Matlab/Octave Kurzhilfe

Version 1.1, Copyright 2010 Boris von Loesch

Allgemeines

- `help, doc`: Aufrufen der Hilfe, `help <command>` gibt die Hilfe zu `<command>` aus.
- `;`: Um die Ausgabe eines Befehls zu unterdrücken, den Befehl mit Semikolon abschließen.
- `,`: Zum Trennen von mehreren Befehlen in einer Zeile wenn `;` nicht verwendet wird.
- `%, %{ ... }%`: `%` kommentiert den Rest der Zeile aus, längere Blöcke können mit `%{...}%` auskommentiert werden.
- `...`: Lange Zeile kann in mehrere Zeilen aufgeteilt werden, dabei jeweils `...` ans Ende schreiben.
- `ans`: Rückgabewert der letzten Operation
- `global`: Macht eine Variable global, muss vor der ersten Zuweisung stehen.
- `keyboard`: Unterbricht die Ausführung und wechselt in den Keyboard Mode in dem Variablen abgefragt oder verändert werden können.
- `Komplexe Zahlen`: Komplexe Zahlen werden im Syntax `<Realteil>+<Imaginärteil>i` eingegeben.

```
>> x=4+3i;                    >> x=...
>> x=4+3i                     4+3i;
x =                           x =
   4.0000 + 3.0000i              4.0000 + 3.0000i
```

- `tic, toc`: Misst die Ausführungszeit der Befehlen zwischen `tic` und `toc` und gibt sie aus.

Erzeugen von Matrizen

Befehle

- `eye, ones, zeros(n,m)`: Einheits-, Eins oder Nullmatrix mit `n` Zeilen und `m` Spalten. Hinweis: `ones(n)==ones(n,n)`

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

- `rand, randn(n,m)`: Zufallsmatrix mit gleich-, normalverteilten Einträgen in $(0, 1)$. Hinweis: `rand(1,1)==rand`.
- `diag(v,i)` Diagonalmatrix mit dem Vektor `v` auf Haupt- (`i`=0) oder `i`-ter Nebendiagonalen, Hinweis: `diag(v,0)==diag(v)`.

```
>> diag([1,3])                >> diag([1,3],-1)
ans =                         ans =
     1     0                        0     0     0
     0     3                        1     0     0
                                    0     3     0
```

- `repmat(A,m,n)`: Erzeugt eine Blockmatrix bestehend aus `m`×`n` Kopien von `A`. Hinweis: `repmat(A,n)==repmat(A,n,n)`

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

Doppelpunkt

- `<Start>:<Abstand>:<Ende>`: Erzeugt Vektor mit Elementen von `<Start>` bis `<Ende>` (inklusiv) mit Abstand `<Abstand>` (kann auch negativ sein). Hinweis: `<Start>:<Ende>==<Start>:1:<Ende>`.

```
>> 2:3.1:9                    >> 9:-2:3
ans =                         ans =
    2.0000    5.1000    8.2000       9       7       5       3
```

- `linspace(i,j,n)`: Erzeugt einen Vektor mit `n` Elementen von `i` bis `j` mit gleichem Abstand, `i` $\geq$ `j` ist zulässig.

Direkt

Vektoren und Matrizen können direkt erzeugt werden mit Hilfe von `[...]`. Die Einträge einer Zeile werden dabei mit `,` oder Leerzeichen getrennt; eine neue Zeile bekommt man durch `;` oder einem Zeilenumbruch. Die Anzahl der Einträge muss in jeder Zeile gleich sein.

```
>> [1,3,-1 4]                 >> [1,2;3,4]
ans =                         ans =
     1     3    -1     4            1     2
                                    3     4
```

Anstelle von Zahlen können auch Vektoren bzw. Matrizen verwendet werden (Blockmatrizen).

```
>> [ones(1,2); 2 5]           >> [eye(2) [3;3]; 2:4]
ans =                         ans =
     1     1                        1     0     3
     2     5                        0     1     3
                                    2     3     4
```

Indizierung

Unter Indizierung versteht man das direkte Ansprechen der Elemente eines Feldes (Vektor oder Matrix). **Achtung:** Die Indizes müssen immer ≥ 1 sein.

In den Beispielen ist stets `A = [1,2,3,4;5,6,7,8]` und `x = 2:5`.

- `x(i)`: Das `i`-te Element des Vektors.
- `A(i,j)`: Das Element in der `i`-ten Zeile und `j`-ten Spalte der Matrix
- `A(i)`: Das `i`-te Element der Matrix wobei die Einträge **spaltenweise** durchnummeriert sind.

```
>> A(2)                       >> A(end)
ans =                         ans =
     5                             8
```

- `end`: Der letze Eintrag der jeweiligen Dimension. Achtung: Kann nur beim Indizieren verwendet werden.

```
>> x(end)                     >> A(1,end)
ans =                         ans =
     5                             4
```

Es können auch Vektoren und Matrizen zum Indizieren verwendet werden:

- `x(v), x(C)`: Die Elemente von `x` deren Indices im Vektor `v` bzw. der Matrix `C` stehen.

```
>> x(2:end)                   >> x([1,2;2,3])
ans =                         ans =
     3     4     5                 2     3
>> x([1,2,1,3])                     3     4
ans =
     2     3     2     4
```

- `A(v,w)`: Matrix mit Elementen a_{ij} mit $i \in$ `v` und $j \in$ `w`. Hinweis: `:==1:end`.

```
>> A(2,[2,3,1,2])             >> A(:,2:end)
ans =                         ans =
     6     7     5     6            2     3     4
                                    6     7     8
```

- `A(v), A(C)`: Einträge der Matrix mit Indices aus `v` bzw. `C`, wobei die Einträge der Matrix spaltenweise durchnummeriert sind.
- `x(l), A(L)`: Indizieren mit logischen Feldern, siehe *Relationale und logische Operatoren*

Felder verändern

- `reshape(A,i,j)`: Gibt ein neues Feld mit geänderten Dimensionen zurück. **Achtung:** Anzahl der Elemente muss gleich `i*j` sein. Hinweis: `i` oder `j` kann auch durch `[]` ersetzt werden.

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

- `x(i)=[]`: Löschen des `i`-ten Elements eines Vektors
- `A(i,:)=[], A(:,j)=[]`: Löschen der `i`-ten Zeilen bzw. `j`-ten Spalte einer Matrix.
- `x(end+i)=s`: Verlängert den Vektor um `i` Null Einträge und setzt den letzen auf `s`. **Achtung:** Langsame Operation!

Operatoren

- `+, -, .*, ./, .^`: Elementweise Operationen, angewendet auf zwei Felder gleicher Dimensionen oder auf Skalar und Feld. **Achtung:** Punkt bei `.*, ./, .^` nicht vergessen!

```
>> x.*2                       >> A(:,[2,3])+A([2,1],[4,1])
ans =                         ans =
     4     6     8    10            10     8
>> x+x.^2                           10     8
ans =
     6    12    20    30
```

- `', .'`: Adjungierte bzw. transponierte Matrix. Die Operationen unterscheiden sich nur bei Matrizen mit komplexen Einträgen.
- `*, ^`: Operationen im Matrix-Matrix- bzw. Matrix-Vektor-Produkt Sinn.

```
>> A*x'            >> x*x'          >> A(:,[2,3])^2
ans =              ans =            ans =
    40                 54               22    27
    96                                  54    67
```

- `\, /`: Lösen von linearen Gleichungssystemen bzw. Bestimmen einer kleinste Quadrate Näherung. `A\b` entspricht $A^{-1} \cdot b$ und `b/A` entspricht $b \cdot A^{-1}$ (Dimensionen beachten).

```
>> B=ones(2)+eye(2);b=[2;3];        >> b=[2,3];
>> y=B\b           >> B*y           >> y = b/B
y =                ans =            y =
    0.3333             2.0000           0.3333    1.3333
    1.3333             3.0000
```

Relationale und logische Operatoren

Die relationen Operatoren sind `<, <=, >, >=, == , ~=` (ungleich). Rückgabe ist ein logisches Feld mit Nullen (falsch) und Einsen (wahr). **Achtung:** `=` ist immer eine Zuweisung, nie ein Vergleich!
Die logischen Operatoren sind `& (and), | (or), xor, ~ (not)`.

```
>> A>3
ans =
     0     0     0     1
     1     1     1     1
>> x>3 & x<5
ans =
     0     0     1     0
```

```
>> xor(x>3, x<5)
ans =
     1     1     0     1
>> x>(log(x)+2)
ans =
     0     0     1     1
```

- `any(l)`: wahr (`1`), wenn der logische Vektor `l` mind. eine `1` enthält.
- `all(l)`: wahr (`1`), wenn der logische Vektor `l` nur `1`sen enthält.

```
>> any(x>3 & x<5)
ans =
     1
```

```
>> all(x>3 & x<5)
ans =
     0
```

- `x(l)`: Gibt einen Vektor mit Einträgen aus `x` zurück, für die die Einträge in `l` wahr sind.
- `find(l)`: Indizes der wahren Einträge eines Feldes.

```
>> x(x>2 & x<5)
ans =
     3     4
```

```
>> find(x>2 & x<5)
ans =
     2     3
```

Einfache Funktionen

- `length`: Gibt die größte Dimension eines Feldes zurück

```
>> length(x)
ans =
     4
```

```
>> length(A)
ans =
     4
```

- `size`: Gibt die Dimensionen eines Feldes als Vektor zurück
- `numel`: Gibt die Anzahl Elemente eines Feldes zurück

Hinweis: `numel(A)==prod(size(A))==length(A(:))`

Skalare Funktionen

Skalare Funktionen bilden von $\mathbb{R}$ bzw. $\mathbb{C}$ nach $\mathbb{R}$ bzw. $\mathbb{C}$ ab. Ist das Eingabeargument ein Vektor oder eine Matrix, wird die Funktion komponentenweise angewendet. Beispiele sind `sin, sqrt, exp, log, factorial` (Fakuktät), `gamma, abs` (Betrag), `...`

Vektorfunktionen

Der Definitionsbereich dieser Funktionen ist $\mathbb{R}^n$ bzw. $\mathbb{C}^n$. Übergibt man eine Matrix, wird die Funktion auf **die Spalten der Matrix** angewandt. Möchte man die Funktion auf die Matrix als Vektor anwenden, muss als Eingabeargument `A(:)` statt `A` übergeben werden. Beispiele sind

- `min, max`: Gibt das kleinste bzw. größte Element eines Vektors zurück.
- `mean, median`: Mittelwert bzw. Median
- `sort`: Sortiert den Vektor
- `sum, prod`: Summe bzw. Produkt der Komponenten
- `diff`: Differenz je zwei aufeinanderfolgender Elemente
- `any, all`: Siehe *Relationale und logische Operatoren*

```
>> prod(x)
ans =
   120
>> diff(x)
ans =
     1     1     1
```

```
>> sum(A)
ans =
     6     8    10    12
>> sum(A(:))
ans =
    36
```

Schleifen

- `for`-Schleife: Syntax (`v` Vektor):

```
for i=v
   <Körper>
end
```

  Wiederhole den Code in `<Körper>` `length(t)` mal, dabei ist `i` in der j-ten Wiederholung gleich dem j-te Element von `v`. `v` ist häufig von der Form `1:N`.
  Achtung: `for`-Schleifen können häufig durch deutlich schnellere vektorwertige Ausdrücke ersetzt werden.
- `while`-Schleife: Syntax

```
while (<logischer Ausdruck>)
   <Körper>
end
```

  Wiederholt den Schleifenkörper solange bis `<logischer Ausdruck>` falsch ist. **Achtung:** Gefahr einer Endlosschleife!
- `break`: Springt aus der aktuellen Schleife
- `continue`: Beendet die aktuelle Iteration und geht zurück zum Schleifenkopf.

Bedingte Ausführung

```
if (<logischer Ausdruck 1>)
   <Körper 1>
elseif (<logischer Ausdruck 2>)
   <Körper 2>
elseif ...
else
   <Körper 3>
end
```

Der Code in `<Körper 1>` wird ausgeführt wenn `<logischer Ausdruck 1>` wahr ist, danach wird der Block verlassen. Wenn `<logischer Ausdruck 1>` falsch ist, werden die logischen Ausdrücke der `elseif` Teile untersucht (sofern vorhanden), wenn einer wahr ist, wird der zugehöriger Code ausgeführt und der Block verlassen. Wenn kein Zweig wahr ist wird der zum `else`-Teil gehörige Code ausgeführt (sofern vorhanden).

Lineare Algebra

- `det`: Determinante einer quadratischen Matrix
- `lu, qr, chol, svd`: LR-, QR-, Cholesky- und Singulärwertzerlegung.
- `norm, normest, cond, condest`: Norm, bzw. genäherte Norm einer Matrix/Vektors. `cond` berechnet die Kondition einer Matrix, mit `condest` kann sie günstiger genähert werden.
- `eig`: Eigenwerte und Eigenvektoren einer quadratischen Matrix
- `inv, pinv`: Inverse bzw. Pseudoinverse einer Matrix. **Achtung:** Lineare Gleichungssysteme nicht mit Hilfe von `inv` lösen, sondern `\` bzw. `/` verwenden.

Funktionen definieren

Eine Funktion `<fName>` die von Außen aufgerufen werden soll muss in einer Datei mit dem Namen `<fName>.m` liegen. Jede von Außen verfügbare Funktion muss also in eine einzelne Datei. Syntax:

```
function [A1, A2, ...] = <fName> (E1, E2, ...)
   %FNAME <Kurzbeschreibung>
   %<ausführliche Beschreibung,
   %wird bei "help fName" angezeigt >
   <Körper in dem die Variablen A1, A2,...
   richtig besetzt werden>
```

Hinweis: Wenn nur ein Ausgabeargument verwendet wird, können die `[ ]` weggelassen werden. Beispiel:

```
<Dateiname: fakultaet.m>
function r=fakultaet (n)
r=prod(2:n);
```

```
>> which fakultaet
<Pfad der Datei>/fakultaet.m
>> fakultaet(10)
ans =
     3628800
```

Kurze Funktionen können auch direkt mit
`<fName> = @(E1, E2,...) <Ausdruck>;`
definiert werden ohne eine eigene Datei zu schreiben.
Funktionen können selber als Argument übergeben werden, dazu muss `@` vor den Funktionsnamen geschrieben werden. Aufrufen kann man diese Funktionen dann durch Aufruf der Variable oder in älteren Versionen durch `feval (<handle>, <arg1>, <arg2>, ...)`.

```
>> fakultaet2 = @(n) prod(2:n);
>> fakultaet2(10)
ans =
     3628800
```

```
>> fun = @(f,n) f(1:n);
>> fun(@prod, 10)
ans =
     3628800
```

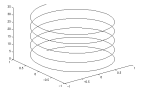
Plotten

- `plot`: Plotten von Funktionswerten.

```
>> x=0:0.1:2*pi;y=sin(x);
>> plot(x,y);
```

- `semilogx, semilogy, loglog`: Plotten mit logarithmischen Achsen
- `subplot`: Mehrere Plots in einem Fenster
- `hold`: Nach `hold on` plotten alle nachfolgenden Operationen in das gleiche Fenster. `hold off` (Standard) erzeugt ein neues Fenster für jeden Plotbefehl.
- `plot3`: Plotten von Funktionswerten in 3-D (Kurven $\mathbb{R} \to \mathbb{R}^3$)

```
>> t=0:0.1:10*pi;x=sin(t);y=cos(t);z=t;
>> plot3(x,y,z);
```



- `[X,Y]=meshgrid(x,y)`: Erstellt eine Diskretisierung eines quadratischen Gebietes nach Angabe der Diskretisierung des Randes. Hinweis: Wird in der Regel zum Plotten von $\mathbb{R}^2 \to \mathbb{R}$ Funktionen gebraucht.
- `contour`: Zeichnen der Höhenlinien von $\mathbb{R}^2 \to \mathbb{R}$ Funktionen.
- `mesh, meshc, surf, surfc(X,Y,Z)`: 3-D Plot von Funktionen $\mathbb{R}^2 \to \mathbb{R}$.

```
>> x=-1:0.1:1;y=-1:0.1:1;
>> [X,Y]=meshgrid(x,y);
>> Z=X.^2-Y.^2;
>> surf(X,Y,Z)
```