

Thuselt, Frank

**Research Report**

## Das Arbeiten mit Numerik-Programmen: MATLAB, Scilab und Octave in der Anwendung

Beiträge der Hochschule Pforzheim, No. 129

**Provided in Cooperation with:**

Hochschule Pforzheim

*Suggested Citation:* Thuselt, Frank (2009) : Das Arbeiten mit Numerik-Programmen: MATLAB, Scilab und Octave in der Anwendung, Beiträge der Hochschule Pforzheim, No. 129, Hochschule Pforzheim, Pforzheim,  
<https://nbn-resolving.de/urn:nbn:de:bsz:951-opus-565>

This Version is available at:

<https://hdl.handle.net/10419/97597>

**Standard-Nutzungsbedingungen:**

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

**Terms of use:**

*Documents in EconStor may be saved and copied for your personal and scholarly purposes.*

*You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.*

*If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.*



<https://creativecommons.org/licenses/by-nc-nd/4.0/deed.de>

**BEITRÄGE DER HOCHSCHULE PFORZHEIM**

---

**Frank Thuselt**

**Das Arbeiten mit Numerik-Programmen**

**MATLAB, Scilab und Octave in der Anwendung**

---

**Nr. 129**

**Herausgeber:** Prof. Dr. Ansgar Häfner, Prof. Dr. Norbert Jost, Prof. Dr. Karl-Heinz Rau, Prof. Dr. Roland Scherr, Prof. Dr. Christa Wehner, Prof. Dr. Hanno Beck (geschäftsführend; Hanno.beck@hs-pforzheim.de)

**Sekretariat:** Frau Alice Dobrinski  
Hochschule Pforzheim  
Tiefenbronner Str. 65  
75175 Pforzheim  
[alice.dobrinski@fh-pforzheim.de](mailto:alice.dobrinski@fh-pforzheim.de)  
Telefon: 07231/28-6201  
Telefax: 07231/28-6666

**Ausgabe:** Juni 2009

**Frank Thuselt**

**Das Arbeiten mit Numerik-Programmen**

**MATLAB, Scilab und Octave in der Anwendung**

Diese Arbeit wurde gefördert durch die Firma

THALES Defence GmbH, Pforzheim

Prof. Dr. Frank Thuselt  
Tiefenbronner Str. 65  
75175 Pforzheim  
[Frank.Thuselt@hs-pforzheim.de](mailto:Frank.Thuselt@hs-pforzheim.de)

Frank Thuselt ist Professor für

## Inhaltsverzeichnis

1. Einleitung.....	7
2. Kommerzielle Mathematik-Programme: MATLAB im Vergleich mit Maple und Mathematica.....	7
3. Numerik-Programme. Grundsätzliches zu MATLAB, Scilab und Octave .....	9
3.1. MATLAB .....	9
3.2. Scilab .....	11
3.3. Octave .....	11
4. Einstieg in MATLAB, Scilab und Octave .....	12
4.1. Installation der Programme.....	12
4.1.1. MATLAB .....	12
4.1.2. Scilab.....	12
4.1.3. Octave .....	13
4.2. Arbeiten auf Kommandozeilenebene (Taschenrechner-Funktion).....	14
4.2.1. Einfache Operationen mit Zahlen und Variablen .....	15
4.2.2. Einige grundlegende Eigenschaften .....	17
4.3. Einfache Vektoren und Matrizen .....	20
4.4. Rechnen mit komplexen Zahlen .....	22
4.5. Graphik .....	24
4.6. Matrizenalgebra und Polynome .....	28
5. Script-Dateien und Funktionen .....	36
5.1. Script-Dateien .....	36
5.1.1. Grundsätzliches .....	36
5.1.2. Einrichten des Arbeitsverzeichnisses in MATLAB/Octave .....	37
5.1.3. Einrichten des Arbeitsverzeichnisses in Scilab .....	37
5.1.4. Einfaches Beispiel in MATLAB .....	38
5.2. Funktionen .....	39

5.2.1. Allgemeines über Funktionen .....	39
5.2.2. Funktionsaufrufe .....	40
5.2.3. Funktionen von Funktionen .....	42
6. Kontrollstrukturen .....	45
6.1. Die if-Bedingung .....	45
6.2. Die switch-Bedingung .....	46
6.3. Die for-Schleife .....	47
6.4. Die while-Schleife .....	50
7. Einfache Benutzer-Interfaces (GUI) .....	53
7.1. Dialogbox „menu“ .....	53
7.2. Eingabe-Listen „listdlg“ und „xchoose“ .....	55
8. Arbeitsgeschwindigkeit der einzelnen Programme .....	57
9. Simulink und Scicos .....	57
10. Erfahrungen aus Lehre und Ausbildung, Forschung und Entwicklung .....	67
11. Literaturverzeichnis .....	71

## **Zusammenfassung**

Innerhalb der Mathematik-Software nehmen die Numerik-Programme MATLAB, Scilab und Octave eine herausragende Rolle ein. In der vorliegenden Arbeit werden sie mit solchen Programmen verglichen, die sich vorwiegend der Computeralgebra widmen, wie Mathematica oder Maple. Dabei betrachten wir die grundsätzlichen Eigenschaften, das Installationsverhalten und die ersten Schritte des täglichen Gebrauchs beim Arbeiten mit den Numerikprogrammen. Dies geschieht am Beispiel von komplexen Zahlen, von Graphik und von Polynomen. Die Verwendung von Script-Files und Funktionen wird erklärt. Anhand zahlreicher Beispiele stellen wir dabei MATLAB/Octave und Scilab direkt gegenüber. Zusätzlich vergleichen wir einige Aspekte von Simulink mit Scicos. Am Schluss wird über Erfahrungen in Entwicklung und Ausbildung berichtet.

## **Summary**

Within mathematical software the numerical programs MATLAB, Scilab, and Octave play an outstanding role. In this paper, they shall be compared with programs that are mostly dedicated to computer algebra, as Mathematica or Maple. Moreover, their basic features, their installation behaviour, and first steps of everyday use are considered. For this purpose, complex numbers, graphics, and polynoms are used as examples. Besides, the application of script files and functions will be explained. In a large number of examples, MATLAB and Octave are directly opposed to Scilab. Additionally, some aspects of Simulink and Scicos are discussed. Finally, we report on some experiences in engineering development and education.



## 1. Einleitung

In den Natur- und Ingenieurwissenschaften ist die Benutzung mathematischer Hilfsmittel unumgänglich. Hierzu zählen insbesondere Programme zum Verarbeiten numerischer Daten. Sie werden meist zusammenfassend kurz als Mathematikprogramme oder Numerik-Programme bezeichnet. Diese Bezeichnung ist allerdings nicht für alle Programme zutreffend. Richtiger müsste man unterscheiden zwischen echten *numerischen Programmen* und so genannten *Computeralgebra-Systemen*. Neben den Marktführern MATLAB, Maple und Mathematica<sup>1</sup> gibt es unter ihnen noch eine Vielzahl weiterer Programme, wie MathCAD, DERIVE, MACSYMA<sup>2</sup> usw., deren Bedeutung allerdings bei weitem nicht so umfassend ist.

Schließlich existieren noch kleinere Programmpakete, die das Ziel haben, zu günstigeren Konditionen, zum Beispiel als Freeware oder Open Source, die wichtigsten Funktionen der „großen“ Programme ebenfalls anzubieten.

Im Folgenden soll das Programmpaket MATLAB im Vergleich mit zwei der kleineren Programme betrachtet werden, insbesondere hinsichtlich seiner Eignung für die studentische Ausbildung. Die Vergleichskandidaten sind dabei Octave und Scilab.

## 2. Kommerzielle Mathematik-Programme: MATLAB im Vergleich mit Maple und Mathematica

Nach Benker [1] unterscheidet man bei integrierten Softwaresystemen *Computeralgebra-Systeme* (CAS) und *Numerik-Systeme*.

Als *Computeralgebra-Systeme* bezeichnet man ein Computerprogramm, das Rechenaufgaben aus verschiedenen Bereichen der Mathematik lösen und dabei nicht nur (wie ein Taschenrechner) mit Zahlen, sondern auch vor allem mit symbolischen Ausdrücken (Variablen, Funktionen, Matrizen) umgehen kann. Dazu gehören neben Maple und Mathematica weiterhin Mathcad, Derive, Axiom und MuPAD<sup>3</sup>.

Die Grenzen zwischen den Systemen sind jedoch aufgeweicht, da Systeme der einen Gruppe auch Merkmale der jeweils anderen Gruppe beinhalten.

Zu den echten Numerik-Programmen gehört an erster Stelle MATLAB. Es hat seinen dominierenden Platz in den Ingenieurwissenschaften eingenommen. Weitere anspruchsvolle Programme sind Mathematica und Maple. Mathematica und Maple waren ursprünglich *Computeralgebra-Systeme*, inzwischen haben sie jedoch wie MATLAB auch weitgehend

---

<sup>1</sup> *Hinweis:* MATLAB<sup>®</sup> ist ein eingetragenes Warenzeichen von The MathWorks, Inc., MAPLE<sup>®</sup> ist ein eingetragenes Warenzeichen von Waterloo Maple Inc., Mathematica<sup>®</sup> ist ein eingetragenes Warenzeichen von Wolfram Research, Inc.

<sup>2</sup> Die Namen sind teilweise eingetragene Markenzeichen ihrer jeweiligen Hersteller.

<sup>3</sup> MuPAD wurde in Deutschland entwickelt. und in der Zwischenzeit von The MathWorks

numerische Funktionen übernommen, während umgekehrt MATLAB durch Lizenzübernahme des Computeralgebra-Pakets Maple Algebra-Funktionen integriert hat.

Algebraische Aufgaben im engeren Sinne sind:

- algebraische Ausdrücke vereinfachen und vergleichen
- algebraische Gleichungen lösen
- lineare Gleichungssysteme lösen und Matrizenberechnung durchführen
- Funktionen differenzieren und integrieren
- mit Dezimalzahlen mit beliebiger Genauigkeit rechnen (mit einem guten CAS kann man z. B. mit geringem Programmieraufwand die Zahl  $\pi$  (pi) auf zehntausende Nachkommastellen genau bestimmen)

Zu den Aufgaben von Numerik-Programmen gehören zum Beispiel:

- Lösen von Integralen und Differentialgleichungen durch numerische Integration („Quadratur“)
- Lösen von linearen Gleichungssystemen beliebig hoher Dimension
- Lösen von nichtlinearen Gleichungen und Gleichungssystemen
- Bestimmung von Eigenwerten und Eigenvektoren
- Optimierung und Simulation komplizierter Zusammenhänge
- Signalanalyse und -verarbeitung
- Graphische Darstellung von Funktionen und Daten in zwei oder drei Dimensionen
- Bereitstellung einer Benutzerschnittstelle für das Einbinden eigener Algorithmen in einer höheren Programmiersprache

In der Regel arbeiten alle Systeme mit Interpreter. Interpreter sind zwar oft langsamer, aber dafür einfacher zu bedienen als Compiler.

In einer Artikelserie der IEEE-Zeitschrift *Computer Science and Engineering* ([2] bis [5]) wurden die drei Marktführer unter den Mathematikprogrammen MATLAB, Maple und Mathematica einem umfangreichen Vergleichstest unterzogen, vor allem hinsichtlich ihres Einsatzes in der Ausbildung. Viele Ergebnisse sind jedoch auch für andere Situationen aufschlussreich. Folgende Aufgabenfelder und Ziele sind darin formuliert worden [4]:

#### 1. Simulation

- Studenten sollen befähigt werden, in kleinen Gruppen als Design- und Entwicklungsteams zusammenzuarbeiten
- Medium für Lehrkräfte zur Interaktion mit Studenten „just in time“
- Möglichkeit für zusätzliches Experimentieren durch besonders motivierte Studenten

## 2. Tutorium

- Studenten sollen befähigt werden, individuell zu arbeiten
- Medium für interaktives Lernen

## 3. Computerprogrammierung

- Studenten sollen befähigt werden, individuell oder in Gruppen zu arbeiten
- Medium für interaktives Erlernen des Designs, der Implementierung und des Tests von Computeralgorithmen
- Möglichkeit für projektbezogenes Experimentieren (z.B. Sammlung von Algorithmen, Eingabe- und Ausgabemechanismen) für umfangreichere Anwendungen

Der Vergleich der Entwicklungsumgebungen läuft auf folgende Kernaussagen hinaus [5]:

Während Mathematica und Maple mit Standard-Benutzeroberflächen (GUIs) einschließlich Symbolleisten und Pull-down-Menüs ausgestattet sind – die Paletten lassen sich in Mathematica sogar verschieben –, bietet MATLAB hingegen lediglich eine Kommandozeilen-Oberfläche in einem von mehreren Fenstern (Kommandozeile, Kommando-Stack und Directory tree). Beispielsweise steht bei Maple für die Quadratwurzel ein Button mit dem Wurzelsymbol zur Verfügung, welches im Editorfeld automatisch eine Codezeile, wie z.B. `sqrt()` erzeugt. Erst in der neuesten Version R2008b hat sich bei MATLAB die Benutzerfreundlichkeit etwas verbessert.

Die einzelnen Programme bedienen verschiedene Zielgruppen in unterschiedlicher Weise. So ist zum Beispiel Mathematica bei Physikern sehr beliebt wegen der zahlreichen Tools, die spezielle physikalische Fragestellungen unterstützen. MATLAB hingegen hat sich im ingenieurwissenschaftlichen Bereich als nahezu alternativlos etabliert, insbesondere wenn es gilt, anspruchsvolle Aufgaben und Simulationen auf den Gebieten der Signalverarbeitung und der Regelungstechnik zu bewältigen. Da sich die Syntax zwischen den drei Programmen erheblich unterscheidet, bleiben jedoch Anwender in der Regel bei dem Programm, das sie einmal kennengelernt haben.

## 3. Numerik-Programme. Grundsätzliches zu MATLAB, Scilab und Octave

Im Folgenden beschränken wir uns auf die eigentlichen Numerik-Programme. Heute sind zumindest drei solcher Numerik-Programme verfügbar, deren Arbeitsweise etwa vergleichbar ist: MATLAB, Scilab und Octave. Der große Vorteil beim Arbeiten mit diesen Numerik-Programmen ist, dass gleichzeitig mit ganzen Zahlengruppen, das heißt Matrizen, operiert werden kann. Bei MATLAB drückt sich dies auch im Namen aus: MATLAB ist die Abkürzung für Matrix Laboratory.

### 3.1. MATLAB

MATLAB wurde seit 1984 von der Firma The Mathworks, Inc. entwickelt und als kommerzielle Software vor allem für Windows und Linux-/Unixrechner vertrieben. Seit Version 6.5 existiert auch eine Version für den Macintosh. MATLAB ist heute unangefochtener Marktführer im Bereich der Numerik-Software. The MathWorks mit Sitz in Natick, Massachusetts, hat heute weltweit über 2000 Mitarbeiter [6].

Ursprünglich wurde MATLAB als Bedienerschnittstelle für den Zugang zu FORTRAN-Programmen entwickelt und auch in FORTRAN programmiert. Seit langer Zeit allerdings wurde der Code in der schnelleren Sprache C implementiert.

Die folgenden Grundfunktionen kennzeichnen die typischen Merkmale von MATLAB:

- Formalismus für schnelle numerische Berechnungen,
- Graphikfunktionen zur Visualisierung und Datenanalyse,
- interaktive Sprache und Programmierumgebung,
- Unterstützung des Datenimports aus Dateien und externen Geräten,
- Integrationsmöglichkeit von externen Anwendungen, die mit anderen Sprachen entwickelt wurden, vor allem Fortran, aber auch C, C++, Java und Excel,
- Kompilierungsmöglichkeit nach C und C++,
- Tool zur Entwicklung graphischer Benutzeroberflächen (GUIs).

MATLAB ist ein Grundprogramm, zu welchem insgesamt über 100 verschiedene ergänzende Toolboxen angeboten werden. Eine der wichtigsten ist Simulink<sup>4</sup>. Dabei handelt es sich um ein graphikorientiertes Programm zum symbolischen Lösen von Differentialgleichungen. Ein wichtiges Strukturelement sind Graphik-Boxen für Übertragungsfunktionen, die sich an der Darstellung der Laplace-Transformierten orientieren und sich wie diese benutzen lassen. Etwa 40 der 100 MATLAB-Toolboxen sind Ergänzungen zu Simulink.

Für symbolische Berechnungen, zum Beispiel zur Formelmanipulation, bietet MATLAB eine eigene Toolbox an, die wesentliche Elemente des Computeralgebra-Systems Maple enthält und von der Firma Maplesoft dazugekauft wurde.

Trotz der hohen Kosten von MATLAB (Einzelplatzversion incl. Simulink/Symbolic für Hochschulen ca. 1.000 \$, mit allen Toolboxen über 20.000 \$ für private Nutzer, als Studentenversion ca. 60 \$) konnte sich bisher Scilab nicht in gewünschtem Maße eine Nutzergemeinde erobern.

Um eine Vorstellung von der Leistungsfähigkeit zu vermitteln, sollen einige der MATLAB-Toolboxen, die für technische Anwendungen von Bedeutung sind, genannt werden:

- Parallel Computing Toolbox
- MATLAB Distributed Computing Server
- Math and Optimization (Optimization, Partial Differential Equation, Genetic Algorithm and Direct Search),
- Statistics and Data Analysis (Statistics, Neural Networks, Curve Fitting, Splines, Model-Based Calibration),
- Control System Design and Analysis (Control Systems, System Identification, Fuzzy Logic, Robust Control, Model Predictive Control, Aerospace)
- Signal Processing and Communications,
- Image Processing,
- Test & Measurement (Data Acquisition, Instrument Control...),
- Application Deployment (MATLAB Compiler, Spreadsheet Link EX für Microsoft Excel).

Zur Simulink-Produktfamilie gehören zum Beispiel:

- Fixed-Point Modeling
- Event-Based Modeling
- Physical Modeling
- Simulation Graphics
- Control System Design and Analysis
- Signal Processing and Communications
- Code Generation

---

<sup>4</sup> Simulink<sup>®</sup> ist ein eingetragenes Warenzeichen von The MathWorks, Inc.

Die aktuelle Version ist R2008b (Version 7.7 mit Simulink Version 7.2). Die Neuerungen in der MATLAB/Simulink-Version 2008b betreffen vor allem folgendes:

- Neuer Browser,
- Erzeugung von Parallel Computing-Anwendungen mit dem Compiler,
- Zugriff auf die symbolische MuPAD-Engine und die MuPAD-Sprache direkt von MATLAB aus (Maple ist jedoch weiterhin verfügbar).
- Physical Modeling in Simulink durch MATLAB-basierte Sprache in Simscape,
- SimElectronics - ein neues Produkt zur Modellierung und Simulation elektronischer und elektromechanischer Systeme.

### **3.2. Scilab**

Das Numerik-Programm Scilab [7] wurde ursprünglich etwa mit Beginn der 1980er Jahre im französischen INRIA (Institut National de Recherche en Informatique et en Automatique) unter dem Namen BASIL entwickelt. Ungefähr um 1990 wurde die Arbeit nach einer längeren Unterbrechung gemeinsam mit der Elitehochschule ENPC (École Nationale des Ponts et Chaussées) wieder aufgenommen. Seit 2003 hat diese Arbeit das Scilab Consortium übernommen. Scilab ist heute ein freies wissenschaftliches Programmpaket. Es arbeitet sowohl auf Unix/Linux-Rechnern als auch auf Windows-PCs. Analog zu Simulink existiert auch zu Scilab ein graphisches Paket zur Lösung von Differentialgleichungen. Es ist unter dem Namen Scicos bereits Bestandteil des Scilab-Programms. Für Scilab gibt es analog zu MATLAB zahlreiche verschiedene Toolboxen, hier Module genannt. An einer Möglichkeit zur Darstellung und Simulation elektronischer Systeme wird ebenfalls bereits seit längerer Zeit gearbeitet. Sie basiert auf einer Zusammenarbeit mit der Modelica Association, einer nichtkommerziellen Vereinigung mit Sitz in Linköping [8]. Die aktuelle Version von Scilab ist die Version 5.1, dazu gehört Scicos 4.2.1. In dieser Kombination gibt es jedoch Probleme mit einigen Graphikkarten beim Aufrufen von Scicos. Wahrscheinlich auch deshalb wurde Scicos innerhalb der alten Scilab-Version 4.3 von einigen Mitgliedern des INRIA/ENPC-Teams noch weiter entwickelt und seit Dezember 2008 unter dem Namen ScicosLab 4.3 zusammen mit Scicos 4.3 als eigenes Paket angeboten. Diese Version wird vom Scilab-Konsortium jedoch nicht unterstützt.

Scilab ist eine freie Software und entsprach ursprünglich der CeCILL (CEA CNRS INRIA Logiciel Libre), das ist eine an der GPL (GNU General Public License) orientierte französische Lizenzvereinbarung. Inzwischen hat sich Scilab den GPL-Richtlinien angeschlossen.

### **3.3. Octave**

Die Entwicklung von Octave begann im Jahre 1992 unter der Leitung von John W. Eaton, Computeradministrator an der Universität von Wisconsin-Madison. Die Entwicklung baute auf einem Programm auf, das zuvor zwei Studenten für Berechnungen chemischer Prozesse geschrieben hatten. Ausdrückliches Ziel ist, anders als bei Scilab, eine weitgehende Kompatibilität zu MATLAB [6],[10].

Octave wurde eigentlich für LINUX-Systeme entwickelt. Darüber hinaus gibt es LINUX-Emulationen, sogenannte shells, damit ist Octave auch unter anderen Betriebssystemen lauffähig, zum Beispiel unter Mac OS X, Sun Solaris und Windows [11]. Die aktuelle Octave-Version ist 3.0.3. Octave unterliegt der GNU General Public License (GPL) und ist damit Freeware bzw. Open Source Software. Diese Lizenzrichtlinien werden von der Free Software Foundation, Inc. verwaltet.

Octave beinhaltet mehrere Packages – sie entsprechen weitgehend den Toolboxen in MATLAB. Die Packages werden unter dem Namen Octave-Forge als gemeinsame Datei angeboten. Eine Simulink-ähnliche Toolbox ist jedoch nicht vorhanden.

## 4. Einstieg in MATLAB, Scilab und Octave

### 4.1. Installation der Programme

#### 4.1.1. MATLAB

MATLAB ist direkt von der gelieferten CD installierbar, aber auch durch Download erhältlich. Seit dem Release R2008a ist für alle MATLAB- und Simulink-Produkte eine Online-Aktivierung erforderlich.

Dadurch wird leider die gleichzeitige Installation auf PC und Notebook verhindert. Für Privatanwender ist das ein erheblicher Nachteil, denn sie müssten dafür gleich zwei Lizenzen erwerben.

Die MATLAB-Dokumentation und Hilfe existiert im PDF- und HTML-Format, beide sind zusätzlich auch online verfügbar. Darüber hinaus erhält man Hilfe auf Kommandozeilenebene über den Befehl `help (name)`. Die MATLAB-Hilfdateien benutzen wie das Programm selbst grundsätzlich Englisch.

Im Gegensatz zu MATLAB ist gedruckte Literatur zu Scilab und Octave nur in geringem Umfang verfügbar. Zu Scilab sind die Bücher von Campbell et al. [12] sowie Urroz [13],[14] zu erwähnen, für Octave existiert das Manual von Eaton [6]. Einen Vergleich von Scilab mit MATLAB hinsichtlich der Programmierung findet man zum Beispiel im Internet unter [15].

#### 4.1.2. Scilab

Für die Installation von Scilab muss zuerst eine ca. 79 MB große Installationsdatei aus dem Internet heruntergeladen werden. Scilab läuft auf GNU/Linux und Windows/Vista, und zwar auch als 64-Bit-Version.

Die Installation der neusten Version Scilab Version 5.1 [7] dauert merklich länger als die der Vorgängerversion. Eine Hilfe als HTML-Dokument wird ebenfalls wie bei MATLAB mitinstalliert. Die Scilab-Hilfe ist in Englisch oder Französisch zu benutzen. Eine ausführliche Dokumentation kann über die Scilab-Webseite heruntergeladen werden. Im Gegensatz zu MATLAB wird mit dem Befehl `help (name)` keine Hilfe auf Kommandozeilenebene geliefert, sondern der Help-Browser (als HTML-Hilfe) aufgerufen.

Da Scilab in vieler Hinsicht nicht vollständig kompatibel zu MATLAB ist, kann zusätzlich die Bibliothek `plotlib` installiert werden [16]. Sie enthält vor allem Graphikfunktionen, die der Anwender beim Umstieg von MATLAB sonst schmerzlich vermisst. Allerdings existiert hierzu kaum eine Hilfe.

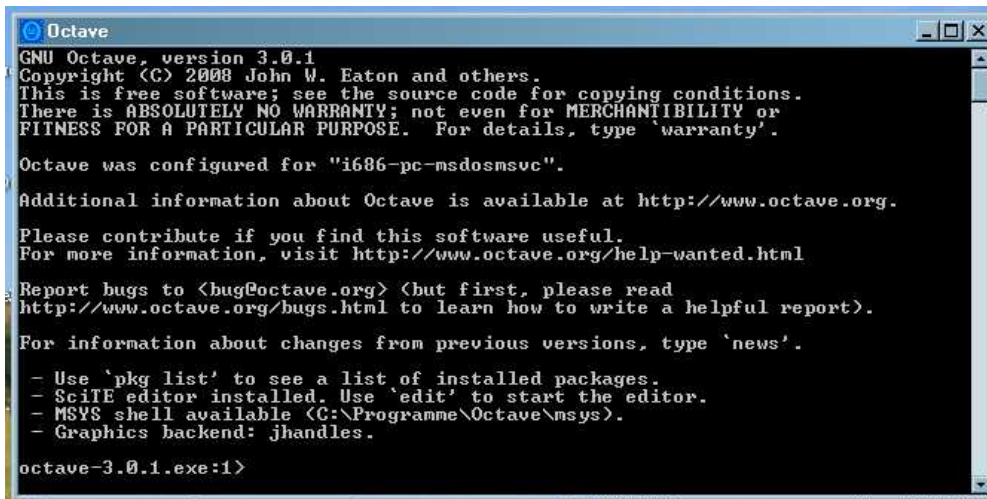
Unter Scilab existieren automatische Konvertierungstools für die Umwandlung von MATLAB-Programmen. Es wird dringend davon abgeraten, diese zu benutzen. Sie sind sehr fehlerträchtig

und verlangen erhebliche Nacharbeit. Mit einiger Erfahrung hat man MATLAB-Programme schneller selbst in die benötigten Scilab-Routinen umgewandelt.

Scilab unterstützt im Gegensatz zu MATLAB mehrere parallel laufende Instanzen.

#### 4.1.3. Octave

Zur Installation von Octave auf einem Windows-PC gibt es zwei Möglichkeiten. In einem Fall benötigt man Cygwin als Zusatzsoftware. Es lädt beim Start von Octave eine unsichtbar im Hintergrund arbeitende Linux-Emulation. Zunächst läuft das Programm nur im DOS-Fenster, zu erreichen unter der „Eingabeaufforderung“ von Windows (Abbildung 1). Für die Graphik bietet Octave zwei verschiedene Varianten an: *GNUplot* (das ist die ältere und stabile Version) und *JHandles* (es soll bessere 3D-Graphikfunktionen enthalten, befindet sich aber derzeit noch in der Entwicklung). Die integrierte Hilfe und ein Handbuch werden mitinstalliert. Unter Linux ist sogar eine einfache graphische Benutzeroberfläche verfügbar.



```
Octave
GNU Octave, version 3.0.1
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i686-pc-msdosmsvc".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).

For information about changes from previous versions, type 'news'.

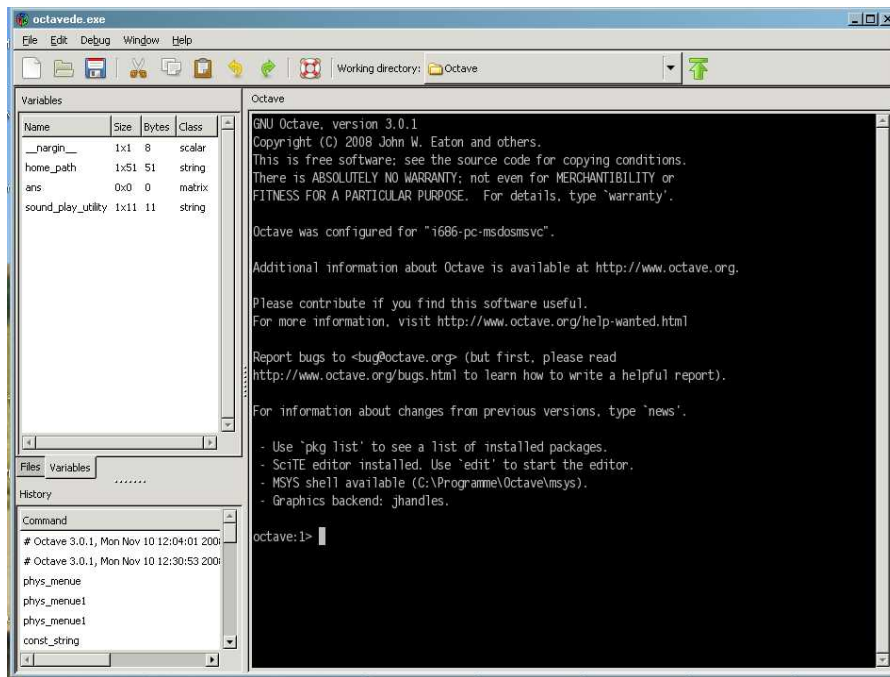
- Use 'pkg list' to see a list of installed packages.
- SciTE editor installed. Use 'edit' to start the editor.
- MSYS shell available (C:\Programme\Octave\msys).
- Graphics backend: jhandles.

octave-3.0.1.exe:1>
```

Abbildung 1 Octave unter DOS

Eine alternative Installationsmöglichkeit geschieht über das Paket Octave-Forge (54 MB). Für dessen Installation wird Cygwin nicht benötigt. Im Rahmen des Octave-Forge-Projekts ist ein verbesserter Zugriff auf Toolboxen analog zu denen von MATLAB möglich. Momentan wird für diese Umgebung an einer graphischen Benutzeroberfläche („Octave UI“) für Windows gearbeitet (Abbildung 2). Sie erinnert an die von Benutzeroberfläche MATLAB, ist jedoch bei weitem nicht so leistungsfähig.

Leider ist die Installation dieser graphischen Benutzeroberfläche nicht auf allen Umgebungen zuverlässig durchführbar. Auf einem Notebook konnte sie beispielsweise nicht verwendet werden. Gleiches trifft übrigens für die neueste Version von Scicos unter Scilab 5.0.2 zu. Offensichtlich kommen beide Programme mit einigen Graphikkarten nicht zurecht.



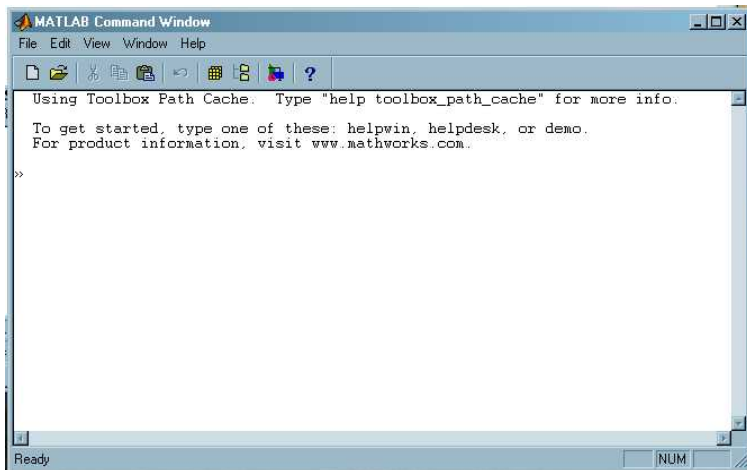
**Abbildung 2** Octave UI mit graphischer Benutzeroberfläche

#### 4.2. Arbeiten auf Kommandozeilenebene (Taschenrechner-Funktion)

Die einfachste Anwendung eines Numerik-Programms ist seine Verwendung als komfortabler Taschenrechner. Was macht man mit einem Taschenrechner? Man gibt auf seiner Tastatur eine mathematische Formel ein und bekommt das Ergebnis auf dem Display angezeigt. Ein PC hat ebenfalls eine Tastatur und sogar ein viel komfortableres Display als der Taschenrechner. Warum also nicht gleich auf dem PC rechnen?

Mit MATLAB, Octave oder Scilab lässt sich all das erledigen, was mit einem Taschenrechner auch bearbeitet werden soll. Allerdings gehört auch die nahezu sofortige Verfügbarkeit dazu. Die ist bei den neueren Versionen (5.1) von Scilab und vor allem von MATLAB (ab Version 6) nicht mehr gegeben. Die Ursache ist darin zu suchen, dass die Java-Laufzeitumgebung eine längere Ladezeit benötigt. Lediglich die frühere MATLAB-Version 5.3 war praktisch sofort nach dem Aufrufen arbeitsfähig. Mit der Startoption `matlab.exe -nojvm` lässt sich eine solche „abgespeckte“ Version jedoch auch in den höheren MATLAB-Versionen noch laden. In Version 6.5 war mit diesem Befehl die gleiche Kommandooberfläche zu sehen wie in der vorhergehenden Version 5.3 (vgl. Abbildung 3), in der späteren Version 7.5 zum Beispiel erscheint leider nur noch eine ganz rudimentäre Form ohne jegliche Buttons. Damit entfällt leider auch die Möglichkeit, zum Beispiel Abbildungen zu kopieren. Auch der Editor und der Suchpfad können nicht mehr von der Kommandooberfläche aus aufgerufen werden. Innerhalb der Scilab-Familie ist Scicos-Lab ebenfalls sofort präsent, im Gegensatz zum eigentlichen Scilab 5.3.





**Abbildung 3** Kommandooberfläche von MATLAB 5.3 und MATLAB 6.5, unter 6.5 aufgerufen mit Option -nojvm

Die vorherrschende Arbeitsweise ist bei MATLAB, Scilab und Octave das Arbeiten auf der Ebene der Kommandozeile. Nach dem Prompt wird der gewünschte Befehl eingegeben. Das Prompt-Zeichen wird bei Scilab durch das Symbol

-->

dargestellt, bei MATLAB durch

>>

und bei Octave durch

octave-3.0.1.exe:x>

Die Variable x zählt in Octave aufwärts mit jeder Eingabe.

#### 4.2.1. Einfache Operationen mit Zahlen und Variablen

Im Folgenden werden einige der einfachsten Befehle für MATLAB (ähnlich bei Octave) und Scilab anhand von Beispielen dargestellt und verglichen. Da die Octave-Syntax in vielen Fällen mit der von MATLAB kompatibel ist, werden die Octave-Befehle nur dort besonders erwähnt, wo sie sich von denen von MATLAB unterscheiden. Solche Fälle gibt es durchaus. Octave ist sehr viel mehr an der C-Syntax orientiert als MATLAB. Beispielsweise kennt Octave die Befehle `a++` oder `++a` zum Weiterzählen von Variablen, die in MATLAB nicht implementiert sind.

MATLAB, Octave	Scilab
<pre>&gt;&gt; 3+4 ans =      7 &gt;&gt; pi ans =   3.1416 &gt;&gt; pi/4 ans =   0.7854 &gt;&gt; sin(pi/4)</pre>	<pre>--&gt;3+4 ans =      7. --&gt;%pi %pi =   3.1415927 --&gt;%pi/4 ans =   0.7853982 --&gt;sin(%pi/4)</pre>

MATLAB, Octave	Scilab
<pre>ans =     0.7071 » pi/4 ans =     0.7854 » sin(ans) ans =     0.7071 » sqrt(2) ans =     1.4142</pre> <p>Bestimmte Schlüsselwörter (keywords) sind den in MATLAB vordefinierten Variablen vorbehalten. Hierzu gehören zum Beispiel <code>pi</code> oder die imaginäre Einheit <code>i</code> bzw. <code>j</code>. Diese Schlüsselwörter können jedoch überschrieben werden, was leider öfter Anlass zu Fehlern gibt.</p> <p>Weiterzählen einer Variablen (funktioniert nur in Octave, nicht unter MATLAB oder Scilab):</p> <pre>octave:17&gt; x=1 x = 1 octave:18&gt; x++ ans = 1 octave:19&gt; x++ ans = 2 octave:20&gt; x++ ans = 3 octave:21&gt; x++ ans = 4 octave:22&gt; █</pre>	<pre>ans =     0.7071068 --&gt;sin(ans) --&gt;%pi/4 ans =     0.7853982 --&gt;sin(ans) ans =     0.7071068 --&gt;sqrt(2) ans =     1.4142136</pre> <p>Konstanten werden in Scilab mit vorangestelltem Prozentzeichen dargestellt, z.B: <code>%pi</code>, <code>%eps</code>. Damit wird die in MATLAB mögliche versehentliche Neudefinition dieser Variablen ausgeschlossen.</p>

Grundsätzlich arbeitet man bei den drei betrachteten Programmen auf der Ebene der Kommandozeile, im sogenannten *Workspace*. Die Ergebnisse der Berechnungen sind in der Variablen `ans` (steht für *answer*) gespeichert. Der Inhalt dieser Variablen wird also mit jeder neuen Eingabe immer wieder überschrieben. Mit dem aktuellen Inhalt können Sie auch weiter arbeiten, beispielsweise `ans/4` berechnen. Ein Semikolon nach der Eingabe verhindert das Echo auf der nächsten Zeile der Konsole. Das ist besonders wichtig bei Variablen, die aus sehr vielen Einzelwerten (z.B. Vektoren oder Matrizen) bestehen. Hier ist Scilab besonders benutzerfreundlich, da nach jeweils drei Zeilen der Ausgabe eine Abfrage erfolgt, sie hat die Form

```
[Continue display? n (no) to stop, any other key to continue]
```

MATLAB und Octave hingegen „müllen“ im Default-Modus unter Umständen das gesamte Kommandofenster mit Zahlenwerten zu. Hier hilft bei MATLAB der Befehl `more on/off` beziehungsweise `more(n)`.

Einige grundsätzliche Kommandos bei MATLAB und Scilab sind

`who` – listet die Variablen des verfügbaren Arbeitsbereichs auf

`whos` - liefert zusätzliche Informationen zu diesen Variablen (z.B. Speichergröße)

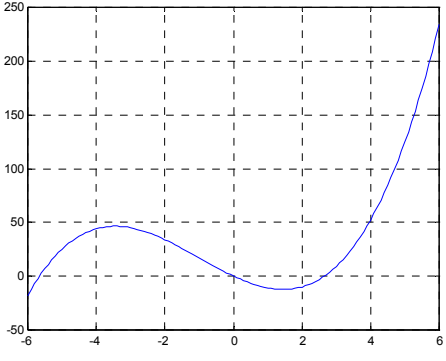
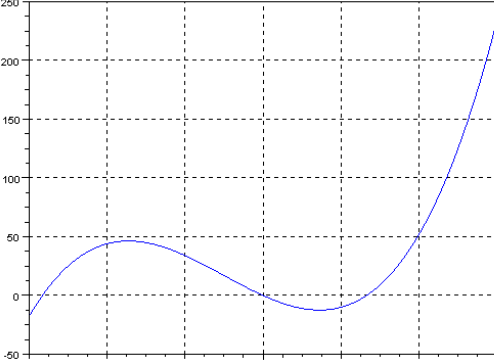
what - zeigt die Dateien im aktuellen Arbeitsverzeichnis an, bei Scilab außerdem sämtliche internen Funktionen und Variablen.

Mit dem Kommando `pause` wird die Ausführung einer Funktion unterbrochen. Dieser Befehl kann ausgesprochen nützlich für das Debuggen sein. Unter Scilab wird mit `pause` an eine neue, höhere Arbeitsebene („Workspace“) geöffnet. Dies wird durch ein verändertes Prompt-Zeichen angezeigt, zum Beispiel `-1->`. In diesem Zustand kann der Benutzer auf alle Variablen der gerade bearbeiteten Ebene zugreifen. Darüber hinaus sind alle Variablen der jeweils darunter liegenden Arbeitsebenen verfügbar, nicht jedoch umgekehrt. Auch Abbrüche oder Programmfehler können auf eine höhere Arbeitsebene führen. Durch eines der Kommandos `resume`, `return` oder `abort` gelangt man zur nächstniedrigen Arbeitsebene zurück. Ein Beispiel für die Arbeit mit dem `pause`-Befehl in Scilab soll hier angegeben werden:

```
-->a = 3;
-->pause
-1->pause
-2->resume
-1->a1=3
  a1 =
    3.
-1->c=return(a1)
-->c
  c =
    3.
-->a1
  !--error 4
Undefined variable: a1
```

#### 4.2.2. Einige grundlegende Eigenschaften

MATLAB, Octave	Scilab
Variablenwerte bleiben während der gesamten Sitzung in einer Arbeitsebene erhalten.	
<pre>» a=2; b=12.4; c=pi/4; » (a+b) / c ans =   18.3346</pre>	<pre>--&gt;a=2; b=12.4; c=%pi/4; --&gt;(a+b) / c ans =   18.334649</pre>
Variablen brauchen nicht nur aus einem Zahlenwert zu bestehen, sondern können für einen vieldimensionalen Vektor oder eine Matrix stehen.	
<pre>» x = [0 1 2 3 4 5 6 7 8 9 10] x =    0   1   2   3   4   5   6   7   8   9  10 » x = 0:1:10 x =    0   1   2   3   4   5   6   7   8   9  10 » x = 0:.1:10 x =   Columns 1 through 7    0   0.1000   0.2000  0.3000   0.4000   0.5000   0.6000 ..... » x = 0:.1:10; » y = sin(x);</pre>	<pre>--&gt; x = [0 1 2 3 4 5 6 7 8 9 10] x =    0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10. --&gt; x = 0:1:10 x =    0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10. --&gt;x = 0:.1:10 x =            column 1 to 10    0.   0.1   0.2   0.3   0.4  0.5   0.6   0.7   0.8   0.9            column 11 to 20 ... [Continue display? n (no) to stop, any</pre>

<p>MATLAB, Octave</p>	<p>Scilab</p>
	<pre>other key to continue] --&gt;x = 0:.1:10; --&gt;y = sin(x);</pre>
<p>Wirkung unterschiedlicher Ausgabeformate</p>	
<p>» format short g, format long g, format short g, format hex Das Hexformat zeigt an, wie die Zahl im Rechner gespeichert wird.</p>	<p>Das Kommando format erlaubt im Gegensatz zu MATLAB nur zwei Einstellungen: „type“ für variables Format, „long“ für die Zahl der auszugebenden Stellen. Umstellungen sind möglich durch Eingabe des Befehls format([type],[long]). Die Einstellung type kann bedeuten: 'v', variables Format (Default), d.h. je nach Zweckmäßigkeit Exponential- oder Gleitkommadarstellung) oder 'e', d.h. immer Exponentialdarstellung. Die Einstellung long gibt die Zahl der Stellen an (voreingestellt sind 10 Stellen).</p> <pre>--&gt;format('v',16), --&gt;format('v',12), --&gt;format('e',12),</pre> <p>Wenn der Formatstring weggelassen wird, wird nur die Zahl der angezeigten Ziffern verändert, der Typ des Formats bleibt bestehen:</p> <pre>--&gt;format(12) //12 Ziffern zur Anzeige</pre> <p>Mittels format() ohne Bezeichner wird nur das gerade eingestellte Format angezeigt.</p>
<p>Minimum und Maximum einer Potenzfunktion</p>	
<p>Numerisch lässt sich das Minimum einer Funktion (hier am Beispiel einer Potenzfunktion) sehr leicht ermitteln, ohne die Ableitung zu bilden:</p> <pre>» x = [-6: 0.01: 6]; » y = x.^3 + 3*x.^2 - 15*x; » plot(x,y) ; grid on ;</pre>  <pre>» min(y) ans = -12.3939</pre> <p>Durch Einschränken des Definitionsbereichs und feinere Unterteilung kann man das Minimum numerisch recht genau</p>	<pre>--&gt; x = [-6: 0.01: 6]; --&gt;y = x.^3 + 3*x.^2 - 15*x; --&gt;plot(x,y); xgrid</pre> 

MATLAB, Octave	Scilab
<p>finden:</p> <pre> » x = [-4: 0.001: 6]; » y = x.^3 + 3*x.^2 - 15*x; » min(y) ans = -12.39387515100000                     </pre> <p>Die Funktion <code>min</code> liefert den Minimalwert und den Index, der zu diesem Wert gehört.</p> <pre> » [a,b] = min(y) a = -12.393875000000000 b = 5450                     </pre> <p>Mit Kenntnis des Index' kann man jetzt den x-Wert ermitteln, der diesen Minimalwert erzeugt:</p> <pre> » x1 = x(5450) x1 = 1.4490000000000000                     </pre> <p>Das Minimum wird also bei <math>x = 1,45</math> erreicht. Wir prüfen das durch Einsetzen in die Funktion.</p> <pre> » y1 = x1.^3 + 3*x1.^2 - 15*x1 y1 = -12.39387515100000                     </pre>	<pre> --&gt;x = [-4: 0.001: 6]; --&gt;y = x.^3 + 3*x.^2 - 15*x; min(y) ans = - 12.393875                     </pre> <pre> --&gt;[a,b] = min(y) b = 5450. a = - 12.393875                     </pre> <pre> --&gt;x1 = x(5450) x1 = 1.449                     </pre> <pre> --&gt;y1 = x1.^3 + 3*x1.^2 - 15*x1 y1 = - 12.393875                     </pre>

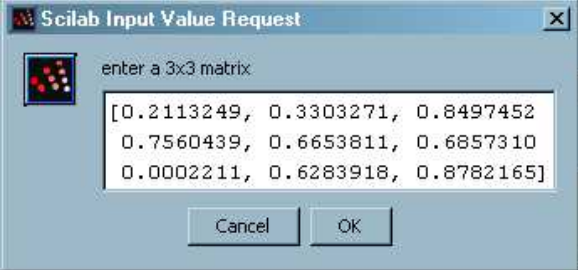
Alle Rechenoperationen wie Multiplikation, Division, Potenzieren werden grundsätzlich als Matrizenoperationen ausgeführt. Die „normale“ (elementweise) Multiplikation wird im Gegensatz hierzu durch einen Punkt vor dem Operationssymbol signalisiert. Häufige Fehler beim Arbeiten gerade bei den ersten Schritten entstehen, wenn der Punkt bei skalaren Operationen (`.*`, `./` und `.^`) vergessen wird. Hier unterscheiden sich die Anforderungen bei Scilab und MATLAB ein wenig. Bei Scilab ist zu beachten, dass ein Punkt nach einer Zahl immer als Dezimalpunkt interpretiert wird. Ein weiterer Punkt oder ein Leerzeichen zwischen der ganzen Zahl und dem Punkt ist deshalb nötig, um die Operation als skalare Operation zu kennzeichnen. Beachtet man dies nicht, können bei der Übertragung von MATLAB-Programmen nach Scilab auf diese Weise überraschende Fehlermeldungen entstehen.

Beispiel:

MATLAB, Octave	Scilab
<pre> » x = [-1 -.5 .5 1] x = -1.0000 -0.5000 0.5000 1.0000 » 1 ./x.^2 ans = 1 4 4 1                     </pre> <p>Die nächsten beiden Eingaben liefern jedoch unterschiedliche Ergebnisse:</p> <pre> » 1./x.^2 ans = 1 4 4 1                     </pre>	<pre> --&gt;x = [-1 -.5 .5 1] x = - 1. - 0.5 0.5 1. --&gt;1 ./x.^2 ans = 1. 4. 4. 1.                     </pre> <p>(Das Leerzeichen sichert die Interpretation des Punktes als zum Divisionssymbol gehörig.)</p> <pre> --&gt;1./x.^2 ans = 0.4705882 0.1176471                     </pre>

MATLAB, Octave	Scilab
<pre> » 1/x.^2 ??? Error using ==&gt; / Matrix dimensions must agree. Die Fehlermeldung bei MATLAB bewahrt vor Irrtümern, im Gegensatz zu Scilab, wo etwas Unkontrollierbares berechnet wird.                     </pre>	<pre> 0.1176471 0.4705882 Dies ist offensichtlich nicht das gewünschte Ergebnis. --&gt;1/x.^2 ans = 0.4705882 0.1176471 0.1176471 0.4705882                     </pre>

### 4.3. Einfache Vektoren und Matrizen

MATLAB, Octave	Scilab
Schreibweise von Matrizen	
Hier: am Beispiel zweier (3 x 3)-Matrizen A und B:	
$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 6 & 7 & 8 \end{pmatrix}, \quad B = \begin{pmatrix} -1 & 1 & 3 \\ 1 & -2 & 3 \\ 4 & -1 & 6 \end{pmatrix}$	
<pre> » A = [1 2 3; 2 4 6; 6 7 8] A =     1     2     3     2     4     6     6     7     8 » B = [-1 1 3; 1 -2 3; 4 -1 6] B =    -1     1     3     1    -2     3     4    -1     6                     </pre> <p>Elemente einer Zeile werden durch Zwischenraum oder Komma abgetrennt, Spalten voneinander durch Semikolon.</p>	<pre> --&gt;A = [1 2 3; 2 4 6; 6 7 8] A =     1.     2.     3.     2.     4.     6.     6.     7.     8.                     </pre> <p>usw.</p> <p>Scilab besitzt übrigens eine Benutzerschnittstelle, die es erlaubt, graphisch Matrizen einzugeben oder vorgegebene Matrizen zu verändern. Dieses Fenster wird mit dem Befehl <code>evstr</code> erzeugt:</p> <pre> --&gt;m=evstr(x_matrix('enter a 3x3 matrix',rand(3,3)))                     </pre>  <pre> m =     0.2113249    0.3303271    0.8497452     0.7560439    0.6653811    0.685731     0.0002211    0.6283918    0.8782165                     </pre>
Addition und Subtraktion von Matrizen (bei MATLAB und Scilab im Wesentlichen gleich)	
Addition	
<pre> » A+B ans =     0     3     6     3     2     9    10     6    14                     </pre>	

**Element-by-Element-Multiplikation:**

```
» Q = A.*B
```

```
Q =
    -1     2     9
     2    -8    18
    24    -7    48
```

**Matrizen-Multiplikation:**

```
» R = A*B
```

```
R =
    13    -6    27
    26   -12   54
    33   -16   87
```

A(1:3,2) extrahiert die zweite Spalte der Matrix A.

```
» A(1:3,2)
```

```
ans =
     2
     4
     7
```

**Weitere Beispiele:**

```
» A(1,:) 
```

```
ans =
     1     2     3
```

(Steht der Doppelpunkt allein, bedeutet das „alle“.)

**Zusammensetzen einzelner Vektoren:**

```
» Z = [A(1,:) B(3,:)]
```

```
Z =
     1     2     3     4    -1     6
```

**Matrix transponieren:**

```
» Z'
```

```
ans =
     1
     2
     3
     4
    -1
     6
```

**Summe aller Elemente einer Matrix:**

```
» sum(Z)
```

```
ans =
    15
```

**Ergänzung:** Der Befehl A(:) erzeugt aus jeder Matrix (oder jedem Vektor) einen Spaltenvektor. Dies kann benutzt werden, um das Einlesen von beliebigen Vektoren in Funktionen zu vereinfachen. Die Funktionen nehmen dann Vektoren jeder Art an, gleich ob Zeilen- oder Spaltenvektor.

```
Z =
     1     2     3     3     6
```

```
» Z(:)
```

```
ans =
     1
     2
     3
     3
     6
```

```
» Y=Z'
```

```
Y =
     1
```

```

2
3
3
6
» Y(:)
ans =
1
2
3
3
6
- Matrix vertikal spiegeln (left-right) (In Scilab: mtlb_fliplr(Z))5
» fliplr(Z)
ans =
6 -1 4 3 2 1
- Matrix oben-unten-spiegeln (up-down) (Hierzu gibt es keine Entsprechung in Scilab)
» flipud(Z')
ans =
6
-1
4
3
2

```

#### 4.4. Rechnen mit komplexen Zahlen

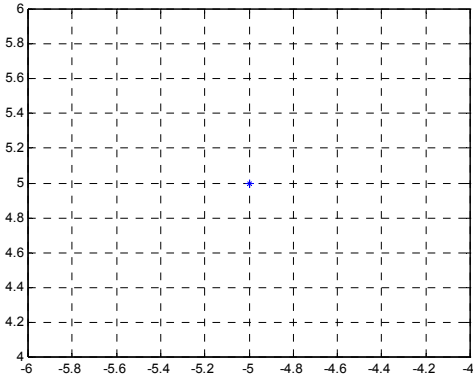
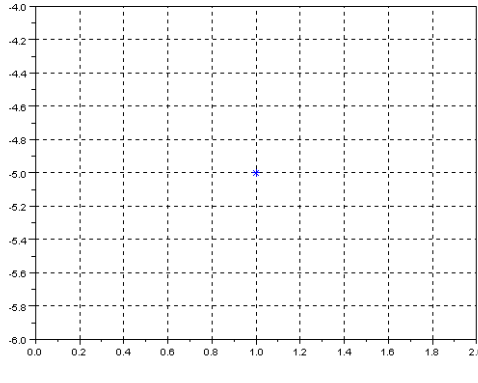
Komplexe Zahlen werden in MATLAB in der Form  $z = -5+5i$  oder  $z = -5+5j$  dargestellt. Die Buchstaben  $i$  und  $j$  sind demnach, wie auch zum Beispiel die Variable `pi`, vordefiniert. In Scilab ist  $j$  nicht vorgesehen und der imaginären Einheit  $i$  muss als einer vordefinierter Variablen ein Prozentzeichen vorangestellt werden: `%i`. Das ist schade, denn es bringt leider etwas erhöhten Schreibaufwand mit sich.

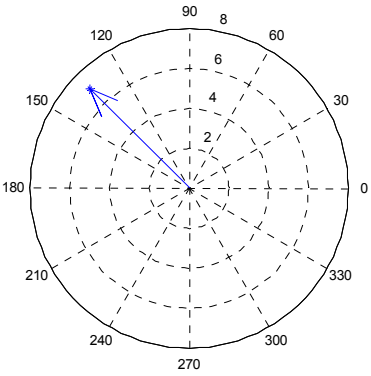
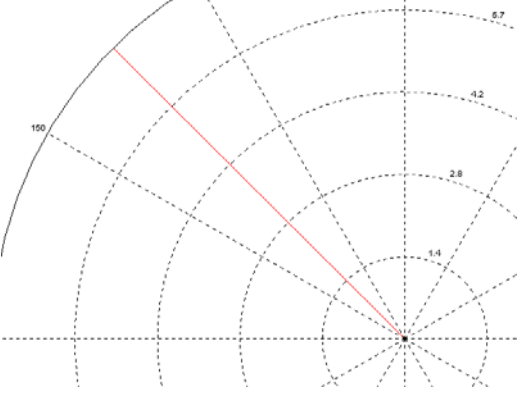
Als Beispiel werden jetzt Betrag und Phase einer komplexen Zahl  $z$  berechnet. Die Darstellung in der komplexen Ebene geschieht mit dem Befehl `plot` und in MATLAB außerdem mit `compass`.

MATLAB, Octave	Scilab
<pre> » z=-5+5i z = -5.0000 + 5.0000i </pre>	<pre> --&gt;z=-5+5*%i z = - 5. + 5.i </pre>
<p><b>Betrag:</b></p> <pre> » r = abs(z) r </pre>	<p><b>Betrag:</b></p> <pre> --&gt;r = abs(z) r = </pre>

<sup>5</sup> *Hinweis:* In Scilab gibt es eine Reihe von „Kompatibilitätsfunktionen“ (*compatibility functions*), die die Äquivalenz zu verschiedenen MATLAB-Funktionen herstellen. Die Umkehrung der Spalten einer Matrix zum Beispiel wird in MATLAB mit der Funktion `fliplr` bewirkt. In Scilab gibt es dafür keine unmittelbare Entsprechung, jedoch die MATLAB-kompatible Funktion `mtlb_fliplr`. Eine zu `flipud` (Spiegelung der Zeilen einer Matrix) kompatible Funktion existiert jedoch in Scilab nicht.



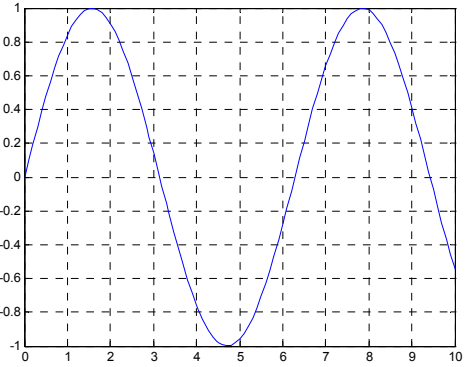
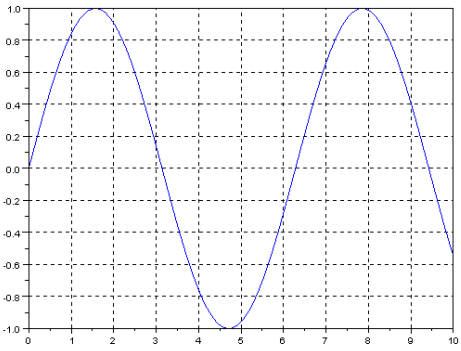
MATLAB, Octave	Scilab
<p>7.0711</p> <p>Phasenwinkel:</p> <pre>» phi = angle(z) phi =     2.3562</pre>	<p>7.0710678</p> <p>Eine Funktion wie <code>angle</code> in MATLAB ist in Scilab nicht verfügbar. Um den Phasenwinkel einer komplexen Zahl zu berechnen, kann man stattdessen schreiben:</p> <pre>--&gt;phase_z = atan(imag(z),real(z)) phase_z =     2.3561945</pre>
	<p><b>Achtung! Wird stattdessen</b></p> <pre>--&gt;phase_z = atan(imag(z)./real(z)) phase_z =     - 0.7853982</pre>
<p>Die Exponentialform ergibt wieder das richtige z:</p> <pre>» z1 = r * exp(i*phi) z1 =     -5.0000 + 5.0000i</pre>	<p>benutzt (also mit Bruchstrich, aber ohne das Komma), so wird nur der Hauptwert des Arkustangens ausgegeben (zwischen <math>-\pi/2</math> und <math>+\pi/2</math>).</p> <p>Analog dazu kann auch die Funktion</p> <pre>--&gt;[r,phi] = polar(z) phi =     2.3561945 + 1.110D-16i r =     7.0710678</pre>
<p>Automatische Darstellung in der komplexen Zahlenebene:</p> <pre>» plot(z, '*'); grid;</pre>	<p>verwendet werden.</p> <p>Die Exponentialform ergibt wieder das richtige z:</p> <pre>--&gt;z1 = r * exp(phi*i) z1 =     - 5. + 5.i</pre>
	<p>Komplexe Zahlenebene:</p> <pre>--&gt;plot(z, '*'); xgrid;</pre> 
<p>Zur Darstellung der Zahl z in Polarkoordinaten dient die Funktion <code>polar</code> (zeichnet einen Punkt) und <code>compass</code> (zeichnet einen Pfeil).</p> <p>Gemeinsames Bild von <code>compass(z)</code> und <code>polar(z)</code>:</p> <pre>» polar(zphase, zbetrag, '*') » hold on » compass(z)</pre>	<p>Eine etwa <code>polar</code> entsprechende Funktion ist <code>polarplot</code>:</p> <pre>--&gt;polarplot([0 phi],[0 abs(r)],[5,,])</pre> <p>Ohne den Startpunkt bei null wird keine Linie gezeichnet. Durch das letzte (optionale) Argument werden verschiedene Informationen übergeben, die Ziffer 5 z.B. bedeutet, dass die Linie rot dargestellt wird.</p>

MATLAB, Octave	Scilab
	
<p>Einige Beispiele für Operationen mit zwei komplexen Zahlen (Ergebnis in algebraischer Form als Real- plus Imaginärteil):</p>	
<pre>» abs(3-4i)*abs(4+3i) ans =     25</pre>	<pre>--&gt;abs(3-4%i)*abs(4+3%i) ans =     25.</pre>
$\left  \frac{1}{1+3i} - \frac{1}{1-3i} \right $ <pre>» abs(1/(1+3i) - 1/(1-3i)) ans =     0.6000</pre>	$\left  \frac{1}{1+3i} - \frac{1}{1-3i} \right $ <pre>--&gt;abs(1/(1+3%i) - 1/(1-3%i)) ans =     0.6</pre>
<p>Das ergibt gerade 3/5. Zum Vergleich:</p> <pre>» 3/5 ans =     0.6000</pre>	
$(-1+\sqrt{3}i)^{10}$ <pre>» (-1+sqrt(3)*i)^10 ans = -5.1200e+002 +8.8681e+002i</pre>	$(-1+\sqrt{3}i)^{10}$ <pre>--&gt;(-1+sqrt(3)*%i)^10 ans =</pre>
<p>Zum Vergleich:</p> <pre>» 512*sqrt(3) ans =     886.8100</pre>	<pre>- 512. + 886.81001i</pre>

#### 4.5. Graphik

Ergebnisse können als zwei- und dreidimensionale farbige Graphik ausgegeben werden. Für eine zweidimensionale Liniengrafik wird der Befehl `plot` benutzt. Scilab kennt dafür die Befehle `plot` oder (äquivalent) `plot2d`. Das zugehörige Gitter wird mittels `grid` oder `grid on` (Scilab: `xgrid`) gezeichnet. In MATLAB wird mit jeder neuen Graphik das vorherige Graphikfenster gelöscht. Will man das vermeiden, muss dies mit einem dem `plot`-Befehl nachgeschalteten Befehl `hold` oder `hold on` kenntlich gemacht werden. In Scilab bleibt eine Graphik prinzipiell erhalten, solange man sie nicht mittels `clf` löscht.

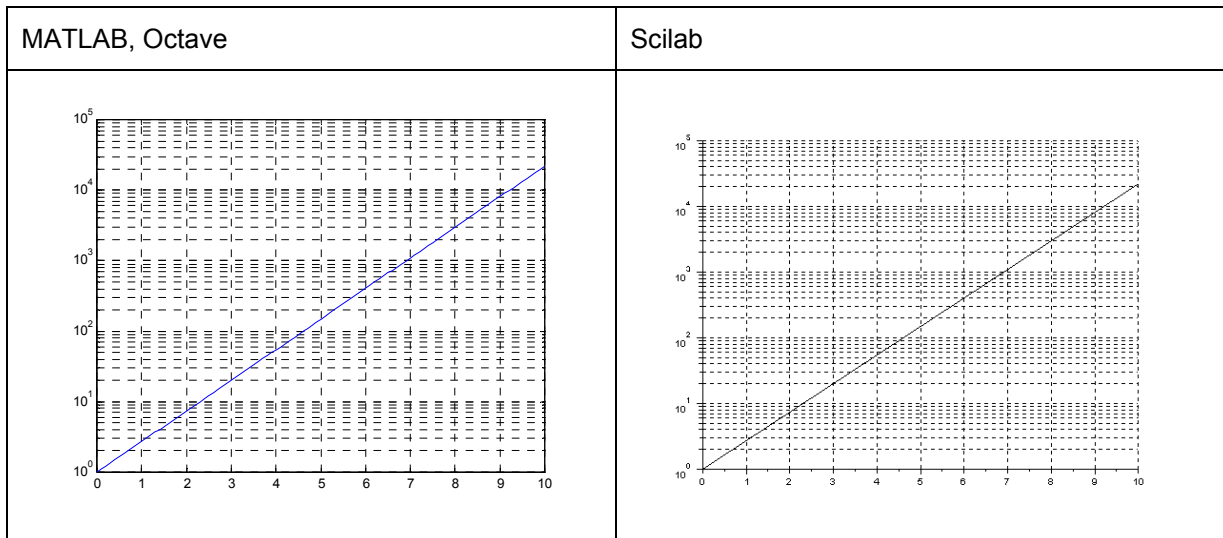
Wir stellen zur Illustration die Ergebnisse einer Sinusberechnung dar.

MATLAB, Octave	Scilab
<pre>» x = 0:.1:10; » y = sin(x); » plot(x,y) » grid</pre>	<pre>--&gt;x = 0:.1:10; --&gt;y = sin(x); --&gt;plot(x,y) --&gt;xgrid</pre>
	

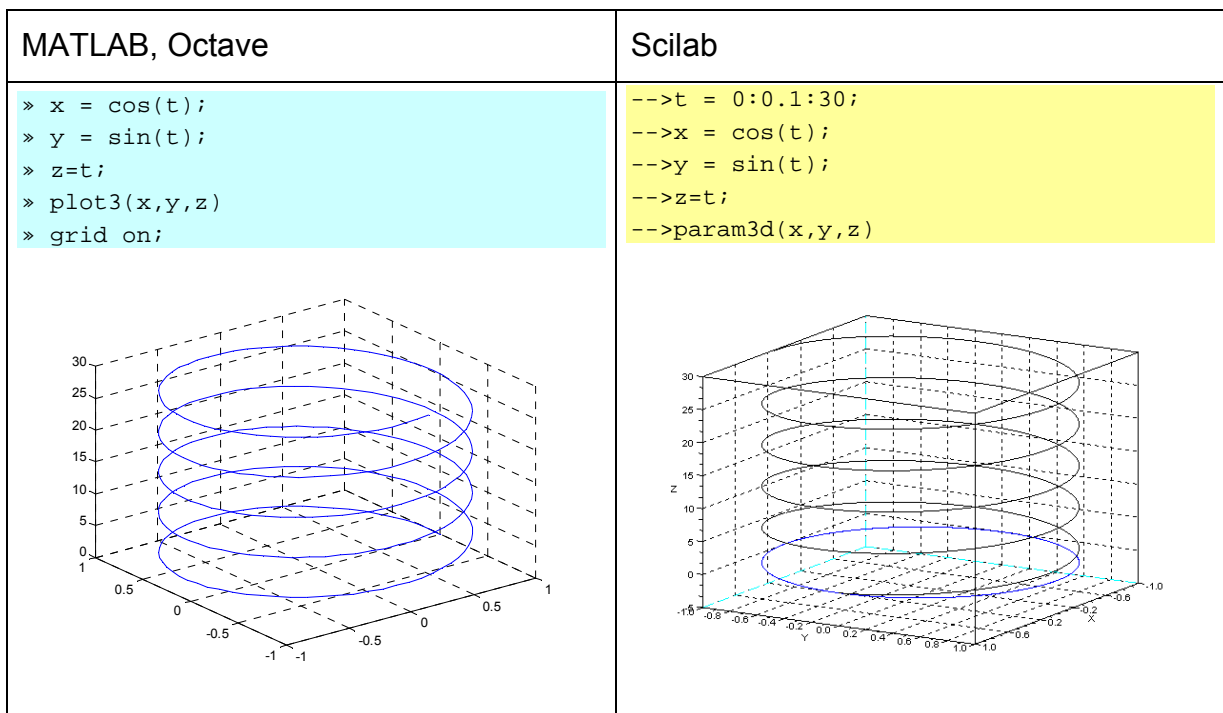
Weitergehende Graphikfunktionen unterscheiden sich teilweise erheblich. So existiert in Scilab beispielsweise kein Befehl, der unmittelbar eine logarithmische oder halblogarithmische Achseneinteilung bewirkt. In MATLAB stehen dafür `loglog`, `semilogx` und `semilogy` zur Verfügung. In Scilab muss mit `plot2d` gearbeitet werden. Hier hat man als Option `logflag = xx` einzutragen. Für `xx` kann dabei `ll`, `nl` oder `ln` stehen.

Eine Alternative stellt bei Scilab (sofern implementiert) das Funktionspaket `myplot` dar. In der folgenden Tabelle sind diese Optionen gegenübergestellt:

MATLAB, Octave	Scilab
Achseneinteilung linear oder logarithmisch	
<p>Lineare Achseneinteilung</p> <pre>» plot(x,y)</pre>	<p>Lineare Achseneinteilung</p> <pre>--&gt;plot(x,y) oder --&gt;plot2d(x,y) oder --&gt;plot2d(x,y,logflag='nn')</pre>
<p>x-Achse logarithmisch</p> <pre>» semilogx(x,y)</pre>	<p>x-Achse logarithmisch</p> <pre>--&gt;plot2d(x,y,logflag='ln')</pre>
<p>y-Achse logarithmisch</p> <pre>» semilogy(x,y)</pre>	<p>y-Achse logarithmisch</p> <pre>--&gt;plot2d(x,y,logflag='nl')</pre>
<p>Beide Achsen logarithmisch</p> <pre>» loglog(x,y)</pre>	<p>Beide Achsen logarithmisch</p> <pre>--&gt;plot2d(x,y,logflag='ll')</pre>
<p>Beispiel</p> <pre>» x=0:.1:10; » y=exp(x); » semilogy(x,y) » grid on</pre>	<p>Beispiel</p> <pre>--&gt;x=0:.1:10; --&gt;y=exp(x); --&gt;plot2d(x,y,logflag='nl') --&gt;xgrid</pre>



Für eine farbige *dreidimensionale Liniengraphik* steht der Befehl `plot3` (Scilab: `param3d`) zur Verfügung. Hier als Beispiel die Parameterdarstellung einer Schraubenlinie.



Für die *Darstellung von Flächen*  $z = z(x,y)$  ist entscheidend, vorher die abzubildende Funktion in allen gewünschten Punkten auch tatsächlich zu berechnen. Hierzu muss ein Gitternetzwerk in der  $(x,y)$ -Ebene festgelegt werden. Es wird mittels `meshgrid` erzeugt. An einem sehr einfachen Beispiel mit einem noch sehr groben Gitternetz soll das in MATLAB gezeigt werden:

Die  $x$ - und  $y$ -Vektoren

```

x = [-2 1 0 1 2]
y = [-1 0 1]
                    
```

erzeugen zum Beispiel folgende Matrix:

```

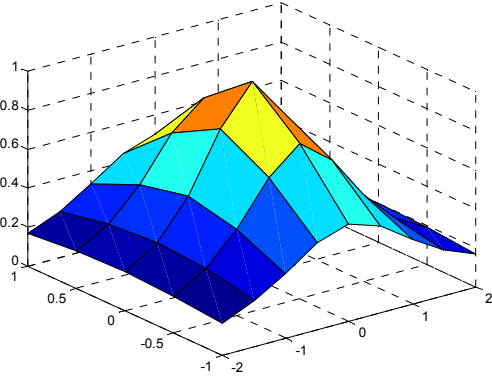
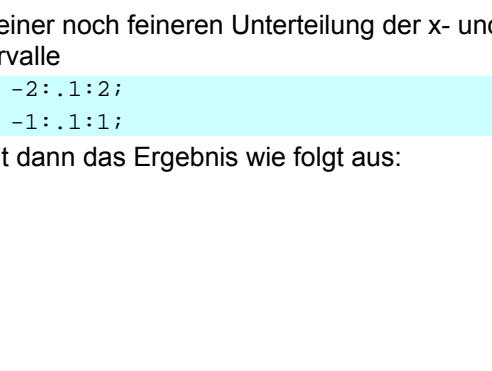
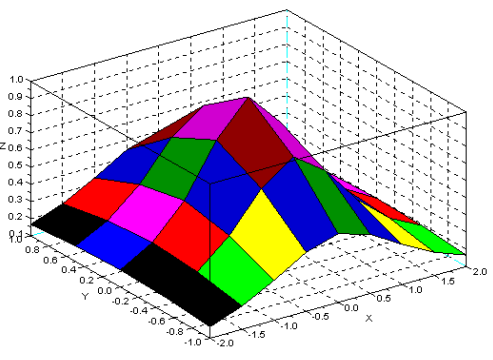
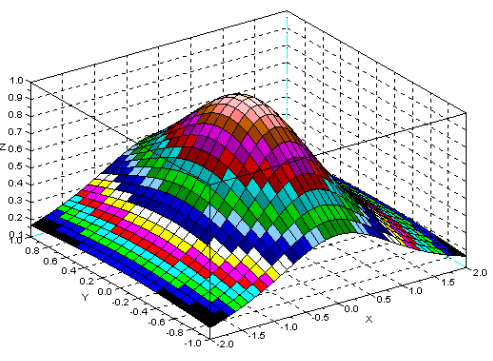
[Xgrid Ygrid] = meshgrid(x,y)
Xgrid =
-2     1     0     1     2
-2     1     0     1     2
                    
```

```
-2    1    0    1    2
Ygrid =
-1    -1    -1    -1    -1
 0     0     0     0     0
 1     1     1     1     1
```

Damit können nun die Funktionswerte berechnet werden:

```
z = 1./(1+(Xgrid.^2+Ygrid.^2))
z =
 0.1667 0.3333 0.5000 0.3333 0.1667
 0.2000 0.5000 1.0000 0.5000 0.2000
 0.1667 0.3333 0.5000 0.3333 0.1667
```

Mit feinerer Unterteilung ergibt sich natürlich eine bessere Darstellung der Funktion.

MATLAB, Octave	Scilab
<pre>x = -2:.5:2; y = -1:.5:1;</pre> <p>Damit werden nun sehr viele Funktionswerte erzeugt. Sie sollen deshalb hier nicht wiedergegeben werden.</p> <pre>[Xgrid Ygrid] = meshgrid(x,y); z = 1./(1+(Xgrid.^2+Ygrid.^2));</pre> <p>Ausgabe von Text und Graphik</p> <pre>surf(Xgrid,Ygrid,z);</pre>  <p>Mit einer noch feineren Unterteilung der x- und y-Intervalle</p> <pre>x = -2:.1:2; y = -1:.1:1;</pre> <p>sieht dann das Ergebnis wie folgt aus:</p> 	<pre>--&gt;x = -2:.5:2; --&gt;y = -1:.5:1;</pre> <pre>--&gt;[Xgrid Ygrid] = meshgrid(x,y); --&gt;z = 1./(1+(Xgrid.^2+Ygrid.^2)) --&gt;surf(Xgrid,Ygrid,z) --&gt;xgrid //zeichnet das Gitternetz</pre>  <p>Die Scilab-Graphik mit der feineren Unterteilung:</p> 

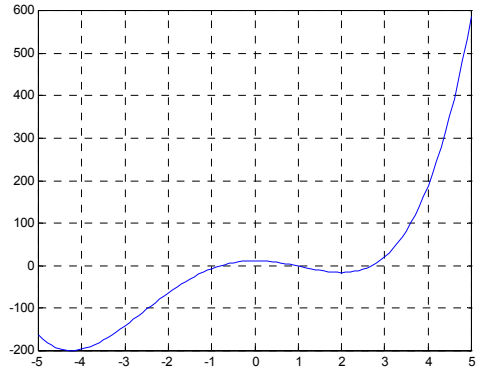
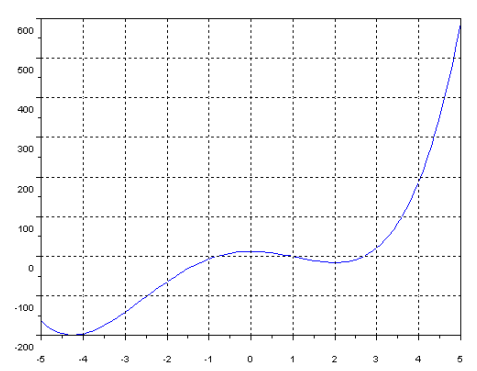
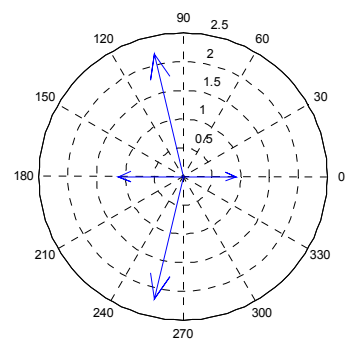
MATLAB, Octave	Scilab
<div data-bbox="252 297 746 674" data-label="Figure"> </div> <p data-bbox="199 723 331 752">Der Befehl</p> <pre data-bbox="199 759 790 788">» contour3 (Xgrid,Ygrid,z)</pre> <p data-bbox="199 792 782 922">zeichnet 3-dimensionale Höhenlinien. Mittels colormap kann die Farbe sowohl des "surf"-Plots als auch des "contour3"-Plots verändert werden, z.B. colormap copper oder:</p> <pre data-bbox="199 929 790 958">» colormap ([0,0,0])</pre> <div data-bbox="252 1003 746 1379" data-label="Figure"> </div>	<p data-bbox="815 255 1366 344">Die Verteilung der Farbstufen in der MATLAB-Graphik erscheint im Vergleich mit der von Scilab viel harmonischer.</p>

#### 4.6. Matrizenalgebra und Polynome

Polynome werden in vielen Bereichen der Technik benötigt. In der Regelungstechnik beispielsweise müssen rationale Übertragungsfunktionen durch Zähler- und Nennerpolynome dargestellt werden. Deshalb existieren besondere Verfahren zur Polynomrechnung. Polynome werden sowohl in MATLAB/Octave als auch in Scilab durch den Vektor ihrer Koeffizienten dargestellt, in MATLAB/Octave beginnend mit der höchsten Potenz, in Scilab allerdings gerade umgekehrt.

MATLAB, Octave	Scilab
<p><b>Darstellung von Polynomen. Nullstellen</b></p>	
<p><i>Beispiel:</i> Polynom <math>y_1 = p_1(x) = x^4 + 3x^3 - 17x^2 + 12</math></p>	
<p>MATLAB sortiert nach absteigenden Potenzen:</p>	<p><i>Achtung:</i> Scilab dagegen sortiert nach aufsteigenden Potenzen. Die Reihenfolge der Polynomkoeffizienten ist also umgekehrt zu der in</p>

MATLAB, Octave	Scilab
<p><math>y_1 = p_1(x) = x^4 + 3x^3 - 17x^2 + 12</math></p> <p>Es wird dargestellt durch</p> <pre>&gt;&gt; p1 = [1 3 -17 0 12] p1 =      1     3    -17     0    12</pre>	<p><b>MATLAB !</b></p> <p><math>y_1 = p_1(x) = 12 - 17x^2 + 3x^3 + x^4</math></p> <pre>--&gt;p1 = [12 0 -17 3 1] p1 =     12.    0.   -17.    3.    1.</pre> <p>Die Umkehrung der Reihenfolge der Koeffizientenreihenfolge kann z.B. mittels <code>mtlb_fliplr</code> geschehen:</p> <pre>--&gt;p1 = [1 3 -17 0 12] p1 =      1     3    -17     0    12. --&gt;p1_scilab = mtlb_fliplr(p1) p1_scilab =     12.    0.   -17.    3.    1.</pre> <p>Aus der Polynommatrix erhalten wir das Polynom selbst mit Hilfe der Funktion <code>poly</code>.</p> <pre>--&gt;q1 = poly(p1,'x','c') //mit 'c':Polynombestimmung anhand der Koeffizienten; //mit 'r' oder ohne Bezeichner: anhand der Wurzeln (ähnlich wie in MATLAB, jedoch hat die Ausgabe eine andere Gestalt).</pre> <pre>q1 =            2     3     4     12 - 17x + 3x + x</pre>
<p>Soll dieses Polynom zum Beispiel für einen bestimmten <math>x</math>-Wert, zum Beispiel <math>x = 3</math> berechnet werden, so verwenden wir die Funktion <code>polyval</code>:</p> <pre>&gt;&gt; polyval(p1,3) ans =      21</pre>	<p>Rückwärts kann man aus <code>q1</code> wieder <code>p1</code> durch die Funktion <code>coeff</code> erhalten:</p> <pre>c = coeff(q1) c =     12.    0.   -17.    3.    1.</pre> <p>Tatsächlich ist <code>c</code> gleich <code>p1</code>.</p> <p>Soll dieses Polynom für einen bestimmten <math>x</math>-Wert, zum Beispiel <math>x = 3</math>, berechnet werden, so verwenden wir die Funktion <code>horner</code> als äquivalente Funktion zu <code>polyval</code>.</p> <pre>--&gt;horner(q1, 3) ans =      21.</pre>
<p>In MATLAB hat die Funktion <code>poly</code> allein die Aufgabe, ein Polynom aus seinen Wurzeln zu bilden.</p> <p>Verlauf des Polynoms <math>y_1 = p_1(x)</math> als MATLAB-Plot im Bereich <code>-5:.1:5</code>.</p> <pre>&gt;&gt; x = -5:.1:5; &gt;&gt; y1 = polyval(p1,x); &gt;&gt; plot(x,y1); grid</pre>	<p>Verlauf des Polynoms <math>y_1 = p_1(x)</math> als Scilab-Plot im Bereich <code>-5:.1:5</code>.</p> <pre>--&gt;xa = -5:0.1:5; // vorgegebene X-Werte --&gt;y = horner(q1, xa); // X-Werte den // Y-Werten zuweisen --&gt;plot(xa,y) --&gt;xgrid(1) //Gitter zeichnen // in Klammern Linienformat, z.B. Farbe</pre>

MATLAB, Octave	Scilab
 <p>Berechnung der Nullstellen mittels <code>roots</code>.</p> <pre data-bbox="199 739 790 929">&gt;&gt; roots(p1) ans = -5.8473  2.6950  0.9521 -0.7998</pre>	 <p>Berechnung der Nullstellen mittels <code>roots</code>.</p> <pre data-bbox="805 716 1396 907">--&gt;roots(p1) ans =  0.9521264 - 0.7997929  2.694968 - 5.8473014</pre>
<p>Weiteres Beispiel: Nullstellen von <math>p_2(x) = x^4 + 1,2x^3 + 4x^2 - 5</math><sup>6)</sup></p>	
<pre data-bbox="199 996 790 1254">&gt;&gt; p2 = [1 1.2 4 0 -5]; &gt;&gt; r2 = roots(p2) r2 = -0.4940 + 2.1407i -0.4940 - 2.1407i  0.9173 -1.1294</pre> <p>Darstellung der Nullstellen als Zeiger in der komplexen Ebene</p> <pre data-bbox="199 1355 790 1388">&gt;&gt; compass(r2)</pre> 	<pre data-bbox="805 996 1396 1377">--&gt;p2 = [-5 0 4 1.2 1]; --&gt;q2 = poly(p2, 'x', 'c') x =       2      3      4 - 5 + 4x + 1.2x + x --&gt;r2 = roots(p2) r2 = - 0.1023469 + 0.4435272i - 0.1023469 - 0.4435272i - 0.8854621  1.0901559</pre> <p>Eine zu <code>compass</code> analoge Funktion existiert in Scilab nicht.</p>
<p><b>Operationen mit zwei Polynomen</b></p> $p_1(x) = x^4 + 3x^3 - 17x^2 + 12$	

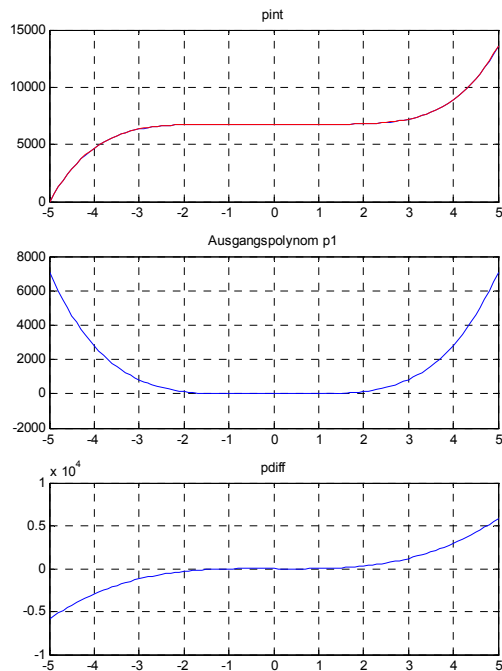
<sup>6)</sup> Hinweis: In Scilab werden mit der Funktion `poly` die Potenzen „halbgraphisch“ ausgegeben. Das bedeutet, in einer Zeile stehen die Exponenten, in der Zeile darunter die Koeffizienten mit `x`.



MATLAB, Octave	Scilab
$p_2(x) = x^4 + 1,2x^3 + 4x^2 - 5$	
<pre>&gt;&gt; p1 = [1 3 -17 0 12]; &gt;&gt; p2 = [1 1.2 4 0 -5];</pre> <p><b>- Addition</b>  <i>Hinweis:</i> Die Addition funktioniert nur, wenn beide Polynome gleichen Grad besitzen, ansonsten sind die fehlenden Koeffizienten mit Nullen aufzufüllen.</p> <pre>&gt;&gt; p_add = p1 + p2 p_add =     2.0000 4.2000 -13.0000 0 7.0000</pre>	<pre>--&gt;p1 = [12 0 -17 3 1]; --&gt;p2 = [-5 0 4 1.2 1];</pre> <p><b>- Addition</b>  <pre>--&gt;p_add = p1+p2 p_add =     7.    0.   -13.    4.2    2.</pre> <p>Die Addition funktioniert in Scilab auch mit den "halbgraphischen" Polynomen</p> <pre>--&gt;q1 =poly(p1,'x','c') x =       2      3      4     12 - 17x + 3x + x</pre> <pre>--&gt;q2 = poly(p2,'x','c') x =       2      3      4     - 5 + 4x + 1.2x + x</pre> <pre>--&gt;q_add = q1+q2 q_add =       2      3      4     7 - 13x + 4.2x + 2x</pre> </p>
<p><b>- Multiplikation</b> <math>p_{mult}(x) = p_1 \cdot p_2</math></p> <pre>p_mult = conv(p1,p2) p_mult =     1.0000 4.2000 -9.4000 -8.4000 -1.0000 -0.6000    133.0000 0 -60.0000</pre> <pre>p4 = p_mult</pre> <p><b>- Division</b> <math>p_{div}(x) = p_{mult} / p_1</math></p> <pre>&gt;&gt; p_div = deconv(p_mult,p1) p_div =     1.0000 1.2000 4.0000 0.0000 -5.0000</pre>	<p><b>- Multiplikation</b> <math>p_{mult}(x) = p_1 \cdot p_2</math></p> <pre>--&gt;q_mult = q1*q2 q_mult =       2      3      4      5      6      7      8     - 60 + 133x - 0.6x - 61x - 8.4x - 9.4x + 4.2x + x</pre> <p><b>- Division</b> <math>p_{div}(x) = p_{mult} / p_1</math></p> <pre>--&gt;q_mult/q1 ans =       2      3      4     - 5 + 5.874D-15x + 4x + 1.2x + x     -----       1</pre> <pre>--&gt;[a,b] = pdiv(q2,q_mult)  b =     0. a =       2      3      4     - 5 + 4x + 1.2x + x</pre>
<p><b>- Differentiation</b></p> <pre>&gt;&gt; p_diff = polyder(p1) p_diff =     4      9     -34      0</pre> <p><b>- unbestimmtes Integral</b>  Zur Berechnung von</p>	<p>Man erhält also wieder das Polynom <math>q_2</math>.</p> <pre>q2 =       2      3      4     - 5 + 4x + 1.2x + x</pre> <p><b>- Differentiation</b></p> <pre>--&gt;q_diff=derivat(q1) q_diff =       2      3     - 34x + 9x + 4x</pre> <p>Eine Funktion, die <code>polyint</code> entspricht, ist in Scilab nicht vorhanden. In [12], Seite 398, ist jedoch ein entsprechendes Programm angegeben, mit dem diese Aufgabe ebenfalls ausgeführt werden kann.</p>

MATLAB, Octave	Scilab
$y_{\text{int}} = \int_{-5}^x p_1(x) dx$ <p>kann <code>polyint</code> benutzt werden (erst ab MATLAB-Version 6.5 verfügbar).</p> <pre>&gt;&gt; p_int = polyint(p1) p_int =     0.2000  0.7500 -5.6667  0 12.0000  0</pre> <p>Die drei Funktionen</p> <ul style="list-style-type: none"> <li>- <code>p_int = polyint(p1)</code></li> <li>- <code>p1</code> und</li> <li>- <code>p_diff = polyder(p1)</code>.</li> </ul> <p>sollen in <i>einer</i> Abbildung mit drei Plots übereinander dargestellt werden. Der Wertebereich soll wie oben verwendet werden, also</p> <pre>&gt;&gt; x = -5:.1:5;</pre> <p>Wir benutzen dazu die Funktion <code>subplot</code>. Die Funktion <code>subplot(m,n,p)</code> stellt eine <math>(m \times n)</math>-Matrix für die Graphikausgabe zur Verfügung. In unserem Fall definiert <code>subplot(3,1,1)</code> demnach eine <math>(3 \times 1)</math>-Matrix (3 Zeilen, 1 Spalte), in der die Graphik des nächstfolgenden <code>plot</code>-Befehls im ersten (oberen) Fenster platziert wird.</p> <pre>&gt;&gt; subplot(3,1,1); &gt;&gt; plot(x,polyval(p_int,x)); grid on; &gt;&gt; hold on; &gt;&gt; title ('pint');</pre> <p>(Da das letzte Argument eine 1 ist, wird die Graphik in der 1. Zeile platziert.)</p> <pre>&gt;&gt; subplot(3,1,2); &gt;&gt; plot(x,polyval(p1,x)); grid; &gt;&gt; title ('Ausgangspolynom p1');</pre> <p>(Da das letzte Argument eine 2 ist, wird die Graphik in der 2. Zeile platziert.)</p> <pre>&gt;&gt; subplot(3,1,3); &gt;&gt; plot(x,polyval(p_diff,x)); grid; &gt;&gt; title ('pdiff');</pre> <p>(Graphik wird in der 3. Zeile platziert.)</p> <pre>&gt;&gt; y_int = ...     polyval(p_int,x) - ...     polyval(p_int,x(1)); ...     % bestimmtes Integral mit Anfangswert &gt;&gt; subplot(3,1,3); &gt;&gt; plot(x,y_int); grid on;</pre>	

## MATLAB, Octave



Jetzt könnte zur Ergänzung das bestimmte Integral alternativ nach der Trapezregel berechnet werden. Hierzu sind die numerischen Werte der Funktion `y1` explizit zu benutzen.

```
y1 = polyval(p1,x);
```

Damit ergibt sich

```
y1_int = cumtrapz(y1)*.1;;
```

Hinweis: Die MATLAB-Trapezregel arbeitet normalerweise nur für Intervalle der Breite 1, deshalb wird hier der Wert des Integrals mit der Intervallbreite  $\Delta x = 0.1$  multipliziert.

Wir stellen `y1_int` im gleichen Fenster dar wie `y_int`, allerdings jetzt als rote Linie

```
>> hold on
>> subplot(3,1,1), plot(x,y1_int,'r');
>> grid on;
```

Analog soll nun numerisch die Ableitung `z_diff` berechnet und im unteren Fenster zusätzlich dargestellt werden. Um die Ableitung zu bilden, berechnen wir die Differenzenquotienten

$$y_{\text{diff}} = \frac{dy}{dx} = \frac{d p_1(x)}{dx} \approx \frac{\Delta p_1(x)}{\Delta x}$$

nach der Beziehung

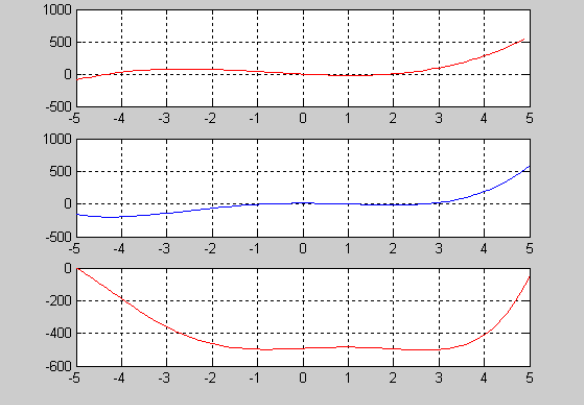
```
z_diff = diff(y1)./diff(x).
```

*Hinweis:* Durch das Bilden der Differenz werden die `x`-Werte um einen Wert vermindert, denn zum Startwert allein gibt es noch keine Differenz.

```
>> xd = x(1:end-1);
>> hold on
>> subplot(3,1,1), plot(xd,z_diff,'r')
```

## Scilab

Für Berechnungen nach der Trapezregel steht in Scilab die Funktion `inttrap` zur Verfügung.

MATLAB, Octave	Scilab												
 <pre data-bbox="204 694 790 840"> % Numerische Ableitung: &gt;&gt; z_diff = diff(y1)./diff(x); &gt;&gt; xd = x(1:end-1); &gt;&gt; subplot(3,1,3), plot(xd,z_diff,'r'); &gt;&gt; grid on;                     </pre>													
<p><b>Anpassung von Messwerten durch Polynome</b></p>													
<p>Polynome eignen sich hervorragend zur Anpassung von Messkurven.                  Beispiel : <i>Ausgleichsgerade</i>                  Bei einer Messung erhält man die folgende Tabelle</p> <table border="1" data-bbox="210 1108 778 1187"> <tr> <td><i>x</i></td> <td>2</td> <td>4</td> <td>6</td> <td>8</td> <td>10</td> </tr> <tr> <td><i>y(x)</i></td> <td>3.11</td> <td>6.01</td> <td>8.97</td> <td>11.01</td> <td>14.89</td> </tr> </table> <p>Von den Messwerten sei bekannt, dass sie auf einer Geraden liegen müssen. Die Lage dieser Geraden ist so zu bestimmen, dass die Abstände der Messpunkte hiervon einen minimalen quadratischen Fehler haben.                  Diese Aufgabe kann mittels <code>polyfit</code> gelöst werden.</p> <pre data-bbox="204 1444 790 1859"> &gt;&gt; X = [2 4 6 8 10] X =     2    4    6    8   10 &gt;&gt; Y =     3.11 6.01 8.97 11.01 14.89 &gt;&gt; Y = [3.11 6.01 8.97 11.01 14.89] Y =     3.1100    6.0100    8.9700   11.0100 14.8900 &gt;&gt; y_appr = polyfit(X,Y,1) y_appr =     1.4280    0.2300                     </pre> <p>Die gesuchte Gerade hat daher die Gestalt</p> $y = mx + b = 1,428x + 0,23 .$	<i>x</i>	2	4	6	8	10	<i>y(x)</i>	3.11	6.01	8.97	11.01	14.89	<p>In Scilab ist dies etwas umständlicher:</p> <pre data-bbox="813 929 1396 1724"> --&gt;X = [2 4 6 8 10] X =     2.  4.  6.  8. 10. --&gt;Y = [3.11 6.01 8.97 11.01 14.89] Y =     3.11 6.01 8.97 11.01 14.89 --&gt;Z = [X;Y] Z =     2.    4.    6.    8.    10.     3.11 6.01 8.97 11.01 14.89 def('e=G(a,z)', 'e=z(2)-a(1)-a(2)*z(1)'); // Definition einer On-line-Funktion zur Approximation p0=[1;1]; // Anfangswert  [p,er] = datafit(G,Z,p0); // scilab-Funktion datafit(G,Z,p0) mit G als Polynom 1. Grades --&gt;[p,er] = datafit(G,Z,p0) er =     0.59392 // Fehlerwert  p =     0.2300002     1.428                     </pre> <p>Das Ergebnis ist dasselbe wie bei MATLAB.                  Die gesuchte Gerade hat die Gestalt</p> $y = mx + b = 1,428x + 0,23$ <p>Eine genaue Entsprechung zu <code>polyfit</code> existiert in Scilab nicht.</p> <p>Die Anpassung von Messwerten durch eine Gerade kann auch nach folgendem Schema als Lösung eines überbestimmten linearen Gleichungssystems ermittelt werden (Das funktio-</p>
<i>x</i>	2	4	6	8	10								
<i>y(x)</i>	3.11	6.01	8.97	11.01	14.89								

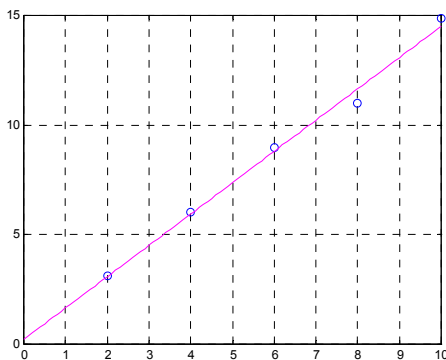
MATLAB, Octave

Scilab

Die Gerade soll zusammen mit den drei Messpunkten in ein Bild gezeichnet werden (Messpunkte als Kreise ('o')) und die Ausgleichsgerade dazu.

```

» x = 0:.1:10;
» plot(X, Y, 'o'); grid on; hold on;
% zeichnet Bild der fünf Messpunkte
» plot(x,polyval(y_appr,x),'m')
% zeichnet Gerade
    
```



nicht allerdings in MATLAB ebenso!

```

-->X = [2 4 6 8 10]
X =
    2. 4. 6. 8. 10.
-->Y = [3.11 6.01 8.97 11.01 14.89]
Y =
    3.11 6.01 8.97 11.01 14.89

-->A = [X' ones(length(X),1)]
A =
    2.    1.
    4.    1.
    6.    1.
    8.    1.
   10.    1.

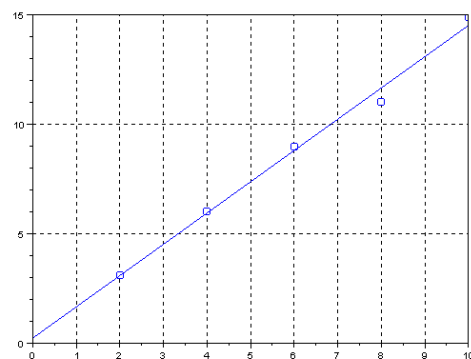
-->mb = A\Y'
mb =
    1.428
    0.23
// Lösung eines überbestimmten LGS
-->bm = mtlb_fliplr(mb')
bm =
    0.23    1.428
// Transponieren und Vertauschen der Reihenfolge
-->y1 = poly(bm,'x','c')
y1 =
    0.23 + 1.428x
    
```

Dies führt zu einer Geradengleichung

$$y = mx + b = 1,428x + 0,23 .$$

```

-->plot(X, Y, 'o'); xgrid;
// zeichnet Bild der fünf Messpunkte
-->x = 0:.1:10;
// kontinuierlich verteilte x-Werte eingeben
-->y = horner(y1, x);
-->plot(x, y);
// Approximationsgerade berechnen und zeichnen
    
```



## 5. Script-Dateien und Funktionen

### 5.1. Script-Dateien

#### 5.1.1. Grundsätzliches

Script-Dateien (in MATLAB und Octave: M-Files, „dateiname.m“, in Scilab: „dateiname.sce“ oder „...sci“ sind Folgen von Befehlen, die als ausführbare Programme mit einem einzigen Kommando aufgerufen und verwendet werden können. Dadurch muss zum Beispiel nicht jedes Kommando einer Befehlsfolge erneut eingegeben werden, wenn die Folge wiederholt werden soll.

Wir müssen *Script-Files* von *Funktions-Files* unterscheiden. Bei letzteren bleiben die Variablen „gekapselt“, sind also im aufrufenden Programm bzw. auf der Kommandoebene nicht verfügbar. Wir kommen auf sie noch zurück.

Script-Dateien speichern die Variablen im Workspace von MATLAB oder Scilab ab. Die in der Script-Datei benutzten Variablen können also nicht nur innerhalb der Script-Datei selbst, sondern auch anschließend von der Kommandozeile aus aufgerufen werden. Der Aufruf einer Script-Datei erfolgt in MATLAB einfach durch Angabe des Dateinamens, in Scilab durch `exec(„file_name.sce“)` oder `exec „file_name.sce“`<sup>7</sup>. Script-Dateien in MATLAB *müssen* die Endung `.m` aufweisen, Script-Dateien in Scilab sollten, müssen aber nicht, die Endung `.sce` besitzen.

Der Aufruf einer Script-Datei sollte jedoch in Scilab mit Semikolon abgeschlossen werden, denn andernfalls werden auf der Kommandozeile bei der Ausführung alle Zeilen der Script-Datei wiedergegeben, einschließlich der Kommentare. Mit nachgestelltem Semikolon geschieht die Ausgabe wie bei MATLAB.

MATLAB, Octave	Scilab
<p>Die Script-Datei <code>testscript.m</code> führt folgende Operationen aus</p> <pre>% testscript.sci a = 1; b = 2; a + b</pre> <p>Sie wird ausgeführt nach Eingabe des Befehls <code>testscript</code>. In der Kommandozeile erscheint</p> <pre>&gt; testscript ans =     3</pre> <p>Ein Semikolon nach dem Ausführungsbefehl bringt in MATLAB keine Veränderung:</p> <pre>&gt; testscript; ans =</pre>	<p>Die Script-Datei <code>testscript.sce</code> führt folgende Operationen aus:</p> <pre>// testscript.sci a = 1; b = 2; a + b</pre> <p>Sie wird ausgeführt nach Eingabe des Befehls <code>exec testscript.sci</code>. In der Kommandozeile erscheint</p> <pre>--&gt;// testscript.sci --&gt;a=1; --&gt;b=2; --&gt;a+b ans =     3.</pre> <p>Wird der Befehl <code>exec testscript.sci</code></p>

<sup>7</sup> Der Dateiname kann beim Aufruf mit oder ohne Klammer stehen sowie mit oder ohne Anführungszeichen. Statt Anführungszeichen kann der Name auch in Hochkomma eingeschlossen werden. Hier ist Scilab sehr tolerant.

3	mit Semikolon abgeschlossen, so wird nur das Ergebnis angezeigt: <pre>--&gt;exec testscript; ans =     3.</pre>
---	--

Die Befehlsfolge einer Script-Datei stellt ein (einfaches) Programm im Sinne der Informatik dar. Es sollte deshalb üblicherweise folgende Abschnitte enthalten:

- ein Teil zum Vorgeben bzw. Einlesen von Werten oder Parametern,
- ein Rechenteil,
- ein Ausgabeteil (Werte an Kommandoebene oder Graphik).

### 5.1.2. Einrichten des Arbeitsverzeichnisses in MATLAB/Octave

Vor Aufruf einer Script-Datei (wie auch einer Funktion) muss zunächst das aktuelle Arbeitsverzeichnis eingestellt werden. Dies geschieht in *MATLAB* mittels `Current Directory >` (*Verzeichnis*). Außerdem besteht die Möglichkeit, dieses Verzeichnis zu den Zugriffspfaden hinzuzufügen. Auch in *Octave UI* lässt sich das aktuelle Arbeitsverzeichnis über den Menübefehl „Working Directory“ ändern.

Der aktuelle Suchpfad kann in *MATLAB* oder *Octave* mit `path` angezeigt werden. Ein neuer Pfad wird mit dem Befehl `addpath` hinzugefügt, zum Beispiel durch

```
addpath("C:/Programme/Octave/share/octave/3.0.1/m/phys").
```

Wenn in *Octave* ohne graphische Benutzeroberfläche gearbeitet wird, müssen alle Eingaben in der Kommandozeile ausgeführt werden. Da die Eingabe eines Suchpfad-Befehls bei längeren Verzeichnisbäumen sehr empfindlich gegenüber Tippfehlern ist, empfiehlt es sich, besser die zu bearbeitende Datei unter *Windows* schon vorher ins Arbeitsverzeichnis von *Octave* zu kopieren. So ein Vorgehen kann auch beim Arbeiten mit älteren *MATLAB*-Versionen vorteilhaft sein.

### 5.1.3. Einrichten des Arbeitsverzeichnisses in Scilab

In *Scilab* ist der Zugriffspfad anfangs immer auf den Pfad eingestellt, in dem das Programm gestartet wurde. Das kann der Desktop sein, wenn man dort das entsprechende Icon angeklickt hat. In vielen Fällen wird die Datei, welche die benötigte Funktion enthält, jedoch in einem anderen Verzeichnis liegen. Den Suchpfad kann man mit `getenv` ermitteln und mit dem von *DOS* her bekannten Befehl `chdir` (äquivalent: `cd`) modifizieren:

```
-->getenv('sci')
ans =
    C:/PROGRA~1/scilab-5.1
cd("C:/Programme/Octave/share/octave/3.0.1/m/phys")
```

Vor dem Aufruf der Datei muss deshalb dieser Befehl eingegeben werden oder stattdessen das Suchverzeichnis mit dem Menübefehl

File > Change current directory

dorthin umgestellt werden, wo die zu bearbeitende Datei liegt. Wie Sie wissen, lässt sich jedoch das Arbeitsverzeichnis auch unter Windows einstellen. Dazu ist mit der rechten Maustaste das Icon anzuklicken und unter

„Eigenschaften > Ausführen in“

das Arbeitsverzeichnis anzugeben. Eine zusätzliche Möglichkeit ist die Einrichtung eines „Startup-Files“. Es trägt die Bezeichnung `scilab.ini` und wird beim Aufruf von Scilab automatisch geladen. Dieses Startup-File muss die Kommandozeile

```
chdir("(neues Arbeitsverzeichnis)");
```

enthalten, sie muss mit einem Zeilenvorschub abgeschlossen werden. Alternativ kann es auch sinnvoll sein, den Pfad der selbst erstellten Dateien in die Liste der Verzeichnisse aufzunehmen, die Scilab durchsucht. Am besten sollte man jedoch Scilab gleich von dem Verzeichnis aus starten, in dem auch die jeweilige Funktionsdatei liegt. Dies geschieht beispielsweise durch Doppelklick auf den entsprechenden Dateinamen oder das Icon unter Windows.

Danach wird mit dem Befehl `File > Execute` die Datei, ihr Name sei hier `myprog.sci`, in den Arbeitsspeicher geladen. Damit hat man Zugriff auf alle in dieser Datei enthaltenen Funktionen. Anstatt über den Menübefehl `Execute` zu gehen, kann man die Datei auch mit Hilfe der Funktion `getf` unter Angabe des kompletten Suchpfades laden:

```
getf('C:\Daten.....\myprog.sci')
```

Dieser Befehl lädt alle Funktionen, die in der Datei `myprog.sci` deklariert wurden. Anstelle von `getf` kann man auch den Befehl `exec` verwenden, und zwar, wie schon oben erwähnt, zum Beispiel in der Form

```
exec('myprog.sci').
```

#### 5.1.4. Einfaches Beispiel in MATLAB

Als *Beispiel* soll ein Demo-Programm zur Umrechnung von Radiant in Grad dienen:

Der Wert des Winkels in Radiant wird über den Befehl `input` eingegeben. Die formatierte Ausgabe erfolgt über `sprintf` und `disp`. Zur Illustration sind auch einige Beispiele für Ausgabeformate angegeben. Kommentare werden übrigens, wie schon häufig benutzt, in MATLAB mit dem Prozentzeichen `%`, in Scilab mit Doppelstrich `//` und in Octave mit Prozentzeichen oder doppeltem Balkenkreuz `##` eingeleitet.

```
% rad_grad2.m
% M-File zur Umwandlung von Radiant in Winkelgrad
% keine Angabe von Winkelminuten und Winkelsekunden, sondern von Zehntel-
% und Hunderstelgrad

clear;
% Werte eingeben
% *****
disp('Umrechnung eines Winkels aus Radiant in Grad')
alpha_rad = input('Winkel in Radiant eingeben: ');
```



```

% Rechnung
% *****
alpha_grad = alpha_rad * 360/2/pi;

% Ausgabe von Text (formatierte Ausgabe)
% *****
string1 = sprintf('Winkel in Grad = %4.3f °\n', alpha_grad);
disp(string1);
% Alternativ:
disp('Festkommaformat:');
    sprintf('Winkel in Grad = %4.3f °\n', alpha_grad);
    % Festkommaformat: 4 Stellen insgesamt, 3 Nachkommastellen
% Andere Formate:
disp('Festkommaformat:');
    sprintf('Winkel in Grad = %4.0f °\n', alpha_grad);
    % Festkommaformat: 4 Stellen insgesamt, keine Nachkommastellen
disp('Dezimalformat:');
    sprintf('Winkel in Grad = %4.1d °\n', alpha_grad);
disp('Exponentialformat:');
    sprintf('Winkel in Grad = %4.3e °\n', alpha_grad);
    % Exponentialformat

```

Zuweilen kommt es vor, dass ein Eingabewert nicht immer benötigt wird und stattdessen ein Default-Wert benutzt werden soll. In diesem Falle leistet die Funktion `isempty` gute Dienste. Sie fragt ab, ob der Eingabewert (in unserem Falle `alpha_rad`) evtl. leer ist und ordnet ihm dann einen Standardwert zu. In unserem Beispiel hat das zwar nicht allzu viel Sinn – es dient nur zur Illustration, aber in vielen anderen Fällen kann so etwas sehr nützlich sein.

```

alpha_rad = input('Winkel in Radiant eingeben: ');
if isempty(alpha_rad)
    alpha_rad = pi/2; disp('    pi/2');
end

```

## 5.2. Funktionen

### 5.2.1. Allgemeines über Funktionen

Neben den Script-Dateien gibt es auch Dateien, die Funktionen beinhalten. Funktionen stellen im Gegensatz zu Script-Dateien Programme dar, die nur über ganz dedizierte Schnittstellen ihre Werte mit dem aufrufenden Programmteil oder der Kommandoebene austauschen. Grundsätzlich bestehen auch Funktionen aus den Teilen *Eingabe* -> *Rechenteil* -> *Rückgabe*. Sowohl mit MATLAB als auch in Scilab oder Octave werden bereits zahlreiche vordefinierte Funktionen mitgeliefert. Es handelt sich um sogenannte *Built-in-Funktionen*. Deren Namen stellen *Schlüsselwörter* dar, die nicht anderweitig verwendet werden sollten, zum Beispiel `sin`, `tan`, `sqrt` usw. Darüber hinaus ist es dem Anwender möglich, eigene *anwenderdefinierte Funktionen* zu schreiben und zu verwenden.

Interne Variablen einer Funktion sind von außen (also im Workspace von MATLAB, Octave oder Scilab) nicht verfügbar.

In einer anwenderdefinierten Funktion muss die erste Zeile in der Funktionsdatei die *Deklaration* der Funktion enthalten. Dies geschieht nach dem folgendem Schema:

```
function alpha_grad = rad_grd(alpha_rad)
```

↑ Schlüsselwort (key word)

↑ Rückgabewert (output argument)

↑ Funktionsname (function name)

↑ Eingabewert (input argument)

Im Unterschied zu Java, C usw. ist in MATLAB oder Scilab keine Variablendeklarationen erforderlich.

Als einfaches Beispiel soll hier zur Illustration das Programm zur Umrechnung von Radiant in Grad angegeben werden, doch diesmal nicht als Script-File, sondern als Funktion geschrieben:

```
function alpha_grad = rad_grd(alpha_rad)
% Funktions-M-File zur Umwandlung von Radiant in Winkelgrad

alpha_grad = alpha_rad * 180/pi;
```

Für eine Funktion können auch mehrere Rückgabewerte vereinbart werden, sie stehen dann in eckigen Klammern, das heißt, sie werden als Vektor ausgegeben:

```
function [grad,min,sec] = rad_grd1(alpha_rad)
% Funktions-M-File zur Umwandlung von Radiant in Winkelgrad
% mit Angabe von Grad, Minuten und Sekunden
alpha_grad = alpha_rad * 180/pi;
grad = fix(alpha_grad);
minuten = (alpha_grad - grad)*60;
min = fix(minuten);
sec = (minuten - min)*60;
```

Die auf den Funktionsnamen folgenden ersten Kommentarzeilen erscheinen übrigens nach einem `help`-Aufruf im Kommando-Fenster von MATLAB. Gleiches gilt übrigens für Script-Dateien.

### 5.2.2. Funktionsaufrufe

In MATLAB muss jede Funktion in einer eigenen Datei stehen, der Dateiname muss mit dem Namen der Funktion übereinstimmen. Er hat wie bei Script-Files die Endung `.m`. In Scilab dagegen können mehrere Funktionen in einer Datei zusammengefasst werden. Diese Datei trägt die Endung `.sci` oder `.sce`. In MATLAB und Octave wird beim Aufruf die Funktion durch die Angabe des Funktionsnamens (ohne die Endung `.m`) auf der Kommandozeile automatisch geladen und ausgeführt. In Scilab muss dagegen eine Funktionsdatei erst eingebunden werden. Hierzu benutzt man den Befehl `getf` oder wieder `exec`, zum Beispiel mit

```
getf('datei.sci').
```

Das erinnert an die Vorgehensweise in der Programmiersprache C. [12]

Wie erwähnt, kann in Scilab eine Datei durchaus mehrere Funktionen enthalten. Nehmen wir zum Beispiel eine solche Datei mit dem Namen `myfunct.sce`. In ihr könnten die drei Funktionen `f1`, `f2` und `x2exp` enthalten sein, wie das folgende Beispiel illustriert.

MATLAB, Octave	Scilab
<p>In MATLAB ist für jede Funktion eine eigene Datei erforderlich, zum Beispiel: Die Datei <code>f1.m</code> enthält</p> <pre>function y = f1(x)     y = 2 .* x;</pre> <p>Die Datei <code>f2.m</code> enthält</p> <pre>function y = f2(x)     y = x.^2;</pre> <p>Die Datei <code>x2exp.m</code> enthält</p> <pre>function y = x2exp(x)     y = x.^2-exp(x);</pre>	<p>In Scilab kann eine Datei (zum Beispiel <code>myfunct.sce</code>) mehrere Funktionen enthalten:</p> <pre>function y = f1(x)     y = 2 .* x; endfunction  function y = f2(x)     y = x.^2; endfunction  function y = x2exp(x)     y = x.^2-exp(x); endfunction</pre>

Der Abschluss mit `endfunction` in Scilab kann auch entfallen. Da in Scilab jedoch mehrere Funktionen in einer Datei stehen können, ist es empfehlenswert, ihn beizubehalten. Auf die einzelne Funktionen kann, nachdem die Funktionsdatei geladen wurde, wie folgt zurückgegriffen werden:

MATLAB, Octave	Scilab
<pre>&gt;&gt; x = 5; &gt;&gt; y = x2exp(x) y = -123.4132</pre>	<pre>--&gt;getf ('myfunct.sce') --&gt;x = 5; --&gt;y = x2exp(x) y = - 123.41316</pre>

Wenn Funktionen sehr kurz sind, können sie auch als *Inline-Funktionen* (Scilab: *Online-Funktionen*) in einer einzigen Zeile definiert werden, zum Beispiel auf der Kommandozeile oder in einer Zeile eines Script-Files oder der aufrufenden Funktion. Dadurch erspart man es sich, eine separate Funktionsdatei anzulegen.

MATLAB, Octave	Scilab
<p>Inline-Funktion in MATLAB:</p> <pre>&gt;&gt; f1 = inline('x.^2-exp(x)') f1 =     Inline function:     f1(x) = x.^2-exp(x)</pre> <p>Die Inline-Funktion kann wie jede andere Funktion ausgewertet werden:</p> <pre>&gt;&gt; x = 5; &gt;&gt; y = f1(x) y = -123.4132</pre>	<p>Die Definition einer Online-Funktion in Scilab ist etwas komplizierter. Dazu benutzt man den Befehl <code>deff</code>.</p> <pre>a = deff('y = newfunct(x)', 'y = sin(x) + x')</pre> <p>In Scilab ist es darüber hinaus möglich, Funktionen unmittelbar über einen einzeiligen Funktionsstring zu definieren:</p> <pre>--&gt;function y=ff(x); y = x.^2-exp(x);endfunction</pre> <p>Der Abschluss mit <code>endfunction</code> ist dabei unbedingt notwendig.</p> <p>Auswertung der Online-Funktion:</p> <pre>--&gt;x = 5; --&gt;y=ff(x) y = - 123.41316</pre>

Eine Alternative ist die Deklaration einer Funktion über einen Funktionsstring. Ihn wertet man mit dem Befehl `eval` aus.

MATLAB, Octave	Scilab
<pre> &gt; f2='x.^2-exp(x)' f2 = x.^2-exp(x) &gt; x = 5; &gt; f = eval(f2) f = -123.4132                     </pre>	<pre> --&gt;f2='x.^2-exp(x)' f2 = x.^2-exp(x) --&gt;x = 5; --&gt;fy = eval(f2) fy = - 123.41316                     </pre>

Abschließend noch ein weiteres Beispiel für eine Online-Funktion in Scilab:

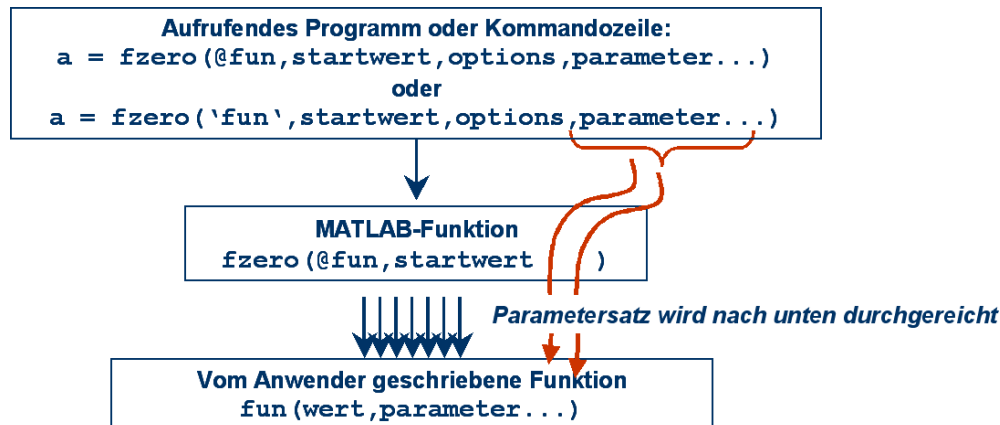
```

-->deff('[y]=f(x)', 'y = x.^2-exp(x)')
-->x = 5;
-->y = f(x)
y =
- 123.41316
    
```

### 5.2.3. Funktionen von Funktionen

Oftmals werden mit Hilfe einer Built-in-Funktion andere, zum Beispiel selbst erstellte Funktionen aufgerufen. Ein Beispiel ist die Funktion `fzero` in MATLAB (`fsolve` in Scilab). Sie dient dazu, die Nullstelle einer anderen Funktion zu bestimmen.

Die Wirkungsweise des Aufrufs von Funktionen über Funktionen sieht man am Beispiel von `fzero` unter MATLAB:



Die Funktionen werden gemäß folgender Syntax aufgerufen:

```
>> fzero('fun', alpha_start, options, ...)
```

Hier soll dies am Beispiel von  $x^2 - e^x = 0$  demonstriert werden. Die Nullstelle dieser Funktion ist nicht elementar bestimmbar, deshalb ist eine numerische Lösung unbedingt erforderlich.

MATLAB, Octave: <code>fzero</code>	Scilab: <code>fsolve</code>
Die Syntax sieht vor, dass beim Aufruf von <code>fzero</code> der Name der Funktionsdatei in Hochkomma angegeben wird, also	Vorher mit „Change current directory“ das aktuelle Suchverzeichnis auf den Pfad umstellen, in dem die aktuelle Datei liegt. Dann

MATLAB, Octave: <code>fzero</code>	Scilab: <code>fsolve</code>
<pre>&gt;&gt; fzero('x2exp',2) ans = -0.7035</pre> <p>Ab MATLAB 6.5 ist es auch möglich, die Funktion mittels eines Zeigers (ähnlich wie in C) aufzurufen:</p> <pre>&gt;&gt; fzero('x2exp',2) ans = -0.7035</pre> <p>Darüber hinaus kann die Funktion (wenn sie durch einen Befehl ausgedrückt werden kann) durch direkte Eingabe des Funktionsstrings angegeben werden:</p> <pre>&gt;&gt; fzero('x.^2-exp(x)',2) ans = -0.7035</pre> <p>(nicht möglich in Octave) oder alternativ über die Definition einer Inline-Funktion:</p> <pre>&gt;&gt; x_2exp=inline('x.^2-exp(x)') x_2exp =     Inline function:     x_2exp(x) = x.^2-exp(x) &gt;&gt; fzero(x_2exp,2) ans = -0.7035</pre>	<p>im Pull-Down-Menü „File &gt; execute“ ausführen. Alternativ ist es möglich, den Befehl</p> <pre>exec('x2exp.sci')</pre> <p>über die Kommandozeile eingeben. Danach lässt sich folgendes Problem lösen:</p> <pre>--&gt;fsolve(2,x2exp) ans = - 0.7034674</pre> <p>Es ist jedoch auch möglich, eine Online-Funktion über <code>def</code> zu definieren und darauf mittels <code>fsolve</code> zurückzugreifen:</p> <pre>def('[y]=f(x)', 'y = x.^2-exp(x)') --&gt;def('[y]=f(x)', 'y = x.^2-exp(x)') --&gt;fsolve(2,f) ans = - 0.7034674</pre>

*Beispiel:* Nullstellensuche für die um  $-1/2$  verschobene „Höckerfunktion“

$$y = \frac{1}{1+x^2} - \frac{1}{2}$$

```
function y = hoecker(x)
% Höckerfunktion in Abhängigkeit von einem Parameter a
% und einem möglichen y-Wert y1

y = 1./(1+x.^2);
```

Die Nullstelle wird direkt gesucht mit

```
>> a = fzero('hoecker',1)
a =
    1
```

Die hier vorgestellte Art des Funktionsaufrufs mittels Hochkomma beinhaltet die Werteübergabe als „Call by value“, das heißt, die Daten werden in eigene Speicherbereiche übergeben. Effektiver ist die in MATLAB ab Version 6.5 und ebenfalls in Octave vorgesehene Möglichkeit der Datenübergabe als „Call by reference“. Hierbei wird dem ausführenden Programm (im Beispiel dem Programm `fzero`) lediglich der Speicherort der zu bearbeitenden Funktion mitgeteilt (im Beispiel der Ort der Funktion `hoecker.m`). Das erinnert an die Vorgehensweise in C, wenn dort Zeiger benutzt werden. Für die Übergabe des Speicherorts wird ein sogenanntes Handle benutzt, symbolisiert durch das Symbol `@`. Die Nullstellensuche für die oben definierte Höckerfunktion wird dann so formuliert:

```
>> a = fzero(@hoecker,1)
a =
    1
```

Als weitere Eingabedaten können in MATLAB zusätzliche Optionen als Information zur Ausführung des Programms eingegeben werden. Default-Optionen werden durch „optimset“ geliefert.

```
>> a = fzero('hoecker',1,optimset)
```

Wenn man die Funktion etwas erweitert und einen Parameter  $a$  mitnimmt,

$$y = \frac{1}{1+x^2} - \frac{1}{2} - ax,$$

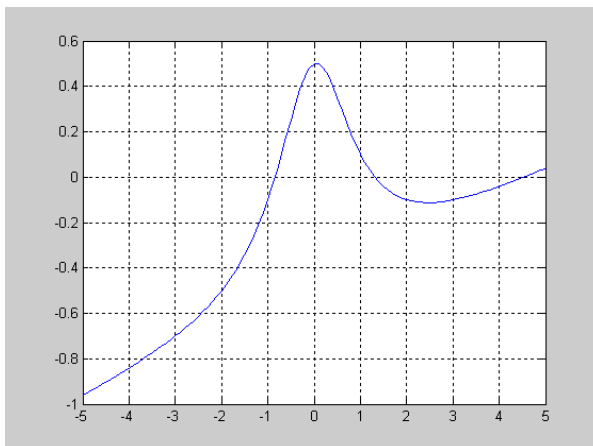
ergibt sich

```
function y = hoecker(x,a)
% Höckerfunktion in Abhängigkeit von einem Parameter a
y = 1./(1+x.^2) + a*x ;
```

Dieser zusätzliche Parameter wird bei der Nullstellensuche als nächste Übergabevariable in die Funktion `fzero` eingefügt, sie steht dann nach `optimset`. Die Nullstelle ist jetzt gegeben durch:

```
>> a = fzero('hoecker',1,optimset,.1)
a =
    1.3068
```

(vgl. Abbildung 4):



**Abbildung 4** „Höckerfunktion“ für Parameter  $a = 0,1$

```
function y = hoecker2(x,a,y1)
% Höckerfunktion in Abhängigkeit von einem Parameter a
% und einem moeglichen y-Wert y1
y = 1./(1+x.^2) - .5 + a*x - y1;
```

Ebenso wie eine Nullstelle kann auch ein  $x$ -Wert berechnet werden, für den der zugehörige Funktionswert  $y(x)$  gegeben ist. Letztlich läuft diese Fragestellung nur auf eine Verschiebung der  $x$ -Achse hinaus, so dass jetzt die Nullstelle der verschobenen Funktion  $y(x) = y(x) - y_{\text{ziel}}$  gesucht wird.

Neben der Nullstellensuche gibt es noch andere Fälle, in denen „Funktionen von Funktionen“ untersucht werden müssen. Solche Probleme sind zum Beispiel:

`fzero` – Nullstellensuche

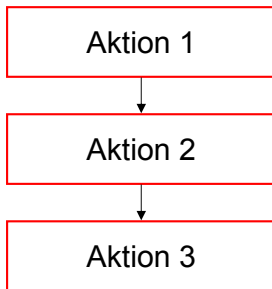
`fminbnd` – Minimumsuche

quad – Integration

ode45 – Lösung von Differentialgleichungen

### 6. Kontrollstrukturen

Der Begriff *Kontrollstrukturen* (als Übersetzung des englischen *control structures*) hat sich im Deutschen zwar durchgesetzt, er sollte jedoch eigentlich besser durch „Steuerstrukturen“ bezeichnet werden. Ein normaler Programmablauf besteht aus einer linearen, unbedingten Folge von Anweisungen (Abbildung 5).

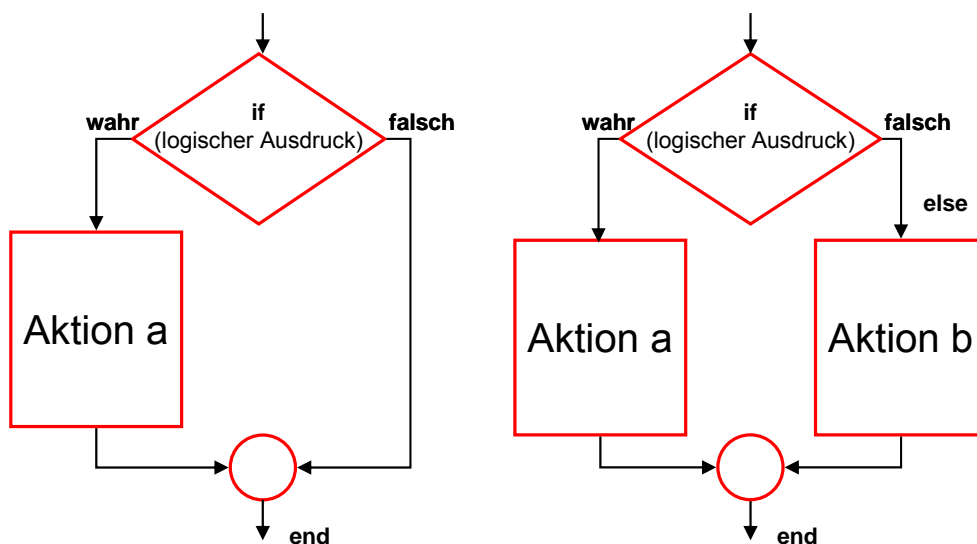


**Abbildung 5** Linearer Ablauf von Aktionen

Durch bedingte Anweisungen und Wiederholungen, eben die „Kontrollstrukturen“, können Abläufe strukturiert werden. [17], [18].

#### 6.1. Die if-Bedingung

Mit der if-Bedingung werden Verzweigungen strukturiert. In der folgenden Abbildung 6 ist das deutlich gemacht: links: die einseitige Verzweigung, rechts: die zweiseitige Verzweigung.



**Abbildung 6** Struktur der if-Bedingung

Diese Verzweigungen werden ausgedrückt durch folgende Programmabläufe:

```

if ...
...
    
```

```
elseif ...% nicht zwingend erforderlich
...
else ...% nicht zwingend erforderlich
...
end
```

Für die Vergleiche benötigt man folgende *logische Operationen*:

- < kleiner als
- <= kleiner oder gleich
- > größer als
- >= größer oder gleich
- == gleich (Beachte: *zwei* Zeichen bei logischem Vergleich!)
- ~= nicht gleich

Darüber hinaus werden unter Umständen NOT (~), OR(|), AND (&) sowie XOR als weitere logische Operationen benötigt.

*Beispiel:* Ausschnitt aus einem Programm, in dem alle negativen Funktionswerte durch null ersetzt werden:

MATLAB, Octave	Scilab
<pre>if y1 &lt; 0     y1 = 0; end</pre>	<pre>if y1 &lt; 0     y1 = 0; end</pre>
<p>Besonders für eine Eingabe über die Konsole ist auch die folgende Form vorteilhaft, bei der alles in einer Zeile ausgeführt wird.</p>	
<pre>if y1 &lt; 0, y1 = 0; end</pre>	<pre>if y1 &lt; 0, y1 = 0; end</pre>

### 6.2. Die switch-Bedingung

Die switch-Bedingung (Scilab: `select`) entspricht praktisch aufeinander folgenden `if-else`-Befehlen.

MATLAB, Octave	Scilab
<pre>% test_select.m  a = input('Geben Sie eine Zahl a zwischen 1 und 3 ein'); switch(a)     case 1         disp(a);     case 2         disp(a);     case 3         disp(a);     otherwise         disp('a war nicht 1,2, oder 3')         % break end</pre>	<pre>//test_select.sce  a = input('Geben Sie eine Zahl a zwischen 1 und 3 ein'); select(a)     case 1 then         disp(a);     case 2         disp(a);     case 3         disp(a); else     disp("a war nicht 1,2, oder 3")     //break end</pre>



### 6.3. Die for-Schleife

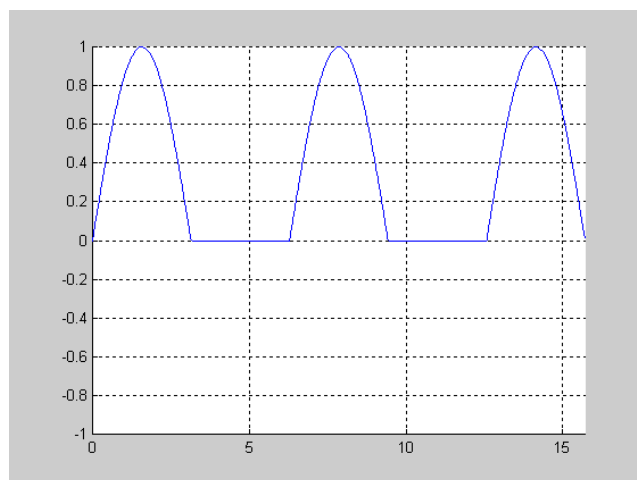
In der Informatik unterscheidet man Zählschleifen und Bedingungsschleifen. Eine Zählschleife wird verwendet, wenn bereits vor der Ausführung bekannt ist, wie oft die in ihr enthaltenen Anweisungen ausgeführt werden sollen. In den von uns betrachteten Sprachen wird sie als `for`-Schleife ausgeführt. Sie enthält deshalb eine numerische Schleifenvariable, die zu Beginn auf einen Startwert gesetzt und dann jeweils um die vorgesehene Schrittweite verändert wird, bis der Zielwert erreicht ist. In MATLAB und Scilab kann die `for`-Schleife jederzeit durch den Befehl `break` beendet werden.

Die Struktur der `for`-Schleife ist in MATLAB, Octave und Scilab gleich:

```
for i = 1:n
...
end
```

*Beispiel:* Zeichnen einer Kurve für eine gleichgerichtete Wechselspannung (Abbildung 7). Dabei wird außerdem die oben früher verwendete `if`-Abfrage benutzt.

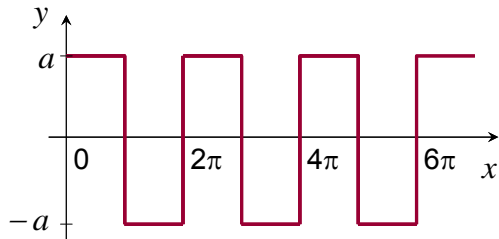
MATLAB, Octave	Scilab
<pre>% Programm sinustest.m % Gleichrichtung einer Sinusfunktion clf  axis([0 5*pi -1 1]); hold on % Skalierung des Bildes wird festgelegt x=[]; y=[]; for x1 = 0:.1:5*pi     y1 = sin(x1);     if y1 &lt; 0         % "Gleichrichtung"         y1 = 0;     end     y = [y,y1]; x=[x,x1]; % Fuegt zu x und y noch eine weitere Spalte hinzu end plot(x,y); grid;</pre>	<pre>// Programm sinustest.sce // Gleichrichtung einer Sinusfunktion clear();  mtlb_axis([0 5*pi -1 1]); // Skalierung des Bildes wird festgelegt x=[]; y=[]; for x1 = 0:.1:5*pi     y1 = sin(x1);     if y1 &lt; 0         // "Gleichrichtung"         y1 = 0;     end     y = [y,y1]; x=[x,x1]; // Fuegt zu x und y noch eine weitere Spalte hinzu end plot(x,y);xgrid;</pre>



**Abbildung 7** Gleichgerichtete Wechselspannung als Ergebnis des Programms `sinustest.m`

Eine weitere Anwendung von for-Schleifen stellt zum Beispiel die Approximation eines Signals durch Fourier-Reihen dar. Ein Rechtecksignal (Abbildung 8) beispielsweise wird durch die folgende Reihenentwicklung erzeugt:

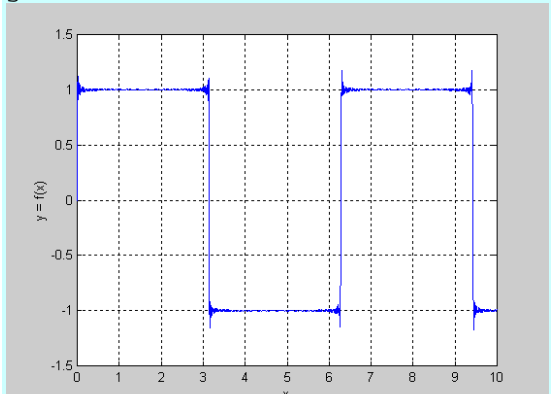
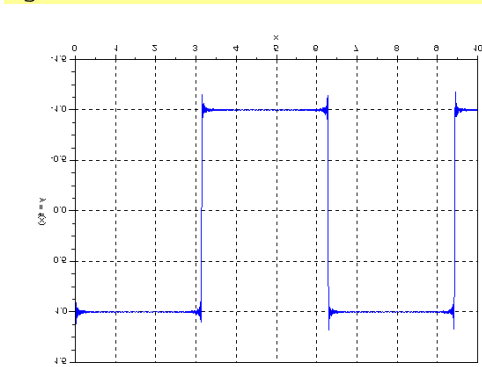
$$y = \frac{4a}{\pi} \left( \sin x + \frac{\sin 3x}{3} + \frac{\sin 5x}{5} + \dots \right)$$



**Abbildung 8** Rechtecksignal

Zur Umsetzung kann das folgende Programm benutzt werden:

MATLAB, Octave	Scilab
<pre> % Programm fourier.m % Fourierzerlegung eines periodischen Rechtecksignals close all % Arbeitsbereich reinigen und Abbildungen schließen:  % Werte eingeben % ***** a = 1; % 'Impulshöhe = 1) fend = input('Zahl der Sinuswellenzüge (Default: 100): '); if isempty(fend), fend = 100; end xend = input('Maximaler x-Wert (Default: 10): '); if isempty(xend), xend = 10; end xstep = 0.01;          % x-Inkrement  % Rechnung % ***** x = 0:xstep:xend; % x-Zwischenwerte bilden y = zeros(1,length(x)); % Startwerte: Überall y = 0  for f = 1:fend % Iteration     y = y + sin((2*f-1)*x)/(2*f-1); end y = 4*a/pi * y; % Normierung 4*a/pi  % Graphikausgabe % ***** </pre>	<pre> // Programm fourier.sce // Fourierzerlegung eines periodischen Rechtecksignals  close() // evtl. vorhandenes Bild loeschen  // Werte eingeben // ***** a = 1;          // Impulshöhe = 1 fend = input('Zahl der Sinuswellenzüge (Default: 100): '); if isempty(fend), fend = 100; end xend = input('Maximaler x-Wert (Default: 10): '); if isempty(xend), xend = 10; end xstep = 0.01;          // x-Inkrement  // Rechnung // ***** x = 0:xstep:xend; // x-Zwischenwerte bilden y = zeros(1,length(x)); //Startwerte: Überall y = 0  for f = 1:fend // Iteration     y = y + sin((2*f-1)*x)/(2*f-1); end y = 4*a/%pi * y; // Normierung 4*a/pi  // Graphikausgabe // ***** </pre>

MATLAB, Octave	Scilab
<pre>plot(x,y) xlabel('x'); ylabel('y = f(x)'); grid</pre> 	<pre>plot(x,y) xlabel('x'); ylabel('y = f(x)'); xgrid;</pre> 

*Hinweis:* Um Default-Werten festzulegen, kann man zweckmäßig die Funktion `isempty` verwenden:

```
if isempty(fend), fend = 100; end
```

Falls der Benutzer auf eine Eingabeaufforderung `input` keinen Wert eingegeben hat, ist die betreffende Variable „leer“, also nicht mit einem Zahlenwert belegt. Die Funktion `isempty` fragt genau diese Situation ab, und durch die `if`-Bedingung wird in diesem Falle ein Default-Wert zugewiesen.

Auch hier lässt sich übrigens das Programm wieder optimieren, wenn Schleifen vermieden werden (Diese Version geht auf R. Bartholomä zurück):

```
% Programm fourier.m
% Fourierzerlegung eines periodischen Rechtecksignals

% *****
% ursprünglicher Autor: R. Bartholomae

clear
close all % Arbeitsbereich reinigen und Abbildungen schließen:

% Werte eingeben
% *****
a = input('Impulshöhe (Default: 1): ', 1);
if isempty(a), a = 1; end
fend = input('Maximale Oberwelle (Default: 100): ', 100);
if isempty(fend), fend = 100; end
xend = input('Maximaler x-Wert (Default: 10): ', 10);
if isempty(xend), xend = 10; end
xstep = input('x-Inkrement (Default: 0.01): ', 0.01);
if isempty(xstep), xstep = .01; end

% calculate row vectors for f and vectors for x
f = 1:2:fend;
x = 0:xstep:xend;

% create a matrix FX which holds the required argumentes for the sine function
% FX(i,j) = f(i)*x(j)
```

```
FX = f' * x;
% calculate the sine value for each matrix element of FX and scale each row i
% with scaling a factor 1/f(i). This can be done by multiplying the resulting
% matrix sin(FX) with a diagonal matrix in which the diagonal elements equals
% the scaling factors.
Y = diag(1./f) * sin(FX);

% finally add alls rows of matrix Y resulting in a row vector which can be
ploted.
% dont forget the scaling factor 4*a/pi
y = 4*a/pi * sum(Y);

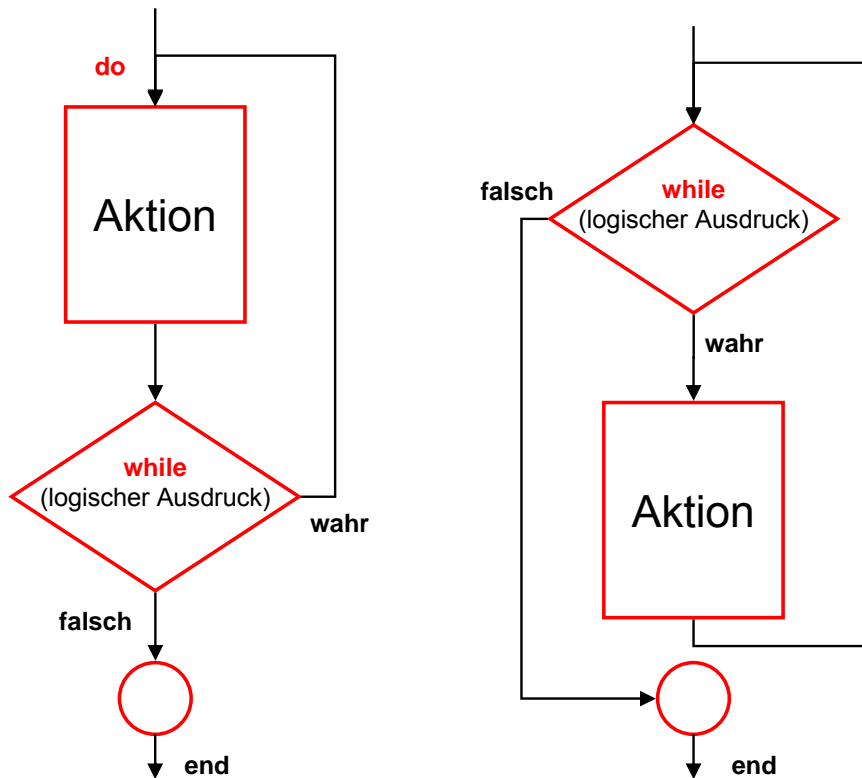
% Now plot the result
plot(x, y)
xlabel('x'); ylabel('y = f(x)'); grid
```

#### 6.4. Die while-Schleife

Prinzipiell unterscheidet man in der Informatik zwei Wiederholungsanweisungen für die Ausführung von Bedingungsschleifen: DO (Bedingungsprüfung am Schleifenende) und WHILE (Bedingungsprüfung am Schleifenanfang). Die DO-Schleife (Abbildung 9 links) ist eine so genannte „nichtabweisende Schleife“. Sie wird vor der Prüfung mindestens einmal durchlaufen. Bei MATLAB, Octave und Scilab ist sie nicht implementiert. Die andere Möglichkeit einer Schleife mit Prüfung am Schleifenanfang (Abbildung 9 rechts) steht dagegen als while-Schleife zur Verfügung. Es kann übrigens unter Umständen passieren, dass eine while-Schleife nicht ein einziges Mal durchlaufen wird. Die Programmierung der while-Schleife erfolgt nach folgendem Schema:

```
while ...
...
end
```

In MATLAB und Scilab kann die while-Schleife wie auch die for-Schleife jederzeit durch den Befehl `break` beendet werden.



**Abbildung 9** Schleife mit Prüfbedingung am Schleifenende (do ... while) bzw. am Schleifenanfang (while)

Als *Beispiel* soll noch einmal die Kurve des Programms `sinustest.m` aus 6.3 dargestellt werden, diesmal jedoch unter Verwendung einer `while`-Schleife.

MATLAB, Octave	Scilab
<pre> % Programm sinustest_while.m  clf axis([0 5*pi -1 1]); hold on % Skalierung des Bildes wird festgelegt x=[]; y=[]; for x1 = 0:.01:5*pi;     y1 = sin(x1);     while y1 &lt; 0         y1 = 0;     end     y = [y,y1]; x=[x,x1]; % Fuegt zu x und y noch eine weitere Spalte hinzu end plot (x,y); grid;                 </pre>	<pre> // Programm sinustest_while.sce  clear () mtlb_axis([0 5*pi -1 1]); // Skalierung des Bildes wird festgelegt x=[]; y=[]; for x1 = 0:.01:5*pi;     y1 = sin(x1);     while y1 &lt; 0         y1 = 0;     end     y = [y,y1]; x=[x,x1]; // Fuegt zu x und y noch eine weitere Spalte hinzu end plot (x,y); xgrid;                 </pre>

Die Bearbeitung dieses Problems mit einer `for`- oder `while`-Schleife ist jedoch nicht optimal für Programme wie MATLAB oder Scilab. Die erste Verbesserung würde darin bestehen, zu Beginn den für `x` und `y` benötigten Speicher bereitzustellen (zu „allokieren“). Die zweite und entscheidende Verbesserung besteht im kompletten Verzicht auf die gesamte Schleife. Dafür wird ausgenutzt, dass MATLAB und Scilab eigentlich für die Berechnung mit Matrizen und Vektoren ausgelegt sind

und gerade damit optimal arbeiten. In unserem Beispiel wird das erreicht, indem zuerst alle Sinuswerte berechnet und anschließend mit dem Befehl `find` diejenigen Werte herausgesucht werden, die kleiner als null sind.

MATLAB, Octave	Scilab
<pre> % Programm sinustest_opt.m % Gleichrichtung einer Sinusfunktion % optimale Berechnung ohne Schleifen  clf axis([0 5*pi -1 1]); hold on % Skalierung des Bildes wird festgelegt x = 0:.01:5*pi; % Festlegung des x-Vektors y = sin(x); ii = find (y &lt; 0); % sucht die Indizes aller y-Werte &lt; 0 heraus y(ii) = 0; % "Gleichrichtung" plot(x,y); grid;                     </pre>	<pre> // Programm sinustest_opt.sce // Gleichrichtung einer Sinusfunktion // optimale Berechnung ohne Schleifen  clear() mtlb_axis([0 5*pi -1 1]); // Skalierung des Bildes wird festgelegt x = 0:.01:5*pi; // Festlegung des x-Vektors y = sin(x); ii = find (y &lt; 0); // sucht die Indizes aller y-Werte &lt; 0 heraus y(ii) = 0; // "Gleichrichtung" plot(x,y); xgrid;                     </pre>

Diese Variante benötigt die geringste Rechenzeit. Allerdings wird in den neueren MATLAB-Versionen auch schon der übliche Schleifenablauf mittels des JIT-Accelerators (vgl. Kap. 8) teilweise optimiert.

Auch das bereits früher in 6.3 mittels `fourier.m` erzeugte periodische Rechtecksignal lässt sich mit einer `while`-Schleife gewinnen. Im folgenden Beispiel ist einmal das für MATLAB und Octave gezeigt. Damit haben wir gleich einmal die Möglichkeit, einige Besonderheiten aufzuzeigen, die bei der Octave-Programmierung möglich sind:

MATLAB/Octave	Octave (Octave-Besonderheiten sind rot unterlegt)
<pre> % Programm rechteck.m % Fourierzerlegung eines periodischen Rechtecksignals % ***** % Dieses Programm läuft sowohl unter MATLAB als auch unter Octave  close all % Arbeitsbereich reinigen und Abbildungen schließen:  % Werte eingeben % ***** a = 1; % Impulshöhe = 1 fend = input('Zahl der Sinuswellenzüge (Default: 100): ');     if isempty(fend), fend = 100; end xend = input('Maximaler x-Wert (Default: 10): ');     if isempty(xend), xend = 10; end xstep = 0.01; % x-Inkrement                     </pre>	<pre> # Programm rechteck_oct.m # Fourierzerlegung eines periodischen Rechtecksignals # ***** # Dieses Programm läuft wegen des Befehls f++ nur unter Octave  close all # Arbeitsbereich reinigen und Abbildungen schließen:  # Werte eingeben # ***** a = 1; # Impulshöhe = 1 fend = input('Zahl der Sinuswellenzüge (Default: 100): ');     if isempty(fend), fend = 100; end xend = input('Maximaler x-Wert (Default: 10): ');     if isempty(xend), xend = 10; end xstep = 0.01; # x-Inkrement                     </pre>

MATLAB/Octave	Octave (Octave-Besonderheiten sind rot unterlegt)
<pre> % Rechnung % ***** x = 0:xstep:xend; % x-Zwischenwerte bilden y = zeros(1,length(x)); % Startwerte: Überall y = 0  for f = 1:fend % Iteration     y = y + sin((2*f-1)*x)/(2*f-1); end y = 4*a/pi * y; % Normierung 4*a/pi  % Graphikausgabe % ***** plot(x,y) xlabel('x'); ylabel('y = f(x)'); grid </pre>	<pre> # Rechnung # ***** x = 0:xstep:xend; # x-Zwischenwerte bilden y = zeros(1,length(x)); # Startwerte: Überall y = 0 f=1; while f &lt;=fend # Iteration     y = y + sin((2*f-1)*x)/(2*f-1); f++; ## typisch für Octave end y = 4*a/pi * y; # Normierung 4*a/pi  # Graphikausgabe # ***** plot(x,y) xlabel('x'); ylabel('y = f(x)'); grid </pre>

## 7. Einfache Benutzer-Interfaces (GUI)

MATLAB stellt ein professionelles Werkzeug zur Entwicklung von graphischen Benutzeroberflächen bereit. Es trägt den Namen GUIDE und kann als eigene Toolbox erworben werden. Das Arbeiten damit ist trotzdem nicht unbedingt einfach. Hier sollen deshalb nicht alle Varianten zur Programmierung von Benutzerschnittstellen diskutiert werden. MATLAB und Scilab bieten jedoch bereits vordefinierte Dialogboxen an, die ohne vertieftes Eindringen in die Materie schon Hilfestellungen bieten können. Verständlicherweise kann Octave solche Möglichkeiten nicht bereitstellen, da es ein Programm ist, das auf der Ebene der Kommandozeile im DOS-Fenster arbeitet. Zum Glück reagiert Octave jedoch auf einige der in MATLAB verfügbaren Funktionen jedoch so, dass zumindest Alternativen zur Programmfortführung geboten werden. Im Unterschied zu echten graphischen Benutzeroberflächen verschwinden alle der hier vorgestellten Dialogboxen nach der Benutzung, und das Programm kehrt wieder zur Kommandozeile zurück.

Im folgenden sollen einige Möglichkeiten vorgestellt werden. Als Beispiel sollen die Werte einiger physikalischer Konstanten ausgewählt, angezeigt und auf der Kommandozeile weiter verwendet werden. Grundsätzlich werden in diesem Programm die Werte aller Konstanten verfügbar gemacht, unabhängig davon, welche Konstante angezeigt werden soll. Es handelt sich hier nur um eine Illustration, für die nur fünf Konstanten ausgewählt wurden.

### 7.1. Dialogbox „menu“

Anhand eines einfachen Programmbeispiels wird die Verwendung der Dialogbox `menu` beschrieben.

```

% Programm phys_testmenue.m
%*****
% Laden physikalischer Parameter und Konstanten in die Kommandooberfläche
% Der Wert einer gewaehlten Konstanten kann jeweils angezeigt werden
% Es werden jedoch trotzdem alle Konstanten geladen.

%*****
% Bereitstellen der Werte:
%*****
c = 2.99792458e8;      % Lichtgeschwindigkeit in m/s
q = 1.60217649e-19;   % Elementarladung in Amperesekunden
eV = q;               % 1 eV = {q}J = 1.60217653e-19 J
h_J = 6.6260693e-34; % Plancksche Konstante in Js
h = h_J/eV;          % Plancksche Konstante in eVs
% (Rest weggelassen)

%*****
% Auswahl zur Anzeige der Konstanten:
%*****
choice = menu ('Bereitstellung folgender physikalischer Konstanten. Bitte wählen
Sie eine',...
    'Lichtgeschwindigkeit c in m/s',...      %1
    'Elementarladung in As',...             %2
    'Joule in eV',...                       %3
    'Planck'sche Konstante h in Js',...     %4
% ... (weitere Daten hier weggelassen)
    'Verlassen');

const_string = ('Abbruch durch den Benutzer');
if isempty(choice); choice = 0; end % Fängt Verlassen durch Eingabe von "ctrl-
X" ab

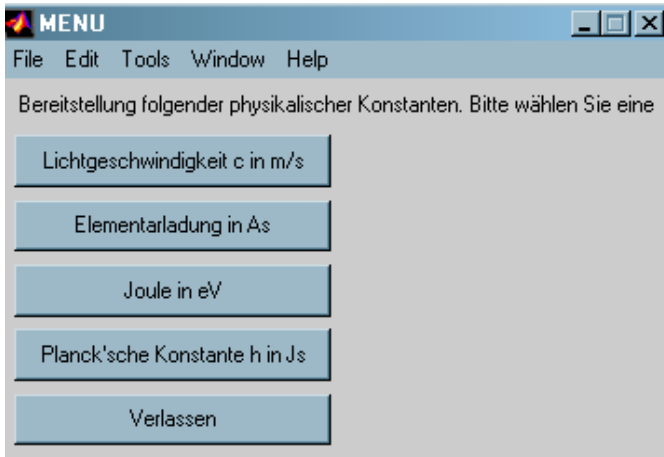
switch (choice)
case {1} % Lichtgeschwindigkeit in m/s
    const_string = sprintf('c = %4.4e m/s', c);
case {2} % Elementarladung in Amperesekunden
    const_string = sprintf('q = %4.4e As', q);
case {3} % 1 eV = {q}J = 1.60217653e-19 J
    const_string = sprintf('eV = %4.4e J', eV);
case {4} % Planck'sche Konstante in Js
    const_string = sprintf('h_J = %4.4e Js', h_J);
otherwise
end

disp(const_string);

```

Das zugehörige Menü sieht dann wie folgt aus:





Anstelle der Dialogbox wird in Octave eine Auswahl auf der Kommandozeile dargestellt.

```

octave:2> phys_testmenue
Bereitstellung folgender physikalischer Konstanten. Bitte wählen
Sie eine

[ 1] Lichtgeschwindigkeit c in m/s
[ 2] Elementarladung in As
[ 3] Joule in eV
[ 4] Planck'sche Konstante h in Js
[ 5] Verlassen

pick a number, any number: 2
q = 1.6022e-019 As
octave:3>
    
```

Eine Dialogbox analog zu menu in MATLAB ist in Scilab nicht bekannt. Lediglich die Funktion buttndialog entspricht dieser MATLAB-Funktion, wobei allerdings die Buttons in einer Zeile angeordnet sind (Für das hier gezeigte Beispiel wäre das jedoch sehr unhandlich!).

### 7.2. Eingabe-Listen „listdlg“ und „xchoose“

Anstelle der Dialogbox kann in MATLAB auch eine Eingabeliste verwendet werden. Hierzu gibt es eine entsprechende Möglichkeit in Scilab, jedoch keine in Octave.

MATLAB	Scilab
2. Eingabe-Listbox (listdlg) (keine Entsprechung in Octave)	2. Eingabe-Listbox (x_choose)
Die nicht angezeigten Zeilen entsprechen dem Programm phys_testmenue.m ... ... %***** % Auswahl zur Anzeige der Konstanten:	<pre> // Programm phys_testmenue.sce //***** //***** // Bereitstellen der Werte: //***** c = 2.99792458e8; q = 1.60217649e-19; n eV = q; h_J = 6.6260693e-34;  //***** // Auswahl zur Anzeige der Konstanten //***** choice=x_choose(['Lichtgeschwindigkeit c in m/s';...                 </pre>

MATLAB	Scilab
<pre> %***** str1 = ...     {'Lichtgeschwindigkeit c in m/s';... %1     'Elementarladung in As';...     %2     'Joule in eV';...     %3     'Planck'sche Konstante h in Js';... %4     'Verlassen'}; choice = listdlg ('PromptString','Bereitstellung folgender physikalischer Konstanten. Bitte wählen Sie eine',...     'SelectionMode','single',...     'ListSize',[350 260],...     'ListString',str1) ... </pre> 	<pre> 'Elementarladung in As'; 'Joule in eV';... 'Planck'sche Konstante h in Js';... 'Verlassen'],... ['Physikalische Konstanten';'Bitte wählen Sie eine zur Anzeige']);  const_string = ('Abbruch durch den Benutzer'); if isempty(choice); choice = 0; end // Fängt Verlassen durch "X" ab  select (choice) case (1)     const_string = sprintf('c = %4.4e m/s', c); case (2)     const_string = sprintf('q = %4.4e As', q); case {3}     const_string = sprintf('eV = %4.4e J', eV); case {4}     const_string = sprintf('h_J = %4.4e Js', h_J); end  disp(const_string); </pre>
<p>Diese Funktion ist in Octave nicht implementiert.!</p>	

## 8. Arbeitsgeschwindigkeit der einzelnen Programme

Die einzelnen Numerik-Programme benötigen unterschiedlich viel Rechenzeit. Sogar innerhalb ein- und desselben Programms unterscheiden sich von Version zu Version die Zeiten zum Teil erheblich.

Einen ersten Vergleich findet man zum Beispiel bei der Lösung eines linearen Gleichungssystems mit  $n$  Unbekannten. Die hierfür benötigte Rechenzeit ergibt sich rein theoretisch nach der Formel

$$z = \frac{n^3}{3} + n^2 - \frac{1}{3}n,$$

beinhaltet also für große  $n$  im Wesentlichen eine Proportionalität zu  $n^3$ . Die wichtigsten Ergebnisse solcher Untersuchungen, die bereits früher von Hamann [19] vorgenommen wurden, sind in folgender Tabelle zusammengefasst:

**Tabelle 1** Vergleich der Rechenzeiten verschiedener Programme (aus [19])

$n$	MATLAB 5.3	MATLAB 6.5	MATLAB 7.0	Scilab	Octave
500	0,28s	0,13s	0,08s	0,08s	0,59s
1000	2,05s	0,58s	0,48s	0,59s	0,87s
5000	238,95s	56,34s	43,16s	*)	137,95s
6000	414,24s	93,44s	74,11s	*)	265,83

\*) Abbruch wegen unzureichenden Stacks. Der Stack lässt sich in Scilab nicht beliebig erweitern.

Die verkürzte Rechenzeit bei MATLAB 6.5 und 7.0 gegenüber der früheren Version ist vor allem auf die Beschleunigung durch den so genannten JIT-Accelerator zurückzuführen<sup>8</sup> (JIT – Just in Time). Während die prozeduralen Sprachen der 3. Generation recht schnell übersetzt werden, durchläuft MATLAB als Programmiersprache der 4. Generation einen Interpreter. Dieser ist naturgemäß langsamer als ein Compiler. Ab Version 6.5 wird ein beschleunigter Interpreter benutzt, der so genannte JIT-Accelerator (JIT – „Just-In-Time Code Generation“). Er arbeitet insbesondere Schleifen rascher ab [20]. Dabei zeigt sich natürlich der Vorteil eines kommerziell entwickelten Programms wie MATLAB gegenüber freien Programmen.

## 9. Simulink und Scicos

Für das Simulieren von Schaltungen, des Verhaltens von Bauelementen, etc. stellt MATLAB mit Simulink [21], [22], [23] ein eigenes Tool zur Verfügung. Dies ist ein sehr umfassendes Simulationswerkzeug, das eng verzahnt mit MATLAB arbeitet und dessen Routinen zur graphischen Ausgabe nutzt. Im Grunde ist Simulink ein Programm zur graphischen Darstellung von Differentialgleichungen. Dies stellt jedoch eine zu einfache Charakterisierung dar. Simulink ist

---

<sup>8</sup> Der JIT-Accelerator lässt sich allerdings zum Test mit dem Befehl `feature jit off` und `feature accel off` auch ausschalten.

eigentlich ein interaktives Tool zur Modellierung, Simulation und Analyse von dynamischen Systemen. Es erlaubt die Erstellung von Blockdiagrammen und deren nachfolgende Simulation. Dabei fügt es sich in die MATLAB-Umgebung, die für seinen Start unbedingt erforderlich ist, nahtlos ein. Seine Hauptanwendungsgebiete liegen in der Regelungstechnik, der Signalverarbeitung und der Nachrichtentechnik.

Simulink wird mit dem Befehl `simulink` von der MATLAB-Kommandozeile aus gestartet.

Unter Simulink erzeugte Programme lassen sich als Dateien abspeichern. Sie tragen die Endung „.mdl“. Leider sind die Dateien, die mit einer neueren Simulink-Version erstellt wurden, nicht abwärtskompatibel zu einer älteren Version; sie lassen sich also nicht auf älteren Umgebungen starten. Das ist bedauerlich für eine Nutzung in unterschiedlichen Arbeitsumgebungen.

Ein zu Simulink ähnliches Programm existiert bei Scilab unter dem Namen Scicos. Der Name Scicos bedeutet *Scilab Connectes Object Simulator*. Scicos läuft wie auch Scilab selbst unter der GNU public license, ist also von jedermann benutzbar. Allerdings ist die gegenwärtige Version von Scilab nicht geeignet, Scicos stabil arbeiten zu lassen. Hierzu sollte auf ScicosLab [24] zurückgegriffen werden. Weiterführende Literatur zu Scicos findet man zum Beispiel in [12],[25]. Scicos-Dateien werden mit der Endung „.cos“ gespeichert.

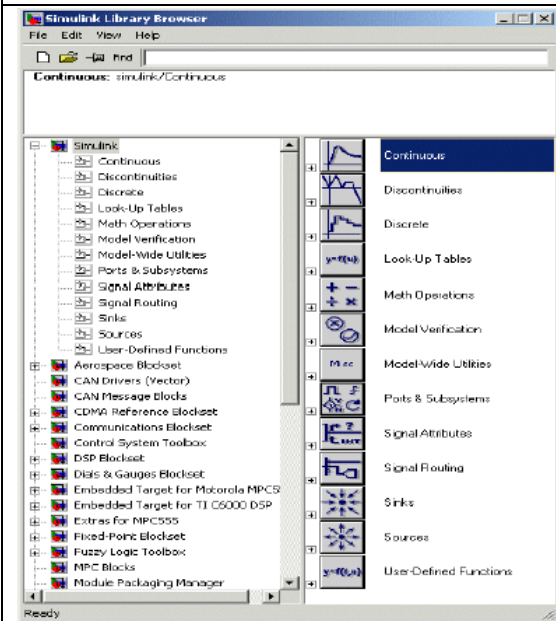
Unter Octave gibt es kein Äquivalent zu Simulink oder Scicos.

Simulink und Scicos verwenden Blöcke, die untereinander durch Wirkungslinien verbunden sind, um dynamische Systeme grafisch darzustellen. Die Blöcke werden Bibliotheken entnommen und ihre Parameter den jeweiligen Erfordernissen angepasst. Simulink und Scicos stellen eine große Menge an Standard-Blöcken bereits in eigenen Bibliotheken bereit.

Anhand eines Beispiels soll die Vorgehensweise zur Erstellung einer Simulation vorgestellt werden. Um das prinzipielle Vorgehen zu illustrieren, soll zunächst nur eine einfache Sinusfunktion dargestellt werden.

MATLAB/Simulink	Scilab/Scicos
<p>Simulink wird aus der Kommandozeile von MATLAB heraus mit <code>simulink</code> aufgerufen (alternativ auch durch einen entsprechenden Button in der MATLAB-Werkzeugleiste). Dabei öffnet sich sofort das Fenster des Simulink Library Browser.</p>	<p>Scicos wird aus der Kommandozeile von Scilab heraus aufgerufen mit <code>scicos</code> oder <code>scicos()</code>. Damit wird sofort ein neues Projekt angelegt (Name: <code>untitled</code>)                  Das Programm sollte am besten gleich unter einem neuen Namen gespeichert werden. Für ein erstes Beispiel nennen wir es <code>erster_versuch.cos</code>.                  Als nächstes sind die benötigten Graphikpaletten zu laden. Dazu ist es zumindest nötig, die Paletten <code>Sources</code> und <code>Sinks</code> aufrufen. In unserem Beispiel wird noch der <code>Sinusoid Generator</code> benötigt. Ihn kann man einfach in das Arbeitsfenster herüberziehen.. Zusätzlich wird aus der Palette <code>Sources</code> die <code>Event clock</code> benötigt.                  Die einzelnen Elemente werden nun durch Links verbunden. In Scicos gibt es zwei Sorten von Links: reguläre Links (schwarze</p>

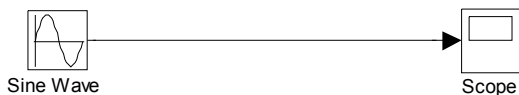
MATLAB/Simulink



Anschließend wird im Simulink Library Browser

File > New > Model angewählt. Dabei wird das Fenster eines neuen Simulink-Modells geöffnet, es trägt den Namen untitled.

Jetzt sind im Simulink Library Browser Sinks und Sources anzuklicken. Für unser Beispiel ist aus Sources der Sinusgenerator Sine Wave in das Graphikfenster hinüber zu ziehen und analog aus Sinks der Scope.



Unter Simulation > Parameters können nun die Simulationsparameter eingestellt werden.

Mit Simulation > Run wird nun die Simulation gestartet. Das Ergebnis erhält man im Fenster Scope. Indem auf das Fernglas gedrückt wird, passt sich die Graphik optimal ins Fenster ein.

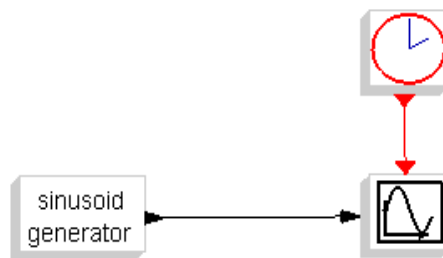
Scilab/Scicos

Dreiecke, üblicherweise an den Seiten der Diagramme positioniert) und Activation links (rote Dreiecke, üblicherweise an den Oberkanten positioniert). Achtung: Es gibt auch zwei Sorten des Timers: Wir benötigen für unsere Anwendung die Activation clock (ein rotes Symbol).

Um ein geeignetes Anzeigeelement zu finden, wählen wir unter Scicos im Menü Palette den Menüpunkt PalTree. Dort kann man alle verfügbaren Elemente finden. Leider werden die Palettenamen bei Scicos nicht deutlich angezeigt. Man kann jedoch das Palettsymbol mit der rechten Maustaste anklicken und mittels Help-Befehl die Beschreibung aufrufen.

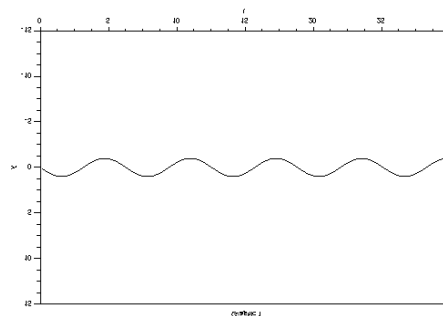
Zur Anzeige wird hier ein Oszilloskop verwendet, das die Zeitabhängigkeit wiedergibt (im Gegensatz zur XY-Darstellung, die wir gleich besprechen). Diese Palette trägt die Bezeichnung

CSCOPE – Single Display scope

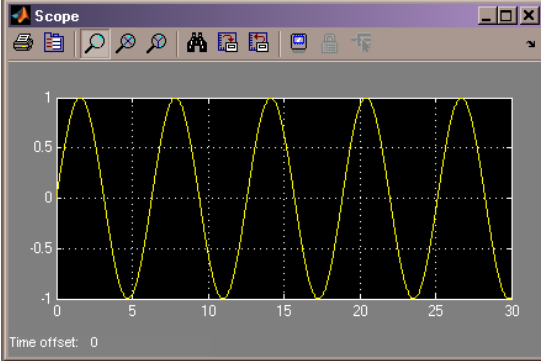
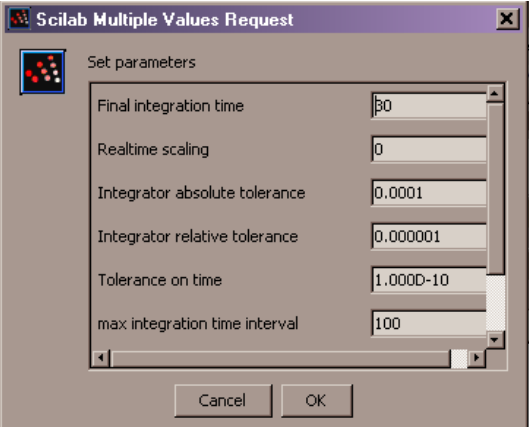


Nachdem die Symbole durch Links verbunden sind, lässt sich die Simulation starten. Dazu ist im Pull-down-Menü

Simulate > Run zu bedienen. Es wird dann automatisch ein Fenster geöffnet, auf dem die Graphik erscheint. Sie wird fortlaufend aktualisiert, bis Stop gedrückt wird.



Wenn unüberlegt zu viele Daten geladen werden, erscheint das Signal Overloading. Wichtig ist deshalb, möglichst sofort nach Aufruf von Scicos die Simulationsparameter geeignet einzustellen.

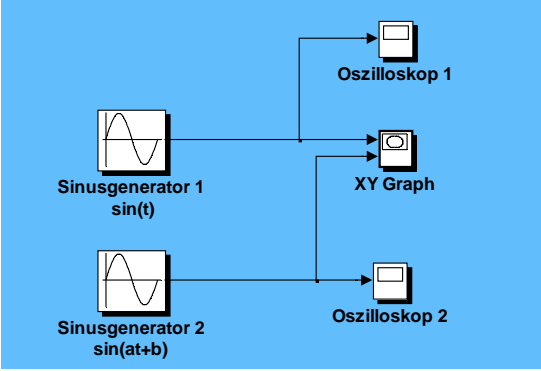
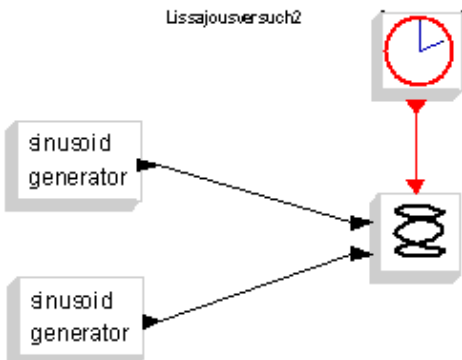
MATLAB/Simulink	Scilab/Scicos
 <p>Bei Simulink kann man die Simulationsparameter im Menü <code>Simulation &gt; Parameters</code> einstellen.</p>	<p>Dies kann im Menü <code>Simulate &gt; Setup</code> erledigt werden (Abbildung unten). Die <code>Final integration time</code> sollte für die Tests nicht zu hoch gewählt werden, der Default-Wert ist unhandlich groß, nämlich 100000.</p> 

Nach dem Aufruf von Scicos ist das Kommandofenster von Scilab übrigens für den Anwender blockiert, obwohl ständig Werte ausgegeben werden. Wenn man darauf während der Scicos-Sitzung trotzdem zugreifen möchte, ist unter Scicos der Menüpunkt

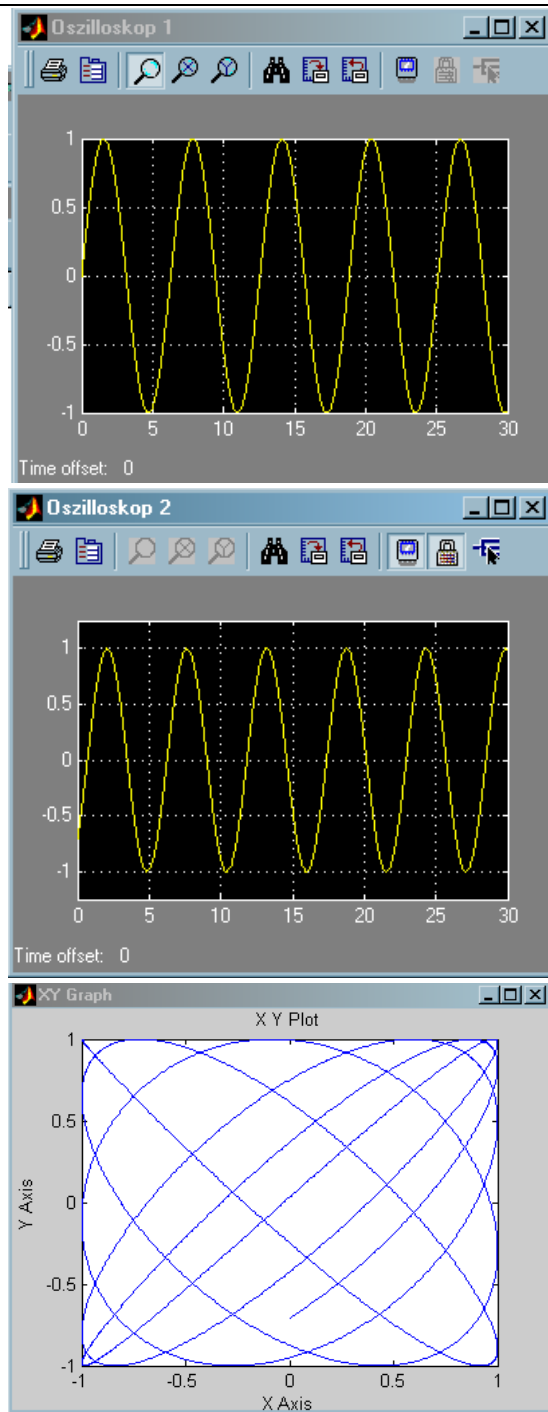
`Tools > Activate Scilab Window`

zu betätigen. Bei MATLAB/Simulink dagegen kann während des Arbeitens mit Simulink stets auf das MATLAB-Kommandofenster zugegriffen werden.

Wir wollen nun sogenannte Lissajous-Figuren darstellen. Sie werden in der Elektrotechnik erzeugt, wenn an den x- beziehungsweise y-Eingang eines Oszilloskops zwei unterschiedliche Sinusfunktionen angelegt werden. Dazu wird ein neues Programm wie folgt erstellt:

MATLAB/Simulink	Scilab/Scicos
<p>Blockschaltbild des Lissajousversuches:</p>  <p>Unten sind die Kurven dargestellt, die auf den Oszilloskopen 1 und 2 zu sehen sind.</p>	<p>Blockschaltbild des Lissajousversuches:</p>  <p>Hier wurde zur Anzeige die Palette <code>CSCOPTY</code> – <code>y=f(x)</code> permanent viewer verwendet. Dabei handelt es sich um ein</p>

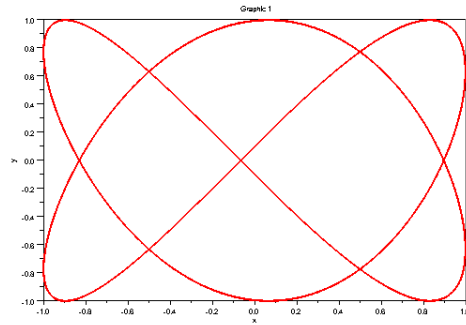
MATLAB/Simulink



Scilab/Scicos

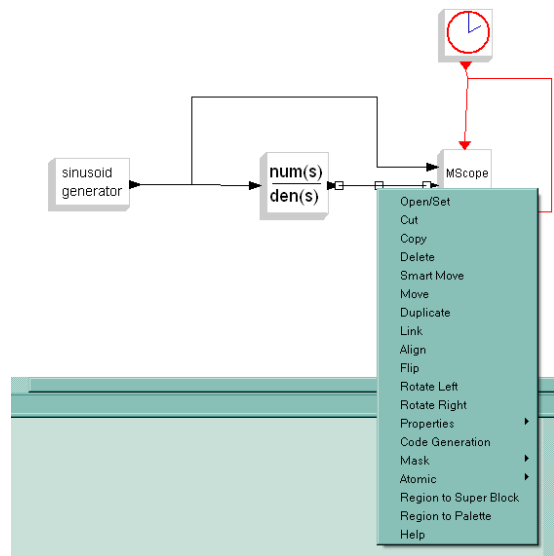
Oszilloskop, das einen XY-Graphen zeichnet.

Als Ergebnis der Simulation wird automatisch das folgende Bild erzeugt:

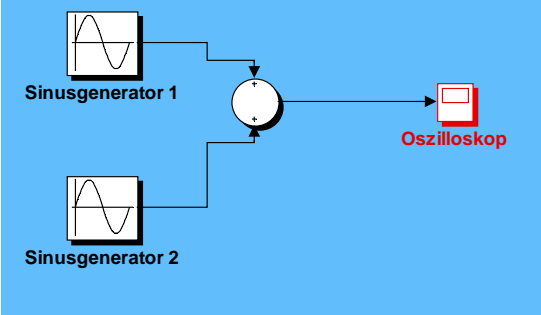
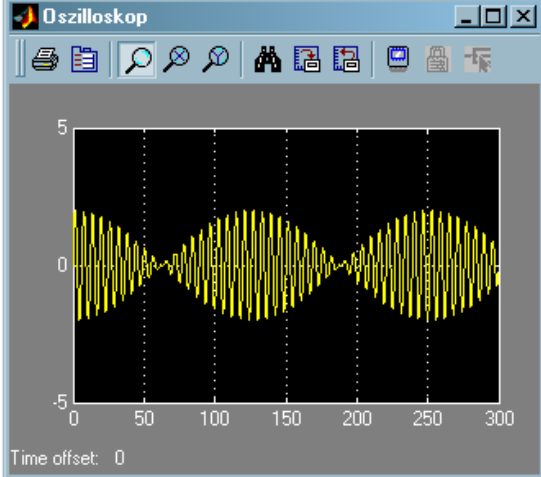
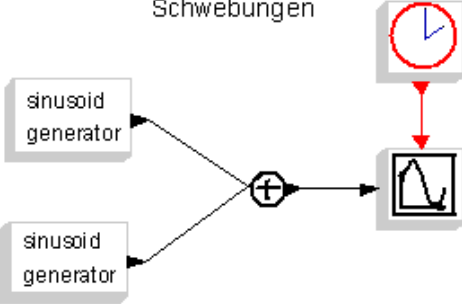
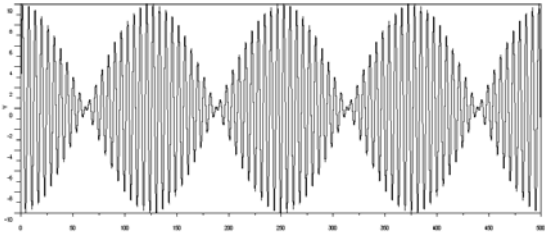


Das Zeichnen der Links in Scicos gestaltet sich übrigens etwas schwerfälliger als bei Simulink. Insbesondere wenn eine Verzweigung geöffnet werden soll, muss man erst einige Möglichkeiten durchprobieren – deutsche Beschreibungen sind ja kaum verfügbar. Dabei ist wie folgt vorzugehen:

1. Mit der rechten Maustaste auf die bereits bestehende Verbindungslinie drücken (siehe Bild)
2. Open/Set auswählen
3. Jetzt die neue Verbindungslinie bis zum Eingang des Zielblocks ziehen.
4. Zum Abschließen wieder die rechte Maustaste drücken.



Als nächstes Beispiel sollen Schwebungen dargestellt werden. Hierzu benötigen wir zwei Sinusgeneratoren, die mit leicht gegeneinander verstimmt Frequenzen arbeiten. Die Summe der Schwingungen aus diesen beiden Sinusgeneratoren wird in einer Graphik dargestellt.

MATLAB/Simulink	Scilab/Scicos
<p>Die Anzeige in Simulink geschieht am einfachsten über den Block „Scope“ (in untenstehendem File in „Oszilloskop“ umbenannt)</p>  <p>Ergebnis der Simulation:</p> 	<p>Hier wird wieder der Block Single Display Scope (CSCOPE) benötigt, also ein einfaches Oszilloskop.</p>  <p>Ergebnis der Simulation:</p> 

Häufig steht man vor der Aufgabe, Graphiken aus Simulink oder Scicos in andere Dateien, zum Beispiel in Word-Texte, zu exportieren. Dazu kann man wie folgt vorgehen:

MATLAB/Simulink	Scilab/Scicos
<p>In Simulink wird das Blockschaltbild über <code>Edit &gt; Copy model to clipboard</code> ins Clipboard aufgenommen – das funktioniert jedoch nicht beim Menü <code>Scope</code>. Um dieses zu kopieren, hilft nur der Weg über eine Bildschirmkopie.</p>	<p>Der Export von Scicos-Graphiken geschieht über das Menü <code>File &gt; Export</code>. Dann müssen Sie auf <code>Graphic Window</code> doppelklicken. Damit werden die Daten in das Clipboard kopiert. Alternativ ist es auch möglich, die Graphik über Doppelclick auf <code>File</code> als PNG- oder EPS-Datei in das Clipboard aufzunehmen.</p>

Ein weiteres Beispiel, das hier besprochen werden soll, arbeitet mit einer Parametereingabe von der Konsole aus und mit der weiteren Möglichkeit, Simulationsergebnisse an die Konsole zurückzugeben. Hierzu wird die Lösung der inhomogenen Schwingungsgleichung mit Dämpfung als Beispiel herangezogen. Damit liegt ein Beispiel vor, das gleichzeitig die Verwendung von



Übertragungsgliedern in der Regelungstechnik zeigt. Sowohl Simulink als auch Scicos greifen bei der Lösung von Schwingungsgleichungen die Schreibweise auf, die auch in der Regelungstechnik verwendet wird, um eine Übertragungsfunktion im Bildraum der Laplace-Transformierten darzustellen. Im Falle der Schwingungsgleichung mit Dämpfung handelt es sich, in der Sprache der Regelungstechnik ausgedrückt, um ein PT2-Glied. Etwas „volkstümlicher“ kann man dieses Übertragungsglied durch folgende Festlegungen charakterisieren:

1. Im Nenner steht der homogene Teil der Differentialgleichung. Darin werden alle Ableitungen durch einen Parameter  $s$  dargestellt.

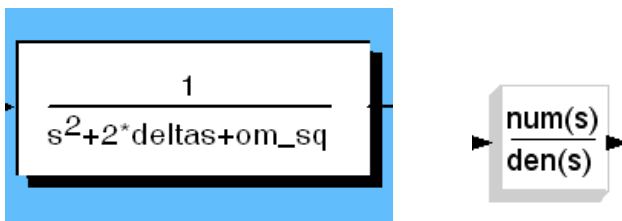
$$\frac{d}{dt} \leftrightarrow s$$

2. Im Zähler steht eine Eins, wenn man die üblichen Anfangsbedingungen als null annimmt. Damit wird der Eingang der Funktion (der inhomogene Teil der Differentialgleichung, also üblicherweise die rechte Seite) multipliziert.

Die Korrespondenz lautet demnach wie folgt:

$$\underbrace{\frac{d^2}{dt^2} x + 2\delta \cdot \frac{d}{dt} x + \omega_0^2 x}_{\text{homogener Teil der Dgl.}} = \underbrace{1 \cdot A \cos \omega_a t}_{\text{inhomogener Teil der Dgl.}} \leftrightarrow \underbrace{s^2 \cdot x + 2\delta \cdot s \cdot x + \omega_0^2 x}_{\text{Nenner}} = \underbrace{1}_{\text{Zähler}} \cdot \underbrace{A \cos \omega_a t}_{\text{Eingang des Simulationsblocks}}$$

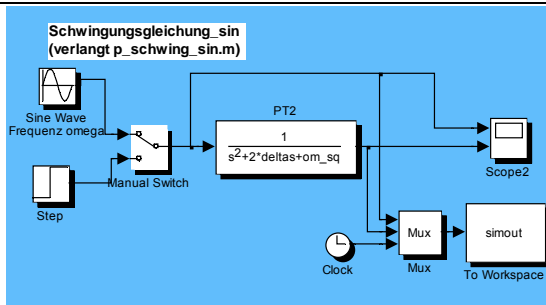
In Simulink wird das sehr schön dargestellt: Die Form der Übertragungsfunktion (mit anderen Worten: die Gestalt der Differentialgleichung) erscheint direkt als Text im Übertragungsblock (Abbildung 10). Leider ist die Darstellung in Scicos nicht so augenfällig, sondern trägt nur eine sehr allgemeine Bezeichnung.



**Abbildung 10** Darstellung des Übertragungsblocks eines PT2-Glieds in Simulink (links) und in Scicos (rechts)

MATLAB/Simulink	Scilab/Scicos
Erstellen des Modells	
Mit dem SIMULINK-Modell Schwingungsgleichung_sin.mdl wird die Schwingungsgleichung mit Sinusanregung dargestellt. Das Modell benutzt die Übertragungsfunktion TransferFcn, die auch in der Regelungstechnik benötigt wird.	Das entsprechende Scicos-Modell trägt die Bezeichnung Schwingungsgleichung_sin.cos. Es benutzt den Block CLR (Continuous Transfer Function).
Simulink-Blockschaltbild:	Scicos-Blockschaltbild:

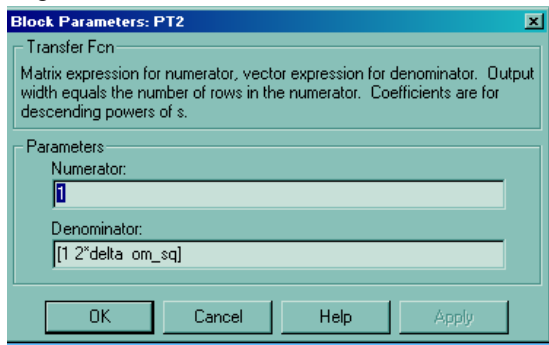
MATLAB/Simulink



(Den Block ToWorkspace können wir an dieser Stelle noch außer Acht lassen, wir gehen später noch auf ihn ein.)

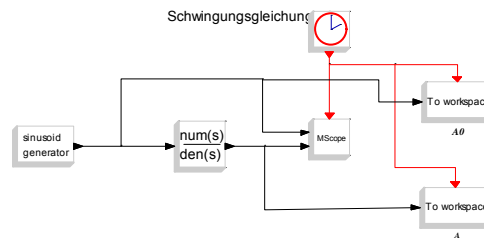
*Hinweis:* Graphiken in Simulink können nach Bedarf eingefärbt werden, hier zum Beispiel mit blauer Hintergrundfarbe.

Wenn man mit der rechten Maustaste auf das Symbol der Übertragungsfunktion klickt und Block Parameters auswählt – oder einfach mittels Doppelklick –, öffnet sich das folgende Fenster:



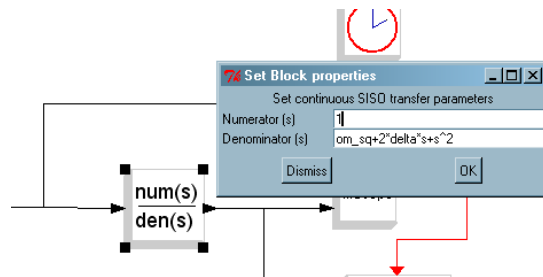
Hier können jetzt die Daten oder Variablenamen eingeschrieben werden (was in der vorliegenden Abbildung bereits geschehen ist)

Scilab/Scicos



(Auch hier betrachten wir den Block ToWorkspace noch nicht.)

Die Parameter dieses Blocks werden ebenfalls wie bei Simulink durch Doppelklick auf das Bild eingegeben beziehungsweise modifiziert.



Die Beschreibung wird mittels rechter Maustaste und Help aufgerufen.

Import von Daten:

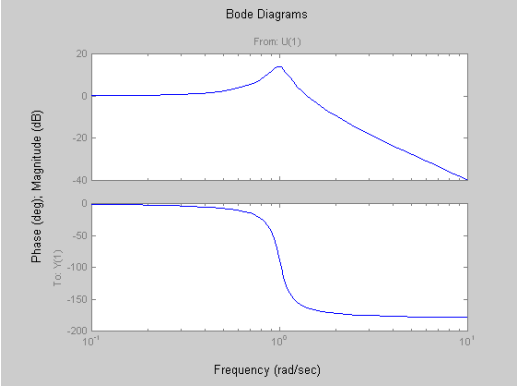
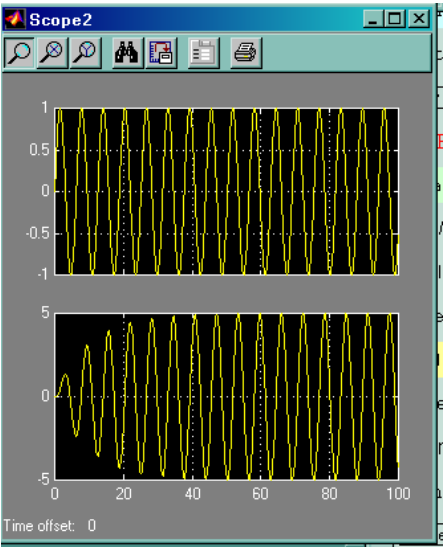
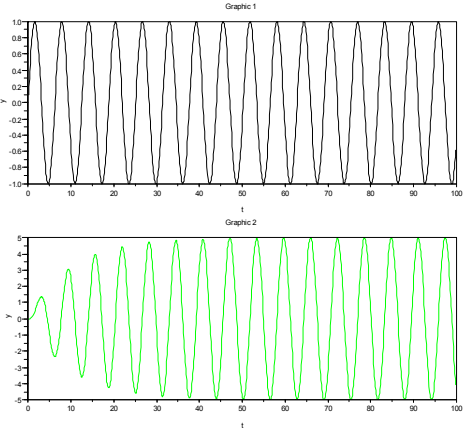
Die Simulationsparameter sind in diesem Modell nicht als Zahlenwerte, sondern als Variablen eingetragen. Sie müssen vorher über die Konsole zur Verfügung gestellt werden. Dies kann über die Kommandozeile geschehen oder über eine vorher zu startendes M-File (bei Scilab: sci-Datei). Eine solche Datei soll im folgenden vorgestellt werden.

Zur Parametereingabe wird vor der Simulation das M-File p\_schwing\_sin.m aufgerufen.  $\delta$  und  $\omega^2$  (entsprechend delta und om\_sq) sind die Parameter, die im Modell Schwingungsgleichung\_sin.mdl benötigt werden.

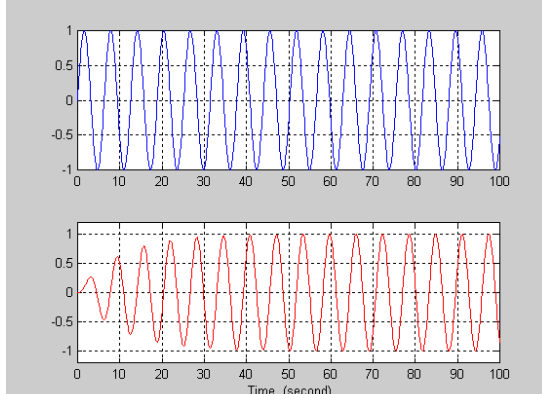
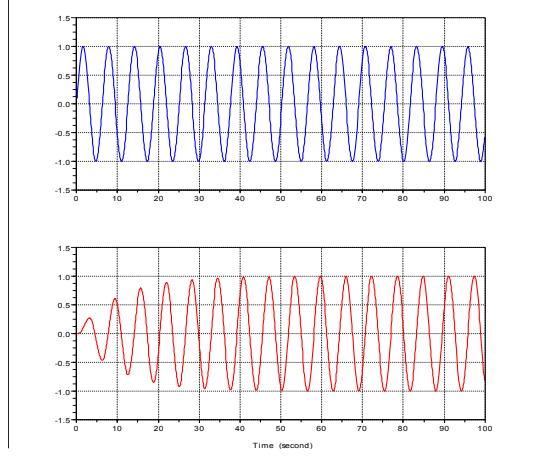
```
omega_0 = 1;
% Eigenkreisfrequenz
delta = input('delta = ');
% Dämpfung
if isempty(delta), delta=1; end
omega = input('omega = ');
if isempty(omega), omeg = omega_0; end
amp = input('Schwingungsamplitude (Default = 1) ');
```

Die sci-Datei p\_schwing\_sin.sci ermöglicht die Parametereingabe für das Modell Schwingungsgleichung\_sin.cos. Diese Datei muss vor dem Aufruf des Scicos-Modells in Scilab ausgeführt werden.

```
omega_0 = 1;
// Eigenkreisfrequenz
delta = input('delta = ');
// Dämpfung
if isempty(delta), delta = 1; end
omega = input('omega = ');
if isempty(omega), omega=omega_0;end
amp =input('Schwingungsamplitude (Default = 1) ');
```

<p><b>MATLAB/Simulink</b></p> <pre> if isempty(amp), amp = 1; end om_sq = omega_0^2; iomega = sqrt(delta.^2-omega_0^2); lambda = -delta + iomega; tmax=100*max(1/sqrt(omega), ... 1/sqrt(omega_0)); % Ende der Simulation                 </pre> <p>Nach Bedarf könnte hier auch noch zur Übersicht die Berechnung eines Bode-Diagramms (in MATLAB, nicht in Simulink!) angeschlossen werden:</p> <pre> % Bode-Diagramm num = 1; den = [1 2*delta om_sq]; bode(num, den);                 </pre> 	<p><b>Scilab/Scicos</b></p> <pre> om_sq = omega_0^2; iomega = sqrt(delta.^2-omega_0^2); lambda = -delta + iomega; tmax = 100 * max(1/sqrt(omega), ... 1/sqrt(omega_0)); // Ende der Simulation                 </pre> <p>Berechnung des Bode-Diagramms:</p> <pre> // Bode-Diagramm //num = 1; //den = [1 2*delta om_sq]; //bode(num, den);                 </pre>
<p><b>Simulationsergebnis</b></p>	
<p>Das Simulationsergebnis wird in der folgenden Graphik dargestellt; hier geschieht dies für die Eingabeparameter</p> <p>delta = 0.1,  omega = 1 (Default),  amp = 1 (Default).</p> 	<p>Das Simulationsergebnis wird in der folgenden Graphik dargestellt; hier für die Eingabeparameter</p> <p>delta = 0.1,  omega = 1 (Default),  amp = 1 (Default).</p> 
<p><b>Übergabe der Daten an MATLAB bzw. Scilab (Block ToWorkspace)</b></p> <p>Mit dem Block ToWorkspace können die Ergebnisse von Simulink oder Scicos zur Kommandozeile des aufrufenden Programms übergeben werden. Hierzu gibt es in beiden Programmen eine Dialogbox ToWorkspace. Bei Benutzung von Simulink stehen diese Daten sofort im</p>	

MATLAB/Simulink	Scilab/Scicos
<p>MATLAB-Arbeitsraum zur Verfügung. Im Unterschied dazu muss Scicos erst beendet sein, wenn man von Scilab aus auf die Daten zugreifen möchte – ein unter Umständen nicht unerheblicher Nachteil.</p>	
<p>In Simulink gibt es die Möglichkeit, <i>mehrere Variablen</i> in einer Box ToWorkspace zu übergeben. Damit können sie im aufrufenden MATLAB-Programm weiter verwendet werden, beispielsweise um eine MATLAB-Graphik zu zeichnen. In unserem Fall wurde der Variablenname mit der bereits als Default-Namen vorgeschlagenen Bezeichnung <code>simout</code> belassen. Zusätzlich müssen noch zeitliche Schritte vorgegeben werden, dazu wird die Uhr (<code>clock</code>) benötigt.</p> <p>In MATLAB wird in unserem Fall die Variable <code>simout</code> als dreispaltige Matrix ausgegeben. Die Spalten der Matrix sind die Zeit sowie die beiden Simulationswerte die auch im Block Scope zu sehen sind.</p> <pre data-bbox="199 981 751 1182"> » whos simout    Name      Size      Bytes    Class    simout    1002x3      24048    double array Grand total is 3006 elements using 24048 bytes                 </pre> <p>Zur Visualisierung dient das Ausgabe- programm <code>schwing_sin_out.m</code></p> <pre data-bbox="199 1870 751 2049"> % ***** % Programm (schwing_sin_out.m) % Darstellung der % Simulationsergebnisse % für Schwingungsgleichung_sin.m % *****                 </pre>	<p>In Scicos gibt es keine Möglichkeit, mehrere Variablen in einer Box zu übergeben. Man benötigt deshalb für jede Variable eine eigene Box ToWorkspace mit zugehörigem Clock-Signal. In unserem Falle wurden die beiden Variablen mit <code>A0</code> (ungestörtes Eingangssignal) und <code>A</code> (über die Transferfunktion übertragenes Signal) bezeichnet.</p> <p>Bei der späteren Verwendung in Scilab wird jede Variable als zweispaltiges Array dargestellt, also <code>A0.time</code> und <code>A0.value</code> sowie <code>A.time</code> und <code>A.value</code></p> <p>Die Arrays <code>A</code> und <code>A0</code> haben die Struktur</p> <pre data-bbox="774 869 1321 1646"> --&gt;whos -name A Name      Type      Size      Bytes A         st        1 by 1    16184 A0        st        1 by 1    16184 --&gt;A A =    values: [999x1 constant]    time: [999x1 constant] --&gt;A0 A0 =    values: [999x1 constant]    time: [999x1 constant] --&gt;A0.time ans =    0.1    0.2    0.3    0.4 ...USW. --&gt;A0.values ans =    0.0998334    0.1986693    0.2955202 ...USW.                 </pre> <p>Zur Visualisierung dient auch hier das Ausgabeprogramm <code>schwing_sin_out.sci</code></p> <pre data-bbox="774 1792 1321 2049"> // ***** // Programm (schwing_sin_out.sci) // Darstellung der Simulationsergebnisse // für Schwingungsgleichung_sin.m // *****  subplot(2,1,1); plot(A0.time,A0.values); xgrid; mtlb_axis([0 tmax -1.2 1.2]);                 </pre>

MATLAB/Simulink	Scilab/Scicos
<pre data-bbox="199 257 742 481">t = simout(:,3); subplot(2,1,1); plot(t, simout(:,1), 'b'); grid on; subplot(2,1,2); plot(t, simout(:,2)/max(simout(:,2)), 'r'); grid on; xlabel(' Time (second)'); axis([0 tmax -1.2 1.2]);</pre> 	<pre data-bbox="774 257 1316 392">subplot(2,1,2); plot(A.time,A.values/max(A.values), 'r'); xgrid; xlabel(' Time (second)'); mtlb_axis([0 tmax -1.2 1.2]);</pre> 

Häufig wird sich die Frage stellen, dass man vom „Mutterprogramm“ MATLAB oder Scicos aus die Simulationen starten möchte, ohne die Blockdiagramme selbst aufrufen zu müssen. Hierzu wird ein sogenanntes Batchfile gestartet. Von MATLAB aus ist hierzu `sim (Dateiname)` einzugeben, beispielsweise

```
sim Lissajousversuch.
```

Damit wird die Simulation gestartet, ohne das Blockschaltbild selbst anzuzeigen. Nur das Ergebnis wird angezeigt, in unserem Beispiel der bereits oben gezeigte xy-Plot der Lissajouskurven.

Der Aufruf von Scicos geschieht mit `scicosim` und den Namen der zu bearbeitenden Anwendung, zum Beispiel

```
-->scicosim ('erster_versuch.cos',0,30,, 'start',1e-3)
```

oder

```
scicos_simulate ('erster_versuch.cos')
```

## 10. Erfahrungen aus Lehre und Ausbildung, Forschung und Entwicklung

Infolge seiner professionellen Unterstützung und stetigen Weiterentwicklung hat sich MATLAB zu einem unentbehrlichen Standard-Werkzeug bei der Simulation komplizierter technischer Vorgänge etabliert. Hierzu trägt bei, dass im Laufe der Zeit immer mehr Toolboxen für die verschiedensten Spezialanwendungen entwickelt wurden. Aber auch die Optimierung der Verarbeitungsgeschwindigkeit hat sich positiv ausgewirkt – sie hat insbesondere beginnend mit der Version 6 enorme Fortschritte gemacht.

Große Unternehmen setzen heute in der Regel MATLAB als Standard-Entwicklungswerkzeug ein. Insbesondere für die Simulation komplizierter Probleme der Regelungstechnik und Signalverarbeitung bietet MATLAB entscheidende Vorteile. Die Anwendungen reichen bis zur Code-Generierung für den nachfolgenden Einsatz in Embedded Mikroprozessorsystemen oder zur Steuerung von Feldbussen.

Auf der anderen Seite kann sich MATLAB zu einem kostenintensiven Werkzeug entwickeln, wenn in einem Unternehmen zahlreiche Arbeitsplätze ausgestattet werden sollen, und dies auch noch mit unter Umständen mehreren Tollboxen geschieht. Allein schon die Installation sowohl auf PC als auch auf Notebook kann für einen Einzelanwender, zum Beispiel in einem Ingenieurbüro, bereits einen erheblichen Kostenfaktor darstellen.

Beim Einsatz von kommerzieller Software wie MATLAB spielen nicht nur deren eigentliche Kosten, sondern auch die Folgekosten eine große Rolle. Softwaresysteme müssen unter Umständen über einen Zeitraum von 10 Jahren oder länger gewartet werden. Wenn das System in diesem Zeitraum nicht mehr intensiv genutzt wird, sind die dadurch entstehenden Aufwendungen für die Aktualisierung der Programme inakzeptabel hoch. Es hat sich deshalb gezeigt, dass es insbesondere bei Aufträgen aus dem öffentlichen Sektor interessant sein kann, Anwendungen mit Scilab zu erstellen. Derselbe Grund führte ja in der Vergangenheit beispielsweise in Behörden auch an anderer Stelle dazu, Software aus dem Freeware-Bereich zu nutzen, beispielsweise Linux. Der Aufwand für die Konvertierung von MATLAB- in Scilab-Dateien und umgekehrt wird in diesem Zusammenhang als nicht dramatisch eingeschätzt.

Prinzipiell orientiert sich die Entscheidung, welche Software in der Industrie eingesetzt werden soll, vorwiegend an folgenden Anforderungen:

- Höhe des Einarbeitungsaufwands,
- Bedienerfreundlichkeit,
- Rechenzeit,
- Möglichkeit der Batchverarbeitung,
- Entwicklungskosten, umgerechnet auf die Einarbeitungszeit.

Ein ganz wesentlicher Gesichtspunkt stellt auch der Support dar, auf den ein Entwicklungsteam bei seiner Arbeit zugreifen kann. Er wird in der Regel bei kommerzieller Software als wesentlich höher eingeschätzt. Trotzdem dürfen die Support-Möglichkeiten bei Public-Domain-Software nicht unbedingt schlecht bewertet werden. Es gibt zahlreiche öffentliche Foren, die anstelle einer Firmenberatung zu Rate gezogen werden können – leider sind jedoch die Anfragen und Hinweise in solchen Foren nicht in allen Fällen auch verlässlich und hilfreich.

Ähnlich verhält es sich mit dem Einsatz von Mathematikprogrammen in der Ausbildung an der Hochschule. Einerseits sollten Studenten natürlich praxisnah ausgebildet werden. Unter diesem Aspekt sollten eigentlich solche Programme genutzt werden, die auch später in der Industrie vorwiegend zum Einsatz kommen. Auf der anderen Seite stellen gerade sie einen nicht unerheblichen Kostenfaktor dar. Zwar werden die Hochschul- und Studentenlizenzen von MATLAB

heute zu einigermaßen günstigen Konditionen angeboten, dennoch ist ihre Nutzung selbst für reine Ausbildungszwecke nicht uneingeschränkt möglich. Wie auch rein kommerziell genutzte Lizenzen können auch Studentenlizenzen infolge der Online-Registrierung nur auf einem PC allein installiert werden, eine parallele Nutzung auf einem Notebook durch den gleichen Anwender ist nicht möglich. Darüber hinaus haben Studentenlizenzen nur begrenzte Gültigkeit. Ein Student, der sich in seinem Studium mit dem Programm einmal vertraut gemacht hat, kann es folglich danach nicht mehr nutzen.

Auch die Campuslizenzen verschlingen im Laufe der Zeit erhebliche Mittel, sollen sie ständig über Wartungsverträge aktualisiert werden. Durch die Einführung von Studiengebühren ist es zum Beispiel an der Hochschule Pforzheim zunächst möglich geworden, jedem Studenten eine persönliche Studentenlizenz zur Verfügung zu stellen und darüber hinaus auch die Campuslizenz aktuell zu halten. Trotzdem ist es aufgrund der Kosten auf die Dauer fraglich, ob zukünftig nicht mehr Augenmerk auf Octave gelegt werden sollte. Die Entwicklung einer graphischen Benutzeroberfläche, die gegenwärtig für Octave in Arbeit ist, wird diese Überlegungen sicher günstig beeinflussen. Ein schwerwiegender Mangel bei Octave ist allerdings, dass ein Tool fehlt, welches etwa die graphischen Möglichkeiten von Simulink beinhaltet. Möglicherweise wird aber auch das im Laufe der Zeit von der Entwickler-Community noch wenigstens in einem solchen Umfang bereit gestellt, dass es für die Erfordernisse der Ausbildung ausreicht. Allerdings sind derzeit keine Überlegungen bekannt, dass innerhalb in der GNU-Gemeinde Anstrengungen in dieser Richtung unternommen würden.

Auf der anderen Seite reichen die Möglichkeiten, die Scilab/Scicos bietet, für die meisten Zwecke der Lehre, aber auch für einfache Entwicklungsaufgaben aus. Leider ist der Umgang mit Scilab von vornherein etwas komplizierter als der mit MATLAB, und auch die Stabilität der jüngsten Scilab-Version ist nicht unter allen Hardwarekonfigurationen gesichert. Hier ist mit ScicosLab eine stabilere, wenn auch gegenüber Scilab 5 weniger komfortable Version verfügbar. Es wäre wünschenswert, dass die Scilab-Version 5 mit der ScicosLab-Version 4.3 bald zusammengeführt wird. An Hochschulen sind bereits vielfältige Erfahrungen zum Arbeiten mit Scilab vorhanden [26].

Für Einrichtungen der Berufsausbildung oder für technische Gymnasien und Technikerschulen dürften Octave und Scilab in jedem Falle günstige Alternativen zu MATLAB darstellen. Dort wird heute oft mit MAPLE oder auch noch mit speziellen Computer-Algebra-Taschencomputern (zum Beispiel mit dem TI-Nspire CAS von Texas Instruments) gearbeitet. Das ist leider eine Investition, die sich nicht auszahlt, da eine spezielle Hardware erforderlich ist und diese Systeme bereits an der Hochschule sowie später in der Ingenieurpraxis nicht mehr verwendet werden.

In den letzten Jahren wurde an der Hochschule Pforzheim damit begonnen, bereits im ersten Semester die Arbeit mit einem PC-Numeriksystem in das Ausbildungsprogramm aufzunehmen. Wir haben uns entschieden, zunächst MATLAB zu benutzen. Daneben sind jedoch immer auch einzelne Arbeiten mit Scilab durchgeführt worden. Die breite Einführung von MATLAB zu einem sehr frühen Zeitpunkt ermöglicht es den Studenten, bereits in der parallelen Mathematik-Vorlesung damit zu arbeiten. Dies wurde auch vielfach genutzt. In späteren Veranstaltungen zur Signalverarbeitung, zur Numerischen Mathematik und zur Regelungstechnik konnten dann die

Kenntnisse von MATLAB bereits vorausgesetzt werden. Aber auch bei der Auswertung von Laborpraktika ist es sinnvoll, ein solches professionelles Werkzeug zu benutzen. In einigen Veranstaltungen wurde teilweise noch auf Excel zurückgegriffen [27]. Mit der Verwendung von Tabellenkalkulationsprogrammen allein wird sicher eine pragmatische Lösung aufgezeigt, die Studierenden werden jedoch auf eine spätere Anwendung im Beruf nur unzureichend vorbereitet. Insgesamt besteht deshalb Bedarf, auch in weiteren Lehrveranstaltungen im eigenen Bereich den Nutzen eines Numerikprogramms deutlich zu machen und dessen Verwendung zu üben.

Für das Arbeiten mit MATLAB im ersten Semester ist allerdings eine intensive Betreuung seitens der Hochschule erforderlich, und zwar durch Professor, wissenschaftliche Mitarbeiter und Tutoren gemeinsam. Insgesamt gesehen kann dieses Konzept jedoch als außerordentlich erfolgreich eingeschätzt werden.

## **Danksagungen**

Diese Arbeit entstand im Rahmen eines Forschungssemesters in Zusammenarbeit mit der THALES Defence GmbH, Pforzheim. Der Autor dankt diesbezüglich insbesondere Herrn Dipl.-Ing. Günther Hornung, Director Business Area Electronic Warfare, für seine tatkräftige Hilfe und Unterstützung sowie den Herren Dipl.-Ing. Werner Kranzpiler und Dipl.-Ing. Norbert Sonnenberg für nützliche Diskussionen. Freundlicherweise wurde von THALES auch der Druck der Arbeit übernommen. Wegen zahlreicher fruchtbaren Diskussionen bin ich außerdem Herrn Dipl.-Phys. Michael Bauer, Hochschule Pforzheim, zu Dank verpflichtet. Nicht zuletzt waren aber auch Anregungen und Kommentare von Studenten immer eine wertvolle Hilfe. Einzelne Ergebnisse wurden bereits früher in einer vom Autor betreuten Diplomarbeit [19] gewonnen.



## 11. Literatur

- [1] H. Benker, Ingenieurmathematik mit Computeralgebra-Systemen. AXIOM, DERIVE, MACSYMA, MAPLE, MATHCAD, MATHEMATICA, MATLAB und MuPAD in der Anwendung (Taschenbuch). Vieweg, Wiesbaden, 1998
- [2] Norman Chonacky and David Winch, *Maple, Mathematica, and Matlab: The 3M's without the Tape*, Computing in Science & Engineering, vol. 7, no. 1, 2005, pp. 8-16.
- [3] Norman Chonacky and David Winch, *Reviews of Maple, Mathematica, and Matlab: Coming Soon to a Publication Near You*. Computing in Science & Engineering, vol. 7, no. 2, 2005, pp. 9-10
- [4] Norman Chonacky and David Winch, *3Ms for Instruction: Reviews of Maple, Mathematica, and Matlab*, Computing in Science & Engineering, vol. 7, no. 3, 2005, pp. 7-13.
- [5] Norman Chonacky and David Winch, *3Ms for Instruction, Part 2: Maple, Mathematica, and Matlab*. Computing in Science & Engineering, vol. 7, no. 4, July/August 2005, pp. 14-23.
- [6] Über The MathWorks. <http://www.mathworks.de/company/aboutus/> (2006)
- [7] Scilab-Webseite, <http://www.scilab.org/> (2009)
- [8] MODELICA - Modeling of Complex Physical Systems. 2009  
vgl. auch die Webseite URL: <http://www.modelica.org/>
- [9] J.W. Eaton, GNU Octave Manual. Network Theory Limited, Bristol 2005, PDF-Version (1997):  
<http://pcmap.unizar.es/softpc/OctaveManual.pdf>, <http://www.gnu.org/software/octave/>
- [10] J.W. Eaton, Octave Manual (2009), URL: <http://www.gnu.org/software/octave/doc/interpreter/>
- [11] About Octave. <http://www.octave.org>, 2006
- [12] S. L. Campbell, J.-P. Chancelier, R. Nikoukhah, Modeling and Simulation in Scilab/Scicos. Springer, NewYork 2000
- [13] Gilberto E. Urroz, Numerical and Statistical Methods with SCILAB for Science and Engineering, vol. 1, greatunpublished.com, 2001
- [14] Gilberto E. Urroz, Numerical and Statistical Methods with SCILAB for Science and Engineering, vol. 2, greatunpublished.com, 2001
- [15] Srikanth S V, Comparative Study of MATLAB and its Open Source Alternative Scilab, Technical Report OSS 0601, OSSRC (Open Source Software Resource Center), National Institute of Technology Calicut, Mumbai, o.J.
- [16] Mottelet, Matlab-like Plotting library for Scilab. 2009  
URL: <http://www.dma.utc.fr/~mottelet/myplot.html>
- [17] Lehr- und Übungsbuch Informatik 1. Grundlagen und Überblick (Gebundene Ausgabe), Hrsg: Christian Horn, Immo O. Kerner und Peter Forbrig. Fachbuchverlag im Carl Hanser Verlag, Leipzig 2003
- [18] U. Rembold, P. Levi, Einführung in die Informatik für Naturwissenschaftler und Ingenieure. Hanser Fachbuch; 3. Aufl, 2002

- [19] M. Hamann, Messsignalauswertung mit MATLAB und alternativen Verfahren der numerischen Mathematik, Diplomarbeit, Hochschule Pforzheim, 2007
- [20] The MathWorks, Accelerating MATLAB. The MATLAB JIT-Accelerator (2006)  
[http://www.mathworks.com/company/newsletters/digest/sept02/accel\\_matlab.pdf](http://www.mathworks.com/company/newsletters/digest/sept02/accel_matlab.pdf)
- [21] Using Simulink. The MathWorks Inc., 2008. (Handbuch)
- [22] F. Grupp und F. Grupp. Simulink – Grundlagen und Beispiele. Oldenbourg, München 2007
- [23] W. D. Pietruszka, MATLAB in der Ingenieurspraxis. Teubner, Wiesbaden 2005
- [24] ScicosLab 2009. <http://www.scicoslab.org/>
- [25] eBook Grundlagen digitale Regelungstechnik und Mechatronik, <http://www.ebookaktiv.de>
- [26] Paulo S. Motta Pires and David A. Rogers, Free/Open Source Software: An Alternative for Engineering Students, 32nd ASEE/IEEE Frontiers in Education Conference, T3G7, 2002
- [27] K. Blankenbach, F. Schuhmacher, M. Jentsch, EXCEL und MATLAB als Ergänzung im Elektrotechnik-Labor. Beiträge zum 7. Tag der Lehre, Biberach 2007, S. 105

Zusätzliche Quellen zu Scilab:

- [28] C. Gomez (Ed.), Engineering and Scientific Computing with Scilab, Birkhäuser, Boston 1999
- [29] Jean-Marie Zogg, Arbeiten mit Scilab und Scicos für numerische Berechnungen  
[http://www.fh-htwchur.ch/uploads/media/Arbeiten\\_mit\\_Scilab\\_und\\_Scicos\\_v1\\_01.pdf](http://www.fh-htwchur.ch/uploads/media/Arbeiten_mit_Scilab_und_Scicos_v1_01.pdf)  
Dort auch Demos:
- [30] Alexander Stoffel, Kurzeinführung in Scilab, Institut für Nachrichtentechnik der Fachhochschule Köln,  
<http://alex.nt.fh-koeln.de/mapdf/scilabein.pdf>

## Verzeichnis der bisher erschienenen Beiträge

---

- |  |   |
|--|---|
| <p>52. <b>Werner Pepels</b> Aug. 1990<br/>Integrierte Kommunikation</p> <p>53. <b>Martin Dettinger-Klemm</b> Aug. 1990<br/>Grenzen der Wirtschaftsfreiheit. Überlegungen zum Thema: Freiheit und Verantwortung des Wissenschaftlers</p> <p>54. <b>Werner Pepels</b> Sept. 1990<br/>Mediaplanung – Über den Einsatz von Werbegeldern in Medien</p> <p>55. <b>Dieter Pflaum</b> Sept. 1990<br/>Werbeausbildung und Werbemöglichkeiten in der DDR</p> <p>56. <b>Rudi Kurz (Hrsg.)</b> Nov. 1990<br/>Ökologische Unternehmensführung – Herausforderung und Chance</p> <p>57. <b>Werner Pepels</b> Jan. 1991<br/>Verkaufsförderung – Versuch einer Systematisierung</p> <p>58. <b>Rupert Huth, Ulrich Wagner (Hrsg.)</b> Aug. 1991<br/>Volks- und betriebswirtschaftliche Abhandlungen. Prof. Dr. Dr. h.c. Tibor Karpati (Universität Osijek in Kroatien) zum siebzigsten Geburtstag. Mit einem Vorwort von R. Huth und Beiträgen von H.-J. Hof, H. Löffler, D. Pflaum, B. Runzheimer und U. Wagner</p> <p>59. <b>Hartmut Eisenmann</b> Okt. 1991<br/>Dokumentation über die Tätigkeit einer Industrie- und Handelskammer – Dargestellt am Beispiel der IHK Nordschwarzwald</p> <p>60. <b>Ursula Hoffmann-Lange</b> Dez. 1991<br/>Eliten und Demokratie: Unvereinbarkeit oder notwendiges Spannungsverhältnis?</p> <p>61. <b>Werner Pepels</b> Dez. 1991<br/>Elemente der Verkaufsgesprächsführung</p> <p>62. <b>Wolfgang Berger</b> Dez. 1991<br/>Qualifikationen und Kompetenzen eines Europa-managers</p> <p>63. <b>Günter Staub</b> Jan. 1992<br/>Der Begriff „Made in Germany“ – Seine Beurteilungskriterien</p> <p>64. <b>Martin W. Knöll, Hieronymus M. Lorenz</b> Mai 1992<br/>Gegenstandsbereich und Instrumente der Organisationsdiagnose im Rahmen von Organisationsentwicklungs (OE)-Maßnahmen</p> <p>65. <b>Werner Lachmann</b> Juni 1992<br/>Ethikversagen – Marktversagen</p> <p>66. <b>Paul Banfield</b> Juni 1993<br/>Observations On The Use Of Science As A Source Of Legitimation In Personnel Management</p> <p>67. <b>Bernd Noll</b> Aug. 1993<br/>Gemeinwohl und Eigennutz. Wirtschaftliches Handeln in Verantwortung für die Zukunft – Anmerkungen zur gleichnamigen Denkschrift der Evangelischen Kirche in Deutschland aus dem Jahre 1991</p> <p>68. <b>Siegfried Kreutzer, Regina Moczadlo</b> Aug. 1993<br/>Die Entdeckung der Wirklichkeit – Integrierte Projektstudien in der Hochschulausbildung</p> | <p>69. <b>Sybil Gräfin Schönfeldt</b> Aug. 1993<br/>Von Menschen und Manieren. Über den Wandel des sozialen Verhaltens in unserer Zeit. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Wintersemester 1992/93</p> <p>70. <b>Hartmut Löffler</b> Dez. 1993<br/>Geld- und währungspolitische Grundsatzüberlegungen für ein Land auf dem Weg zur Marktwirtschaft – Das Beispiel Kroatien</p> <p>71. <b>Hans-Georg Köglmayr, Kurt H. Porkert</b> Nov. 1994<br/>Festlegen und ausführen von Geschäftsprozessen mit Hilfe von SAP-Software</p> <p>72. <b>Alexa Mohl</b> Febr. 1995<br/>NLP-Methode zwischen Zauberei und Wissenschaft. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Wintersemester 1994/95</p> <p>73. <b>Bernd Noll</b> Mai 1995<br/>Marktwirtschaft und Gerechtigkeit: Anmerkungen zu einer langen Debatte</p> <p>74. <b>Rudi Kurz, Rolf-Werner Weber</b> Nov. 1995<br/>Ökobilanz der Hochschule Pforzheim. 2. geänderte Auflage, Jan. 1996</p> <p>75. <b>Hans Lenk</b> Mai 1996<br/>Fairneß in Sport und Wirtschaft. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Wintersemester 1995/96</p> <p>76. <b>Barbara Burkhardt-Reich, Hans-Joachim Hof, Bernd Noll</b> Juni 1996<br/>Herausforderungen an die Sozialstaatlichkeit der Bundesrepublik</p> <p>77. <b>Helmut Wienert</b> März 1997<br/>Perspektiven der Weltstahlindustrie und einige Konsequenzen für den Anlagenbau</p> <p>78. <b>Norbert Jost</b> Mai 1997<br/>Innovative Ingenieur-Werkstoffe</p> <p>79. <b>Rudi Kurz, Christoph Hubig, Ortwin Renn, Hans Diefenbacher</b> Sept. 1997<br/>Ansprüche in der Gegenwart zu Lasten der Lebenschancen zukünftiger Generationen</p> <p>80. <b>Björn Engholm</b> Okt. 1997<br/>Ökonomie und Ästhetik. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Wintersemester 1996/97. 2. geänderte Auflage. Jan. 1998</p> <p>81. <b>Lutz Goertz</b> Sept. 1998<br/>Multimedia quo vadis? – Wirkungen, Chancen, Gefahren. Vortrag gehalten im Rahmen des Studium Generale der Fachhochschule Pforzheim, Wintersemester 1996/97</p> <p>82. <b>Eckhard Keßler</b> Nov. 1998<br/>Der Humanismus und die Entstehung der modernen Wissenschaft. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Wintersemester 1996/97</p> <p>83. <b>Heinrich Hornef</b> Febr. 1998<br/>Aufbau Ost – Eine Herausforderung für Politik und Wirtschaft. Vortrag gehalten im Rahmen des Studium Generale der Fachhochschule Pforzheim, Wintersemester 1997/98</p> |
|--|---|

84. **Helmut Wienert** Juli 1998  
50 Jahre Soziale Marktwirtschaft – Auslaufmodell oder Zukunftskonzept? Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Sommersemester 1998  
Peter Kern, Wilhelm Bauer, Rolf Ilg; Heiko Dreyer; Johannes Wößner und Rainer Menge
85. **Bernd Noll** Sept. 1998  
Die Gesetzliche Rentenversicherung in der Krise
86. **Hartmut Löffler** Jan. 1999  
Geldpolitische Konzeptionen - Alternativen für die Europäische Zentralbank und für die Kroatische Nationalbank
87. **Erich Hoppmann** Juni 1999  
Globalisierung. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Sommersemester 1999
88. **Helmut Wienert (Hrsg.)** Dez. 1999  
Wettbewerbspolitische und strukturpolitische Konsequenzen der Globalisierung. Mit Beiträgen von Hartmut Löffler und Bernd Noll
89. **Ansgar Häfner u.a. (Hrsg.)** Jan. 2000  
Konsequenzen der Globalisierung für das internationale Marketing. Mit Beiträgen von Dieter Pflaum und Klaus-Peter Reuthal
90. **Ulrich Wagner** Febr. 2000  
Reform des Tarifvertragsrechts und Änderung der Verhaltensweisen der Tarifpartner als Voraussetzungen für eine wirksame Bekämpfung der Arbeitslosigkeit
91. **Helmut Wienert** April 2000  
Probleme des sektoralen und regionalen Wandels am Beispiel des Ruhrgebiets
92. **Barbara Burkhardt-Reich** Nov. 2000  
Der Blick über den Tellerrand – Zur Konzeption und Durchführung eines „Studium Generale“ an Fachhochschulen
93. **Helmut Wienert** Dez. 2000  
Konjunktur in Deutschland - Zur Einschätzung der Lage durch den Sachverständigenrat im Jahresgutachten 2000/2001
94. **Jürgen Wertheimer** Febr. 2001  
Geklonte Dummheit: Der infantile Menschenpark. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Wintersemester 2000/01
95. **Konrad Zerr** März 2001  
Erscheinungsformen des Online-Research – Klassifikation und kritische Betrachtung
96. **Daniela Kirchner** April 2001  
Theorie und praktische Umsetzung eines Risikomanagementsystems nach KontraG am Beispiel einer mittelständischen Versicherung
97. **Bernd Noll** Mai 2001  
Die EU-Kommission als Hüterin des Wettbewerbs und Kontrolleur von sektoralen und regionalen Beihilfen  
**Peter Frankenfeld**  
EU Regionalpolitik und Konsequenzen der Osterweiterung
98. **Hans Joachim Grupp** Juni 2001  
Prozessurale Probleme bei Beschlussmängelstreitigkeiten in Personengesellschaften
99. **Norbert Jost (Hrsg.)** Juli 2001  
Technik Forum 2000: Prozessinnovationen bei der Herstellung kaltgewalzter Drähte. Mit Beiträgen von
100. **Urban Bacher, Mikolaj Specht** Dez. 2001  
Optionen – Grundlagen, Funktionsweisen und deren professioneller Einsatz im Bankgeschäft
101. **Constanze Oberle** Okt. 2001  
Chancen, Risiken und Grenzen des M-Commerce
102. **Ulrich Wagner** Jan. 2002  
Beschäftigungshemmende Reformstaus und wie man sie auflösen könnte  
**Jürgen Volkert**  
Flexibilisierung durch Kombi-Einkommen? Die Perspektive der Neuen Politischen Ökonomie
103. **Mario Schmidt, René Keil** März 2002  
Stoffstromnetze und ihre Nutzung für mehr Kostentransparenz sowie die Analyse der Umweltwirkung betrieblicher Stoffströme
104. **Kurt Porkert** Mai 2002  
Web-Services – mehr als eine neue Illusion?
105. **Helmut Wienert** Juni 2002  
Der internationale Warenhandel im Spiegel von Handelsmatrizen
106. **Robert Wessolly, Helmut Wienert** Aug. 2002  
Die argentinische Währungskrise
107. **Roland Wahl (Hrsg.)** Sept. 2002  
Technik-Forum 2001: Weiterentwicklungen an Umformwerkzeugen und Walzdrähten. Mit Beiträgen von Roland Wahl, Thomas Dolny u.a., Heiko Pinkawa, Rainer Menge und Helmut Wienert
108. **Thomas Gulden** April 2003  
Risikoberichterstattung in den Geschäftsberichten der deutschen Automobilindustrie
109. **Günter Altner** Mai 2003  
Lasset uns Menschen machen – Der biotechnische Fortschritt zwischen Manipulation und Therapie. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Sommersemester 2003
110. **Norbert Jost (Hrsg.)** Juni 2003  
Technik-Forum 2002: Innovative Verfahren zur Materialoptimierung. Mit Beiträgen von Norbert Jost, Sascha Kunz, Rainer Menge/Ursula Christian und Berthold Leibinger
111. **Christoph Wüterich** Februar 2004  
Professionalisierung und Doping im Sport. Vortrag gehalten im Rahmen des Studium Generale der Hochschule Pforzheim, Sommersemester 2003
112. **Sabine Schmidt** Mai 2004  
Korruption in Unternehmen – Typologie und Prävention
113. **Helmut Wienert** August 2004  
Lohn, Zins, Preise und Beschäftigung – Eine empirische Analyse gesamtwirtschaftlicher Zusammenhänge in Deutschland
114. **Roland Wahl (Hrsg.)** Sept. 2004  
Technik-Forum 2003: Materialentwicklung für die Kaltumformtechnik. Mit Beiträgen von Andreas Baum, Ursula Christian, Steffen Nowotny, Norbert Jost, Rainer Menge und Hans-Eberhard Koch
115. **Dirk Wenzel** Nov. 2004  
The European Legislation on the New Media: An Appropriate Framework for the Information Economy?

116. **Frank Morelli, Alexander Mekyska, Stefan Mühlberger** Dez. 2004  
Produkt- und prozessorientiertes Controlling als Instrument eines erfolgreichen Informationstechnologie-Managements
117. **Stephan Thesmann, Martin Frick, Dominik Konrad** Dez. 2004  
E-Learning an der Hochschule Pforzheim
118. **Norbert Jost (Hrsg.)** Juni 2005  
Technik-Forum 2004: Innovative Werkstoffaspekte und Laserbehandlungstechnologien für Werkzeuge der Umformtechnik
119. **Rainer Gildeggen** Juni 2005  
Internationale Produkthaftung
120. **Helmut Wienert** Oktober 2005  
Qualifikationsspezifische Einkommensunterschiede in Deutschland unter besonderer Berücksichtigung von Universitäts- und Fachhochschulabsolventen
121. **Andreas Beisswenger, Bernd Noll** Nov. 2005  
Ethik in der Unternehmensberatung – ein vermintes Gelände?
122. **Helmut Wienert** Juli 2006  
Wie lohnend ist Lernen? Ertragsraten und Kapitalendwerte von unterschiedlichen Bildungswegen
123. **Roland Wahl (Hrsg.)** Sept. 2006  
Technik-Forum 2005: Umformwerkzeuge - Anforderungen und neue Anwendungen. Mit Beiträgen von Edmund Böhm, Eckhard Meiners, Andreas Baum, Ursula Christian und Jörg Menno Harms
124. **Mario Schmidt** Dez. 2006  
Der Einsatz von Sankey-Diagrammen im Stoffstrommanagement
125. **Norbert Jost (Hrsg.)** Okt. 2007  
Technik-Forum 2006: Innovative neue Techniken für Werkzeuge der Kaltverformung. Mit Beiträgen von Franz Wendl, Horst Bürkle, Rainer Menge, Michael Schiller, Andreas Baum, Ursula Christian, Manfred Moik und Erwin Staudt.
126. **Roland Wahl (Hrsg.)** Okt. 2008  
Technik-Forum 2007: Fortschrittsberichte und Umfeldbetrachtungen zur Entwicklung verschleißreduzierter Umformwerkzeuge. Mit Beiträgen von Klaus Löffler, Andreas Zilly, Andreas Baum und Paul Kirchhoff.
127. **Julia Tokai, Christa Wehner** Oktober 2008  
Konzept und Resultate einer Online-Befragung von Marketing-Professoren an deutschen Fachhochschulen zum Bologna-Prozess
128. **Thomas Cleff, Lisa Luppold, Gabriele Naderer, Jürgen Volkert** Dez. 2008  
Tätermotivation in der Wirtschaftskriminalität
129. **Frank Thusselt** Juni 2009  
Das Arbeiten mit Numerik-Programmen. MATLAB, Scilab und Octave in der Anwendung