Einführung in MATLAB

MATLAB = Matrizenlaboratorium

MATLAB = Programmiersprache

+ Datenvisualisierung + Computeralgebra

Alle Daten werden in MATLAB als Felder (Vektoren, Matrizen) abgelegt und bearbeitet.

startup.m – File

- **Vor dem ersten Aufruf** von MATLAB wird im Verzeichnis I: ein Ordner mit dem Namen „matlab“ angelegt, der das File Startup.m oder startup17.m enthalten muss.
- In diesem Verzeichnis wird alles abgelegt, was mit MATLAB im Zusammenhang steht.
- Im File startup.m sind alle Pfade anzugeben, die zu persönlichen Verzeichnissen führen und selbst-geschriebene Skript- oder m-Files enthalten. Deshalb sollte das startup.m – File auf keinen Fall gelöscht werden.

Starten von MATLAB

– vor dem Unterricht!

* Rufe Matlab auf – vom Desktop.
* Suche für das aktuelle Arbeitslaufwerk i:\matlab heraus.
* Rufe in Matlab startup.m oder startup17.m
* Richte ein Skriptfile zur Dokumentation des Praktikums ein.
* Speichere dieses vor der Abarbeitung.

Command – Window

* **Einfache Berechnungen** können im Command - Window ausgeführt werden.
* Rechenoperationen

  + ; - ; * ; / ; \ ; ^ **Klammern ( ) und [ ]**
* Berechnung von links nach rechts mit üblichen Prioritäten
* Ergebnisausschrift mit **Variabler ans** (answer)

Beispiel:

```
>> (5 - 3*6)/(7 + 35 - 18*3)
ans =
    1.0833
>>
```

Anlegen eines Protokollfiles

>> diary <filename>

Schreibt alles, was im Command-Window ein- oder ausgegeben wird, in ein File mit dem Namen <filename>, das im Verzeichnis I:\matlab angelegt wird.

>> diary off %unterbrechen,
Voraussetzung für die Betrachtung des Inhaltes des Files

>> diary on %weiterschreiben in das File

Anlegen eines Skriptfiles

* Skriptfile: **m-Datei**, enthält eine Sammlung von Matlabbefehlen, die **unter dem Namen des Skriptfiles gespeichert** und durch Eingabe des Namens im Command-Window abgearbeitet werden können.
* Erstellen eines Formulars für eine Skriptdatei: Anklicken von „New Skript“
* Achtung: vor jeder Abarbeitung speichern!!
* Kommentare beginnen mit „%“

Beispiel / Namenskonvention

- **Beispiel eines Skriptfiles: Name: KU.m**

```
% Skriptfile: Berechnung des Kreisumfangs u
r = input(' Eingabe des Radius r: ')
u=2*pi*r
```

- **Aufruf: >> KU**
  >> Eingabe des Radius r: 4
  >> u=25.1327
- **Namen von selbsterzeugten Skriptfiles für den Beleg** werden nach den Hinweisen zum Beleg vergeben: **Mustermann_SG _A1_B1_4.m**

Nutzung von M-Files

- Alle selbst erzeugten M-Files werden im Verzeichnis I:\matlab\ *.* abgelegt.
- Werden M-Files in einem Unterverzeichnis von I:\matlab\ abgelegt: z.B.: I:\matlab\lingleich\ *.m,

  dann ist ein Eintrag in startup.m nötig:

  path(path ‘I:\matlab\lingleich\‘);
- startup.m nie löschen, aber vervollständigen!
- Tipp: Funktioniert das m-File beim Aufruf durch den Namen nicht, so zieht man es mit der Maus in das Command-Window. Auch wenn das eine Fehlermeldung ergibt, ist das m-File danach in Matlab bekannt und durch den Namen aufgerufen werden.

Variable

- Ausdrücke können mit Variablen bezeichnet werden.

Beispiel:

```
>> a = 3 * 5;
>> b = ((a - 1)*6 -3);
>> c = a - b
c = -66
```

- Display der definierten Variablen mit *who* bzw. *whos*
- **Variablennamen sind „case-sensitiv“**
  (Länge von Variablennamen bis zu 31 characters)
- **Löschen** einer Variablen mit ***clear* <variablenname>**
- Löschen aller Variablen mit *clear*.

Vordefinierte Variable

- pi $= \pi$
- i = j = 0 + i imaginäre Einheit
- eps = 2.2204 e-016 Maschinengenauigkeit
- NAN - not a number (Antwort bei 0/0)
- Matlab-Processing wird unterbrochen mit Ctrl – C
- Semikolon am Ende eines Ausdruckes verhindert das Ausdrucken seines Wertes

Ausgabeformat

- Intern wird mit 16 Mantissenstellen gerechnet
- Ausgabeformat kann festgelegt werden
  z.B.: >> format short (ist Standard)
  >> format long
- Format besser im Menü „Preferences“ im einstellen
- Löschen des Bildschirminhaltes mit „clc“
- Fortsetzen eines Ausdruckes auf der nächsten Zeile durch „ …“

Komplexe Zahlen

c = 1 – 2 i d = 3 + 5 i

kein *- Zeichen

aber e = 1 + cos(pi/4) * i

Spezielle Funktionen für komplexe Zahlen

real(c) =1 imag(c) = -2

angle(c) = -1.1071 % im Bogenmaß

Berechnung im Gradmaß:

angle(c)*180/pi

abs(c) = 2.2361 % Betrag von c

Matlabhilfen

- Help – Button
- f_x - Button
- Demos als Examples im Help-Button
- help name
- helpwin – helpwin name
- doc – doc name — Aufruf Dokumentation
- who – whos — Variablensuche
- what — UP-Suche
- which name … — Suche nach name

Array - Processing

```
>> A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
```

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Ansprechen des Feldelementes
in der 1. Zeile und der 3. Spalte:

```
>> A(1,3)
```

Felderzeugung

```
>> x = 0 : 0.1 : 1        % x = [0, 0.1, 0.2,…,0.9, 1.0]

>> x = pi*(0:0.1:1)       % x = pi*[0, 0.1, 0.2,…,0.9, 1.0]

>> a = 1 : 5              % a = [1, 2, 3, 4, 5]
>> a = 1: 0.5 : 9.7       % a = [1, 1.5, 2.0,…,9.0, 9.5]

>> a = 1 : 5;
>> b = 3 : 0.5 : 7;
>> x = [a, b]             % Zeilenvektor
>> y = x‘                 % Spaltenvektor
```

Teilfelder

$$A([1,2],2) = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$$

$$A(1:2:3,:) = \begin{pmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \end{pmatrix}$$

$$A(1:2:3,1:2:3) = \begin{pmatrix} 1 & 3 \\ 7 & 9 \end{pmatrix}$$

$$A([1,2],[1,3]) = \begin{pmatrix} 1 & 3 \\ 4 & 6 \end{pmatrix}$$

Matrixoperationen

```
>> A = [2, 1, 1; 1, 2, 1; 1, 1, 2];
```

% A =

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

```
>> B = [4, 1, 0; 1, 4, 1; 0, 1, 4];
```

% B =

$$\begin{pmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{pmatrix}$$

```
>> A + B        % Summe A + B
>> A – B        % Differenz A – B
>> A * B        % Produkt A * B
```

Es gelten die Gesetze der Matrizenrechnung!

Matrixfunktionen

```
>> det(A)              % Determinante von A
>> norm(A)             % 2 – Norm von A;  ||A||_2
>> norm(A, inf)        % Zeilenmaximumnorm von A; ||A||_∞
>> size(A)             % Zeilen- u. Spaltenanzahl von A
      >> size(A,1)     % Zeilenanzahl von A
      >> size(A,2)     % Spaltenanzahl von A
>> cond(A)             % Konditionszahl: ||A||_2||A^-1||_2
>> rank(A)             % Rang von A
>> poly(A)             % Koeff. des charkteristischen Polynoms
>> inv(A)              % Inverse Matrix zu A
```

Spezielle Matrizen (1)

Setzen :

```
>> n = 5;
>> m = 3;
>> eye(n)          % Einheitsmatrix
>> zeros(n,m)      % Nullmatrix (speziell: >> zeros(n)  )
>> ones(n,m)       % Matrix ausschließlich mit „Einsen“
                   % besetzt
>> diag(a)         % Matrix mit Diagonalenvektor a
```

Beispiel:

```
>> a = [1, 2, 3];
>> A = diag(a)
```

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Spezielle Matrizen (2)

```
>> triu(A)      % obere (upper) Dreiecksmatrix von A
>> tril(A)      % untere (lower) Dreiecksmatrix von A
>> rand(n)      % Zufallsmatrix (Elemente gleichverteilt)
>> randn(n)     % Zufallsmatrix (Elemente normalverteilt)
>> hilb(n)      % Hilbertmatrix
>> magic(n)     % magisches Quadrat
```

Lineare Gleichungssysteme

```
>> [L, U] = lu(A)      % LU – Zerlegung von A: A = LU
>> L = chol(A)         % Cholesky-Zerlegung von A: A =L^T L
>> [Q, R] = qr(A)      % QR – Zerlegung von A: A = QR
>> [U, D] = eig(A)     % U – Matrix der Eigenvektoren von A
                       % D – Diagonalmatrix mit EW von A
```

Lösen des linearen Gleichungssystems Ax = b:

>> x = A\b

Beispiel:

```
>> A = [2, 1, 1; 1, 2, 1; 1, 1, 2];
>> b = [4, 4, 4]';
>> x = A\b
```

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Matrixinversion

- Die Operationszeichen „/“ *und* „\“ *stehen* beim Rechnen mit Matrizen *stellvertretend für eine Multiplikation mit der inversen Matrix*,
- wobei das obere Ende des Zeichens auf die zu invertierende Matrix zeigt.
- **Es gelten weiterhin die Regeln der Matrizenrechnung!**

- Bsp: size(A)=(n,n), size(b)=(n,1), size(c)=(1,n)
  x = A\b steht für x = (A^{-1})*b, size(x)=(n,1)
  y = c/A steht für y = c* (A^{-1}), size(y)=(1,1)
  A/b und c\A können aufgrund der nicht passenden Typen der Matrizen nicht gebildet werden!

Skalare Funktionen über Arrays

```
>> sin()
>> cos()
>> tan()
>> asin()
>> acos()
>> atan()
>> exp()
>> abs()
>> sqrt()
```

```
>> log()          % natürlicher
                  % Logarithmus
>> rem()          % Divisionsrest
>> sign()         % Vorzeichen
>> round()        % Runden zur
   % nächstliegenden ganzen Zahl
>> floor()        % Runden zur
   % nächstkleineren ganzen Zahl
>> ceil()         % Runde zur
   % nächstgrößeren ganzen Zahl
```

Beispiele

```
>> exp([0:0.5:1])      %  (1.0000, 1.6487, 2.7183)
>> abs([-5,pi,i])      %  (5.0000, 3.1416, 1.0000)
>> asin([-1:0.5:1])    %  (-1.5708, -0.5236, 0, 0.5236, 1.5708)
>> sqrt([0:5:10])      %  (0, 2.2361, 3.1623)
>> sign([-3:3])        %  (-1   -1   -1    0    1    1    1)

>> round([pi,exp(1)])        %  (3,   3)
>> floor([pi,exp(1)])        %  (3,   2)
>> ceil([pi,exp(1)])         %  (4,   3)
>> rem(pi,2)                 %  1.1416
```

Komponentenweise Arbeit mit Arrays

- **Punkte vor den Operationszeichen schalten das Matrizenkalkül aus (bzw. A.‘ entspricht A^T nicht A*)!**
- Die jeweilige Rechenoperation wird **dann komponentenweise** in dem Array ausgeführt.
- **Voraussetzung**: zu verknüpfende Arrays haben die **gleiche Größe**

  Bsp.: >> A = [2, 1, 1; 1, 2, 1; 1, 1, 2];
  >> B = [4, 1, 0; 1, 4, 1; 0, 1, 4];

$$A*B = \begin{pmatrix} 9 & 7 & 5 \\ 6 & 10 & 6 \\ 5 & 7 & 9 \end{pmatrix} \qquad A.*B = \begin{pmatrix} 8 & 1 & 0 \\ 1 & 8 & 1 \\ 0 & 1 & 8 \end{pmatrix}$$

Vektorfunktionen

>> **A** = [a_1,…,a_n]

>> max(**A**)
>> sum(**A**)
>> sort(**A**)

>> min(**A**)
>> median(**A**)

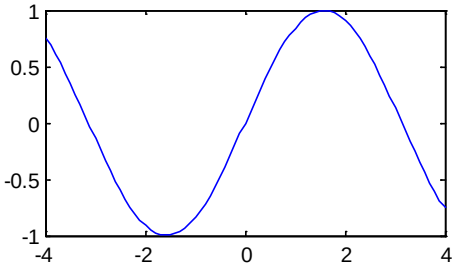
Beispiel:

>> **A** = [3, 5, 1, 9, 7];
>> max(**A**) = 9
>> min(**A**) = 1
>> sum(**A**) = 25
>> median(**A**) = 5
>> sort(**A**) = (1, 3, 5, 7, 9)

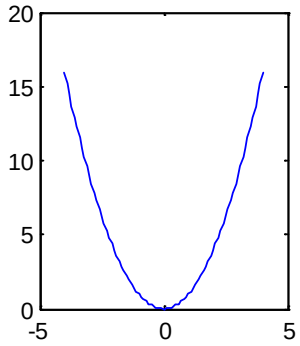
Funktionsplots /1

Funktionsdefinierte Kurven

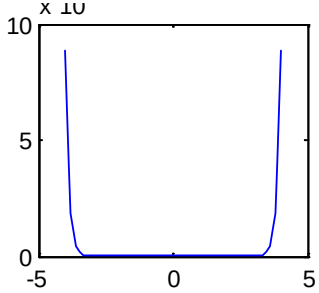
```
>> x = -4 : 0.1 : 4;
>> y = sin(x);

>> plot(x,y)
```




```
>> y = x.^2

>> plot(x,y)
```



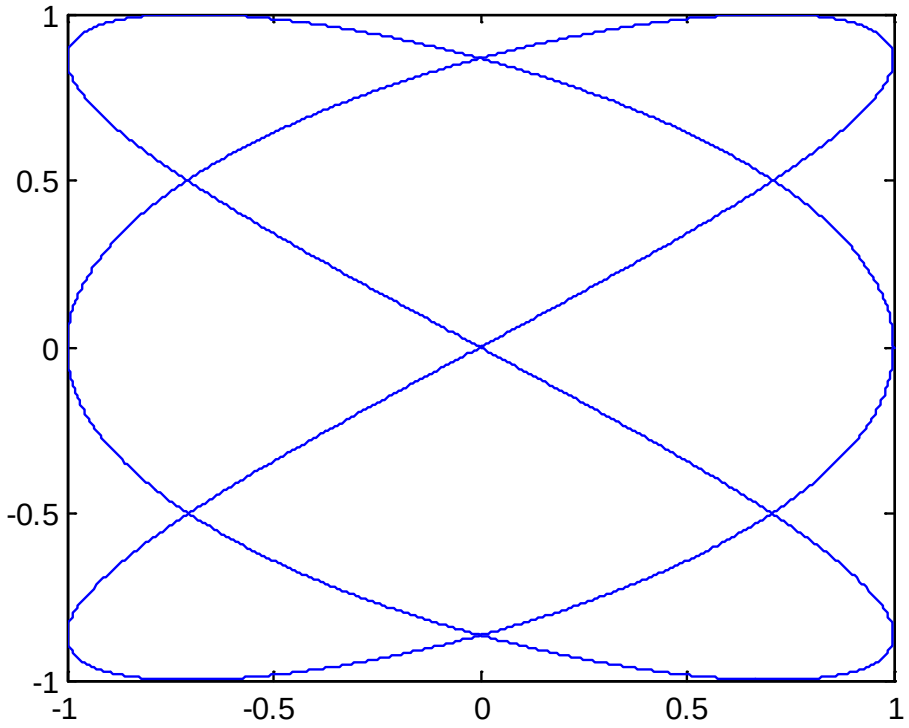

```
>> plot(x,exp(x.^2))
```



Funktionsplots /2

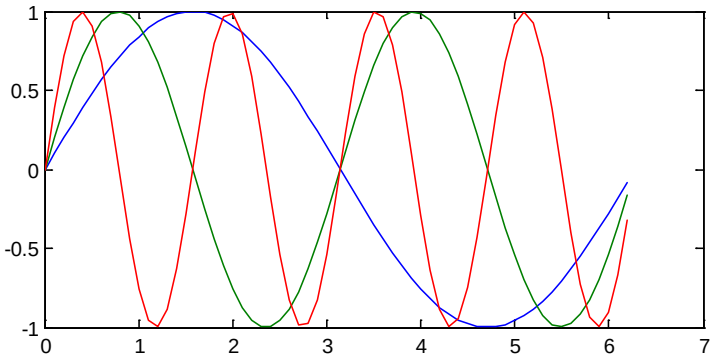
Parametrisch definierte Kurven

```
>> t = 0 : 0.001 : 2*pi;

>> x = cos(3*t);

>> y = sin(2*t);

>> plot(x,y)
```

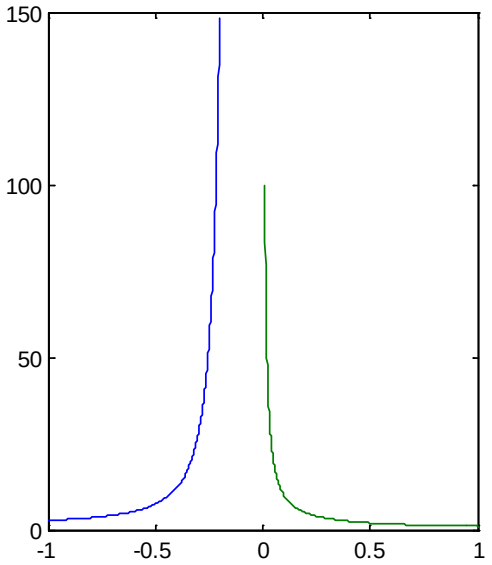


Multiple Plots / 1

```
>> x   = 0 : 0.1 : 2*pi;
>> y1 = sin(x);
>> y2 = sin(2*x);
>> y3 = sin(4*x);
>> plot(x,y1,x,y2,x,y3)
```




```
>> x = -1 : 0.001 : -0.2;
>> y = 0.01 : 0.001 : 1;
>> plot(x,exp(-1./x),y,1./y)
```



Computeralgebra / 1

- Symbolisch definierte Variable: z.B. syms a b c x;
- Umwandlungsroutinen:

  * numerische Variable => symbolische Variable

    z.B. x=sym(5)/sym(2) liefert **x=5/2**

  * symbolische Variable => numerische Variable

    z.B. a=double(x) liefert **a=2.5**

- Vereinfachungsroutinen: z.B.

```
rho=sym(1+sqrt(sym(5)))/sym(2)
f=rho^2 - rho - 1
simplify(f)
```

liefert **f = 0**

Computeralgebra /2

Funktionen auf symbolischen Variablen

- Substitution von Variablen
  f=sym('a*x^2+b*x+c');
  g=subs(f,x,1) => g=a+b+c
- Zeichnen von Funktionen
  z=sin(x), ezplot(z)
- Differenzieren
  diff(sin(x)) => cos(x)
- Integrieren
  int(sin(x)) => -cos(x)
- Gleichungslösen:
  f(x) = 0
  f(x) = g(x)
  solve(f)
  solve(f(x) == g(x))

Unterprogramm-Files / 1

- Sinnvoll zur **Strukturierung** und zum Zugriff aus verschiedenen Hauptprogrammen heraus
- **Skriptfiles** oder selbstdefinierte **Funktionen** (Prozeduren) <filename>.m, die über **filename** bzw. **filename(input_args)** aufgerufen werden
- filename muss bei Prozeduren dem Funktionsnamen entsprechen
- Erzeugung von Funktionen im MATLAB-Editor (ASCII-Format): “New => function“ oder Doppelklick auf den Namen

Unterprogramm-Files / 2

Namenskonvention:

- **Namen von selbsterzeugten Skriptfiles** werden nach den Hinweisen zum Beleg vergeben (s.o):
  Bsp.: **Mustermann_17w1_A1_B1_4.m**
- **Namen von Funktionen** enthalten an Stelle von Aufgaben- und Belegnummer eine frei wählbare Bezeichnung:
  Bsp.: **Mustermann_17w1_cramer.m**

Funktionsparameter/Funktionsaufbau

```
function [ output_args ] = Name( input_args )

    % zu beachten ist:
    % Filename = Name beim Abspeichern
    % [ output_args ] : Feld der Ausgabeparameter
    % Einfachster Fall: Name der Ausgabevariable
    % input_args: Aufzählg. der Eingabeparameter
Anweisungen
end
```

Beispiel einer Funktion

```
function x = hghilbert(n)
% Lösen der Gleichung A*x = b mit der Hilbert-Matrix A
% und der rechten Seite b=A*x0 (x0=(1 .... 1)) nach dem
% Gauss Algorithmus.
if nargin < 1  % number of function input arguments
    n = input(' Eingabe der Gleichungsanzahl n: ');
end;
A = hilb(n);
b = A*ones(n,1);
x = A\b;
```

Aufruf (Beispiel):

```
>> hghilbert(5)
>> n = 5
```

Control Flow (1)

for – Schleife:

```
for x = array
      Anweisungen
end
```

while – Schleife:

```
while Ausdruck
      Anweisungen
end
```

Beispiel 1:

```
for n = 1:2:10
      x(n) = sin(n*pi/10);
end
```

Beispiel 2:

```
for k = 1:5
      kk = k^2;
end
```

Beispiel 3:

```
t = 0;
while t < 100;
      t = t+1;
end
```

A break statement terminates both loops … !

Control Flow (2)

if – else – end - Konstruktionen:

```
if  Ausdruck
   Anweisungen
end
```

```
if  Ausdruck
   Anweisungen
else
   Anweisungen
end
```

```
if  Ausdruck
   Anweisungen
elseif  Ausdruck
   Anweisungen
elseif  Ausdruck
   Anweisungen
end
```

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

Beispiel zur Codierung von y = sign(x)

Control Flow (3)

Vergleichsoperatoren:

< ; > ; <= ; >=

== „gleich“

~= „ungleich“

Logische Operatoren:

& „und“

| „oder“

~ Negation

Beispiel:

```
if  ~a
   x = 0;   % wenn a = 0
else
   x = 1;   % wenn a <> 0
end
```

Control Flow (4)

switch – case - Konstruktionen

```
switch  Ausdruck
    case  Test-Ausdruck 1
        Anweisungen
    case  Test-Ausdruck 2
        Anweisungen
    ……..……
    case  Test-Ausdruck n
        Anweisungen
    otherwise
        Anweisungen
end
```

Beispiel:

```
switch  k
    case  3
        x = 5;
    case  27
        x = 100;
    otherwise
        x = 0;
end
```

Quadratische Gleichung (1)

$$x^2 - 2px + q = 0$$

```
function QUADGLEI
% Berechnung der NST der obigen quadratischen Gleichung
char = 'y';
while char == 'y' | char == 'Y'
    % Eingabe der Parameter
    ……..
    % Berechnung der Nullstellen
    ……..
   disp('  ')
   char = input(' Noch eine quadratische Gleichung? ', 's');
end
```

Quadratische Gleichung (2)

Eingabe der Parameter:

```
disp(' Loesung der quadratischen Gleichung x^2 - 2px + q = 0 ')
  disp('  ')
  p = input(' Eingabe des Parameters p:  ');
  disp('  ')
  q = input(' Eingabe des Parameters q:  ');
  disp('  ')
  disp(' Nullstellen x1 und x2 der quadratischen Gleichung: ')
  disp('  ')
```

Quadratische Gleichung (3)

Berechnung der Nullstellen:

```
r = p*p - q;          % Berechnung der Diskriminante

if r < 0              % Komplexwertige NST
    r = sqrt(-r);
                      %      x1 = p + i*r;
                      %      x2 = p -  i*r;
    str1 = ' Nullstelle x1: %5g + %5g i\n';
    str2 = ' Nullstelle x2: %5g -  %5g i\n';
    fprintf(str1,p,r)
    fprintf(str2,p,r)
```

Quadratische Gleichungen (4)

```
else                    % Reellwertige NST
    r = sqrt(r);
    if p < 0
       r = -r;
    end
    x2 = p + r;
    if x2 == 0
       x1 = 0;
    else
       x1 = q/x2;
    end
    fprintf(' Nullstelle x1:  %5g\n Nullstelle x2:  %5g\n', x1, x2)
end
```

3D – Funktionsplots (1)

```
function testplot
% Verschiedene grafische Darstellungen einer Funktion z = z(x,y)
xbeg = -3.0;  xend =  3.0;
ybeg = -3.0;  yend =  3.0;
nx = 60;  ny = 60;
xpoint = linspace(xbeg,xend,nx);
ypoint = linspace(ybeg,yend,ny);
[x,y] = meshgrid(xpoint,ypoint);
% Funktionswerte z(i,j) der darzustellenden Funktion definieren
z =  3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
   - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
   - 1/3*exp(-(x+1).^2 - y.^2);
```

3D – Funktionsplots (2)

```
% 2D Contour Plot von z(x,y)
contour(x,y,z)
hold on
pause

% Darstellung des Gradienten von z(x,y) durch Pfeile
dx = x(1,2) - x(1,1);     % Spacing in x - direction
dy = y(2,1) - y(1,1);     % Spacing in y - direction
[dzdx,dzdy] = gradient(z,dx,dy);
quiver(x,y,dzdx,dzdy)  % Gradientendarstellung durch Pfeile
hold off
pause
```

3D – Funktionsplots (3)

```
% Farbliche räumliche Darstellung der Oberfläche
% Abstufung gibt die Krümmung der Oberfläche wieder
L = del2(z,dx,dy); % Approximation des LAPLACE-Operators
surf(x,y,z,L)          % 3-D shaded surface plot
shading interp         % Interpolation der Farbtabelle
pause
% 3D Netzoberfläche der Funktion z(x,y)
mesh(x,y,z)
pause
% 3D Netzoberfläche mit hidden off von z(x,y)
mesh(x,y,z)
hidden off
pause
```

3D – Funktionsplots (4)

```
% Mesh plot with underlying contour plot
hidden on
meshc(x,y,z)
pause
% Mesh plot with zero plane
meshz(x,y,z)
pause
% Filled Contourplot with 12 colours
contourf(x,y,z,12)
colorbar
pause
```

3D – Funktionsplots (5)

```
% Generate pseudocolor plot
% 12 contour lines in black
pcolor(x,y,z)
shading interp
hold on
c = contour(x,y,z,12,'k');
pause
clabel(c)
pause
hold off
close all
```