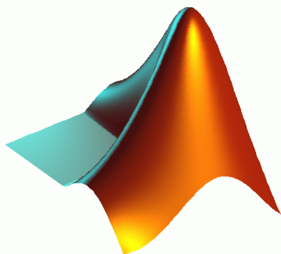
ulm university universität
uulm



Numerische Lineare Algebra

Matlab Einführung

Prof. Dr. Dirk Lebiedz,
M.Sc. Pascal F. Heiter,
Dipl. Math. oec. Klaus Stolle

Institut für Numerische Mathematik

Wintersemester 2013/14, 16.10.2013

Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Polynome, Zeichenketten etc.

Programmieren

Funktionen

Debuggen

Warum Matlab?

Was ist Matlab?

Matlab (Matrix laboratority)

- ist ein Softwarepaket für numerische Berechnungen und zur Visualisierung;
- wurde in den 1970er Jahren zur Unterstützung von Kursen der Linearen Algebra und numerischen Analysis entwickelt.

Was kann Matlab?

Matlab bietet

- eine einfache Syntax basierend auf dem Matrix-Datentyp;
- ein breites Spektrum mathematischer Funktionen und Algorithmen aus verschiedenen Anwendungsbereichen;
- eine plattformübergreifende Programmiersprache;
- einfach zu bedienende Visualisierungsmöglichkeiten.

Wo finde ich Matlab?

- Pools und Server des kiz (zeus.rz.uni-ulm.de)

```
zeus$ module avail math/matlab

----------------- /soft/common/modulefiles/Linux-x86_64 -----------------
math/matlab/R2011b          math/matlab/R2012a          math/matlab/R2012b
math/matlab/R2013a          math/matlab/R2013b(default)
zeus$ module load math/matlab
zeus$ matlab
```

oder

```
zeus$ cat .profile | grep matlab
module load math/matlab;
zeus$ matlab
```

- Nutzung auf dem eigenen Rechner
  - im Netz der Uni Ulm
  - mit einer Studentenlizenz
    (siehe www.uni-ulm.de → Hochschulportal→Software für Studierende)
    (http://portal.uni-ulm.de/PortalNG/content.title.software.html, erhältlich am Schalter des kiz für 20 Euro)

Matlab starten

Matlab wird gestarten

- über das Symbol auf dem Desktop oder in der Menüleiste oder
- durch Eingabe von `matlab` im Terminal/in der Shell.

Dadurch wird ein Matlab Fenster geöffnet.

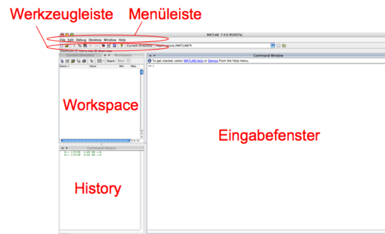
Matlab kann auch durch `matlab -nodesktop` ohne graphische Oberfläche gestartet werden:

```
zeus$ matlab -nodesktop

                              < M A T L A B (R) >
             Copyright 1984-2013 The MathWorks, Inc
                  R2013b (8.2.0.701) 64-bit (glnxa64)
                              August 13, 2013

   To get started, type one of these: helpwin, helpdesk, or demo.
   For product information, visit www.mathworks.com.

>>
```

Matlab starten

Nach erfolgreichem Start erscheint ein dreigeteiltes Fenster bestehend aus

- **Eingabefenster (Command Window):** Hier werden die Matlab Befehle eingegeben;
- **Workspace Fenster:** Zeigt die definierten Variablen an;
- **History Fenster:** Zeigt die zuletzt eingegebenen Befehle an;



Weitere Fenster wie z.B. der Matlab Editor oder Grafikfenster können beliebig in das Matlabfenster integriert werden.

Matlab wird durch Eingabe von `exit` oder `quit` im Eingabefenster beendet.

Beispiel 1: Übersicht Matlab

Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Polynome, Zeichenketten etc.

Programmieren

Funktionen

Debuggen

Elementares Rechnen in Matlab: erstes Beispiel

Beispiel 2: Berechne zu einem Kreisradius r die Fläche und den Umfang des Kreises und den Umfang eines flächengleichen Quadrates

```
>> r = 3
r =
     3
>> A_Kreis = r^2*pi
A_Kreis =
   28.2743
>> U_Kreis = 2*r*pi
U_Kreis =
   18.8496
>> U_Quadrat = 4*sqrt(A_Kreis)
U_Quadrat =
   21.2694
```

- Variablen werden durch Zuweisungen eines Wertes mit „=" definiert.
- Namen müssen mit einem Buchstaben anfangen und dürfen Buchstaben, Zahlen und den Unterstrich enthalten. WICHTIG: Dabei wird Groß- und Kleinschreibung berücksichtigt.
- Die Grundrechenarten sind durch die Zeichen $+, -, *, /, \hat{}$ (potenzieren) definiert.
- Bei den Operatoren gilt die übliche Auswertungsreihenfolge:
  Potenzieren vor Punktrechnung vor Strichrechnung.
  Auswertungsreihenfolgen können durch Klammerung geändert werden.

Elementare Funktionen

Es gibt eine Vielzahl elementarer Funktionen in Matlab:

`exp`, `pow2`	Exponentialfunktion zur Basis e bzw. 2
`log`, `log10`, `log2`	Logarithmus Funktionen
`sqrt`, `realsqrt`	Wurzelfunktionen
`sin`, `cos`, `tan`	Trigonometrische Funktionen
`asin`, `acos`, `atan`	Inverse der trigonometrischen Funktionen
`sinh`, `cosh`, `tanh`	Hyperbelfunktionen
`asinh`, `acosh`, `atanh`	Area Hyperbolicus Funktionen
`abs`, `sign`	Betragsfunktion bei skalarem Argument bzw. Signum
`round`, `floor`, `ceil`	runden, abrunden, aufrunden
`mod`, `rem`, `sign`	Modul, Divisionsrest, Vorzeichen

```
>> sin(pi)
ans =
   1.2246e-16
>> cos(pi)
ans =
    -1
...
```

```
...
>> exp(1)
ans =
    2.7183
>> sqrt(-1)
ans =
        0 + 1.0000i
```

Hilfeseite: `>> help elfun`.

Konstanten in Matlab

In Matlab sind einige spezielle Zahlen definiert:

`realmin, realmax`	kleinste bzw. größte darstellbare Gleitpunktzahl
`eps`	relative Genauigkeit von Gleitpunktzahlen
`inf, -inf`	$\pm\infty$
`NaN`	Not a number, nicht definierter Ausdruck, z.B. 0/0
`pi`	Kreiszahl π
`i, j`	imaginäre Einheit

```
>> pi*sqrt(-1)
ans =
        0 + 3.1416i
>> 0/0
ans =
   NaN
>> 1/0
ans =
   Inf
>> realmax
ans =
  1.7977e+308
>> realmin
ans =
  2.2251e-308
  >> 1+eps
ans =
    1.0000
```

Variablen

- In Matlab werden Variablen durch Zuweisungen ohne vorherige Deklaration angelegt.
- Variablennamen können aus Buchstaben, Ziffern und dem Zeichen _ bestehen, das erste muss ein Buchstabe sein.
- Matlab unterscheidet zwischen Groß- und Kleinschreibung bei Variablennamen (case-sensitive).
- In einem Workspace definierte Variablen können mit den Funktionen `who` und `whos` angezeigt werden.
- Durch Variablendefinition können vorhandene Matlab Funktionen und Variablen überschrieben werden.
- Mit `clear <Variablenname>` bzw. `clear` kann eine Variable bzw. alle Variablen im Workspace gelöscht werden.
- WICHTIG: Vorsicht mit den Variablen `i` und `j`:

```
>> i=2
i =
     2
>> pi*i
ans =
    6.2832
>> clear i
>> pi*i
ans =
        0 + 3.1416i
```

Speichern von Variablen und IO

- Variablen eines Workspace können mit `save <Dateiname> <Variablenname>` gespeichert und mit `load <Dateiname> <Variablenname>` wieder geladen werden.
- Ebenso kann mit `load` eine Textdatei mit einer Liste von Werten als Matrix eingelesen werden.
- Mit `save <Dateiname>` und `load <Dateiname>` werden alle Variablen des Workspace gespeichert bzw. alle Variablen der Datei geladen.
- Die Ein- und Ausgaben des Workspace in einer Matlab-Sitzung können mit dem Befehl `diary` aufgezeichnet werden (Beispiel 3: sitzung.txt):

```
>> diary sitzung.txt
>> r=3
r =
     3
>> h=5
h =
     5
>> V=r^2*h
V =
    45
>> diary off
```

sitzung.txt

```
r=3
r=
     3
h=5
h=
     5
V=r^2*h
V=
    45
diary off
```

Einfache Skripte

- Matlab Befehle können in **Textdateien mit Endung** `.m` gespeichert und im Workspace durch Eingabe des Dateinamens (ohne Endung) ausgeführt werden. Dazu kann der Matlab Editor `edit` oder jeder andere Texteditor benutzt werden.

Kegel.m

```
% Berechnung des Volumens
% eines Kegels
r=3
h=5
V=1/3*r^2*h
```

```
>> Kegel
r =
     3
h =
     5
V =
    15
>>
```

- Beispiel 4: Kegel.m
- Zeilen, die mit einem % beginnen, werden als Kommentarzeilen behandelt.
- Lange Eingaben können durch `...` auf mehrere Zeilen verteilt werden.
- Beim Aufruf im Workspace werden alle Skripte im aktuellen Verzeichnis und im Suchpfad berücksichtigt.
- Mit den Befehlen `pwd, cd, mkdir` können das aktuelle Arbeitsverzeichnis angezeigt, geändert bzw. neue Verzeichnisse angelegt werden.
- Mit `edit <Dateiname>` wird der Matlab-Texteditor aufgerufen, `type <Skriptname>` zeigt den Inhalt eines m-Files an.
- Die Funktion `what` listet alle m-Files im aktuellen Verzeichnis auf.

Matlab Hilfe im Command Window

In Matlab gibt es ein umfassendes Hilfe-System, um Informationen zu allen Funktionen zu bekommen. Es gibt verschiedene Möglichkeiten die Hilfe in Matlab zu nutzen:

- `help` oder `help <Thema>`

  Zeigt eine Übersicht über Hilfethemen oder über ein Thema bzw. einer Funktion im Command Window an;

- `lookfor <Text>`

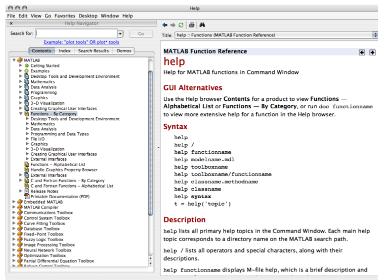
  Sucht in den Kurzbeschreibungen der Funktionen nach `<Text>` ;

```
>> help sin
 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.


    Reference page in Help browser
       doc sin

>> lookfor lookfor

LOOKFOR Search all M-files for keyword.
```

Matlab Hilfefenster

- `helpbrowser`
  Öffnet das graphische Hilfesystem;
- `doc <Thema>`
  Öffnet die Hilfe zum Thema oder zum Funktionsnamen im graphischen Hilfefenster zu einem Thema.

```
>> doc help
```



Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Polynome, Zeichenketten etc.

Programmieren

Funktionen

Debuggen

Rechnen mit Matrizen und Vektoren - Beispiel

Lösen des Gleichungssystems $Ax = b$ (Beispiel 5: LoeseLGS.m)

```
>> alpha = pi/4;
>> A = [cos(alpha), -sin(alpha); sin(alpha), cos(alpha)]
A =
    0.7071   -0.7071
    0.7071    0.7071
>> b = 1/sqrt(2)*[1; 1]
b =
    0.7071
    0.7071
>> x = A\b
x =
    1.0000
    0.0000
>>
```

- Matrizen und Vektoren können in Matlab duch Angabe der Elemente in eckigen Klammern definiert werden.
- Dabei werden die Werte zeilenweise angegeben, Elemente einer Zeile werden durch Kommata oder Leerzeichen voneinander getrennt, verschiedene Zeilen werden durch Semikolon oder Zeilenumbruch getrennt.
- Vektoren werden als Matrizen definiert, wobei die Zeilen- oder Spaltendimension 1 ist.
- In Matlab sind Operatoren zum Rechnen mit Matrizen, Vektoren und Skalaren definiert.

Operatoren für Matrizen

- Operationen zwischen zwei Matrizen / Vektoren: +, −, ∗ zum Addieren, Subtrahieren, Multiplizieren.(Beispiel 6: MatrixOperatoren.m)

```
>> alpha=pi/5;
>> A=[cos(alpha), -sin(alpha); sin(alpha) cos(alpha)];
>> B=[cos(-alpha), -sin(-alpha); sin(-alpha) cos(-alpha)];
>> C=A+B
C =
    1.6180         0
         0    1.6180
>> A*B
ans =
     1     0
     0     1
```

- Operatoren zum Lösen linearer Gleichungssysteme: /, \.

```
>> B
B =
    0.8090    0.5878
   -0.5878    0.8090
>> A\[1 0; 0 1]
ans =
    0.8090    0.5878
   -0.5878    0.8090
```

$$A \cdot B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$A \backslash \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = B$$

Operatoren für Matrizen und Skalare

- Operationen zwischen Matrizen/Vektoren und Skalar: **skalare Multiplikation** mit den Operatoren $*$ und $/$:

```
>> 3*[1 1; 0 1]
ans =
      3      3
      0      3
>> [1 1; 0 1]/3
ans =
    0.3333    0.3333
         0    0.3333
```

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

$$\frac{1}{3} \cdot \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.3333 & 0.3333 \\ 0 & 0.3333 \end{pmatrix}$$

- **Potenzieren** mit ^:

```
>> [1 1; 0 1]^2
ans =
      1      2
      0      1
```

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

- **Addition** und **Subtraktion** mit + und −:

```
>> [1 1; 0 1]+3
ans =
      4      4
      3      4
```

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 3 & 3 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 4 & 4 \\ 3 & 4 \end{pmatrix}$$

Komponentenweise Operationen für Matrizen/Vektoren

- **Komponentenweise Multiplikation und Division**: skalare Multiplikation mit den Operatoren `.*` und `./`:

```
>> A=[2 4; 6 9];
>> B=[2 2; 3 6];
>> A.*B
ans =
     4      8
    18     54
>> A./B
ans =
    1.0000    2.0000
    2.0000    1.5000
```

$$\begin{pmatrix} 2\cdot 2 & 4\cdot 2 \\ 6\cdot 3 & 8\cdot 6 \end{pmatrix} = \begin{pmatrix} 4 & 8 \\ 18 & 54 \end{pmatrix}$$

$$\begin{pmatrix} 2/2 & 4/2 \\ 6/3 & 8/6 \end{pmatrix} = \begin{pmatrix} 1.0000 & 2.0000 \\ 2.0000 & 1.5000 \end{pmatrix}$$

- **Komponentenweises Potenzieren** mit `.^`

```
>> A.^B
ans =
                 4
16
              216
531441
```

$$\begin{pmatrix} 2^2 & 4^2 \\ 6^3 & 8^6 \end{pmatrix} = \begin{pmatrix} 16 & 4 \\ 531441 & 216 \end{pmatrix}$$

Spezielle Matrizen

Funktionen zum Erzeugen für häufig verwendeter Matrizen:

- Einsmatrix- bzw. -vektor: `ones(n)`, `ones(n,m)`
- Nullmatrix- bzw. -vektor: `zeros(n)`, `zeros(n,m)`
- Einheitsmatrix bzw. -vektor: `eye(n)`, `eye(n,m)`
- Zufallsmatrix bzw. -vektor: `rand(n,m)`, `randn(n,m)`
- Diagonalmatrix zu einem Vektor mit Diagonalelementen oder Vektor der Diagonalelemente einer Matrix: `diag(x)`, `diag(A)`
- Magisches Quadrat: `magic(n)`

Mit gallery können noch weitere spezielle Matrixformen erzeugt werden.

Hilfeseite: `help elmat`.

```
>> eye(2,3)
ans =
     1     0     0
     0     1     0
>> ones(3)
ans =
     1     1     1
     1     1     1
     1     1     1
>> rand(3)
ans =
    0.4898    0.7094    0.6797
    0.4456    0.7547    0.6551
    0.6463    0.2760    0.1626
>> magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2
>> gallery('jordbloc',3,2)
ans =
     2     1     0
     0     2     1
     0     0     2
```

Spezielle Vektoren

Ebenso gibt es Funktionen für häufig verwendete Vektortypen (Zeilenvektoren!):

- **Sequenz von Zahlen mit festem Inkrement:** `a:b:c` bzw. `a:c` für das Inkrement 1 (Doppelpunkt-Notation!).

```
>> 2.3:.7:5.4
ans =
    2.3000    3.0000    3.7000    4.4000    5.1000
>> x=[1:-.7:-2.4]
x =
    1.0000    0.3000   -0.4000   -1.1000   -1.8000
>> 2.4:5.6
ans =
    2.4000    3.4000    4.4000    5.4000
```

- **lineare Unterteilung** eines Intervalls $[a, b]$ in n Teilintervalle: `linspace(a,b,n)`

```
>> linspace(1,4,3)
ans =
    1.0000    2.5000    4.0000
```

- **logarithmische Unterteilung** eines Intervalls $[a, b]$ in Teilintervalle n: `logspace(a,b,n)`

```
>> logspace(pi,exp(1),5)
ans =
   1.0e+03 *
    1.3855    1.0858    0.8510    0.6670    0.5227
```

Matrixindizierung

Auf Komponenten von Matrizen/Vektoren kann mit dem ( ) Operator zugegriffen werden. Dazu können die Elemente auf zwei verschiedene Arten indiziert werden:

über Zeilen- und Spaltenindizes

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & & a_{2,n} \\ \vdots & & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

Zugriff: `A(2,2)`

über Indizes der Elemente

$$A = \begin{pmatrix} a_1 & a_{m+1} & \cdots & a_{(n-1)m+1} \\ a_2 & a_{m+2} & & a_{(n-1)m+2} \\ \vdots & & \ddots & \vdots \\ a_m & a_{2m} & \cdots & a_{mn} \end{pmatrix}$$

`A(2)`

Hierbei werden die Elemente, im Gegensatz zur Eingabe, spaltenweise nummeriert.

```
 >> A=[1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
>> A(2, 2)
ans =
     5
>> A(2)
ans =
     4
```

Matrixindizierung

Mit dem Klammeroperator kann ...

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

...auf einzelne **Elemente** ...

$a_{2,2}$

a_2

```
>> A(2, 2)
ans =
     5
>> A(2)
ans =
     4
```

...oder auf **Teilmatrizen** zugegriffen werden.

a_1, a_2, a_3, a_4

a_{12}, a_{13}

```
>> A(1:4)
ans =
     1      4      2      5
>> A(1, 2:3)
ans =
     2      3
```

Verändern und Zusammensetzen von Matrizen - cont'd

Über den Zugriff auf Komponenten einer Matrix kann diese ausgelesen und verändert werden:

```
>> A(1, 2:3)=zeros(1,2)
A =
     1     0     0
     4     5     6
```

Zusammensetzung aus Teilmatrizen passender Größe

```
>> A=[1:4; eye(1,2), zeros(1,2)]
A =
     1     2     3     4
     1     0     0     0
```

Ebenfalls nützlich:

- `blckdiag` Matrizen werden hierbei entlang der Diagonalen anordnet.

Veränderung der Dimension:

- `reshape`

```
>> B=reshape(1:4, 2, 2)
B =
     1     3
     2     4
```

Verändern und Zusammensetzen von Matrizen

Mit dem Operator [] können Zeilen oder Spalten von Matrizen und Vektoren gelöscht werden.

```
>> A(:,2:3)=[]
A =
     1     4
     1     0
```

Um eine Matrix zu transponieren bzw. die komplex konjugierte zu bestimmen gibt es die Operatoren .' und ':

```
>> B'
ans =
     1     2
     3     4
```

Mit den Funktionen fliplr bzw. flipud kann die Reihenfolge der Spalten bzw. Zeilen der Matrix vertauscht werden:

```
>> fliplr(B)
ans =
     3     1
     4     2
>> flipud(B)
ans =
     2     4
     1     3
```

Matrix-Abmessungen

Die Abmessungen einer Matrix kann mit der Funktion `size A` bzw. `[z,s]=size(A)` ermittelt werden,

```
>> A=[1 2 3; 4 5 6];
>> [z, s]=size(A)
z =
     2
s =
     3
```

mit `length(a)` kann man die Länge eines Vektors ermitteln.

```
>> format compact
>> a = [2.3:0.4:8.9]
a =
  Columns 1 through 6
    2.3000    2.7000    3.1000    3.5000    3.9000    4.3000
  Columns 7 through 12
    4.7000    5.1000    5.5000    5.9000    6.3000    6.7000
  Columns 13 through 17
    7.1000    7.5000    7.9000    8.3000    8.7000
>> length(a)
ans =
    17
```

Funktionen für skalare Kenngrößen einer Matrix

Funktionen, mit denen Matrizen charakterisiert werden, sind

die Spur trace

$$\operatorname{tr}(A) = \sum_{i=1}^{n} a_{ii},$$

```
>> trace(A)
ans =
    15
```

den Rang rank, also die Dimension des Bildraums einer Matrix

$$\dim(\operatorname{Bild}(A)),$$

```
>> rank(A)
ans =
     3
```

die Determinante det

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)},$$

```
>> det(A)
ans =
  -360
```

(wird nicht so berechnet!!)

und die Kondition cond

$$\kappa(A) = \frac{\max_{\|x\|=1} \|Ax\|}{\min_{\|x\|=1} \|Ax\|}$$

```
>> cond(A)
ans =
    4.3301
```

Weitere Kenngrößen von Vektoren und Matrizen

Die Norm eines Vektors oder einer Matrix:

$$\|x\| = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}},$$

$$\|A\| = \max_{\|x\|=1} \|Ax\|;$$

```
>> norm(A)
ans =
    15
>> a=A(1,:)
a =
     8     1     6
>> norm(a)
ans =
    9.4340
```

die Summer der Elemente eines Vektors, bzw. die Summe der Elemente in jeder Spalte einer Matrix

```
>> sum(a)
ans =
    15
```

maximales bzw. minimales Element eines Vektors:

```
>> min(a)
ans =
     3
>> max(a)
ans =
     8
```

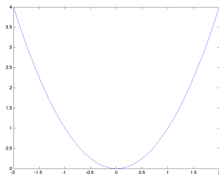
Elementares Plotten

Funktionen können in Matlab auf zwei Arten geplottet werden:

- Durch Plotten von endlich vielen Wertepaaren $(x, f(x))$, die ausgewertet und anschließend an die Funktion `plot` übergeben werden:

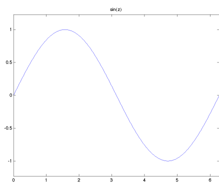
```
>> x=[-2:0.1:2];
>> y=x.^2;
>> plot(x,y);
```

Mit `plot` können auch mehrere Graphen mit unterschiedlichen Linientypen geplottet werden. Näheres dazu in der Hilfe... (Beispiel 7: ElementaresPlotten.m)



- Durch Angabe einer Inline-Funktion, die zusammen mit einem Intervall an die Funktion `ezplot` übergeben wird:

```
>> ezplot('sin(x)', [0,2*pi]);
```



Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Polynome, Zeichenketten etc.

Programmieren

Funktionen

Debuggen

Polynome

Polynome werden in Matlab durch Koeffizienten-vektoren dargestellt.

- `p = [ones(10,1)]` Bsp. für Koeffizientenvektor
- `poly` erzeugt das charakteristische Polynom einer Matrix als Vektor der Koeffizienten.

```
>> A=magic(3);
>> p=poly(A)
p =
    1.0000   -15.0000
-24.0000   360.0000
```

Auswertung eines Polynoms

- mit einer skalaren Größe kann die Funktion `polyval` verwendet werden.
- mit einer Matrix durch die Funktion `polyvalm`

```
>> polyval(p, 2)
ans =
  260.0000
>> polyvalm(p, A)
ans =
   1.0e-12 *
    0.4547   -0.5116   -0.0853
   -0.3340    0.1705    0.0568
   -0.2700    0.1705         0
```

Nullstellenbestimmung:

- `roots`

```
>> roots(p)'
ans =
   15.0000   -4.8990    4.8990
```

Hilfeseite: `>> help polyfun`.

Zeichenketten

Ein weiterer Datentyp in Matlab sind die Zeichenketten.

- Zeichenketten werden in Matlab in einfachen Hochkommata '' angegeben, gespeichert werden sie als Vektor von Buchstaben (char Array).

```
>> a='Hallo␣Ulm'
a =
Hallo Ulm
>> whos
  Name         Size                Bytes  Class     Attributes

  a            1x9                    18  char
```

- Auf die Buchstaben einer Zeichenkette kann wie auf Elemente von Matrizen zugegriffen werden.

```
>> a(1:5)
ans =
Hallo
```

- Mit den Funktionen `double` und `char` können Strings in Gleitzahlvektoren und umgekehrt konvertiert werden. Dabei werden Zeichen entsprechend der ASCII-Tabelle codiert.

```
>> b=double(a)
b =
    72    97   108   108   111    32    85   108   109
>> c=char(b+1)
c =
Ibmmp!Vmn
```

Zeichenketten modifizieren und auswerten

- Mit `findstr` bzw. `strrep` können Zeichenketten gesucht und ersetzt werden

```
>> b=strrep(a, 'Ulm', 'Welt')
b =
Hallo Welt
```

- Zum Vergleichen von Strings gibt es die Funktion `strcmp`. Diese gibt eine (logische) 1 zurück falls die Strings übereinstimmen. Mit der Funktion `findstr` können Teilstrings gesucht werden. Das Ergebnis ist der Index des ersten Vorkommens des Teilstrings.

```
>> strcmp(a,b)
ans =
     0
>> strcmp(a(1:5),b(1:5))
ans =
     1
>> findstr('Ulm', a)
ans =
     7
```

- Mit `upper` und `lower` kann eine Zeichenkette in Groß- bzw. Kleinbuchstaben übersetzt werden:

```
>> upper(b)
ans =
HALLO WELT
```

Zeichenketten und Zahlen

- Um Zahlen als Zeichenketten auszugeben oder eingelesene Zeichenketten als Zahl zu interpretieren können die Funktionen `num2str` und `str2num` verwendet werden.

```
>> fast_pi='3.14'
fast_pi =
3.14
>> str2num(fast_pi)
ans =
    3.1400
>> Kreiszahl=num2str(pi)
Kreiszahl =
3.1416
>> whos
  Name               Size                Bytes  Class     Attributes

  Kreiszahl          1x6                    12  char
  ans                1x1                     8  double
  fast_pi            1x4                     8  char
```

Hilfeseite: `>> help strfun`

Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Polynome, Zeichenketten etc.

Programmieren

Funktionen

Debuggen

Skripte

- Definitionen, Operationen auf Objekten und Funktionsauswertungen können in Matlab in Textdateien mit der Dateiendung `.m` (**m-Files**) zusammengefasst und zusammen ausgeführt werden.
- Aufruf von m-Files im Command Window durch Eingabe des Dateinamens ohne Dateiendung `.m`
- Kommentare beginnen mit `%` und gehen bis zum Ende der Zeile. Kommentarblöcke können in `%{ }%` eingeschlossen werden wobei die Kommentarzeichen für Anfang und Ende jeweils in einer eigenen Zeile stehen müssen.
- Nach m-Files wird in dem aktuellen Arbeitsverzeichnis und im Installationsverzeichnis von Matlab gesucht. Weitere Pfade können mit `path` hinzugefügt werden.
- Insbesondere bietet sich die Verwendung der Kontrollstrukturen in Skripten an:
  - `if`
  - `switch`
  - `for`
  - `while`

if Anweisung

- Syntax:

```
if <Bedingung>
    <Anweisung>
elseif <Bedingung>
    <Anweisung>
else
    <Anweisung>
end
```

- Der else Block und der elseif Block ist optional und kann weggelassen werden;
- Die if Anweisung kann beliebig viele elseif Blöcke enthalten;
- Beispiel 9: ifbsp.m

ifbsp.m

```
x=rand(2,1)
abstand=norm(x)
disp('Der␣Punkt␣liegt...');

if(abstand>1)
    disp('...ausserhalb...');
elseif(abstand<1)
    disp('...im␣Innern...');
else
    disp('...auf␣dem␣Rand...');
end
disp('des␣Einheitskreises');
```

```
>> ifbsp
x =
    0.7060
    0.0318
abstand =
    0.7068
Der Punkt liegt
...im Innern...
des Einheitskreises
```

switch Anweisung

- Syntax:

```
switch <Ausdruck>
    case Wert
        <Anweisung>
    case {Wert1, Wert2, ...}
        <Anweisung>
    otherwise
        <Anweisung>
end
```

- Der Ausdruck wird von oben nach unten mit den Werten verglichen und die Anweisungen der ersten Übereinstimmung ausgeführt. Spätere Übereinstimmungen werden ignoriert.
- Falls es keine Übereinstimmung gibt werden die Anweisungen des `otherwise` Blocks ausgeführt.

switchbsp.m

```
n=mod(floor(rand(1)*10), 9)+1

switch n
    case {1,4,9}
        disp('ist␣Quadratzahl');
    case {2,3,5,7}
        disp('ist␣Primzahl');
    case {6}
        disp('hat␣2␣Primfaktoren');
    otherwise
        disp('ist␣Kubikzahl');
end
```

```
>> switchbsp
n =
     2
ist Primzahl
```

for Schleife

- Syntax:

```
for <Variable>=<Matrix>
        <Anweisung>
end
```

- In der for Schleife wird der Variablen nacheinander die Spalten der Matrix zugewiesen und die Anweisungen ausgeführt.
- In einer for Schleife kann mit `continue` zur nächsten Zuweisung gesprungen und mit `break` der Schleifendurchlauf beendet werden.

forbsp.m

```
% Berechet Fibonacci Zahlen

n=6;
f=[0, 1];

for i=2:n
    f=[f, f(i)+f(i-1)];
end

disp(f);
```

```
>> forbsp
     0
     1
     1
     2
     3
     5
     8
```

while Schleife

- Syntax:

```
while <Ausdruck>
        <Anweisung>
end
```

- Durch `break` bzw. `continue` kann wieder die Schleife beendet bzw. zur Überprüfung des Ausdrucks gesprungen werden.

whilebsp.m

```
% Berechnet Naeherung von e

e=1;
n=1;

while abs(e-exp(1))>0.1
    e=e+1/factorial(n)
    n=n+1
end
```

```
>> whilebsp
e =
     2
n =
     2
e =
    2.5000
n =
     3
e =
    2.6667
n =
     4
```

Performance - Schleifen vermeiden

- In vielen Fällen können Scheifen in Matlab durch Anwendung von Vektorbefehlen umgangen werden. In den meisten Fällen sind Vektorfunktionen deutlich schneller als entsprechende Operationen, die mit Schleifen durchgeführt werden.

Loop_vs_Vector1.m

```
n=1000000;
summe=0;
x=[1:n];
disp('Summenberechnung');
disp('...mit For-Schleife');
tic
for i=x
    summe=summe+1/i;
end
toc
disp(summe)
```

Loop_vs_Vector2.m

```
n=1000000;
summe=0;
x=[1:n];
disp('Summenberechnung');
disp('...mit Vektrobefehl');
tic
summe=sum(1./x);
toc
disp(summe)
```

```
>> Loop_vs_Vector1
Summenberechnung
...mit For-Schleife
Elapsed time is 2.182345 seconds.
   14.3927
```

```
>> Loop_vs_Vector2
Summenberechnung
...mit Vektorbefehl
Elapsed time is 0.037873 seconds.
   14.3927
```

Performance - Vorinitialisieren

- Um die Performance von Anweisungen zu messen kann ein Zeitintervall mit den Funktionen `tic` und `toc` gemessen werden.
- Es ist effizienter große Matrizen und Vektoren mit Nullen zu initialisieren anstatt die Größe während der Ausführung zu ändern.

Dyn_vs_Stat1.m

```
disp('Dynamische␣Erweiterung:');
clear;
n=10000;
tic
f=[0; 1];
tic
for i=2:n
    f=[f; f(i-1)+f(i)];
end
toc
```

Dyn_vs_Stat2.m

```
disp('Statischer␣Vektor:');
clear;
n=10000;
f=[0; 1; zeros(n-1,1)];
tic
for i=2:n
    f(i+1)=f(i)+f(i-1);
end
toc
```

```
>> Dyn_vs_Stat1
Dynamische Erweiterung:
Elapsed time is 1.239487 seconds.
```

```
>> Dyn_vs_Stat2
Statischer Vektor:
Elapsed time is 0.001487 seconds.
```

Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Polynome, Zeichenketten etc.

Programmieren

Funktionen

Debuggen

m-Files Teil 2: Funktionen

- Für komplexere Funktionen können **Funktions-m-Files** verwendet werden. Hier werden, wie in Skripten, Anweisungen in einer Textdatei gespeichert, die der Reihe nach abgearbeitet werden.
- Um die Datei als Funktion zu kennzeichnen, wird die Funktion durch

  ```
  function <Ausgabeargumente> = <Funktionsname>(<Eingabeargumente>)
  ```

  eingeleitet.
- WICHTIG: Hierbei sollte `<Funktionsname>` mit dem Dateinamen übereinstimmen! Falls der Funktionsname und der Dateiname *nicht* übereinstimmen, wird der Dateiname zum Funktionsaufruf verwendet.

gerade.m

```
function [a, b] = gerade(x1, x2, y1, y2)
%GERADE Berechnet die Koeffizienten
% der linearen Funktion ax+b durch die
% Punkte (x1, y1) und (x2, y2)

a=(y2-y1)./(x2-x1);
b=y1-a.*x1;
```

```
>> [c1,c2]=gerade(0, 3, 1, 2)
c1 =
    0.3333
c2 =
     1
```

Funktions-Workspace

- Im Gegensatz zu Skripten wird eine Funktion in einem eigenen Workspace ausgeführt.
- In diesem Workspace werden Variablen unabhängig vom Matlab-Workspace angelegt und gelöscht. Nach dem Beenden der Funktion werden die Variablen des Funktionsworkspace gelöscht.
- Soll auf eine Variable des Matlab-Workspace zugegriffen werden, die nicht als Argument übergeben wird, so muss diese mit `global <Variablenname>` im Matlab-Workspace und im Funktions-Workspace sichtbar gemacht werden.
- Variablen, die nach beenden der Funktion ihren Wert behalten sollen, müssen mit `persistent <Variablenname>` deklariert werden.

modulfunc.m

```
function y=modulfunc(x)
modul=7;
global letzterAufruf;
persistent funcnt;
if isempty(funcnt)
    funcnt=1;
else
    funcnt=funcnt+1;
end
disp('Funktionsaufrufe:')
disp(funcnt);
letzterAufruf=datestr(now,13);
y=x.^2+3.*x+2;
y=mod(y,modul);
```

```
>> global letzterAufruf
>> letzterAufruf=0
letzterAufruf =
     0
>> modulfunc(3)
Funktionsaufrufe:
     4
ans =
     6
>> disp(letzterAufruf)
12:57:09
>> who
Your variables are:
ans               letzterAufruf
```

m-File Funktionen: Kommentare

- Mit % kann ein einzeiliger Kommentar und mit %{ }% ein Kommentarblock definiert werden. Kommentare mit % beginnen beim Kommentarzeichen und gehen bis zum Ende der Zeile. Bei der Definition eines Kommentarblocks mössen die %{ %} Zeichen jeweils in einer eigenen Zeile stehen.
- Die ersten zusammenhängenden Kommentarzeilen vor der ersten Anweisung werden beim Aufruf von `help <Funktionsname>` angezeigt.
- Die erste Kommentarzeile dieses Blocks wird beim Aufruf von `lookfor` durchsucht.

eratosthenes.m

```
function x = eratosthenes(n)

%ERATOSTHENES Berechnet alle Primzahlen bis zu einer oberen Schranke
%
% INPUT : n  Obere Schranke der Primzahlen
% OUTPUT: x  Vektor der Primzahlen der groesse kleiner oder gleich n
%
% Die Funktion berechnet Primzahlen mit dem Sieb des Eratosthenes

...
```

```
>> lookfor eratosthenes
ERATOSTHENES Berechnet alle Primzahlen bis zu einer oberen Schranke
```

m-File Funktionen: Ein- und Ausgabeargumente

- Eingabeparameter werden via call-by-value übergeben, d.h. Änderungen der Eingabevariablen in der Funktion ändert die beim Aufruf benutzte Variable nicht.
- Zur Verwendung einer Ein- oder Ausgabevariablenliste mit variabler Länge können in der Funktionsdefinition die Argumente `varargin` und `varargout` verwendet werden.
- Zur Abfrage der Anzahl von Ein- und Ausgabeargumenten können die Funktionen `nargin` und `nargout` verwendet werden.
- Mit `exist` kann geprüft werden, ob eine Variable definiert ist.
- Die Funktionen `nargchk` und `nargoutchk` überprüfen die Anzahl der Eingabe- bzw. Ausgabeparameter. Beispiel aus der Matlab-Hilfe:

mysize.m

```
function [s, varargout] = mysize(x)
msg = nargoutchk(1, 3, nargout);
if isempty(msg)
    nout = max(nargout, 1) - 1;
    s = size(x);
    for k = 1:nout, varargout(k) = {s(k)}; end
else
    disp(msg)
end
```

Funktion vs. Kommando

In Matlab wird zwischen Funktionsaufruf und Kommandoaufruf unterschieden.

- Bei Funktionsaufrufen werden die Argumente in runden Klammer aufgelistet. Dabei werden Variablen interpretiert und ihr Wert wird an die Funktion übergeben.

```
>> x=1:3
x =
     1     2     3
>> disp(x)
     1     2     3
```

- Bei Kommandoaufrufen werden die Argumente ohne Klammerung aufgelistet. Dabei werden die Argumente als Strings interpretiert.

```
>> disp x
x
```

- Viele Funktionen sind auch als Kommando aufrufbar und liefern dann mit dem falschen Aufruf unerwartete Ergebnisse.

Unterfunktionen

Beim Aufruf einer Funktion in einem m-File wird nur die erste Funktion der Datei ausgeführt. Zur Strukturierung können Unterfunktionen in einem m-File definiert werden. Diese können *nur* aus den Funktionen des m-Files aufgerufen werden.

funktion.m

```
function y = funktion(calls)
y=0;
for i=1:calls
    unterfunktion(i)
    y=y+i;
end

function []=unterfunktion(n)
fprintf(1, '%d-ter␣Aufruf\n', n);
```

```
>> funktion(2)
1-ter Aufruf
2-ter Aufruf
ans =
     3
```

Dabei hat jede Unterfunktion einen eigenen Workspace. Variablen, die in mehreren Funktionen verwendet werden sollen bzw. ihren Wert behalten sollen, müssen wieder als `global` oder `persistent` deklariert werden.

Verschachtelte Funktionen

In Matlab ist es auch erlaubt Funktionen in Funktionen zu definieren. In diesem Fall müssen die Funktionen mit einem `end` abgeschlossen werden.

funktion.m

```
function y = funktion(calls)
y=0;
for i=1:calls
    unterfunktion(i)
    y=y+i;
end

function []=unterfunktion(n)
    fprintf(1, '%d-ter␣Aufruf\n', n);
end

end
```

```
>> funktion(2)
1-ter Aufruf
2-ter Aufruf
ans =
     3
```

Die Funktionen haben dabei wieder einen eigenen Workspace, können aber auf den Workspace der umschliessenden Funktion zugreifen.

Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Polynome, Zeichenketten etc.

Programmieren

Funktionen

Debuggen

Debuggen von Skripten und Funktionen mit Standardfunktionen

Wie in anderen Programmiersprachen auch gibt es mehrere Methoden m-File-Funktionen und Skripte zu debuggen. Eine Möglichkeit ist das Einfügen von Ausgaben und Anweisungen zur Überprüfung:

- `echo`, `disp`, `fprintf` zur Ausgabe von Werten und zur Ablaufkontrolle;
- `size`, `who`, `whos` zur Überprüfung von Variablen im Workspace und deren Größe;
- `lasterr`, `lasterror`, `lastwarn` zur Ausgabe der letzten Fehlermeldung oder Warnung;
- `keyboard`, `input`, `return` zur Unterbrechung des Programmflusses durch Benutzereingaben.
- Mit `ginput` kann bei graphischen Benuzteroberlächen auf eine Mauseingabe gewartet werden. Die Funktion liefert dann die Koordinaten des Mauszeiger zurück.

Matlab Debugger

Darüber hinaus bietet Matlab einen Debugger, mit dem die Programmausführung an beliebiger Stelle unterbrochen, Kommandos einzeln ausgeführt und Werte von Variablen überprüft werden können.

- Mit `dbstop` und `dbclear` können Haltepunkte gesetzt und wieder entfernt werden, `dbstatus` zeigt die gesetzten Haltepunkte an.

```
>> dbstop in modulfunc at 11
>> dbstop in modulfunc at 3
>> dbstatus
Breakpoints for modulfunc are on lines 3, 11.
>> dbclear in modulfunc at 3
```

- Mit `dbtype` können m-Files mit Zeilennummern angezeigt werden.

```
>> dbtype modulfunc.m
1     function y=modulfunc(x)
2
3     modul=7;
4     global letzterAufruf
5     persistent funcnt
6     if isempty(funcnt)
7         funcnt=1;
8     else
      ...
```

Matlab Debugger (cont'd)

- Beim Aufruf der Funktion wird der Programmlauf an den Haltepunkten unterbrochen. Wird die Abarbeitung einer Funktion an einem Haltepunkt unterbrochen, so können an dem Punkt beliebige Funktionsaufrufe durchgeführt werden.
- Mit `dbstep` und `dbcont` kann die Funktion in Einzelschritten oder bis zum nächsten Haltepunkt fortgesetzt werden.

```
>> modulfunc(3)
11  disp('Funktionsaufrufe:')
K>> funcnt
funcnt =
     1
K>> dbstep
Funktionsaufrufe:
12  disp(funcnt);
K>> dbcont
     3
ans =
     6
```

- `dbstack` zeigt den Namen der aktuell ausgeführten Funktion an

```
K>> dbstack
> In modulfunc at 11
```

Matlab Debugger (cont'd)

- Mit den Funktionen `dbup` und `dbdown` kann zwischen dem aufrufendem und dem Funktions-Workspace gewechselt werden.

```
K>> dbup
In base workspace.
K>> who
Your variables are:
ans
K>> dbdown
In workspace belonging to modulfunc at 11
K>> who
Your variables are:
funcnt          modul
letzterAufruf   x
```

- Mit dbquit kann die Ausführung abgebrochen werden.

```
K>> dbquit
>>
```

Graphischer Debugger

Der Debugger kann auch über den Matlab-Texteditor bedient werden.

Editor – /Users/danieln/Desktop/Praktikum/Kursmaterial/Pr...

File Edit Text Go Cell Tools Debug Desktop Window Help

```
function y=modulfunc(x)

modul=7;
global letzterAufruf
persistent funcnt
if isempty(funcnt)
    funcnt=1;
else
    funcnt=funcnt+1;
end
disp('Funktionsaufrufe:')
disp(funcnt);
letzterAufruf=datestr(now, 13);
y=x.^2+3.*x+2;
y=mod(y,modul);
```

modulfunc Ln 1 Col 1