MATLAB-Crashkurs

Niklas Borg

Institut für Angewandte Analysis und Numerische Simulation

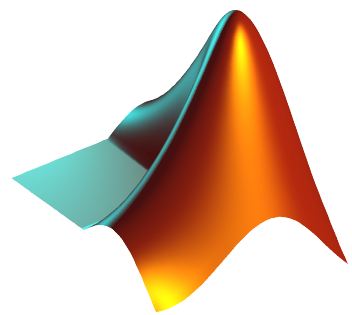
13. Mai 2015

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

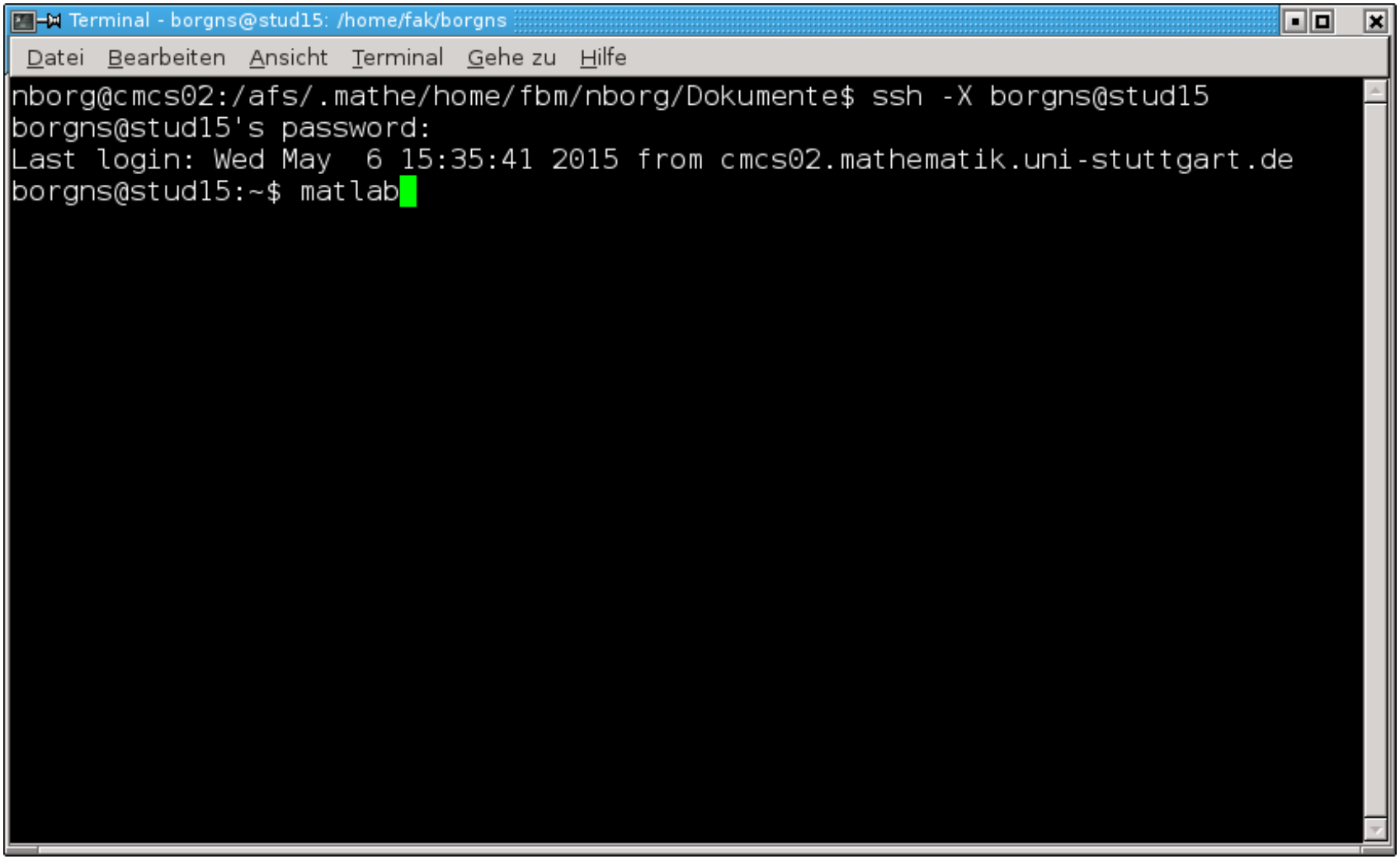
Bezugsquelle

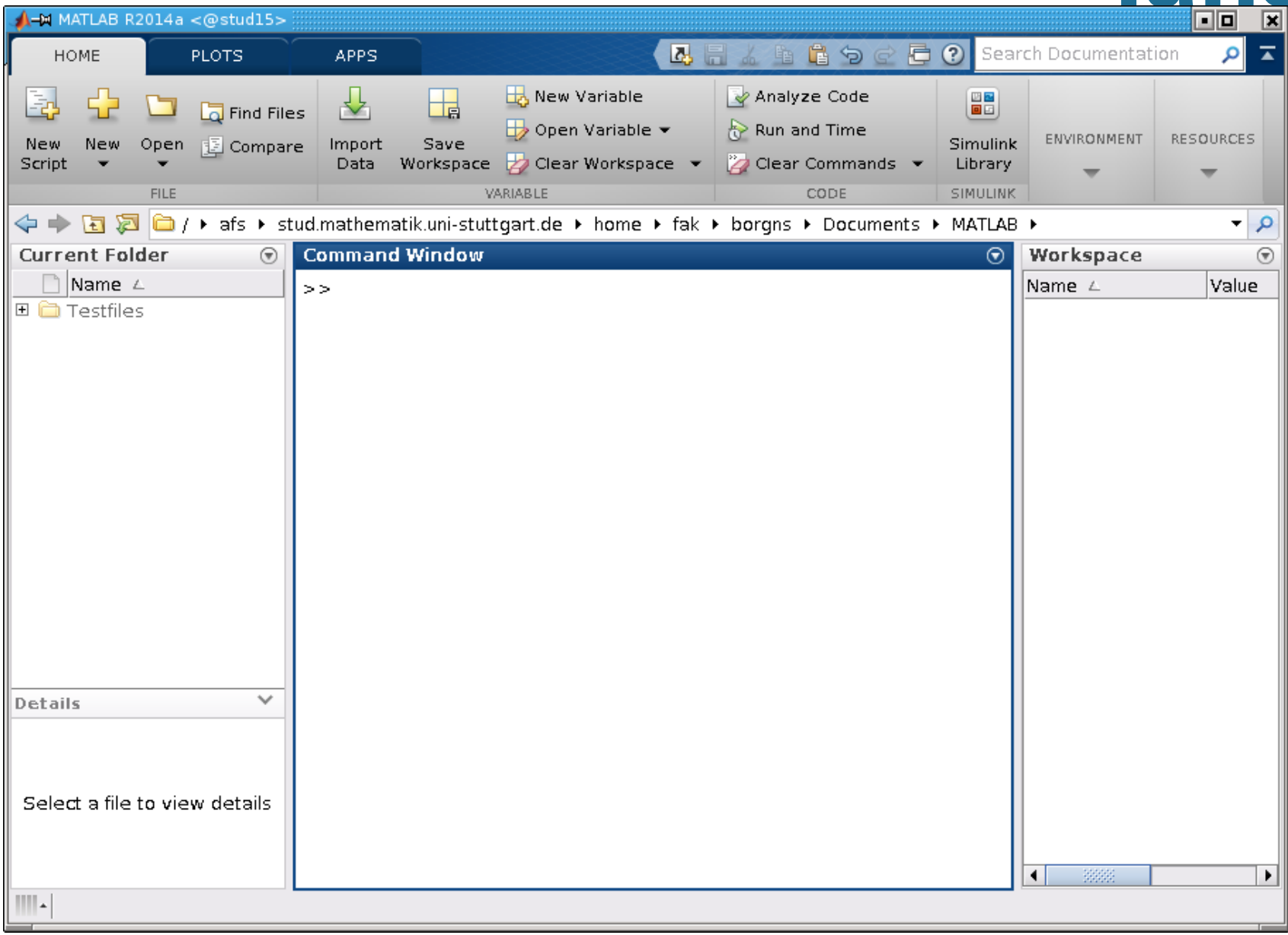
- Kommerzieller Vertrieb durch die Firma MathWorks http://www.mathworks.de/
- CIP-Pools sind mit MATLAB ausgestattet
- Kostenlose Studentenversion erhältlich MATLAB



MATLAB starten

- Terminal öffnen → „Matlab“ eingeben und bestätigen




MATLAB R2014a <@stud15>
HOME
PLOTS
APPS
Search Documentation
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
Analyze Code
Run and Time
Clear Commands
Simulink Library
ENVIRONMENT
RESOURCES
FILE
VARIABLE
CODE
SIMULINK
/ ▸ afs ▸ stud.mathematik.uni-stuttgart.de ▸ home ▸ fak ▸ borgns ▸ Documents ▸ MATLAB ▸
Current Folder
Name
Testfiles
Details
Select a file to view details
Command Window
>>
Workspace
Name
Value

Hilfreiches

- Standard Windows Shortscuts benutzen:
  Preferences → Matlab → Keyboard → Shortcut → „Active settings“ → Windows Default Set
- Schriftart/-größe ändern:
  Preferences → Matlab → Fonts → Desktop Code Font

Überblick

- integriertes Hilfesystem
- Variablen, Vektoren und Matrizen
- mathematische Operationen
- M-Dateien: Skripte und Funktionen
- Schleifen (`for`, `while`)
- if-Anweisungen
- Plots

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
- Aussagekräftige Ausgaben mittels:
  `fprintf('Text1 %f und Text2 %.3f \n', Variable1, Variable2)`
- Command Window mittels `clc` bereinigen

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

  Ohne Angabe der Schrittweite wird automatisch 1 angenommen

- Schritteweite mittels weiterem Doppelpunkt einfügen

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

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
b =
         0    1.5708    3.1416    4.7124    6.2832
```

`pi` ist als π vordefiniert

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 Werte 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
```

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

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

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

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
```

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

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

```
>> a=1:10; n=norm(a)
n =
    19.6214
```

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

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

$p = 1$	Betragssummennorm	$\|\mathbf{a}\|_1 = \sum_{k=1}^{n} \|a_k\|$
$p = \inf$	Maximumsnorm	$\|\mathbf{a}\|_\infty = \max_{k=1,k} \|a_k\|$
$p \in \mathbb{R}$	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*).

- 2×3 Matrix

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

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

  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

```
>> N = zeros(2)
N =
     0     0
     0     0
```

```
>> N = zeros(2,3)
N =
     0     0     0
     0     0     0
```

- „Einsmatrix"

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

```
>> E = ones(2,3)
E =
     1     1     1
     1     1     1
```

- Einheitsmatrix

```
>> I = eye(2)
I =
     1     0
     0     1
```

```
>> I = eye(2,3)
I =
     1     0     0
     0     1     0
```

Matrixoperationen

- Sämtliche Operationen wie +, −, * und ^ werden unterstützt

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

Produkt der Matrixeinträge

```
>> A*B
 ans =
    3    6
    7   14
```

Summe der Matrixeinträge

```
>> A+B
 ans =
    2    4
    4    6
```

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)                 >> r=rank(A)
d =                         r =
  0.0833                      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
```

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

- Es können auch Eigenwerte und Eigenvektoren berechnet werden

```
>> [R D]=eig(A)
R =                              D =
     0.4719   -0.8817                 0.0657         0
    -0.8817   -0.4719                      0    1.2676
```

Beachte: Bei der Eigenwertberechnung können Rundungsfehler auftreten.

Lineare Gleichungssysteme

- Lineare Gleichungssysteme der Form

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

  können mittels Linksdivision (`mldivide`)

  `x=A\b` oder direkt `x=inv(A)*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
```
```

Elementweise Operationsausführung

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

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

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*...*A` während `A.^k` jeden Eintrag der Matrix potenziert.

Teilmatrizen

- Matrizen können aus einzelnen Teilmatrizen aufgebaut werden

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

```
>> B = [A zeros(2); ...
        ones(2) eye(2)]
B =
     1   2   0   0
     3   4   0   0
     1   1   1   0
     1   1   0   1
```

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

```
>> C = repmat(eye(2),2,3)
C =
     1   0   1   0   1   0
     0   1   0   1   0   1
     1   0   1   0   1   0
     0   1   0   1   0   1
```

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

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

```
>> B=diag(A)
B =
     1
     5
     9
```

```
>> C=diag(A,1)
C =
     4
     8
```

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

```
>> D=diag(1:3)
D =
     1     0     0
     0     2     0
     0     0     3
```

```
>> E=diag(1:2,-1)
E =
     0     0     0
     1     0     0
     0     2     0
```

```
>> F=diag(diag(A))
F =
     1     0     0
     0     5     0
     0     0     9
```

M-Dateien

- Dateiendung `.m` kennzeichnet eine ausführbare Datei
- Algorithmen können in einer M-Datei editiert/gespeichert und mehrfach (mit unterschiedlichen Daten) aufgerufen werden
- **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 ;-)

Skripte

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 „%"
- Aufruf des Skripts im Kommando-Fenster durch maxentry (ohne .m!!!)

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)`
- Aufruf der Funktion im Kommando-Fester durch maxentry1(A)

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] = maxentry2(A)`

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

Übergabe von Funktionen

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

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);`
- Ausgabe `ans = 48.0001` $\leftarrow$ $p'(4) = 48$

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, ...`
- 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 100 Zufallszahlen zwischen 0 und 1 und speichere sie in einem Vektor.

```
for i=1:100
  Zufallszahlen(i) = rand(1);
end
```

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

- Schleifen über Wertelisten sind möglich

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

- Schleifen über Vektoren sind möglich

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

Schleifen mit variabler Anzahl an Wiederholungen

Aufgabe: Berechne solange Zufallszahlen zwischen 0 und 1 bis sich eine Zahl echt größer 0.8 ergibt und lasse die Anzahl der berechneten Zufallszahlen ausgeben.

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

```
while Bedingung
      Befehlsfolge
end
```

if Anweisungen - Beispiel

Aufgabe: Berechne 33 Zufallszahlen zwischen 0 und 1, sortiere diese in drei Blöcke der Größe $\frac{1}{3}$ und lasse diese am Ende ausgeben.

```
j=1; k=1; l=1;
for i=1:33
  Zufallszahl = rand(1);
  if Zufallszahl <= 1/3
    first(j) = Zufallszahl;
    j = j+1;
  elseif Zufallszahl > 1/3 && Zufallszahl <= 2/3
    second(k) = Zufallszahl;
    k = k+1;
  else
    third(l) = Zufallszahl;
    l = l+1;
  end
end
```

`if` Anweisungen - Allgemeine Syntax

```
if Bedingung
  Befehlsfolge
elseif Bedingung
  Befehlsfolge
else
  Befehlsfolge
end
```

- Zulässige Befehle für *Bedingung*: $<, <=, >, >=, ==, \sim=$
- Mehrere *Bedingungen* können mittels && (mathematisches *und*) oder || (mathematische *oder*) miteinander verknüpft werden
- Wird eine `if` oder `elseif` Bedingung erfüllt, gilt der gesamte `if` Block als beendet

Grafikausgabe in 2D

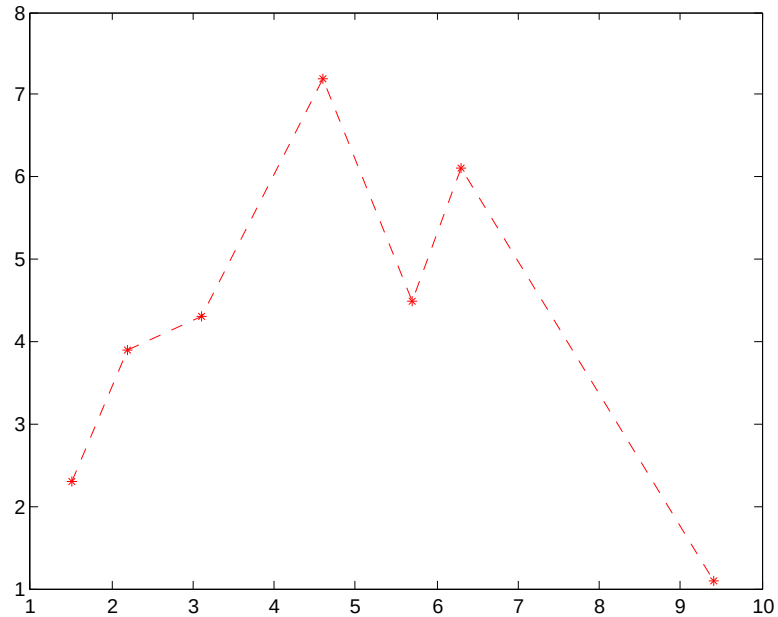
Aufgabe: Stelle folgende Daten grafisch dar

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 `x1`, `y1`, `x2` und `y2`
  - `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--')
```



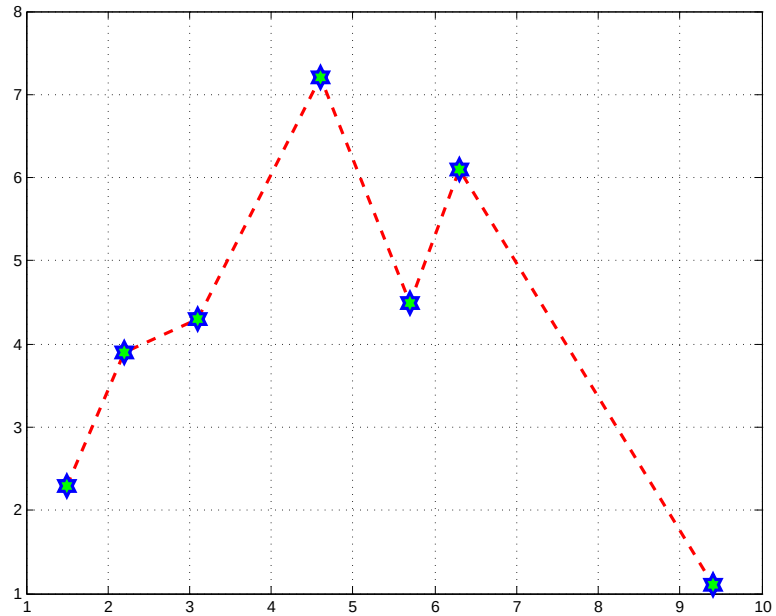
`plot(x,y,`*Zeichenkette*`)`

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

-	durchgezogen
--	gestrichelt
:	gepunktet
-.	abwechselnd

o	○
*	∗
.	·
+	+
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])`
  - $\pm$`inf` berechnet Grenze(n) automatisch
- Achsenverhältnis kann mittels `axis` angepasst werden
  `axis auto`, `equal`, `square`, `tight`, `off`
- Bearbeitung der Achseneinstellungen im *Property Editor*