

Computerorientiertes Problemlösen

25. September – 29. September 2017

Steffen Basting



technische universität
dortmund

fakultät für
mathematik
lehrstuhl für
angewandte mathematik

m!

WS 2017-2018

Ziele der Veranstaltung

- Vorbereitung auf die Vorlesung Numerik I mit praktischen Übungen
(numerische Algorithmen sollen in MATLAB® oder Octave programmiert werden)
- Computerorientiertes Lösen mathematischer Probleme erlernen:

- Aufgabenstellung in numerischen Algorithmus umformulieren

Bsp: "invertiere Matrix A " → " $A^{-1} = \text{Gaußelimination}(A)$ "

- Algorithmus in für Computer verständliche Anweisungen zerlegen

Bsp: "für alle Zeilen $i = 1, \dots, n$ " → "for i=1:n"

- Berechnete Ergebnisse mit mathematischem Verstand bewerten

Bsp: warning: inverse: matrix singular to machine precision, rcond = 4.89645e-18

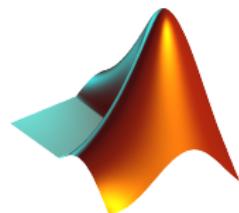
Numerisches Softwarewerkzeug MATLAB®

Eigenschaften

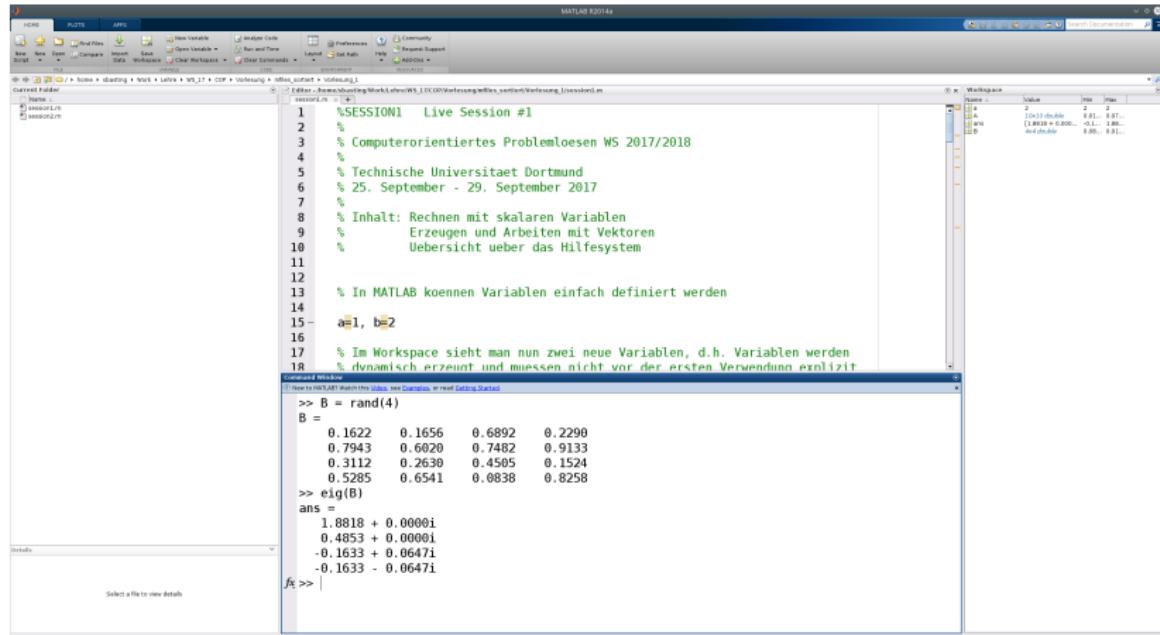
- Etablierte kommerzielle Software für numerische Berechnungen
- Unterstützung aller gängigen Betriebssysteme (M-files sind portabel)
- Interaktive, benutzerfreundliche Programmierumgebung (in Java)
- Sehr umfangreiche Bibliothek von mathematischen Funktionen
- Erweiterbar durch (kommerzielle) Toolboxen und kostenlose M-files
- Schnittstellen zu Hochsprachen wie C und Fortran sind vorhanden

Bezugsquelle

- Kommerzieller Vertrieb durch die Firma MathWorks® <http://www.mathworks.de/>
- Studentenversion kostet ca. 100 €



MATLAB® Umgebung



Open-Source Alternative GNU Octave

Eigenschaften

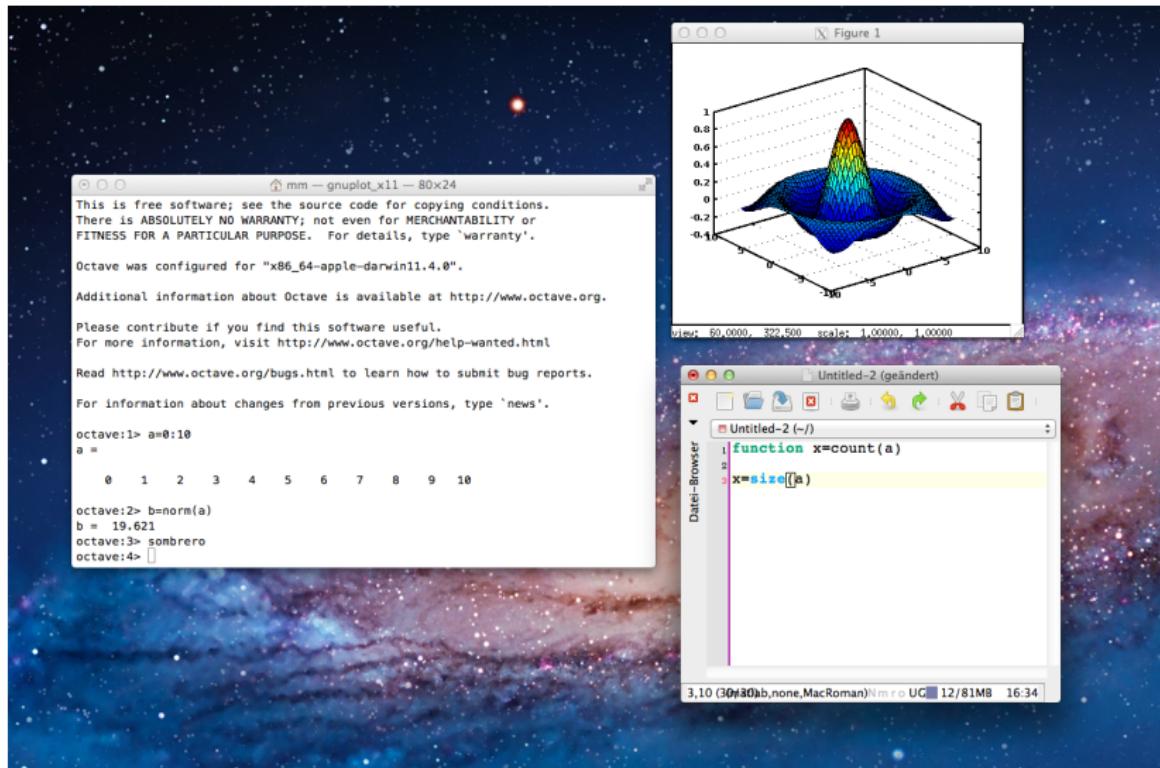
- Weitgehend sprachkompatibel zu MATLAB® (M-files lauffähig)
- Funktionsumfang entspricht der Basisversion von MATLAB®
- Unterstützung aller gängigen Betriebssysteme
- Erweiterbar durch Packages <http://www.gnu.org/software/octave/>
- Benutzerfreundliche Programmierumgebung (seit Version 4.0)
- Kostenlose Nutzung auf privaten Laptops möglich

Bezugsquelle

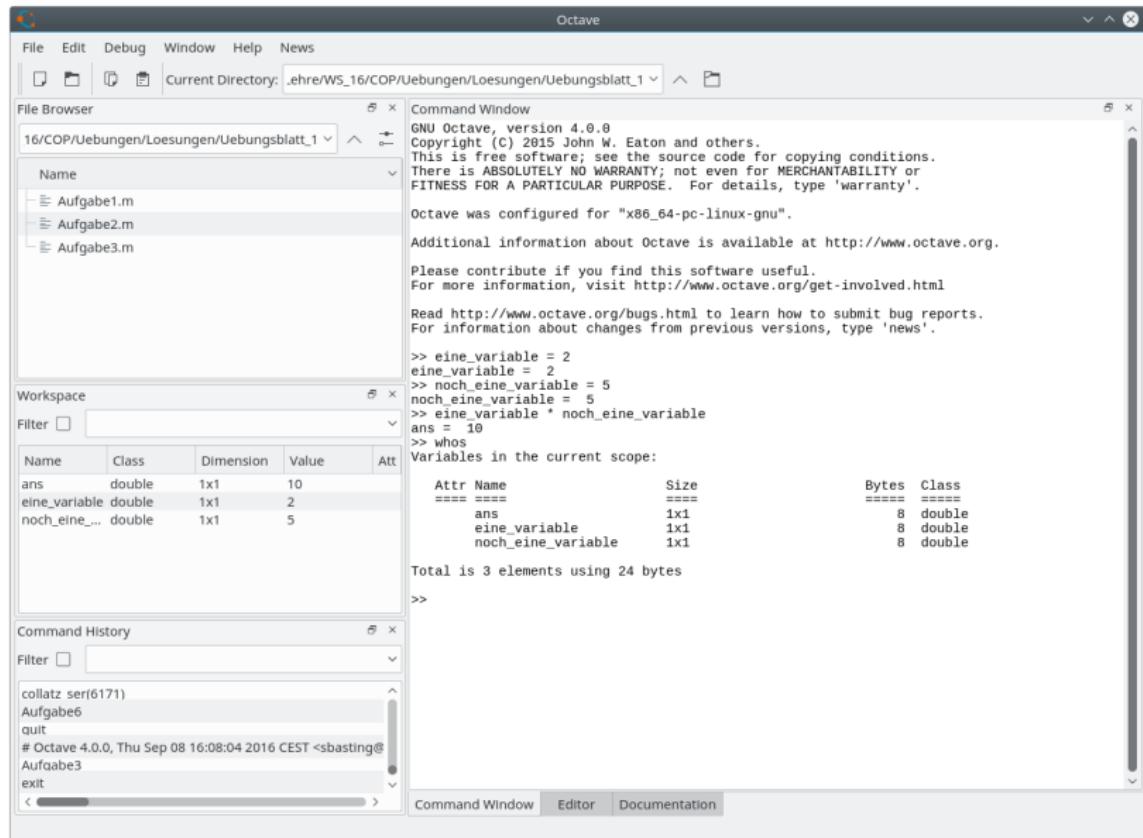
- Quellcode und z.T. Binärpakete frei verfügbar
<http://www.gnu.org/software/octave/>
- Image für USB-Stick auf der Kurshomepage



GNU Octave 3.x "Umgebung"



GNU Octave 4.x Umgebung (aktuell)



MATLAB[®] vs. GNU Octave

- Grundfunktionalität beider Softwarepakete vergleichbar
- GNU Octave reicht zum Bearbeiten der Übungsaufgaben in diesem Kurs und in den Vorlesungen Numerik 1-2 aus
- MATLAB[®] bietet mehr Komfort beim Programmieren und Debuggen speziell von größeren Programmen (z.B. mit GUI)
- Funktionsumfang von MATLAB[®] lässt sich durch kommerzielle Toolboxen deutlich erweitern (Zielgruppe: Ingenieurusanwendungen)
- MATLAB[®] ist insb. in den Ingenieurwissenschaften etabliert
- Eine Übersicht der Unterschiede liefert die Octave FAQ Abschnitt 10

Literatur zu MATLAB®/GNU Octave

- D.J. Higham, N.J. Higham, MATLAB Guide, SIAM, 2005.
- W. Schweizer, MATLAB kompakt, Oldenbourg-Verlag, 2009.
- G. Gramlich, W. Werner, Numerische Mathematik mit MATLAB:
Eine Einführung für Naturwissenschaftler und Ingenieure.
Dpunkt-Verlag, 2000.
- C. Überhuber, S. Katzenbeisser, D. Praetorius, MATLAB 7:
Eine Einführung. Springer-Verlag, 2005.
- A. Quarteroni, F. Saleri, Scientific Computing with MATLAB and
Octave, Springer, 2006.

Internet

- alternative Vorlesungsskripte und Kurzanleitungen findet man leicht
mit gängigen Suchmaschinen

Überblick 1. Vorlesung

- integriertes Hilfesystem
- Variablen, Vektoren und Matrizen
- mathematische Operationen
- Ein- und Ausgabe von Daten
- Speichern von Befehlsvorschriften

Zusammenfassung: Hilfesystem

- `help <Thema>` gibt Hilfetexte auf dem Bildschirm aus
- `lookfor <Thema>` durchsucht Hilfetexte nach Stichwort
- `doc <Thema>` öffnet grafisches Hilfesystem/Browser
- `demo <Thema>` öffnet interaktive MATLAB Demos

Nützliche Hilfethemen

- `general` Generelle Befehle (`who`, `clear`, ...)
- `ops` Operationen (+, -, *, /, ^, [], ...)
- `elfunc` Mathematische Funktionen (`min`, `max`, `sqrt`, ...)
- `elmat` Matrix Funktionen
- `lang` Programmierung

Zusammenfassung: Variablen

- Es wird zwischen Groß- und Kleinschreibung unterschieden
- Variablen werden dynamisch erzeugt und sind veränderbar
- Wenn keine Variable angegeben wird, dann wird das zuletzt berechnete Ergebnis in `ans` (=answer) gespeichert
- Eine Übersicht über alle Variablen liefern `who` bzw. `whos`
- Variablen lassen sich mittels `clear <Variablenname>` löschen; alle Variablen werden mit `clear` oder `clear all` gelöscht
- Die Bildschirmausgabe lässt sich mittels Semikolon unterdrücken

Komplexe Zahlen

- Komplexe Zahlen $z = x + iy \in \mathbb{C}$ können wie folgt definiert werden

```
>> z=3+2i
```

```
>> z=3+2*i
```

```
>> z=3+2j
```

```
>> z=3+2*j
```

```
>> z=complex(3,2)
```

```
z =
```

```
3.000 + 2.000i
```

- Variablen *i* und *j* sind mit der komplexen Einheit $\sqrt{-1}$ vordefiniert
- Wird der Wert von *i* oder *j* überschrieben, so ist nur $z=3+2i$ gültig

- `>> whos z`

Name	Size	Bytes	Class	Attributes
z	1x1	16	double	complex

entspricht 2 Doubles á 8 Bytes

Komplexe Zahlen

Funktionen für komplexe Zahlen $z = x + iy$

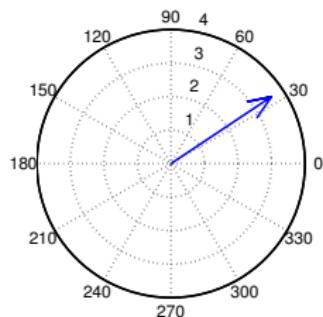
abs	Absolutwert	$ z = \sqrt{x^2 + y^2} = r$
imag	Imaginärteil	$\text{Im}(z) = b$
real	Realteil	$\text{Re}(z) = a$
conj	konjugiert komplex	$\bar{z} = x - iy$
angle	Winkel	$\arg(z) = \varphi$

■ Polarkoordinatendarstellung

$$z = re^{i\varphi} = r(\cos \varphi + i \sin \varphi)$$

■ Grafische Darstellung komplexer Zahlen

`>> z=3+2i; compass(z)`



Vektoren

- Zeilenvektoren können direkt erzeugt werden

```
>> a=[1 3 5 7 9 11 13 15 17 19]  
a =  
    1 3 5 7 9 11 13 15 17 19
```

oder mittels Doppelpunkt-Operator

```
>> a=1:2:20  
a =  
    1 3 5 7 9 11 13 15 17 19
```

- Ohne Angabe der Schrittweite wird automatisch 1 angenommen

```
>> a=1:20  
a =  
    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Vektoren

- Negative Schrittweiten sind zulässig, wenn gilt: Startwert > Endwert

```
>> a=20:-2:1
```

```
a =
```

```
20 18 16 14 12 10 8 6 4 2
```

- Anfangs-, Endwert und Schrittweite können nichtganzzahlig sein

```
>> a=0.3:0.1:0.7
```

```
a =
```

```
0.3000 0.4000 0.5000 0.6000 0.7000
```

```
>> b=0:pi/2:2*pi
```

pi ist als π vordefiniert

```
b =
```

```
0 1.5708 3.1416 4.7124 6.2832
```

Vektoren

- Spaltenvektoren können direkt mittels ' ; ' erzeugt werden oder durch Transponieren des entsprechenden Zeilenvektors

```
>> a=[1; 2; 3; 4; 5]
```

```
a =
```

```
1  
2  
3  
4  
5
```

```
>> b=transpose(1:2:10)
```

```
b =
```

```
1  
3  
5  
7  
9
```

- Kurzform ist a.' und b' entspricht komplex konjugiert transponiert

```
>> a=(1:3).'
```

```
a =
```

```
1  
2  
3
```

```
>> b=[1+i 2+i 3+i]'
```

```
b =
```

```
1.0000 - 1.0000i  
2.0000 - 1.0000i  
3.0000 - 1.0000i
```

Vektoradressierung

- Einen einzelnen Vektoreintrag adressiert/verändert man mit

```
>> a=1:5; a(3)=pi
```

```
a =
```

```
1.0000 2.0000 3.1416 4.0000 5.0000
```

- Auf den letzten Vektoreintrag greift man mit end zu

```
>> b=1:5; b(end)=1
```

```
b =
```

```
1 2 3 4 1
```

- Teilvektoren können direkt oder mittels ':' adressiert werden

```
>> a(2:4)=42
```

```
a =
```

```
1 42 42 42 5
```

```
>> b([1 3 end])=42
```

```
b =
```

```
42 2 42 4 42
```

Vektoradressierung

- Vektoren können zu größeren Vektoren zusammengesetzt werden

```
>> a=1:5; b=6:10; c=[a, b]
```

c =

1 2 3 4 5 6 7 8 9 10

Dabei kann das Komma bei Zeilenvektoren entfallen.

- Vektoreinträge/Teilvektoren können mittels ' [] ' entfernt werden

```
>> c(3:8)=[]
```

c =

1 2 9 10 ← end entspricht jetzt dem Eintrag 4

Vektoroperationen

- min bzw. max berechnet kleinsten bzw. größten Wert eines Vektors

```
>> a=[2 5 4]; m=min(a)  
m =  
2
```

```
>> m=max(a)  
m =  
5
```

- Die Position des kleinsten bzw. größten Eintrags liefert

```
>> [m i]=min(a)  
m =      i =  
2          1
```

```
>> [m i]=max(a)  
m =      i =  
5          2
```

- sum berechnet die Summe, prod das Produkt der Vektoreinträge

```
>> sum(a)  
ans =  
11
```

$$\hat{=} \sum_{k=1}^n a_k$$

```
>> prod(a)  
ans =  
40
```

$$\hat{=} \prod_{k=1}^n a_k$$

Vektoroperationen

- dot berechnet das Skalarprodukt zweier Vektoren $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$

```
>> a=1:5; b=6:10; d=dot(a,b)
```

```
d =
```

```
130
```

$$\hat{\equiv} (\mathbf{a}, \mathbf{b}) = \sum_{k=1}^n a_k b_k$$

- norm berechnet die Euklidische Norm ($p = 2$) eines Vektors $\mathbf{a} \in \mathbb{R}^n$

```
>> a=1:10; n=norm(a)
```

```
n =
```

```
19.6214
```

$$\hat{\equiv} \|\mathbf{a}\| = \sqrt{\sum_{k=1}^n a_k^2}$$

- weitere Normen können mittels `norm(a,p)` berechnet werden

$p = 1$	Betragssummennorm	$\ \mathbf{a}\ _1 = \sum_{k=1}^n a_k $
$p = \inf$	Maximumsnorm	$\ \mathbf{a}\ _\infty = \max_{k=1, \dots, n} a_k $
$p \in \mathbb{R}$	p -Norm, $p \geq 1$	$\ \mathbf{a}\ _p = (\sum_{k=1}^n a_k ^p)^{1/p}$

MATLAB = MATrix LABoratory

- Eine $m \times n$ Matrix ist in MATLAB ein zweidimensionales Array mit m Zeilen (engl. *rows*) und n Spalten (engl. *columns*).

- 2 \times 3 Matrix

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

```
>> A=[1 2 3; 4 5 6]
```

```
A =
```

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

Zeilen werden durch Semikolon getrennt (vgl. Spaltenvektoren)

- `>> whos A`

Name	Size	Bytes	Class	Attributes
A	2x3	48	double	

Matrixdimensionen

■ Dimension der Matrix A

```
>> s = size(A)
```

```
s =
```

```
2 3
```

```
>> [m n] = size(A)
```

```
m = n =
```

```
2 3
```

```
>> m = size(A,1)
```

```
m =
```

```
2
```

```
>> n = size(A,2)
```

```
n =
```

```
3
```

■ Größte Dimension der Matrix A

```
>> l = max(size(A))
```

```
l =
```

```
3
```

```
>> l = length(A)
```

```
l =
```

```
3
```

Spezielle Matrizen

- Nullmatrix $\gg N = \text{zeros}(2)$ $\gg N = \text{zeros}(2,3)$
 $N =$

$$\begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

- „Einsmatrix“ $\gg E = \text{ones}(2)$ $\gg E = \text{ones}(2,3)$
 $E =$

$$\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$$

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

- Einheitsmatrix $\gg I = \text{eye}(2)$ $\gg I = \text{eye}(2,3)$
 $I =$

$$\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$

- Weitere spezielle Matrizen `hilb`, `rand`, `magic`, ... → `doc elmat`

Matrixoperationen

- Sämtliche Operationen wie +, -, * und ^ werden unterstützt
- Vektoroperationen lassen sich (automatisch) spaltenweise anwenden

```
>> A=magic(3); [m i]=min(A)
```

m =

3 1 2

i =

2 1 3

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

- oder gezielt spalten- bzw. zeilenweise durch Angabe der Dimension

```
>> p=prod(A,1)
```

p =

96 45 84

```
>> p=prod(A,2)
```

p =

48

105

72

Matrixoperationen

- `norm(A,p)` berechnet die p -Norm einer Matrix A
zulässige Werte sind $p=1, 2, \inf$ und `'fro'` (Frobeniusnorm)

- `inv` berechnet die Inverse A^{-1} einer regulären Matrix A

```
>> A=hilb(2), B=inv(A), C=A*B
```

A =	B =	C =
1.0000 0.5000	4.0000 -6.0000	1 0
0.5000 0.3333	-6.0000 12.0000	0 1

- `det` und `rank` bestimmen Determinante und Rang einer Matrix A

```
>> d=det(A)
```

```
d =
```

```
0.0833
```

```
>> r=rank(A)
```

```
r =
```

```
2
```

Matrixoperationen

- `eig` berechnet die Eigenwerte einer Matrix $A \in \mathbb{R}^{n \times n}$

```
>> A=hilb(2); l=eig(A)
```

$l =$

0.0657
1.2676

$$Ax = \lambda x, \quad \lambda \in \mathbb{C}, x \in \mathbb{C}^n \setminus \{0\}$$

- Es können auch Eigenwerte und Eigenvektoren berechnet werden

```
>> [R D]=eig(A)
```

$R =$

0.4719 -0.8817
-0.8817 -0.4719

$D =$

0.0657 0
0 1.2676

Beachte: Bei der Eigenwertberechnung können Rundungsfehler auftreten.

Lineare Gleichungssysteme

- Lineare Gleichungssysteme der Form

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n \times n}, \quad \mathbf{b} \in \mathbb{R}^n$$

können mittels Linksdivision (`mldivide`)

$$\mathbf{x} = A \backslash \mathbf{b} \quad \text{oder direkt} \quad \mathbf{x} = \text{inv}(A) * \mathbf{b}$$

gelöst werden (falls Matrix A invertierbar ist)

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

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

$$\Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- `>> A=[1 2; 2 1]; b=[1; 2];`

`>> x=A\b`

`x =`

1

0

`>> x=inv(A)*b`

`x =`

1

0

- Linksdivision ist i.d.R. numerisch stabiler (\rightarrow Übungen)

Lineare Gleichungssysteme

- Lineare Gleichungssysteme der Form

$$\mathbf{x}A = \mathbf{b}, \quad A \in \mathbb{R}^{n \times n}, \quad \mathbf{b}^T \in \mathbb{R}^n$$

$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

können mittels Rechtsdivision (`mrddivide`)

$$= \begin{pmatrix} 5 & 4 \end{pmatrix}$$

$$\mathbf{x} = \mathbf{b}/\mathbf{A} \quad \text{oder direkt} \quad \mathbf{x} = \mathbf{b} * \text{inv}(\mathbf{A})$$

$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

gelöst werden (falls Matrix A invertierbar ist)

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

`>> x=b/A`

`x =`

$$\begin{matrix} 1 & 2 \end{matrix}$$

`>> x=b*inv(A)`

`x =`

$$\begin{matrix} 1 & 2 \end{matrix}$$

- Rechtsdivision ist i.d.R. numerisch stabiler

Elementweise Operationsausführung

- Operatoren $*$, $/$, \backslash , $^{\wedge}$ werden elementweise auf jeden Eintrag des Arrays angewendet, wenn ein „.” vorangestellt wird

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$$

Produkt der Matrixeinträge

```
>> A.*B
```

```
ans =
```

1	4
3	8

Division vom Typ „ b_{ij}/a_{ij} “

```
>> B./A
```

```
ans =
```

1.0000	1.0000
0.3333	0.5000

Beachte: A^k meint die k -fache Matrizenmultiplikation $A * \dots * A$ während $A.^k$ jeden Eintrag der Matrix exponenziert.

Teilmatrizen

- Matrizen können aus einzelnen Teilmatrizen aufgebaut werden

```
>> A = [1 2; 3 4]
```

```
A =
```

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

```
>> B = [A zeros(2); ...
```

```
ones(2) eye(2)]
```

```
B =
```

$$\begin{matrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{matrix}$$

- Blockmatrizen können mittels `repmat(A,m,n)` erzeugt werden

```
>> C = repmat(eye(2),2,3)
```

```
C =
```

$$\begin{matrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{matrix}$$

```
>> D = repmat([1; 2],2)
```

```
D =
```

$$\begin{matrix} 1 & 1 \\ 2 & 2 \\ 1 & 1 \\ 2 & 2 \end{matrix}$$

Matrixumformungen

- Matrizen können mittels `reshape(x,m,n)` verändert werden

```
>> A=reshape(9:-1:1,3,3)
```

```
A =
```

9	6	3
8	5	2
7	4	1

i Matrizen werden

spaltenweise gespeichert

`A(:)=9 8 7 6 5 4 3 2 1`

(Der Zugriff `A(4)` ist also erlaubt)

- Matrizen können mittels `transpose(A)` transponiert werden

```
>> B=transpose(A)
```

```
B =
```

9	8	7
6	5	4
3	2	1

i Kurznotation `A.'`

`ctranspose(A)` berechnet
die komplex Konjugierte \bar{A}
Kurznotation `A'`

Diagonalmatrizen

- Diagonalteil einer Matrix kann mittels `diag(A,k)` bestimmt werden

```
>> A=reshape(1:9,3,3)    >> B=diag(A)          >> C=diag(A,1)  
A =                                B =                                C =  
 1   4   7           1                               4  
 2   5   8           5                               8  
 3   6   9           9
```

- Diagonalmatrizen können mittels `diag(v,k)` erzeugt werden

```
>> D=diag(1:3)      >> E=diag(1:2,-1)    >> F=diag(diag(A))  
D =                                E =                                F =  
 1   0   0           0   0   0           1   0   0  
 0   2   0           1   0   0           0   5   0  
 0   0   3           0   2   0           0   0   9
```

Dreiecksmatrizen

- Die linke untere bzw. rechte obere Dreiecksmatrix von

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

kann mittels `tril(A,k)` bzw. `triu(A,k)` bestimmt werden

<code>>> L=tril(A)</code>	<code>>> U=triu(A)</code>	<code>>> T=tril(A,-1)</code>
$L = \begin{matrix} 1 & 0 & 0 \\ 2 & 5 & 0 \\ 3 & 6 & 9 \end{matrix}$	$U = \begin{matrix} 1 & 4 & 7 \\ 0 & 5 & 8 \\ 0 & 0 & 9 \end{matrix}$	$T = \begin{matrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 6 & 0 \end{matrix}$

- Es gilt $A \equiv \text{tril}(A) + \text{triu}(A) - \text{diag}(\text{diag}(A))$
und $A \equiv \text{tril}(A,-1) + \text{triu}(A,1) + \text{diag}(\text{diag}(A))$

Laden und Speichern von Variablen

- Inhalt des Workspaces kann mittels save gespeichert werden

```
>> save
```

```
Saving to: matlab.mat
```

```
>> save Dateiname[.Ext]
```

```
Standardendung ist .mat
```

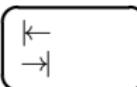
- save Dateiname x y z gespeichert nur die Variablen x, y und z
- save Dateiname Var* speichert alle Variablen mit Namen Var...
- Option -ascii speichert Daten als plattformunabhängige Textdatei
- Option -mat speichert Daten als Binärdatei (Standardformat)
- load liest gespeicherte Variablen aus einer Datei in den Workspace

Speichern von Befehlsvorschriften

Demo-Übungsaufgabe

- Erzeugen Sie eine Hilbertmatrix H der Dimension 10, sowie den Spaltenvektor $b = (1, \dots, 1)^T \in \mathbb{R}^{10}$
- Lösen Sie das Gleichungssystem $Hx = b$ indem Sie
 - i) die Hilbertmatrix invertieren, also $x = H^{-1}b$ berechnen, und
 - ii) den MATLAB®-Befehl $y=H\backslash b$ verwenden
- Welche der beiden Berechnungen ist genauer? Berechnen Sie hierfür die Normen der jeweiligen Residuen $Hx - b$ bzw. $Hy - b$ und vergleichen Sie diese!

Nützliche Tipps

-  wiederholt vorherige Befehle
-  vervollständigt bekannte Befehle/Variablen
- reelle Zahlen werden standardmäßig im Format `short` (vier Nachkommastellen) ausgegeben

`>> x=1.23456`

`x =`

`1.2346`

Überblick 2. Vorlesung

- M-Dateien: Skripte und Funktionen
- Sichtbarkeit von Variablen
- Vergleichsoperatoren und -funktionen
- Logische Operatoren
- Verzweigungen und Schleifen

M-Dateien

- Dateiendung `.m` kennzeichnet eine ausführbare Datei
- Algorithmen können in einer M-Datei editiert/gespeichert und mehrfach (mit unterschiedlichen Daten) aufgerufen werden
- <http://www.mathworks.com/matlabcentral/fileexchange/>
- **Skripte** besitzen keine Ein- und Ausgabeparameter und arbeiten mit den global im Workspace vorhandenen Variablen
- **Funktionen** besitzen Ein- und Ausgabeparameter und arbeiten intern mit lokalen Variablen, die automatisch gelöscht werden
- M-Dateien **müssen** eine gute Dokumentation enthalten ;-)

Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A.

M-Datei: maxentry.m

```
%MAXENTRY Betragsmäßig größter Matrixeintrag  
% MAXENTRY berechnet betragsmäßig größten Wert von A  
  
B=abs(A); m=max(B); max(m)
```

- Kurzbeschreibung in H1-Zeile wird von help, lookfor verwendet
- Hilfetext endet vor der ersten Zeile ohne Kommentarzeichen „%“
- Globale Variablen B und m bleiben im Workspace erhalten

Funktionen

Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

M-Datei: maxentry1.m

```
function y = maxentry1(A)
%MAXENTRY1    Betragsmäßig größter Matrixeintrag
% MAXENTRY1 berechnet betragsmäßig größten Wert von A
B=abs(A); m=max(B); y=max(m);
```

- Funktions- und Dateiname müssen übereinstimmen
- Deklaration `function` Ausgabe = `name`(Eingabe)
- Variablen `B` und `m` werden nach Verlassen der Funktion gelöscht

Funktionen mit mehreren Ausgabeparametern

Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A und bestimme dessen Zeilen- und Spaltennummer.

M-Datei: maxentry2.m

```
function [y i j] = maxentry2(A)
%MAXENTRY2    Betragsmäßig größter Matrixeintrag
% MAXENTRY2 berechnet betragsmäßig größten Wert von A
[x k] = max(abs(A));
[y j] = max(x);
i      = k(j);
```

- Beim Aufruf können Ausgabeparameter von rechts nach links weggelassen werden, z.B. entfällt j bei $[y \ i] = \text{maxentry2}(A)$

Sichtbarkeit von Variablen

- Variablen in Skripten sind **global** und verbleiben im Workspace
- In Funktionen definierte Variablen sind **lokal**, d.h. sie sind außerhalb der Funktion nicht sichtbar, und werden automatisch gelöscht

Workspace

- definiere Variablen `a,b,c` im Workspace

Skript

- sieht Variablen `a,b,c`
- definiere Variable `x`

Funktion

- sieht Variablen `a,b,c,x`
- definiere Variable `y`

- Variablen `a,b,c,x` im Workspace vorhanden

Vergleichsoperatoren

- Logische Variablen haben den Wert 1 (=wahr) oder 0 (=falsch)

```
>> a=1, b=2, c=a==b
```

```
a =      b =      c =  
1          2          0
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x1	8	double
b	1x1	8	double
c	1x1	1	logical

=	==	gleich
≠	~=	ungleich
>	>	größer
≥	≥=	größer gleich
<	<	kleiner
≤	≤=	kleiner gleich

Vergleichsoperatoren

- Neben double wird der Datentyp logical unterstützt
- Zuweisungsoperator ($c=42$) ist **kein** Vergleichsoperator ($a==b$)

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (elementweiser Vergleich)

```
>> diag([1; 1])==0  
ans =  
0     1  
1     0
```

```
>> ones(2)==1  
ans =  
1     1  
1     1
```

- Vergleich zwischen Matrizen **gleicher** Größe

```
>> diag([1; 1])==zeros(2)  
ans =  
0     1  
1     0
```

```
>> ones(3)==ones(2)  
??? Error using ==> eq  
Matrix dimensions must  
agree.
```

- Vergleichsfunktionen **is*** für **beliebige** Variablen

```
>> isequal(diag([1; 1]),...  
           zeros(2))  
ans =  
0
```

```
>> isequal(ones(3),...  
           ones(2))  
ans =  
0
```

Übersicht über Vergleichsfunktionen

■ Vergleichsfunktionen mit skalarem Rückgabewert (doc `is`)

<code>isequal(A,B,...)</code>	Test, ob A , B , etc. identisch sind
<code>isempty(A)</code>	Test, ob A die leere Matrix [] ist
<code>islogical(A)</code>	Test, ob A vom Typ logical ist
<code>isnumeric(A)</code>	Test, ob A ein Zahlwert ist
<code>isscalar(A)</code>	Test, ob A ein Skalar (1×1) ist
<code>isvector(A)</code>	Test, ob A ein Vektor ($1 \times n$, $n \times 1$) ist

■ Vergleichsfunktionen mit mehrdimensionalem Rückgabewert

<code>isfinite(A)</code>	Test auf endliche Matrixeinträge
<code>isinf(A)</code>	Test auf unendliche Matrixeinträge vom Typ Inf
<code>isnan(A)</code>	Test auf unzulässige Matrixeinträge vom Typ NaN

■ Vergleich `x==NaN` ist falsch, selbst wenn `x` „not a number“ ist

```
>> 0.0/0.0==NaN  
ans = 0
```

```
>> isnan(0.0/0.0)  
ans = 1
```

Logische Operatoren

- Logische Operatoren (doc relop)

&	logisches Und
	logisches Oder
~	logisches Nicht
xor	logisches exklusives Oder

0 1 0 1	0 1 0 1
& 0 0 1 1	0 0 1 1
= 0 0 0 1	= 0 1 1 1
	0 1 0 1
	xor 0 0 1 1
	= 0 1 1 0

- „short circuit“ Varianten `&&` bzw. `||` für skalare Ausdrücke

- **Effizienz:** `Ausdruck1 || Ausdruck2`

- wenn `Ausdruck1` wahr ist, dann wird `Ausdruck2` nicht ausgewertet

- **Fehlererkennung:** `x>0 && log(1/x)<=1`

- verhindert Division durch 0 sowie Logarithmusberechnung für $x \leq 0$

Logische Operatoren

- Logische Operatoren all bzw. any für Vektoren und Matrizen
 - **Mengen-Und:** ist wahr, wenn **alle** Elemente wahr sind
 - `all([1 1 1]==1)` ist wahr
 - `all([1 2 3]==1)` ist nicht wahr
 - **Mengen-Oder:** ist wahr, wenn **ein** (beliebiges) Element wahr ist
 - `any([1 2 3]==1)` ist wahr
 - `any([1 2 3]==0)` ist nicht wahr
- all und any können auf Teilvektoren/-matrizen angewendet werden
`>> v=1:10; all(v(1:5)<=5)` ist **wahr**

find Funktion

- `k=find(Ausdruck)` findet alle Indizes k des Vektors v , für die die im *Ausdruck* definierte Bedingung wahr ist

```
>> v=1:10; k=find(v<=5)  
k =  
1 2 3 4 5
```

```
>> k=find(mod(v,2)==0)           mod elementweise zu betrachten  
k =  
2 4 6 8 10
```

```
>> v=1:10; k=find(v<=5 & mod(v,2)==0)  
k =  
2 4
```

find Funktion

- $[i \ j \ s] = \text{find}(\text{Ausdruck})$ findet alle Indizes (i, j) der Matrix A , für die die im *Ausdruck* definierte Bedingung wahr ist und liefert die entsprechenden Werte in s

```
>> [i j]=find(ones(2)==diag([1 1]))  
i =           j =  
    1   2           1   2
```

- die absoluten Positionen $k = \text{ncols} \times (j - 1) + i$ liefert

```
>> k=find([1 2 3; 1 2 3; 1 2 3]==ones(3))  
k =  
    1   2   3   ← spaltenweise Nummerierung der Matrixeinträge!
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} == \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Anwendung der find Funktion

- Berechnung auf Teilvektoren bzw. Teilmatrizen einschränken

```
>> v=[0 -4 2 6]; k=find(v>0); w=log(v(k))  
w =  
0.6931 1.7918
```

- Teilvektoren bzw. Teilmatrizen auswählen

```
>> A=rand(10); k=find(A<=0.8); l=length(k)  
l =  
83
```

- Alle Nicht-Nulleinträge auswählen

```
>> A=diag(10:20); [i j s]=find(A)  
i = 1 2 ... j = 1 2 ... s = 10 11 ...
```

Bedingte Verzweigung

Aufgabe: Überprüfe ob die Variable x gerade oder ungerade ist.

```
if mod(x,2) == 0  
    disp('Variable x ist gerade.')  
else  
    disp('Variable x ist ungerade.')  
end
```

```
if Ausdruck  
    Befehlsfolge1  
else  
    Befehlsfolge2  
end
```

- Mehrere if Abfragen können ineinander geschachtelt werden
- Es wird immer genau ein Zweig der if Abfrage ausgeführt

Bedingte Verzweigung

Aufgabe: Überprüfe ob die Variable x gerade, ungerade oder Null ist.

```
if x == 0
    disp('Variable x ist Null.')
elseif mod(x,2) == 0
    disp('Variable x ist gerade.')
else
    disp('Variable x ist ungerade.')
end
```

```
if Ausdruck1
    Befehlsfolge1
elseif Ausdruck2
    Befehlsfolge2
else
    Befehlsfolge3
end
```

- Nach einer if Abfrage können beliebig viele elseif Zweige folgen
- else Zweig wird ausgeführt, wenn keine Bedingung erfüllt ist

Auswahlabfrage

Aufgabe: Gib für jeden Monat die zugehörige Quartalszahl an.

```
switch x
case {1,2,3}
    q = 1;
case {4,5,6}
    q = 2;
...
otherwise
    disp('Ungültiger Monat.')
end
```

```
switch Ausdruck
case Ausdruck1
    Befehlsfolge1
case {Ausdruck2}
    Befehlsfolge2
...
otherwise
    Befehlsfolge
end
```

- Erster zutreffender case Abschnitt oder otherwise wird ausgeführt
- Hinweis für C-Programmierer: break ist nicht erforderlich

Gebrauch von if und switch case

- Ausdrücke können von weiteren Variablen abhängen

```
if x==sin(y)                      switch x
    disp('x ist gleich sin(y)');      case sin(y)
end                                ...
...
```

- if Anweisungen können in einer Zeile geschrieben werden

```
if x==sin(y), disp('x ist gleich sin(y)'); end
```

- case Anweisungen können einen einzelnen Ausdruck

```
case 1
```

oder eine Liste von mehreren Ausdrücken enthalten

```
case {inf,-inf}
```

Schleifen

- Schleifen dienen zur wiederholten Ausführung von Befehlen
- Einfachste Schleife ist der Doppelpunktoperator
 $A=1:5$ ist Kurzform für $A(1)=1, A(2)=2, A(3)=3, \dots$
- Es gibt zwei unterschiedliche Schleifenarten
 - **for** Schleifen mit **fester** Anzahl an Wiederholungen
 - **while** Schleifen mit **variabler** Anzahl an Wiederholungen
- Schleifen (auch unterschiedliche) können geschachtelt werden
- Programmierfehler können zu Endlosschleifen führen, die mit der Tastenkombination Strg+C abgebrochen werden können

Schleifen mit fester Anzahl an Wiederholungen

Aufgabe: Berechne **für** 100 Zufallszahlen zwischen 0 und 1
die Anzahl der Zufallszahlen kleiner als 0.8.

```
z=0; for i=1:100
    if rand(1)<0.8, z=z+1; end
end
```

```
for Variable=Ausdruck
    Befehlsfolge
end
```

- Schleifen über Wertelisten sind möglich

$$\text{for } x=[\pi \ \pi/2 \ \pi/4] \rightarrow x^{(1)} = \pi, \quad x^{(2)} = \frac{1}{2}\pi, \quad x^{(3)} = \frac{1}{4}\pi$$

- Schleifen über Vektoren sind möglich

$$\text{for } x=\text{eye}(3) \rightarrow x^{(1)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad x^{(2)} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad x^{(3)} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Schleifen mit fester Anzahl an Wiederholungen

- for Schleifen können mittels break vorzeitig verlassen werden

M-Datei: liesganzzahlen.m

```
function z = liesganzzahlen(n)

z=[];
for k=1:n
    i=input('Ganzzahl eingeben:'); % Benutzereingabe
    if fix(i)-i ~= 0, % auf Ganzzahl runden
        disp('Zahl ist keine Ganzzahl'); break
    else
        if isempty(z),
            z = i; % erste Ganzzahl setzen
        else
            z = [z i]; % Ganzzahl hinzufügen
        end
    end
end
```

Schleifen mit variabler Anzahl an Wdhlgcn

Aufgabe: Berechne **solange** Zufallszahlen zwischen 0 und 1 bis eine Zahl größer oder gleich 0.8 auftritt.

```
z=0; while rand(1)<0.8  
    z=z+1;  
end
```

```
while Ausdruck  
    Befehlsfolge  
end
```

- while Schleifen werden wiederholt solange *Ausdruck* wahr ist
- while 1, ..., end erzeugt eine Endlosschleife
- Endlosschleifen *müssen* mittels break verlassen werden

Tipps zum Umgang mit Schleifen

1. Tip: while Schleifen können zwei unterschiedliche Formen haben.

```
i=0;  
while i<n  
    i=i+1;  
end
```



```
j=0;  
while 1  
    j=j+1;  
    if j>=n, break, end  
end
```

- Abbruchbedingung kann **zu Beginn** (vorprüfend) oder **am Ende** (nachprüfend) des Schleifendurchlaufs überprüft werden
- Hinweis für Programmierer: es gibt keine REPEAT-UNTIL Schleifen; verwende als Alternative `while 1, ..., end`

Tipps zum Umgang mit Schleifen

2. Tip: Schleifen können oft durch Arrayoperationen ersetzt werden
(Stichwort: Vektorisieren).

- Beispiel: Erzeugung der Hilbertmatrix $H = \{h_{ij}\}$, $h_{ij} = \frac{1}{i+j-1}$

```
for i=1:n
    for j=1:n
        h(i,j)=1/(i+j-1);
    end
end
```



```
I=repmat(1:n,n,1);
H=ones(n)./(I+I'-1);
```

- Arrayoperationen sind übersichtlicher und bereits ausgetestet
- Arrayoperationen sind manchmal effizienter (Built-In Funktionen)

Tipps zum Umgang mit Schleifen

- Gauss Elimination (T. Driscoll, Learning MATLAB, SIAM, '09)

```
n=length(A);
for k=1:n-1
    for i=k+1:n
        s=A(i,k)/A(k,k);
        for j=k:n
            A(i,j)=A(i,j)-s*A(k,j);
        end
    end
end
```



```
n=length(A);
for k=1:n-1
    row=k+1:n;
    col=k:n;
    s=A(row,k)/A(k,k);
    A(row,col)=...
        A(row,col)-s*A(k,col);
end
```

- Eine geschickte Implementierung kommt auch mit nur **einer** for-Schleife aus.

CPU Zeit	n=200	400	600	800	1000
3 Schleifen	0.18	1.57	5.64	14.05	27.36
1 Schleife	0.02	0.17	0.64	0.88	3.15

Rekursive Funktionen

Aufgabe: Schreibe eine Funktion zur Berechnung von $f(n) = n!$.

```
function f = fakultaet(n)
if n==1
    f = 1;
else
    f = n * fakultaet(n-1);
end
```

- Rekursionen können in manchen Situationen vermieden werden
 - 1 $f=1; \text{ for } i=1:n,$
 $f=f*i;$ end
 - 2 $f = \text{prod}(1:n);$

- Rekursive Funktionen brauchen eine **Abbruchbedingung**
- Rekursive Funktionen können sich selbst mehrfach aufrufen
(MATLABs Rekursionslimit liegt bei 500)

Beispiel: Fibonacci-Funktion: $\text{Fib}(1) = 1, \quad \text{Fib}(0) = 0$
 $\text{Fib}(x) := \text{Fib}(x - 1) + \text{Fib}(x - 2), \quad \text{für } x \geq 2$

Nützliche Tipps

- **Strg** + **C** bricht die aktuelle Berechnung ab
- `disp('Hallo')` schreibt die Ausgabe *Hallo* in die Kommandozeile (nützlich für eine kommentierte Ausgabe)
- `clc` löscht alle Eingaben/Ausgaben im Kommandofenster
- mit ... kann ein Code in der nächsten Zeile fortgesetzt werden
- `return` beendet die Ausführung einer Funktion (Rückgabewert ist u.U. dann nicht definiert)
- `NaN` (Not a Number) repräsentiert einen undefinierten Wert
`>> 0/0`
`ans =`
`NaN`
- mit Hilfe des Befehls `t=cputime; <Befehle>; cputime-t` wird die benötigte CPU Zeit angezeigt

Überblick 3. Vorlesung

- Unterfunktionen
- Übergabe von Funktionen
- Anonyme Funktionen
- Zeichenketten
- Grafikausgabe in 2D

Unterfunktionen

- Berechne für die Funktion

$$f(x) = \sin(x)^2$$

die erste Ableitung

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

mit Parameter $h > 0$

```
>> fderiv(pi/4, 1e-5)
ans =
1.0000
```

M-Datei: fderiv.m

```
function y = fderiv(x,h)
%FDERIV Berechnet f'(x)
y = (f(x+h) - f(x))/h;
```

```
function w = f(x)
%F Berechnet sin(x)^2
w = sin(x)^2;
```

- Unterfunktion `f` ist nur **innerhalb** der Funktion `fderiv` sichtbar
- Unterfunktion `f` kann von **außen** nicht verändert werden :-(

Übergabe von Funktionen

Aufgabe: Schreibe eine Funktion, die für eine beliebige Funktion $f(x)$ deren Ableitung $f'(x)$ approximiert.

M-Datei: fderiv1.m

```
function y = fderiv1(f,x,h)
y = (feval(f,x+h) - feval(f,x))/h;
```

- Funktionsauswertung mittels `feval(fun,x1,x2,...,xn)`
- Aufruf für sin Funktion mittels `fderiv1('sin',pi,1e-5)`
- Übergabeparameter `'Funktion'` als Zeichenkette
- Aufruf `fderiv1(sin,pi,1e-5)` erzeugt Fehlermeldung

Function Handles (ab MATLAB 7)

Aufgabe: Schreibe eine Funktion, die für eine beliebige Funktion $f(x)$ deren Ableitung $f'(x)$ approximiert.

M-Datei: fderiv2.m

```
function y = fderiv2(f,x,h)
f = fcncchk(f); Diese Funktion fehlt in Octave!!!
y = (f(x+h) - f(x))/h;      % Function Handle
```

- Aufruf für sin Funktion mittels `fderiv2(@sin,pi,1e-5)`
- Aufruf `fderiv2('sin',pi,1e-5)` erzeugt Fehlermeldung
- `fcncchk` konvertiert eine **Funktion** in ein **Function Handle**

Function Handles (in GNU Octave)

- Konvertierung einer **Funktion** in ein **Function Handle** ist in Octave nicht erforderlich, d.h. der Aufruf `fderiv2('sin',pi,1e-5)` ohne die Zeile `f = fcnchk(f)` erzeugt **keine** Fehlermeldung
- Um kompatiblen Code für MATLAB **und** Octave zu schreiben, kann die folgende Hilfsfunktion in Octave benutzt werden

M-Datei: `fcnchk.m`

```
function f=fcnchk(x, n)
    f = x;
end
```

Übergabe von Funktionen

- Beispiefunktion $p(x) = x^3$

M-Datei: polynom.m

```
function y = polynom(x)
y = x^3;
```

- Approximation der Ableitung $p'(x) = 3x^2$ im Punkt $x = 4$

- `fderiv1('polynom',4,1e-5);` 'polynom' wird mittels feval ausgewertet
- `fderiv2(@polynom,4,1e-5);` Aufruf mit Function Handle
- `fderiv2('polynom',4,1e-5);` 'polynom' wird mittels fcncchk in Function Handle konvertiert

Ausgabe in allen Fällen $\text{ans} = 48.0001 \leftarrow p'(4) = 48$

Anonyme Funktionen

- Einfache Definition von „einzeiligen“ Funktionen

```
>> p = @(x) x^3;  
>> p(4)  
ans = 64.0000
```

Allgemeine Form

$f = \text{@(Variablen) Funktion}$

- Approximation der Ableitung $p'(x) = 3x^2$ im Punkt $x = 4$

- ```
fderiv2(p,4,1e-5);
fderiv2(@(x) x^3,4,1e-5);
```

- Anonyme Funktionen können von mehreren Variablen abhängen

```
>> f = @(x,y,z) x^3+y^2+x*z-y*z-z^3;
```

# Anonyme Funktionen

---

**Aufgabe:** Berechne die Approximation der zweiten Ableitung  $p''(x) = 6x$ .

- 1 Erstelle anonyme Funktion zur Berechnung der ersten Ableitung in  $x$

```
>> p1 = @(x) fderiv2(@(x) x^3,x,1e-5)
```

- 2 Berechne mit Hilfe des Function Handles p1 die zweite Ableitung

```
>> p2 = fderiv2(p1,4,1e-5)
```

```
p2 = 24.0000
```

- Kombination der Schritte 1 und 2 mittels anonymer Funktionen

```
>> fderiv2(@(x) fderiv2(@(x) x^3,x,1e-5),4,1e-5)
```

# Zeichenketten

---

- Zeichen(ketten) werden durch Anführungszeichen erzeugt

```
>> zeichen='a'; zeichenkette='abc';
```

- Zeichenketten können zusammengesetzt werden

```
>> zk1=['Dies ', 'ist ', 'eine ', 'Zeichenkette']
zk1 =
```

```
 Dies ist eine Zeichenkette
```

```
>> zk2=strcat('Eine ', 'zweite ', 'Zeichenkette')
zk2 =
```

```
 Eine zweite Zeichenkette
```

```
>> zk3=strvcat('Eine', 'dritte', 'Zeichenkette')
zk3 =
```

```
 Eine
```

```
 dritte
```

```
 Zeichenkette
```

# Ersetzungen in Zeichenketten

---

- upper und lower konvertieren zwischen Groß- und Kleinbuchstaben

```
>> upper('gross')
ans = GROSS
```

```
>> lower('KLEIN')
ans = klein
```

- strrep ersetzt Teile einer Zeichenkette durch andere Zeichen

```
>> strrep('Deckfehler', 'eck', 'uck')
ans = Druckfehler
```

- char und double wandeln zwischen ASCII-Code und Zeichen um

```
>> char([97 98 99])
ans = abc
```

```
>> double('abc')
ans = 97 98 99
```

Datentyp char!

Datentyp double!

# Der ASCII-Code

```
>> char([97 98 99])
ans = abc
```

```
>> double('abc')
ans = 97 98 99
```

| Scan-code | ASCII hex dez | Zeichen  | Scan-code | ASCII hex dez | Zch. | Scan-code | ASCII hex dez | Zch. | Scan-code | ASCII hex dez | Zch. |     |     |
|-----------|---------------|----------|-----------|---------------|------|-----------|---------------|------|-----------|---------------|------|-----|-----|
| 00        | 0             | NUL ^@   | 20        | 32            | SP   | 40        | 64            | @    | 0D        | 60            | 96   |     |     |
| 01        | 1             | SOH ^A   | 02        | 21            | 33   | !         | 41            | 65   | A         | 1E            | 61   | 97  |     |
| 02        | 2             | STX ^B   | 03        | 22            | 34   | "         | 42            | 66   | B         | 30            | 62   | 98  |     |
| 03        | 3             | ETX ^C   | 29        | 23            | 35   | #         | 43            | 67   | C         | 2E            | 63   | 99  |     |
| 04        | 4             | EOT ^D   | 05        | 24            | 36   | \$        | 44            | 68   | D         | 20            | 64   | 100 |     |
| 05        | 5             | ENQ ^E   | 06        | 25            | 37   | %         | 45            | 69   | E         | 12            | 65   | 101 |     |
| 06        | 6             | ACK ^F   | 07        | 26            | 38   | &         | 46            | 70   | F         | 21            | 66   | 102 |     |
| 07        | 7             | BEL ^G   | 0D        | 27            | 39   | '         | 47            | 71   | G         | 22            | 67   | 103 |     |
| 0E        | 8             | BS ^H    | 09        | 28            | 40   | (         | 48            | 72   | H         | 23            | 68   | 104 |     |
| 0F        | 9             | TAB ^I   | 0A        | 29            | 41   | )         | 49            | 73   | I         | 17            | 69   | 105 |     |
| 0A        | 10            | LF ^J    | 1B        | 2A            | 42   | *         | 4A            | 74   | J         | 24            | 6A   | 106 |     |
| 0B        | 11            | VT ^K    | 1B        | 2B            | 43   | +         | 4B            | 75   | K         | 25            | 6B   | 107 |     |
| 0C        | 12            | FF ^L    | 33        | 2C            | 44   | ,         | 4C            | 76   | L         | 26            | 6C   | 108 |     |
| 1C        | 0D            | 13 CR ^M | 35        | 2D            | 45   | -         | 4D            | 77   | M         | 32            | 6D   | 109 |     |
| 4C        | 0D            | 43 CB ^N | 32        | 5D            | 42   | -         | 4D            | 77   | M         | 35            | 6D   | 109 |     |
| 0C        | 43            | EE ^P    | 33        | 5C            | 44   | ,         | 50            | 4C   | 78        | P             | 50   | 6C  | 108 |
| 0B        | 44            | AL ^K    | 4B        | 5B            | 43   | +         | 52            | 4B   | 72        | K             | 52   | 6B  | 107 |
| 0A        | 40            | FE ^U    | 4B        | 5A            | 45   | *         | 54            | 4C   | 71        | U             | 54   | 6C  | 108 |

# Vergleiche von Zeichenketten

---

- strcmp vergleicht zwei Zeichenketten miteinander

```
>> strcmp('Matlab','Maple')
ans = 0
```

- strncmp vergleicht die ersten  $n$  Zeichen zweier Zeichenketten

```
>> strncmp('Matlab','Maple',2)
ans = 1
```

- strfind sucht in der ersten Zeichenkette nach der zweiten Zeichenkette und gibt deren Startindex bzw. [] zurück

```
>> strfind('Matlab the matrix laboratory','mat')
ans = 12
```

- strtok gibt den ersten Token der Zeichenkette zurück

```
>> strtok('Matlab!Maple!MuPAD',' !')
ans = Matlab
```

# Zeichenketten für Benutzerinteraktion nutzen

---

- input fordert den Benutzer zur Eingabe auf

```
>> s=input('Bitte eine Zahl eingeben:', 's')
```

Bitte eine Zahl eingeben:**42**

s = 42

← vom Typ char

- disp gibt eine Zeichenkette aus und führt das Programm fort

```
>> disp('Dies ist nur eine Information.')
```

Dies ist nur eine Information.

- warning gibt eine Warnung aus und führt das Programm fort

```
>> warning('Es wurde durch Null geteilt.')
```

**Warning: Es wurde durch Null geteilt.**

- error gibt eine Fehlermeldung aus und beendet den Programmablauf

```
>> error('Gleichungssystem ist nicht loesbar.')
```

**Error: Gleichungssystem ist nicht loesbar.**

# Konvertieren zwischen Zeichenketten und Zahlwerten

---

- num2str konvertiert einen Zahlwert in eine Zeichenkette

```
>> a=num2str(42), b=num2str(pi)
a = 42 b = 3.1416 ← vom Typ char
```

- Beispiel

```
>> disp(['Die gesuchte Antwort lautet ', num2str(42)])
Die gesuchte Antwort lautet 42
```

- str2num konvertiert eine Zeichenkette in einen Zahlwert

```
>> c=str2num('42.0'), d=str2num='1.5e3'
c = 42 d = 1500 ← vom Typ double
```

- **Achtung:** Beim Konvertieren gehen u.U. Nachkommastellen verloren

```
>> isequal(1/3, str2num(num2str(1/3)))
ans = 0 ← vom Typ logical
```

# Konvertieren zwischen Zeichenketten und Zahlwerten

---

- Beim Konvertieren von Zahlwerten in Zeichenketten kann die Anzahl der Nachkommastellen explizit festgelegt werden

```
>> a=num2str(pi,5), b=num2str(pi,10), c=num2str(pi,16)
a = 3.1416 b = 3.141592654 c = 3.141592653589793
```

- Beispiel

```
>> isequal(1/3, str2num(num2str(1/3,10)))
ans = 0

>> isequal(1/3, str2num(num2str(1/3,16)))
ans = 1
```

- Matlab betrachtet zwei Zahlwerte (vom Typ double) als gleich, wenn sie bis auf 15 Nachkommastellen übereinstimmen

# Konvertieren zwischen Zeichenketten und Zahlwerten

---

- Beim Konvertieren von Zahlwerten in Zeichenketten kann das Format des Zahlwerts explizit festgelegt werden

```
>> a=num2str(zahl,formatzeichenkette)
```

- Ganzzahlen in dezimaler bzw. hexadezimaler Darstellung

|                     |                     |
|---------------------|---------------------|
| >> num2str(42,'%d') | >> num2str(42,'%x') |
| 42                  | 2a                  |

- Zahldarstellung mit fester Anzahl an Nachkommastellen

|                      |                         |
|----------------------|-------------------------|
| >> num2str(1/3,'%f') | >> num2str(1/3,'%6.4f') |
| 0.333333             | 0.3333                  |

- Exponentaldarstellung

|                      |                         |
|----------------------|-------------------------|
| >> num2str(1/3,'%e') | >> num2str(1/3,'%6.4e') |
| 3.333333e-01         | 3.3333e-01              |

- Automatische Wahl der Darstellung

|                      |                           |
|----------------------|---------------------------|
| >> num2str(1/3,'%g') | >> num2str(0.000001,'%g') |
| 0.333333             | 1e-06                     |

# Grafikausgabe in 2D

---

**Aufgabe:** Stelle folgende Daten grafisch dar

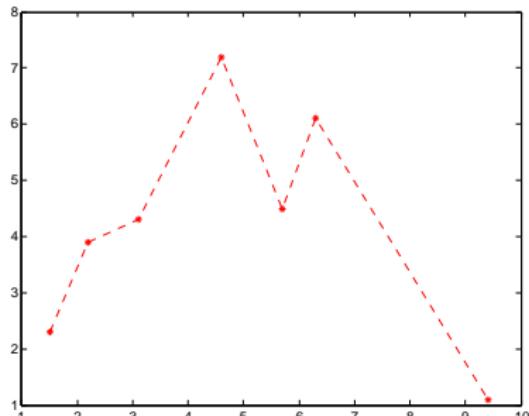
|   |     |     |     |     |     |     |     |
|---|-----|-----|-----|-----|-----|-----|-----|
| x | 1.5 | 2.2 | 3.1 | 4.6 | 5.7 | 6.3 | 9.4 |
| y | 2.3 | 3.9 | 4.3 | 7.2 | 4.5 | 6.1 | 1.1 |

- $xy$ -Diagramme lassen sich mittels `plot(x, y)` Befehl erzeugen
- `plot` Befehl kann mehrere Datensätze gleichzeitig anzeigen
  - `plot(x1, y1, x2, y2, ...)` mit Vektoren  $x_1$ ,  $y_1$ ,  $x_2$  und  $y_2$
  - `plot(x, Y)` mit  $m$ -Vektor  $x$  und  $m \times n$ -Matrix  $Y$
  - `plot(X, Y)` mit  $m \times n$ -Matrizen  $X$  und  $Y$
- Darstellung kann durch optionale Parameter beeinflusst werden

# Optionen des plot Befehls

---

```
>> plot(x,y,'rp--')
```



plot(x,y,*Zeichenkette*)

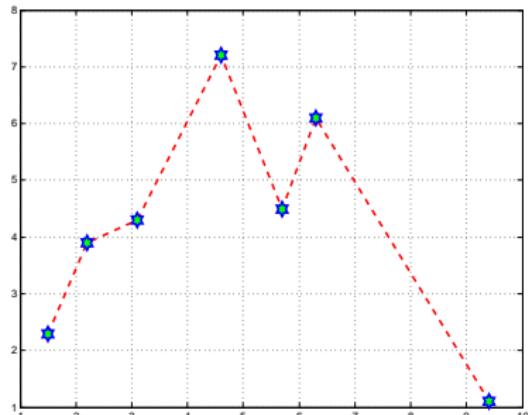
|   |          |
|---|----------|
| r | Rot      |
| g | Grün     |
| b | Blau     |
| c | Blaugrün |
| m | Pink     |
| y | Gelb     |
| k | Schwarz  |
| w | Weiß     |

|    |              |
|----|--------------|
| -  | durchgezogen |
| -- | gestrichelt  |
| :  | gepunktet    |
| -. | abwechselnd  |

|   |   |
|---|---|
| o | o |
| * | * |
| . | . |
| + | + |
| x | x |
| s | □ |
| d | ◊ |
| ^ | △ |
| v | ▽ |
| > | ◀ |
| < | ▶ |
| p | ★ |
| h | ★ |

# Weiterführende Optionen des plot Befehls

---



```
>> plot(x,y,'rh--',...
 'LineWidth',2.0,...
 'MarkerSize',15,...
 'MarkerEdgeColor','b',...
 'MarkerFaceColor','g')
```

- Einstellungen können im *Property Editor* vorgenommen werden  
Menüeintrag: Tools → Edit Plot
- Manuelle Einstellungen können als M-Datei exportiert werden  
Menüeintrag: File → Generate M-File (Generate Code)
- `createfigure(x1,y1)` wendet generierte M-Datei auf x1, y1 an

# Anpassen von Diagrammen

---

- `title('Überschrift')` erzeugt eine Überschrift
- Gitterlinien können mittels `grid` Befehl eingeblendet werden
  - `grid on|off` blendet Gitterlinien ein bzw. aus
  - `grid minor` schaltet zwischen Gitterlinienauflösungen um
- `legend` Befehl fügt eine Abbildungslegende ein
  - `legend('Eintrag1', 'Eintrag2', Optionen)`
  - Position, Ausrichtung, Umrahmung optional festlegbar
- `hold on` erlaubt Einfügen in ein bestehendes Diagramm
- `clf (clear figure)` löscht ein bestehendes Diagramm
- `close` schließt das aktuell ausgewählte Diagramm, `close all` schließt alle Diagramme

# Anpassen der Diagrammachsen

---

- Achsen können linear oder logarithmisch skaliert werden  
`plot, semilogx, semilogy, loglog`
- Achsen können mittels `xlabel`, `ylabel` beschriftet werden  
`xlabel('x-Achse'), ylabel('y-Achse')`
- Achsengröße kann mittels `axis`, `xlim`, `ylim` festgelegt werden
  - `axis([xmin xmax ymin ymax])`
  - `xlim([xmin xmax]), ylim([ymin ymax])`
  - `±inf` berechnet Grenze(n) automatisch
- Achsenverhältnis kann mittels `axis` angepasst werden  
`axis auto, equal, square, tight, off`
- Bearbeitung der Achseneinstellungen im *Property Editor*

# Speichern und Drucken von Diagrammen

---

- Diagramme können in einem MATLAB/Octave-eigenem Format (fig) gespeichert werden, um diese später zu editieren

Menüeintrag: File → Save As (fig)

oder mit Hilfe des saveas Befehls

```
>> saveas(gcf,'dateiname','fig')
```

Hierbei bezeichnet gcf das aktuell ausgewählte Diagramm

- Diagramme können in verschiedenen grafischen Formaten gespeichert werden

Menüeintrag: File → Save As (eps, pdf, jpeg, png, tiff, ...)

- Diagramme können mit Hilfe des print Befehls ausgedruckt werden

- „Drucken“ des aktuellen Diagramms in eine Datei

```
>> print -deps2 gebirge.eps
```

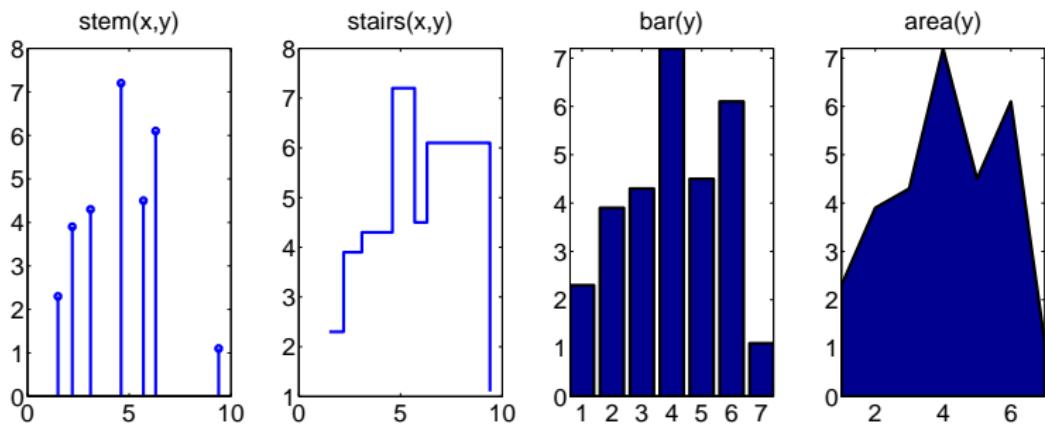
- Option -d legt das Ausgabeformat fest

```
-d{ps2, ps2c, eps2, eps2c, jpeg, tiff, png}
```

# Abbildungen mit mehreren Diagrammen

---

- `subplot(m,n,p)` erzeugt  $m \times n$  Array und aktiviert  $p$ -tes Diagramm
- `>> subplot(1,4,1); stem(x,y); ...`  
`subplot(1,4,2); stairs(x,y); ...`

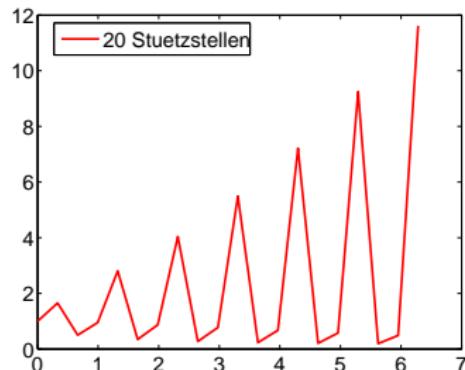


- Dokumentation zur Grafikausgabe in 2D: [doc graph2d](#)

# Darstellung von expliziten Funktionen

**Aufgabe:** Stelle die Funktion  $f(x) = e^{\sqrt{x}} \sin 2\pi x$  für  $x \in [0, 2\pi]$  dar.

- Erzeuge 20 äquidistante Stützstellen  $x_i$  im Intervall  $[0, 2\pi]$   
`>> x=linspace(0,2*pi,20);`       $\hat{=}$       `0:2*pi/(20-1):2*pi`
- Werte die Funktion  $f$  elementweise in den Stützstellen  $x_i$  aus  
`>> f=exp(sqrt(x).*sin(2*pi*x));`
- Stelle die Funktion grafisch dar  
`>> plot(x,f)`
- Auflösung hängt von der Anzahl der verwendeten Stützstellen ab  
→ Wiederhole Vorgehen mit 50 statt 20 Stützstellen

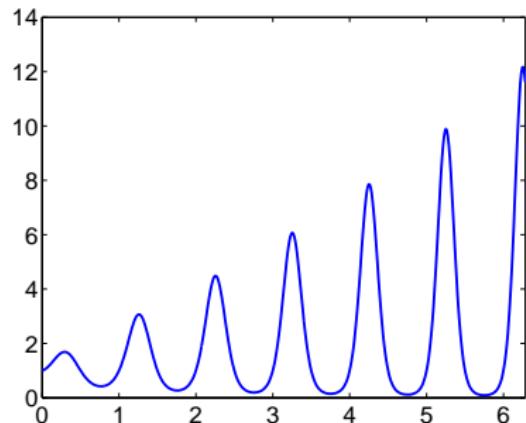


# Funktionsplotter für explizite Funktionen

**Aufgabe:** Stelle die Funktion  $f(x) = e^{\sqrt{x}} \sin 2\pi x$  für  $x \in [0, 2\pi]$  dar.

- Funktionen können einfach mit dem fplot Befehl dargestellt werden  
`>> fplot('exp(sqrt(x))*sin(2*pi*x)', [0 2*pi])`

- Anzahl der Stützstellen wird automatisch bestimmt
- Achsen werden angepasst
- fplot Befehl kann durch Parameter angepasst werden



# Optionen des fplot Befehls

---

```
>> fplot(fun,lims,tol,N,'LineSpec',p1,p2,...)
```

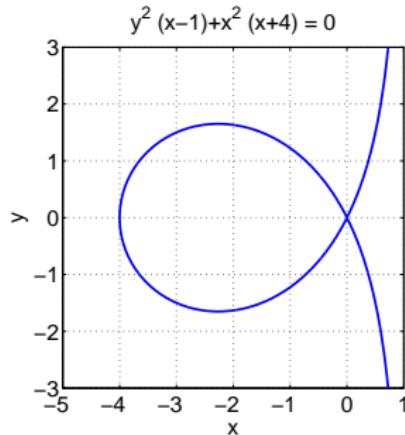
- fun gibt die darzustellende(n) Funktion(en) an  
`'exp(sqrt(x))*sin(2*pi*x))'`    [sin(x), cos(x)]
- lims legt die Wertebereiche für  $x$  und  $y$  fest  
`[xmin xmax]`      oder      `[xmin xmax ymin ymax]`
- tol legt den maximal zulässigen relativen Fehler fest
- Parameter N legt die Mindestanzahl an Stützstellen fest
- 'LineSpec' legt die Darstellung der Funktionsgraphen fest
- Parameter p1, p2, ... werden an Funktionen übergeben ( $\rightarrow$  später)

# Funktionsplotter für implizite Funktionen

**Aufgabe:** Stelle folgende implizit definierte Funktion dar

$$y^2(x - 1) + x^2(x + 4) = 0$$

```
>> ezplot('y^2*(x-1)+x^2*(x+4)',...
[-5 1 -3 3])
```



- `ezplot` (*easy-to-use plot*) ist ein einfacher Funktionsplotter
- `ezplot` kann auch explizite Funktionen darstellen  
`>> ezplot('sin(x)', [0 2*pi])`
- Standardwertebereich ist  $-2\pi \leq x \leq 2\pi$  und  $-2\pi \leq y \leq 2\pi$

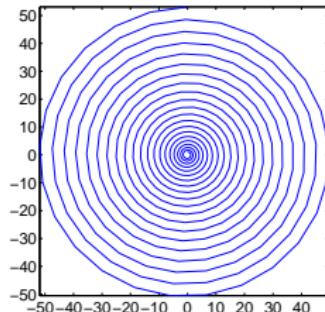
# Funktionsplotter für Parameterkurven

---

**Aufgabe:** Stelle folgende Kurve dar

$$f : [0, 2\pi] \rightarrow \mathbb{R}^2$$

$$t \mapsto \begin{bmatrix} (1+t)^2 \sin(20t) \\ (1+t)^2 \cos(20t) \end{bmatrix}$$



```
>> t=linspace(0,2*pi,500); >> ezplot('((1+t)^2*sin(20*t))',...
 x=(1+t).^2.*sin(20*t); '(1+t)^2*cos(20*t)',...
 y=(1+t).^2.*cos(20*t); [0 2*pi]) % 2D

>> plot(x,y); % 2D >> ezplot3('((1+t)^2*sin(20*t))',...
 % 3D '(1+t)^2*cos(20*t)',...
>> plot3(x,y,t); % 3D 't',[0 2*pi]) % 3D
```

# Erstellen von Tortendiagrammen

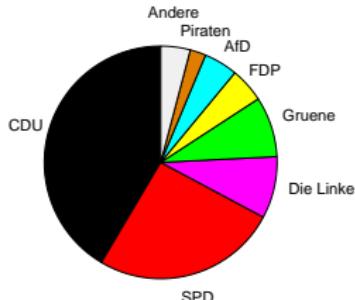
Wahlergebnis der Bundestagswahl 2013 (nach Spiegel)

| Union | SPD  | Linke | Grüne | FDP | AfD | Piraten | Andere |
|-------|------|-------|-------|-----|-----|---------|--------|
| 41,5  | 25,7 | 8,6   | 8,4   | 4,8 | 4,7 | 2,2     | 4,1    |

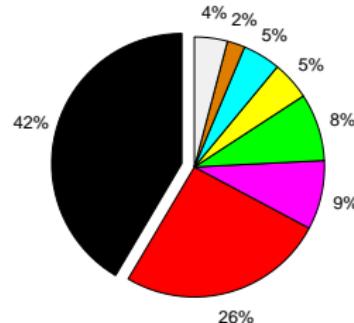
```
>> s=[41.5 25.7 8.6 8.4 4.8 4.7 2.2 4.1];
>> explode=[1 0 0 0 0 0 0 0];
>> lab={'Union' 'SPD' 'Die Linke' 'Gruene' 'FDP' 'AfD' ...
 'Piraten' 'Andere'}

```

```
>> pie(s,lab)
```



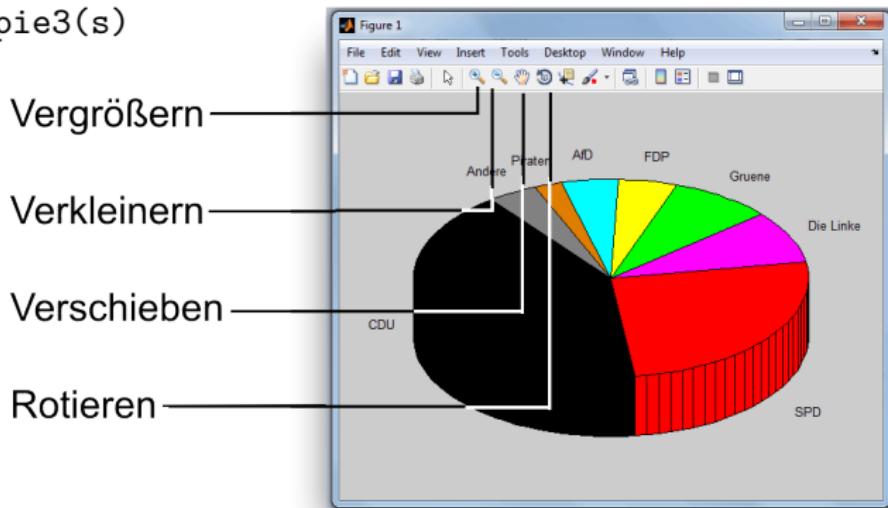
```
>> pie(s,explode)
```



# Tortendiagramme in 3D

- Zu einigen Befehlen gibt es eine 3D-Variante wie `plot3`, `ezplot3`

```
>> pie3(s)
```



- In MATLAB® kann der Blickwinkel interaktiv mit dem `rotate3d` Befehl bzw. per Klick auf das Symbol 'Rotate 3D' verändert werden

## Überblick 4. Vorlesung

---

- Grafikausgabe in 3D
- dünnbesetzte Matrizen
- spezielle Datenstrukturen
- flexible Argumentlisten

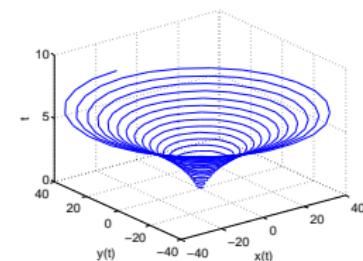
# Erinnerung an bekannte 3D-Varianten

Wir hatten bereits zu einigen Befehlen entsprechende 3D-Varianten kennengelernt

- plot3 und ezplot3

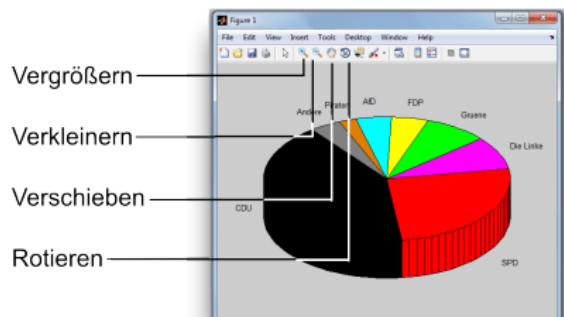
```
>> plot3(x,y,t);

>> ezplot3(' (1+t)^2 * sin(20*t)', ...
 ' (1+t)^2 * cos(20*t)', ...
 ' t', [0 2*pi])
```



- pie3

```
>> pie3(s)
```



# Anpassen des Blickwinkels

---

- In GNU Octave bzw. direkt aus dem Programm kann der Blickwinkel mit dem view Befehl angepasst werden

- `view(2)` erzeugt 2D Ansicht

- `view(3)` erzeugt 3D Ansicht

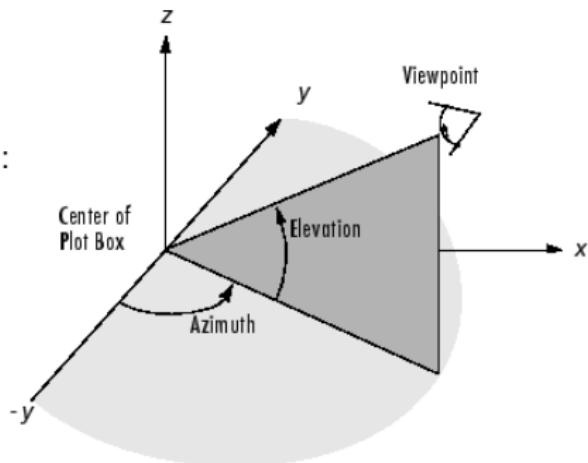
- `view(az,el)` setzt Blickwinkel:

- az Rotationswinkel um die  $z$ -Achse

- el vertikale Anhebung

- `view([x,y,z])` setzt Blickpunkt auf feste Koordinaten

- Dokumentation zur Blickwinkelanpassung: doc `view`, `viewmtx`



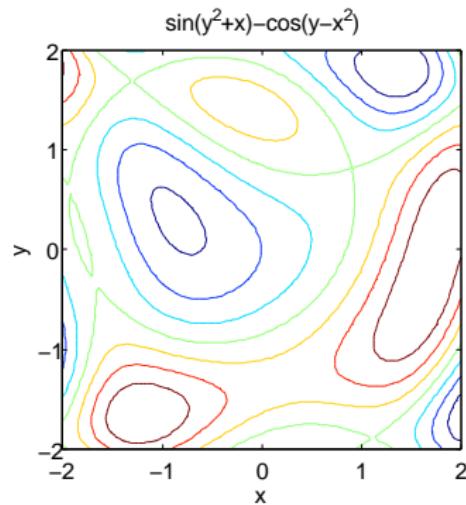
# Funktionen mit mehreren Veränderlichen

**Beispielfunktion:**  $f(x, y) = \sin(y^2 + x) - \cos(y - x^2)$   $x, y \in \mathbb{R}$

- Darstellung der Höhenlinien  $f(x, y) = c_i$  für Konstanten  $c_i \in \mathbb{R}$

```
>> ezcontour(...
 'sin(y^2+x)-cos(y-x^2)',
 [-2 2 -2 2], 100)
```

- Definitionsgebiet ist optional;  
Standardwerte  $-2\pi \leq x, y \leq 2\pi$
- Gitterauflösung ist optional;  
 $60 \times 60$  Punkte standardmäßig
- Anzahl der Höhenlinien wird  
automatisch vorgegeben



# Darstellung von Funktionswerten in 3D

---

- Erzeuge Vektoren  $x$  und  $y \in \mathbb{R}$  mit 3 bzw. 4 Stützstellen

```
>> x=linspace(0,1,3)
```

```
>> y=linspace(0,1,4)
```

| x = |        |        |  | y = |        |        |        |
|-----|--------|--------|--|-----|--------|--------|--------|
| 0   | 0.5000 | 1.0000 |  | 0   | 0.3333 | 0.6667 | 1.0000 |

- Erzeuge daraus die Koordinaten eines rechteckigen 2D-Gitter

```
>> [X Y]=meshgrid(x,y)
```

| X = |        |        |  | Y =    |        |        |        |
|-----|--------|--------|--|--------|--------|--------|--------|
| 0   | 0.5000 | 1.0000 |  | 0      | 0.3333 | 0.6667 | 1.0000 |
| 0   | 0.5000 | 1.0000 |  | 0.3333 | 0.3333 | 0.3333 |        |
| 0   | 0.5000 | 1.0000 |  | 0.6667 | 0.6667 | 0.6667 |        |
| 0   | 0.5000 | 1.0000 |  | 1.0000 | 1.0000 | 1.0000 |        |

- Matrixeintrag  $[X(j,i), Y(j,i)]$  steht für die Stützstelle  $(x_i, y_j)$

# Darstellung von Funktionswerten in 3D

---

- Auswertung einer Funktion in einer Stützstelle  $(x_i, y_j)$ :

```
>> i=2; j=3;
```

```
>> f=sin(x(i))*cos(y(j))
```

```
f =
```

0.3768

```
>> i=2; j=3;
```

```
>> f=sin(X(j,i))*cos(Y(j,i))
```

```
f =
```

0.3768

- Auswertung einer Funktion in allen Stützstellen  $(x_i, y_j)$ :

```
>> F=sin(X).*cos(Y) elementweise Multiplikation!
```

```
F =
```

0 0.4794 0.8415

0 0.4530 0.7952

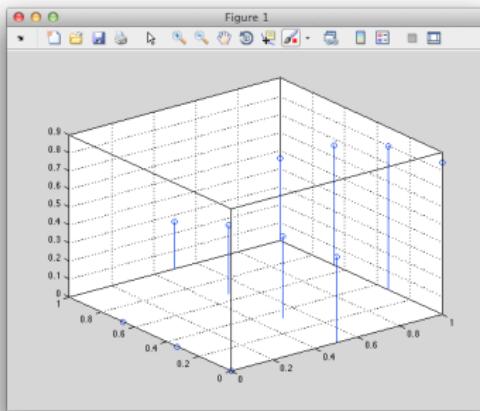
0 0.3768 0.6613

0 0.2590 0.4546

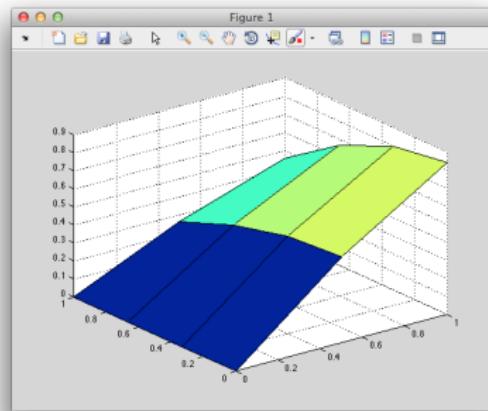
- $F(j, i)$  ist der Funktionswert zur Stützstelle  $(x_i, y_j)$

# Darstellung von Funktionswerten in 3D

**Allgemein:** `befehl(x,y,F)` mit  $x, y$  Vektoren,  $F$  Matrix



```
>> stem3(x,y,F)
```



```
>> surf(x,y,F)
```

# Anwendung auf die Beispielfunktion

---

- Matrizen X und Y mit sich spalten-/zeilenweise wiederholenden Werten

```
>> x=linspace(-2,2,20);
>> y=linspace(-2,2,20);
>> [X Y]=meshgrid(x,y);
```

- Funktionswerte

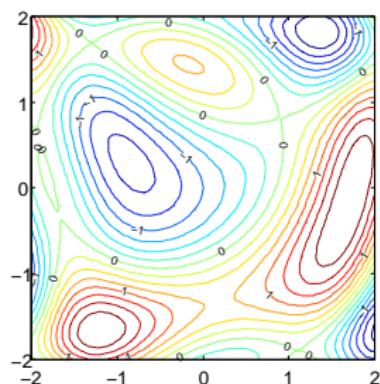
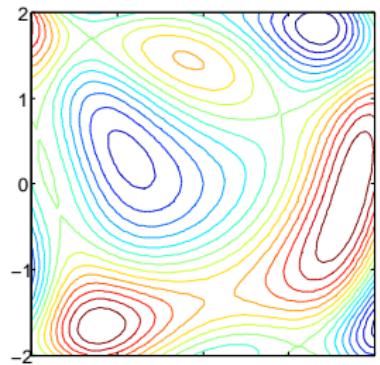
```
>> Z=sin(Y.^2+X)-cos(Y-X.^2);
```

- Höhenliniendarstellung

```
>> cwerte=linspace(-2,2,20);
>> [C h]=contour(x,y,Z,cwerte);
```

- Höhenlinienbeschriftung

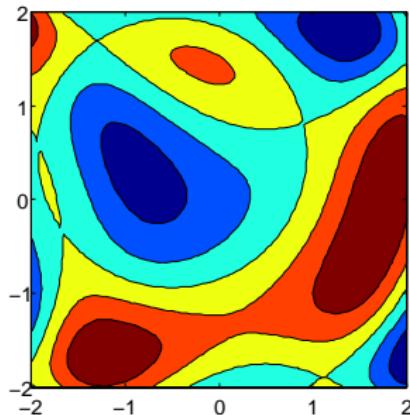
```
>> clabel(C,h,cwerte(1:4:17));
```



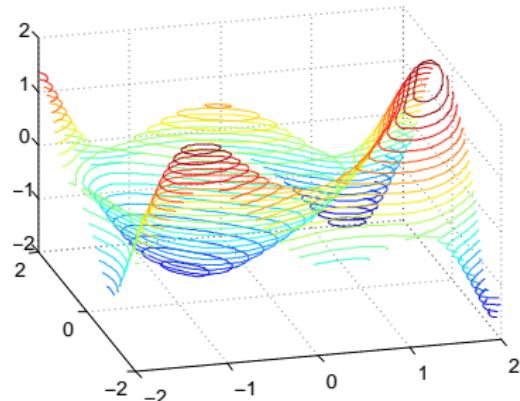
# Alternative Höhenliniendarstellung

---

```
>> contourf(x,y,Z,5)
```



```
>> contour3(x,y,Z,25)
```

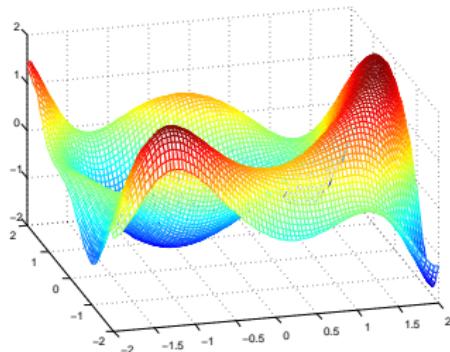


- Dokumentation zur Höhenliniendarstellung: `doc ezcontour`, `ezcontourf`, `contour`, `contourf` und `contour3`

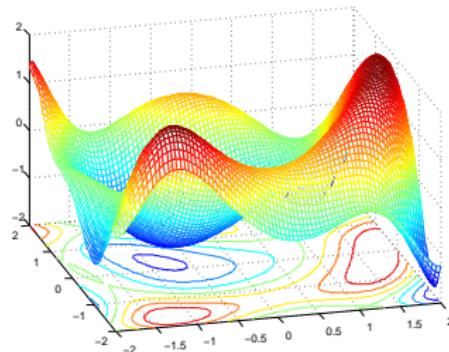
# Gitterliniendarstellung

---

```
>> mesh(x,y,Z)
```



```
>> meshc(x,y,Z)
```



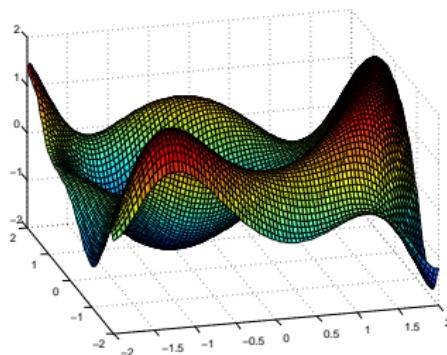
- Alternative (easy-to-use) Befehle `ezmesh` und `ezmeshc`

```
>> ezmesh('sin(y^2+x)-cos(y-x^2)', [-2 2 -2 2], 25)
```

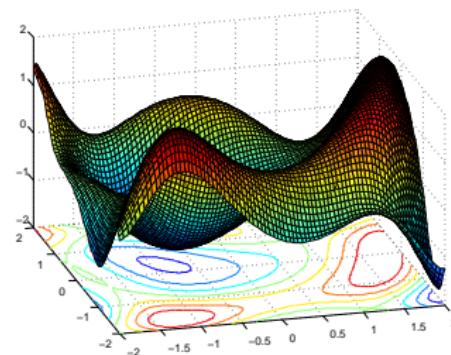
# Oberflächendarstellung

---

```
>> surf(x,y,Z)
```



```
>> surfc(x,y,Z)
```



- Alternative (easy-to-use) Befehle `ezsurf` und `ezsurf`
- Farbschattierung mittels `shading flat|interp|faceted`

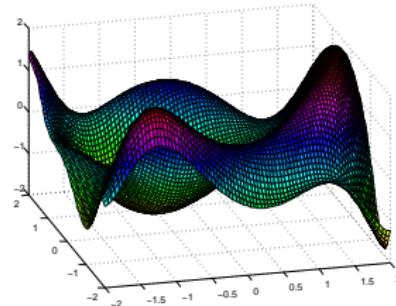
# Farbpaletten

---

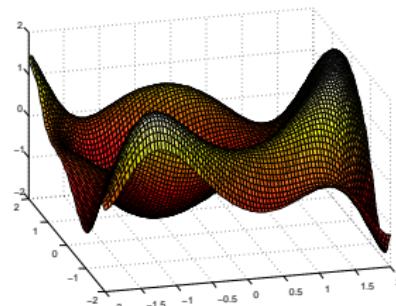
- `colormap('Farbpaletten')` aktiviert eine vordefinierte Farbpalette jet, hsv, hot,...
- `map=colormap` speichert aktuelle Farbpalette in `map`
- `colormap(map)` setzt  $m \times 3$  Matrix `map` als Farbpalette

| R | G | B | Farbe   |
|---|---|---|---------|
| 0 | 0 | 0 | Schwarz |
| 1 | 1 | 1 | Weiss   |
| 1 | 0 | 0 | Rot     |

für beliebige Tripel aus  $[0, 1]^3$

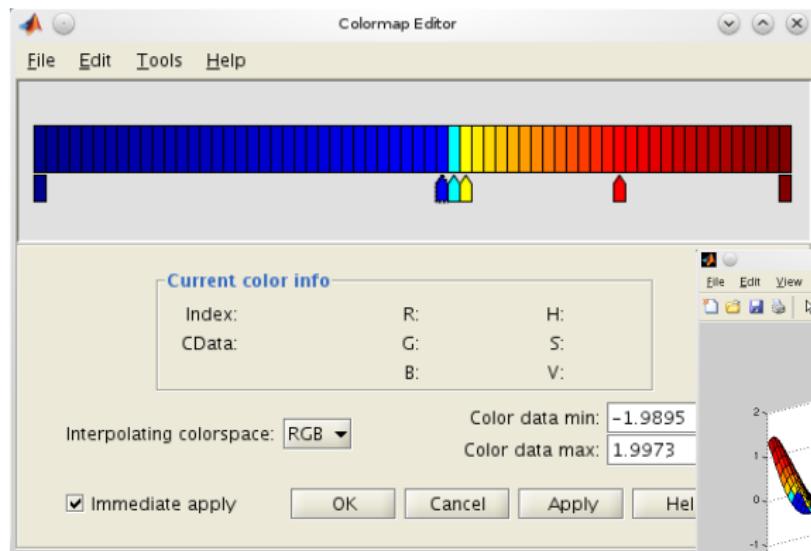


`colormap('hsv')`



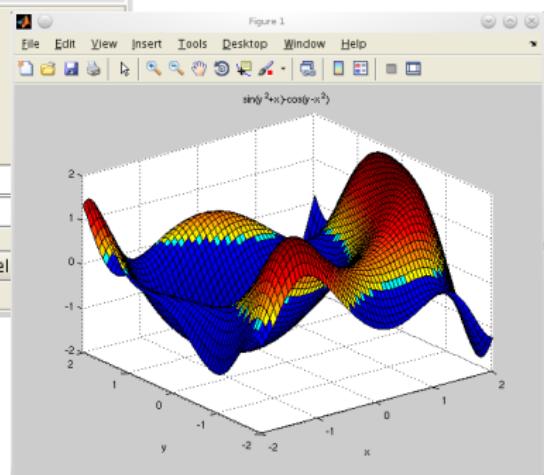
`colormap('hot')`

# Der Colormapeditor

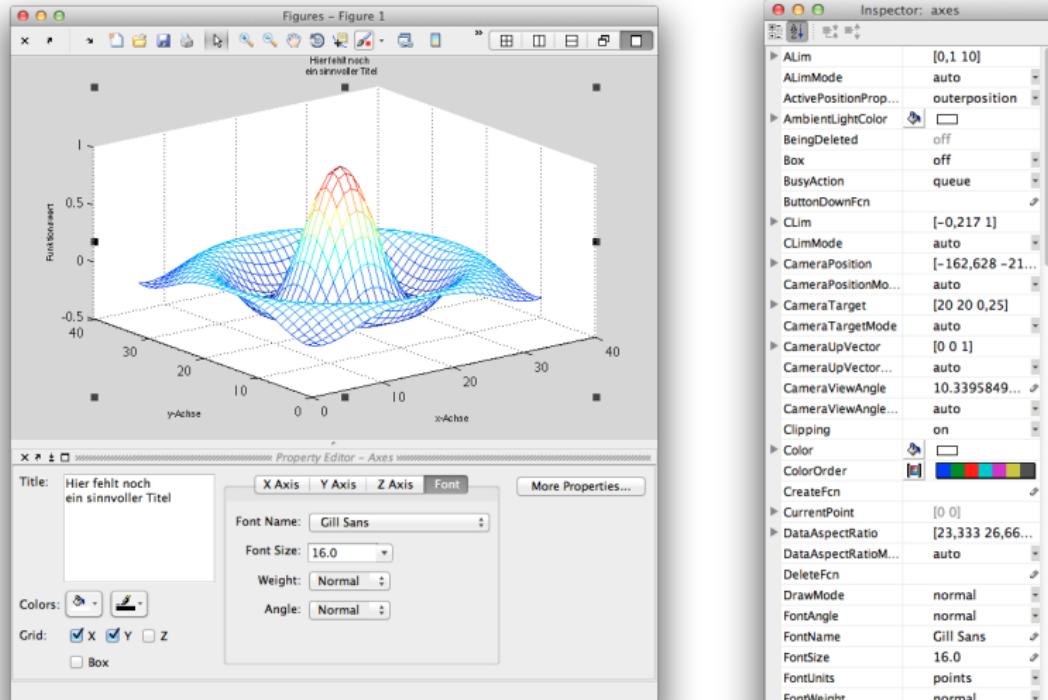


>> colormapeditor

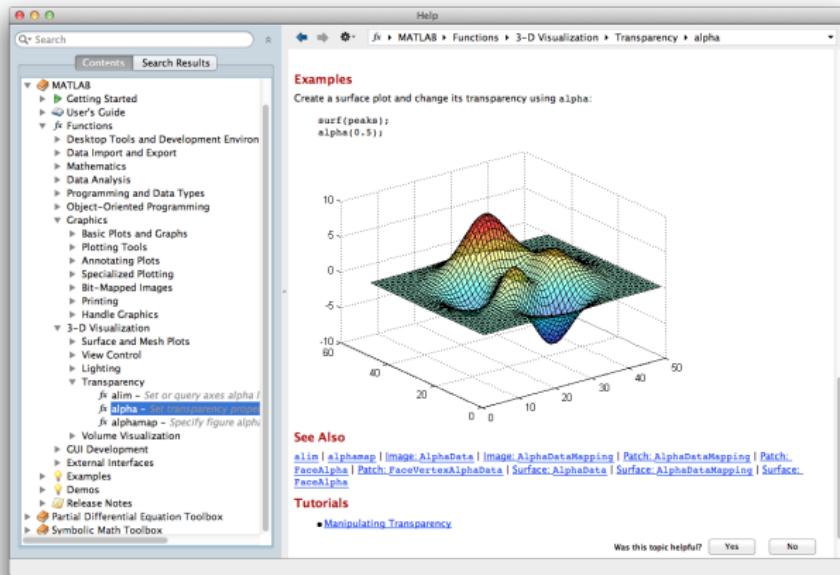
manuelle Anpassung der Farbpalette



# Der Propertyeditor



# Ausblick Graphics und 3D-Visualization



- Eine Stärke von MATLAB® ist die Vielzahl an hochwertigen Visualisierungsmöglichkeiten (siehe doc und demo)

# Übersicht über Datentypen/-strukturen

---

## Datentypen

- Variablen besitzen standardmäßig den Datentyp `double`
- Komplexe Variablen besitzen das Attribut `complex`
- Logischen Variablen besitzen den Datentyp `logical`
- Function Handles besitzen den Datentyp `function_handle`
- Zeichenketten sind Arrays vom Datentyp `char`
- MATLAB unterstützt weitere Datentypen wie `single`,  
`int8`, `int16`, `int32`, `int64`, `uint8`, ... (`doc class`)

## Datenstrukturen

- Mehrdimensionale Variablen, z.B. Vektoren, Matrizen,  $n$ D-Arrays, ...
- MATLAB® unterstützt einige spezielle Datenstrukturen

# Dünnbesetzte Matrizen

---

$$A = \begin{bmatrix} 1 & 4 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 7 & 0 & 0 & 8 \\ 0 & 0 & 9 & 2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1_{(1,1)} & 4_{(1,2)} & & \\ & 3_{(2,2)} & & \\ 7_{(3,1)} & & & \\ & & 9_{(4,3)} & 8_{(3,4)} \\ & & & 2_{(4,4)} \end{bmatrix}$$

- Alle Matrixeinträge werden gespeichert (inkl. „ $a_{ij} = 0$ “)
- Speicherplatzbedarf ist  $n \times m$   
`>> prod(size(A))`  
`ans = 16`
- Nur Matrixeinträge  $a_{ij} \neq 0$  werden gespeichert
- Spalten- und Zeilenindizes werden gespeichert
- Speicherplatzbedarf ist  $\mathcal{O}(nnz)$   
`>> nnz(A)`  
`ans = 7`

# Dünnbesetzte Matrizen

---

- Finden aller Nicht-Nulleinträge einer Matrix

```
>> [i j s]=find(A)
```

$$\begin{array}{lll} i = & j = & s = \\ 1 & 1 & 1 \\ 3 & 1 & 7 \\ 1 & 2 & 4 \\ \vdots & \vdots & \vdots \end{array} \quad A = \begin{bmatrix} 1 & 4 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 7 & 0 & 0 & 8 \\ 0 & 0 & 9 & 2 \end{bmatrix}$$

- Erzeugen einer dünnbesetzten Matrix

```
>> B=sparse(i,j,s)
```

$$B = \begin{array}{ll} (1,1) & 1 \\ (3,1) & 7 \\ (1,2) & 4 \\ \vdots & \vdots \end{array}$$

**Kurzform:**

```
>> B=sparse(A);
```

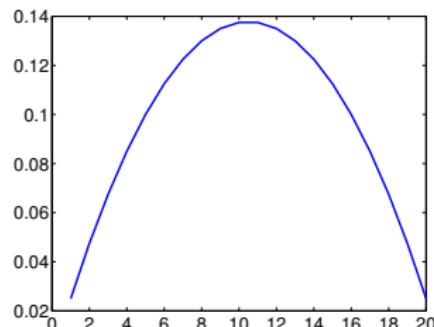
# Dünnbesetzte Matrizen

---

- Dünnbesetzte Matrizen werden wie „normale“ Matrizen behandelt  
→ Doppelpunktoperator, Matrizenmultiplikation, Linksdivision, ...
- speye Befehl erzeugt die dünnbesetzte Einheitsmatrizen
- sparse(n,m,nzmax) erzeugt dünnbesetzte  $n \times m$  Nullmatrix und reserviert Speicherplatz für maximal  $nzmax$  Nicht-Nulleinträge
- spdiags Befehl erzeugt dünnbesetzte Diagonalmatrizen

```
>> b=ones(5,1);
>> A=spdiags([-b 2*b -b],...
 [-1 0 1],5,5);
>> full(A)
ans =
 2 -1 0 0 0
 -1 2 -1 0 0
 0 -1 2 -1 0
 0 0 -1 2 -1
 0 0 0 -1 2
```

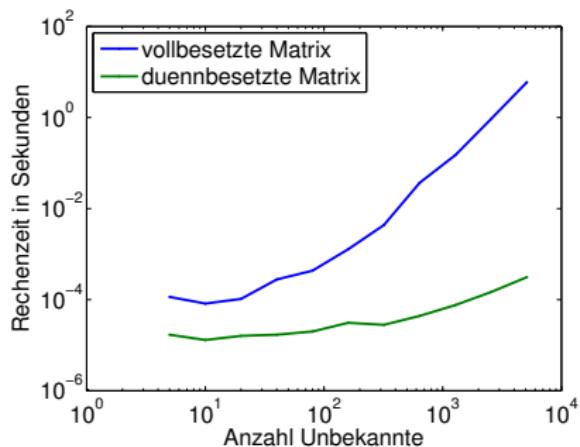
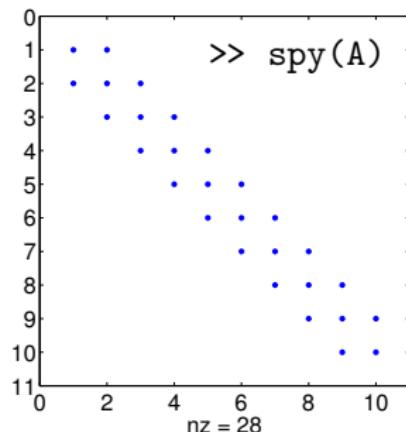
Lösung von  $Ax = b$



# Dünnbesetzte vs. vollbesetzte Matrizen

M-Datei: `spoisson.m` bzw. `poisson.m`

```
function [x time] = spoisson(n)
b=ones(n,1);
A=spdiags([-b 2*b -b],[-1 0 1],n,n).*n^2; A=full(A)
tstart=tic; % beginnt Zeitmessung
x=A\b;
time=toc(tstart); % beendet Zeitmessung
```



# Zellvariablen

---

- Arrays können nur Werte desselben Datentyps enthalten :-(

```
>> A = (1, 2; 3, 4);
```

- Zellvariablen sind „Arrays von beliebigen Datentypen“

```
>> C = {'Id', eye(3); rank(eye(3)), poly(sym(eye(3)))}
C =
```

```
'Id' [3x3 double]
[3] [1x1 sym]
```

- Auf einzelne Zelleinträge kann mittels „{ }“ zugegriffen werden

```
>> C{2, 2}
ans =
(x-1)^3
```

$\Leftarrow$  Zelleintrag  
    Arrayeintrag  $\Rightarrow$

```
>> A(2, 2)
ans =
4
```

- Alle von Arrays bekannten Zugriffsmöglichkeiten lassen sich auf Zellvariablen übertragen, z.B. C{1, :} oder C{1, [2 1]}

# Darstellung von Zellvariablen

---

```
>> celldisp(C)
```

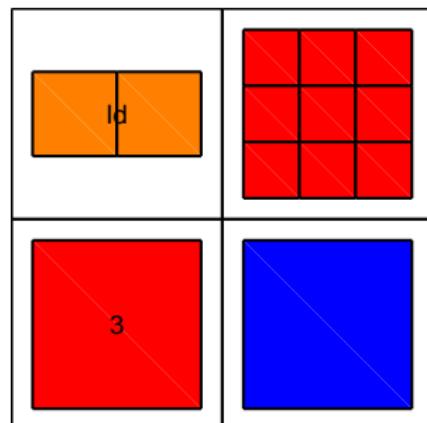
```
C{1,1} =
Id
```

```
C{2,1} =
3
```

```
C{1,2} =
1 0 0
0 1 0
0 0 1
```

```
C{2,2} =
(x-1)^3
```

```
>> cellplot(C)
```



⇒ „Container“ mit Platznummern

# Strukturen (= „Container“ mit Namensschildern)

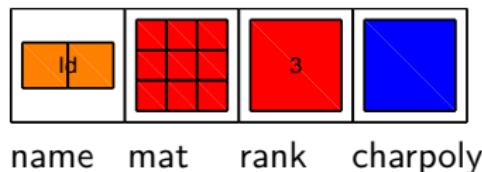
---

- Jedes Element einer Struktur besitzt einen Feldnamen

```
>> S = struct('name', 'Id',...
 'mat', eye(3),...
 'rank', rank(eye(3)),...
 'charpoly', poly(sym(eye(3))))
```

S =

```
name: 'Id'
mat: [3x3 double]
rank: 3
charpoly: [1x1 sym]
```



name mat rank charpoly

- Auf einzelne Struktureinträge kann direkt zugegriffen werden

```
>> S.charpoly
ans =
(x-1)^3
```

```
>> S.rank
ans =
3
```

# Anwendung von Zellvariablen und Strukturen

---

- einige Funktionen geben Strukturen als Rückgabewert zurück

```
>> struct=importdata('tudo.dat')
struct =
 data: [11x3 double]
 textdata: {12x4 cell}
```

- `struct.textdata` ist selbst wieder eine Zellvariable

```
>> struct.textdata
ans =
 'Jahr' ' Statistik' ' Mathematik'
 'WS98_99' '' ''
 'WS99_00' '' ''
```

. . .

# Anwendung von Zellvariablen und Strukturen

---

- Funktionen mit vielen verschiedenen Parametern sind unübersichtlich

```
>> x = -pi:pi/10:pi;
>> y = tan(sin(x)) - sin(tan(x));
>> plot(x,y,'--rs','LineWidth',2,...
 'MarkerEdgeColor','k',...
 'MarkerFaceColor','g',...
 'MarkerSize',10)
```

- Parameterlisten müssen überall im Code angepasst werden, wenn ein neuer Parameter hinzukommt bzw. geändert werden soll
- Parameter können in einer Struktur gesammelt werden

```
>> opts=struct('LineWidth',2,...
 'MarkerEdgeColor','k',...
 'MarkerFaceColor','g',...
 'MarkerSize',10)
>> plot(x,y,'--rs', opts)
```

# Wie kommt man an die Strukturinhalte?

---

M-Datei: structdemo.m

```
function structdemo(s)

f=fieldnames(s); % Hole Feldnamen aus der Struktur s
c=struct2cell(s); % Konvertiere Rest in Zellvariable

for i=1:length(f),
 switch f{i},
 case('LineWidth')
 disp(['LineWidth = ', num2str(c{i})]);
 case('MarkerEdgeColor')
 disp(['MarkerEdgeColor = ', c{i}]);
 ...
 otherwise
 disp(['Unbekannter Parameter ', f{i}]);
 end
end
```

# Funktionen mit beliebigen Parameterlisten

**Aufgabe:** Schreibe eine Funktion, die das Maximum  $\max(x_1, x_2, \dots)$  aus einer beliebigen Anzahl von Vektoren  $x_1, x_2 \dots$  berechnet.

M-Datei: allmax.m

```
function m = allmax(varargin)

error(nargchk(2,inf,nargin)) % >= 2 Parameter vorhanden?
m=-inf; for i=1:nargin
 m=max([m varargin{i}]);
end
```

- Struktur **varargin** nimmt beliebig viele Eingabeparameter entgegen
- Variable **nargin** ist **automatisch** die Anzahl der Eingabeparameter
- Allgemeine Form **nargchk**(low,high,nargs)

# Funktionen mit beliebigen Parameterlisten

---

- Funktionen können variable Anzahl an Ausgabeparametern besitzen

```
function [op1 varargout] = myfun(a)
```

- Variable **nargout** ist **automatisch** die Anzahl der Ausgabeparameter
- Allgemeinste Form einer Funktion

```
function [op1 op2 varargout] = myfun(ip1 ip2 varargin)
```

- mit erforderlichen Eingabeparametern ip1, ip2
- beliebig vielen optionalen Eingabeparametern varargin
- Ausgabeparametern op1, op2, wobei beim Aufruf  
`>> y=myfun(1,2)`  
nur der Rückgabewert y=op1 übernommen wird
- beliebig vielen optionalen Rückgabeparametern varargout

# Überblick 5. Vorlesung

---

- Symbolisches Rechnen
- Polynome (numerisch/symbolisch)
- Fehlersuche mit dem Debugger
- Effizienzsteigerung mit dem Profiler
- Gleitkommazahlen

# Polynome

---

- Darstellung von Polynomen der Form

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

mittels Koeffizientenvektor  $p = [p_1 \ p_2 \ \dots \ p_n \ p_{n+1}]$

- Auswertung von Polynomen (mittels Horner Schema)

$$\begin{aligned} >> \text{polyval}([1 \ -8 \ 15], 4) \quad p(x) = x^2 - 8x + 15 \\ \text{ans} = -1 \quad \quad \quad p(4) = 16 - 32 + 15 = -1 \end{aligned}$$

- Wird der `polyval` Befehl mit einem Vektor  $x$  oder einer Matrix  $X$  aufgerufen, so erfolgt die Polynomauswertung **elementweise**

$$\begin{aligned} >> y = \text{polyval}([1 \ -8 \ 15], [1 \ 2; \ 3 \ 4]) \\ y = \begin{matrix} 8 & 3 \\ 0 & -1 \end{matrix} \quad \quad \quad p(1) = 1 - 8 + 15 = 8 \\ \quad \quad \quad p(2) = 4 - 16 + 15 = 3 \\ \quad \quad \quad p(3) = 9 - 24 + 15 = 0 \end{aligned}$$

# Nullstellen von Polynomen

---

- Berechnung aller Nullstellen von Polynomen der Form

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

>> p=[1 -8 15];

$$p(x) = x^2 - 8x + 15$$

>> z=roots(p)

$$= (x - 5) \cdot (x - 3)$$

z = 5 3

$$x_1 = 5, \quad x_2 = 3$$

- Aufstellen des Polynoms zu gegebenen Nullstellen  $z_1, \dots, z_n$

$$p(x) = (x - z_1) \cdot (x - z_2) \cdots (x - z_n)$$

>> z=[5 3];

>> p=poly(z)

p = 1 -8 15

$$p(x) = (x - 5)(x - 3)$$

$$= x^2 - 8x + 15$$

# Ableiten von Polynomen

---

- Berechnung der Ableitung von Polynomen der Form

$$p(x) = p_1 x^n + p_2 x^{n-1} + \dots + p_n x + p_{n+1}$$

$$p(x)' = n p_1 x^{n-1} + (n-1) p_2 x^{n-2} + \dots + p_n$$

```
>> p=[1 -8 15]; p(x) = x2 - 8x + 15
```

```
>> p1=polyder(p) p'(x) = 2x - 8
p1 = 2 -8
```

```
>> p2=polyder(p1) p''(x) = 2
p2 = 2
```

- `polyder` Befehl berechnet die Koeffizienten des abgeleiteten Polynoms, wertet die Ableitung jedoch nicht aus ( $\rightarrow$  `polyval`)

# Polynommultiplikation

---

- conv Befehl (engl. *convolution*) führt Polynommultiplikation durch

```
>> q=[1 -4];
```

$$q(x) = x - 4$$

```
>> s=[1 -2 4];
```

$$s(x) = x^2 - 2x + 4$$

```
>> r=[0 0 0 8];
```

$$r(x) = 8$$

```
>> p=conv(q,s)+r
```

$$p(x) = x^3 - 6x^2 + 12x - 8$$

```
p =
```

```
1 -6 12 -8
```

- Polynome mit gleicher Koeffizientenzahl können addiert bzw. subtrahiert werden (vgl. Rechenregeln für Vektoren)

# MATLAB®Symbolic Math Toolbox

---

- Erweiterungspaket für symbolisches Rechnen in MATLAB
- Bis MATLAB Version 7.6 (R2008a) wird intern der Maple<sup>TM</sup> Kernel ( $\leq$  Ver. 3.2.3) benutzt, der teilweise separat erworben werden muss
- Ab MATLAB Version 7.7 (R2008b) wird intern der MuPAD Kernel ( $\geq$  Ver. 4.9) benutzt, der zur MATLAB Standardinstallation gehört
- Informationen über installierte Toolboxen liefert der ver Befehl

```
>> ver
```

```

MATLAB Version 7.13.0.564 (R2011b)
```

```
...
```

```

MATLAB Version 7.13 (R2011b)
```

```
Partial Differential Equation Toolbox Version 1.0.19 (R2011b)
```

```
Symbolic Math Toolbox Version 5.7 (R2011b)
```

# GNU Octave Symbolic Package

---

- Für GNU Octave existiert das Paket symbolic, mit dem **einfache** symbolische Berechnungen durchgeführt werden können
- Paket symbolic ist nicht kompatibel zur Symbolic Math Toolbox

- Informationen über die installierten Pakete auflisten

```
>> pkg list
```

- Zusätzliche Pakete von Octave-Forge nachinstallieren

```
>> pkg install <paketname> -global -forge
```

- Installierte Pakete zur Benutzung laden

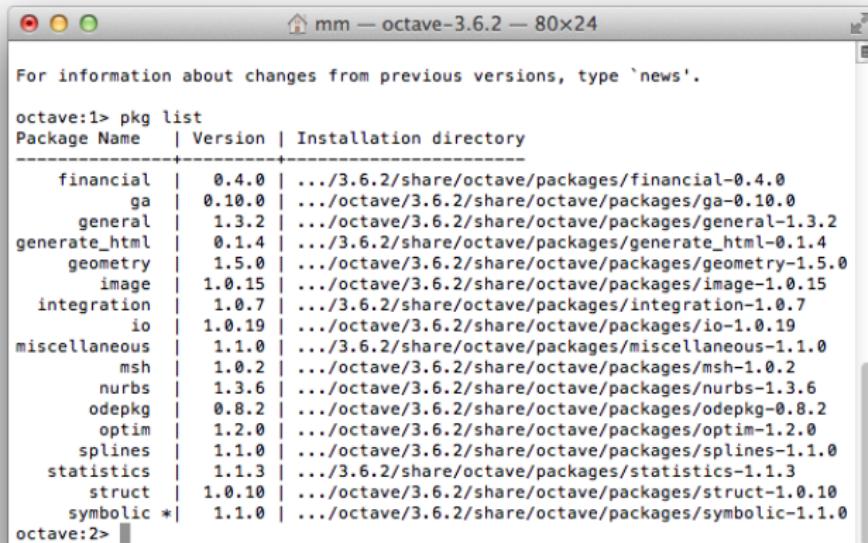
```
>> pkg load <paketname>
```

- Installierte Pakete beim Start automatisch laden

```
>> pkg rebuild -auto <paketname>
```

# GNU Octave Symbolic Package

- Die auf der Homepage bereitgestellten GNU Octave Versionen beinhalten die folgenden zusätzlichen Pakete von OctaveForce



```
mm — octave-3.6.2 — 80x24

For information about changes from previous versions, type 'news'.

octave:1> pkg list
Package Name | Version | Installation directory
-----+-----+-----
 financial | 0.4.0 | .../3.6.2/share/octave/packages/financial-0.4.0
 ga | 0.10.0 | .../octave/3.6.2/share/octave/packages/ga-0.10.0
 general | 1.3.2 | .../octave/3.6.2/share/octave/packages/general-1.3.2
generate_html | 0.1.4 | .../3.6.2/share/octave/packages/generate_html-0.1.4
 geometry | 1.5.0 | .../octave/3.6.2/share/octave/packages/geometry-1.5.0
 image | 1.0.15 | .../octave/3.6.2/share/octave/packages/image-1.0.15
 integration | 1.0.7 | .../3.6.2/share/octave/packages/integration-1.0.7
 io | 1.0.19 | .../octave/3.6.2/share/octave/packages/io-1.0.19
miscellaneous | 1.1.0 | .../3.6.2/share/octave/packages/miscellaneous-1.1.0
 msh | 1.0.2 | .../octave/3.6.2/share/octave/packages/msh-1.0.2
 nurbs | 1.3.6 | .../octave/3.6.2/share/octave/packages/nurbs-1.3.6
 odepkg | 0.8.2 | .../octave/3.6.2/share/octave/packages/odepkg-0.8.2
 optim | 1.2.0 | .../octave/3.6.2/share/octave/packages/optim-1.2.0
 splines | 1.1.0 | .../octave/3.6.2/share/octave/packages/splines-1.1.0
 statistics | 1.1.3 | .../3.6.2/share/octave/packages/statistics-1.1.3
 struct | 1.0.10 | .../octave/3.6.2/share/octave/packages/struct-1.0.10
symbolic *| 1.1.0 | .../octave/3.6.2/share/octave/packages/symbolic-1.1.0
octave:2>
```

# Symbolische Variablen

---

- `sym` Befehl **deklariert** eine einzelne symbolische Variable  
`>> a=sym('a');`
- `syms` Befehl **deklariert** eine Gruppe von symbolischen Variablen  
`>> syms b c;` **nicht in Octave verfügbar!!!**  
`>> b=sym('b'); c=sym('c');`
- Symbolische Variablen sind vom Datentype „symbolic object“

```
>> whos
```

| Name | Size | Bytes | Class | Attributes |
|------|------|-------|-------|------------|
| a    | 1x1  | 126   | sym   |            |
| b    | 1x1  | 126   | sym   |            |
| c    | 1x1  | 126   | sym   |            |

# Symbolische Zahlen

---



- Symbolische Zahlen können mit dem `sym` Befehl deklariert werden

```
>> x=sym('2'); y=sym('1/3'); z=sqrt(x);
```

- Symbolische Zahlen bleiben solange „exakt“ bis sie mit Hilfe des `double` Befehls in einen numerischen Wert konvertiert werden

```
>> y >> double(y) >> z >> double(z)
y = 1/3 ans = 0.3333 z = 2^(1/2) ans = 1.4142
```

- `sym` Befehl versucht `double` Werte als rationale Zahlen darzustellen

```
>> a=cos(pi/4), b=sym(a) >> sym(cos(pi-0.00000001))
a = b = ans =
0.7071 sqrt(1/2) -1
```

# Variablensubstitution

---

- Symbolische Funktionen können direkt definiert werden

```
>> a=sym('a');b=sym('b');c=sym('c'); f=a^2+b*c;
```

- Symbolische Variablen können mit subs substituiert

```
>> g=subs(f,a,3)
g = 9+b*c
```

```
>> h=subs(f,{a b},{3 c^2})
h = 9+c^3
```

- Wenn alle Variablen eines symbolischen Ausdrucks durch Werte ersetzt werden, dann wird das Ergebnis als double ausgegeben

```
>> d=subs(f, {a b c}, {3 4 2})
d = 17
```

- >> whos d f

| Name | Size | Bytes | Class  | Attributes |
|------|------|-------|--------|------------|
| d    | 1x1  | 8     | double |            |
| f    | 1x1  | 190   | sym    |            |

# Polynome

---



- Definition von symbolischen Polynomen der Form

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

```
>> x=sym('x'); p=x^2-8*x+15 ← p(x) = x2 - 8x + 15
```

- Auswertung von symbolischen Polynomen

```
>> subs(p,x,4) ← p(4) = 42 - 8 · 4 + 15 = -1
ans = -1
```

- Zerlegung in Polynomkoeffizienten  $p_k$  und Monome  $x^k$

```
>> [c m]=coeffs(p,x)
```

c =

[1, -8, 15]

symbolisches Objekt!

m =

[x<sup>2</sup>, x, 1]

symbolisches Objekt



# Nullstellen von Polynomen

- Berechnung aller Nullstellen von Polynomen der Form

$$\begin{aligned} p(x) &= p_1x^n + p_2x^{n-1} + \cdots + p_nx + p_{n+1} \\ &= (x - z_1) \cdot (x - z_2) \cdots (x - z_n) \end{aligned}$$

```
>> syms x; p=x^2-8*x+15; ← p(x) = (x - 3) · (x - 5)

>> solve(p) >> solve(p,x)
ans = 3 5 ans = 3 5

>> solve('x^2-8*x+15=0', 'x')
ans = 3 5 ans ist symbolisches Objekt!
```

- Wenn keine Variable angegeben ist, dann ruft der `solve` Befehl intern die `findsym` Funktion auf, um diejenige symbolische Variable zu bestimmen, die der Variablen *x* alphabetisch am nächsten ist ;-)



# Polynomdivision/-multiplikation

„Teilen eines Polynoms  $p$  durch  $q$  mit Rest  $r$ “

Zu zwei Polynomen  $p$  und  $q$  (nicht Nullpolynom) gibt es zwei eindeutig bestimmte Polynome  $s$  und  $r$  mit  $\deg r < \deg q$ , so dass gilt:  $p = sq + r$ .

- `quorem` Befehl (engl. *quotient* und *remainder*) führt Polynomdivision für symbolische Polynome durch

```
>> syms x; [s r]=quorem(x^3-6*x^2+12*x-8, x-4)
```

s =

$x^2 - 2x + 4$

r =

8

symbolisches Objekt!

symbolisches Objekt!

- ? Befehl führt Polynommultiplikation durch

```
>> p=s*(x-4)+r
```

```
p = (x - 4)*(x^2 - 2*x + 4) + 8
```

# Polynomformungen

---



- Ausmultiplizieren von symbolischen Ausdrücken

```
>> syms x; expand((x-4)*(x^2-2*x+4)+8)
ans =
x^3 - 6*x^2 + 12*x - 8
```

- Faktorisieren von symbolischen Ausdrücken

```
>> syms x; factor(x^3-6*x^2+12*x-8)
ans =
(x - 2)^3
```

- Vereinfachen von symbolischen Ausdrücken

```
>> syms x; simplify(x^3-6*x^2+12*x-8)
ans =
(x - 2)^3
```



# Charakteristisches Polynom

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 2 & 1 \\ 4 & 2 & 1 \end{pmatrix}, \quad p_A(\lambda) = \det(\lambda I - A) = \lambda^3 - 4\lambda^2 - \lambda + 4$$

- Definition der symbolischen Matrix  $A$

```
>> A=sym([1 0 1;2 2 1;4 2 1]);
```

- Berechnung des charakteristischen Polynoms  $p_A$

```
>> p=poly(A)
p = x^3 - 4*x^2 - x + 4
```

```
>> p=poly(A,'l')
p = 1^3 - 4*1^2 - 1 + 4
```

- Bestimmung der Nullstellen des Polynoms  $p_A$

```
>> solve(p)
```

```
ans =
```

-1  
1  
4

↔

```
>> eig(A)
```

```
ans =
```

-1  
1  
4

# Fehlersuche mittels Debugger

---

- Debugger sind Werkzeuge zur systematischen Fehlersuche
- Beobachtung des Programms während der Ausführung

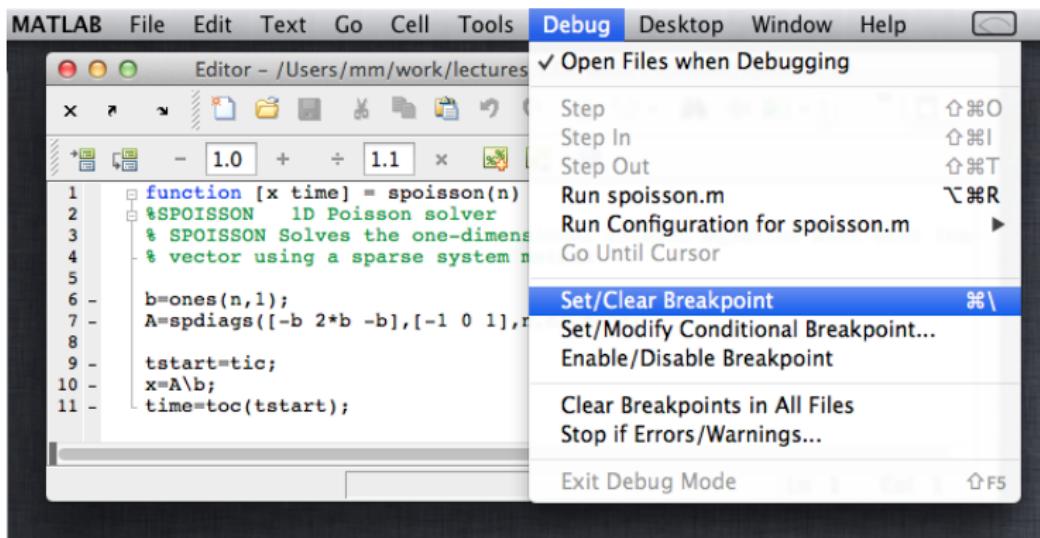
```
>> help debug
```

```
debug List M-file debugging functions
```

|          |                                   |
|----------|-----------------------------------|
| dbstop   | - Set breakpoint.                 |
| dbclear  | - Remove breakpoint.              |
| dbcont   | - Resume execution.               |
| dbdown   | - Change local workspace context. |
| dbmex    | - Enable MEX-file debugging.      |
| dbstack  | - List who called whom.           |
| dbstatus | - List all breakpoints.           |
| dbstep   | - Execute one or more lines.      |
| dbtype   | - List M-file with line numbers.  |
| dbup     | - Change local workspace context. |
| dbquit   | - Quit debug mode.                |

# Der MATLAB® Debugger

- MATLAB® enthält einen einfach zu bedienenden grafischen Debugger, der für jede im Editor geöffnete M-Datei verfügbar ist



# Der MATLAB® Debugger

- Breakpoints sind vom Benutzer aufgestellte "Stopschilder", an denen die Programmausführung angehalten wird
- An Conditional Breakpoints wird die Programmausführung angehalten, sofern die zugehörige Bedingung erfüllt ist

The screenshot shows the MATLAB Editor window with the following details:

- Title Bar:** Editor - /Users/mm/work/lectures/COP.WS1213/mfiles/spoisson.m
- Toolbar:** Standard MATLAB toolbar with icons for file operations, zoom, and help.
- Code Area:** The script file 'spoisson.m' is displayed:

```
function [x time] = spoisson(n)
%SPOISSON 1D Poisson solver
% SPOISSON Solves the one-dimensional Poisson equation
% with unit load vector using a sparse system matrix
%
% b=ones(n,1);
% A=spdiags([-b 2*b -b],[-1 0 1],n,n).*n^2;
% tstart=tic;
% x=A\b;
% time=toc(tstart);
```
- Breakpoints:** Two breakpoints are set:
  - Line 6: A red circle with a cross (conditional breakpoint).
  - Line 7: A red circle with a green arrow (regular breakpoint).
- Status Bar:** Shows "2 usages of 'A' found" and the current file name "spoisson". It also displays the line number (Ln 7) and column number (Col 1).

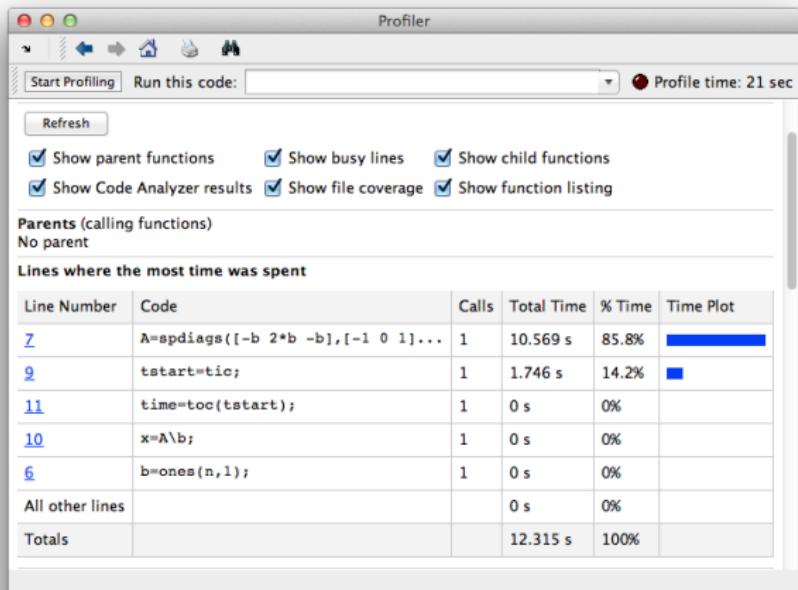
# Der MATLAB® Profiler

- Profiler sind Werkzeuge zum Auffinden von "Bottlenecks"

```
>> profile on
>> spoisson(100);
>> profile viewer
```

Dokumentation,  
Tipps und Beispiele

```
>> doc profile
```



# Gleitkommazahlen

```
Ax = b >> A=hilb(10); b=ones(10,1); x=A\b;
=> Ax - b = 0 >> norm(A*x-b)
ans = 2.3498e-10 ???
```

- Rechner approximieren Zahlen durch Gleitkommazahlen und führen Berechnungen mittels Gleitkommaarithmetik durch; es gilt **nicht**
  - Assoziativgesetz:  $(x + y) + z \neq x + (y + z)$   
 $(x \cdot y) \cdot z \neq x \cdot (y \cdot z)$
  - Distributivgesetz:  $x \cdot (y + z) \neq x \cdot y + x \cdot z$
- **Fazit:** niemals auf exakte Gleichheit überprüfen  
falsch: `isequal(A*x-b, 0)`      richtig: `norm(A*x-b)< tol`

# Gleitkommazahlen

---

- Beispiel: Lichtgeschwindigkeit im Vakuum  $2,99792458 \cdot 10^8 [m/s]$ 
  - **Basis**  $b = 10$  (Computer verwenden i.d.R. das Dualsystem,  $b = 2$ )
  - **Mantisse**  $m = 2,99792458$  enthält die Ziffern der Gleitkommazahl
  - **Exponent**  $e = 8$  speichert die genaue Position des Kommas ab
- Eindeutige Darstellung erhält man durch **Normalisierung**:

Festlegung des Wertebereichs der Mantisse auf  $1 \leq m < b$

richtig:  $1 \leq m = 2,99792458 < 10$

falsch:  $1 \leq m = 29,9792458 > 10$

falsch:  $1 \geq m = 0,29792458 < 10$

# Gleitkommazahlen

---

- Die Menge der Gleitkommazahlen  $\mathbb{F}$  ist eine *endliche* Teilmenge der rationalen Zahlen  $\mathbb{Q}$  und wird durch folgende Größen charakterisiert

- $b$  Basis
- $p$  Anzahl der Mantissenziffern
- $e_{\min}$  kleinster Exponent
- $e_{\max}$  größter Exponent

- IEEE-754-Standard

|               | $b$ | $p$ | $e_{\min}$ | $e_{\max}$ | MATLAB Datentyp |
|---------------|-----|-----|------------|------------|-----------------|
| einfach genau | 2   | 24  | -126       | 127        | single          |
| doppelt genau | 2   | 53  | -1022      | 1023       | double          |

- MATLAB Kommando `isieee` liefert Rückgabewert 1 zurück, wenn der IEEE-Standard unterstützt wird
- MATLAB rechnet intern stets mit doppelter Genauigkeit (`double`)

# Gleitkommazahlen

- Allgemeine Form einer normalisierten Gleitkommazahl

$$x = \pm \left( d_0 + \frac{d_1}{b} + \frac{d_2}{b^2} + \cdots + \frac{d_{p-1}}{b^{p-1}} \right) b^e = \pm d_0, d_1 d_2 \dots d_{p-1} \cdot b^e$$

- für die Mantissenziffern gilt  $0 \leq d_l \leq b - 1$  für  $l = 1, \dots, p - 1$
- für den Exponenten gilt  $e_{\min} \leq e \leq e_{\max}$

Beispiel: Lichtgeschwindigkeit im Vakuum

$$2.99792458 \cdot 10^8 =$$

$$+ \left( 2 + \frac{9}{10} + \frac{9}{10^2} + \frac{7}{10^3} + \frac{9}{10^4} + \frac{2}{10^5} + \frac{4}{10^6} + \frac{5}{10^7} + \frac{8}{10^8} \right) 10^8$$

- Zahl 0 kann nicht normalisiert dargestellt werden

# Gleitkommasysteme

---

## ■ Anzahl der normalisierten Gleitkommazahlen

$$|\mathbb{F}_{b,p,[e_{\min}, e_{\max}]}| = 2(b-1)b^{p-1}(e_{\max} - e_{\min} + 1) + 1$$

- Vorzeichen  
 $VZ \in \{+, -\}$  2 Möglichkeiten
- erste Mantissenziffer  
 $d_0 \in \{1, \dots, b-1\}$   $b-1$  Möglichkeiten
- übrige Mantissenziffern  
 $d_k \in \{0, 1, \dots, b-1\}$  je  $b$  Möglichkeiten
- Exponent  
 $e_{\min} \leq e \leq e_{\max}$   $e_{\max} - e_{\min} + 1$  Möglichkeiten
- Darstellung der Zahl Null +1

## ■ IEEE-754-Standard

einfach genau  $|\mathbb{F}_{2,24,[-126,127]}| = 4,261412865 \times 10^9$

doppelt genau  $|\mathbb{F}_{2,53,[-1022,1023]}| = 1,84287296752001 \times 10^{19}$

# IEEE-754-Standard

---

Der IEEE-754-Standard nutzt noch einige Besonderheiten aus, die aber zum grundsätzlichen Verständnis von Gleitkommazahlen unerheblich sind:

- erste Stelle einer normalisierten Mantisse im **Binärsystem** ist stets 1 und muss daher nicht gespeicher werden → *hidden bit*
- Anstatt den Exponenten  $e$  (Ganzzahl mit Vorzeichen)

$$e_{\min} \leq e \leq e_{\max}$$

zu speichern, reicht es aus die positive Ganzzahl

$$E = e + B \geq 1$$

und den Biaswert  $B$  (Verschiebung  $e_{\min} + B = 1$ ) anzugeben

# Runden

---

Wenn eine Zahl  $x \in \mathbb{R}$  nicht exakt durch eine Gleitkommazahl  $\hat{x} \in \mathbb{F}$  dargestellt werden kann, dann muss gerundet werden:

Beispiel:  $x = \pi = 3.14159 \dots$ ,  $\hat{x} \in \mathbb{F}_{b,p}$  mit  $b = 10$  und  $p = 4$

- Runden durch Abschneiden       $\hat{x} = 3.141$
- konventionelles Runden       $\hat{x} = 3.142$

Regel: Ist  $d_4 < 5$  so wird abgerunden, gilt  $d_4 \geq 5$  wird aufgerundet

- Optimales Runden (mit round to even)     $\hat{x} = 3.142$

Regel:  $x \in \mathbb{R}$  wird auf die nächstgelegene Gleitkommazahl gerundet;  
liegt  $x$  genau zwischen zwei Gleitkommazahlen, wird diejenige  
Näherung gewählt, deren letzte Mantissenziffer gerade ist

# Absoluter und relativer Fehler

---

Der Fehler zwischen dem exakten Wert  $x \in \mathbb{R}$  und der (gerundeten) Gleitkommazahl  $\hat{x} \in \mathbb{F}$  kann auf verschiedene Arten gemessen werden:

- absoluter Fehler       $err_{abs} = |x - \hat{x}|$

$$|\pi - 3.142| = 4.07346\ldots e-04$$

$$|100 \cdot \pi - 100 \cdot 3.142| = 4.07346\ldots e-02$$

- relativer Fehler       $err_{rel} = \frac{|x - \hat{x}|}{|x|}$

$$\frac{|\pi - 3.142|}{|\pi|} = 1.29662\ldots e-04$$

$$\frac{|100 \cdot \pi - 100 \cdot 3.142|}{|100 \cdot \pi|} = 1.29662\ldots e-04$$

# Auslöschung (engl. *cancellation*) → Numerik I

Beispiel: Gleitkommasystem  $b = 10, p = 4$

Subtrahiere  $y = 9.999 \cdot 10^{-1}$  von  $x = 1.000$  in Gleitkommaarithmetik

exakte Rechnung

$$\begin{array}{rcl} x & = & 1.000 \\ y & = & 0.9999 \\ \hline x - y & = & 0.0001 \\ & = & 1 \cdot 10^{-4} \end{array}$$

Rechnung in Gleitkommaarithmetik

$$\begin{array}{rcl} \hat{x} & = & 1.000 \cdot 10^0 \\ \hat{y} & = & 0.999 \cdot 10^0 \\ \hline \hat{x} - \hat{y} & = & 0.001 \cdot 10^0 \\ & = & 1.000 \cdot 10^{-3} \end{array}$$

Fehler:  $err_{\text{abs}} = 9 \cdot 10^{-4}, \quad err_{\text{rel}} = 9$

Bei der Subtraktion zweier fast gleich großer Zahlen werden relevante Ziffern durch die Angleichung der Exponenten ausgelöscht; der relative Fehler kann maximal  $b - 1$  betragen.

# Größenordnungen (engl. *absorption*)

Beispiel: Gleitkommasystem  $b = 10, p = 4$

Addiere  $x = 1.000$  und  $y = 1.000 \cdot 10^{-4}$  in Gleitkommaarithmetik

exakte Rechnung

$$\begin{array}{rcl} x & = & 1.000 \\ y & = & 0.0001 \\ \hline x + y & = & 1.0001 \end{array}$$

Rechnung in Gleitkommaarithmetik

$$\begin{array}{rcl} \hat{x} & = & 1.000 \cdot 10^0 \\ \hat{y} & = & 0.000 \cdot 10^0 \\ \hline \hat{x} - \hat{y} & = & 1.000 \cdot 10^0 \end{array}$$

Fehler  $err_{\text{abs}} = 1 \cdot 10^{-4}, \quad err_{\text{rel}} = 9.9990001 \cdot 10^{-5}$

Bei der Addition/Subtraktion einer betragsmäßig viel kleineren Zahl ändert sich die größere Zahl nicht. Die MATLAB Variable `eps` gibt die kleinstmögliche Gleitkommazahl an, für die gilt  $1 + \text{eps} \neq 1$ .

# Nützliche Tipps

---

- beim Start von GNU Octave werden automatische Befehle aus der Datei

```
./Octave/Octave3.6.4_gcc4.6.2/share/ ...
... octave/site/m/startup/octaverc
```

gestartet. Hier kann also die Editor Variable mittels

```
EDITOR('Pfad_zu_meinem_Editor\\meinEditor.exe');
```

gesetzt werden.

- mittels addpath('Pfad\_zu\_meinen\_MDateien') können entsprechend abgelegte M-Dateien automatisch gefunden und ausgeführt werden (ohne das diese manuell gesucht werden müssen). Um die so temporär modifizierte PATH Variable zu speichern sollte der Befehl savepath ausgeführt werden.

# Nicht behandelte Themen

---

- Interpolations- bzw. Extrapolationsverfahren  
→ Numerik I (doc polyfit, spline, interp1)
- Iterative Löser für große lineare Gleichungssystemen  
→ Numerik I (doc bicg, bicgstab, gmres)
- Numerische Behandlung von gewöhnlichen Differentialgleichungen  
→ Numerik II, spezielle MATLAB Funktion (doc ode45)
- Benutzung des MATLAB-Debuggers und -Profilers
- MEX-Compiler zum Anbinden von MATLAB and C-Programme
- Erstellung von grafischen Benutzeroberflächen mit GUIDE