Einführung in

SCILAB/XCOS 5.4

Januar 2013

Helmut Büch

Dieser Arbeit liegt die

Introduzione a Scilab 5.3
Versione 2.3 Gennaio 2011
von Gianluca Antonelli und Stefano Chiaverini
Dipartimento di Automazione, Elettromagnetismo,
Ingegneria dell'Informazione e Matematica Industriale
Università degli Studi di Cassino, Italia

zugrunde.

Nach meiner Meinung hätten Gianluca Antonelli und Stefano Chiaverini als Autoren genannt werden müssen. Herr Prof. Antonelli hat aber darauf bestanden, dass ich als Autor firmiere. Das hat u.a. zur Folge, dass nun alle Fehler auf meine Kappe gehen. Vorschläge zur Verbesserung dieser Einführung sind willkommen auf Helmut@Buech-Gifhorn.de.

Ich habe die Arbeit der Übersetzung und der Anpassung an die Version 5.4.0 samt einiger kleinerer Ergänzungen auf mich genommen, weil ich denke, dass Scilab im allgemeinen und Xcos im besonderen eine größere Aufmerksamkeit und Verbreitung verdient. Das gilt nicht zuletzt für die regelungstechnischen Möglichkeiten von Scilab/Xcos, die die italienischen Autoren in ihrer Introduzione hervorgehoben haben.

Inhaltsverzeichnis

1 Einführung

1.1 Was Scilab ist

Scilab ist ein Programm für numerische Rechnungen in wissenschaftlichen und technischen Anwendungen. Scilab ist open source und seit 1994 als ausführbares Programm sowie als Quellcode kostenfrei verfügbar. Es findet Verwendung in der Wissenschaft, in Universitäten und in Unternehmen. Es kann nicht-kommerziellen Zwecken dienen und in bestimmten Grenzen auch kommerziellen.

Scilab enthält Hunderte von mathematischen Funktionen und die Möglichkeit, die eigene - höhere - Programmiersprache einzusetzen; erlaubt aber ebenso den Einsatz von Punktionen, die in einer anderen Sprache geschrieben sind wie z.B. C, C++ oder Fortran. Die verfügbaren Funktionen ermöglichen Operationen wie

- 2D- und 3D-Grafik und Animation
- Lineare Algebra und Matrizen
- Polynome, rationale Funktionen
- Interpolation, Approximation
- Simulationen: Löser ODE und DAE
- Xcos: grafischer Simulator für dynamische Systeme (hieß in früheren Versionen Scicos)
- klassische und robuste Regelung für dynamische Systeme LMI
- Optimierung
- Theorie der Zeichen
- Graphen und Netze
- Statistik
- Maple-Interface zur Erzeugung von Scilab-Code
- Interfaces für Fortran, Tcl/Tk, C, C++, Java, LabVIEW
- eine gewachsene Zahl von Paketen, die von Anwendern für Scilab geschrieben sind

1.2 Wie man Informationen zu Scilab und das Programm bekommt

Die Hauptseite von Scilab ist über die Adresse http://www.scilab.org zu erreichen. Von dieser URL kann man für die Betriebssysteme GNU/Linux, Windows und Ma-cOSX die neueste stabile Version des Programms herunterladen (ausführbar und als Quellcode). Auf derselben Seite findet man auch die Links zu älteren und zur Entwicklungsversion sowie zur Dokumentation. Dann gibt es Links zu weiteren Dokumentationen, freien wie auch kostenplichtigen, zu Toolboxen, zur Xcos-Seite, zur LabVIEW-Seite und zur Newsgroup comp.soft-sys.math.scilab. Noch eine nützliche Seite mit Dokumentation ist http://www.scilab.org/support/dociimentation.

1.3 Ein wenig Geschichte

Das Scilab-Projekt entsteht zu Jahresbegin 1980 bei INRIA (www.inria.fr) in Frankreich, angeregt durch das MIT-Projekt Matlab, seinerzeit noch public domain. Es wird in den ersten Jahren noch unter einem anderen Namen kommerziell verwertet und wird Anfang der 90er Jahre unter dem Namen Scilab zu open source erklärt.

Seitdem wird Scilab kontinuierlich gepflegt und es wird auch der grafische Simulator für dynamische Systeme entwickelt: Scicos, das mit einem großen Teil seiner Grafik-und Optimierungsbefehle Simulink von Matlab ähnelt.

Eine erste Teilung in der Entwicklung erfolgt 2005 mit Einführung einer auf Java basierenden Version, d.h. ab Release 5. Es wird entschieden, eine stark auf Release 4 basierende Version zu pflegen, die ScicosLab genannt wird. Insbesondere pflegt Scicos-Lab die Version Scicos, während Scilab 5.x Xcos entwickelt.

1.4 Scilab und die anderen

Es existieren verschiedene Programme für numerische Rechnungen, z.B.

- ScicosLab (http://www.scicoslab.org);
- Matlab (http://www.mathworks.com);
- Octave (http://www.octave.org);
- Freemat (http://freemat.sourceforge.net);
- Euler Math Toolbox (http://euler.rene-grothmann.de/).

In vielen Newsgroups, auch auf der Scilab-Seite selbst, kann man Kommentare, Vergleiche und numerische Leistungstests dieser Programme finden. Die Bezugsgröße für numerisches Rechnen ist das kommerzielle Programm Matlab von der Firma Mathworks. Scilab stellt auch Funktionen für die automatische Übersetzung von Matlab-Code in Scilab-Code zur Verfügung und eine Gegenüberstellung einiger der wichtigsten Funktionen. Weitere Einzelheiten kommen in Kapitel 6 zur Sprache.

1.5 Gibt es Scilab wirklich kostenlos?

Scilab ist gratis. Es wird nicht mit einer GPL-Lizenz (http://www.gnu.org/license) verteilt, sondern mit einer Lizenz CeCILL (http://www.cecill.info/index.en.html), die die GPL-Lizenz an das französische Recht anpasst. Scilab ist open source, wofür eine Lizenz an der Adresse http://www.scilab.org/legal/license.html verfügbar ist.

2 Programmieren in Scilab

2.1 Installieren von Scilab, Starten und Beenden des Programms

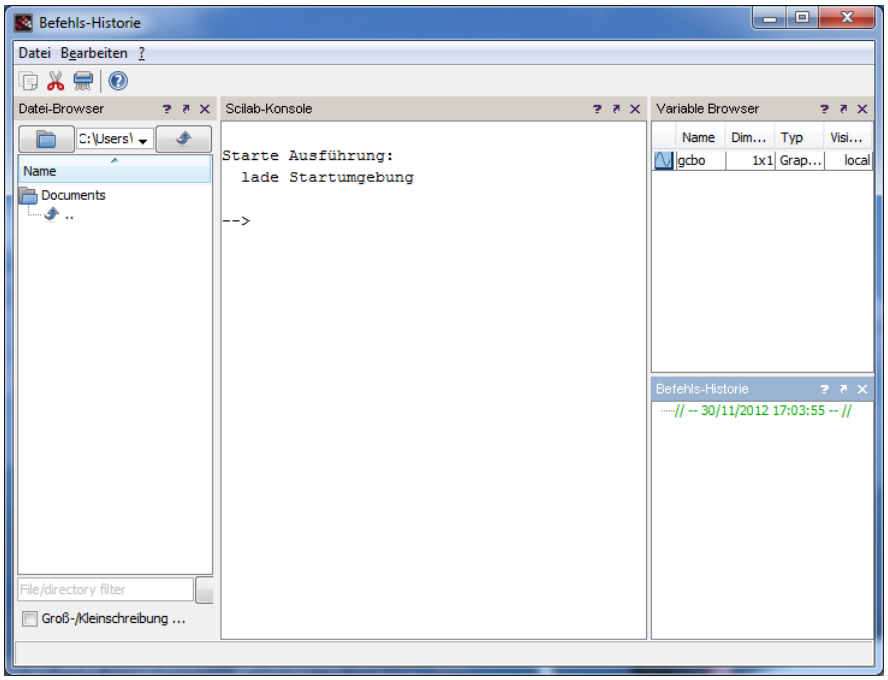

Abb. 2.1: Das Startfenster von Scilab

2.1.1 Das Arbeitsverzeichnis

Nach dem Start arbeitet Scilab in einem vordefinierten Verzeichnis, unter Windows kann das je nach Version „Eigene Dateien“ oder „Dokumente“ sein; unter Linux ist es das Verzeichnis $HOME. Um das zu prüfen, genügt die Eingabe des Befehls pwd. Alle Operationen mit einer Anwenderdatei, die einen Lese- oder Schreibvorgang auf der Festplatte bedingen, z.B. das Speichern oder Ausführen eines Anwenderskripts, werden im aktuellen Verzeichnis ausgeführt.

Das Arbeitsverzeichnis kann durchaus in ein anderes Verzeichnis verlegt werden. Es ist aber nicht ratsam, die eigenen Dateien mit denen des Programms zu vermengen.

Unter Linux kann die Verlegung z.B. so erfolgen:

```
--> cd /home/meinname/meinscilabdir
```

Wenn der Pfadname Leerzeichen enthält, ist die zu verwendende Syntax

```
--> cd '/home/meinname/mein scilab dir'
```

Dieselbe Operation kann unter Windows und Linux im Menü Datei -> aktuelles Verzeichnis ändern... ausgeführt werden.

2.2 Die Hilfe

Es gibt verschiedene Möglichkeiten, Syntax und Semantik der Scilab-Befehle in Erfahrung zu bringen. Außer dem Gebrauch von Handbüchern, wie dem hier vorliegenden, kann man mit dem Befehl `help` Informationen *online* erhalten. Ein neues Fenster öffnet sich wie Abb. 2.2 zeigt.

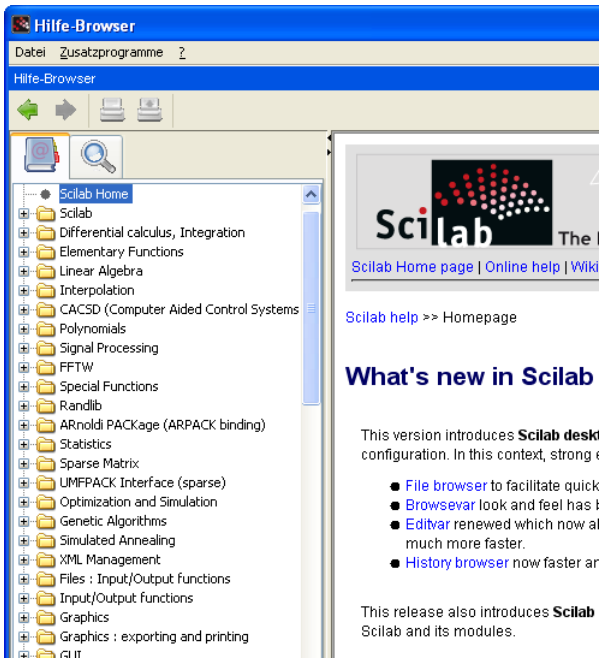

Abb. 2.2: Startfenster des Befehls `help`

Im Hilfe-Fenster kann man nach Befehlen suchen, die nach Sachgebieten geordnet sind. Man kann auch direkt zur Beschreibung eines einzelnen Befehls gelangen, indem man seinen Namen anklickt. Dasselbe erhält man auch, wenn man am Prompt eingibt

```
-->help <Name des Befehls>
```

Auch der Befehl `apropos` kann nützlich sein. Er sucht nach allen Befehlen, in denen ein bestimmtes Schlüsselwort vorkommt, zum Beispiel öffnet

```
-->apropos exponential
```

ein Fenster wie in Abb. 2.3, in dem die Befehle nach der Häufigkeit des Vorkommens des gesuchten Wortes in absteigender Ordnung aufgelistet sind.

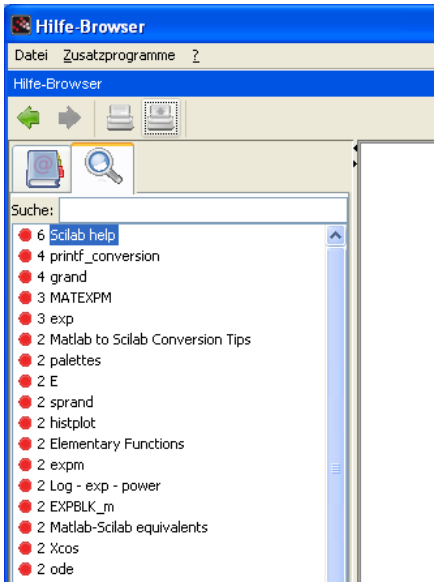

Abb. 2.3: Ergebnisse des Befehls apropos exponential

Die Seite http://www.scilab.org/support/documentation ist Ausgangspunkt für eine Fülle von Informationen. Man findet dort neben einem Matlab®/Scilab Dictionary, das in Kapitel 6 behandelt wird, u.a. auch einen Link zur offiziellen Scilab-Youtube-Seite.

Einen Verweis auf die Seite http://www.openeering.com/scilab_tutorials werden besonders die an Regelungsaufgaben interessierten Leser zu schätzen wissen.

2.3 Variablen und Arbeitsraum

Wie jedes andere Programm für numerische Rechnungen kann auch Scilab zum Rechnen eingesetzt werden, indem man die auszuführenden Operationen einfach in eine Befehlszeile schreibt. Abgesehen von solchem elementaren Gebrauch wird es immer erforderlich sein, Variablen zu definieren, Funktionen oder Programme (Skripte) und diese zu verwenden.

Generell ist jede Anweisung in Scilab eine Zeile der Art

```
--> Variablenbezeichner = Anweisung
```

worin die Ausgabe der `Anweisung` der Variablen `Variablenbezeichner` zugewiesen wird. Die Variable `ans` wird von Scilab automatisch gesetzt, sobald eine Anweisung geschrieben wird, ohne eine Variable für die Ausgabe anzugeben.

Jeder Variablenbezeichner besteht aus alphanumerischen Zeichen. Das erste darf keine Zahl sein, Sonderzeichen sind außer dem Unterstrich nicht erlaubt. Also sind `name`, `name01`, `name01_a` gültige Variablenbezeichner, während `01name` das nicht ist. Variablenbezeichner sind *case sensitive*, will sagen große und kleine Buchstaben sind verschiedene Zeichen.

Im Unterschied zu vielen Sprachen wie z.B. Pascal oder C, doch ebenso wie in Octave oder Matlab verlangt Scilab vor der Verwendung einer Variablen keine Deklaration des Typs oder der Dimension.

Die Variablen sind im Arbeitsraum sichtbar und werden mit dem Befehl `who` aufgelistet.

Um nur die vom Anwender definierten Variablen zu erhalten, kann der Befehl `who_user` verwendet werden.

Zur Auflistung aller Variablen gibt es auch ein Grafikfenster wie in Abb. 2.4, das sich mit dem Befehl `browsevar` öffnet. Es öffnet sich auch, wenn man im Menü Applications/Anwendungen den Punkt Browser Variables/Variablen-Browser anklickt.

Man beachte, dass Funktionen und Bibliotheken von Funktionen in Scilab Variablen sind und deshalb auf den Befehl `who` ebenfalls erscheinen. Wie Programme (Skripte) und Funktionen definiert und benutzt werden, ist Gegenstand der Abschnitte 2.14 bzw. 2.15.

Eine Variable wird mit dem ersten Gebrauch zum Arbeitsraum hinzugefügt und ihre Entfernung daraus geschieht mit den Befehl

```
-->clear Variablenbezeichner
```

der auch zum Löschen sämtlicher vom Anwender definierter Funktionen dient, indem `clear` allein eingegeben wird. Man bedenke aber, dass der Befehl `clear` auch die Punktionen löscht (siehe Abschnitt 2.15) und etwaige Umgebungsvariablen von Scilab; der Befehl ist also mit Vorsicht zu gebrauchen.

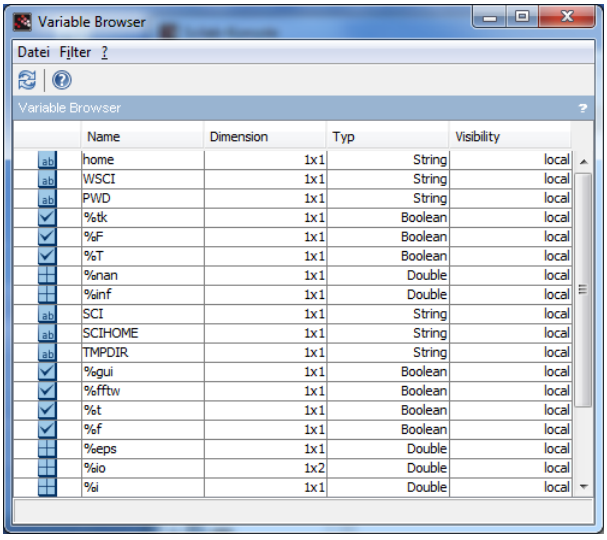

Abb. 2.4: Fenster zur Visualisierung des Arbeitsraumes.

2.3.1 Die vordefinierten Konstanten in Scilab

Im Arbeitsraum sind etliche vordefinierte Konstanten ständig verfügbar. Ihr Bezeichner beginnt mit dem Zeichen %. Schreibt man z.B.

```
-->%pi
 %pi  =

    3.1415927
```

erhält man einen Näherungswert für π, das Verhältnis von Kreisumfang zu Durchmesser. Die vordefinierten Konstanten sind schreibgeschützt, sie können weder geändert noch gespeichert werden. Die Tabelle 2.1 enthält eine Liste einiger in Scilab vordefinierter Konstanten. Es gibt noch weitere, die die Konfiguration des Programms betreffen und in diesem Zusammenhang ignoriert werden können. Das Prozentzeichen dient auch als Direktive bei der Formatierung in Befehlen zum Laden oder Speichern von Dateien, wie in Abschnitt 2.18 erwähnt.

%i	$i = \sqrt{-1}$	imaginäre Einheit
%pi	$\pi = 3.1415927\ldots$	Kreiszahl
%e	$e = 2.718281\ldots$	Eulersche Zahl
%eps	$2.20D - 16$	Maschinengenauigkeit
%inf		unendlich
%nan		not a number
%s	s	Polynomvariable
%z	z	Polynomvariable
%t %T	wahr (true)	Wahrheitswert
%f %F	unwahr (false)	Wahrheitswert

Tab. 2.1: Vordefinierte Konstanten

Hinweis: Als Ausgabe liefert Scilab die Wahrheitswerte T oder F ohne das Prozentzeichen.

2.3.2 Die Typen der Variablen

In Scilab kann einer Variablen ein Wert zugewiesen weiden, ohne sie vorher deklariert zu haben, was z.B. in C oder Fortran unabdingbar ist. Der Interpreter versucht, den richtigen Typ der Variablen zu herauszufinden, indem er den zugehörigen Befehl betrachtet. Diese Charakteristik vereinfacht das Schreiben von Code merklich, erhöht aber die Gefahr durch Flüchtigkeitsfehler.

Der wichtigste Datentyp ist eine Matrix (auch als Vektor oder Skalar) reeller oder komplexer Zahlen. Für Scilab ist dieser Datentyp konstant, ebenso wie fast alle vordefinierten Konstanten.

Ein für das Programmieren wichtiger Datentyp ist `boolean`, der in Scilab die beiden Werte %T (true, wahr) und %F (false, falsch) annehmen kann.

Andere Datentypen sind die Zeichenketten, in Scilab `string`, zu deren Definition die Zeichen in Anführungszeichen oder Hochkommas eingeschlossen werden müssen, zwischen denen Scilab nicht unterscheidet. In künftigen Versionen von Scilab wird das Mischen von Hochkomma und Anführungszeichen nicht mehr möglich sein.

```
-->str1 = 'hallo'
 strl =

 hallo

-->str2 = "hallo"
 str2 =

 hallo

-->str1 == str2
 ans  =

 T
```

Die beiden Variablen `strl` und `str2` sind also gleich. Um ein Hochkomma oder ein Anführungszeichen in eine Zeichenkette einzufügen, muss es zweimal eingegeben werden. Weitere Informationen zum Typ string finden sich im Abschnitt 2.11.

Den Datentyp `polynomial` definiert man mit dem Befehl `poly`. Eine einfache Möglichkeit, Polynome mit ihren bekannten Koeffizienten einzugeben, ist

```
-->s=poly(0,'s');

-->meinpoly = 3*s^2-s+2
 meinpoly  =

              2
   2 - s + 3s
```

Auf Variablen des Typs `polynomial` kann eine Reihe von Operationen ausgeführt werden, die in Abschnitt 2.9 präzisiert werden. Insbesondere ist es möglich, auf Verhältnissen von Polynomen zu arbeiten, d.h. auf gebrochen rationalen Funktionen vom Typ `rational`, der in Abschnitt 2.10 vorgestellt wird und auf dem Gebiet der dynamischen Systeme große Bedeutung hat. Er wird in Kap. 4 ausführlich behandelt.

Zur Ermittlung des Typs einer Variablen stehen die Befehle `type` und `typeof` zur Verfügung, die den Typ als Zahl oder als Text ausgeben. Die folgende Liste enthält die Namen der Typen, die man mit dem Befehl `typeof` bekommen kann.

```
constant
polynomial
function
handle
string
boolean
list
```

```
rational
state-space
sparse
boolean sparse
hypermat
st
ce
fptr
pointer
sizeimplicit
library
```

2.4 Formate für die Zahlendarstellung

Mit dem Befehl `format` kann man das Format der auszugebenden Zahlenwerte wählen. Er hat die Syntax

```
format ([type],[long])
format ()
```

worin beide Parameter optional sind. Der Parameter `type` ist ein Zeichen und nimmt nur zwei Werte an: 'v' für ein variables Format und 'e' für die Exponentialschreibweise. Der Parameter `long` bedeutet die Zahl der geforderten Ziffern plus Dezimalpunkt und evtl. führender Null. Das voreingestellte Format ist `format('v',10)`. Die Eingabe von format() ohne Parameter liefert das aktuelle Format. Der folgende Code zeigt Anwendungsbeispiele des Befehls.

```
-->format()
 ans  =

    1.   10.

-->x=0.123456789;

-->format('v',10); x
 x  =

    0.1234568

-->format('e'); x
 x  =

    1.235D-01

-->format('e',5)
              !--error 999
format: Falscher Wert für Eingangsargument #2: MUSS im Intervall [8, 25] liegen.
```

2.5 Wissenschaftliche Funktionen

Scilab erlaubt numerisches Rechnen mit Dutzenden von Funktionen. Vollständige Listen dieser Funktionen findet man nach Eingabe von help in den Abschnitten „Elementary Functions“ und „Special Functions“ des Hilfe-Browsers. Die Tabelle 2.2 enthält die wichtigsten elementaren Funktionen.

abs	absoluter Wert, Betrag
acos	arcus cosinus
acosh	arcus cosinus hyperbolicus
acot	arcus cotangens
acoth	arcus cotangens hyperbolicus
acsc	arcus cosecans
acsch	arcus cosecans hyperbolicus
amell	Jacobis Amplitudenfunktion
asec	arcus secans
asech	arcus secans hyperbolicus
asin	arcus sinus
asinh	arcus sinus hyperbolicus
atan	arcus tangens
atanh	arcus tangens hyperbolicus
cos	cosinus
cosh	cosinus hyperbolicus
cotd	cotangens (Gradmaß)
cotg	cotangens (Bogenmaß)
coth	cotangens hyperbolicus
exp	Exponentialfunktion
log	natürlicher Logarithmus
log10	Briggscher Logarithmus
log1p	natürlicher Logarithmus (Argument + 1)
log2	dualer Logarithmus
sec	secans
sech	secans hyperbolicus
sign	Vorzeichenfunktion
signm	Matrixvorzeichen
sin	sinus
sinc	sinus cardinalis: sinc(x)=(sin x)/x
sinh	sinus hyperbolicus
sqrt	Quadratwurzel
tan	tangens
tanh	tangens hyperbolicus

Tab. 2.2: Die wichtigsten elementaren Funktionen

2.6 Manipulation von Skalaren, Vektoren und Matrizen

Am einfachsten ist eine Matrix einzugeben, wenn man sie in die Befehlszeile schreibt:

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

Auf diese Weise hat man eine 3×3-Matrix reeller Zahlen in den Arbeitsraum eingefügt, Um das zu verifizieren, gibt man einen der Befehle `who` bzw. `who_user` ein oder man schreibt einfachen den Namen der Variablen in die Befehlszeile:

```
-->A
 A   =

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

Folgende Vereinbarungen gelten für die Eingabe einer Matrix in die Befehlszeile:

- die Elemte derselben Zeile werden durch Kommas oder durch Leerzeichen getrennt,
- jede Zeile, außer der letzten, wird durch ein Semikolon beendet,
- alle Elemente der Matrix müssen in eckige Klammern [...] eingeschlossen werden.

Und deshalb kann diese Matrix A auch mit folgender Syntax eingegeben werden:

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

Eine weitere Möglichkeit besteht darin, zunächst die erste Zeile einzugeben

```
-->A =[1 2 3
```

und die Enter-Taste zu drücken, um dann die anderen Zeilen einzugeben. Scilab beendet die Anweisung erst bei Eingabe der schließenden eckigen Klammer.

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

Das Semikolon am Ende der Anweisung zeigt, dass der Anwender keine Ausgabe des Ergebnisses auf dem Bildschirm wünscht. Eine sehr lange Anweisung kann durch drei Punkte am Zeilenende getrennt werden:

```
-->a = [1 2 3... <Enter>
```

Scilab kehrt zur Befehlszeile zurück, erwartet aber das Ende der Anweisung so, als befände es sich noch auf der vorigen Zeile und indem man z.B. mit

```
-->4 5 6] <Enter>
```

endet, erhält man einen Vektor a der Dimension 1×6:

```
 a  =

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

Es sei angemerkt, dass Vektoren und Skalare nur Sonderfälle von Matrizen sind. Bei Eingabe eines Skalars kann man die eckigen Klammern auch weglassen und die Variable wird einfach so definiert:

```
-->g=9.81
 g  =

    9.81
```

Zur Eingabe einer komplexen Zahl verwendet man die Konstante %i. Entsprechend wird der komplexe Skalar $1 - 3i$ eingegeben mit der Syntax

```
-->a = 1-3%i
 a  =

    1.  -3.i
```

Einige Standardmatrizen

In Scilab sind Punktionen zur Erzeugung einiger immer wieder gebrauchter Matrizen verfügbar, nämlich `eye(n,m)` für die Einheitsmatrix, `zeros(n,m)` für die Nullmatrix, `ones(n,m)` für die Einsermatrix oder `rand(a,m)` und `grand` für eine Matrix aus Zufallszahlen. Darin legt n die Anzahl der Zeilen und m die der Spalten fest. Zum Beispiel:

```
-->I3=eye(3,3)
 I3  =

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

Auch eine Matrix kann dabei als Argument dienen. Dann liefert Scilab als Ausgabe die gewünschte Matrix mit den Dimensionen des Arguments:

```
-->Null=zeros(I3)
 Null  =

    0.    0.    0.
    0.    0.    0.
    0.    0.    0.
```

Wegen der Erzeugung von Matrizen mit Zufallszahlen oder anderer Matrizen, die in Tabelle 2.3 aufgelistet sind, wird die Online-Hilfe von Scilab empfohlen.

eye	Einheitsmatrix
zeros	Nullmatrix
ones	Einsermatrix
rand, grand	Matrizen mit Zufallszahlen
diag	Diagonalmatrix
triu	obere Dreiecksmatrix
tril	untere Dreiecksmatrix
companion	Begleitmatrix
hank	Hankel-Matrix
toeplitz	Toeplitz-Matrix

Tab. 2.3: Funktionen zur Erzeugung von Matrizen

Matrizen mit konstantem Inkrement

Manchmal möchte man einen Vektor definieren, dessen Elemente mit einer bestimmten Zahl beginnen und in gleichen Abständen bis zu einer anderen Zahl anwachsen. Mit der Syntax

```
-->x=1:2:10
 x  =

    1.    3.    5.    7.    9.
```

bildet Scilab einen Vektor, dessen Elemente mit 1 beginnen und die jeweils um 2 wachsen, solange die Endzahl nicht übeschritten ist. Für Folgen mit dem Abstand 1 genügt die Schreibweise

```
-->x=1:4
 x  =
    1.    2.    3.    4.
```

Eine Alternative bietet der Befehl `linspace`. Für den zu erzeugenden Vektor wird das erste und das letzte Element sowie die Anzahl der Elemente angegeben.

```
-->x=linspace(0,1,6)
 x  =

    0.    0.2    0.4    0.6    0.8    1.
```

Zugriff auf die Elemente einer Matrix

Wenn im Arbeitsraum eine Matrix A mit den Dimensionen 3×3 definiert ist

```
-->A
 A  =

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

kann es notwendig werden, auf ein einzelnes Elemenent der Matrix oder auf eine ihrer Untermatrizen zuzugreifen. Die Anweisung

```
-->b=A(3,2)
 b  =

    8.
```

weist der Variablen b den Wert des Elementes in der 3. Zeile und 2. Spalte von A zu. Generell kann man Elemente der Matrix A mit der Syntax

```
-->b=A(3,2:3)
 b  =

    8.    9.
```

herausziehen. Die Variable b ist nun ein $l \times 2$-Vektor mit den Elementen aus Zeile 3 in den Spalten 2 bis 3. Verallgemeinernd kann man sagen, dass die Indizes der Zeilen und der Spalten, aus denen Elemente ausgelesen werden sollen, durch ein Komma zu trennen sind, wie diese Anweisung beispielhaft zeigt:

```
-->B=A([1 3],2:3)
 B  =

    2.    3.
    8.    9.
```

Um auf ganze Zeilen bzw. ganze Spalten zuzugreifen, genügt das Symbol : zur Bezeichnung:

```
-->B=A(:,2:3)
 B  =

    2.    3.
    5.    6.
    8.    9.
```

So werden alle Zeilen von A ausgewählt und eine 3×2-Matrix wird ausgegeben.

Da ein Vektor eine Matrix ist mit nur einer Zeile bzw. nur einer Spalte, kann auf ein einzelnes Element zugegriffen werden, indem der Index 1 für die Zeile bzw. für die Spalte angegeben wird. Dieser Index darf auch ganz weggelassen werden.

```
-->b=[1 2 3 4]; c=b(1,2), d=b(2)
 c  =

    2.
```

```
 d  =

    2.
```

Auch kann man mit dem Befehl `diag` nur die Diagonale einer Matrix entnehmen:

```
-->b=diag(A)
 b  =

    1.
    5.
    9.
```

Mit dem Diagonalenvektor kann man mit demselben Befehl eine Diagonalmatrix erzeugen:

```
-->C=diag(b)
 C  =

    1.    0.    0.
    0.    5.    0.
    0.    0.    9.
```

Man beachte, dass in Scilab der Index eines Vektors oder einer Matrix mit 1 beginnt. Das ist die gleiche Konvention wie in Matlab, während der Index z.B. in C mit 0 beginnt.

2.6.1 Mehrdimensionale Matrizen

Die Erweiterung einer Matrix auf den mehrdimensionalen Fall ist einigermaßen intuitiv. Man fügt die entsprechenden Indizes einfach hinzu, um Matrizen mit mehr als 2 Dimensionen zu generieren oder zu bearbeiten. Man definiert z.B. eine 2×2-Matrix:

```
-->A=[1 2; 3 4]
 A  =
    1.    2.
    3.    4.
```

kann dann eine dritte Dimension hinzufügen

```
-->A(:.:,2)=[5 6; 7 8]
 A  =
    1.    2.    3.    4.
    5.    6.    7.    8.
```

und auf die einzelnen Elemente zugreifen, indem die für den zweidimensionalen Fall gezeigte Syntax einfach erweitert wird.

2.7 Operationen auf Matrizen

Scilab bietet eine Bibliothek mit Funktionen für die Matrizenrechnung; alle Operationen auf Matrizen sind in Scilab mit ausreichend intuitiver Syntx möglich.

Die Summe von Matrizen

Summe zweier Matrizen gleicher Dimension:

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

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

-->A+A
 ans  =

    2.    4.    6.
    8.    10.   12.
```

Die Transponierte einer Matrix

Die Transponierte einer Matrix bekommt man mit dem Hochkomma ':

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

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

-->A'
 ans  =

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

Ist die Matrix komplex, liefert das Hochkomma eine konjugierte Transponierte.

Produkt von Matrizen, Vektoren und Skalaren

Das Produkt von Matrizen entsprechender Dimension oder einer Matrix mit einem Vektor erhält man ganz intuitiv mit dem Symbol $*$.

```
-->A=[1 2 3; 4 5 6]; B=[1 2; 3 4; 5 6]; C=A*B
 C  =

    22.    28.
    49.    64.
```

Bei der Multiplikation eines Skalars mit einer Matrix oder einem Vektor wird jedes Element mit diesem Skalar multipliziert.

```
-->A=ones(4,1); c=3; A*c
 ans  =

    3.
    3.
```

Potenzieren einer Matrix

Das Potenzieren einer Matrix gelingt mit dem Symbol $\wedge$.

```
-->A=[1 2; 3 4]; A^3
 ans  =

    37.    54.
    81.    118.
```

Man beachte, dass dieselbe Syntax bei einem Vektor die Elemente einzeln potenziert.

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

    1.    8.    27.
```

Exponentialfunktion einer Matrix

Für eine gegebene Matrix A ist eine Exponentialfunktion definiert durch

$$e^A = \sum_{i=0}^{\infty} \frac{A^i}{i!}$$

und man erhält mit der Syntax

```
-->A=[1 0; 2 1]; expm(A)
 ans  =

    2.7182818    0.
    5.4365637    2.7182818
```

was mit $\exp(A)$, wodurch das Exponential jedes einzelnen Elementes berechnet wird, nicht verwechselt werden darf.

```
-->A=[1 0; 2 1]; exp(A)
 ans  =
    2.7182818    1.
    7.3890561    2.7182818
```

Rang, Determinante, Bild und Kern einer Matrix

Den Rang einer Matrix erhält man mit dem Befehl `rank()` , die Determinante mit dem Befehl `det()`. Die Basen für die Unterräume Bild und Kern sind mit den Befehlen `orth` bzw. `kernel` erhältlich.

```
-->A =

  - 7.   - 5.   - 9.   - 3.
  - 5.   - 3.   - 6.   - 14.
    5.     6.   - 3.   - 2.
    6.   - 4.     8.     10.
    7.   - 5.   - 6.   - 4.
   10.   - 10.    7.     7.

-->orth(A)
 ans  =

    0.0192660    0.2772380    0.8497724
    0.5126586  - 0.4292629    0.4286271
    0.2664641    0.2720885  - 0.1775415
    0.5504762    0.1003252    0.0344690
    0.0207670  - 0.7193396  - 0.1060547
    0.6019564  - 0.3705749    0.2240734

-->kernel(A)
 ans  =
  - 0.5
  - 0.5
  - 0.5
    0.5
```

Division von Matrizen

Die Syntax $A \setminus B$ ist mathematisch nicht korrekt, denn die Matrizenmultiplikation ist nicht kommutativ. Tatsächlich wäre es notwendig, die Division als Multiplikation mit der inversen Matrix darzustellen, wie z.B. AB^{-1}. Deshalb bewirkt der Backslash eine etwas andere Operation: er wird für die Division von links benutzt. Die Syntax

```
x = A\B
```

löst die Gleichung $Ax = b$, wobei die Matrizen passende Dimensionen haben müssen und die Matrix A quadratisch oder rechteckig sein kann. Scilab erzeugt eine zur anderen Inverse bzw. Pseudoinverse mit der Dimension bzw. dem Rang von A.

Ganz analog findet die kompakte Syntax

```
x = A/b
```

die Lösung der algebraischen Gleichung $xb = A$.

Das charakteristische Polynom

Eine Anwendung des Befehls `poly` ist die Berechnung des charakteristischen Polynoms einer Matrix, das als `p(x) = |xI - A|` (2.1)

definiert ist, worin A eine qudratische Matrix, I die Einheitsmatrix gleicher Dimension, x die Polynomvariable ist und mit | der Determinantenoperator bezeichnet wird. Ein Beispiel liefert

```
-->A=[ 2 4; 0  1]; p = poly (A ,"x"), typeof(p)
 P  =

              2
    2 + 3x + x
 ans  =

 polynomial
```

Abschnitt 2.9 zeigt eine weitere Anwendung des Befehls poly.

2.7.1 Operationen auf den Elementen einer Matrix

In Scilab können Matrixoperationen elementweise ausgeführt werden. Das bedeutet, dass ein einziger Befehl etliche Zeilen Code einpart, mit denen dieselbe Operation auf allen Elementen der Matrix wiederholt werden müsste. Der Befehl

```
-->A=[1 %pi/2; 0 1]; sin(A)
 ans  =

    0.8414710   1.
    0.          0.8414710
```

hat keine mathematische Bedeutung, er berechnet die Sinuswerte der einzelnen Elemente der Matrix.

Die elementweise Opeation hat noch eine andere Bedeutung. Sind die beiden Matrizen A und B mit den Elementen a_{ij} und b_{ij} gegeben, dann liefert die Syntax

```
-->A=[1 %pi/2; 0 1]; B=[1 0; 0 -2]; C=A.*B
 C  =

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

eine Matrix, deren Elemente $c_{ij} = a_{ij} \cdot b_{ij}$ sind. Es genügt also, einen Punkt vor das Operationszeichen zu setzen, um eine elementweise Verarbeitung zu erhalten. Außer der Multiplikation, die man mit .∗ erhält, kann auch die Division mit ./ und das Potenzieren mit . elementweise ausgeführt werden.

Man beachte, dass die Syntax A^n als Potenz einer Matrix interpretiert wird, wenn A quadratisch ist und als elementweise Operation, wenn A rechteckig ist. Es ist guter Brauch, Mehrdeutigkeit zu vermeiden und die Syntax A^n auch bei reckteckigen Matrizen zu verwenden.

Ist die Variable c ein Skalar und ist A eine Matrix beliebiger Dimension, dann liefert die Syntax $c.^A$ eine Matrix mit den Elementen $c^{a_{ij}}$.

2.7.2 Erweitern und Löschen von Matrizen und Vektoren

Es gibt eine einfache Syntax um Matrizen oder Vektoren als Blöcke zu einer neuen Matrix zusammenzusetzen. Man trennt die Variablen mit einem Leerraum, einem Komma oder einem Semikolon, und Scilab versucht, die resultierende Matrix so zu erzeugen, als wären die Elemente Skalare. Zum Beispiel

```
-->a=1; B=[2 3; 4 5]; c=[-1 -1]; D=[a c; c' B]
 D  =

    1.  - 1.  - 1.
  - 1.    2.    3.
  - 1.    4.    5.
```

Es ist mit einer geeigneten Anweisung auch möglich, einzelne Elemente einer Matrix zu entfernen:

```
-->D(2,:)=[]
 D  =
    1.  - 1.  - 1
  - 1     4.    5.
```

So wurde in der Matrix D die zweite Zeile gelöscht und in der Konsequenz verändern sich auch die Dimensionen.

Um die Variable D vollständig aus dem Arbeitsraum zu entfernen, benutze man den Befehl `clear D` (siehe auch Abschnitt 2.3 zum Umgang mit Variablen).

2.8 Operationen auf komplexen Zahlen

Scilab behandelt die komplexen Zahlen auf dieselbe Weise wie die reellen Zahlen. Um eine komplexe Zahl einzugeben, bedient man sich der vordefinierten Konstanten $\%i$.

```
-->a=3+2*%i
 a  =

    3. + 2.i

-->isreal(a)
 ans  =

  F
```

wobei die Punktion `isreal` verwendet wurde, die bei reellen Zahlen T zurückgibt. Man beachte;

```
-->a=3+0*%i
 a  =

    3.

-->isreal(a)
 ans  =

  F
```

Die Zahl wird also als komplex erkannt, obwohl ihr Imaginärteil null ist.

Die Konjugierte einer komplexen Zahl erhält man mit dem Befehl `conj`. Die Transponiete einer Konjugierten erhält man ganz einfach mit dem Hochkomma $'$.

2.8.1 Isolieren von Real- und Imaginärteil

Die dafür anzuwendenden Befehle sind `real` für den Realteil und `imag` für den Imaginärteil.

2.8.2 Berechnung von Betrag und Argument

Den Betrag einer komplexen Zahl bekommt man mit dem Befehl `abs`, ihr Argument mit dem Befehl `phasemag` in Grad. Um gleichzeitig Betrag und Argument zu erhalten, letzteres dann im Bogenmaß, verwendet man den Befehl `polar`.

```
-->a=3+4*%i
 a  =

    3. + 4.i

-->abs(a)
 ans  =

    5.

-->phasemag(a)
 ans  =

    53.130102

-->[bet,arg]=polar(a)
 arg  =

    0.9272952 - 8.327D-17i
 bet  =

    5.
```

isreal()	ergibt T (true) für eine reelle Zahl
conj()	konjugiert komplexe Zahl
real()	Realteil
imag()	Imaginärteil
abs()	Betrag
phasemag()	Argument (in Grad)
[bet,arg]=polar()	Betrag und Argument (im Bogenmaß)

Tab. 2.4: Funktionen für komplexe Zahlen

Hinweis: `polar` liefert das Argument als komplexe Zahl, weil diese Funktion auf quadratischen Matrizen definiert ist (in Scilab ist ein Skalar eine $l \times 1$-Matrix). `clean(arg)` würde hier den vernachlässigbar kleinen Imaginärteil entfernen.

2.9 Operationen auf Polynomen

Für den Typ `polynomial` stehen verschiedene Operationen zur Verfügung, eine vollständige Liste findet man mit `help Polynomials` in der online-Hilfe direkt unterhalb des Stichwortes `rational`, wo die Befehle für die Operationen auf Variablen des Typs `rational` liegen.

2.9.1 Definition eines Polynoms durch seine Wurzeln

Der Befehl `poly` ermöglicht die Definition eines Polynoms aus seinen Wurzeln. So können wir eine Variable `meinpoly` mit den Wurzeln -1 und 2 erzeugen:

```
-->meinpoly=poly([-1, 2], 's')
 meinpoly  =

             2
  - 2 - s + s
```

Um die Polynomvariab le s zu definiern, erzeugt man ein Polynom vom Grad 1 mit Wurzel 0:

```
-->s=poly(0,'s')
 s  =

    s
```

oder man verwendet eine vordefinierte Variable, die es in Scilab für die Laplace-Transformierte und die Z-Transformierte gibt:

```
-->s=%s
 s  =

    s

-->z=%z
 z  =

    z
```

2.9.2 Definition eines Polynoms durch seine Koeffizienten

Der Befehl `poly` erlaubt nicht nur die Definition eines Polynoms aus seinen Wurzeln, was die Voreinstellung ist, sondern auch aus seinen Koeffizienten:

```
-->vekt=[-2 -1 1]
 vekt  =

  - 2.  - 1.    1

-->meinpoly=poly(vekt,'s','c')
 meinpoly  =

             2
  - 2 - s + s
```

2.9.3 Definition eines Polynoms durch einen Ausdruck

Auf recht einfache Weise kann man ein Polynom durch Eingabe des Ausdrucks mit der Polynomvariablen s definieren:

```
-->s=%s; meinpoly=-2-s+s^2
 meinpoly  =

             2
  - 2 - s + s
```

oder alternativ auch so:

```
-->meinpoly=(s+1)*(s-2)
 meinpoly  =

             2
  - 2 - s + s
```

2.9.4 Berechnung der Wurzeln eines Polynoms

Der Befehl `roots` liefert die Wurzeln eines Ploynoms. Er akzeptiert als Eingabe außer einer Variablen des Typs `polynomial` auch einen Vektor. Dann werden die Elemente des Vektors als Koeffizienten des Polynoms interpretiert, und zwar in absteigender Ordnung (Koeffizient der höchsten Potenz zuerst). Mit dem zuvor definierten Polynom `meinpoly` oder seinen Koeffizienten bekommt man:

```
-->roots(meinpoly)
 ans  =

    2.
  - 1.

-->roots([1  -1 -2])
 ans  =

    2.
  - 1.
```

2.9.5 Ermittlung der Koeffizienten aus der Polynomvariablen

Um die Koeffizienten eines Polynoms zu ermitteln, gibt es den Befehl `coeff`:

```
-->meinekoeff=coeff(meinpoly)
 meinekoeff  =

  - 2.  - 1.   1.
```

Man beachte, dass anders als in Matlab hier die Koeffizienten in aufsteigender Ordnung des Grades des Polynoms ausgegeben werden.

2.9.6 Berechnung des Wertes eines Polynoms für ein Argument

Zur Berechnung des Wertes eines Polynoms in einem bestimmten Punkt kann man den Befehl `horner` verwenden.

```
-->horner(meinpoly,3.2)
 ans  =

    5.04
```

2.9.7 Symbolische Substitution einer Polynomvariablen

Mit dem Befehl `horner` kann auch eine Polynomvariable durch eine andere polynomiale oder rationale Variable ersetzt werden:

```
-->s=poly(0,'s'), p1=s+1, p2=2*s^2+2
 s  =

    s
 p1  =

    1 + s
 p2  =
          2
    2 + 2s

-->p3=horner(p1,p2)
 p3  =

          2
    3 + 2s
```

Man beachte, dass man das Produkt zweier Polynome einfach mit dem Multiplikationsoperator $*$ erhält, wohingegen der Divisionsoperator / eine gebrochen ratonale Variable des Typs `rational` erzeugt:

```
-->out=p1*p2, typeof(out)
 out  =

               2    3
    2 + 2s + 2s + 2s
 ans   =

 polynomial

-->out=p1/p2, typeof(out)
 out  =
    1 + s
    -----
          2
    2 + 2s
 ans  =

 rational
```

Für die Polynomdivision muss man auf den Befehl `[q,rest]=pdiv(p1,p2)` zurückgreifen.

2.9.8 Übersicht über die Polynom-Befehle

bezout	- Bezout equation for polynomials or integers
clean	- cleans matrices (round to zero small entries)
cmndred	- common denominator form
coeff	- coefficients of matrix polynomial
coffg	- inverse of polynomial matrix
colcompr	- column compression of polynomial matrix
degree	- degree of polynomial matrix
denom	- denominator
derivat	- rational matrix derivative
determ	- determinant of polynomial matrix
detr	- polynomial determinant
diophant	- diophantine (Bezout) equation
factors	- numeric real factorization
gcd	- gcd calculation
hermit	- Hermite form
horner	- polynomial/rational evaluation
hrmt	- gcd of polynomials
htrianr	- triangularization of polynomial matrix
invr	- Inversion of (rational) matrix
lcm	- least common multiple
lcmdiag	- least common multiple diagonal factorization
ldiv	- polynomial matrix long division
numer	- numerator
pdiv	- polynomial division
pol2des	- polynomial matrix to descriptor form
pol2str	- polynomial to string conversion
polfact	- minimal factors
residu	- residue
roots	- roots of polynomials
routh_t	- Routh's table
rowcompr	- row compression of polynomial matrix
sfact	- discrete time spectral factorization
simp	- rational simplification
simp_mode	- toggle rational simplification
sylm	- Sylvester matrix
systmat	- system matrix

2.10 Operationen auf gebrochen rationalen Funktionen

Die gebrochen rationalen Funktionen sind für dynamische Systeme von großer Bedeutung. Nach einer Einführung in ihren Gebrauch in diesem Abschnitt wird wegen Details zu ihrer Anwendung bei Aufgaben der Regelungstechnik das Kapitel 4 empfohlen.

2.10.1 Definition einer gebrochen rationalen Funktion

Mit bekannten Polynomen für Zähler und Nennen kann eine gebrochen rationale Funktion leicht erzeugt werden, indem man sie ins Verhältnis setzt. Im folgenden Beispiel sind Zahler und Nenner als Ausdrücke definiert:

```
-->s=7,s; num=1+s; den=(s+2)*(s+2); meinerat=num/den
 meinrat  =

      1 + s
    ---------
              2
    4 + 4s + s

-->typeof(meinerat)
 ans  =

 rational
```

Wenn möglich, vereinfacht Scilab das Ergebnis, indem die Faktoren gekürzt werden, die beiden Polynomen gemeinsam sind:

```
-->s=%s; num=(1+s)*(s+2); den=(s+2)*(s+2); meinerat=num/den
 meinerat  =

    1 + s
    -----
    2 + s
```

Man vermeidet die Vereinfachung, indem man die vordefinierte Funktion simp_mode anpasst:

```
-->simp_mode(%F)

-->s=%s; num=(1+s)*(s+t); den=(s+2)*(s+2); meinerat=num/den
 meinerat  =

              2
    2 + 3s + s
    ----------
              2
    4 + 4s + s
```

2.10.2 Ermittlung von Zähler und Nenner

Mit den Befehlen `numer` und `denom` kann auf Zähler und Nenner des Bruchs zugegriffen werden:

```
-->meinerat
 meinerat  =

    1 + s
    -----
    2 + s

-->numer(meinerat)
 ans  =

    1 + s
-->denom(meinerat)
 ans  =

    2 + s
```

Man beachte, dass auf Zähler bzw. Nenner auch dadurch zugegriffen werden kann, dass die Listeneigenschaft des Typs rational ausgenutzt wird:

```
-->meinerat.num
 ans  =

    1 + s

-->meinerat.den
 ans  =

    2 + s
```

2.10.3 Berechnung der Pole und Nullsteilen einer gebrochen rationalen Funktion

Zur Berechnung der Pole und Nullstellen einer gebrochen rationalen Punktion muss zunächst auf Zähler und Nenner separat zugegriffen werden, um deren Wurzeln zu erhalten:

```
-->s=%s; num=1+s; den=3+s^2; meinerat=num/den
 meinerat =
```

```
    l + s
    -----
         2
    3 + s

-->roots(meinerat.den)
 ans  =

    1.7320508i
  - 1.7320508i
```

2.10.4 Zerlegung in Partialbrüche

Man kann eine gebrochen rationale Funktion mit dem Befehl `pfss` einfach in Partialbrüche zerlegen, so wie im folgenden die Funktion

$$F(s) = \frac{3(s+2)}{(s+3)*(s+1)}$$

zerlegt wird in

$$F(s) = \frac{1.5}{s+1} + \frac{1.5}{s+3}$$

```
-->s=%s; F=syslin('c',3*(s+2),(s+3)*(s+1)); out=pfss(F)
 out  =

       out(1)

     1.5
    -----
    l + s

       out(2)

     1.5
    -----
    3 + s

-->typeof(out)
 ans  =

 list
```

Man beachte, dass die Ausgabe des Befehls `pfss` eine Liste ist. Man kann diesen Befehl auch auf ein lineares System im Zustandsraum anwenden (siehe Kap. 4 wegen Einzelheiten zu dynamischen Systmen).

2.10.5 Berechnung des Wertes einer rationalen Funktion für ein Argument

Der Befehl `horner` akzeptiert als Eingabeparameter sowohl Polynome als auch rationale Funktionen.

```
-->horner(meinerat,4.33)
 ans  =

    0.2450699
```

2.10.6 Substitution einer Variablen in einer rationalen Funktion

Der Befehl `horner` ermöglicht auch die Ersetzung einer symbolischen Variablen einer rationalen Punktion durch einen anderen rationalen Ausdruck. Angenommen, wir haben die rationale Punktion

$$F(s) = \frac{1+s}{5+s}$$

und wir wollen mit

$$s = \frac{z-1}{z+1}$$

eine Substitution durchführen, die eine solche Lösung liefert:

$$F(z) = F(s)|_{s=\frac{z-1}{z+1}} = \frac{z}{2+3z}$$

Dafür haben wir in Scilab

```
-->s=%s; rat1=(s+1)/(s+5)
 rat1  =

    1 + s
    -----
    5 + s

-->z=%z; rat2=(z-1)/(z+1)
 rat2  =

  - 1 + z
    -----
    1 + z

-->horner(rat1,rat2)
 ans  =

     0.3333333z
    ------------
    0.6666667 + z
```

2.11 Zeichenketten

Zeichenketten haben den Typ `string`. Sie werden in Anführungszeichen oder Hochkommas eingeschlossen. Scilab unterscheidet sie nicht, man darf sie sogar mischen, was aber in späteren Versionen nicht mehr möglich sein wird.

```
--> s = 'ciao'
 s  =

 ciao
```

Man kann auch eine Matrix mit Zeichenketten definieren:

```
-->matstr = ['c' 'ci' 'cia' 'ciao']
 matstr  =

!c  ci  cia  ciao  !

-->typeof(matstr)
 ans  =

 string

-->size(matstr)
 ans  =

    1.    4.
```

Es gibt unzählige Befehle zur Behandlung von Zeichenketten, das sind hauptsächlich

ascii	- string ascii conversions
blanks	- Create string of blank characters
code2str	- returns character string associated with Scilab integer codes
convstr	- case conversion
emptystr	- zero length string
grep	- find matches of a string in a vector of strings
isalphanum	- check that characters of a string are alphanumerics
isascii	- tests if character is a 7-bit US-ASCII character
isdigit	- check that characters of a string are digits between 0 and 9
isletter	- check that characters of a string are alphabetic letters
isnum	- tests if a string represents a number
justify	- Justify character array
length	- length of object
part	- extraction of strings
regexp	- find a substring that matches the regulär expression string
sci2exp	- converts an expression to a string
str2code	- return scilab integer codes associated with a character string
strcat	- concatenate character strings
strchr	- Find the first occurrence of a character in a string
strcnip	- compare character strings
strcmpi	- compare characterstrings (case independent)
strcspn	- Get span until character in string
strindex	- search position of a character string in an other string
string	- conversion to string
strings	- Scilab Object, character strings
stripblanks	- Strips leading and trailing blanks (and tabs) of strings
strncmp	- Copy characters from strings
strrchr	- Find the last occurrence of a character in a string
strrev	- returns string reversed
strsplit	- split a string into a vector of strings
strspn	- Get span of character set in string
strstr	- Locate substring
strsubst	- substitute a character string by another in a character string
strtod	- Convert string to double
strtok	- Split string into tokens
tokenpos	- returns the tokens positions in a character string
tokens	- returns the tokens of a character string
tree2code	- generates ascii definition of a Scilab function

2.12 Strukturen und Listen

Wie in vielen Programmiersprachen gibt es auch in Scilab die Struktur, eine Zusammenfassung von Objekten; der Typ jedes Objektes muss in Scilab definiert sein. Die Struktur bietet sich an, wenn man ein komplexes Objekt definieren möchte, das aus mehreren Variablen unerschiedlichen Typs besteht.

Will man beispielsweise eine Simulation schreiben für die Planung der Bewegung von Fahrzeugrobotern, ist jedes Fahrzeug charakterisiert durch einen Namen, seine Position, Geschwindigkeit und weitere Informationen, z.B. ob es angeschaltet ist oder ausgeschaltet. Alle diese Informationen können mit dem Befehl `struct` in einer Struktur zusammengefasst werden:

```
-->robot=struct  ('name',[],'pos',[],'vel',[],'on',[])
 robot  =

   name: [0x0 constant]
   pos: [0x0 constant]
   vel: [0x0 constant]
   on: [0x0 constant]
```

deren Typ mit dem Befehl `typeof` ermittelt wird:

```
-->typeof(robot)
 ans  =

 st
```

Nun ist es möglich, auf die Elemente der Struktur zuzugreifen. Man benutzt ihren Bezeichner und kann dann, um z.B. den Namen des ersten Fahrzeugs festzulegen, schreiben:

```
-->robot.name='neu';
```

Ganz analog kann man Position und Geschwindigkeit durch Vektoren definieren und den Zustand an/aus mit `T` oder `F` angeben.

Es ist hilfreich, in der Struktur Vektoren zu definieren, im vorliegenden Fall einen Vektor Fahrzeug.

Das zweite Fahrzeug definiert man einfach folgendermaßen:

```
-->robot(2).name='morpheus';
```

Die Variabile `robot` hat jetzt zwei Elemente, jedes davon ist eine Struktur. Die Namen der verschiedenen Fahrzeuge können mit dem folgenden Befehl abgefragt werden:

```
-->robot.name
 ans  =

       ans(1)

 neu

       ans(2)

 morpheus
```

Die Definition der Struktur kann zusammen mit der Definition des ersten Elements erfolgen:

```
-->robot=struct  ('name','neo','pos',[0 0] ,'vel',[0 0],'on',%T)
 robot  =

   name: "neo"
   pos: [0,0]
   vel: [0,0]
   on: %t
```

Es ist immer möglich, der Definition noch ein weiteres Feld hinzuzufügen. Dazu bezieht man sich auf eines der schon definierten Fahrzeuge. Will man das Feld `modell` ergänzen, das die Angabe des Fahrzeugmodells enthält, schreibt man die Anweisung

```
-->robot(1).modell='beta 0.9';
```

die dieses Feld auch für das zweite Fahrzeug hinzufügt, obwohl der Wert dafür noch nicht festgelegt ist.

```
-->robot(2).modell
 ans  =

     []
```

Auf den Feldern einer Struktur sind dann alle Operationen möglich, die für den entsprechenden Typ definiert sind.

Ähnlich wie Strukturen sind *Listen* aufgebaut, die mit den Befehlen `list, tlist` und `mlist` definiert werden.

2.13 Symbolisches Rechnen

Scilab ist nicht wie Maple, Mathematica oder selbst Matlab mit einer Engine für symbolische Rechnungen ausgestattet. Es macht aber einfache symbolische Operationen möglich, indem Befehle ausgenutzt werden, die für Zeichenketten definiert sind. Insbesondere sind die in Tabelle 2.5 zusammengestellten Befehle dafür anwendbar.

addf	Addition
subf	Subtraktion
mulf	Multiplikation
rdivf	Division
cmb_lin	Linearkombination
trianfml	Triangulation
solve	Lösung von Ax=b
eval	Auswertung eines Ausdrucks

Tab. 2.5: Funktionen für symbolisches Rechnen

Wir definieren eine symbolische Variable mit dem Befehl `addf`. Solche Variablen haben den Typ string.

```
-->out=addf("a","x")
 out  =

 a+x

->typeof(out)
 ans  =

 string
```

Es sind nur simple Vereinfachungen möglich wie in folgendem Fall:

```
-->addf("x+2","y-2")
 ans  =

 x+y
```

Entsprechendes gilt für Multiplikation und Division:

```
-->mulf("x+l","y-l")
 ans  =

 (x+l)*(y-1)

-->rdivf("x^2+x-1","x+1")
 ans  =

 (x^2+x-1)/(x+l)
```

So ist es dann auch möglich, durch Angabe von vier Parametern eine Linearkombination einzuführen (man beachte das Minuszeichen):

```
-->z=cmb_lin("c(l)","x"',"c(2)","y+2")
 z  =

 c(l)*x-c(2)*(y+2)
```

Der Ergebnisausdruck, der in der Variablen z gespeichert ist, kann nach Belegung der benutzten Variablen mit Zahlenwerten ausgewertet werden:

```
-->c=[1 2];

-->x=3;

-->y=5;

-->z=eval(z)
 z  =
  - 11.
```

Jetzt ist das Ergebnis eine Zahl:

```
-->typeof(z)
 ans  =

 constant
```

Eine weitere symbolische Operation besteht in der Triangulation, die über elementaren Operationen auf den Zeilen der Matrix ausgeführt werden:

```
-->A=["a11","a12";"a21","a22"]
 A  =

!all a12  !
!          !
!a21 a22  !

-->trianfml(A)
 ans  =
 !a21  a22             !
 !                      !
 !0    a21*a12-a11*a22  !
```

Schließlich ermöglicht der Befehl `solve` das Lösen von linearen Gleichungssystemen:

```
Ax = b
```

wobei die Koeffizientenmatrix A obere Dreiecksgestalt haben muss:

```
-->A=["a","b";"0","d"]
 A  =

 !a b  !
 !     !
 !0 d  !

-->b=['2';'4']
```

```
 b  =
!2  !
!   !
!4  !

-->solve(A,b)
 ans  =
!a\(-b*(d\4)+2)  !
!                !
!d\4             !
```

Man schenke der Tatsache Beachtung, dass die Dreiecksgestalt von A nicht geprüft wird. Einträge unterhalb der Hauptdiagonalen werden schlicht ignoriert.

Hier kommt nun eine einfache Übung, um diese Funktionen auch für die Lösung eines Gleichungssystem mit nicht dreieckiger Matrix A zu verwenden und die Inverse einer Matrix zu berechnen,

Für die Aufgabe $Ax = b$ mit nicht dreieckiger Matrix A bildet man die erweiterte Koeffizientenmatrix

$$\bar{A} = [Ab]$$

auf der man mit dem Befehl `trianfml` eine Triangulation durchführt. Dadurch erhält man eine Matrix $\bar{A}_1$, die in zwei Matrizen mit den gleichen Dimensionen wie die Matrizen A und b der ursprünglichen Aufgabe gteilt werden kann:

$$\bar{A}_1 = [Aib_1]$$

.

Das gewünschte Resultat ist die Ausgabe des Befehls `solve` mit A_1 und b_1 als Eingabe.

Auch für Matrizengleichungen der Form $AX = B$ kann der Befehl `solve` verwendet werden. Man erhält die symbolische Inverse einfach dadurch, dass man für die Matrix B eine Einheitsmatrix passender Dimension setzt und für den Fall, dass die Koeffizientenmatrix nicht dreieckig ist, das zuvor beschriebene Verfahren anwendet.

2.14 Skripte

Ein Skript ist ein Programm, also eine Liste von Anweisungen, die in einer Datei gespeichert sind und durch einen einzigen Befehl ausgeführt werden.

Um ein Skript ausführbar zu machen, muss es in einer Datei mit der Endung `sce` gespeichert werden. Das Speichern ist nach jeder Änderung zu wiederholen. Zu beachten ist, dass

die `*.sce`-Dateien unformatierte Textdateien sein müssen. Sie können im Prinzip mit einem der unzähligen Texteditoren editiert werden, doch kommt in Scilab ein eigener Texteditor `SciNotes` zum Einsatz, der eigens für die Scilab-Syntax geschrieben wurde. Um ihn zu starten, geht man mit der Maus im Hauptfenster von Scilab in das Menü „Anwendungen“ oder man gibt `scinotes` in der Befehlszeile ein. Die verschiedenen Icons geben bei Berührung mit dem Mauszeiger ihre Bedeutung preis, in der deutschsprachigen Version noch teilweise in Englisch.

Es ist zweckmäßig, im Skript Kommentare einzufügen (wie auch in Funktionen - siehe den nächsten Abschnitt), was nach einem doppelten Schrägstrich // wie in FreePaseal oder C bis zum Ende der Zeile geschehen kann.

Zur Ausführung eines Skriptes wird entweder ein Icon in der Kopfzeile von SciNotes angeklickt oder „Ausführen“ im Menü „Datei“. Oder man gibt in der Befehlszeile von Scilab den Befehl

`-->exec('dateiname.sce')` ein. Oder man speichert und startet das Skript mit der Funktionstaste F5.

2.15 Funktionen

Funktionen sind Unterprogramme und werden in Scilab in einer Datei mit der Endung `sci` gespeichert (Skripte haben die Endung `sce`). Auch die `*.sci`-Dateien müssen unformatierte Text-Dateien sein. Eine `*.sci`-Datei kann mehrere Funktionen enthalten, die z.B. eine Toolbox zu einem bestimmten Thema bilden.

Eine Funktion besitzt die folgende Struktur:

```
function[outl, ...,  outm] = funktionsbezeichner(inputl, ..., inputn)
    Befehle
endfunction
```

Darin hängt die Zahl der Eingabeparameter `input1` bis `inputn` wie der Ausgabepameter `out1` bis `outm` von der jeweiligen Funktion ab und kann u.U. auch null sein. Eine und dieselbe Funktion kann mit unterschiedlich vielen Eingabeparametern aufgerufen werden und kann daher beispielsweise mit optionalen Parametern umgehen. In einem solchen Fall erfährt die Funktion die Anzahl der Eingabe- bzw. Ausgabeparameter durch Aufruf der Funktion `argn()`.

Ist eine Funktion einmal geschrieben und evtl. mit anderen Funktionen in einer Datei gespeichert, muss diese Datei mit dem Befehl exec('dateiname.sci') erst aufgerufen werden, damit die Funktionen in den Hauptspeicher gelangen und verwendet werden können. Dazu ist es zweckmäßig die Befehle zu kennen, um das Verzeichnis zu bestimmen, in dem sich die Datei befindet: mit dem Befehl `pwd` erfährt man das aktuelle Arbeitsverzeichnis, das mit dem Befehl chdir geändert werden kann oder grafisch mit der Maus im Menü „Datei“ und „Verzeichnis ändern“.

Und schon können wir eine erste Bibliothek mit zwei einfachen Funktionen erzeugen. Wir schreiben die Datei `erstebiblio.sci`: // *berechnet den Abstand zweier Punkte in der Ebene*

```
function out = abstand(x,y)

    out = sqrt((x(1)-y(1))^2+(x(2)-y(2))^2);

endfunction

// und liefert das Ergebnis in dezibel

function out = abstanddb(x,y)

    out = abstand(x,y);
    out = 20*log10(out);

endfunction
```

Die Leerzeilen dienen nur der besseren Lesbarkeit und können weggelassen werden. Der Aufruf mit dem Parameter -1 (siehe `help exec`) unterdrückt die Ausgabe der Skript- bzw. Funktionstexte. Ein Semikolon am Zeilenende ist nur sinnvoll bei zeilenweiser Eingabe auf der Console, schadet hier aber nicht. Zwischenwerte während der Ausführung von Skripten und Funktionen können z.B. mit dem Befehl `disp` ausgegeben werden (siehe Abschnitt 2.19).

Um diese Bibliothek mit zwei Funktionen nutzen zu können, muss man den Inhalt der Datei mit der Anweisung `exec(erstebibliothek.sci')`; in den Hauptspeicher laden. Danach werden mit dem Befehl `who` auch diese beiden Punktionen angezeigt und können vom Anwender benutzt werden.

Man beachte, dass eine oder auch mehrere Funktionsbibliotheken von einer Funktion oder einem Skript geladen werden können. Auch kann eine Funktion oder ein Skript andere Funktionen ansprechen, die sich im Hauptspeicher befinden. Schließlich können `*.sci`-Dateien auch aus einem Skript heraus geladen werfden, sodass umfangreichere Programme nur einen einzigen Aufruf benötigen.

2.15.1 Parameterübergabe an eine Funktion

Besondere Sorgfalt muss man der Parameterübergabe widmen und allgemein der Sichtbarkeit der Variablen.

Anders als in Matlab oder C ist in Scilab aus einer Funktion heraus der gesamte Arbeitsraum sichtbar und so können alle darin vorhandenen Variablen verwendet werden. Daher ist es guter Programmierstil, nur solche Variablen zu verwenden, die der Funktion bei ihrem Aufruf übergeben wurden. Man achte sehr darauf, dass in Scilab ein Fehler sehr leicht unbemerkt bleiben kann, sobald innerhalb einer Funktion irrtümlich eine außerhalb initialisierte Variable benutzt wird. Angenommen, eine Variable wurde in der aufrufenden Funktion definiert, dann wird ihr Gebrauch in der aufgerufenen Funktion von Scilab nicht beanstandet, was als „Seiteneffekt“ den klassischen semantischen Fehler hervorruft. Nach unserer bescheidenen Meinung sollte das zu einem Syntaxfehler gemacht und somit unterbunden werden, indem für Funktionen die vollständige Sichtbarkeit des Arbeitsraums aufgehoben wird.

Die Parameterübergabe erfolgt in Scilab mit `call by value`. Das garantiert, dass die übergebenen Parameter nur innerhalb, nicht aber außerhalb der Funktion verändert werden können. Diese Art der Behandlung der Parameter beeinflusst die Ausführungsgeschwindigkeit des Codes und sollte beim Schreiben von Funktionen genau beachtet werden. Das folgende Beispiel illustriert dieses Konzept, alle drei Funktionen sind in einer Datei namens `test_uebergabe.sci` gespeichert:

```
// Übergabe mit call-by-reference

function out = testl(x,i)

    out = 2*x(i);

endfunction

// Übergabe mit call-by-value

function out = test2 (x,i)

    x(i) = 2*x(i);
    out = x(i);

endfunction

// Aufruf der beiden Funktionen

function test

    n = 100000;
    x = rand(n,1);
    timer(); for i =1: n; out = testl(x,i); end; tl = timer()/n
    timer(); for i =1: n; out = test2(x,i); end; t2 = timer()/n
    printf('tl = %f/n t2 = %f/n', tl,t2)

endfunction
```

Ist die Datei geladen und die Funktion `test` gestartet, bekommt man nach einiger Zeit, die vom benutzten Computer abhängt, z.B.

```
-->test
 tl = 0.000010 t2 = 0.000678
```

Eine weitere wichtige Charakteristik, die den Umgang mit Variablen einschränkt, ist das Fehlen statischer Variablen. Sie werden innerhalb einer Funktion definiert und verlieren zwischen zwei Aufrufen dieser Funktion nicht ihren Wert. Diese Option gibt es in C mit dem Befehl `static` sowie in Matlab, Freemat oder Octave mit `persistent`.

Das gleiche Ergebnis wird erzielt, wenn man eine globale Variable definiert entweder in der aufrufenden Funktion oder in der aufgerufenen. Es ist aber klar, dass diese Lösung die Portabilität und die Robustheit des Codes herabsetzt.

2.16 Logische und Vergleichsoperationen

In Tabelle 2.6 sind die in Scilab vorhandenen logischen Operatoren aufgelistet und in Tabelle 2.7 die Vergleichsoperatoren.

&	und
|	oder
∼	nicht

Tab. 2.6: Logische Operatoren

==	gleich wie
<	kleiner als
>	größer als
≤	kleiner oder gleich
≥	größer oder gleich
∼= oder <>	verschieden von

Tab. 2.7: Vergleichsoperatoren

2.17 Befehle zur Ablaufsteuerung

Wie die anderen Programmiersprachen stellt auch Scilab Befehle für Schleifen bereit, die den Programmfluss steuern. Es muss hervorgehoben werden, dass man gut beraten ist, den Gebrauch von Schleifen auf ein Minimum zu reduzieren, weil Scilab, genau wie Matlab, jeden Schleifendurchlauf neu interpretiert. Wo immer möglich sollte man für Matrixoperationen die elementweisen Operatoren einsetzen.

2.17.1 Die for-Schleife

Eine `for`-Schleife bietet mit einer Syntax wie

```
for Variable = Ausdruck

    Befehle

end
```

in allen Programmiersprachen ein ähnliches Aussehen. Sie wird generell dann verwendet, wenn die Anzahl der Schleifendurchläufe vorher bekannt ist. In Scilab lautet ihre Syntax wie folgt:

```
-->n=10;

~->for i=l:n, printf(' %d ', i), end
 1  2  3  4  5  6  7  8  9  10
```

oder abwärts zählend

```
-->for i=n:-2:l, printf(' %d ',i), end
 10  8  6  4  2
```

2.17.2 Die `while`-Schleife

Eine `while`-Schleife hat die Gestalt

```
while Ausdruck

    Befehle

end
```

und auch hier ähnelt ihre Syntax derjenigen anderer Programmiersprachen, wie dieses Beispiel zeigt:

```
-->x=2; while x<8, x=2*x, end
 x  =

    4.
 x  =

    8.
```

2.17.3 Die Anweisung `if then else`

Dies ist nun die Syntax einer `if-then-else`-Anweisung:

```
if Bedingungl then

    Befehle

elseif Bedingung2 then

    Befehle

elseif Bedingung3 then
```

```
    Befehle
else
    Befehle
end
```

2.17.4 Die Anweisung `case select`

Folgende Syntax gilt für die bedingte Anweisung `case select`:

```
select Variable
case Ausdruck1
    Befehle
case Ausdruck2
    Befehle
case Ausdruck3
    Befehle
else
    Befehle
end
```

2.17.5 Die Anweisung `break`

Sie erlaubt das Verlassen einer `for`- oder `while`-Schleife und lässt das Programm zu der auf die Schleife folgenden Anweisung springen.

Wie in anderen Sprachen auch ist die Anweisung `break` mit Vorsicht zu verwenden, denn sie ist eine potentielle Fehlerquelle.

2.17.6 Die Anweisungen `error` und `warning`

Zum Beenden eines Programms oder einer Funktion bei einem Fehler gibt es die Anweisung `error`.

Die Anweisung `warning` kann dem Anwender eine anomale Situation anzeigen, unterbricht die Ausführung jedoch nicht.

2.17.7 Die Anweisungen `pause, return, abort`

Eine etwas grobschlächtige Möglichkeit seinen Code zu debuggen (für die Sprachpuristen: zu entlausen) liefern die Anweisungen `pause-return-abort`. Wird `pause` in ein Programm eingefügt, erscheint der Cursor und gibt damit die Möglichkeit, Befehle in die Befehlszeile zu schreiben, um z.B. den Wert einer Variablen auszugeben. Mit `return` wird die Ausführung fortgesetzt, während `abort` sie abbricht.

2.18 Import und Export von Daten

Es existieren mehrere Möglichkeiten, eigene Variablen in einer Datei zu speichern. Eeine davon bietet der Befehl `save`. Er erzeugt eine Binärtei mit allen oder einigen der im Arbeitsraum vorhandenen Dateien. Der folgende Code erzeugt drei Variablen unterschiedlichen Typs, speichert sie in einer Datei, löscht sie im Arbeitsraum und lädt sie dann wieder mit dem Befehl `load`.

```
-->A=[1 2 3; 4 5 6]; b='hallo'; c=3.12;

-->save('daten')

->clear

-->load('daten')
```

Man bedenke, dass eine Binärdatei nur von einer Scilab-Anwendung zur anderen übertragen werden kann. Die Portabilität ist also davon abhängig, ob Scilab auf der Maschine installiert ist.

Scilab stellt auch Befehle bereit, welche die Schreib- und Lesefunktionen für Dateien der Programmiersprache C emulieren. Zum Öffnen bzw. Schließen einer Datei dienen die Befehle `mopen` bzw. `mclose` und zum Schreiben bzw. Lesen die Befehle `mfprintf` bzw. `mfscanf`. Um in das Scilab-Fenster zu schreiben, hat man analog die entsprechenden C-Befehle `mprintf` und `mscanf`. Für jemand, der mit C vertraut ist, bedeuten diese Befehle keine zusätzlichen Schwierugkeiten.

Um eine Matrix in einer Textdatei zu speichern, stellt Scilab die Funktion `fprintfMat` bereit. Im Format von C kann die Matrix zusammen mit einem eventuellen Kommentar gespeichert und mit dem Befehl `fscanfMat` zurückgelesen werden.

Für mit der Sprache Fortran Vertraute gibt es die beiden Befehle `write` und `read`. Sie erlauben das Speichern von Variablen in Textdateien im Fortran-Format. Mit dem Befehl:

```
-->v=1:10; write('daten3',v,'(10(i2,3x))')
```

wird eine Textdatei erzeugt, deren einzige Zeile mit einem beliebigen Texteditor gelesen werden kann:

```
 1       2       3       4       5       6       7       8       9       10
```

Weitere Details sind einem Fortran-Handbuch zu entnehmen.

Schließlich gibt es noch die Möglichkeit, sowohl Binär- als auch Textdateien im Format der Matlab-Version 6 zu schreiben und zu lesen. Wegen der Syntax der Befehle `savematfile` und `loadmatfile` sei auf die Online-Hilfe von Scilab verwiesen.

2.19 Ausgabe von Daten im Befehlsfenster von Scilab

Um Daten in das Befehlsfenster von Scilab zu schreiben, kann man den Befehl `printf` mit derselben Syntax wie in C verwenden oder den Befehl `disp`, der ohne weitere Formatierung jeden beliebigen Scilab-Typ ausgeben kann, zum Beispiel:

```
-->s=poly(0,'s'); x=s^2+1; y=s-1; F=y/x; disp([x y F])
          2
    1 + s   - 1 + s   - 1 + s
    -----     -----     -----
                              2
      1         1       1 + s
```

2.20 Anpassen des Befehlsfensters von Scilab

In Scilab kann man das Aussehen des Befehlsfenster seinen persönlichen Bedürfnissen anpassen, und zwar sowohl unter Windows als auch unter Linux: man wählt im Menü Editieren -> Preferences die gewünschte Option und modifiziert sie oder man klickt mit der Maus auf eines der Icons.

2.21 Aufruf eines Startskripts

Vielleicht möchte man ein Skript schreiben, das beim Start von Scilab eine Reihe von Operationen ausführt, die vom Anwender definiert sind. Dazu sucht man nach der Datei `scilab.start` und fügt *an ihrem Ende* die gewünschten Befehle an. In der Linux-Distribution Ubuntu 9.04 befindet sich die Datei in
`/usr/share/scilab/etc/scilab.start`.

Als Alternative kann man der Datei `scilab.start` als letzte Zeile eine Anweisung wie `exec ('$PATH/startscript.sce');` hinzufügen und dann die Datei `startscript.sce` bearbeiten.

Es gibt noch eine gute Möglichkeit, die Startbefehle in Mehrbenutzer-Systemen zu personalieren. Beim Start sucht Scilab nach der Datei `SCIHOME/.scilab`, und falls sie existiert,

führt es ihren Inhalt aus. Somit wird es unnötig, eine Datei im Programmverzeichnis zu verändern. Wenn beide Dateien vorhanden sind, führt Scilab die Datei `SCIHOME/.scilab` vor `scilab.start` aus. Das Verzeichnis `SCIHOME` befindet sich in

Linux/Unix:	/home/<User>/.Scilab/<Scilab-Version>
Windows XP:	C:\Dokumente und Einstellungen\<User> \Anwendungsdaten\Scilab\<Scilab-Version>
Vista:	C:\<User>\AppData\Roaming\Scilab\<Scilab-Version>
Windows 7:	C:\Programme\scilab-5.4.0\etc

worin für `<User>` naürlich der Name des Anwenders zu setzen ist.

2.22 Aufruf von Systembefehlen

Es kann erforderlich werden, Systembefehle einzugeben. Dafür gibt es den Befehl `host()`, dessen Argument eine Zeichenkette mit dem auszuführenden Befehl ist. Man beachte aber, dass `host()` keine Ausgabe liefert, sodass man keine Befehle verwenden kann, die eine Ausgabe von Text erzeugen, wie z.B. `dir` unter Windows oder `ls` unter Linux.

Unter Linux gibt es noch Varianten des Befehls `host()`, die eine größere Flexibilität erlauben, die Beschränkungen des Befehls `host()` überwinden und auch Ausgaben im Scilab-Fenster zulassen. Näheres liefert der Aufruf `help unix`.

2.23 Gibt es keine Null? Anmerkungen zur numerischen Genauigkeit

Programme zur numerischen Rechnung unterliegen bei der Ausführung der Rechnungen einer ganzen Reihe von Fehlermöglichkeiten. Ihre Duskussion ist hier im einzelnen noch nicht einmal oberflächlich möglich. Wir möchten aber doch die Aufmerksamkeit des Lesers auf die Fehler bei Näherungsrechnungen lenken, die selbst bei einfach scheinenden Rechnungen auftreten können.

Wir erzeugen eine quadratische Matrix $T \in \mathrm{M}_{33}(\mathbb{R})$ mit gleichverteilten Zufallszahlen zwischen 0 und 1:

```
-->T=grand(3,3,'def')
 T  =

    0.8147237    0.8350086    0.9133759
    0.135477     0.1269868    0.2210340
    0.9057919    0.9688678    0.6323592
```

und aus der Algebra ist bekannt, dass

$$T^{-1}T = I$$

worin I die Einheitsmatrix ist. Führen wir diese Operation nun aus,

```
-->inv(T)*T
 ans  =

    1.           0.           7.105D-15
    7.105D-15    1.           7.105D-15
    1.776D-15    8.882D-16    1.
```

sehen wir, dass diese Matrix nicht genau die Einheitsmatrix ist. Wir erinnern uns, dass die in Scilab vordefinierte Konstante `%eps` die „Maschinengenauigkeit“ enthält, die vor allem auch von Scilab selbst abhängt, und die ist

```
->%eps
 %eps  =

    2.220D-16
```

Der Befehl `clean` kann nun die sehr kleinen Einträge einer Matrix auf 0 setzen. Das führt mit unserer Matrix T von vorhin auf

```
-->clean(inv(T)*T)
 ans  =

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

Ein weiteres Beispiel enthält Abschnitt 4.1.3, wo eine einfache Umwandlung der Darstellung eines linearen Systems zu einem Rundungsfehler führt.

2.24 Tipps und Tricks

- Die Pfeiltaste „nach oben“ holt frühere Befehle in die Befehlszeile. Man kann die Befehle auch anhand ihrer Anfangsbuchstaben filtern.
- Es ist auch Autovervollständigung eines Befehls möglich. Man gibt den Anfang ein und drückt dann auf die TAB-Taste. Das funktioniert auch mit den Bezeichnern von Verzeichnissen und Dateien.
- Mit der Syntax `x=[]` kann man leere Variablen definieren. So kann der Anwender eine Variable erzeugen, deren Dimension und deren Elemente erst später definiert werden.

3 Grafik

Scilab stellt eine Vielzahl von Grafikbefehlen bereit. Es muss aber darauf hingewiesen werden, dass die Einrichtung einer Grafik im Allgemeinen mehr Arbeit macht als in Matlab.

3.1 Einrichten des Grafifensters

Ein Grafikbefehl wird in einem anderen Fenster als dem Hauptfenster ausgeführt, das der Befehlsinterpreter verwaltet. Es können mehrere Grafikfenster gleichzeitig existieren, von denen jedes durch eine ganze Zahl identifiziert wird.

Gibt man nun einen Grafikbefehl ein, z.B. `plot2d` (siehe Abschnitt 3.2), wird Scilab ein Fenster und eine bei 0 beginnende ganze Zahl erzeugen. Anders als Matlab werden nacheiander eingegebene Grafikbefehle im aktuellen Fenster ausgeführt, wenn nicht ausdrücklich verlangt wird, dass Scilab ein neues öffnet.

Tabelle 3.1 versammelt die grundlegenden Operationen für den Umgang mit Grafik-Fenstern und einige Scilab-Befehle zu ihrer Ausführung. Wenn sich eine Operation auf ein nicht vorhandenes Fenster bezieht, wird dieses Fenster erzeugt. Im folgenden werden einige der Befehle besprochen, wegen der ausführlichen Beschreibung und den Einzelheiten zur Syntax verweisen wir aber auf die Online-Hilfe.

Der Befehl `scf` (set current figure) aktiviert ein Grafikfenster und macht es zum aktuellen, sodass sich alle Befehlseingaben nun auf dieses Fenster beziehen. Mit der Syntax `scf(num)` wird das Fenster mit der Nummer `num` aktiviert, wenn es existiert, andernfalls wird es zuerst erzeugt und dann aktiviert. Alternativ zur Nummer `num` kann man auch ein Handle `h` verwenden. Der Befehl `scf()` erzeugt ein neues Fenster mit einem Index, der um eins größer ist als die höchste jetzt existierende und aktiviert es. In jedem Fall bewirkt die Verwendung einer Ausgabevariablen (z.B. `h=scf(num)` die Erneuerung des Handles zum angesprochenen Bild.

Erzeugen eines neuen Fensters	`figure(num), scf(num)`
Fokus auf das Fenster `num`	`scf(num)`
Zeigen des aktuellen Fensters	`xselect()`
Löschen der Grafik im Fenster `num`	`clf(num), xclear(num)`
Schließen des Fensters `num`	`xdel(num)`
Liste aller Grafikfenster	`winsid()`

Tab. 3.1: Basis-Operationen auf Grafikfenstern und Befehle zu ihrer Umsetzung

Die Anweisung `clf` dient zum Löschen des Inhalts eines Grafikfensters: `clf()` oder `clf('clear')` entfernt alle im aktuellen Fenster enthaltenen Objekte, während `clf, 'reset')` darüberhinaus alle Eigenschaften der Darstellung auf die Anfangswerte zurücksetzt. Verwendet man die Syntax `clf(num), clf (num,'clear')` oder `clf(num, 'reset')`, bezieht sich der Befehl auf die mit `num` indizierte Darstellung; wenn die Darstellung `num` nicht existiert, wird sie erzeugt.

Alternativ zur Zahl `num` kann man einen Handle `h` verwenden.

Die Anweisung `xclear` säubert eines oder mehrere Fenster gleichzeitig; für den letzteren Fall ist das Argument des Befehls ein Vektor mit den Indizes der zu löschenden Darstellungen; wie üblich wird ein nicht existierendes Fenster erzeugt, wenn es Gegenstand der Anweisung ist. Die Syntax `xclear()` bezieht sich auf das aktuelle Fenster.

Die Anweisung `xdel` schließt eines oder mehrere Grafikfenster gleichzeitig; für den letzteren Fall ist das Argument des Befehls ein Vektor mit ganzzahligen Werten, die die zu schließenden Fenster indizieren. Die Syntax `xdel()` bezieht sich auf das laufende Fenster.

Der Befehl `figure` kann ohne Argument verwendet werden und fügt den vorhandenen Fenstern ein weiteres hinzu. In diesem Fall ist der Hintergrund grau, um ihn weiß zu machen, muss man den Befehl `clf` verwenden (oder ihn vermittels Handle modifizieren, wie in diesem Kapitel erklärt).

In Scilab gibt es keinen Befehl, der alle Fenster auf einmal schließt; es ist daher erforderlich, eine Anwenderfunktion zu erstellen. Um sie dann ständig verfügbar zu haben, ist es ist hilfreich, sie in die Datei `startscript.sce` zu kopieren, die beim Start ausgeführt wird, wie in Abschnitt 2.21 erklärt.

Ein mögliches Listing für die Funktion `xdelall` ist hier angegeben:

```
function xdelall()

    wins = winsid();
    for w = wins
        xdel(w);
    end

endfunction
```

3.2 2D-Grafiken

3.2.1 Der Befehl `plot2d`

Die elementare Anwendung des Befehls `plot2d` ist die folgende:

```
x = (-1:.1:3)';
y = x.^2;
plot2d(x,y)
```

Darin wird zunächst ein Vektor für die Abszisse definiert (als Spaltenvektor!) und dann bildet man den Ordinatenvektor. In diesem Fall haben wir uns für eine Parabel im Intervall $[-l, 3]$ mit Schrittweite 0.1 entschieden. Abbildung 3.1 zeigt das Ergebnis (um es in einer Datei zu speichern, siehe Abschnitt 3.5).

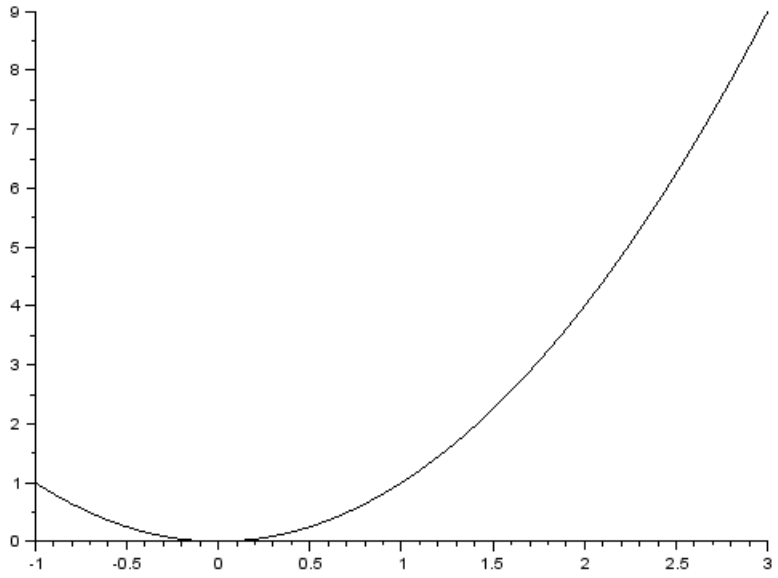

Abb. 3.1: Ein einfacher 2D-Plot

Der Befehl `plot2d` kann als zweites Argument eine Matrix aufnehmen, in diesem Fall interpretiert Scilab den Befehl als Überlagerung mehrerer Einzelbefehle, die sich auf denselben Spaltenvektor beziehen. Will man z.B. die Betragsfunktion der Abszissen zusammen mit der Parabel darstellen, kann man folgenden Code hinzufügen:

```
x =  (-1:.1:3)';
y = x.^2;
y2 = abs(x);
plot2d(x,[y y2]);
```

Abbildung 3.2 zeigt das Ergebnis. Beachten Sie, dass Scilab den Graphen für den zweiten Vektor `y2` blau gezeichnet hat.

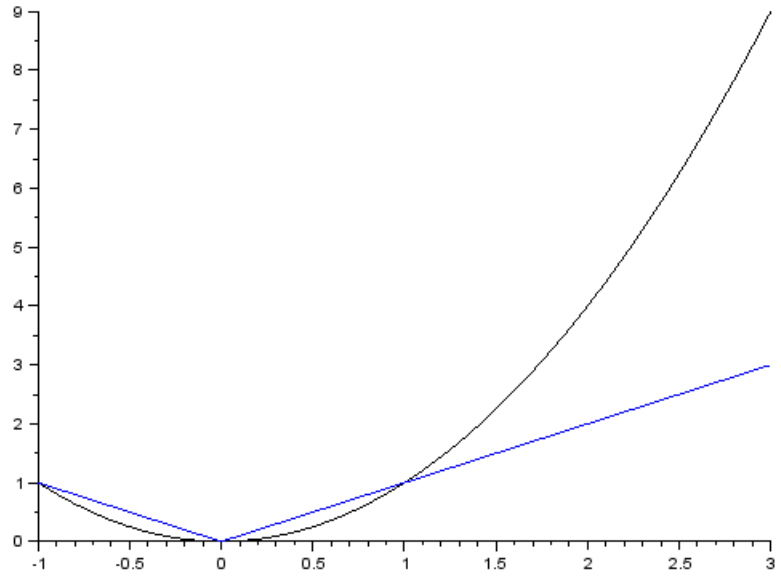

Abb. 3.2: Der Befehl plot2d mit einer Matrix als Eingabeparameter

3.2.2 Farbwahl

Mit weiteren Parametern kann man mit diesem Befehl auch die Farben festlegen, mit denen die Kurven gezeichnet werden sollen:

```
plot2d(x,[y y2],[optl opt2]);
```

führt nach den darzustellenden Daten in den Variablen `opt1` und `opt2` die Farbcodes auf. Die Tabelle 3.2 enthält die Codes für die 32 Farben der voreingestellten Palette, die Abbildung 3.3 zeigt die Palette wie sie mit dem Befehl `getcolor()` angezeigt wird und der die interaktive Wahl der Farbe gestattet. Wegen Details zur Farbverwaltung sei auf die Online-Hilfe verwiesen. Es ist möglich, verschiedene Paletten zu verwenden und sogar eigene Farben über ihre RGB-Codes zu definieren. Bei Fehlen des Farbarguments entnimmt Scilab die Werte der Tabelle in aufsteigender Ordnung, und tatsächlich waren die Farben im vorhergehenden Beispiel schwarz und blau.

Beachten Sie, dass von einem weiteren `plot2d`-Befehl - anders als in Matlab - kein neues Grafikfenster geöffnet wird, sondern Scilab zeichnet in das aktuelle Fenster mit schwarz beginnend, wenn nicht explizit eine andere Farbe festgelegt wird. Verlieren Sie aber keine Zeit mit der Suche nach einer unsichtbaren Kurve, wenn ihre Matrix mindestens 8 Spalten enthält: Die Farbe mit der Nr. 8 ist weiß

1	schwarz
2	blau
3	hellgrün
4	hellblau
5	rot
6	magenta
7	gelb
8	weiß
9-12	Blautöne
13-15	Grüntöne
16-18	helle Blautöne
19-21	Rottöne
22-24	Magentatöne
25-27	Brauntöne
28-31	Rosatöne
32	golden

Tab. 3.2: Codes der 32 Farben in der voreingestellten Farbpalette

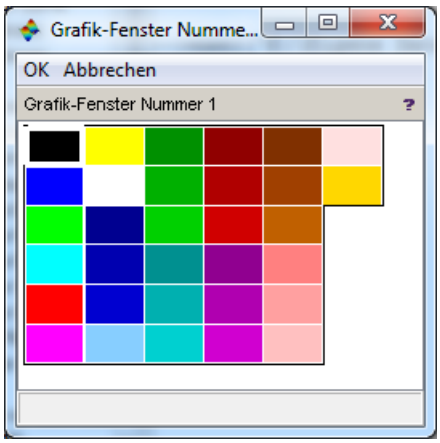

Abb. 3.3: Die voreingestellte Palette mit 32 Farben, die der Befehl `getcolor()` liefert. Die Zahlen sind den Feldern spaltenweise von oben nach unten zugeordnet.

3.2.3 Einstellen der Farbpalette

Die Palette mit den 32 voreingestellten Farben hat außer dem Problem mit Weiß auch den Mangel, dass sie nicht kontinuierlich ist. Viele Darstellungen erforden, dass in der Palette benachbarte Farben ähnlich sind. Ein Beispiel ist die Zeichnung der Oberfläche in Abschnitt 3.3.3. Eine Farbpalette ist als Matrix aufgebaut, die in der gewünschten Anzahl von Zeilen je drei Spalten für die drei Komponenten RGB (Rot Grün Blau) aus dem Interval $[0, 1]$ enthält. Der Anwender kann eine solche Farbpalette selber definieren oder er kann eine der vordefinierte Paletten

- C = graycolormap(n). Eine Skala mit n Graustufen,
- C = hotcolormap(n). Eine Skala von rot zu weiß mit n Stufen,

- C = jetcolormap(n). Eine Skala von blau über grün, gelb, orange zu rot mit n Stufen,

mit folgenden Anweisungen laden:

```
x =  (-1:.1:3)';
y = x.^2;
y2 = abs(x);
plot2d(x,[y y2]);
f = gcf();
f.color_map = C;
```

Zu beachten ist dabei, dass die Farbpalette nur für die aktuelle Zeichnung und nicht generell geladen wird, weshalb zuerst ein `plot`-Befehl einzugeben ist und dann die neue Farbpalette, evtl. gefolgt von weiteren `plot`-Befehlen. Einzelheiten zum Befehl `gcf` findet man in Abschnitt 3.4. Abbildung 3.4 enthält die 16 Farbstufen des Typs hotcolormap

Abb. 3.4: Farbpalette hotcolormap mit 16 Stufen

3.2.4 Titel, Beschriftungen, Legende

Die Grafik in Abb. 3.2 kann ohne weiteres besser verständlich gemacht werden, wenn ihr ein Titel hinzugefügt wird, Beschriftung der Achsen und eine Legende. Außerdem wird die Farbe der Kurven mit rot und blau vorgeschrieben:

```
x = (-1:.1:3)';
y = x.^2;
y2 = abs(x);
plot2d(x,[y y2], [2 5],leg='Parabel@Absolutwert');
xtitle ('Zwei einfache Funktionen','x','y')
```

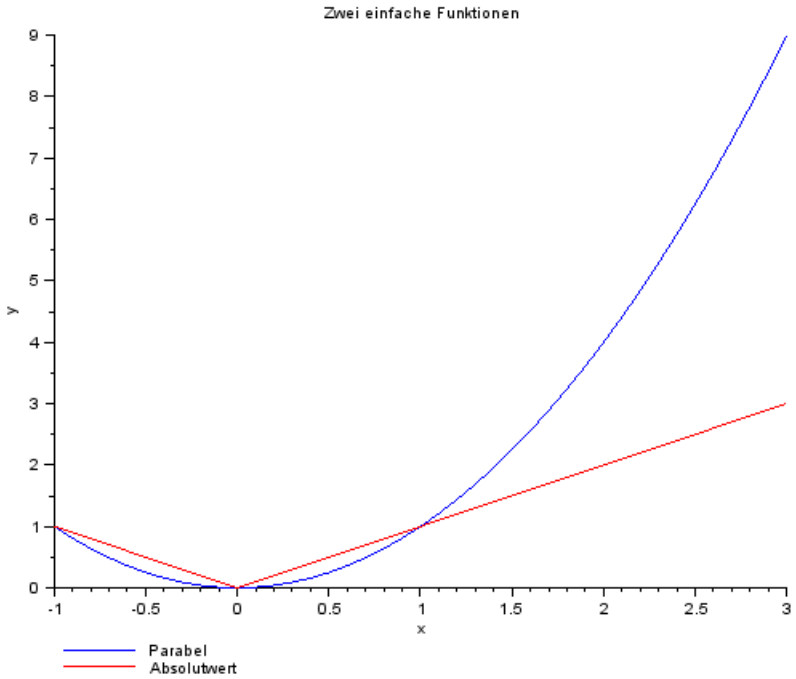

Abb. 3.5: Grafik mit Titel, Beschriftung und Legende.

Die voreingestellte Schriftgröße kann wie im vorliegenden Fall für die Bildschirmdarstellung richtig sein, nicht jedoch für das Speichern in einer Datei und anschließendes Ausdrucken. Zur Änderung der Schriftgröße muss man einige Zeilen Code hinzufügen:

```
x=(-1:.1:3)';
y=x.^2;
y2=abs(x);
nice_font=4;
plot2d(x,[y y2],[2 5]);
legend('Parabel','Absolutwert');
h = gce();
h.font_size = nice_font;
xtitle('Zwei einfache Funktionen','x','y')
a=gca();
a.font_size = nice_font;
a.x_label.font_size = nice_font;
a.y_label.font_size = nice_fout;
a.title.font_size = nice_font;
```

Abbildung 3.6 zeigt das Ergebnis. Die Befehle `gce` und `gca` werden in Abschnitt 3.4 erläutert, wo auch eine interaktive Methode zur Beeinflussung dieser Parameter angegeben wird.

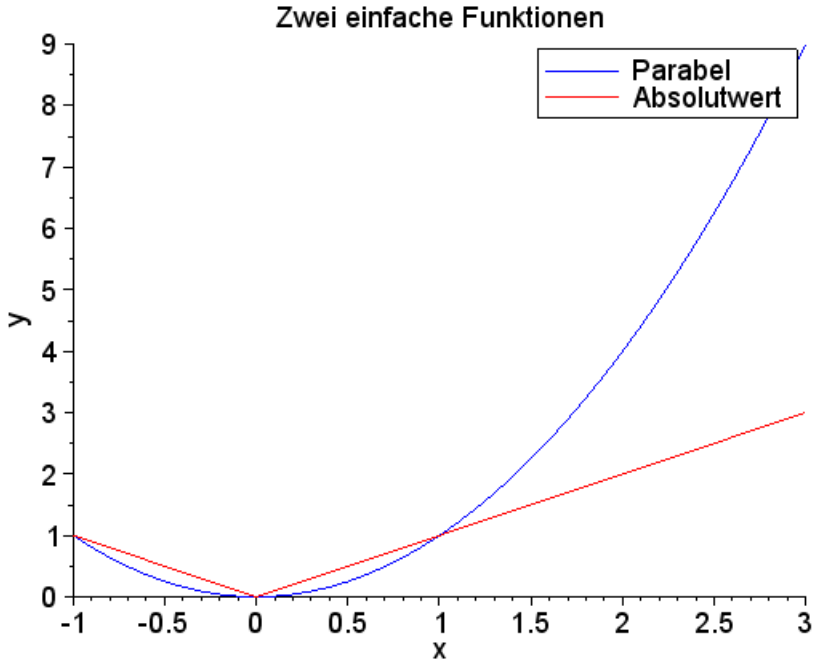

Abb. 3.6: Grafik mit Titel, Achsenbeschriftung und Legende in Schriftgröße 4

3.2.5 Zeichnen mit Symbolen

Statt mit Linien, genauer: statt mit Strichen, die die Punkte des Ordinatenvektors verbinden, kann eine Kurve auch mit Symbolen dargestellt werden. Diesem Zweck dient die Option `style`. Dabei bewirkt eine positive Zahl die Wahl einer Farbe, während eine negative Zahl schwarze Symbole gemäß Tabelle 3.3 zeichnet.

style	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
Symbol	·	+	×	⊕	♦	◊	△	▽	♣	◯

Tab. 3.3: Symbole für `style`

Das folgende Beispiel zeigt den Gebrauch der Option `style`, wenn für die Werte eines Vektors ein Symbol gezeichnet werden soll. Abbildung 3.7 zeigt das Ergebnis.

```
x = (-1:.1:3)';
y = x.^2;
plot2d(x,y,-7);
```

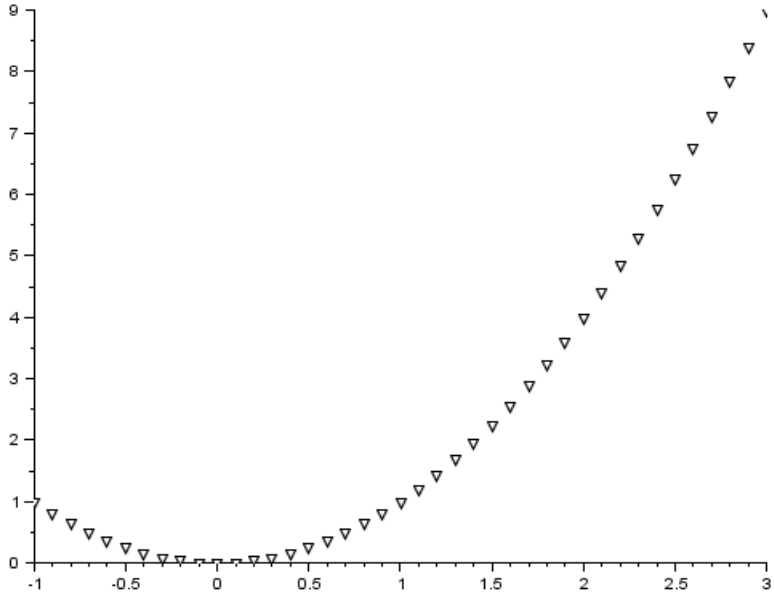

Abb. 3.7: Ein Beispiel für die Option style

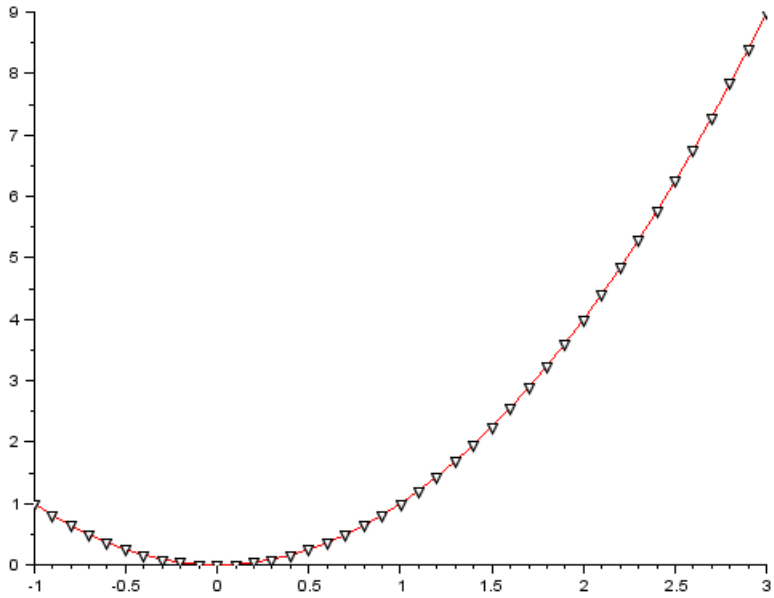

Abb. 3.8: Ein Beispiel für kombinierte Optionen

Abb. 3.8 zeigt ein Beispiel, wie die Optionen kombiniert werden können, um scharze Dreiecke auf einer roten Linie zu erhalten.

```
x = (-1:.1:3)';
y = x.^2;
plot2d(x,[y y],[5 -7]);
```

3.2.6 Erzeugung mehrerer Grafiken in einem Fenster

Um mehrere Grafiken in einem einzigen Fenster unterzubringen, gibt es den Befehl `subplot`, der zwei verschiedene Schreibweisen zulässt. Man muss diesem Befehl die drei Parameter `m`, `n` und `p` übergeben, wobei `m` und `n` Zeilen- und Spaltenzahl einer virtuellen Matrix bedeuten, in die das Fenster aufgeteilt werden soll und `p` den ausgewählten Eintrag. Der folgende Code prodziert die Abbildung 3.9:

```
scf
x = (-1:.1:3)';
y = x.^2;
subplot(221),plot2d(x,y,-6)
subplot(222),plot2d(x,y,-1)
subplot(2,2,3),plot2d(x,[y y],[5 -9])
subplot(2,2,4),plot2d(x,[y y],[-9 5])
```

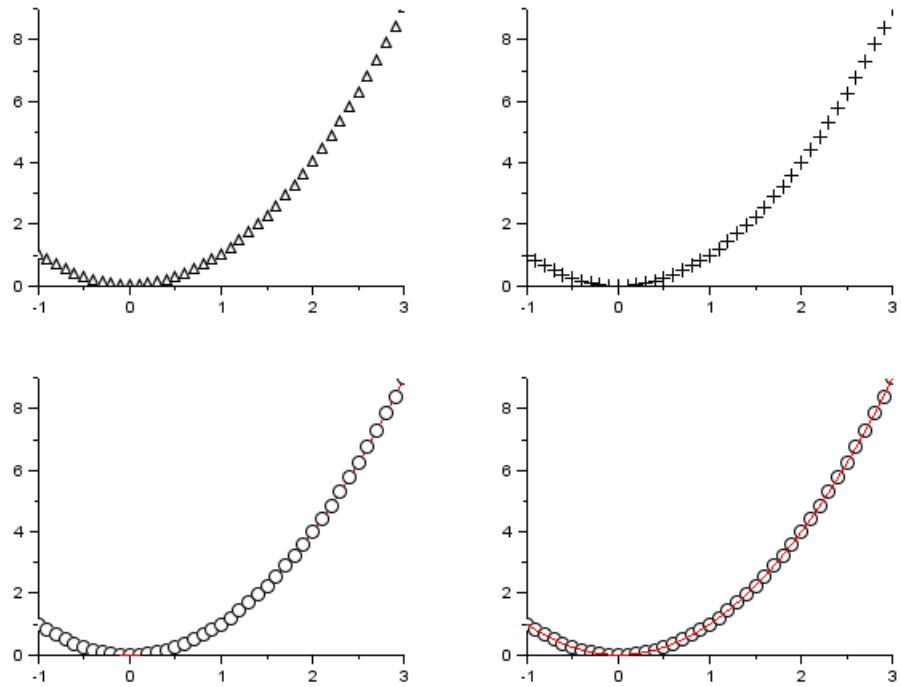

Abb. 3.9: Wirkung des Befehls `subplot`

3.2.7 Festlegung der Maßstäbe

In der Voreinstellung legt Scilab die Länge der Achsen so fest, dass alle Punkte des Befehls `plot2d` gezeichnet werden. Mit der Option `frameflag` kann die Länge der Achsen so manipuliert werden, wie es der folgende Code bzw. Tabelle 3.4 oder die Abbildungm 3.10 zeigen.

```
scf
theta=0:.1:2*%pi;
x=cos(theta);
y=sin(theta);
subplot(221),plot2d (x,y,5);  xgrid
subplot(222),plot2d (x,y,5,frameflag=1,rect=[-2 -2 2 2]);xgrid
subplot(2,2,3),plot2d (x,y,5,frameflag=3,rect=[ -2 -2 2 2]);xgrid
subplot(2,2,4),plot2d (x,y,5,frameflag=4);xgrid
```

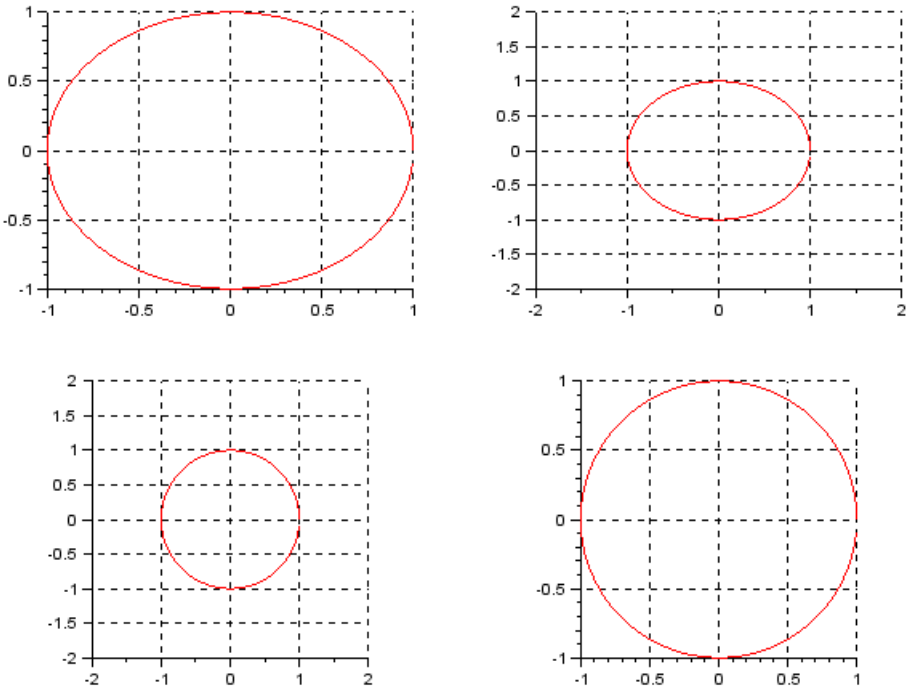

Abb. 3.10: Ein Kreis mit unterschiedlichen Skalen

Man kann die Skalen auch noch nach Ausführung von `plot2d` verändern. Eine Möglichkeit bietet das Grafik-Werkzeug `Vergrößern und Markieren` des interessierenden Gebietes mit der Maus. Genauer ist die Verlängerung der Achsen durch Verändern der Eigenschaften des Grafikobjektes. Siehe dazu Abschnitt 3.4, die zu verändernde Eigenschaft ist `data_bounds`

3.2.8 Einführung einer logarithmischen Skala

Man kann als Option des Befehls `plot2d` auch logarithmische Skalenteilungen erhalten, sowohl für eine als auch für beide Achsen. Der folgende Code erzeugt zwei Vektoren für Punkte, eine Teilung ist linear, die andere ist logarithmisch und zeichnet dann zwei Grafiken in dasselbe Fenster; die zweite Grafik ist halblogarithmisch und der dargestellte Graph ist ein Segment.

```
x=l:l:50
y=logspaee(l,6,50)
subplot(211)
plot2d(x,y,logflag="nn")
subplot(212)
plot2d(x,y,logflag="nl")
```

3.2.9 Gestaltung der Rahmen und Achsen

Wie die Skalen sind auch die Achsen mit Optionen des Befehls `plot2d` veränderbar, die in Tabelle 3.5 aufgelistet sind.

axesflag	Achsen
axesflag=0	ohne Rahmen, weder Achsen noch Skalen
axesflag=l	mit Rahmen, Achsen und Skalen (x unten, y links)
axesflag=2	mit Rahmen, aber ohne Achsen und Skalen
axesflag=3	mit Rahmen, Achsen und Skalen (x unten, y rechts)
axesflag=4	ohne Rahmen, aber mit Achsen und Skalen
axesflag=5	ohne Rahmen, Achsenkreuz mittig mit Skalen

Alternativ kann mit `gca()` und dem Parameter `data_bounds` des Handle gearbeitet werden, wie in Abschnitt 3.4 beschrieben.

3.2.10 Einfügen oder Ändern von Text

Der Befehl zum Einfügen von Text in eine Grafik ist `xstring`

```
-->xstring(x,y,'Text')
```

Darin sind `x` und `y` die Koordinaten der unteren linken Ecke des Textes.

3.2.11 Das Werkzeug `datatip`

Mit `datatip` wird ein vor allem grafisches Werkzeug bezeichnet, das die interaktive Visualisierang von Punkten einer zweidimensionalen Grafik ermöglicht. Jedes Grafik-Fenster erlaubt die Aktivierung bzw. Deaktivierung des `datatip`-Modus sowohl über ein Icon als auch mit dem entsprechenden Befehl im Menü `Editieren`. Den klickt man mit der linken Maustaste an, und am unteren Fensterrand erscheint ein Hinweis zur Vorgehensweise. Ein linker Mausklick auf einen Kurvenpunkt lässt dessen Koordinaten erscheinen.

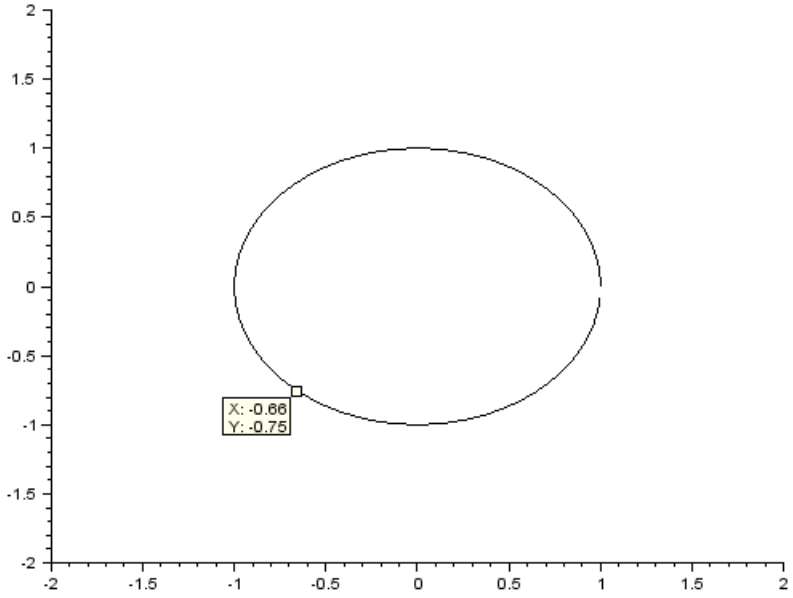

Abb. 3.11: Beispiel für die Anwendung von `datatip` zur Anzeige der Koordinaten eines Punktes

3.2.12 Weitere Grafiktypen

Außer dem Befehl `plot2d` gibt es noch andere vordefinierte Grafiktypen, die in Abschnitt 3.8 vorgestellt werden.

3.3 3D-Grafiken

3.3.1

3.3.2 Zeichnen einer Kurve

Der Befehl zum Zeichnen von Punkten ist natürlich auch geeignet zum Zeichnen von Kurven in 3D. Es gibt zwei Versionen desselben Befehls: `param3d()` und `param3d1()`, die sich nur für den Fall unterscheiden, dass Sie zwei Kurven gleichzeitig zeichnen möchten. Die einfachste Syntax ist:

```
-->param3d1(x,y,z)
```

worin die drei Vektoren, Zeilen- oder Spaltenvektoren, die Koordinaten der Punkte enthalten, die durch Sekanten verbunden werden. Man kann auch Matrizen als Eingabeparameter

verwenden. Dann werden die Spalten als getrennte Befehle behandelt. Das folgende Beispiel liefert die Darstellung in Abbildung 3.13.

```
t=0:0.01:5*%pi; t=t';
param3d1([sin(t) 3+sin(t)],[cos(t) cos(t)],[t/5 t])
```

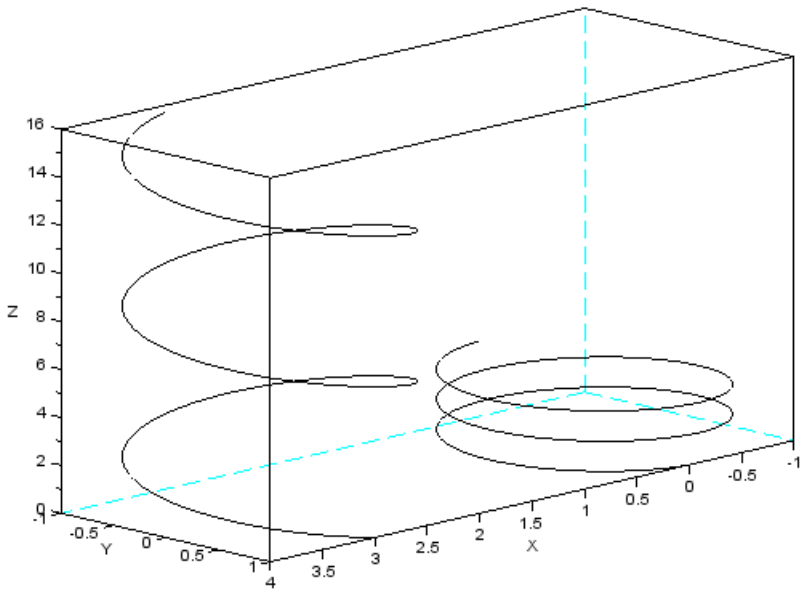

Abb. 3.13: Mit dem Befehl `param3d1()` erzeugte Raumkurven

3.3.3 Darstellung einer Fläche

Der einfachste Weg eine Fläche zu zeichnen, ist die Auswertung einer Funktion

$$x = f(x, y) (3.1) \tag{3.1}$$

mit Angabe des rechteckigen Definoitionsbereiches. Als Beispiel nehmen wir die Funktion

$$f(x, y) = cos(x)cos(y) \text{ für } (x, y) \in [0, 2\pi] \times [0, 2\pi] (3.2) \tag{3.2}$$

Dann definiert man einfach zwei Vektoren für den Definitionsbereich und eine Matrix $z(i, j) = f(x(i), y(j))$:

```
x=linspace(0,2*%pi,31)
y=x;
z=cos(x)'*cos(y)
```

Der Befehl `plot3d(x,y,z,[theta,alpha,leg,flag,ebox])` wird einfachheitshalber nur mit dem Parameter `flag`, einem dreielementigen Vektor, aufgerufen, der die Darstellung der Fläche steuert. Im einzelnen

- flag=[mode, type, box]:
  - mode: ganze Zahl für die Farbwahl,
    - mode>0: Fläche in gewählter Farbe mit Drahtgitter,
    - mode=0: Fläche nur als Drahtgitter (mesh),
    - mode<0: Fläche in gewählter Farbe ohne Drahtgitter,
  - type: ganze Zahl zwischen 0 und 6 zur Wahl der Skalen (Details in der Online-Hufe),
  - box: ganze Zahl zwischen 0 und 4 für den Rahmen (Details in der Online-Hilfe)

Schließlich erzeugt der folgende Code mit zwei Werten für den Parameter `mode` die Abbildung 3.14:

```
subplot(121)
plot3d(x,y,z,flag=[2 4 4])
subplot(122)
plot3d(x,y,z,flag=[0 4 4])
```

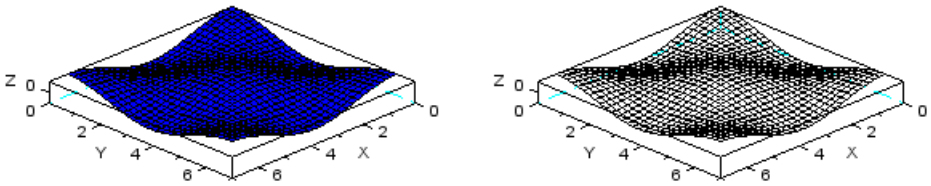

Abb. 3.14: Beispiele dreidimensionaler Flächen - plot3d() mit mode=2 (links) und mode=0 (rechts)

Ein offenkundidiges Problem stellt in Abbildung 3.13 die Schwierigkeit dar, die Höhe der einzelnen Punkte der Fläche zu erhalten. Es kann hilfreich sein, den Höhen verschiedene Farben zuzuordnen. Dafür muss der Befehl `plot3d1()` verwendet werden. Es kann auch notwendig werden, die Farbpalette zu wechseln, wie in Abschnitt 3.2.3 beschrieben.

Die Abbildungen 3.15 und 3.16 erhält man durch Laden der vordefinierten Farbpaletten mit abgestuften Rot- bzw. Grautönen mit dem folgenden Code

```
h=scf() // Abb. 3.15
h.color_map = hotcolormap(32)
subplot(121)
plot3d1(x,y,z,flag=[1 4 4])
subplot(122)
plot3d1(x,y,z,flag=[-1 4 4])

h=scf() // Abb. 3.16
h.color_map = graycolormap(32)
subplot(121)
plot3d1(x,y,z,flag=[1 4 4])
subplot(122)
plot3d1(x,y,zjflag=[-1 4 4])
```

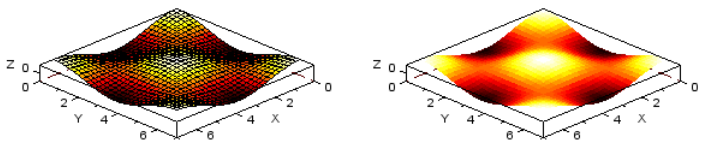

Abb. 3.15: Beispiele dreidimensionaler Flächen
`plot3d1()` und `hotcolormap` mit mode>0 (links) und mode<0 (rechts)

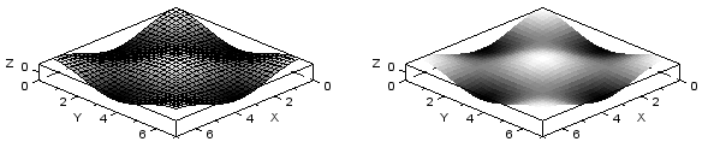

Abb. 3.16: Beispiele dreidimensionaler Flächen
`plot3d1()` und `greycolormap` mit `mode > 0` (links) und `mode < 0` (rechts)

3.4 Eigenschaften von Grafik-Objekten erforschen

Die Grafik-Befehle haben eine Reihe von Optionen, mit denen das Aussehen der in einer Abbildung dargestellten Objekte beeinflusst werden kann. Jedoch sind nicht alle Möglichkeiten eines Befehls über diese Optionen erreichbar. So kann beispielsweise die Farbe des Symbols über den Befehl nicht verändert werden.

Daher ist es nötig, auf die Eigenschaften eines Objektes auf anderem Wege zuzugreifen. Zeichnen wir eine Grafik mit zwei „Achsen“ und mit einem Text:

```
h=scf(9);
theta=0:.l:2*%pi
x=cos(theta)
y=sin(theta)
subplot(121)
plot2d(x,y)
h1=gca()
h1.font_size=2
subplot(122)
h2=gca()
h2.font_size=2
plot2d(x,y,-7,frameflag=4)
xstring(-0.2,0,'ein Text')
```

wie Abbildung 3.17 zeigt.

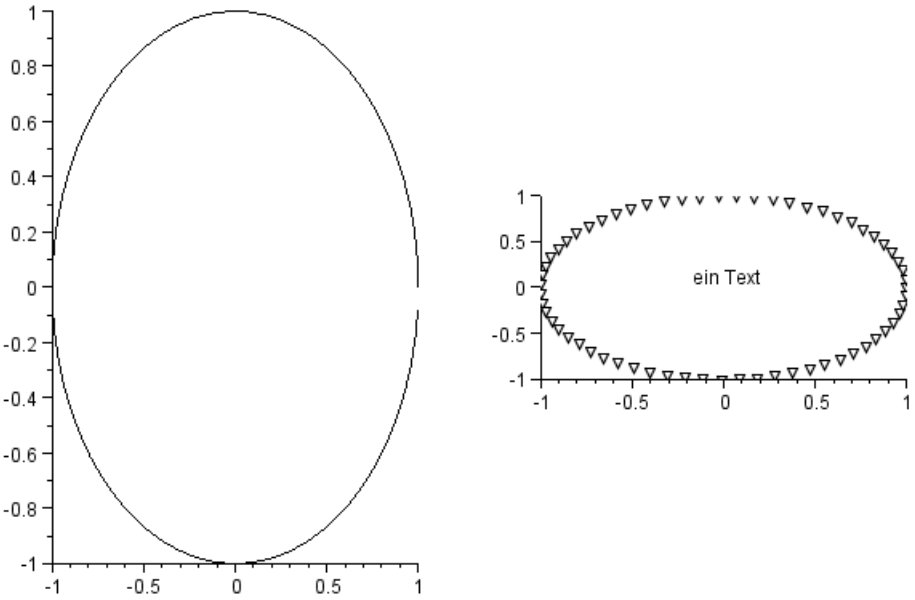

Abb. 3.17: Einfache Grafik mit zwei Objekten

Der Befehl `gcf()` (get current figure) liefert einen Handle zur Abbildung:

```
-->gcf()
 ans  =

Handle of type "Figure" with properties:
========================================
children: ["Axes","Axes"]
figure_position = [200,200]
figure_size = [614,591]
axes_size = [610,460]
auto_resize = "on"
viewport = [0,0]
figure_name = "Grafik-Fenster Nummer %d"
figure_id = 9
info_message = ""
color_map = matrix 32x3
pixmap = "off"
pixel_drawing_mode = "copy"
anti_aliasing = "off"
immediate_drawing = "on"
background =  -2
visible = "on"
rotation_style = "unary"
event_handler = ""
event_handler_enable = "off"
user_data = []
resizefcn = ""
closerequestfcn = ""
tag = ""
```

woraus man erkennt, dass es zwei *children* gibt entsprechend den beiden gezeichneten Objekten. Um die Eigenschaften, z.B. des linken anzuzeigen, schreibt und erhält man:

```
-->h.children(1)
 ans  =

Handle of type "Axes" with properties:
======================================
parent: Figure
children: ["Text","Compound"]

visible = "on"
```

und etwa 40 weitere Zeilen.

Um evtl. interessierende Parameter zu modifizieren, um z.B. nur die senkrechten Gitterlinien zu erhalten, gibt man ein:

```
--> h.children(1).grid=[1 -1]
```

Nun ist klar, dass auch dieses Objekt zwei *children* besitzt, wovon eines die mit Symbolen dargestellte Kurve ist, deren Farbe verändert werden kann:

```
--> h.children(1).children.children.mark_foreground=5
```

oder Stil und Strichstärke der linken Kurve.

```
--> h.children(2).children.children.line_style=2;
--> h.children(2).children.children.thickness=2;
```

was die Abbildung 3.18 zeitigt.

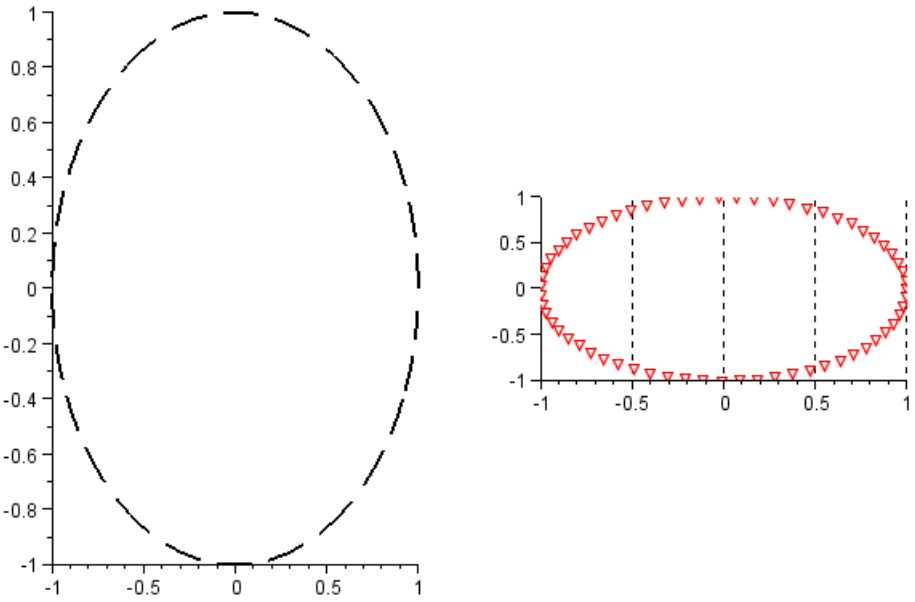

Abb. 3.18: Einfache Grafik mit zwei über ihre Handles modifizierten „Achsen“

Die Modifikation von Grafiken über ihren Handle ist angezeigt, wenn die Operation wiederholbar sein muss. Für eine einmalige Änderung kann im Menü Editieren des Grafikfensters der Menupunkt `Voreinstellung der Darstellung` angewählt werden. Man gelangt nun in das Dialogfenster `Figure Editor` und kann schließlich `Axes(1)` anwählen (siehe Abbildungen 3.19 und 3.20).

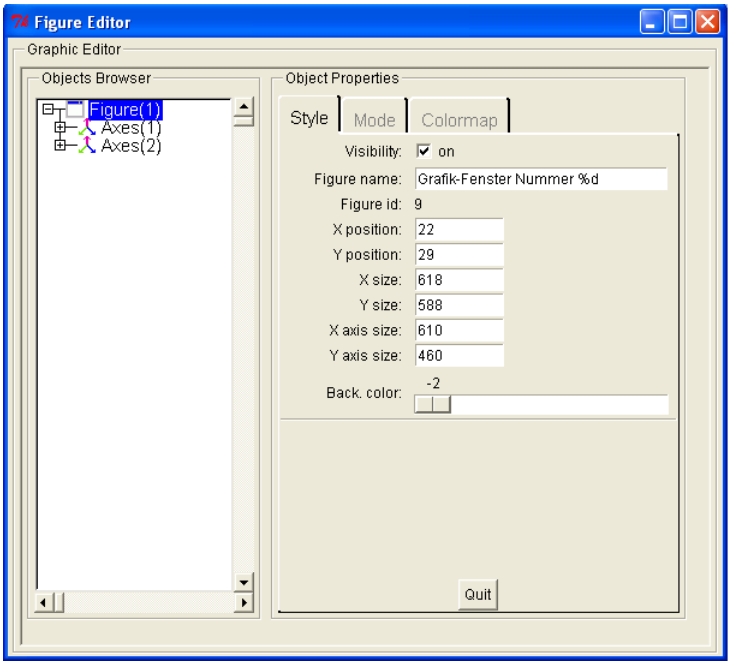

Abb. 3.19: Dialogfenster `Figure Editor`

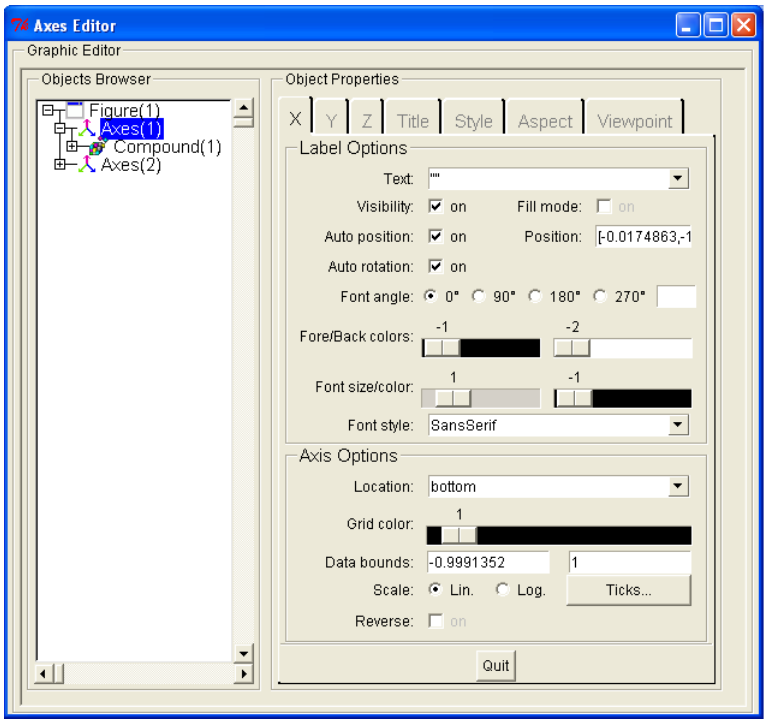

Abb. 3.20: Dialogfenster `Axes Editor`

Zwei weitere Befehle bieten Zugriff auf den Handle eines Objektes und können von Nutzen sein:

```
h = gca();
h = gce();
```

Sie beziehen sich auf das aktuelle Objekt bzw. die aktuelle Einheit. Zum ersten Befehl ist dem, was zum Befehl `gcf` gesagt worden ist, nichts hinzuzufügen, der das *Parent*-Objekt repräsentiert. Der zweite bezieht sich auc das zuletzt erzeugte Objekt und kann helfen, schlankeren Code zu schreiben.

3.5 Exportieren der Grafik

Man kann Grafik mit dem Befehle `xs2...` in den üblichen Formaten speichern, also mit `xs2gif`, `xs2jpg`, `xs2png`, `xs2ppm`, `xs2eps`, `xs2pdf`, `xs2svg`, `xs2ps` und - nur für Windows - mit `xs2emf`.

Das Verhältnis der Zeichengröße auf dem Bildschirm und in einer Datei kann sich - und das gilt auch für Matlab - beim Speichern verändern. Deshalb ist seine Anpassung erforderlich.

Zur Anpassung der Pixelzahl eines Bildes kann auf einen geeigneten Parameter zugegriffen werden. Um z.B. ein Bild mit 640×480 Pixeln zu erzeugen, schreibt man:

```
h=gcf();
h.axes_size=[640 480]
```

wobei sichergestellt sein muss, dass die Pixelzahl in `h.figure_size` größer ist.

3.6 Programmierung einer grafischen Benutzeroberfläche (GUI)

In Scilab kann man eine grafische Benutzeroberfläche (Graphic User Interface - GUI) für interaktive Anwendungen erstellen. Der GUI können Steuerelemente wie Buttons, Schieberegler, Textfelder und vieles andere eingefügt werden, was auch in anderen Programmmiersprachen dafür zur Verfügung steht. Es ist auch möglich, in einem solchen Fenster Objekte zu erzeugen, die Grafiken enthalten.

Der wichtigste Befehl zur Verwaltung einer GUI ist `uicontrol`. Er ermöglicht die Definition der Objekte, mit denen interagiert werden soll. Die Tabelle 3.6 enthält die wichtigsten Objekte bzw. die Befehle, mit denen diese Objekze erzeugt werden können.

`Pushbutton`	Button, generell verwendet zum Aufruf eines Callback
`Radiobutton`	Button mit zwei sich gegenseitig ausschließenden Zuständen
`Checkbox`	Button mit zwei Zuständen (für mehrere unabhängige Zustände
`Edit`	Feld für eine veränderbare Zeichenkette
`Text`	Feld für eine im allgemeinen unveränderliche Zeichenkette
`Slider`	Schieberegeler zur Wahl eines Wertes aus einem Intervall mit der Maus
`Frame`	Feld zur Gruppierung ähnlicher Steuerelemente
`Listbox`	Liste, aus der mit der Maus ausgewählt werden kann
`Popupmenu`	List, die sich auf Mausklick für die Auswahl öffnet

Tab. 3.6: Mit dem Befehl `uicontrol` definierbare Objekte

Anhand eines einfachen Beispiels werden wir in diesem Abschnitt die Grundlagen liefern, um eine GUI mit `Slider, Pushbutton, Text` und `Radiobutton` zu erstellen. Das folgende Listing, das unter `manuale_gui.sce` zu speichern ist, erzeugt ein Fenster, in dessen rechtem Teil eine einfache Sinuskurve dargestellt ist, und in deren linkem Teil Raum ist für Steuerelemente, mit denen die Grafik interaktiv verändert werden kann. Abbildung 3.21 zeigt den Anfangszustand der GUI.

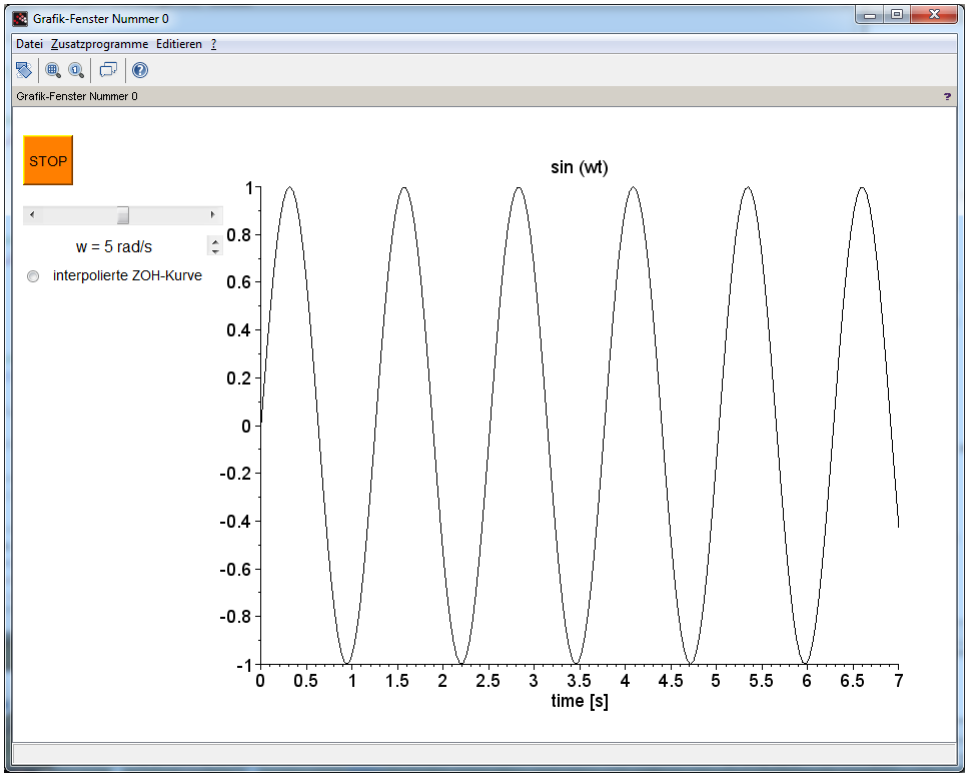

Abb. 3.21: Anfangszustand des Fensters

```
//
// exec('manuale_gui.sei');
//
```

```
// update nach Bewegung des Schiebereglers
function update_slider()

  mydraw();

endfunction

// update der durchgezogenen/abgesetzten Kurve
function update_radio()

  mydraw();

endfunction

// erneutes Zeichen der Kurve
function mydraw()

  t = linspace(0,7,200)
  hf = gcf();
  hf.immediate_drawing="off";
  h=gca();
  if (h.children~=[])
    delete(h.children);
  end
  w = h_slider.value/10;
  plot2d(t,sin(w.*t));
  h.axes_bounds=[0.15,0,.9,1]
  xtitle("sin (wt)")
  xlabel("time [s]")
  h.font_size = 4;
  h.x_label.font_size = 4;
  h.title.font_size = 4;

  if (h_radio.value==0)
    h.children.children.polyline_style=1;
  else
    h.children.children.polyline_style=2;
  end

  hf.immediate_drawing="on";
endfunction

// erstes Zeichnen der Kurve
function myInitDraw()

  t = linspace(0,7,200)
```

```
  w = 5;
  plot2d(t,sin(w.*t));
  h = gca();
  h.axes_bounds=[0.15,0,.9,1]
  xtitle("sin (wt)")
  xlabel("time [s]")
  h.font_size = 4;
  h.x_label.font_size = 4;
  h.title.font_size = 4;

endfunction

//
// MAIN
//
//xdelall(); // siehe Abschnitt 3.1
funcprot(0);

screen_size = get(0,"screensize_px");
sizex = .75*screen_size(3);
sizey = .75*screen_size(4);
h_graf = scf(0);
h_graf.figure_size = [sizex sizey];
h_graf.figure_position =...
  [(screen_size(3)-sizex)/2 (screen_size(4)-sizey)/2];

myInitDraw();
Text = strcat(["w = 5 rad/s"]);

// stop
h_stop=uicontrol(h_graf,...
  "style","pushbutton",...
  "string","STOP",...
  "fontsize",14,...
  "backgroundcolor",[1 0.5 0],...
  "position",[10 sizey-210 50 50],...
  "callback","xdel(0)");

// Slider
h_text_slider = uicontrol(h_graf,...
  "style","text",...
  "horizontalalignment","center",...
  "string",Text,...
  "fontsize",16,...
  "backgroundcolor",[1 1 1],...
  "position",[10 sizey-280 200 20]);
```

```
h_slider=uicontrol(h_graf,...
  "style","slider",...
  "Min",0,...
  "Max",100,...
  "value",50,...
  "position",[10 sizey-250 200 20],...
  "callback","update_slider();...
  Text = strcat([''w = '' string(h_slider.value/10) '' rad/s'']);...
  h_text_slider.string = Text");

// radiobutton
h_text_radio = uicontrol(h_graf,...
  "style","text",...
  "horizontalalignment","left",...
  "string","interpolierte ZOH-Kurve",...
  "backgroundColor" , [1 1 1],...
  "fontsize",14,...
  "position",[40 sizey-310 170 20]);
h_radio = uicontrol(h_graf,...
  "style","radiobutton",...
  "Min",0,...
  "Max",1,...
  "value",0,...
  "backgroundcolor",[1 1 1],...
  "position",[10 sizey-310 20 20],...
  "callback", "update_radio()");

//
// ENDE MAIN
//
```

Als erstes betrachten wir die Struktur des Programms. Es gibt einige Funktione, die vor dem eigentlichen Skipt definiert werden und schließlich das von den Kommentaren `MAIN` und `ENDE MAIN` begrenzte Hauptprogamm.

Die beiden ersten Funktionen sind *dummy*-Funktion, die jeweils bei einer Interaktion mit den Schaltern `Slider` bzw. `Radiobutton` aufgerufen werden, eine Funktion zur Initialisierung der Grafik und eine Funktion zur Anpassung dieser Grafik. Für den Schalter `pushbutton` wird keine Funktion benötigt, hier genügt ein einziger String für das Callback.

Die beiden ersten Anweisungen des Hauptprogramms

```
xdelall();
funcprot(0);
```

sind Befehle, die alle Fenster schließen (siehe Abschnitt 3.1) und Warnmeldungen verhindern, wenn Funktionen bei wiederholtem Aufruf des Skripts neu definiert werden. Sie sind nicht unbedingt erforderlich.

Die folgenden Befehle

```
screen_size = get(0,"screensize_px");
sizex = .75*screen_size(3);
sizey = .75*screen_size(4);
h_graf = scf(0);
h_graf.figure_size = [sizex sizey];
h_graf.figure_position =...
 [(screen_size(3)-sizex)/2 (screen_size(4)-sizey)/2];
```

dienen zur Definition des Grafikfensters und darin zum Aufbau der GUI, die an die aktuellen Bildschirmabmessungen gebunden wird. Nach der Initialisierung der Grafik mit `myInitDraw()` definieren wir die nötigen Befehle. Der erste betrifft den Pushbutton:

```
// stop
h_stop=uicontrol(h_graf,...
"style","pushbutton",...
"string","STOP",....
"fontsize",14,...
"backgroundcolor",[1 0.5 0],...
"position",[10 sizey-210 50 50],...
"callback","xdel(0)");
```

wofür der Handle auf das Grafikfenster zu definieren ist. Darin ist das Objekt festzulegen, dessen Typ, die Parameter für seine Gestaltung und schließlich das Callback, d.h. der Befehl, der ausgeführt wird, wenn der Anwender auf das Steuerelement klickt. In diesem Fall ist es ein banaler Schließbefehl für das einzige geöffnete Fenster: `xdel(0)`.

Die Definition der übrigen Objekte verläuft im Prinzip ähnlich, der einzige Unterschied ist, das nun das Callback eine Anwenderfunktion aufruft. Dann sehen wir uns einige interessante Anweisungen aus der Funktion `MyDraw` an, die bei jeder Interaktion mit den Grafikobjekten aufgerufen wird. Als erstes ist darauf hinzuweisen, dass alle Anweisungen der Funktion auf diese Weise ummantelt werden:

```
hf = gcf()
hf.immediate_drawing = "off"
...
// Anweisungen
...
hf.immediate_drawing = "on"
```

Es ist ratsam, sich diese Struktur zu merken, um Probleme, auch mit Bildschirmflimmern, zu vermeiden. Ein weiteres Code-Element dient zum Löschen der aktuellen Kurve auf dem Bildschirm

```
h=gca();
if (h.children~=[])
  delete(h.children);
end
```

worin von *children* Gebrauch gemacht wird, das in Abschnitt 3.4 vorgestellt wurde.

Zu beachten ist, dass die Funktion `MyDraw` auf den Zustand der Grafikobjekte einwirkt und dass man ihn deshalb für seine Zwecke nutzen kann. In diesem besonderen Fall benutzt man den Wert des Schiebereglers, den man über die Variable `h_slider.value` erreicht, und den Wert des `Radiobutton`, der über die Variable `h_radio.vaue` zugänglich ist.

Abbildung 3.22 zeigt schließlich den Zustand der GUI nach dem Verstellen des Schiebereglers und Aktivierung des `Radiobutton`.

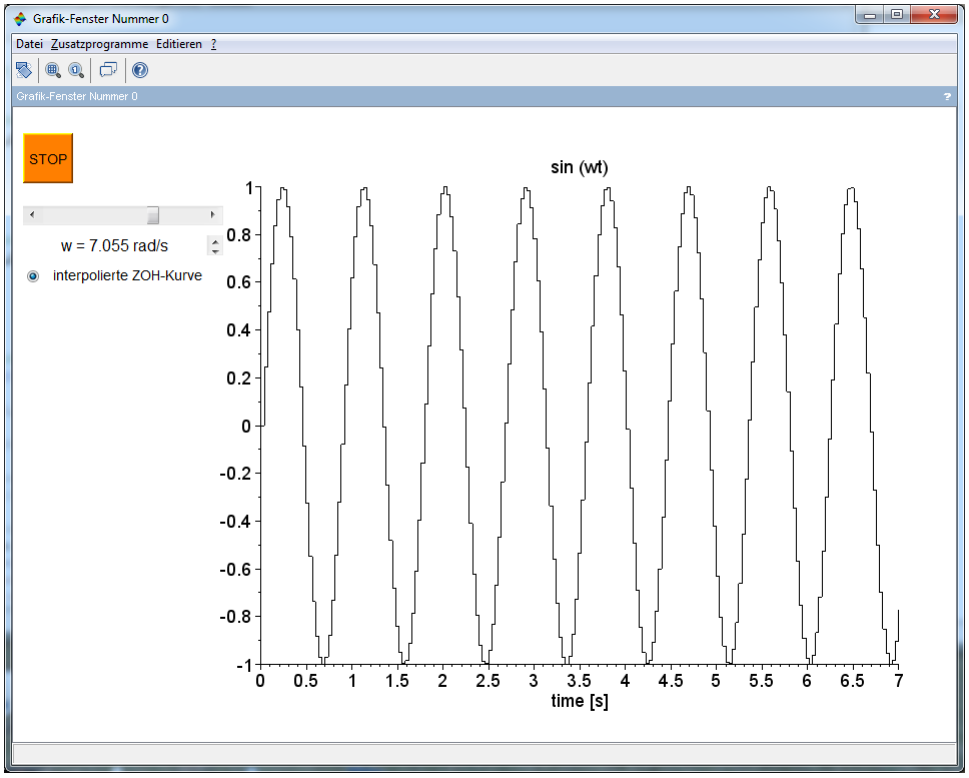

Abb. 3.22: Zustand der GUI nach dem Verstellen des Schiebereglers und Aktivierung des `Radiobutton`

Es folgt eine Liste mit Befehlen zur Gestaltung einer grafischen Benutzeroberfläche

`about`	- show 'about scilab' dialog box
`addmenu`	- interactive button or menu definition
`clipboard`	- Copy and paste strings to and from the system clipboard
`close`	- close a figure
`delmenu`	- interactive button or menu deletion
`exportUI`	- Call the file export graphical interface
`figure`	- create a figure
`findobj`	- find an object with specified property
`gcbo`	- Handle of the object whose callback is executing
`getcallbackobject`	- Return the handle of the object whose callback is executing
`getinstalledlookandfeels`	- returns a string matrix with all Look and Feels
`getlookandfeel`	- gets the current default look and feel
`getvalue`	- xwindow dialog for data acquisition
`messagebox`	- Open a message box
`printfigure`	- Opens a printing dialog and prints a figure
`printsetupbox`	- Display print dialog box
`progressionbar`	- Draw a progression bar
`root_properties`	- description of the root object properties
`setlookandfeel`	- sets the current default look and feel
`setmenu`	- interactive button or menu activation
`toolbar`	- show or hide a toolbar
`toprint`	- Send text or figure to the printer
`uicontrol`	- create a Graphic User Interface object
`uigetcolor`	- Opens a dialog for selecting a color
`uigetdir`	- dialog for selecting a directory
`uigetfile`	- dialog window to get a file(s) name(s), path and filter index
`uigetfont`	- Opens a dialog for selecting a font
`uimenu`	- Create a menu or a submenu in a figure
`unsetmenu`	- interactive button or menu or submenu de-activation
`usecanvas`	- Get/Set the main component used for Scilab graphics
`waitbar`	- Draw a waitbar
`x_choices`	- interactive Xwindow choices through toggle buttons
`x_choose`	- interactive window choice (modal dialog)
`x_choose_modeless`	- interactive window choice (not modal dialog)
`x_dialog`	- Xwindow dialog
`x_matrix`	- Xwindow editing of matrix
`x_mdialog`	- Xwindow dialog
`x_message`	- X window message
`x_message_modeless`	- X window modeless message
`xgetfile`	- dialog to get a file path

3.7 Erzeugung von Animationen

Um Animationen zu erzeugen, die nicht flimmern, muss man auf die bekannte Grafik-Technik des Doppelpuffers zurückgreifen. Für eine einfache Animation (ein roter Ball bewegt sich über den Bildschirm) zeigt der folgende Code die einfachste, wenn auch nicht die beste Methode. Man beachte, dass für die Aktivierung des Doppelpuffers nur ein paar Befehle erforderlich sind, nämlich `clf()` zum Löschen der Abbildung, während der Befehl `drawlater()` dafür sorgt, dass das neue Bild zunächst in den unsichtbaren Puffer gezeichnet werden kann, bevor es mit `drawnow` angezeigt wird.

```
N = 1000;
r1 = .2; r2 = .2;
scf();
for i=1:N
    clf();
    drawlater();
    plot2d (%inf ,%inf , frameflag=3, rect=[-2,-2,2,2], axesflag=0)
    xtitle('Animation')
    theta = i*2*%pi/N;
    theta2 = i*20*%pi/N;
    c = [cos(theta)+r2*cos(theta2) sin(theta)+r2*sin(theta2)];
    xfarcs([c(1)-r1 c(2)+r1 2*r1 2*r1 0 360*64]',5);
    drawnow();
end
```

3.8 Verzeichnis der Grafikbefehle

Ein nützliches Verzeichnis, wie es in der Onlihne-Hilfe von Scilab angeboten wird:

- 2d plotting

`plot2d`	:	plot a curve
`plot2d2`	:	plot a curve as step function
`plot2d3`	:	plot a curve with vertical bars
`plot2d4`	:	plot a curve with arrows
`fplot2d`	:	plot a curve defined by a function
`champ`	:	2D vector field
`champ1`	:	2D vector field with colored arrows
`fchamp`	:	direction field of a 2D first order ODE
`contour2d`	:	level curves of a surface on a 2D plot
`fcontour2d`	:	level curves of a surface defined by a function on a 2D plot
`grayplot`	:	2D plot of a surface using colors
`fgrayplot`	:	2D plot of a surface defined by a function using colors
`Sgrayplot`	:	smooth 2D plot of a surface using colors
`Sfgrayplot`	:	smooth 2D plot of a surface defined by a function using colors
`xgrid`	:	add a grid on a 2D plot
`errbar`	:	add vertical error bars on a 2D plot
`histplot`	:	plot a histogram
`Matplot`	:	2D plot of a matrix using colors

- 3d plotting

`plot3d`	:	plot a surface
`plot3d1`	:	plot a surface with gray or color level
`fplot3d`	:	plot a surface defined by a function
`fplot3d1`	:	plot a surface defined by a function with gray or color level
`param3d`	:	plot one curve
`param3d1`	:	plots curves
`contour`	:	level curves on a 3D surface
`fcontour`	:	level curves on a 3D surface defined by a function
`hist3d`	:	3D representation of a histogram
`genfac3d`	:	compute facets of a 3D surface
`eval3dp`	:	compute facets of a 3D surface
`geom3d`	:	projection from 3D on 2D after a 3D plot

- Line and polygon plotting

`xpoly`	:	draw a polyline or a polygon
`xpolys`	:	draw a set of polylines or polygons
`xrpoly`	:	draw a regular polygon
`xsegs`	:	draw unconnected segments
`xfpoly`	:	fill a polygon
`xfpolys`	:	fill a set of polygons

- Rectangle plotting

`xrect`	:	draw a rectangle
`xfrect`	:	fill a rectangle
`xrects`	:	draw or fill a set of rectangles

- Arc plotting

`xarc`	:	draw a part of an ellipse
`xarcs`	:	draw parts of a set of ellipses
`xfarc`	:	fill a part of an ellipse
`xfarcs`	:	fill parts of a set of ellipses

- Arrow plotting

`xarrows`	:	draw a set of arrows

- Strings

`xstring`	:	draw strings
`xstringl`	:	compute a box which surrounds strings
`xstringb`	:	draw strings into a box
`xtitle`	:	add titles on a graphics window
`titlepage`	:	add a title in the middle of a graphics window
`xinfo`	:	draw an info string in the message subwindow

- Frames and axes

`xaxis`	:	draw an axis
`graduate`	:	pretty axis graduations
`plotframe`	:	plot a frame with scaling and grids

- Coordinates transformations

`isoview`	:	set scales for isometric plot (do not change the size of the window)
`square`	:	set scales for isometric plot (change the size of the window)
`scaling`	:	affine transformation of a set of points
`rotate`	:	rotation of a set of points
`xsetech`	:	set the sub - window of a graphics window for plotting
`subplot`	:	divide a graphics window into a matrix of sub-windows
`xgetech`	:	get the current graphics scale
`xchange`	:	transform real to pixel coordinates

- Colors

`colormap`	:	using colormaps
`getcolor`	:	dialog to select colors in the current colormap
`addcolor`	:	add new colors to the current colormap
`graycolormap`	:	linear gray colormap
`hotcolormap`	:	red to yellow colormap

- Graphics context

`xset`	:	set values of the graphics context
`xget`	:	get current values of the graphics context
`xlfont`	:	load a font in the graphics context or query loaded font
`getsymbol`	:	dialog to select a symbol and its size

- Save and load

`xsave`	:	save graphics into a file
`xload`	:	load a saved graphics
`xbasimp`	:	send graphics to a Postscript printer or in a file
`xs2fig`	:	send graphics to a file in Xfig syntax
`xs2gif`	:	send graphics to a file in Gif syntax
`xs2png`	:	send graphics to a file in PNG syntax
`xs2ppm`	:	send graphics to a file in PPM syntax

- Graphics primitives

`xbasc`	:	clear a graphics window and erase the associated recorded graphics
`xclear`	:	clear a graphics window
`driver`	:	select a graphics driver
`xinit`	:	initialisation of a graphics driver
`xend`	:	close a graphics session
`xbasr`	:	redraw a graphics window
`replot`	:	redraw the current graphics window with new boundaries
`xpause`	:	suspend Scilab
`xselect`	:	raise the current graphics window
`xclea`	:	erase a rectangle
`xclip`	:	set a clipping zone
`xdel`	:	delete a graphics window
`winsid`	:	return the list of graphics windows
`xname`	:	change the name of the current graphics window

- Mouse position

`xclick`	:	wait for a mouse click
`locate`	:	mouse selection of a set of points
`xgetmouse`	:	get the current position of the mouse

- Interactive editor

`edit_curv`	:	interactive graphics curve editor
`gr_menu`	:	simple interactives graphic editor
`sd2sci`	:	gr_menu structure to scilab instruction convertor

4 Anwendungsfelder: Dynamische Systeme

4.1 Definition linearer dynamischer Systeme

Der Befehl zur Definition linearer dynamischer Systeme heißt `syslin`. Er kann sowohl für die Definition von Übertragungsfunktionen als auch für Darstellungen im Zustandsraum verwendet werden.

4.1.1 Die Übertragungsfunktion

Vom mathematischen Standpunkt ist eine Übertragungsfunktion eine gebrochen rationale Funktion und deshalb überrascht es nicht, dass ihre Definition derjenigen ähnelt, die in Abschnitt 2.10 gezeigt wurde und dass ihr Typ ebenfalls `rational` ist. Ein Beispsiel:

```
-->s=%s;

-->num = 1+s;

-->den = (s+2)*(s+3);

-->Stf = syslin('c',num,den)
 Stf  =

      1 + s
    ---------
              2
    6 + 5s + s

-->typeof(Stf)
 ans  =

 rational
```

Die Option `'c'` des Befehls `syslin` bezeichnet die Definition eines zeitkontinuierlichen Systems, für eine zeitdiskrete Übertragungsfunktion hat man ganz analog:

```
-->z=%z;

-->num = 1+s;
```

```
-->Stfd = syslin('d',1,z-0.5)
 Stfd  =

       1
    -------
    - 0.5 + z

-->typeof(Stfd)
 ans  =

 rational
```

4.1.2 Die Darstellung im Zustandsraum

Die Definition eines linearen dynamischen Systems in der Form des Zustandsraums verläuft immer über den Befehl `syslin`, aber mit einer anderen Syntax: `syslin('c',A,B,C,D)`. So erhält man z.B. das System

$$\begin{cases} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{cases}$$

mit

$$A = \begin{bmatrix} -5 & -1 \\ 6 & 0 \end{bmatrix}$$
$$B = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$
$$C = \begin{bmatrix} -1 & 0 \end{bmatrix}$$
$$D = 0$$

mit den Anfangsbedingungen $x(0) = [0 \quad 0]^T$ mit dem Code

```
-->A = [-5 -1
-->      6  0];
-->B = [-1; 1];
-->C = [-1 0];
-->D = 0;

--> Sss = syslin('c',A,B,C,D)
 Sss  =
```

```
       Sss(1)   (state-space system:)

!lss  A  B  C  D  X0  dt  !

       Sss(2) = A matrix =
  - 5.  - 1.
    6.    0.

       Sss(3) = B matrix =
  - 1.
    1.

       Sss(4) = C matrix =
-   1.    0.

       Sss(5) = D matrix =
    0.

       Sss (6) = X0 (initial state) =
    0.
    0.

       Sss(7) = Time domain =
 c

-->typeof(Sss)
 ans  =

 state-space
```

wobei es unnötig ist, die Anfangsbedingungen zu spezifizieren, da sie voreingestellt null sind.

4.1.3 Umformungen zwischen Zustandsraum und Übertragungsfunktion

Es ist möglich, ein dynamisches System aus seiner Form im Zustandsraum in seine rationale Gestalt umzuformen - und umgekehrt. Die Umformung ist nicht schmerzlos, weshalb dem Leser zur Vertiefung ein Text über dynamische Systeme empfohlen sei. In Scilab sind die entsprechenden Befehle `tf2ss` (transfer function to state space) und `ss2tf` (state space to transfer function). Die beiden vorstehenden Beispiele sind dynamische Systeme mit dem selben Übertragungsverhalten zwischen Eingang und Ausgang, man würde also erwarten, dass beide Umformungen die gleichen Koeffizienten ergeben, die in der Definition verwendet wurden. Das ist offensichtlich nicht der Fall:

```
-->ss2tf(Sss)
 ans  =
```

```
      1 + s
    ---------
              2
    6 + 5s + s

-->tf2ss (Stf)
 ans  =

       ans(1)   (state-space system:)
!lss  A  B  C  D  X0  dt !

       ans(2) = A matrix =
  - 5.  - 1.
    6.  - 4.441D -16

       ans(3) = B matrix =
  - 1.
    1.

       ans(4) = C matrix =
  - 1.    1.110D-16

       ans(5) = D matrix =
    0.

       ans(6)=X0 (initial state) =
    0.
    0.

       ans(7) = Time domain =
 c
```

Tatsächlich bemerkt man zwei Zahlen, deren Wert man zu null erwartet hätte und die jetzt in der Größenordnung von 10^{-16} auftreten. Es ist klar, dass es sich um *sehr kleine* Werte handelt, die praktisch null sind. Aber bevor Sie die Werte nehmen, wie sie sind, lesen Sie den Abschnitt 2.3 über die numerische Genauigkeit.

4.1.4 Minimierte Darstellungen

Der Befehl `minreal` erlaubt die Berechnung einer minimierten Darstellung eines linearen Systems in der Darstellung im Zustandsraum. Derselbe Befehl kann aber auch für Übertragungsfunktionen verwendet werden und bewirkt das Kürzen von Polen und Nullstellen. In folgendem Beispiel hat die Übeertragungsfunktion $F1$ bei $s = 2$ einen Pol und eine Nullstelle. Scilab kürzt den Ausdruck, nachdem es die Übertragungsfunktion in eine Darsteluung im Zustandsraum umgeformt hat (die Ausgabe von `minreal` ist auch im Zustandsraum).

```
-->s=%s;
-->num=(s-2)*(s-3);den=(s-2)*(s -4)*(s -5);
-->F1= syslin('c',num,den)
 F1  =
                    2
          6 - 5s + s
     ------------------
                    2   3
   - 40 + 38s - 11s + s

-->F2= minreal(tf2ss(F1));
--> ss2tf(F2)
 ans  =
   - 3 + s
    -----------
               2
    20 - 9s + s
```

Der Befehl für die minimale Realisierung ist `minss`.

4.1.5 Extraktion von Informationen aus einer Variablen, die ein System repräsentiert

Unser lineares System sei als Übertragungsfunktion definiert, während aus der Übergangsform im Zustandsraum Informationen über Zähler, Nenner, Pole und Nullstellen der Übertragungsfunktion erhalten werden können, wie das für rationale Funktionen gezeigt wurde.

```
-->poli = roots(Stf.den)

 poli  =
  - 2.
  - 3.
```

Ist das System mittels einer Variablen des Typs `state-space` definiert, kann auf seine Matrizen $A, B.C.D$ und einen evtl. Anfangszustand zugegriffen werden:

```
-->A = Sss.A
 A  =

  - 5.  - 1.
    6.    0.

-->typeof(A)
 ans  =

 constant
```

Man beachte, dass der Typ der Variablen A `constant` ist und daher mit A die entsprechenden Matrixoperationen vorgenommen werden können.

Man kann mit dem Befehl `abcd` auch sämtliche Matrizen aus der Übergangsfunktion eines dynamischen Systems bekommen:

```
-->[A,B,C,D]=abcd(Sss)
 D  =
    0.

 C  =
  - 1.   0.

 B  =
  - 1.
    1.

 A  =
  - 5.  - 1.
    6.    0.
```

4.1.6 Visualisierung von Systemen im Zustandsraum

Zur Visualisierung von Systemen im Zustandsraum gibt es einen Befehl, der im Vergleich zu den vorstehenden Beispielen eine besser lesbare Ausgabe produziert.

```
--> ssprint(Sss)

    |-5 -1 |    |-1 |
x = | 6  0 |x + | 1 |u

y = |-1  0 |x
```

oder zeitdiskret

```
-->ssprint(tf2ss(Stfd))

+
x = | 0.5 |x + | 1 |u

y = | 1 |x
```

4.2 Die Tabelle von Routh-Hurwitz

4.2 Der Befehl `routh_t` ermöglicht die Berechnung der Tabelle von Routh-Hurwitz, die zur Beurteilung der Stabilität eines dynamischen Systems dient. Wegen der Einzelheiten ziehe man die Fachliteratur hinzu.

Das folgende Beispiel zeigt die Anwendung der Tabele auf ein Polynom, genauer, den Nenner der Übertragungsfunktion eines linearen Systems.

```
-->s=%s;

-->d = (s+3)*(s-1)
 d  =

              2
  - 3 + 2s + s

-->routh_t(d)
 ans  =

    1.  - 3.
    2.    0.
  - 3.    0.
```

Auch kann man diesen Befehl zur Berechnung der Tabelle eines geschlossenen Systems mit Rückführung benutzen. Dazu verwendet man den Parameter k als Freiheitsgrad. Der Eingang muss nun eine Variable des Typs `rational` sein, also eine Übertragungsfunktion. Mit dem bereits definierten Nenner ist ein Anwendungsbeispiel:

```
-->F = 1/d;

-->k=poly(0, 'k')
 k  =

    k

-->routh_t(F,k)
 ans  =
    1         - 3 + k

    2           0
  - 6 + 2k      0
```

woraus man erkennt, dass mit $k = 3$ das geschlossene System mit Rückführung als asymptotisch bzw. als stabil angesehen werden kann.

4.3 Grafische Darstellungen in der komplexen Ebene

4.3.1 Karte der Nullstellen und Pole

Oft ist eine grafische Darstellung, der Pole und Nullstellen eines dynamischen Systems von Nutzen. Der Befehl `plzr` akzeptiert als Eingabe sowohl Variablen des Typs `rational` als auch Zeitfunktionen.

Gibt man dann noch den Befehl `sgrid` ein, werden die geometrischen Örter gezeichnet, aus denen man die Zeitkonstantenten und die Dämpungskonstanten ablesen kannn. Das Ergebnis des Codes

```
-->plzr(Sss)

-->sgrid
```

zeigt Abbildung 4.1.

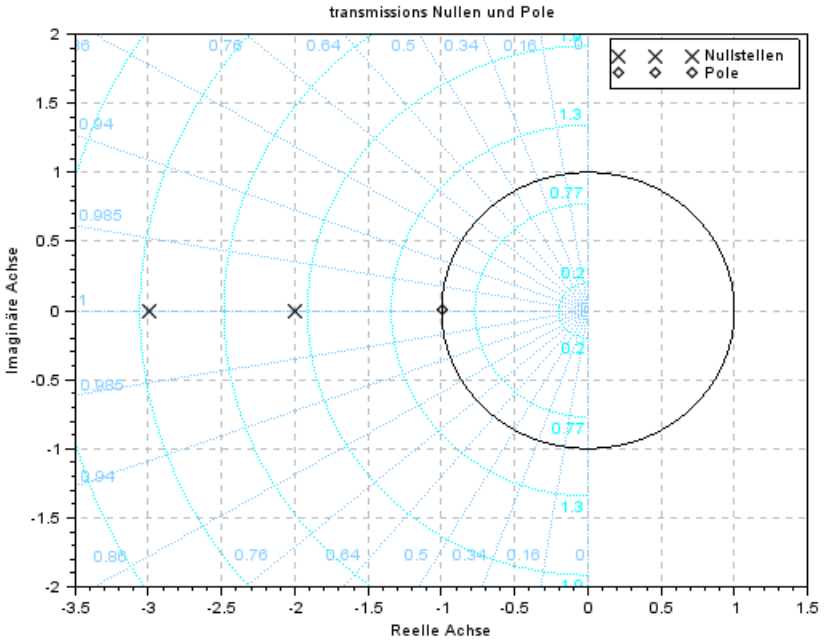

Abb. 4.1: Beispiel für die Anwendung von `plzr`

Beachten Sie, dass im zeitkontinuierlichen Fall bei fehlendem Befehl `sgrid` auch die Einheitsumgebung gezeichnet wird, was bar jeden Nutzens ist.

Im zeitdiskreten Fall dient derselbe Befehl zum Zeichnen der Karte, doch muss nun der Befehl `zgrid` für das Gitternetz genommen werden, was mit dem Code

```
-->plzr(Stfd)

-->zgrid
```

die Abbildung 4.2 ergibt, worin sich der einzige Pol bei $x = 0.5$ zeigt.

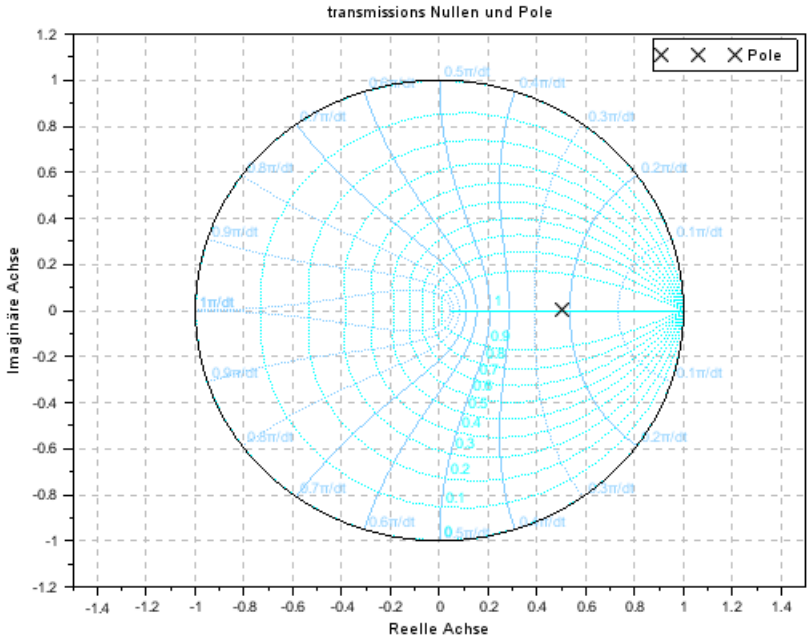

Abb. 4.2: Beispiel für die Anwendung von `plzr` im zeitdiskreten Fall

4.3.2 Der Wurzelort

Der Wurzelort ist der geometrische Ort der Punkte, für welche

$$1 + kF(s) = 0$$

gilt mit $F(s)$ als Übertragungsfunktion des geschlossenen Kreises. Man kann ihn mit dem Befehl `evans` grafisch darstellen, wie das folgendem Beispiel

$$F(s) = \frac{s + 10}{(s + 2)(s + 3)}$$

mit der entsprechenden Abbildung 4.3 zeigt:

```
s=%s;
num = 10+s;
den = (s+2)*(s +3);
Stf = syslin ('c',num ,den)
evans (Stf)
```

Die Festlegung der Achsen ist für die Visualisierung des Wurzelortes nicht immer optimal, weshalb es nötig werden kann, mit dem *zoom*-Werkzeug im Menu `Zusatzprogramme` und `Vergrößern` den interessierenden Ausschnitt einzugrenzen.

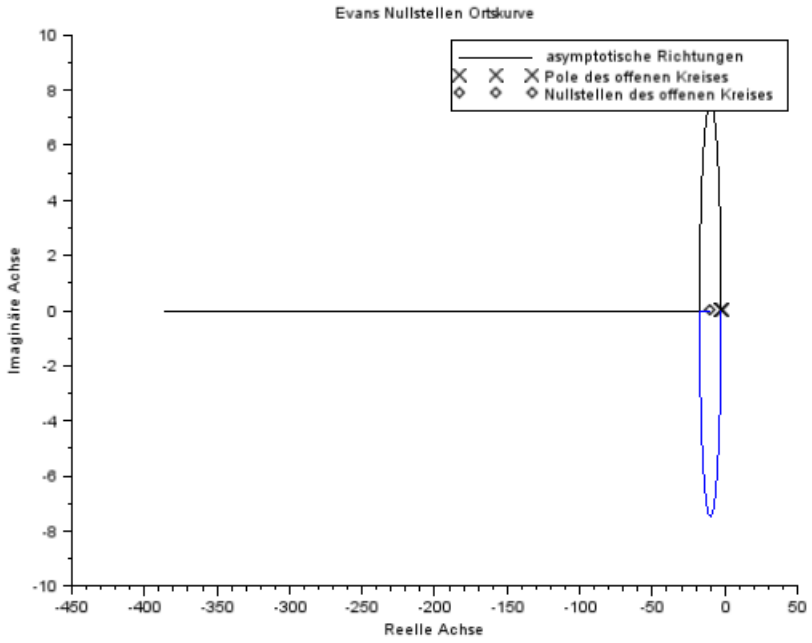

Abb. 4.3: Wurzelort einer Übertragungsfunktion mit zwei Polen und einer Nullstelle.

Verstehen Sie auch, das die Beschriftungen interessierende Teile der Grafik verdecken können. Im vorliegenden Fall hätte wahrscheinlich eine Festlegung der Achsen wie in Abbildung 4.4 zweckmäßig sein können.

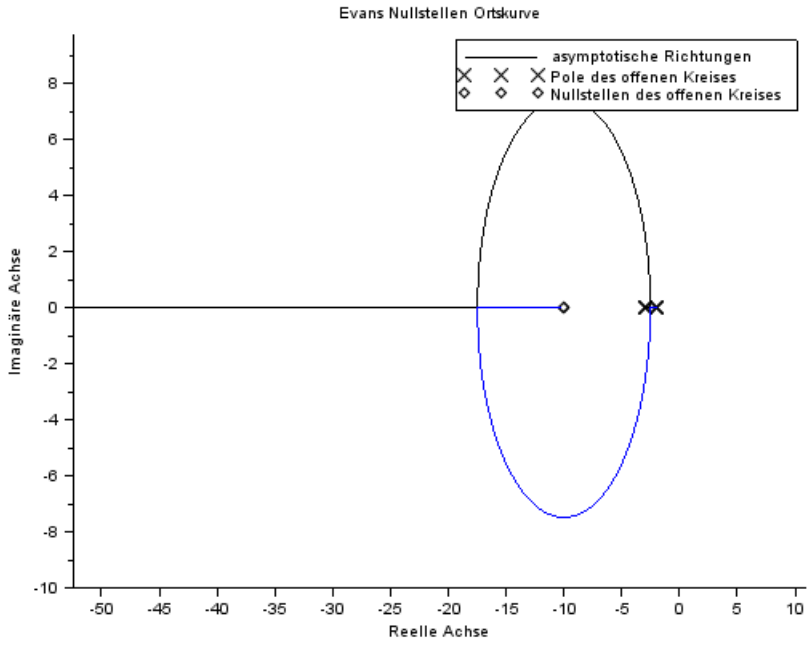

Abb. 4.4: Wurzelort einer Übertragungsfunktion mit zwei Polen und einer Nullstelle im Ausschnitt.

Ein anderes Problem mit dem Befehl `evans` besteht in der für die Zeichnung gewählten Diskretisierung. In der Online-Hilfe wird der Befehl als

`\texttt{evans(H[,kmax]}`

also mit dem fakultativen Parameter `kmax` gezeigt. Der voreingestellte Wert ist nicht immer zufriedenstellend. Man könnte dem abhelfen durch Eingabe des größten k, dafür müsste man jedoch *apriori* wissen, dass die Grafik falsch wird...

Die Fallstricke dieses Befehls können anhand der vier Grafiken in Abbildung 4.5 eingeschätzt werden, in denen die Wurzelorte der Übertragungsfunktionen

$$P1 = \frac{s+1}{s(s+0.1)} \quad P2 = \frac{s+10}{s(s+1)} \quad P3 = \frac{s+100}{s(s+10)} \quad P4 = \frac{s+1000}{s(s+100)}$$

gezeigt werden.

Aus der Theorie ist bekannt, dass sie *qualitativ* nicht unterscheidbar sein sollten, während der Befehl untereinander sehr verschiedene Ergebnisse zeitigt.

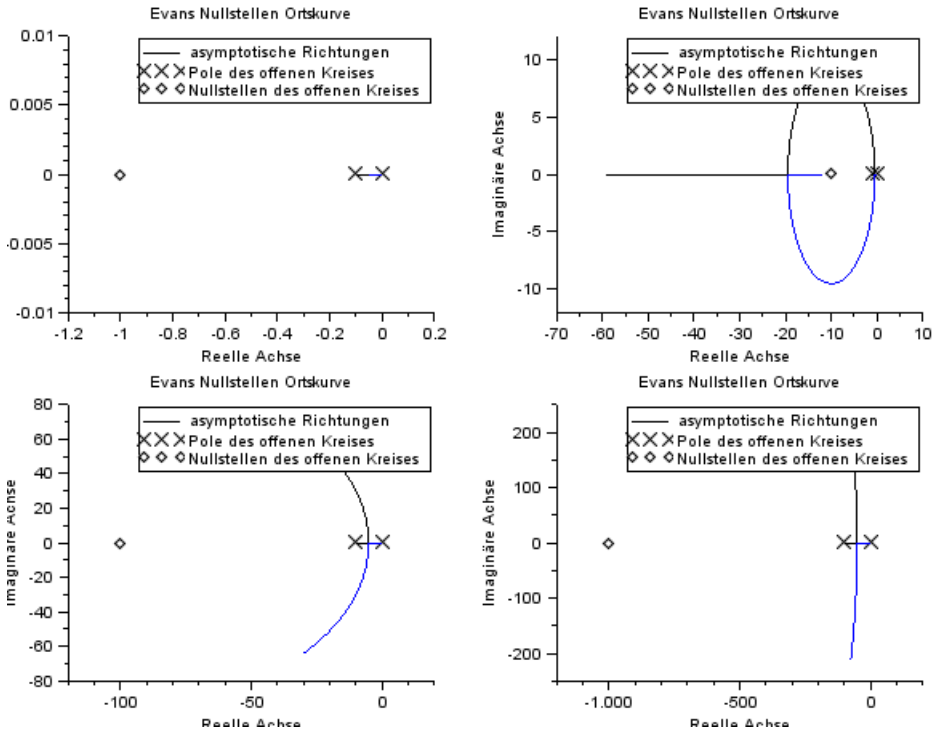

Abb. 4.5: Wurzelorte von 4 Übertragungsfunktionen, die qualitativ gleich aussehen sollten

Die Darszellung des Wurzelortes in z ist formal identisch mit dem Fall in s, also benutzt man dafür dieselben Befehle.

4.3.3 Synthese mittels Wurzelortverfahren

Der Befehl `evans` bietet vermittels des Wurzelortes ein numerisches Werkzeug zum Abgleich eines Reglers. Ist nämlich die Wurzelortkurve einmal gezeichnet, muss man wissen, für welchen Wert von k sich die Pole beim geschlossenen Kreis in einem bestimmten Punkt treffen.

Es gibt dafür zwei mögliche grafische Verfahren. Das eine benutzt den Befehl `horner`, das andere das in Abschnitt 3.2.11 diskutierte Werkzeug `datatip`.

Ohne in die algorithmischen Einzelheiten zu gehen, funktioniert das erste Verfahren mit der Anweisung

```
-->k=-1/real(horner(Stf,[1,%i]*locate(1)))
```

die das Anklicken des gewünschten Punktes (mit dem Befehl `locate`) erlaubt und ihn dann zur Auflösung der Gleichung $1 + kF(s) = 0$ nach k verwendet. Man muss nicht genau auf die Kurve klicken, aber es ist hilfreich, vorher in das interessierende Gebiet hineinzuzoomen.

Auch für die Synthese ist der Fall z mit dem Fall s formal identisch, weshalb die gleichen Befehle Verwendung finden.

4.4 Der Frequenzgang

4.4.1 Das Nichols-Diagramm

Das Nichols-Diagramm stellt die harmonische Antwortfunktion $F(j\omega)$ in der Ebene dar, in der auf der Abszisse die Phase, üblicherweise in Grad aufgetragen ist und auf der Ordinate das Argument, üblicherweise in dezibel[1] (dB).

Der Befehl für die Darstellung des Nichols-Diagramms ist `black`. Für die Überlagerung mit konstantem Argument und Phase kann man den Befehl `chart` benutzen. Für sich allein produziert er ein Nichols-Diagramm, in dem der Ursprung der Achsen im Punkt $(-90°, 0)$ liegt und in dem die konstanten Örter von Argument und Phase dargestellt werden. Die Abbildung 4.6 zeigt ein Beispiel.

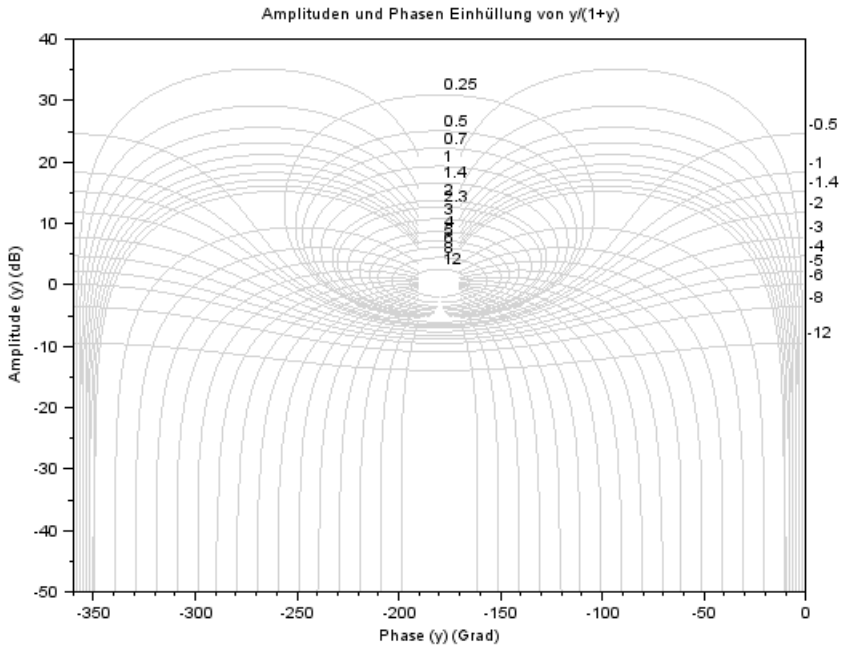

Abb. 4.6: Grafik mit dem Befehl `chart()`

Mit dem Befehl `chart` kann man auch nur wenige interessierende Kurven zeichnen oder sie von den anderen unterscheiden. Wegen der Einzelheiten siehe die Online-Hilfe. Der folgende Code zeigt das Nichols-Diagramm der harmonischen Antwortfunktion

$$F(j\omega) = \frac{10 + j\omega}{j\omega(j\omega + 3)}$$

wobei die Kurve in rot gezeichnet werden soll, die dem Punkt mit dem konstanten Argument von -3 dB entspricht. Dazu gehört die Abiilding 4.7.

[1] $X_{dB} = 20 \log_{10}(X)$

```
s=%s;
num = 10+s;
den = s*(s+3);
Stf = syslin('c',num,den);
chart();
black(Stf)
chart(-3,[],list(1 ,0 ,5))
xgrid
```

Auch hier kann mit der zoom-Funktion der gewünschte Ausschnitt erzeugt werden. Zu beachten ist, das die Kurve in Hz parametrisiert ist. Für die kontinuierliche Zeit auf der Kurve sind einige Werte aus dem Intervall $\left[10^{-3}, 10^{3}\right]$ Hz aufgeführt.

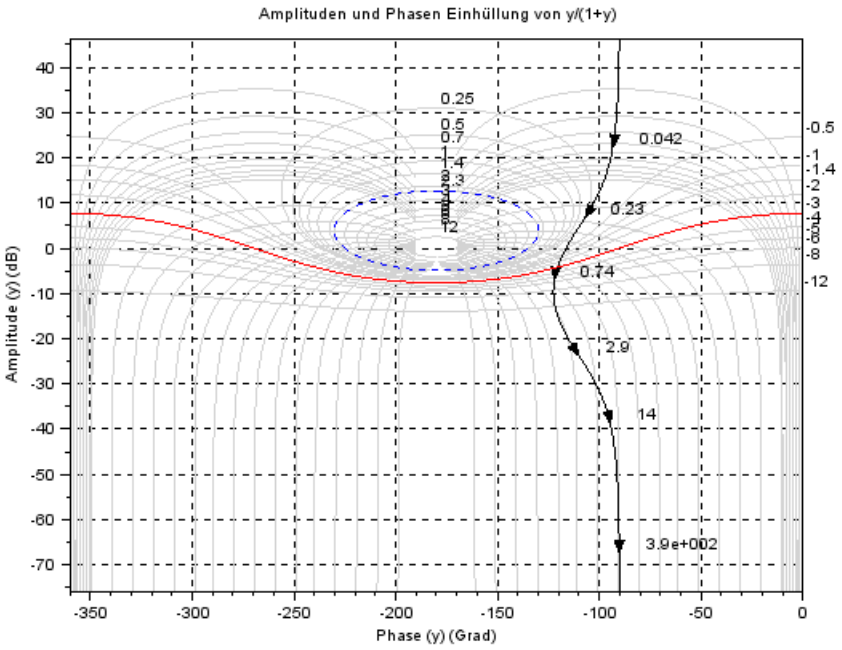

Abb. 4.7: Beispiel eines Nichols-Diagramms

Für diskrete Zeit berechnet man den Frequenzgang mit

$$z = e^{j\omega}$$

mit $\theta \in \ ]-\pi, \pi]$ rad. Wegen der Symmetrie genügt es oft, die Kurve nur für $\theta \in \ [0, \pi]$ rad zu zeichnen. Befehl erwartet als Eingabe eine Variable des Typs

$$z = e^{2\pi j\omega T}$$

also eine Syntax für Systeme mit abgetasteten Daten. Die Eingabe des Befehls ist ω und wird daher in Hz angegeben. Das voreingestellte Intervall ist $\omega \in \ \left[10^{-3}, 0.5\right]$Hz. Für zeitdiskrete Systeme, die nicht von einer Abtastung erzeugt werden, ist die Variable $T = 1$ und um eine Grafik bis $\theta = \pi$ zu bekommen, muss man genau $\omega = 0.5$ machen.

4.4.2 Das Bode-Diagramm

Das Bode-Diagramm erhält man mit dem Befehl `bode` wie im folgenden Code-Beispiel:

```
s=%s;
num = 20*(10+s);
den = (s+5)*(s+3);
Stf = syslin('c',num ,den);
bode(Stf)
```

das die Funktion

$$F(j\omega) = \frac{20(10 + j\omega)}{(j\omega + 3)(j\omega + 5)}$$

zeigt. Es ist in Abbildung 4.8 visualisiert.

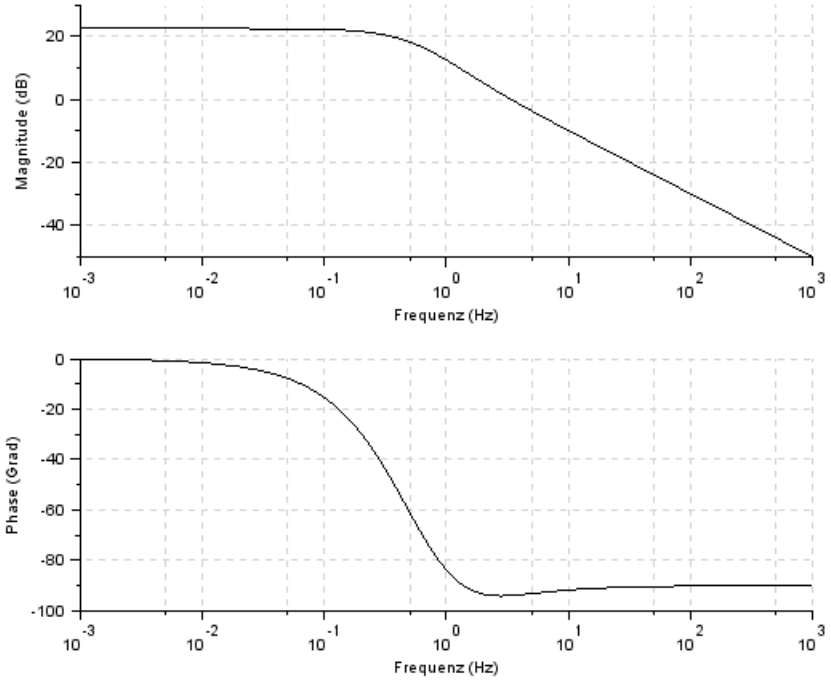

Abb. 4.8: Beispiel eines Bode-Diagramms

Man beachte, dass in Scilab die Abszisse in Hz eingeteilt ist, während andere Programme, z.B. Matlab hier nur rad/sec verwenden. Das bedeutet, dass z.B. ein abgelesener Wert von 4 Hz einem Wert von $4 \times 2\pi \approx 25 rad/s$ entspricht und umgekehrt, wenn man den Wert z.B. für $10 rad/s$ sucht, muss man an der Abszisse $10/2\pi \approx 1.6 Hz$ ablesen.

Ein Diagramm nur für die Argumente erhält man mit dem Befehl `gainplot`. Der folgende Code zeichnet einen Fall, in dem die in Scilab voreingestellten Frequenzen für die Darstellung der Funktion nicht opteimal sind. Nehmen wir z.B. die Funktion

$$F(j\omega) = \frac{10^6 + j\omega}{10^5 + j\omega}$$

für die der Code

```
s=%s;
num = (s+1e6);
den = (s+1e5);
Stf = syslin('c',num,den);
gainplot(Stf)
```

die wenig aussagefähige Abbildung 4.9 ergibt.

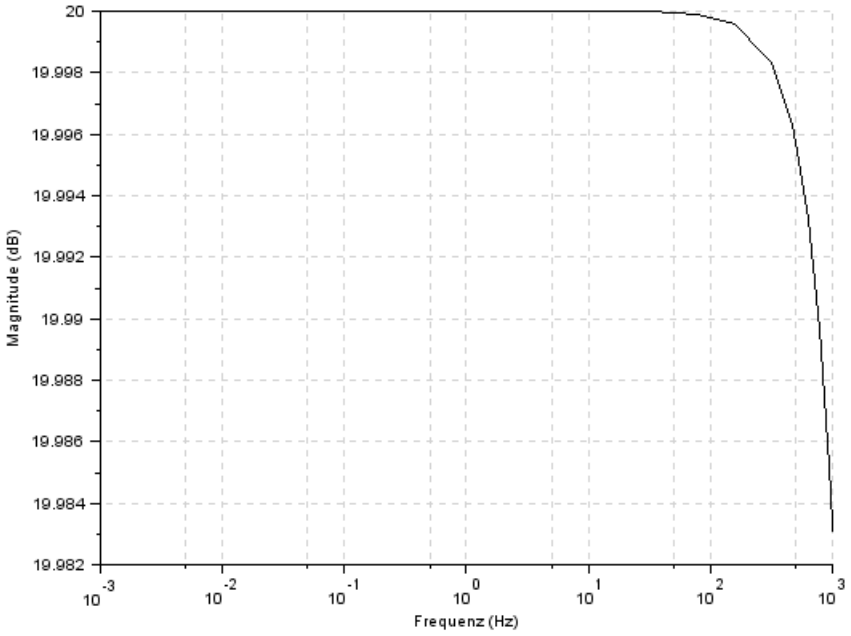

Abb. 4.9: Beispiel eines Bode-Diagramms (nur Argumente), worin der voreingestellte Definitionsbereich unbefriedigend ist.

Durch Festlegung eines geeigneten Intervalls auch aufgrund der Kenntnis dieses Ergebnisses erhält man die aussgefähige Abbildung 4.10. Der Code dafür ist

```
s=%s;
num = (s+1e6);
den = (s+1e5);
```

```
Stf = syslin('c',num,den);
f=logspace(-1, 8, 50);
gainplot(Stf,f)
```

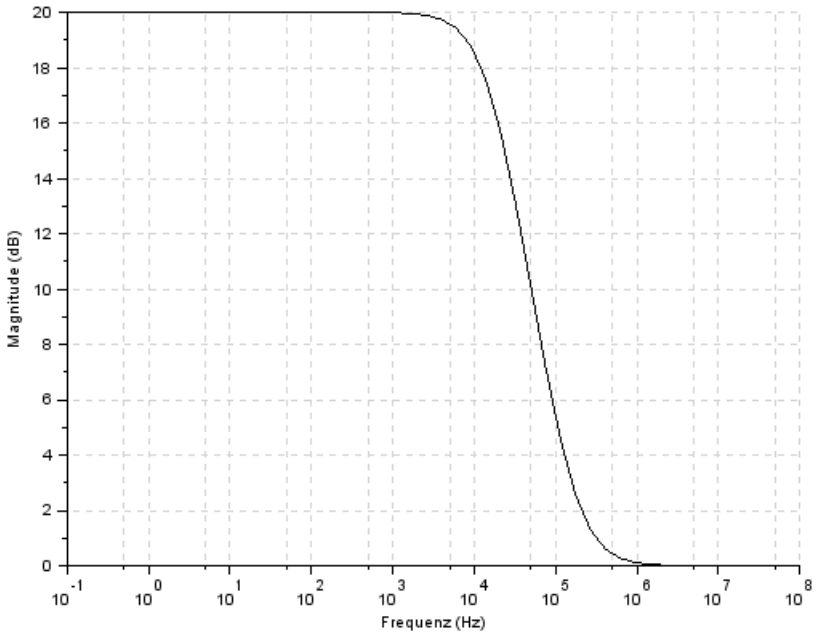

Abb. 4.10: Dieselbe Funktion wie in Abb. 4.9 mit `gainplot` und einem geeigneten Definitionwsbereich dargestellt

4.4.3 Das Nyquist-Diagramm

Das Nyquist-Diagramm und die Punkte mit konstantem Argument erhält man mit Hilfe der Befehle `nyquist` und `m_circle`, deren Gebrauch mit den für das Nichols-Diagramm verwendeten Befehlen völlig übereinstimmt. Daher ist auch hier die Kurve in Hz parametrisiert.

Beachten Sie, dass die Grafik für die Frequenz auch hier mit einem voreingestellten Definitionsbereich arbeitet, der sich für eine adäquate Darstellung der jeweiligen Funktion als ungeeignet erweisen kann. Bei zeitkontinuierlichen Funktionen mit Polen im Ursprung oder für zeitdiskrete Funktionen mit Polen in 1 kann es zu schlechter Lesbarkeit kommen, was den Einsatz der *zoom*-Funktion erforderlich macht.

4.4.4 Berechnung von Modul und Phase in speziellen Punkten

Der Befehl `horner` ermöglicht die Berechnung des Wertes einer rationalen Funktion an einer oder mehreren Stellen. Er kann zur Berechnung des Wertes einer harmonischen Antwortfunktion dienen, um sie dann in Argument und Phase, ausgedrückt in dB und Grad, zu konvertieren.

Im folgenden Beispiel wird die harmonische Antwortfunktion

$$F(j\omega) = \frac{1}{(j\omega + 3)}$$

berchnet für $\omega = 3$ rad/s::

```
-->s=%s;

-->num = 3;

-->den = (s+3);

-->Stf = syslin('c',num,den);

-->out = horner(Stf,3*%i)
 out  =

    0.5 - 0.5i

-->[phi, db]=phasemag(out)
 db  =

  - 3.0103
 phi  =

  - 45.
```

und das Resultat ist wie bekannt, dass an der fraglichen Stelle das Argument -3 dB ist mit einer Phase von -45 Grad.

Es existieren verschiedene Befehle um numerische Informationen über die Frequenzantwort einer Übertragungsfunktion in einem Intervall von Punkten zu bekommen, siehe insbesondere `repfreq` und `dbphi`

Es ist ratsam, sich stets zu vergewissern, in welcher Maßeinheit die Eingaben zu erfolgen haben, in Hz oder in rad/s.

4.4.5 Berechnung der Stabilitätsgrenzen

Die für die Berechnung der Stabilitätsgrenzen zu verwendenden Befehle sind `g_margin` und `p_margin` für die Grenze der Verstärkung bzw. die Grenze der Phase.

4.5 Umwandlung zeitkontinuierlich - zeitdiskret

Man kann mit dem Befehl `cls2dls` ein zeitkontinuierliches System in das entsprechende zeitdiskrete umwandeln. Dieser Befehl implementiert die bilineare Transformation

$$s = \frac{2}{T} \cdot \frac{z-1}{z+1}$$

mit T als Abtastschritt. Der Befehl akzeptiert als Eingabe nur Systeme im Zustandsraum und liefert auch ein System im Zustandsraum zurück.

```
Sd = cls2dls(Sc, T, [,fp])
```

worin Sc das umzuwandelnde System, T der Abtastschritt und das optionale fp die prewarp-Frequenz.

In dem häufigen Fall, dass mit Zeitfunktionen und Übertragungsfunktionen gearbeitet wird, ist es erforderlich, eine unbequeme doppelte Konversion vorzusehen: dafür die Syntax

```
Fd = ss2tf(cls2dls(tf2ss(Fc),T))
```

Man beachte, dass man die Konversion auch mit dem in Abschnitt 2.9.6 beschriebenen Befehl `horner` vornehmen kann. Der Befehl `horner` ermöglicht auch die Transformationen durch Integration vorwärts und rückwärts.

4.6 Simulation eines einfachen dynamischen Systems

Für die Simulation komplexer dynamischer Systeme ist es Verwendung von Xcos zweckmäßig, wenn nicht sogar unverzichtbar. Für die Visualisierung zumindest der Standardantworten eines einfachen dynamischen Systems, ist es möglich, die entsprechen Befehle unter Scilab zu verwenden.

Für den zeitkontinuierlichen Fall ist der Befehl `csim` zu verwenden:

```
[y [,x]]=csim(u,t,sl,[x0 [,toll]])
```

worin

u	:	function, list or string (control)
t	:	real vector specifying times with, t(1) is the initial time (x0=x(t(1))).
sl	:	list (syslin)
y	:	a matrix such that y=[y(t(i)], i=1,..,n
x	:	a matrix such that x=[x(t(i)], i=1,..,n
tol	:	a 2 vector [atol rtol] defining absolute and relative tolerances for `ode` solver (see `ode`)

Man beachte, dass das dynamische System im Zeitbereich formuliert sein muss (und man kann dann auch den Fortgang des Zustands visualisieren). Die Eingabe u kann eine Funktion sein, ein Vektor, der die Eingabe in den entsprechenden Punkten repräsentiert oder eine Zeichenkette mit den Werten `'impulse'` oder `'step'` zur Erzeugung der Standardeingänge.

Der entsprechende zeitdiskrete Befehl ist `dsimul`, merkwürdigerweise ist die Syntax dieses Befehls verschieden vom entsprechenden zeitkontinuierlichen Befehl:

```
y=dsimul(sl, u)
```

worin *sl* wieder ein dynamisches System im Zustandsraum ist und *u* der Eingang. Hier ist die einzige zulässige Eingabe ein Vektor passender Dimension um außer einer allgemeinen Eingabe auch die Standardeingaben zu bilden. Dieser Befehl liefert nur den Ausgang. Um den Fortgang des Zustands zu erhalten muss ein anderer Befehl benutzt werden:

```
[X]=ltitr(A,B,U,[x0])
[xf,X]=ltitr((A,B,U,[x0])
```

4.7 Die Differentialgleichungen

Die Programme zur numerischen Berechnung dynamischer Systeme sind mit einer Bibliothek mit Funktionen zur Lösung von Differntialgleichungen ausgestattet. In Scilab ist `ode` ein solcher Befehl. Wegen Einzelheiten befrage man die Online-Hilfe oder die Fachliteratur. Zur Simulation einfacher dynamischet Systeme wird die Verwendung der sehr einfachen hier soeben vorgestellten Befehle vorgeschlagen oder Xcos, das in Kapitel 5 beschrieben wird.

4.8 Befehlsübersicht für dynamische Systeme

- Befehle für dynamische Systeme und ihre Regelung

`abcd`	- state - space matrices
`abinv`	- AB invariant subspace
`arhnk`	- Hankel norm approximant
`arl2`	- SISO model realization by L2 transfer approximation
`balreal`	- balanced realization
`bilin`	- general bilinear transform
`cainv`	- Dual of abinv
`calfrq`	- frequency response discretization
`canon`	- canonical controllable form
`cls2dls`	- bilinear transform
`colregul`	- removing poles and zeros at infinity
`cont_frm`	- transfer to controllable state - space
`cont_mat`	- controllability matrix
`contr`	- controllability , controllable subspace , staircase
`contrss`	- controllable part
`csim`	- simulation (time response) of linear system
`ctr_gram`	- controllability gramian
`dbphi`	- frequency response to phase and magnitude representation
`ddp`	- disturbance decoupling

`des2tf`	- descriptor to transfer function conversion
`dscr`	- discretization of linear system
`dsimul`	- state space discrete time simulation
`dt_ility`	- detectability test
`equil`	- balancing of pair of symmetric matrices
`equil1`	- balancing (nonnegative) pair of matrices
`feedback`	- feedback operation
`flts`	- time response (discrete time, sampled system)
`frep2tf`	- transfer function realization from frequency response
`freq`	- frequency response
`freson`	- peak frequencies
`g_margin`	- gain margin
`gfrancis`	- Francis equations for tracking
`imrep2ss`	- state - space realization of an impulse response
`invsyslin`	- system inversion
`kpure`	- continuous SISO system limit feedback gain
`krac2`	- continuous SISO system limit feedback gain
`lin`	- linearization
`linmeq`	- Sylvester and Lyapunov equations solver
`lqe`	- linear quadratic estimator (Kalman Filter)
`lqg`	- LQG compensator
`lqg2stan`	- LQG to standard problem
`lqr`	- LQ compensator (full state)
`ltitr`	- discrete time response (state space)
`markp2ss`	- Markov parameters to state - space
`minreal`	- minimal balanced realization
`minss`	- minimal realization
`obs_gram`	- observability gramian
`obscont`	- observer based controller
`observer`	- observer design
`obsv_mat`	- observability matrix
`obsvss`	- observable part
`p_margin`	- phase margin
`pfss`	- partial fraction decomposition
`phasemag`	- phase and magnitude computation
`ppol`	- pole placement
`projsl`	- linear system projection
`repfreq`	- frequency response
`ricc`	- Riccati equation
`rowregul`	- removing poles and zeros at infinity
`rtitr`	- discrete time response (transfer matrix)
`sm2des`	- system matrix to descriptor
`sm2ss`	- system matrix to state - space
`specfact`	- spectral factor
`ss2des`	- (polynomial) state - space to descriptor form

`ss2ss`	- state - space to state - space conversion, feedback, injection
`ss2tf`	- conversion from state - space to transfer function
`st_ility`	- stabilizability test
`stabil`	- stabilization
`svplot`	- singular - value sigma - plot
`sysfact`	- system factorization
`syssize`	- size of state - space system
`tf2ss`	- transfer to state - space
`time_id`	- SISO least square identification
`trzeros`	- transmission zeros and normal rank
`ui_observer`	- unknown input observer
`unobs`	- unobservable subspace
`zeropen`	- zero pencil

5 Xcos

5.1 Einführung

Xcos ist eine Scilab-Anwendung für die grafische Simulation dynamischer Systeme. Es enthält eine Sammlung von Blöcken, die Eingänge, Funktionen und Ausgänge für ein weites Anwendungsfeld der Simulation liefern und zwar sowohl für zeitdiskrete wie für zeitkontinuierliche Systeme. Es ist eine Weiterentwicklung von Scicos, das bis zur Version 5.1 von Scilab Verwendung fand und stellt eine Art Clone von Simulink dar, dem berühmten Paket von Matlab.

Zweckmäßigerweise stellt man sich Xcos als eine Art grafischer Programmiersprache vor, die wie andre Programmiersprachen auch eine Reihe von Befehlen bereitstellt, daneben aber dem Anwender auch ermöglicht, selbst angepasste Blöcke einzufügen, die eigentlich Funktionen sind. Insbesondere können in Xcos Blöcke zur Verwendung mit der Scilab-eigenen Sprache, mit C oder mit Fortran eingesetzt werden.

Eine weitere Fähigkeit von Scilab ist die automatische Erzeugung von C-Code. Ist ein Modell in Xcos erst einmal erstellt, kann seine Verwendung auch in eingebetteten Anwendungen vorgesehen werden. Es ist möglich, Xcos zur Realisierung aller Stadien der Projektierung und der Implementierung einer Regeleinrichtung einzusetzen, von der Simulation über Versuche in HIL (Hardware-In-the-Loop) bis zur Erstellung eines in Echtzeit ausführbaren Programms.

Das Aussehen eines einfachen Modells in Xcos zeigt die Abbildung 5.1.

5.1.1 Was bedeutet die Simulation eines zeitkontinuierlichen Systems durch ein digitales System?

Ein Computer ist eine Maschine, die auf Digitaltechnik basiert. Was bedeutet es, dass ein zeitkontinuierliches dynamisches System durch eine digitale Machine simuliert wird? Die Simulation besteht in der numerischen Approximation der Differentialgleichungen, die das zeitkontinuierliche Modell charakterisieren, durch Differenzengleichungen, die das zeitdiskrete Modell charakterisieren. Wegen der Näherungen ist die numerische Siumulation fehlerbehaftet, eine weitere Fehlerquelle ist die Verwendung rationaler Näherungswerte für reelle Zahlen. Dieses Verhalten, dass in der Automatisierungstechnik als Diskretisierung bekannt ist, wird in der Fachliteratur ausführlich behandelt, auf die hier zur Vertiefung verwiesen sei.

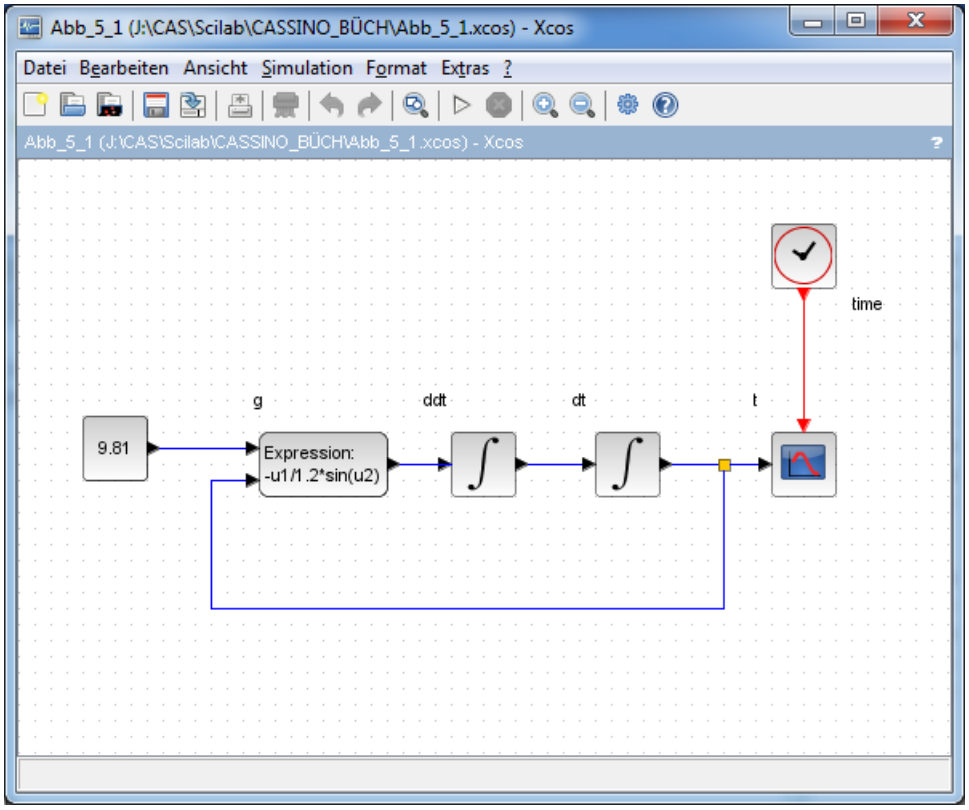

Abb. 5.1: Ein Strukturbild in Xcos

Wer beabsichtigt, Xcos für die Simulation einfacher, insbesondere linearer und stationärer dynamischer Modelle, zu verwenden, muss wegen der Diskretisierung keine signifikanten Fehler befürchten, zumindest wenn er keine Systeme simuliert, deren Teile sehr unterschiedliche Dynamiken aufweisen (Eigenwerte oder voneinander weit entfernte Pole unterschiedlicher Ordnung).

Man beachte, dass der Diskretisierungsfehler etwas anderes ist als ein Einstellungsfehler des Oszilloskops (siehe Abschnitt 5.2.2).

5.1.2 Xcos starten

Zum Start von Xcos muss auf der Kommandozeile von Scilab nur

```
--> xcos;
```

eingegeben werden oder man folgt mit der Maus dem Menu `Anwendungen -> Xcos`, wie in Abbildung 5.2 gezeigt.

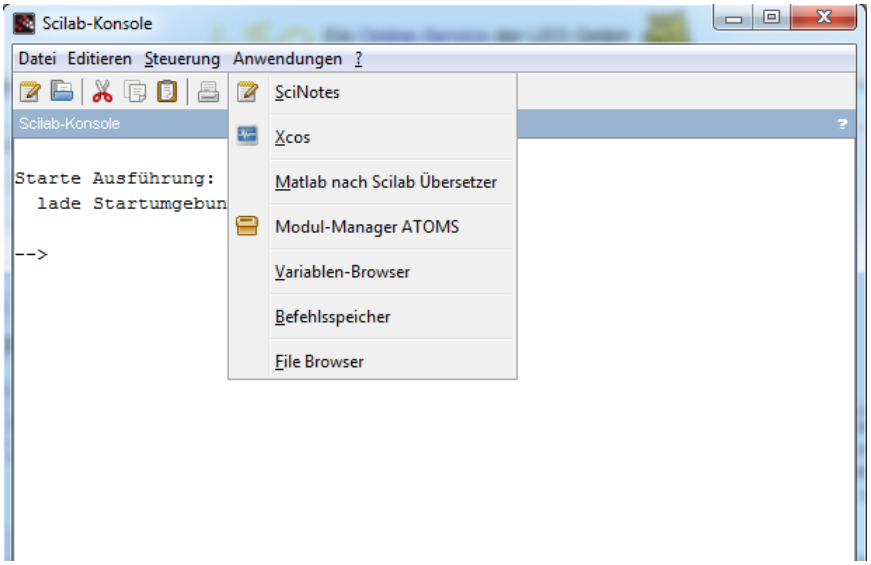

Abb. 5.2: Start von Xcos mit der Maus im Startfenster von Scilab

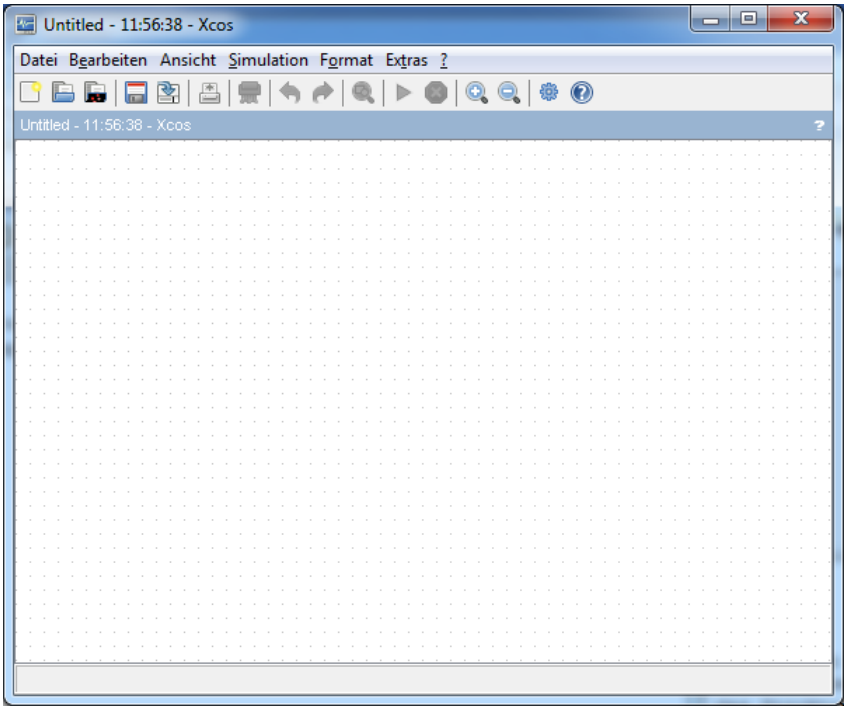

Abb. 5.3: Das noch leere Fenster für die Konstruktion des Simulationsmodells

Nun öffnet sich ein Fenster für die Konstruktion des Strukturbildes der Simulation mit dem voreingestellten namen `Untitled` (siehe Abbildung 5.3). Dieser Name kann beim Abspeichern in eine `xcos`-Datei geändert werden. Will man z.B. das Strukturbild in `meinschema` umbenennen, führt man den Befehl `Speichern als...` im Menu `Datei` des Fensters `Untitled` aus und gibt als Namen `meinschema.xcos` an.

5.1.3 Was ist ein Block?

Ein Block ist eine grafische Funktion, die für folgende Elemente stehen kann:

- Eingang
- Zustand
- Augang
- skalare/vektorielle Größen
- zeitkontinuierlich/zeitdiskret
- Quellen
- Senken

Die ersten Elemente sind selbsterklärend. Die Quellen beinhalten, dass die Funktion möglicherweise nur dann ausgewertet wird, wenn eine Anforderung eintrifft. Somit kann eine Behandlung von Ereignissen erfolgen, die nicht an den Zeitablauf sondern an den Wert von Variablen gekoppelt sind. Entsprechend kann ein Block ein Signal zur Aktivierung eines anderen Blocks erzeugen. Eingänge und Ausgänge werden durch schwarze Dreiecke dargestellt, Quellen und Senken durch rote Dreiecke.

5.2 Die Paletten

Mit dem Ausdruck *Palette* bezeichnet man die Bibliothek von Blöcken, die für die Konstruktion von Strukturbildern in Xcos verfügbar sind.

Der *Paletten-Browser* offnet sich automatisch beim Aufruf von Xcos. Ihn zeigt die Abbildung 5.4.

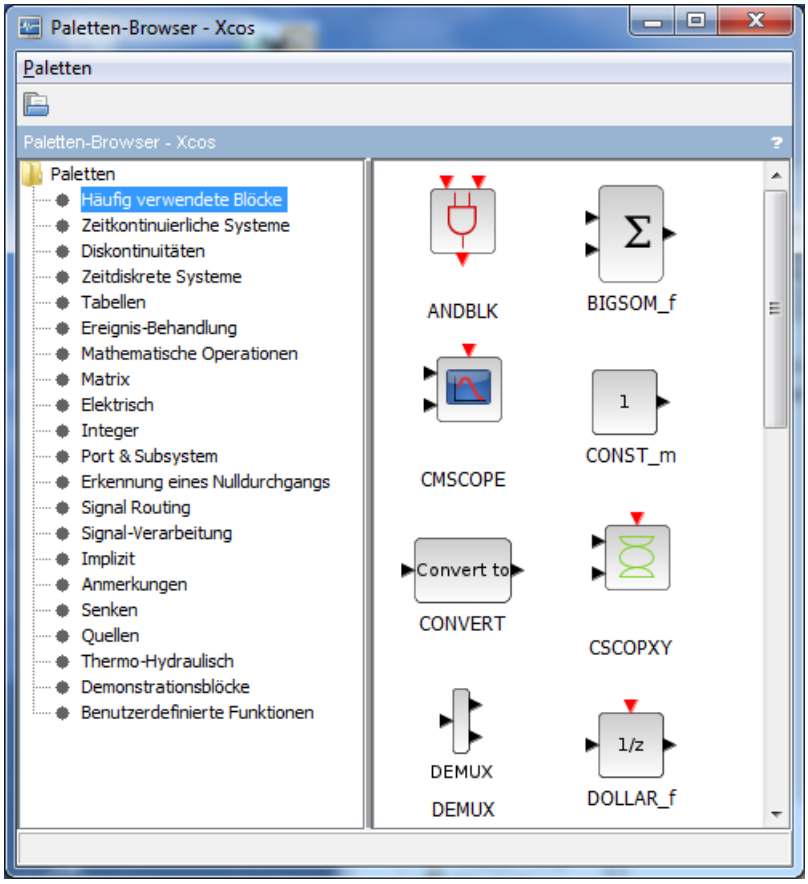

Abb. 5.4: Das Fenster des Paletten-Browsers

Hat man einen interessierenden Block in einer der Biblioteken identifiziert, muss er vom Paletten-Browser in das Xcos-Fenster gebracht werden, am besten durch *drag and drop*.

5.2.1 Quellen

Das Verzeichnis *Quellen* enthät, wie Abbildung 5.5 zeigt, alle Blöcke, die in der Lage sind, Signale zu erzeugen, und somit als Quellen einem Xcos-Strukturbild als Quellen dienen können. Außer dass sie Signale mit analytischen Funktionen erzeugen können, wie eine konstante Funktione, eine Sinusfunktion, eine Sägezahnfunktion usw. ist es möglich, ein Signal aus dem Arbeitsraum zu laden oder aus einer Datei. Der Anwender kann folglich ein beliebiges Signal erzeugen.

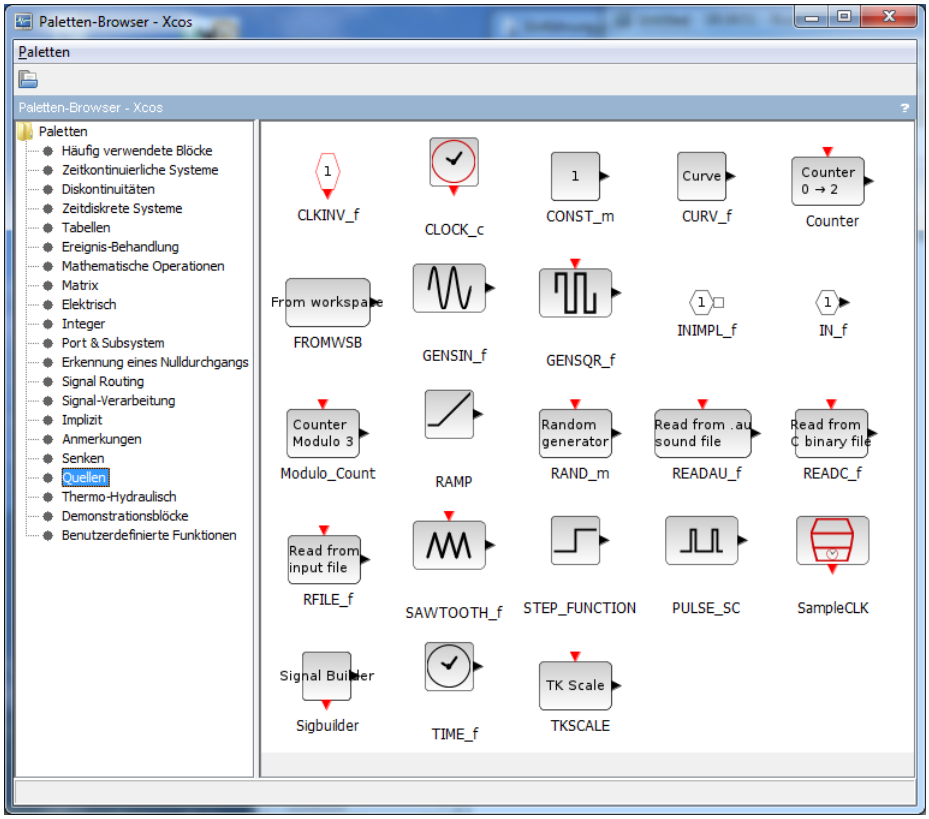

Abb. 5.5: Die Blöcke der Quellen in Xcos

5.2.2 Senken

Senken sind alle die Blöcke, die Ergebnisse einer Simulation visualisieren bzw. speichern. Es gibt grafische Blöcke, wie Oszilloskope, oder numerische, die die Speicherung der Daten im Arbeitsraum oder in einer Datei erlauben.

5.2.3 Funktionen für dynamische Systeme

In den Verzeichnissen `Zeitkontinuierliche Systeme` und `Zeitdiskrete System` sind zahlreiche Blöcke für die Simulation linearer dynamischer Systeme verfügbar. Es ist damit möglich, das Verhalten von System mit Eingang-Zustand-Ausgang oder Eingang-Ausgang zu simulieren und die System-Matrizen und die Übertragungsfunktion zu erhalten. In diesem Verzeichnis gibt es auch Blöcke zur Berechnung des Integrals und der Ableitung nach der Zeit.

5.2.4 Weitere Blöcke

Der beste Weg zum Kennenlernen der Blöcke ist das Öffnen der Verzeichnisse und das Studium der zugehörigen Online-Hilfe. Viele nützliche Blöcke gibt es auch in `Mathematische Operationen` und in `Signal-Verarbeitung`. Wir verweisen auch auf die Blöcke, die in C, in

Fortran oder auch in Scilab geschriebenen Code ausführen und damit einen beliebigen mathematischen Ausdruck realisieren.

5.3 Einstellen von Blockparametern

Ist der gewünschte Block ausgewählt, genügt es, ihn in das Strukturbild zu ziehen und in geeigneter Weise zu verbinden. Einen Block zu verbinden, heißt festzulegen, welches Signal gegebenenfalls seinen Eingang bildet und was gegebenenfalls mit seinem Ausgang geschehen soll. Der Block ist, wie gesagt, eine grafische Funktion; für einen jeden Block werden vor seiner Verwendung die Simulationsparameter festgelegt. Stellt ein Block z.B. einen sinusförmigen Eingang dar, werden Amplitude, Frequenz und u.U. Anfangsphase des Sinus eingestellt. Viele Blöcke haben voreingestellte Parameter, man hat also darauf zu achten, nicht eine Simulation zu starten, ohne vorher die Parameter sämtlicher Blöcke angepasst zu haben. Die voreingestellten Parameter vermeiden zwar Syntaxfehler, begünstigen aber das Auftreten semantischer Fehler.

Für den Zugriff auf die Parameter eines Blocks, klickt man mit der rechten Maustaste auf den Block und wählt `Block Parameter` aus dem Aufklappmenu. Man aktiviert dieses Dialogfenster auch direkt durch Doppelklick auf das Icon des Blocks.

In Abbildung 5.6 sieht man das Dialogfenster des Blocks `CONST_m`; in die Zeile kann der numerische Wert der Konstanten oder ein Ausdruck zur Berechnung ihres Wertes eingegeben werden (siehe insbesondere Abschnitt 5.5).

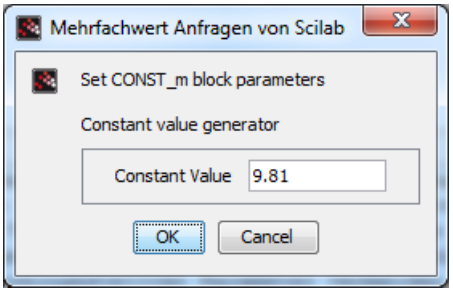

Abb. 5.6: Das Dialogfenster des Blocks `CONST_m`

5.4 Optionen für die Simulation

Die Simulation selbst benötigt Parameter und auch hier startet die Simulation mit den voreingestellten Werten. Für den Zugriff und eine evtl. Anpassung ist `Simulation -> Einstellungen` anzuwählen und man erhält ein Fenster wie es die Abbildung 5.7 zeigt.

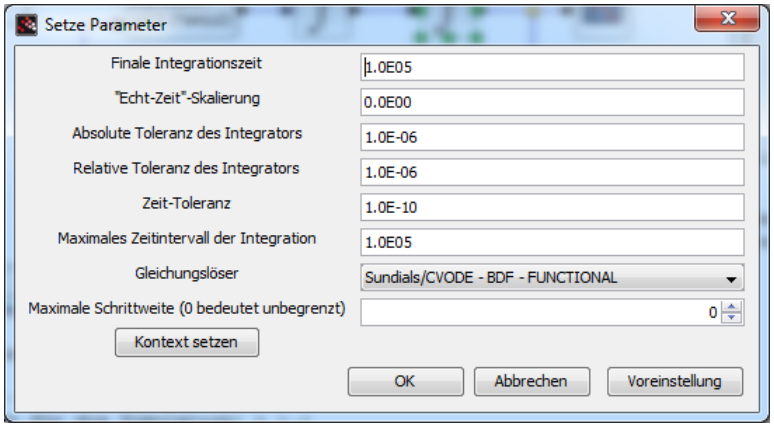

Abb. 5.7: Fenster mit den Optionen für die Simulation

Für einfache Systeme ist als einziger Parameter die Dauer der Simulation einzustellen (`Finale Integrationszeit`), während man die übrigen unverändert lässt. Die Integrationszeit wird am besten vor dem Start der Simulation aufgrund der Eigenschaften des zu simulierenden Systems fstgelegt, weil man andernfalls schwer zu interpretierende Grafiken erhalten kann. Will man z.B. ein elektromechanisches System mit einer Zeitkonstanen im Bereich von Millisekunden simulieren, und setzt man die Integrationszeit auf 100 Sekunden, wird das Resultat gerne missverständlich sein. Ähnliche Überlegungen gelten für die Parameter des Oszilloskops.

5.5 Verwaltung der Variablen

Es ist zweckmäßig, den Gebrauch von Konstanten zu vermeiden. Das gilt sowohl für einzelne Blöcke als auch für die Simulation. Für die Minimierung der Fehlermöglichkeiten ist es sinnvoll, die Definition sämtlicher Variablen zu zentralieren und in den Blöcken nur symbolische Variablen zu setzen. So ist es z.B. möglich und zweckmäßig, für alle Oszilloskope im Strukturbild dasselbe Zeitintervall festzulegen.

Eine empfehlenswerte Vorgehensweise besteht z.B. darin, den `Kontext` zu setzen. Das geht so:

- Man schreibt alle Variablen in eine Datei und nennt sie z.B. `var.sce`.
- Man öffnet `Simulation -> Kontext setzen` und schreibt `exec('var.sce')`.
- Man benutzt die symbolischen Variablen als Parameter.

Man beachte, dass die Kontext-Variablen nur von den Blöcken gesehen werden. Der Kontext kann also nicht zur Einstellung der Parameter der Simulation dienen, was in Abnschnitt 5.4 gezeigt wurde.

Eine Möglichkeit, die Simulationszeit über den Kontext einzustellen, ist mit Verwendung des Blocks `ENDBLK` im Verzeichnis `Senken` gegeben. Hat man etwa die Zeile `tf=3` in der Datei `var.sce`, benutzt man die symbolische Variable `tf` als Parameter im Block `ENDBLK`.

Eine letzte Möglichkeit stellen globale Variablen dar.

5.6 Starten der Simulation

Für den Start der Simulation wählt man den Befehl `Simulation -> Start` oder klickt auf das dreieckige Icon.

5.7 Aufbau eines einfachen Modells

Wir wollen jetzt die Simulation eines einfachen zeitkontinuierlichen dynamischen Systems erstellen. Es habe die Übertagungsfunktion

$$F(s) = \frac{1}{s^2 + s + 1}$$

Dafür braucht man ein Eingangssignal. Wir entscheiden uns hier für ein konstantes Signal, nehmen einen Block, der die Funktion $F(s)$ darstellt und wählen als Ausgang ein einfaches Oszilloskop.

Die Abbildung 5.5 zeigt das Verzeichnis der Quellen, woraus wir den konstanten Block leicht auswählen und in das Strukturbild hinüberziehen.

Auf ähnliche Weise kann der Block `CLR` (Continuous Transfer Function) aus dem Verzeichnis `Zeitkontinuierliche Systeme` ausgewählt werden. Schließlich braucht man noch das Oszilloskop aus dem Verzeichnis `Senken`. Dieser Block hat einen Eingang zur Aktivierung mit einem *event clock generator*, der eine regelmäßige Abtastung des Signals der Übertragungsfunktion bewirkt.

Den Aufbau des Strukturbildes zeigt Abbildung 5.8.

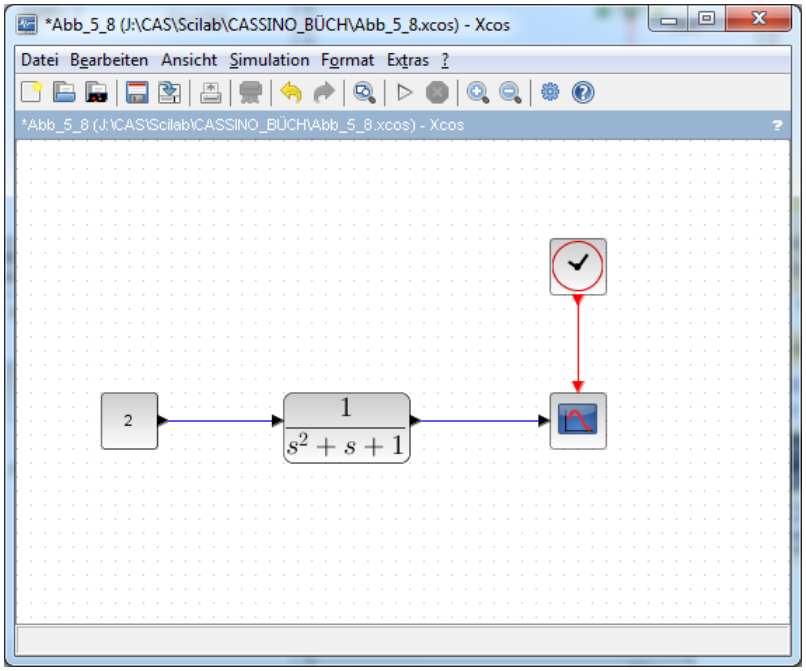

Abb. 5.8: Ein einfaches Strukturbild in Xcos

Nun müssen alle Parameter der Blöcke und der Simulation gesetzt werden, wie das in den vorstehenden Abschnitten kurz gezeigt wurde und man erhält den grafischen Ausgang des Oszilloskops, wie ihn Abbildung 5.9 zeigt.

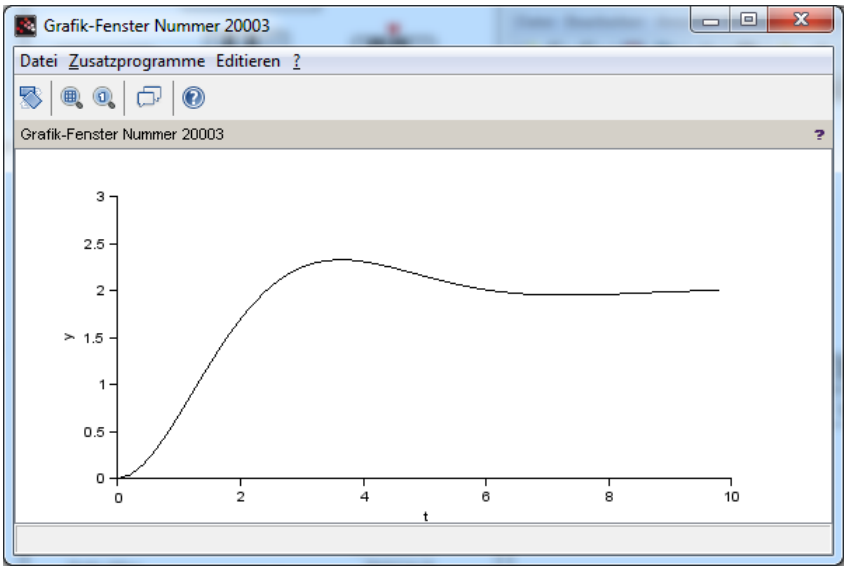

Abb. 5.9: Anzeige des Ergebnisses für das Strukturbild in Abbildung 5.8

5.8 Import und Export von Daten

Die Strukturbilder in Xcos können Daten außerhalb von Xcos nutzen und speichern. Dafür gibt es zwei Möglichkeiten, nämlich die Interaktion mit einer Datei oder mit dem Arbeitsraum von Scilab. Es ist möglich, mit einer geeigneten Syntax in eine Datei zu schreiben (`WRITEC_f` und von ihr `WFILE_f`) und zu lesen (`READC_f` und `RFILE_f`), sowie mit den Blocks `TOWS_c` und `FROMWS_c` auf Variablen im Arbeitsraum schreibend und lesend zuzugreifen.

Da ist es hilfreich, dass bei geöffnetem Xcos-Fenster der Scilab-Workspace über das Menu `Datei` zugänglich ist.

5.9 Aufbau eines angepassten Blocks

Wie in der traditionellen Programmierung können auch in Xcos angepasste Blöcke erzeugt werden und damit auch´eine Bibliothek analog zu eine Funktionenbibliothek. Dafür müssen nur die entsprechenden Befehle aus dem Paletten-Browser verwendet werden.

5.10 Der Superblock

Der Begriff Superblock meint eine Zusammenfassung mehrerer Blöcke zu einem einzigen mit entsprechenden Ein- und Ausgängen. Zur Erzeugung eines Superblocks muss nur das infrage kommende Gebiet markiert werden, wobei auf die Ein- und Ausgänge des Superblocks besonders zu achten ist, und dann der Menupunkt `Bearbeiten -> Region zu Superblock` auszuwählen. Ein existierender Superblock kann dann nach einem Doppelklick auch modifiziert werden: es öffnet sich ein Fenster, dass den Inhalt dieses Blocks aufklappt.

5.11 Beispiel: Simulation eines einfachen Pendels

5.11.1 Modellierung des Pendels

Für das Pendel in Abbildung 5.10 betrachten wir die folgenden Parameter:

- l - Länge des starren Pendels
- m - Masse (punktförmig) am Ende des Pendels konzentriert
- μ - Trägheitsmoment bezüglich des Aufhängepunktes des Pendels

und die Größen

- ϑ - Auslenkung des Pendels aus der Vertikalen (gegen den Uhrzeigersinn positiv)
- g - Erdbeschleunigung (nach unten positiv)
- τ_g - Drehmoment der Gewichtskraft bezüglich des Aufhängepunktes (bei Drehung gegen den Uhrzeigersinn positiv)

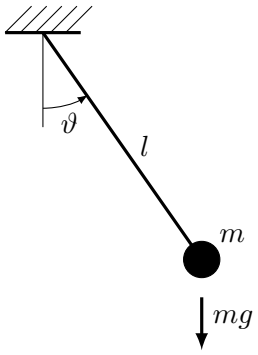

Abb. 5.10: Schematische Darstellung eines einfachen Pendels

Aus der Mechanik hat man

$$\mu\ddot{\vartheta} = \tau_g$$
$$\mu = ml^2$$
$$\tau_g = mgl\sin(\vartheta)$$

Mit ϑ als Ausgang und g als Eingang, können wir die Differentialgleichung

$$\ddot{\vartheta} + \frac{g}{l}\sin(\vartheta) = 0.$$

herleiten. Um uns indessen auf die Struktur der Abbildung 5.11 zu beziehen, müssen wir von der allgemeinen Form

$$\begin{cases} \dot{x} = f(x,u) \\ y = g(x,u) \end{cases} \tag{5.1}$$

ausgehen.

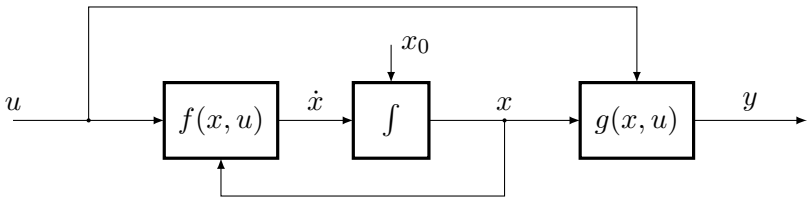

Abb. 5.11: Strukturbild für das Gleichungssystem (5.1)

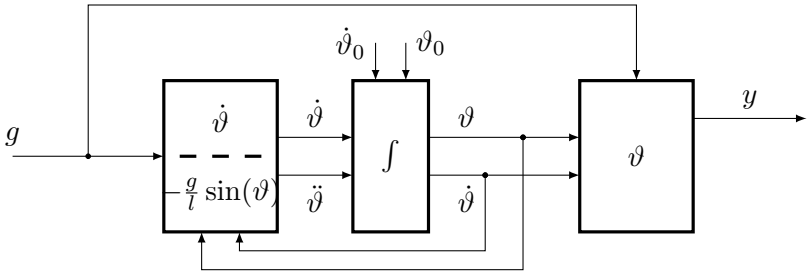

Abb. 5.12: Strukturbild für das Gleichungssystem (5.2)

In unserem Fall setzen wir $y = \vartheta$, $u = g$ und mit

$$x = \begin{pmatrix} \vartheta \\ \dot{\vartheta} \end{pmatrix}$$

kann man schreiben

$$\begin{cases} \begin{pmatrix} \dot{\vartheta} \\ \ddot{\vartheta} \end{pmatrix} &= \begin{pmatrix} \dot{\vartheta} \\ -\frac{g}{l}\sin(\vartheta) \end{pmatrix} \\ y &= \vartheta \end{cases} \tag{5.2}$$

Das entspricht dem Strukturbild in Abbildung 5.12. Man beachte, dass die Zustandsvariable $x_1 = \vartheta$ an die potentielle Energie des Pendels gebunden ist und die Variable $x_2 = \dot{\vartheta}$ an seine kinetische Energie. Überdies darf die erste Gleichung nicht als banale Identität verstanden werden, vielmehr zeigt die Gleichheit

$$\dot{x}_1 = x_2$$

den integralen Zusammenhang zwischen den beiden Zustandsvariablen.

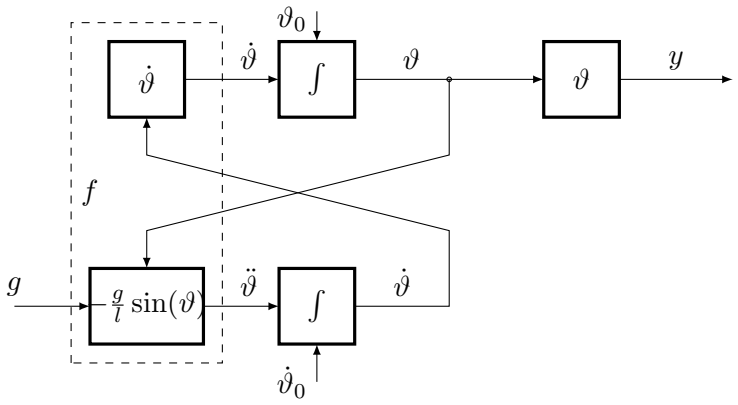

Abb. 5.13: Transformation des Strukturbildes aus Abb. 5.12

Durch Umbau der Abbildung 5.12 erhält man den Informationsfluss von Abbildung 5.13. Von hier führt eine nochmalige Vereinfachung der Grafik auf die Abbildung 5.14. Im folgenden behandeln wir ein Pendel mit den Parametern

$$m = 1 \text{ kg} \qquad l = 1 \text{ m}$$

bei einer konstanten Fallbeschleunigung von $g = \hat{g} = 9.81 \text{ m/s}^2$.

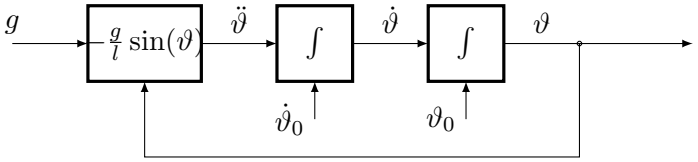

Abb. 5.14: Vereinfachung des Strukturbildes aus Abb. 5.13

5.11.2 Konstruktion des Strukturbildes in Xcos

Zur Simulation des Pendels schreiten wir nun zur Konstruktion des Strukturbildes in Xcos auf der Basis der Abbildung 5.14.

Zu diesem Zweck rufen wir das Grafik-Interface auf wie in Abschnitt 5.1.2 beschrieben und speichern die Datei z.B. unter dem Namen `pendel.xcos`.

Bei unserer Aufgabe kommen folgende Blöcke zum Einsatz:

- ein Block zur Erzeugung des konstanten Signals, das den Eingang darstellt (Block `CONST_m`) im Verzeichns `Quellen`,
- ein Block zur Berechnung einer algebraischen Funktion (Block `EXPRESSION` im Verzeichnis `Benutzerdefinierte Funktionen`),
- zwei Blöcke zur Ausführung von Integrationen (Block `INTEGRAL_m` im Verzeichnis `Zeitkontinuierliche Systeme`),

- einen Oszilloskop-Block zur Visualisierung des Verlaufs des Ausgangs (Block `CSCOPE` im Verzeichnis `Senken`). Man beachte, dass der Block `CSCOPE` außer der zu visualisierenden Variablen auch einen Zeiteingang benötigt, was durch einen Block `CLOCK_c` im Verzeichnis `Quellen` bewerkstelligt wird.

Zur Berechnung der algebraischen Funktion $-(g/l)\sin(\vartheta)$ müssen am Eingang des Blocks `EXPRESSION` zwei Größen bereitgestellt werden, die die Werte der Variablen g und ϑ liefern. Verbinden wir z.B. die Größe g mit dem ersten und die Größe ϑ mit dem zweiten Eingangsport, wird die im Dialogfenster des Blocks `EXPRESSION` einzugebende Funktion, wie Abbildung 5.15 zeigt,

$$-\frac{g}{l}\sin(\vartheta) \qquad \rightarrow \qquad -u1/l * sin(u2),$$

worin, statt die Variable l im Kontext zu definieren, der Zahlenwert auch direkt eingegeben werden kann.

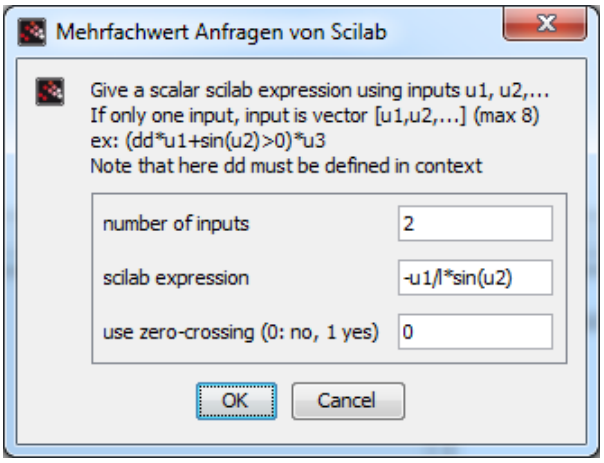

Abb. 5.15: Das Dialogfenster des Blocks `EXPRESSION`

Die beiden Blöcke `INTEGRAL_m` benötigen jeweils Anfangsbedingungen, die in der dafür vorgesehenen Zeile des Dialogfensters (siehe Abbildung 5.16) eingegeben werden können. Zu beachten ist, dass die Winkel in Radiant einzugeben sind. Dabei ist Eingabe mit den Umrechnungsfaktoren am bequemsten. Bei einer Anfangsauslenkung von z.B. 45° schreibt man 45° in die Zeile `Initial Condition` des zweiten Integrals. Der Anfangswert des ersten ist null, weil sich das Pendel bei 45° im Umkehrpunkt befindet, wo die Geschwindigkeit null ist.

Werden dann noch die Verbindungen zwischen den Blöcken entsprechend dem Schema in Abbildung 5.14 hergestellt, bekommt man das in Abbildung 5.1 gezeigte Xcos-Strukturbild.

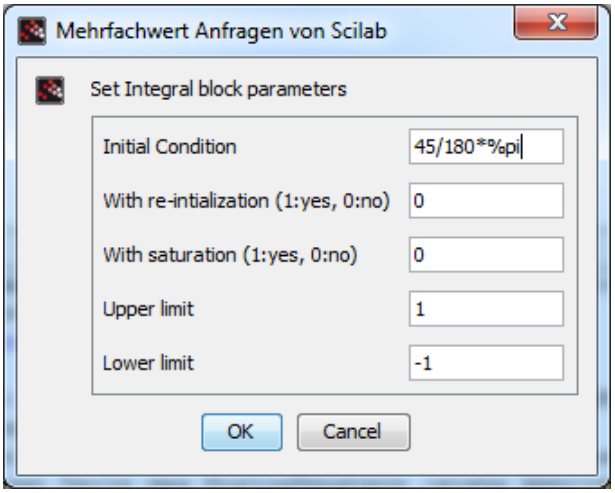

Abb. 5.16: Das Dialogfenster des Blocks `INTEGRAL_m`

An dieser Stelle empfiehlt es sich, das Strukturbild zu speichern.

5.11.3 Ausführung der Simulation

Nach der Erstellung des Strukturbildes in Xcos können wir die Simulation starten. Dafür müssen zunächst die Parameter der Simulation im Dialogfenster unter `Simulation -> Einstellungen` konfiguriert werden.

Von den vielen Parametern, die eingestellt werden können, ist es im ersten Anlauf sinnvoll, nur die Dauer der Simulation anzupassen (`Finale Integrationszeit`). In unserem Fall ergeben die physikalischen Parameter des Pendels eine Schwingungsdauer im Sekundenbereich. Wir können uns also an einer Dauer der Simulation von 5 Sekunden orientieren.

Außerdem ist die Konfiguration der Parameter des Blocks `CLOCK_c` sinnvoll. Der voreingestallte Wert von 0.1 sec für die Periodenlänge `Period` reicht für die Auflösung der grafischen Darstellung zwar aus, doch erscheint eine Periode von 0.01 s bei 5 s Dauer besser geeignet.

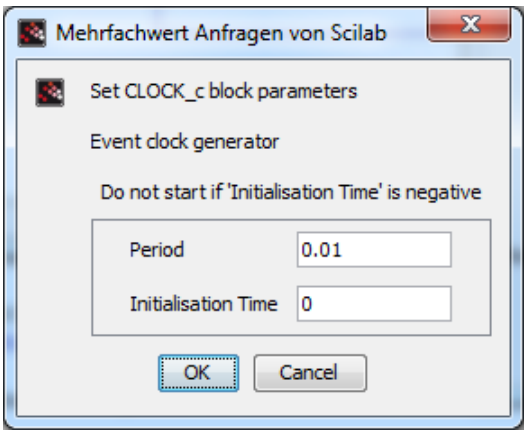

Abb. 5.17: Das Dialogfenster des Blocks `CLOCK_c`.

Außerdem schließt der voreingestellte Wert von `Initialiation Time` von 0.1 sec die Anzeige der Anfangswerte aus; wir setzen ihn daher auf 0 (siehe Abbildung 5.17).

Nun können wir die Simulation starten, z.B. indem wir den Befehl `Start` im Menu `Simulation` anwählen; das Ergebnis der Simulation wird in einem Grafikfenster sichtbar, das durch den Block `CSCOPE` aktiviert wird. Z.B. zeigt sich das Grafikfenster mit den Anfangsbedingungen ($\dot{\vartheta} = 0°/s$ und $\vartheta = \pi/4$) wie die Abbildung 5.18 zeigt.

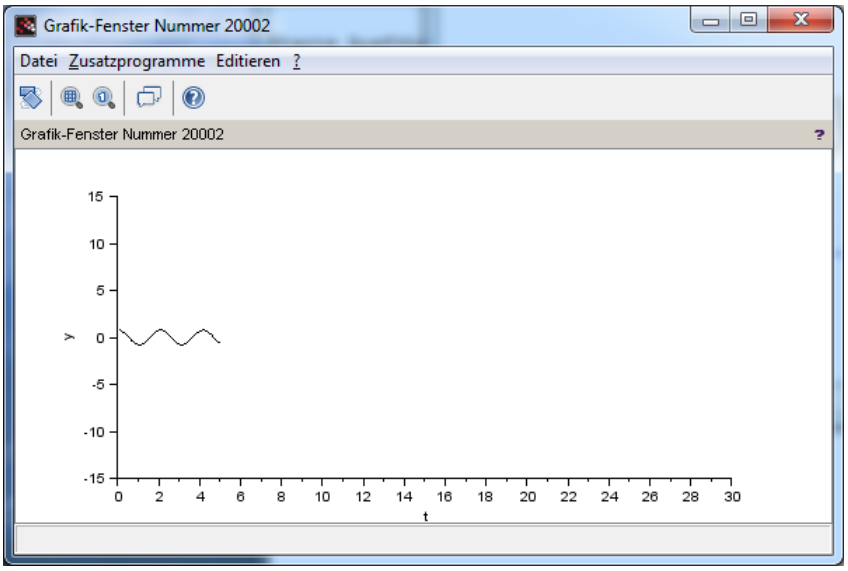

Abb. 5.18: Grafikfenster mit den Einstellungen des Blocks `CSCOPE` aus Abb. 5.17 am Ende der Simulation mit `Finale Integrationszeit = 5 sec, Initial Condition` für den ersten Integrator = 0 und `Initial Condition = 45/180*%pi` für den zweiten

Man kann erkennen, dass der Maßstab der horizontalen Achse dem Zeitintervall `Refresh period` entspricht, das bei den Eigenschaften des Blocks `CSCOPE` definiert wurde, wohingegen sich der vertikale Maßstab vom Wert `Ymin` bis zum Wert `Ymax` erstreckt (siehe Abb. 5.19).

Zur besseren Anpassung der Maßstäbe der Achsen an die Größe des Bildes der Simulation gibt es zwei Möglichkeiten:

- Festlegung neuer Werte für die Parameter `Refresh period, Ymin` und `Ymax` in den Eigenschaften des Blocks `CSCOPE` im Dialogfenster, das Abbildung 5.19 zeigt. Dieser Ansatz erfordert den Neustart der Simulation, bewahrt aber die Werte für alle zukünftigen Ausführungen, sowie abgespeichert auch für alle zukünftigen Sitzungen,
- Festlegung der Abmessungen von Abszisse und Ordinate im Dialogfenster `Voreinstellungen der Achsen` im Menu `Editieren` des Grafikfensters (siehe Abschnitt 3.4).

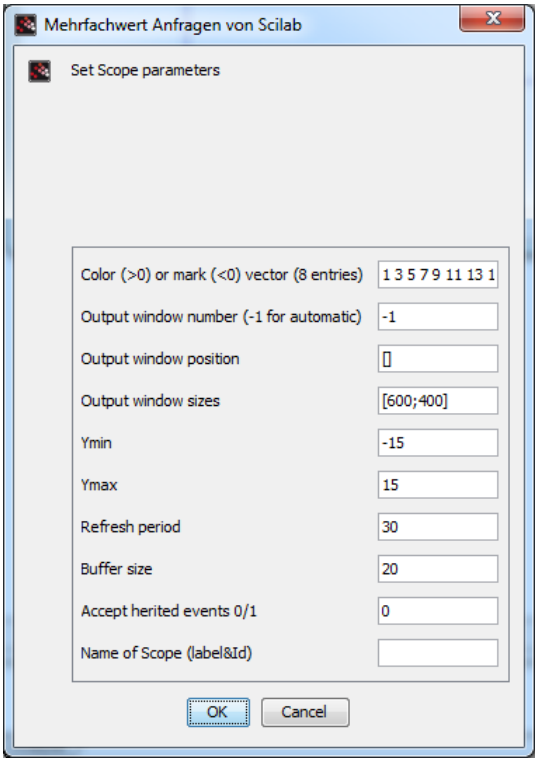

Abb. 5.19: Dialogfenster des Blocks `CSCOPE`

Wenn wir den Maßstab der Darstellung also derart verändern, dass wir auf der Abszisse die gesamte Dauer der Simulation (5 s) und auf der Ordinate eine Auslenkung von -1 bis $+1$ abbilden, dann erhalten wir für den geänderten Block `CSCOPE` ein Grafikfenster gemäß Abbildung 5.20.

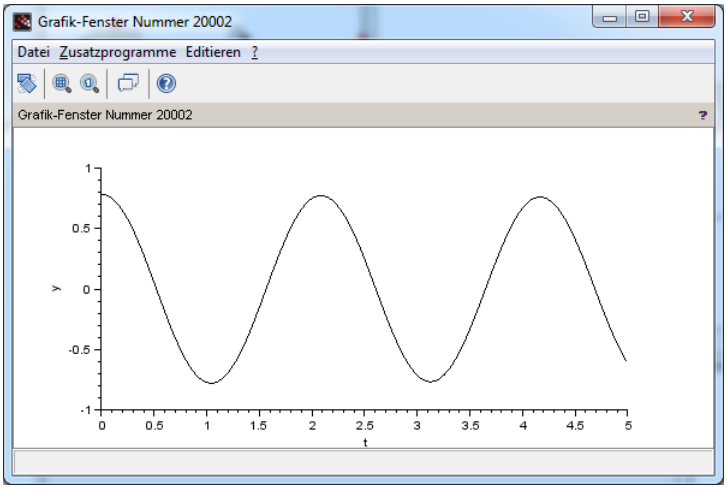

Abb. 5.20: Simulation wie in Abb. 5.18 nach Reskalierung der Achsen

5.11.4 Linearisierung

Wenn wir eine Lösung des Problems

$$f(\hat{x}, \hat{u}) = 0$$

suchen und dabei von der Gleichung im Zustandsraum (5.2) ausgehen, finden wir leicht, dass dass Pendel mit $k \in \mathbb{N}k$ über stationäre Gleichgewichtslagen verfügt, die durch

$$\hat{x}_1 = \hat{\vartheta} = k\pi \qquad \hat{x}_2 = \hat{\dot{x}}_2 = 0 \qquad \hat{u} = \hat{g}$$

charakterisiert sind. Führen wir Variablen für die Abweichung ein mit

$$\delta\vartheta = \vartheta - \hat{\vartheta} \qquad \delta\dot{\vartheta} = \dot{\vartheta} - \hat{\dot{\vartheta}} = \dot{\vartheta}$$

lautet die Übergangsfunktion des linearisierten Modells

$$\left\{ \begin{array}{rl} \begin{pmatrix} \dot{\vartheta} \\ \ddot{\vartheta} \end{pmatrix} & = \begin{pmatrix} 0 & 1 \\ -\frac{\hat{g}}{l}\cos(\vartheta) & 0 \end{pmatrix} \begin{pmatrix} \delta\vartheta \\ \dot{\vartheta} \end{pmatrix} = \begin{pmatrix} \dot{\vartheta} \\ -\frac{\hat{g}}{l}\cos(\hat{\vartheta})\delta\vartheta \end{pmatrix} \\ y & = \delta\vartheta \end{array} \right. \tag{5.3}$$

Mit Bezug auf das durch

$$\hat{\vartheta} = 0 \qquad \hat{\dot{\vartheta}} = 0 \qquad \hat{g} = 9.81 \text{ m/s}^2$$

charakterisierte Gleichgewicht reduziert sich das linearisierte Modell zu

$$\left\{ \begin{array}{rl} \begin{pmatrix} \dot{\vartheta} \\ \ddot{\vartheta} \end{pmatrix} & = \begin{pmatrix} \dot{\vartheta} \\ -\frac{\hat{g}}{l}\vartheta \end{pmatrix} \\ y & = \vartheta \end{array} \right. \tag{5.4}$$

Diese Gleichung kann analog zur Vorgehensweise wie in Abschnitt 5.11.1 auf das Strukturbild in Abbildung 5.21 reduziert werden.

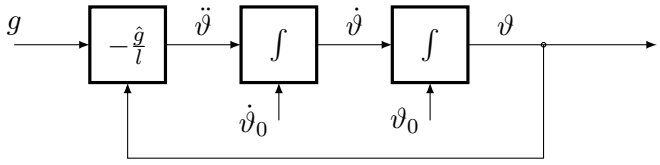

Abb. 5.21: Vereinfachtes Strukturbild der Übergangsfunktion (5.4)

Angesichts der strukturellen Ähnlichkeit mit dem Schema in Abbildung 5.14 kann man zur Konstruktion des Xcos-Strukturbildes zu Abbildung 5.21 das Schema in Abbildung 5.1 kopieren und entsprechend anpassen. Zu diesem Zweck markiert man zuerst das ganze Strukturbild des Pendels, z.B. mit dem Cursor oder mit der Tastenkombination `Ctrl+A` und fährt mit der klassischen Sequenz `Ctrl+C, Ctrl+V` fort, unter Linux und Windows das Werkzeug zur Duplikation. Um die Überdeckung des originalen Bildes durch das kopierte zu vermeiden, klickt man auf einen geeigneten Punkt des zweiten Schemas und und zieht es mit der Maus in die

gewünschte Position. Nun wird das durch die Duplikation erhaltene zweite Strukturbild an die Simulation gemäß Abbildung 5.21 angepasst, indem die Funktion im Block `EXPRESSION` von

$$-\frac{g}{l}\sin(\vartheta) \qquad \rightarrow \qquad -u1/l * sin(u2)$$

nach

$$-\frac{\hat{g}}{l}\vartheta \qquad \rightarrow \qquad -u1/l * u2$$

geändert wird.

Zu beachten ist, dass durch das Kopieren den duplizierten Objekten auch alle Eigenschaften der jeweiligen originalen Objekte mitgeteilt werden, insbesondere trifft das auf die beiden Integrator-Blöcke zu, sodass der Vergleich der beiden Simulationen ohne weiteres mit denselben Anfangsbedingungen erfolgen kann.

Um einen direkten Vergleich der beiden Ergebnisse zu bekommen, müssen beide Ausgänge auf dasselbe Oszilloskop gelegt werden. Das kann auf zwei Arten geschehen:

- durch Verwendung eines Blocks `CMSCOPE` statt des Blocks `CSCOPE`; beide Eingangssignale werden in jeweils eigenen Koordinatensystem dargestellt,
- durch Zusammenführung der Ausgänge in einem Block `MUX` aus dem Verzeichnis `Signal Routing` werden beide Signale zu einem einzigen multivariablen Signal vereinigt, das einem einzelnen Block `CSCOPE` als Eingang zugeführt wird. Man erhält eine einzige Grafik, die beide Kurven im selben Koordinatensystem enthält.

Hier verwenden wir die zweite Lösung, weil der Vergleich der beiden Simulationen hier besser ins Auge springt. Das resultierende Strukturbild ist in Abbildung 5.22 dargestellt und das Ergebnis der Simulation von 10 s Dauer in Abbildung 5.23. Leicht erkennt man, dass sich das Ergebnis der Simulation des linearisierten System bei Ausschlägen von ±45° sehr schnell von demjenigen des korrekten Systems entfernt.

Wiederholt man indessen die Simulation mit einem Anfangsinkel von 5°, liefert das linearisierte Modell eine sehr gute Näherung für das genaue Modell (siehe Abbildung 5.24).

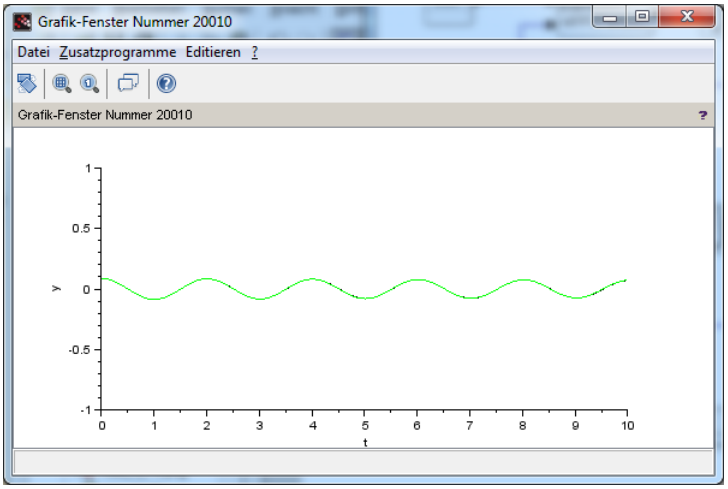

Abb. 5.24: Fenster mit dem Block `CSCOPE` in Abbildung 5.22 am Ende der Simulation mit 5°

Insbesondere mit dem *Zoom*-Werkzeug kann man nach einer Simulation von etwa 10 s Dauer einen Unterschied zwischen beiden Lösungen von nur etwa 4.5 ms ablesen.

5.11.5 Liste von Blöcken in Xcos

Eine nicht erschöpfende Zusammenstellung von Xcos-Blöcken wie sie im Fenster des Hilfe-Browsers erscheinen:

- Annotations palette

Annotations_pal	Annotations palette
TEXT_f	Free annotation

- Commonly used blocks palette

Commonlyusedblocks_pal	Commonly used blocks palette
LOGICAL_OP	Logical operation
RELATIONALOP	Relational operation

- Continuous time system palette

Continuous_pal	Continuous time systems palette
CLINDUMMY_f	Dummy
CLR	Continuous transfer function
CLSS	Continuous state-space system
DERIV	Derivative
INTEGRAL_f	Integration
INTEGRAL_m	Integration
PID	PID regulator
TCLSS	Continuous linear system with jump
TIME_DELAY	Time delay
VARIABLE_DELAY	Variable delay

- Demonstrations blocks palette

Demonstrationsblocks_pal	Demonstrations blocks palette
AUTOMAT	automata (finite state machine)
BOUNCE	Balls coordinates generator
BOUNCEXY	Balls viewer
BPLATFORM	Balls under a platform viewer
PDE	1D PDE block

- Discontinuities palette

discontinuities_pal	discontinuities palette
BACKLASH	Backlash
DEADBAND	Deadband
HYSTHERESIS	Hystheresis
RATELIMITER	Rate limiter
SATURATION	Saturation

- Discrete time systems palette

Discrete_pal	Discrete time systems palette
DELAYV_f	Variable delay
DELAY_f	Discrete time delay
DLR	Discrete transfer function
DLRADAPT_f	Discrete Zero-Pole
DLSS	Discrete state-space system
DOLLAR_f	Delay operator
REGISTER	Shift Register

- Electrical palette

Electrical_pal	Electrical palette
CCS	Controllable Modelica current source
CVS	Controllable Modelica voltage source
Capacitor	Electrical capacitor
ConstantVoltage	Electrical DC voltage source
CurrentSensor	Electrical current sensor
Diode	Electrical diode
Ground	Ground (zero potential reference)
Gyrator	Modelica Gyrator
IdealTransformer	Ideal Transformer
Inductor	Electrical inductor
NMOS	Simple NMOS Transistor
NPN	NPN transistor
OpAmp	Ideal opamp (norator-nullator pair)
PMOS	Simple PMOS Transistor
PNP	PNP transistor
PotentialSensor	Potential sensor
Resistor	Electrical resistor
SineVoltage	Sine voltage source
Switch	Non-ideal electrical switch

VVsourceAC	Variable AC voltage source
VariableResistor	Electrical variable resistor
VoltageSensor	Electrical voltage sensor
VsourceAC	Electrical AC voltage source

- Event handling palette

Events_pal	Event handling palette
ANDBLK	Activation and
ANDLOG_f	Logical and
CEVENTSCOPE	Activation scope
CLKFROM	Receives data from a corresponding CLKGOTO
CLKGOTO	Pass block input to CLKFROM block
CLKGotoTagVisibility	Define Scope of CLKGOTO tag visibility
CLKSOMV_f	Activation union
EDGE_TRIGGER	EDGE_TRIGGER block
ESELECT_f	Synchronous block Event-Select
EVTDLY_c	Event delay
EVTGEN_f	Event generator
EVTVARDLY	Event variable delay
Extract_Activation	Extract_Activation block
HALT_f	Halt
IFTHEL_f	Synchronous block If-Then-Else
MCLOCK_f	MCLOCK_f title
MFCLCK_f	MFCLCK_f title
M_freq	Multiple Frequencies
VirtualCLK0	Triggered Always Active Blocks
freq_div	Frequency division

- Implicit palette

Implicit_pal	Implicit palette
CONSTRAINT_c	Constraint
DIFF_f	Derivative

- Integer palette

Integer_pal	Integer palette
BITCLEAR	Clear a Bit
BITSET	Set a Bit
CONVERT	Data Type Conversion
DFLIPFLOP	D flip-flop
DLATCH	D latch flip-flop
EXTRACTBITS	Bits Extraction
INTMUL	Integer matrix multiplication
JKFLIPFLOP	JK flip-flop
LOGIC	Combinatorial Logic
SHIFT	Shift/Rotates Bits
SRFLIPFLOP	SR flip-flop

- Lookup tables palette

Lookuptables_pal	Lookup tables palette
INTRP2BLK_f	2D interpolation
INTRPLBLK_f	Interpolation
LOOKUP_f	Lookup table

- Math operations palette

Mathoperations_pal	Math operations palette
ABS_VALUE	Absolute value
BIGSOM_f	Scalar or vector Addition/Soustraction
COSBLK_f	Cosine
EXPBLK_m	Exponential of a scalar
GAINBLK_f	Gain
INVBLK	Inverse
LOGBLK_f	Logarithm
MATMAGPHI	Complex from/to Magnitude and Angle Conversion
MATZREIM	Complex decomposition/composition
MAXMIN	Maximum or minimum value of vectors's elements
MAX_f	Maximum value of a vector's elements
MIN_f	Minimum value of a vector's elements
POWBLK_f	Array power
PRODUCT	Element-wise vector multiplication/division
PROD_f	Element-wise product
SIGNUM	Sign

SINBLK_f	Sine
SQRT	Square root
SUMMATION	Matrix Addition/Subtraction
SUM_f	Addition
TANBLK_f	Tangent
TrigFun	Trigonometric function

- Matrix operation palette

Matrix_pal	Matrix operation palette
CUMSUM	Cumulative Sum
EXTRACT	Matrix extractor
EXTTRI	Triangular or Diagonal Extraction
MATBKSL	Left matrix division
MATCATH	Horizontal Concatenation
MATCATV	Vertical Concatenation
MATDET	Matrix Determinant
MATDIAG	Create Diagonal Matrix
MATDIV	Matrix division
MATEIG	Matrix Eigenvalues
MATEXPM	Matrix Exponential
MATINV	Matrix Inverse
MATLU	LU Factorization
MATMUL	Matrix Multiplication
MATPINV	Matrix PseudoInverse
MATRESH	Matrix Reshape
MATSING	SVD Decomposition
MATSUM	Sum of Matrix's Elements
MATTRAN	Matrix Transpose
MATZCONJ	Conjugate of Matrix's Elements
RICC	Riccati Equation
ROOTCOEF	Polynomial Coefficient Computation
SUBMAT	Sub-matrix Extraction

- Port & Subsystem palette

Portaction_pal	Port & Subsystem palette
CLKINV_f	Input activation port
CLKOUTV_f	Output activation port

INIMPL_f	Implicit Input port
IN_f	Regular Input Port
OUTIMPL_f	Output implicit port
OUT_f	Regular Output Port

- Signal processing palette

Signalprocessing_pal	Signal processing palette
QUANT_f	Quantization
SAMPHOLD_m	Sample and hold

- Signal routing palette

Signalrouting_pal	Signal routing palette
DEMUX	Demultiplexer
EXTRACTOR	Extractor
FROM	FROM Receives data from a corresponding GOTO
FROMMO	Receives data from a corresponding GOTOMO
GOTO	GOTO Pass block input to From block
GOTOMO	Pass block input to FROMMO block
GotoTagVisibility	Define Scope of GOTO tag visibility
GotoTagVisibilityMO	Define Scope of GOTOMO tag visibility
ISELECT_m	Iselect
MUX	Multiplexer
M_SWITCH	Multi-port switch
NRMSOM_f	Merge data
RELAY_f	Relay
SELECT_m	Select
SELF_SWITCH	Switch
SWITCH2_m	Switch2
SWITCH_f	Switch

- Sinks palette

Sinks_pal	Sinks palette
AFFICH_m	Display
BARXY	y=f(x) animated viewer
CANIMXY	y=f(x) animated viewer
CANIMXY3D	z=f(x,y) animated viewer
CFSCOPE	Floating point scope

CMAT3D	Matrix z values 3D viewer
CMATVIEW	Matrix Colormapped viewer
CMSCOPE	Multi display scope
CSCOPE	Single Display Scope
CSCOPXY	y=f(x) permanent viewer
CSCOPXY3D	z=f(x,y) permanent viewer
ENDBLK	END block
END_c	END_c block
TOWS_c	Data to Scilab workspace
TRASH_f	Trash block
WFILE_f	Write to output file. This function is obsolete.
WRITEAU_f	Write AU sound file
WRITEC_f	Write to C binary file

- Sources palette

Sources_pal	Sources palette
CLOCK_c	Activation clock
CONST_m	Constant
CURV_f	Curve
Counter	Counter
FROMWSB	Data from Scilab workspace to Xcos
GENSIN_f	Sine wave generator
GENSQR_f	Square wave generator
Modulo_Count	Modulo counter (0 to N counter)
PULSE_SC	Pulse Generator
RAMP	Ramp
RAND_m	Random generator
READAU_f	Read AU sound file
READC_f	Read binary data
RFILE_f	Read from input file
SAWTOOTH_f	Sawtooth generator
STEP_FUNCTION	Step Function
SampleCLK	Sample Time Clock
Sigbuilder	Signal creator/generator
TIME_f	Time
TKSCALE	Adjust value with a graphical widget.

- Thermohydraulics palette

ThermoHydraulics_pal	Thermal-Hydraulics toolbox
Bache	Thermal-hydraulic tank (reservoir)
PerteDP	Thermal-hydraulic pipe
PuitsP	Thermal-hydraulic drain (well)
SourceP	Thermal-hydraulic constant pressure source
VanneReglante	Thermal-hydraulic control valve

- User defined functions palette

Userdefinedfunctions_pal	User defined functions palette
CBLOCK	New C
DSUPER	Masked super block
EXPRESSION	Mathematical expression
MBLOCK	Modelica generic block
SUPER_f	Super block
c_block	C language
fortran_block	Fortran
generic_block3	Generic block
scifunc_block_m	Scilab function block

- Zero crossing detection palette

Zerocrossingdetection_pal	Zero crossing detection palette
GENERAL_f	GENERAL_f title
NEGTOPOS_f	Threshold negative to positive
POSTONEG_f	Threshold positive to negative
ZCROSS_f	Threshold detection at zero

6 Scilab versus Matlab

Matlab ist an den Universitäten wahrscheinlich das am meisten verbreitete wissenschatliche Rechenprogramm, in den letzten Jahren war auch seine Verbreitung im industriellen Bereich bemerkenswert.

Die ersten Versionen von Matlab erschienen Ende der 80er Jahre, seidem ist es zu einem sicher erprobten Produkt geworden und es ist im Wesentlichen fehlerfrei. Die Vorzüge bei der Anwendung von Matlab sind bemerkenswert:

- Es ist im universitären wissenschatlichen Bereich beinahe zum Standard geworden,
- es enhält in einer einzelnen Umgebung eine umfangreiche Bibliothek von Funktionen,
- die Community der Matlab-Anwender ist riesig, auf der Mathworks-Seite ist es z.B. möglich, auf unzählige Funktionen zuzugreifen, die von Anwendern für Anwender geschrieben wurden.

Es gibt auch Nachteile beim Gebrauch von Matlab:

- Es ist ein kommerzielles Programm, dessen Preis nicht banal ist,
- es ist mit einer großen Anzahl von zusätzlichen Toolboxen ausgestattet, die kostenpflichtig sind,
- der Code unterliegt dem Copyright und kann nur auf einem einzigen Rechner installiert werden.
- die Studenten können auf dem eigenen Rechner nicht damit üben, ohne dass sie eine eigene Lizenz erwerben oder den Code knacken,
- es kommt eine aggressive Verkaufsstrateige zum Einsatz, die immer wieder neue Versionen mit oft nur kosmetischen Modifikationen auf den Markt bringt, die die Kompatibilität mit früheren Versionen nicht immer garantieren.

Auch Scilab hat offensichtlich Vor- und Nachteile:

- Der Code ist kostenlos, jeder Anwender darf ihn ohne zu bezahlen oder Unrecht zu begehen installieren wo und wie er will,
- es befreit den Anwender von der Notwendigkeit ständiger Aktualisierung, deren Erfordernis oft fragwürdig ist,

mtlb_0	mtlb_a	mtlb_all	mtlb_any
mtlb_axis	mtlb_beta	mtlb_box	mtlb_close
mtlb_colordef	mtlb_conv	mtlb_cumprod	mtlb_cumsum
mtlb_dec2hex	mtlb_delete	mtlb_diag	mtlb_diff
mtlb_dir	mtlb_double	mtlb_e	mtlb_echo
mtlb_eig	mtlb_eval	mtlb_exist	mtlb_eye
mtlb_false	mtlb_fft	mtlb_fftshift	mtlb_find
mtlb_findstr	mtlb_fliplr	mtlb_fopen	mtlb_format
mtlb_fprintf	mtlb_fread	mtlb_fscanfb	mtlb_full
mtlb_fwrite	mtlb_grid	mtlb_hold	mtlb_i
mtlb_ifft	mtlb_imp	mtlb_int16	mtlb_int32
mtlb_int8	mtlb_is	mtlb_isa	mtlb_isfield
mtlb_isletter	mtlb_isspace	mtlb_l	mtlb_legendre
mtlb_linspace	mtlb_load	mtlb_logic	mtlb_logical
mtlb_lower	mtlb_max	mtlb_min	mtlb_more
mtlb_num2str	mtlb_ones	mtlb_plot	mtlb_prod
mtlb_rand	mtlb_randn	mtlb_rcond	mtlb_realmax
mtlb_realmin	mtlb_repmat	mtlb_s	mtlb_save
mtlb_setstr	mtlb_size	mtlb_sort	mtlb_strcmp
mtlb_strcmpi	mtlb_strfind	mtlb_strrep	mtlb_sum
mtlb_t	mtlb_toeplitz	mtlb_tril	mtlb_triu
mtlb_true	mtlbuint16	mtlb_uint32	mtlb_uint8
mtlb_upper	mtlb_zeros		

Tab. 6.1: In Scilab emulierte Matlab-Funktionen

- es ist im Wesentlichen ein Matlab-Clone, der Übergang von Matlab nach Scilab ist, auch wenn er nicht schmerzfrei ist, ziemlich einfach,
- außer dass es kostenlos ist, ist es auch noch *open source* mit allem Pro und Kontra dieser Entscheidung,
- Die Community der Scilab-Anwender hat verschiedene Softwarepakete und Handbücher zur allgemeinen Nutzung erstellt,
- es ist nicht frei von Fehlern,
- im Vergleich zu Matlab ist die Community reduziert, nicht triviale Probleme finden nicht immer sofort eine Lösung,
- in diversen Punkten scheint es nicht ausgereift, z.B. ist die Syntax einiger *ähnlicher* Befehle voneinander völlig *verschieden*, allgemein ist die Syntax einiger Befehle zu *maschinennah* und könnte mit wenig Aufwand mehr *benutzerfreundlich* gemacht werden.

Scilab stellt eine Reihe von Funktionen bereit, die das Verhalten von Matlab-Funktionen *emulieren*. Diese Funktionen haben das Präfix `mtlb_` und sind in Tabelle 6.1 aufgelistet. Auch steht ein Tool zur Verfügung, um die automatische Übersetzung von Matlab-Code zu versuchen.

Die Operatoren sind in Matlab und Scilab gleich, doch ist die Semantik etwas anders, z.B. ergibt

$$A > B$$

mit leeren Matrizen A und B eine leere Matrix in Matlab und einen Fehler in Scilab. Sind A und B komplexe Matrizen, arbeitet Matlab mit dem Realteil, Scilab gibt einen Fehler aus. Die größten Unterschiede gibt es bei Stringvariablen.

Der Typ `string` wird in Matlab und Scilab auf besonders unterschiedliche Weise behandelt. Matlab betrachtet z.B. `'mystring'` als Kette von Zeichen, während das in Scilab zu `[m,y,s,t,r,i,n,g]` wird. In Scilab ist eine Zeichenkette also ein Objekt vom Typ `string` mit der Dimension 1×1, während sie in Matlab ein Vektor ist, dessen Elemente die Zeichen sind. Ein kleiner Vorteil von Scilab ist die Möglichkeit, Zeichenketten unterschiedlicher Länge zu stapeln, was in Matlab nur geht, wenn eine `cell of character strings` definiert wird.

Man muss präzisieren, dass bei gleichen Bezeichnungen für Befehle nicht nur die Syntax anders sein kann, sonder bei gleicher Syntax kann auch die Semantik verschieden sein.

Die Seite
http://help.scilab.org/docs/5.4.0/en_US/section_36184e52ee88ad558380be4e92d3de21.html
enthält ein nützliches Wörterbuch und man muss nur auf eine Matlab-Funktion klicken, um Zugang zur äquivalenten Scilab-Funktion zu bekommen, oder, wenn ein Äquivalent nicht existiert, Zugang zu Code-Zeilen, die implementiert werden können, um ein äquivalentes Verhalten zu bekommen.

7 Free Documentation Licence

Version 1.3, 3 November 2008.

1. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document „free“ in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of „copyleft“, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The „Document“, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as „you“. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A „Modified Version“ of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language. A „Secondary Section“ is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The

relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The „Invariant Sections“ are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The „Cover Texts“ are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words. A „Transparent“ copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not „Transparent“ is called „Opaque“.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machinegenerated HTML, PostScript or PDF produced by some word processors for output purposes only.

The „Title Page“ means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, „Title Page“ means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text. The „publisher“ means any person or entity that distributes copies of the Document to the public.

A section „Entitled XYZ“ means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as „Acknowledgements“, „Dedications“, „Endorsements“, or „History“.) To „Preserve the Title“ of such a section when you modify the Document means that it remains a section „Entitled XYZ“ according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

3. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

4. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

5. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled „History“, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled „History“ in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the „History“ section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled „Acknowledgements“ or „Dedications“, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled „Endorsements“. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled „Endorsements“ or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled „Endorsements“, provided it contains nothing but endorsements of your Modified Version by various parties - for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard. You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled „History“ in the various original documents, forming one section Entitled „History“; likewise combine any sections Entitled „Acknowledgements“, and any sections Entitled „Dedications“. You must delete all sections Entitled „Endorsements“.

7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an „aggregate“

if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled „Acknowledgements“, „Dedications“, or „History“, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been

terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License „or any later version“ applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

12. RELICENSING

„Massive Multiauthor Collaboration Site“ (or „MMC Site“) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A „Massive Multiauthor Collaboration“ (or „MMC“) contained in the site means any set of copyrightable works thus published on the MMC site.

„CC-BY-SA“ means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

„Incorporate“ means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is „eligible for relicensing“ if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant

Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled „GNU Free Documentation License“.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the „with ... Texts.“ line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software