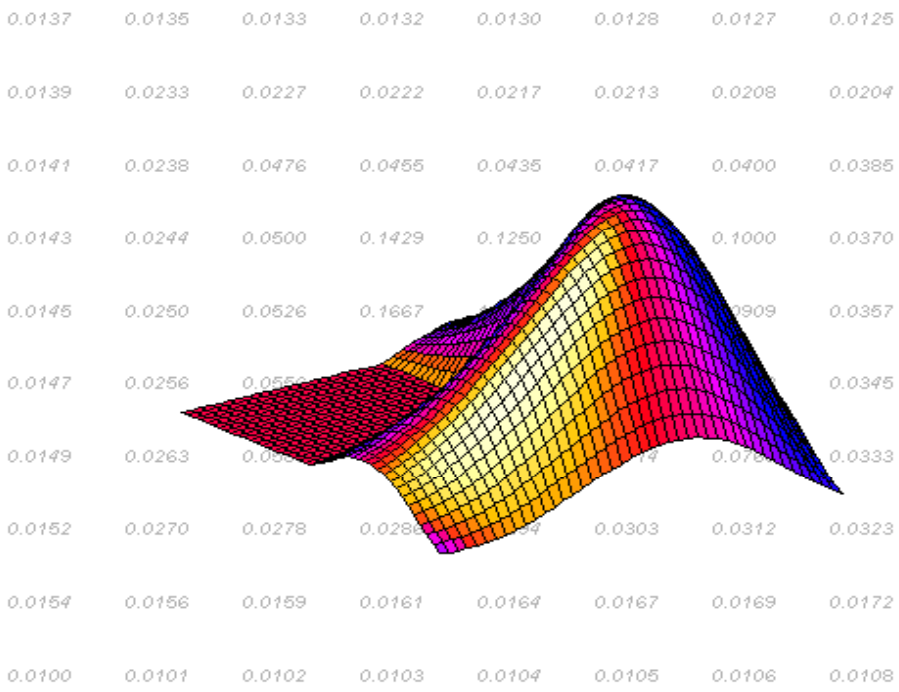
SCILAB - Eine Einführung

Basierend auf einem MATLAB-Skript von Susanne Teschl

An SCILAB angepasst von Sabine Beil und Hannes Grimm-Strele



Dieses Skript beinhaltet eine Einführung in SCILAB (Version 5.0.3). Es basiert weitestgehend auf einem MATLAB-Skript von Susanne Teschl, das auf ihrer Homepage

http://www.esi.ac.at/~susanne/

unter folgendem Link heruntergeladen werden kann:

http://www.esi.ac.at/~susanne/MatlabSkriptum.pdf

Wir danken Susanne Teschl, dass sie uns ihr Skript zur Verfügung gestellt hat!

Aktuelle Informationen zu MATLAB finden Sie unter anderem auf der offiziellen Webpage

http://www.mathworks.com

SCILAB kann unter

http://www.scilab.org/

kostenlos für gängige Betriebssysteme heruntergeladen werden.

Vorwort zum MATLAB-Skript:

Dieses Skriptum soll Ihnen einen schnellen Einstieg in MATLAB (Versionen 5 und 6) ermöglichen. Es ist als kurze Einführung gedacht, die an Hand von Beispielen die Arbeitsweise von MATLAB demonstriert. Das Skriptum enthält zahlreiche Übungen mit Lösungen und kann daher auch zum Selbststudium verwendet werden.

Inhaltsverzeichnis

0.1. Was ist MATLAB? 3
0.2. Was ist SCILAB? 4
1. Erste Schritte 5
2. Vektoren und Matrizen – die Grundbausteine von MATLAB 11
3. Grafik 20

Wien, März 2009

Sabine Beil, Hannes Grimm-Strele
beil.s@gmx.at, hannes.grimm-strele@univie.ac.at

0.1. Was ist MATLAB?

Kurz gesagt: MATLAB ist ein Softwarepaket für **numerische Berechnungen** und für die **Visualisierung von Daten** 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 ("**Mat**rix **Lab**oratory") 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 werden mehrere Anweisungen als 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.

Darüber hinaus können auch **C- und Fortran-Programme** in Form von sog. *mex-Files* von MATLAB exekutiert werden.

Für spezielle Anwendungen gibt es ***Toolboxes***, das sind Bibliotheken von m-Files zu bestimmten Aufgabenbereichen. So sind zum Beispiel elementare Befehle zum symbolischen Rechnen in der *Symbolic Math Toolbox* enthalten.

Die **Studentenversion** (*Student Edition*) von MATLAB unterscheidet sich in mehreren Punkten von der professionellen Version:
- sie ist preisgünstiger,
- sie läßt nur Matrizen mit beschränkter Größe,
- die Studentenversion enthält standardmäßig die *Symbolic Math Toolbox* und die *Signals and Systems Toolbox,*
- sie kann keine mex-Files linken.

Die Studentenversion kann auf die professionelle Version ergänzt werden.

Achtung: Die *Symbolic Math Toolbox* und die *Signals and Systems Toolbox* sind Studentenversionen von kommerziell erhältlichen Toolboxes. Sie sind **nur mit der Studentenversion verwendbar**.

Berechnungen mit MATLAB können mithilfe eines MATLAB Notebooks (***M-book***) übersichtlich dokumentiert werden. Ein M-book ist ein Word-Dokument, das neben Text auch MATLAB-Anweisungen und deren Output enthält.

0.2. Was ist SCILAB?

SCILAB ist eine seit 1994 existente, in Frankreich entwickelte Open Source-Software. Es ähnelt MATLAB in vielen Bereichen und wird sehr ähnlich bedient. Mit Hilfe von im Internet erhältlichen Konvertern (oder einer in SCILAB eingebauten Option) kann MATLAB-Code in SCILAB-Code und umgekehrt umgewandelt werden.

Ziel dieses Skriptums ist es, eine Einführung in die Bedienung von SCILAB zu geben und dabei insbesondere die Unterschiede zu MATLAB herauszuarbeiten.

Für weitere Informationen zu SCILAB sei auf die Projekthomepage verwiesen:

http://www.scilab.org/

Alle im Skript wiedergegebenen SCILAB-Outputs beziehen sich auf SCILAB Version 5.0.3 bzw. 5.1 unter Windows.

Unter

http://www.scilab.org/product/dic-mat-sci/

findet sich eine Liste aller MATLAB- und SCILAB-Befehle mitsamt den Unterschieden.

1. Erste Schritte

Einfache Rechenoperationen

Nach dem Start von SCILAB öffnet sich die Konsole (*console)* und es erscheint das *Prompt* `-->`. In die Konsole werden SCILAB-Anweisungen eingegeben und numerische Ergebnisse ausgegeben. Zur Darstellung von Grafiken wird ein eigenes Grafikfenster geöffnet.

Alle Anweisungen werden nach dem Prompt eingegeben und **mit ↵ *(Return)* bestätigt**. SCILAB nennt das Ergebnis **`ans`** (kurz für *answer*):

```
12/3 +7*5 -1
```

```
ans =
    38.
```

Unterschied zu MATLAB: Das Ausgabeformat für numerische Ergebnisse von MATLAB und SCILAB ist unterschiedlich. Dazu später mehr.

Addition (+), Subtraktion (-), Multiplikation (*) und Division (/) werden wie gewohnt bezeichnet und verwendet.

Unterschied zu MATLAB: Leerzeilen in der Ausgabe können nicht vermieden werden.

Braucht man einen Ausdruck öfters, so kann man ihn als Variable definieren. SCILAB speichert den Wert der Variablen automatisch im sogenannten ***Workspace***:

```
a = 48/3 - 3^2
```

```
a =
     7.
```

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 (der Rest wird automatisch abgeschnitten).

```
Variable_1 = 4*2
```

```
Variable_1 =
     8.
```

Ein **Strichpunkt** (*semi-colon)* am Ende der Eingabezeile bewirkt, dass SCILAB das Ergebnis zwar auswertet, aber nicht ausgibt:

```
A = 3*a^2 - Variable_1;
```

SCILAB unterscheidet zwischen **Groß- und Kleinbuchstaben** *(upper case* und *lower case letters)*. Daher sind `a` und `A` verschiedene Variablen (man sagt, die Variablennamen sind *case sensitive*).

Beachten Sie, dass Potenzieren (z.B. `a^2`) vor einer Multiplikation oder Division ausgewertet wird. Danach kommen Addition oder Subtraktion. Durch Klammersetzung kann man diese Reihenfolge aber ändern.

Sie können auch mehrere Anweisungen in eine Zeile eingeben: sind sie durch einen Beistrich (*comma*) getrennt, so folgt eine Ausgabe, werden sie durch einen Strichpunkt getrennt, so folgt keine Ausgabe:

```
b = (3+5)*6; c = (b/3)^2, d = A/c^2
```

```
c =
   256.
d =
    0.0021210
```

Das Komma einer Dezimalzahl wird, wie im Englischen allgemein üblich, als Punkt (*decimal point*) geschrieben.

Wichtig: Bei der Multiplikation darf das **Multiplikationszeichen `(*)` nicht weggelassen** werden:

```
3a
    !--error 276
Missing operator, comma, or semicolon.

3*a
ans =
    21.
```

Übung 1:

(a) Sie geben nacheinander folgende Anweisungen ein. Was wird SCILAB ausgeben?

```
u = 2,v = 5;
(u+6)/4
y = x+1
y = 3u
```

(b) Welche der folgenden Variablennamen sind nicht zulässig?

```
anzahl, Summe_a+b, 5_Tageskarte, dauer_phase3
```

Mit dem Befehl **`who`** kann man herausfinden, welche Variablen momentan im Workspace gespeichert sind.

Unterschied zu MATLAB: `who` liefert nicht nur die vom Benutzer eingespeicherten Variablen, sondern auch alle (lokalen und globalen) Systemvariablen.

Unterschied zu MATLAB: Konstanten wie π werden mit vorhergehendem Prozentzeichen eingegeben, z.B. **`%pi`**.

Vom Benutzer eingespeicherte Variable werden wieder **gelöscht** mit der Anweisung

```
clear a            % löscht die Variable a
```

oder

```
clear              % löscht alle Variablen im Workspace
```

Zwei Schrägstriche (`//`) nach einem eingegebenen Befehl bedeuten, dass ein **Kommentar folgt**. SCILAB ignoriert alles, was in der Zeile nach den Schrägstrichen steht.

Unterschied zu MATLAB: In MATLAB werden statt der Schrägstriche Prozentzeichen verwendet.

Nach Auswertung des `clear`-Befehls sind keine Variablen mehr im Workspace gespeichert:

help oder **HELP-Menü**

Der Befehl **`help`** ist sehr hilfreich, wenn man Näheres zu einer SCILAB-Anweisung wissen möchte. Wir erhalten zum Beispiel weitere Informationen zum Befehl **`who`** (und zu verwandten Befehlen) mit

```
help who
```

Im sich öffnenden Fenster erscheint eine kurze Beschreibung des Befehls sowie der Möglichkeiten des Aufrufes.

Unterschied zu MATLAB: Die Hilfe von SCILAB ist deutlich weniger intuitiv und verständlich. Außerdem sind die Möglichkeiten, sie zu nutzen, verschieden.

Eine **Liste aller Hilfe-Themen** (*help topics*) erhält man über den HELP-Menüpunkt oder einfach durch Eingabe von

```
help
```

Im sich öffnenden ***Help Browser*** können Informationen zu allen Befehlen angezeigt werden. Links befindet sich ein Verzeichnisbaum, mit dem zwischen verschiedenen Handbüchern gewechselt werden kann. Öffnet man das oberste Handbuch ***Scilab***, wird eine Liste der wichtigsten in SCILAB verfügbaren Befehle angezeigt. Durch einfachen Klick auf einen Befehl wird rechts der entsprechende Eintrag angezeigt.

Übung 2:
Mit welchem Befehl wird das Commandfenster gelöscht (dh., der Bildschirm wird gelöscht, es bleiben aber trotzdem noch alle Variablenwerte gespeichert) ?

Übung 3:
Sie möchten die Variablen `a=2` und `b=1` zugleich aus dem Workspace löschen. Bisher kennen Sie die Möglichkeit:

```
clear a; clear b
```

Sie versuchen nun, sich Tipparbeit zu sparen, und geben ein:

```
clear a,b
```

Warum mißversteht MATLAB Ihren Befehl? Wie lautet die richtige Verwendung von `clear`, um mehrere Variable auf einmal zu löschen?

Eingebaute Funktionen

Es gibt - wie auch in jedem Taschenrechner - in SCILAB bereits **"eingebaute" Funktionen**. Ein Beispiel ist die Wurzelfunktion (*square root*)

```
a = sqrt(2)/2
```

```
a =
    0.7071068
```

Hier haben wir `a` $= \frac{\sqrt{2}}{2}$ berechnet. Nähere Informationen zur Funktion **`sqrt`** erhält man mit

```
help sqrt
```

Nun öffnet sich im ***Help Browser*** der entsprechende Eintrag.

Hinweis: Beachten Sie, dass die Anweisung `y = sqrt(x)` nur dann von SCILAB ausgewertet werden kann, wenn `x` einen Zahlenwert hat (oder allgemein, wenn `x` eine numerische Matrix ist)! MATLAB ist vor allem auf die *Verarbeitung von numerischen Daten* ausgerichtet. Ist `x` noch nicht definiert, so erhält man eine Fehlermeldung:

```
y = sqrt(x)

         !--error 4
Undefined variable: x
```

Tipp: Die **vorangehende Eingabe** bekommt man mit der Cursor-Taste ↑. Das ist zum Beispiel praktisch, wenn man einen Tippfehler ausbessern möchte. Wiederholtes Tippen von ↑ oder ↓ ermöglicht ein *Scrollen* in den vorangehenden Eingaben.

Übung 4:
Berechnen Sie den natürlichen Logarithmus von 1.36.

Übung 5:
Berechnen Sie `cos(π)` und `cos(π/2)`. Das Argument der Kosinusfunktion wird von SCILAB immer im Bogenmaß interpretiert. Die Konstante π ist in SCILAB bereits eingebaut und wird mit **`%pi`** bezeichnet.

Darstellung von numerischen Ergebnissen

Unterschied zu MATLAB: Die Darstellung von numerischen Ergebnissen ist komplett unterschiedlich.

SCILAB gibt standardmäßig ein ganzzahliges Ergebnis als Zahl ohne Nachkommastellen, aber mit Dezimalpunkt und ein nicht-ganzzahliges Ergebnis als Dezimalzahl mit bis zu 7 Nachkommastellen aus:

```
9.0/3.0
ans =
     3.
13/5
ans =
    2.6
13/6
ans =
    2.1666667
```

Der Dezimalpunkt auch bei ganzzahligen Ergebnissen ist als Hinweis darauf zu interpretieren, dass SCILAB stets mit Maschinenzahlen rechnet und diese um die Maschinengenauigkeit (s.u.) von der exakten Zahl abweichen können. Bei nicht ganzzahligen Ergebnissen werden standardmäßig bis zu 7 von 0 verschiedene Nachkommastellen ausgegeben, genauer gesagt die führenden acht Stellen.

Man kann das **Ausgabeformat** numerischer Resultate **ändern**. Der dazu benötigte Befehl heißt **`format`**. Er kann auf zwei Arten aufgerufen werden: **`format('v',20)`** stellt die Ausgabe dauerhaft auf 20 Stellen um (ebenso **`format(20)`**):

```
format('v',20); pi
```

```
ans =
   3.14159265358979312
```

Um das Ergebnis in Gleitkommadarstellung auszugeben, gibt man **`format('e',20)`** ein, wobei 20 durch eine beliebige andere Zahl ersetzt werden kann.

```
format('e',20);1/1000000
ans  =
    1.00000000000000D-06
```

Eine Beschreibung der verschiedenen numerischen Formate erhalten Sie zum Beispiel mit `help format`. Der Standardwert ist **`format('v',10)`**.

Wichtig: Bei Änderung des Formats ändert sich **nur die Ausgabe**, nicht aber die interne Darstellung (und damit die Genauigkeit) der Zahlen.

(Relative) Genauigkeit der Darstellung numerischer Daten:

```
%eps
```

```
%eps =
  2.220D-16
```

`%eps` gibt den Abstand von 1 zur nächst größeren (Gleitkomma-)Zahl an, die darstellbar ist.

Übung 6:
Berechenen Sie den natürlichen Logarithmus von 1.45 auf 14 Nachkommastellen genau.

Übung 7:
Finden Sie nur durch Wahl eines geeigneten Ausgabeformats eine möglichst einfache Bruchdarstellung von 0.784544.

Komplexe Zahlen

Komplexe Zahlen können wie reelle Zahlen eingegeben werden:

```
z = 3+4*%i    //Eingabe der komplexen Zahlen z
```

```
z =
   3. + 4.*%i
```

Unterschied zu **MATLAB**: Für den Imaginärteil kann statt i nicht j geschrieben werden. Weiters muss man % vor i schreiben und kann das Multiplikationszeichen nicht weglassen.

Addition, Subtraktion, Multiplikation, Division und Funktionswerte werden wie bei den reellen Zahlen gebildet:

```
z1 = 3+4*%i
z2 = 1-%i

z = z1+z2
z =
  4. + 3.i

sqrt(1+2*%i)          // Wurzel einer komplexen Zahl
ans =
   1.2720 + 0.7862i
```

Die **konjugiert komplexe Zahl zu `z`** erhalten wir mit

```
z = 1+3*%i; conj(z)
```

```
ans =
   1. - 3.i
```

Tipp: Auch durch Anfügen eines Apostrophs `'` wird eine Zahl komplex konjugiert. Das ist ein Spezialfall - im allgemeinen bewirkt `A'`, dass die Matrix `A` transponiert und komplex konjugiert wird.

```
z'
```

```
ans =
   1. - 3.i
```

Absolutbetrag und **Phasenwinkel** berechnet man mit **`abs(z)`** und **`angle(z)`**:

```
abs(z)
```

```
ans =
    3.1623
```

Der Befehl `angle(z)` funktioniert in Scilab nicht. Man muss stattdessen den Arcustangens verwenden.

In SCILAB kann man mit dem Befehl **`polar`** die Polardarstellung einer komplexen Zahl berechnen. Die Ausgabe des Winkels ist dabei stets im Bogenmaß.

```
[r,phi]=polar(1+%i)
```

```
 phi  =
    0.7853982 + 2.498D-16i
 r  =
    1.4142136
```

Man beachte die numerische Ungenauigkeit bei der Angabe des Winkels (Fehler in der Größenordnung der Rechengenauigkeit).

Übung 8:
Gegeben ist `z = 4i/(1+i)`. Wie lautet die zu `z` konjugiert komplexe Zahl? Wie groß ist Absolutbetrag?

Übung 9:
Finden Sie heraus, wie man den Real- und Imaginärteil einer komplexen Zahl berechnet. Berechnen Sie dann den Real- und Imaginärteil von `z = (3+2i)/(1-i)`.

2. Vektoren und Matrizen – die Grundbausteine von MATLAB

Vektoren

Nehmen wir an, wir möchten die Werte von $\sqrt{x}$ berechnen, sagen wir, für die Werte `x = 0,2,4,6,8,10,12`. Hier kommt uns MATLAB/ SCILAB sehr entgegen. Wir fassen die `x`-Werte zu einem **Vektor** (*list* oder auch *array*) zusammen:

```
x =[0 2 4 6 8 10 12]
x =
     0.     2.     4.     6.     8.    10.    12.
```

Vektoren werden immer in **eckigen Klammern** (*brackets*) eingegeben. Die einzelnen Elemente des Vektors *(elements)* sind durch **Beistriche oder Leerzeichen** *(spaces)* zu trennen.
Die Funktionswerte für die einzelnen Werte von `x` müssen nun nicht einzeln berechnet werden, sondern sie können alle auf einmal mit

```
y = sqrt(x)
y =
         0.    1.4142    2.    2.4495    2.8284    3.1623    3.4641
```

erhalten werden! Der Befehl **`sqrt` angewendet auf** einen Vektor **`x`** gibt nämlich die Anweisung: Nimm **von jedem Element von `x` die Wurzel** und schreibe das Ergebnis als entsprechendes Element eines neuen Vektors (hier `y` genannt).

Diese unkomplizierte Art, einen Befehl (hier das Wurzelziehen) elementweise anzuwenden, ist eine große Stärke von MATLAB/ SCILAB. 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 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 = 29:-2:0
u =
  Column 1 to 8
    29.    27.    25.    23.    21.    19.    17.    15.
  Column 9 to 15
    13.    11.     9.     7.     5.     3.     1.
```

Dieser Vektor hat 15 Elemente (Spalten - *columns*). 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
```

Einzelne Werte eines Vektors können einfach herausgegriffen werden. So erhalten wir z.B. das 4. Element von `z` mit

```
z(4)
```

```
ans =
    10
```

Achtung: Enthält der Vektor `y` die Funktionswerte von $\sqrt{x}$, so ist dementsprechend **`y(1)`** in MATLAB/ SCILAB **das erste Element des Vektors `y`**

```
x = 0:2:12; y = sqrt(x); y(1)
```

```
ans =
     0
```

und nicht der Wert von $\sqrt{x}$ an der Stelle `x=1`!

Übung 1:
Erzeugen Sie einen Vektor `y`, der die Funktionswerte des natürlichen Logarithmus an den Stellen `x = 1,3,5,7,9` enthält. Was gibt MATLAB/ SCILAB aus, wenn Sie `y(1)` eingeben?

Übung 2:
Geben Sie die Vektoren `a` und `b` mit den Elementen `-10,-8,-6,...,6,8,10` bzw. `10,9,8,...0` mit kurzen Anweisungen ein.

Elementweise Operationen

Vektoraddition und die Multiplikation eines Vektors mit einem Skalar sind wie üblich definiert, nämlich **elementweise**:

```
w = [1 2 3]*2                // Vektor * Skalar
```

```
w =
     2.     4.     6.
```

Hier wird jedes Element des Vektors `[1 2 3]` mit `2` multipliziert und der so entstandene Vektor wird `w` genannt.

```
[2  -1  9] + [1  3  6]          // Vektor + Vektor
```

```
ans =
     3.     2.    15.
```

Zu jedem Element von `[2 -1 9]` wird das entspechende Element von `[1 3 6]` addiert. Die Differenz zweier Vektoren wird analog gebildet.

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

```
w + [1 4 6 7]
```

```
              !--error 8
Inconsistent addition.
```

Die Länge eines Vektors (dh. die Anzahl seiner Elemente) erhalten wir mit

```
length(w)
```

```
ans =
     3.
```

Neben diesen aus der Mathematik bekannten Vektoroperationen sind in MATLAB/ SCILAB **weitere elementweise Vektoroperationen** definiert:

```
[2  5  7] + 3                // Vektor + Zahl
```

```
ans =
     5.     8.     10.
```

Zu jedem Element von `[2 5 7]` wird `3` addiert.

Neu ist auch die in MATLAB/ SCILAB definierte **elementweise Multiplikation.** 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 das erste Element von `a` mit dem ersten Element von `b` multiplizieren, das zweite Element von `a` mit dem zweiten von `b`... Der MATLAB/SCILAB-Befehl für diese Operation ist allerdings nicht, wie vielleicht erwartet, `a*b`, sondern

```
a.*b          // elementweise Multiplikation
```

```
ans =
     1.     6.     15.    28.    45.
```

Das Ergebnis ist wieder ein Vektor der Länge 5.

Achtung: Der **Punkt vor dem Stern** *(asterisk)* ist **notwendig**! Die Anweisung **`a*b`** bedeutet die in der Mathematik übliche **Matrixmultiplikation**. Sie 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 10
Inconsistent multiplication.
```

Die **elementweise Division** in MATLAB ist ganz analog definiert:

```
a./b
```

```
ans =
    1.    1.5    1.6667    1.75    1.8
```

Jedes Element von `a` wird durch das entsprechende Element von `b` dividiert. Auch ein **elementweises Potenzieren** ist möglich:

```
a.^2          // elementweises Potenzieren
```

```
ans =
     1.     9.     25.    49.    81.
```

Jedes Element von `a` wird quadriert.

Polynome

Polynome sind besonders einfache mathematische Funktionen, die vielfach Anwendung finden. Ein Polynom, etwa $y = -x^3 + 2 \cdot x^2 + 5$, kann über Punktoperationen für einen bestimmten Vektor x errechnet werden. Einfacher kann ein Polynom in SCILAB durch seine Koeffizienten angegeben werden.
Im Folgenden soll das Polynom $y = -x^3 + 3 \cdot x^2 + 1$ für die Werte x = -1, 0, 1, 2, 3, 4 und 5 berechnet werden. x wird als Vektor definiert; dazu werden die Polynomkoeffizienten –1, 3, 0 und 1 ebenfalls als Vektor, etwa `koeff`, zusammengefasst.

Unterschied zu MATLAB: In MATLAB kann ein Polynom an mehreren Stellen gleichzeitig mit Hilfe des **`polyval`** Befehls ausgewertet werden. In SCILAB ist dieser Befehl nicht verfügbar – stattdessen:

Um das Polynom in SCILAB zu definieren, benutzt man den **`poly`** Befehl, der in zwei Modi aufgerufen werden kann. Zuerst definiert man den Vektor x der Stellen, an denen das Polynom ausgewertet werden soll, und den Koeffizientenvektor, wobei man mit dem Koeffizienten der niedrigsten Potenz beginnt:

```
x = -1:5;
koeff = [1 0 3 -1];  // Koeffizient der niedrigsten Potenz zuerst
```

Nun definiert man das Polynom mittels des folgenden Aufrufes:

```
p=poly(koeff,"x","coeff")
p  =
    1 + 3x2 - x3
```

Im zweiten Feld kann man den Namen der Variablen ändern und im dritten Feld den Modus auswählen. Im Modus `coeff` wird der an erster Position stehende Vektor als Koeffizientenvektor des Polynoms benutzt, im Modus `roots` wird ein Polynom ausgegeben, dass die Werte als Nullstellen hat.
Um jetzt die Werte des Polynoms an den im Vektor x gespeicherten Stellen auszurechen, benutzt man den Befehl **`horner`**:

```
horner(p,x)
 ans  =
    5.    1.    3.    5.    1.  - 15.  - 49.
```

Übung 4:
`a = [1 4 6]` und `b = [-1 2 1]`. Was gibt SCILAB aus, wenn Sie eingeben: `a+b, a*2, a/2, a+3, a*b, a.*b, a./b`? Geben Sie die Antwort, bevor Sie mit SCILAB rechnen!

Übung 5:
Sie möchten einen Vektor `x` mit den Elementen `0, 0.5`π`, ...,2`π (d.h, Schrittweite `0.5`π) definieren. Dazu können Sie zum Beispiel die elementweise Multiplikation mit π

```
x = (0:0.5:2)*pi
```

verwenden. Warum sind hier die Klammern notwendig? Wie würde SCILAB die Anweisung

```
x = 0:0.5:2*pi
```

interpretieren?

Übung 6:

a) Berechnen Sie die Werte der Polynomfunktion $y = -0{,}01 \cdot x^3 + 0{,}02 \cdot x^2 + 1{,}2$ für $x = 0, 1, 2, 3, 4, 5$ und 6

b) Die Sinusfunktion kann näherungsweise in der Form $\sin x \approx x - \frac{x^3}{6} + \frac{x^5}{120}$ (Winkel x im Bogenmaß) geschrieben. Vergleichen Sie den genauen Sinuswert und den Näherungswert für $x = 0°, 10°, 20°, ..., 50°$.

Matrizen

SCILAB: Hier funktioniert alles ziemlich gleich.

Matrizen sind wertvolle Hilfsmittel bei der Verarbeitung von Daten. Betrachten wir zum Beispiel das 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 Spaltenvektoren `x` und `b`. Matrizen werden in MATLAB zwischen eckigen Klammern eingegeben. Das Ende **einer Zeile** wird **durch einen Strichpunkt gekennzeichnet:**

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

```
A =
     3.     4.    -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 und die Elemente werden komplex konjugiert.

Geben wir den Vektor `b` in der gewohnten Form ein,

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

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

```
b = b'
```

```
b =
     4.
    -1.
     3.
```

(Der Ausdruck links von einem Gleichheitszeichen wird immer durch den Ausdruck rechts definiert.)

Tipp: Einen Spaltenvektor erhalten Sie auch mit **`b(:)`** .

Wie bei Vektoren können wir auch ein bestimmtes 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 ganze zweite 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 "*alle Spalten*").
Entsprechend erhalten wir zum Beispiel die ganze erste Spalte von `C` durch

```
C(:,1)
```

```
ans =
     3.
     4.
    -2.
```

Hier steht der Doppelpunkt für "*alle Zeilen*".
Möchten wir **ein bestimmtes Element der Matrix C ändern**, zum Beispiel `C(3,2)` auf den Wert `7`, so geben wir ein

```
C(3,2) = 7
```

```
C =
     3.    -1.     2.
     4.     2.     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**, indem wir eingeben:

```
C(:,1) = []
```

```
C =
    -1.     2.
     2.     0.
     7.    -5.
```

Die **Dimension einer Matrix** (d.h., ihre Zeilen- und Spaltenanzahl), gibt der Befehl

```
size(C)
```

```
ans =
     3.      2.
```

Als Ergebnis erhält man den Vektor [*Zeilenanzahl*, *Spaltenanzahl*], hier `[3 2]`.

Tipp: DieAnweisung **`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 z.B. erzeugt durch

```
zeros(3,3)            // 3×3 Nullmatrix
```

```
ans =
     0.      0.      0.
     0.      0.      0.
     0.      0.      0.
```

```
eye(3,3)              // 3×3 Einheitsmatrix (3-by-3 identity matrix )
```

```
ans =
     1.      0.      0.
     0.      1.      0.
     0.      0.      1.
```

```
ones(3,3)              // 3×3 Einsmatrix
```

```
ans =
     1.      1.      1.
     1.      1.      1.
     1.      1.      1.
```

Im **Gegensatz zu Matlab** ist in Scilab immer eine Eingabe eye(m,n), zeros(m,n), ones(m,n) für eine mxn-Matrix notwendig.

Übung 7:
`A = [1 3 5; 2 0 1; 2 4 6];` Was ist das Ergebnis der folgenden Anweisungen? `A', A(1,:), A(:,3), A(2,2), size(A).` Überlegen Sie, bevor Sie mit SCILAB rechnen.

Übung 8:
`B = rand(3,4)` erzeugt eine 3×4 Matrix mit **zufällig bestimmten Elementen**. Ersetzen Sie in der so erhaltenen Matrix `B` das erste.Element in der ersten Zeile durch `0` und streichen Sie die gesamte zweite Spalte von `B`.

Übung 9:
Erzeugen Sie eine 2×5 Matrix `A` mit Zufallselementen. Bilden Sie daraus
- eine Matrix `B`, die aus den ersten drei Spalten von `A` besteht
- eine Matrix `C`, die aus der zweiten und vierten Spalte von `A` besteht.

Finden Sie die dazu nötigen Anweisungen mit **`help colon`**.

Matrixoperationen

`A+B, A-B, A*B` bezeichnen die üblichen Matrizenoperationen.

Wichtig: Der **Stern `' * '` allein** bedeutet die übliche **Matrizenmultiplikation** (die nur definiert ist, wenn die Anzahl der Spalten von `A` gleich der Anzahl der Zeilen von `B` ist). Darüber hinaus ist, wie bei den Vektoren, auch die **elementweise Multiplikation** definiert, die mit dem Symbol **`'.* '`** 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 entsprechendem Element von `[3  6; 0 -1]` multipliziert.

Existiert die Inverse A^{-1}, so ist die Lösung x von A*x = b gegeben durch $x = A^{-1}*b$. A ist invertierbar genau dann, wenn die Determinate von A ungleich Null ist:

```
det(A)     // Determinante der Matrix A
```

```
ans =
    22.
```

Wir erhalten die Inverse A^{-1} mit **`inv(A)`**:

```
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 (*left division*)**. Der Name kommt daher, dass nun der **Nenner links** steht und der **Bruchstrich nach links geneigt** ist. Hier steht `A\` 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 ja kommutativ). Bei Matrizen aber ist `inv(A)*b` (Linksdivision) ungleich `b*inv(A)` (Rechtsdivision), der letzte Ausdruck ist nämlich **gar nicht definiert**. Daher erhalten wir eine Fehlermeldung, wenn wir eingeben:

```
b/A
```

```
!--error 266
A and B must have equal number of columns.
```

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 wird das Gaußsche Eliminationsverfahren verwendet). Daher erhält man so vor allem bei größeren Problemen **schneller eine Lösung**.
- Zweitens liefert die Eingabe von `x = A\b` auch dann eine Lösung, wenn das **Gleichungssystem nicht eindeutig lösbar** ist (und daher die Inverse A^{-1} nicht existiert). Im Fall eines **überbestimmten** Systems erhält man zum Beispiel eine Lösung, die **den quadratischen Fehler von A*x - b = 0 minimiert**.

Für nähere Informationen siehe Handbuch Teil 3 (*Reference*), Stichwort "*Arithmetic Operators*" oder z.B. **`help slash`**.

Eigenwerte/Eigenvektoren

Sei A eine nxn-Matrix. Der Befehl **spec(A)** gibt alle Eigenwerte an in der entsprechenden algebraischen Vielfachheit und der Befehl **[D,X]=bdiag(A)** gibt Matrizen X und D aus, sodass D=X^(-1)*A*X.

Übung 10:
`A = eye(3,3), B = [1 2 3; 4 5 6; 7 8 9]`. Was ist das Ergebnis von `A+B, A*2, A/2, B.^2, A.*B, A*B`? Überlegen Sie, bevor Sie mit SCILAB rechnen!

Übung 11:
Lösen Sie das Gleichungssystem

$$-x + 4y + 6z = 5$$

$$\begin{aligned} 3x \quad &- \quad z = 7 \\ 2x + 7y &- 3z = -1 \end{aligned}$$

Beispiel: der ε-Tensor

Der ε-Tensor in n Dimensionen hat n Indizes und ist durch folgende zwei Eigenschaften definiert:

- $\varepsilon_{1,2,\ldots,n}=1$
- Unter Vertauschung zweier Indizes ändert es das Vorzeichen: $\varepsilon_{\ldots,i,\ldots,j,\ldots} = -\varepsilon_{\ldots,j,\ldots,i,\ldots}$

Daraus kann man die folgende alternative Definition ableiten, die im Spezialfall n=3 wie folgt aussieht: $\varepsilon_{i,j,k}=-(j-i)*(k-j)*(i-k)$ (analog für höhere Dimensionen). Mit Hilfe dieser Definition kann man den ε-Tensor in drei Dimensionen wie folgt in SCILAB abspeichern:

```
eps=zeros(3,3,3);  // Initialisierung
for i=1:n, for j=1:n, for k=1:n eps(i,j,k)=-(i-j)*(k-j)*(i-
k)/2;end;end;end;
eps
 eps  =
(:,:,1)
    0.    0.    0.
    0.    0.    1.
    0.  - 1.    0.
(:,:,2)
    0.    0.  - 1.
    0.    0.    0.
    1.    0.    0.
(:,:,3)
    0.    1.    0.
  - 1.    0.    0.
    0.    0.    0.
```

Mit Hilfe des ε-Tensors kann man zum Beispiel das Kreuzprodukt zweier Vektoren berechnen, für das es in SCILAB keine integrierte Funktion gibt. Die 2-te Komponente des Kreuzproduktes zweier Vektoren ist dann:

```
a=[2 3 -2];b=[0 1 -3];
komp2=0;
for j=1:3,for k=1:3,komp2=komp2+eps(2,j,k)*a(j)*b(k);end;end;komp2
 komp2  =
    6.
```

Die hier benutzte `for`-Schleife werden wir später genauer erklären.

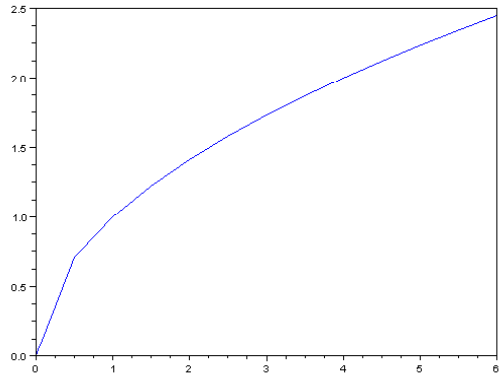
3. Grafik

`plot` veranschaulicht numerische Daten

In SCILAB kann man mit wenigen Befehlen sehr leicht Daten veranschaulichen. Die Anweisung **`plot(x,y)`** erzeugt ein **Grafikfenster**. (Achtung: es ist oft hinter den anderen geöffneten Fenstern versteckt!). `x` und `y` sind Vektoren, sie enthalten die x- und y-Koordinaten der zu zeichnenden Datenpunkte.

Wir wollen zum Beispiel die Wurzelfunktion im Intervall `[0,6]` zeichnen:

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

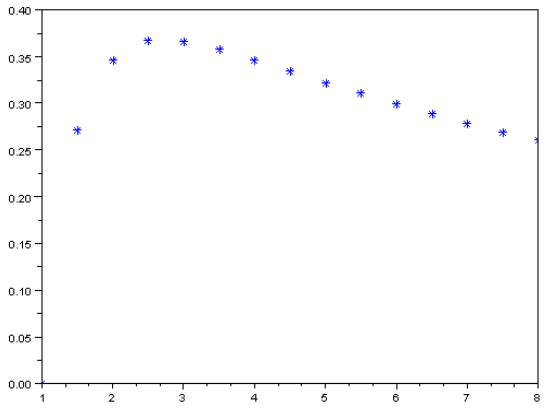


Es öffnet sich ein Graphikfenster und die Funktionswerte von $y = \sqrt{x}$ werden für die Werte `x = 0,0.5,1, ..., 5.5,6` gezeichnet. SCILAB wählt die Achsenbegrenzungen automatisch, beschriftet die Achsen und **verbindet die Datenpunkte durch gerade Linien**.

Übung 1: Zeichnen Sie die Gerade `y = ¾ x – 1` im Intervall `[0,4]`.

Das nächste Beispiel zeigt, wie praktisch die in SCILAB möglichen elementweisen Operationen 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,'*')
```



Der Stern `(*)` unter **einfachen Anführungszeichen** bewirkt, dass **nur die Datenpunkte gezeichnet** werden, und zwar in Form von Sternen (anstelle der Sterne sind zum Beispiel auch Kreise (o) oder

Kreuze (+) möglich, aber auch vieles anderes). Genauere Informationen zu den verschiedenen Darstellungsmöglichkeiten gibt es in der Hilfe unter **`help plot.`**
Achtung: Es darf nur das Anführungszeichen **'** oder " verwendet werden. Verwendet man ´ oder `, so erhält man eine Fehlermeldung.

Unterschied zu MATLAB: In MATLAB darf " nicht verwendet werden.

Übung 2:
Wie sieht der Graph von `y = 1-e^-2t` im Intervall `[0,3]` aus?

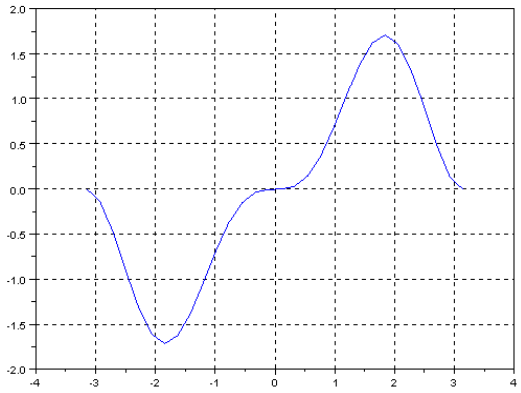
Übung 3:
Zeichnen Sie folgende Kurve in Parameterdarstellung: `x(t) = sin(2t), y(t) = cos(t),` $0 \le t \le 2\pi$, und als strichlierte Linie. Verändern Sie die Anzahl der gezeichneten Punkte und beobachten Sie die daraus resultierenden Veränderungen.

Möchte man die **Anzahl der gezeichneten Datenpunkte vorgeben**, so erzeugt man den Vektor `x` am besten mit **`linspace:`**

```
x = linspace(-%pi,%pi,30);
```

Hier besteht der Vektor `x` aus 30 Werten im gleichen Abstand, wobei der erste Wert $-\pi$ und der letzte Wert π ist. **Merkhilfe: `linspace` (*erster Wert, letzter Wert, Anzahl der Werte*) .** Fehlt die Angabe für die Anzahl der Werte, so wird dafür 100 genommen.

```
y = x.*sin(x).^2;
plot(x,y); xgrid
```



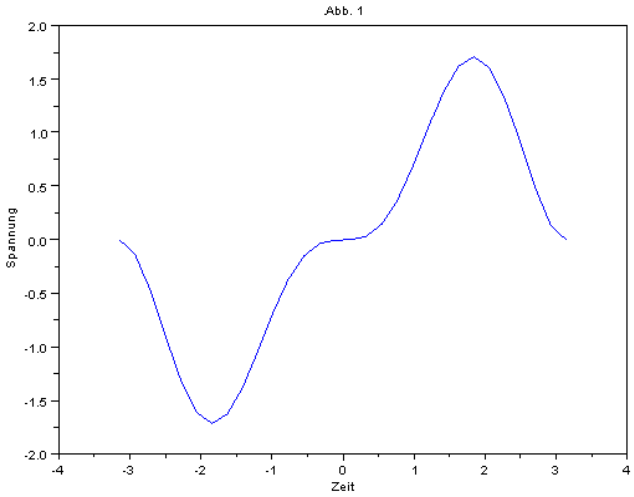
Die Anweisung **`xgrid`** erzeugt **Gitterlinien**.

Unterschied zu MATLAB: In MATLAB können Gitterlinien mit der Option **`grid`** erzeugt werden.

Wie kann ich eine Grafik beschriften?

Die Koordinatenachsen werden mit **`xlabel`** bzw. **`ylabel`** beschriftet. Die Anweisung **`title`** versieht die Graphik mit einer Überschrift.

```
plot(x,y);
xlabel('Zeit'), ylabel('Spannung')
title('Abb. 1')
```



Unterschied zu MATLAB: Dort kann man mittels `text(-0.8,0.5,'x*sin(x)^2')` eine Beschriftung in die Grafik setzen. Dann erscheint im Bild beginnend im Punkt mit den **Koordinaten** `(-0.8,0.5)` der Text `'x*sin(x)^2'`.

In der aktuellen Graphik wird die x-Achse mit `'Zeit'` beschriftet, die y-Achse mit `'Spannung'`. Die ganze Grafik wird mit `'Abb.1'` betitelt.

Achtung: Es ist wichtig, **zuerst** die Anweisung **`plot(x,y)`** einzugeben, und dann erst die Befehle zur Beschriftung der Graphik. Die Beschriftung wir nämlich immer in die **aktuelle Grafik** eingefügt.

Tipp: **Grafiknachbearbeitung** (am Besten ausprobieren!)
Im Zeichenfenster kann mit Hilfe des Menüpunktes EDIT Darstellung der Achsen und der Kurve/Fläche, Beschriftung, … geändert werden.

Tipp: Grafik in WORD übertragen
COPY TO CLIPBOARD im Menüpunkt FILE des Zeichenfenstesr anwählen und danach die Grafik mit Einfügen in den WORD-File übertragen.

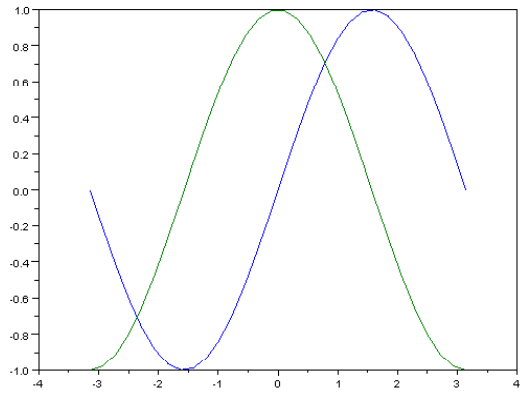
Übung 4:
Beschriften Sie den Graphen von $y = 1 - e^{-2t}$ im Intervall `[0,3]`.

Übung 5:
Beschriften Sie den Plot: $x(t) = \sin(2t),\ y(t) = \cos(t),\ 0 \le t \le 2\pi$.

Wie kann ich mehrere Funktionen gleichzeitig zeichnen?

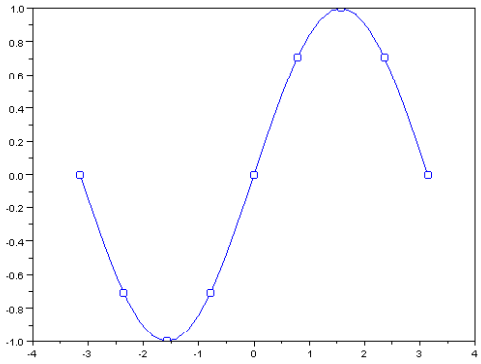
Die Anweisung **`plot(x,y,x,z)`** zeichnet die Funktionen `y(x)` und `z(x)` in das gleiche Koordinatensystem.

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



Die Sinus- und Kosinusgraphen werden automatisch in verschiedenen Farben in ein- und dasselbe Graphikfenster gezeichnet.
Gibt man nacheinander mehrere Plots in die Konsole ein, werden diese in das stets in das gleiche Fenster gezeichnet, allerdings immer in der gleichen Farbe. Möchte man eine neue Zeichnung erstellen, schließt man entweder das Fenster oder löscht alle bisherigen Zeichnungen mit dem Befehl **`clf`**.

```
x1=linspace(-%pi,%pi,30);
x2=-%pi:%pi/4:%pi;
y1=sin(x1); y2=sin(x2);
plot(x1,y1); plot(x2,y2,'o');
```

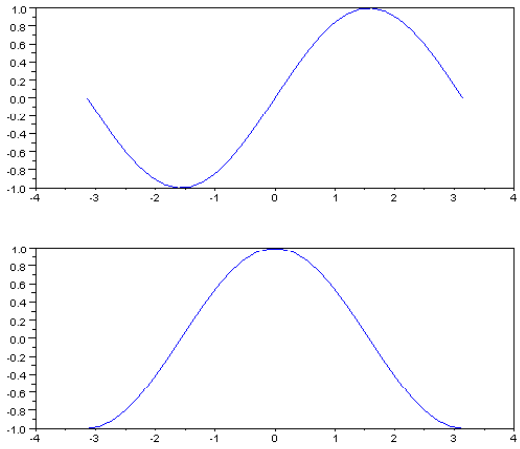


Unterschied zu MATLAB: Der Befehl **`hold`** ist somit in SCILAB nicht notwendig, um mehrere Grafiken in einem Fenster zu erstellen.

Möchte man die nächste Grafik in ein **weiteres Grafikfenster** gezeichnet haben, so muss man zuvor ein neues Grafikfenster mit dem Menüpunkt FILE/NEW.../FIGURE (im alten Grafikfenster) öffnen.

SCILAB ermöglicht es auch, das Graphikfenster mir Hilfe des Befehls **`subplot(m,n,p)`** in **mehrere Teilfenster** zu zerlegen.

```
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);
```



Die erste Zahl m sagt, dass in vertikaler Richtung m Grafikteilfenster, die zweite Zahl n, dass in horizontaler Richtung n Graphikteilfenster vorgesehen sind. Die Zahl p gibt an, in welches der m mal n Teilfenster die nächste Grafik gezeichnet werden soll. Gezählt wird zeilenweise von links oben nach rechts unten.

Übung 6:
Zeichnen Sie `y = x^2` und `z = 2 + |x|` im Intervall `[-3,3]` und lesen Sie aus der Graphik ab, für welche `x` die Beziehung `2 + |x| < x^2` gilt!
Tipp: Falls Sie `linspace` verwenden, wählen Sie eine ungerade Anzahl an Datenpunkten (warum?).

Übung 7:
Zeichnen Sie die Funktionen `y = x, w = x^2` und `z =` $\sqrt{x}$ in eine Graphik.

Wie erstellt man eine Grafik in Polarkoordinaten?

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

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

Unterschied zu MATLAB: Dort heißt der entsprechende Befehl einfach **`polar`**. Beachten Sie, dass es eine gleichnamige SCILAB-Funktion gibt (s. oben), die jedoch die Polardarstellung einer komplexen Zahl bestimmt!

Übung 8: Zeichnen Sie folgende Kurve in Polarkoordinaten:

`r = 2*cos(phi), -π/2 ≤ phi ≤ π/2.`

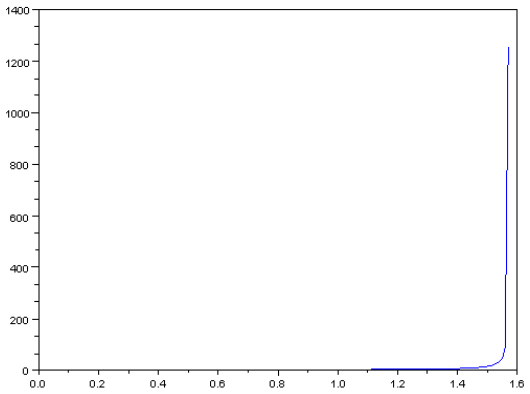
Tipp: Sie können eine **Grafik** mithilfe des Menüpunktes FILE/PRINT des Grafikfensters **ausdrucken**.

Unterschied zu MATLAB: Der Befehl **`print`** druckt nicht die aktuelle Grafik aus, sondern speichert den Wert von Variablen in einer Datei.

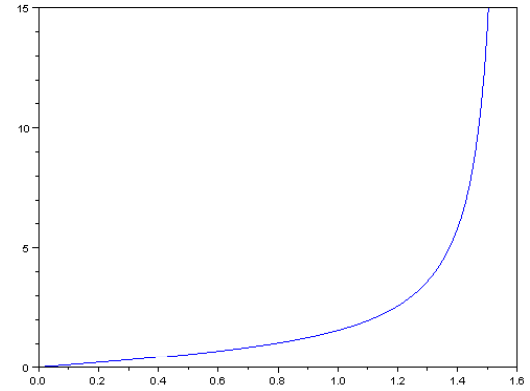
Wie kann ich die Skalierung verändern?

Angenommen, wir möchten die Funktion `y = tan(x)` im Intervall `[0,π/2]` zeichnen:

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



Die Funktion wird nicht gut dargestellt, denn der von SCILAB automatisch gewählte y-Bereich ist zu groß! Wir können den x- und y-Bereich, in dem die Funktion gezeichnet wird, verändern, indem wir im Grafiikfenster die Option EDIT -> FIGURE PROPERTIES anwählen und den Data Range der Achsen beschränken.



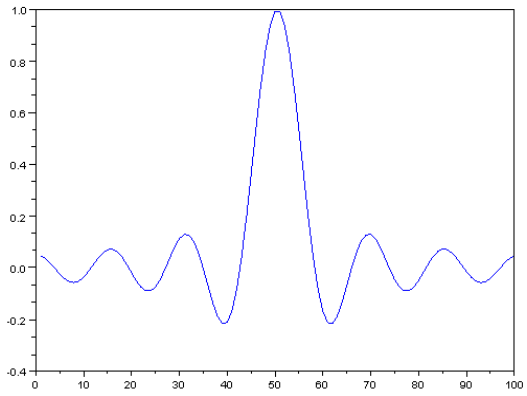
Unterschied zu MATLAB: Die Funktion **`axis`** existiert in SCILAB nicht.

Übung 9:
Zeigen Sie grafisch, dass die Funktion $y = x^3 - x$ nicht monoton wachsend ist.

Kann ich eine Funktion zeichnen, ohne Datenpunkte anzugeben?

Wenn man den Befehl **`plot`** nur mit einem Vektor aufruft, nimmt SCILAB für die x-Achse eine äquidistante Skalierung vor, wobei die Zahl der Punkte der Dimension des Vektors y entspricht.

```
x=linspace(-20,20);y=sin(x)./x;plot(y)
```



Unterschied zu MATLAB: Die Funktion **`fplot`** ist in SCILAB nicht notwendig und daher nicht existent.

Übung 10 (nur MATLAB):
Zeichnen Sie die Funktion `y = sin(x)` im Intervall `[0,2π]` mit `fplot`.

Übung 11 (nur MATLAB):
Zeichnen Sie die Funktion `y = x + 1/(10000x)` im Intervall `[-1,1]` mit `fplot`. Warum gibt die Graphik das Verhalten der Funktion nicht richtig wieder?
(Tipp: Was kann man über das Verhalten von `f` am Nullpunkt aussagen?). Stellen Sie die Grafik richtig, indem Sie den y-Bereich einschränken und die Anzahl der gezeichneten Kurvenpunkte erhöhen.

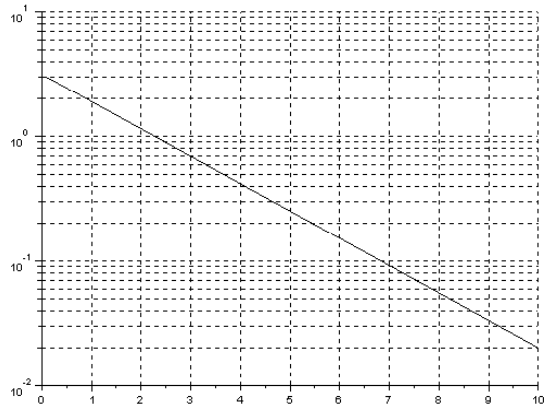
Übung 12: Wurfparabel
Zeichnen Sie die Wurfparabel: `x(t) = v t cos(phi), y(t) = v t sin(phi) - g t²/2`. Anfangsgeschwindigkeit: `v = 100 km/h`, Abwurfwinkel: `phi = 60°`. (Achtung: Einheiten in `m/s` bzw. Radiant umwandeln!)

Logarithmische Skalierung einer Achse

Es kann vorteilhaft sein, eine oder beide Achsen eines rechtwinkligen Koordinatensystems logarithmisch zu skalieren. So wird eine Exponentialfunktion $y = a \cdot e^{b \cdot x}$ in einem "ordinatenlogarithmischen Papier" (d.h. die y-Achse ist logarithmisch geteilt) als Gerade dargestellt.

```
x = linspace(0.1,10);
y = 3*exp(-0.5*x);

plot2d(x,y,logflag="nl");xgrid
```



`plot2d` wird wie **`plot`** verwendet, man kann jedoch einige zusätzliche Optionen angeben. So kann man die Achsen mittels logflag logarithmisch skalieren, wobei der erste Buchstabe für die x-Achse, der zweite für die y-Achse, „l“ für logarithmisch und „n“ für normal steht. Mit dem Befehl **`logspace`** kann man Vektoren erzeugen, deren logarithmierte Einträge denselben Abstand haben.

Der SCILAB-Befehl **`plot`** ist dem entsprechenden MATLAB-Befehl sehr ähnlich, **`plot2d`** bietet dagegen deutlich mehr Möglichkeiten.

Unterschied zu MATLAB: In MATLAB gibt es dafür die Befehle **`semilogy`**, **`semilogx`** und **`loglog`**.

Übung 13: Doppeltlogarithmische Skalierung

Zeichnen Sie die Funktion $y = \sqrt{x}$ in einem doppeltlogarithmischen Papier zwischen 1 und 1000.

Dreidimensionale Plots

Will man eine Funktion z = f(x,y) graphisch veranschaulichen, so kann man den Graph dieser Funktion, eine Fläche, über der (x,y)-Ebene zeichnen. Dazu ist anzugeben, über welchem x-Bereich und y-Bereich gezeichnet werden soll.
Wir wollen die Funktion z = x·y zuerst sehr grobflächig als *Maschenplot* (*Drahtgitter*) über den folgenden 15 Gitterpunkten (-2/-1), (-2/0), ..., (2/1) zeichnen, die aus den fünf x-Werten –2, -1, 0, 1, 2 sowie den drei y-Werten –1, 0 und 1 gebildet werden können.

```
x = -2:1:2
y = -1:1:1
[xx, yy] = meshgrid(x,y)
```

```
x  =
  - 2.  - 1.    0.    1.    2.
y  =
  - 1.    0.    1.
yy  =

  - 1.  - 1.  - 1.  - 1.  - 1.
    0.    0.    0.    0.    0.
    1.    1.    1.    1.    1.
 xx  =

  - 2.  - 1.    0.    1.    2.
  - 2.  - 1.    0.    1.    2.
```

```
  - 2.   - 1.    0.    1.    2.
```

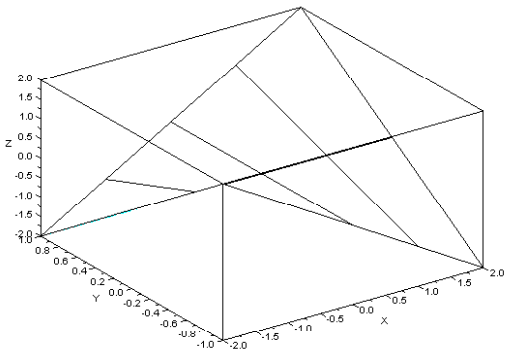
Von besonderer Bedeutung ist hier die Anweisung **`meshgrid`** (*mesh* = *Masche, grid* = *Gitter*). Es erzeugt zwei Matrizen, hier `xx` und `yy` genannt. Einander entsprechende Elemente dieser beiden Matrizen bilden gerade die beiden Koordinaten unserer Gitterpunkte. Anwendung der Punktmultiplikation ergibt:

```
z = xx.*yy
```

```
z  =
    2.    1.    0.  - 1.  - 2.
    0.    0.    0.    0.    0.
  - 2.  - 1.    0.    1.    2.
```

Dies sind die gewünschten Funktionswerte. Zuletzt soll jeder dieser Funktionswerte über "seinem" Gitterpunkt als Punkt dargestellt und in x- und in y-Richtung durch Strecken mit den Nachbarpunkten verbunden werden. Wir erhalten den *Maschenplot* oder das *Drahtmodell* der Funktion. Dies erfolgt mit Hilfe der Anweisung **`mesh`**:

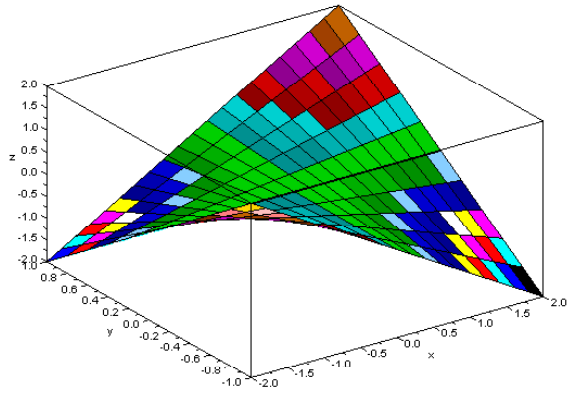
```
mesh(x,y,z)
```



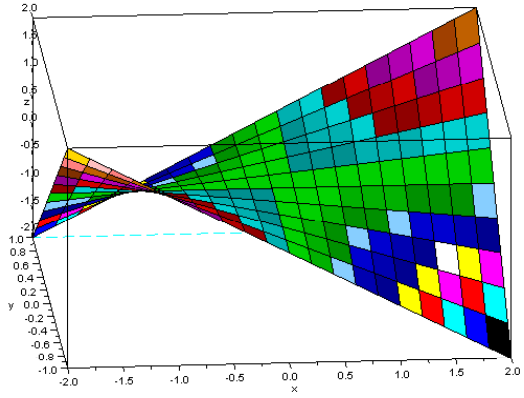
x und y geben die x- bzw. die y-Werte der Gitterpunkte an, z die Funktionswerte über den Gitterpunkten an.

Statt die Punkte durch Strecken zu verbinden, können auch kleine Flächenstücke zwischen den Punkten verwendet werden. Man erhält so ein *Kachelmodell* der Funktion, indem man `mesh` durch **`surf`** (von *surface*) ersetzt. Im Folgenden ist die Fläche außerdem durch mehr Gitterpunkte feinmaschiger gezeichnet.

```
x = -2:0.2:2; y = -1:0.2:1;
[xx,yy] = meshgrid(x,y);
z = xx.*yy;
surf(x,y,z); xlabel('x'); ylabel('y'); zlabel('z')
```



Der Blickwinkel kann mit der Rotationstaste im Grafikfenster verändert werden.



Unterschied zu MATLAB: In SCILAB ist der Befehl view zum Einstellen des Blickwinkels nichts vorhanden.

Übung 14: Lineare Funktion
Stellen Sie die lineare Funktion $z = -0{,}4 \cdot x - 0{,}8 \cdot y + 3$ für $0 \le x \le 5$ und $0 \le y \le 5$ graphisch dar. Um welche Art von Fläche handelt es sich dabei?

Übung 15: Dreidimensionaler Plot
Stellen Sie die Funktion $z = x^2 + y^2$ graphisch für $-3 \le x \le 3$ und $-3 \le y \le 3$ graphisch dar.

Übung 16: Betrag einer komplexwertigen Funktion
Stellen Sie graphisch den Betrag des Kehrwertes von $z = x + j \cdot y$ über der komplexen Ebene dar, also die Funktion $\frac{1}{|z|} = \frac{1}{\sqrt{x^2 + y^2}} = f(x, y)$. Nehmen Sie $-1 \le x \le 1$ und $-1 \le y \le 1$.

Beispiel: Stochastische Integration

Als Anwendung für die in den vorhergehenden Kapiteln vorgestellten Befehle werden wir nun mittels stochastischer Integration eines Kreisviertels eine numerische Näherung für π berechnen.

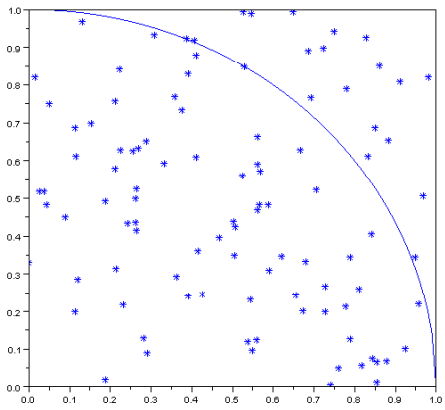
π sei definiert als die Fläche des Einheitskreises (=Kreis um (0,0) mit Radius 1). Die Idee ist, sich viele Zufallszahlen im Quadrat [0,1]x[0,1] zu erzeugen und diejenigen zu zählen, die innerhalb des Einheitskreises liegen. Aus dem Verhältnis jener Zahlen zur Gesamtzahl erhält man nun eine Schätzung für π/4.

Zuerst lassen wir uns von SCILAB beliebig viele Zufallszahlen erstellen (wir nehmen 100):

```
n=100;
A=rand(2,n);
```

Die Matrix A besteht aus 100 zweidimensionalen Vektoren, die alle im Quadrat [0,1]x[0,1] liegen (`rand` erzeugt gleichverteilte Zufallszahlen zwischen 0 und 1). Zeichnen wir uns die Zufallszahlen zusammen mit dem Kreisviertel:

```
plot(A(1,:),A(2,:),"*")
x=linspace(0,1);plot(x,sqrt(1-x^2))
```



Jetzt erstellen wir den Vektor B, in dem der Abstand jedes Zufallspunktes zu 0 eingespeichert wird.

```
B=sqrt(A(1,:)^2+A(2,:)^2);
```

Mittels einer `for`-Schleife über alle Punkte und einer `if`-Abfrage (Details siehe später) erhalten wir unsere Schätzung:

```
count=0; for i=1:n,if B(i)<1 then count=count+1;end;end;count/n
ans  =
    0.81
```

Abhängig von den Zufallszahlen erhält man nun eine mehr oder weniger genaue Schätzung für π/4. Wiederholt man das „Experiment“ oft mit einer wechselnden Anzahl von Zufallszahlen, kann man eine recht gute Genauigkeit erzielen.

Dieses Prinzip findet übrigens in der Wirtschaftswissenschaft unter dem Namen „Monte-Carlo-Simulation“ eine häufige Anwendung.