Technische Universität Berlin

Institut für Konstruktion, Mikro - und Medizintechnik

Fachgebiet Mikrotechnik

Prof. Dr. rer. nat. Heinz Lehr

Integrierte Lehrveranstaltung

Engineering Tools / Bachelor

Sommersemester 2016

Übungseinheit

MATLAB

Übungsleiter

Dipl.-Ing. Kilian Helfmeier

Tel. 030 - 314 - 79886

helfmeier@fmt.tu-berlin.de

1 Einführung

1.1 Was ist MATLAB?

MATLAB ist ein Softwarepaket für numerische Berechnungen und für die Visualisierung vonDaten im technisch-wissenschaftlichen Bereich.

MATLAB wurde in den 70er Jahren an der University of New Mexico und der Stanford University entwickelt, um Kurse aus Lineare Algebra und Numerische Analysis zu unterstützen. Der Name ("Matrix Laboratory") erinnert noch daran. Heute ist MATLAB ein universelles Werkzeug, das in weiten Bereichen der angewandten Mathematik eingesetzt wird.

Bei MATLAB stehen numerische Rechnungen und die Darstellung von Zahlenmaterial im Vordergrund. Der Grundbaustein ist eine Matrix, deren Dimensionen nicht explizit definiert werden müssen. Dadurch können numerische Probleme innerhalb kürzester Zeit gelöst werden. Man kann MATLAB auf zwei Arten verwenden:

- Bei der interaktiven Verwendung werden Anweisungen direkt über die Tastatur eingegeben und sofort ausgeführt.
- Für umfangreiche Probleme ist es empfehlenswert, MATLAB als Programmiersprache einzusetzen. Dabei wird eine Abfolge von Anweisungen in sogenannte M-Files abgespeichert. m - Files sind ASCII-Files und werden mit einem Texteditor geschrieben. Sobald sie im Commandfenster aufgerufen werden, führt sie der MATLAB-Interpreter wie ein Programm aus.

Für spezielle Anwendungen, wie beispielsweise Bildverarbeitung oder die Simulation von Regelkreisen, gibt es sogenannte Toolboxen. An diesem Institut stehen die folgenden Toolboxen zur Verfügung:

SymbolicMath Toolbox, Simulink, Image Processing Toolbox und die Image Aquisition Toolbox

1.2 Programmübersicht

Nach dem Start von MATLAB öffnen sich die Fenster *Command Window, Workspace, Command History* und *Current Folder* (siehe Abbildung 1-1). Für den Fall, dass eines der Fenster nicht geöffnet ist, lässt sich dieses im Menü *Desktop* wieder öffnen.

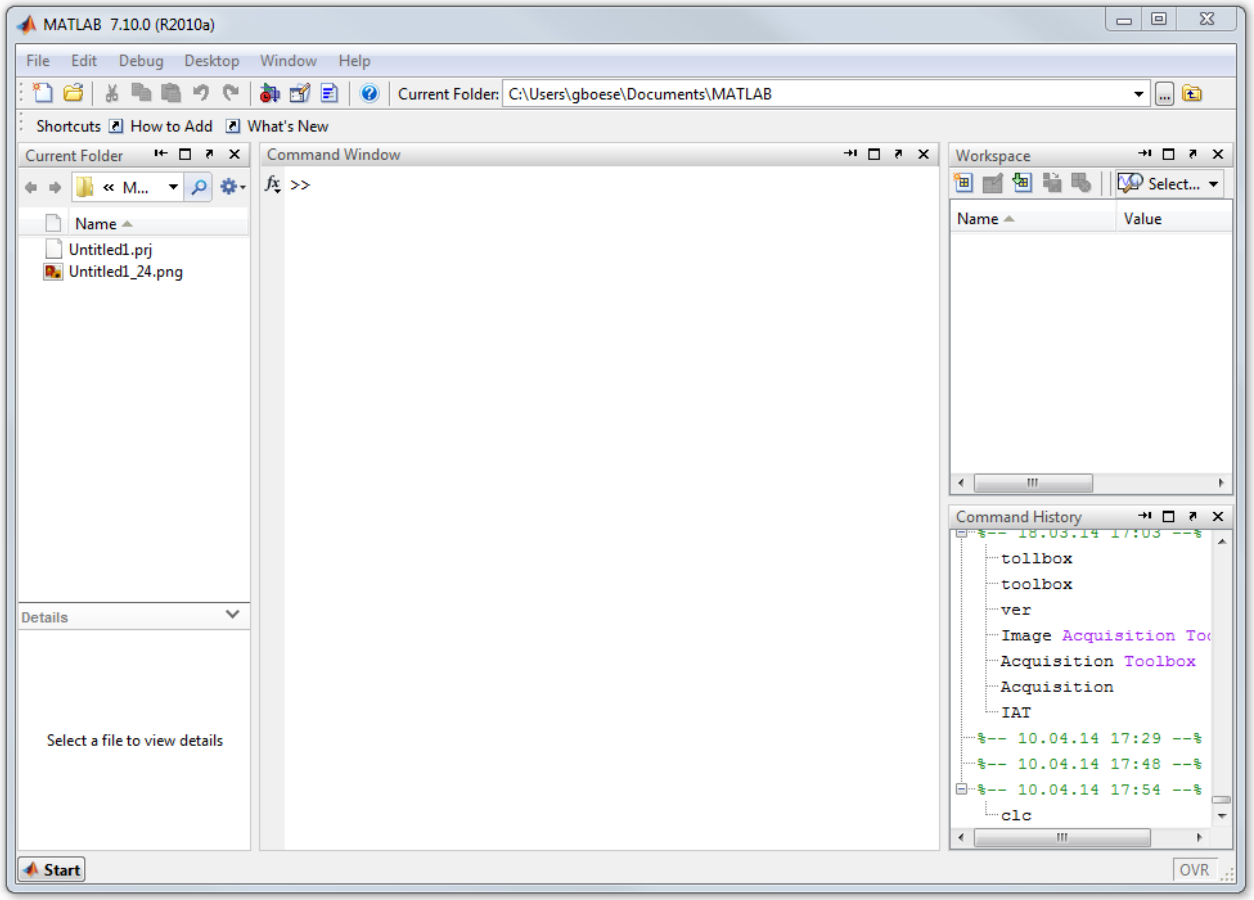

Abbildung 1-1MATLAB-Desktop

Nach dem Start von MATLAB öffnet sich das Kommandofenster (commandwindow) und es erscheint das Prompt ». In demKommandofenster werden MATLAB-Anweisungen eingegeben und die Ergebnisse ausgegeben. Zur Darstellung von Graphiken wird ein eigenes Graphikfenster geöffnet. Die Ausgabe von Graphiken wird in einem späteren Kapitel behandelt.

Alle Anweisungen werden nach dem Prompt » eingegeben und mit RETURN bestätigt. MATLAB nennt das Ergebnis **ans** (kurz für answer):

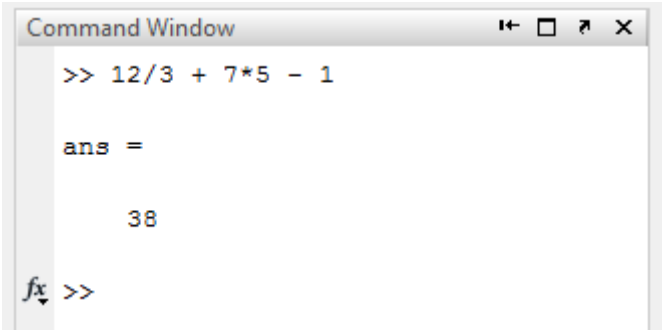

Abbildung 1-2 Ein- und Ausgabe im Command Window

Die Grundfunktionen Addition +, Subtraktion -, Multiplikation *,Division / und Potenz ^ werden wie gewohnt bezeichnet und verwendet.

Ein mehrfach benötigter Ausdruck kann in eine Variable gespeichert werden. MATLAB legt die Variablen automatisch in dem so genannten Workspace ab.

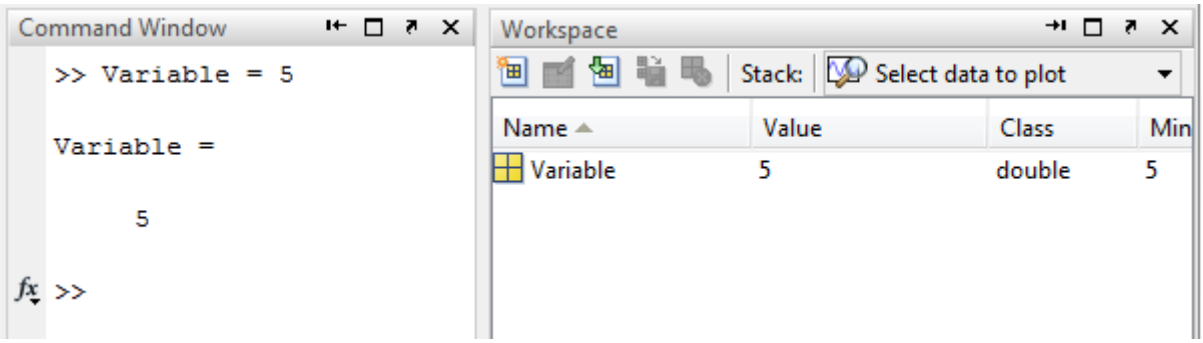

Abbildung 1-3Variablen im Workspace

Die Variablen werden im Workspace tabellarisch mit allen relevanten Informationen aufgeführt und lassen sich wenn nötig auch editieren und löschen.Um eine Variable aufzurufen, genügt die Eingabe des Variablennamens im Command Window.

Folgende Regeln sind bei der Definition von Variablen zu beachten:

- Ein Variablenname darf keine Sonderzeichen außer dem Unterstrich _ enthalten.
- Das erste Zeichen muss ein Buchstabe sein.
- Der Name darf nicht mehr als 19 Zeichen enthalten.

MATLAB unterscheidet zwischen Groß- und Kleinbuchstaben. Daher sind a und Aunterschiedliche Variablen (man sagt Variablen sind case sensitive).

Ein Semikolon am Ende der Eingabe bewirkt, dass MATLAB das Ergebnis zwar auswertet, aber nicht ausgibt.

Die Reihenfolge der Rechenoperationen entspricht der bekannten Arithmetik (Potenz vor Punkt vor Strich) und kanndurch Klammersetzung geändert werden.

Das Dezimaltrennzeichen ist, wie im Englischen allgemein üblich, ein Punkt. Die Eingabe „0,5“ interpretiert MATLAB als zwei getrennte Kommandos.

>> 0,5
ans = 0
ans = 5

Die richtige Eingabe lautet:

>> 0.5
ans = 0.5000

Das bedeutet weiterhin, dass mehrere Befehle, durch Kommas getrennt, in einer Zeile eingegeben werden können. Der Ausdruck links von einem Gleichheitszeichen wird immer durch den Ausdruck rechts definiert.

>> a=3, b=4, c=a+b
a = 3
b =4
c = 7

2 Teil I

2.1 Eingebaute Funktionen

Es gibt in MATLAB eine Vielzahl eingebauter Funktionen. Ein Beispiel ist die Wurzelfunktion(squareroot):

>> a=sqrt(9)
a = 3

Die Argumente werden in nachfolgenden Klammern an die Funktionen übergeben. Es gibt natürlich noch eine weitere Anzahl von verschiedenen mathematischen Funktionen.Einen Überblick kann man sich mit der Hilfe **helpelfun**geben lassen. Nachfolgend sind einige häufig verwendete Funktionen aufgeführt:

sin( )	- Sinus
cos( )	- Cosinus
tan( )	- Tangens
sqrt()	- Wurzelfunktion
exp( )	- Exponentialfunktion
log( )	- natürlicher Logarithmus
log10( )	- dekadischer Logarithmus
abs( )	- Betrag

2.2 Vektoren

Nehmen wir an, wir möchten die Werte von $\sqrt{x}$berechnen, für x = 0, 2, 4, 4, 5, 6.Bei dieser Aufgabenstellung spielt MATLAB sein ganzes Können aus. Mit einer einfachenProgrammiersprache müsste jetzt eine Schleife geschrieben werden, in der jeder einzelneWert berechnet wird. In MATLAB kann man die Werte vektorisieren d.h. die x-Werte lassen sichin einem Vektor zusammenfassen.

>> x=[0 2 4 6 8 10 12]
x = 0 2 4 6 8 10 12

Vektoren werden in eckigen Klammern eingegeben. Die einzelnen Elemente eines Zeilenvektorssind durch Leerzeichen oder Komma zu trennen und die Elemente eines Spaltenvektors durch

Semikolons. Auf die Funktionen, die im letzten Kapitel genannt sind, können auch Vektoren angewendet wenden. Dabei wird diese für die einzelnen Elemente ausgeführt.

```
>> y=sqrt(x)
y = 0   1.4142   2.00002.44952.82843.16233.4641
```

Diese unkomplizierte Art, einen Befehl (hier die Wurzelfunktion) elementweise anzuwenden, ist eine große Stärke von MATLAB. Sie ermöglicht eine rasche Verarbeitung von großen Datenmengen.

Tipp: Der Vektor x = [0 2 4 6 8 10 12] kann kürzer eingegeben werden:

```
>>x = 0:2:12
x =0 2 4 6 8 10 12
```

Die Schreibweise 0:2:12 bedeutet: beginne mit **0** und zähle **2** dazu, dann zähle solange wieder **2** dazu, bis die Grenze **12** erreicht ist. Diese Anweisung ist sehr nützlich bei der Eingabe von Vektoren mit vielen Elementen. Man kann 0:2:12 auch in runden oder eckigen Klammern schreiben: (0:2:12) oder [0:2:12]. Die Schrittweite (increment) darf auch negativ sein:

```
>>u = 25:-2:0
u = 25:-2:0
u =
    25     23     21     19     17     15     13     11      9      7      5      3      1
```

Dieser Vektor hat 13 in Spalten angeordnete Elemente. Beachten Sie, dass das letzte Element hier 1 ist und nicht 0 (zieht man immer wieder 2 von 29 ab, so trifft man nie auf 0. Die letzte Zahl größer 0, die man erhält, ist 1). Ist die Schrittweite gleich 1, so kann man die Angabe der Schrittweite weglassen:

```
>>z = 7:11
z =7 8 9 10 11
```

Auf die einzelnen Werte eines Vektors kann einfach zugegriffen werden, indem der Index in Klammern hinter die Variable geschrieben wird. Die Indizierung beginnt bei MATLAB mit 1.

```
>>z(4)
ans =10
```

Achtung: Enthält der Vektor y die Funktionswerte von x, so ist dementsprechend **y(2)** inMATLAB das zweite Element des Vektors **y**

```
>>x = 0:2:12; y = sqrt(x); y(2)
ans = 1.4142
```

und nicht der Wert von x an der Stelle x=2!

2.3 Elementweise Operationen

Vektoraddition und die Multiplikation eines Vektors mit einem Skalar erfolgtnach den üblichen mathematischen Regeln,nämlich elementweise. Nachfolgend wird jedes Element des Vektors [1 2 3] mit 2 multipliziert und der so entstandene Vektor wird w genannt.

```
>>w = [1 2 3]*2          % Vektor * Skalar
w = 2 4 6
```

Zu jedem Element von [2 -1 9] wird das entsprechende Element von [1 3 6] addiert. Die Subtraktion zweier Vektoren erfolgt analog zur Addition.

```
>>[2 -1 9] + [1 3 6]     % Vektor + Vektor
ans =3 2 15
```

Achtung: Es können nur Vektoren bzw. Matrizen mit gleichen Dimensionen addiert odersubtrahiert werden! Man sagt auch, sie müssen dieselbe Länge (length oder dimension) haben:

```
>> w + [1 4 6 7]
??? Error using ==> plus
Matrix dimensions must agree.
```

Die Länge eines Vektors (d.h. die Anzahl seiner Elemente) erhalten wir mit dem Befehl *lenght*:

```
length(w)
ans =3
```

Neben den aus der Mathematik bekannten Vektoroperationen sind in MATLAB weitereElementweise Vektoroperationen definiert, wie beispielsweise die elementweise Addition:

```
[2 5 7] + 3             % Vektor + Zahl
ans =5 8 10
```

Zu jedem Element von [2 5 7] wird die Zahl 3 addiert. Neu ist auch die in MATLAB definierte elementweiseMultiplikation. Dazu benötigen wir zwei Vektoren der gleichen Länge:

```
>>a = 1:2:10,        b = 1:5

a =1 3 5 7 9
b =1 2 3 4 5
```

Nun wollen wir einen neuen Vektor bilden, indem wir die Elemente mit dem gleichen Index aus den Vektoren a und b miteinander multiplizieren. (d.h. a(1)*b(1), a(2)*b(2), usw.) Der MATLAB-Befehl für diese Operation ist allerdings nicht der Stern (Asterisk) „*“, sondern das Zeichen mit einen Punkt davor („ .* „). Beispiel:

```
>>a.*b                  % elementweise Multiplikation
ans =1 6 15 28 45
```

Das Ergebnis ist wieder ein Vektor der Länge 5.

Achtung: Der Punkt vor dem Asterisk ist notwendig! Die Anweisung **a*b** bedeutet die in der Mathematik übliche Matrixmultiplikation. Diese ist völlig verschieden von der elementweisen-Multiplikation und kann nur durchgeführt werden, wenn die Anzahl der Spalten von a gleich der Anzahl der Zeilen von b ist:

```
>>a*b
??? Error using ==> *
Inner matrix dimensions must agree.
```

Die elementweise Division in MATLAB ist ganz analog definiert:

```
>>a./b
ans =1.0000 1.5000 1.6667 1.7500 1.8000
```

Jedes Element von a wird durch das entsprechende Element von b dividiert. Das elementweise-Potenzieren erfolgt analog:

```
>>a.^2                % elementweises Potenzieren
ans =1 9 25 49 81
```

2.4 Polynome

Polynome sind besonders einfache mathematische Funktionen, die vielfach Anwendung finden. Ein Polynom, beispielsweise $y = -x^3 + 2x^2 + 5$, kann über elementweise Operationen für einen bestimmten Vektor x errechnet werden. Einfacher kann ein Polynom in Matlab durch seine Koeffizienten angegeben werden.

Im Folgenden soll das Polynom $y = -x^3 + 3x^2 + 1$ für die Werte x = -1, 0, 1, 2, 3, 4 und 5 berechnet werden. Die Variable x wird als Vektor definiert. Ebensowerden die Polynomkoeffizienten – 1, 3, 0 und 1 zu dem Vektor „koeff“ zusammengefasst, beginnend mit der höchsten Potenz, in diesem Fall $–x^3$. Die Berechnung erfolgt mit der **polyva**l-Anweisung.

```
>>x = -1:5;
>>koeff = [-1 3 0 1];          % Koeffizient der höchsten Potenz zuerst
>>polyval(koeff, x)
ans = 5 1 3 5 1 -15 -49
```

2.5 Matrizen

Matrizen sind wertvolle Hilfsmittel bei der Verarbeitung von Daten. Als Beispieldient das folgende lineare Gleichungssystem:

```
3*x + 4*y – 2*z = 4
-x + 2*y +8*z = -1
2*x + – 5*z = 3
```

Es kann auch in der Form A*x = b geschrieben werden, mit einer Matrix A und zwei Spaltenvektorenx und b. Matrizen werden in MATLAB zwischen eckigen Klammern eingegeben.Das Ende einer Zeile wird durch einen Semikolon gekennzeichnet:

```
>>A= [ 3 4 -2; -1 2 8; 2 0 -5]
A =     34 -2
        -1 2 8
        2 0 -5
```

Die Transponierte einer (reellen) Matrix erhalten wir, indem wir ein Apostroph **'** anhängen

```
>>C = A'
C =      3 -1 2
         4 2 0
         -2 8 -5
```

Hat die Matrix auch komplexe Elemente, so wird sie durch Anhängen von **'** transponiert unddie Elemente werden komplex konjugiert. Geben wir den Vektor b in der gewohnten Form:

```
b = [4 -1 3];
```

ein, so erhalten wir eine einzeilige Matrix, d.h., einen Zeilenvektor. Die Schreibweise A*x = bverlangt aber, dass b ein Spaltenvektor ist. Mit **b'** entsteht aus dem Zeilenvektor b durchTransponieren ein Spaltenvektor. Wir definieren den Vektor b neu durch

```
>>b = b'
b =     4
        -1
        3
```

Tipp: Einen Spaltenvektor erhalten Sie auch mit **b(:)**. Wie bei Vektoren können wir auch einbestimmtes Element einer Matrix herausgreifen:

```
>>C(3,2)
ans =8
```

C(3,2) bedeutet das Element in der dritten Zeile und zweiten Spalte der Matrix C. Die ganzezweite Zeile von C wird ausgegeben mit:

```
>>C(2,:)
ans = 4 2 0
```

C(2,:) bedeutet: nimm die 2. Zeile und alle Spalten (der Doppelpunkt steht hier für "alleSpalten"). Entsprechend erhalten wir zum Beispiel die ganze erste Spalte von C durch

```
>>C(:,1)
ans =  3
       4
       -2
```

Der Doppelpunkt stehtfür „alle Zeilen“ für das Beispiel *>>C(:,1)* und „alle Spalten“ für *>>C(2,:)*.

Möchten wir ein bestimmtes Element der Matrix C ändern, zum Beispiel C(3,2) auf den Wert7, so geben wir ein

```
>>C(3,2) = 7
C =     3 -1 2
        42 0
        -2 7 -5
```

Soll eine ganze Spalte oder Zeile von C gelöscht werden, so setzen wir diese Spalte (bzw.Zeile) gleich**[]**. So wird etwa die erste Spalte von C gelöscht, und die Dimension wird verändert.

```
>>C(:,1) = []
C =     -12
        2 0
        7 –5
```

Die Dimension einer Matrix (d.h., ihre Zeilen- und Spaltenanzahl) wird über den Befehl **size**ermittelt.

```
>>size(C)
ans = 3 2
```

Als Ergebnis erhält man den Vektor [Zeilenanzahl, Spaltenanzahl], hier [3 2]. Tipp: Die Anweisung**whos**gibt ausführliche Informationen über die im Workspace gespeicherten Variablen.

Zwei spezielle quadratische Matrizen werden oft gebraucht: die Nullmatrix und die Einheitsmatrix.Im Fall von 3 Zeilen und Spalten werden sie erzeugt durch:

```
>>zeros(3)                    % 3x3 Nullmatrix
ans =   0 0 0
        0 0 0
        0 0 0
>>eye(3)                      % 3x3 Einheitsmatrix (3-by-3 identitymatrix )
ans =   1 0 0
        0 1 0
        0 0 1
```

2.6 Matrixoperationen

A+B, A-B, A*B bezeichnen die aus der Mathematik üblichen Matrizenoperationen.

Wichtig: Der Stern ' ***** ' allein bedeutet die übliche Matrizenmultiplikation (die nur definiertist, wenn die Anzahl der Spalten von A gleich der Anzahl der Zeilen von B ist). Darüber hinausist, wie bei den Vektoren auch, die elementweise Multiplikation definiert, die mit demSymbol '**.*** ' bezeichnet wird:

```
>>[1 2; -2 5] .* [3 6; 0 -1]
ans =  3 12
       0 -5
```

Jedes Element von [1 2; -2 5] wird mit dem Element der gleichen Position von [3 6; 0 -1] multipliziert.

Mehr Informationen zur Matrizeneingabe und –manipulation erhalten Sie zum Beispiel mit**helpelmat**, **helpmatfun**oder auch **helpcolon**und **helpparen**.

Existiert die Inverse A^{-1}, so ist die Lösung x von A*x = b gegeben durch $x = A^{-1}*b$. A istinvertierbar genau dann, wenn die Determinante von A ungleich Null ist. Die Determinante einer Matrix errechnet sich mit dem Befehl **det( )**.

```
>>det(A)                    % Determinante der Matrix A
ans = 22
```

Wir erhalten die Inverse A^{-1}mit **inv(A)**. Damit kann nun auch für das lineare Gleichungssystem die Werte für den Vektor x berechnet werden.

```
>>x = inv(A)*b
x =    2.1818
      -0.5000
       0.2727
```

MATLAB bietet noch eine andere Schreibweise für $x = A^{-1}*b$:

```
>>x = A\b
x =    2.1818
      -0.5000
       0.2727
```

Das Symbol **'\'** bezeichnet die sogenannte Linksdivision (leftdivision). Der Name kommtdaher, dass nun der Nenner links steht und der Bruchstrich nach links geneigt ist. Hier stehtA\ sozusagen für die Inverse von A. Im Fall von Skalaren ist die Linksdivision

```
>>2\3
ans = 1.5000
```

gleich der Rechtsdivision 3/2 (die Multiplikation von Skalaren ist kommutativ). Bei Matrizenaber ist inv(A)*b (Linksdivision) ungleich b*inv(A) (Rechtsdivision), der letzte Ausdruckist nämlich gar nicht definiert. Daher erhalten wir eine Fehlermeldung, wenn wir eingeben:

```
>>b/A
??? Error using ==> /
Matrix dimensions must agree.
```

Hinweis: Die Verwendung von **x = A\b** anstelle von **x = inv(A)* b** hat verschiedene Vorteile:

- Erstens werden bei Eingabe von A\b weniger interne Rechenschritte durchgeführt (es wirddas Gauß'sche Eliminationsverfahren verwendet). Daher erhält man so vor allem bei größerenProblemstellungen schneller eine Lösung.
- Zweitens liefert die Eingabe von x = A\b auch dann eine Lösung, wenn das Gleichungssystemnicht eindeutig lösbar ist, demnach die Inverse A^{-1}nicht existiert. Im Fall einesüberbestimmten Systems erhält man zum Beispiel eine Lösung, die den quadratischenFehler von A*x - b = 0 minimiert.

2.7 m-Files: MATLAB als Programmiersprache

Man unterscheidet zwei Arten von m-Files: Script-Files und Function-Files. Beide werden mitder Endung „ .m " abgespeichert, haben jedoch unterschiedliche Aufgaben.

2.7.1 Script-Files

In einem Script-File (auch: Script m-File oder kurz Script) kann eine Abfolge von MATLAB-Anweisungen gespeichert und ausgeführt werden. Dies ist insbesondere dann interessant, wenn eine bestimmte Befehlsfolge mehrfach benötigt wird. Wird der Name des Script-Files im Commandfenster nach demPrompt eingegeben, so werden die Befehle im Script-File der Reihe nach ausgeführt. Bei seinem Aufruf stehen dem Script-File alle im Workspace gespeicherten Variablen zur Verfügung. Umgekehrt sind alle Variablen,die im Script-File definiert werden, nach dessen Ablauf noch im Workspace definiert(d.h., die Variablen des Script-Files sind **global**).

Beispiel:

Es soll ein Script-File geschrieben werden, das bei Aufruf eine Wurfparabel zeichnet. Die Anfangsgeschwindigkeitv_kmh und den Abwurfwinkel phi_grad soll das Script-File aus demWorkspace nehmen. Um ein Script-File zu erzeugen, wählt man in der Menüzeile NEWund dann m-FILE. Es öffnet sich daraufhin ein Texteditor-Fenster. Hier geben wir die Befehle ein, die MATLAB ausführen soll:

```
% Plot einer Wurfparabel
% übernimmt aus Workspace: Abwurfgeschwindigkeit v_kmh und Abwurfwinkel phi_grad

g = 9.81;
v_ms = v_kmh/3.6;                % Umrechnung in m/s
phi_rad = phi_grad*pi/180;       % Umrechnung ins Bogenmaß
wurfzeit = 2*v_ms*sin(phi_rad)/g;
t = 0:0.01:wurfzeit;
x = v_ms*t*cos(phi_rad);
y = v_ms*t*sin(phi_rad) - 0.5*g*t.^2;
plot(x,y);
grid on
```

Nun speichern wir dieses File mit der Endung **.m** ab, zum Beispiel als **wurf.m**. Im Commandfenstermüssen noch die Variablen „v_kmh“ und „phi_grad“ definiert werden, beispielsweise mit:

>>v_kmh = 100; phi_grad = 60;

Geben wir nun *wurf*ein, so führt MATLAB die Befehle aus dem Filewurf.m nacheinander aus, so, alsob sie im Commandfenster nach dem Prompt eingegeben werden.

Achtung: Bei Eingabe von *wurf* durchsucht MATLAB das aktuelle Verzeichnis und alle Verzeichnisseim MATLAB-Suchpfad (dieser kann mit **path**ausgegeben werden). Wenn Sie dieFehlermeldung:

>>wurf
??? Undefinedfunctionor variable 'wurf'.

erhalten, so müssen Sie entweder in das Verzeichnis, in dem sich wurf.m befindet, wechselnoder

- den vollständigen Pfad von wurf.m eingeben oder
- das Verzeichnis, in dem wurf.m sich befindet, dem Suchpfad hinzufügen (verwenden Siehelppath)

Eine Liste aller im aktuellen Verzeichnis enthaltenen m-Files erhält man mit dem Befehl**what**.

Zum besseren Verständnis sollten die erstenpaar Zeilen eines Script-Files Kommentarzeilen sein, die den Zweck und die Arbeitsweise desScript-Files beschreiben. Die Kommentarzeilen haben noch eine andere nützliche Funktion:sie werden ausgegeben, wenn der **help**-Befehl verwendet wird. So erhalten wir die erstenKommentarzeilen von wurf.m mit

>>helpwurf
Plot einer Wurfparabel
übernimmt aus Workspace: Abwurfgeschwindigkeit v_kmh,Abwurfwinkel phi_grad

2.7.2 Function-Files

Über Function-Files (auch: Function m-Files) können in MATLAB neue Funktionen definieren werden.Ein Function-File wird, wie auch ein Script-File, mit der Endung .m abgespeichert.Der Unterschied zu einem Script-File besteht darin, dass die im Function-File definierten Variablen**lokal** sind und demnach nicht im Workspace erscheinen.

Beim Aufruf eines Function-Files werden ihm eine oder mehrereVariablen als Argumente übergeben. Es können auch Matrizen übergeben werden. Nach Auswertung der Befehle im Function-File wird ein Wert, der Funktionswert, ausgegeben. Wir wollen als einführendes Beispieleine Funktion definieren, die einen Winkel x vom Bogenmaß ins Gradmaß umrechnet.Die Funktion soll mit r2d(x) (radianttodegree) bezeichnet werden:

```
function d = r2d(x)
% r2d(x) rechnet einen Winkel x vom Bogenmaß ins Gradmaß um
d = x*180/pi;
```

Wir speichern das File unter dem Namen r2d.m ab.

Wichtig: Der Funktionsname muss immergleich sein, wie der Name unter dem das File abgespeichert wird. Nun können wir die Funktionr2d(x) verwenden. Um zum Beispiel x = 3.14 vom Bogenmaß ins Gradmaß umzurechnen,ist nur eine einfache Eingabe im Commandfenster nötig:

```
>>r2d(3.14)
ans =179.9087
```

Natürlich kann das Argument x auch ein Vektor oder eine Matrix sein:

```
>>x = (0:0.5:2)*pi; r2d(x)
ans = 0 90 180 270 360
```

Ein Function-File muss nach folgender Struktur aufgebaut sein:

```
functionFunktionswert = Funktionsname(Argumentliste)
% Kommentare
Anweisungen
Funktionswert =...;
```

Es beginnt immer mit dem Wort '**function**'. Danach folgt der Funktionswert, dem im Anweisungsteil Werte übergeben werden. Dieserkann auch ein Vektor oder eine Matrix sein:

```
functionFunktionswert = Funktionsname(Argumentliste)
% Kommentare
Anweisungen
a =...; b =...; ...
Funktionswert = [a,b,..];
```

Die auf die erste Zeile folgenden Kommentarzeilen haben dieselbe Bedeutung wie bei einemScript-File. Sie werden ausgegeben, wenn man die help-Anweisung verwendet.Die allererste Kommentarzeile wird auch von der **lookfor**-Anweisung durchsucht.

Wie schonzu Beginn erwähnt, sind alle im Function-File definierten Variablen lokal und werden nicht im Workspace gespeichert. Umgekehrt kanndas Function-File auch auf keine Variablen aus dem Workspace zugreifen. Die Variablen müssen als Argument übergeben werden.Dies soll an dem folgenden Beispiel deutlich werden. Zunächst wird ein Funktion-File „f.m“ erstellt.

```
function y = f(x)
% Parabel um c nach oben/unten verschoben
c = 3;
y = x^2 + c;
```

Werten wir nun die Funktion an der Stelle x = 2 aus:

>>f(2)
ans = 7

Es wird, wie erwartet, der Funktionswert mit dem Wert c = 3 berechnet. Die Variable c ist jedoch im Workspace nicht vorhanden und kann somit nicht weiterverarbeitet werden.

>>c
??? Undefined function or variable 'c'.

Geben wir nun der Variablen c im Workspace den Wert c = 1 und werten wir die Funktionwieder an der Stelle x = 2 aus:

>>c = 1, f(2)
ans = 7

Für die Auswertung der Funktion wird also nach wie vor der Wert c = 3 verwendet! Das "c"im Workspace hat mit dem "c" im Function-File nichts zu tun!

Um c vom Workspace aus ändern zu können, müssen wir es als globale Variable definieren.Das müssen wir sowohl im Workspace als auch im Function-File tun. Die hier notwendigeMATLAB Anweisung lautet**global**. Wir schreiben also einerseits das Function-File um,

function y = f(x)
% Parabel um c nach oben/unten verschoben
global c
y = x^2 + c;

und definieren auch im Commandfenster c als globale Variable:

>>global c;
c = 1, f(2)
ans =5

Nun verwenden Workspace und Function-File dasselbe "c". Viele von MATLAB zur Verfügunggestellten Funktionen sind selbst m-Files, im Gegensatz zu bereits im MATLAB-Prozessor eingebauten Funktionen wie sin(x). Ein Beispiel ist die Funktion **angle(x)**. Den Pfad des zugehörigen m-Files erhalten wir mit der Anweisung **which**:

>>which angle
C:\PROGRAMME\MATLAB\toolbox\matlab\elfun\angle.m

Den Code von angle können wir mit **edit angle** lesen (es öffnet sich ein Editor-Fenster mit den Anweisungen zu angle.m).

Tipp: Sie können in Ihrem m-File auch Kontrollstrukturen einbauen. MATLAB stellt dazu die **IF**-Anweisung sowie **FOR-** und **WHILE**-Schleifen zur Verfügung. Für nähere Informationen steht die Hilfe zur Verfügung:**helpif, helpfor, helpwhile**.

3 TEIL II

3.1 Graphik

In MATLAB kann man mit wenigen Befehlen sehr leicht Daten veranschaulichen. Die Anweisung**plot(x,y)** erzeugt ein Graphikfenster. (**Achtung**: dieses ist oft hinter den anderen geöffneten-Fenstern versteckt!). Die Argumente x und y sind Vektoren und enthalten die x- und y -Koordinaten der zu zeichnenden Datenpunkte. Es soll als Beispiel die Wurzelfunktionim Intervall [0,6] gezeichnet werden:

>>x = 0:0.5:6, y = sqrt(x),plot(x,y)

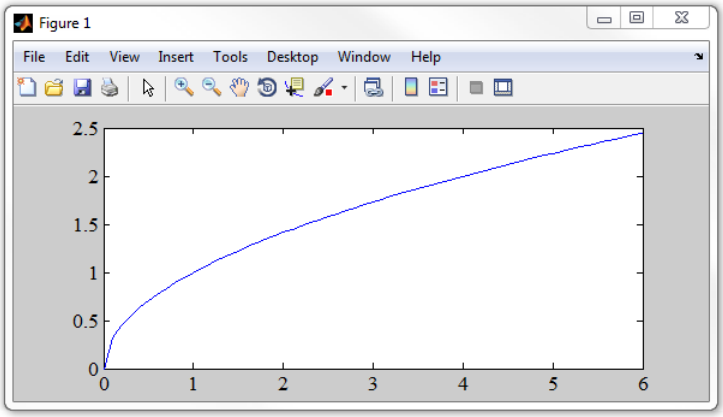

Abbildung 3-1 Plot einer Wurzelfunktion

Es öffnet sich ein Graphikfenster und die Funktionswerte von y =sqrt(*x)*werden für die Werte x = 0,0.5,1, ..., 5.5,6 gezeichnet. MATLAB wählt die Achsenbegrenzungen automatisch, beschriftetdie Achsen und verbindet die Datenpunkte durch gerade Linien.

Das nächste Beispiel zeigt, wie praktisch die in MATLAB möglichen elementweisenOperationen sind. Es ist die Funktion y = ln(x)/x im Intervall [1,8] zu zeichnen:

*>>x = 1:0.5:8, y = log(x)./x, plot(x,y,'*r')*

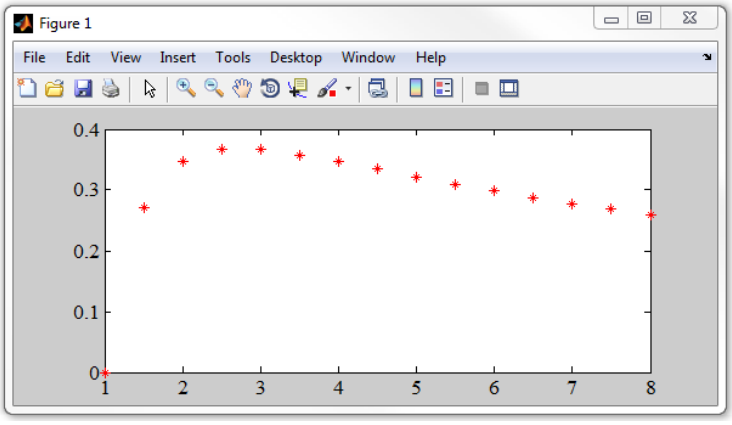

Abbildung 3-2 Plot des natürlichen Logarithmus

Der Stern (*) unter einfachen Anführungszeichen bewirkt, dass nur die Datenpunkte gezeichnet-werden, und zwar in Form von Sternen. Anstelle der Sterne sind auch Kreise(o), Punkte (.), Kreuze (+) und viele andere Symbole möglich. Der Buchstabe „r“ hinter dem Stern bewirkt, dass die Sterne in rot dargestellt werden. Andere Farben und Symbole sind unter **helpplot** zu finden.

Achtung: Es darf nur das Anführungszeichen **'**verwendet werden. Verwendet man ´ oder `oder" , so erhält man eine Fehlermeldung.

Möchte man die Anzahl der gezeichneten Datenpunkte vorgeben, so erzeugt man den Vektorx am besten mit der Anweisung**linspace**:

>>x = linspace(-pi,pi,30);

Hier besteht der Vektor x aus 30 Werten im gleichen Abstand, wobei der erste Wert und derletzte Wert π ist. Der **linspace** Befehl benötigt zwei bis drei Argumente: der erste Wert, der letzte Wert und fakultativ kann noch die Anzahl der Werte angegeben werden.Fehltdie Angabe für die Anzahl der Werte, so wird dafür standardmäßig 100 genommen.

*>>y = x.*sin(x).^2;*
>>plot(x,y);
>>grid on

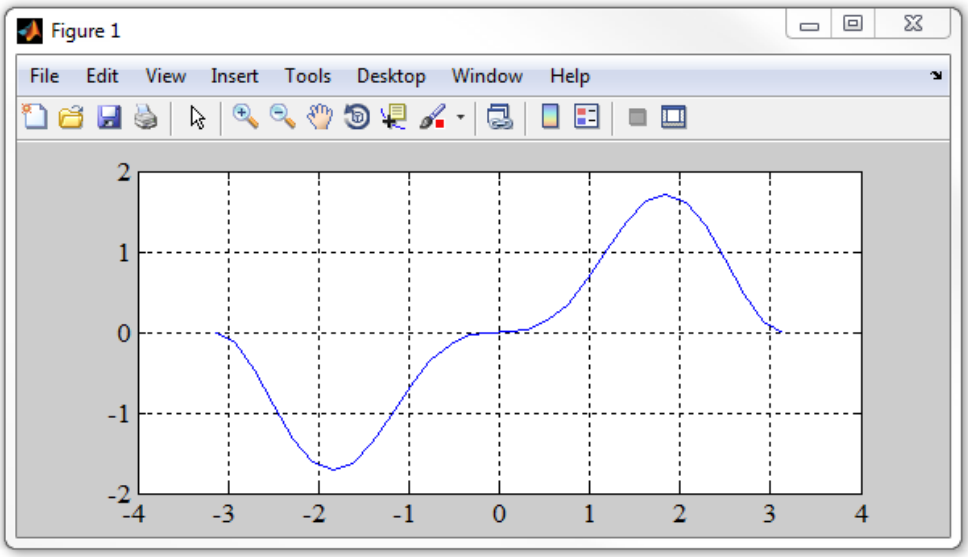

Abbildung 3-3 Plot von mit linspace erzeugten Werten

Die Anweisung **grid on**erzeugt Gitterlinien.

3.2 Beschriften einer Graphik

Die Koordinatenachsen werden mit **xlabel**und**ylabel**beschriftet. Die Anweisung **title** versiehtdie Graphik mit einer Überschrift und mit **text**kann man eine Beschriftung hineinsetzen.

>>plot(x,y);
>>xlabel('Zeit'), ylabel('Spannung')
>>title('Abb. 1')
*>>text(-0.8,0.5,'x*sin(x)^2')*

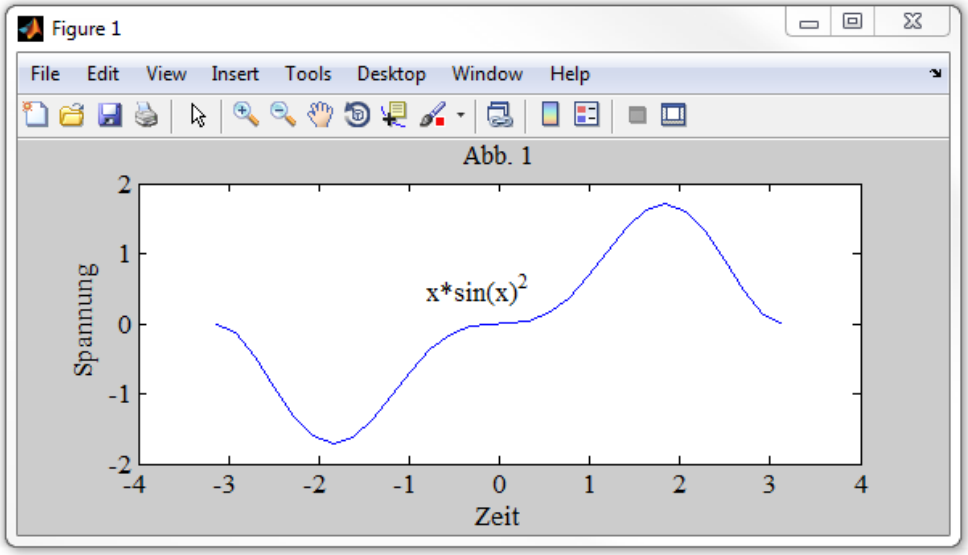

Abbildung 3-4 Beschriftung eines Plots

In der aktuellen Graphik wird die x-Achse mit ' Zeit ' beschriftet, die y-Achse mit 'Spannung'.Die ganze Graphik wird mit ' Abb.1 ' betitelt. Im Punkt mit den Koordinaten (-0.8,0.5)beginntder Text ' x*sin(x)^2 '.

Achtung: Es ist wichtig, zuerst die Anweisung **plot(x,y)** einzugeben und dann erst die Befehlezur Beschriftung der Graphik. Die Beschriftung wird immer nachträglich in die aktuelle Graphikeingefügt.

3.3 Änderung der Skalierung

Angenommen, es soll die Funktion y = tan(x) im Intervall [0,/2] gezeigt werden:

>>x=0:0.01:pi/2;
>>plot(x,tan(x));

Die Funktion wird nicht gut dargestellt, denn der von MATLAB automatisch gewählte y- Bereich ist zu groß.Der Zeichnungsbereich kann mit dem Befehl **axis([xmin, xmax, ymin, ymax])** geändert werden. Die Funktion wird dann im x- Bereich xmin ≤ x ≤xmax und im y-Bereich ymin ≤y ≤ymax dargestellt.

>>axis([0, pi/2, 0, 15]);

Gleiche Skalierung auf der x- und y-Achse erreichen wir mit **axisequal**. Die ursprüngliche (von MATLAB gewählte) Skalierung wird mit **axis normal** wiederhergestellt.

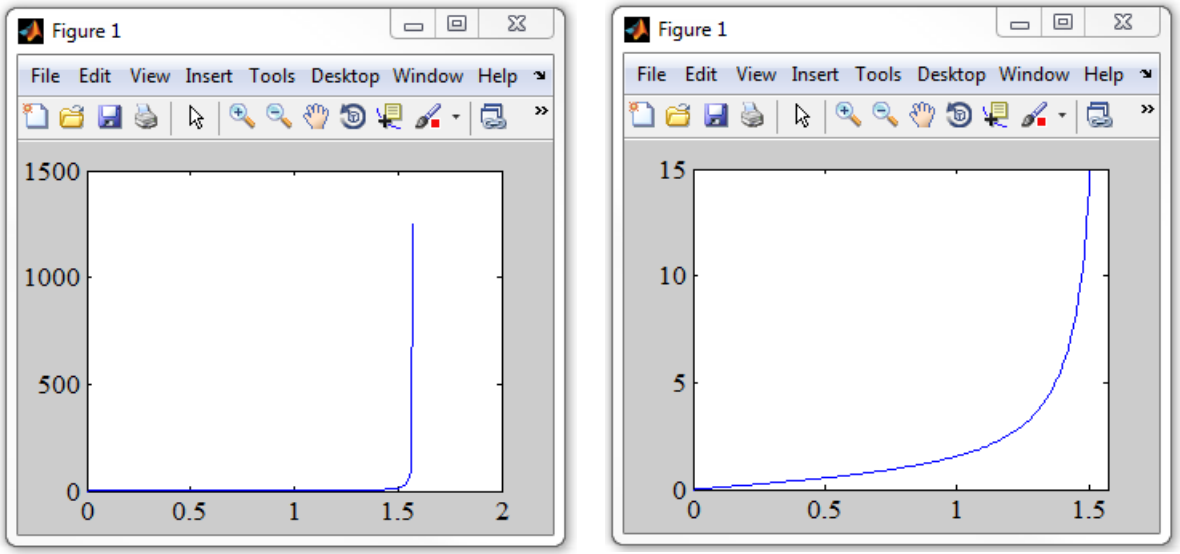

Abbildung 3-5 Plot ohne und mit Skalierung

3.4 Graphiknachbearbeitung

Eine einfachere Nachbearbeitung der Graphiken bieten die Menüpunkte des Plots *Insert* oder- *Tools*.

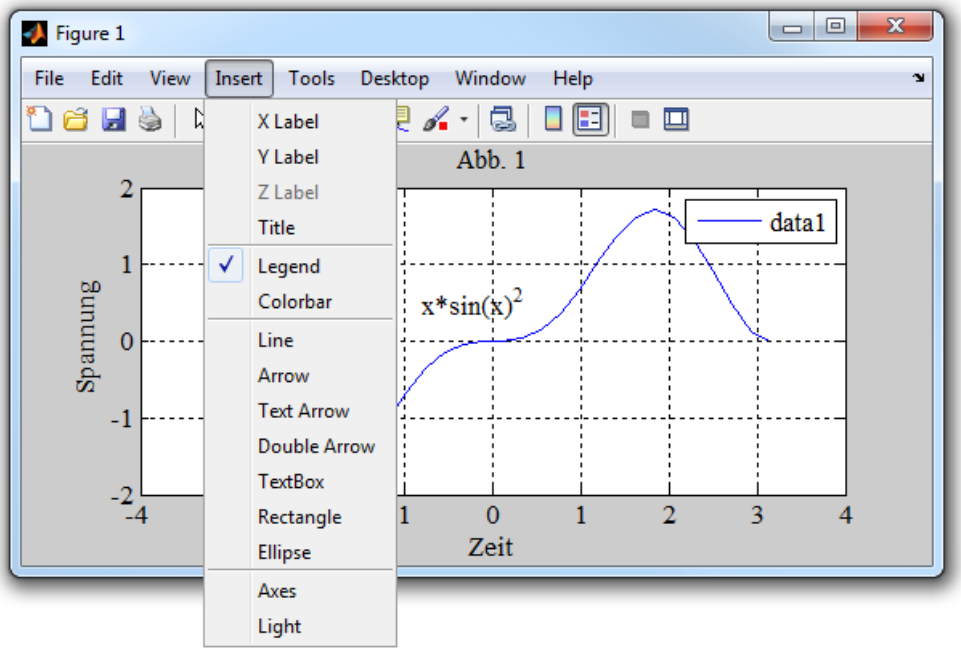

Abbildung 3-6 Menüpunkt für die Nachbearbeitung von Graphiken

Graphiken können auch auf andere Weise erstellt werden als durch den Befehl **plot(x,y).** Dazu werden im Workspace die entsprechenden Variablen ausgewählt, hier x und y, und dann per Rechtsklickder Plot-Befehl aufgerufen.

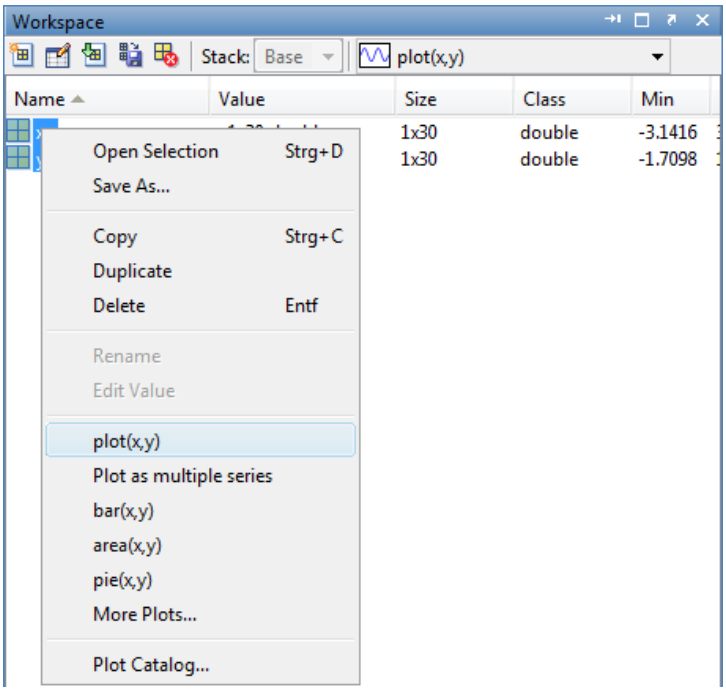

Abbildung 3-7 Alternative zu dem Befehl plot()

3.5 Gleichzeitiges zeichnen von mehreren Funktionen

Die Anweisung **plot(x,y,x,z)** zeichnet die Funktionen y(x) und z(x) in das gleiche Koordinatensystem.

>>x = linspace(-pi,pi,30); y = sin(x); z = cos(x);
>>plot(x,y,x,z)

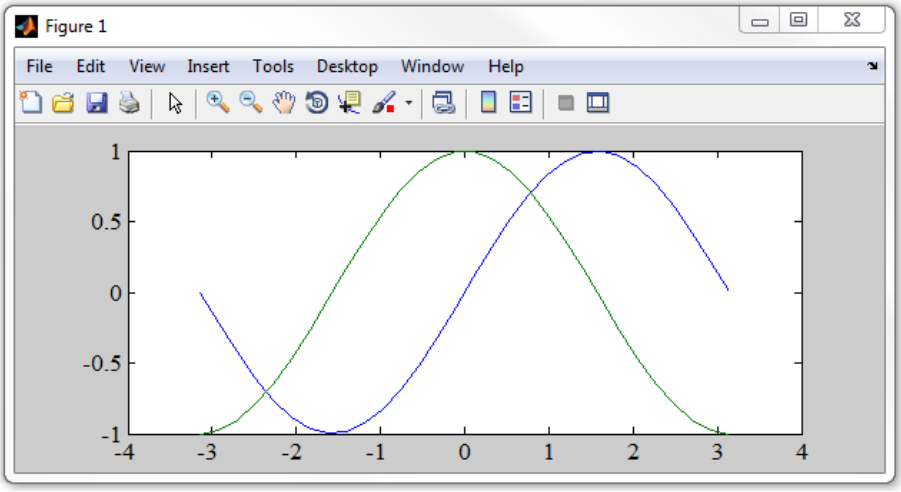

Abbildung 3-8 Mehrere Funktionen in einem Plot

Die Graphen der Sinus- und Kosinusfunktion werden automatisch in verschiedenen Farben in ein- und dasselbeGraphikfenster gezeichnet.

Tipp: Eine andere Möglichkeit bietet die Anweisung **hold on**. Sie bewirkt, dass die aktuelleGraphik im Graphikfenster festgehalten wird. Anschließende plot-Anweisungen fügen dieGraphiken in dasselbe Achsenkreuz hinzu. **hold off** beendet diese Funktion. Bei dieser Variante muss jeder Graphik die Farbe separat zugewiesen werden mit beispielsweise ‘r‘ (rot) oder ‘b‘ (blau).

```
>> x=linspace(-pi,pi,30);
>> y1=sin(x);
>>y2=cos(x);
>> plot(x,y1,'r')
>> hold on
>> plot(x,y2,'b')
>> hold off
```

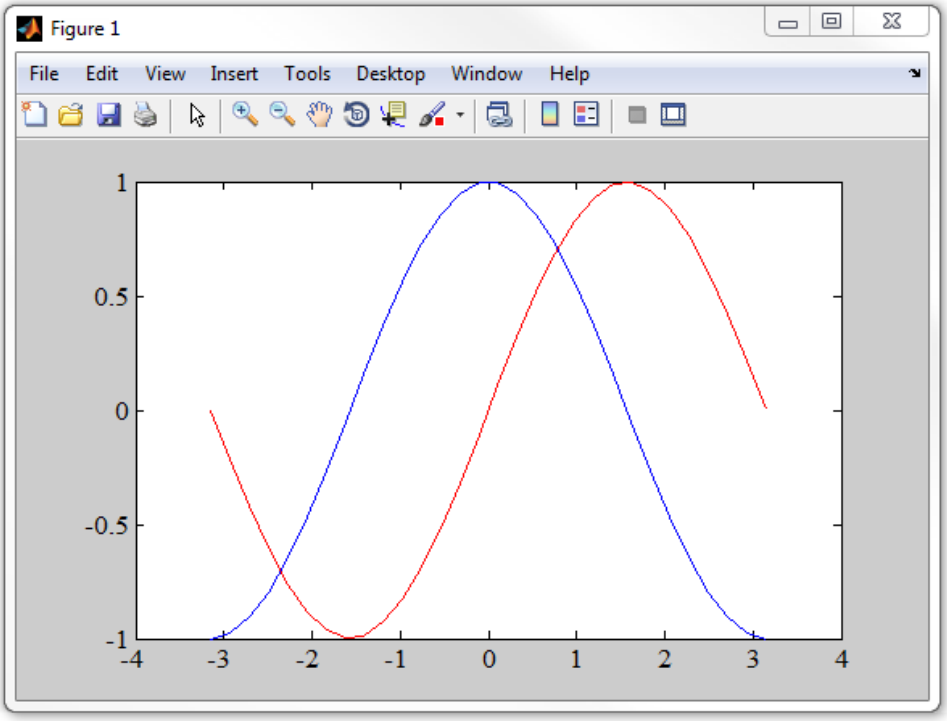

Abbildung 3-9 Zwei Plots in einem Fenster mit dem Befehl hold

Matlab ermöglicht es auch, das Graphikfenster mir Hilfe des Befehls **subplot(m,n,p)** in mehrereTeilfenster zu unterteilen. Das erste Argument (m) gibt die Anzahl der Zeilen und das zweite (n) die Spaltenanzahl an. In dem letzten Argument wird die jeweilige Position angewählt. Gezählt wird zeilenweise von links oben nach rechts unten.

```
>>x=linspace(-pi,pi,30); y = sin(x); z = cos(x);
subplot(2,1,1); plot(x,y);
subplot(2,1,2); plot(x,z);
```

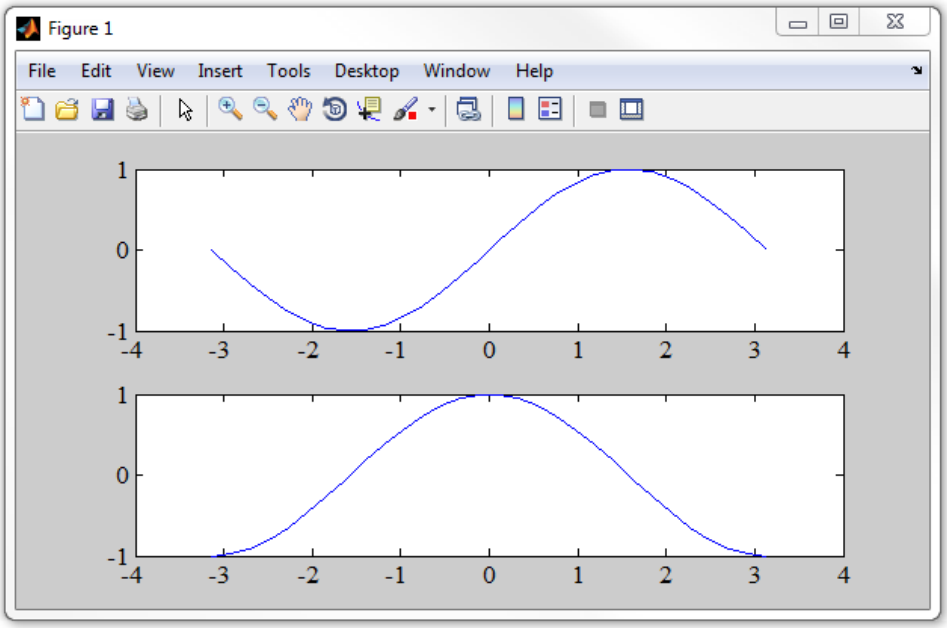

Abbildung 3-10 Zwei Graphen in einem Fenster mit dem Befehl subplot()

3.6 Graphik in Polarkoordinaten

Der Befehl **polar(phi,r)** erzeugt eine Graphik in Polarkoordinaten. Das Argument **phi** ist der Winkel in Radiant,**r** der Radius. Hier sind zum Beispiel die Anweisungen um die Kurver = sin(2*phi)cos(2*phi) zu zeichnen:

*>>phi = (-1:0.01:1)*pi; r = sin(2*phi).*cos(2*phi);*
>>polar(phi,r)

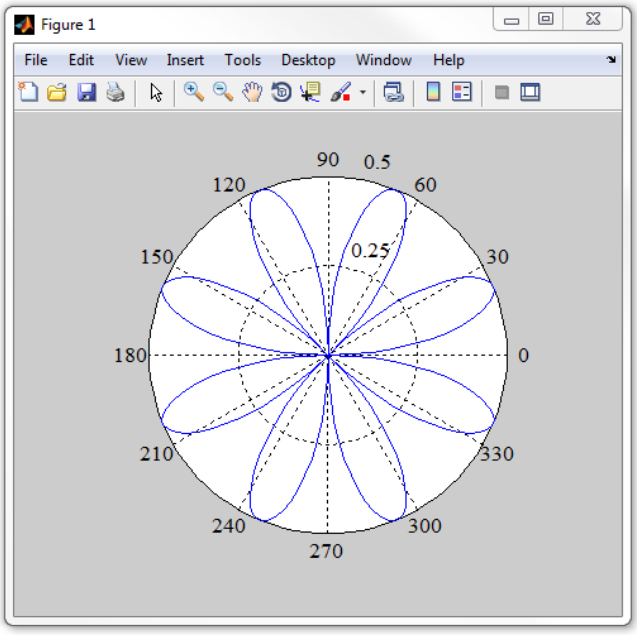

Abbildung 3-11 Darstellung in Polarkoordinaten

3.7 Funktion zeichnen, ohne Datenpunkte anzugeben

Das ist mit der Anweisung **fplot**möglich. Mit ihr kann man Funktionen zeichnen, ohne vorherei-nen Vektor x definieren zu müssen. MATLAB wählt die Stellen, an denen die Funktionswertebe-rechnet werden, automatisch.

>>fplot('sin(x)./x', [-20,20,-0.4,1.2])

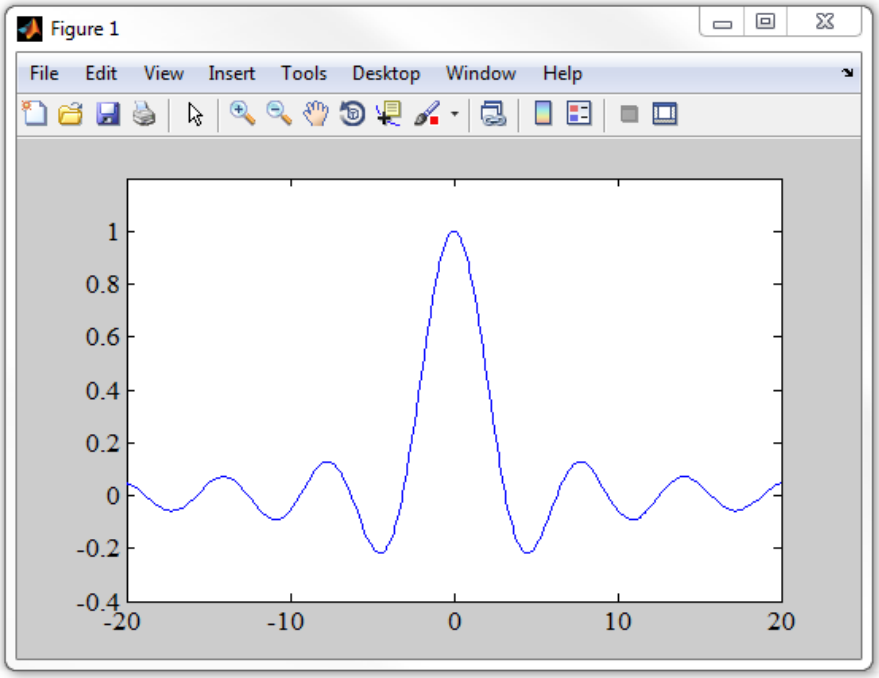

Abbildung 3-12 Funktionsplot mit fplot()

Die zu zeichnende Funktion wird zwischen Anführungszeichen eingegeben, danach der ge-wünschtex-Bereich (hier: -20 ≤x ≤20) und der gewünschte y-Bereich (hier: -0.4 ≤y ≤1.2).Der y-Bereich kann auch weggelassen werden, dann wählt MATLAB ihn automatisch.

Wichtig: x und y sind intern nach wie vor Vektoren, daher müssen elementweise Operationen verwendet werden (also zum Beispiel . / anstelle von /)!

Tipp: Um eine Graphik in WORD zu übertragen muss man EDIT/COPY FIGURE im Zeichen-fenster anwählen und danach die Graphik mit Einfügen in den WORD-File übertragen. Wenn gewünscht können auch Optionen durch EDIT/COPY OPTIONS gesetzt werden.

3.8 Speichern in und lesen aus Dateien

Damit die meist großen Datenmengen in MATLAB auch verarbeitet werden können, müssen diese zuvor eingelesen werden. Die gängigsten Datenformate sind Exceltabellen und CSV-Dateien (CommaSeperated Value Files). Diese können mit den Befehlen **xlsread** und **csvread** gelesen werden. Befinden sich die Dateien in dem Arbeitsverzeichnis, reicht es den Dateinamen als Argument anzugeben, andernfalls muss der vollständige Dateipfad übergeben werden. Beispiel:

>>Data = xlsread('c:\..\..\..\Tabelle')

In diesem Beispiel werden alle Daten vom ersten Worksheet aus der Datei „Tabelle.xls“ in die Variable „Data“ geschrieben, die in den meisten Fällen eine Matrix ist. Matlab ignoriert beim Auslesen von Exceldateien die erste Zeile / Spalte, sofern hier eine Beschriftung für die Tabelle steht. Fakultativ kann aber auch ausgewählt werden aus welchem Worksheet und aus welchen Zellen die Daten geladen werden sollen (**xlsread(File, Sheet, Cell Range)**):

>>Data = xlsread('Tabelle', 'Sheet6', 'B1:H4')

Sollte ein Feld in der entsprechenden Tabellen leer sein oder Text enthalten fügt Matlab an dieser Stelle ein „NaN“ ein. Auch die neuen Excel-Formate können mit der MATLAB Version R2010 und höher eingelesen werden. Dazu muss jedoch die Dateiendung mit angegeben werden:

>>Data = xlsread('c:\..\..\..\Tabelle.xlsx')

Der Befehl **csvread** funktioniert äquivalent. Mehr dazu findet sich mit **helpcsvread**.

Mit den Befehlen **xlswrite**und**csvwrite**können Ergebnisse aus einer Matrixin Excel- und CSV-Dateienexportiert werden. Dabei wirddie entsprechende Datei im aktuellen Arbeitsverzeichnis erzeugt, sofern nicht schon vorhanden.Für die Variable M, die exportiert wird, gilt folgende Syntax:

*>>xlswrite('Dateiname', **M**, Tabellenblatt, 'Bereich im Tabellenblatt')*

und

*>>csvwrite('Dateiname', **M**)*

Soll beispielsweise der Inhalt der Matrix „MatrixA“ in das Tabllenblatt „Tabellenblatt1“ in der Exceltabelle„Tab.xls“ im Bereich D2 bis H7 hinterlegt werden, lautet der entsprechende Befehl:

>>xlswrite('Tab', MatrixA, Tabellenblatt1, 'D2:H7')

In diesem Fall müssen die Dimensionen von MatrixA denen des angegebenen Bereichs entsprechen(D2 bis H7). Sollten die genauen Dimensionen nicht bekannt sein, kann auch lediglichdie obere linke Ecke, hier D2, angegeben werden.