1. Aufbau und Bedienung von MATLAB

Struktur von MATLAB:
- **Kern von MATLAB** (numerisches Rechnen), wird immer geladen. Es erscheint die
- **Kommandooberfläche** zum Dialog Nutzer – System, Ansicht der Fenster über *Desktop* einstellbar, Schriftgröße über *File/Preferences* veränderbar.
- **Zusatzpakete** (sog. **Toolboxes**), werden bei Bedarf geladen.

MATLAB-Operationen lassen sich einteilen in
arithmetische Operationen, logische Operationen, math. Funktionen, Grafik, Ein-/Ausgabe-Operationen für Daten.

Variable können frei auf der Kommandooberfläche definiert werden.
Variablenname: bis zu 31 Zeichen, Beginn Buchstabe, gefolgt von beliebigen Buchstaben, Ziffern und Unterstrichen, keine Leer- und Interpunktionszeichen, Groß- und Kleinbuchstaben werden unterschieden.

Hilfe s. *help*, Hilfe auf Kommandoebene zu Befehl XX: >> **help** XX
Auflistung der **Hilfen** zu einzelnen **Themenbereichen** >> **helpwin**.
Liste der **allgemeinen Kommandos** >> **help general**.

```
>> x=1.23                    % Mit '=' wird x der Wert 1,23 zugewiesen, Angabe
>>                           % mit Dezimalpunkt, Eingabe über Enter-Taste.
>>                           % Die Variable wird mit 4 Stellen nach dem Dez.punkt
 x =                         % ausgegeben und erscheint im Workspace-Fenster.
    1.2300                   % '%' markiert Kommentar.

>> format compact            % Komprimierte Ausgabe bezüglich Leerzeilen,
>>                           % über File/Preferences als Voreinstellung festlegbar.
>> 3*x                       % Zwischen 2 Faktoren muss ein Malzeichen stehen.
 ans =                       % Ergebnis einer Operation ohne Zuweisung zu einer
    3.6900                   % Variablen wird automatisch ans zugewiesen.
>> format long               % Ausgabeformat ändern: Dezimalzahl mit 15 Stellen
>> x=pi                      % π=pi, es wird der numerische Wert von π zugewiesen
>> format short              % Rückschalten auf Standardausgabeformat (4 Stellen)
>> y=sin(x)                  % Argumente von Funktionen in runden Klammern.
>>                           % Argument trigonometrischer Funktionen im Bogenmaß.
 y =1.2246e-016              % Ergebnis liegt außerhalb d. Bereiches für die
                             % Standardausgabe, Ausgabe als Gleitpunktzahl
                             % mit Faktor 10^-16 (format short e).

>> y=abs(-3),w=3/4,z=sqrt(4); % Mehrere Kommandos in einer Zeile durch Komma
 y =3                        % trennen, ';' nach Eingabe unterdrückt Ausgabe.
 w =0.7500                   % abs Betrag einer Zahl, sqrt Quadratwurzel

>> z='mathe'                 % Zuweisung einer Zeichenkette (character string).
 z =mathe

>> whos                      % Die Variablen im Arbeitsspeicher ausgeben lassen.

>> % Eingabe wiederholen/ändern: Zeile über ↑ -Taste bzw. ↓ -Taste ansteuern,
>> % Zeile erscheint beim aktuellen >>, ev. korrigieren, neu eingeben
>> % Schnelles Ansteuern einer Zeile: Zeilenanfang vor ↑ -Taste angeben.
>> % Speichern von Variablen über File/save workspace as, Import von Daten über load.
>> % Protokoll der MATLAB-Sitzung automatisch im Fenster Command history oder als
>> % ASCII-Datei: diary on  schaltet Prot. an, Datei mit diary filename  zuweisen.

>> clear x                   % Löschen der Variablen x.
>> clear                     % Löschen aller Variablen.

>> % Achtung! Einige Variablennamen werden standardmäßig intern belegt und
>> % sollten nicht für eigene Zwecke benutzt werden :
>> % i, j, ans, pi, eps, flops, inf, NaN, nan, nargin, nargout, realmin, realmax
```

2. Symbolisches Rechnen mit MATLAB

Standard: **Numerisches Rechnen**, **symbolisches Rechnen** ist nur mit **symbolischen Variablen/Konst.** möglich, benötigt die **Symbolic Math Toolbox**, s. a. **help symbolic**

```
>> clear, x=pi, y=sin(x)          % numerische Berechnung
>> x=sym(pi), y=sin(x)            % sym wandelt in symbolische Größe um, dann ist
 x =pi                           % x symbolisch und das Ergebnis der Operation
 y =0                            % mit x ist automatisch wieder symbolisch.
>> x=double(x)                    % Symbolische Größe in numerische verwandeln.
 x =3.1416
>> clear, syms x y                % x und y zu symbolischen Variablen erklären.
>> z= x+x+x+y*y                   % Mit symbolischen Variablen kann formal exakt
 z =3*x+y^2                      % gerechnet werden ohne vorherige Wertzuweisung
>> (x^2-y^2)/(x-y)                % Symbol. Division wird so nicht durchgeführt.
>> simple((x^2-y^2)/(x-y))        % simple umfasst eine Reihe von Routinen, um
 ans =x+y                        % den Ausdruck auf d. kürzeste Form zu bringen.
>> expand((1-5*y)^4)
 ans =1-20*y+150*y^2-500*y^3+625*y^4
>> factor(ans)
 ans = (5*y-1)^4
>> solve('1/R=1/R1+1/R2+1/R3','R') % Symbolische Lös. einer algebraischen Gleich.
 ans =R1*R2*R3/(R2*R3+R1*R3+R1*R2) % Gleichung und Variable, nach der aufzulösen
                                 % ist, als Zeichenkette.
>> syms x                         % oder aber alle Variablen als symbolisch def.,
>> x0= solve(2*x^2+2*x-112)       % Ausdruck in solve wird automatisch gleich 0
                                 % gesetzt, wenn kein Gleichheitszeichen da.
```

3. Erstellung von Grafiken mit dem plot-Befehl, Funktionen zeichnen

plot verbindet Punkte geradlinig miteinander. Die x-Werte der Punkte müssen dazu in einem x-Zeilenvektor und die y-Werte in einem y-Zeilenvektor zusammengefasst werden (**Prinzip Wertetabelle**). **Funktionen** lassen sich auf einen **x-Zeilenvektor** anwenden, die **Auswertung erfolgt elementweise** und das Ergebnis ist ein neuer Zeilenvektor mit den Funktionswerten.

```
>> x=[1 2 3], y=[0 2 0.5]      % Zeilenvektoren eingeben in [], Trennzeichen Blank
>> plot(x,y)                   % plot öffnet grafisches Fenster 'figure' passender
>>                             % Größe und verbindet (1|0),(2|2) und (3|0.5).
>> x1=0:0.1:3;                 % Zeilenvektor Anfang 1, Ende 3, Schrittweite 0.1
>> y1=sin(4*x1);               % Vektor mit zugehörigen Funktionswerten erzeugen
>> plot(x1,y1)                 % (elementweise Auswertung),  y=sin(4x) zeichnen.
                               % Neuer Zeichenbefehl überschreibt aktuelles Bild.
>> clf                         % Für aktuelles Bildfenster Einträge löschen.
>> figure                      % Neues Bildfenster öffnen /(schließen mit close).
>> plot(x,y,'k+--'), hold on   % Farbe, Marker, Linestyle als Zeichenkette
>> plot(x1,y1,'bd-')           % hold on/off aktuelles Bild erhalten/überschreiben
>> plot(x,y, 'r', 'linewidth',3)% Strichstärke auf 3 point setzen.
>> grid on                     % Koordinatennetz einzeichnen / analog grid off.
>> title('Testbild')           % Bildüberschrift einfügen.
>> gtext('sin4x')              % Text einfügen, Positionierung mit der Maus.
>> axis([0 3 -2 4])            % Bildbereich neu festlegen:[xmin xmax ymin ymax]
>> eckp=axis                   % Bildbereich abfragen
 eckp =  0  3  -2  4
>> eckp(2)                     % Zugriff auf das 2. Element des Zeilenvektors eckp
```

Weitere Befehle s. help graph2d (Achsenbeschriftung, Legende,…).
Mit **subplot** lassen sich mehrere Grafiken in einem Bildfenster erzeugen.
Beachten Sie den **grafischen Editor** zur Bildbearbeitung! **Symbolische Funktionen** werden mit **ezplot** gezeichnet, **funtool** bietet einen einfachen Funktionsplotter.

Achtung: Der y-Vektor für y=x^2 entsteht durch elementweises Quadrieren. Diese **elementweisen Operationen** erzeugt man durch einen **Punkt vor dem Operationszeichen.**

```
y2=0.5*x1.^4 +1; plot(x1,y2,'k')
```

4. Symbolisches und numerisches Arbeiten mit Funktionen

Liste der **elementaren mathematischen MATLAB-Funktionen** unter **help elfun**.
Die Definition komplizierter eigener Funktionen ist auch in Form eines Unterprogramms möglich (Function-m-File, s. 5.)

```
>> syms x, y=x^2              % Definition einer symbolischen Funktion,
 y =x^2                       % y ist durch die Zuweisung automatisch symbolisch.
>> syms x, ezplot(x^2,[-1 2]) % ezplot zeichnet eine symbolische Funktion
                              % im Intervall [a,b].
>> y1=diff(y), y2=int(y)      % Diffenzieren u. Integrieren symbol. Funktionen
 y1 =2*x                      % ist problemlos möglich.
 y2 =1/3*x^3
>> int(y,0,1)                 % Bestimmtes Integral(symbol.) im Intervall [0,1].
 ans =1/3
>> subs(y,x,4)                % Werte für eine symbol. Funktion berechnen:
 ans =16                      % In y  x durch 4 ersetzen.
>> syms x y, z=x^2+3*y        % Eine symbol. Funktion von mehreren Variablen
>> y1=diff(z,y)               % lässt sich nach den einzelnen Variablen
 y1 = 3                       % ableiten (partielle Ableitungen).
```

```
>> x=-1:0.1:2, y=exp(x)       % Numerische Beschreibung einer Funktion über den
```

Vektor y der Funktionswerte (Wertetabelle). **MATLAB-Funktionen** dürfen **Vektor/Matrix** als **Argument** haben, die Auswertung erfolgt elementweise. Eigene numerische Funktionen müssen auch so definiert werden, dass sie einen Vektor/eine Matrix elementweise auswerten: y = x.^2, der '.' kennzeichnet die elementweise Operation.

```
>> plot(x,exp(x))             % Numerisch definierte Funktion zeichnen (s. 3)
>> semilogy(x,exp(x)),grid on % Halblog. Darstellung (s. auch semilogx, loglog)
```

Für **Polynome** gibt es in MATLAB eine **spezielle numerische Beschreibung** und **besondere numerische Operationen.** Das Polynom y **= x^3-2*x^2-5*x+6** lässt sich in MATLAB beschreiben durch den **Zeilenvektor seiner Koeffizienten p=[1 -2 -5 6], Potenzen in absteigender Reihenfolge**. Falls eine Potenz in der Reihenfolge fehlt, ist der Koeffizient 0 einzufügen.

```
>> p=[1 -2 -5 6];                    % Koeff.vektor für  x^3-2*x^2-5*x+6
>> x1=2; polyval(p,x1)               % Wert eines Polynoms mit Koeff.vektor p
 ans = -4                            % an der Stelle x1 bestimmen.
>> r=roots(p)                        % Nullstellen (Wurzeln) eines Polynoms
 r = 3.0000                          % numerisch aus dem Koeff.vektor berechnen,
    -2.0000                          % Ausgabe als Spaltenvektor.
     1.0000
>> p1=poly(r)                        % Koeff.vektors eines Polynoms in Normal-
 p1 =1.0000  -2.0000  -5.0000  6.0000 % form aus den Nullstellen bestimmen.
>> poly2sym(p1)                      % Koeff.vektor → symbolisches Polynom
 ans = x^3-2*x^2-5*x+6
>> sym2poly(x^3-2*x^2-5*x+6)         % Symbolisches Polynom → Koeff.vektor
 ans = 1    -2    -5     6

>> p2=[2 1 3];
>> [p3 rest]=deconv(p,p2)            % Division des Polynoms zu p durch das zu
>>                                   % p2 gehörende Polynom.
 P3 =                                % p3 enthält die Koeff. des ganzen Anteils,
    0.5000   -1.2500                 % rest die Zählerkoeff. des Divisionsrestes
 rest =                              % (Potenzen jeweils in absteigender
       0      0   -5.2500    9.7500  %  Reihenfolge).
```

Numerische Integration: trapz(x,y) (für diskrete Werte), **quad** (für Funktionen).
Nullstellenbestimmung mit **solve** (symbolisch, s. 2) oder **fzero** (numerisch).

5. Programmieren mit MATLAB

Script-m-File	**Function-m-File**
Abfolge von Befehlen Eingabe: Menüleiste File → New → Script öffnet MATLAB-Editor-Fenster. Eingabe der Befehle, Syntax gemäß MATLAB. Abspeichern in einer Datei (~.m) (in Ordner, der global verfügbar ist). Aufruf im Command-Window: MATLAB den Pfad mitteilen (Menüleiste Current Folder), dann Datei aufrufen: >> Dateiname Die Befehle werden so abgearbeitet, als ob sie aktuell eingegeben würden. Es wird der Arbeitsspeicher von MATLAB benutzt, d.h. alle Variablen auf Kommandoebene stehen zur Verfügung und können verändert werden.	**Abfolge von Befehlen in einem Unterprogramm mit eigenem Arbeitsspeicher.** Die Kommunikation mit dem Arbeitsspeicher der Kommandoebene erfolgt nur über spezielle Ein- und Ausgabeparameter. Eingabe: Menüleiste File → New → Function **1. Zeile: function** y= *fname*(x) oder **function** y= *fname*(x1,..,xn) x,x1,..,xn Eingabeparameter, y Ausgabepar. Parameter können Skalare, Vektoren oder Matrizen sein. Eingabe der Befehle, Syntax gemäß MATLAB. Speicherung in Datei *fname*. Aufruf im Command-Window: MATLAB den Pfad mitteilen (Menüleiste Current Folder), aufrufen mit v=*fname*(u) oder *fname*(u).

Änderung einer Variablen in einer **Function** verändert die Variable im zentralen MATLAB-Arbeitsspeicher nicht. Variable in einer Function existieren nur temporär. Variable, die in mehreren Arbeitsspeichern verfügbar sein sollen, müssen dort jeweils als **global** deklariert werden.

Beipiel für einen einfachen Function-m-File

```
function y=test(x)
% x Zeilenvektor der Länge>=3, test berechnet die Summe der erste 3 Komponenten.
y=x(1)+x(2)+x(3);
```

Kommandos für die Gestaltung eines m-Files (s. auch help lang):
echo, pause, input, disp, return, error, eval, feval.

Logische Variable (z.B. benötigt für Ablaufsteuerung): **1-wahr, 0-falsch**

```
>> clear, a=3; b=4;
>> v=(a>b)                      % a und b werden bezüglich > verglichen.
 v =0
>> v=(a==b)                     % == (gleich, übereinstimmend)
 v =0                           %    (zu unterscheiden von der Zuweisung '=')
>> v=(a~=b), w=(a>=b)           % ~= (ungleich), >= (größer oder gleich)
```

Spezielle Sprachkonstrukte zur Ablaufsteuerung (Verzweigungen, Schleifen)

if - else - Konstruktionen

```
   if   logischer Ausdruck v    Die Befehle zwischen if und end werden
           bel. Befehle         ausgeführt, wenn v wahr ist.
   end

   if   logischer Ausdruck v
           Befehle              werden ausgeführt, wenn v wahr
   else
           Befehle              werden ausgeführt, wenn v falsch
   end
```

Schleifen (for-Schleifen lassen sich häufig durch Matrixschreibweise vermeiden)

```
   for n=1:10
      bel. Befehle              werden für jedes n einmal ausgeführt
   end                          (als Laufindex niemals i oder j benutzen!)

   while logischer Ausdruck v
           bel. Befehle         werden solange ausgeführt, wie v wahr ist.
   end
```

6. Spalten-/Zeilenvektoren, Matrizen, Lineare Gleichungssysteme

Kommandoliste unter **help elmat** sowie **help matfun**.
MATLAB ist Matrix-orientiert und bietet einfachen Umgang mit Matrizen! Insbesondere sind Vektoren und Matrizen auch als Argumente von Funktionen zulässig. Die Auswertung erfolgt dann elementweise und liefert einen Vektor bzw. eine Matrix mit den Funktionswerten. Erspart for-Schleifen!

Ein-/Ausgabe von ein- und zweidimensionalen Feldern (Vektoren und Matrizen)

```
>> u=[1 2 3]                          % Zeilenvektor eingeben in [], Trennung
 u = 1     2     3                    % der Elemente durch Blank oder Komma.
>> A=[3 4;5 6]                        % Matrix, Eingabe in [] zeilenweise,
 A = 3     4                          % Trennung der Zeilen durch ';'.
     5     6
```

Vektoren und Matrizen erzeugen, darauf zugreifen und verändern

```
>> u=1:7
 u = 1     2     3     4     5     6     7
>> v=0:2:10
 v = 0     2     4     6     8    10
>> v=linspace(0,10,7)                 % 7 äquidist. Zahlen von 0 bis 10.
 v = 0    1.6667    3.3333    5.0000    6.6667    8.3333   10.0000
>> w=logspace(0,1,6)                  % 6 Zahlen von 10^0 bis 10^1,
>>                                    % logarithmische Teilung.
 w = 1.0000    1.5849    2.5119    3.9811    6.3096   10.0000
>> u=ones(2,3)
 u = 1     1     1                    % Analog zeros  für Nullmatrix.
     1     1     1
>> eye(3);                            % Einheitsmatrix E.
>> C=[eye(2) zeros(2,2)]              % Matrix aus Matrizen zusammen-
 C = 1     0     0     0              % setzen.
     0     1     0     0
>> v=[0 1]';                          % ' transponiert konj. komplex, .' transp.
>> w(2),C(2,1), w(3:end)              % Zugriff auf einzelne Elemente.
>> C(:,3)=4;                          % 3. Spalte mit 4en belegen.
>> s=size(C), le=length(C)            %
 s = 2   4                           % Zeilen- und Spaltenanzahl.
 le =4                               % Max(Zeilenzahl, Spaltenzahl)
```

Wird eine Matrix nur mit 1 Index angesprochen (C(5)), erfolgt die Zählung den Spalten folgend. Matrizen lassen sich bequem mit d. **Array-Editor** ändern (öffnen mit Doppelklick auf Variable im Workspace-Browser).

Rechenoperationen für Matrizen und Vektoren: + Matrixaddition, * -multiplikation
Elementweise Operationen für Felder sind mit . (davor) zu kennzeichnen.

```
>> x=[1 3 4]; y=x.^2;  x.*y           % Multiplikation von 2 gleichgroßen
ans = 1    27    64                   % Feldern elementweise.
>> y=3+x                              % Addition einer Zahl zu jedem Feldelement
 y = 4     6     7                    % (ohne Punkt).
>> sum(x)                             % Summe der Vektorelemente; bei Matrizen:
 ans = 8                             % Summmation der einzelnen Spalten.
```

Lösung von linearen mxn-Gleichungssystemen A∘x = b

```
>> A=[1 1 1;4 2 1;1 -1 3]; b=[4;7;10]; % A Koeff.matrix, b rechte Seite, m=n.
>> x=A\b                              % Lösung über LU-Faktorisierung,
 x = 1.0000                          % ähnliches Prinzip wie Gauß,
     0.0000                          % funktioniert auch für symbol. Variable!
     3.0000                          % s. auch help slash.
```

Für **m > n** liefert x=A\b die **Least-Squares-Lösung** (s. help slash), s. auch Kap.7.
Für **m < n** (System unterbestimmt)kann die allg. Lösung mit **solve** ermittelt werden.

det(A) Determinante von A, **inv(A)** Inverse von A, **rank(A)** Rang von A.

Vergleichsoperationen für Felder

```
>> A=[1 -2;0 0];B=[0 4;0 1];
>> V=(A>B)                          % Die bekannten Vergleichsoperationen
 V = 1     0                       % arbeiten für Felder elementweise;
     0     0                       % Ergebnis: logisches Feld.
>> V=(B>1)                          % Die Elemente der Matrix können auch
 V = 0     1                       % mit einer Zahl verglichen werden.
     0     0
>> isequal(A,B)                     % Vergleichsfunktion, prüft A und B
 ans = 0                          % insgesamt auf Gleichheit.
```

7. Messdatenauswertung

Der Mittelwert einer Messreihe lässt sich mit **mean**, die Standardabweichung der einzelnen Messwerte mit **std** berechnen.

Auswertung von Messdatenpaaren (x,y) durch eine Ausgleichskurve

Für lineare Ausgleichsansätze $\mathbf{y = \lambda_1 f_1(x)+\lambda_2 f_2(x)+...}$ mit bel. Funktionen $f_i(x)$ liefert Einsetzen der Messdaten das überbestimmte Gleichungssystem $\mathbf{A \cdot \lambda = y}$. MATLAB berechnet über $\boldsymbol{\lambda}$ **= A\y** die gesuchte **Least-Squares-Lösung**. Z.B. **Ausgleichsgerade**:

```
>> x=1:4; y=[6 6.8 10 10.5];        % Eingabe der Messdaten
>> A=[x' ones(4,1)]; la=A\y'        % A Koeff.matrix des überbest. Gleich.systems
```

Ausgleich speziell durch Polynome mit **polyfit**

```
>> x=-3:3;                          %
>> y=[0.1 0.2 0.5 0.7 0.6 0.3 0.2]; % n=7 Messdatenpaare
>> m=2; la=polyfit(x,y,m)           % Ausgleichspolynom der Ordnung m berechnen
>>                                  % (m=1: Ausgleichsgerade)
>>                                  % x und y Vektoren mit den Messpunktkoord.
 la =  -0.0548  0.0214  0.5905      % Ausgabe: Zeilenvektor la mit d. Koeff. des
                                    % Ausgleichspolynoms (Potenzen absteigend).
>> x1=-4:0.1:4;                     % x-Raster zum Zeichnen des Ausgleichspol.
>> plot(x,y,'ro',x1,polyval(la,x1),'k')  % Messpunkte und Ausgleichspol. zeichnen
```

Beim Aufruf **[la,S]=polyfit(x,y,m)** enthält S in normr die Quadratwurzel der Summe der Abweichungsquadrate.

Interpolation von Messdaten (x,y) (x, y, x1 s.o. wie beim Ausgleich mit polyfit)

```
>> m=6; p=polyfit(x,y,m)            % Es liegen 7 Messdaten vor. Für m=n-1
                                    % liefert polyfit ein durchgehendes Polynom,
                                    % das die Messdaten interpoliert.
                                    % p enthält die Koeffizienten des Polynoms.
>> u=interp1(x,y,x1,'linear');      % Stückweise Interpolation der Messp. (x,y)
>> plot(x,y,'ro',x1,u,'b')          % durch Polynome. Ausgabe u: Werte der
                                    % Interpol.kurve an den Stellen x1.
```

Interpolationsarten: Stückw. linear: **'linear'**, stückw. kubisch: **'spline'**, **'pchip'**

Fehlerfortpflanzung (Vorschlag zur Berechnung mit MATLAB)

```
function [f0,Fehler]= MaxFehler(f,x0,y0,deltax,deltay)
% MaxFehler berechnet den max. Fehler bei linearer Fehlerfortpflanzung
% für eine Funktion von 2 Variablen x und y.
% Eingabe:
% f: Characterstring mit der Funktionsvorschrift, Variablennamen x und y,
% z. B. für die Parallelschaltung von 2 Widerständen: 'x*y/(x+y)'
% x0, y0: Arbeitspunkt
% deltax, deltay: Abweichungen in x und in y
% Ausgabe:
% f0: Wert der Funktion im Arbeitspunkt
% Fehler:  max. Fehler
syms x y
f=eval(f);          % eval führt einen String als MATLAB-Kommando aus
DX=diff(f,x);
DY=diff(f,y);
f0=subs(f,[x y],[x0 y0]);
Fehler=abs(subs(DX,[x y],[x0 y0])*deltax)+abs(subs(DY,[x y],[x0 y0])*deltay);
```