Arbeiten mit Scilab und Scicos

Scilab für numerische Berechnungen

Scicos für grafische Simulationen

HTW Chur
Hochschule für Technik und Wirtschaft

Fachhochschule Ostschweiz
University of Applied Sciences

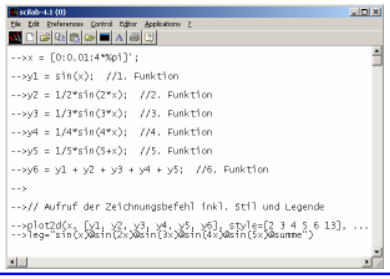
Jean-Marie Zogg

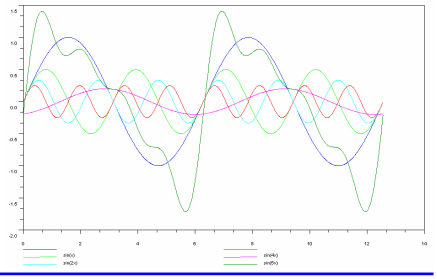
Arbeiten mit Scilab und Scicos

1 Einführung

Scilab ist ein Rechenprogramm. Ähnlich einem wissenschaftlichen Taschenrechner können Zahlen eingegeben und Formeln, Funktionen etc. programmiert, berechnet und ausgeführt werden. Mit **Scilab** wird der Verlauf von Funktionen in grafischen Fenstern angezeigt.

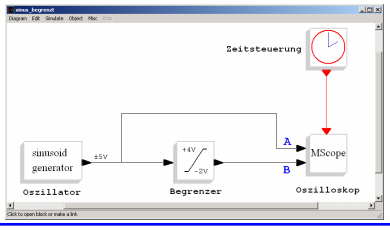
Scilab für numerische Berechnungen

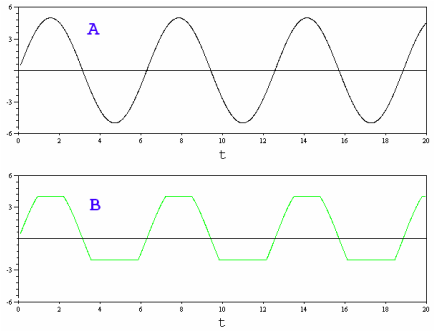




Scicos, eine Ergänzung zu **Scilab**, ist ein Programm, um ein Verhalten zu modellieren bzw. zu simulieren: Anstatt Formeln zu schreiben, dienen vorbereitete Funktionsblöcke (= Kästen) zur Beschreibung und Simulation eines Ablaufs.

Scicos für grafische Simulationen





Scilab und **Scicos** sind keine Lernprogramme für Mathematik. Wer **Scilab** und **Scicos** anwendet, sollte die Gesetzmässigkeiten der Mathematik bereits kennen!
Die folgende Einleitung gibt Ihnen einen Überblick über die wichtigsten Möglichkeiten von **Scilab** und **Scicos**. Damit die Anleitung übersichtlich bleibt, wurden viele Funktionen von **Scilab** und **Scicos** nicht behandelt. Die Anleitung zeigt Ihnen aber auch, wie Sie verfahren können, sollten Sie nicht mehr weiterkommen.
Dem Autor war es ein Anliegen, den Text einfach und kurz zu halten, die Abschnitte und die Erklärungen sinnvoll zu gliedern und anregende Beispiele zu geben.
Sie sollten versuchen, Ihre mathematischen, physikalischen und technischen Probleme möglichst oft mit **Scilab** und **Scicos** zu lösen und diese Anleitung zu verwenden. Ergänzen Sie diese Anleitung mit eigenen Kommentaren und teilen Sie dem Autor (mailto:jean-marie.zogg@fh-htwchur.ch) Unstimmigkeiten und Verbesserungsvorschläge bitte mit. Weitere Nutzer dieser Anleitung werden Ihnen dankbar sein.

2 Inhaltsverzeichnis

3 Die verschiedenen Anwendungen von Scilab

Um was geht es in diesem Kapitel?
Dieser Abschnitt gibt Ihnen einen Überblick über die wichtigsten Anwendungen von **Scilab** und **Scicos**. Eine detaillierte Beschreibung der einzelnen Funktionen folgt in einem späteren Abschnitt.

Was werden Sie erreichen?
Sie werden eine Vorstellung von den vielfältigen Möglichkeiten von **Scilab** und **Scicos** haben. Sie werden gespannt sein, was sich noch alles mit diesem Programm realisieren lässt. Sie werden Lust kriegen, es selbst auszuprobieren!

Was müssen Sie tun?
Sie sollten beobachten und versuchen, sich den Nutzen dieser Anwendung vorzustellen. Später (ab Kapitel 5) sollten Sie versuchen, die Beispiele selbst durchzuspielen.

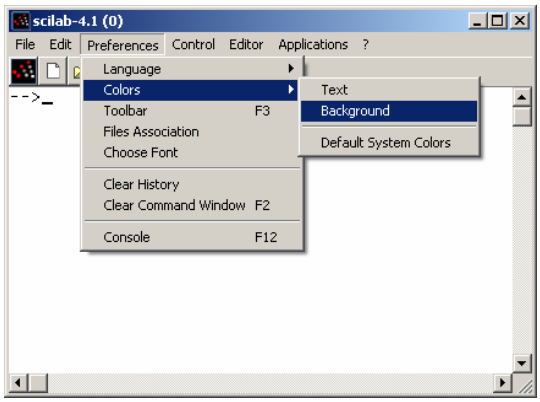
Erklärung zu der Darstellung
Damit klar ist, was eingegeben wurde, wird zwischen **Eingabe** und **Resultat** farblich unterschieden. Drucken Sie diese Unterlagen in Farbe aus.

```
Prompt
Eingabe
Resultat
//Kommentar
```

```
Prompt-Zeichen (-->): Es wird automatisch von Scilab generiert und
zeigt an, dass Scilab bereit ist, Eingaben entgegen zu nehmen
Eingabe: Das geben Sie ein
Resultat: Das ist die Antwort von Scilab
Kommentar: Das ist ein Kommentar und muss mit // beginnen
```

Scilab-Menübefehle
Scilab-Menübefehle werden im Text folgendermassen angezeigt:
Befehl1 ⇨ Befehl2 ⇨ Befehl3

Beispiel:
Die Menü-Befehlsreihenfolge (siehe Bild) *Preferences, Colors* und *Background* wird in folgender Kurzform angezeigt:
Preferences ⇨ Colors ⇨ Background



Scilab-Befehle werden folgendermassen angezeigt: ***Befehl()***

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

%pi	%e	sin(x)	exec(‘ ‘)	x = a
sqrt(x)	log(x)	^	plot2d(x, [f(x])	for a = x, b = y, end
/	*	+	clf()	\
-	rand(x)	;	conj(Z)	atan(x,y)
[a b c ; e f g]	[a b c ; e f g]’	%i	imag(Z)	abs(Z)
real(Z)				

3.1 Scilab als Taschenrechner

Scilab kann als gewöhnlicher Taschenrechner verwendet werden. Der Befehl wird nach dem Prompt-Zeichen (-->) eingegeben. Nach der Betätigung der Enter-Taste (↵) erscheint das Resultat.

Beispiel:
Gezeigt werden Wertzuweisungen, die Eingabe von zwei Konstanten und die Anwendung von einfachen Grundfunktionen.

Wertzuweisung (mit Bildschirmausgabe):

$x = 6.5 \cdot 10^3$

Wertzuweisung (ohne Bildschirmausgabe):

$a = 4;\ b = 7;$

Eingabe von Konstanten (immer mit %...):

π

e

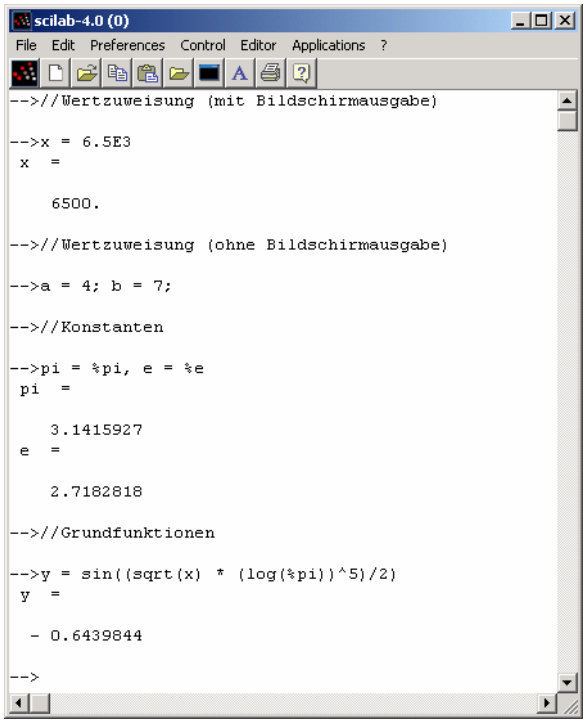
Mit Grundfunktionen rechnen:

$$y = \sin\left[\frac{\left(\sqrt{x} \cdot \ln(\pi)\right)^5}{2}\right]$$

```
-->//Wertzuweisung (mit Bildschirmausgabe)

-->x = 6.5E3
 x  =

    6500.

-->// Wertzuweisung (ohne Bildschirmausgabe)

a = 4; b = 7;

-->//Konstanten

-->pi = %pi, e = %e
 pi  =

    3.1415927
 e  =

    2.7182818

-->//Grundfunktionen

-->y = sin((sqrt(x) * (log(%pi))^5)/2)
 y  =

  - 0.6439844

-->
```

Ansicht des Scilab-Bildschirms

```
Prompt
Eingabe
Resultat
//Kommentar
```



So sieht der Bildschirm von Scilab tatsächlich aus

3.2 Scilab als Vektoren- und Matrizen-Rechner

Scilab wurde speziell entwickelt, um mit Matrizen und Vektoren zu rechnen. Weitere Informationen zu diesem Thema finden Sie im Kapitel 7.

Beispiel:
Gezeigt werden die Erzeugung von Matrizen und von Vektoren, sowie eine einfache Matrixumrechnung.

Erzeugung einer 3x3-Matrix:

$$Matrix\_1 = \begin{bmatrix} 1 & 7 & 3 \\ 4 & 9 & 6 \\ 2 & 8 & 5 \end{bmatrix}$$

Inverse Matrix:

$$Matrix\_1\_invers = \begin{bmatrix} 1 & 7 & 3 \\ 4 & 9 & 6 \\ 2 & 8 & 5 \end{bmatrix}^{-1}$$

Spaltenvektor:

$$Spalten\_Vektor = \begin{bmatrix} 7 \\ 34 \\ -39 \end{bmatrix}$$

Matrix mit Zufallswerten:
Verwendung des Befehls „***rand***“

```
-->//Erzeugung einer 3x3-Matrix

-->Matrix_1 = [1 7 3 ; 4 9 6 ; 2 8 5]
 Matrix_1  =

    1.    7.    3.
    4.    9.    6.
    2.    8.    5.

-->// Berechung der inversen Matrix

-->Matrix_1_invers = inv(Matrix_1)
 Matrix_1_invers  =

    0.1764706    0.6470588  - 0.8823529
    0.4705882    0.0588235  - 0.3529412
  - 0.8235294  - 0.3529412    1.1176471

-->// Erzeugung eines Spaltenvektors

-->Spalten_Vektor = [7 ; 34 ; -39]
 Spalten_Vektor  =

    7.
    34.
  - 39.

-->// 3x2-Matrix mit Zufallswerten

-->MZ = rand(3,2)
 MZ  =

    0.7263507    0.2320748
    0.1985144    0.2312237
    0.5442573    0.2164633

-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

3.3 Scilab als Rechner für komplexe Zahlen

Mit **Scilab** können Rechnungen mit komplexen Zahlen durchgeführt werden. Komplexe Zahlen bestehen aus Real- und Imaginärteil. Weitere Informationen zu diesem Thema finden Sie im Kapitel 8.

Beispiel:
Gezeigt wird, wie komplexe Zahlen eingegeben und wie einzelne Grössen (z. B. Betrag, Argument) aus komplexen Zahlen extrahiert werden.

Kartesische Form:
$Z\_1 = 1 + i$

Trigonometrische Form:
$Z\_2 = \sqrt{2} \cdot cis\left(\frac{\pi}{4}\right)$

Extraktion von:
- **Realteil**
- **Imaginärteil**
- **Betrag**

Bestimmung der konjugiert-komplexen Zahl

Berechnung des Arguments (in Grad umgerechnet) einer komplexen Zahl

```
-->// Eingabe in kartesischer Form

-->Z_1 = 1  + 1*%i
 Z_1  =

    1. + i

-->//Eingabe in trigonometrischer Form (= Polarform)

-->Z_3 = sqrt(2) * (cos(%pi/4) + %i*sin(%pi/4))
 Z_3  =

    1. + i

-->// einzelne Operationen

-->Real_Teil = real(Z_1)
 Real_Teil  =

    1.

-->Imag_Teil = imag(Z_1)
 Imag_Teil  =

    1.

-->Betrag = abs(Z_1)
 Betrag  =

    1.4142136

-->Z_conj = conj(Z_1)
 Z_conj  =

    1. - i

-->// Argument einer komplexen Zahl

-->Winkel = ((atan(imag(Z_1),real(Z_1)))/(2*%pi))*360
 Winkel  =

    45.

-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

3.4 Scilab zum Lösen von Gleichungen

Mit **Scilab** können lineare und nichtlineare Gleichungssysteme einfach gelöst werden.
Weitere Informationen zu diesem Thema finden Sie im Kapitel 9.

Beispiel: Lösung eines linearen Gleichungssystems
Gesucht sind drei unbekannte Grössen: X_1, X_2 und X_3.
Gegeben sind folgende drei Gleichungen:

Erste Gleichung:	X_1 – X_2 + X_3	=	0
Zweite Gleichung:	2•X_1 + 3•X_2	=	2
Dritte Gleichung:	3•X_2 + 4•X_3	=	4

Gleichungssystem:

$$\begin{bmatrix} 1 & -1 & 1 \\ 2 & 3 & 0 \\ 0 & 3 & 4 \end{bmatrix} \bullet \begin{bmatrix} X\_1 \\ X\_2 \\ X\_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix}$$

Aufgelöst nach X_1, X_2 und X_3:

$$\begin{bmatrix} X\_1 \\ X\_2 \\ X\_3 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 2 & 3 & 0 \\ 0 & 3 & 4 \end{bmatrix}^{-1} \bullet \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix}$$

1. Lösungsweg

2. Lösungsweg

Lösung 1 und Lösung 2 sind identisch!

```
-->//Koeffizienten-Matrix KM

-->KM = [1 -1 1 ; 2 3 0 ; 0 3 4]
 KM  =

    1.   - 1.     1.
    2.     3.     0.
    0.     3.     4.

-->//Spaltenvektor SV

-->SV = [0 ; 2 ; 4]
 SV  =

    0.
    2.
    4.

-->//Lösungsweg 1
-->//(Gesucht ist X-Vektor mit X_1, X_2, X_3)

-->X = inv(KM) * SV
 X  =

    0.0769231
    0.6153846
    0.5384615

-->//oder Lösungsweg 2

-->X = KM\SV
 X  =

    0.0769231
    0.6153846
    0.5384615

-->
```

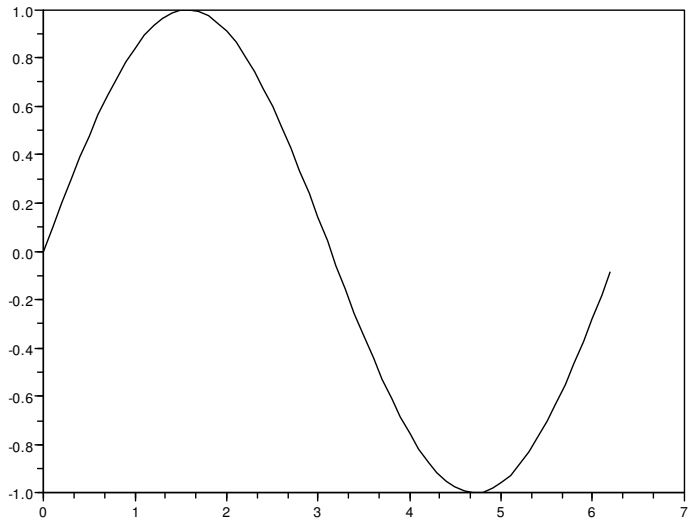
```
Prompt
Eingabe
Resultat
//Kommentar
```

3.5 Scilab als Kurvenzeichner

Mit **Scilab** können Grafen von Funktionen dargestellt werden. Weitere Informationen zu diesem Thema finden Sie im Kapitel 10.

Beispiel:
Gezeichnet wird eine Sinuskurve im Bereich von 0 bis 2π.

```
-->// x Initialisierung
-->x=[0:0.1:2*%pi]';
-->// Löschen von älteren Grafiken
-->clf()
-->// Zeichnen der Sinuskurve im Bereich von x
-->plot2d(x,[sin(x)])
-->
```




```
Prompt
Eingabe
Resultat
//Kommentar
```

3.6 Scipad, der Editor zu Scilab

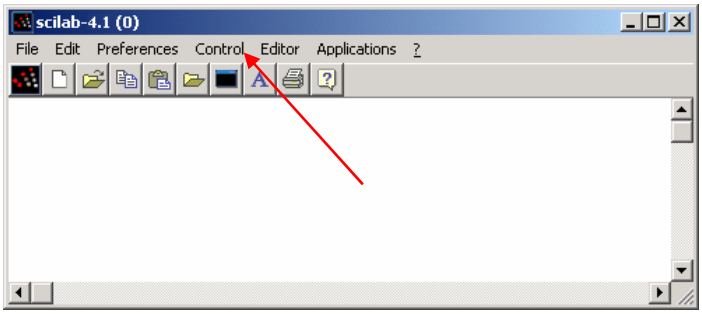
Anstatt Zeile um Zeile in die **Scilab**-Oberfläche einzugeben und jeweils das Resultat abzuwarten, kann die gesamte Berechnung mit dem Editor **Scipad** geschrieben und aus **Scipad** ausgeführt werden. Dies erlaubt einfache Korrekturen, ohne jeweils alle Befehle neu einzugeben. Die editierte Datei wird als **Scilab-Skript**-Datei abgespeichert und erhält die **Endung .sce**. Weitere Informationen zu diesem Thema finden Sie im Kapitel 6.

Beispiel:
Wird ein Gleichungssystem mit **Scilab** aufgelöst und will man nachträglich einen Koeffizienten ändern, so müssen alle Befehle neu eingegeben werden. Wird **Scipad** benützt, so muss im Editor nur der entsprechende Wert geändert werden.

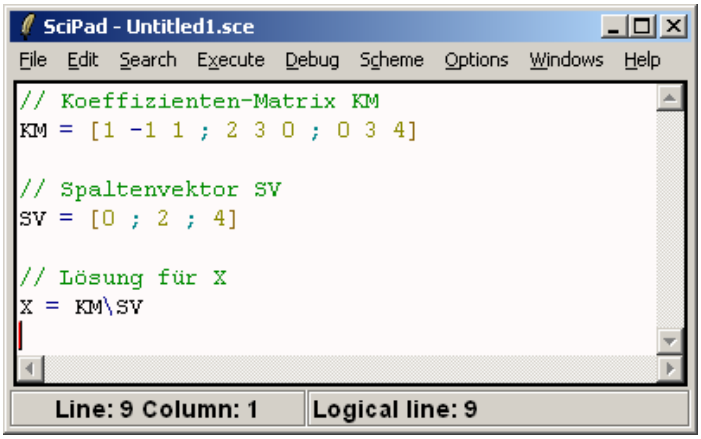
Lösung der Gleichung mit **Scilab**:

```
-->// Koeffizienten-Matrix KM

-->KM = [1 -1 1 ; 2 3 0 ; 0 3 4]
 KM  =

    1.   - 1.     1.
    2.     3.     0.
    0.     3.     4.

-->// Spaltenvektor SV

-->SV = [0 ; 2 ; 4]
 SV  =

    0.
    2.
    4.

-->// Lösung für X

-->X = KM\SV
 X  =

    0.0769231
    0.6153846
    0.5384615
```

So wird der Editor **Scipad** von **Scilab** aus aufgerufen:



In **Scipad** werden Grössen und Befehle eingegeben



Aus **Scipad** werden die Befehle ausgeführt
Execute ⇨ Load into **Scilab**.
Das Resultat erscheint direkt auf der **Scilab** Oberfläche:

```
--> X  =
    0.0769231
    0.6153846
    0.5384615
-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

3.7 Scilab kann programmiert werden

Mit **Scilab** können automatische Abläufe von Berechnungen programmiert werden. Befehle für Schleifen wie for, while, case, else, etc. stehen zur Verfügung. Um Programme zu schreiben und auszuführen, eignet sich der Editor ***Scipad***. Weitere Informationen zu diesem Thema finden Sie im Kapitel 12.

Beispiel:
Eine Möglichkeit, Quadrat- und Kubikzahlen zu bestimmen, ist die Verwendung einer for-Schleife. Im Beispiel wird das Programm im Editor **Scipad** geschrieben und von dort ausgeführt.

Das Programm wird im **Scipad**-Editor geschrieben:

```
// Berechnung des Quadratwertes (QW)
// und des Kubikwertes (KW)
// für Zahlen (Z) im Bereich von A bis B
A = 0; B = 5;
for Z = A:1:B, QW = [Z Z^2 Z^3], end
```

Anzeige in **Scilab** nach der Ausführung:

```
--> Wert  =

    0.    0.    0.
 Wert  =

    1.    1.    1.
 Wert  =

    2.    4.    8.
 Wert  =

    3.    9.    27.
 Wert  =

    4.    16.    64.
 Wert  =

    5.    25.    125.
```

Hinweis: Die gleiche Aufgabe könnte auch direkt in **Scilab** als Rechnung ausgeführt werden:

```
-->A = 0; B = 5;

-->Z = [A:1:B]';

-->Wert = [Z Z^2 Z^3]
 Wert  =

    0.    0.    0.
    1.    1.    1.
    2.    4.    8.
    3.    9.    27.
    4.    16.    64.
    5.    25.    125.

-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

3.8 Scilab kann Funktionen erzeugen

Mit **Scilab** können Sie eigene lokale (nur gültig in der verwendeten Datei, siehe Abschnitt 5.6) und globale (können von einer beliebigen Datei aus aufgerufen werden, siehe Abschnitt 10.6) Funktionen definieren, schreiben und abspeichern. Die globalen Funktionen können später aus **Scilab** aufgerufen und verwendet werden. Globale Funktionen werden wie Scilab-eigene Funktionen eingesetzt. Bevor die globale Funktion ausgeführt wird, muss sie in **Scilab** geladen (eingebunden) sein. Globale Funktionen haben die **Endung .sci**. Weitere Informationen zu diesem Thema finden Sie im Kapitel 11.

Beispiel:
Wir wollen eine eigene globale **Scilab**-Funktion erstellen. Diese Funktion bestimmt die Polargrössen (Argument und Betrag) von einer komplexen Zahl, dargestellt in kartesischer Form. Die Funktion hat die Bezeichnung *rect2polar.sci*. Die Funktion speichern wir im Ordner *C:\Scilab*.

Die in **Scipad** selbst erstellte globale Funktion *rect2polar.sci* wird im Ordner *C:\Scilab* gespeichert:

```
// rect2polar
// Funktion zur Umwandlung einer komplexen Zahl Z
// von der kartesischen- zur Polarform
// (a + ib) zu (r, phi)
// Der Winkel wird in Grad ausgegeben
function [Betrag, Winkel] = rect2polar(Z)
Winkel = ((atan(imag(Z), real(Z)))/(2*%pi))*360
Betrag = abs(Z)
endfunction
```

Die neu erstellte globale Funktion „rect2polar“ wird in **Scilab** eingesetzt:

```
-->// Funktion rect2polar.sci wird geladen

-->exec('C:\Scilab\rect2polar.sci');

-->// Eine komplexe Zahl Z wird definiert

-->Z = -12 -7*%i;

-->//Winkel und Betrag werden bestimmt:
-->//Dazu wird die Funktion rect2polar aufgerufen

-->[Betrag, Winkel] = rect2polar(Z)
 Winkel  =
  - 149.74356
 Betrag  =
    13.892444
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Hinweis: der Befehl zum Einbinden einer globalen Funktion `-->exec('C:\Scilab\rect2polar.sci');` muss nicht manuell eingegeben werden, sondern kann direkt aus der **Scilab**-Befehlsleiste ***File ⇨ Exec ...*** aufgerufen werden.

3.9 Lesen von Dateien und schreiben in Dateien

Mit **Scilab** können fremde Dateien (z. B. eine Textdatei) erstellt, geöffnet und geschlossen werden. Der Inhalt solcher fremder Dateien kann gelesen und verändert werden.
Weitere Informationen zu diesem Thema finden Sie im Kapitel 13.

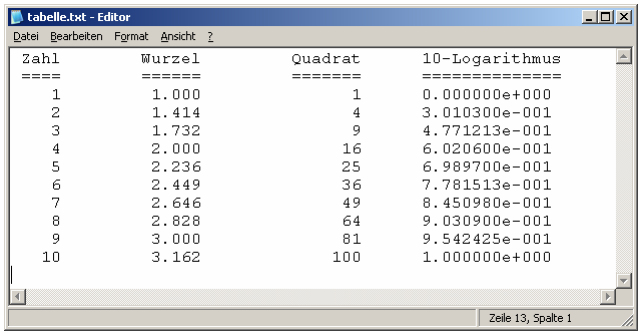
Beispiel: Eine Tabelle wird in einer Text-Datei erstellt

In einer Datei soll eine Tabelle erstellt werden. Für die Zahlen 1 bis 10 sollen der Wurzelwert, der Quadratwert und der Zehner-Logarithmus aufgelistet werden.

Befehle in **Scipad**:

```
// ============= Tabelle ==========================
// Beispiel zum Thema "Schreiben in einer Datei"
// Tabelle mit den Wurzelwerten, Quadratwerten und Zehner-Logarithmus wird erstellt
// Verwenden Sie die Schriftart "Courier" oder "Courier-New" um eine sinnvolle Anzeige zu haben.

// Datei wird erstellt
Datei = mopen('c:\scilab\tabelle.txt', 'w');

// Titelzeile wird erstellt
mfprintf(Datei, '%5s\t %10s\t %10s\t %10s\n', 'Zahl', 'Wurzel', 'Quadrat', '10-Logarithmus');
mfprintf(Datei, '%5s\t %10s\t %10s\t %10s\n', '====', '======', '=======', '==============');

// Tabelle wird erstellt und Zeile um Zeile ausgedruckt
for i = 1:10
  wurzel_wert = sqrt(i);
  quadrat_wert = i^2;
  log_wert = log10(i);
  mfprintf(Datei, '%5d\t %10.3f\t %10d\t %10e\n', i, wurzel_wert, quadrat_wert, log_wert);
end

// Datei wird geschlossen
mclose(Datei);
```

Inhalt der Datei “tabelle.txt“. Diese Text-Datei wird mit dem Editor-Programm von Windows betrachtet:

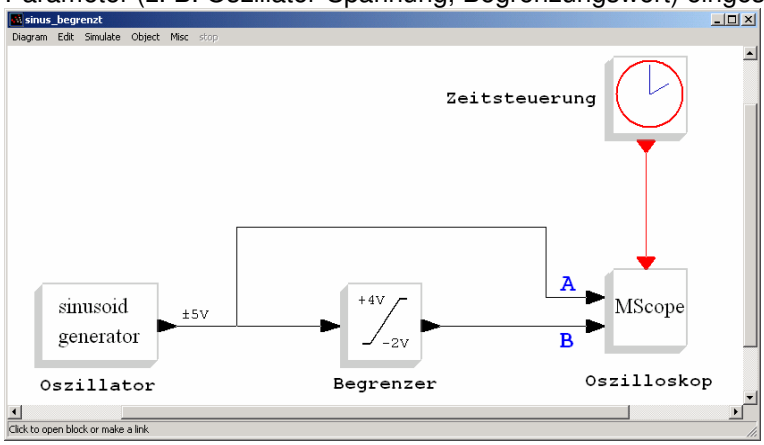


3.10 Scicos, der Blockschema-Simulator von Scilab

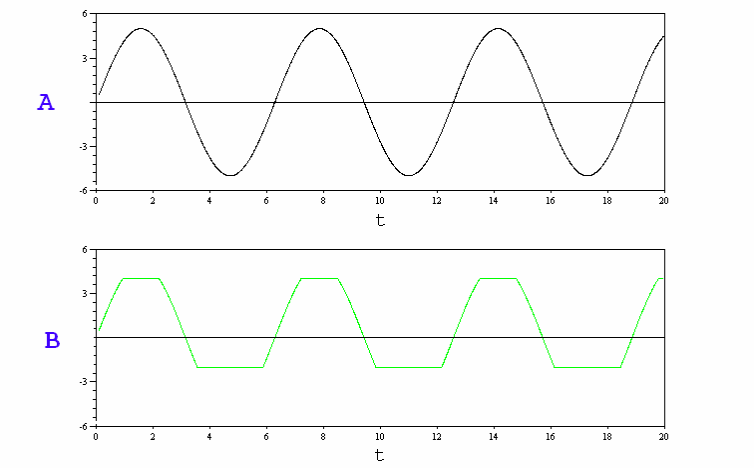
Scicos ist integriert in **Scilab**. **Scicos** ist ein blockorientiertes grafisches Simulationssystem, d.h. die Modelle werden nicht durch Gleichungen beschrieben, sondern es wird ein Blockschaltbild aufgebaut, das unmittelbar zur Simulation und Analyse dient. Aus einer Bibliothek lassen sich die gewünschten Blöcke auswählen. Mit Hilfe der grafischen Oberfläche werden aus diesen Blöcken durch Zeichnen von Blockdiagrammen ganze Systeme modelliert. Nachdem man ein Modell erstellt hat, kann man dieses simulieren. Mit Hilfe von ‚Scopes' (Oszilloskopen) werden die Simulationsergebnisse während der Simulation angezeigt. Weitere Informationen zu diesem Thema finden Sie im Kapitel 14.

Beispiel:
Was passiert, wenn ein Sinussignal durch eine Schaltung in der Amplitude begrenzt wird? Die Antwort gibt uns **Scicos**. Die Amplitude des Sinussignals beträgt 5V. Die Begrenzerschaltung limitiert das Signal auf +4V und -2V. Untersucht wird das Verhalten im Zeitbereich 0s bis 20s.

In **Scicos** wird das Blockschema erstellt und die einzelnen Parameter (z. B. Oszillator-Spannung, Begrenzungswert) eingestellt:



Das erstellte Blockschema wird simuliert und angezeigt:



4 Installation und Hilfestellungen von Scilab

Um was geht es in diesem Kapitel?

Dieser Abschnitt erklärt Ihnen, wie **Scilab** (inkl. **Scicos**) auf Ihrem Computer installiert wird, worauf Sie bei der Installation achten müssen und wie **Scilab** gestartet wird. Wenn später beim Einsatz von **Scilab** Fragen auftauchen, oder wenn Sie die genaue Bedeutung eines Befehls wissen möchten, wird Ihnen gezeigt, wo Sie sich eine „Erste Hilfe“ holen können.

Was werden Sie erreichen?

Sie werden **Scilab** installiert haben und wissen, wie Sie das Programm aktualisieren können. Sie werden die verschiedenen Möglichkeiten kennen, um Hilfe zu bekommen. Sie werden **Scilab** starten und eine kleine Anwendung testen.

Was müssen Sie tun?

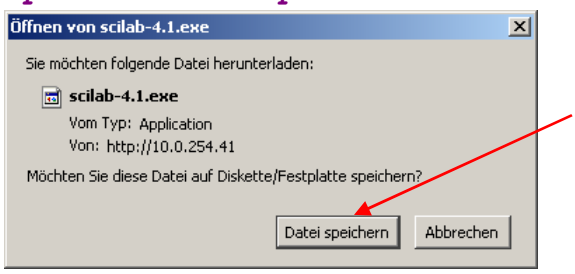
Sie sollten **Scilab** auf Ihrem Computer installieren, ev. aktualisieren und starten. Zudem sollten Sie die verschiedenen Hilfestellungen kritisch begutachten und sich vorstellen, wie Sie sie später nutzen werden.

4.1 Installation

Scilab wird von der Seite *http://www.Scilab.org/* heruntergeladen und auf Ihrem Computer installiert.

a. Gehen Sie zur Seite *http://www.Scilab.org/* und klicken Sie auf den Button oben links *Download Scilab 4.1* (Dies gilt für die Windows-Version, ansonsten müssen Sie den Link *Others systems & source files* anklicken).

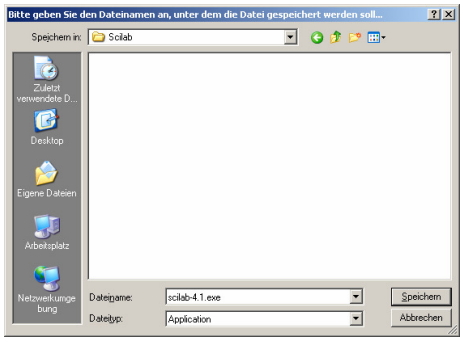
b. Je nach Betriebssystem oder installiertem Browser erscheint ein Fenster ***Öffnen von Scilab-4.1.exe*** oder ***Dateidownload***. Betätigen Sie den Button ***Datei Speichern*** oder ***Speichern*** und drücken Sie den Button ***OK***.



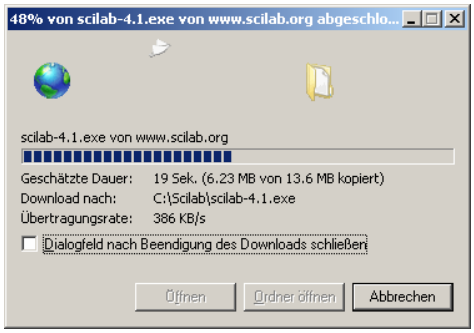
Firefox

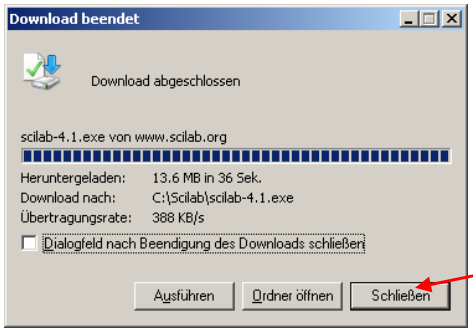
Windows Internet-Explorer

c. Im erscheinenden Fenster müssen Sie den Speicherort angeben. Speichern Sie die Datei in einem Ordner mit dem Namen `Scilab`

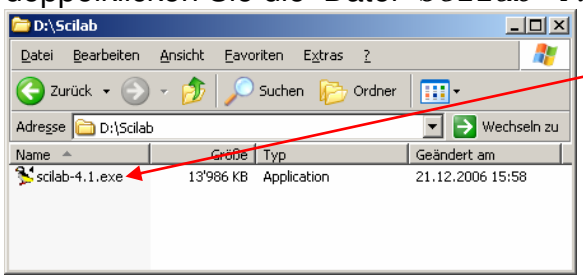


d. Der Datei-Download wird gestartet (linkes Bild). Ist der Download beendet (rechtes Bild), müssen Sie den Button ***Schliessen*** drücken.



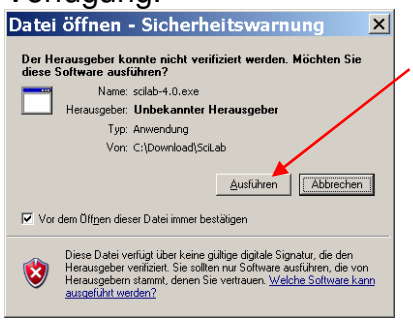


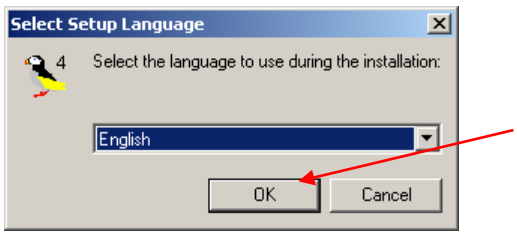
e. Gehen Sie zum Ordner, in dem **Scilab** gespeichert wurde (siehe Punkt c.) und doppelklicken Sie die Datei `Scilab-4.1.exe`



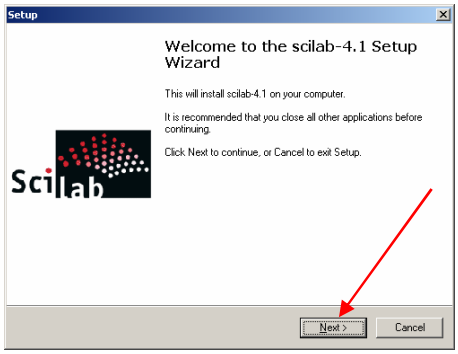
f. Die Installationsausführung beginnt. Vielleicht müssen Sie zuerst noch Sicherheitswarnungen akzeptieren (Bild links). Bevor die Installation beginnt, dürfen Sie

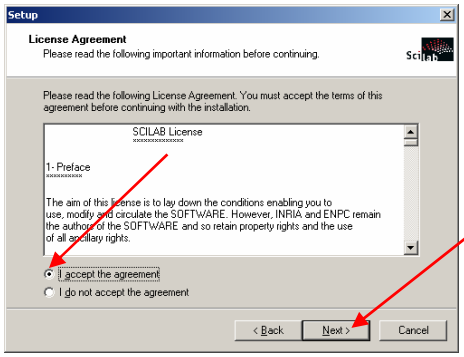
die Sprachversion auswählen (Bild rechts). Zurzeit stehen Englisch und Französisch zur Verfügung.



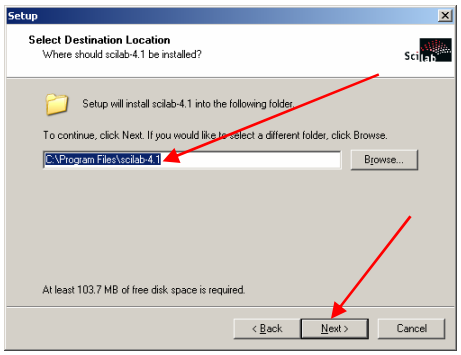


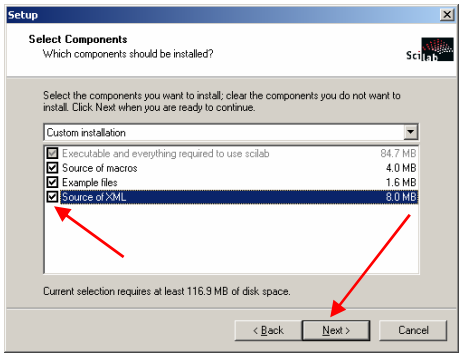
g. **Scilab** begrüsst Sie und fordert Sie auf, den Button ***Next>*** zu aktivieren (Bild links). Die Lizenzvereinbarung sollten Sie sorgfältig lesen und akzeptieren (bild rechts). Aktivieren Sie den Button ***Next>***



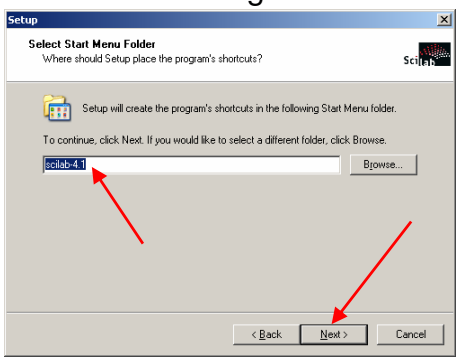


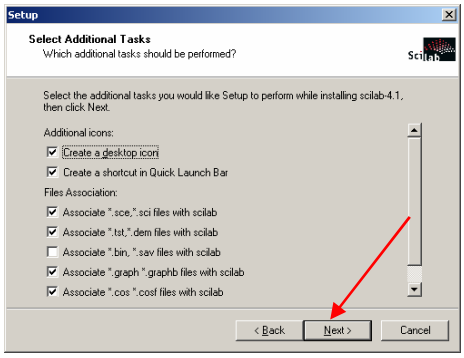
h. Das Installationsprogramm fragt Sie nach dem Zielort des Programms. Grundsätzlich können Sie den vorgeschlagenen Ort durch Aktivierung des Button ***Next>*** akzeptieren (Bild links). Beim Fenster `Select Components` sollten Sie **alle** Optionen anklicken und ***Next>*** klicken.



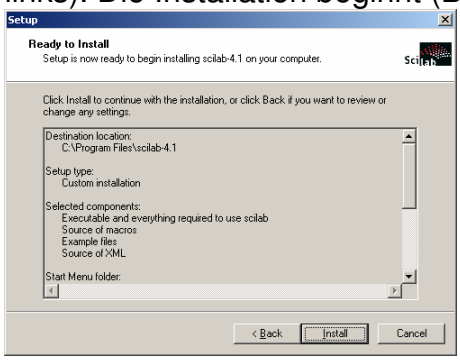


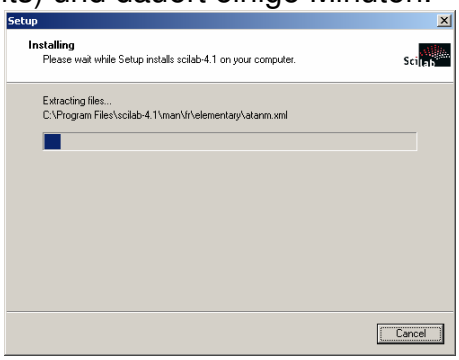
i. Das Installationsprogramm möchte von Ihnen wissen, wie die Kurzbezeichnung des Programms in der Programmleiste benannt werden soll (Bild links). Akzeptieren Sie den Vorschlag durch Klicken des Buttons ***Next>***. Das Installationsprogramm fragt Sie, welche zusätzlichen Funktionen noch installiert werden sollen (Bild rechts). Akzeptieren Sie den Vorschlag durch Drücken des Button ***Next>***



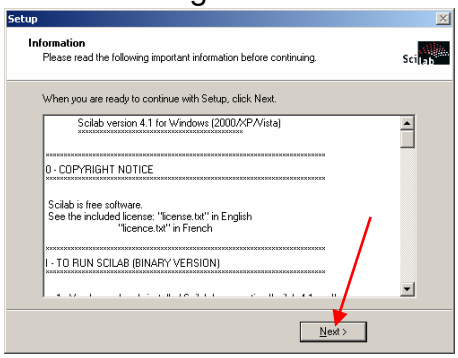


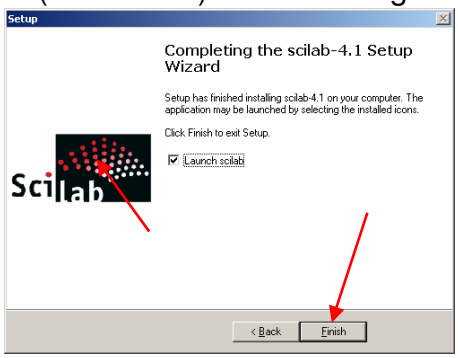
j. Endlich kann die Installation durch klicken der Taste ***Install*** gestartet werden (Bild links). Die Installation beginnt (Bild rechts) und dauert einige Minuten.



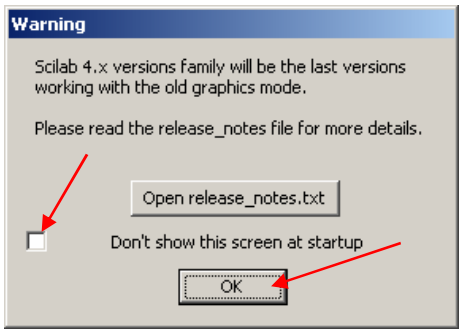


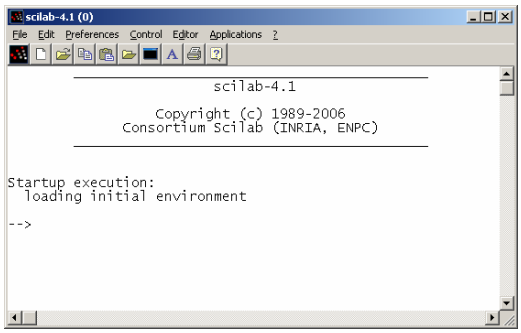
k. War die Installation erfolgreich, wird dies gemeldet und Sie sollten nach dem Lesen der Mitteilung die Taste ***Next>*** drücken (Bild links). Das Installationsprogramm fordert Sie nun auf, die Installation durch Klicken des Button ***Finish*** zu beenden. Belassen Sie die Voreinstellung ***Launch Scilab*** aktiviert (Bild rechts) und das Programm wird starten.



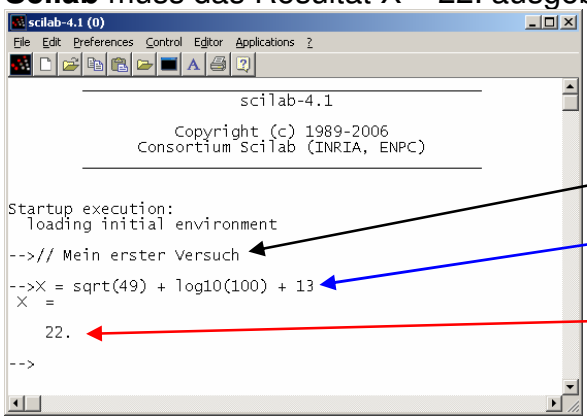


l. **Scilab** wird gestartet. Vielleicht erscheint noch eine Warnung (Bild links). Aktivieren Sie das kleine Feld ***Don't show this screen at startup*** und klicken Sie die Taste ***OK***. Der Startbildschirm von **Scilab** erscheint.





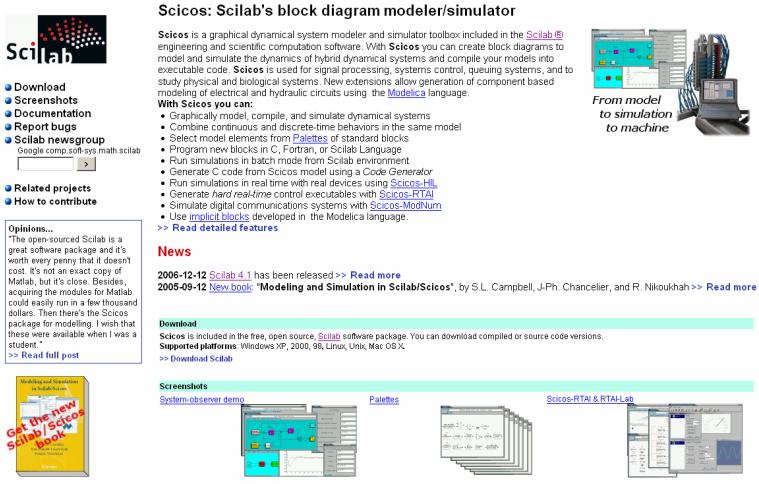
m. Testen Sie **Scilab** durch Eingabe folgender zwei Zeilen:
// Mein erster Versuch ↵
// x = sqrt(49) + log(100) + 13 ↵
Scilab muss das Resultat X = 22. ausgeben!




```
Prompt
Eingabe
Resultat
//Kommentar
```

```
-->// Mein erster Versuch

-->X = sqrt(49) +log10(100) + 13
 X  =

    22.

-->
```

n. Sie sollten noch auf der Homepage von **Scicos** *http://www.Scicos.org* kontrollieren, ob irgendwelche Patches (Reparaturmodule) für **Scicos** notwendig sind. Wenn Ja, dann müssen diese installiert werden, damit **Scicos** richtig funktioniert.



Nach der Aktualisierung von **Scicos** (wenn notwendig!) ist die Installation von **Scilab** abgeschlossen. Viel Spass bei der Anwendung!

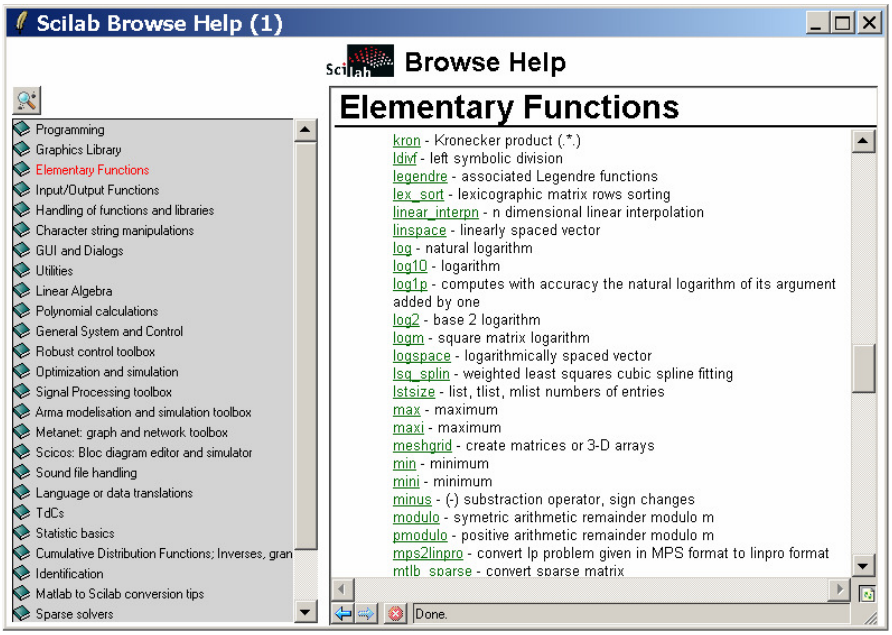
4.2 Hilfestellungen

Wenn Sie nicht mehr weiter wissen oder wenn Sie unsicher in der Anwendung von **Scilab** sind, dann verfügen Sie über starke Hilfepartner. Einige von diesen Hilfestellungen werden hier vorgestellt.

4.2.1 F1 / Help

Offline-Help: Das Pogramm **Scilab** stellt Ihnen ein internes Hilfemenü zur Verfügung. Aufgerufen wird das Hilfemenü mittels ***F1*** oder über ***? ⇨ Scilab Help F1*** in der Menüleiste.
Der Help-Browser ist dreiteilig:

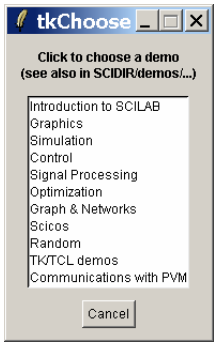
- Links des Browsers sind alle Befehlsgruppen aufgelistet
- Rechts sind die Inhalte der jeweiligen angewählten Befehlsgruppe (im Beispiel: Elementary Functions). Wenn Sie im rechten Fenster einen unterstrichenen Befehl anklicken, dann erhalten Sie Detailinformationen darüber.
- Durch Aktivierung des Lupensymbols (oben links) können Sie einen Suchbegriff eingeben und der Hilfebrowser listet Ihnen alle passenden Möglichkeiten im linken Teil des Fensters auf.



Online-Help: Auf der Homepage von **Scilab** *http://www.Scilab.org/* finden Sie unter dem Button *Technical aera* den Menüpunkt *Documentation and Support center* unter *Documentation* den weiteren Menü-Punkt *On line help*. Diese Hilfestellung ist gleich aufgebaut wie die Offline-Hilfestellung.

4.2.2 Scilab-Demo

Mit dem Menübefehl **? ⇨ Scilab Demos** öffnet sich das Fenster *tkChoose*

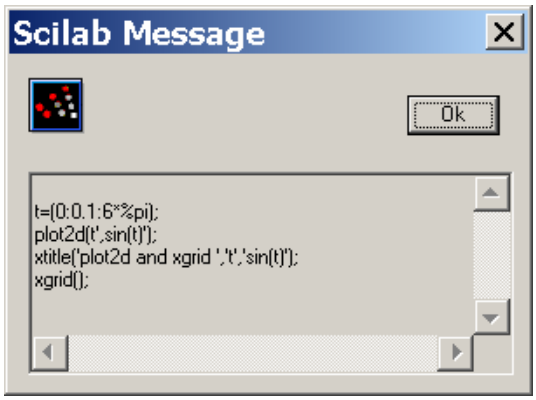


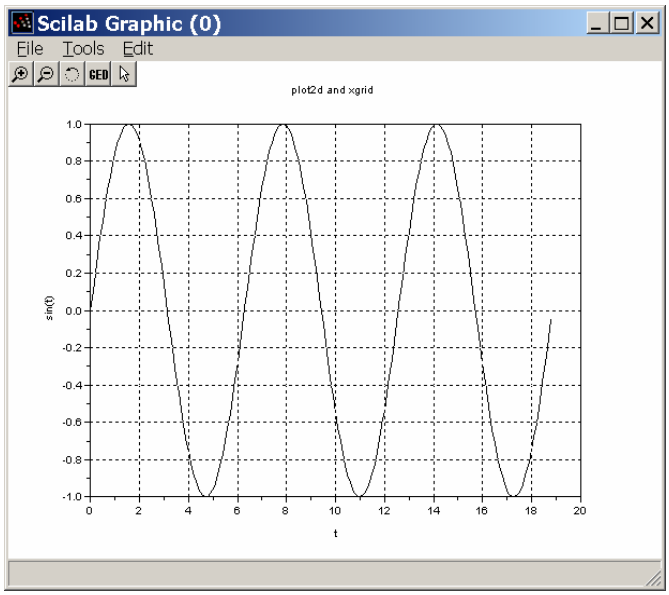
Sie müssen eine Gruppe wählen (und ev. noch Weitere im nachfolgenden Fenster) bis die gewünschte Demonstration abläuft.

Beispiel einer 2D-Grafik:

1. Im tkChoose-Fenster: Graphics wählen
2. Im Fenster *choose a demo*: *2D and 3D plots* wählen
3. Im nächsten Fenster: *plot2d* wählen

Nach dieser Prozedur erscheint das Fenster *Scilab Message* (Bild links). Im Fenster sind alle Befehle aufgeführt, welche die Grafik erzeugen. Diese Befehle können für eigene Anwendungen kopiert werden. Um die Grafik zu starten, müssen Sie den Button **Ok** drücken. Die Grafik erscheint (Bild rechts).





4.2.3 Scilab-Homepage

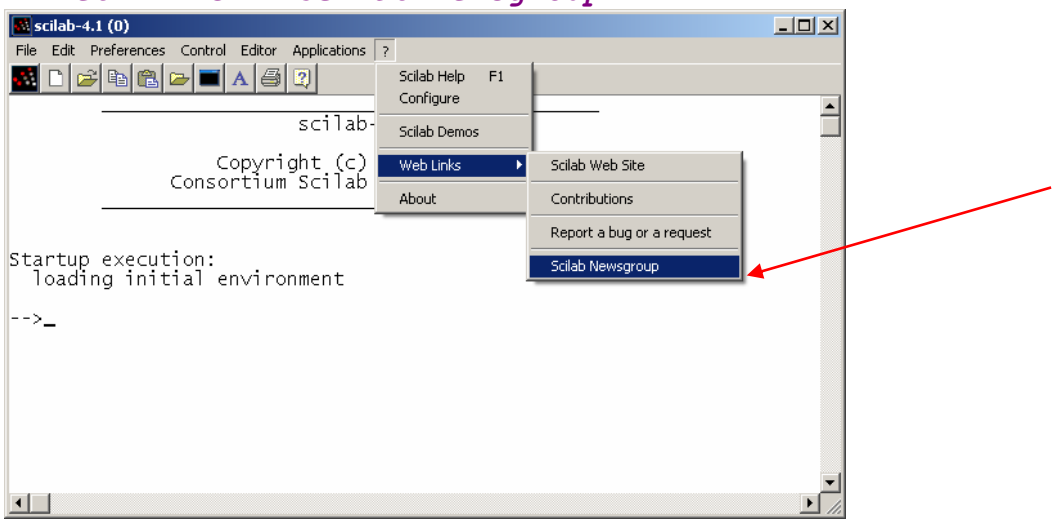
Auf der **Scilab**-Homepage *http://www.Scilab.org/*, die Sie aus dem Menü **? ⇨ Web Links ⇨ Scilab Web Site** direkt anwählen können, finden Sie unter den Menüpunkten

- Documentation & Support
- Books, Reports & Articles

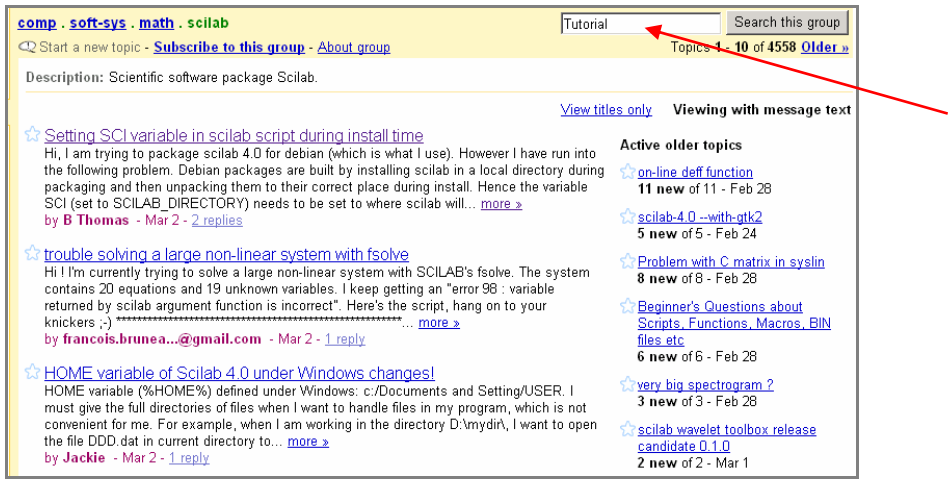
wertvolle Dokumentation (auch in Deutsch) und Hilfe.

4.2.4 Scilab-Newsgroup

Viele Antworten und Anregungen finden Sie in der **Scilab**-Newsgroup. Um diese Seite zu sehen, können Sie entweder *http://groups.google.com/group/comp.soft-sys.math.Scilab* oder Sie können direkt aus **Scilab** einsteigen über den Menü-Punkt **?** ***⇨ Web Links ⇨ Scilab Newsgroup***



Hier können Sie selbst Fragen stellen (und hoffen, dass sie beantwortet werden) oder nach einem Stichwort suchen. Im Beispiel sehen Sie die Rückmeldung nach der Eingabe des Suchbegriffs *Tutorial*

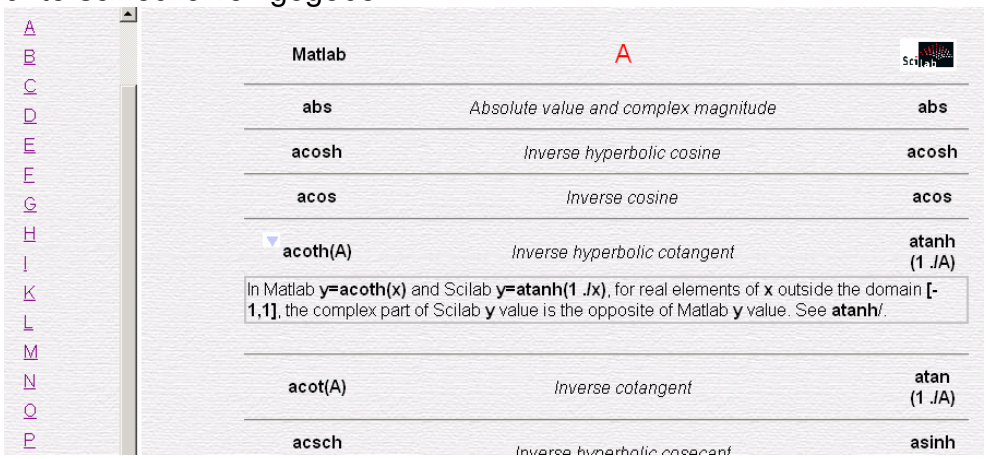


4.2.5 Für Matlab-Umsteiger

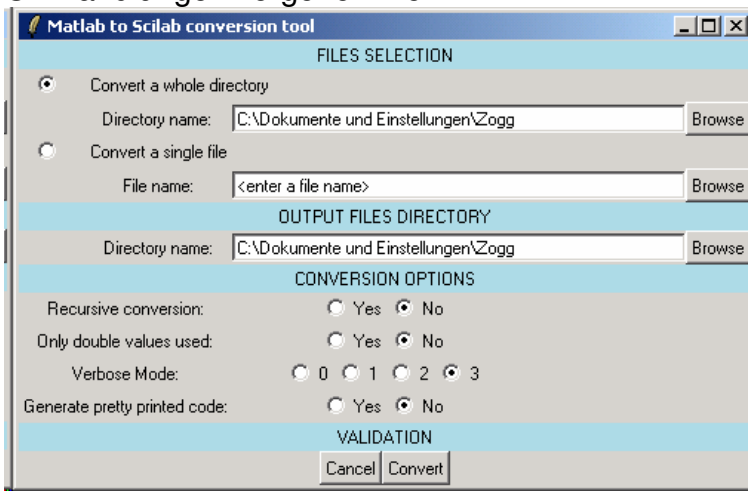
Wenn Sie Erfahrungen mit Matlab (einem kommerziellen Programm) haben, finden Sie auf der Homepage von **Scilab** *http://www.Scilab.org/* dem Button **Technical aera** den Menüpunkt **Documentation & support center** den weiteren Menü-Punkt **Matlab/Scilab functions** unter **Documentation:** Tipps und Tricks bezüglich Unterschied und Konvertierungsmöglichkeiten von Matlab zu **Scilab**.

Auf dieser Seite finden Sie in alphabetischer Reihenfolge alle Befehle aufgelistet. Sie sehen den Unterschied in der Schreibweise und allenfalls vertiefende Erklärungen. Zu diesen Erklärungen gelangen Sie durch Anklicken des blauen Dreiecks (▷), links des Matlab-Befehls.

Beispiel: Die inverse hyperbolische Cotangensfunktion wird in Matlab und in **Scilab** unterschiedlich eingegeben.



Hinweis: Scilab kann Matlab-Dateien oder Matlab-Verzeichnisse direkt von Matlab in **Scilab** konvertieren. Mit dem **Scilab**-Menüpunkt ***Applications ⇨ m2sci*** werden Umwandlungen vorgenommen.



5 Basiswissen zu Scilab (Getting started)

Um was geht es in diesem Kapitel?
Dieser Abschnitt gibt Ihnen eine Einführung in grundsätzliche Einstellungen und einfache Berechnungen, die Sie mit **Scilab** durchführen können.

Was werden Sie erreichen?
Sie werden einfache Berechnungen (ähnlich wie mit einem Taschenrechner) mit **Scilab** durchführen. Sie werden in der Lage sein, das Format der Ausgabe zu beeinflussen.

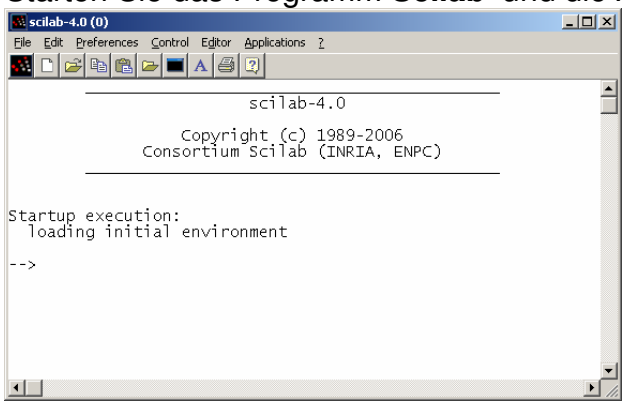
Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

x = a	ans	%pi	%e	%i
%inf	%T oder %t	%pi	+	-
*	/	sqrt(x)	x^y	“abcd“ ’abcd’
;	//	E, D	format(....)	abs(x)
acos(x)	acosh(x)	asin(x)	asinh(x)	atan(x)
atan(x,y)	atanh(x)	ceil(x)	cos(x)	cos(x)
cosh(x)	cotg(x)	exp(x)	floor(x)	imag(Z)
int(x)	log(x)	log10(x)	log2(x)	rand()
real(Z)	round(x)	sin(x)	sinh(x)	sqrt(x)
tan(x)	tanh(x)	hex2dec(“H“)	oct2dec(“O“)	base2dec(“Z“,b)
dec2hex(d)				

5.1 Starten

Starten Sie das Programm **Scilab** und die Arbeitsoberfläche erscheint:




```
                scilab-4.1

            Copyright (c) 1989-2006
        Consortium Scilab (INRIA, ENPC)


Startup execution:
  loading initial environment

-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

5.2 **Zuweisungen**, Konstanten, Basisoperationen und Datentypen

Eine **Zuweisung** (eine Variable erhält einen Wert) erfolgt bei **Scilab** auf klassische Weise:
Variable = Ausdruck

```
-->myVariable = 4 + 9
   myVariable  =

    13.

-->
```

Wird **keine Variable definiert**, d. h. nur eine Berechnung oder einen Wert eingegeben, wird eine interne (temporäre) Variable *ans* erzeugt. Die Variable *ans* behält ihren Wert:

a) bis sie durch eine neue Eingabe überschrieben wird
b) bis zu einer Neuformatierung der Ausgabe.

```
--> 4 + 9
   ans  =

    13.

--> 8 * 9
   ans  =

    56.

-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

- Die Variable *ans* kann zum Weiterrechnen verwendet werden.

Der Wert einer definierten Variablen wird durch das Eingeben des Variablennamens abgefragt.

```
-->myVariable
   myVariable  =

    13.

-->
```

Folgende **vordefinierten Konstanten** werden oft angewendet:

Symbol	Konstante
%pi	Pi, 3.1415926535897931…, π
%e	Eulersche Zahl, 2.7182818, e
%i	Imaginäre Einheit i (oder j), $\sqrt{-1}$
%inf (Infinite)	Unendlich ∞
%T oder %t (True)	Aussage: Wahr
%F oder %f (False)	Aussage: Falsch

Die einfachsten **Basisoperationen** sind:

Symbol	Operation
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
sqrt(x)	Quadratwurzel
x^y	x^y

Bei **Scilab** gibt es folgende **Datentypen**:

- Reelle Zahlen: Z. B. ***4.5678***
- Komplexe Zahlen: Z. B. ***4.32 + 5.6i***
- Zeichen (z. B. Buchstaben und Buchstabenfolgen), immer in Hochkommas (" oder ') eingebettet: Z. B. ***"Text"*** oder ***'Text'***
- Logischer Datentyp: ***T*** oder ***t*** (für True = Wahr), ***F*** oder ***f*** (für False = Falsch)

```
-->reelle_Zahl = 4.5658
 reelle_Zahl  =

    4.5658

-->komplexe_Zahl = 3.45 + 6.78*%i
 komplexe_Zahl  =

    3.45 + 6.78i

-->Zeichen = "Text"
 Zeichen  =

 Text

-->4.57 > 3.96  //Ist 4.57 grösser als 3.96?
 ans  =

  T

-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

5.3 Formatierung

Die **Anzeige** des Resultats wird mit einem Semikolon (`;`) **verhindert**.

```
--> myVar_2 = 23 + 78;

-->
```

Leerzeichen in der Berechnung beeinflussen das Resultat nicht, machen die Berechnung aber leserlicher!

```
--> my_Var_3=4+3;

--> my_Var_4  =  4 + 3;

-->
```

Mit `//` wird ein **Kommentar** eingeleitet.

```
--> // Dies ist eine Erklärung

-->
```

Zehnerpotenzen (z.B 10^3 oder 10^6) werden mit ***D*** oder ***E*** eingeben. Die Ausgabe erfolgt immer mit D.

```
-->A = 6.7E9
 A  =

    6.700D+09
```

Lange Eingabezeilen werden mit der Eingabe von drei Punkten und Enter (***...↵***) unterteilt.
Bsp.: myVar = sqrt(4 + 5 + 6 * (5/7)) + (sqrt(8) + log10(120) + (4^5 +%e^5 + log(56)))

```
--> myVar = sqrt(4 + 5 + 6 * (5/7)) + ...
-->         (sqrt(8) + log10(120) + ...
-->         (4^5 +%e^5 + log(56)))
 myVar  =

    1184.9911

-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Die **Darstellung der Ausgabe** (Festkomma oder Fliesskomma bzw. Anzahl der Ziffern) wird mit dem Befehl ***format("v", z)*** oder ***format("e", z)*** eingestellt. Die Default-Einstellung ist ***format("v", 10)***.
Erklärung der Parameter im Befehl ***format***:

- ***"v"*** steht für Festkomma
- ***"e"*** steht für Fliesskomma (E- bzw. D-Form)
- z (Zahl) steht für die Anzahl der angezeigten Ziffern (inkl. Vorzeichen und Dezimalexponenten)

```
-->0.0001/3
 ans  =

    0.0000333

-->format("e",18)

-->0.0001/3
 ans  =

    3.33333333333D-05

--> format("v",8)

-->0.0001/3
 ans  =

    0.00003

-->
```

Die gesamte **Bildschirm-Anzeige** wird **gelöscht** mit dem Befehl ***Preferences ⇨ Clear Command Window F2*** oder durch Drücken der Taste **F2**.

Bereits eingegebene **Befehle** (History) werden folgendermassen **wiederholt**:

- Mit dem Menü-Befehl ***Edit ⇨ History***
- Mit den Tasten *Pfeil-nach-oben* (⬆) oder *Pfeil-nach-unten* (⬇)

Die gesamte **Befehlsvorgeschichte** (History) wird mit dem Menü-Befehl ***Preferences ⇨ Clear History*** **gelöscht.**

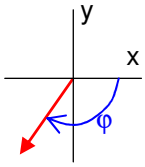
Um alle selbstdefinierten und vordefinierten **Variablen** bzw. **Konstanten aufzulisten,** wird der Befehl ***who*** eingegeben.

```
-->who
your variables are...

myVar   scicos_pal %helps   WSCI home  SCIHOME  PWD      TMPDIR  MSDOS   SCI
guilib  sparselib  xdesslib percentlib polylib  intlib   elemlib utillib statslib
alglib  siglib     optlib   autolib    roblib   soundlib metalib armalib
tkscilib  tdcslib   s2flib
mtlblib   %F       %T        %z        %s        %nan      %inf      COMPILER
%gtk      %pvm      %tk
 $         %t       %f        %eps      %io       %i        %e
 using       6433 elements  out of    5000000.
          and          49 variables out of       9231
```

Alle selbstdefinierten Variablen werden mit der Eingabe des Befehls ***clear*** **gelöscht.**

Eine einzelne selbstdefinierte Variable wird mit der Eingabe des Befehls ***clear myVar*** **gelöscht.** myVar steht für die zu löschende Variable.

5.4 Wichtige Funktionen

Befehl	Bedeutung	Beispiel
abs(x)	Absolutbetrag	Betrag = abs(4 + 4*%i)
acos(x)	Arcuscosinus	Wert8 = acos(1)
acosh(x)	Arcuscosinus hyperbolicus	Wert14 = acosh(%pi)
asin(x)	Arcussinus	Wert9 = asin(sqrt(1/2))
asinh(x)	Arcusinus hyperbolicus	Wert15 = asinh(2.5)
atan(x)	Arcustangens	Wert10 = atan(2)
atan(y,x)	 Der Winkel φ eines Ursprungsvektors mit den Komponenten y und x wird ausgegeben. Die Bestimmung von φ erfolgt im Bereich $-\pi$ bis $+\pi$.	phi = atan(-10,-5) → ergibt phi = - 2.0344439
atanh(x)	Arcustangens hyperbolicus	Wert16 = atanh(0.2)
ceil(x)	Ganzzahliger Anteil von x, auf die nächsthöhere ganze Zahl aufgerundet	c = ceil(4.1) → ergibt c = 5 d = ceil(4.9) → ergibt d = 5
cos(x)	Cosinus (Argument im Bogenmass)	Wert4 = cos(%pi)
cosh(x)	Cosinus hyperbolicus	Wert11 = cosh(4)
cotg(x)	Cotangens (Argument im Bogenmass)	Wert7 = cotg(2)
exp(x)	Exponentialfunktion e^x	Wert = exp(5)
floor(x)	Ganzzahliger Anteil von x, immer auf die nächst tiefere ganze Zahl abgerundet	a = floor(4.1) → ergibt a = 4 b = floor(4.9) → ergibt b = 4
imag(Z)	Imaginärteil der komplexen Zahl Z	Z = 2 + 5*%i r = imag(Z) → ergibt r = 5
int(x)	Ganzzahliger Anteil (abgeschnitten) von x	g = int(4.1) → ergibt g = 4 h = int(4.9) → ergibt h = 4
log(x)	Natürlicher Logarithmus (ln!)	Wert2 = log(56)
log10(x)	10er-Logarithmus	Wert3 = log10(1000)
log2(x)	Logarithmus mit der Basis 2	Bit_Breite = log2(4096)
rand()	Erzeugung einer Zufallszahl im Bereich 0 ...1	
real(Z)	Realteil der komplexen Zahl Z	Z = 2 + 5*%i i = real(Z) → ergibt i = 2
round(x)	Rundet auf die nächste ganze Zahl ab- oder auf	e = round(4.1) → ergibt e = 4 f = round(4.9) → ergibt f = 5
sin(x)	Sinus (Argument im Bogenmass)	Wert5 = sin(%pi / 4)
sinh(x)	Sinus hyperbolicus	Wert12 = sinh(5)
sqrt(x)	Quadratwurzel	Wert17 = sqrt(2)
tan(x)	Tangens (Argument im Bogenmass)	Wert6 = tan(1)
tanh(x)	Tangens hyperbolicus	Wert13 = tanh(0.5)

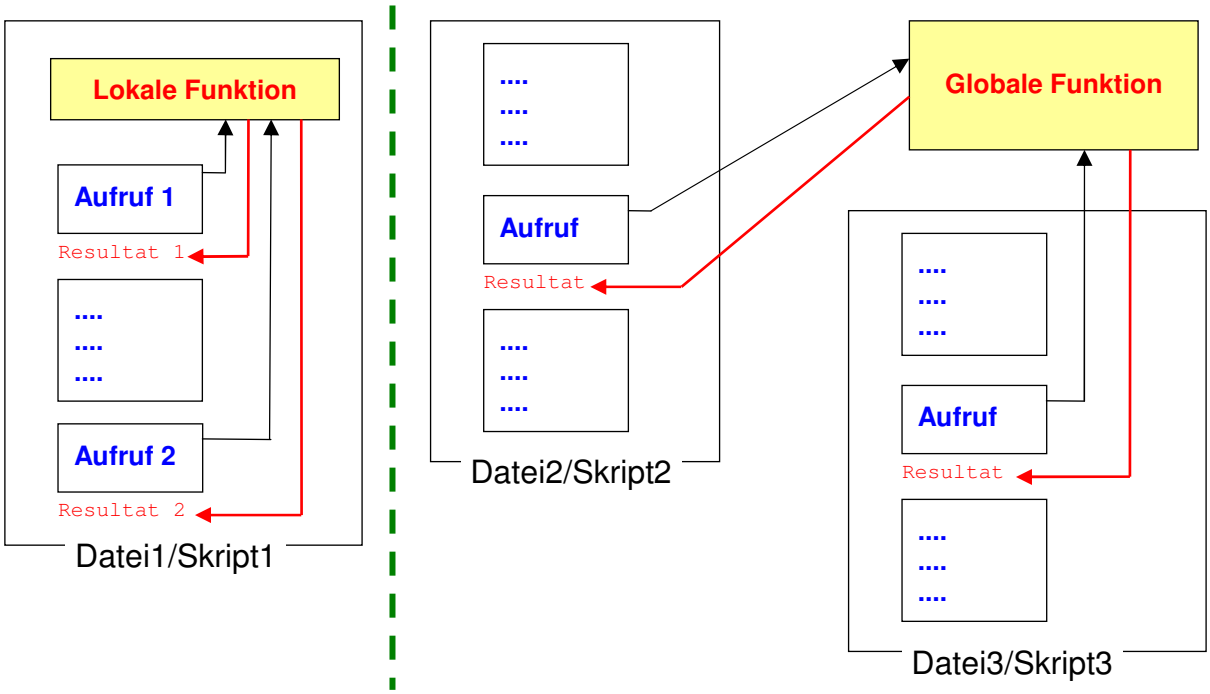
5.5 Umwandlung in verschiedene Zahlensysteme

Mit **Scilab** können Zahlen (zum Teil als Zeichen dargestellt) in verschiedene **Zahlensysteme umgewandelt** werden:

Befehl	Bedeutung	Beispiel
hex2dec("H")	Hexadezimalzahl H wird in eine Dezimalzahl umgewandelt. H ist als Zeichen dargestellt.	Dz1 = hex2dec("2A") → ergibt Dz1 = 42
oct2dec("O")	Octalzahl O wird in eine Dezimalzahl umgewandelt. O ist als Zeichen dargestellt.	Dz2 = oct2dec("17") → ergibt Dz2 = 15
base2dec("Z",b)	Die Zahl Z mit der Basis b wird in eine Dezimalzahl umgewandelt. Z ist als Zeichen dargestellt.	Dz3 = base2dec("1101",2) → ergibt Dz3 = 13
dec2hex(d)	Die Dezimalzahl d wird in eine Hexadezimalzahl (als Zeichen dargestellt) umgewandelt.	Hx1 = dec2hex(24455) → ergibt Hx1 = 5F87

5.6 Definieren einer lokalen Funktion

Eine **lokale Funktion** ist nur innerhalb der verwendeten Datei bzw. des verwendeten Skriptes gültig. Eine **globale Funktion** wird gespeichert und kann von einer beliebigen Datei bzw. einem beliebigen Skript aufgerufen werden. In diesem Abschnitt wird gezeigt, wie eine lokale Funktion erzeugt und angewendet wird. Im Abschnitt 3.9 wurde eine kurze Einführung zum Erstellen und Anwenden von globalen Funktionen vorgestellt. Im Abschnitt 10.6 wird vertieft auf das Prinzip von globalen Funktionen eingegangen.



Jede Funktion muss mit dem folgenden Befehl beginnen:
function [y1,y2,...,yn] = Funktionsname(x1,x2,...,xm)

wobei x1, x2, ..., xm Eingabeargumente, y1, y2, ..., yn Ausgabeargumente sind. Darauf folgen die Anweisungen, die zur Funktion gehören und aus den Eingabeargumenten die Ausgabeargumente berechnen.

Eine Funktion endet mit dem Schlüsselwort:
endfunction.

Beispiel 1: Funktion mit zwei Eingabe- und zwei Ausgabeargumenten.

```
function [x,y] = square(a,b)
    x = a² + b²
    y = a² - b²
endfunction
```

Der Aufruf (die Benutzung) der Funktion erfolgt:

1. entweder durch direkte Übergabe der Eingabeargumente
   [v,w] = square (3,2)
2. oder durch vorgängige Definition der Eingabeargumente erfolgen.
   c = 3
   d = 2
   [r,s] = square (c,d)

Die Bezeichnung der Ein- und Ausgabeargumente kann frei gewählt werden. Wichtig für die Zuweisung ist nur der Platz in der Aufzählung der Argumente.

```
--> function [x,y] = square(a,b)
-->  x = a^2 + b^2
-->  y = a^2 - b^2
--> endfunction

--> // direkte Übergabe der Eingabeargumente
--> [v,w] = square(3,2)
 w  =
    5.
 v  =
    13.

--> // vorgängige Definition der Eingabeargumente
-->c = 3;
-->d = 2;

--> [r,s] = square(c,d)
 s  =
    5.
 r  =
    13.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Beispiel 2: Funktion mit Vektoren als Eingabe- und Ausgabeargumente.

```
--> // Die lokale Funktion kreuzprodukt wird definiert

-->function [x]= kreuzprodukt(a,b)
-->  x(1) = (a(2)*b(3) - a(3)*b(2))
-->  x(2) = (a(3)*b(1) - a(1)*b(3))
-->  x(3) = (a(1)*b(2) - a(2)*b(1))
-->endfunction

-->a = [-2  5  8];
-->b = [7 -13 -5];

--> // Anwendung der Funktion kreuzprodukt
-->KP2 = kreuzprodukt(a,b)
 KP2  =
    79.
    46.
  - 9.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Beispiel 3: Funktionen können in einer Zeile geschrieben werden

```
--> function y = mysin(x), y = sin(x) + 3, endfunction

-->res = mysin(5)
 res  =
    2.0410757
```

Beispiel 4: Funktionen können verschachtelt sein

```
--> function y = verschacht(x)
-->       a = sin(x) + 2*4*%pi
-->            function y = schach(x),
-->                 y = x^2 - sqrt(x),
-->            endfunction
-->       y = schach(a)+3^2
-->endfunction

--> wert = verschacht(5)
 wert =
    588.45674
```

Beispiel 5: Berechnung der Fakultät

```
-->// n muss eine natürliche Zahl sein
--> function y = fakultaet(n)
--> y = prod(1:n)
--> endfunction

--> myfac = fakultaet(10)
 myfac  =
    3628800
```

5.7 Lösen eines bestimmten Integrals

Scilab kann das bestimmte Integral einer kontinuierlichen Funktion lösen.

Mit dem Befehl ***intg(a,b,f)*** wird das bestimmte Integral der Funktion f berechnet. Die untere Grenze des bestimmten Integrals ist a und die obere Grenze b.
Die Funktion f muss kontinuierlich sein!

Beispiel 1:

$$I = \int_0^{2\pi} f(x) \bullet dx$$

```
--> function y=f(x)
-->       y=x*sin(30*x)/sqrt(1-((x/(2*%pi))^2)),
--> endfunction

-->I = intg(0,2*%pi,f)
 I  =
  - 2.5432596
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Beispiel 2:

$$E = \int_0^{\pi} test(v) \bullet dv$$

```
-->function power = test(v)
-->  power = sin(v)
-->endfunction

-->E = intg(0, %pi, test)
 E  =
    2.
```

6 Arbeiten mit dem Editor Scipad

Um was geht es in diesem Kapitel?
Dieser Abschnitt erklärt Ihnen, was der Editor **Scipad** ist, den Zweck einer Skript-Datei, den Nutzen von **Scipad** und seine Anwendung.

Was werden Sie erreichen?
Sie werden eigene Skript-Dateien mit dem Editor **Scipad** durchspielen, abspeichern und diese ablaufen lassen.

Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

6.1 Was ist eine Skript-Datei und was nützt sie?

Eine Skript-Datei ist eine Liste von **Scilab**-Befehlen, welche auch Kommentare beinhalten kann. Die Ausführung erfolgt nach dem Schreiben aller Befehle.

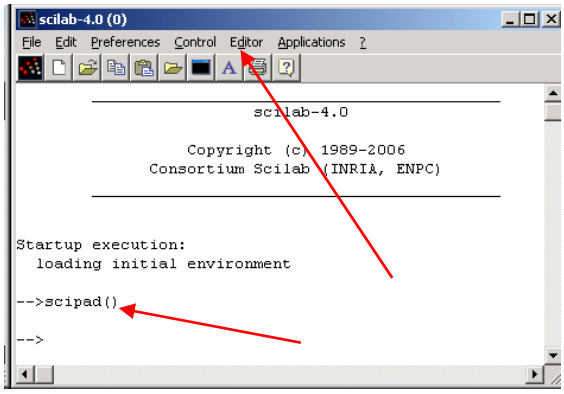
Beispiel einer Skript-Datei:

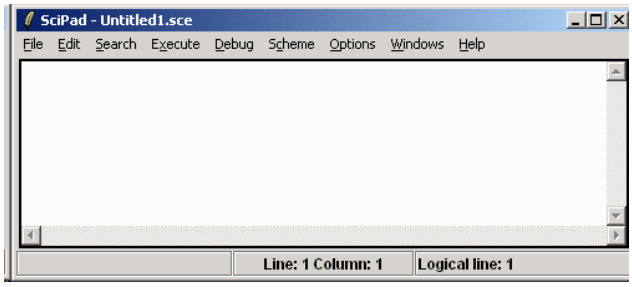
```
// Berechnung des Quadratwertes (QW)
// und des Kubikwertes (KW)
// für Zahlen (Z) im Bereich von A bis B
A = 0; B = 5;
for Z = A:1:B, QW = [Z Z^2 Z^3], end
```

In das Standard-Fenster von **Scilab** werden Befehle eingegeben, ausgeführt, eingegeben, ausgeführt usw. Merkt ein Anwender erst gegen Schluss der Berechnung, dass seine ersten Befehle fehlerhaft waren, muss er wieder sämtliche (korrigierten) Befehle eingeben und ausführen. Für solche Aufgaben ist es eleganter, zuerst alle Befehle (das Skript) zu schreiben und anschliessend alle gemeinsam auszuführen. Da das Skript abgespeichert wird, können einzelne Befehle korrigiert werden, ohne alles neu einzugeben.

6.2 Die Erstellung von Skript-Dateien mit dem Editor Scipad

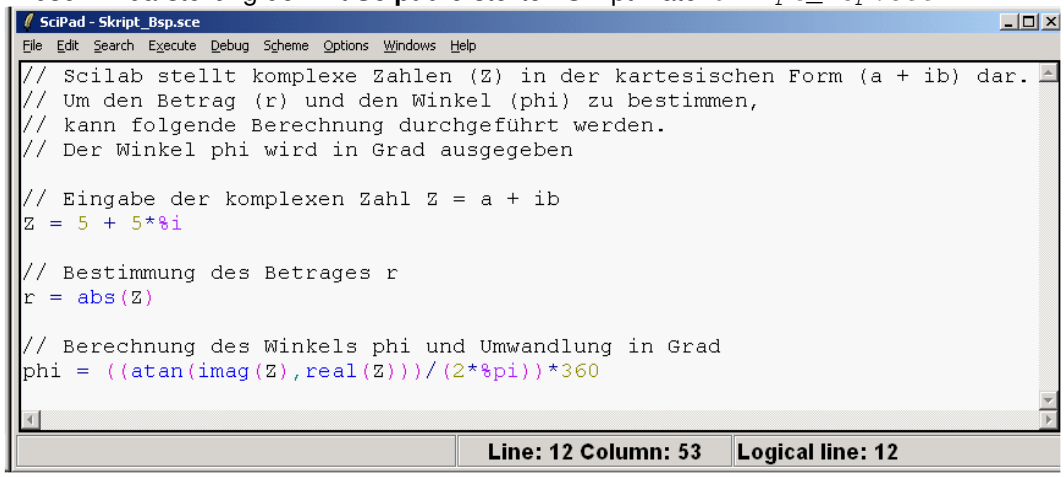
Ein zentrales Instrument von **Scilab** ist der integrierte Editor/Debugger **Scipad**. Mit ihm können Skript-Dateien komfortabel erzeugt und einfach getestet werden. Starten kann man diesen Editor im Hauptfenster mit dem Befehl ***scipad()*** oder im **Scilab**-Hauptmenü mit der Anwahl des Menüpunktes ***Editor***.





Mit dem Editor **Scipad** wird eine einfache Text-Datei erstellt. Speziell ist jedoch, dass der Editor die Syntax von **Scilab** kennt und diese, je nach Bedeutung, mit anderen Farben darstellt. Mit dem integrierten Debugger ist es auch möglich, Haltepunkte zu setzen und **Scilab** bis zu diesem Punkt rechnen zu lassen (dies ist nur für sogenannte .sci-Dateien, d.h. selbsterstellte Funktionen möglich).
Um die Skript-Datei auszuführen, muss sie immer zuerst abgespeichert werden. Die abgespeicherte Datei erhält die Endung *.sce*.

Bildschirmdarstellung der mit **Scipad** erstellten Skript-Datei *Skript_Bsp.sce*:



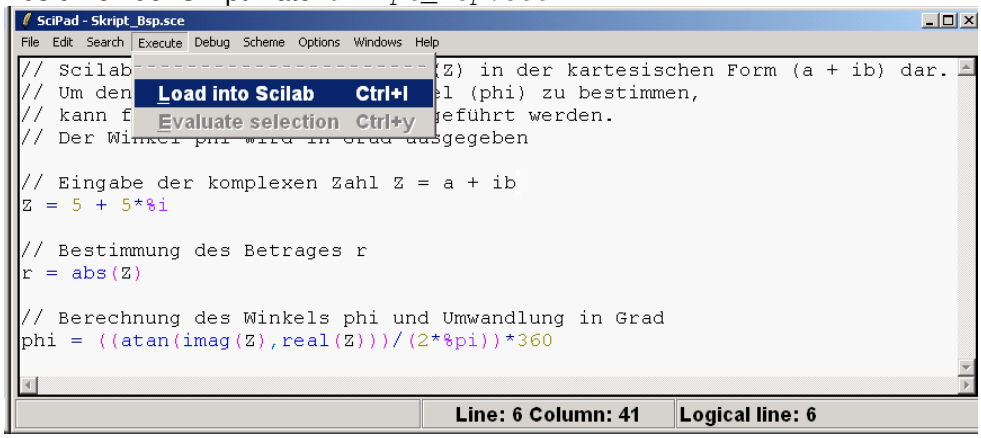
Skript-Datei *Skript_Bsp.sce*:

```
// Scilab stellt komplexe Zahlen (Z) in der kartesischen Form (a + ib) dar.
// Um den Betrag (r) und den Winkel (phi) zu bestimmen,
// kann folgende Berechnung durchgeführt werden.
// Der Winkel phi wird in Grad ausgegeben

// Eingabe der komplexen Zahl Z = a + ib
Z = 5 + 5*%i

// Bestimmung des Betrages r
r = abs(Z)

// Berechnung des Winkels phi und Umwandlung in Grad
phi = ((atan(imag(Z),real(Z)))/(2*%pi))*360
```

Wie in allen anderen Programmiersprachen ist es auch in **Scilab** unerlässlich, den Quelltext gut zu dokumentieren. Zeilen mit zwei Schrägzeichen (//) am Anfang gelten als Kommentar und werden bei der Ausführung ignoriert.

Um die Skript-Datei ausführen zu lassen, wird das Skript direkt aus dem Editor gestartet: Mit dem Menü-Befehl ***Execute ⇨ Load into Scilab***.

Ausführen der Skript-Datei *Skript_Bsp.sce*:



Die Ausgabe der Berechnung erfolgt im Arbeitsfenster von **Scilab**:

```
--> Z   =

    5. + 5.i
 r   =

    7.0710678
 phi   =

    45.
-->
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

7 Vektoren und Matrizen

Um was geht es in diesem Kapitel?
Dieser Abschnitt zeigt, wie Vektoren und Matrizen erzeugt und verarbeitet werden. Die wichtigsten Vektoren- und Matrizenoperationen werden eingehend erklärt.

Was werden Sie erreichen?
Sie werden in der Lage sein, Vektoren und Matrizen mit **Scilab** zu erzeugen, zu verändern und die wichtigsten Operationen durchzuführen.

Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

v = [a b c]	v = [a ; b , c]	v = [a b c]'	v = [a:b:c]	linspace(a,b,c)
logspace(a,b,c)	rand(1,a)	rand(b,1)	zeros(a,1)	ones(a,1)
size(v)	length(v)	v(a)	v(a:b)	v(a) = a
+	-	*	/	log(v)
log10(v)	exp(v)	sqrt(v)	^	norm(v)
sum(v)	.*	./	deff( )	m = [1 2; 3 4; 6 7]
rand(z,s)	zeros(z,s)	ones(z,s)	eye(z,s)	diag(m)
rank(m)	tril(m)	triu(m)	m(:)	sqrt(m)
exp(m)	sin(m)	./	.^	inv(m)
sum(m)	prod(m)	spec(m)	min(m)	max(m)
mean(m)				

7.1 Vektoren

7.1.1 Eingabe und automatische Erzeugung von Vektoren

Grundsätzlich unterscheidet man zwischen Zeilen- und Spaltenvektoren. Vektoren können auch als einzeilige oder einspaltige Matrizen betrachtet werden.

Eingabe eines Zeilenvektors: Die Elemente werden mit Leerschlag oder Komma getrennt.

```
-->a = [2 3 4]
 a  =

    2.    3.    4.

-->a = [2 , 3 , 4]
 a  =

    2.    3.    4.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Eingabe eines Spaltenvektors: Die Elemente werden mit einem Semikolon (`;`) getrennt.

```
-->c = [7 ; 8 ; 9]
 c  =

    7.
    8.
    9.
```

Umwandlung eines Zeilenvektors in einen Spaltenvektor (und umgekehrt): Mit dem Transpositionszeichen (`'`).

```
-->d = a'
 d  =

    2.
    3.
    4.
```

Vektorenreihen können generiert werden mit `[x:y:z]`: x ist der Anfangswert, y ist das Inkrement und z den nicht zu überschreitenden Wert. Wird y weggelassen, dann wird der Wert 1 angenommen.

```
-->a = [3:2:14]
 a  =

    3.    5.    7.    9.    11.    13.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Linear ansteigende (oder abfallende) Komponenten eines Vektors werden durch `linspace(x,y,z)` eingegeben: x ist die erste Komponente, y ist die letzte Komponente und z ist die Anzahl der Komponente innerhalb des Vektors.

```
-->b = linspace(3,13,5)
 b  =

    3.    5.5    8.    10.5    13.
```

Logarithmisches Ansteigen der Komponente eines Vektors werden mit ***logspace(x,y,z)*** generiert: x definiert die erste Komponente in der Form 10^x, y definiert die letzte Komponente in der Form 10^y und z die Anzahl der Komponente innerhalb des Vektors.

```
-->c = logspace(2,3,5)
 c  =

    100.    177.82794    316.22777    562.34133
```

Zeilenvektoren mit x Zufallskomponente werden mit ***rand(1,x)*** erzeugt.

```
-->d = rand(1,3)
 d  =

    0.4818509    0.2639556    0.4148104
```

Spaltenvektoren mit x Zufallskomponente werden mit ***rand(x,1***) erzeugt.

```
-->e = rand(3,1)
 e  =

    0.2806498
    0.1280058
    0.7783129
```

Zeilenvektoren mit x Nullen oder x Einsen werden mit ***zeros(1,x***) oder ***ones(1,x)*** erzeugt.
Spaltenvektoren mit x Nullen oder x Einsen werden mit ***zeros(x,1)*** oder ***ones(x,1***) erzeugt.

```
-->a = zeros(7,1);

-->b = ones(7,1);

-->Ausgang = [a b]
 Ausgang  =

    0.    1.
    0.    1.
    0.    1.
    0.    1.
    0.    1.
    0.    1.
    0.    1.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Ein **Vektor mit komplexen Komponente** kann durch Kombination der obigen Möglichkeiten eingegeben werden.

```
-->KV_1 = [(2:5)*(1 + 1.5*%i)]
 KV_1  =

    2. + 3.i    3. + 4.5i    4. + 6.i    5. + 7.5i

-->KV_2 = [(4 + 5*%i) ; (9 - 7*%i)]
 KV_2  =

    4. + 5.i
    9. - 7.i
```

7.1.2 Extraktion aus Vektoren und Veränderung von Komponente

Die Dimension eines Vektors x in der Darstellung Zeile-Spalte wird mit dem Befehl ***size(x)*** bestimmt.

```
-->x_1 = [1 2 3];

-->x_2 = x_1';

-->size(x_1)
 ans  =

    1.    3.

-->size(x_2)
 ans  =

    3.    1.
```

Die **Anzahl der Komponente eines Vektors** V wird mit ***length(V)*** ermittelt.

Einzelne Komponente werden aus einem Vektor extrahiert durch Angabe der Position.

```
-->x = [1:3:17]
 x  =

    1.    4.    7.    10.    13.    16.

-->x(4) //Komponente 4 wird extrahiert!
 ans  =

    10.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Mehrere nacheinander stehende Komponente des Vektors y werden mit ***y(a:b)*** **extrahiert:** a ist die erste Komponente und b die letzte. Die Extraktion ergibt wieder einen Vektor.

```
-->y = [1:3:17]
 y  =

    1.    4.    7.    10.    13.    16.

-->vkt = y(2:5) //Komponente 2 bis 5 werden extrahiert!
 vkt  =

    4.    7.    10.    13.
```

Mehrere Komponenten können in beliebiger Reihenfolge aus einem Vektor extrahiert werden.

```
-->y = [1:3:17]
 y  =

    1.    4.    7.    10.    13.    16.

-->a = y([2,5,6]) //Komponente 2, 5 und 6 werden extrahiert!
 a  =

    4.    13.    16.
```

Eine Komponente Vektors kann verändert werden.

```
-->y = [1:3:17]  //vorher
 y  =

    1.    4.    7.    10.    13.    16.

-->y(3) = 314; //Die dritte Komponente wird zu 314

-->y             //nachher
 y  =

        1.    4.    314.    10.    13.    16.
```

Vektoren können zusammengesetzt und ergänzt werden.

```
-->a = [1 2 3];

-->b = [5 8 9];

-->c = [a 314 b] //Vektor c wird neu zusammengesetzt!
 c  =

    1.    2.    3.    314.    5.    8.    9.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

7.1.3 Operationen mit Vektoren

Bei allen Operationen mit Vektoren muss darauf geachtet werden, dass die **Formate korrekt** gewählt sind. Z.B können keine Spalten- und Zeilenvektoren miteinander addiert werden. Wird dies trotzdem versucht, gibt **Scilab** eine Fehlermeldung aus.

```
-->a = [1 2 3]; //Zeilenvektor 1

-->b = [6 7 8]; //Zeilenvektor 2

-->c = [4 5 9]; //Spaltenvektor 2

-->x = a + c    //Versuch einer Addition!
          !--error 8
inconsistent addition

-->y = a * b    //Versuch einer Multiplikation!
          !--error 10
inconsistent multiplication
```

Addition und Subtraktion von Vektoren: Vektoren gleichen Formats können addiert (+) und subtrahiert (-) werden.

```
-->a = [1 2 3];

-->b = [5 8 9];

-->c = a + b //Addition!
 c  =

    6.    10.    12.

-->d = b - a //Subtraktion!
 d  =

    4.    6.    6.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Ein **Vektor wird mit einer reellen oder komplexen Zahl multipliziert**.

```
-->Z = 4 + 5*%i;  // Komplexe Zahl

-->V = [5 ; 6 ; 7]; // Spaltenvektor

-->M = V * Z
 M  =

    20. + 25.i
    24. + 30.i
    28. + 35.i
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Die **Operationen** ***log***, ***log10***, ***exp***, ***sqrt*** und **^** werden immer pro Komponente ausgeführt.

```
-->a = [1 4 9];

-->b = sqrt(a)
 b  =

    1.    2.    3
```

Der **Betrag (Länge)** eines Vektors wird mit dem Befehl ***norm(x)*** ermittelt.

Zur Erinnerung: $Betrag\ von\ \vec{a} = \left|\vec{a}\right| = \sqrt{a_1{}^2 + a_2{}^2 + a_3{}^2 + a_4{}^2 + ...}$

```
-->a = [1 4 9];

-->B = norm(a)
 B  =

    9.8994949
```

Die Summe aller Komponenten eines Vektors wird mit ***sum(x)*** ermittelt.

```
-->a = [1 4 9];

-->S = sum(a)
 S  =

    14.
```

Zwei **Vektoren werden komponentenweise miteinander multipliziert oder dividiert** mit den sogenannten Dot-Befehlen (***.****) oder (***./***).

```
-->a = [1 2 3];

-->b = [5 8 9];

-->X = a .* b
 X  =

    5.    16.    27.
```

Das **Skalarprodukt** (Zeilenvektor mal Spaltenvektor gleicher Länge) eines Vektors wird mit der Operation * berechnet.

```
-->a = [1 2 3];

-->b = [5 ; 8 ; 9];

-->SKP = a * b
 SKP  =

    48.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Scilab hat keine eigene Funktion für das **Kreuzprodukt (= Vektorprodukt)**. Wir können selbst eine lokale Funktion *kreuzprodukt* mit dem Befehl ***function*** / ***endfunction*** erstellen, um das Kreuzprodukt zweier Vektoren mit je drei Elementen zu berechnen.

```
--> // Die lokale Funktion kreuzprodukt wird definiert

-->function [x]= kreuzprodukt(a,b)
-->  x(1) = (a(2)*b(3) - a(3)*b(2))
-->  x(2) = (a(3)*b(1) - a(1)*b(3))
-->  x(3) = (a(1)*b(2) - a(2)*b(1))
-->endfunction

--> // 1. Anwendung der Funktion kreuzprodukt
-->KP1 = kreuzprodukt([-2 5 8] , [7 -13 -5])
KP1  =
    79.
    46.
  - 9.

-->a = [-2  5  8];
-->b = [7 -13 -5];

--> //2. Anwendung der Funktion kreuzprodukt
-->KP2 = kreuzprodukt(a,b)
 KP2  =
    79.
    46.
  - 9.
```

7.2 Matrizen

Eine Matrix besteht aus einer Anzahl Zeilen und Spalten. Die Position eines Elementes (= Zelle) innerhalb der Matrix wird mit der Angabe von Zeilen- und Spaltennummer definiert.

	Spalten →					
Zeilen ↕	1,1	1,2	1,3	1,4	1,5	1,6
	2,1	2,2	2,3	2,4	2,5	2,6
	3,1	3,2	3,3	3,4	3,5	3,7

Wird obige Matrix mit M bezeichnet, kann der Inhalt des Elements in der 2. Zeile und 4. Spalte folgendermassen eingegeben werden: M(2,4) = 314

Ist die Anzahl der Zeilen gleich der Anzahl der Spalten, dann ist die Matrix quadratisch:

1,1	1,2	1,3
2,1	2,2	2,3
3,1	3,2	3,3

7.2.1 Eingabe und automatische Erzeugung von Matrizen

Eingabe einer Matrix: Spalten werden durch Leerschlag oder Komma getrennt, eine neue Zeile wird mit einem Semikolon gestartet.

```
-->M1 = [1 2 3; 4 5 6; 7 , 8 , 9]
 M1  =

    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Transposition einer Matrix: Die Zeilen werden in Spalten (und umgekehrt) mit dem Transpositionszeichen (') umgewandelt.

```
-->M2 = M1'
 M2  =

    1.    4.    7.
    2.    5.    8.
    3.    6.    9.

-->M3 = [1 2 3 ; 4 5 6]'
 M3  =

    1.    4.
    2.    5.
    3.    6.
```

Matrizen können **aus Vektoren zusammengesetzt** werden:

```
-->V1 = [1 2 3];

-->V2 = [5 6 7];

-->M4 = [V1 ; V2]
 M4  =

    1.    2.    3.
    5.    6.    7.

//oder

-->M5 = [V1' V2']
 M5  =

    1.    5.
    2.    6.
    3.    7.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Automatische Erzeugung von Matrizen

- Matrix **mit Zufallswerten**: ***rand(z,s)***, (z = Anzahl der Zeilen, s = Anzahl der Spalten)
- Matrix **mit 1 (Einsen):** ***ones(z,s)***.
- Matrix **mit 0 (Nullen):** ***zeros(z,s)***. Hinweis: In der linearen Algebra ist eine **Nullmatrix** eine Matrix, deren Elemente alle 0 sind.
- Matrix **mit 1 (Einsen) in der Diagonale**: ***eye(z,s)***. Hinweis: In der linearen Algebra ist eine **Einheitsmatrix** (auch **Identitätsmatrix** genannt) eine quadratische Matrix, deren Hauptdiagonale nur aus Einsen besteht. Alle anderen Elemente sind 0.

```
--> M6 = rand(3,2)
 M6  =
    0.2113249    0.3303271
    0.7560439    0.6653811
    0.0002211    0.6283918

--> M7 = ones(2,3)  //Nullmatrix
 M7  =
    1.    1.    1.
    1.    1.    1.

--> M8 = zeros(3,2)
 M8  =
    0.    0.
    0.    0.
    0.    0.

--> M9 = eye(4,4)  //Einheitsmatrix
 M9  =
    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
    0.    0.    0.    1.
```

Beliebige **Reihen** können aus **Kombinationen von Vektoren** erzeugt werden. Hinweis: Je nach Form der Matrix sollte die Anzahl der Zeilen bzw. der Spalten bei jedem Vektor übereinstimmen:

```
--> M12 = [[2:3:15] ; linspace(100,87,5) ;logspace(2,3,5)]
 M12  =
    2.      5.          8.          11.         14.
    100.    96.75       93.5        90.25       87.
    100.    177.82794   316.22777   562.34133   1000.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Matrizen mit **komplexen Elementen**:

```
--> Z1 = [(1 + 1*%i) (2 + 2*%i) ; (3 + 3*%i) (4 + 4*%i)]
 Z1  =
    1. + i      2. + 2.i
    3. + 3.i    4. + 4.i
```

Eine **Leere Matrix** wird mit ***[]*** erzeugt:

```
--> M_leer = []
 M_leer  =
     []
```

7.2.2 Extraktion aus Matrizen und Veränderung von Elementen

Die **Dimension einer Matrix** wird mit ***size(M)*** abgefragt. Als Antwort erhält man die Anzahl der Zeilen und Spalten:

```
--> size(M12)
 ans  =
    3.    5.
```

Die **Anzahl der Elemente** einer Matrix wird mit ***length(M)*** geliefert:

```
-->  M14 = ones(8,9);

--> Anz = length(M14)
 Anz  =
    72.
```

Ein Element aus einer Matrix **extrahieren:**

```
--> M15 = rand (5,3)
 M15  =
    0.8497452    0.6623569    0.2312237
    0.6857310    0.7263507    0.2164633
    0.8782165    0.1985144    0.8833888
    0.0683740    0.5442573    0.6525135
    0.5608486    0.2320748    0.3076091

--> //Das Element in der 3. Zeile und 2. Spalten wird dargestellt
-->El_4 = M15(3,2)
 El_4  =
    0.1985144
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Die **Hauptdiagonale einer Matrix** als Spaltenvektor mit ***diag(M)*** extrahieren.

```
-->M16 = eye(6,6);

-->SV = diag(M16)
 SV  =
    1.
    1.
    1.
    1.
    1.
    1.
```

Der **Rang einer Matrix** mit ***rank(M)*** abfragen. Hinweis: Der Rang einer Matrix M ist die Maximalzahl linear unabhängiger Zeilenvektoren bzw. die Maximalzahl linear unabhängiger Spaltenvektoren.

```
--> M = rand(6 , 9);

--> M_rang = rank(M)
 M_rang  =
    6.
```

Extraktion eines **Zeilenvektors** oder eines **Spaltenvektors** aus einer Matrix:

```
--> M17 = [1 2 3 ;  4 5 6 ; 7 8 9]
 M17  =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.

// Die 3. Zeile wird als Zeilenvektor extrahiert
-->Z3 = M17(3 ,:)
 Z3  =
    7.    8.    9.

// Die 2. Spalte wird als Spaltenvektor extrahiert
-->Z4 = M17(: , 2)
 Z4  =
    2.
    5.
    8.
```

Extraktion einer Teilmatrix TM aus einer Matrix M:
Ausgewählt werden die Elemente aus der Matrix M, welche sich im Kreuzungspunkt der definierten Zeilen und Spalten befinden. Die Zeilen und Spalten werden folgendermassen definiert: ***M([Def. der Zeilen] [Def. der Spalten])***.

```
--> // Zuerst wird eine Zufallsmatrix M erstellt

--> M = rand(4 ,5)
 M  =
    0.2113249    0.6653811    0.8782165    0.7263507    0.2312237
    0.7560439    0.6283918    0.0683740    0.1985144    0.2164633
    0.0002211    0.8497452    0.5608486    0.5442573    0.8833888
    0.3303271    0.6857310    0.6623569    0.2320748    0.6525135

--> // Beispiel 1
--> // Auswahl der gemeinsamen Elemente der Zeilen 2, 4, 1 und der Spalten 1, 5

--> TM1 = M([2 4 1],[1 5])
 TM1  =
    0.7560439    0.2164633
    0.3303271    0.6525135
    0.2113249    0.2312237

--> // Beispiel 2
--> // Auswahl der Elemente der Zeilen 4 und 1 von sämtlichen Spalten

--> TM2 = M([4 1],[:])
 TM2  =
    0.3303271    0.6857310    0.6623569    0.2320748    0.6525135
    0.2113249    0.6653811    0.8782165    0.7263507    0.2312237
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Extraktion der unteren (triangle lower) und **der oberen** (triangle upper) **Dreiecksmatrix** mit ***tril(M)*** und ***triu(M)***, d.h. Extraktion unterhalb und oberhalb der Diagonalen (inkl. Diagonale). Die nichtverwendeten Zellen werden mit Nullen ausgefüllt.

```
--> M17 = [[1:5] ; [2:6] ; [3:7]]
 M17  =
    1.    2.    3.    4.    5.
    2.    3.    4.    5.    6.
    3.    4.    5.    6.    7.

--> ob_Dreieck = triu(M17)
 ob_Dreieck  =
    1.    2.    3.    4.    5.
    0.    3.    4.    5.    6.
    0.    0.    5.    6.    7.

--> unt_Dreieck = tril(M17)
 unt_Dreieck  =
    1.    0.    0.    0.    0.
    2.    3.    0.    0.    0.
    3.    4.    5.    0.    0.
```

Änderung der Komponenten einer Matrix:

```
--> M20 = [1 2 3 ;  4 5 6 ; 7 8 9]
 M20  =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.

--> // Element in der 3. Zeile und 2. Spalte erhält den Wert 314
--> M20(3,2) = 314
 M20  =
    1.    2.      3.
    4.    5.      6.
    7.    314.    9.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Matrizen aus Teilmatrizen bzw. Vektoren erweitern und zusammenstellen. Hinweis: die neu erstellte Matrix muss eine rechteckige bzw. quadratische Form.

```
--> TM_1 = [1 2 3 ; 4 5 6]
 TM_1  =
    1.    2.    3.
    4.    5.    6.

-->TM_2 = [45 46 ; 47 48]
 TM_2  =
    45.    46.
    47.    48.

-->Vk_1 = [83 84 85 86 87]
 Vk_1  =
    83.    84.    85.    86.    87.

-->neu_Matrix = [TM_1  TM_2 ; Vk_1]
 neu_Matrix  =
    1.     2.     3.     45.    46.
    4.     5.     6.     47.    48.
    83.    84.    85.    86.    87.
```

Erzeugung einer Diagonalmatrix mit *diag(v)*: Die Diagonalelemente werden aus einem Vektor übernommen, die Dimension der Matrix entspricht derjenigen des Vektors.

```
--> v = [9 8 7 6];

-->M11 = diag(v)
 M11  =
    9.    0.    0.    0.
    0.    8.    0.    0.
    0.    0.    7.    0.
    0.    0.    0.    6.
```

Die **Dimension einer Matrix ändern** mit ***matrix(M, z, s)***. Hinweis: die Anzahl der Koeffizienten darf sich nicht ändern d. h, das Produkt Zeilen mal Spalten muss gleich bleiben.

```
--> Pr = rand(3,4) // Zufallsmatrix erzeugen
 Pr  =
    0.2693125    0.9184708    0.2639556    0.1280058
    0.6325745    0.0437334    0.4148104    0.7783129
    0.4051954    0.4818509    0.2806498    0.2119030

--> // Die Matrix Pr wird umgestaltet zu 2 Zeilen und 6 Spalten
-->Pr_neu = matrix(Pr,2,6)
 Pr_neu  =
    0.2693125    0.4051954    0.0437334    0.2639556    0.2806498    0.7783129
    0.6325745    0.9184708    0.4818509    0.4148104    0.1280058    0.2119030
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Eine Matrix **in einen Spaltenvektor umwandeln:**

```
--> M13 = [1 2 3; 4 5 6]
 M13  =

    1.    2.    3.
    4.    5.    6.

--> V13 = M13(:)
 V13  =

    1.
    4.
    2.
    5.
    3.
    6.
```

7.2.3 Operationen mit Matrizen

Wichtig: Bei allen Operationen mit Matrizen muss darauf geachtet werden, dass die Dimension (Anzahl der Zeilen und Anzahl der Spalten) korrekt gewählt sind.

Addition und Subtraktion von Matrizen untereinander: Es können nur Matrizen mit der gleichen Anzahl von Zeilen und Spalten addiert bzw. subtrahiert werden.

```
--> M1 = rand(3,3);

--> M2 = rand (3,3);

--> M = M1 + M2
 M  =
    0.3119692    1.382483     0.7588441
    1.2475557    1.7321743    1.2657905
    0.7427394    0.9104238    1.0361258
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Addition und Subtraktion aller Elemente einer Matrix mit einem Skalar (reale oder komplexe Zahl)

```
--> M1 = rand(2,3);

--> M_neu = M1 + (3 + 6*%i)
 M_neu  =
    3.9488184 + 6.i    3.3760119 + 6.i    3.2615761 + 6.i
    3.3435337 + 6.i    3.7340941 + 6.i    3.4993494 + 6.i
```

Multiplikation von Matrizen untereinander: Um zwei Matrizen zu multiplizieren, müssen die Spaltenanzahl der linken mit der Zeilenanzahl der rechten Matrix übereinstimmen. Hat die linke Matrix das Format a x b und die rechte Matrix das Format b x c, dann hat das Resultat das Format a x c.

```
--> M_links = ones(3 , 7)
 M_links  =
    1.    1.    1.    1.    1.    1.    1.
    1.    1.    1.    1.    1.    1.    1.
    1.    1.    1.    1.    1.    1.    1.

-->M_rechts = rand(7 , 2)
 M_rechts  =
    0.2638578    0.0485566
    0.5253563    0.6723950
    0.5376230    0.2017173
    0.1199926    0.3911574
    0.2256303    0.8300317
    0.6274093    0.5878720
    0.7608433    0.4829179

-->M_mult = M_links * M_rechts
 M_mult  =
    3.0607126    3.2146479
    3.0607126    3.2146479
    3.0607126    3.2146479
```

Multiplikation oder Division einer Matrize mit einem Skalar (real oder komplex):

```
--> M = ones(3 , 4);

--> M_skal = M * (3 + 2*%i)
 M_skal  =

    3. + 2.i    3. + 2.i    3. + 2.i    3. + 2.i
    3. + 2.i    3. + 2.i    3. + 2.i    3. + 2.i
    3. + 2.i    3. + 2.i    3. + 2.i    3. + 2.i
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Elementenweise Operationen mit ***sqrt(x)***, ***exp(x)***, ***sin(x)*** etc. (siehe Funktionen, Abschnitt 5.4):

```
--> M = [1 2 3; 4 5 6; 7 8 9];

--> M_Wurzel = sqrt(M)
 M_Wurzel  =
    1.          1.4142136   1.7320508
    2.          2.236068    2.4494897
    2.6457513   2.8284271   3.
```

Elementenweise Multiplikation, Division und Potenzierung zweier Matrizen gleicher Dimension mit dem Dot-Operator (**.**):

```
--> M1 = [1 2 3 ; 4 5 6]
 M1  =
    1.    2.    3.
    4.    5.    6.

-->M2 = [11 12 13 ; 14 15 16]
 M2  =
    11.    12.    13.
    14.    15.    16.

-->M_mul = M1 .* M2
 M_mul  =

    11.    24.    39.
    56.    75.    96.
-->M_div = M2 ./ M1
 M_div  =
    11.    6.    4.3333333
    3.5    3.    2.6666667

-->M_pot = M1 .^ M2
 M_pot  =
    1.          4096.       1594323.
    2.684D+08   3.052D+10   2.821D+12
```

Inverse Matrix. Hinweis: Die inverse Matrix ist nur für quadratische Matrizen definiert.

```
--> A = [1 2 ; 3 4];
--> A_inv = inv(A)
 A_inv  =
  - 2.     1.
    1.5   - 0.5
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Eigenwert einer quadratischen Matrix mit ***spec(M)***. Hinweis: Eine Zahl x heisst Eigenwert einer Matrix M, wenn es einen Vektor v gibt, so dass M•v = x•v gilt. Der Aufruf von ***spec(M)*** liefert die Eigenwerte der Matrix M als Spaltenvektor.

```
--> M = rand(5,5)
 M  =
    0.7901072    0.9229532    0.5175369    0.8433565    0.4920584
    0.9808542    0.1000746    0.8325452    0.0748595    0.7489608
    0.8187066    0.4678218    0.6104832    0.8532815    0.9414957
    0.4256872    0.3950498    0.1871112    0.0124590    0.2124056
    0.2461561    0.0366117    0.0189575    0.1867539    0.579502

--> spec(M)
 ans  =
    2.4584555
  - 0.3384634 + 0.1755862i
  - 0.3384634 - 0.1755862i
    0.0358589
    0.2752384
```

Eigenvektoren einer Matrix M (wenn sie diagonalisierbar ist) erhält man mit ***bdiag(M)***.

Summe oder Produkt aller Elemente einer Matrix mit ***sum(M)*** bzw.***prod(M)***.
Summe oder Produkt aller Spaltenelemente einer Matrix mit ***sum(M, "r")*** bzw. ***prod(M, "r")***.
Summe oder Produkt aller Zeilenelemente einer Matrix mit ***sum(M, "c")*** bzw. ***prod(M, "c")***.

```
--> M = [1 2 3; 4 5 6]
 M  =
    1.    2.    3.
    4.    5.    6.

-->sum(M)
 ans  =
    21.

-->sum(M, "r")
 ans  =
    5.    7.    9.

-->prod(M, "c")
 ans  =
    6.
    120.
```

Minimum und Maximum eines Vektors oder einer Matrix mit ***min(M)*** oder ***max(M)***.
Minimum und Maximum aller Spaltenelemente mit ***min(M, "r")*** bzw. ***max(M, "r")***.
Minimum und Maximum aller Zeilenelemente mit ***min(M, "c")*** bzw. ***max(M, "c")***.

```
--> M = [1 2 3; 4 5 6];

--> min(M)
 ans  =
    1.

--> min(x, "c")
 ans  =
    1.
    4.

--> max(M, "r")
 ans  =
    4.    5.    6.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Um die **Position** (Zeile, Spalte) **des Maximums bzw. Minimums** gleichzeitig mit dem Wert zu erhalten, muss folgende Eingabe getätigt werden: ***[M_min min_pos] = min(M)*** bzw. ***[M_max max_pos] = max(M)***.

```
--> M = [1 2 3; 4 5 6];

--> [M_min min_pos] = min(M)
 min_pos  =
    1.    1.
 M_min  =
    1.

--> [M_max max_pos] = max(M)
 max_pos  =
    2.    3.
 M_max  =
    6.
```

Der **Mittelwert aller Elemente einer Matrix** bestimmt man mit ***mean(M)***.
Der **Mittelwert,** spaltenweise oder zeilenweise berechnet, wird mit ***mean(M, "r")*** bzw. ***mean(M, "c")*** bestimmt.

```
--> M = [1 2 3; 4 5 6]
 M  =
    1.    2.    3.
    4.    5.    6.

--> mean(M)
 ans  =
    3.5

--> mean(M, "r")
 ans  =
    2.5    3.5    4.5

--> mean(M, "c")
 ans  =
    2.
    5.
```

8 Komplexe Zahlen in Scilab

Um was geht es in diesem Kapitel?
Dieser Abschnitt zeigt, wie komplexe Zahlen erzeugt und verarbeitet werden. Die Umwandlungsmöglichkeiten und Extraktion einzelner Komponenten werden eingehend erklärt.

Was werden Sie erreichen?
Sie werden in der Lage sein, komplexe Zahlen mit **Scilab** zu erzeugen und die wichtigsten Operationen auszuführen.

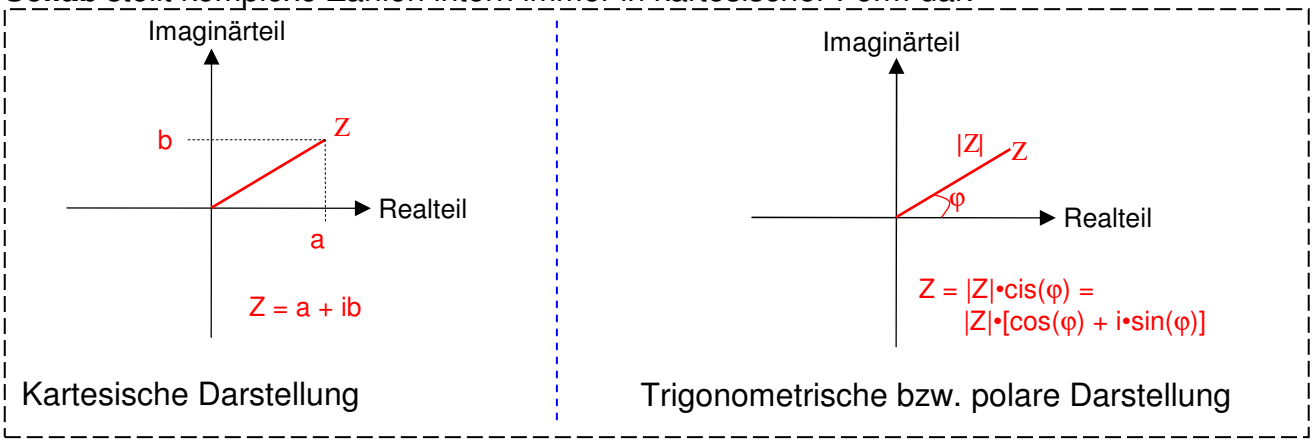
Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

Z = a + %i*b	real(Z)	imag(Z)	abs(Z)	conj(Z)
atan()	ones(z,s)	sin()	cos()	

8.1 Einführung

Scilab stellt komplexe Zahlen intern immer in kartesischer Form dar.



Eingabe einer komplexen Zahl **in der kartesischen Form** (a + ib): ***Z_1 = 1 + 1*%i***

Eingabe einer komplexen Zahl in der trigonometrischen Form (|Z|•[cos(φ) + i•sin(φ)]). Der Winkel wird in Radiant eingegeben. Die Ausgabe erfolgt in kartesischer Form.

```
--> Z_2 = 7 * (cos(0.3) + %i*sin(0.3))
 Z_3  =
    6.6873554 + 2.0686414i
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Einzelne Grössen einer komplexen Zahl werden mit folgenden Befehlen erhalten:

- Der **Realteil** mit ***real(Z)***
- Der **Imaginärteil** mit ***imag(Z)***
- Der Betrag mit ***abs(Z)***

- Die **konjugiert komplexe Zahl** mit ***conj(Z)***
- Das **Argument** (im Bereich von ±180°) mit ***((atan(imag(Z),real(Z)))/(%pi))*180***

```
--> Z_3 = -1 + %i;

-->real_teil = real(Z_3)
 real_teil  =
  - 1.

-->imag_Teil = imag(Z_3)
 imag_Teil  =
    1.

-->Betrag = abs(Z_3)
 Betrag  =
    1.4142136

-->conj_Z = conj(Z_3)
 conj_Z  =
  - 1. - i

-->Winkel = ((atan(imag(Z_3),real(Z_3)))/(%pi))*180
 Winkel  =
    135.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Matrizen und Vektoren können mit **komplexen Komponenten bzw. Elementen** erstellt werden:

```
--> Z1 = [(1 + 1*%i) (2 + 2*%i) ; (3 + 3*%i) (4 + 4*%i)]
 Z1  =
    1. + i      2. + 2.i
    3. + 3.i    4. + 4.i
```

Vektoren und Matrizen können mit einer komplexen Zahl multipliziert werden:

```
--> Z = 4 + 5*%i;

--> M = ones(3 , 4);

--> Z_M = Z * M
 Z_M  =
    4. + 5.i    4. + 5.i    4. + 5.i    4. + 5.i
    4. + 5.i    4. + 5.i    4. + 5.i    4. + 5.i
    4. + 5.i    4. + 5.i    4. + 5.i    4. + 5.i
```

9 Lösen von Gleichungen

Um was geht es in diesem Kapitel?
Dieser Abschnitt zeigt, wie Gleichungssysteme aufgestellt und gelöst werden. Die wichtigsten Regeln zum Lösen von linearen und nichtlinearen Gleichungssystemen werden eingehend erklärt.

Was werden Sie erreichen?
Sie werden in der Lage sein, lineare und nichtlineare Gleichungssysteme mit **Scilab** aufzustellen und die einzelnen Lösungen zu berechnen.

Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

linsolve(KM, -LV)	\	inv(m)	fsolve()	function
endfunction	sqrt()	sin()		

9.1 Lineares Gleichungssystem mit reellen Koeffizienten

Scilab kennt verschiedene Möglichkeiten, um lineare Gleichungssysteme zu lösen. Die übersichtlichsten Methoden teilen das System in eine Koeffizientenmatrix KM und einen Spaltenvektor SV ein. Mit den **Scilab**-Befehlen ***linsolve()*** und **** wird der Wert der verschiedenen Unbekannten berechnet.

Beispiel:
Um vier unbekannte Grössen (w, x, y, z) zu bestimmen, sind folgende vier Gleichungen gegeben:

1. Gleichung: 4w + 13x -5y + 6z = 12
2. Gleichung: 6w -7x + 2y – 2z = 0
3. Gleichung: -2w + 9x – 13y + 15z = -11
4. Gleichung: w – 4x + 8y + z = 3

Die vier obigen Gleichungen können in Koeffizientenmatrix (KM), Lösungsvektor (LV) und Spaltenvektor (SV) aufgeteilt werden:

$$\begin{bmatrix} 4 & 13 & -5 & 6 \\ 6 & -7 & 2 & -2 \\ -2 & 9 & -13 & 15 \\ 1 & -4 & 8 & 1 \end{bmatrix} \bullet \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 12 \\ 0 \\ -11 \\ 3 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} 4 & 13 & -5 & 6 \\ 6 & -7 & 2 & -2 \\ -2 & 9 & -13 & 15 \\ 1 & -4 & 8 & 1 \end{bmatrix}}_{KM} \bullet \underbrace{\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}}_{LV} = \underbrace{\begin{bmatrix} 12 \\ 0 \\ -11 \\ 3 \end{bmatrix}}_{SV}$$

Die Koeffizientenmatrix KM ist quadratisch (q x q), der Spaltenvektor SV ist ein Spaltenvektor mit genau q Elementen.

Es gibt bei **Scilab** verschiedene Möglichkeiten, um den Wert der vier Unbekannten (LV) zu bestimmen. Drei Methoden sind hier vorgestellt:

- ***KM\SV***
- ***inv(KM) * SV***
- ***linsolve(KM,-SV)***

```
--> KM = [4 13 -5 6; 6 -7 2 -2; -2 9 -13 15; 1 -4 8 1]
 KM  =
    4.    13.  - 5.     6.
    6.  - 7.     2.   - 2.
  - 2.    9.   - 13.    15.
    1.  - 4.     8.     1.

-->SV
 SV  =
    12.
    0.
  - 11.
    3.

-->Loesung_1 = KM\SV              // 1. Lösungsweg
 Loesung_1  =
    0.9528660
    1.2380776
    0.9415687
  - 0.5331049

-->Loesung_2 =inv(KM)*SV          // 2. Lösungsweg
 Loesung_2  =
    0.9528660
    1.2380776
    0.9415687
  - 0.5331049

-->Loesung_3 =linsolve(KM ,-SV)   // 3. Lösungsweg
 Loesung_3  =
    0.9528660
    1.2380776
    0.9415687
  - 0.5331049
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

9.2 Lineares Gleichungssystem mit komplexen Koeffizienten

Grundsätzlich werden lineare Gleichungssysteme mit komplexen Zahlen gleich wie Gleichungssysteme mit reellen Koeffizienten gelöst (siehe Abschnitt 9.1). Da aber oft die Aufstellung des Systems Sorge bereitet, wird sie in diesem Kapitel behandelt.

Beispiel:
Um vier unbekannte komplexe Grössen (w, x, y, z) zu bestimmen, sind folgende vier komplexe Gleichungen gegeben:

1: (100 + 101.25*%i)* w + (-101.25*%i)*x + (-207.35*%i)* y = (1 - 2.0735*%i)
2: (-101.25*%i)*w + (150 + 239.48*%i)*x + (207.35*%i)*y = (2.0735*%i)
3: (-207.35*%i)*w + (207.35*%i)*x + (220 + 502*%i)*y + (-295.31*%i)*z = (-2 + 2.0735*%i)
4: (-295.35*%i)*y + (136.16*%i)* z = 2

Die vier obigen Gleichungen können in Koeffizientenmatrix (KM), Lösungsvektor (LV) und Spaltenvektor (SV) aufgeteilt werden:

$$\begin{bmatrix} 100 + 101.25i & -101.25i & -207.35i & 0 \\ -101.25i & 150 + 239.48i & 207.35i & 0 \\ -207.35i & 207.35i & 220+502i & -295.31i \\ 0 & 0 & -295.31i & 136.16i \end{bmatrix} \bullet \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 - 2.0735i \\ 2.0735i \\ -2+2.0735i \\ 2 \end{bmatrix}$$

KM • LV = SV

Die komplexe Koeffizientenmatrix KM ist quadratisch (q x q), der komplexe Spaltenvektor SV hat genau q Elementen.

```
--> KM= [(100 + 101.25*%i) (-101.25*%i) (-207.35*%i) (0) ; ...
-->      (-101.25*%i) (150 + 239.48*%i) (207.35*%i) (0) ; ...
-->      (-207.35*%i) (207.35*%i) (220 + 502*%i) (-295.31*%i) ; ...
-->      (0) (0) (-295.35*%i) (136.16*%i)]
 KM  =
    100. + 101.25i  - 101.25i         - 207.35i         0
  - 101.25i           150. + 239.48i    207.35i         0
  - 207.35i           207.35i           220. + 502.i  - 295.31i
    0                 0               - 295.35i         136.16i

-->SV = [(1 - 2.0735*%i) ; (2.0735*%i) ; (-2 + 2.0735*%i) ; (2)]
 SV  =
    1. - 2.0735i
    2.0735i
  - 2. + 2.0735i
    2.

-->Loesung_1 = KM \ SV                      // 1. Lösungsweg
 Loesung_1  =
  - 0.0019859 - 0.0020949i
    0.0050170 - 0.0032267i
    0.0055701 + 0.0063332i
    0.0120824 - 0.0009511i

-->Loesung_2 = inv(KM) * SV                 // 2. Lösungsweg
 Loesung_2  =
  - 0.0019859 - 0.0020949i
    0.0050170 - 0.0032267i
    0.0055701 + 0.0063332i
    0.0120824 - 0.0009511i

-->Loesung_3 = linsolve(KM, -SV)            // 3. Lösungsweg
 Loesung_3  =
  - 0.0019859 - 0.0020949i
    0.0050170 - 0.0032267i
    0.0055701 + 0.0063332i
    0.0120824 - 0.0009511i
```

Prompt
Eingabe
Resultat
//Kommentar

9.3 Nichtlineares Gleichungssystem

Es gibt verschiedene Möglichkeiten, um mit **Scilab** nichtlineare Gleichungen zu lösen. Nicht immer führen sie zu einer Lösung. Oft müssen mehrere Versuche gemacht werden, bis ein plausibles Resultat vorliegt. In der Regel muss der Anwender die Lösung im Voraus abschätzen. Je genauer seine Schätzung ist, umso schneller wird das richtige Resultat ermittelt.

Die unbekannten Grössen des unlinearen Gleichungssystems werden mit dem Befehl ***fsolve*** berechnet. Eingegeben werden die Schätzwerte für alle unbekannten Grössen und eine externe Funktion, welche die verschiedenen Gleichungen definiert. Die gesuchten Grössen und die Schätzwerte sind als Spaltenvektoren zu definieren.

```
[X] = fsolve ([Schätzwert_X1, Schätzwert_X2, Schätzwert_X3],
Funktion_für_X)
```

Die Gleichungen der externen Funktion müssen in die Form `... = 0` gebracht werden.

Beispiel:
Die folgende Gleichung sei gegeben:

$$a \cdot X(1)^4 + b \cdot X(2)^2 - \sqrt{X(3)} \cdot c = d \cdot X(4)$$

Die unbekannten vier Grössen sind X(1), X(2), X(3) und X(4), welche zum Spaltenvektor [X] zusammengefasst werden.
Obige Gleichung muss umgestellt werden.

$$a \cdot X(1)^4 + b \cdot X(2)^2 - \sqrt{X(3)} \cdot c - d \cdot X(4) = 0$$

Beispiel 1 zur Anwendung des Befehls *fsolve*
Gegeben sind folgende drei nichtlinearen Gleichungen (R, S und T sind unbekannt):

$$\sqrt{(R-a)^2 + (S-b)^2} + T \cdot c = d$$

$$\sqrt{(R-e)^2 + (S-f)^2} + T \cdot c = g$$

$$\sqrt{(R-h)^2 + (S-j)^2} + T \cdot c = k$$

Schritt 1: Definition der externen Funktion (hier genannt Position)
Die drei unbekannten Grössen werden im Spaltenvektor [X] zusammengefasst

$$\begin{bmatrix} R \\ S \\ T \end{bmatrix} = \begin{bmatrix} X(1) \\ X(2) \\ X(3) \end{bmatrix} = [X]$$

Die Gleichungen werden in die Form `... = 0` umgestellt.
Der Ausdruck = 0 muss nicht geschrieben werden.

```
function Position = Gleichungen(X)
      Position = [sqrt((X(1) - a)^2 + (X(2) - b)^2) + X(3)*c - d; ...
                  sqrt((X(1) - e)^2 + (X(2) - f)^2) + X(3)*c - g; ...
                  sqrt((X(1) - h)^2 + (X(2) - j)^2) + X(3)*c - k]
endfunction
```

Schritt 2: Berechnung der drei Unbekannten, zusammengefasst im Vektor [X].
Für die drei unbekannten Grössen R, S, T (bzw. X(1), X(2), X(3) wird ein Schätzwert SW mit eingegeben.

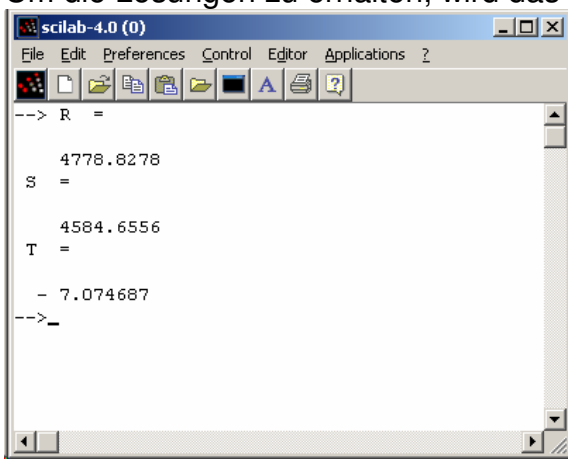
X = ***fsolve***([SW_R; SW_S; SW_T], Gleichungen);

Schritt 3: Die drei gesuchten Grössen R, S und T werden angezeigt
R = X(1)
S = X(2)
T = X(3)

Anschauungsbeispiel: Der Lösungsalgorithmus wird im Editor **Scipad** eingegeben.

```
clear

// Lösung eines nichtlinearen Gleichungssystems mit drei Unbekannten
// ================================================================

// Gegebene drei Gleichungen
// =========================
// Gleichung 1: sqrt[(R-a)² + (S-b)²] + T*c = d
// Gleichung 2: sqrt[(R-e)² + (S-f)²] + T*c = g
// Gleichung 3: sqrt[(R-h)² + (S-j)²] + T*c = k

// Konstanten
// ==========
a = 0;      b = 0;      c = 300;    d = 4500;   e = 0;
f = 10000;  g = 5100;   h = 10000;  j = 10000;  k = 5400;

// Gesucht
// =======
// R, S und T

// Lösung
// ======
// 1. Die drei (= n) Gleichungen müssen in die Form ... = 0 gebracht werden
// 2. Die Null wird bei der Eingabe nicht mitgeschrieben
// 3. Eine Funktion (hier Position) in Abhängigkeit von X(n) wird definiert
//    (n ist die Anzahl der Unbekannten).
// 4. Die Funktion (als Spaltenvektor) beinhaltet die drei Gleichungen.
// 5. Die drei (= n) unbekannten Grössen (R, S, T) werden zu X(1), X(2) bzw. X(3) umbenannt.

function Position = Gleichungen(X)
  Position =  [ sqrt((X(1)-a)^2 + (X(2)-b)^2) + X(3)*c - d; ...
                sqrt((X(1)-e)^2 + (X(2)-f)^2) + X(3)*c - g; ...
                sqrt((X(1)-h)^2 + (X(2)-j)^2) + X(3)*c - k ]
endfunction

// Das Gleichungssystem wird aufgelöst.
// Das Ergebnis wird als Spaltenvektor für X(1), X(2) bzw. X(3) ( = X_pos, Y_pos, d_t) berechnet.
// Bei der Auflösung muss ein Schätzwert (Spaltenvektor) für (R, S, T) mit definiert werden,
// hier z. B. [2000, 3000, 4].

X = fsolve([2000; 3000; 4], Gleichungen);

// Die Resultate werden wieder den gesuchten Grössen zugewiesen

R = X(1)
S = X(2)
T = X(3)
```

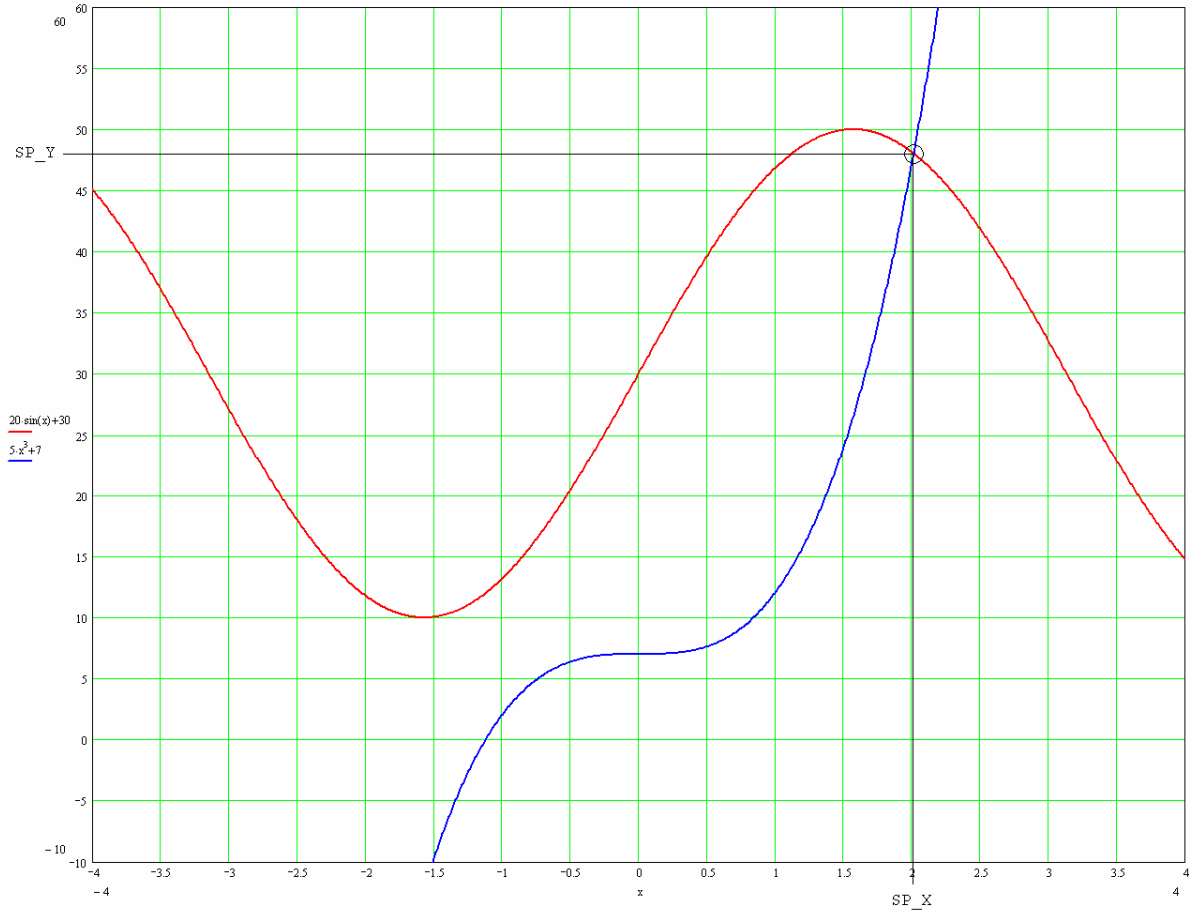
Um die Lösungen zu erhalten, wird das Skript ausgeführt:



Beispiel 2: Anwendung des Befehls *fsolve*.

Gegeben sind zwei Funktionen y_1 (x) und y_2(x).
Y_1(x) = 20sin(x) + 30
Y_2(x) = $5x^3$ + 7

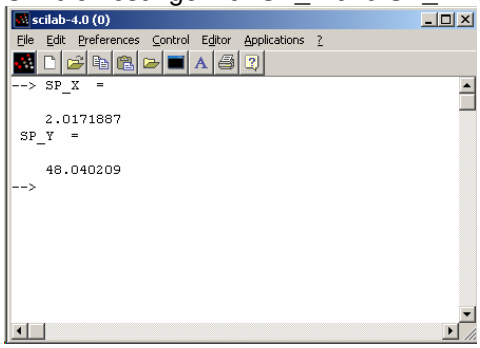
Gesucht ist der Schnittpunkt (SP_X, SP_Y) beider Funktionen.



Der Lösungsalgorithmus wird im Editor **Scipad** eingegeben:

```
clear

// Lösung einer nichtlinearen Gleichungssystems mit zwei Unbekannten (SP_X,
SP_Y)
// ============================================================================

// Gegeben sind zwei Funktionen y_1(X) und y_2(X).
// Gesucht ist der Schnittpunkt (SP_X und SP_Y) beider Funktionen.

// y_1(x) = 20sin(x) + 30
// y_2(X) = 5x³ + 7


// Konstanten
// ==========
a = 20; b = 30; c = 5; d = 7;


// Gesucht
// =======
// Der Schnittpunkt [SP_X, SP_Y]


// Lösung
// ======
// 1. Die zwei Gleichungen müssen in die Form ... = 0 gebracht werden.
// 2. Die Null wird bei der Eingabe nicht mit geschrieben.
// 3. Eine Funktion (hier Schnittpunkt) in Abhängigkeit von Q wird definiert.
// 4. Die Funktion (als Spaltenvektor) beinhaltet die zwei Gleichungen.
// 5. Die zwei unbekannten Grössen (SP_X und SP_Y) werden zu Q(1) und X(2) umbenannt.

function Schnittpunkt = Gleichungen(Q)
  Schnittpunkt =  [ a*sin(Q(1)) + b - Q(2); ...
                    c*Q(1)^3 + d - Q(2)]
endfunction

// Das Gleichungssystem wird aufgelöst.
// Das Ergebnis wird als Spaltenvektor für Q(1) und Q(2) berechnet.
// Bei der Auflösung muss ein Schätzwert (Spaltenvektor) für (SP_X und SP_Y) mit definiert werden,
hier z. B. [5, 30].

Q = fsolve([5; 30], Gleichungen);

SP_X = Q(1)
```

Um die Lösungen für SP_X und SP_Y zu erhalten, wird das Skript ausgeführt:



10 Grafische Darstellung

Um was geht es in diesem Kapitel?
Dieser Abschnitt zeigt, wie Funktionsgrafen in einer Grafik aufgezeichnet werden. Die wichtigsten Möglichkeiten zum Anzeigen von zweidimensionalen Grafen werden eingehend erklärt.

Was werden Sie erreichen?
Sie werden in der Lage sein, Grafiken zu generieren und ansprechend zu gestalten.

Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

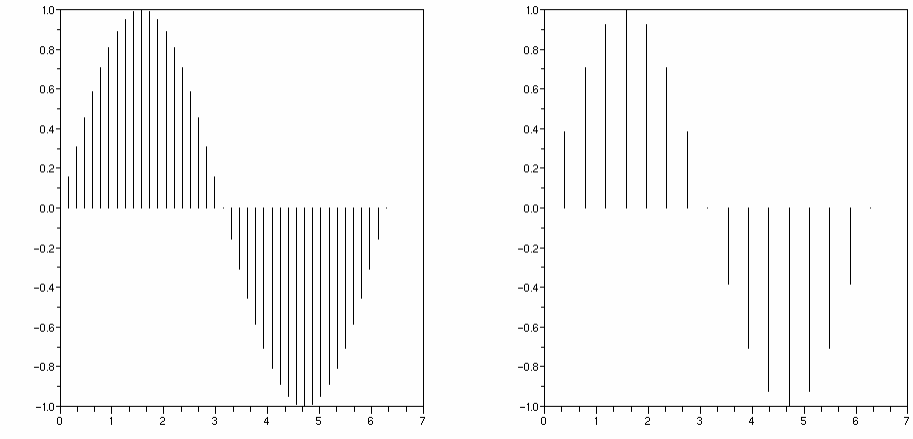
clear	clf()	xdel	plot2d()	plot2d2()
plot2d3()	plot2d(4)	xtitle("text")	subplot()	style = i
rect=[a, b, c, d]	logflag="xy"	frameflag=j	axesflag=n	xgrid()
leg="text@text"	logspace()	sin()	abs()	

10.1 Einleitung

10.1.1 Definition der x-Achse

Um eine zweidimensionale Grafik (x-y-Grafik) zu zeichnen, muss zuerst der Bereich auf der Abzisse (x-Bereich) als Vektor definiert werden. Zur Definition gehört die x-Schrittweite zum Aufzeichnen und Berechnen der Werte der x-Achse.

Beispiel einer x-Achse beginnend bei 0, endend bei 2π und mit einer Schrittweite von:
a) π/20 (linkes Diagramm):`x = [0: (%pi/20):2*%pi];`
b) π/8 (rechtes Diagramm): `x = [0: (%pi/8):2*%pi];`



10.1.2 Initialisierung

Bevor gezeichnet wird, sollten die alten Grafiken gelöscht und Variablen rückgestellt werden. Der Initialisierungsvorgang kann folgendermassen aussehen:

```
clear          // löscht alle Variablen
clf()          // löscht die aktuelle bisherige Grafik
xdel           // schliesst alle offenen Grafikfenster
```

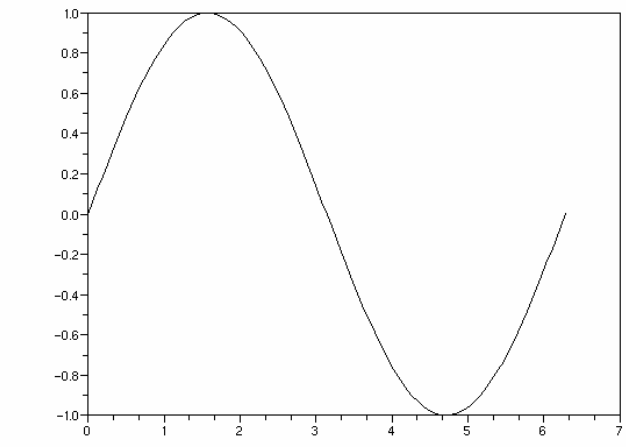
10.1.3 Einfache Grafik mit plot2d()

Nach Initialisierung und Definition der x-Achse wird der Graf der Funktion aufgezeichnet:

```
-->clear  // löscht alle Variablen
-->clf()  // löscht die aktuelle bisherige Grafik
-->xdel   // schliesst alle offenen Grafikfenster

-->// Definition der x-Achse und der y-Werte
-->x = [0: (%pi/20):2*%pi];

-->// Definition der Funktion y = f(x)
-->y = sin(x);

-->// Befehl zum Aufzeichnen von y = f(x)
-->plot2d(x,y)
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Scilab öffnet ein grafisches Fenster und zeichnet die Funktion:



Die Funktion `y = f(x)` kann direkt im Zeichnungsbefehl `plot2d()` definiert werden. Die beiden Befehle

```
-->y = sin(x);
-->plot2d(x,y)
```

können zusammengefasst werden zu

```
-->plot2d(x,sin(x))
```

10.1.4 Darstellung mehrerer Funktionen im gleichen Koordinatensystem

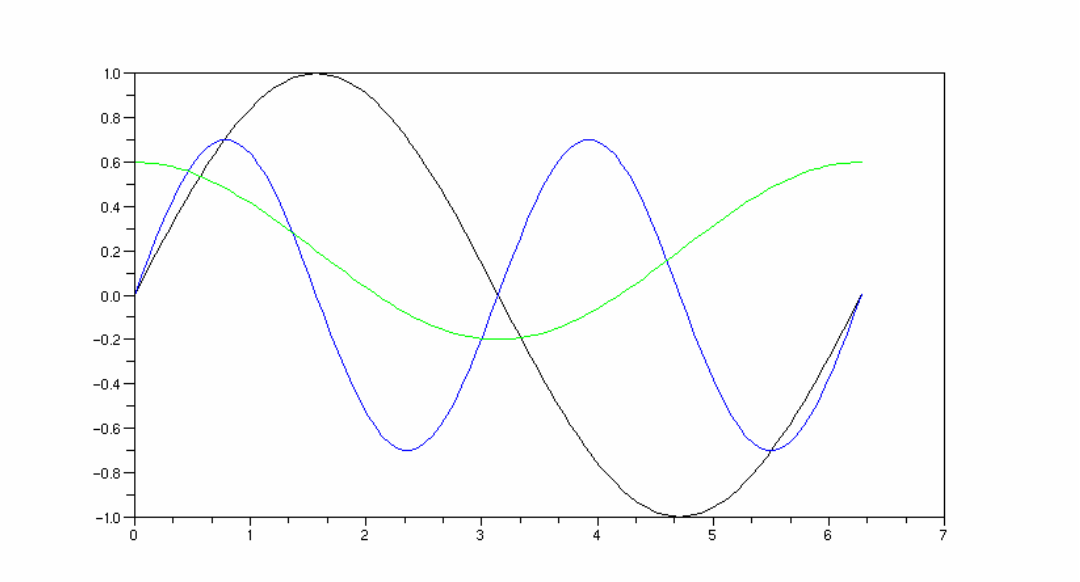
Mehrere Kurven können innerhalb der gleichen x- und y-Achsen gleichzeitig aufgezeichnet werden.
Werden mehrere Funktionen gleichzeitig dargestellt, dann muss die x**-Achse als Spaltenvektor** und die verschiedenen **Funktionen als Zeilenvektor** definiert werden.

Beispiel: gleichzeitige Darstellung von drei Funktionen im gleichen Koordinatensystem.

```
-->clear
-->clf()
-->xdel

-->// Definition der X-Achse (Spaltenvektor)
-->x = [0:(%pi/50):2*%pi]';


-->// Definition der drei Funktionen
-->y1 = sin(x);              // Funktion 1
-->y2 = 0.7*sin(2*x);        // Funktion 2
-->y3 = 0.4*cos(x) + 0.2;    // Funktion 3

-->// Plotbefehl
-->// Die einzelnen Funktionen werden als Zeilenvektor eingegeben
-->plot2d(x,[y1 y2 y3])
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Scilab öffnet ein grafisches Fenster und zeichnet die Grafen drei Funktionen. Die einzelnen Farben werden automatisch zugewiesen.



10.2 Die verschiedenen plot2d()-Varianten und die Darstellung in verschiedenen Grafen

Scilab kennt verschiedene Darstellungsmöglichkeiten und kann gleichzeitig Grafen mit unterschiedlichen Skalierungen darstellen.

- ***plot2d()***: Kurvendarstellung, der Graf wird als kontinuierliche Linie gezeichnet.

- ***plot2d2()***: Treppendarstellung, der Graf wird in einzelnen Schritten gezeichnet.
- ***plot2d3()***: Liniendarstellung, der Graf wird mit vertikalen Linien gezeichnet.
- ***plot2d4()***: Pfeildarstellung, der Graf wird mit einzelnen Pfeilen gezeichnet.

Mehrere Funktionen können im gleichen Fenster, aber mit unterschiedlichen Achsen und Skalierungen gezeichnet werden. Die einzelnen Grafen werden im Zeichnungsfenster matrixartig angeordnet. Verwendet wird der Befehl ***subplot(z, s, n)***.

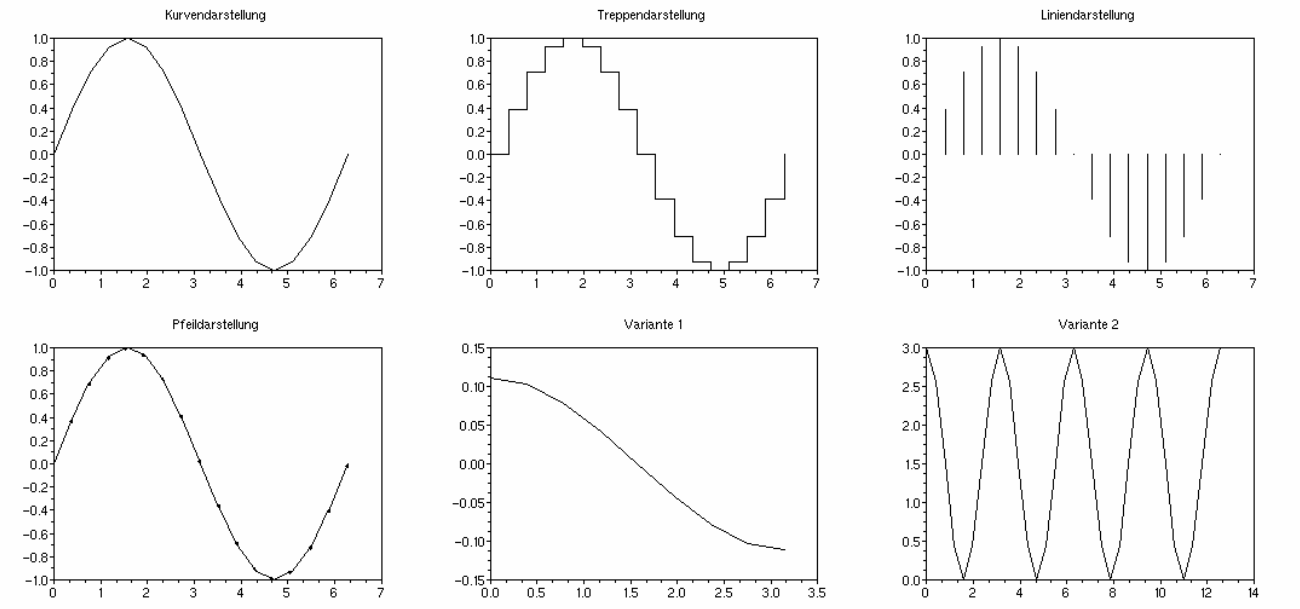
- ***z*** steht für Anzahl der Zeilen
- ***s*** steht für Anzahl der Spalten
- ***n*** steht für die Position der Grafik im Fenster

Beispiel:
Sechs verschiedene Grafen werden in einem Fenster mit 2 Zeilen und 3 Spalten gezeichnet. Die verschiedenen Möglichkeiten des Befehls ***plot2d(d***) werden dabei visualisiert.

Mit dem Befehl ***xtitle("titel")*** kann eine Grafik betitelt werden.

```
-->clear
-->clf()
-->xdel

-->x = [0:(%pi/8):2*%pi];
-->y1 = sin(x);
-->subplot(2,3,1);
-->// 2: Anzahl Zeilen, 3: Anzahl Spalten, 1 fortlaufende Nummer
-->plot2d(x, y1)
-->xtitle("Kurvendarstellung")

-->y2 = sin(x);
-->subplot(2,3,2); // Grafik an 2. Position
-->plot2d2(x,y2)
-->xtitle("Treppendarstellung")

-->y3 = sin(x);
-->subplot(2,3,3); // Grafik an 3. Position
-->plot2d3(x,y3)
-->xtitle("Liniendarstellung")

-->y4 = sin(x);
-->subplot(2,3,4); // Grafik an 4. Position
-->plot2d4(x,y4)
-->xtitle("Pfeildarstellung")

-->x1 = [0:(%pi/8):%pi];
-->y5 = cos(x1)/9;
-->subplot(2,3,5); // Grafik an 5. Position
-->plot2d(x1,y5)
-->xtitle("Variante 1")

-->x2 = [0:(%pi/8):4*%pi];
-->subplot(2,3,6); // Grafik an 6. Position
-->plot2d(x2,3*cos(x2)^2)
-->xtitle("Variante 2")
```

```
Prompt
Eingabe
Resultat
//Kommentar
```


Kurvendarstellung
Treppendarstellung
Liniendarstellung
Pfeildarstellung
Variante 1
Variante 2

10.3 Weitere Optionen zur Gestaltung einer Grafik

10.3.1 Parameter zum Befehl plot2d()

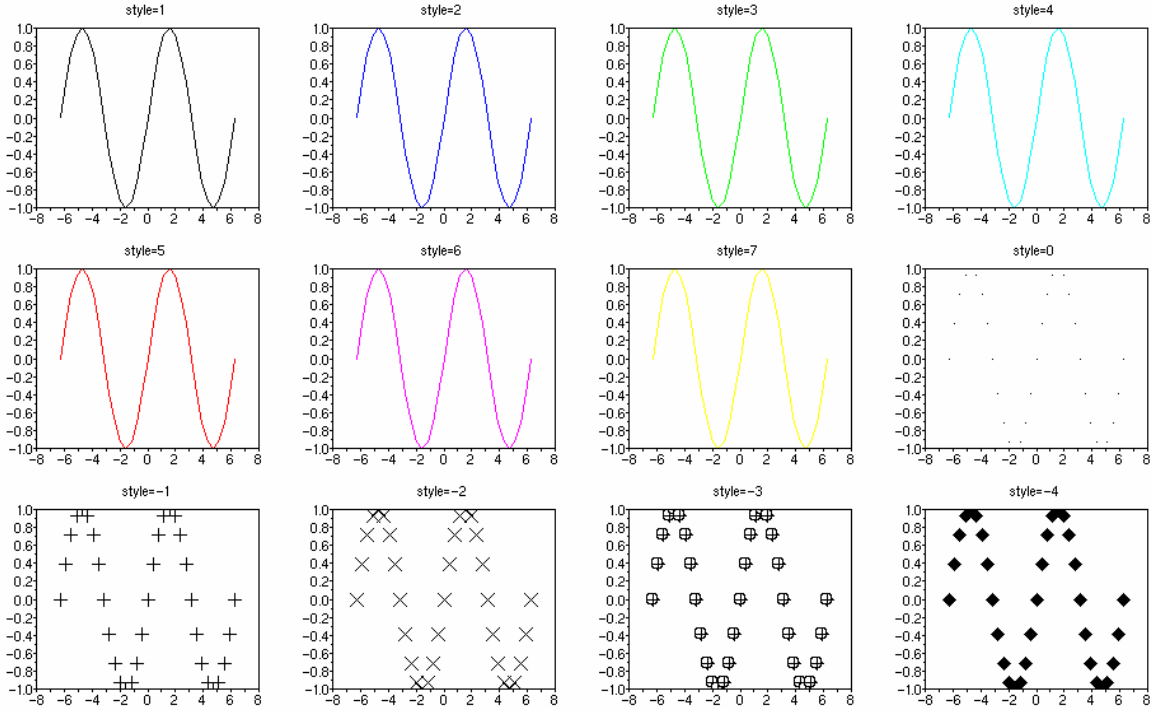
Der Befehl ***plot2d()*** kennt weitere Parameter zur Gestaltung der Grafiken. Die wichtigsten Parameter sind in diesem Kapitel erklärt.
Die Struktur des Befehls ***plot2d()*** mit den Parametern sieht folgendermassen aus:

```
plot2d (x, y, style=i, rect=[xmin,ymin,xmax,ymax], logflag="xy",
frameflag=j, axesflag=3)
```

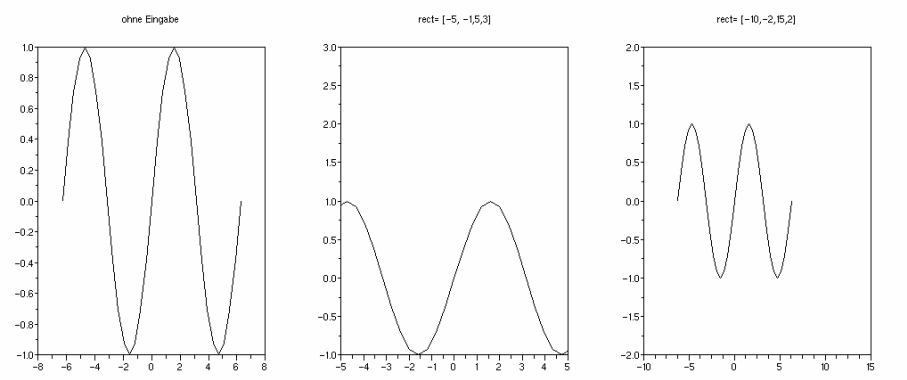
- ***x***: Verwendete x-Achse (sie wurde vorgängig definiert)
- ***y***: Angezeigte Funktion
- ***style = i***: Definiert Farbe und Gestaltung der Kurve.

Beispiel:

i=1...7 und i=0...-4

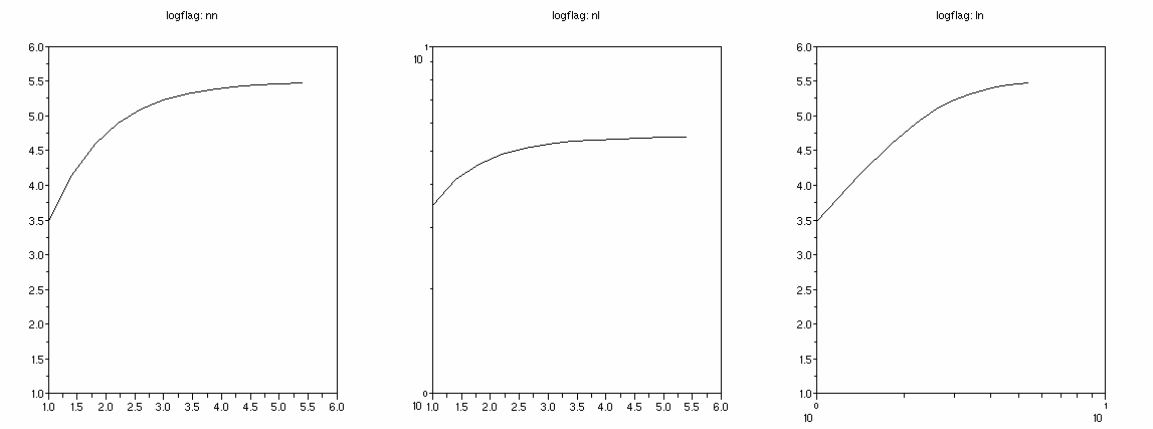


- ***rect=[xmin,ymin,xmax,ymax]***: Definiert Bereich des Vierecks um die gezeichnete Kurve. Wenn ***rect*** nicht spezifiziert wird, dann passt **Scilab** den Bereich automatisch an. Beispiel mit drei Varianten:
  - linkes Bild: ohne Spezifikation von ***rect***
  - Bild in der Mitte: rect= [-5,-1,5,3]
  - Rechtes Bild: rect= [-10,-2,15,2]



- ***logflag="xy"***: Dieser Parameter bestimmt, ob die x- und y-Achse logarithmisch oder linear skaliert werden:
  - ***"nn"***: normal/normal
  - ***"nl"***: normal/logarithmisch
  - ***"ln"***: logarithmisch/normal
  - ***"ll"***: logarithmisch/logarithmisch

  Beispiel: Die gleiche Funktion mit den Skalierungen ***"nn"***, ***"nl"*** und ***"ln"***:

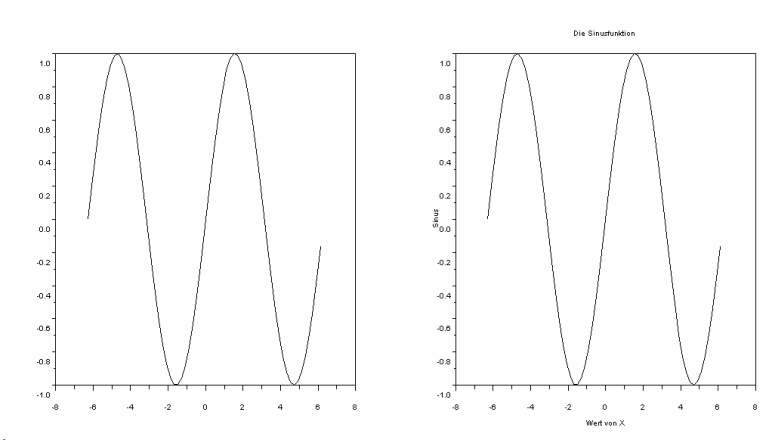


- ***frameflag=j:*** Der Wert j (1 bis 6) gibt an, wie die Achsen skaliert werden.
  - ***j=0***: Benutzung des vorherigen Skalierung oder der Voreinstellung
  - ***j=1***: Die Skalierung ergibt sich aus den Werten des Parameters rect
  - ***j=2***: Die Skalierung wird aus dem Minimum und Maximum der x- und y-Werten berechnet
  - ***j=3***: Isometrische Skalierung, abgeleitet von ***rect***
  - ***j=4***: Isometrische Skalierung, abgeleitet aus dem Minimum und Maximum der x- und y-Werten
  - ***j=5***: Wie j=1, mit einer Anpassung für eine harmonische Unterteilung
  - ***j=6***: Wie j=2, mit einer Anpassung für eine harmonische Unterteilung

10.3.2 Bezeichnung der Achsen mit xtitle()

Mit dem Befehl ***xtitle("Titel", "X-Achse", "Y-Achse")*** werden die Achsen beschriftet. Beispiel:

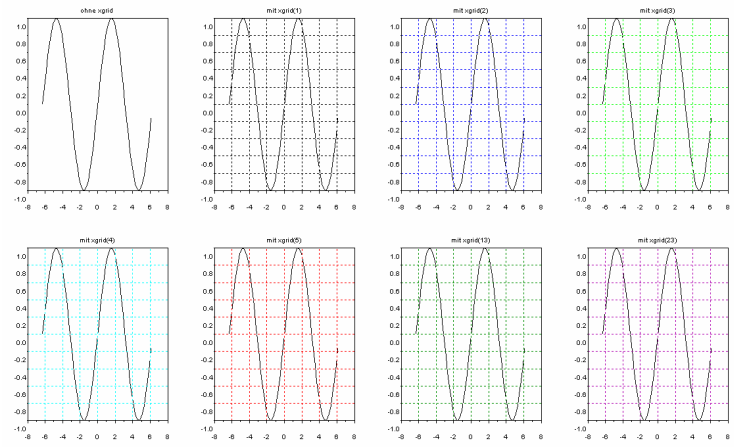
- Linkes Bild: ohne xtitle()-Zeile
- Rechtes Bild: mit Zeile: ***xtitle("Die Sinusfunktion", "Wert von X", "Sinus")***



10.3.3 Gestaltung des Gitters mit xgrid()

Mit ***xgrid()*** wird definiert, ob die Grafik mit einem Gitter versehen wird, und welche Farbe dieses Gitter erhalten soll.

- Fehlt der Befehl ***xgrid(),*** wird kein Gitter gezeichnet.
- Der Parameter f von ***xgrid(f)*** definiert die Farbe des Gitters:
  - 1: Farbe schwarz
  - 2: Farbe blau
  - 3: Farbe hellgrün
  - 4: Farbe hellblau
  - 5: Farbe rot
  - 13: Farbe dunkelgrün
  - 16: Farbe türkis



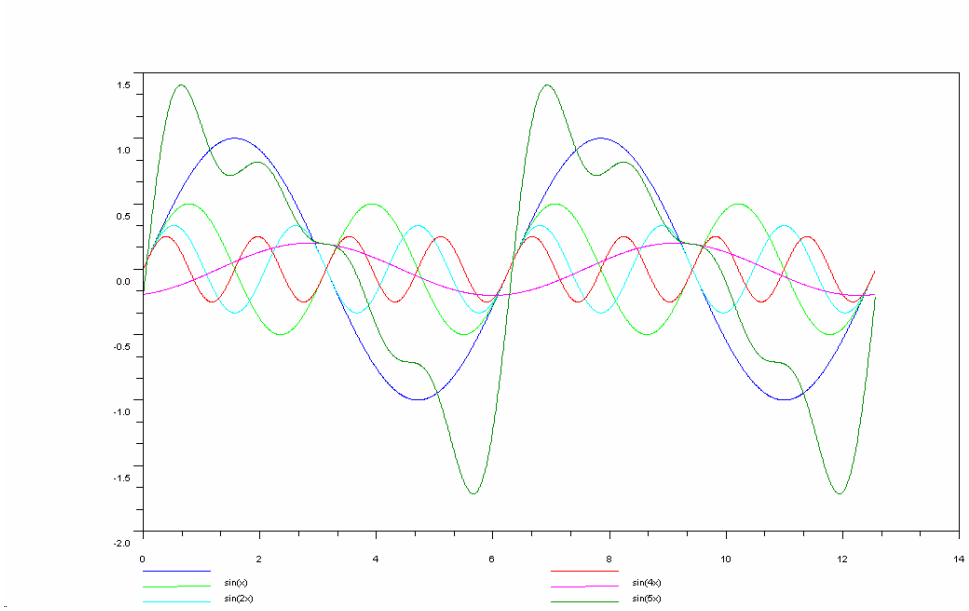
10.3.4 Mehrere Funktionen in der gleichen Grafik zeichnen und mit einer Legende versehen

Scilab kann mehrere Funktionen in der gleichen Grafik zeichnen. Sind mehrere Funktionen in der der gleichen Grafik, ist es sinnvoll, eine Legende zu den Funktionen einzuführen. Werden mehrere Funktionen gezeichnet, müssen sie im Befehl ***plot2d()*** als Vektor aufgelistet werden. Stil (***style()***) und Legende (***leg()***) müssen in der gleichen Reihenfolge wie der Funktionsvektor aufgelistet werden. Die einzelnen Legenden werden mit dem Zeichen @ getrennt.

Beispiel:

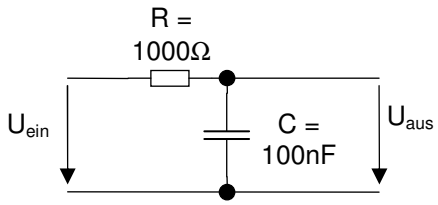
```
-->clear
-->clf()
-->xdel

-->x = [0:0.01:4*%pi]';
-->y1 = sin(x);                  // 1. Funktion
-->y2 = 1/2*sin(2*x);            // 2. Funktion
-->y3 = 1/3*sin(3*x);            // 3. Funktion
-->y4 = 1/4*sin(4*x);            // 4. Funktion
-->y5 = 1/5*sin(5+x);            // 5. Funktion
-->y6 = y1 + y2 + y3 + y4 + y5;  // 6. Funktion

-->// Aufruf des Zeichnungsbefehls inkl. Stil und Legende.
-->plot2d(x, [y1, y2, y3, y4, y5, y6], style=[2 3 4 5 6 13], ...
-->leg="sin(x)@sin(2x)@sin(3x)@sin(4x)@sin(5x)@summe")
```

Prompt
Eingabe
Resultat
//Kommentar



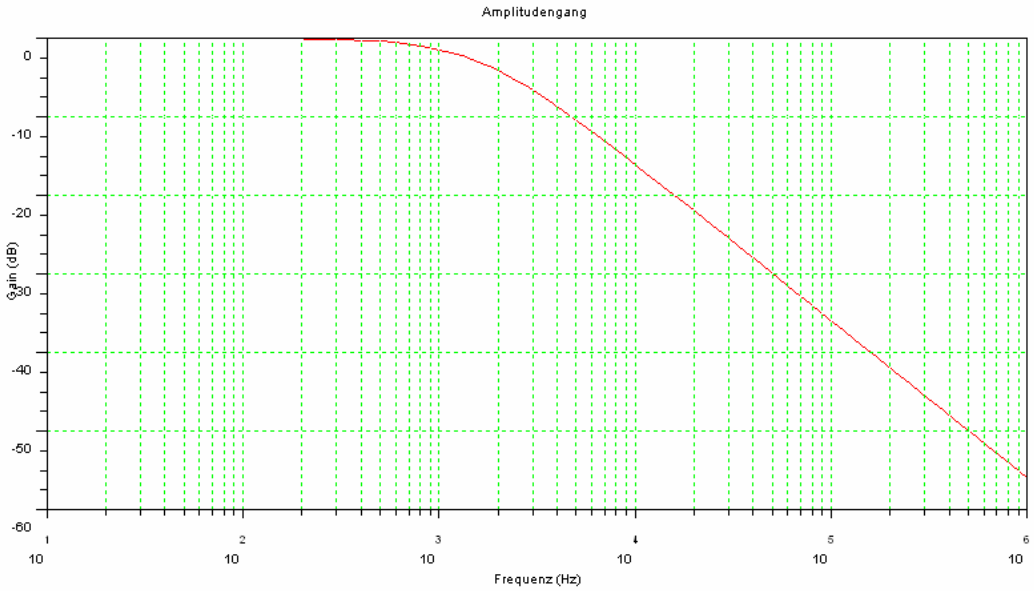
10.4 Beispiel 1 einer Grafik: Bode-Plot

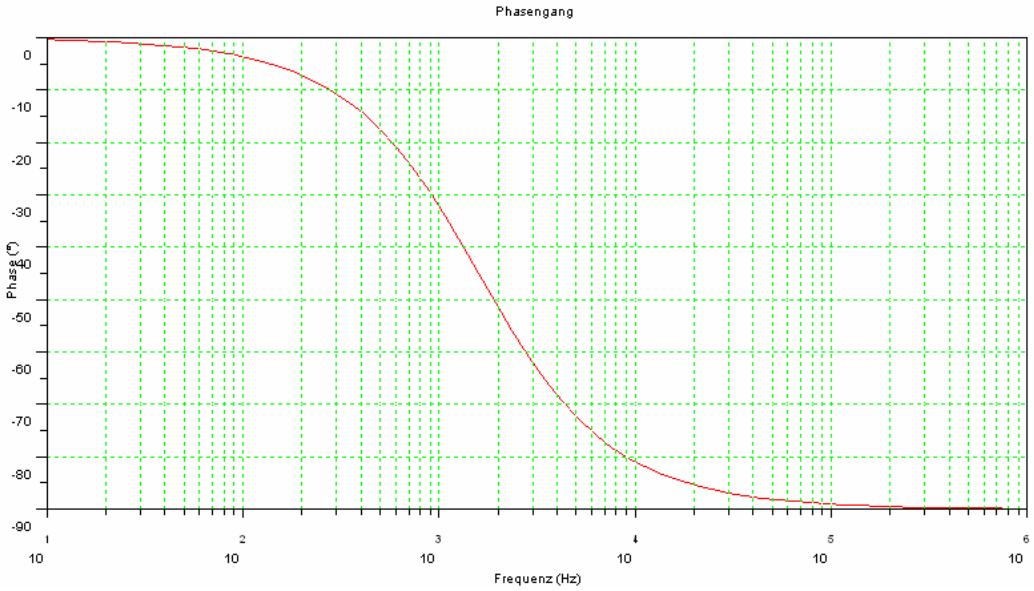
Die Übertragungsfunktion eines einfachen RC-Gliedes (Tiefpass 1. Ordnung) soll gezeichnet werden. Amplituden- und Phasengang werden separat dargestellt.




```
-->// Bode-Diagramm eines RC-Gliedes (Tiefpass 1. Ordnung)
-->// ====================================================
-->clear
-->clf()
-->xdel

-->R = 1000;        // Wert des Widerstandes in Ohm
-->C = 100E-9;      // Wert der Kapazität in Farad

-->freq = logspace(1,6,60);  //Bereich der Darstellung
-->// Übertragungsfunktion (aufgepasst: Divisionszeichen > ./):
-->Gain = 1 ./ (1 + 2*%pi*freq*R*C*%i);

-->// Bestimmung des Gains in dB
-->Gain_dB = 20*log10(abs(Gain));

-->// Bestimmung der Phase in Grad
-->Phase = ((atan(imag(Gain), real(Gain)))/(%pi))*180;

-->// Verteilung der Plots (2 Zeilen, 1 Spalte und Nummer 1)
-->subplot(2,1,1);

-->plot2d(freq, Gain_dB, logflag="ln", style=5)
-->// x-Achse wird Frequenz, y-Achse wird Gain in dB

-->xgrid(3)    // Gitter ist ON

-->xtitle("Amplitudengang","Frequenz (Hz)","Gain (dB)")
-->// Beschriftung des Titels und der Achsen

-->subplot(2,1,2
-->// Verteilung der Plots (2 Zeilen, 1 Spalte und Nummer 2)

-->// x-Achse wird Frequenz, y-Achse wird Phase in Grad
-->plot2d(freq,Phase, logflag="ln", style=5)

-->xgrid(3)         //Gitter ist ON

-->xtitle("Phasengang","Frequenz (Hz)","Phase (°)")
-->// Beschriftung des Titels und der Achsen
```

```
Prompt
Eingabe
Resultat
//Kommentar
```


Amplitudengang
Gain (dB)
0
-10
-20
-30
-40
-50
-60
10
10
10
10
10
10
Frequenz (Hz)


Phasengang
Phase (°)
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
10
10
10
10
10
10
Frequenz (Hz)

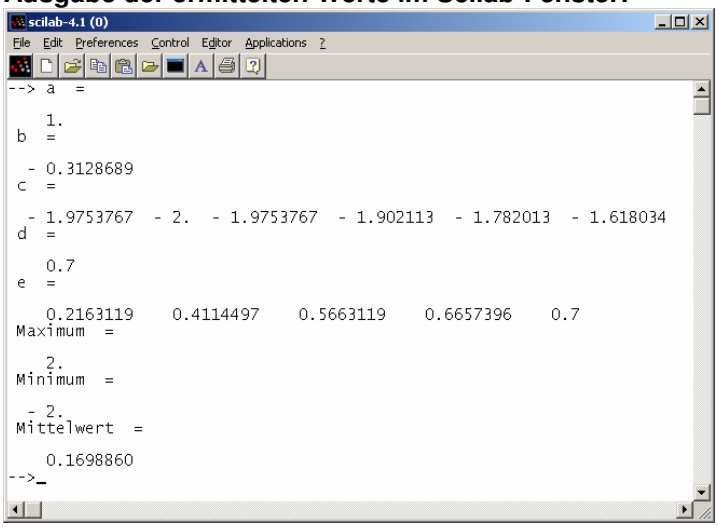
10.5 Beispiel 2: Zusammensetzung und Analyse einer Grafik

Ein Signal wird aus verschiedenen Signalen zusammengesetzt und einzelne Kenndaten des Signals werden berechnet und bestimmt.

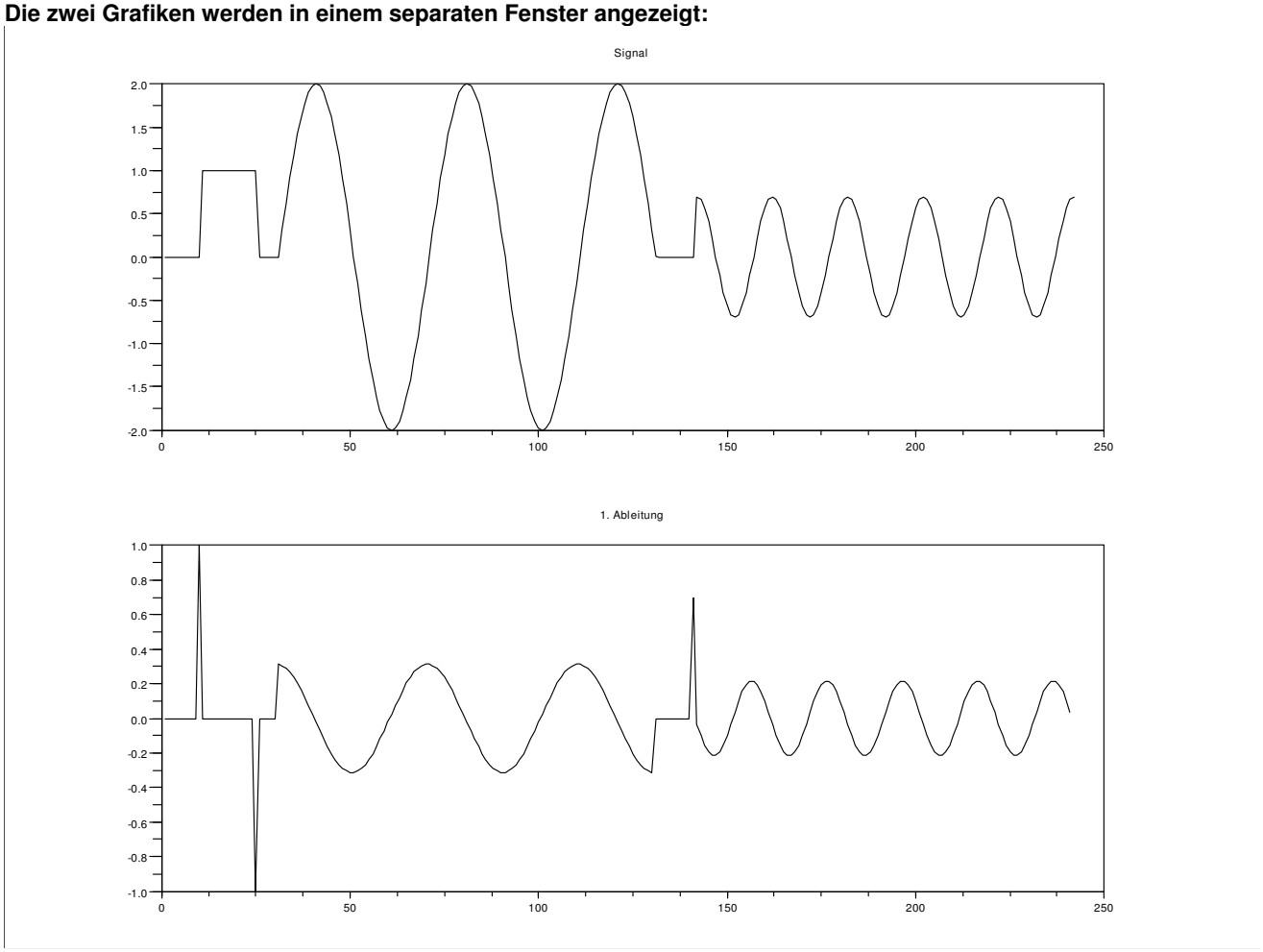
Skript des Beispiels 2, erstellt mit Scipad:

```
// Zusammensetzung und Analyse einer Grafik
clear
clf()
xdel

// Teil-Signale werden in einem Vektor zusammengefasst
t = 0:0.01:1;
Signal =
[zeros(1,10),ones(1,15),zeros(1,5),2.0*sin(1*%pi*5.0*t),zeros(1,10),0.7*cos(1*%pi*10*t)];

// Ausgabe des Signals
subplot(2,1,1);
plot2d(Signal);
xtitle("Signal")

// Aus dem Signal können einzelne Werte extrahiert werden
a = Signal(13)
b = Signal(70)

// Aus dem Signal kann ein Ausschnitt extrahiert werden
c = Signal(100:105)

// Der letzte Wert des Signals:
d = Signal(length(Signal))

// Die letzten 5 Werte werden extrahiert
// Mit $ wird das letzte Element im Vektor angesprochen
e = Signal($-4:$)

// Statistische Werte werden aus dem Signal ermittelt
Maximum = max(Signal)
Minimum = min(Signal)
Mittelwert = mean(Signal)

// Die erste Ableitung des Signal kann angezeigt werden
subplot(2,1,2);
plot2d(diff(Signal))
xtitle("1. Ableitung")
```

Ausgabe der ermittelten Werte im Scilab-Fenster:



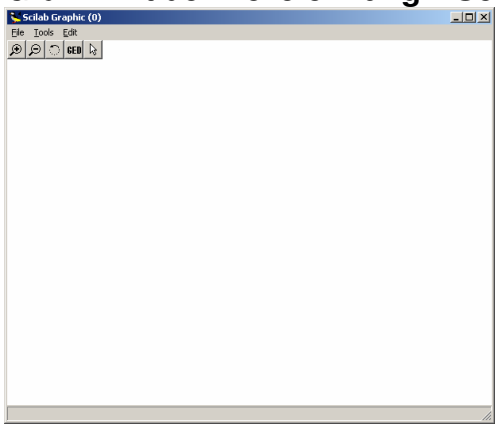
Die zwei Grafiken werden in einem separaten Fenster angezeigt:



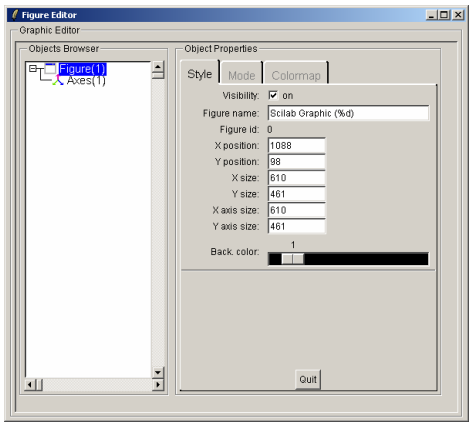
10.6 Alternativer Weg zur Erstellung von Grafiken

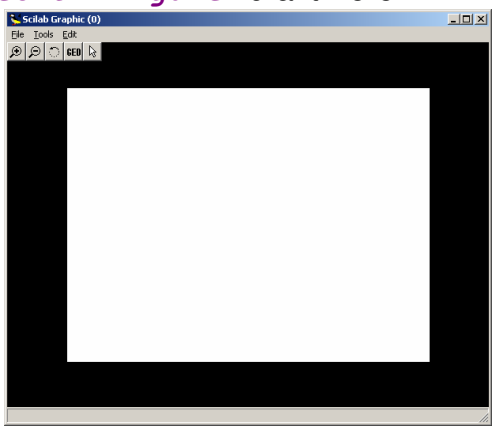
Manchmal ist es mühsam alle Parameter einer Grafik als Befehl einzugeben. Eine alternative Möglichkeit um ansprechende Grafiken einfach zu erstellen wird gezeigt. Grundsätzlich wird zuerst das Gerüst (Achsen, Skalierung, Grösse, Rahmen, Gitter, Beschriftung, ..) im Grafik-Fenster generiert. Dieses Grundgerüst wird abgespeichert. Bei Bedarf wird die abgespeicherte Grafik aufgerufen und die neuen Kurven in diese Grafik gezeichnet. Das Vorgehen wird Schritt um Schritt erklärt.

1. **Eine leere Grafik wird aus Scilab mit dem Befehl `gcf();` aufgerufen: Die leere Grafik mit der Bezeichnung "Scilab Grapfic (0))" wird geöffnet (siehe Bild).**

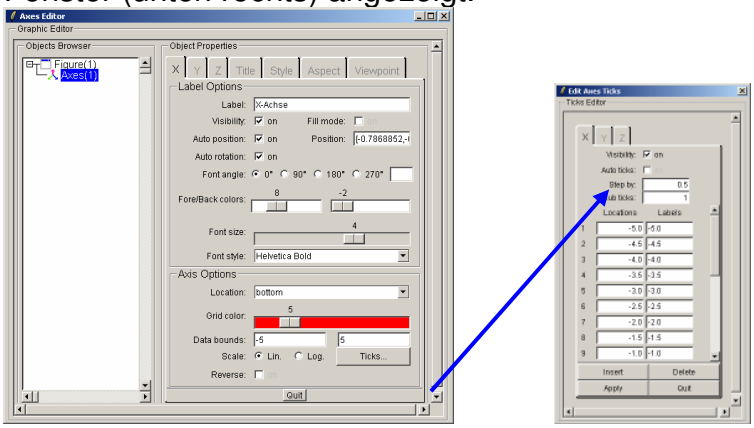


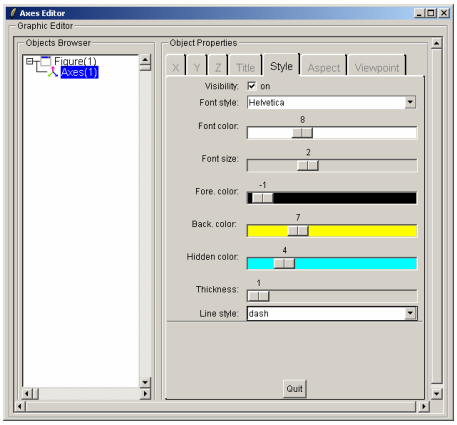
2. **Im Grafik-Fenster mit dem Befehl `Edit Figure ⇨ Properties` den Editor aufrufen.** Standardmässig wird zuerst der **Figure Editor** (Rahmen Editor) eingeblendet. Die gewünschten Einstellungen (Grösse, Farbe, etc.) können direkt im Editor-Fenster (linkes Bild) eingegeben und die Auswirkungen im Grafik-Fenster (rechtes Bild) gesichtet werden. Um die Auswirkungen zu sehen, ist es manchmal notwendig, im Grafik-Fenster den Befehl `Edit ⇨ Redraw Figure` zu aktivieren.

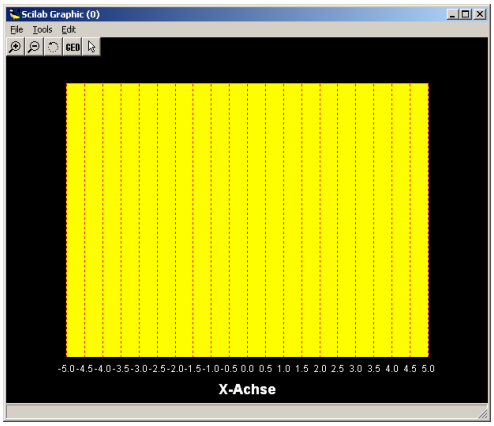




3. **Die X- und Y-Achsen werden konfiguriert.** Entweder im Figure Editor links auf Axes anklicken oder im Grafik-Fenster auf ***Edit ⇨ Current axes properties*** klicken. Im Axes Editor Fenster werden alle Einstellungen (Titel der Achsen, Skalierung, Farben, Gitter, Einteilung des Gitters (ticks), etc.) vorgenommen. Das Ergebnis wird im Grafik-Fenster (unten rechts) angezeigt.

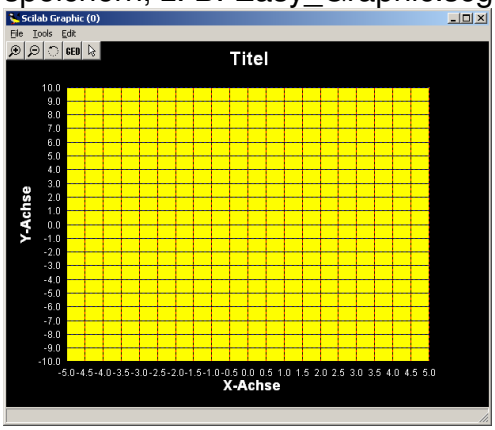






Sämtliche Achsen und Titel definieren.

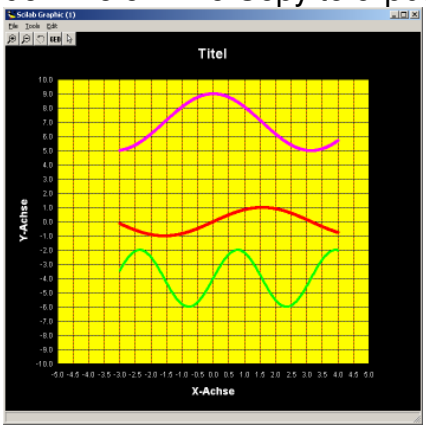
4. **Die vollständig erstellte Grafik kann als Vorlage gespeichert werden.** Im Grafik-Fenster den Befehl ***File ⇨ Save…*** anklicken und Grafik mit Erweiterung ".scg" speichern, z. B. Easy_Graphic.scg

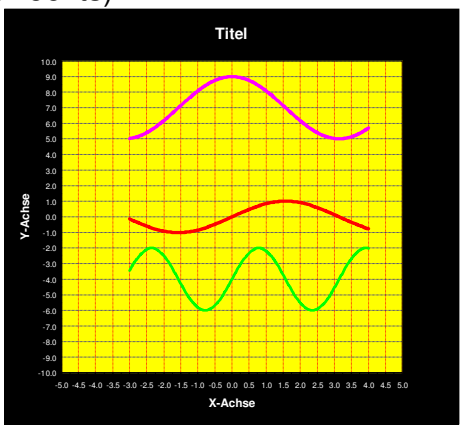


5. **Skript mit gewünschten Funktion schreiben.** Die gespeicherte Grafik-Vorlage (Easy_graphic.scg) kann zur Anzeige eines Graphs genutzt werden. Im Beispiel-Skript werden drei verschiedene Funktionen (Y1, Y2 und Y3) gezeichnet. Gleichzeitig kann noch die Farbe des Graphs mit "style = …" definiert werden. Um die Strichdicke des Plots zu definieren, wird die Befehlsfolge ***e=gce();e.children(1).thickness=5;*** genutzt.

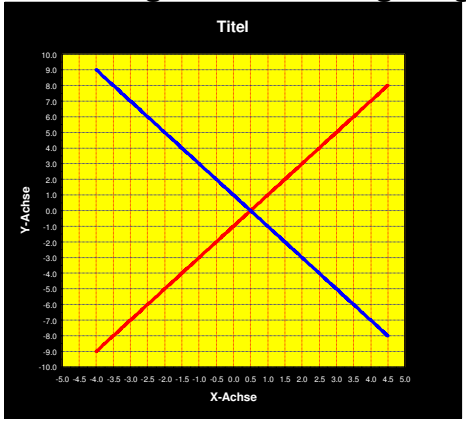
```
clf()
load('Easy_graphic.scg');
X=[-3:0.01:4];
Y1=sin(X);
     plot2d(X,Y1, style=5);
          e=gce();
          e.children(1).thickness=5;
Y2 = 2*cos(X) + 7;
     plot2d(X,Y2, style=6);
          e=gce();
          e.children(1).thickness=5;
Y3 = 4*(cos(X).*sin(X)-1);
     plot2d(X,Y3, style=3);
          e=gce();
          e.children(1).thickness=4;
```

6. **Skript ausführen und Zeichnung ev. exportieren.** Das Skript kann ausgeführt werden, (Bild links) und die erstellte Grafik kann exportiert werden (z. B. in Word) mit dem Befehl File Copy to clipboard (Bild rechts)





7. **Die Vorlage kann beliebig oft genutzt werden.**



11 Eigene Funktionen in Scilab erstellen

Um was geht es in diesem Kapitel?
Dieser Abschnitt zeigt, wie Sie eigene lokale und globale Funktionen definieren und aufrufen können. Lokale Funktionen sind nur gültig in der verwendeten Datei, und globale Funktionen können von einer beliebigen Datei aufgerufen werden. Die globalen Funktionen können später aus **Scilab** aufgerufen und verwendet werden.

Was werden Sie erreichen?
Sie werden in der Lage sein, lokale und globale Funktionen zu generieren und zu verwenden.

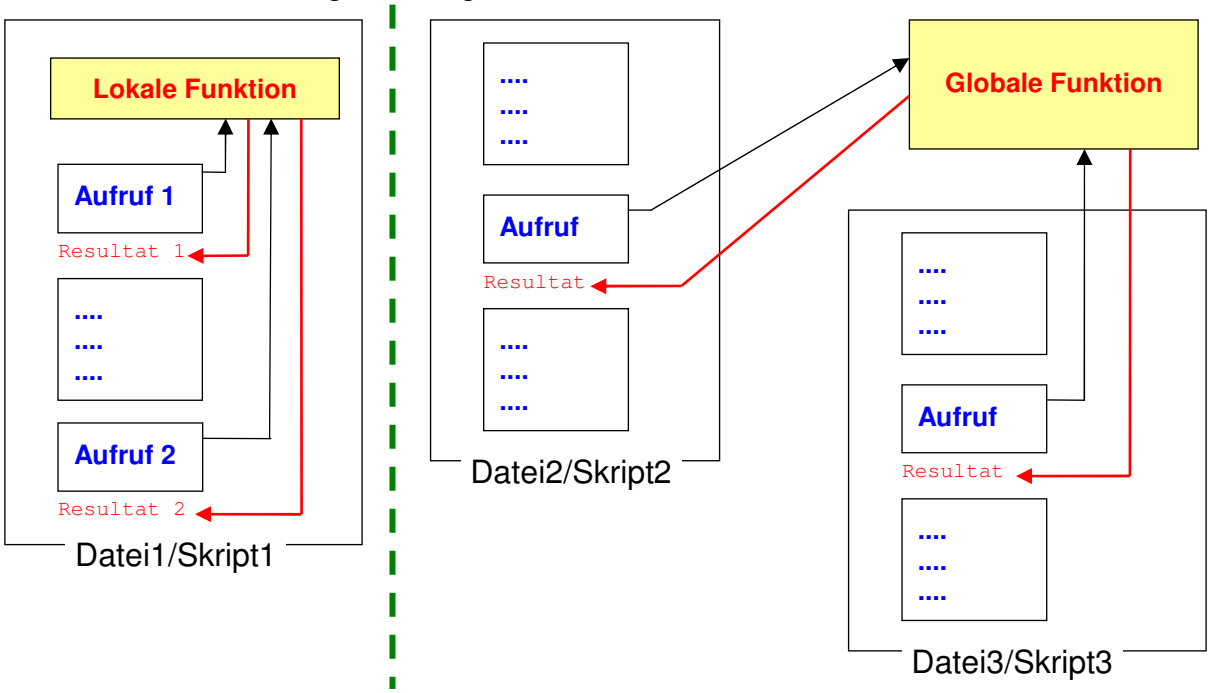
Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

function	endfunction	exec()		

11.1 Lokale und globale Funktionen

Eine **lokale Funktion** ist nur innerhalb der verwendeten Datei bzw. des Skriptes gültig. Eine **globale Funktion** wird gespeichert und kann von einer beliebigen Datei bzw. einem beliebigen Skript aufgerufen werden. In diesem Abschnitt wird gezeigt, wie lokale und globale Funktionen erzeugt und angewendet werden.



11.2 Lokale Funktionen

Mit **Scilab** können Sie eigene lokale Funktionen definieren, schreiben und anwenden. Lokale Funktionen sind nur gültig in der verwendeten Datei. Das Prinzip von lokalen Funktionen wurde bereits eingehend im **Abschnitt 5.6 (Definieren einer lokalen Funktion)** besprochen.

Zur Erinnerung: Die Definition einer Funktion beginnt mit dem Schlüsselwort ***function*** und endet mit dem Schlüsselwort ***endfunction***.

Jede Funktion muss mit dem folgenden Befehl beginnen:
function [y1,y2,...,yn] = Funktionsname(x1,x2,...,xm)
wobei x1, x2, …, xm Eingabeargumente, y1, y2, ..., yn Ausgabeargumente sind. Darauf folgen die Anweisungen, die zur Funktion gehören und aus den Eingabeargumenten die Ausgabeargumente berechnen.

Eine Funktion endet mit dem Schlüsselwort:
endfunction.

11.3 Globale Funktionen

Globale Funktionen können von einer beliebigen Datei aufgerufen werden. Die globalen Funktionen können später aus **Scilab** aufgerufen und verwendet werden. Globale Funktionen werden wie eigene Funktionen eingesetzt. Bevor die globale Funktion ausgeführt wird, muss sie mit dem Befehl ***exec()*** in **Scilab** geladen (eingebunden) sein. Dazu muss der gesamte Pfad der Funktion eingegeben werden, z. B. ***exec('C:\Scilab\kreuz_prod.sci');***. Globale Funktionen haben die **Endung .sci**.

Beispiel:
Wir wollen eine eigene globale **Scilab**-Funktion erstellen. Diese Funktion bestimmt das Kreuzprodukt von zwei Vektoren mit je drei Komponenten. Die Funktion hat die Bezeichnung *kreuz_prod.sci*.

Die in **Scipad** selbst erstellte globale Funktion *kreuz_prod.sci* wird im Ordner *C:\Scilab* gespeichert:

```
// Funktion kreuz_prod.sci
// Zum Berechnen des Kreuzprodukts zweier Vektoren mit drei Komponenten
// Die Funktion wird definiert
// Ausgegeben wird der Wert x (Vektor mit drei Komponenten)
// Zur Berechnung müssen der Funktion die Vektoren a und b übergeben werden.
// Die Vektoren a und b besitzen ebenfalls drei Komponente

function [x]= kreuz_prod(a,b)
  x(1) = (a(2)*b(3) - a(3)*b(2))
  x(2) = (a(3)*b(1) - a(1)*b(3))
  x(3) = (a(1)*b(2) - a(2)*b(1))
endfunction
```

Die neu erstellte globale Funktion *kreuz_prod* wird in **Scilab** eingesetzt:

```
-->// Funktion kreuz_prod.sci wird geladen

-->exec('C:\Scilab\kreuz_prod.sci');

-->// Vektoren h und g werden definiert
--> h = [4 6 8];
-->g = [ -12 8 78];

-->//Das Kreuzprodukt Y wird bestimmt
-->//Dazu wird die Funktion kreuz_prod aufgerufen

-->Y = kreuz_prod(h,g)
 Y  =
    404.
  - 408.
    104.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Hinweis: der Befehl zum Einbinden einer globalen Funktion ***exec('C:\Scilab\kreuz_prod.sci');*** muss nicht manuell eingegeben werden, sondern kann direkt aus der **Scilab**-Befehlsleiste ***File ⇨ Exec ...*** aufgerufen werden. Wird dieser Weg zum Aufrufen gewählt, werden zwei Mitteilungen von **Scilab** ausgegeben:
`disp('exec done'); exec done`

12 Programmierung in Scilab

Um was geht es in diesem Kapitel?
Dieser Abschnitt zeigt, wie Abläufe automatisiert werden. Die wichtigsten Programmierbefehle bzw- Anweisungen (for, while-then-else, if-then-else) werden erklärt und in Beispielen veranschaulicht.

Was werden Sie erreichen?
Sie werden in der Lage sein, Abläufe zu programmieren.

Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

for end	while then else	==	<	>
<=	>=	<>	%t oder %T	%f oder %F
&	|	~	if then else	elseif then
bool2s()				

12.1 Die for-end-Anweisung

Die for-end-Anweisung dient der wiederholten Durchführung einer Anweisung unter der Kontrolle eines Schleifenzählers. Die for-Anweisung wird auch Zählschleife genannt.

Sie hat die allgemeine Form:

for k=a:i:e, Instruktion_1, Instruktion_2; Instruktion_3, ..., ***end***

k=a:i:e

- k ist die Zählvariable
- a ist der Anfangswert
- i ist das Inkrement (wenn es weggelassen wird, dann gilt automatisch 1)
- e ist der Endwert

Instruktionen:

- Wenn mit einem Komma (,) abgeschlossen wird, wird das Resultat angezeigt.
- Wenn mit einem Semikolon (;) abgeschlossen wird, wird das Resultat nicht angezeigt.

For-end-Anweisungen können verschachtelt werden (siehe Beispiel 3).

Beispiel 1:
Für die Zahlen 1 bis 4 werden Wurzel- und Quadratwerte berechnet. Das Resultat von jeder Zahl wird mit einem Vektor ausgegeben.

Befehle in **Scipad**:

```
n = 4;
for k=1:1:n, wurzel=sqrt(k); quadrat=k^2; V1= [k, wurzel, quadrat], end
```

Ausführung von **Scilab**:

```
V1  =
    1.    1.    1.

V1  =
    2.    1.4142136    4.

V1  =
    3.    1.7320508    9.

V1  =
    4.    2.    16.
```

Beispiel 2:
Für die Zahlen 1 bis 4 werden Wurzel- und Quadratwerte berechnet. Das gesamte Resultat wird in der Matrix M2 ausgegeben.

Befehle in **Scipad**:

```
n = 4;
M2 = [];  //Erzeugung einer Leeren Matrix M2 (als Platzhalter)
for k=1:1:n, wurzel=sqrt(k); quadrat=k^2; a=[k, wurzel, quadrat]; M2(k,:)=a; end
// Ausgabe von M2
M2
```

Ausführung von **Scilab**:

```
M2  =

    1.    1.           1.
    2.    1.4142136    4.
    3.    1.7320508    9.
    4.    2.           16.
```

Beispiel 3 (Verschachtelung von Anweisungen):
Eine Matrix (n x m) mit aufsteigenden Werten wird erzeugt.

Befehle in **Scipad**:

```
n = 4;
m = 6;
for i = 1:n,
  for j = 1:m, a(i,j) = j+i-1;
  end
end
a
```

Ausführung von **Scilab**:

```
a  =
    1.    2.    3.    4.    5.    6.
    2.    3.    4.    5.    6.    7.
    3.    4.    5.    6.    7.    8.
    4.    5.    6.    7.    8.    9.
```

12.2 Die while-then-else-Anweisung

Die *while-then-else*-Anweisung dient der wiederholten Ausführung einer Anweisung in Abhängigkeit vom Wert eines booleschen Ausdrucks.

Sie hat die allgemeine Form:

while Bedingung ***then*** Instruction_1, Instruktion_2, ...[,***else*** Instruction_n], ***end***

- Bedingungen: sind logische Vergleiche. Ist der Vergleich wahr (erfüllt), dann werden die Instruktionen vor dem *else*-Befehl ausgeführt.
- Ist die Bedingung nicht erfüllt, dann werden nur die Instruktionen nach *else* ausgeführt. Der Ausdruck [***else*** Instruction_n] ist optional, d.h. er muss nicht immer aufgeführt werden.
- **Vergleichsoperationen** sind:
  - ***==*** für gleich
  - ***<*** für kleiner
  - ***>*** für grösser
  - ***<=*** für kleiner gleich
  - ***>=*** für grösser gleich
  - ***<>*** für ungleich
- **Logische bzw. boolsche Konstanten** sind:
  - ***%t*** oder ***%T*** für wahr (true)
  - ***%f*** oder ***%F*** für falsch (false)
- **Logische Operatoren** sind:
  - ***&*** für UND (AND)
  - ***|*** für ODER (OR)
  - ***~*** für NICHT (NOT)

Beispiel 1:
Die natürliche Zahl d wird mit der Zahl f verglichen. Nach jedem Vergleich wird d um 1 erhöht. Solange d kleiner als f ist, wird die Mitteilung „immer noch kleiner“ ausgegeben. Sobald der Wert von d demjenigen von f entspricht, wird die Mitteilung „endlich gleich“ ausgegeben.

Befehle in **Scipad**:

```
d=1;
f=4;
while d < f then "immer noch kleiner", d = d+1; else "endlich gleich", end
```

Ausführung von **Scilab**:

```
ans  =
 immer noch kleiner

ans  =
immer noch kleiner

ans  =
immer noch kleiner

ans  =
 endlich gleich
```

Beispiel 2:
Wie Beispiel 1. Der Antwort des Vergleichs d < f (wahr oder falsch) wird in der Variable v gespeichert.

Befehle in **Scipad**:

```
d = 1;
f = 4;
v = d < f;
while v == %T then "immer noch kleiner", v = d<f; d = d+1; else "endlich gleich", end
```

Ausführung von **Scilab**:

```
ans  =
 immer noch kleiner

ans  =
immer noch kleiner

ans  =
immer noch kleiner

ans  =
 endlich gleich
```

12.3 Die if-then-else-Selektion

Eine Selektion wird auch Auswahlanweisung oder Alternative genannt. Sie wird in Abhängigkeit eines Ausdrucks gesteuert. Die if-then-else-Anweisung dient der bedingten Ausführung von alternativen Programmzweigen in Abhängigkeit von dem Wert eines booleschen Ausdrucks.

Sie hat die allgemeine Form:

```
if Bedingung_1 then
Instruktionen_1
      elseif Bedingung_2 then
      Instruktionen_2
            elseif Bedingung_n then
            Instruktionen_n
                  else
                  Instruktionen_m
end
```

Ist Bedingung_1 erfüllt (wahr) dann werden nur die Instruktionen_1 ausgeführt, ist Bedingung_1 nicht erfüllt und dafür Bedingung_2 wahr, dann werden nur die Instruktionen_2 ausgeführt. Sind die Bedingungen_1 bis Bedingungen_n nicht erfüllt, dann werden die Instruktionen_m ausgeführt.

Die Vergleichsoperationen, die logischen bzw. booleschen Konstanten und die logischen Operatoren sind diejenigen, die im Abschnitt 12.2 vorgestellt wurden.

Beispiel 1:
Eine natürliche Zahl m wird analysiert. Liegt der Wert von m zwischen 1 und 4, dann wird der Wert ausgegeben. Ist der Wert von m grösser als 4, wird dies mitgeteilt.

Befehle in **Scipad**:

```
m = 5;
if m == 1 then
"m ist 1"
  elseif m==2 then
  "m ist 2"
    elseif m ==3 then
    "m ist 3"
      elseif m== 4 then
      "m ist 4"
        else
        "m ist grösser als 4"
end
```

Ausführung von **Scilab**:

```
ans  =
m ist grösser als 4
```

12.4 Beispiel aus der Digitaltechnik

Das Verhalten einer kombinatorischen digitalen Schaltung soll analysiert und veranschaulicht werden.
Die Schaltung hat folgende Eigenschaften:
Sie hat einen Ausgang: Ausgang
Sie hat fünf Eingänge: Ein_Var_1, Ein_Var_2, Ein_Var_3, Ein_Var_4 und Ein_Var_5

Die logische Verknüpfung lautet:

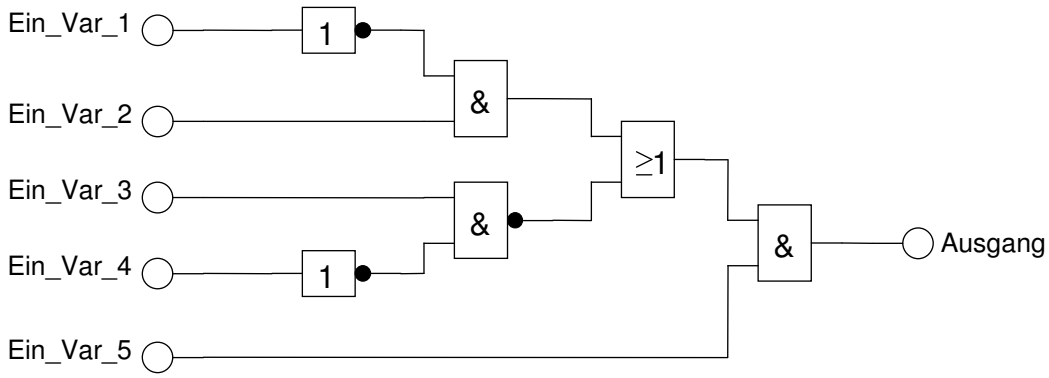
$$\text{Ausgang} = \left[\left(\overline{\text{Ein\_Var\_1}} \wedge \text{Ein\_Var\_2} \right) \vee \left(\overline{\text{Ein\_Var\_3} \wedge \overline{\text{Ein\_Var\_4}}} \right) \right] \wedge \text{Ein\_Var\_5}$$

$\wedge$: UND-Verknüpfung

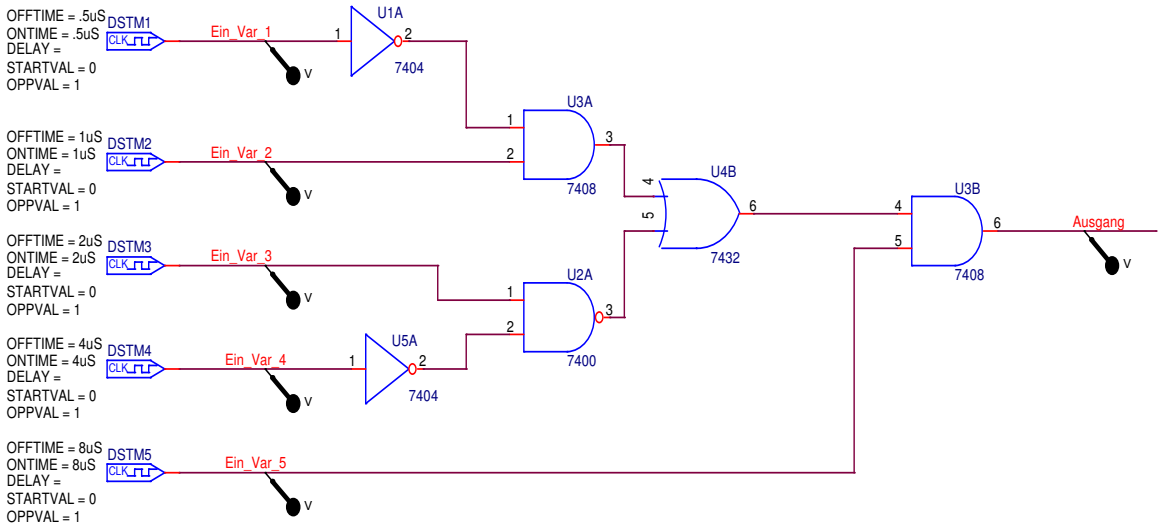
$\vee$: OR-Verknüpfung

$\overline{x}$: Inversion von x

Die Schaltung ist folgendermassen aufgebaut:
a) Gezeichnet mit europäischen Symbolen



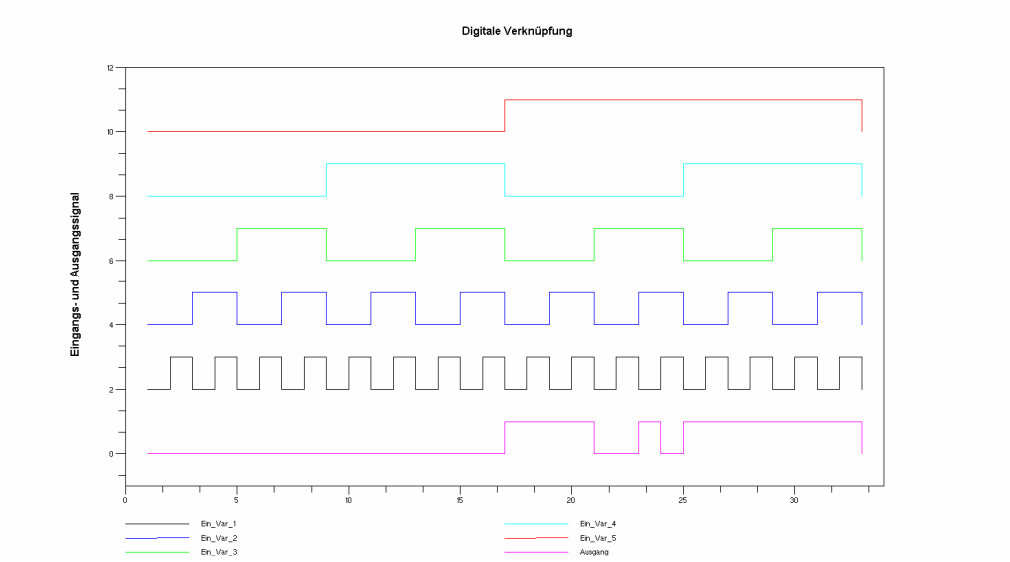
b) Gezeichnet mit PSPICE/ORCAD, inkl. Simulationsgeneratoren (amerikanische Symbole)



Befehle in **Scipad**:

```
// =============== Binär-Folge Kompakt ============================
// Generiert Boolesche Funktionen mit NOT-, OR-, NAND-Verknüpfungen

// Initialisierung:
clear
clc

// Eingabe von m: Anzahl der Eingangs-Variablen
m = 5;    // m = Anzahl Eingangs-Variablen (Ein_Var: Ein_Var_1 ... Ein_Var_m) die betrachtet
werden


// Erzeugung einer Matrix mit allen Eingangskombinationen
n = 2^m + 1;    // n ergibt Anzahl der Spalten, die ausgegeben werden
for i = 1:m   // i ist die aktuelle Zeile (es gibt soviele Zeilen wie Variablen)
w = 2^i;      // w bestimmt die Wertigkeit der aktuellen Variable (2, 4, 8, 16, 32, ...)
      for j = 1:n    // j ist die aktuelle Spalte
       v = j/w - int(j/w);      // v bestimmt den Zwischenwert von 0 bis 1
          if ((v > 0) & (v <= 0.5)) then
                Ein_Var(i,j) = 0;
          else
                Ein_Var(i,j) = 1;
          end
      end
end

// Extrahierung der einzelnen Zeilen und Benennung der Eingangsvariablen
Ein_Var_1 = Ein_Var(1,1:n);
Ein_Var_2 = Ein_Var(2,1:n);
Ein_Var_3 = Ein_Var(3,1:n);
Ein_Var_4 = Ein_Var(4,1:n);
Ein_Var_5 = Ein_Var(5,1:n);

// Vorgeben der Ausgangsfunktion: (~ = NOT), (& = AND), (| = OR)
Ausgang = bool2s(((~Ein_Var_1 & Ein_Var_2) | ~(Ein_Var_3 & ~Ein_Var_4)) &Ein_Var_5) ;
// Die Funktion bool2s convertiert T (wahr) und F (falsch) zu 1 und 0

// Grafik-Ausgabe
clf() // löscht alte Grafiken

plot2d2(1:n, ...
        [Ein_Var_1'+2, Ein_Var_2'+4, Ein_Var_3'+6, Ein_Var_4'+8, Ein_Var_5'+10, Ausgang'], ...
        rect=[0, -1, n+1, 2*m+2], style =[1 2 3 4 5 6], frameflag=1, ...
        leg="Ein_Var_1@Ein_Var_2@Ein_Var_3@Ein_Var_4@Ein_Var_5@Ausgang")
xtitle("Digitale Verknüpfung", "", "Eingangs- und Ausgangssignal")
```

Ausführung von **Scilab**:



13 Lesen von Dateien und schreiben in Dateien

Um was geht es in diesem Kapitel?
Dieser Abschnitt erklärt die verschiedenen Möglichkeiten, um Dateien von **Scilab** aus zu schreiben und um Dateien oder deren Inhalte in **Scilab** zu lesen bzw. zu übernehmen. Es werden nur einfache Befehle eingesetzt.

Was werden Sie erreichen?
Sie werden in der Lage sein, Ihre Eingaben und Ausgaben in **Scilab** automatisch abzuspeichern, Dateien zu öffnen, deren Inhalte zu extrahieren und Daten bzw. Informationen in einer Datei abzulegen. Mit einem ausführlichen Beispiel wird gezeigt, wie aus einer bestehenden Datei Daten gelesen, verändert und in einer neuen Datei geschrieben werden.

Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

In diesem Abschnitt verwendete Scilab-Befehle, Funktionen und Konstanten:

diary('name')	diary(0)	mopen()	mclose()	mfprintf()
\n	\t	%d	%f	%s
%e	%10.3f	mfscanf()	mseek()	fprintfMat()
fscanfMat()	mgetl()	grep()	part()	mclose('all')
evstr()				

Hinweis: Bei allen Beispielen und Demonstrationen sollten Sie die Schriftart *Courier* oder *Courier-New* verwenden, um eine sinnvolle Anzeige der Werte in den Dateien zu erhalten.

13.1 Führen eines Tagebuches

Sie können sämtliche Eingaben, die Sie in **Scilab** tätigen, und die Ausgaben von **Scilab** in einer Tagebuchdatei (englisch: diary file) protokollieren. Das Protokollieren der Eingaben und Ausgaben kann genutzt werden, um eine nachträgliche Skript-Datei zu erzeugen.

Der Befehl zum Starten des Eintrages heisst ***diary('dateiname')***.
Der Befehl zum Beenden des Eintrages heisst ***diary(0)***.

Als Dateiname kann ein vollständiger Pfad oder nur der Name der Datei eingegeben werden. Wird nur der Name der Datei eingeben, wird die Datei im Pfad, der unter ***File ⇨ Get Current Directory*** spezifiziert ist, gespeichert. Dieser vorgegebene Pfad kann mit dem Befehl ***File ⇨ Change Directory*** geändert werden.

Beispiele für ***diary(dateiname)***:
Mit vollständiger Pfadsangabe: ***diary('C:\Scilab_Dateien\Tagebuch.txt')***
Verwendung des Standard-Pfads: ***diary('Tagebuch.txt')***

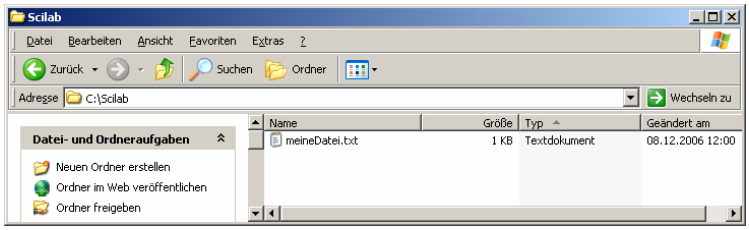
Es ist sinnvoll, den Namen der Datei mit der Erweiterung *.txt* zu versehen. Mit dieser Erweiterung ist ersichtlich, dass es sich um eine Textdatei handelt, die mit einem Texteditor (z. B. **Scipad** oder mit Windows XP: *WordPad*) betrachtet und bearbeitet werden kann.

Beispiel: Eingaben in **Scilab**

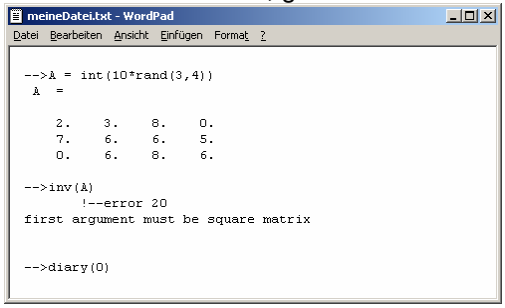
```
-->C = 5;
-->D = 9;
-->F = C + D
 F  =
    14.

-->// Ab hier sollen die Ein- bzw. Ausgaben gespeichert werden
-->// Die Datei soll meineDatei.txt benannt werden

-->diary('C:\Scilab\meineDatei.txt')
-->A = int(10*rand(3,4))
 A  =
    2.    3.    8.    0.
    7.    6.    6.    5.
    0.    6.    8.    6.
-->inv(A)
       !--error 20
first argument must be square matrix
-->diary(0)
-->// Hier endet die Protokollierung

-->F = C * D
 F  =
    45.
```

```
Prompt
Eingabe
Resultat
//Kommentar
```

Im Ordner *C:\Scilab* wurde die Datei *meineDatei.txt* erstellt:



Die neu erstellte Datei, geöffnet mit *WordPad*:



13.2 Befehle zum Lesen, Schreiben und Verarbeiten

Folgende Befehle werden vorgestellt:

- Um Dateien zu erstellen, öffnen und schliessen: ***mopen()*** und ***mclose()***
- Um in Dateien zu schreiben: ***mfprintf()*** und ***fprintfMat()***
- Um aus Dateien zu lesen: ***mfscanf()*** und ***fscanfMat()***
- Um Dateien zu verändern: ***mseek()***

In den Beispielen 2 und 3 werden noch weitere Befehle eingesetzt. Eine Kurzbeschreibung ist jeweils im entsprechenden Scipad-Skript vorhanden.

13.2.1 Eine Datei anlegen oder öffnen mit mopen()

Bevor in einer Datei geschrieben wird, muss sie existieren. Mit ***mopen()*** wird eine neue Datei erstellt oder eine bestehende Datei geöffnet. Eine Datei wird geöffnet oder erstellt mit folgender Syntax:

fd = mopen('meineDatei.text','mode')

fd (file descriptor) steht für den zugewiesenen, temporären Namen der geöffneten Datei. **Scilab** weist einer neu erstellten Datei eine Nummer zu. Diese Nummer wird in der Variablen fd (file descriptor) abgespeichert.

'meineDatei.txt' ist die Datei oder der vollständige Pfad der Datei, die erstellt bzw. geöffnet wird

'mode' definiert, was mit der Datei geschehen soll:

- ***'w'*** zum Erstellen (wenn sie noch nicht existiert) einer neuen Datei
- ***'w'*** zum Schreiben in einer bestehenden Datei. Existiert die Datei schon, wird bei der Verwendung von ***'w'*** der Inhalt gelöscht
- ***'a'*** um eine Datei zu öffnen und mit Inhalten zu erweitern
- ***'r'*** für lesen des Inhalts der Datei (wenn sie schon existiert)

Beispiel: ***Temperatur = mopen('C:\Scilab\messungen.txt','w')***
Entwicklung = mopen("werte.txt", "a"). In diesem Fall wird die Datei "werte.txt" im voreingestellten Pfad (siehe Beschreibung des Befehls "Diary") abgespeichert.

13.2.2 Eine Datei schliessen mit mclose()

Eine geöffnete Datei muss nach dem Bearbeiten (lesen, schreiben oder bearbeiten) mit folgender Syntax wieder geschlossen werden:
mclose(fd) oder ***mclose('meineDatei.txt')***
Alle geöffneten Dateien können gemeinsam mit ***mclose('all')*** geschlossen werden.

Beispiel: ***mclose(Temperatur***) oder ***mclose('C:\Scilab\messungen.txt')***

13.2.3 Formatierungsanweisungen

Um in Dateien zu schreiben bzw. aus Dateien zu lesen, werden Formatierungsanweisungen benötigt. Mit diesen Anweisungen wird die Darstellung der entsprechenden Inhalte verändert.
Die wichtigsten Formatierungsanweisungen sind:

- **`%d`** für Ganzzahldarstellung (z.B. 123)
- **`%f`** für Fliesskommadarstellung (z. B. 123.4568)
- **`%e`** für Exponentialnotationsdarstellung (z. B. 1.234568e+002)
- **`%s`** für Textdarstellung (s für String) (z. B. Guten Tag)
- **`%10.3f`** um Ausgabegrösse zu definieren
  - 10 für gesamte Anzahl der Ziffern (vorlaufende Nullen werden nicht angezeigt)
  - 3 für Anzahl der Ziffern nach dem Komma
- **`\n`** für die Anweisung “auf neue Linie gehen“
- **`\t`** für die Anweisung “horizontalen Tabulator verwenden“

Einige Kombinationen sind möglich. In diesem Fall müssen sie zusammengeschrieben werden. Zum Beispiel **`%6.2f\n`** ist eine sinnvolle Kombination, hingegen macht %e%d keinen Sinn.

13.2.4 Daten in der geöffneten Datei schreiben mit mfprintf()

Mit dem Befehl **`mfprintf()`** werden Daten und Werte in einer geöffneten Datei geschrieben.

Die Struktur des Befehls sieht folgendermassen aus:
`mfprintf(fd` oder **`'DateiName.txt'`** oder **`'C:\pfad\DateiName.txt',`**
`'text_a format_1 text_b format_2 text_c format_3 ...' , wert_1, wert_2, wert_3, ...)`

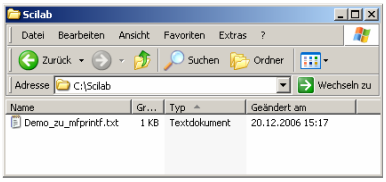
Die Anweisung **`format_n`** gilt für den Wert **`wert_n`** (Bsp. **`format_2`** formatiert **`wert_2`**)
Für **`format_n`** stehen Angaben wie Darstellung der Zahl, Anzahl der Stellen, Zeilenumbruch, usw.
Pro Wert werden alle Formatanweisungen zusammengeschrieben, z. B. **`%10.3f\n`**.
Für **`text_n`** kann ein beliebiger Begleittext eingegeben werden.

Beispiel: Verschiedene Varianten der Ausgabe werden demonstriert.

Befehle in **Scipad**:

```
// Neue Datei Demo_zu_mfprintf.txt wird erstellt:
fd = mopen('C:\Scilab\Demo_zu_mfprintf.txt','w');

// Die Variablen zum Anzeigen werden erstellt:
A = 123.45678901;
a = 0.2;
b = 1.23e-02;
c = a + %i*b;
text = 'Guten Tag';

// Verschiedene Varianten der Ausgabe werden demonstriert:
mfprintf(fd, ' %d\n %10d\n %20d\n %8.4f\t %8.4f\n %5.2f\t %5.2f\t %5.2f\n', A, A, A, A, A, A, A, A);
mfprintf(fd, ' %d\n %f\t %e\n %10.3f\t %6.2f\n kompl. Zahl = %f +i%f\n\n', A, A, A, A, A, real(c), imag(c));
mfprintf(fd, ' %e\t %5.2e\n %s\n %5s\t %10s\t %15s\t %20s\t\n',  A, A, text, text, text, text, text);

// Die geöffnete Datei muss mit mclose() geschlossen werden:
mclose(fd);
```

Im Windows-Ordner C:\Scilab wurde die Datei 'Demo_zu_mfprintf.txt' erstellt:



Der Inhalt der Datei 'Demo_zu_mfprintf.txt':

13.2.5 Lesen des Inhalts einer Datei mit mfscanf() und mseek()

Mit dem Befehl ***mfscanf()*** werden Daten aus einer Datei gelesen. Gelesen werden kann entweder der gesamte Inhalt der Datei oder nur einzelne Sätze. Wenn aus der Datei gelesen wird, verschiebt sich ein Positionszeiger. Dieser Zeiger teilt **Scilab** mit, wie weit der Datei schon Daten entnommen wurden. Der Zeiger kann mit ***mseek(0, fd)*** auf den Beginn der Datei verschoben werden.

Die Struktur des Befehls sieht folgendermassen aus:

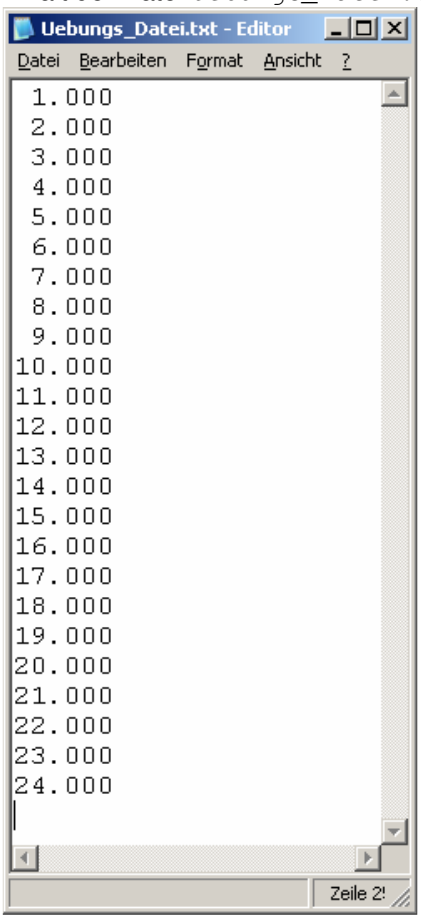
- Gesamter Inhalt auslesen und formatieren mit ***mfscanf(-1, fd, '%f')***
- Einzelne Daten auslesen und formatieren mit ***mfscanf(fd, '%f %f %f %f %f')***

Beispiel:
Verschiedene Varianten des Lesens aus einer Datei werden demonstriert.

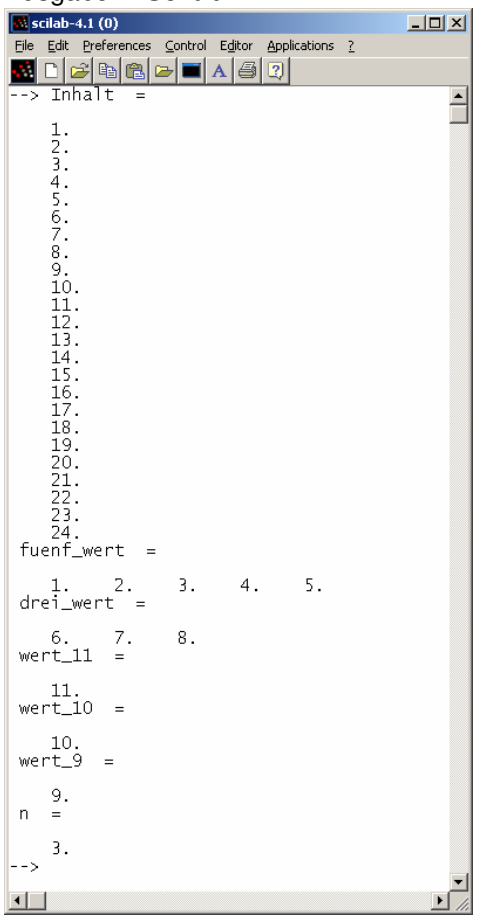
Befehle in **Scipad**:

```
// Eine Übungs-Datei wird erstellt und geöffnet:
fd = mopen('C:\Scilab\Uebungs_Datei.txt','w');

// Daten werden erstellt und in der Übungsdatei geschrieben:
t = (1:1:24)';
mfprintf(fd, '%6.3f\n', t);

// Die Übungsdatei wird geschlossen:
mclose(fd);

// Die Übungsdatei wird zum Lesen geöffnet:
fd = mopen('C:\Scilab\Uebungs_Datei.txt','r');

// Der gesamte Inhalt der Datei wird ausgelesen (-1 steht für 'gesamte Datei lesen') und formatiert:
Inhalt = mfscanf(-1, fd, '%f')

// Nach dem Lesen befindet sich der Positions-Zeiger am Ende der Datei.
// Um erneut auszulesen, muss der Positionszeiger wieder an den Anfang verschoben werden:
mseek(0, fd)

// Die ersten fünf Werte der Datei werden ausgelesen und einzeln formatiert:
fuenf_wert = mfscanf(fd, '%f %f %f %f %f')
// Die nächsten drei Werte werden ausgelesen und formatiert:
drei_wert = mfscanf(fd, '%f %f %f')
// Die nächsten zwei Werte auslesen, mit direkter Zuordnung zu einem Variablennamen
// n gibt die Anzahl der ausgelesenen Werte zurück.
[n, wert_9, wert_10, wert_11] = mfscanf(fd, '%f %f %f')

// Die Datei muss geschlossen werden:
mclose(fd);
```

Inhalt der Datei *Uebungs_Datei.txt*



Ausgabe in **Scilab**:



13.2.6 Lesen und schreiben einer Matrix

Mit den Befehlen ***fprintfMat() und fscanfMat()*** wird eine Matrix in eine Datei geschrieben bzw. wird eine Matrix aus einer Datei gelesen.

Schreiben einer Matrix mit ***fprintfMat()***:
fprintfMat('neue_Matrix.txt',A,'%9.3f', 'Titel der Matrix:')

- ***'neue_Matrix.txt'*** (mit und ohne Pfadsangabe) für den Namen der Datei
- ***A*** ist die zu schreibende Matrix
- ***'%9.3f'*** Formatierungsanweisung zum Schreiben der Matrix
- ***'Titel der Matrix:'*** zum Definieren eines Textes. Dieser muss zwingend auf der 1. Zeile der Datei stehen.

Die Datei wird automatisch erstellt und geschlossen.

Lesen einer Matrix mit ***fscanfMat()***:

- ***M = fscanfMat('neue_Matrix.txt')*** zum Lesen der Matrix ohne Titel auf der 1. Zeile

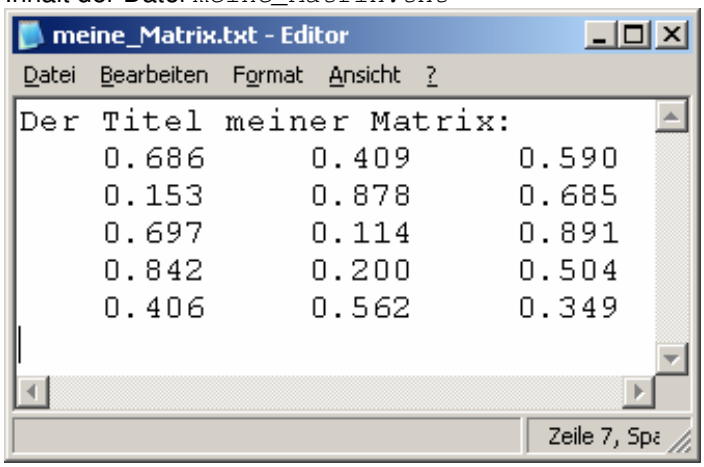
- ***[G, text] = fscanfMat('zogg_Matrix.txt')*** zum Lesen der Matrix inkl. des Textes auf der 1. Zeile

Beispiel:
Verschiedene Varianten des Schreibens und Lesen einer Matrix werden demonstriert.

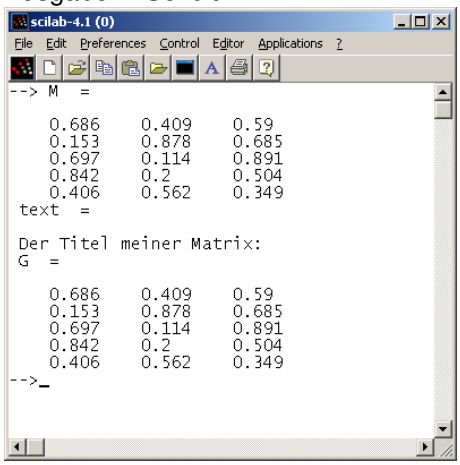
Befehle in **Scipad**:

```
// Speichern einer Matrix mit Text auf der 1. Zeile
// Die Datei wird automatisch erstellt geschlossen

// Eine Matrix wird erstellt:
A=rand(5,3);

// Die Matrix A wird in der Datei meine_Matrix.txt gespeichert
fprintfMat('C:\Scilab\meine_Matrix.txt',A,'%9.3f', 'Der Titel meiner Matrix:');

// Lesen der Matrix aus der Datei
// Nur der Inhalt der Matrix meine_Matrix.txt wird gelesen
M = fscanfMat('C:\Scilab\meine_Matrix.txt')

// Lesen der Matrix, inkl. des Textes auf der 1. Zeile:
[G, text] = fscanfMat('C:\Scilab\meine_Matrix.txt')
```

Inhalt der Datei *meine_Matrix.txt*



Ausgabe in **Scilab**:



13.3 Beispiele

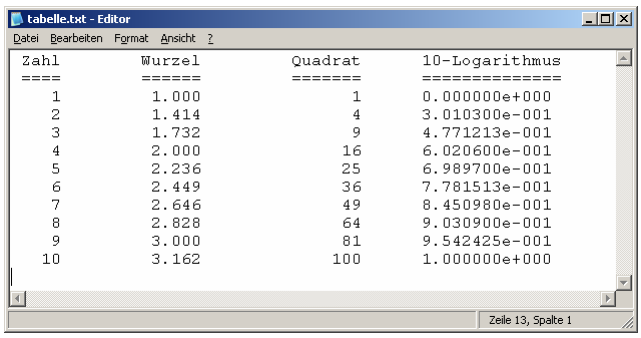
13.3.1 Beispiel 1: Erstellen einer Tabelle

In einer Datei soll eine Tabelle erstellt werden. Für die Zahlen 1 bis 10 sollen der Wurzelwert, der Quadratwert und der Zehner-Logarithmus aufgelistet werden.

Befehle in **Scipad**:

```
// ============= Tabelle ==========================
// Beispiel 1 zum Thema "Schreiben in einer Datei"
// Tabelle mit den Wurzelwerten, Quadratwerten und Zehner-Logarithmus wird erstellt
// Verwenden Sie die Schriftart "Courier" oder "Courier-New" um eine sinnvolle Anzeige zu haben.

//Datei wird erstellt
Datei = mopen('c:\scilab\tabelle.txt', 'w');

// Titelzeile wird erstellt
mfprintf(Datei, '%5s\t %10s\t %10s\t %10s\n', 'Zahl', 'Wurzel', 'Quadrat', '10-Logarithmus');
mfprintf(Datei, '%5s\t %10s\t %10s\t %10s\n', '====', '======', '=======', '==============');

// Tabelle wird erstellt und Zeile um Zeile ausgedruckt
for i = 1:10
  wurzel_wert = sqrt(i);
  quadrat_wert = i^2;
  log_wert = log10(i);
  mfprintf(Datei, '%5d\t %10.3f\t %10d\t %10e\n', i, wurzel_wert, quadrat_wert, log_wert);
end

//Datei wird geschlossen
mclose(Datei);
```

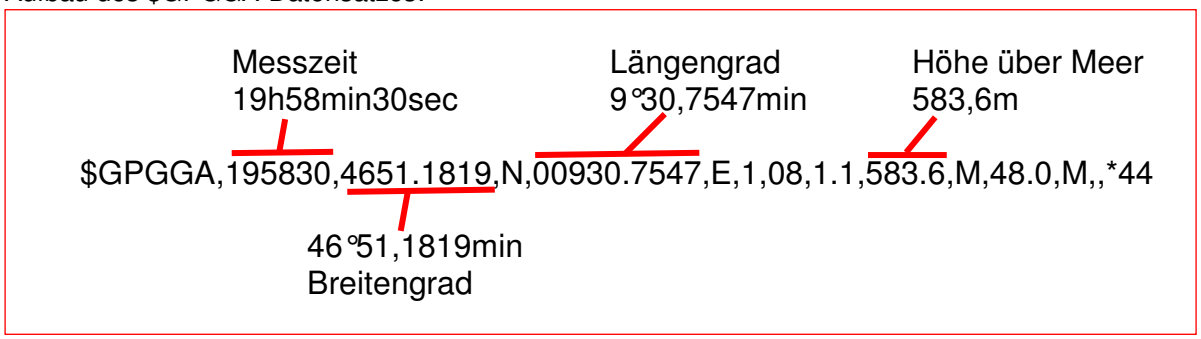
Inhalt der Datei *tabelle.txt*



13.3.2 Beispiel 2: Auswertung von Daten und neue Anzeige (1)

Bei diesem Beispiel wird demonstriert, wie Werte selektiv aus einer Datei extrahiert, verarbeitet und in einer neuen Datei geschrieben werden können. Konkret werden die Datensätze einer GPS-Messung (Globales Positionierungs-System) verarbeitet. Die Datensätze wurden gemäss NMEA-Protokoll (National Marine Electronics Association) erstellt. Der Datensatz $GPGGA listet Messzeit, Breitengrad (Latitude), Längengrad (Longitude) und Höhe über Meer (Altitude) auf.

Aufbau des $GPGGA-Datensatzes:



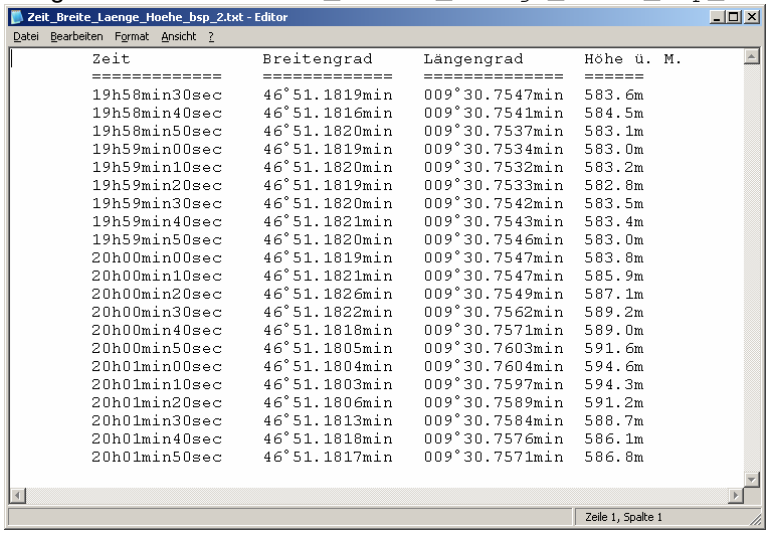
Inhalt der NMEA-Datei (Auszug aus der Datei *NMEA_Daten_15_12_2006.txt*

```
$GPRMC,195830,A,4651.1819,N,00930.7547,E,000.0,298.3,151206,001.0,E*7C
$GPGGA,195830,4651.1819,N,00930.7547,E,1,08,1.1,583.6,M,48.0,M,,*44
$GPGSV,3,1,10,01,43,053,49,04,09,306,00,11,55,151,52,13,13,208,44*77
$GPRMC,195840,A,4651.1816,N,00930.7541,E,000.0,298.3,151206,001.0,E*72
$GPGGA,195840,4651.1816,N,00930.7541,E,1,08,1.2,584.5,M,48.0,M,,*4D
$GPGSV,3,2,10,14,01,038,00,17,38,289,49,20,78,317,49,23,44,197,51*7A
$GPRMC,195850,A,4651.1820,N,00930.7537,E,000.0,298.3,151206,001.0,E*77
$GPGGA,195850,4651.1820,N,00930.7537,E,1,08,1.1,583.1,M,48.0,M,,*48
$GPGSV,3,3,10,25,29,085,49,31,24,076,49*7C
$GPRMC,195900,A,4651.1819,N,00930.7534,E,000.0,298.3,151206,001.0,E*7A
$GPGGA,195900,4651.1819,N,00930.7534,E,1,08,1.1,583.0,M,48.0,M,,*44
$GPGSV,3,1,10,01,43,053,50,04,09,306,00,11,55,151,52,13,13,208,44*7F
$GPRMC,195910,A,4651.1820,N,00930.7532,E,000.0,298.3,151206,001.0,E*77
$GPGGA,195910,4651.1820,N,00930.7532,E,1,08,1.1,583.2,M,48.0,M,,*4B
$GPGSV,3,2,10,14,01,038,00,17,38,289,50,20,78,317,50,23,45,197,52*78
$GPRMC,195920,A,4651.1819,N,00930.7533,E,000.0,298.3,151206,001.0,E*7F
$GPGGA,195920,4651.1819,N,00930.7533,E,1,08,1.1,582.8,M,48.0,M,,*48
$GPGSV,3,3,10,25,29,084,50,31,24,075,49*76
$GPRMC,195930,A,4651.1820,N,00930.7542,E,000.0,298.3,151206,001.0,E*72
$GPGGA,195930,4651.1820,N,00930.7542,E,1,08,1.1,583.5,M,48.0,M,,*49
$GPGSV,3,1,10,01,43,053,49,04,09,306,00,11,55,151,51,13,13,208,44*74
$GPRMC,195940,A,4651.1821,N,00930.7543,E,000.0,298.3,151206,001.0,E*75
$GPGGA,195940,4651.1821,N,00930.7543,E,1,08,1.1,583.4,M,48.0,M,,*4F
$GPGSV,3,2,10,14,01,038,00,17,38,289,50,20,78,317,49,23,45,197,51*73
$GPRMC,195950,A,4651.1820,N,00930.7546,E,000.0,298.3,151206,001.0,E*70
$GPGGA,195950,4651.1820,N,00930.7546,E,1,08,1.1,583.0,M,48.0,M,,*4E
$GPGSV,3,3,10,25,29,084,50,31,24,075,49*76
```

Befehle in **Scipad**:

```
// Lesen einer NMEA-GPS-Datei, Werte extrahieren und neue Datei schreiben

// Öffnen der NMEA-Datei mit Moppen(), "r" steht für reading (lesen),
// weitere Möglichkeiten: "w" für writing, "a" für appending (anhängen)
// Zum Schliessen der Datei wird mclose("DateiName.txt") oder mclose('all') verwendet
// Die Datei muss sich im Ordner, spezifiziert durch File/Get Current Directory, befinden
// sonst muss der vollständige Pfad angegeben werden
fd_NMEA = mopen("NMEA_daten_15_12_2006.txt","r");

// Lesen der Textlinien mit mgetl(Datei, m), m steht für Anzahl der zu lesenden Zeilen
// Defaultwert für m ist -1, d .h. alle Zeilen der Datei werden gelesen
Text_Linien = mgetl(fd_NMEA, -1);

// Suchen und extrahieren von einzelnen Zeilen mit grep()
// Nur die Zeilen mit dem Ausdruck $GPGGA werden in der neuen Datei geschrieben
GGA_Zeilen = Text_Linien(grep(Text_Linien,"$GPGGA"));

// Aus den einzelnen GGA-Zeilen werden die Information über die Zeit, die Breite, die Länge und die Höhe extrahiert
// Die einzelnen Werte werden in Bestandteile zerlegt, z. B. Stunde, Minute, Sekunde, Grad
// Die Extrahierung erfolgt mit dem Befehl part(Name_Zeile, (a:b])
// Name_Zeile gibt an, welche Zeile abgesucht wird, a und b geben an, welche Positionsbereichen in der Zeile extrahiert werden
// Beispiel: Die Stundenangabe beginnt bei Position 8 und endet bei Position 9
Zeit_h = part(GGA_Zeilen,[8:9]);       // Stunden-Wert der Zeit
Zeit_min = part(GGA_Zeilen,[10:11]);   // Minuten-Wert der Zeit
Zeit_sec = part(GGA_Zeilen,[12:13]);   // Sekunden-Wert der Zeit
Breite_grad = part(GGA_Zeilen,[15:16]); // Grad-Wert des Breitengrades
Breite_min = part(GGA_Zeilen,[17:23]);  // Minuten-Wert des Breitengrades
Laenge_grad = part(GGA_Zeilen,[27:29]); // Grad-Wert des Längengrades
Laenge_min = part(GGA_Zeilen,[30:36]);  // Minuten-Wert des Längengrades
Hoehe = part(GGA_Zeilen,[49:53]);      // Höhe über Meer

// Erstellen einer neuen Datei. In dieser Datei wird geschrieben
fd = mopen("Zeit_Breite_Laenge_Hoehe_bsp_2.txt", 'w');

// Schreiben und formatieren des Titels
mfprintf(fd, '\t%s\t\t %s\t %s\t %s\n', 'Zeit', 'Breitengrad', 'Längengrad', 'Höhe ü. M.');
mfprintf(fd, '\t%s\t %s\t %s\t %s\n', '==========', '==========', '==========', '======');

// Schreiben und formatieren der Zeile mit den Werten
mfprintf(fd, '\t%sh%smin%ssec\t %s°%smin\t %s°%smin\t %sm\n ', Zeit_h, Zeit_min, Zeit_sec, Breite_grad, Breite_min, Laenge_grad,
Laenge_min, Hoehe);

// Schliessen aller geöffneten Dateien
mclose('all');
```

Ausgabe in der Datei *Zeit_Breite_Laenge_Hoehe_bsp_2.txt*



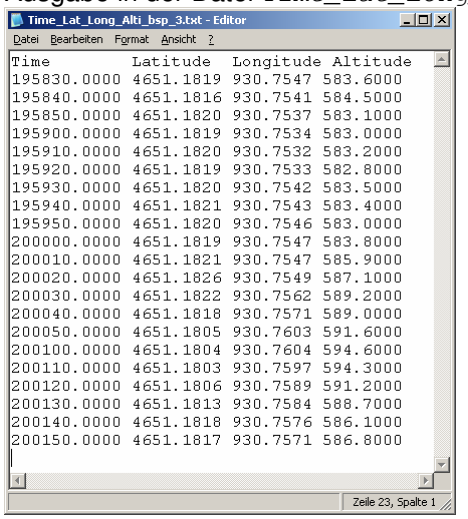
13.3.3 Beispiel 3: Auswertung von Daten und neue Anzeige (2)

Dies ist eine alternative Lösung von Beispiel 2.

Befehle in **Scipad**:

```
// Lesen einer Datei, Werte extrahieren, verarbeiten und neue Datei Schreiben
// Öffnen der NMEA-Datei mit Moppen(), "r" steht für reading (lesen),
// Die Datei muss sich im Ordner, spezifiziert durch File/Get Current Directory, befinden
// sonst muss der vollständige  Pfad angegeben werden
fd_datei = mopen("NMEA_daten_15_12_2006.txt","r");

// Lesen der Textlinien mit mgetl(Datei, m), m steht für Anzahl der zu lesenden Zeilen
// Defaultwert für m ist -1, d.h. alle Zeilen der Datei werden gelesen
alle_Zeilen = mgetl(fd_datei, -1);

// Suchen nach einzelnen Zeilen mit grep()
// Nur die Zeilen mit dem Ausdruck $GPGGA werden in der neuen Datei geschrieben
GGA_Zeilen = alle_Zeilen(grep(alle_Zeilen,"$GPGGA"));

// Aus den einzelnen GGA-Zeilen werden die Information über die Zeit, die Breite, die Länge und die Höhe extrahiert
// Die Extrahierung erfolgt mit dem Befehl part(Name_Zeile, (a:b])
// Name_Zeile gibt an, welche Zeile abgesucht wird. a und b geben an, welche Positionsbereichen in der Zeile extrahiert werden
// Beispiel: Die Breiteangabe beginnt bei Position 15 und endet bei Position 23
time = part(GGA_Zeilen,[8:13]);   // Extrahieren der Zeit
lat = part(GGA_Zeilen,[15:23]);   // Extrahieren der Breite
lon = part(GGA_Zeilen,[27:36]);   // Extrahieren der Länge
alt = part(GGA_Zeilen,[49:53]);   // Extrahieren der Höhe

// Schreiben der Matrix mit den Werten in einer neuen Datei.
// Da die Werte für Zeit, Breite, Länge und Höhe als Zeichenkette (=String) vorliegen, müssen die Zeichen in Zahlen-Werte
// umgewandelt werden.
// Die Umwandlung erfolgt mit der Funktion evstr()
fprintfMat("Time_Lat_Long_Alti_bsp_3.txt",evstr([time lat lon alt]), '%8.4f', 'Time        Latitude  Longitude Altitude' );

// Schliessen aller geöffneten Dateien
mclose('all');
```

Ausgabe in der Datei *Time_Lat_Long_Alti_bsp_3.txt*



14 Der grafische Simulator Scicos

Um was geht es in diesem Kapitel?
Dieser Abschnitt erklärt die Grundlagen und die Anwendung des grafischen Simulators **Scicos**. Einfache Schaltungen werden aufgebaut und simuliert. Die Hilfefunktionen werden erklärt und das Aktualisieren des Programms demonstriert.

Was werden Sie erreichen?
Sie werden in der Lage sein, einfache Modelle zu zeichnen und zu simulieren. Wenn Sie nicht mehr weiter wissen werden Sie der Lage sein, die umfangreichen Hilfefunktionen zu verwenden.

Was müssen Sie tun?
Sie sollten den Text lesen und die aufgeführten Beispiele durcharbeiten und nachvollziehen.

Wichtiger Hinweis:
Bevor Sie **Scicos** einsetzen, sollten Sie überprüfen, ob Ihre **Scicos**-Version aktualisiert ist und alle Update-Patches installiert sind.
Auf der Homepage von **Scicos** http://www.Scicos.org/ finden Sie vielleicht ähnliche Mitteilungen wie die nachfolgende:

> **News**
> **2006-04-12** Due to an error in the **Scicos** compiler, in some exceptional circumstances, the order of block executions is not computed correctly. This could lead to simulation errors. To fix the problem, download and use this patch.

Wenn nötig installieren Sie die notwendigen Patches gemäss den im Patch beigefügten Anleitungen. Die Patches bestehen aus komprimierten Dateien, eine davon ist die Installationsanleitung.

14.1 Einleitung

Scicos (**Scilab** Connected Object Simulator, mit **Scilab** verbundener Modell Simulator) ist ein **Scilab**-Paket für das Modellieren und Simulieren von dynamischen Systemen. **Scicos** beinhaltet einen grafischen Editor, um Modelle zu konstruieren. Für die Konstruktion eines Modells werden Blöcke (vordefinierte Basisfunktionen oder vom Benutzer definierte Funktionen) zusammengeschaltet.

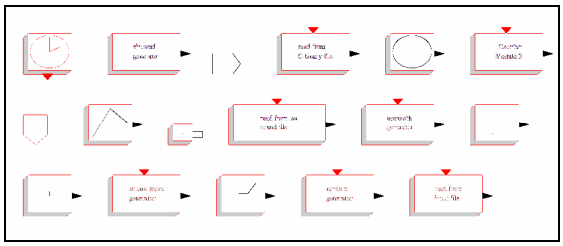
Die Konstruktion und Simulation eines **Scicos**-Modells besteht gewöhnlich aus den folgenden sechs Schritten:

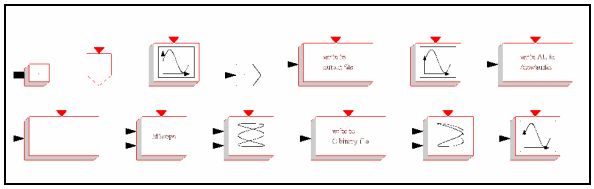
1. **Scicos** aus **Scilab** starten
2. Paletten wählen
3. Platzieren der Blöcke auf der Zeichnungsoberfläche
4. Blöcke untereinander verbinden
5. Block-Parameter einstellen
6. Simulationsparameter einstellen
7. Simulation starten

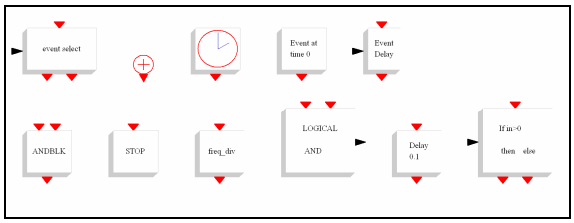
Beispiel zum Veranschaulichen eines Simulationsablaufs:
Das Signal eines einfachen Sinusgenerators wird auf einem Oszilloskop anzeigt.

1. **Scicos aus Scilab starten**
   Im **Scilab**-Menü ***Applications ⇨ Scicos*** wählen oder durch den Befehl `Scicos();` auf der **Scilab**-Oberfläche eingegeben. Die Oberfläche von **Scicos** öffnet sich.

2. **Paletten wählen**
   Im **Scicos**-Menü ***Edit ⇨ Palletes ⇨ Sources*** , ***Sinks*** und ***Events*** anklicken. Folgende drei Paletten sind sichtbar:

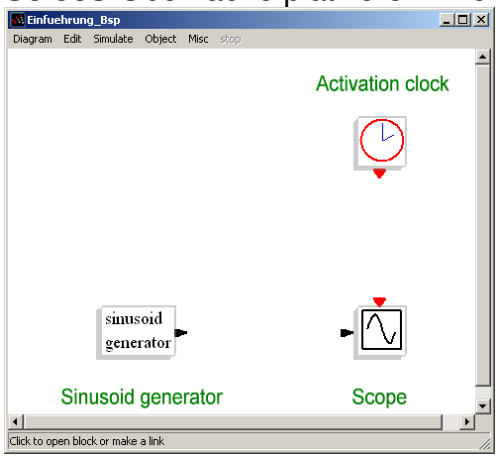

Palette Sources


Palette Sinks



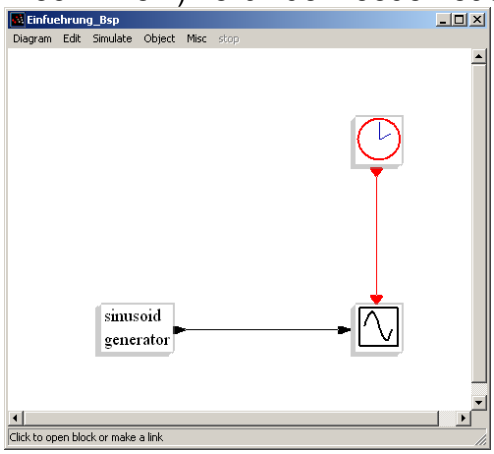
Palette Events

3. **Platzieren der Blöcke auf der Zeichnungsoberfläche.**
   Aus der Palette ***Sources ⇨ Sinusoid generator***, aus der Palette ***Sinks ⇨ Scope*** und aus der Palette ***Events ⇨ Activation clock*** anklicken und auf der **Scicos**-Oberfläche platzieren. Die **Scicos**-Oberfläche sieht folgendermassen aus:



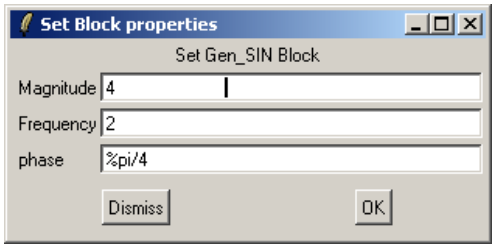
4. Blöcke untereinander verbinden

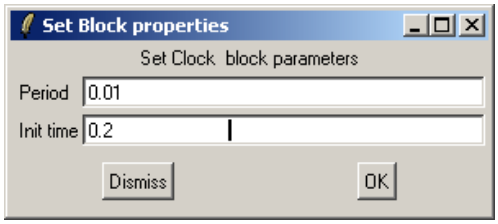
Aus dem **Scicos**menü ***Edit ⇨ Link*** anklicken und die drei Blöcke von einem Ausgang (Pfeil zeigt aus dem Block nach Aussen) zu einem Eingang (Pfeil zeigt in den Block hinein) verbinden. Jede neue Verbindung muss im Menü neu gewählt werden.

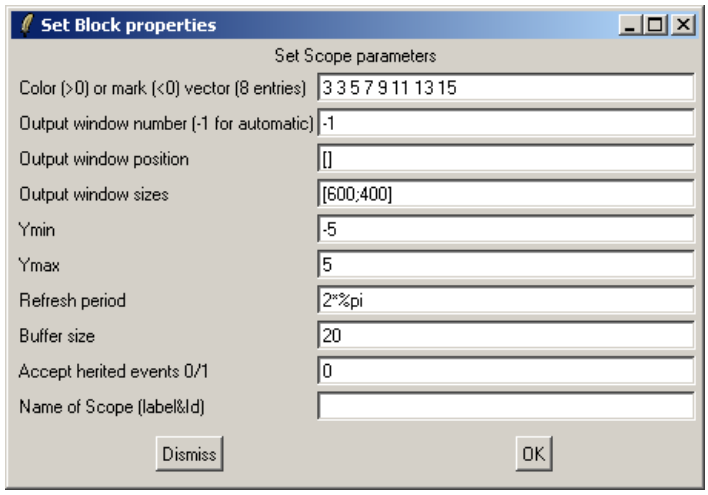


5. Block-Parameter einstellen

Auf der **Scicos**-Oberfläche jeweils mit der linken Maustaste auf die drei Blöcke klicken und die entsprechenden Werte einstellen. Die Einstellungen jeweils mit *OK* bestätigen und abschliessen.







Erklärungen zu den Einstellungen des Sinusgenerators:

Magnitude = Amplitude des Signals, **Frequency** = Frequenz des Signals, **phase** = Nullphase des Signals (in radiant, hier π/4).

Erklärungen zu den Einstellungen des Activation clocks:

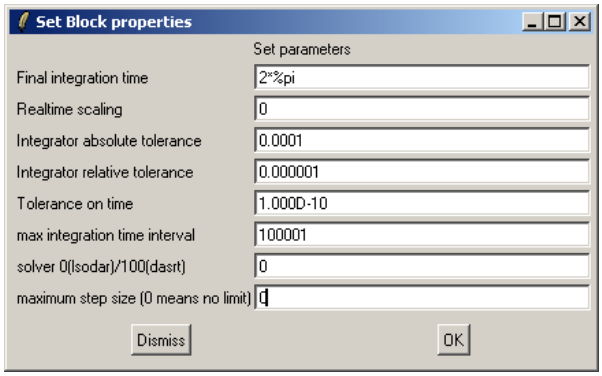
Period = Häufigkeit einer Messauslösung (je kleiner der Wert, desto mehr Punkte werden berechnet bzw. angezeigt), **Int time** = Zeit, die zu Beginn verstreicht, bis die erste Messung ausgelöst wird.

Erklärungen zu den wichtigsten Einstellungen des Scopes:

Color = Farbe der einzelnen Bildschirmstrahlen (hier wurde der erste und einzige Strahl auf 3/Grün eingestellt), **Ymin bzw. Ymax** = Anzeigebereich des Oszilloskopes, **Refresh period** = Anzeigebereich der x-Achse (hier 2π).

6. Simulationsparameter einstellen

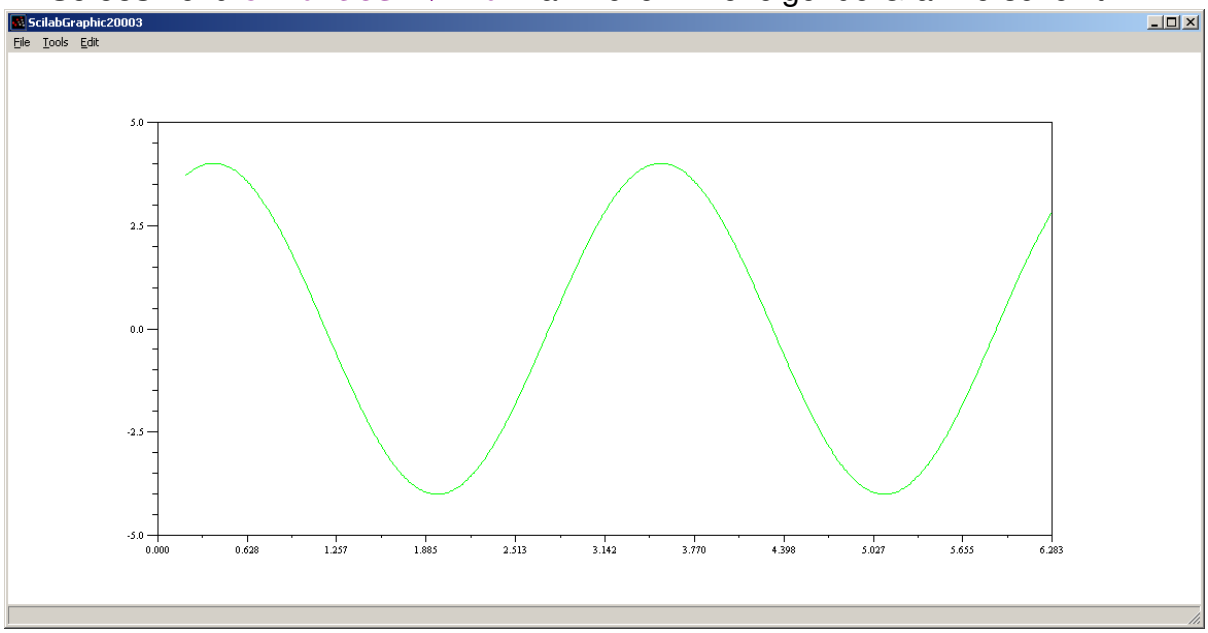
Im **Sicos**-menü ***Simulate ⇨ Setup*** anklicken und folgende Einstellungen vornehmen:



Wichtig ist die `Final integration time` (=2π). diese Simulationszeit sollte der `Refresh period` bei der Einstellung des Scopes entsprechen. Die Einstellungen mit *OK* abschliessen.

7. Modell simulieren

Im **Scicos**menü ***Simulate ⇨ Run*** anklicken. Die folgende Grafik erscheint:

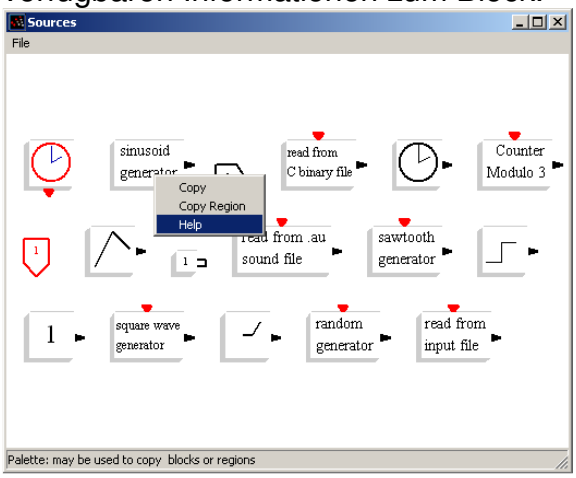


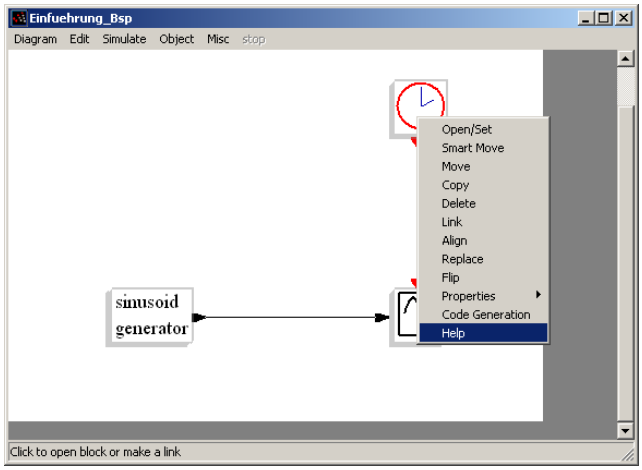
14.2 Hilfe zu Scicos

Scicos besteht aus einer Unmenge von Blöcken und Funktionen, und es würde den Rahmen dieser Anleitung sprengen, auf sämtliche Möglichkeiten einzugehen.
Wenn Sie mehr wissen möchten, bietet **Scicos** bzw. **Scilab** verschiedene Hilfemöglichkeiten.

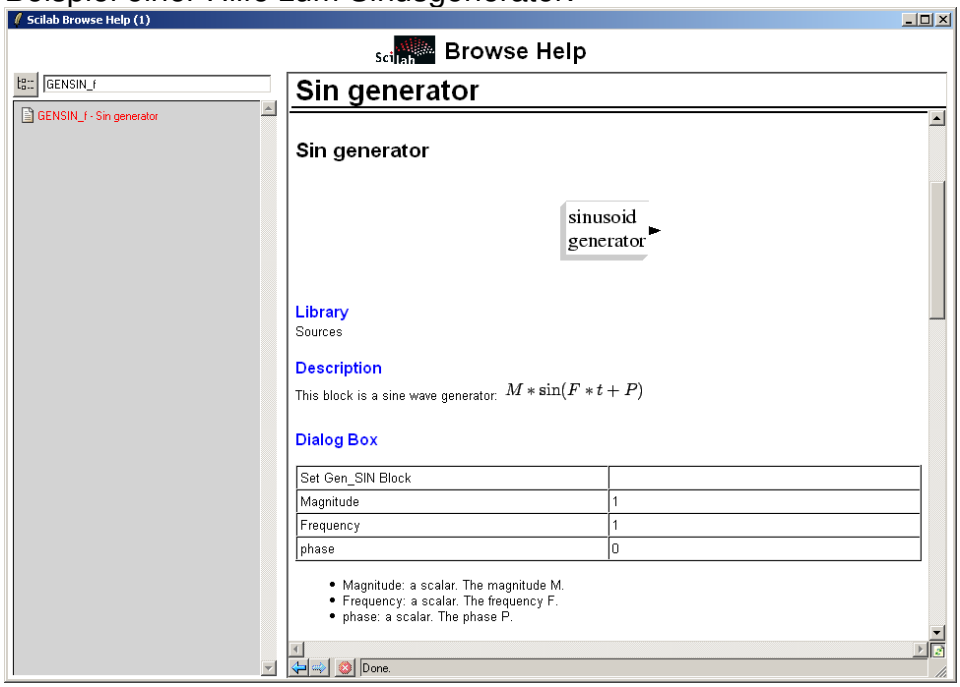
14.2.1 Hilfe zu den einzelnen Blöcken

Wird in einer geöffneten Palette oder auf der **Scicos**-Oberfläche mit der rechten Maustaste auf einen Block geklickt, erscheint ein kleines Menü. Wählen Sie `Help` und Sie erhalten alle verfügbaren Informationen zum Block.



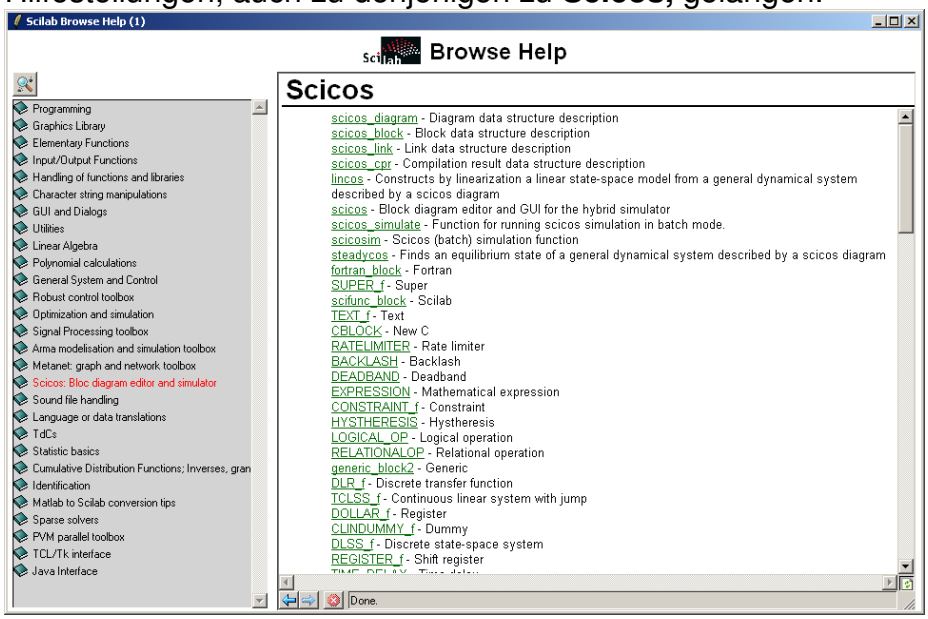


Beispiel einer Hilfe zum Sinusgenerator:



14.2.2 Hilfe aus Scilab

Im Hilfemenü von **Scilab** `? ⇨ Scilab Help F1` können Sie zu sämtlichen Hilfestellungen, auch zu denjenigen zu **Scicos,** gelangen.



14.2.3 Hilfe aus der Scicos-Homepage

Auf der **Scicos**-Homepage http://www.Scicos.org/index.html#documentation finden Sie ebenfalls Ratschläge und Hilfe:

Documentation

Note: some of these documents are old. You can find up-to-date information in Scilab's help pages and Scicos demos in your Scilab software directory under Scilab-.../demos/scicos.

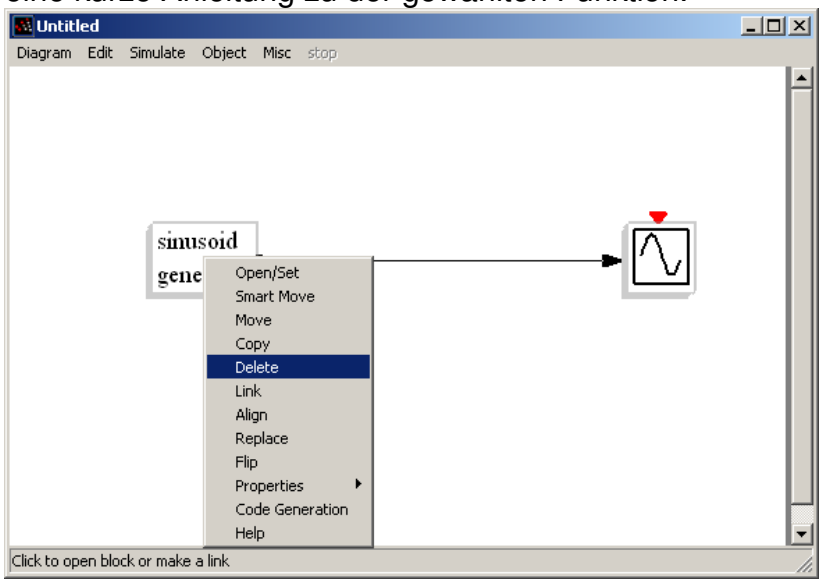
- **Books**
  - New Book (in German): "**Grundlagen digitale Regelung und Mechatronik simulation mit SCILAB/SCICOS**"
- **Examples**
  - Examples [from the book *Modeling and Simulation in Scilab/Scicos*]
- **Tutorials**
  - Sample chapter [from the book *Modeling and Simulation in Scilab/Scicos*]
  - Scicos tutorial Wiki (in French).
  - Old Scicos tutorial
  - wolffdata.se Tutorials by Peter Wolff on Control Systems with Scilab/Scicos, GPS data processing, serial port interfacing
- **Presentations**
  - Scicos presentation
- **Reference**
  - Scilab's help pages
  - Scicos online reference
  - Brief Scicos user guide (15 pages)

14.3 Ein Blockschema gestalten

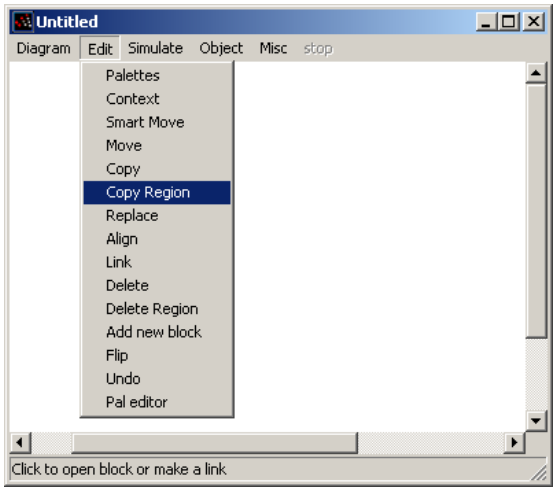
Blöcke können:

- Eingestellt werden (Open/Set)
- Verschoben werden (Smart Move, Move)
- Kopiert werden (Copy)
- Gelöscht werden (Delete)
- Verbunden werden (Link)
- Automatisch auf eine Linie gebracht werden (Align)
- Ersetzt werden (Replace)
- Um 180° gedreht werden (Flip)

Um eine Aktion auszulösen, müssen Sie mit der rechten Maustaste auf den Block klicken und die gewünschte Funktion auswählen. In der untersten Zeile des Bildschirms finden Sie eine kurze Anleitung zu der gewählten Funktion.

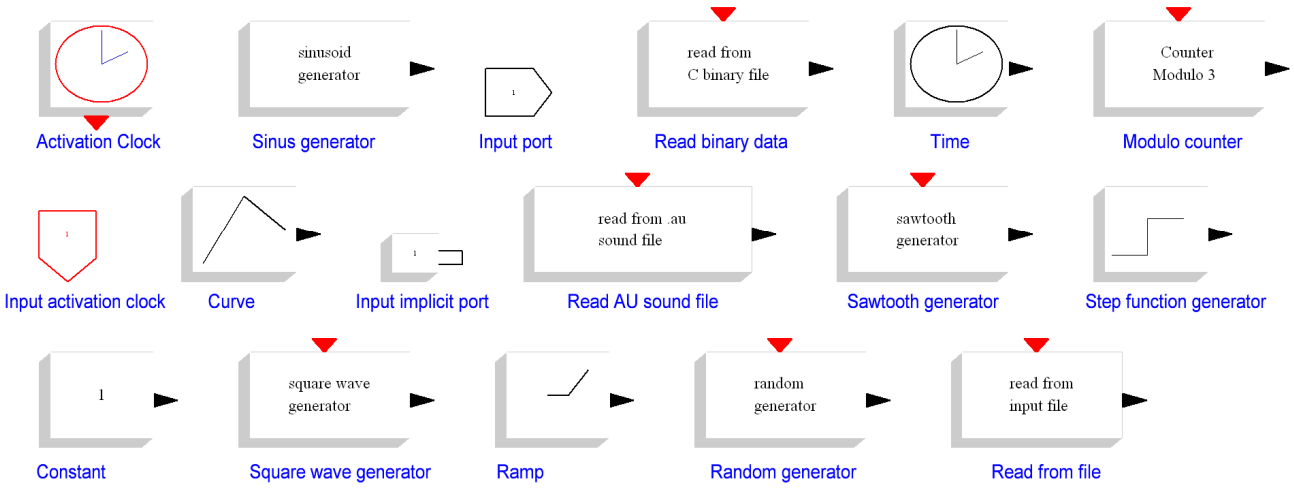


Im **Scicos-**menü ***Edit*** können noch zusätzliche Funktionen (z. B. Copy Region, Undo etc.) aktiviert werden:

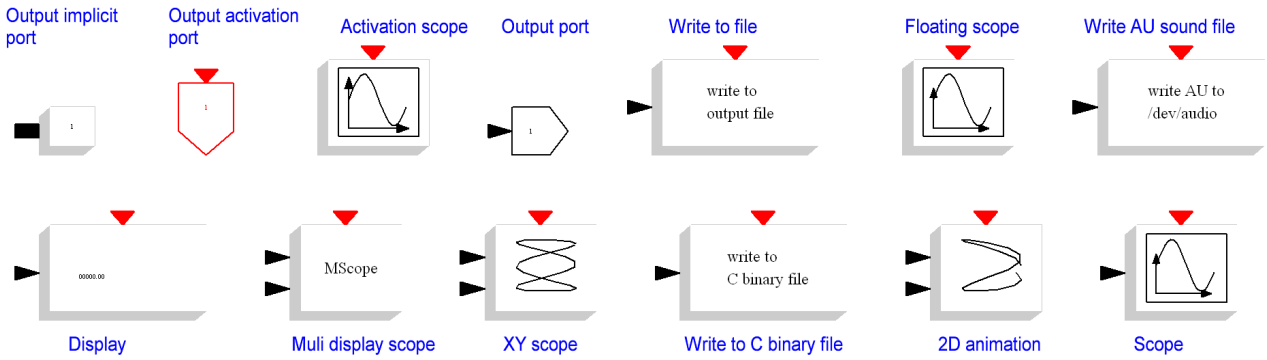


14.4 Die einzelnen Paletten

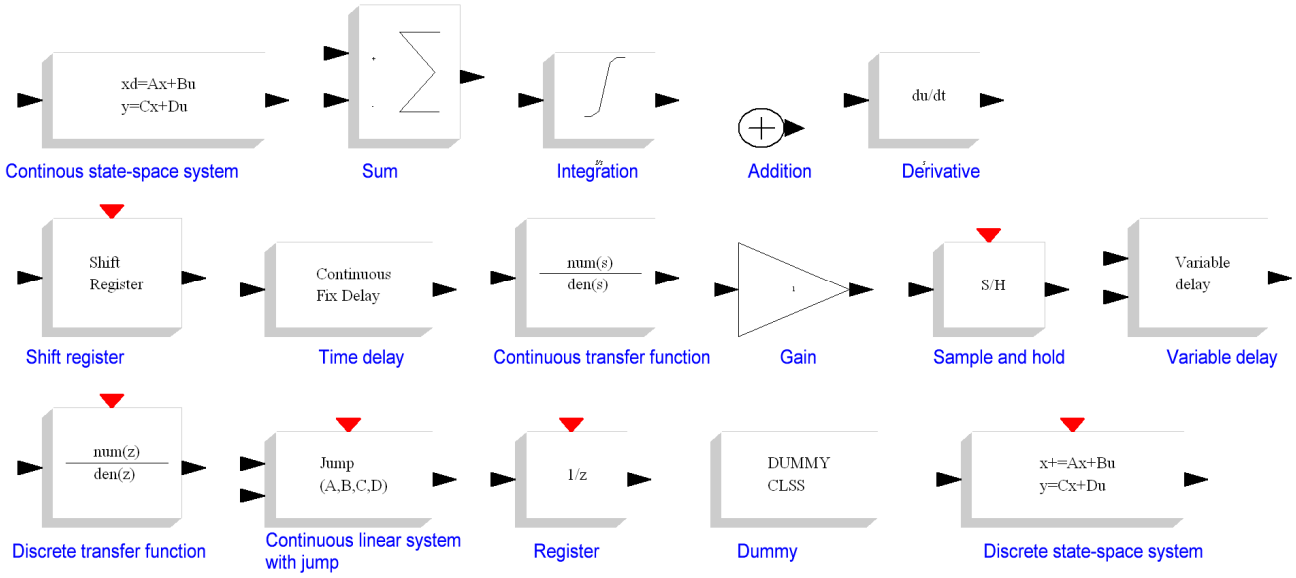
14.4.1 Die Sources-Palette



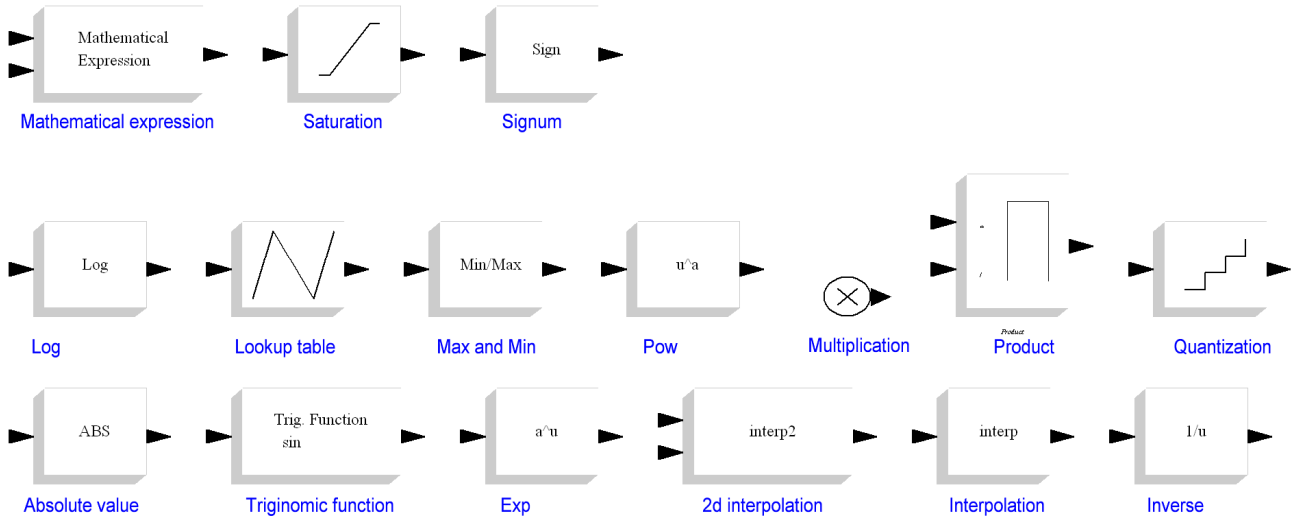
14.4.2 Die Sinks-Palette



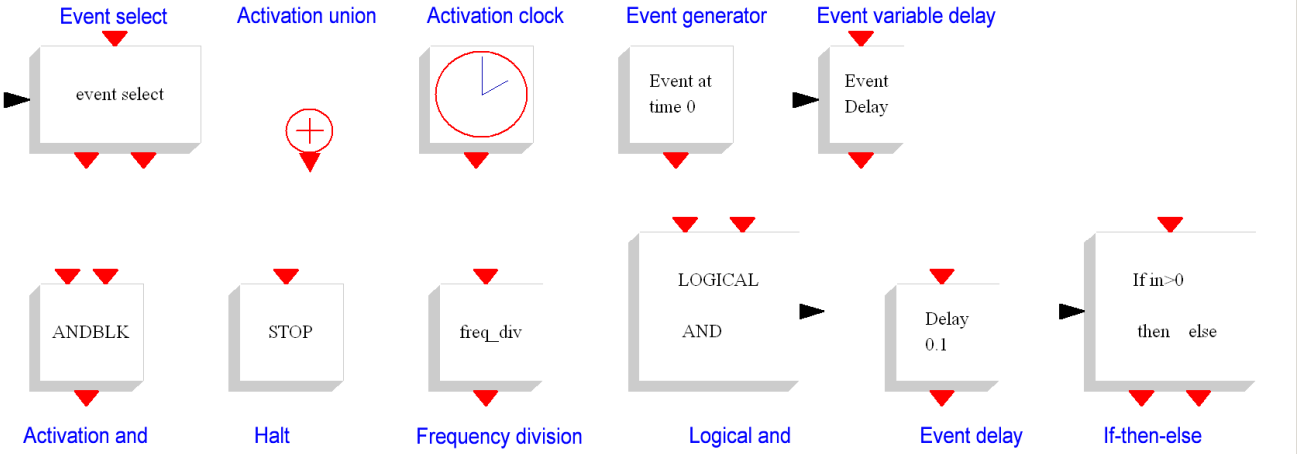
14.4.3 Die Linear-Palette



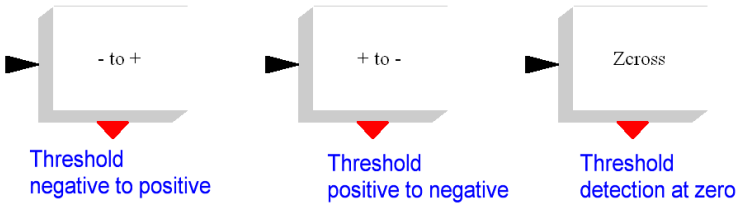
14.4.4 Die Nonlinear-Palette



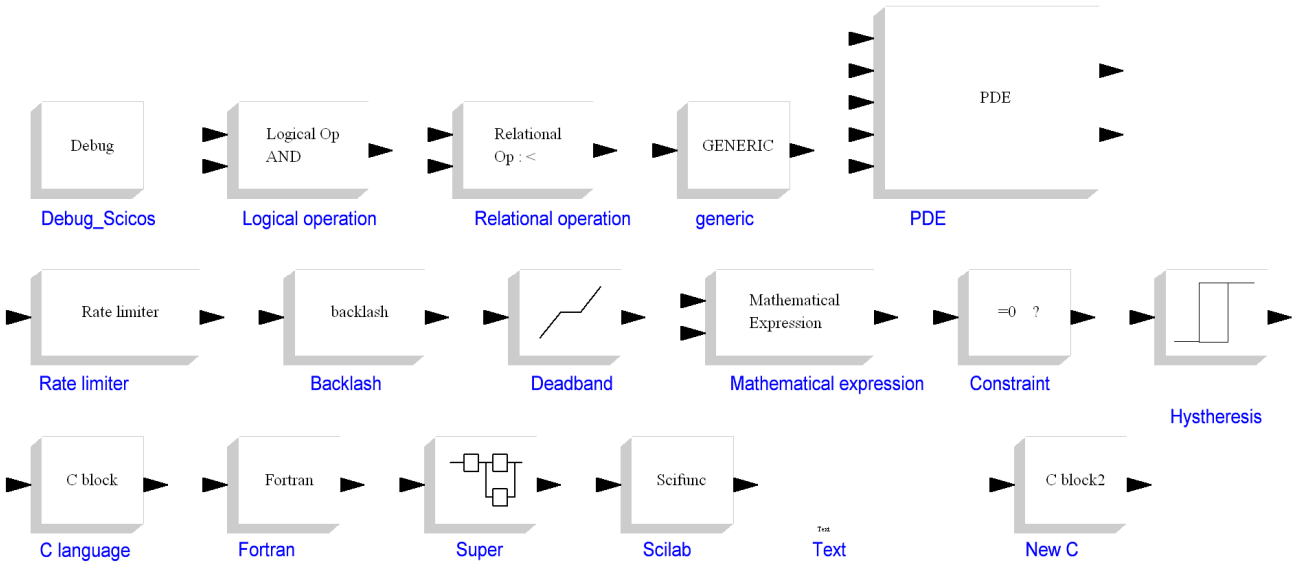
14.4.5 Die Events-Palette



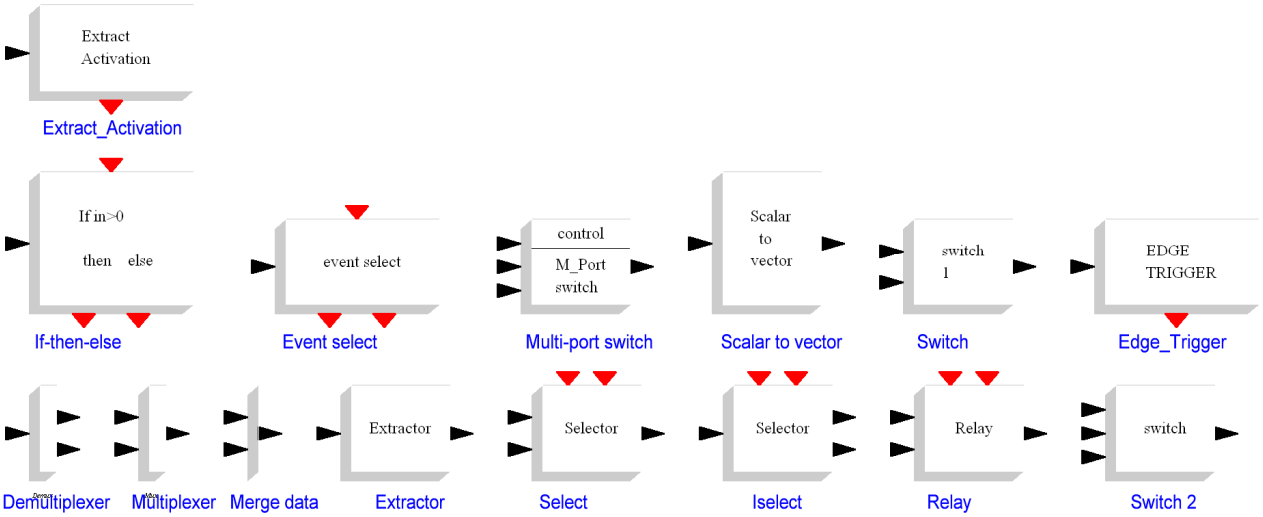
14.4.6 Die Threshold-Palette



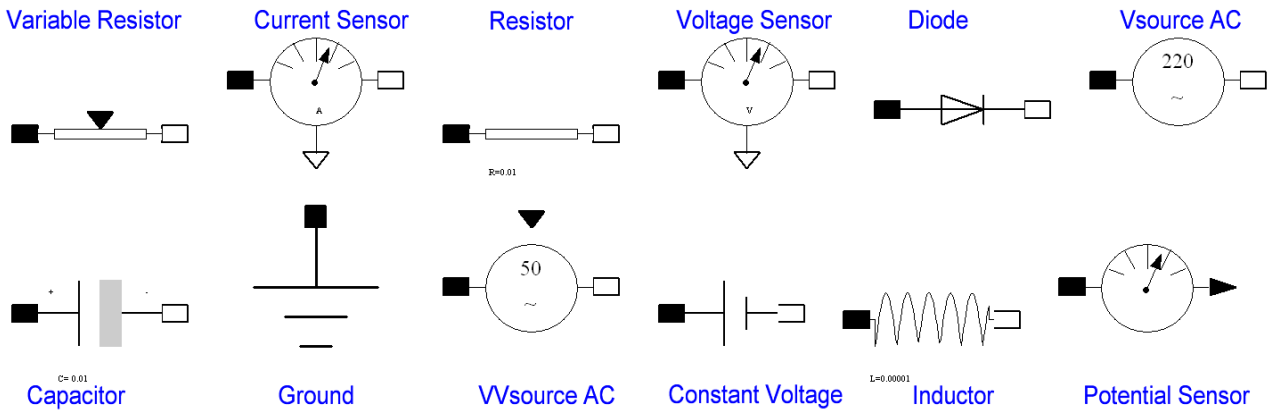
14.4.7 Die Others-Palette



14.4.8 Die Branching-Palette



14.4.9 Die Electrical-Palette

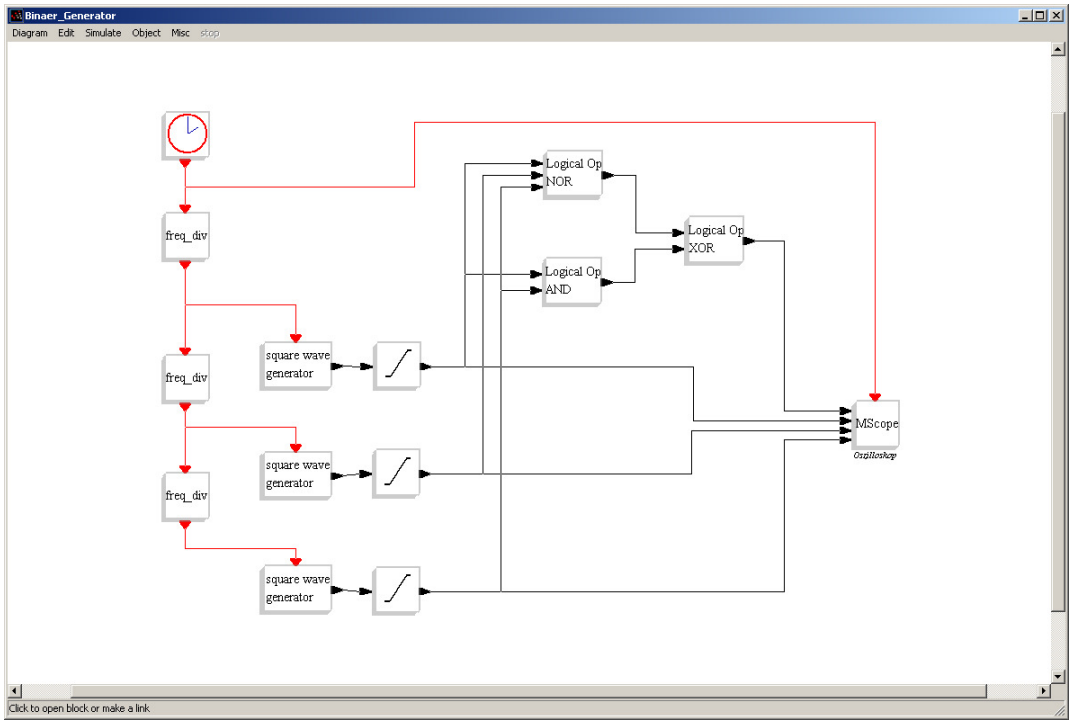


14.5 Beispiele

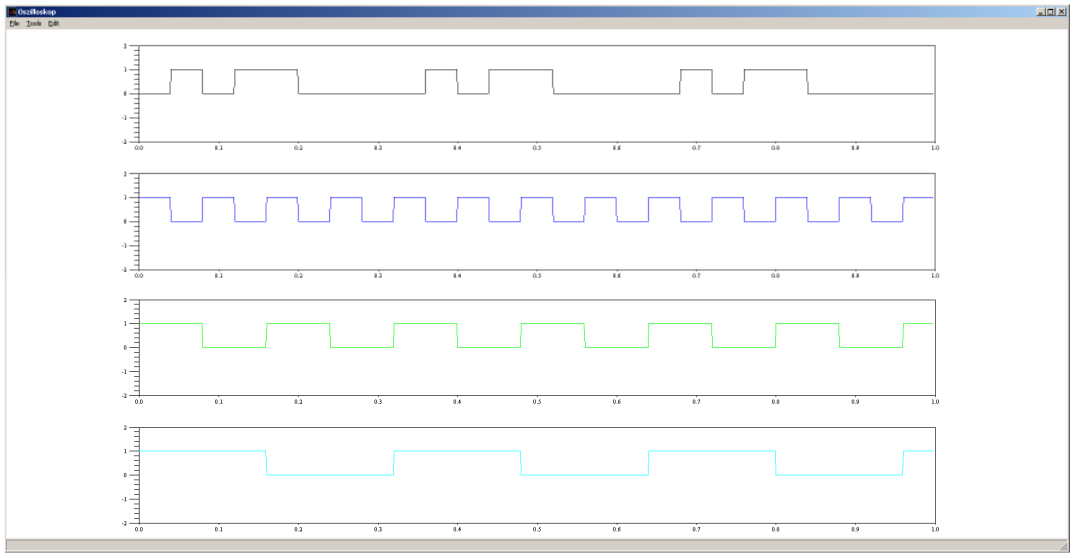
Alle hier gezeigten Beispiele können herunter geladen werden (www. …?????……). Dies kann sinnvoll sein, um die eingestellten Parameter einzusehen.

14.5.1 Beispiel aus der Digitaltechnik

Simuliert wird eine digitale kombinatorische Schaltung mit einem NOR, einem AND und einem EXOR.

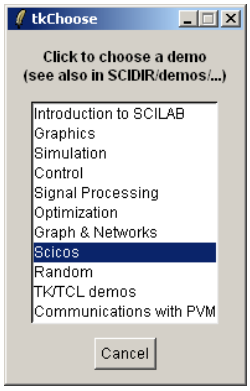


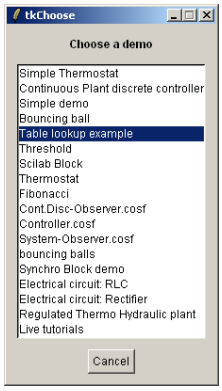
Die Simulation der Schaltung ergibt folgendes Bild. Hinweis: Die Diagramme erscheinen in der gleichen Reihenfolge wie die Signale am MScope (Oszilloskop) angeschlossen worden sind.



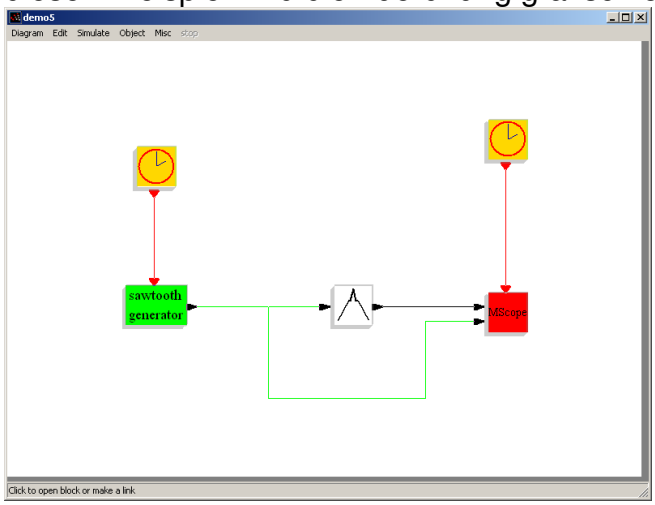
14.5.2 Allgemeines Beispiel: Anwendung einer Lookup-Tabelle

Dieses Beispiel stammt aus den **Scilab**-Demonstrationsbeispielen.
Im **Scilab**-menü `? ⇨ Scilab Demos ⇨ Scicos` können viele Beispiele geladen werden.

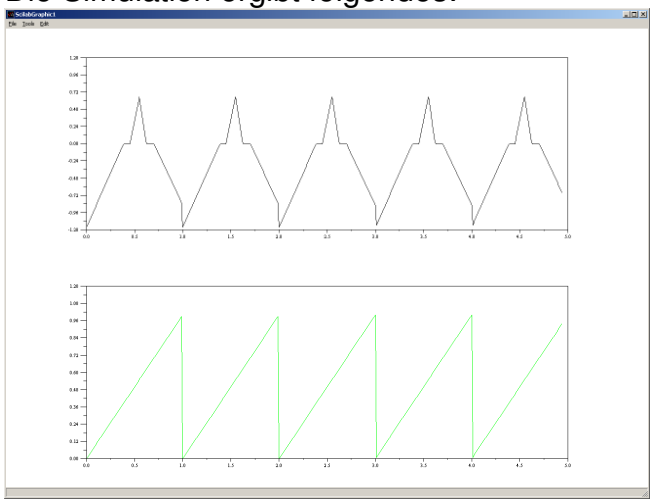




Wir wählen hier das Beispiel `Table lookup example`.
Eine Lookup-Tabelle ist eine tabellarische Zuordnung eines Ausgangs zu einem Eingang. In diesem Beispiel wird die Zuordnung grafisch erstellt.

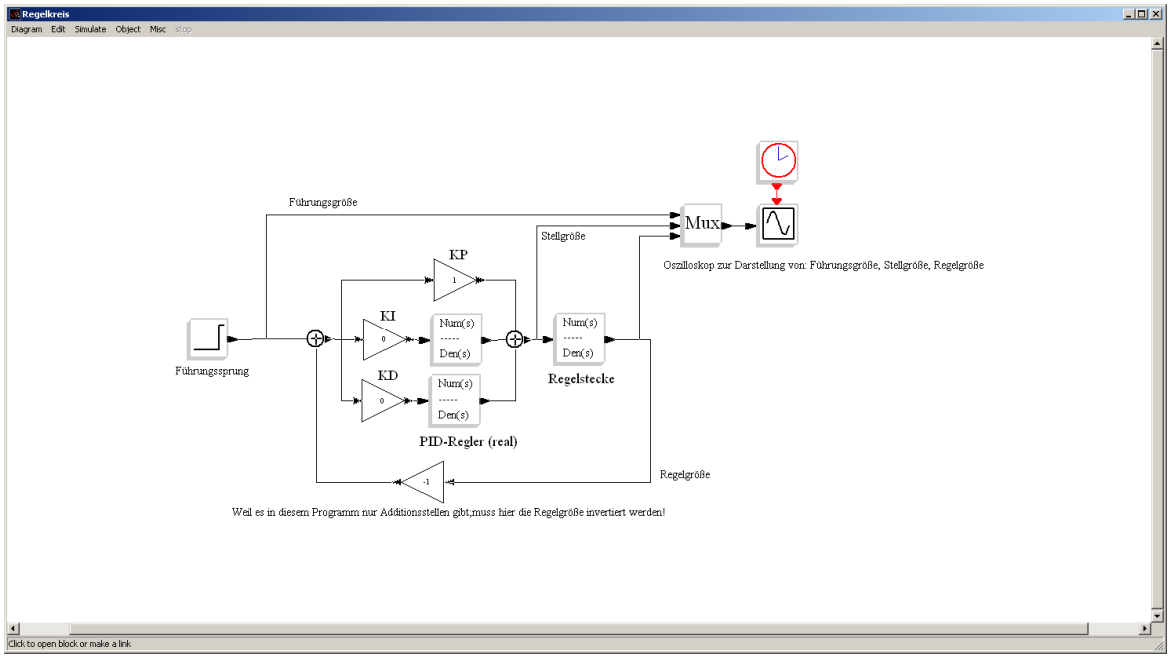


Die Simulation ergibt folgendes:

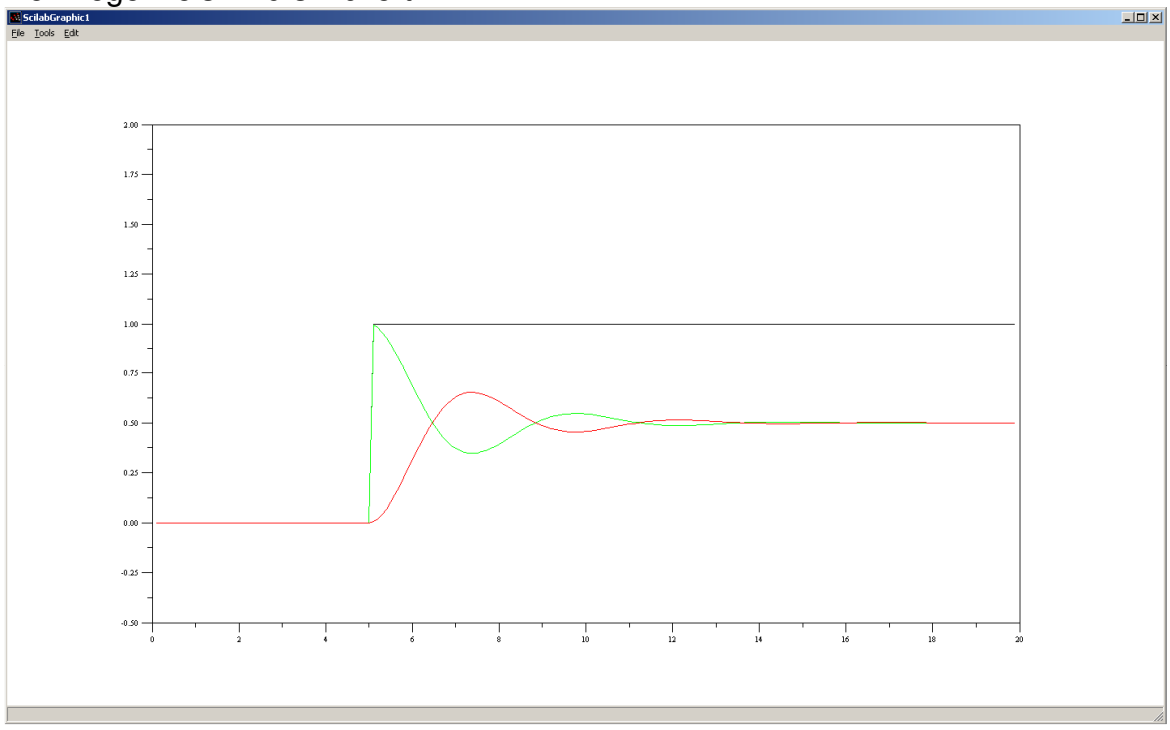


14.5.3 Beispiel aus der Regelungstechnik: Regelkreis mit PID-Regler

Dieses Beispiel wurde aus http://www.mb.uni-siegen.de/d/imr5/Scilab.htm kopiert. Auf dieser Seite des Fachbereichs Maschinenbau der Universität Siegen können noch weitere wertvolle Informationen gefunden werden.



Der Regelkreis wird simuliert:



15 Stichwortverzeichnis

F

G

H

I

K

L

M

U

V

W

X

Z