zum Befehl inv steht unter Tipps:

It is seldom necessary to form the explicit inverse of a matrix. A frequent misuse of inv arises when solving the system of linear equations $Ax = b$.

One way to solve the equation is with x = inv(A)*b. A better way, from the standpoint of both execution time and numerical accuracy, is to use the matrix backslash operator x = A\b.

This produces the solution using Gaussian elimination, without explicitly forming the inverse.

3: Funktionen & Operatoren

Wichtige Funktionen

```
>> any(A)
ans =
  1×3 logical array
   1   1   1
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix}$$

```
>> all(A)
ans =
  1×3 logical array
   0   0   0
```

```
>> all(b)
ans =
  logical
   1
```

3: Funktionen & Operatoren

Wichtige Funktionen

Weitere Funktionen finden Sie über die MatLab-Suche.

3: Funktionen & Operatoren

Inhaltsverzeichnis

3: Funktionen & Operatoren

Relationsoperatoren

Es gibt folgende Relationsoperatoren:

MatLab-Syntax	mathematischer Syntax
`A > B`	$A > B$
`A < B`	$A < B$
`A >= B`	$A \geq B$
`A <= B`	$A \leq B$
`A == B`	$A = B$
`A ~= B`	$A \neq B$

3: Funktionen & Operatoren

Relationsoperatoren

Relationsoperatoren können auf Skalare, Vektoren und Matrizen angewandt werden.

Bei Vektoren und Matrizen werden die einzelnen Komponenten verglichen.

→ A und B müssen die selbe Dimension haben.

3: Funktionen & Operatoren

Relationsoperatoren

Das Ergebnis eines Relationstests ist der boolsche Operator, d. h. `1` (true) oder `0` (false)

Beispiel:

```
>> 5>2
ans =
  logical
   1
```

3: Funktionen & Operatoren

Relationsoperatoren

```
>> 4==4
ans =
  logical
   1

>> 4~=4
ans =
  logical
   0
```

3: Funktionen & Operatoren

Relationsoperatoren

```
>> x=[1 2 3]; y=[-1 4 6]; z=[1 0 -7];
>> y>x
ans =
  1×3 logical array
   0   1   1
>> z==x
ans =
  1×3 logical array
   1   0   0
```

$$x=(1 \quad 2 \quad 3)$$

$$y=(-1 \quad 4 \quad 6)$$

$$z=(1 \quad 0 \quad -7)$$

3: Funktionen & Operatoren

Relationsoperatoren

Boolsche Operatoren und damit auch die Relationsoperatoren können zur Indizierung von Matrizen verwendet werden.

```
>> x=[1 2 3 4 5];    y=[-5 3 2 4 1];
>> x([0 1 0 1 0])
```

Subscript indices must either be real positive integers or logicals.

$$x=(1 \quad 2 \quad 3 \quad 4 \quad 5)$$

$$y=(-5 \quad 3 \quad 2 \quad 4 \quad 1)$$

3: Funktionen & Operatoren

Relationsoperatoren

Logische Matrizen werden mit `logical(A)` erzeugt. Alle Elemente gleich 0 werden zu einem logischen false, alle anderen zu einem logischen true.

A darf weder `NaN`, noch `Inf` enthalten.

```
>>A=[0 -7 0;-2 5 0; 5 0 -2]; logical(A)

ans =

  3×3 logical array

   0   1   0
   1   1   0
   1   0   1
```

$$A=\begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

3: Funktionen & Operatoren

Relationsoperatoren

```
logical([0 1 0 1 0])
ans =
  1×5 logical array
   0   1   0   1   0
```

$$x=(1 \quad 2 \quad 3 \quad 4 \quad 5)$$

$$y=(-5 \quad 3 \quad 2 \quad 4 \quad 1)$$

```
>>x(logical([0 1 0 1 0]))
ans =
     2     4
```

```
>>x(y>=3)
ans =
     2     4
```

3: Funktionen & Operatoren

Relationsoperatoren

Vorsicht bei komplexen Zahlen!

> , >= , < und <= vergleichen nur den Realteil,
~= und == sowohl Real- als auch Imaginärteil.

Wie vergleiche ich, ob zwei Matrizen gleich sind?

3: Funktionen & Operatoren

Relationsoperatoren

Methode 1: Befehl `all` (Kapitel 3, wichtige Funktionen)

```
>> all(A(:)==B(:))
ans =
  logical
    0
>> all(A(:)==A(:))
ans =
  logical
    1
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

3: Funktionen & Operatoren

Relationsoperatoren

Der `:` – Operator wandelt eine Matrix in einen Longvector um, indem die Spalten untereinander geschrieben werden.

```
>> A(:)
ans =
     0
    -2
     5
    -7
     5
     0
     0
     0
    -2
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

3: Funktionen & Operatoren

Relationsoperatoren

`all` testet, ob alle Elemente einer Spalte ungleich 0 sind.

```
>> all(B)
ans =
  1×3 logical array
   0   1   0
>> all(B(:))
ans =
  logical
   0
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

3: Funktionen & Operatoren

Relationsoperatoren

Methode 2: Einfacher ist der Befehl `isequal( , )`

```
>> isequal(A,A)
ans =
  logical
   1
>> isequal(A,B)
ans =
  logical
   0
```

$$A = \begin{pmatrix} 0 & -7 & 0 \\ -2 & 5 & 0 \\ 5 & 0 & -2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & -7 & 2 \\ 3 & 2 & 6 \\ 1 & 2i & 0 \end{pmatrix}$$

3: Funktionen & Operatoren

Inhaltsverzeichnis

3: Funktionen & Operatoren

Logische Operatoren

Es gibt vier logische Operatoren:

- und (`&`)
- oder (`|`)
- nicht (`~`)
- aussschließendes oder (`xor`)

3: Funktionen & Operatoren

Logische Operatoren

Es gilt folgende Wahrheitstabelle:

		und	oder	nicht	ausschließendes oder
A	B	`A & B`	`A | B`	`~A`	`xor(A,B)`
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Ebenso wie die Relationsoperatoren wirken logische Operatoren komponentenweise, wenn sie auf Matrizen angewandt werden.

3: Funktionen & Operatoren

Logische Operatoren

```
>> u=[0 0 1 -3 0 1];
>> v=[0 1 7  0 0 -1];
```

$$u = (0 \quad 0 \quad 1 \quad -3 \quad 0 \quad 1)$$
$$v = (0 \quad 1 \quad 7 \quad 0 \quad 0 \quad -1)$$

~u gibt an den Stellen, an denen u gleich 0 ist eine 1 aus, an allen anderen Stellen eine 0.

```
>> ~u
ans =
1×6 logical array
 1  1  0  0  1  0
```

```
>> ~v
ans =
 1×6 logical array
   1  0  0  1  1  0
```

3: Funktionen & Operatoren

Logische Operatoren

`&` liefert komponentenweise eine 1, wenn sowohl u, als auch v an dieser Stelle ungleich 0 ist. Sonst wird 0 ausgegeben.

$$u=(0 \quad 0 \quad 1 \quad -3 \quad 0 \quad 1)$$
$$v=(0 \quad 1 \quad 7 \quad 0 \quad 0 \quad -1)$$

```
>> u&v
ans =
  1×6 logical array
   0   0   1   0   0   1
```

3: Funktionen & Operatoren

Logische Operatoren

| liefert komponentenweise eine 1, falls u, v oder u und v an dieser Stelle ungleich 0 ist. Sonst wird 0 ausgegeben.

$$\begin{vmatrix} u = (0 & 0 & 1 & -3 & 0 & 1) \\ v = (0 & 1 & 7 & 0 & 0 & -1) \end{vmatrix}$$

```
>> u|v
ans =
  1×6 logical array
   0   1   1   1   0   1
```

3: Funktionen & Operatoren

Logische Operatoren

`xor` liefert komponentenweise eine 1, falls entweder u, oder v an dieser Stelle ungleich 0 ist, wobei entweder u oder v gleich 0 sein muss. Sonst wird 0 ausgegeben.

$$\begin{array}{l} u = (0 \quad 0 \quad 1 \quad -3 \quad 0 \quad \quad 1) \\ v = (0 \quad 1 \quad 7 \quad \quad 0 \quad 0 \quad -1) \end{array}$$

```
>> xor(u,v)
ans =
  1×6 logical array
   0   1   0   1   0   0
```

3: Funktionen & Operatoren

Logische Operatoren

Vorsicht beim hintereinander ausführen von logischen Operatoren! Sind die Operatoren gleich, wird von links nach rechts ausgewertet.

```
u | v | w = (u | v) | w
```

Allerdings wird ~ vor &, sowie & vor | ausgeführt.

```
u | ~v & w = u | ((~v) & w)
```

Bei Unsicherheit setzen Sie Klammern.

3: Funktionen & Operatoren

Logische Operatoren

Short-circuit Operatoren `&&` und `||`

`&&`, sowie `||` wirken auf skalare Größen und entsprechen dort den Operatoren `&` und `|`, sind jedoch effizienter.

Bei dem Befehl

```
expr_1 & expr_2 & expr_3 & expr_4 & expr_5
```

testet MatLab alle Ausdrücke und entscheidet dann, ob er `true` (1) oder `false` (0) ist.

3: Funktionen & Operatoren

Logische Operatoren

Short-circuit Operatoren && und ||

```
expr_1 && expr_2 && expr_3 && expr_4
```

Im Gegensatz dazu untersucht MatLab hier zunächst `expr_1`. Ist diese schon falsch, wird der Rest nicht mehr ausgewertet und das Ergebnis ist `false` (0). Ist er war, wird der nächste Ausdruck ausgewertet, usw.
Somit ist diese Variante schneller.

3: Funktionen & Operatoren

Logische Operatoren

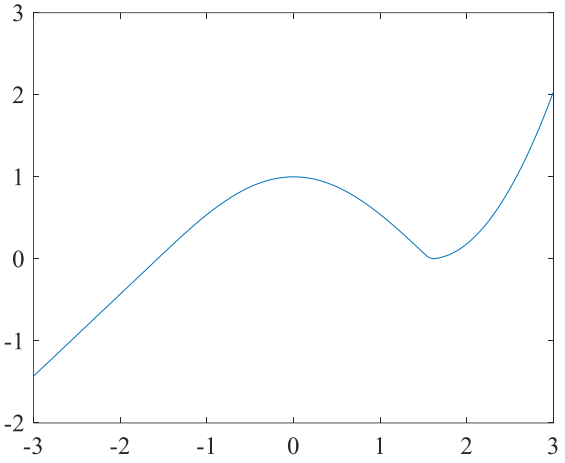
Anwendungsbeispiel: Stückweise definierte Funktionen

$$f(x)=\begin{cases} x+\frac{\pi}{2} & , \quad x<-\frac{\pi}{2} \\ \cos(x) & , \quad x\in\left[-\frac{\pi}{2},\frac{\pi}{2}\right] \\ \left(\frac{\pi}{2}-x\right)^2 & , \quad x>\frac{\pi}{2} \end{cases}$$

3: Funktionen & Operatoren

Logische Operatoren

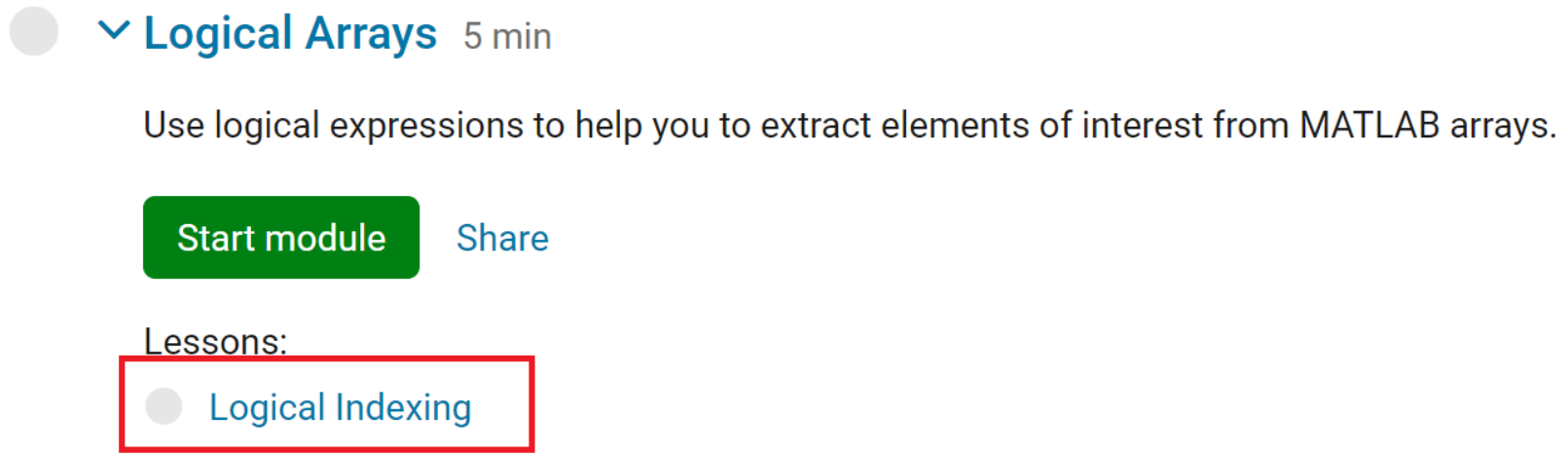
$$f(x)=\begin{cases} x+\frac{\pi}{2} & , \quad x<-\frac{\pi}{2} \\ \cos(x) & , \quad x\in\left[-\frac{\pi}{2},\frac{\pi}{2}\right] \\ \left(\frac{\pi}{2}-x\right)^2 & , \quad x>\frac{\pi}{2} \end{cases}$$




```
>> x=linspace(-3,3,100);
>> f=(x+pi/2).*(x<-pi/2)...
     +cos(x).*(x>=-pi/2 & x<=pi/2)...
     +(-x+pi/2).^2.*(x>pi/2);
>> plot(x,f)
```

3: Funktionen & Operatoren

Online-Tutorial: ca. 10min



3: Funktionen & Operatoren

Inhaltsverzeichnis

3: Funktionen & Operatoren

Polynome

Für das Rechnen mit Polynomen und deren Darstellung gibt es eigene Funktionen.

Ein Polynom

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

wird in MatLab durch

```
>> p = [a_n    a_n-1    a_n-2    …    a_1    a_0];
```

dargestellt. Fehlende Potenzen werden durch 0 deklariert.

3: Funktionen & Operatoren

Polynome

MatLab kann (u. a.) Polynome auswerten, multiplizieren, dividieren und deren Nullstellen berechnen.

Beispiel: $p(x)=x^3-15x-4$

```
>> p=[1 0 -15 -4];
```

Polynome werden mit `polyval(Polynom,Punkte)` ausgewertet

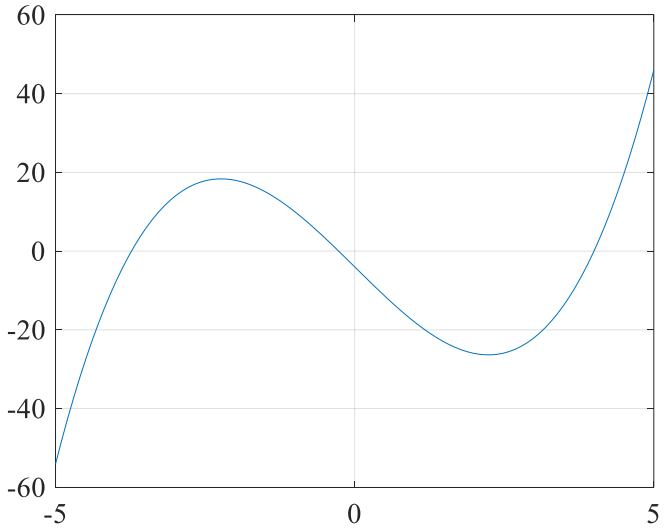
```
>> polyval(p,[-1 0 1])
ans =
    10    -4   -18
```

3: Funktionen & Operatoren

Polynome

Zum Plotten eines Polynoms kann `polyval` ebenfalls verwendet werden.

```
>>x=linspace(-5,5,100);
>>plot(x,polyval(p,x))
```

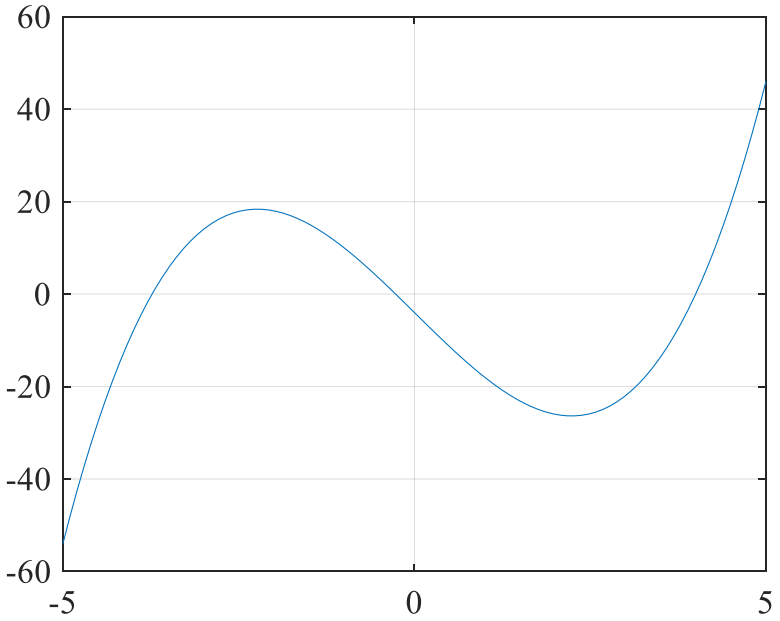


3: Funktionen & Operatoren

Polynome

Nullstellen werden mit `roots(Polynom)` berechnet.

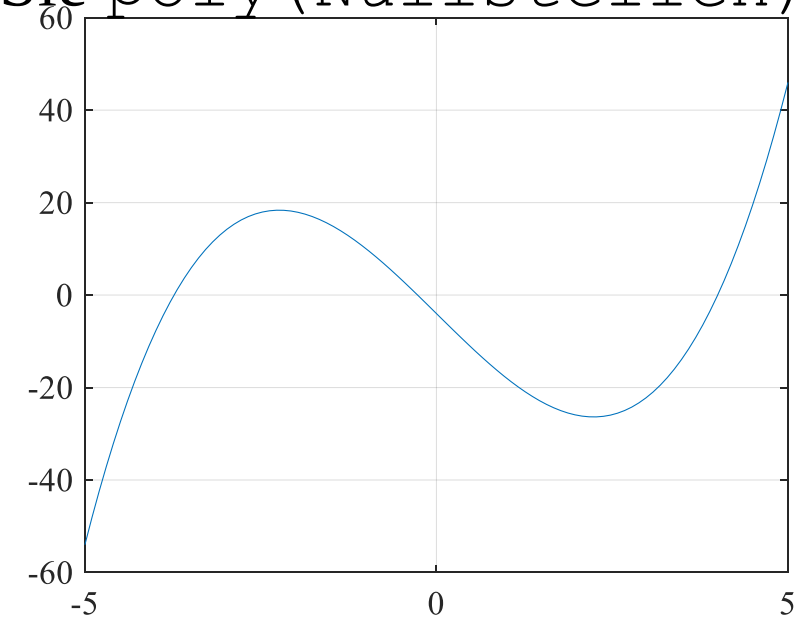
```
>> r=roots(p)
r =
    4.0000
   -3.7321
   -0.2679
```



3: Funktionen & Operatoren

Polynome

Möchten Sie ein Polynom erzeugen, das vorgegebene Nullstellen hat, verwenden Sie `poly(Nullstellen)`




```
>> poly(r)
ans =
   1.0000   0.0000  -15.0000  -4.0000
```

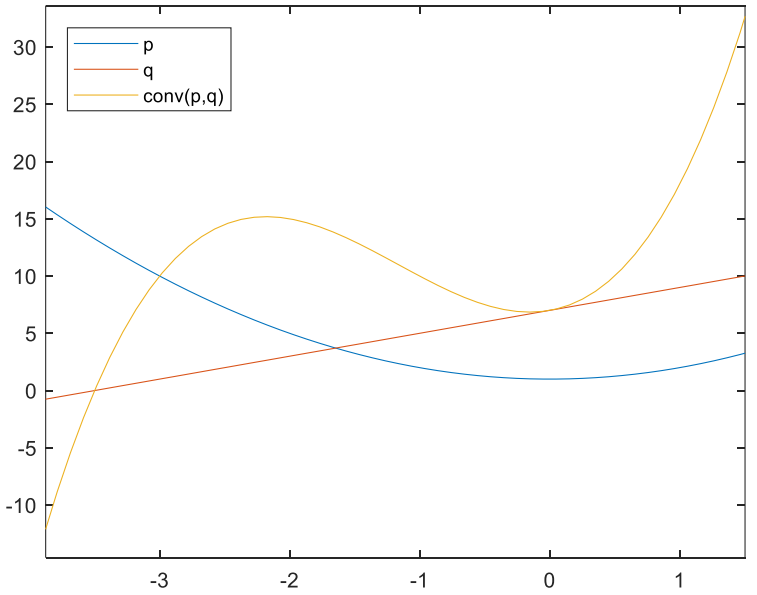
3: Funktionen & Operatoren

Polynome

Multiplikationen werden mit `conv(p,q)` durchgeführt.

Für $p(x) = x^2 + 1$ und $q(x) = 2x + 7$ ergibt die Multiplikation

```
>> p=[1 0 1];
>> q=[2 7];
>> pq=conv(p,q)
pq =
     2     7     2     7
```



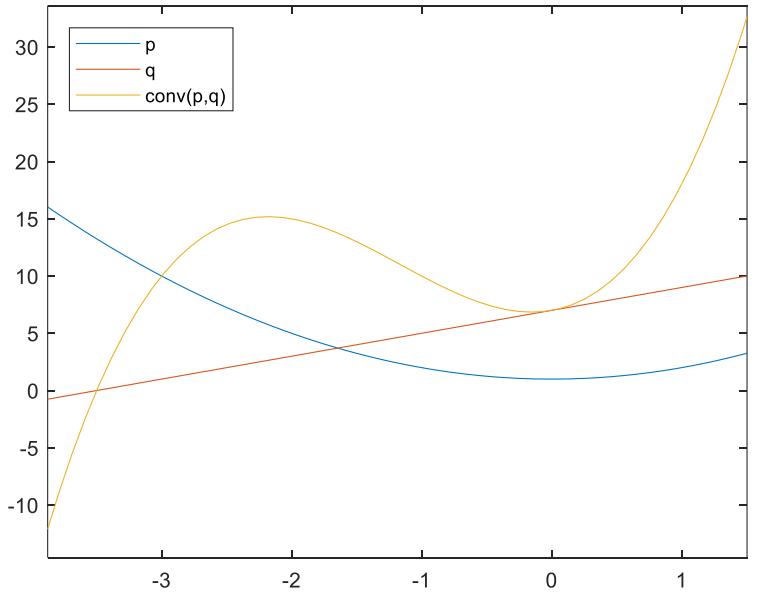
Dies entspricht $p(x)q(x) = 2x^3 + 7x^2 + 2x + 7$.

3: Funktionen & Operatoren

Polynome

Die Polynomdivision wird mit `deconv(p,q)` durchgeführt. Für $p(x)q(x) = 2x^3 + 7x^2 + 2x + 7$ und $q(x) = 2x + 7$ ergibt die Polynomdivision

```
>> p=conv(pq,q)

p =

     1     0     1
```



Dies entspricht dem Polynom $p(x) = x^2 + 1$.

Inhaltsverzeichnis

4: Programmieren in MatLab

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Inhaltsverzeichnis

- Funktionen
- Function handles und anonyme Funktionen
- Entscheidungen und Schleifen
- Dünnbesetzte Matrizen
- Vektorisierung und Zeitmessung

4: Programmieren in MatLab

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Lernziele:

- Sie beherrschen den Umgang mit Funktionen in MatLab.
- Sie können Schleifen und Entscheidungen in MatLab programmieren.
- Sie wissen, wie man eine Zeitmessung in MatLab durchführt um Ihren Code zu beschleunigen.

4: Programmieren in MatLab

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Inhaltsverzeichnis

- Funktionen
- Function handles und anonyme Funktionen
- Entscheidungen und Schleifen
- Dünnbesetzte Matrizen
- Vektorisierung und Zeitmessung

4: Programmieren in MatLab

Funktionen

Funktionen werden ebenso wie gewöhnliche Skripte geschrieben. Dabei muss das erste Wort function sein.

```
function [A,b,C] = Funktionsname(d,E,f)
```

In der ersten Zeile werden die auszugebenden Variablen (hier A, b, C) und die zu übergebenden Variablen (d, E, f) definiert.

4: Programmieren in MatLab

Funktionen

Beispiel: `K4_Rechteck.m`

Das Skript `K4_Rechteck.m` berechnet den Flächeninhalt und die Länge der Diagonale eines Rechtecks. Diese werden anschließend ausgegeben.

Das Skript nutzt die beiden Funktionen
`K4_Flaecheninhalt_Rechteck`
und
`K4_Diagonale_Rechteck`

4: Programmieren in MatLab

Funktionen

Zunächst werden im Skript mit dem Befehl `input()` die beiden Seitenlägen a und b eingelesen.

```
a = input('Bitte a eingeben: ');
b = input('Bitte b eingeben: ,);
```

Dabei wird z. B. „`Bitte a eingeben`“ im Command Window angezeigt. Anschließend muss a über das Command Window eingegeben werden.

4: Programmieren in MatLab

Funktionen

Anschließend ruft das Skript die Funktionen zur Berechnung des Flächeninhalts und der Länge der Diagonalen auf.

```
A = K4_Flaecheninhalt_Rechteck(a,b);
d = K4_Diagonale_Rechteck(a,b);
```

Dieses Ergebnis wird anschließend über den Befehl disp ausgegeben.

4: Programmieren in MatLab

Funktionen

Zu beachten:

- Hauptprogramm und Funktionen müssen sich im selben Ordner befinden, sonst tritt ein Fehler auf.
- Einzelne Teile eines Programms in Funktionen und Skripte auszulagern ist immer dann sinnvoll, wenn das Programm kompliziert ist oder eine Routine häufig benutzt wird.
- Möchten Sie eine Funktion vor deren letzter Zeile abbrechen, können Sie dies mit dem Befehl `return`. In diesem Fall gibt die Funktion den letzten Wert der auszugebenden Variablen aus.

4: Programmieren in MatLab

Funktionen

Vorsicht bei Funktionsnamen.

- Es dürfen keine Sonderzeichen oder Umlaute verwendet werden. Nur der Unterstrich _ ist zulässig.

- Fehler entstehen häufig, wenn Funktionsnamen doppelt vergeben werden. Ob Funktionsnahmen bereits verwendet werden, kann mit dem Befehl `exist` kontrolliert werden.

4: Programmieren in MatLab

Funktionen

Ausgabe	Bedeutung
`0`	Name existiert nicht
`1`	Name ist eine Variable im Workspace
`2`	Name ist ein bestehendes Skript oder eine Datei unbekannten Typs
`3`	Es existiert ein Mex-file mit diesem Namen
`4`	Es existiert ein MDL-file mit diesem Namen
`5`	Name ist an eine MatLab Funktion vergeben (z. B. `sin`)
`6`	Es existiert ein P-file mit diesem Namen
`7`	Es existiert ein Verzeichnis mit diesem Namen

```
>>exist d  >>exist cos  >>exist hello
ans =      ans =       ans =
     1           5                0
```

4: Programmieren in MatLab

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Inhaltsverzeichnis

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Ein function handle ist ein MatLab Dateityp, in dem eine assoziierte Funktion gespeichert wird.

Typische Verwendungszwecke von function handles sind z. B. die Übergabe einer Funktion an eine andere, z. B. an eine Funktion zur numerischen Integration oder Differentiation.

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Die Ableitung einer Funktion in einem festen Punkt x lässt sich über den Differentialquotienten

$$f'(x) \approx D\,f(x) := \frac{f(x+h)-f(x)}{h}, \qquad h>0$$

für eine vorgegebene Schrittweite h berechnen.

4: Programmieren in MatLab

Function handles und anonyme Funktionen

```
x=0:0.1:2*pi;
h=0.001;
Df_1=(sin(x+h)-sin(x))./h;
Df_2=((x+h).^2-x.^2)./h;
Df_3=(cos(x+h)-cos(x))./h;
```

Achten Sie darauf, dass es für viele Funktionen (z. B. die MatLab-Funktion `integral`) nötig ist, dass das function handle Vektoren als Argument x (Punktmenge) verarbeiten können und entsprechend auch Vektoren ausgeben müssen (f(x_i), ... i=1,...,n).

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Die numerische Differentiation kann mit function handles übersichtlicher programmiert werden. Zunächst benötigen wir zwei Funktionen

```
function y=K4_f(x)
y=sin(x);
```

```
function Df=K4_numAbleitung(f,x,h)
Df=(f(x+h)-f(x))./h;
```

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Die Funktion `K4_f` definiert die abzuleitende Funktion.

Die Funktion `K4_numAbleitung` leitet eine beliebige übergebene Funktion f über den Differenzenquotienten ab.

Als letztes benötigen wir das Hauptprogramm.

4: Programmieren in MatLab

Function handles und anonyme Funktionen

```
h=0.02;
x = 0:h:2*pi;
% Berechne f(x) mit der Funktion
K4_f.m:
y = K4_f(x);
% Berechne die Ableitung von f auf dem
Intervall [0,2*pi]
% D.h. Dy ist ein Vektor der gleichen
Länge wie x
Dy = K4_numAbleitung(@K4_f,x,h);
```

4: Programmieren in MatLab

Function handles und anonyme Funktionen

```
Dy = K4_numAbleitung(@K4_f,x,h);
```

Wird eine Funktion (hier `K4_f`) an eine andere Funktion übergeben, schreiben Sie ein @ vor die Funktion. So weiß MatLab, dass die Funktion `K4_f` in der Variablen x ausgewertet werden soll.

Diese Vorgehensweise hat den Vorteil, dass Sie den Differenzenquotienten nicht für jede abzuleitende Funktion neu programmieren müssen wie im ersten Beispiel.

Es reicht die Funktion `K4_f` zu ändern.

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Es ist auch möglich, parametrisierte Funktionen zu nutzen. Beispiel:

$$f_2(x) = x^3 + bx^2 + cx$$

```
function y=K4_f2(x,b,c)
y=x.^3+b.*x+c;
```

Hier muss MatLab wissen, was die Variable, und was die Parameter sind.

4: Programmieren in MatLab

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Function handles und anonyme Funktionen

Das Programm `K4_numAbleitung` muss nicht geändert werden.

Im Hauptprogramm `K4_Abl` müssen folgende Änderungen erfolgen:

```
b=-10; c=-10;
Dy = K4_numAbleitung(@(x)K4_f2(x,b,c),x,h);
```

Dieser Befehl berechnet die Ableitung von `K4_f2` für die Parameter `b=-10` und `c=-10`.

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Es kann unübersichtlich werden, wenn für jede verwendete Funktion ein neues Skript angelegt werden muss.

MatLab erlaubt die Definition sogenannter anonymer Funktionen vom Datentyp function handle.

```
>> f=@(x) sin(x)
f =
  function_handle with value:
    @(x)sin(x)
```

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Das geänderte Hauptprogramm sieht nun wie folgt aus.

```
f=@(x) sin(x);

h=0.02;
x = 0:h:2*pi;
y = K4_f(x);
Dy = K4_numAbleitung(f,x,h);
plot(x,Dy)
```

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Bei der direkten Verwendung anonymer Funktionen müssen die Variablen immer mitgeführt werden.

```
>> f=@(x) cos(x).^2; g=@(x) sin(x).^2;
>> h=@(x) f(x)+g(x);

>> h(0:2:2*pi)
ans =
     1     1     1     1
```

4: Programmieren in MatLab

Function handles und anonyme Funktionen

In diesem Beispiel führt der Befehl

```
>> h=@(x) f(x)+g;
```

zu einer Fehlermeldung bei der Auswertung der Funktion.

```
>> h(1)
Undefined operator '+' for input
arguments of type 'function_handle'.
Error in @(x)f(x)+g
```

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Anonyme Funktionen und function handles können auch auf vektorwertige Funktionen mehrerer Variablen angewandt werden.

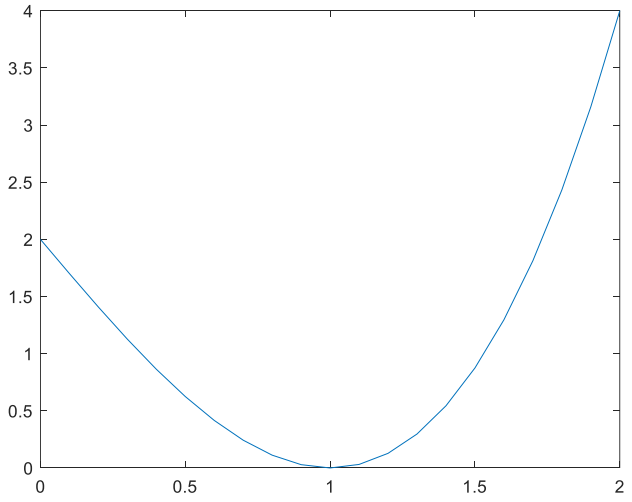
```
>> f=@(x,y) [x+y     x-y
cos(x).^2+sin(x).^2];

>> f(1,2)
ans =
     3    -1     1
```

4: Programmieren in MatLab

Function handles und anonyme Funktionen

Die Parametrisierung anonymer Funktionen kann wie folgt umgesetzt werden.

```
>> b=-3;        c=2;
>> x=0:0.1:2;

>> f=@(x) x.^3+b.*x+c;
>> plot(x,f(x))
```



4: Programmieren in MatLab

Function handles und anonyme Funktionen

Häufig verwendeter Funktionen für function handles.

MatLab Bezeichnung	math. Syntax	Beschreibung
integral(f,a,b)	$\int_a^b f(x)\,dx$	numerische 1D-Integration
integral2(f,a,b,c,d)	$\int_a^b \int_c^d f(x_1, x_2)\,dx_1 dx_2$	numerische 2D-Integration
gradient(F)	$\nabla F(x_1, \ldots x_n)$	Gradient skalarer Funktionen
del2(F)	$\Delta F(x_1, \ldots x_n)$	Laplace-Differentiation skalarer Funktionen
functions		zeigt Informationen über function handles

4: Programmieren in MatLab

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Inhaltsverzeichnis

- Funktionen
- Function handles und anonyme Funktionen
- Entscheidungen und Schleifen
- Dünnbesetzte Matrizen
- Vektorisierung und Zeitmessung

4: Programmieren in MatLab

Entscheidungen und Schleifen

Die (bedingte) Anweisung `if` wertet einen logischen Ausdruck aus und verzweigt zu einer Gruppe von Anweisungen, sofern der Ausdruck wahr ist.

```
if Ausdruck 1
      Anweisungen 1
elseif Ausdruck 2
      Anweisungen 2
else
      Anweisungen 3
end
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

Ist Ausdruck 1 wahr (=1), wird Anweisung 1 ausgeführt. Wenn nicht, wird geprüft ob Ausdruck 2 wahr (=1) ist. Wenn ja, wird Anweisung 2 ausgeführt.

```
if Ausdruck 1
     Anweisungen 1
elseif Ausdruck 2
     Anweisungen 2
else
     Anweisungen 3
 end
```

Ist keiner der Ausdrücke wahr, wird Anweisung 3 ausgeführt.

4: Programmieren in MatLab

Entscheidungen und Schleifen

- Es können beliebig viele `elseif` Befehle benutzt werden.
- Wenn zwei `if / elseif` Abfragen wahr sind, wird nur die Anweisung der obersten Abfrage ausgeführt.
- Auf die `elseif`- und den `else`-Zweig kann verzichtet werden.

4: Programmieren in MatLab

Entscheidungen und Schleifen

Dieses Programm liest n ein und gibt aus, ob n gerade, ungerade oder keine ganze Zahl ist.

```
clear all
n = input('Bitte eine ganze Zahl eingeben: ,');
if mod(n,2)==0
    disp('Die eingegebene Zahl ist gerade!')
elseif mod(n,2)==1
    disp('Die eingegebene Zahl ist ungerade!')
else
    disp('Die eingegebene Zahl ist keine ganze Zahl!')
end
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

```
a = 5;
aux = 0;
if aux == 1
a = a + 2;
end
disp(['a=' num2str(a)])

>> K4_auskommentieren_mit_if
a=5
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

```
a = 5;
aux = 1;
if aux == 1
a = a + 2;
end
disp(['a=' num2str(a)])

>> K4_auskommentieren_mit_if
a=7
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

Wenn verschiedene Anweisungen basierend auf dem Wert einer Variablen ausgeführt werden sollen, kann die Fallunterscheidung `switch` verwendet werden.

Die einzelnen Fälle werden durch `case` definiert.

Wenn die Variable keinen der definierten Fälle erfüllt, wird ähnlich dem `else` der `if`-Abfrage die Anweisung unter dem Schlüsselwort `otherwise` ausgeführt.

Die `switch` Umgebung ist schneller als `if-elseif-else` und sollte, wenn möglich, bevorzugt werden.

4: Programmieren in MatLab

Entscheidungen und Schleifen

```
switch Variable
    case Fall 1
        Anweisungen 1
    case Fall 2
        Anweisungen 2
    ...
    otherwise
        sonstige Anweisungen
end
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

```
clear all
n = input('Bitte eine ganze Zahl eingeben:')
switch mod(n,2)
    case 0
        disp('Die eingegebene Zahl ist gerade!')
    case 1
        disp('Die eingegebene Zahl ist ungerade!')
    otherwise
        disp('Die eingegebene Zahl ist keine ganze Zahl!')
end
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

Für Berechnungen, die wiederholt durchgeführt werden müssen, eignen sich `for`–Schleifen.

`for`-Schleifen sind sehr einfach anzuwenden, dafür aber häufig ineffizient.

```
for Variable = Matrix/Zelle/Array
    Anweisungen
end
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

Der Variablen werden nacheinander die Spalten der Matrix bzw. der Zelle oder dem Array zugewiesen.

```
for Variable = Matrix/Zelle/Array
    Anweisungen
end
```

Für jede Spalte werden die Befehle einmal ausgeführt.

Für einen n-fachen Schleifendurchlauf kann z. B. die Variable `index=1:n` verwendet werden.

4: Programmieren in MatLab

Entscheidungen und Schleifen

Beispiel: Berechne das Hölder-Mittel $\left(\frac{1}{n}\sum_{l=1}^{n} l^k\right)^{\frac{1}{k}}$ der Zahlen l=1,...,n.

```
n = 10;      k = 1;
summe = 0;
for l = 1:n
    summe = summe + l.^k;
end
hoelder = ((1/n).*summe)^(1/k)
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

Beispiel: Berechne das Hölder-Mittel $\left(\frac{1}{n}\sum_{l=1}^{n} l^k\right)^{\frac{1}{k}}$ der Zahlen l=1,...,n.

```
n = 10;      k = 1;
summe = 0; % summe vor Schleife initialisiert
for l = 1:n
    summe = summe + l.^k;
end
hoelder = ((1/n).*summe)^(1/k)
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

while-Schleifen führen ebenfalls eine Befehlssequenz mehrfach aus, solange der logische Ausdruck in der Definition wahr (=1) ist.

```
while logischer Ausdruck
    Anweisungen
end
```

Beispiel: Division mit Rest

4: Programmieren in MatLab

Entscheidungen und Schleifen

Für x,y,q,r ∈ ℕ mit r<y, berechne q,r, sodass x=qy+r.

```
x = input('Bitte eine Zahl x eingeben, die
    groesser als 0 ist: ,');
y = input('Bitte eine Zahl y eingeben, die
    groesser als 0 und kleiner als x ist: ');
disp(' ')
% Initialisierungen:
q = 0;
% Speichere den eingegebenen Wert von x fuer
die Ausgabe
x_old=x;
```

4: Programmieren in MatLab

Entscheidungen und Schleifen

Für x,y,q,r ∈ ℕ mit r<y, berechne q,r, sodass x=qy+r.

```
if x<0 || y<0
    error('x oder y kleiner 0!')
elseif x<y
    error('y ist nicht kleiner als x!')
elseif y == 0
    error('Division durch 0!')
end
```

`error('Text')` bricht das Programm ab und gibt die Fehlermeldung `'Text'` im Command Window aus.

4: Programmieren in MatLab

Entscheidungen und Schleifen

Für x,y,q,r ∈ ℕ mit r<y, berechne q,r, sodass x=qy+r.

```
while x>=y
    x = x - y;
    q = q+1;
end
r = x;
```

Die Befehle innerhalb der `while`-Schleife werden ausgeführt so lange `x>=y` wahr ist.

Achtung vor Endlosschleifen (`x>=y` bleibt immer wahr)!

4: Programmieren in MatLab

Entscheidungen und Schleifen

Für x,y,q,r ∈ ℕ mit r<y, berechne q,r, sodass x=qy+r.

```
disp(['q = ' num2str(q) ' und r = ' num2str(r)])
disp([num2str(x_old) '=' num2str(q) '*'
      num2str(y) '+' num2str(r)])
```

gibt das Ergebnis aus.

4: Programmieren in MatLab

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Inhaltsverzeichnis

- Funktionen
- Function handles und anonyme Funktionen
- Entscheidungen und Schleifen
- Dünnbesetzte Matrizen
- Vektorisierung und Zeitmessung

4: Programmieren in MatLab

Dünnbesetzte Matrizen

Besitzt eine Matrix viele Nullen, ist es effizienter, sie im Sparse-Format zu speichern um Arbeitsspeicher zu sparen.

In dieser Darstellung werden nur von 0 verschiedene Elemente gespeichert.

Alle Matrixoperationen lassen sich auch auf Sparse-Matrizen anwenden.

Matrixoperationen ergeben eine Sparse-Matrix, wenn alle Argumente der Operation Sparse-Matrizen sind.

4: Programmieren in MatLab

Dünnbesetzte Matrizen

Beipiel: 100 x 100 Matrix mit 2 auf der Diagonale und -1 auf der ersten oberen und unteren Nebendiagonale

Von 10.000 Elementen sind nur 298 ungleich 0 (ca. 3%).

```
M_full=diag(2*ones(100,1))+...
  diag(-ones(99,1),-1)+...
  diag(-ones(99,1),1);
```

4: Programmieren in MatLab

Dünnbesetzte Matrizen

n x m-Sparse-Matrizen mit M von Null verschiedenen Einträgen werden mit dem Befehl `spalloc(n,m,M)` initialisiert.

```
>> e = ones(100,1);
>> M_sparse = spdiags([-e 2*e -e],...
      -1:1, 100, 100);
```

spdiags erzeugt eine 100x100 Matrix, auf deren (Neben-) Diagonalen -1,0,1 die Spalten der Matrix `[-e 2*e -e]` gesetzt werden.

4: Programmieren in MatLab

Dünnbesetzte Matrizen

Wird eine Sparse-Matrix ausgegeben, so ist diese wie folgt zu verstehen:

```
>> M_sparse(1,:)
ans =
   (1,1)         2
   (1,2)        -1
```

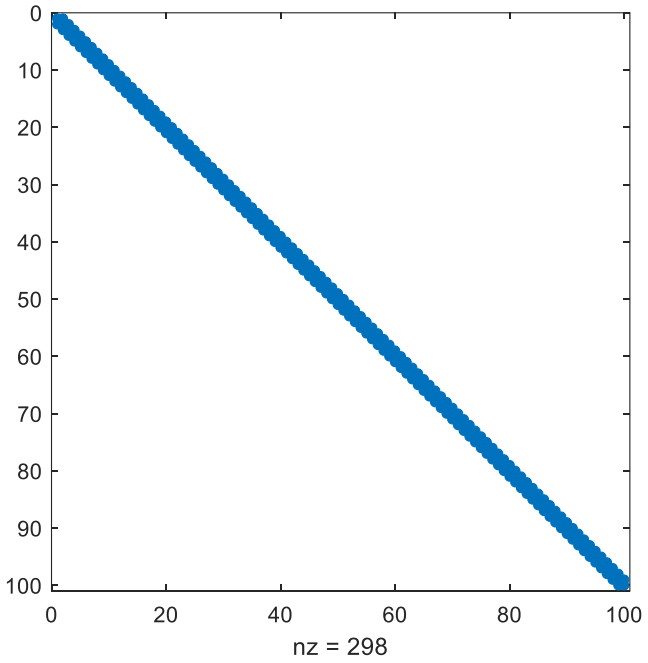
Das Element `(1,1)` hat den Wert 2, das Element `(1,2)` den Wert -1.

4: Programmieren in MatLab

Dünnbesetzte Matrizen

Mit dem Befehl `spy(M_sparse)` wird eine Grafik ausgegeben die zeigt, welche Elemente der Matrix `M_sparse` ungleich Null sind.

```
>> spy(M_sparse)
```



4: Programmieren in MatLab

Dünnbesetzte Matrizen

Vergleich Full-Matrix und Sparse-Matrix

```
>> whos M_full M_sparse
Name       Size     Bytes Class  Attributes
M_full     100x100 80000 double
M_sparse 100x100 5576  double sparse
```

`M_sparse` benötigt nur ca. 7% des Arbeitsspeichers von `M_full`.

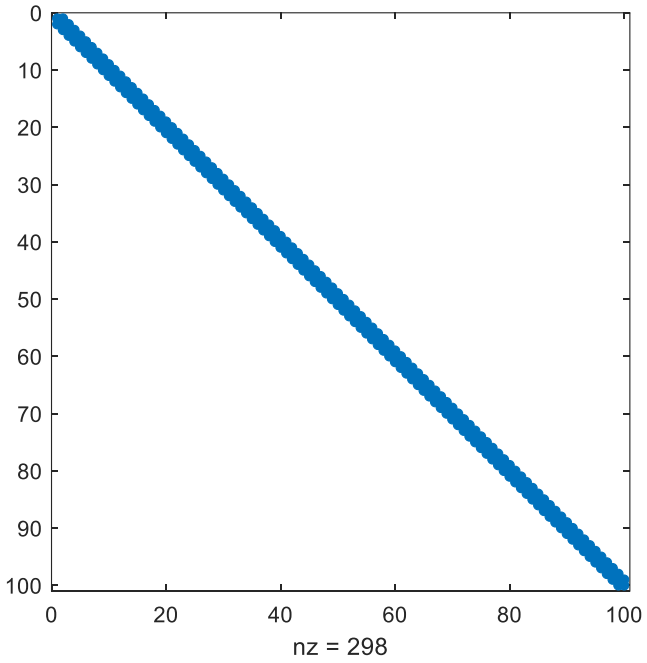
4: Programmieren in MatLab

Dünnbesetzte Matrizen

Mit dem Befehl `spy(M_sparse)` wird eine Grafik ausgegeben die zeigt, welche Elemente der Matrix `M_sparse` ungleich Null sind.

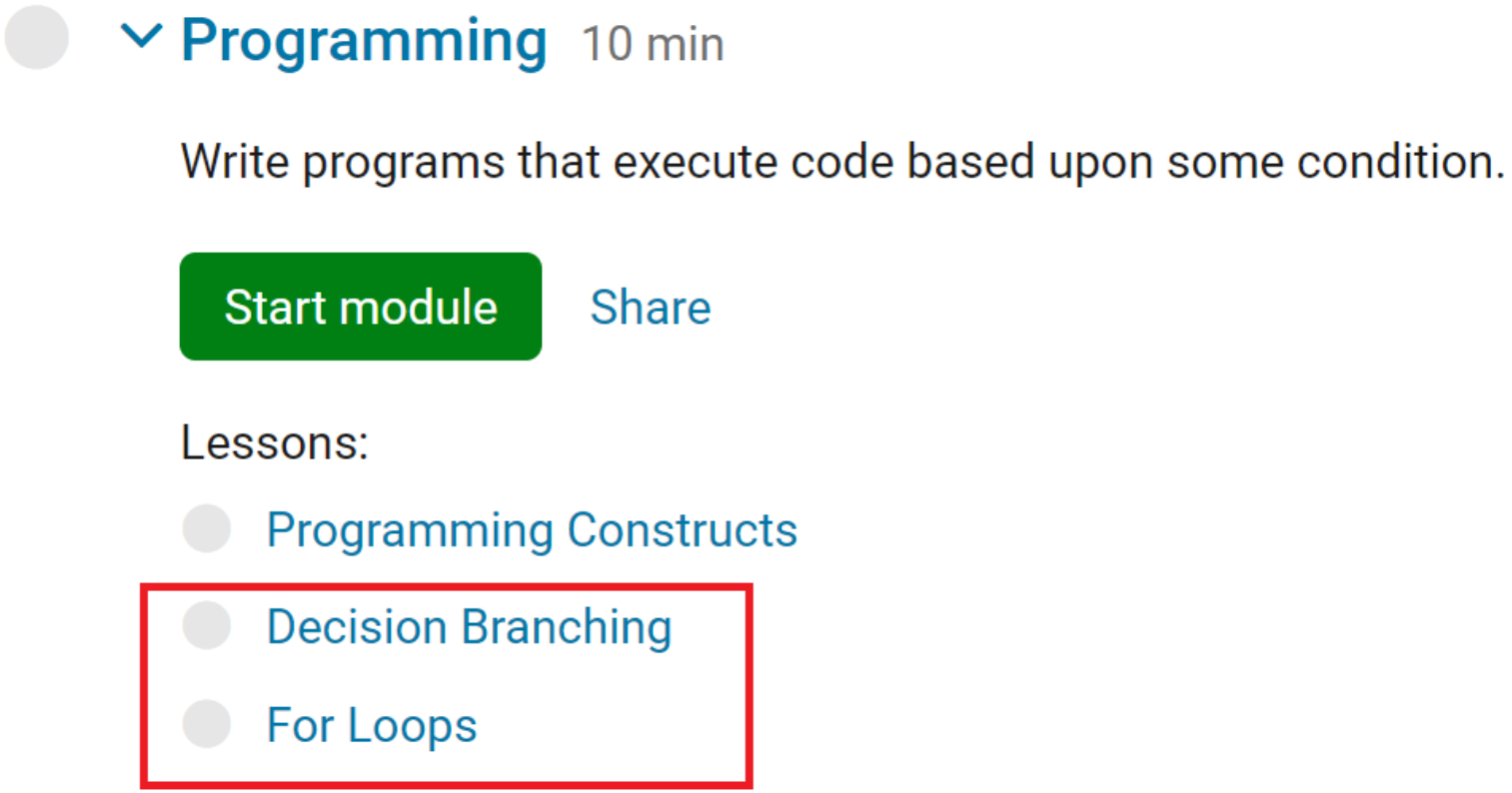
```
>> spy(M_sparse)
```

`speye(100)` erzeugt eine dünnbesetzte 100x100 Diagonalmatrix.



2: Lineare Algebra

Online-Tutorial: ca. 10min



4: Programmieren in MatLab

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Inhaltsverzeichnis

- Funktionen
- Function handles und anonyme Funktionen
- Entscheidungen und Schleifen
- Dünnbesetzte Matrizen
- Vektorisierung und Zeitmessung

4: Programmieren in MatLab

Vektorisierung und Zeitmessung

Die Verwendung von Schleifen ist meist ineffizient (langsam) und sollte wenn möglich durch einen vektorisierten Code ersetzt werden.

MatLab liefert zwei Möglichkeiten zur Zeitmessung, um den Code zu optimieren.

Die Befehle `tic` und `toc` arbeiten wie eine Stoppuhr.

Der Button 

liefert ein detailiertes Laufzeitprofil.

4: Programmieren in MatLab

Vektorisierung und Zeitmessung

`tic` startet einen allgemeinen Timer, auf den sich die Zeitmessung `toc` bei jeder Ausführung bezieht.

Wird `tic` erneut ausgeführt, wird der Timer auf Null gesetzt und neu gestartet.

Sollen verschiedene Startpunkte für die Zeitmessung im selben Programm verwendet werden, wird der Befehl `Startpunkt_Name=tic` verwendet.

4: Programmieren in MatLab

Vektorisierung und Zeitmessung

`toc` gibt die Zeit in Sekunden seit dem (letzten) Starten der Stopuhr mit `tic` aus.

Soll auf einen bestimmten Timer zugegriffen werden, geschieht dies mit dem Befehl
`toc(Startpunkt_Name)`

Mit `Zeit=toc(Startpunkt_Name)` wird die vergangene Zeit zum in der Variablen `Zeit` gespeicherten Zeitpunkt gestoppt.

4: Programmieren in MatLab

Vektorisierung und Zeitmessung

Beispiel: Berechnung von `sin(x)'`

```
% Clear Workspace
clear all
% Definition von f
f = @(x) sin(x);
% Def. Werte x und Auswertung von f
h=0.00002;
x=0:h:4;
y = f(x);
```

4: Programmieren in MatLab

Vektorisierung und Zeitmessung

Berechnung der Anzahl der Einträge des Vektors, der die Ableitung `f'(x)` enthält.

```
n_help = 4/h;


% n_help ist vom Typ double, daher
muss auf ganze Zahlen gerundet werden.


n = round(n_help);
```

4: Programmieren in MatLab

Vektorisierung und Zeitmessung

```
tic;
Dffor = zeros(n,1);
% Berechne die numerische Ableitung in
% einer for-Schleife.
for i = 1:n
    Dffor(i) = (y(i + 1) - y(i))/h;
end
toc;
t1=toc;
```

4: Programmieren in MatLab

Vektorisierung und Zeitmessung

```
% Starte neuen Timer
tstart = tic;
% Berechne die numerische Ableitung
% vektorisiert.
Dfvect = (y(2:n+1)-y(1:n))/h;
toc(tstart);
t2=toc(tstart);
```

Ein Vergleich von `t1` und `t2` zeigt, dass der vektorisierte Code schneller läuft.

4: Programmieren in MatLab

Vektorisierung und Zeitmessung

- Das Programmieren über `for`-Schleifen ist im Allgemeinen leichter und weniger Fehleranfällig.

- EinVektorisierter Code ist meist kompakter und schneller.

- Für Programmieranfänger empfiehlt es sich, zunächst ein lauffähiges Programm zu schreiben, dass die richtigen Ergebnisse liefert. Eine Vektorisierung ist dann in einem zweiten Schritt möglich.

Inhaltsverzeichnis

5: Graphische Ausgaben

Inhaltsverzeichnis

- 2D Plots
- Grafikexport
- Mehrere Plots in einer Figure
- 3D Grafiken
- MatLab-Movies

5: Graphische Ausgaben

Lernziele:

- Sie können 2D- und 3D-Plots in MatLab erzeugen, bearbeiten und speichern.

5: Graphische Ausgaben

Inhaltsverzeichnis

- 2D Plots
- Grafikexport
- Mehrere Plots in einer Figure
- 3D Grafiken
- MatLab-Movies

5: Graphische Ausgaben

2D Plots

Grafiken werden mit dem Befehl `figure` initialisiert.

Mit `figure(Name,Value)` kann die Grafik modifiziert werden.

Name-Value Paare sind z. B.

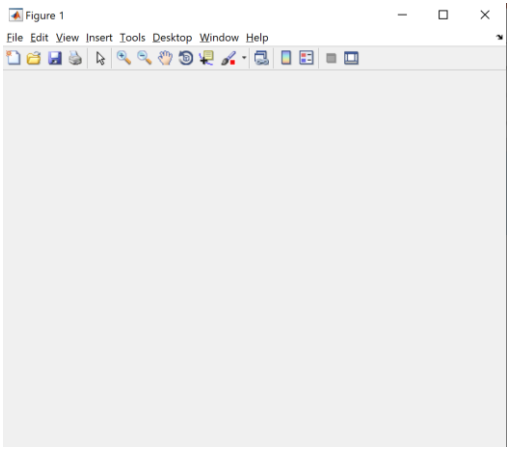
Name	Value	Beschreibung
‘Color‘	RGB-Triplet [r g b] Kurzbezeichnung (z. B. ‘r‘)	Ändert die Hintergrundfarbe der Figure. r,g,b sind Werte im Intervall [0 1].
‘Position‘	[left bottom width height]	Position und Größe der Figure

weitere Paare finden sich in der MatLab Hilfe

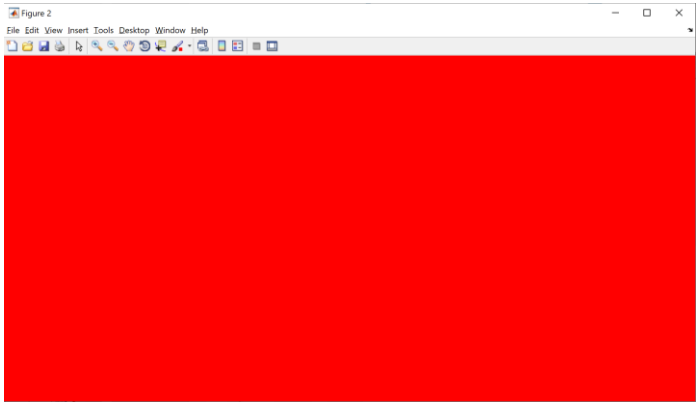
5: Graphische Ausgaben

2D Plots

```
>> figure
```




```
>> figure('Position',[100 200 1000...
500],'Color',[1 0 0])
```

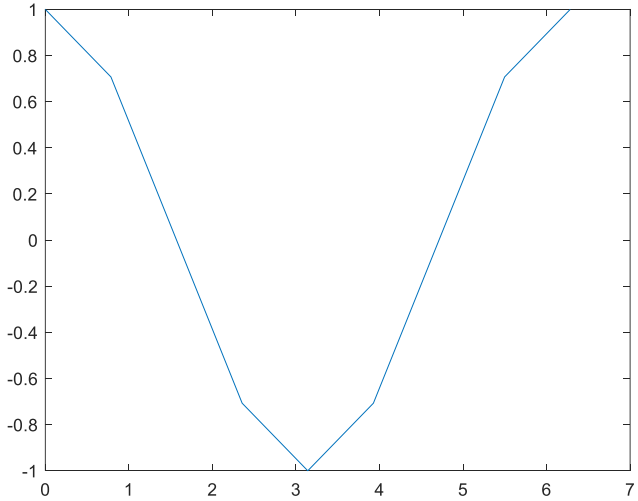


5: Graphische Ausgaben

2D Plots

Der Befehl `plot(x,y)` zeichnet die Werte im Vektor x gegen die Werte im Vektor y ab. Punkte werden dabei mit einer Geraden verbunden.

```
>> x=linspace(0,2*pi,9);
>> plot(x,cos(x))
```

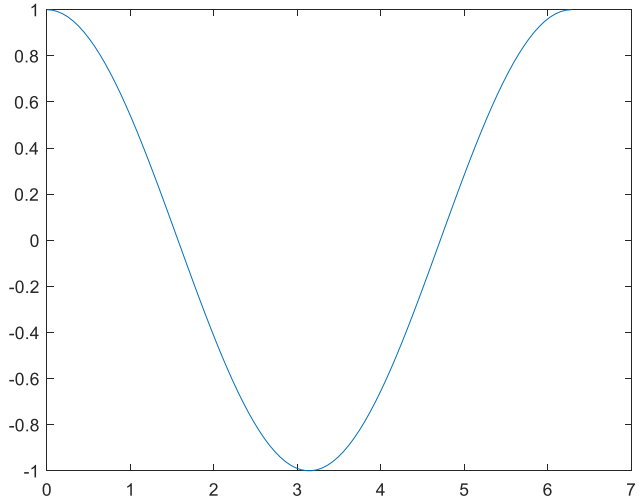


5: Graphische Ausgaben

2D Plots

Der Befehl `plot(x,y)` zeichnet die Werte im Vektor x gegen die Werte im Vektor y ab. Punkte werden dabei mit einer Geraden verbunden.

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



5: Graphische Ausgaben

2D Plots

Figures können mit folgenden Befehlen direkt im Command-Window oder Skript bearbeitet werden.

MatLab-Befehl	Beschreibung
axis([xmin xmax ymin ymax])	Setzt den x- und y-Achsenabschnitt auf die angegebenen Werte.
axis tight	automatische Anpassung der Achsen auf die Daten
axis equal	Gleiche Wahl der Skalierung aller Achsen
grid on	Gitter anzeigen
grid off	Gitter nicht anzeigen (Standard)
xlabel(‘x_Beschriftung‘)	x-Achsen Beschriftung
ylabel(‘y_Beschriftung‘)	y-Achsen Beschriftung
zlabel(‘z_Beschriftung‘)	z-Achsen Beschriftung

5: Graphische Ausgaben

2D Plots

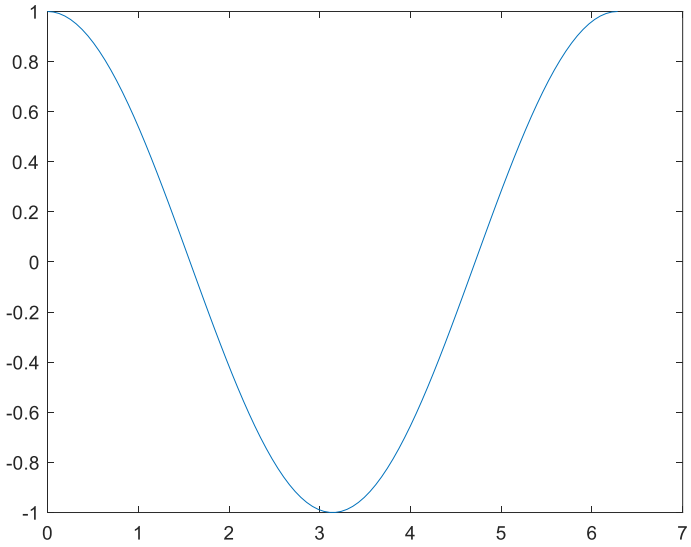
Figures können mit folgenden Befehlen direkt im Command-Window oder Skript bearbeitet werden.

MatLab-Befehl	Beschreibung
titel(‘Titel der Figure‘)	Überschrift der Figure
legend(‘Legende des Plots‘)	Legende des Plots
text(x,y,‘Text‘)	Beschriftung am Punkt (x,y) in der Grafik
set(gca,‘XTick‘,‘Vektor‘)	Setzt die zu Beschrifteten x-Achsenpunkte
set(gca,‘XtickLabel‘,{‘P1‘,‘P2‘…})	Setzt die Beschriftung an den x-Achsenpunkte
colormap(Farbskala)	Auswahl der Farbskala
caxis([cmin cmax])	Setzt Farbskala auf das Intervall [cmin cmax]
caxis auto	automatisches Setzen der Farbskala

5: Graphische Ausgaben

2D Plots

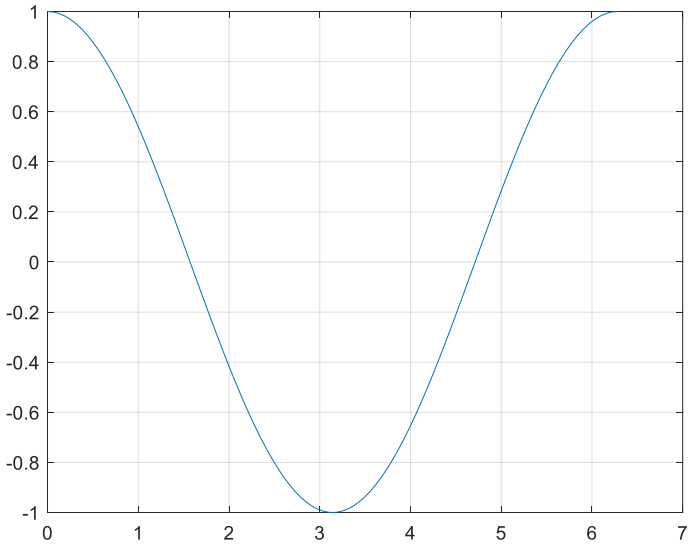
```
>> x=linspace(0,2*pi,100);plot(x,cos(x))
```



5: Graphische Ausgaben

2D Plots

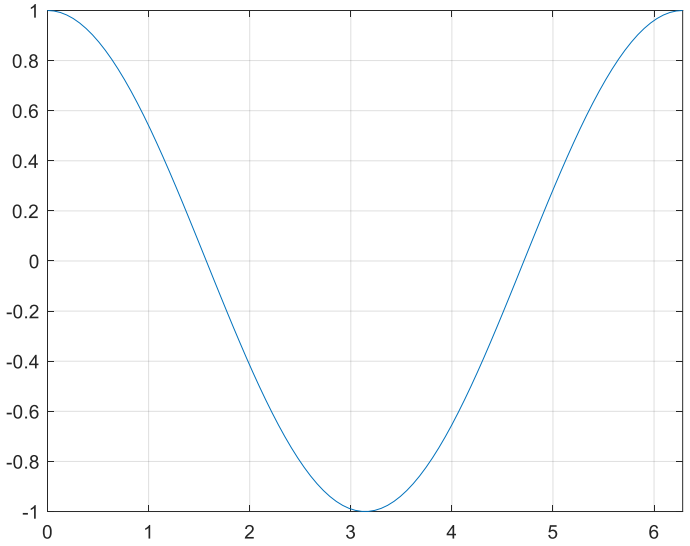
```
>> x=linspace(0,2*pi,100);plot(x,cos(x))
>> grid on
```



5: Graphische Ausgaben

2D Plots

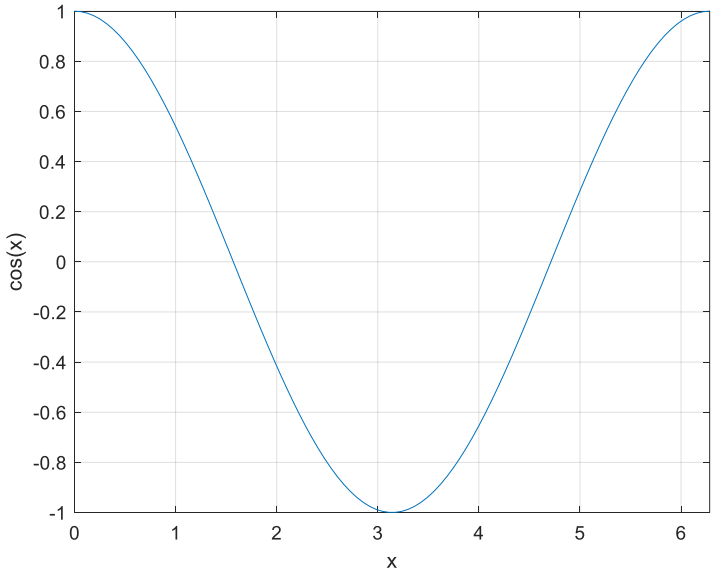
```
>> x=linspace(0,2*pi,100);plot(x,cos(x))
>> grid on
>> axis tight
```



5: Graphische Ausgaben

2D Plots

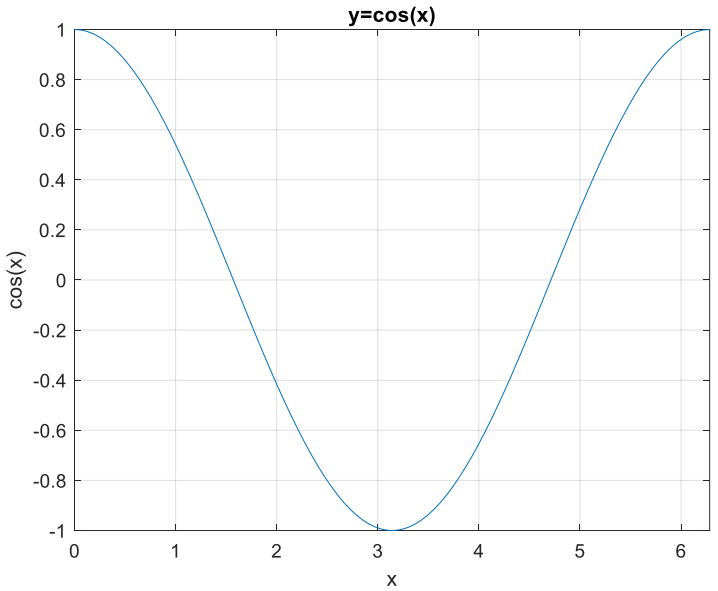
```
>> x=linspace(0,2*pi,100);plot(x,cos(x))
>> grid on
>> axis tight
>> xlabel('x'),ylabel('cos(x)')
```



5: Graphische Ausgaben

2D Plots

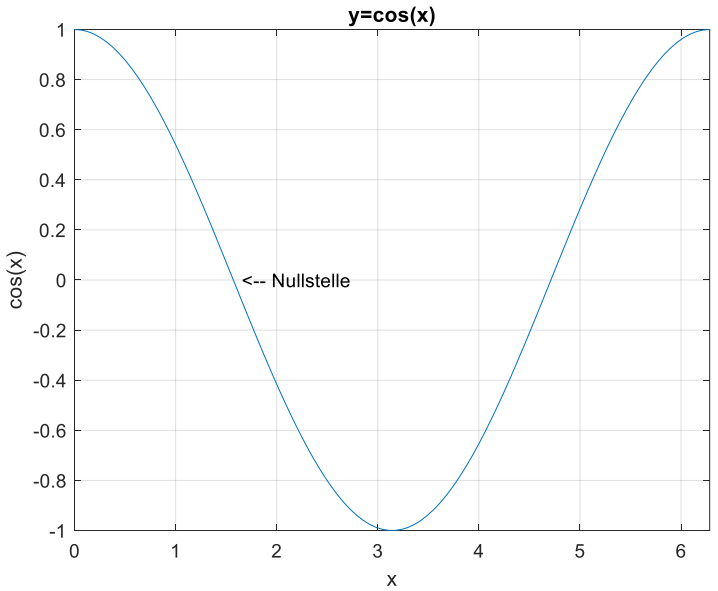
```
>> x=linspace(0,2*pi,100);plot(x,cos(x))
>> grid on
>> axis tight
>> xlabel('x'),ylabel('cos(x)')
>> title('y=cos(x)')
```



5: Graphische Ausgaben

2D Plots

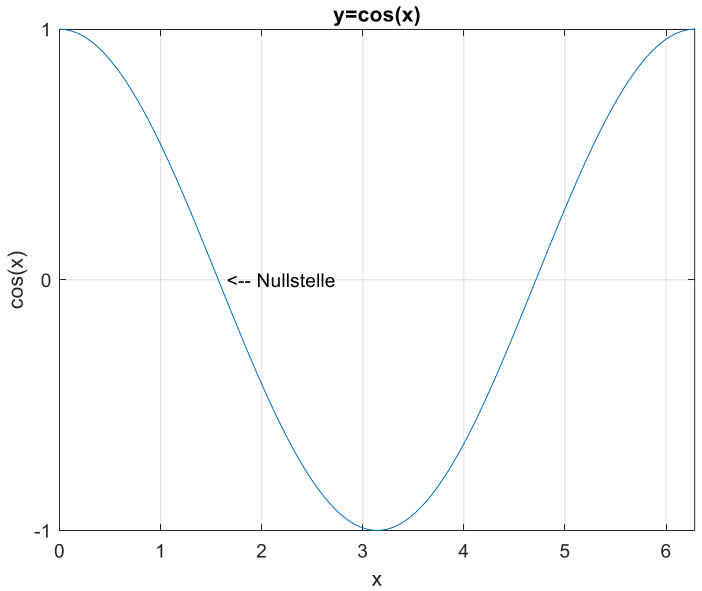
```
>> x=linspace(0,2*pi,100);plot(x,cos(x))
>> grid on
>> axis tight
>> xlabel('x'),ylabel('cos(x)')
>> title('y=cos(x)')
>> text(1.6,0,' <--Nullstelle‘)
```



5: Graphische Ausgaben

2D Plots

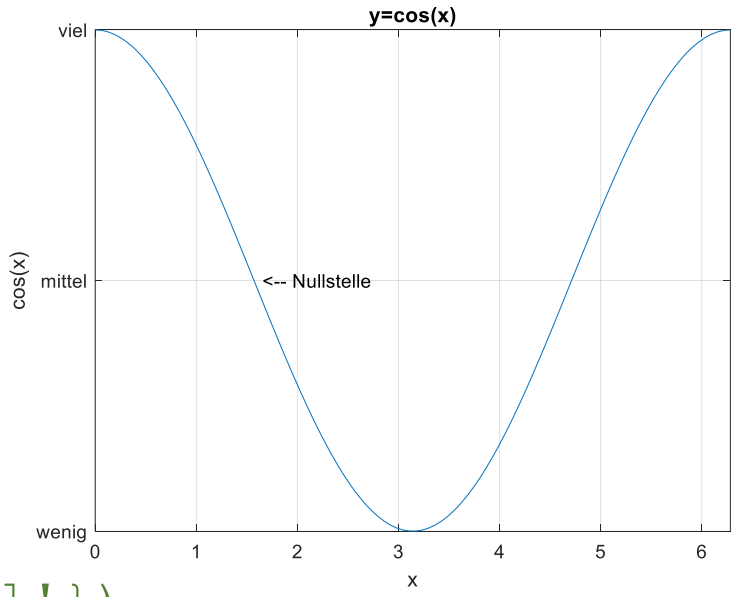
```
>> x=linspace(0,2*pi,100);plot(x,cos(x))
>> grid on
>> axis tight
>> xlabel('x'),ylabel('cos(x)')
>> title('y=cos(x)')
>> text(1.6,0,' <--Nullstelle‘)
>> set(gca,'YTick',[-1 0 1])
```



5: Graphische Ausgaben

2D Plots

```
>> x=linspace(0,2*pi,100);plot(x,cos(x))
>> grid on
>> axis tight
>> xlabel('x'),ylabel('cos(x)')
>> title('y=cos(x)')
>> text(1.6,0,' <--Nullstelle')
>> set(gca,'YTick',[-1 0 1])
>> set(gca,'YTickLabel',…
        {'wenig','mittel','viel'})
```



5: Graphische Ausgaben

2D Plots

- `gca` steht für get current axis.
- Ähnlich zu `XTick` existieren `YTick` und `ZTick`.
- Ähnlich zu `XTickLabel` existieren die Befehle `YTickLabel` und `ZTickLabel`.
- Bei `XTickLabel` müssen so viele Punktbeschriftungen angegeben werden wie `XTicks` existieren.
- Text kann in LaTeX eingegeben und interpretiert werden.

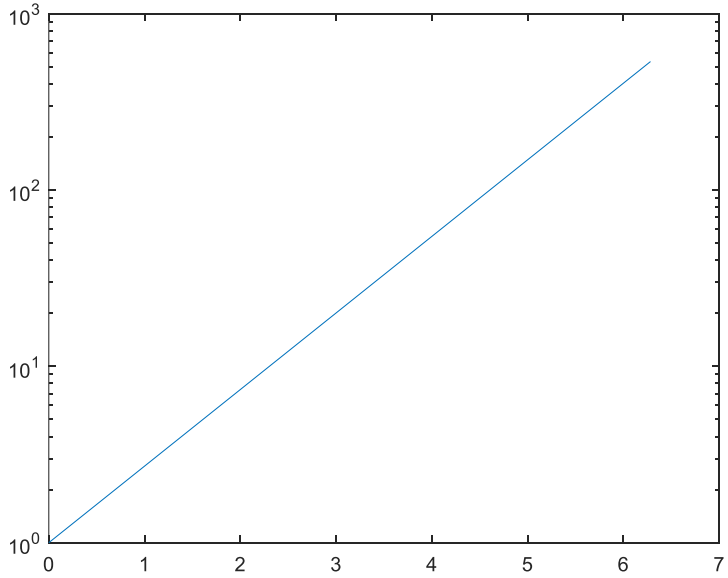
5: Graphische Ausgaben

2D Plots

Logarithmische Skalen können mit den Befehlen `semilogx`, `semilogy` oder `loglog` geplottet werden.

Dabei sind die x-, die y-Achse oder beide Achsen logarithmisch skaliert.

```
>> semilogy(x,exp(x))
```



5: Graphische Ausgaben

Inhaltsverzeichnis

- 2D Plots
- Grafikexport
- Mehrere Plots in einer Figure
- 3D Grafiken
- MatLab-Movies

5: Graphische Ausgaben

Grafikexport

Grafiken können über das Menü *File* → *Save as* gespeichert werden.

Das MatLab Standard Format ist `.fig`. Dieses hat den Vorteil, dass Grafiken vollständig neu geladen werden können. Dies erlaubt eine nachträgliche Bearbeitung, sollten kleinere Fehler (z. B. in der Beschriftung) später auffallen.

Figures können mit `print` automatisch gespeichert werden.

5: Graphische Ausgaben

Grafikexport

Die Syntax lautet `print('Dateiname','Format')`

Befehl	Beschreibung
print –depsc	Export als farbige eps-Datei
print –deps	Export als schwarz/weiß eps-Datei
print –djpeg	Export als jpg-Datei
print –dtiff	Export als tiff-Datei
print –depsc -r600	Export als farbige eps-Datei im einer Auflösung von 600dpi
print –dpng	Export als png-Datei
print –dpdf	Export als pdf-Datei
print –dsvg	Export als svg-Datei

```
>> print('semilogy','-djpeg')
```

5: Graphische Ausgaben

Inhaltsverzeichnis

5: Graphische Ausgaben

Mehrere Plots in einer Figure

Möchten Sie eine Figure in mehrere separate Bilder aufteilen, können Sie den Befehl `subplot(n,m,l)` nutzen.

`subplot(n,m,l)` teilt die Figure in `n` Zeilen und `m` Spalten auf. Der Parameter `l=1,...nm` bezeichnet die Bildnummer.

`l` zählt die Bilder zeilenweise durch, d.h. für `n=m=2` bezeichnet `l=2` das Bild in der ersten Zeile und zweiten Spalte.

5: Graphische Ausgaben

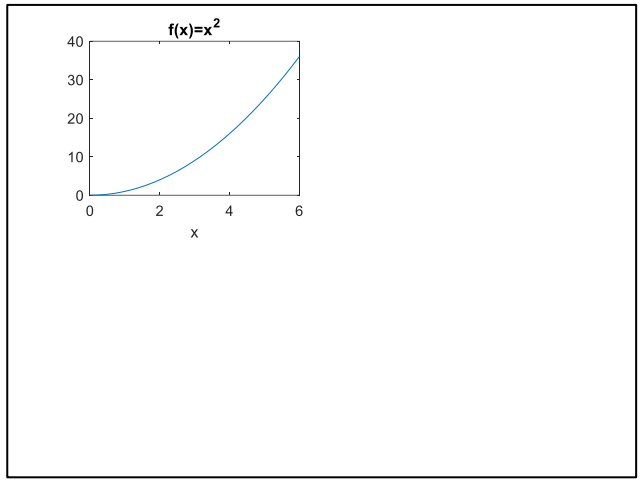
Mehrere Plots in einer Figure

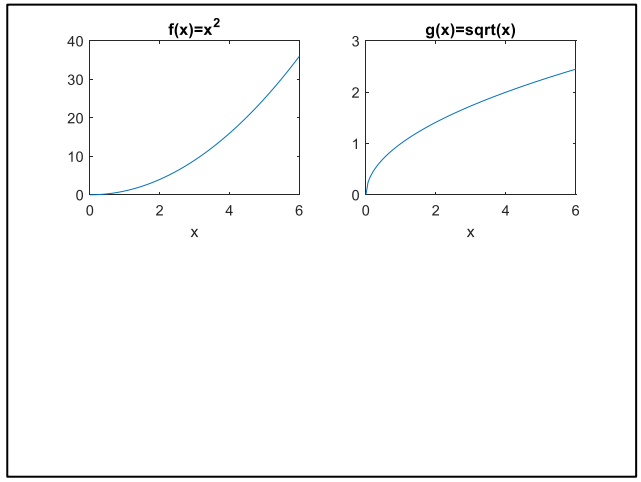
```
x = linspace(0,6,100);


f = x.^2;
g = sqrt(x);
h = x.^3;
j = 3.*x-5;

figure(1)
```

5: Graphische Ausgaben

Mehrere Plots in einer Figure

```
subplot(2,2,1)
plot(x,f)
title('f(x)=x^2')
xlabel('x')

subplot(2,2,2)
plot(x,g)
title('g(x)=sqrt(x)')
xlabel('x')
```

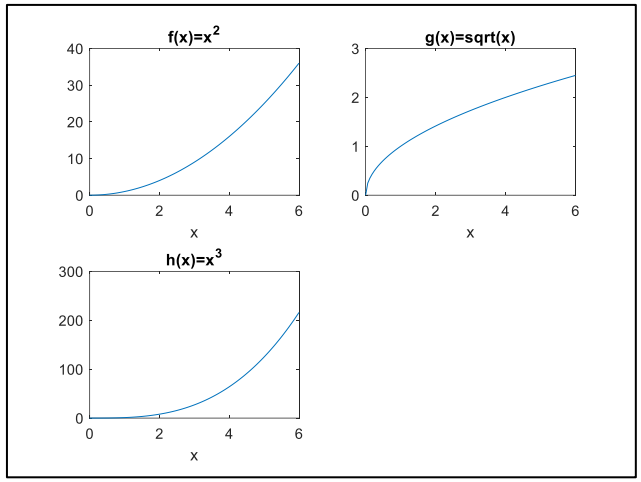


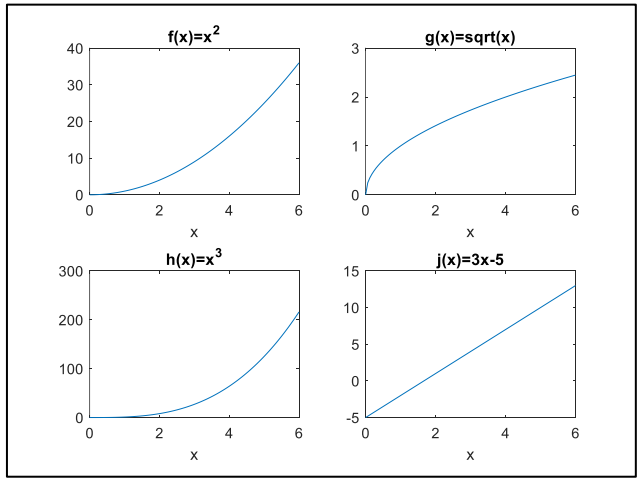


5: Graphische Ausgaben

Mehrere Plots in einer Figure

```
subplot(2,2,3)
plot(x,h)
title('h(x)=x^3')
xlabel('x')

subplot(2,2,4)
plot(x,j)
title('j(x)=3x-5')
xlabel('x')
```





5: Graphische Ausgaben

Mehrere Plots in einer Figure

Mehrere Plots in der selben Figure können mit dem Befehl `hold` erzeugt werden.

`hold on` verhindert, dass die in der Figure existierenden Plots nicht überschrieben werden.

Dies gilt für alle Plots, bis das Zusammenfügen mit `hold off` beendet wird.

5: Graphische Ausgaben

Mehrere Plots in einer Figure

```
x = linspace(0,2,20);
g = sqrt(x);
j = x;

figure(1)
plot(x,g,'Color','k','Linestyle','--')
hold on
plot(x,j,'Color','r','Marker',`d`,...
     'MarkerSize',8)
```

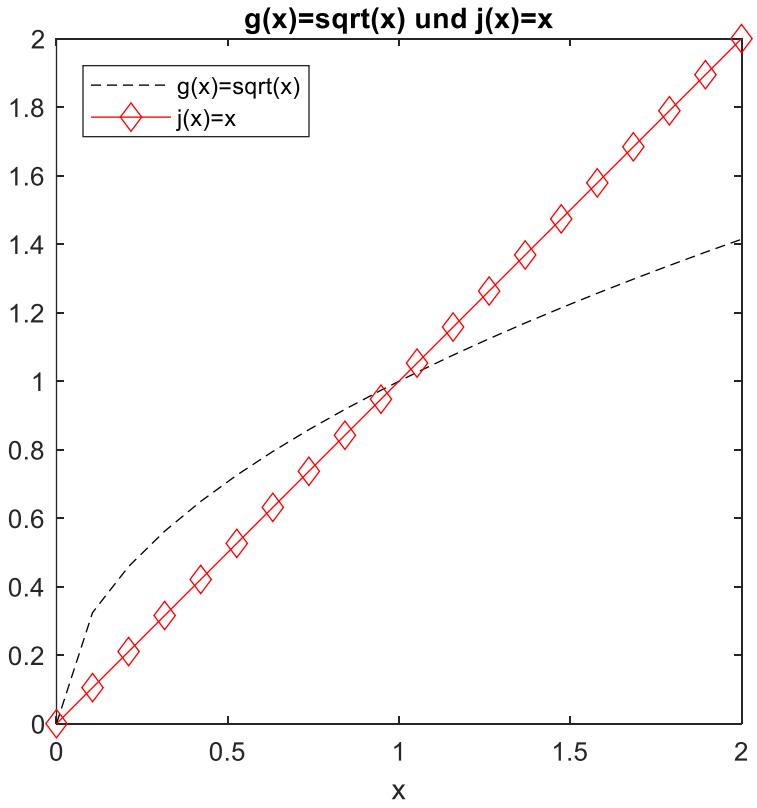
5: Graphische Ausgaben

Mehrere Plots in einer Figure

```
hold off

legend('g(x)=sqrt(x)','j(x)=x',...
    'Location','Northwest')

xlabel('x')

axis square
title('g(x)=sqrt(x) und j(x)=x')
```

5: Graphische Ausgaben

Mehrere Plots in einer Figure



5: Graphische Ausgaben

Mehrere Plots in einer Figure

Zum Befehl `hold` wurden in den Plots die Attribute

- `'Color'` – ändert die Linienfarbe
- `'Linestyle'` – ändert den Linientyp
- `'Marker'` – ändert den Marker
- `'MarkerSize'` – ändert die Markergröße auf den Angegenen Wert

und in der Legende das Attribut

- `'Location'` – ändert die Position der Legende

angepasst.

5: Graphische Ausgaben

Mehrere Plots in einer Figure

Folgende Farben können mit dem Name-Value-Paar `plot(x,y,'Color',Befehl)` verwendet werden.

Befehl	Farbe
`'y'`	gelb
`'m'`	magenta
`'c'`	cyan
`'r'`	rot
`'g'`	grün
`'b'`	blau
`'w'`	weiß
`'y'`	schwarz
`[r g b]`	RGB-Triplet mit r,g,b aus dem Intervall [0,1]

5: Graphische Ausgaben

Mehrere Plots in einer Figure

Folgende Marker können mit dem Name-Value-Paar `plot(x,y,'Marker',Befehl)` verwendet werden.

Befehl	Marker
`'+'`	Pluszeichen
`'*'`	Stern
`'x'`	Kreuz
`'s'`	Quadrat
`'d'`	Diamand
`'p'`	Pentagram
`'v'`	Dreieck unten
`'<'`	Dreieck links

5: Graphische Ausgaben

Mehrere Plots in einer Figure

Folgende Linienarten können mit dem Name-Value-Paar `plot(x,y,'LineStyle',Befehl)` verwendet werden.

Befehl	Linienstil
`'-'`	durchgezogene Linie
`'--'`	gestrichelte Linie
`':'`	gepunktete Linie
`'-.'`	Strich-Punkt-Linie

5: Graphische Ausgaben

Mehrere Plots in einer Figure

Folgende Positionen können mit dem Name-Value-Paar `Legend('label1','label2,...,'Location',Befehl)` verwendet werden.

Befehl	Position
‘north‘	Innerhalb der Figure oben mitte (alternativ ‘south‘, ‘east‘, ‘west‘)
‘southeast‘	Innerhalb der Figure unten rechts (alternativ ‘southwest‘, etc.)
‘northoutside‘	Außerhalb der Figure oben mitte (alternativ ‘southoutside ‘, ‘eastoutside ‘, ‘westoutside ‘)
‘southeastoutside‘	Innerhalb der Figure unten rechts (alternativ ‘southwestoutside‘, etc.)
‘best‘	Position mit wenigstem Konflikt mit den Plots
‘bestoutside‘	Rechts der Achsen

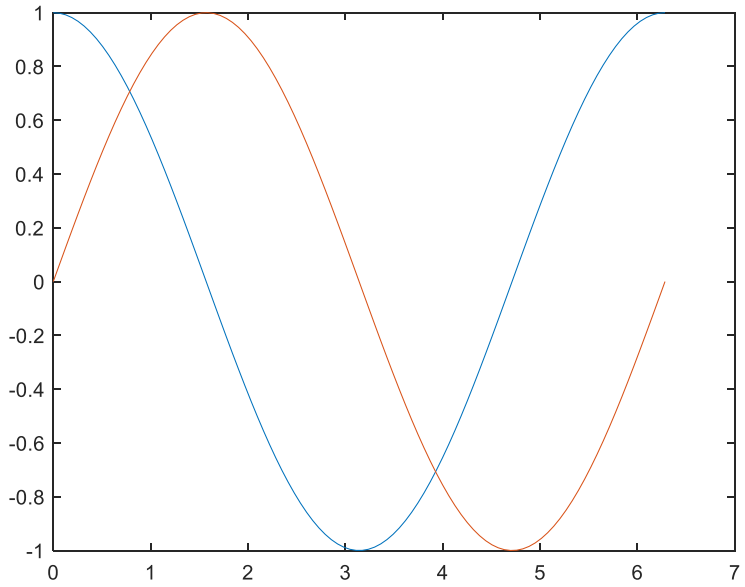
5: Graphische Ausgaben

2D Plots

Schnelle Alternative:

Sind X und Y n x m Matrizen, werden mit `plot(X,Y)` m Linien geplottet, für jede Spalte der Matrix Y eine Linie.

```
>> x=0:2*pi/99:2*pi;
>> plot([x' x'],...
     [cos(x)' sin(x)'])
```

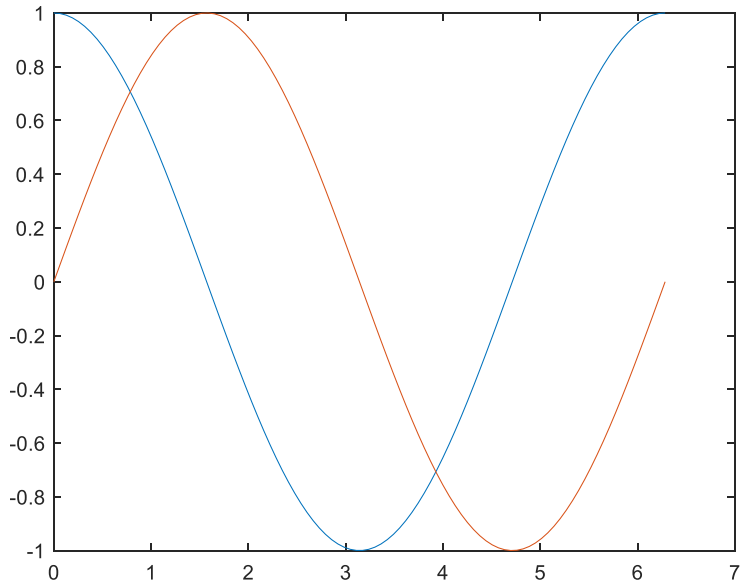


5: Graphische Ausgaben

2D Plots

Sind alle Spalten von X oder Y gleich, kann alternativ auch ein (Spalten- oder Zeilen-) Vektor übergeben werden, dessen Werte für alle Linien verwendet wird.

```
>> x=0:2*pi/99:2*pi;
>> plot(x,...
     [cos(x)' sin(x)'])
```



5: Graphische Ausgaben

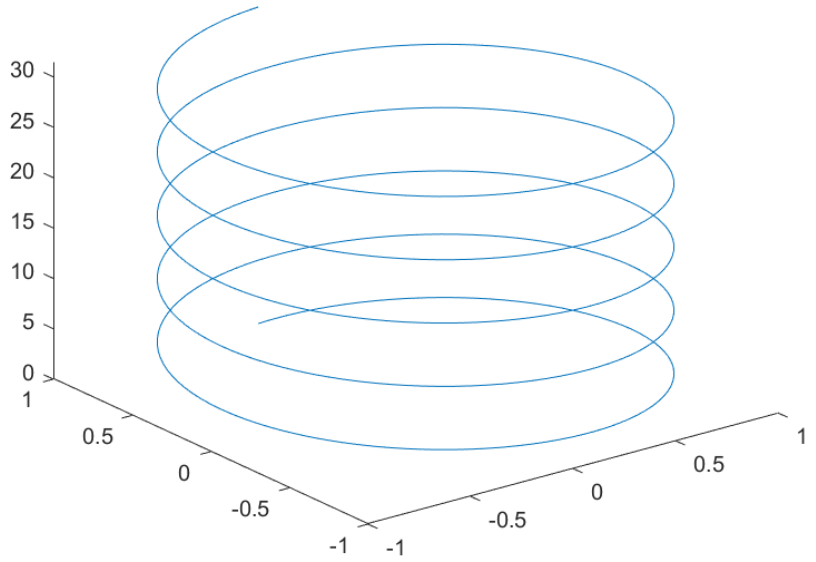
Inhaltsverzeichnis

5: Graphische Ausgaben

3D Grafiken

Um 3D-Linienplots zu erstellen, wird der Befehl `plot3` verwendet.

```
t = 0:pi/50:10*pi
st = sin(t);
ct = cos(t);
plot3(st,ct,t)
```



5: Graphische Ausgaben

3D Grafiken

Die Linien, die mit `plot3` erzeugt werden, können analog dem Befehl `plot` verändert werden (Linienart, Farbe, Marker).

Ein Unterschied: Sind X,Y,Z Matrizen, so erzeugt der Befehl `plot3(X,Y,Z)` für jede Zeile eine Linie.

Hat eine der Matrizen dabei nur eine Zeile, werden die Werte dieser Zeile für alle zu plottenden Linien verwendet.

5: Graphische Ausgaben

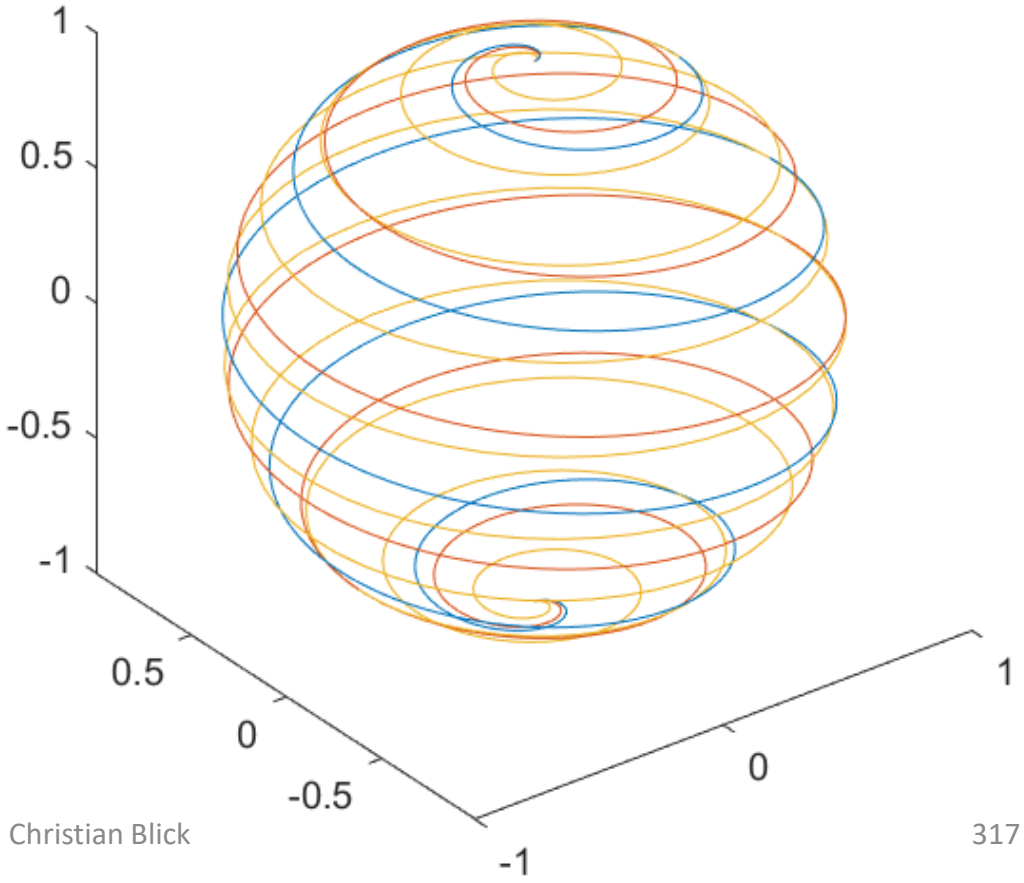
3D Grafiken

```
t = 0:pi/500:pi;

X(1,:) = sin(t).*cos(10*t);
X(2,:) = sin(t).*cos(12*t);
X(3,:) = sin(t).*cos(20*t);

Y(1,:) = sin(t).*sin(10*t);
Y(2,:) = sin(t).*sin(12*t);
Y(3,:) = sin(t).*sin(20*t);
```

5: Graphische Ausgaben

3D Grafiken

```
Z = cos(t); % Z ist ein Vektor

plot3(X,Y,Z)

axis equal
```



5: Graphische Ausgaben

3D Grafiken

Für die Erstellung von Flächenplots müssen zunächst, ähnlich zu `linspace` oder dem Doppelpunktoperator `a:h:b`, Gitter zum Plotten erzeugt werden.

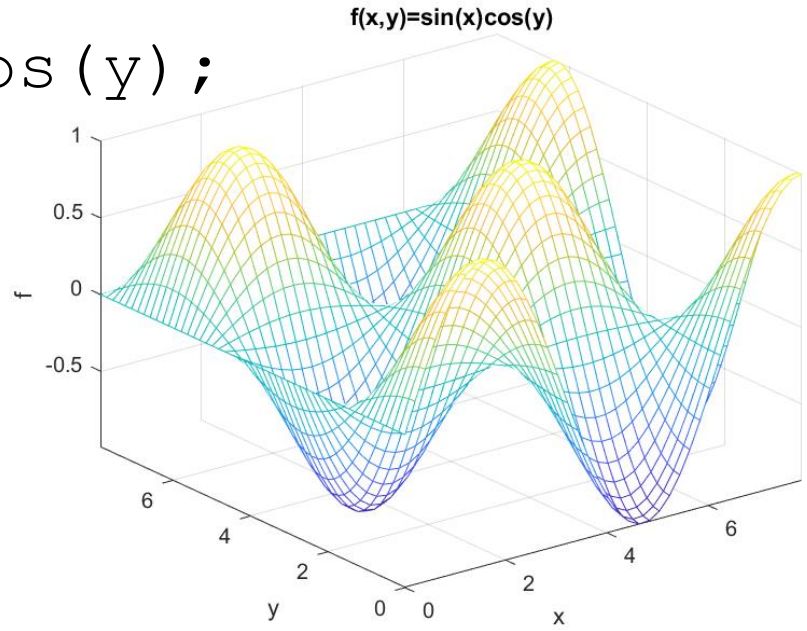
Der Befehl `ndgrid(x1,x2,…,xn)` erzeugt ein n-dimensionales Gitter mit Gitterabständen in x_i-Richtung wie im Vektor x_i definiert.

```
>>[X,Y]=ndgrid(linspace(0,2.5*pi,40)...
     ,linspace(0,2.5*pi,50));
```

5: Graphische Ausgaben

3D Grafiken

```
f=@(x,y) sin(x).*cos(y);
fd=f(X,Y);
```




```
figure(1)
mesh(X,Y,fd)
xlabel('x'), ylabel('y'), zlabel('f')
title('f(x,y)=sin(x)cos(y)')
axis tight
```

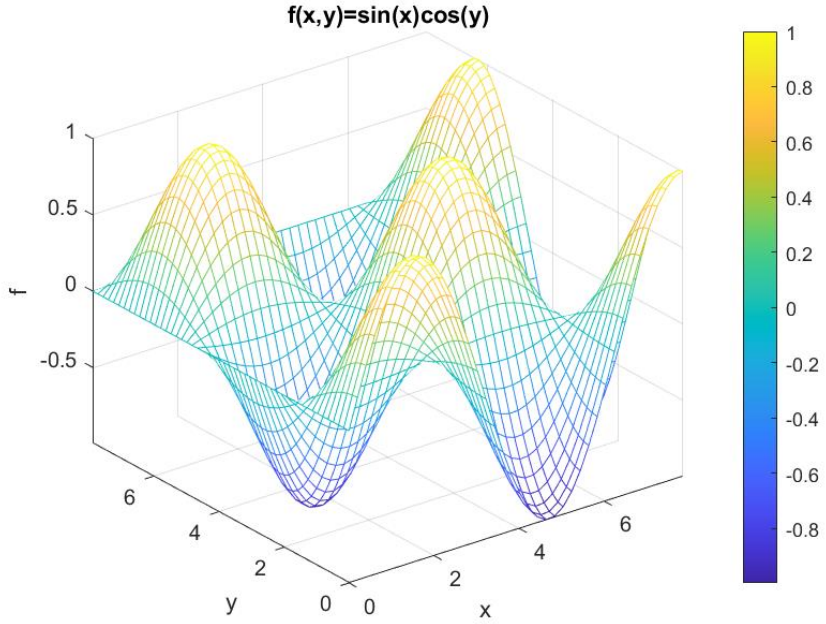
5: Graphische Ausgaben

3D Grafiken

Der Befehl `mesh(X,Y,fd)` erzeugt einen Gitterplot.

Mit dem Befehl colorbar wird dem Plot eine Farbskala hinzugefügt.

```
>> colorbar
```



5: Graphische Ausgaben

3D Grafiken

Die Farben der Colorbar werden standardmäßig den Funktionswerten `fd` gleichgesetzt.

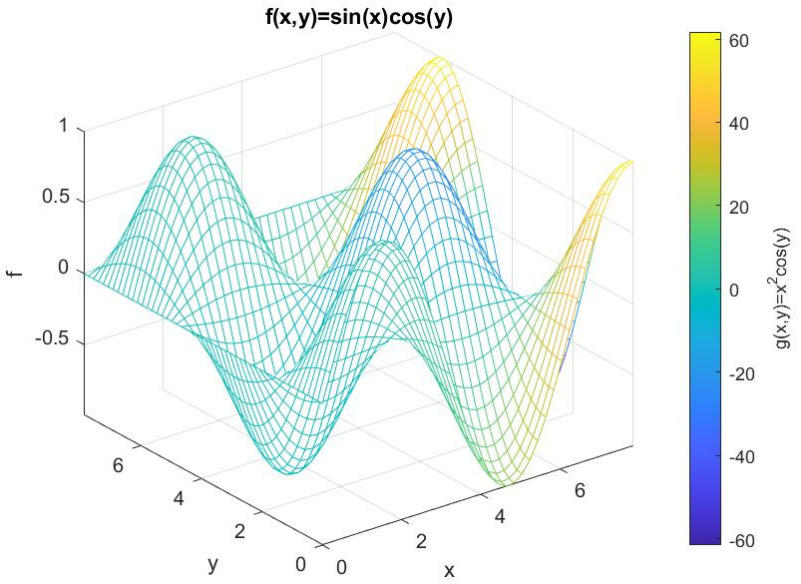
Dies kann mit zusätzlichen Parametern geändert werden.

```
>> mesh(X,Y,fd,CO)
```

`CO` ist eine Matrix der selben Dimension wie `fd` und entählt die Werte, die die Colorbar zeigen soll.

5: Graphische Ausgaben

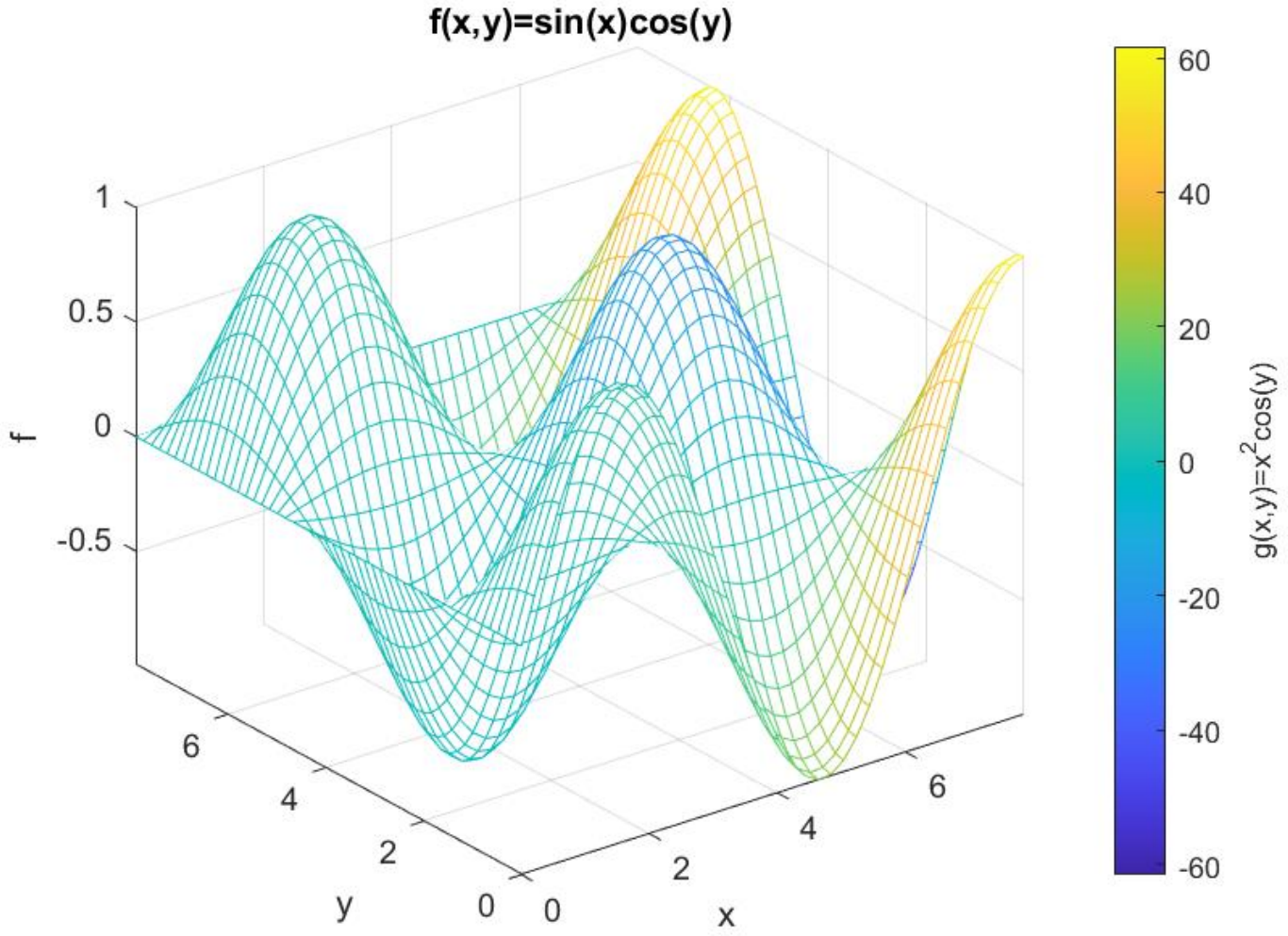
3D Grafiken




```
CO=X.^2.*cos(Y);
mesh(X,Y,fd,CO)
xlabel('x')
ylabel('y')
zlabel('f')
title('f(x,y)=sin(x)cos(y)')
axis tight
c=colorbar;
c.Label.String = 'g(x,y)=x^2cos(y)';
```

5: Graphische Ausgaben

3D Grafiken



5: Graphische Ausgaben

3D Grafiken

Die Colorbar kann, wie jede Achse in einem Plot manipuliert werden.

Hier wird die Colorbar mit dem Befehl

```
c=colorbar;
c.Label.String = 'g(x,y)=x^2cos(y)';
```

beschriftet.

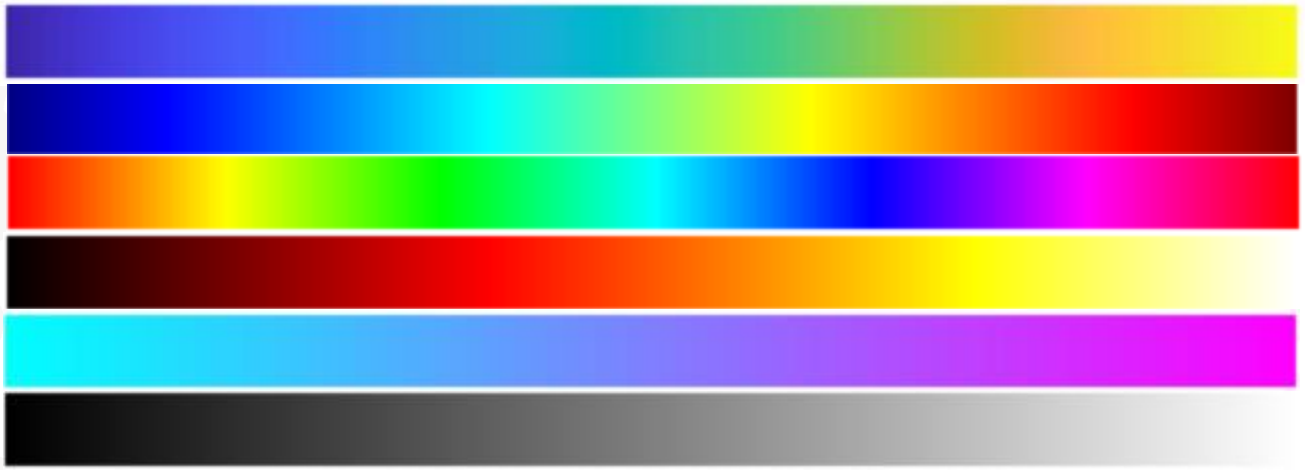
5: Graphische Ausgaben

3D Grafiken

Das Farbschema einer Colormap lässt sich mit dem Befehl `colormap('name')` ändern. Eine Übersicht liefert die MatLab Hilfe.

Häufig genutzte Colorbars sind:

‚name‘	Colorbar
‘parula‘	
‘jet‘	
‘hsv‘	
‘hot‘	
‘cool‘	
‘gray‘	

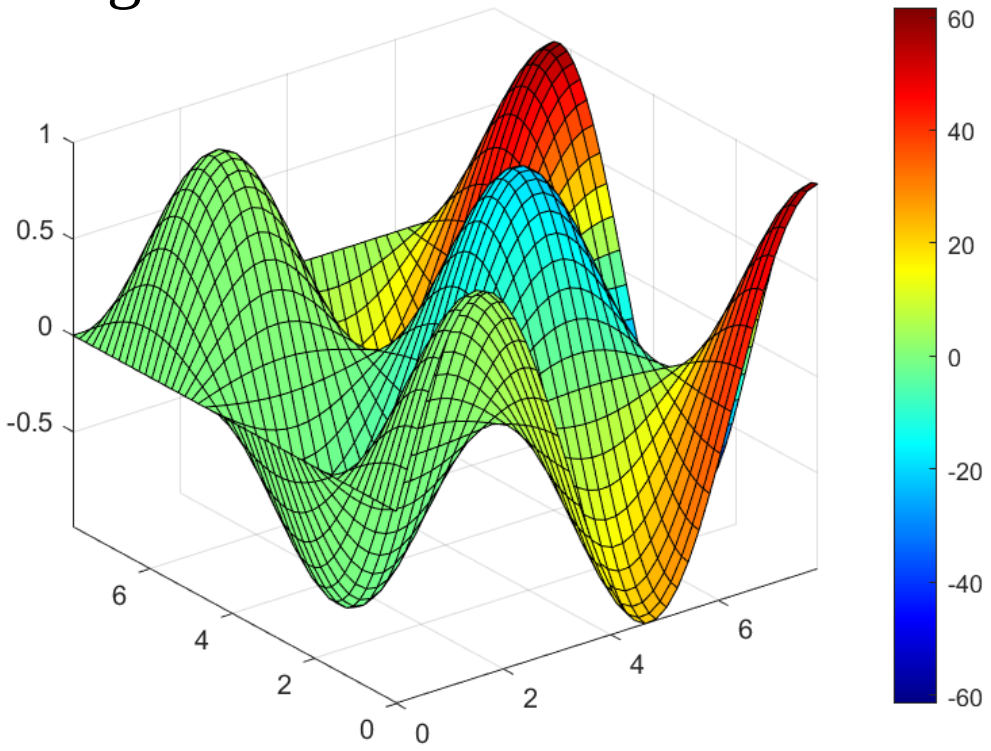


5: Graphische Ausgaben

3D Grafiken

Oberflächenplots werden mit dem Befehl `surf(X,Y,fd,CO)` erzeugt.

```
surf(X,Y,fd,CO)

axis tight
colorbar
colormap('jet')
```

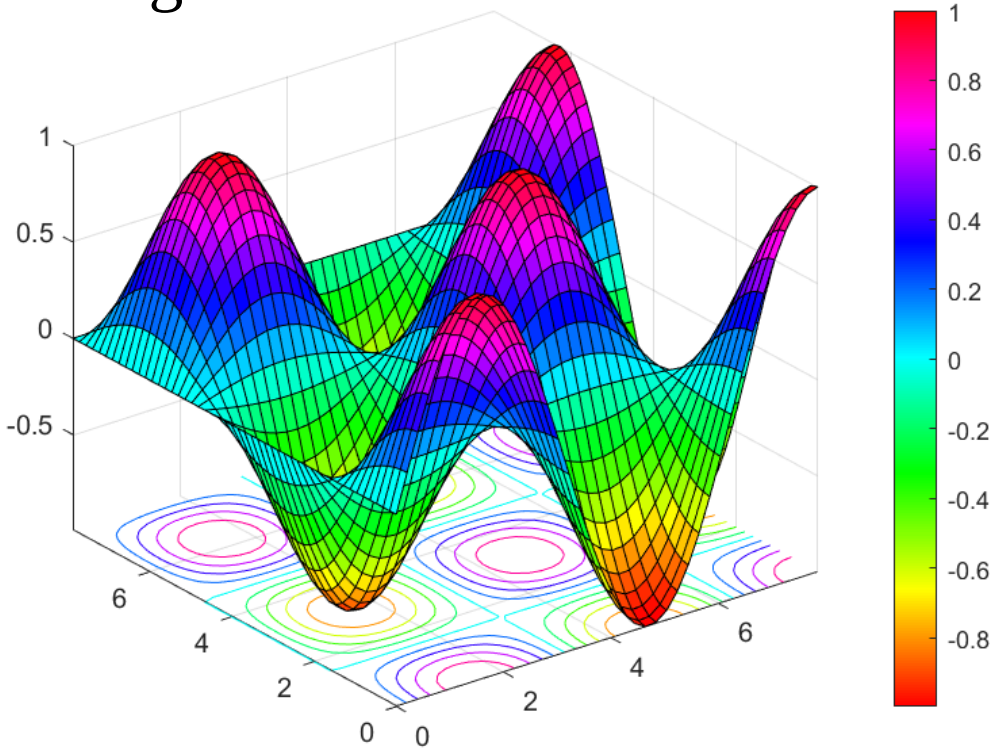


5: Graphische Ausgaben

3D Grafiken

Oberflächenplots mit Konturlinien werden mit dem Befehl `surfc(X,Y,fd,CO)` erzeugt.

```
surfc(X,Y,fd)

axis tight
colorbar
colormap('hsv')
```

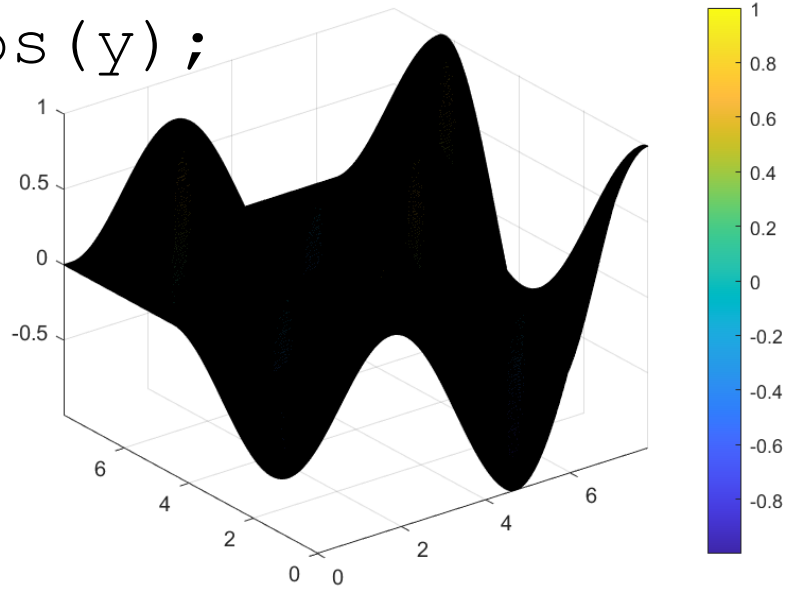


5: Graphische Ausgaben

3D Grafiken

Die schwarzen Gitterlinien werden zum Problem, wenn das Gitter sehr fein wird.

```
[X,Y]=ndgrid(linspace(0,2.5*pi,400)...
     ,linspace(0,2.5*pi,500));
f=@(x,y) sin(x).*cos(y);
fd=f(X,Y);


surf(X,Y,fd)
axis tight
```

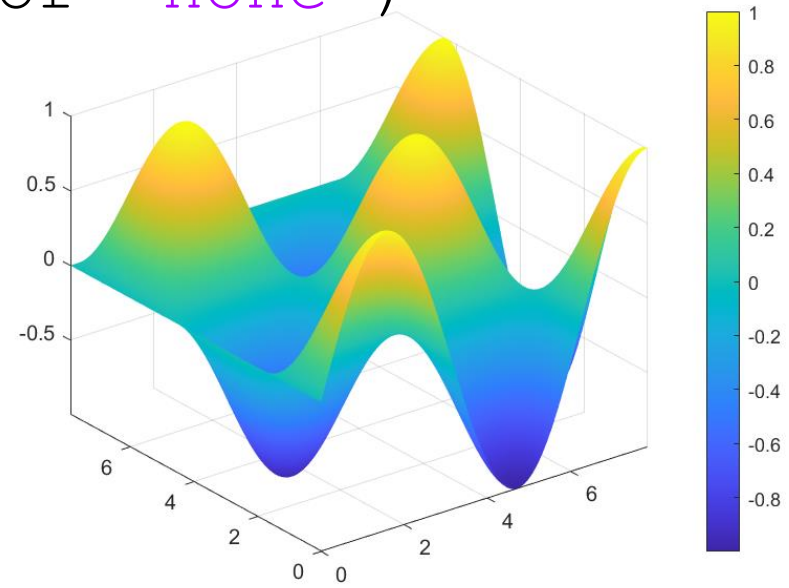


5: Graphische Ausgaben

3D Grafiken

Das Plotten des Gitters kann mit dem Parameter `EdgeColor="none"` unterdrückt werden.

```
surf(X,Y,fd,EdgeColor="none")

axis tight

colorbar
```

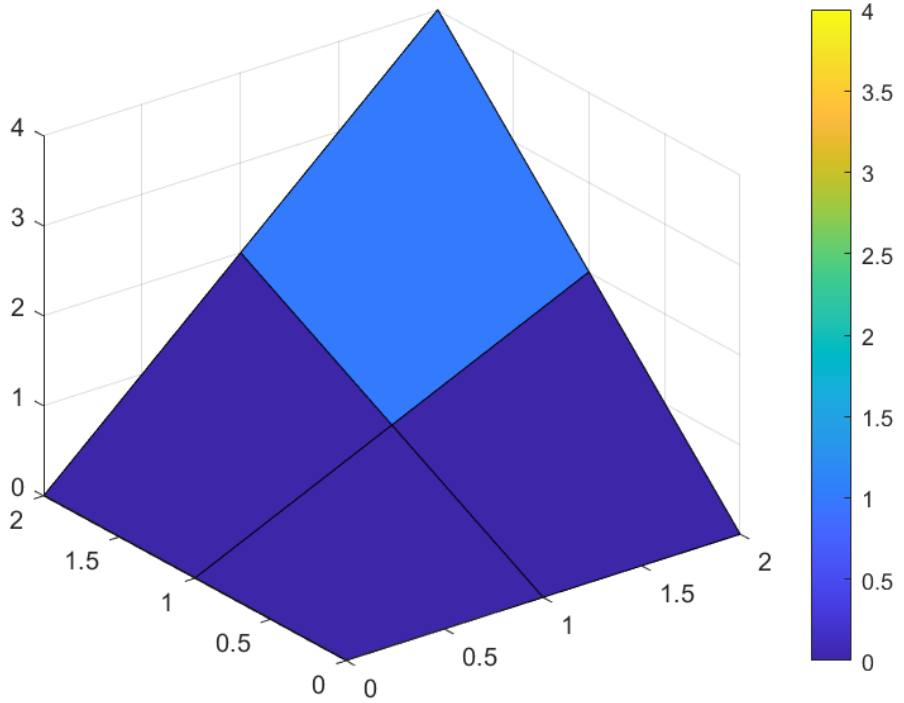


5: Graphische Ausgaben

3D Grafiken

Auswahl des Shadings: Wie werden die Farben gewählt?

```
[X,Y]=ndgrid(0:2,0:2);
f=@(x,y) x.*y;
fd=f(X,Y);

surf(X,Y,fd)
axis tight
colorbar
```

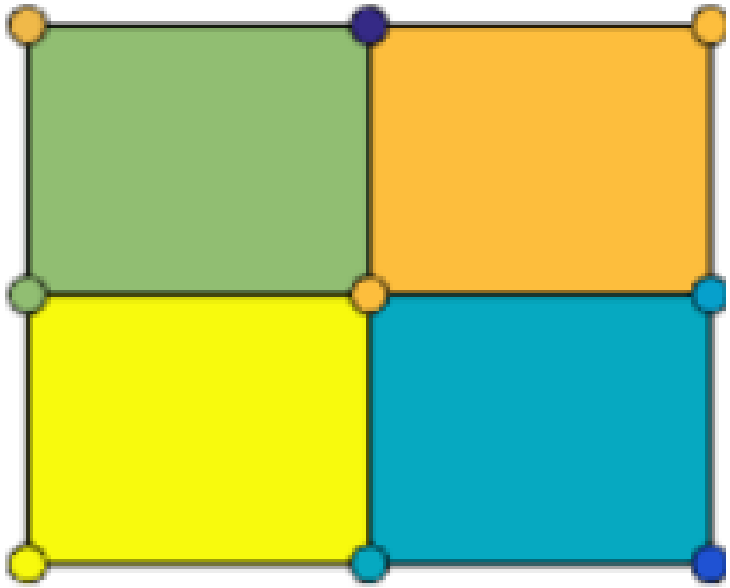


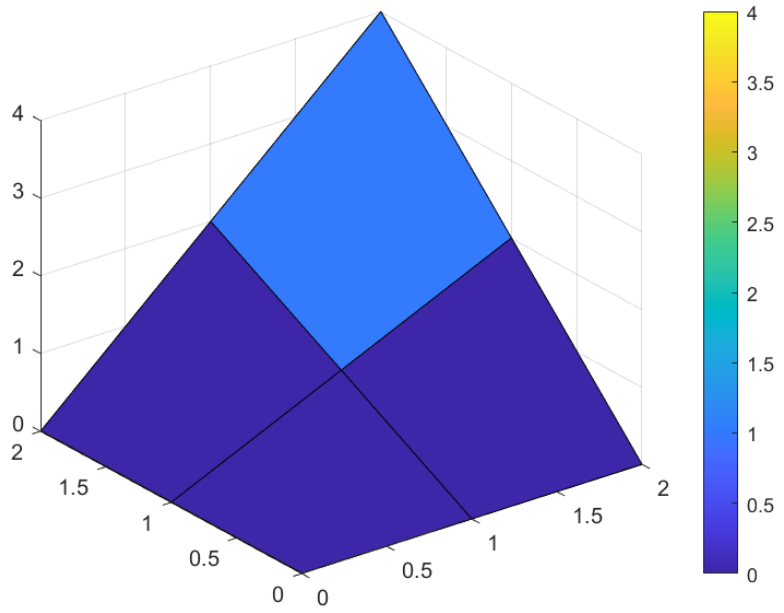
5: Graphische Ausgaben

3D Grafiken

`surf(X,Y,fd,FaceColor="flat")` (standard)

Jedem Flächenelement wird die Farbe zugeordnet, die im ersten Eckpunkt (kleinster x und y Wert) definiert ist.



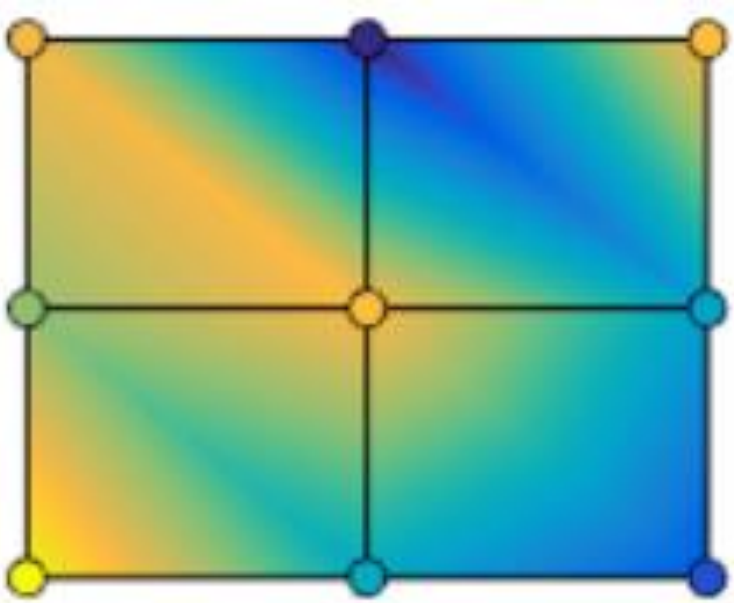


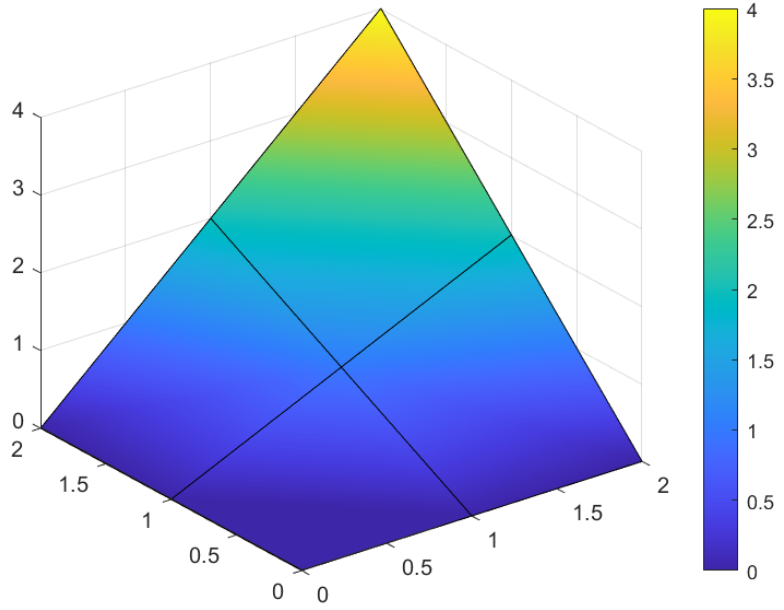
5: Graphische Ausgaben

3D Grafiken

```
surf(X,Y,fd,FaceColor="interp")
```

Die Farben in jedem Flächenelement werden interpoliert.





5: Graphische Ausgaben

Inhaltsverzeichnis

- 2D Plots
- Grafikexport
- Mehrere Plots in einer Figure
- 3D Grafiken
- MatLab-Movies

5: Graphische Ausgaben

MatLab-Movies

Mit MatLab können Videos (z. B. für Vorträge und Präsentationen) erzeugt werden.

Zunächst wird eine Figure vorbereitet.

```
Z = peaks;
surf(Z);
axis tight manual
set(gca,'nextplot','replacechildren');
```

5: Graphische Ausgaben

MatLab-Movies

`Z = peaks` wertet die peak-Funktion aus.

`axis tight manual` schränkt die Achsen auf den genutzten Wertebereich ein (`tight`) und verhindert, dass die Achsen geändert werden (`manual`).

```
set(gca,'nextplot','replacechildren')
```

sorgt dafür, dass jeder neue Plot den Vorigen ersetzt.

5: Graphische Ausgaben

MatLab-Movies

Zum Erstellen eines Videos muss zunächst ein VideoWriter-Objekt erstellt und geöffnet werden.

```
v = VideoWriter('peaks.avi');
open(v);
```

5: Graphische Ausgaben

MatLab-Movies

Als nächstes werden die einzelnen Grafiken geplottet und dem VideoWriter-Objekt mit dem Befehl `frame = getframe(gcf)` hinzugefügt.

```
for k = 1:20
    surf(sin(2*pi*k/20)*Z,Z)
    frame = getframe(gcf);
    writeVideo(v,frame);
end
```

5: Graphische Ausgaben

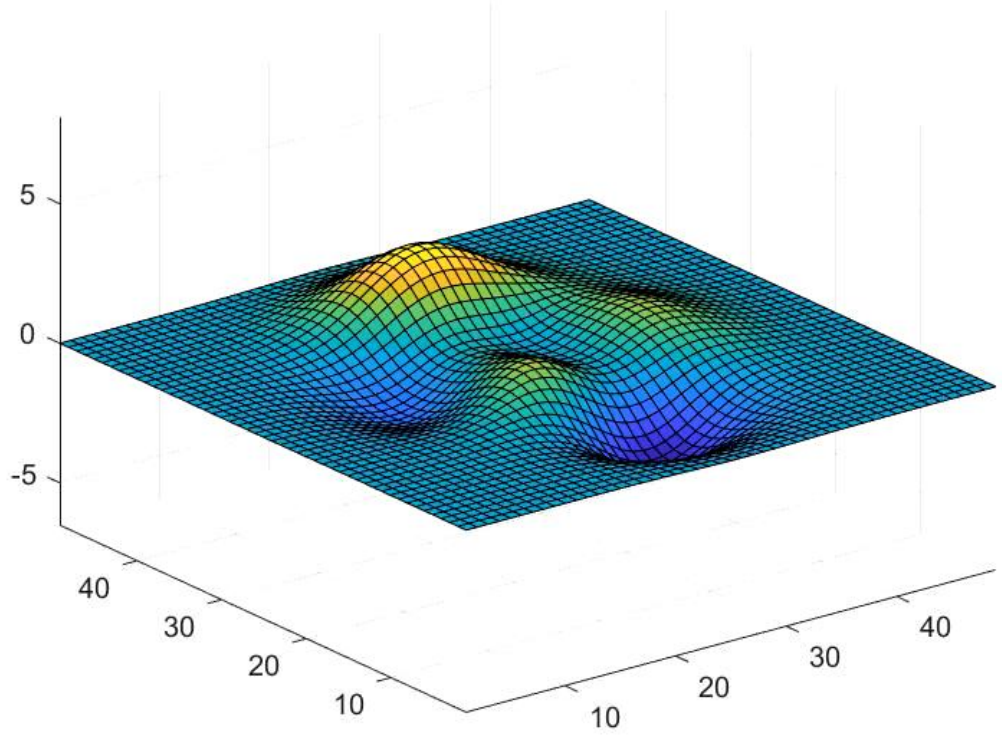
MatLab-Movies

Als nächstes werden die einzelnen Grafiken geplottet und dem VideoWriter-Objekt mit dem Befehl `frame = getframe(gcf)` hinzugefügt. Danach wird das Video mit `close(v)` geschlossen.

```
for k = 1:20
    surf(sin(2*pi*k/20)*Z,Z)
    frame = getframe(gcf);
    writeVideo(v,frame);
end
close(v)
```

5: Graphische Ausgaben

MatLab-Movies



5: Graphische Ausgaben

MatLab-Movies

Die Eigenschaften eines Videos können geändert werden, **bevor** das Video geöffnet wurde.

Zu den wichtigsten gehören:

- FrameRate – gibt an, wie viele Bilder pro Sekunde wiedergegeben werden.
- Quality – gibt die Qualität des Videos zwischen 0 und 100 an.

5: Graphische Ausgaben

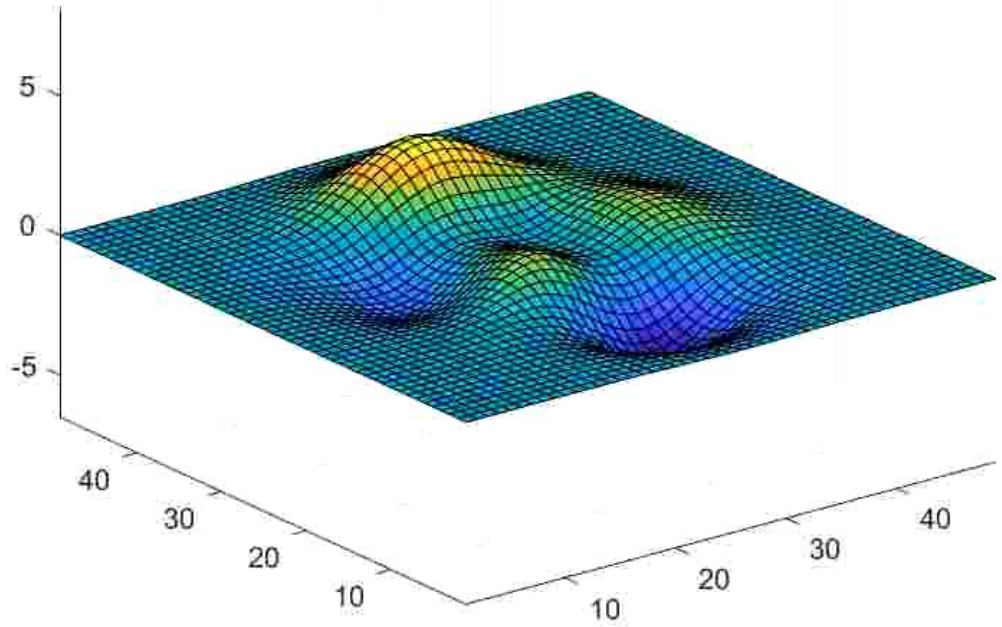
MatLab-Movies

In diesem Beispiel:

```
v = VideoWriter('peaks2.avi');
v.FrameRate=5;
v.Quality=10;
```

5: Graphische Ausgaben

MatLab-Movies



5: Graphische Ausgaben

MatLab-Movies

Vorlesungsevaluation

Inhaltsverzeichnis

6: Symbolic Toolbox

Inhaltsverzeichnis

6: Symbolic Toolbox

Lernziele:

- Sie können symbolische Ausdrücke in MatLab implementieren.
- Sie können grundlegende arithmetische Operationen mit symbolischen Ausdrücken durchführen.
- Sie können symbolische Funktionen plotten.

6: Symbolic Toolbox

Inhaltsverzeichnis

6: Symbolic Toolbox

Einführung

Symbolische Ausdrücke werden mit dem Befehl `sym` erzeugt.

```
>> sym(1/3)
 ans =
 1/3

>> 1/3
ans =
    0.3333
```

6: Symbolic Toolbox

Einführung

Symbolische Variablen werden entweder mit dem Befehl `sym` oder mit `syms` erzeugt. Die Syntax lautet:

```
>> sym('x')
ans =
    x
```

`syms` erlaubt die gleichzeitige Erzeugung mehrerer Variablen.

```
>> syms y z
```

6: Symbolic Toolbox

Einführung

Ein Vektor bestehend aus symbolischen Variablen wird mit `sym` erzeugt.

```
>> clear all
>> A = sym('a', [1 3])
A =
[a1, a2, a3]
>> whos
 Name   Size   Bytes   Class    Attributes
 A      1x3    8       sym
```

6: Symbolic Toolbox

Einführung

Kombiniert man sym und syms, lassen sich schnell einzelne Variablen erzeugen.

```
>> clear all
>> syms(sym('a', [1 3]))
>> whos
  Name   Size   Bytes  Class  Attributes
  a1     1x1    8      sym
  a2     1x1    8      sym
  a3     1x1    8      sym
```

6: Symbolic Toolbox

Einführung

Symbolische Funktionen werden ebenfalls mit `syms` erzeugt und können durch Eingabe von Werten ausgewertet werden.

```
>> syms f(x,y)
>> f(x,y)=x^2*y
 f(x, y) =
 x^2*y
>> f(3,2)
ans =
18
```

6: Symbolic Toolbox

Einführung

Neben der Erzeugung symbolischer Matrizen mit

```
>> A = sym('a', [2 3])
 A =
 [a1_1, a1_2, a1_3]
 [a2_1, a2_2, a2_3]
```

können Matrizen auch kompakt definiert werden.

```
>> syms B [2 3] matrix
>> B
B = B
```

6: Symbolic Toolbox

Einführung

Symbolische Matrizen können in Arrays symbolischer Variablen umgewandelt werden.

```
>> symmatrix2sym(B)
 ans =
[B1_1, B1_2, B1_3]
[B2_1, B2_2, B2_3]
```

6: Symbolic Toolbox

Einführung

Symbolische Matrizen können indiziert werden.

```
>> B(2,1)
 ans =
 B2_1

>> A(2,1)
 ans =
 a2_1
```

6: Symbolic Toolbox

Inhaltsverzeichnis

- Einführung
- Symbolisches Rechnen
- Grafiken von symbolischen Funktionen

6: Symbolic Toolbox

Symbolisches Rechnen

MatLab kann symbolische Ausdrücke mit dem Befehl `diff` ableiten.

```
>> syms x
>> f = sin(x)^2;
>> diff(f)
 ans =
 2*cos(x)*sin(x)
```

6: Symbolic Toolbox

Symbolisches Rechnen

Wird keine Variable, nach der abgeleitet werden soll angegeben, wählt MatLab selbst die Variable (hier `x`).

```
>> syms x y
>> f = sin(x)^2 + cos(y)^2;
>> diff(f)
 ans =
 2*cos(x)*sin(x)
```

6: Symbolic Toolbox

Symbolisches Rechnen

Nun wird `f` nach `y` abgeleitet.

```
>> syms x y
>> f = sin(x)^2 + cos(y)^2;
>> diff(f,y)
 ans =
 -2*cos(y)*sin(y)
```

6: Symbolic Toolbox

Symbolisches Rechnen

Höhere Ableitungen nach der selben Variablen werden wie folgt berechnet:

```
>> diff(f,x,2)
ans =
2*cos(x)^2 - 2*sin(x)^2
```

Dies entspricht:

```
diff(diff(f,x),x)
ans =
2*cos(x)^2 - 2*sin(x)^2
```

6: Symbolic Toolbox

Symbolisches Rechnen

Auch gemischte Ableitungen können berechnet werden.

```
>> diff(f,x,y)
ans =
0
```

Dies entspricht

```
>> diff(diff(f,x),y)
ans =
0
```

6: Symbolic Toolbox

Symbolisches Rechnen

Unbestimmte Integrale werden mit dem Befehl `int` berechnet.

```
>> syms x y n
>> f = x^n + y^n;
>> int(f,x)
ans =
x*y^n + (x*x^n)/(n + 1)
```

6: Symbolic Toolbox

Symbolisches Rechnen

```
>> int(f,y)
ans =
x^n*y + (y*y^n)/(n + 1)

>> int(f,n)
ans =
x^n/log(x) + y^n/log(y)
```

6: Symbolic Toolbox

Symbolisches Rechnen

Bei der Berechnung bestimmter Integrale müssen die Integralgrenzen angegeben werden.

```
>> int(f,x,1,10)

ans =

piecewise(n == -1, log(10) + 9/y,
n ~= -1, (10*10^n - 1)/(n + 1) +
9*y^n)
```

6: Symbolic Toolbox

Symbolisches Rechnen

Für die Berechnung von Grenzwerten wird die Funktion `limit(f,Variable,x)` verwendet.

```
>> f=sin(x-a)/(x-a)
f =
sin(a - x)/(a - x)
>> limit(f,x,a)
ans =
1
```

6: Symbolic Toolbox

Symbolisches Rechnen

MatLab kann symbolische Gleichungen lösen. Die Syntax lautet `solve(Gleichung,Variable)`.

```
>> syms x y
>> solve(6*x^2 - 6*x^2*y + x*y^2 - x*y
+ y^3 - y^2 == 0, y)
ans =
   1
 2*x
-3*x
```

6: Symbolic Toolbox

Symbolisches Rechnen

Auch Gleichungssysteme lassen sich lösen. Dabei werden einzelne Gleichungen mit Komma voneinander getrennt.

```
>> syms x y z
>> [x, y, z] = solve(z == 4*x, ...
     x == y,  z == x^2 + y^2)
x =             y =             z =
0               0               0
2               2               8
```

6: Symbolic Toolbox

Symbolisches Rechnen

Beim Integrieren und Differenzieren komplexer Ausdrücke sind die Resultate oft sehr lang. Um Ausdrücke zu vereinfachen, benutzt man den Befehl `simplify`.

```
>> phi = (1 + sqrt(sym(5)))/2;
>> f = phi^2 - phi - 1
f =
(5^(1/2)/2 + 1/2)^2 - 5^(1/2)/2 - 3/2
```

6: Symbolic Toolbox

Symbolisches Rechnen

Beim Integrieren und Differenzieren komplexer Ausdrücke sind die Resultate oft sehr lang. Um Ausdrücke zu vereinfachen, benutzt man den Befehl `simplify`.

```
f =
(5^(1/2)/2 + 1/2)^2 - 5^(1/2)/2 - 3/2
>> simplify(f)
ans =
0
```

6: Symbolic Toolbox

Symbolisches Rechnen

Ausdrücke können auch vereinfacht werden, indem sie ausmultipliziert werden, sodass sich Terme wegheben. Dies geschieht mit `expand`.

```
>> syms x
>> f = (x ^2- 1)*(x^4 + x^3 + x^2 +...
     x + 1)*(x^4 - x^3 + x^2 - x + 1);
>> expand(f)
ans =
x^10 - 1
```

6: Symbolic Toolbox

Symbolisches Rechnen

Mit dem Befehl `subs` können Ausdrücke ineinander eingesetzt werden.

```
>> syms x y
>> f = x^2*y + 5*x*sqrt(y);
>> subs(f, x, 3)
ans =
9*y + 15*y^(1/2)
```

6: Symbolic Toolbox

Symbolisches Rechnen

```
>> subs(f, y, x)
ans =
x^3 + 5*x^(3/2)

>> subs(f, y, x^2-y)
ans =
5*x*(x^2 - y)^(1/2) - x^2*(- x^2 + y)
```

6: Symbolic Toolbox

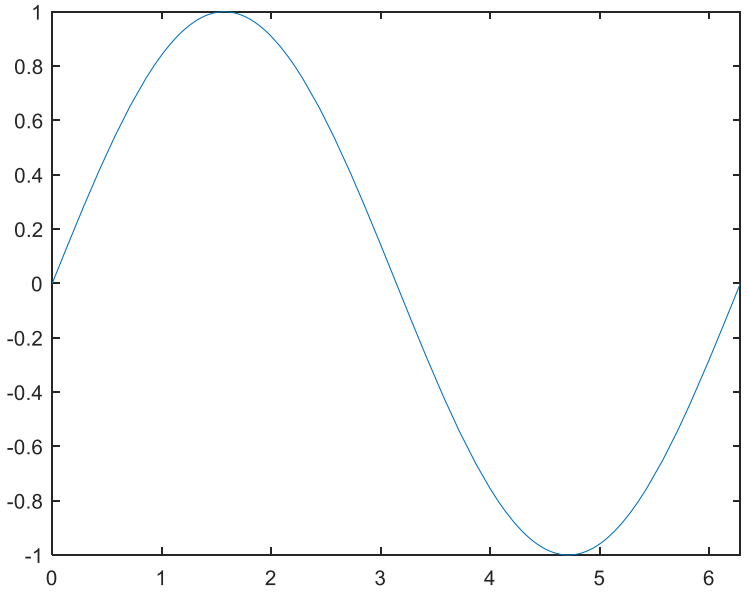
Inhaltsverzeichnis

6: Symbolic Toolbox

Grafiken von symbolischen Funktionen

Ein 2D-Linienplott ähnlich dem Befehl `plot` eines symbolischen Ausdrucks wird mit `fplot(f,Intervall)` erzeugt.

```
>> f=sin(x)

f =

sin(x)


>> fplot(f,[0,2*pi])
```



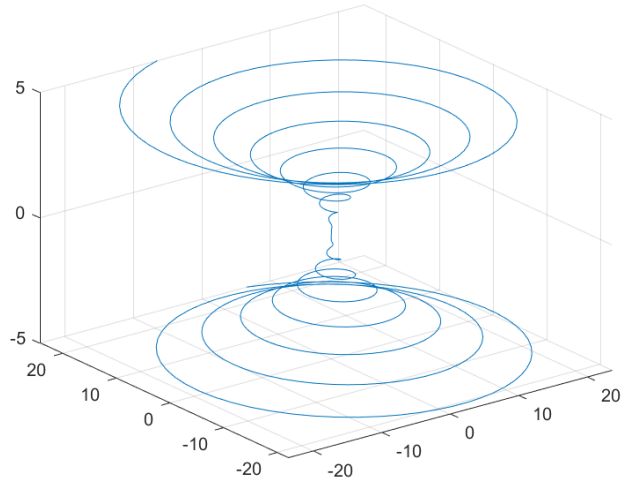
6: Symbolic Toolbox

Grafiken von symbolischen Funktionen

Dem Befehl plot3 entspricht der Befehl fplot3 für symbolische Funktionen. Die Syntax lautet

```
fplot3(x(t),y(t),z(t),Intervall_t)
```

```
>> syms t
>> fplot3(t^2*sin(10*t),t^2*cos(10*t), t)
```



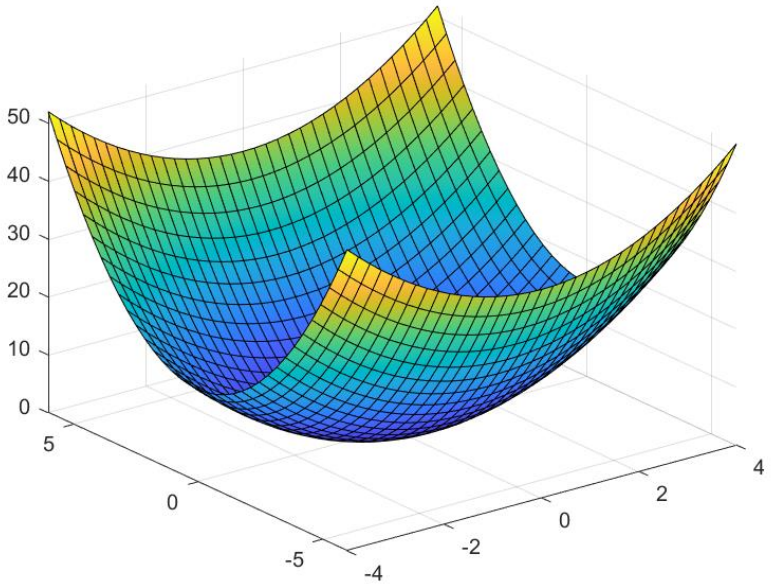
6: Symbolic Toolbox

Grafiken von symbolischen Funktionen

Für Flächenplots wird die Funktion

`fsurf(f,[xmin xmax ymin ymax])` verwendet.

```
>> fsurf(x^2 + y^2,[-4 4 -6 6])
```



6: Symbolic Toolbox

Online-Tutorial: ca. 45 min

https://matlabacademy.mathworks.com/details/introduction-to-symbolic-math-with-matlab/symbolic