Numerische Methoden mit MATLAB

Hans W Borchers, DHBW Mannheim

April – Mai 2017

(Version 0.7-4)

Tag 1: Einführung in MATLAB

- Was ist MATLAB?
- MATLAB Resourcen
- Benutzeroberfläche (GUI)
- Rechnen mit MATLAB
- Mathematische Funktionen
- Plotten und Funktionsgraphen

MATLAB

"MATLAB ist eine kommerzielle Software des US-amerikanischen Unternehmens **MathWorks** zur Lösung mathematischer Probleme und zur grafischen Darstellung der Ergebnisse. MATLAB ist vor allem für numerische Berechnungen mithilfe von Matrizen ausgelegt, woher sich auch der Name ableitet: MATrix LABoratory.

Die Software wird in der Industrie und an Hochschulen vor allem für numerische Simulation sowie Datenerfassung, Datenanalyse und -auswertung eingesetzt.

Die akademische Bindung ist in der Entwicklung und im Vertrieb von relativ preisgünstigen Studentenversionen bis heute erhalten geblieben und war möglicherweise auch die Grundlage für den Erfolg der Software."

Wikipedia.de

MATLAB Funktionsumfang

MATLAB als Umgebung für techn.-wissenschaftl. Rechnen

- Graphische Bedienoberfläche
- Mathematische Operationen / Funktionen
- Eigene Programmiersprache
- 'Just-In-Time' (JIT) Compiler
- Toolboxen, z.B. die 'Symbolic' Toolbox
- 2- und 3-dimensionale Visualisierungen
- Integration mit MS Office Paketen
- Erzeugung graphischer Benutzerschnittstellen
- Simulink: Simulation dynamischer Systeme
- C/C++ Compiler bzw. .NET Integration

MATLAB an der Dualen Hochschule

- (Studenten-)Account an der Dualen Hochschule muss e-mails empfangen *und* versenden können
- Konto bei `de.mathworks.com` mit diesem Account anmelden; persönliche Daten nicht notwendig
- Herunterladen einer neueren Version von MATLAB, z.B. R2014b, über "Eigener Account" -> "Download Products" *Wichtig*: Alle angebotenen 'Toolboxes' mitnehmen
- Auffinden der Lizenznummer über -> "My Licenses"
- Starten des Installers unter Angabe von Benutzer-ID und Lizenznummer
- Installationstest: Aufruf MATLAB aus dem (Windows-)Menu

Matlab Resourcen

MathWorks Internet Seiten

- **MathWorks** `www.mathworks.com, de.mathworks.com`
- **Documentation Center** `.../help/matlab/`
- **Support** `.../support/` (mit Tutorien und Videos)
- **Matlab Central** `.../matlabcentral/`

Open Source und freie Alternativen

- **Octave** `www.gnu.org/software/octave/`
- **Scilab** `www.scilab.org`
- **Python** `python.org` (mit numpy, scipy, matplotlib, etc.)
- **R** `cran.r-project.org` (mit *pracma* Paket)

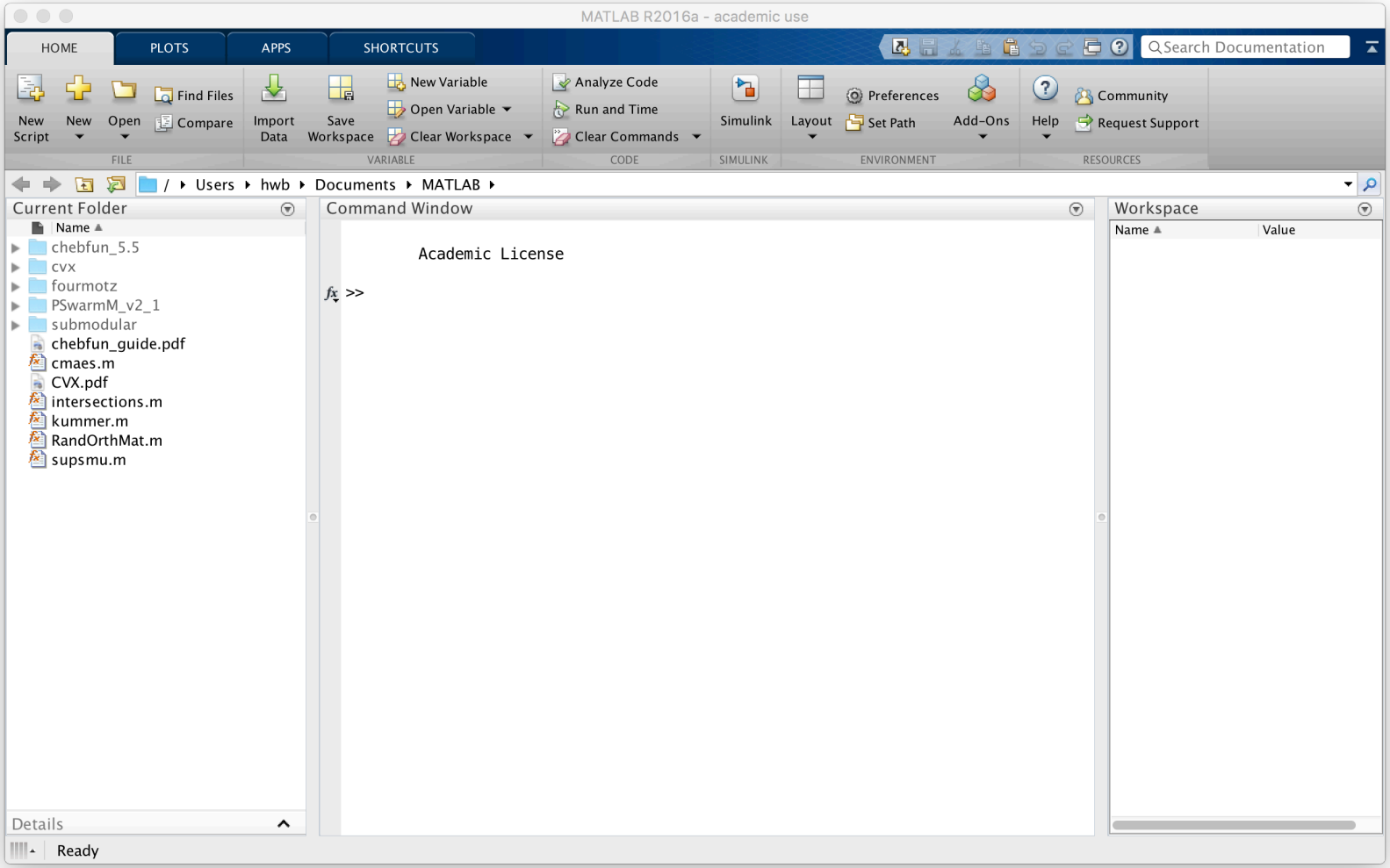
Kursunterlagen

- G. Gramlich, W. Werner. “Numerische Mathematik mit Matlab”. dpunkt.verlag, Heidelberg, 2000. `http://www.hs-ulm.de/users/gramlich/docs/BuchMATLAB.pdf`
- Ulrich Stein. *Einstieg in das Programmieren mit Matlab*. Dritte Auflage, Carl Hanser Verlag, München, 2011.
- Ottmar Beucher. *MATLAB und Simulink: Eine kursorientierte Einführung*. mitp, Hüthig Jehle Rehm, Heidelberg, 2013.

Für diese Kursgruppe:

- Kursfolien, Aufgabenblätter, “Matlab – Quick Reference”
- Prof. Wardenbach, DHBW: “Skript zur Einführung in Matlab (Release R2014b)”

Graphische Benutzeroberfläche



Rechnen mit MATLAB

Zahlen sind in Matlab i.A. doppeltgenaue Gleitkommazahlen (d.h. auf etwa 15 Stellen genau), aber mit engl. *Dezimalpunkt*!

- `0, 1, 2, ...`
- `1.2345, 0.12345*10^1, 12.345*10^-1`
- `1.2345e3, 1.2345e-6` (sog 'engineering' Notation)

Arithmetische Operationen `+ - * / ^`
Spezielle Zahlen `Inf, -Inf, NaN, realmax, realmin, eps`
Zeichenketten `'Abbildung 1'`
Klammern ( ) Gliedern, Prioritäten setzen in math. Ausdrücken
Semikolon ; Trennen von Befehlen, keine Ergebnisausgabe
Kommentar % (bis Zeilenende),
Variablen mit Zuweisung `name =` *wert*

```
pi, e = 2.7181281828 % Ludolphsche, Eulersche Zahl
```

Besonderheiten im “command window”

- **Hilfe** `doc` *`functionname`*, `help` *`functionname`* `lookfor` *`topicname`*
- **Pfeiltasten**: Befehlswiederholung (auch mit ‘Filter’)
- **Kompakte Ausgabe**: `format compact`
- **Zahlenformate**: `format {short|long} {e|g}`
- **Letzter berechneter Wert**: `ans`
- **Zeilenfortsetzung**: ‘`...`’
- **Fensterinhalt löschen**: `clc`
- **Variablen, Funktionen löschen**: `clear`, `clear all`
- **Ausgabe von Variablen**: `a` oder `disp(a)`
- **Anzeige einer Funktion**: `type magic.m`

Mathematische Funktionen

In Matlab vordefinierte Funktionen, zum Beispiel:

- `abs, sign, sqrt, nthroot`
- **Exponentialfunktion** `exp, log, log10, log2`
- **Trigonometrische F.** `sin, cos, tan, cot`
- `asin, acos, atan, acot`
- **Hyperbolische F.** `sinh, cosh, ..., asinh, acosh, ...`
- **Spezielle F.** `beta, expint, gamma, legendre, zeta`

Für natürliche Zahlen definierte Funktionen, z. B.:

- `round, fix, floor, ceil, mod, rem, factorial`

Kurze benutzerdefinierte Funktionen

Kurze, einzeilige Funktionen werden mit der folgenden sehr kompakten Schreibweise definiert (sog. 'anonyme' Funktionen):

- `f1 = @(x) sin(x) .* cos(x);`
- `f2 = @(x, y) sin(x) .* cos(y);`
- `f1(1.5), f2(1.5, 2.5)`

Wenn es sich *nicht* um eine Vektor- oder Matrizen-Operation handelt, sollten immer die Operatoren '.*', './', oder '.^' benutzt werden (als Zeichen für *elementweise* Berechnung)!

'Inline' Funktionen:

- `fun = inline('sin(x) .* cos(x)', 'x');` [deprecated!]

Einfache Funktionsgraphen

Einfaches Plotten von Funktionen mit `ezplot`:

- `ezplot(@sin), ezplot(@sqrt, [0, 2*pi])`
- `ezplot('sin(x)+cos(x)')`
- `ezplot(myfun, 0, 1)`

Parametrische Plots: `x = x(t), y = y(t)`:

- `ezplot(x, y, [0, 1])`
- `ezplot3(x, y, z, [0, 1])`

Zweidimensionale Plots: `f = f(x, y)`:

- `ezcontour(f, [-2*pi, 2*pi])`
- `ezsurf[c](f, [-2*pi, 2*pi])`
- `ezmesh[c](f, [-2*pi, 2*pi])`

Plotten von Funktionen

Aufruf: `plot(x, y, 'farbe linientyp marker')`

- **x** Liste der x-Koordinaten der Kurvenpunkte
- **y** Liste der y-Koordinaten der Kurvenpunkte
- **Farbe**: r, g, b, c, m, y, k, w
- **Linientyp**: '-', ':', '-.', '--'
- **Marker**: '.', o, x, +, *, s, d, v, p, h
- **Text**: `plot(..., 'LineWidth', 2, 'MarkerSize', 3)`

```
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'-rs','LineWidth',2,...
'MarkerEdgeColor','k',...
'MarkerFaceColor','g',...
'MarkerSize',10)
```

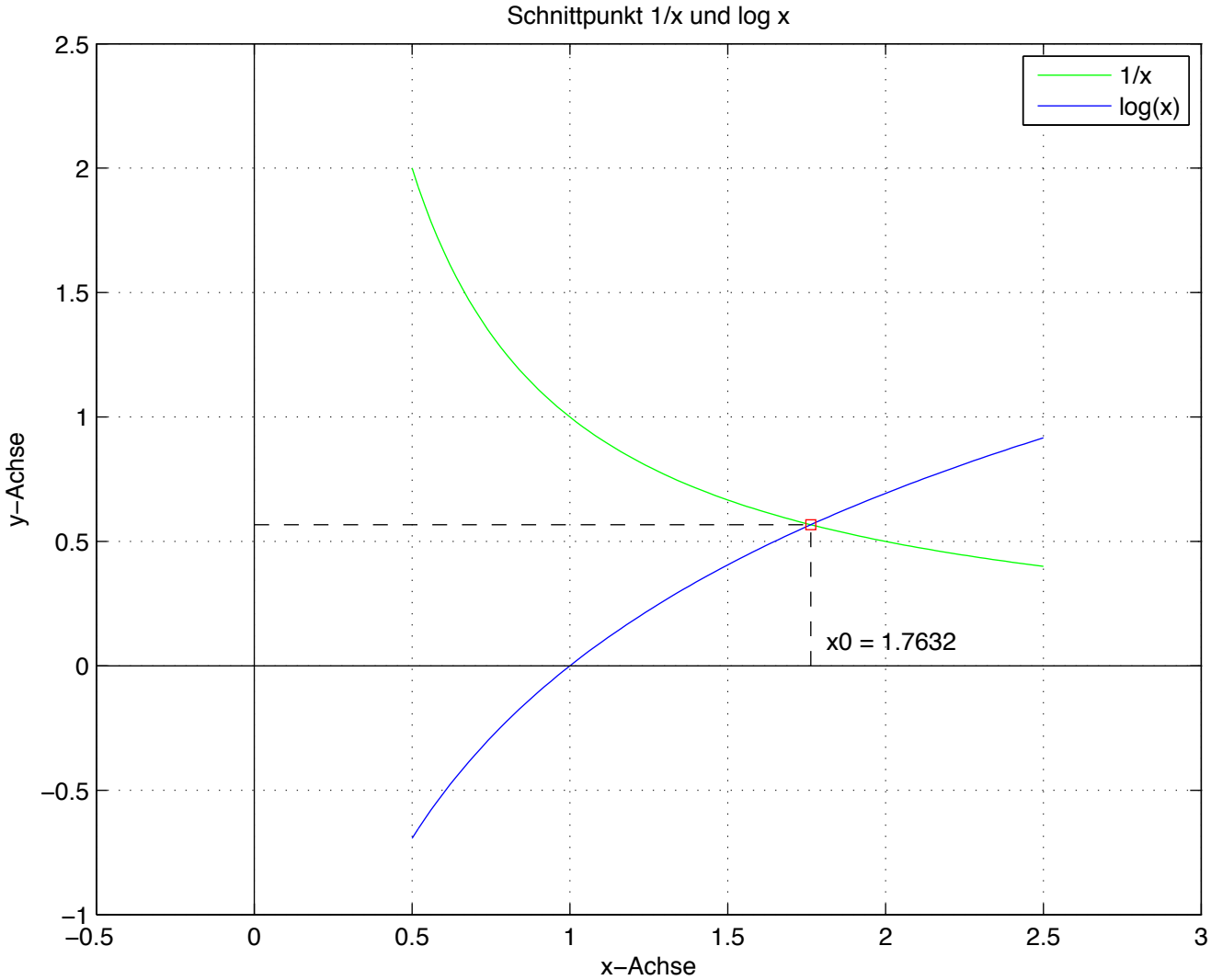
Weitere Plotaufrufe

Mehrere Funktionsgraphen:

```
plot(x, sqrt(x), 'b'); hold on
plot(x, log(x), 'g'); ...
hold off
```

- **Abb.-Titel** `title('sqrt und log Funktion')`
- **Achsenausschnitt** `xlim([-2*pi, 2*pi]); ylim([0, 1])`
- **Achsenbeschriftung** `xlabel('x-Achse'); ylabel('y-Achse')`
- **Legende** `legend('square root', 'logarithm')`
- **Grid** `grid on | off`
- **Text einfügen** `text(xt, yt, 'Nullstelle')`
- **Figur sichern** `print -dtiff filename.tif`
- **Figur löschen** `clf`

Beispielabbildung



Matlab Code für Abbildung 1

```
clf  % oder: fh = figure, ..., figure(fh)
x = linspace(0.5, 2.5, 51);  x0 = 1.7632;
y1 = log(x); y2 = 1 ./ x;
plot(x, y1, 'g', x, y2, 'b', 'LineWidth', 2)
xlim([-0.5, 3]); ylim([-1, 2.5])
xlabel('x-Achse'); ylabel('y-Achse')
plot(x0, 1/x0, 'rs', 'MarkerSize', 5)
text(x0-0.05, -0.1, ['x0 = ', num2str(x0, 4)])
plot([-0.5, 3], [0, 0], 'k')
plot([0 0], [-1.5 2.5], 'k')
plot([0, x0], [1/x0, 1/x0], 'k-')
plot([x0, x0], [0, 1/x0], 'k-')
title('Schnittpunkt der log(x) und 1/x Kurve')
legend('log x', '1/x'); grid('on'); hold('off')
```

Tag 2: Lineare Algebra mit MATLAB

- Vektoren und Matrizen
- Geometrische Anwendungen
- Lineare Gleichungssysteme
- Polynome
- Lineare Regression
- Daten und Zufallszahlen

Vektoren

Ein Vektor ist eine (indizierbare) Liste von Zahlen.

- **Erzeugung** `x = [1 2 3 4 5 6 7 8 9 10]` oder `x = 1:10`
  `x = 0:0.5:2*pi`, `linspace(0, 2*pi, 101)`, `x = 10:-1:1`
- **Indizierung** `x(1)`, `x(2:3)`, `x(4:end)`
  `x(1) = 10`, `x(2:3) = [pi pi]`, `x(4:end) = []`
- **Vektorisierte Funktionen** `x = 0:10; y = sqrt(x)`
- **Funktionen auf Vektoren**
  `sum`, `prod`, `min`, `max`, `mean`, `sort`, `length`
- **Operationen im Vektorraum**
  Skalarmultiplikation: `2 * [1 2 3]`
  Vektoraddition: `[1 2 3]` $\pm$ `[4 5 6]`
- **Elementweise Operationen**
  `[1 2 3].*[4 5 6]`, `[1 2 3]./[4 5 6]`, `[1 2 3].^2`

Geometrische Anwendungen

- **Betrag** eines Vektors: `norm(a)`
- **Skalarprodukt**: `dot(a, b)`
- **Kreuzprodukt** `cross(a, b)`

Beispiele:

- Abstand zwischen zwei Punkten a und b: `norm(a - b)`
- Winkel zwischen zwei Vektoren: $\boxed{\cos\phi = \frac{a \cdot b}{|a||b|}}$

```
phi = acos(dot(a,b) / (norm(a)*norm(b)))
```

- Projektion des Vektors a auf b: $\boxed{a_b = \frac{a \cdot b}{|b|^2}}$

```
a_b = dot(a,b)/norm(b)^2
```

Matrizen I

- **Eingabe**: `A = [1,2,3,4; 0,1,0,0; 0,0,1,0]` ergibt
  $$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$
- Lesen und Schreiben von einzelnen Elementen:
  `a = A(i, j)` bzw. `A(i, j) = 5`
- Indizieren von ganzen Untermatrizen: `B = A(2:3, 2:4)`
- Erweitern von Matrizen um Zeilen oder Spalten:
  `C = [A; 0,0,0,0]` bzw `C = [A, [0;0;0]]`
- Zugriff auf Zeilen/Spalten einer Matrix:
  `Z1 = A(1, :); S3 = A(:, 3);`
- Matrix als Zeilenvektor: `a = A(:)`
- Löschen einer Zeile/Spalte: `A(i, :) = []; A(:, j) = [];`

Matrizen II

- **Operationen** auf Matrizen: `+ - * / ^ .* ./ .^`
  Elementweise Multiplikation: `A .* A; A.^2;`
  Matrizenmultiplikation: `A * B`
  Vorsicht: Matrizendimensionen müssen zusammen passen!
- Inverse Matrix: `B = inv(A)` (A quadr., $\det(A) \neq 0$)
- Transponierte Matrix: `B = A'`
- Diagonale einer Matrix: `diag(A); diag([1,2,3]);`
- Dimensionen einer Matrix: `[n, m] = size(A);`
- **Erzeugung spezieller Matrizen**:
  Einheitsmatrix: `I = eye(n)`, also `B == I * B == B * I`
  Nullmatrix: `zeros(n, m)`
  nxm-Matrix aus lauter Einsen: `ones(n, m)`
- **Zufallszahlen**: `rand(2, 2); randn(1, 1000)`

Matrizen III

- **Funktionen** auf Matrizen:
  Determinante: `det(A)`
  Summe, Min., Max., etc. immer spaltenweise:
  `sum(A), min(A), max(A), mean(A), ...`
  Summe, ... der ganzen Matrix: `S = sum(A(:));`
  Summe, ... zeilenweise: `sum(A');`

- **Eigenwerte und Eigenvektoren**
  Charakteristisches Polynom: `p = poly(A);`
  Eigenwerte: `[V, D] = eig(A);`
  $\Rightarrow$ `A * V(:, 1) == D(1, 1) * V(:, 1)`

- Umformungen: `reshape(A, 2, 6)` — spaltenweise!
- Blockmatrizen: `repmat(A, 2, 1)`

Lineare Gleichungssysteme

- Gegeben ein lineares Gleichungssystem der Form

$$\begin{aligned} x_1 + 1/2x_2 + 1/3x_3 &= 1 \\ 1/2x_1 + 1/3x_2 + 1/4x_3 &= -1 \\ 1/3x_1 + 1/4x_2 + 1/5x_3 &= 1 \\ (1/4x_1 + 1/5x_2 + 1/6x_3 &= 1) \end{aligned}$$

- **Matrizenform**: `A * x = b` mit `b = [1, -1, 1]'` und `A = [1, 1/2, 1/3; 1/2, 1/3, 1/4; 1/3, 1/4, 1/5]`.
- **Lösung**: `x = A \ b;`
  (*Fast* wie die Multiplikation von links mit `A^-1`).
- Der 'Backslash' Operator löst auch überbestimmte Systeme: `AA = [A; [1/4, 1/5, 1/6]]; bb = [b; 1];`
  `xx = AA \ bb; AA * xx`
  nach der Methode der "kleinsten Quadrate"('least-squares').

Polynome

Ein Polynom $a_1x^n + a_2x^{n-1} + ... + a_nx + a_{n+1}$ wird in Matlab dargestellt durch den Vektor `p = [a(1), a(2), ..., a(n), a(n+1)]`.

- **Auswertung** des Polynoms `p` an der Stelle `x`: `y = polyval(p, x)`
- **Nullstellen** des Polynoms `p`: `roots(p)`
- Erzeugung eines Polynom mit Nullstellen in den Punkten `q = [q(1), ..., q(n)]`: `p = poly(q)`
- **Fitten:**Finde ein Polynom vom Grad `n`, dass vorgegebene Datenpunkte `x`, `y` annähert: `p = polyfit(x, y, n)`
- Produkt zweier Polynome: `p = conv(p1, p2)`
- Ableitung des Polynoms `p`: `polyder(p)`
- Integration des Polynoms `p`: `polyint(p)`

Aufgaben zu Polynomen

- Zeichne den Funktionsverlauf von $x^5 - 15x^2 + 9x + 3$.
- Bestimme die (reellen) Nullstellen dieses Polynoms.
- Bestimme die Nullstellen des Polynoms $x^2 - x - 1$ aus der quadratischen Lösungsformel $\frac{p}{2} \pm \frac{1}{2}\sqrt{p^2 - 4q}$ und vergleiche mit dem Ergebnis von `roots()`.
- Bestimme die Nullstellen der folgenden beiden Polynome jeweils mit `roots()` und der Formel. Wie gut stimmen die beiden Ergebnisse überein?
  1. $x^2 - 10^5 x + 1$
  2. $x^2 - 4x + 3.9999999$
- Wie lautet die positive reelle Nullstelle des folgenden Polynoms auf 15 Stellen genau?

$$x^{100} - 2x^{99} + 10x^{50} + 6x - 4000$$

Daten sichern und laden

- `a = input('prompt: '[, 's']);`
  `disp(a), display(a)`
  `s = sprintf('%s = %4.2f', 'a', a);`
- **Variablen speichern**: `save filename x y z`
  Optionen: `-ascii -double -tabs -append`
- **Variablen laden**: `load filename x y`
  `A = load('filename')`
- **Dateien**: `fid = fopen(filename, 'r'); % 'w', 'a'`
  `line = fgetl(fid); % fprintf(fid, '%4.2f', a)`
  `fclose(fid)`
- **Excel Dateien** lesen und schreiben:
  `A = xlsread('filename.xls', 'Sheet1', 'A2:C11')`
  `xlswrite('filename.xls', A, 'Sheet1', 'A2')`

Tag 3: Programmieren mit MATLAB

- Skripte
- Kontrollstrukturen
- Vergleichsoperatoren
- Benutzerdefinierte Funktionen
- Argumente, Fehlermeldungen
- Datentypen

Eigene Funktionen definieren

Jede Funktion in Matlab wird in eine eigene Datei gespeichert.
Dabei stimmen Funktionsname und Dateiname (+ '.m') überein.
Die Datei muss im Pfad oder im aktuellen Arbeitsverzeichnis liegen,
Funktionen können mit STRG-C abgebrochen werden.

myfun.m

```
function y = myfun(x, ...)
% Kommentar für 'help myfun'
...
y = ...
end
```

Der Ergebnisvariablen (hier 'y') muss vor Beendigung der Funktion ein Wert zugewiesen sein. Der Aufruf der Funktion erfolgt mit `x = myfun(5.0, ...)` oder nur mit `myfun(5.0, ...)`.

Schleifen und Verzweigungen

for-Schleifen

```
summe = 0;
for i = 1:100
  summe = summe + i;
end
```

if-Verzweigungen

```
if x > y
  z = 1;
elseif x < y
  z = -1;
else
  z = 0;
end
```

while-Schleifen

```
n = 0;
while c >= a
    c = c/2.0;
    n = n + 1;
end
```

Weitere Anweisungen:

'break' verlässt die innerste for- oder while-Schleife

'continue' fährt mit dem nächsten Schleifendurchgang fort.

'return' beendet die Funktion.

'error' beendet mit Fehlermeldung

'nargin' Anz. Eingabeargumente

Logische Vergleiche

- **Arithmetische Vergleiche**: a kleiner gleich b: `a <= b`
  `==, <, <=, >, >=, ~=`
- **Ergebnis**: `c = (a<=b)`
  c ist 1, wenn `a <= b` wahr ist, sonst 0 (also eine Zahl) !
- **Negation** mit dem ~-Zeichen: `c = ~(a<=b)`
- **Und-Verknüpfung**: `(b < a) && (a < 1.035*b)`
- **Oder-Verknüpfung**: `(b < a-0.6) || (a+0.6 < b)`
- Besonderheiten:
  Alle Zahlen ungleich 0 sind logisch wahr (also 1)
  `A = [0.53 0.67 0.01 0.38 0.07 0.42 0.69]`
  `A <= 0.5` ergibt `[0 0 1 1 1 1 0]`
  `any(A <= 0.5)` ergibt 1
  `all(A <= 0.5)` ergibt 0

Datentypen in Matlab

- **Fließkommazahlen**: *single* (32 bit), *double* (64 bit)
- Ganze Zahlen ('integers'): *int8*, *uint8*, ..., *int64*, *uint64*
- Logisch ('logical', Boolean'): 0, 1
- Zeichenketten ('strings', 'char'): `'Resultat'`
- **Vektoren, Matrizen, Felder** ('arrays')
- Datenfelder ('structures', 'records'):

```
person = struct('name', '', 'tel_nr', 0)
person.name = 'Smith'; person.name
```

- Zellen ('cell', 'cell arrays')

```
my_cell = {'Messung1', [1.1, 1.2, 1.5]}
my_cell{1}, my_cell{2}(1)
```

Tag 4: Numerische Methoden

- Nullstellenbestimmung
- Numerische Differentiation
- Numerische Integration
- Minima und Maxima
- Interpolation
- Funktionsapproximation
- Die ‘symbolische’ Toolbox

Nullstellenbestimmung: Bisektionsmethode

Die Funktion hat in a und b unterschiedliches Vorzeichen. Berechne eine Nullstelle durch fortgesetztes Halbieren des Intervalls $[a, b]$.

```
function x0 = bisect(f, a, b, tol)
x1 = a;
x2 = b;
xm = (x1+x2)/2;
while (abs(f(xm)) > tol)
    if f(x1)* f(xm) < 0
        x2 = xm;
    else
        x1 = xm;
    end
    xm = (x1+x2)/2;
end
x0 = xm;
```

Numerisches Differenzieren

Berechne die Ableitung einer Funktion f an der Stelle x_0 durch

$$f'(x0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}, \quad h << 1$$

deriv.m

```
function xp = deriv(f, x0, h)
xp = (f(x0+h) - f(x0-h)) / (2*h)
end
```

- Berechne die Ableitung von `sin(x)` für `x = 0..pi/2` Wie gross ist die Differenz der numerischen und der 'richtigen' Ableitung? Für welches h wird die Ableitung am genauesten berechnet?

Numerisches Integrieren

Die Simpson Formel berechnet das Integral $\int_a^b f(x)\,dx$, wobei $y_i = f(a + i\,h), h = (b - a)/2n$

$$\int_a^b f(x)\,dx \approx \frac{h}{3}(y_0 + y_{2n} + 4(y_1 + \cdots + y_{2n-1}) + 2(y_2 + \cdots + y_{2n-2}))$$

```
function w = simpson(f, a, b, n)
x = linspace(a, b, 2*n+1);
y = f(x);
h = (b-a)/(2*n);
s1 = y(1) + y(2*n+1);
i = 2:2:2*n; s2 = sum(y(i));
j = 3:2:2*n-1;s3 = sum(y(j));
w = h/3 * (s1 + 4*s2 + 2*s3);
end
```

Funktionsoptimierung

- **Minimierung**: Bestimme das (lokale) Minimum der Funktion f im Intervall von x_1 bis x_2:
  `[x0, fval] = fminbnd(@f, x1, x2)`
- **Maximierung**: Bestimme das Minimum der Funktion $-f$.
- Beispiel: Minimiere $\sin(x) - \cos(x)$ in $[-\pi, \pi]$
- **Funktionen mehrerer Veränderlicher**:
  `[x, fval] = fminsearch(f, x0)`
- Beispiel: Finde ein Minimum der Rosenbrock Funktion mit dem Startwert $(0, 0)$ und visualisiere mit `ezsurf`:
  $rosenbrock(x, y) = (x - 1)^2 + 100(y - x^2)^2$

Regression: Newtons Abkühlungsgesetz

- **Daten**:

```
data = xlsread('name.xls', 'Sheet1', 'A2:B11')
t = data(:, 1)'; T = data(:, 2)', Tu = 20;
```

- **Newton's Abkühlungsgesetz**: $T = (T_0 - T_u)e^{-kt} + T_u$
  $log(T - T_u) = log(T_0 - T_u) - kt$

```
Tlog = log(T - Tu);
p = polyfit(t, Tlog, 1);
k = p(1); T0 = exp(p(2)) + Tu
tp = 0:0.25:20;  % Verlauf für 20 Minuten
Tp = exp(polyval(p, tp) + Tu)
plot(tp, Tp); hold on;
plot(t, T, 'ro')
```

Approximation[1] und Interpolation[2]

- **Funktionsapproximierung** als ein lineares Gleichungssystem

```
xs = [...]'; ys = f(xs);      % Stuetzstellen
A  = [f1(xs), ..., fn(xs)];   % Funktionen
c  = A^-1 * ys                % Gleichungssystem
```

- Beispiel: Bestimme zu den Wertepaaren `x = 1:4; y = [10 5 3 2];` eine Ausgleichsfunktion vom Typ $f(x) = c_0 + \frac{c_1}{x}$.
- 'Splines': `y = spline(xs, ys, x)`
- **Interpolation**: `y = interp1(xs, ys, x, 'linear')`
  Optionen: `'nearest'`, `'linear'`, `'spline'`, `'pchip'`
- Beispiel: Interpoliere $sin(x)$ mit verschiedenen Optionen in den Stellen $x = 0, \frac{\pi}{4}, \frac{3pi}{8}, \frac{3\pi}{4}, \text{und } \pi$.

[1] Approximation: Finde eine Funktion (aus einer Klasse von Funktionen) die eine Funktion oder gegebene diskrete Punkte am besten annähert — im Sinne der kleinsten Quadrate.

[2] Interpolation: Finde eine Näherungsfunktion, die in Stützstellen xs_i mit f übereinstimmt und zwischen den Stützstellen die Funktion f approximiert.

Aufgaben der Numerik

Typische Aufgabenstellungen der Numerik:

- **(Lineare) Regression**: `polyfit(x, y)`
- **Lineare Gleichungssysteme**: `x = A \ b`
- Lineare Algebra: Eigenwerten und Eigenvektoren
- **Nullstellenbestimmung**: `fzero(fun, x1)`, `roots(p)`
  Beispiel: `bisect(fun, x1, x2)`
- **Numerische Differentiation**:
  Beispiel: `deriv(fun, x0)`
- **Numerische Integration**: `quad(f, x1, x2)`
  Beispiel: `simpson(fun, x1, x2)`
- **Funktionsapproximation** und Interpolation
- **Funktionsminimierung**: `fminbnd(fun, x1, x2)`

Symbolic Math Toolbox

- **Symbolische Variablen**: `syms x a b`
- **Funktionen**: `y = x^5 - 15*x^2 + 9*x + 3`
  `f = 1/(5 + 4*cos(x)); ezplot(f, [0, 2*pi])`
- **Nullstellen**: `solve(y, x)`
- **Differenzieren**: `y1 = diff(y, x)`, `y2 = diff(y, x, 2)`
- **Integrieren**: `int(sin(x), x, 0, pi)`
- **Grenzwerte**: `limit(sin(x)/x, x, 0)`
- **Folgen**: `syms n; limit(n^(1/n), n, Inf)`
- **Reihen**: `syms m; a = symsum(1/m^2, m, 1, Inf)`
- **Taylorreihe**: `ft = taylor(sin(x), x, 8)`, `pretty(ft)`
- **Auswerten**: `double(a)`, `subs(ft, x, 0.1)`
- **Vielfache Genauigkeit**: `digits(50); vpa(pi)`