Matlab-Syntax in a nutshell:

Stand: 20.10.2014

GRUNDLAGEN:

Variablen:

Variablennamen bestehen aus Buchstaben, Ziffern und Unterstrichen. Sie beginnen grundsätzlich mit einem Buchstaben. Groß- und Kleinbuchstaben werden unterschieden. Sie können maximal 63 Zeichen lang sein.

a=7	Generiert eine Variable *a* mit Wert 7, oder ändert den Wert der Variablen *a* in 7, falls *a* bereits eine bekannte Variable ist. Ein einfaches Gleichheitszeichen bezeichnet in Matlab eine Definition, "a gesetzt gleich 7".
clear a	Löscht die Variable a aus dem aktuellen Workspace (der Kommandozeile oder einer Funktion).
clear all	Löscht alle Variablen des aktuellen Workspace.

Vektoren:

v=[1 4 78]	Generiert einen Zeilenvektor *v* mit drei Komponenten (die Komponenten innerhalb einer Zeile können entweder durch Leerzeichen oder durch Kommas voneinander getrennt werden.) Eckige Klammern umschließen generell die zusammengehörigen Komponenten eines Vektors oder einer Matrix.
v=[1; 4; 78; 15.8; 0]	Generiert einen Spaltenvektor *v* mit fünf Komponenten (Zeilen werden durch Semikolon getrennt).
l=[]	Generiert einen leeren Vektor *l*.
zaehl=1:10	Generiert einen Vektor *zaehl* mit 10 Komponenten, die mit Werten von 1 bis 10 belegt werden.
rueck=10:-2:1	Generiert einen Vektor mit 5 Komponenten, der von 10 bis 1 in Zweierschritten rückwärts zählt.
neu=[zaehl rueck]	Generiert einen Vektor mit 15 Komponenten, indem die Vektoren *zaehl* und *rueck* zu einem langen Vektor aneinandergehängt werden.
l=linspace(a,b,n)	Generiert einen Vektor *l* mit *n* Komponenten, die mit jeweils gleichem Abstand zwischen *a* und *b* liegen.
l=linspace(a,b)	Generiert einen Vektor mit 100 Komponenten zwischen *a* und *b*.
b=v(2)	Generiert eine Variable *b*, welcher der Wert der zweiten Komponente von *v* zugewiesen wird.

v(2)	Weist der Standard-Variablen *ans* den Wert der zweiten Komponente von *v* zu.
c=v(1:3)	Generiert einen Vektor *c* mit drei Komponenten, denen die ersten drei Werte von *v* zugewiesen werden.
d=v(end)	Generiert eine Variable *d*, welcher der letzte Wert des Vektors *v* zugewiesen wird.
alles=v(:)	Setzt die Variable alles mit Vektor v gleich. Andere Schreibweisen sind *alles=v* oder *alles=v(1:end)*.
zeil=v'	Transponiert *v*: generiert einen Zeilenvektor *zeil*, dem alle Werte des Spaltenvektors *v* zugewiesen werden. (Funktioniert genauso für Spalte nach Zeile).
L=length(v)	Weist der Variablen *L* die Anzahl der Komponenten von *v* zu.

Matrizen:

m=[1 5; 79 0.5; 17 0]	Generiert eine Matrix mit drei Zeilen und zwei Spalten (Zeilen sind durch Semikolon getrennt, die Werte innerhalb einer Zeile durch Leerzeichen oder Komma).
z=zeros(3,4)	Generiert eine Matrix mit drei Zeilen und vier Spalten, deren Werte alle 0 sind.
o=ones(3,4)	Entsprechend, aber alle Werte sind 1.
e=m(2,1)	Weist der Variablen e den Wert der zweiten Zeile und ersten Spalte von *m* zu.
e=m(2)	Weist ebenfalls der Variablen e den Wert der zweiten Zeile und ersten Spalte zu. Bei der linearen Indizierung wird entlang der Spalten gezählt. (*e=m(4)* wäre in diesem Beispiel der Wert der ersten Zeile und zweiten Spalte.)
zeile1=m(1,:)	Erzeugt einen Vektor zeile1 mit allen Werten der ersten Zeile von *m.*
[zeilen,spalten]=size(m)	Weist der Variable *Zeilen* die Anzahl der Zeilen, der Variablen spalten die Anzahl der Spalten in Matrix *m* zu.
zeilen=size(m,1)	Weist der Variable *Zeilen* die Größe der ersten Dimension zu, also die Anzahl der Zeilen. Entsprechend für die anderen Dimensionen einer Matrix (*spalten=size(m,2)*, *tiefe=size(m,3)* etc)
l=length(m(:))	Weist der Variablen *l* die Anzahl aller Elemente der Matrix *m* zu.

mt=m'	Transponiert die Matrix *m*. Die erste Zeile von *m* wird zur ersten Spalte von *mt*, etc.
B = repmat(A,m,n)	Repliziert die Matrix *A*. *B* ist eine Matrix, in der *A* *m*-mal "untereinander" und *n*-mal "nebeneinander" enthalten ist.
B = reshape(A,m,n)	Sortiert die Einträge von Matrix *A* in die aus *m* Zeilen und *n* Spalten bestehende Matrix *B*. (Spaltenweise durchgezählt)
B=squeeze(A)	Wird angewandt auf eine mehrdimensionale Matrix A, die 1-er Dimensionen enthält. Gibt eine niedriger dimensionale Matrix B zurück, bei der diese Dimensionen fehlen. Z.B. aus 3x1x3 wird eine 3x3 Matrix.

Operatoren

Arithmetische Operatoren

*+, - *, /, ^*	Wenden arithmetische Operation auf ganze Matrix bzw ganzen Vektor an
., ./ , .^*	Berechnen arithmetische Operation punktweise für jedes Element

Vergleichsoperatoren

<, >, <=, >=, ==	Geben Wahrheitswerte zurück, *1* für wahr, *0* für falsch. == ist der Test auf Gleichheit (im Gegensatz zu =, was "gesetzt gleich" bedeutet).
a=(2<3); *b=([0.5 7]==[0.5 8]);*	Bei Vektoren und Matrizen erfolgt der Vergleich punktweise
l=isequal(a,b)	Weist der Variablen *l* den Wahrheitswert für den Test auf Gleichheit von *a* und *b* zu. *a* und *b* müssen keine Zahlen, Vektoren oder Matrizen sein, *isequal* funktioniert für alle Datentypen (z.B. auch strings und Strukturen).

Logische Operatoren:

Punktweiser Vergleich, der Wahrheitswerte liefert.

&	logisches Und
|	logisches Oder
xor()	logisches Entweder-Oder
~	logische Negation

Mathematische Funktionen

a=abs(m)	Absolutbetrag, komponentenweise berechnet von Matrix *m*
a=sqrt(m)	Quadratwurzel, komponentenweise berechnet von Matrix *m.*
a=exp(m)	Exponentialfunktion, komponentenweise berechnet von Matrix *m.*
a=log(m)	Natürlicher Logarithmus, komponentenweise berechnet von Matrix *m.*
a=log10(m)	Logarithmus zur Basis 10, komponentenweise berechnet von Matrix *m.*
a=sin(m), a=cos(m)	Sinus und Kosinus, komponentenweise berechnet von Matrix *m.*
a=sum(m)	*a* ist die spaltenweise aufsummierte Matrix *m* (*a* ist ein Zeilenvektor).
a=sum(m,dim)	*a* ist die entlang der Dimension *dim* aufsummierte Matrix *m.*
a=rem(m,n)	Restbeträge wenn die Matrix *m* komponentenweise ganzzahlig durch Matrix *n* geteilt wird.

MATLAB-FUNKTIONEN

Funktionen mit fester Anzahl Ein- und Ausgaben:

Kopfzeile im Programm:

function [Ausgabe1,Ausgabe2] = Funktionsname (Eingabe1, Eingabe2)

Speicherung der Funktion in der Datei Funktionsname.m

Aufruf von Command window oder anderer Funktion / Script:

[Ausgabe1,Ausgabe2] = Funktionsname (Eingabe1, Eingabe2)

function	Schlüsselbegriff *function* in der Kopfzeile bewirkt, dass das Programm als Funktion mit gekapseltem Workspace verwendet wird.
[Ausgabe1,Ausgabe2]	Liste der Ausgabeparameter, die von der Funktion übergeben werden, also dort belegt werden müssen.
(Eingabe1, Eingabe2)	Liste der Eingabeparameter, die an die Funktion übergeben werden und dort genutzt werden können. Die Anzahl der Ein- und Ausgaben müssen im Programmkopf und im Aufruf übereinstimmen, sonst gibt es eine Fehlermeldung.

Funktionen mit variabler Anzahl Ein- und Ausgaben:
Kopfzeile im Programm: *function [varargout] = Funktionsname (varargin)*

function	Schlüsselbegriff *function* in der Kopfzeile bewirkt, dass das Programm als Funktion mit gekapseltem Workspace verwendet wird.
[varargout]	Schlüsselbegriff *varargout* bewirkt, dass abhängig vom Aufruf der Funktion unterschiedlich viele Ausgabeparameter übergeben werden können. *varargout* ist selber ein cell array, das im Programmtext belegt werden muss.
varargout{1}=x	kopiert den Inhalt der Variable *x* an die erste Stelle der zu übergebenden Ausgabeparameter
nargout	Schlüsselbegriff *nargout* ist eine Variable, die in jeder Funktion automatisch verfügbar ist und die Anzahl der im Programmaufruf angeforderten Ausgaben angibt.
(varargin)	Schlüsselbegriff *varargin* bewirkt, dass abhängig vom Aufruf der Funktion unterschiedlich viele Eingabeparameter übergeben werden können. *varargin* ist selber ein cell array, das im Programmtext verwendet werden sollte.
x=varargin{1}	kopiert den Inhalt der ersten übergebenen Eingabe in die Variable *x*
nargin	Schlüsselbegriff *nargin* ist eine Variable, die in jeder Funktion automatisch verfügbar ist und die Anzahl der im Programmaufruf übergebenen Eingaben angibt.

Kommentare in Funktionen und Skripten:

%	Schlüsselzeichen, dass alles weitere in einer Programmzeile als Kommentar gewertet wird.
%%	Schlüsselzeichen für den Beginn eines neuen Programmteils (cell).

KONTROLLFLUSS

Fallunterscheidungen:

IF für wenige unterschiedliche Fälle, *SWITCH* für viele Fälle:

Allgemeine Syntax	**Beispiel**
if Bedingung * Befehle* *end*	*if a==b* * c=a/2;* *end*

if Bedingung * Befehle1* *else* * Befehle2* *end*	*if a==b* * c=a/2;* *else* * c=0;* *end*
if Bedingung1 * Befehle1* *elseif Bedingung2* * Befehle2* *else* * Befehle3* *end*	*if a==b* * c=a/2;* *elseif a>b* * c=b/2;* *else* * c=0;* *end*
switch Ausdruck *case Fall1* * Befehle1* *case {Fall2, Fall3}* * Befehle2* *otherwise* * Befehle3* *end*	*str='a'* *switch str* *case 'a'* * x=1* *case {'b','c'}* * x=2* *otherwise* * x=0* *end*

Schleifen

Allgemeine Syntax	**Beispiel**
ZÄHLSCHLEIFE: *for variable=Ausdruck* * Befehle* *end*	*o=ones(1,10)* *for a=2:10* * o(a)=2*o(a-1);* *end*
BEDINGTE SCHLEIFE: *while Bedingung* * Befehle* *end*	*o=1;* *a=1;* *while o(a)<1000* * a=a+1;* * o=[o 2*o(a-1)];* *end*
DOPPELPUNKT ALS IMPLIZITE SCHLEIFE: *var=[startwert:schrittweite:endwert]*	*a=[1:0.1:2]*

Folgende Möglichkeiten erzeugen den gleichen Vektor *a*:

Doppelpunkt:	**Zählschleife:**	**Bedingte Schleife:**
a=[1:0.1:2]	*a=1;* *for b=1:10* *a=[a a+0.1*b];* *end*	*a=1* *while a(end)<2* *a=[a a(end)+0.1];* *end*

DATENFLUSS:

Dateien-Verwaltung:

save filename	Sichert alle aktuellen Variablen im file *filename.mat* im aktuellen Verzeichnis.
save('filename','var1','var2') alternativ: *save filename var1 var2*	Sichert nur Variablen *var1* und var2 im file *filename.mat*.
save -ascii filename.txt	Sichert alle aktuellen Variablen als Textdatei *filename.txt*, die von einem beliebigen Editor eingeladen und weiterverarbeitet werden kann.
load('filename')	Holt alle in filename.mat gespeicherten Variablen in den Workspace.
cd subfolder	Wechselt das aktuelle Verzeichnis zum Unterordner *subfolder*.
cd ..	Wechselt das aktuelle Verzeichnis zum übergeordneten Verzeichnis.
fscanf	Mächtige Möglichkeit jegliche formatierte Daten einzulesen (viele Optionen, bei Bedarf bitte in der Hilfe nachsehen).
fprintf	Mächtige Möglichkeit Daten formatiert abzuspeichern (viele Optionen, bei Bedarf bitte in der Hilfe nachsehen).
xlswrite('datei.xls',var)	Speichert eine Matlab-Variable in einer xls-Datei ab, um sie mit Excel weiter zu verarbeiten.
var=xlsread('datei.xls')	Lädt eine mit Excel erzeugte Datei in den Matlab Workspace ein.

Bildschirmaus- und Eingaben

;	Hinter einem Befehl unterdrückt die Bildschirmausgabe.
...	Unterbricht eine Programmzeile. Dadurch lassen sich sehr umfangreiche Befehle lesbarer schreiben (ansonsten führt Matlab eine Programmzeile aus, sobald sie mit Return abgeschlossen ist).
echo on	Wiederholt den jeweils ausgeführten Befehl auf dem Bildschirm.
echo off	Stellt echo wieder ab.
clc	Löscht alle aktuellen Bildschirmausgaben, so dass das Eingabefenster wieder leer ist.
pause	Wartet mit der Ausführung des nächsten Befehls auf einen Tastendruck.
pause(10)	Wartet mit der Ausführung des nächsten Befehls 10sec.
disp('hello world') *disp(v)*	Stellt eine Matrix oder Zeichenkette auf dem Bildschirm dar, ohne den Arraynamen darzustellen.
fprintf('hello world \n')	Formatierte Darstellung von Zeichenkette auf dem Bildschirm dar, ohne den Arraynamen darzustellen.
msgbox('Text in der Box')	Öffnet ein Ausgabefenster, in dem der übergebene Text dargestellt wird.
warning('Achtung!')	Stellt eine Warnung auf dem Bildschirm dar, der Ablauf des Programms wird dadurch nicht beeinflusst.
error('Achtung, Fehler!')	Stellt eine Fehlermeldung rot geschrieben auf dem Bildschirm dar und bricht das Programm ab.
type('filename')	Stellt den Inhalt der Datei auf dem Bildschirm dar.
lookfor Begriff	Durchsucht alle Hilfetexte nach dem angegebenen (englischen) Begriff und stellt jeweils die erste Zeile des Textes auf dem Bildschirm dar.
format long, *format long e,* *format short,* *format short e*	Umstellung, wie viele Nachkommastellen bei Fließkommazahlen angezeigt werden.
antwort=input('bitte Text eingeben','s')	Liest eine Benutzereingabe von der Tastatur als Zeichenkette in Variable *antwort* ein.

antwort=input('bitte Zahl eingeben')	Liest eine Benutzereingabe von der Tastatur in Variable *antwort* ein (normalerweise benutzt für Zahlen).
antwort=inputdlg('bitte Zahl eingeben: ')	Öffnet eine Dialogbox mit dem angegebenem Text und liest eine Benutzereingabe in die Variable *antwort* ein (Mehr Optionen siehe Hilfe)

DATENTYPEN

Fließkommazahlen:

double	Fließkommazahlen mit doppelter Präzision, Standardtyp für Zahlen in Matlab
realmax	Größte positive Fließkommazahl, mit der Matlab rechnen kann.
realmin	Kleinste positive Fließkommazahl, mit der Matlab rechnen kann.
eps	Kleinster positiver Unterschied zwischen zwei Fließkommazahlen, den Matlab als Unterschied erkennt

Umwandlung von Datentypen:

double(v)	Wandelt eine Variable (Vektor oder Matrix) in Fließkommazahlen um.
int8(v), int16(v), int32(v), int64(v)	Wandelt eine Variable (Vektor oder Matrix) in ganze Zahlen um, die mit jeweils maximal 8, 16, 32 oder 64 bit dargestellt werden.
logical(v)	Wandelt eine Variable (Skalar, Vektor oder Matrix) in Wahrheitswerte um.
char(v)	Wandelt eine Variable (Vektor) in eine Zeichenkette um.

Tests auf Datentypen:

b=islogical(a);	Gibt den Wahrheitswert *1* zurück, wenn *a* aus Wahrheitswerten besteht
b=isnumeric(a);	Gibt den Wahrheitswert *1* zurück, wenn *a* aus Zahlen besteht (Fließkommazahlen, ganze Zahlen, komplexe Zahlen).
b=isfloat(a);	Gibt den Wahrheitswert *1* zurück, wenn *a* aus Fließkommazahlen besteht.

b=ischar(a);	Gibt den Wahrheitswert *1* zurück, wenn *a* aus Zeichen besteht.
b=isa(a,'class-name');	Gibt den Wahrheitswert *1* zurück, wenn *a* dem mit der Option class-name angegebenen Datenyp entspricht, z.B. *logical, char, integer, int8, struct*.
b=isnan(a);	Gibt eine Matrix aus Wahrheitswerten mit den gleichen Dimensionen wie *a* zurück, für jeden Eintrag wird *1* zurückgegeben, wenn er *NaN* (not a number) - also keine darstellbare Zahl ist (z.B. nachdem man durch 0 geteilt hat).
b=isscalar(a);	Gibt den Wahrheitswert *1* zurück, wenn *a* ein skalarer Wert ist.
b=isvector(a);	Gibt den Wahrheitswert *1* zurück, wenn *a* ein Vektor ist. (Achtung, auch einzelne Werte sind Vektoren, Matrizen aber nicht).

Zeichenketten (string)

s='bla'	Erzeugt eine Variable *s* vom Typ *char*, die mit der Zeichenkette *'bla'* belegt wird.
b=blanks(10)	Erzeugt einen String aus 10 Leerzeichen.
s(2)	Zweites Element (Buchstabe) des strings *s*.
s=sprintf(format, A);	Schreibt den Inhalt von Matrix *A* formatiert in die Zeichenkette *s*.
fprintf(format, A);	Schreibt den Inhalt von Matrix *A* formatiert als Ausgabe in die Console
%g	Belegung für *format* in *sprintf* und *fprintf* kompakte Darstellung von Dezimalzahlen
%s	Belegung für *format* in *sprintf* und *fprintf :* Darstellung von strings
'\n'	Zeichen für Zeilenumbruch, wichtig z.B für formatierte Bildschirmausgaben
strcmp(s,'bla')	vergleicht string *s* mit string *'bla'* auf Gleichheit
strncmp(s,'bla',2) *ind=strfind(str,pattern)*	Vergleicht erste 2 Buchstaben der beiden Strings auf Gleichheit. Sucht string *pattern* in string *str* und gibt Startindex zurück.
antwort=input('bitte Text eingeben','s')	Liest eine Benutzereingabe von der Tastatur als Zeichenkette in Variable *antwort* ein.

Sparse Matrizen

S=sparse(M)	Generiert eine sparse-Matrix, bei der nur die von 0 verschiedenen Elemente von *S* gemeinsam mit ihren Indizes abgespeichert werden.
M=full(S)	Erzeugt aus einer sparse-Matrix *S* die zugehörige normale Matrix *M*.
issparse(s)	Test, ob eine Matrix sparse ist
find	finde Indizes der Elemente ungleich 0 (wie sonst auch)
nonzeros	Werte der Elemente ungleich 0 (wie sonst auch)
n=nnz(M)	Gibt die Anzahl der Werte der Elemente der Matrix *M* zurück, die ungleich 0 sind
speye	generiert sparse Identitätsmatrix
spfun	wendet Funktion auf Elemente ungleich 0 an.
sprand(S)	erzeugt eine sparse Matrix mit der gleichen Struktur wie bei Matrix *S*aber gleichverteilten zufälligen Elementen
sprandn(S)	erzeugt eine sparse Matrix mit der gleichen Struktur wie bei Matrix *S* aber normalverteilten zufälligen Elementen
spones	Ersetzt die Elemente ungleich 0 mit Einsen
spy	Visualisierung des *sparse*-Musters

KOMPLEXE DATENTYPEN

Strukturen:

strukturname.feldname=Belegung
strukturname(Nummer).feldname=Belegung

Die Felder von Strukturen können beliebige, auch verschiedene Datentypen beinhalten.
Bei Strukturen in Strukturen werden mehr Punkte verwendet: *S(2).unterS(3).feld1*

s=struct('f1',0.1,'f2','bla')	erzeugt eine Struktur *s* mit zwei Feldern mit Namen *f1* und *f2* und belegt diese mit den jeweils angegebenen Werten
c=struct2cell(s)	Überträgt den Inhalt der Struktur *s* in das cell array *c*
s=cell2struct(c,fields,dim)	Überträgt den Inhalt des cell array *c* in die Struktur *s* (siehe Hilfe)
L=isstruct(s)	Gibt einen logischen Wert zurück, ob die Variable *s* eine Struktur ist.
fieldnames(s)	Gibt die Namen der Felder der Struktur *s* zurück
L=isfield(s,'fname')	Gibt einen logischen Wert zurück, ob die Struktur *s* ein Feld mit Namen *fname* enthält
s=rmfield(s,'f1')	Löscht das Feld mit Namen *f1* aus der Struktur *s*
v=getfield(s,'f2')	Weist *v* den Inhalt des Feldes *f2* der Struktur *s* zu. Für eindimensionale Strukturen ist der Befehl äquivalent zu *v=s.f2* Für mehrdimensionale siehe Hilfe
s=setfield(s,'f2',v)	Belegt in Struktur *s* das Feld *f2* mit v als Inhalt. Für eindimensionale Strukturen ist der Befehl äquivalent zu *s.f2=v* Für mehrdimensionale siehe Hilfe

Cell Arrays:

cellname{Nummer}=Belegung
cellname{Zeile,Spalte}=Belegung

Die cells in cell arrays können beliebige, auch verschiedene Datentypen beinhalten.

c=cell(N)	erzeugt ein eindimensionales cell array mit *N* Zellen
c=cell(N,M)	erzeugt ein zweidimensionales cell array mit *NxM* Zellen
L=iscell(c)	Gibt einen logischen Wert zurück, ob die Variable *c* ein Cell Array ist.

c=num2cell(m)	Überträgt den Inhalt der Matrix *m* in ein cell array *c,* wobei jedes Matrixelement in eine eigene Zelle geschrieben wird
c=mat2cell(m,rdist,cdist)	Überträgt den Inhalt der Matrix *m* in ein cell array *c,* wobei *rdist* und *cdist* angeben, wie große Untermatrizen in den einzelnen Zellen stehen (s. Hilfe)
c=struct2cell(s)	Überträgt den Inhalt der Struktur *s* in das cell array *c*
s=cell2struct(c,fields,dim)	Überträgt den Inhalt des cell array *c* in die Struktur *s* (siehe Hilfe)
celldisp(c)	Gibt den Inhalt des cell arrays *c* auf dem Bildschirm aus

SUCHEN UND SORTIEREN

Logische Indizierung:

L=M>0	Erzeugt eine logische Matrix *L* mit den gleichen Dimensionen wie Matrix *M*, bei der für jedes Element von Matrix *M* an der gleichen Stelle eine logische *1* steht, wenn das Element positiv ist und sonst eine logische *0*.
v=M(L)	Wendet eine logische Matrix *L* auf eine Matrix *M* mit den gleichen Dimensionen an. Vektor >*v* enthält nur diejenigen Elemente von Matrix *M* , bei denen an der gleichen Stelle in *M* eine logische *1* steht.

Suchen:

ind=find(v)	Gibt einen Vektor aller Indizes des Vektors *v* zurück, die ungleich 0 sind.
ind=find(v,k)	Gibt einen Vektor der ersten *k* Indizes des Vektors *v* zurück, die ungleich 0 sind.
ind=find(v,k,'last')	Gibt einen Vektor der letzten *k* Indizes des Vektors *v* zurück, die ungleich 0 sind.
[row,col]=find(M,...)	Gibt jeweils einen Vektor Zeilen- und Spaltenindizes der Elemente der Matrix *M* zurück, die ungleich 0 sind.
[row,col,val]=find(M,...)	Gibt jeweils einen Vektor der Zeilen- und Spaltenindizes und Werte der Elemente der Matrix *M* zurück, die ungleich 0 sind.

val=nonzeros(M)	Gibt einen Vektor der Werte der Elemente der Matrix *M* zurück, die ungleich 0 sind
n=nnz(M)	Gibt die Anzahl der Werte der Elemente der Matrix *M* zurück, die ungleich 0 sind

Sortieren:

vs1=sort(v1)	sortiert die Elemente eines Vektors *v1* in aufsteigender Reihenfolge
ms1=sort(m,1)	sortiert die Elemente jeder Spalte der Matrix *m* in aufsteigender Reihenfolge(unabhängig voneinander)
ms2=sort(m,2)	sortiert die Elemente jeder Zeile der Matrix *m* in aufsteigender Reihenfolge(unabhängig voneinander)
msd=sort(m,1,'descend')	sortiert die Elemente jeder Spalte der Matrix *m* in absteigender Reihenfolge (unabhängig voneinander)
[ms1,index]=sort(m,1)	gibt zusätzlich zur sortierten Matrix die Indizes zurück
mr1=sortrows(m1,n)	sortiert die Zeilen der Matrix *m1* gemäß ihrer Einträge in der n- ten Spalte in aufsteigender Reihenfolge.
mdesc1=sortrows(m1,-n)	sortiert die Zeilen der Matrix *m1* gemäß ihrer Einträge in der *n*- ten Spalte in absteigender Reihenfolge.
mr_n1=sortrows(m1,[n,1])	sortiert die Zeilen der Matrix *m1* ihrer Einträge in der *n*-ten Spalte in aufsteigender Reihenfolge. Bei gleichen Werten in der *n*-ten Spalte werden diese Zeilen entsprechend der 1. Spalte sortiert.
[mr1,index]=sortrows(m1,n)	gibt zusätzlich einen Vektor der Indizes zurück.
L=issorted(m)	Gibt den logischen Wert *1* zurück, wenn *m* sortiert ist, also wenn *m==sort(m)*
L=issorted(m,'rows')	Gibt den logischen Wert *1* zurück, wenn die Zeilen von *m* gemäß der ersten Spalte sortiert sind, also wenn *m==sortrows(m)*

GRAFIK

Liniengraphik

plot(v)	Trägt die Werte des Vektors *v* gegen ihre Indizes auf.
plot(x,y)	Trägt die Werte des Vektors *y* gegen die des Vektors *x* auf.
plot(x1,y1,x2,y2)	Kombiniert die Auftragungen von *y1* gegen *x1* und *y2* gegen *x2* in einem Bild.
plot(x,y,'ro-')	Benutzt rote, mit Linien verbundene Kreise, um *y* gegen *x* aufzutragen (Reihenfolge der Formatierungsparameter ist egal - Parameter bitte in der Hilfe nachgucken).
plot(x,y,'color',[0.5 0.2 1])	Benutzt eine Linie mit selbst definierter Farbe, um *y* gegen *x* aufzutragen. Der Farbvektor setzt sich zusammen aus den Werten für *[rot grün blau]*,die jeweils zwischen 0 und 1 liegen. (Andere line properties bitte in der Hilfe nachgucken, Syntax ist grundsätzlich *plot(x,y,'property',Wert).* Es können mehrere properties kombiniert werden.
loglog(x,y)	Doppellogarithmische Auftragung von *y* gegen *x.*
semilogx(x,y)	Auftragung mit logarithmischer *x*- und linearer *y*-Achse (*semilogy* entsprechend).
errorbar(x,y,e)	Trägt *y* gegen *x* auf und versieht jeden Datenpunkt mit symmetrischen Fehlerbalken der Länge *2e* (jeweils e nach oben und unten).

Andere Graphik-Typen

pie(x), pie3(x)	Darstellung des Vektors *x* als Kuchengraphik bzw als 3-dimensionale Kuchengrafik.
mesh(x,y,z)	3-dimensionale Darstellung der Matrix *z* als Maschendrahtmodell mit Vektoren x und *y* als Achsen. *x* muss die Dimension *y*x* haben. (Mehr Möglichkeiten in der Hilfe).
surf(x,y,z)	3-dimensionale Darstellung der Matrix z als farbige Oberfläche mit Vektoren *x* und *y* als Achsen. *z* muss die Dimension *y*x* haben. (Mehr Möglichkeiten in der Hilfe).
boxplot(x)	Stellt *x* als Boxplot dar. Wenn *x* eine Matrix ist, wird für jede Spalte Median, Perzentile und Ausreißer berechnet und als eigene "Box" aufgetragen.

imagesc(m)	Zeigt die Elemente einer 2D-Matrix fabkodiert an. Bei der Standardeistellung der Farben (*colormap*) ist das kleinste Element dunkelblau, das größte rot
colormap(gray)	Legt die Farben der Farbkodierung im aktuellen Grafikfenster fest (in diesem Fall Graustufen). Farbauswahl kann aus einer vordefinierten Liste gewählt oder selber definiert werden. Siehe Hilfe.
colorbar *colorbar('off')*	Zeigt die Fabkodierung (colormap) im aktuellen Grafikfenster an. Löscht den colorbar aus dem aktuellen Grafikfenster.
fill(x,y,farbdef)	Füllt einen durch *x,y* definierten geschlossenen Linienzug mit der gewünschten Farbe aus. Farbdefinition entweder durch Kürzel für Standardfarben oder in der RGB-Darstellung
hist(v)	Stellt das Histogramm der Häufigkeit des Auftretens von Werten im Vektor *v* grafisch dar. Standardversion: Teilt den Wertebereich des Vektors hist ohne Ausgabeargument aufgerufen wird, stellt es die Häufigkeit des Auftretens der Klassen als Balkengrafik dar. Wenn Wenn *hist* mit einem Ausgabeargument aufgerufen wird, (*h=hist(v)* oder *[h,xout]=hist(v))* produziert es keine grafische Ausgabe, sondern gibt den Vektor der Häufigkeiten zurück.
hist(v,nbins)	Teilt den Wertebereich des Vektors *v* in *nbins* gleich große Klassen ein.
hist(v,centers)	Benutzt den Vektor *centers* als Mittelpunkte der Klassen, in die die Elemente von *v* aufgeteilt werden. Wenn *hist* ohne Ausgabeargument aufgerufen wird, stellt es die Häufigkeit des Auftretens der Klassen als Balkengrafik dar.

Graphik-Fenster und Formatierungen

figure(2)	Öffnet ein neues Graphikfenster und gibt ihm die Nummer 2.
close(2)	Schließt das zweite Graphikfenster.
clf	Inhalt des aktuellen Graphikfensters löschen.
subplot(m,n,p)	Teilt das aktuelle Graphikfenster in *m***n* Unterabbildungen (*m* nebeneinander, *n* untereinander) und bewirkt, dass beim nächsten *plot*-Befehl die *p*-te Unterabbildung verwendet wird (zeilenweise durchgezählt).
xlabel('bla bla')	Beschriftung der x-Achse (y entsprechend).

title('Tolles Bild')	Bildüberschrift.
legend('f1','f2','f3')	Erzeugt eine Abbildungslegende für das aktuelle Graphikfenster. Für jede Linie in einer Liniengraphik wird eine Zeichenkette übergeben.
axis([xmin xmax ymin ymax])	Setzt die Grenzwerte fest, zwischen denen Werte graphisch aufgetragen werden sollen.
axis equal	Macht das Graphikfenster quadratisch, indem die Achsen gleich skaliert werden
colormap(gray)	Legt die Farben der Farbkodierung im aktuellen Grafikfenster fest (in diesem Fall Graustufen). Farbauswahl kann aus einer vordefinierten Liste gewählt oder selber definiert werden. Siehe Hilfe.

ZUFALL UND STATISTIK

Zufall

r=rand(2,5) *r4=rand(4)*	Generiert eine 2x5-Matrix mit gleichverteilten Zufallszahlen zwischen 0 und1 Wenn nur ein Eingabeparameter übergeben wird, wird eine quadratische Matrix (hier 4x4) aus Zufallszahlen erstellt
rn=randn(2,5) *rn5=randn(5)*	Generiert eine 2x5-Matrix mit normalverteilten Zufallszahlen mit Mittelwert 0 und Standardabweichung 1. (Inputs wie oben)
ri=randi(100,2,4) *ri10=randi(100,10)*	Generiert eine 2x4 Matrix mit ganzzahligen Zufallszahlen zwischen 1 und 100. (Quadratische Matrix wie oben bei Übergabe nur einer Dimensionsgröße)
p=randperm(n)	Gibt einen Vektor *p* zurück, in dem die natürlichen Zahlen 1 bis *n* in zufälliger Reihenfolge stehen.

Beschreibende Statistik

mean(m,dim)	Mittelwert von Matrix *m* entlang Dimension *dim* (wenn keine Dimension angegeben wird: spaltenweise).
std(m,flag,dim)	Standardabweichung von m entlang Dimension *dim* (wenn keine Dimension angegeben wird: spaltenweise). Der Parameter flag entscheidet über die Normalisierung: *flag=0* bei Normalisierung über N-1, *flag=1* bei Normalisierung über N.

var(m, flag, dim)	Varianz von *m* entlang Dimension *dim* (wenn keine Dimension angegeben wird: spaltenweise). Der Parameter *flag* entscheidet über die Normalisierung: *flag=0* bei Normalisierung über N-1, *flag=1* bei Normalisierung über N.
median(m,dim)	Median von *m* entlang Dimension *dim* (wenn keine Dimension angegeben wird: spaltenweise).
Z=prctile(x,p)	Berechnet für den Datenvektor *x* (bzw für jede Spalte der Matrix *x*) das *p*-te Perzentil. (Statistik-Toolbox!)
min(m,dim), max(m,dim)	Minimum und Maximum entlang Dimension *dim* (wenn keine Dimension angegeben wird: spaltenweise).
h=hist(v,nbins)	Histogramm der Häufigkeit des Auftretens von Werten im Vektor *v* wobei *nbins* die Anzahl Klassen angibt, in die der Bereich zwischen dem minimalen und dem maximalen Wert von *v* aufgeteilt wird. Ohne Angabe des Ausgabearguments *h* wird das Histogramm auf dem Bildschirm dargestellt.
[h,xout]=hist(v)	Wenn *hist* mit zwei Ausgabeargumenten aufgerufen wird, produziert es keine grafische Ausgabe, sondern gibt zwei Vektoren zurück: den Vektor *h* der Häufigkeiten und den Vektor *xout* der Klassenmittelpunkte.

Hypothesentests

h=ttest(vektor,mittelwert)	testet, ob die Nullhypothese abgelehnt werden kann, dass die im Vektor *vektor* gespeicherten Messdaten einer Normalverteilung mit dem Mittelwert *mittelwert* entstammen. Das Standard-Signifikanznivieau ist 5%.
h=ttest(vektor, mittelwert, alpha)	wie oben, aber mit Angabe des Signifikanznivieaus *alpha*
h=ttest2(vektor1,vektor2)	testet, ob für zwei Stichproben *vektor1* und *vektor2* zum Standard-Signifikanzniveau 5% die Nullhypothese abgelehnt werden kann, dass beide Stichproben der gleichen Verteilung entstammen.
h=ttest2(vektor1, vektor2, alpha)	wie oben, aber mit Angabe des Signifikanznivieaus *alpha*

Für alle *ttest*-Funktionen gilt: Der **Rückgabewert** ist

- *1* wenn die Nullhypothese abgelehnt und somit die Alternativhypothese angenommen wird (Dann entstammt der zu testende Datensatz zu α% Wahrscheinlichkeit doch einer Verteilung mit dem zu testenden Mittelwert).
- *0* wenn die Nullhypothese nicht abgelehnt werden kann.

Kurvenanpassung

[p,S]=polyfit(x,y,n)	Findet das Polynom *n*-ten Grades, das die durch die Vektoren *x* und *y* gegebenen Messwerte optimal beschreibt, indem der quadratische Abstand zwischen Messwerten und Funktionswerten des Polynoms minimiert wird. *p* gibt die Koeffizienten des Polynoms $p_1x^n+p_2x^{n-1}+...p_nx+p_{n+1}$ an. *S* ist eine Struktur zur Verwendung mit *polyval* um die Güte der Kurvenanpassung abzuschätzen.
y=polyval(p,x)	Bestimmt die Funktionswerte für ein Polynom $p_1x^n+p_2x^{n-1}+...p_nx+p_{n+1}$, dessen Koeffizienten im Vektor *p* angegeben sind, für die im Vektor *x* angegebenen Argumente.
[y,delta]=polyval(p,x,S)	Bestimmt die Funktionswerte für ein Polynom $p_1x^n+p_2x^{n-1}+...p_nx+p_{n+1}$, dessen Koeffizienten im Vektor *p* angegeben sind, für die im Vektor *x* angegebenen Argumente. Zusätzlich wird mit Hilfe der von *polyfit* zurückgegebenen Struktur *S* im Vektor *delta* ein Maß für die Zuverlässigkeit der Schätzung angegeben. Der Bereich von *y-delta* bis *y+delta* enthält mindestens 50% der Schätzungen (bei Normalverteilteilung).

ZEITMESSUNG

tic; toc	Stoppuhr
tic	Startet die Stoppuhr
toc	gibt die Zeit seit dem Aufruf von *tic* auf dem Bildschirm aus.
t=toc	Schreibt die Zeit seit Aufruf von *tic* in die Variable *t*
profile on	Startet den profiler, ein Tool, das die Rechenzeit eines Programms analysiert
profile viewer	Öffnet ein Fenster, in dem die Ergebnisse des profilers dargestellt werden.
str=date	Gibt eine Zeichenkette mit dem aktuellen Datum zurück.
str=clock	Gibt eine Zeichenkette mit dem aktuellen Datum und Uhrzeit zurück.