Matlab - eine Einführung

Thomas Schramm

8. Oktober 1999

Inhaltsverzeichnis

Kapitel 1

Einführung in Matlab

1.1 Was ist Matlab?

Matlab ist ein integriertes, interaktives System zur Berechnung, Visualisierung oder der Programmierung mathematischer Ausdrücke. Historisch ist es als Matrix-Laboratorium zur einfachen Benutzung mathematischer Softwarebibliotheken (LINPACK, EISPACK) zur Matrixberechnung entwickelt worden.
Das Programm wird von der Firma **The Mathworks**^{TM}[1] vertrieben und weiterentwickelt. Günstige etwas eingeschränkte Studentenversionen sind im Buchhandel erhältlich.

1.2 Was finden Sie hier?

In dieser Einführung und in den Beispielseiten beziehen wir uns auf Matlab Version 5.3. Die meisten unserer Beispiele sollten jedoch auch noch unter der Version 4.2 funktionieren. Auf die vielen vorhanden Erweiterungen (Toolboxen) wie Simulink, Signal Processing Toolbox, Image Processing Toolbox oder die Symbolic Math Toolbox etc., die z.T. separat erhältlich sind, gehen wir **nicht** ein.

1.3 Auf welchem Computer läuft Matlab?

Matlab läuft unter vielen Betriebssystemen , z.B. UNIX (auch Linux), Win95/98/NT (Matlab 4.2c auch unter Windows 3.1) oder Macintosh. Eine Minimalinstallation benötigt unter Wind95/NT je nach Clustergröße der Festplatte (Matlab enthält sehr viele sehr kleine Dateien) zwischen 25 und 115MB, (mit Dokumentation zwischen 50 und 250MB), die sich aber bei installierten Toolboxen auf mehrere hundert MB ausdehnen kann. Auf einem PC mit 486-Prozessor (besser Pentium) läßt sich Matlab ab 8/12MB (Win95/98/NT) (sinnvoll ab 16MB) Hauptspeicher einsetzen. Eine Netzwerkinstallation ist möglich.
Auf einem Macintosh (Power, 68020/60030 CPU mit 68881/68882 FPU, 68040 CPU, nicht 68LC040 CPU) mit 26MB Festplattenplatz (plus 60 MB für das optionale Hilfesystem) und 16 MB Hauptspeicher arbeitet Matlab mit dem System 7.1 (besser 7.5) und Color Quickdraw.
Auf allen Systemen ist ein CD-ROM Laufwerk erforderlich.

[1] www.mathworks.com

1.4 Wie funktioniert Matlab?

Nach dem Aufruf Matlabs erscheint ein Fenster, in dem an einem Prompt (≫) Matlabkommandos eingegeben werden können. Die Kommandos werden mit der Enter-Taste (Return-Taste) abgeschlossen. Reicht die Zeilenlänge nicht aus, können drei Punkte (...) als Fortsetzungszeichen eingegeben werden. Wir stellen Eingaben grün und Ausgaben blau dar. Auf vergangene Eingaben kann man mit den Pfeiltasten zurückgreifen und diese dann editieren.
Eingegebene Kommandos werden von Matlab ausgewertet. Ist keine Variable zugeordnet, gibt Matlab die Variable `ans` aus. Beendet man eine Eingabe mit einem ein Semikolon (;), wird die Ausgabe unterdrückt. Es werden die normalen arithmetischen Operatoren unterstützt.

```
» (1+2)/(3-4)*5
ans =
   -15
```

Im folgenden geben wir eine kurze Einführung in die wesentlichen Strukturelemente Matlabs, die man kennen sollte, um mathematische Algorithmen in Matlab umzusetzen:

- Kommandos

  Matlabkommandos sind Ausdrücke, die sich aus folgenden Elementen zusammensetzen:

  - Variable
  - Zahlen
  - Operatoren
  - Funktionen

- Datentypen

  Matlabkommandos operieren mit speziellen Datentypen. Matlab kennt sechs verschiedene multidimensionale Felder (Arrays) als Datentypen:

  - numeric: double, sparse, uint8
  - char
  - cell
  - struct

  Der wichtigste Datentyp ist das numerische, zweidimensionale doppeltgenaue Feld: die Matrix.

- Indexierung

  Auf definierte Daten (z.B. Marixelemente) wird mit Hilfe der Indexierung zugegriffen. Dies ist eine der originellsten Eigenschaften Matlabs, da sehr einfach auf ganze Gruppen von Matrixelementen zugegriffen werden kann. Diese kann dazu benutzt werden, komplizierte Matrixoperationen zu vektorisieren oder mit Hilfe logischer Eigenschaften Teile einer Matrix zu selektieren.

- Programmierung: Skripte und Funktionen

  Eine der Gründe für die verbreitete Benutzung Matlabs ist die sehr einfache Erstellung von Matlabprogrammen. Es gibt zwei Arten:

  1. Skripte, die für häufige wiederkehrende Sequenzen von Matlabkommandos gebraucht werden und
  2. Funktionen, die komplexe Programmabläufe mit Parameterübergabe, globalen und lokalen Variablen und Unterprogrammen realisieren lassen.

  Zur Flußkontrolle in Programmen stehen gängige Strukturelemente zur Verfügung sowie Methoden zur objektorientierten Programmierung.

- Grafik

  Matlab enthält sehr viele Funktionen zur Darstellung zweidimensionaler (2D) und dreidimensionaler (3D) Daten. Diese sind in ihrer Basisfunktionalität sehr einfach auf als Vektoren oder Matrizen vorliegende Daten anzuwenden.

- Animation

  Matlab bietet zwei Möglichkeiten Animationen innerhalb Matlabs zu erstellen: Die movie- und die drawnow-Methode.

1.5 Ausdrücke

Gültige Matlabkommandos sind Ausdrücke, die aus folgenden Elementen zusammengesetzt sind:

Variable sind Zeichenketten von denen Matlab maximal 31 Zeichen berücksichtigt. Matlab unterscheidet Groß- und Kleinschreibung! Variablen können mit dem Gleichheitszeichen (=) Werte (meist Matrizen) zugeordnet werden. Um den Wert einer Variablen anzuzeigen, muß nur der Name eingegeben werden. Im weiteren kann mit den Variablen wie mit den zugeordneten Werten gearbeitet werden.

```
» A=25;
» B=3.1
B =
    3.1000
» C=A-B
C =
   21.9000
```

Zahlen werden in dezimalschreibweise mit einem (optionalen) Punkt und führendem Vorzeichen (+/-) angegeben , oder in wissenschaftlicher Darstellung mit einem `e` oder `E` zwischen Mantisse und Exponent. Komplexe Zahlen können mit `i` oder `j` als imaginäre Einheit angegeben werden.

Intern arbeitet Matlab immer mit doppeltgenauen Fließkommazahlen, d.h. mit einer endlichen Genauigkeit von ca. 16 signifikanten Dezimalstellen. Diese Genauigkeit

läßt sich nicht variieren. Der mögliche Bereich von Zahlen erstreckt sich von 1.0E-308 bis 1.0E+308. Auch läßt sich durch Eingabe von ganzen Zahlen (Integers) ohne Dezimalpunkt kein Speicherplatz sparen, da sie intern wieder in Fließkommazahlen verwandelt werden.

Operatoren Matlab kennt die arithmetischen Standardoperatoren und deren Ausführungsregeln, die für Zahlen (Skalare) und Matrizen gelten:

+	Addition und Matrixaddition
-	Subtraktion und Matrixsubtraktion
*	Multiplikation und die nichtkommutative Matrixmultiplikation
/,\	Division und die Matrixrechts- bzw. Linksdivision
^	Exponentation und Matrixexponentation
'	Komplex konjugierte, transponierte Matrix

Sollen die Operationen Elementenweise auf Matrizen angewendet werden, verwendet man die Punkt-Operatoren:

.*	Elementenweise Multiplikation zweier Matrizen
./,.\	Elementenweise Division zweier Matrizen
.^	Elementenweise Exponentation zweier Matrizen
.'	Nicht komplex konjugierte, transponierte Matrix

Vordefinierte Funktionen und Konstanten Matlab kennt viele elementare mathematische Funktionen wie `sqrt` (Wurzel), `exp`, `sin` etc. Eine Liste der elementaren Funktionen erhält man in Matlab mit:

```
» help elfun
```

Eine Liste speziellerer Funktionen (Bessel, Fehlerfunktion etc.) mit:

```
» help specfun
```

und eine Liste besonderer Funktionen für Matrizen mit:

```
» help elmat
```

Die meisten Funktionen nehmen als Argumente auch komplexe Zahlen an und das Ergebnis kann ebenfalls komplex sein.

□
```
» sqrt(-1)
ans =
     0 + 1.0000i
```

Weiterhin sind einige häufig genutzte Konstanten vordefiniert, z.B.:

pi	3.14159265...
j,i	Imaginäre Einheit, $\sqrt{(-1)}$
eps	2^{-52}, Abstand der nächsten Fließkommazahl von 1.0, eps wird von Matlabfunktionen zur Toleranzermittlung benutzt. eps kann überschrieben (z.B. eps=1.e-10)und zurückgesetzt werden (clear eps).
inf	Unendlich, Ergebnis wenn eine von Null verschiedene Zahl durch Null geteilt wird.
nan	Mathematisch nicht definierte Zahl, z.B. 0/0

1.6 Datentypen

Matlab kennt sechs verschiedene multidimensionale Felder (Arrays) als Datentypen:

numeric: double, sparse, uint8

char

cell

struct

Der wichtigste Datentyp ist das numerische, zweidimensionale doppeltgenaue Feld: Die Matrix.

1.6.1 Matrizen und Zahlen

Der am häufigsten verwendete Datentyp Matlabs ist die doppeltgenaue Matrix (Zahlen sind 1x1-Matrizen). Sie kann direkt eingegeben werden.

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

Die Zahlenwerte werden in eckigen Klammern eingegeben. Alle Elemente einer Zeile werden durch die Lehrtaste oder Komma, die Zeilen durch Semikolon getrennt.
Alternativ existieren viele Funktionen, die bestimmte Matrizen generieren: z.B. Für die Einheitsmatrix `eye`, für eine mit Einsen oder Nullen besetzte Matrix ones bzw. `zeros`, oder eine mit Zufallszahlen besetzte Matrix rand.

1.6.2 Mehrdimensionale Felder

Ab der Version 5 beherrscht Matlab auch mehrdimensionale Felder, die z.B aus Matrizen aufgebaut werden können.

☐ Ein 3-dimensionaler Array C, der in der ersten Schicht eine Matrix A und in der zweiten eine Matrix B enthält :

```
» A=[1 2; 3 4];
» B=[5 6; 7 8];
» C(:,:,1)=A;
» C(:,:,2)=B
C(:,:,1) =
     1     2
     3     4
C(:,:,2) =
     5     6
     7     8
```

1.6.3 Dünn besetzte Matrizen: sparse

Matrizen, die sehr viele Nullen enthalten, können sehr effizient mit dem Datentyp `sparse` behandelt werden. Es werden so nur die Indizes und Werte der von Null verschiedenen Elemente gespeichert. Mit dem Befehl `whos` läßt sich der Speicherverbrauch anzeigen:

☐
```
» A=[1 0 0 0 0;0 0 1 0 0;0 0 0 0 0];
» B=sparse(A)
B =
   (1,1)        1
   (2,3)        1
» whos
  Name      Size          Bytes  Class

  A         3x5             120  double array
  B         3x5              48  sparse array
```

1.6.4 Bilddaten: uint8

In der Bildverarbeitung tauchen häufig große Matrizen mit 8-Bit (z.B. ganze Zahlen von 0-255) Daten auf. Diese als doppeltgenaue Matrizen abzuspeichern wäre eine große Speicherverschwendung. Deshalb sieht Matlab den speziellen Typ `uint8` vor, der aber nur von sehr speziellen Funktionen bearbeitet werden kann.

1.6.5 Zeichenketten: Strings

Strings werden in Matlab durch Zeichenketten, die in Apostrophs eingeschlossen werden, dargestellt. Intern sind sie ebenfalls einzeilige Matrizen, die für jedes Zeichen den ASCII-

Wert (Typ: char) enthalten. Die numerischen Werte eines Strings erhalten wir mit dem Kommando `double`; die den Zahlenwerten entsprechenden Zeichen mit `char`.

☐
```
» b='anton'
b =
    anton
» double(b)
ans =
    97   110   116   111   110
```

oder umgekehrt:

```
» char([122 121 120])
ans =
    zyx
```

1.6.6 Weitere Datentypen: cell, struct

Für die fortschgeschrittene Programmierung werden oft weitere Datentypen zur Zusammenfassung verschiedener Datenarten benötigt. `Cells` sind im wesentlichen Felder, deren Elemente wieder Felder sein können. `Cells` werden ähnlich den Matrizen definiert, nur werden geschweifte Klammern {} benutzt.

☐
```
» A=[1 2;3 4];
» B='anton';
» C={A B}
C =
    [2x2 double]    'anton'
```

Auf die Elemente kann durch Angabe des Indexes in geschweiften Klammern zugegriffen werden.

☐
```
» C{1}
ans =
     1     2
     3     4
```

Strukturen können in der Form :

`Variablenname=struct('Feldbeschreiber','Inhalt',...)`

vereinbart werden.

☐ Für eine Adressenkartei:

```
» Adresse=struct('Name','Otto','Ort','Hamburg','Alter',42)
Adresse =
     Name: 'Otto'
      Ort: 'Hamburg'
    Alter: 42
```

Weitere Elemente können per Punkt angefügt werden:

```
» Adresse.Telefon=777777
Adresse =
       Name: 'Otto'
        Ort: 'Hamburg'
      Alter: 42
    Telefon: 777777
```

oder abgefragt werden:

```
» Adresse.Alter
ans =
    42
```

1.7 Indexierung: Zugriff auf Matrix-Elemnte

Eine der originellsten Eigenschaften Matlabs ist die sehr einfache Indexierung von Matrixelementen. Diese kann dazu benutzt werden, komplizierte Matrixoperationen zu vektorisieren.
Auf ein Element einer Matrix wird durch Angabe des Indexes in runden Klammern zugegriffen. Der erste Index steht für die Reihe, der zweite für die Spalte.

☐
```
» A=[1 2 3 4 ; 5 6 7 8 ;  9 10 11 12]
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Das Element aus der zweiten Reihe und der dritten Spalte

```
» A(2,3)
ans =
     7
```

Mit dem Doppelpunkt (`:`) kann auf ganze Bereich von Indizes zugegriffen werden.

☐ Die ersten bis dritten Elemente der zweiten Reihe:

```
» A(2,1:3)
ans =
     5     6     7
```

Soll auf eine ganze Reihe oder Spalte zugegriffen werden, wird nur der Doppelpunkt angegeben.
Die zweite Spalte:

```
» A(:,2)
ans =
     2
     6
    10
```

Sollen z.B. alle Elemente der ersten Spalte addiert werden, könnte man schreiben:

```
» A(1,2)+A(2,2)+A(3,2)
ans =
    18
```

Einfacher ist jedoch die Matlab-Funktion `sum`:

```
» sum(A(:,2))
ans =
    18
```

Weitere Möglichkeiten:
Die Summe aller Spalten:

```
» sum(A)
ans =
    15    18    21    24
```

Die Summe aller Reihen:

```
» sum(A')'
ans =
    10
    26
    42
```

Matrizen können benutzt weden, um komplexere Matrizen aufzubauen. Wenn die Dimensionierung richtig gewählt ist, können diese einfach zusammengefügt (concatenated) werden.

```
» B=[A A+100 ; A-100 A]
B =
     1     2     3     4   101   102   103   104
     5     6     7     8   105   106   107   108
     9    10    11    12   109   110   111   112
   -99   -98   -97   -96     1     2     3     4
   -95   -94   -93   -92     5     6     7     8
   -91   -90   -89   -88     9    10    11    12
```

Die Summe aller Elemente erhält man mit:

☐
```
» sum(sum(B))
ans =
   312
```

Einzelne Reihen oder Spalten können durch Überschreiben mit der leeren Matrix `[]` gelöscht werden.

☐ Löschen der ersten Spalte und letzten Reihe der Matrix B:

```
» B(:,1)=[];
» B(end,:)=[]
B =
     2     3     4   101   102   103   104
     6     7     8   105   106   107   108
    10    11    12   109   110   111   112
   -98   -97   -96     1     2     3     4
   -94   -93   -92     5     6     7     8
```

Der Doppelpunkt kann auch genutzt werden, um Vektoren oder Matrizen zu erzeugen. Syntax:

`[Anfang:Schrittweite:Ende]`

oder bei Schrittweite=1

`[Anfang:Ende]`

☐
```
» [1:.3:3]
ans =
    1.0000    1.3000    1.6000    1.9000    2.2000    2.5000    2.8000
```

1.7.1 Vektorisierung und Schleifen

Um einen Vektor oder eine Matrix zu besetzten, können alle Elemente der Reihe nach in einer Schleife mit dem `for .. end` Konstrukt zugewiesen werden.

☐ Soll z.B. von allen Werten zwischen 0 und 2*pi in Schritten von 0.6 der Sinus gebildet und in einem Vektor S gespeichert werden, kann man folgende Befehle eingeben (jede Zeile ist mit der Entertaste abzuschließen):

```
i=0;
for t=0:.6:2*pi
    i=i+1;
        S(i)=sin(t);
end
```

Dieses Verfahren ist aus zwei Gründen ungünstig:

1. Die Länge des Vektors S ist bei Beginn der Schleife unbestimmt. Matlab muß also bei jedem Durchlauf der Schleife Speicher beschaffen. Besser ist es, wenn die Vektoren oder Matrizen vorbesetzt (präalloziert) werden.

2. Funktionen wie `sin` verarbeiten ganze Vektoren als Argument und geben auch Vektoren aus. Hierbei werden optimierte Verfahren eingesetzt, die sehr viel schneller als Schleifen arbeiten.

Das bessere Verfahren ist:

☐
```
t=[0:.6:2*pi];
S=sin(t)
```

1.7.2 Logische Indexierung

Es bestehen eine Reihe von Möglichkeiten mit logischen Operationen Indizes zu bestimmen und zu benutzten.

☐ Tritt z.B. in einer Matrix eine `NaN` auf:

```
» A=[-3:3]
A =
    -3    -2    -1     0     1     2     3
» B=A./A
Warning: Divide by zero.
B =
     1     1     1   NaN     1     1     1
```

kann diese mit der Funktion `finite` entfernt werden.

```
» B(finite(B))
ans =
     1     1     1     1     1     1
```

Sollen Elemente mit bestimmten Eigenschaften verändert werden, kann dies leicht durchführt werden.

☐ Aus einer zufälligigen `5x5`-Matrix alle Werte auf Null setzten, die größer als 0.3 sind:

```
» A=rand(5,5)
A =
    0.1934    0.6979    0.4966    0.6602    0.7271
    0.6822    0.3784    0.8998    0.3420    0.3093
    0.3028    0.8600    0.8216    0.2897    0.8385
    0.5417    0.8537    0.6449    0.3412    0.5681
    0.1509    0.5936    0.8180    0.5341    0.3704
» A(A>.3)=0
A =
    0.1934         0         0         0         0
         0         0         0         0         0
         0         0         0    0.2897         0
         0         0         0         0         0
    0.1509         0         0         0         0
```

1.8 Programmierung: Skripte und Funktionen

Eine der Gründe für die verbreitete Benutzung Matlabs ist die sehr einfache Erstellung von Matlabprogrammen. Es gibt zwei Arten:

Skripte sind Textdateien, die einfach zeilenweise eine Folge von Matlabbefehlen enthalten. Diese Dateien können einfach mit dem (unter Windows) mitgelieferten (oder einem beliebigen anderen) Editor erstellt und mit einem Namen der Form name.m als m-file abgespeichert werden. (!Achtung: Windows-gestützte Editoren neigen dazu, bestimmte Endungen z.B. .txt an Dateinamen anzuhängen. Manchmal hilft es nur, diese unter DOS umzubenennen.) In einem Matlabfenster wird dann die Folge der Matlabbefehle abgearbeitet, wenn der Dateiname name (ohne .m) eingegeben wird. Hierzu muß die Datei im Suchpfad enthalten sein. Skripten lassen sich keine Parameter übergeben. Definierte Variable werden im globalen Workspace abgelegt, auf welchen vom Skript auch zugegriffen werden kann. Der Inhalt eines m-files kann mit type name angezeigt werden.

Funktionen werden wie Skripte erzeugt, abgespeichert und aufgerufen. Es lassen sich aber Parameter übergeben, lokale und globale Variablen und (ab Version 5) lokale Unterprogramme erzeugen. Das m-file für eine Funktion namens `fname` ist wie folgt aufgebaut:

Funktionsdefinition Hier wird eine Zeile der Form

```
function dummy=fname(p1,p2,...)
```

erwartet. `function` ist ein Schlüsselwort. `dummy` ist eine beliebig benannte Ausgabevariable. Es können hierfür auch Vektoren etc. verwendet werden z.B. `[dummy1,dummy2]`. Im weiteren Programmverlauf wird dieser Ausgabevariablen das Ergebnis zugewiesen. `fname` ist der Funktionsname, mit welchem

die Funktion von Matlab aus aufgerufen wird. `p1,p2,..` sind die (beliebig vielen) Übergabeparameter. Diese werden als *Call by Value* übergeben, d.h. eine Änderung in der Funktion wirkt sich nicht außerhalb der Funktion aus. Soll dies der Fall sein, muß die Variable innerhalb und außerhalb (im Matlabfenster) der Funktion `global` definiert sein.

H1-zeile Diese Kommentarzeile sollte den Namen der Funktion und eine kurze Funktionserläuterung enthalten. Diese und die folgenden Kommentarzeilen werden angezeigt, wenn man in Matlab `help fname` eingibt. Die H1-Zeile wird mit durchsucht, wenn mit `lookfor` in Matlab nach einem Begriff gesucht wird. Z.B.

```
% FNAME erzeugt eine Matrix mit Eigenschaften
```

Hilfstext Hier werden weitere Hilstexte angegeben, die z.B. den Typ der Parameter beschreibt oder auf Fehlermöglichkeiten hinweist. Es kann so eine regelrechte Onlinehilfe erstellt werden. Z.B.

```
% FNAME(p1,p2)
%             p1,p2 müssen Vektoren sein,
%             die die gleiche Anzahl von Elementen
%             haben müssen.
```

Variablendeklaration muß nicht durchgeführt werden. Die verwendeten Variablen werden automatisch als lokal vereinbart. Globale Variable müssen ausserhalb der Funktion (im Matlabfenster oder im aufrufenden Skript) und innerhalb mit dem Schlüsselwort `global` vereinbart werden. Nur dann wirkt sich eine Veränderung dieser Variablen innerhalb und ausserhalb der Funktion aus. Normalerweise sollte man auf den Einsatz globaler Variable lieber verzichten, da man bei größeren Programmprojekten schnell die Übersicht verliert was in welcher Funktion genau definiert ist.

Funktionskörper Im Funktionskörper wird die eigentliche Berechnung vorgenommen. Das Resultat wird der oben angegebenen Variablen `dummy` zugewiesen. Hier kann jedes gültige Matlabstatement stehen. Insbesondere können andere Funktionen aufgerufen werden (s. Programmierung).

Kommentare Hier können noch Kommentare angehängt werden.

Unterfunktionen Weitere Unterfunktionen können angefügt werden, die aber nur für die erste Funktion sichtbar sind.

□ Wir geben als Beispiel eine Funktion zur Berechnung einer speziellen linearen komplexen Abbildung `linear.m` an .

```
function W=linear(Z)
%LINEAR linare Funktion
%       LINEAR(Z) berechnet die lineare Funktion
%       mit den Parametern:
%       a=1/2*exp(i*pi/3)               Drehstreckung
%       b=1+i                           Verschiebung
%       W=a.*Z +b
%       Z reelle oder komplexe Zahl oder Matrix
%       a,b reelle oder komplexe Parameter

a=1/2*exp(i*pi/3);
b=1+i;
W=a.*Z+b;
```

1.8.1 Programmierung : Flußkontrolle

Matlabprogramme enthalten beliebige Matlabstatements. Es gibt aber eine Reihe von Statements, welche meist nur in Programmen angewendet werden. Dies sind speziell Statements, die sich auf den Ablauf des Programms beziehen. Dies sind:

if, else, elseif Die Syntax einer if-Anweisung ist:

```
if logische Bedingung 1
   Matlabbefehle
   ..
end
```

Die Matlabbefehle werden ausgeführt, wenn die logische Bedingung wahr ist, sonst nicht. Die Bedingung `elseif` wird als weitere Verzweigungsmöglichkeit verwendet, wenn die erste Bedingung nicht erfüllt ist. Die Alternative, wenn auch diese Bedingung nicht erfüllt ist, kann mit `else` angegeben werden.

```
if logische Bedingung 1
   Matlabbefehle
   ..
elseif logische Bedingung 2
   Matlabbefehle
   ..
else
   Matlabbefehle
   ..
end
```

☐ Z.B. eine abgewandelte Signum-Funktion `sigpi`, die 1 ausgeben soll, falls eine Zahl größer als π ist, 0, wenn sie gleich π ist und -1, wenn sie kleiner ist:

```
function out=sigpi(x)
     if x>pi
        out=1;
     elseif x==pi
         out=0
     else
         out=-1
     end
```

switch, case Die `switch`-Anweisung hat folgende Syntax:

```
switch Ausdruck (Skalar oder String)
       case  Wert1
             Matlabstatements
       case  Wert2
             Matlabstatements
         .
         .
         .
       otherwise
             Matlabstatements
end
```

Der `Ausdruck` wird ausgewertet. Die Ausführung verzweigt dann zu den Matlabstatements des Falls (case) dessen Wert (Wert1 .. Wertn) mit dem des `Ausdrucks` übereinstimmt. Trifft kein Fall zu wird in den mit `otherwise` gekennzeichneten Weg verzweigt. Anders als in der Programmiersprache **C** wird jeweils nach der Ausführung nicht in den nächsten Fall, sondern auf das Ende der `switch`-Konstruktion verzweigt. Z.B.: eine etwas seltsame Stufenfunktion, die den ganzzahligen Anteil plus 1 im Bereich zwischen -1 und fünf ausgibt. Sonst wird eine `NaN` (not a number) ausgegeben.

```
function out=stufe(x)
   switch fix(x)
     case 0
          out=1;
     case 1
          out=2;
     case 2
          out=3;
     case 3
          out=4;
     case 4
          out=5;
     case 5
          out=6;
```

```
    otherwise
        out=NaN;
end
```

for-Schleifen werden für eine feste Anzahl von Durchläufen von Matlabstatements benutzt. Die Syntax ist:

```
for index=Anfangswert:Schrittweite:Endwert
    Matlabstatements
    ..
end
```

Die Schrittweite muß nicht angegeben werden und ist dann 1. Sie kann positiv oder negativ gewählt werden. Der Index kann in der Schleife verwendet werden, ist beim Start gleich dem Anfangswert und wird nach jedem Schritt um die Schrittweite erhöht (falls positiv). Ist die Schrittweite positiv, wird die Schleife solange durchlaufen bis der Index den Endwert übersteigt (dann aber nicht mehr ausgeführt).

☐
```
x=1;
for i=1:1.1:3
    x=x+i;
end
```

Diese Schleife wird zweimal ausgeführt. x tritt mit dem Wert 1 in die Schleife ein, wird dort im ersten Durchlauf auf drei und nach dem zweiten auf 4.1 erhöht. i hat dann den Wert 2.1.

Ist die Schrittweite negativ, wird die Schleife solange durchlaufen bis der Index kleiner als der Endwert ist.

Beachten Sie, daß vektorisierte Statements oft viel schneller als Schleifen sind.

while-Schleifen werden solange ausgeführt wie eine Bedingung den Wert 1 (TRUE) hat. Die Syntax ist:

```
while Bedingung
      Matlabstatement
      ..
end
```

☐
```
n=1
while n<10
      disp(n)
      n=n+2;
end
```

In dieser Schleife werden nacheinander die Werte 1,3,5,7,9 angezeigt.

break beendet `while`- und `for`-Schleifen.

1.8.2 OOP

Objektorientierte Programmierung ist ab Matlab Version 5 sehr einfach möglich. Konzepte wie:

- Klassen-, Objektdefinition
- Operator- und Funktionsüberladung
- Vererbung

lassen sich einfach realisieren. Eine Einführung in die OOP in Matlab ist für eine spätere Ausgabe vorgesehen.

1.9 Grafik

Matlab enthält sehr viele Funktionen zur Darstellung zweidimensionaler (2D) und dreidimensionaler (3D) Daten. Diese sind in ihrer Basisfunktionalität sehr einfach auf als Vektoren oder Matrizen vorliegende Daten anzuwenden.
Grafiken werden in Matlab in einem Extrafenster dargestellt. Das Fenster hat eine Nummer, die mit dem Kommando `gcf` herausgefunden wird. Diese Nummer nennt man `handle` und wird bei Referenzen auf die Grafik verwendet. Ein neues Fenster läßt sich mit dem Kommando `figure` erzeugen. Hat dieses die Nummer n, kann man es jederzeit mit `figure(n)` zum aktuellen Fenster machen, in welches geplottet wird.

1.9.1 2D

Typen einfacher 2D-Plots:

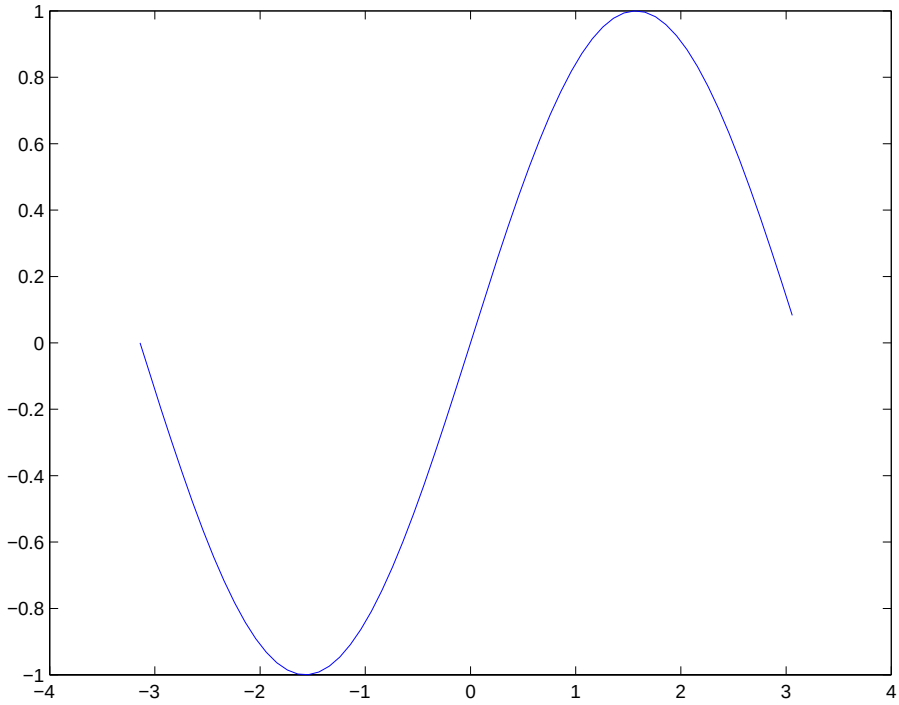
`plot`	Standard 2D-Plot mit linearen Achsen
`fplot`	Funktions-2D-Plot mit linearen Achsen
`loglog`	Doppeltlogarithmischer Plot
`semilogx`	Plot mit logarithmischer X-Achse
`semilogy`	Plot mit logarithmischer Y-Achse
`fill`	Plot von Polygonen

☐ Als Beispiel erzeugen wir uns einen Vektor y, der Sinuswerte eines Vektors x enthält:

```
» x=-pi:.1:pi;
» y=sin(x);
```

Eine einfache Grafik erhalten wir mit dem `plot`-Kommando:

```
» plot(x,y)
```



Alternativ kann dieser Plot auch mit der Funktion `fplot(F,LIMS)` erzeugt werden. `F` ist hierbei entweder der Name einer als m-file abgelegten Funktion oder ein in Apostrophs eingeschlossener String, der die Funktion enthält. Hierbei muß das Argument der Funktion mit einem kleinen "`x`"benannt werden. `LIMS` ist ein Vektor der in der Form `[XMIN XMAX YMIN YMAX]` die Grenzen des Anzeigebereichs enthält.
Für unser Beispiel also:

☐
```
» fplot('sin(x)',[-pi pi])
```

Weitere Graphen lassen sich ohne Probleme in die gleiche plot-Grafik einfügen. Die Syntax ist:

```
plot(x1,y1,x2,y2,x3,y3,...)
```

Als Option läßt sich im `plot`-Kommando die Linienart und -farbe angeben. Erlaubt sind für die Linienart: '-' für durchgehend, '–' für gestrichelt, ':' für gepunktet, '-.' für strichpunkt, 'none' für keine, für die Farbe: 'c', 'm', 'y', 'r', 'g', 'b', 'w' und 'k' für Cyan, Magenta, Gelb, Rot, Grün, Blau, Weiß und Schwarz. Für die verwendeten Symbole läßt sich z.B. '+', 'o', '*', und 'x' angeben.

Beschriftungen sind ebenso einfach anzubringen. Ein Beispiel illustriert dies:

☐ Der Standardplot zweier Funktionsgraphen von Sinus (rot, gepunktet) und Kosinus (blau, Sterne und gestrichelt):

```
» x=[-pi:.1:pi];
» y1=sin(x);
» y2=cos(x);
» plot(x,y1,'r:',x,y2,'b*--')
```

Der dargestellte Bereich von -Pi bis Pi für x und von -1 bis 1 für y:

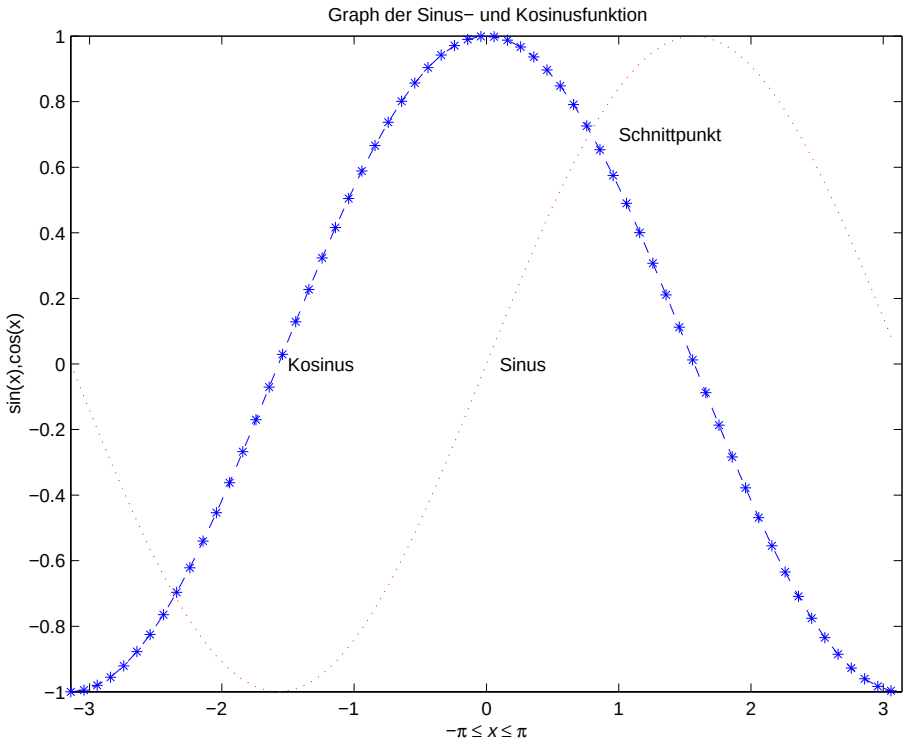
```
» axis([-pi pi -1 1])
```

Beschriftung der X- und Y-Achse. Sonderzeichen im TeX-Stil sind möglich:

```
» xlabel('-\pi \leq \itx \leq \pi')
» ylabel('sin(x),cos(x)')
```

Titel über der Grafik und Text an mit Koordinaten bestimmten Punkten:

```
» title('Graph der Sinus- und Kosinusfunktion')
» text(1, .7, 'Schnittpunkt')
» text(-1.5,0,'Kosinus')
» text(0.1, 0, 'Sinus')
```



1.9.2 3D

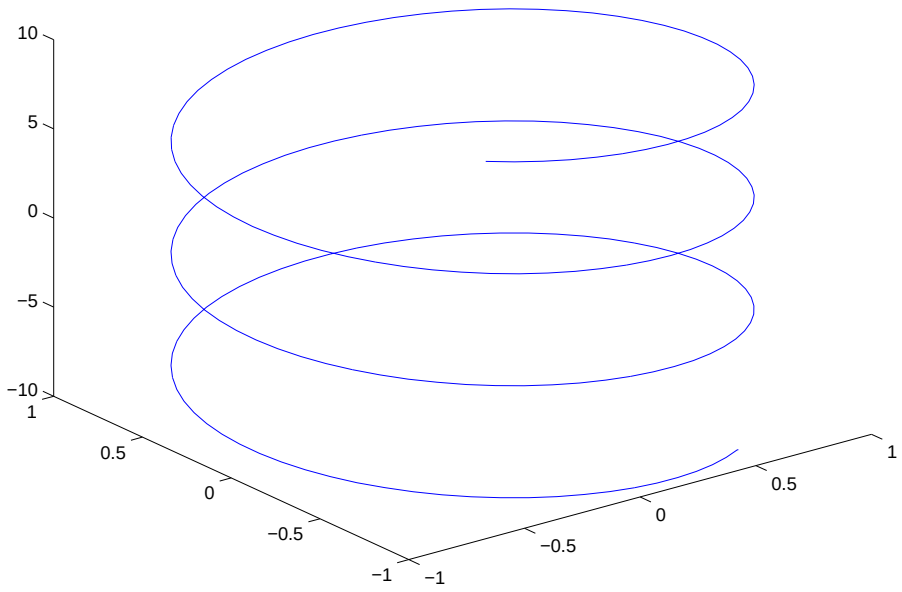
Typen einfacher 3D-Plots

`plot3`	Analogon zum 2D plot-Kommando. Eignet sich zur parametrischen Darstellung von Kurven im Raum.
`mesh`	Gitterlinen- oder Netzplot
`surf`	(Ober-)flächenplot
`contour`	Kontur- oder Höhenlinenplot
`pcolor`	Pixelplot
`surfc`	Flächen- und Konturplot

Kurven im Raum: Ein parametrischer 3D-Plot kann sehr einfach angegeben werden.

☐ Hier eine Spirale:

```
» t=[-10:.1:10];
» x=sin(t);
» y=cos(t);
» plot3(x,y,t)
```



Visualisierung von Funktionen mit zwei Variablen: Flächen im Raum. Hierzu werden in Matlab zwei Matrizen definiert, die jeweil Reihen- bzw. Spaltenweise die jeweiligen x- bzw. y-Koordinaten enthalten. Der Vorteil liegt darin, daß in Formeln die Namen dieser Matrizen wie die Variable benutzt werden. Syntax:

```
x=[Anfangswert:Schrittweite:Endwert];
y=[Anfangswert:Schrittweite:Endwert];
[X,Y]=meshgrid(x,y);
```

Sind die beiden Bereichsvektoren identisch, reicht die Angabe eines der beiden. Der Bereich kann auch direkt in `meshgrid` eingegeben werden.

☐ » `[X,Y]=meshgrid([-2*pi:.2:2*pi]);`

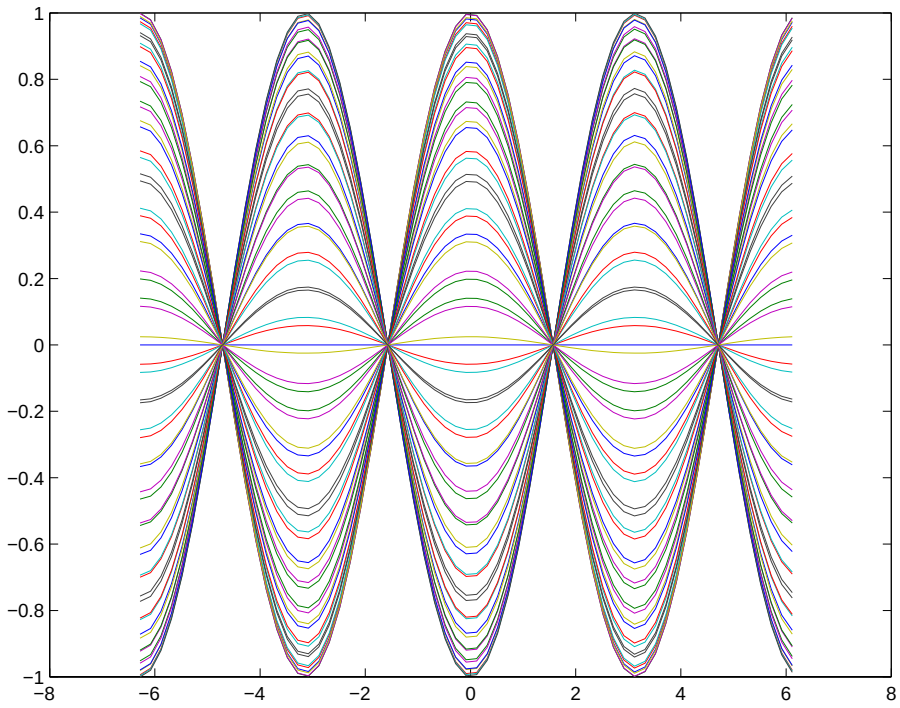
Jetzt kann z.B. eine Matrix Z erzeugt werden, die dann jeweils die Funktionswerte der Koordinaten (X,Y) enthält.

☐ Z.B. der „Eierkarton“:

» `Z=sin(X).*cos(Y);`

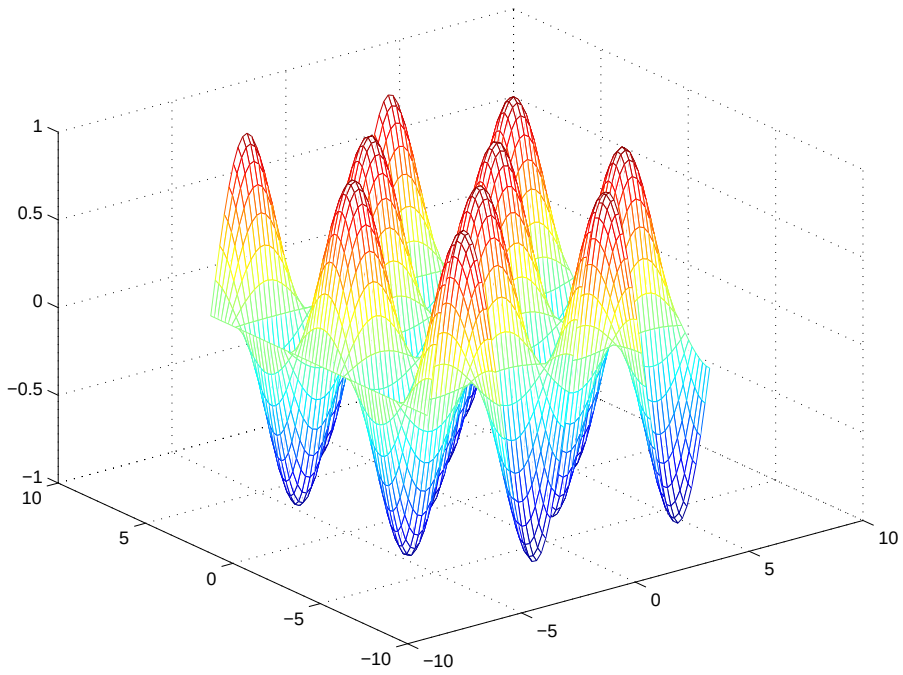
Beachten Sie die **Punkt-Multiplikation**. Die beiden Matrizen `sin(X)` und `cos(Y)` sollen ja elementweise miteinander multipliziert werden.
Die Darstellung des „Eierkartons“kann jetzt auf viele Arten geschehen. Im einfachen 2D-Plot werden alle Z(X)-Plots für jedes Y hintereinander ausgegeben:

☐ » `plot([-2*pi:.2:2*pi],Z)`



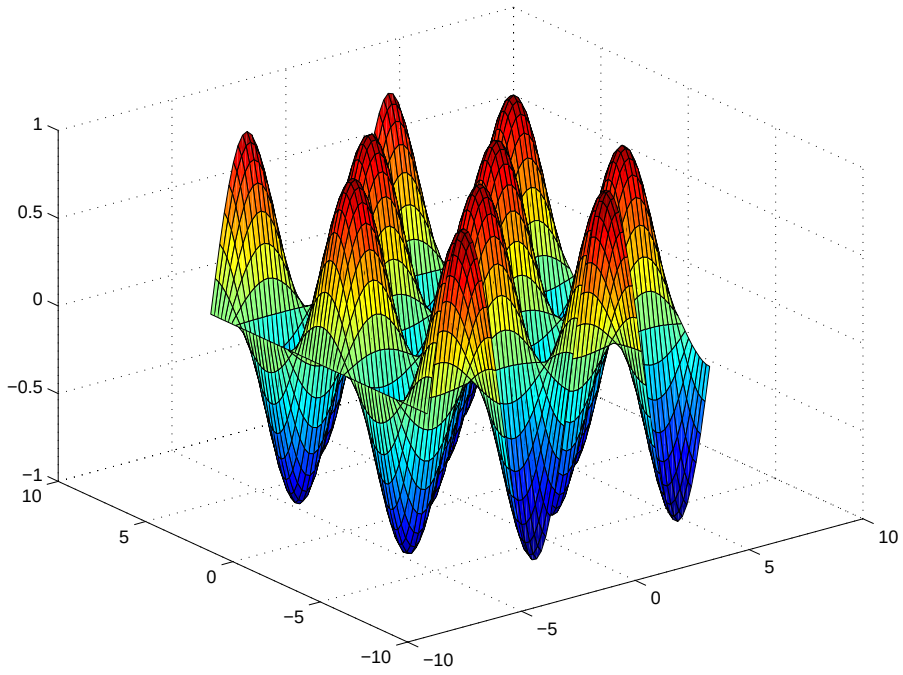
☐ Ein einfaches „Funktionsgebirge“ (Netz) erhält man mit dem `mesh`-Kommando:

```
» mesh(X,Y,Z)
```



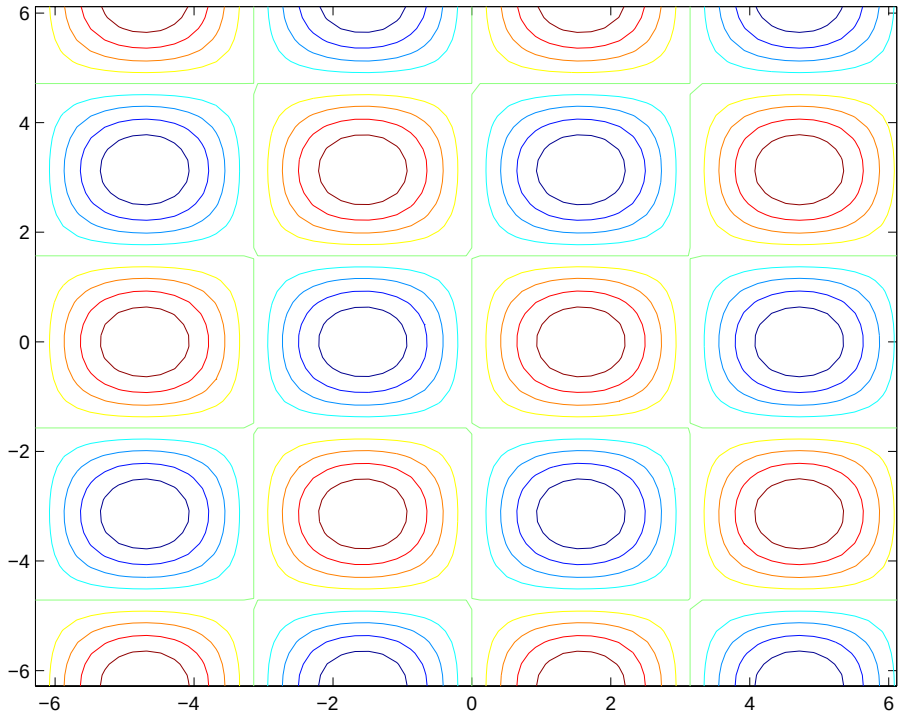
☐ Ein Flächenplot (surface) mit eingefärbter Oberfläche:

```
» surf(X,Y,Z)
```



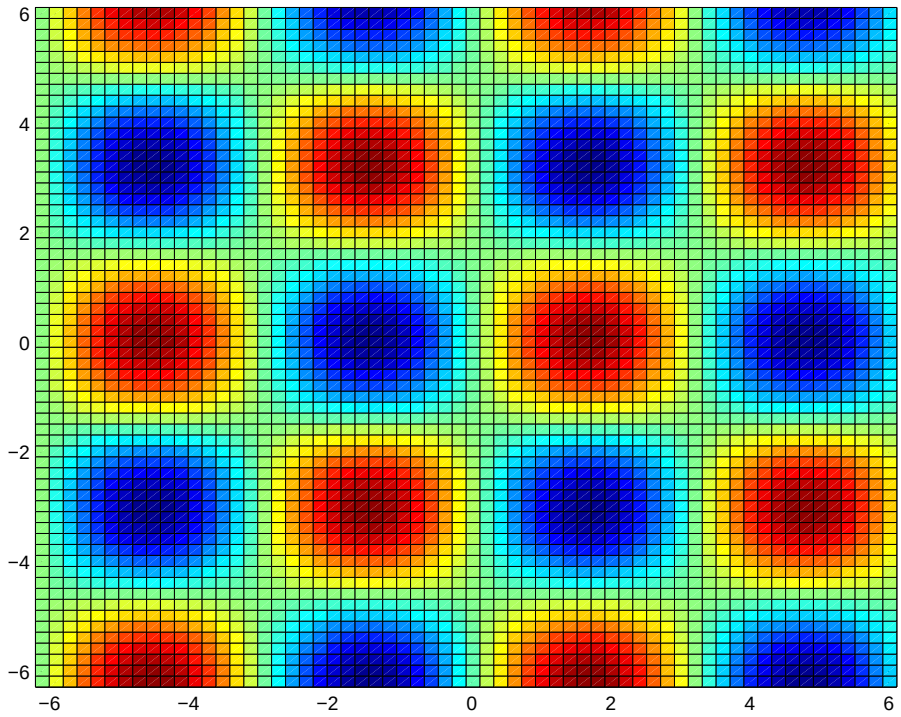
☐ Der klassische Konturplot:

```
» contour(X,Y,Z)
```



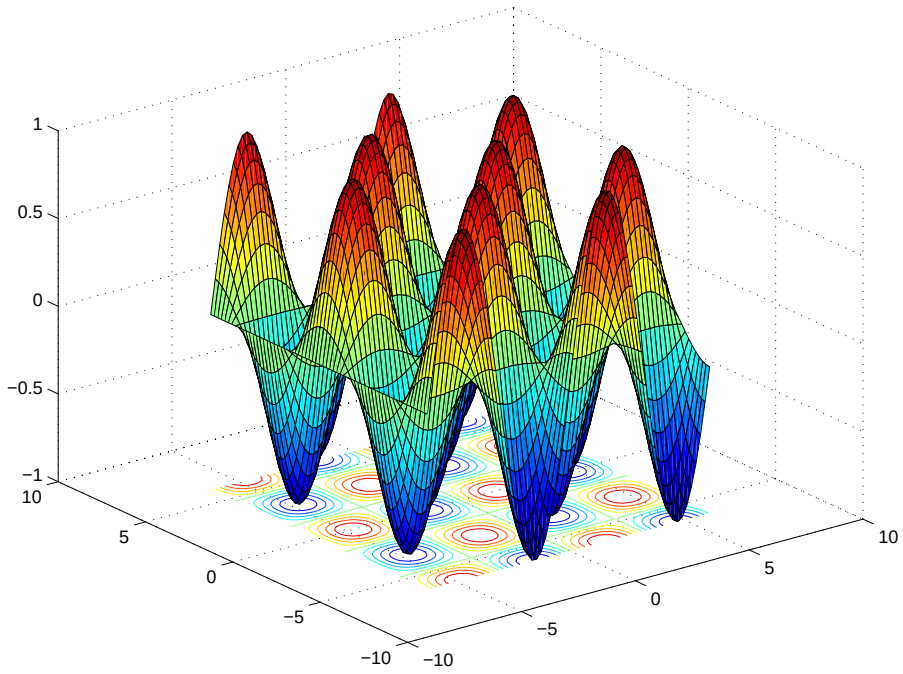
☐ Ein Pixel-Plot. Dieser entspricht einem von oben gesehenen surf-Plot:

```
» pcolor(X,Y,Z)
```



☐ Ein kombinierter Flächen-Konturplot:

```
» surfc(X,Y,Z)
```



Viele weitere spezielle Darstellungen, Plots und Variationsmöglichkeiten sind in Matlab enthalten, deren Besprechung aber unseren Rahmen sprengen würden.

1.10 Animation

In Matlab gibt es zwei Möglichkeiten animierte Grafiken zu erstellen. Die **Animation mit MOVIE** und die **Animation mit DRAWNOW**

1.10.1 Animation mit MOVIE

mit Hilfe des `movie`-, `moviein`-Befehles. Hierzu wird mit einem `axis`-Befehl zuerst die Dimensionierung des Ausgabefenster definiert. Will man N Einzelbilder in einer Animation zusammenfassen, wird hierzu der Speicherplatz in einer Matrix M mit dem Befehl `M=moviein(N)` reserviert. Mit dem Befehl `set(gca,'NextPlot','replacechildren')` werden dann die Achseneinstellungen beibehalten. Jetzt kann nach jedem Plot der Befehl `M(:,j)=getframe;` (Plotnummer j=1 bis N) abgesetzt werden. Sind alle Plots abgespeichert, kann die Animation mit `movie(Anzahl)` abgespielt werden. `Anzahl` gibt die Anzahl der Wiederholungen an. Ist sie negativ, wird jeweils vor- und zurückgespielt. Optional kann die Anzahl der Frames/Sekunde noch mit angegeben werden.
Diese Möglichkeit der Animation empfiehlt sich für aufwendige Rechnungen.
Beispiele zur Animation mit MOVIE:

Ein animierter Sinus. Das folgende Beispiel liegt als Skript ANISIN.M im Beispielverzeichnis. Ist dieses im Suchpfad eingetragen kann es von Matlab direkt ausgeführt werden.

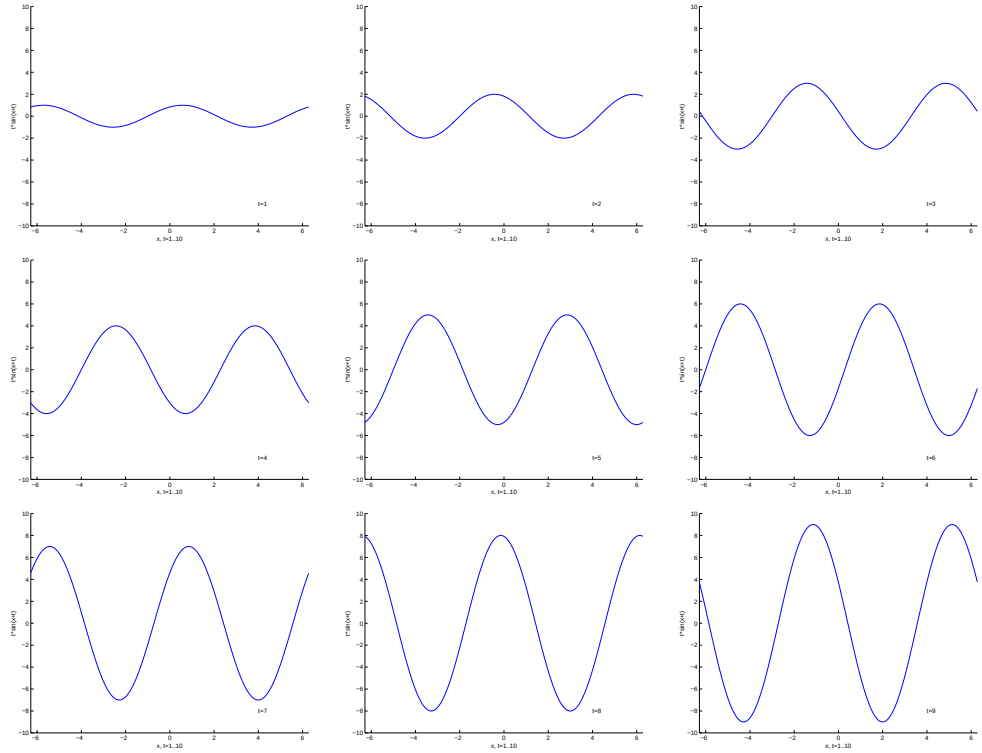
```
% anisin
% Animation einer 2D-Funktion
clear all

axis([-2*pi 2*pi -10  10])
N=10;
M=moviein(N);
set(gca,'NextPlot','replacechildren');
xlabel('x, t=1..10')
ylabel('t*sin(x+t)')
x=[-2*pi:.01:2*pi];
for t=1:N
   plot(x,t*sin(x+t))
   text(4,-8,['t=' int2str(t)] )
   %Optionaler Ausdruck als TIFF-Files
   %print('-dtiff','-r72',['anisin' int2str(t)])
   %Optionaler Ausdruck als Postscript-Files
   %print('-depsc2',['anisin' int2str(t)])
   M(:,t)=getframe;
end

%Abspielen mit:

movie(M,-5)
```

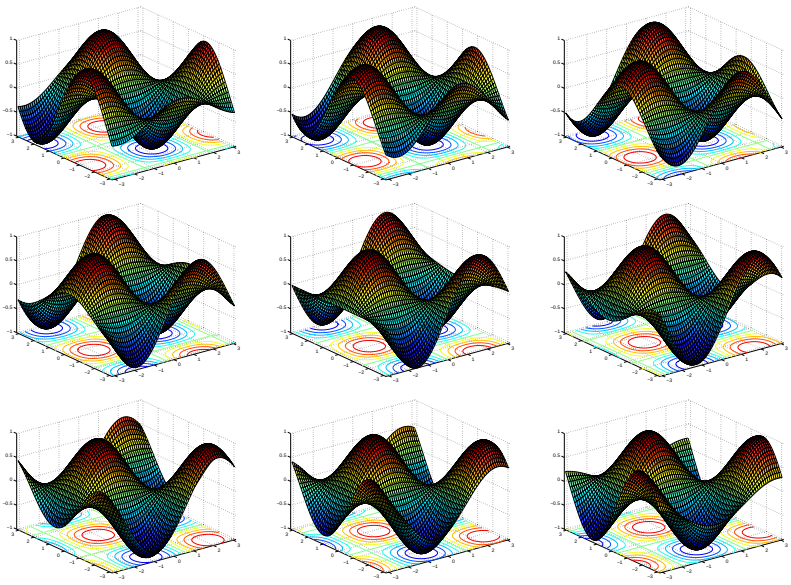
Für die Druckversion geben wir eine Tabelle mit 3×3 Plots aus, die im Movie nacheinander abgespielt werden.



☐ **Der animierter „Eierkarton“** Wir geben ein weiteres kommentiertes Beispiel an, das als File MYMOVIE.M im Beispielverzeichnis vorliegt. Ist dieses im Suchpfad eingetragen kann es von Matlab direkt ausgeführt werden:

```
%MYMOVIE
% Animation des "Eierkartons"
% cos(x-t)*sin(y-t)
clear all
axis([-pi pi -pi pi -1 1])
nframe=10;
M=moviein(nframe);
set(gca,'NextPlot', 'replacechildren')
%Wertebereich
x=[-pi:.1:pi];
y=x;
[X,Y]=meshgrid(x,y);
for j=1:nframe
    t=j/nframe*pi;
    surfc(x,y,cos(X-t).*sin(Y-t))
    %[Optionaler] TIFF-Ausdruck der Einzelbilder
    %print('-dtiff','-r72',['aniei',int2str(j)])
    %[Optionaler] Postscript-Ausdruck der Einzelbilder
    %print('-depsc2',['aniei',int2str(j)])
    M(:,j)=getframe;
end
%Das erzeugte Movie 5-mal hin- und zurückspielen
movie(M,-5)
```

Für die Druckversion geben wir eine Tabelle mit 3×3 Plots aus, die im Movie nacheinander abgespielt werden.



1.10.2 Animation mit DRAWNOW

Die zweite Methode benutzt den `redraw`-Mechanismus Matlabs, der bei weniger rechenintensiven Pozessen benutzt werden kann, um die Ergebnisse önlinemitzuplotten. Sollen die Einzelbilder oder das Endergebnis ausgedruckt werden, kann mit dem capture-Befehl die aktuelle Grafik als Pixelbild gesichert und mit dem image-Befehl in einer neuen Figur dargestellt werden. Diese neue Figur kann dann mit dem print-Befehl z.B. als TIFF-File ausgegeben werden. Das im Handbuch angegebene Verfahren (-zbuffer) funktioniert nicht!
Beispiel zur Animation mit DRAWNOW:

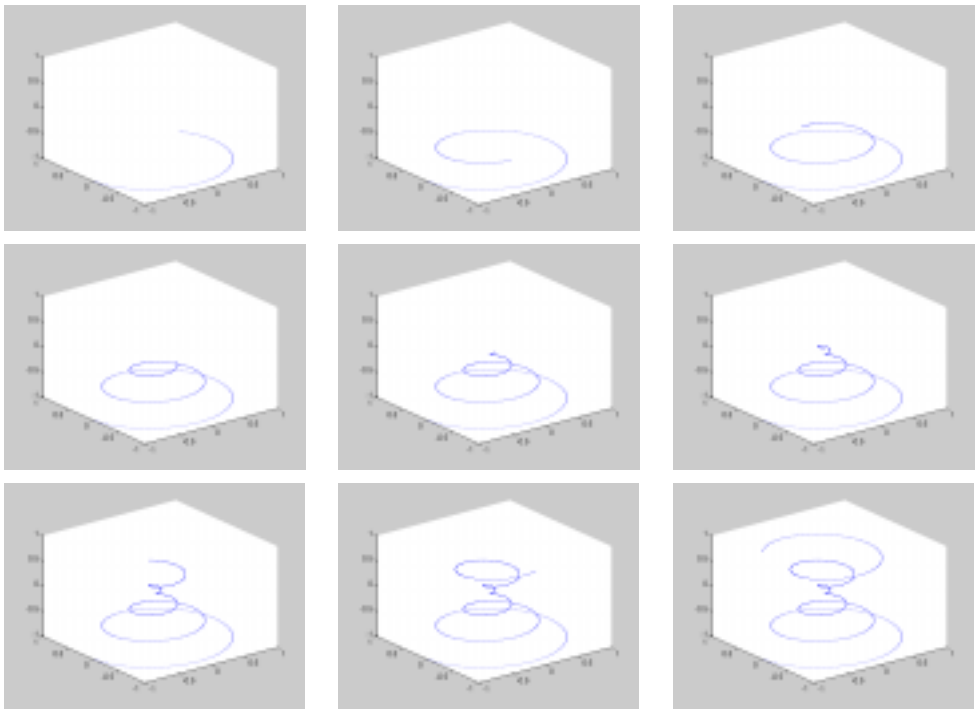
☐ **Eine animierte Spirale:** Wir geben eine kommentiertes Beispiel an, das als File ANISPI.M im Beispielverzeichnis vorliegt. Ist dieses im Suchpfad eingetragen kann es von Matlab direkt ausgeführt werden:

```
%anispi
% Animation mit dem drawnow-Verfahren
% einer Spirale auf einem Doppelkegel.
clear all
%Anzahl der gesetzten Bildpunkte 2*N+1
N=1000;
%Ausdruck alle M Schritte, wenn drucken=1
drucken=0
%drucken=1
M=200;
%EraseMode none verhindert das Löschen des Bildinhaltes
%Plot eines Punktes zum Setup der Grafik
p=plot3(0,0,0,'EraseMode' ,'none')
%Handle auf die Plotfigur bereitstellen
cp=gcf;
%Plotbereich festlegen
axis([-1 1 -1 1 -1 1])
hold on

for i=-N:N
   phi=6*pi*i/N;
   r=i/N;
   %Die Spirale
   x=r*cos(phi);
   y=r*sin(phi);
   z=r;
   %An den Koordinaten x,y,z wird ein Punkt gesetzt
   set(p,'XData',x,'YData',y,'ZData',z);
   %und gezeichnet
   drawnow
```

```
 %[Optionaler] Ausdruck
   if drucken
    if mod(i,M)==0
      k=i/M+N/M
      [X,map] = capture(cp);
      figure('Name','BITMAP' )
      image(X)
      colormap(map)
      axis off
      set(gca,'Position',[0 0 1 1])  % Fill entire figure
      %Ausdruck als TIFF- oder Postscriptdatei
      print( '-dtiff','-r72', ['anispi' int2str(k)])
      %print( '-depsc2', ['anispi' int2str(k)])
    end
   end
end
```

Für die Druckversion geben wir eine Tabelle mit 3×3 Plots aus, die im Movie nacheinander abgespielt werden.



1.11 Ausgewählte Kapitel ...

1.11.1 Links- und Rechtsdivision von Matrizen

In Matlab sind zwei Arten der Divison für Matrizen vorgesehen:

1. **Linksdivision**

   `X=A\B`

   stellt die Lösung der Gleichung

   `A*X=B`

   dar. `A,B` müssen die gleiche Anzahl von Zeilen haben. `X` hat dann die gleiche Zahl der Spalten wie `B`.

2. **Rechtsdivision**

   `B/A`

   stellt die Lösung der Gleichung

   `X*A=B`

   dar.

   `A,B` müssen die gleiche Anzahl von Spalten haben. `X` hat dann die gleiche Anzahl der Zeilen wie `B`.

1.11.2 Lösung von Gleichungssystemen

Matlab löst das allgemeine lineare Problem

`A*X=B`

durch Anwendung der Linksdivision

`X=A\B`

unter recht allgemeinen Voraussetzungen. Ist

- `A` eine allgemeine `N`×`N` Matrix, wird das Ergebnis durch Gaußelimination bestimmt. Ist `A` symmetrisch, positiv definit, wird die Cholesky-Zerlegung benutzt. `B` kann ein Spaltenvektor oder eine Matrix mit `N` Zeilen sein. Im Falle einer Matrix wird das Gleichungssystem simultan für alle Spalten der Matrix als rechte Seiten der Gleichung gelöst. (Anwendungsbeipiel: Rechteckige Platte mit festem Potential am Rand). Es wird eine Warnung ausgegeben, wenn `A` (fast) singulär ist.

- Ist `A` eine `M×N` Matrix mit `M<N` oder `M>N` und ist `B` ein Spaltenvektor oder eine Matrix mit `M` Zeilen, dann stellt `X=A\B` eine Lösung der unter- oder überbestimmten Gleichung im Sinne der kleinsten quadratische Abweichungen dar.

☐ Gegeben sei das System:

```
 2*x + 4*y    -z=2
   x + 2*y + 2*z=3
-4*x +   y + 5*z=4
```

In Matlab:

```
» A=[2 4 -1; 1 2 2 ; -4 1 5]
A =
     2     4    -1
     1     2     2
    -4     1     5

» B=[ 2 3 4]'
B =
     2
     3
     4
```

Die Lösung ist:

```
» X=A\B
X =
    0.1556
    0.6222
    0.8000
```

Test:

```
» A*X
ans =
    2.0000
    3.0000
    4.0000
```

1.11.3 Iterative Lösung linearer Gleichungssysteme

Matlab bietet eine Reihe moderner iterativer Solver für die Lösung von großen linearen Gleichungssystemen an. Eine detaillierte Besprechung würde hier den Rahmen sprengen. Es seien nur einige allgemeine Bemerkungen angefügt.
Folgende Solver sind verfügbar:

Funktion	Methode deutsch	Methode englisch
bicg	Bikonjugiertes Gradientenverfahren	Biconjugate Gradient
bicgstab	Bikonjugiertes stabilisiertes Gradientenverfahren	Biconjugate Gradient Stabilized
cgs	Konjugiertes quadratisches Gradientenverfahren	Conjugate Gradient Squared
gmres		Generalized Minimum Residual
pcg	Vorkonditioniertes konjugiertes Gradientenverfahren	preconditioned conjugate gradient
qmr		Quasiminimal Residual

Alle diese Methoden sind geeignet die Gleichung `A*X=B` zu lösen, wobei `A` eine `N`×`N`-Matrix und `B` ein Spaltenvektor der Länge `N` sein muß. Einzig die Methode `pcg` ist auf symmetrische, positiv definite Matrizen `A` beschränkt. Alle Methoden können dünn besetzte Matrizen (`sparse`) verarbeiten.
Im einfachsten Fall werden die Methoden in der Form `Funktion(A,B)` angewendet. Es werden aber viele weitere Optionen ermöglicht. Es können die Genauigkeit, die Zahl der Iterationen und andere Parameter angegeben werden (s. hierzu help Funktion).
Das besondere an diesen Methoden ist die mögliche Verwendung von sog. Prekonditionierern, d.h. Matrizen `M1, M2`, die die Konvergenz der Iteration beschleunigen. Hierzu wird das Gleichungssystem `A*X=B` in das äquivalente System `M1-1*A*M2-1*M2*X=M1-1*B` überführt und geeignet gewählte Matrizen `M1,M2` den Solvern mit übergeben.

Index