MATLAB® – Simulink® – Stateflow®

Grundlagen, Toolboxen, Beispiele

von
Dr.-Ing. Anne Angermann
Dr.-Ing. Michael Beuschel
Dr.-Ing. Martin Rau
Dipl.-Ing. Ulrich Wohlfarth

7., aktualisierte Auflage

Oldenbourg Verlag München

Dr.-Ing. Anne Angermann, TU München
Dr.-Ing. Michael Beuschel, Conti Temic microelectronic GmbH, Ingolstadt
Dr.-Ing. Martin Rau, BMW AG, München
Dipl.-Ing. Ulrich Wohlfarth, Patentanwaltskanzlei Charrier, Rapp & Liebau, Augsburg

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2098 USA
Tel: 508-647-7000; Fax: 508-647-7001
E-mail: info@mathworks.com; Web: www.mathworks.com

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
www.oldenbourg-verlag.de

Lektorat: Martin Preuß
Herstellung: Constanze Müller
Einbandgestaltung: hauser lacour
Gesamtherstellung: Grafik + Druck GmbH, München

Dieses Papier ist alterungsbeständig nach DIN/ISO 9706.

ISBN 978-3-486-70585-0

Vorwort zur siebten Auflage

Die 7. Auflage wurde im Hinblick auf das MATLAB Release 2011a vollständig überarbeitet und ergänzt, ohne jedoch den Umfang des Buches wesentlich zu erhöhen. Das Ergebnis ist ein kompaktes Lehrbuch für den Einsteiger und gleichzeitig ein übersichtliches Nachschlagewerk für den fortgeschrittenen MATLAB-Anwender.

Die dem Buch beigelegte CD-ROM ist mit einem benutzerfreundlichen HTML-Navigator versehen. Neben den überarbeiteten Beispielen, Übungsaufgaben und Lösungen enthält die CD-ROM wieder eine Bibliothek der Autoren mit nützlichen Extras für MATLAB und Simulink.

Danken möchten wir zuallererst Herrn Professor i. R. Dierk Schröder für seine umfassende Unterstützung bei der Erstellung dieses Buches. Ausgehend von seiner Idee und seinem unermüdlichen Engagement, eine längst notwendige Vorlesung zum Softwarepaket MATLAB und Simulink für Studenten der Fachrichtungen Energietechnik, Automatisierungstechnik und Mechatronik ins Leben zu rufen, konnten wir in sehr freier und kollegialer Arbeitsweise ein Skriptum zu dieser Vorlesung erstellen.

Nach einem ruhestandsbedingten Wechsel kann dieses Engagement unter Leitung von Herrn Professor Ralph Kennel, Ordinarius des Lehrstuhls für Elektrische Antriebssysteme und Leistungselektronik der Technischen Universität München, in vollem Umfang fortgesetzt werden. Für seine immer wohlwollende Unterstützung sowie für die vertrauensvolle und angenehme Zusammenarbeit danken wir ihm daher sehr herzlich.

Die äußerst positive Resonanz von Studenten unterschiedlichster Fachrichtungen sowie zahlreiche Anfragen aus Forschung und Industrie ermutigten uns, das ursprüngliche Skriptum einem größeren Leserkreis zu erschließen und als Buch zu veröffentlichen. Aufgrund der regen Nachfrage erscheint dieses Buch nun bereits in seiner 7. Auflage. Nicht zuletzt danken wir daher unseren zahlreichen Lesern, allen Dozenten, Studenten und Kollegen, die uns dabei mit ihren Anregungen und ihrer stets wohlwollenden Kritik unterstützten und noch unterstützen werden.

Für Verbesserungvorschläge und Hinweise auf noch vorhandene Fehler sind wir jederzeit dankbar und werden sie auf der Internetseite **www.matlabbuch.de** neben weiteren aktuellen Informationen rund um MATLAB veröffentlichen.

Dem Oldenbourg Verlag danken wir für die Bereitschaft, dieses Buch zu verlegen. Besonderer Dank gilt hierbei Frau Mönch, Frau Dr. Bromm, Herrn Dr. Preuß sowie Herrn Schmid für ihre hilfreiche Unterstützung während der Entstehung und für die Übernahme des Lektorats.

München

Anne Angermann
Michael Beuschel
Martin Rau
Ulrich Wohlfarth

Vorwort zur ersten Auflage

Das vorliegende Buch „Matlab – Simulink – Stateflow“ wendet sich an *Studenten* und *Ingenieure*, die das Simulationswerkzeug MATLAB/Simulink effizient einsetzen wollen.

Zielsetzung dieses Buches ist es, dem Leser einen direkten Zugang zum Anwenden der umfangreichen Möglichkeiten dieses Programmpaketes zu ermöglichen. Es wird prägnant dargestellt, welche wesentlichen Funktionen in MATLAB und Simulink verfügbar sind – beispielsweise Ein- und Ausgabe, grafische Darstellungen oder die verschiedenen Toolboxen und Blocksets für die Behandlung zeitkontinuierlicher und zeitdiskreter linearer und nichtlinearer Systeme sowie ereignisdiskreter Systeme – und wie diese Funktionen zu nutzen sind. Dies wird außerdem an zahlreichen Beispielen erläutert. Um den Ansatz *prägnante Einführung* zu unterstützen, sind zudem zu jedem Abschnitt Übungsaufgaben mit deren Lösungen auf einer CD-ROM beigefügt. Der Leser hat somit die Option, sein Verständnis des betreffenden Kapitels selbständig zu vertiefen und sofort praktisch zu überprüfen.

Um den Umfang dieses Buches zu begrenzen, sind keine theoretischen Abhandlungen z.B. über Integrationsverfahren, die Grundlagen der Regelungstechnik oder der Signalverarbeitung bzw. deren Implementierungen enthalten. Für den interessierten Leser finden sich jedoch in jedem Kapitel Hinweise auf vertiefende Literatur.

Ausgangspunkt dieses Buches war meine Überlegung, Studenten so früh wie möglich mit den Vorteilen des wertvollen Werkzeuges *Simulation* bekannt zu machen. Dies beginnt bei regelungstechnischen Aufgabenstellungen bereits bei der Modellbildung der Komponenten des betrachteten Systems und der Erstellung des Simulationsprogramms. Es setzt sich fort bei der Validierung der Modelle sowie der grafischen Veranschaulichung des Systemverhaltens in den verschiedenen Arbeitspunkten und bei unterschiedlichen Randbedingungen, etwa aufgrund variabler Parameter, der Systemanalyse, der Reglersynthese sowie der Optimierung des Gesamtsystems. Ausgehend von diesen Überlegungen wurde ein Konzept für dieses Buch erarbeitet, damit der Leser die verschiedenen Aufgabenstellungen möglichst anschaulich kennen lernt. Dieses Konzept wurde aufgrund langjähriger Erfahrung bei Vorlesungen, Studien- und Diplomarbeiten, Dissertationen und Industrieprojekten kontinuierlich verbessert.

Meine Mitarbeiter und ich hoffen, allen Interessierten in Studium und Beruf mit diesem Buch einen anschaulichen und effizienten Einstieg in die Simulation mit MATLAB und Simulink geben zu können.

München Dierk Schröder

Inhaltsverzeichnis

Datenimport und -export binär – Formate

`uchar`, `uint8`, `uint16`, `uint32`, `uint64`	Formate ohne Vorzeichen
`int8`, `int16`, `int32`, `int64`	Formate mit Vorzeichen
`single`, `double`, `float32`, `float64`	Formate Fließkomma
`bit`N, `ubit`N, $1 \leq N \leq 64$	N Bits mit/ohne Vorzeichen

3.4 Betriebssystemaufruf und Dateiverwaltung

Damit MATLAB ein aufgerufenes M-File oder Daten findet, muss sich die zugehörige Datei im aktuellen Verzeichnis oder im MATLAB-**Pfad** befinden. Der MATLAB-Pfad kann im *Path Browser* eingesehen und geändert werden. Der Aufruf erfolgt über das Menü *File/Set Path* oder den Befehl `pathtool`. Im Path Browser kann man über den Button *Add Folder* dem MATLAB-Pfad ein neues Verzeichnis hinzufügen.

Ein Ausrufezeichen (`!`) am Beginn einer Zeile veranlasst MATLAB, den Rest der Zeile dem **Betriebssystem** als Kommando zu übergeben. Werden Rückgabewerte benötigt, eignet sich der Befehl `[`*status*, *ergebnis*`] = system (`*kommando*`)`. Hilfreich kann hier auch der mächtige (aber meist unübersichtliche!) Befehl `eval(`*string*`)` sein: Dieser Befehl interpretiert einen String als MATLAB-Befehl. Das folgende Beispiel erstellt ein Verzeichnis, weist dessen Namen der Struct-Variablen `verzeichnis` zu und löscht das Verzeichnis wieder. Eine weitere Übersicht erhält man mit `help general`.

```
>> mkdir ('testverzeichnis')              % Verzeichnis anlegen (MATLAB-Befehl)
>> verzeichnis = dir ('testverzeic*');    % Dir-Befehl ausführen (MATLAB-Befehl)
>> eval (['!rmdir ', verzeichnis.name])   % Verzeichnis löschen (Betriebssystem)
```

Die nachfolgend aufgeführten MATLAB-Befehle erlauben, ähnlich wie in gängigen Betriebssystemen, aus MATLAB heraus Verzeichnisse und **Dateien** zu **verwalten** (manuell ist dies natürlich auch im *Current Folder Browser* möglich).

Betriebssystemaufruf und Dateiverwaltung

`cd` *verzeichnis*	Verzeichniswechsel
`pwd`	Anzeige des aktuellen Verzeichnisses
`dir` [*auswahl*]	Anzeige Verzeichnis-Inhalt
`ls` [*auswahl*]	Anzeige Verzeichnis-Inhalt
`mkdir` *verzeichnis*	Neues Verzeichnis erstellen
`copyfile` *quelle ziel*	Datei kopieren
`delete` *datei*	Datei löschen
`!` *kommando*	Betriebssystemaufruf
`system (`*kommando*`)`	Betriebssystemaufruf mit Rückgabewerten
`eval (`*string*`)`	String als MATLAB-Befehl interpretieren

3.5 Grafische Darstellung

Insbesondere bei großen Datenmengen ist eine numerische Ausgabe wenig anschaulich. Daher werden nun Möglichkeiten zur grafischen Ausgabe behandelt. Zuerst werden die MATLAB-Figure sowie die für alle grafischen Ausgaben gleichermaßen gültigen Befehle, wie Achsen-Skalierung und Beschriftung, angesprochen. In den Kapiteln 3.5.3 und 3.5.4 werden die spezifischen Befehle für zwei- und dreidimensionale Grafiken beschrieben und an Beispielen ausführlich dargestellt; Kapitel 3.5.6 zeigt Möglichkeiten zum Import, Export und Drucken von Grafiken.

3.5.1 Die Figure – Grundlage einer MATLAB-Grafik

Den Rahmen für alle grafischen Ausgaben stellt die so genannte MATLAB-*Figure* dar, die mit `figure` erzeugt und mit einer Nummer versehen wird. Eine vorhandene Figure kann mit `figure (`*nummer*`)` angesprochen werden, die Nummer (eigentlich das *Handle*) einer Figure erhält man mit `gcf` (*Get handle to Current Figure*). Alle Grafikbefehle wirken stets auf die zuletzt erzeugte bzw. angesprochene Figure.

Abb. 3.1 wurde mit den folgenden Befehlen erzeugt. Falls keine Nummer explizit angegeben wird, vergibt MATLAB diese automatisch (hier `1`).

```
>> figure
>> plot (diff (primes (500)))              % Abstand zwischen Primzahlen < 500
```

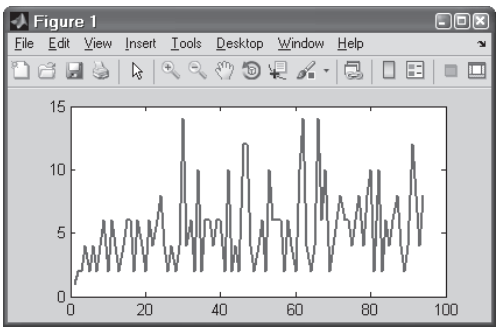

Abb. 3.1: MATLAB-*Figure*

Eine Figure kann mit `subplot (`*zeilen*, *spalten*, *zaehler*`)` gleichmäßig in so genannte *Subplots* unterteilt werden. Der jeweils angesprochene Subplot wird durch die Variable *zaehler* bestimmt; es wird zeilenweise von links oben durchnummeriert. Sind alle Argumente einstellig, können sie ohne Komma (und ohne Leerzeichen!) hintereinander geschrieben werden (siehe Beispiel auf Seite 47).

Der Inhalt einer vorhandenen Figure kann mit `clf` (*Clear Figure*) gelöscht werden. Die Figure selbst kann mit `close` *nummer* geschlossen (d.h. gelöscht) werden. `close` schließt die aktuelle Figure; `close all` schließt alle geöffneten Figures.

MATLAB verwaltet alle Grafiken objektorientiert. Die Objekt-Hierarchie für Grafikobjekte ist in Abb. 3.2 dargestellt. Die Eigenschaften der aktuellen Figure erhält man mit dem Befehl `get (`*FigureHandle*`)` bzw. `get (gcf)`, die Eigenschaften des aktuellen Subplots über das Handle `gca` (*Get handle to Current Axis*) entsprechend mit `get (gca)`.

Des Weiteren gibt `set(gcf)` bzw. `set(gca)` eine Liste der möglichen Parameter aus. Das Abfragen und Setzen einzelner Eigenschaften geschieht wie folgt:[5)]

`get(`*handle*`,'`*eigenschaft*`')`
`set(`*handle*`,'`*eigenschaft*`',`*wert*`)`

Bei etlichen Grafikbefehlen (z.B. `plot`, `xlabel`, `text`, `legend`, siehe ab Kap. 3.5.3) lassen sich Eigenschaften und zugehörige Werte in gleicher Weise als zusätzliche Parameter übergeben.

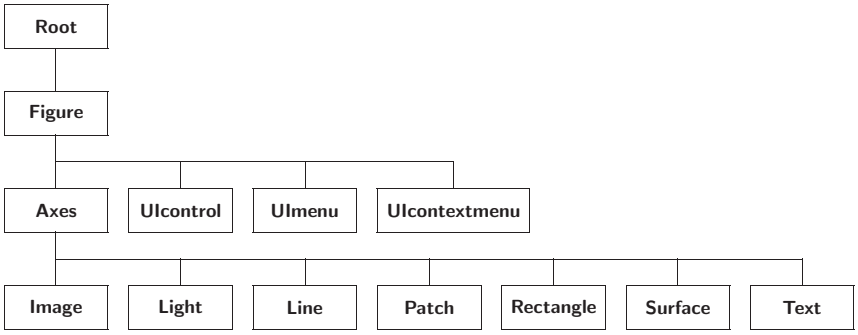

Abb. 3.2: *Objekt-Hierarchie für Grafikobjekte*

Über das Figure-Menü *View* kann der *Property Editor* aufgerufen werden; er erleichtert das Setzen einzelner Eigenschaften der Figure und aller anderen Grafikobjekte, wie Achsen, Beschriftung, Linienstärken, Skalierung etc. Ist der Button aktiviert, können Objekte in der Figure angeklickt und dann deren Eigenschaften verändert werden.

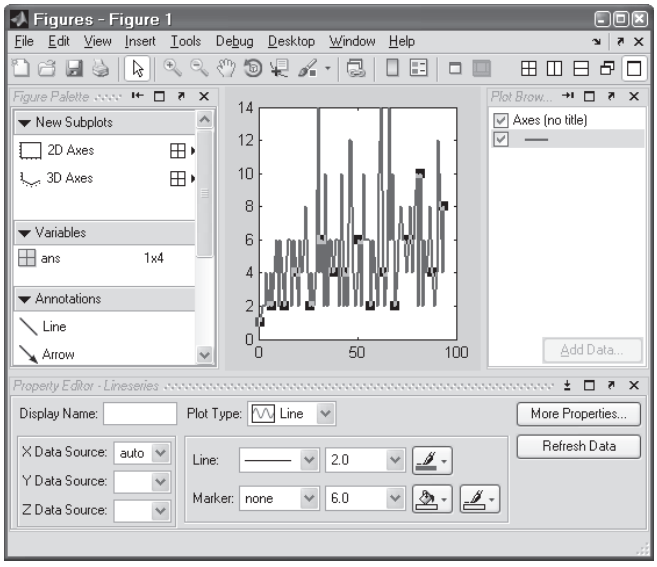

Abb. 3.3: MATLAB-*Figure mit Werkzeugen*

[5)] Die übergebenen Werte können je nach Eigenschaft eine Zahl, ein Array oder ein String sein.

Über *Insert* kann die Figure mit frei platzierbaren Grafik- und Textobjekten versehen werden. Diese Funktionen sind auch über das Figure-Menü *View/Plot Edit Toolbar* und die dann eingeblendete Funktionsleiste verfügbar (siehe Abb. 3.3).

Weitere Werkzeuge können ebenfalls über das Figure-Menü *View* aufgerufen werden: Die *Figure Palette* erlaubt das Hinzufügen weiterer Daten. Über *New Subplots* werden Subplots eingefügt. Ebenso können Subplots mit der Maus verschoben, skaliert sowie auch gelöscht werden. *Variables* bietet im Kontextmenü an, verfügbare Variablen zu plotten. Im *Plot Browser* können Objekte der Figure einzeln unsichtbar geschaltet werden. Die Figure-Buttons aktivieren bzw. deaktivieren die zuvor gewählten Werkzeuge (siehe ebenfalls Abb. 3.3).

Das so gestaltete Layout einer Figure kann über das Figure-Menü *File/Generate Code* als MATLAB-Funktion ausgegeben werden. Diese lässt sich dann mit anderen Daten wiederverwenden.

Grafik allgemein

`figure` [`(`*nummer*`)`]	Erzeugen oder Ansprechen einer Figure
`subplot (`*zeilen*, *spalten*, *zaehler*`)`	Erzeugen oder Ansprechen eines Subplots
`clf`	Rücksetzen der aktuellen Figure
`close` *nummer*	Figure *nummer* schließen (löschen)
`close all`	Alle Figures schließen (löschen)
`gcf`	Aktuelle Figurenummer (*Handle*)
`gca`	Aktueller Subplot (*Handle*)
`get (`*handle*, `'`*eigenschaft*`')`	Eigenschaft auslesen
`set (`*handle*, `'`*eigenschaft*`'`, *wert*`)`	Eigenschaft setzen

3.5.2 Achsen und Beschriftung

Die **Skalierung der Achsen** erfolgt automatisch, kann aber auch manuell mit dem Befehl `axis ([`*xmin*, *xmax*, *ymin*, *ymax*`])` für zweidimensionale bzw. mit `axis ([`$x1$, $x2$, $y1$, $y2$, $z1$, $z2$`])` für dreidimensionale Grafiken eingestellt werden. Mit `axis ('auto')` wird die Skalierung wieder MATLAB überlassen. Die Befehle `xlim ([`*xmin*, *xmax*`])`, `ylim` und `zlim` skalieren jeweils die angegebene Achse.

Der Befehl `grid on` erzeugt ein **Gitternetz** entsprechend der Achsenteilung.[6)]

Zur genaueren Betrachtung von **Ausschnitten eines Plots** kann nach dem Befehl `zoom on` (oder nach Anklicken eines der Buttons im Figure-Fenster) die Figure mit der Maus gezoomt werden. Ein Doppelklick in die Figure stellt die ursprüngliche Skalierung wieder her. Über das Menü *Tools/Pan* oder lässt sich der Ausschnitt einer gezoomten Figure mit der Maus verschieben (weitere Optionen über Kontextmenü).

Zur **Beschriftung** der Plots bestehen mehrere Möglichkeiten: Mit `xlabel (`*string*`)` (und entsprechend `ylabel` und `zlabel`) werden die **Achsen** beschriftet; mit

[6)] Änderung der Achsenteilung mittels Property Editor oder wie im Beispiel auf Seite 49 gezeigt.

dem Befehl `title(`*string*`)` wird eine **Überschrift** erzeugt. Dabei können entsprechend der LaTeX-Konventionen hoch- und tiefgestellte Zeichen sowie griechische Buchstaben verwendet werden.[7] So ergibt z.B. `xlabel ('\alpha_{yz} = b^3/c \rightarrow \pm \infty')` die Beschriftung $\alpha_{yz} = b^3/c \rightarrow \pm\infty$.

Mit `legend(`*string*1, *string*2 ... [, `'Location'`, *richtung*]`)` kann eine **Legende** erzeugt werden. Die Strings werden entsprechend der Reihenfolge der `plot`-Befehle zugeordnet. Die Position der Legende wird mit *richtung* als (englische) Himmelsrichtung angegeben: `'NE'` platziert z.B. rechts oben (Standard), `'W'` links, `'Best'` automatisch und `'BestOutside'` neben dem Plot. Der Button schaltet eine Legende zwischen sichtbar und unsichtbar um. Weitere Infos über `doc legend`.

Der Befehl `text(`*x*, *y*, *string*`)` schließlich **platziert** einen **Text** an einer beliebigen Koordinate im aktuellen Plot. Den genannten Beschriftungsbefehlen können optional weitere Parameter übergeben werden, z.B. `title('Überschrift','FontSize',16, 'FontWeight','Bold')`. Weitere Infos zur grafischen Ausgabe erhält man mit `help graph2d`, `help graph3d` und `help specgraph` (Spezialbefehle); siehe auch Kap. 3.7.

Grafik: Achsen

`axis([`*xmin*, *xmax*, *ymin*, *ymax*`])`	Manuelle Achsen-Skalierung (2D)
`axis([`*x*1, *x*2, *y*1, *y*2, *z*1, *z*2`])`	Manuelle Achsen-Skalierung (3D)
`axis('auto')`	Automatische Achsen-Skalierung
`xlim([`*xmin*, *xmax*`])`	Manuelle Skalierung der x-Achse
`ylim([`*ymin*, *ymax*`])`	Manuelle Skalierung der y-Achse
`zlim([`*zmin*, *zmax*`])`	Manuelle Skalierung der z-Achse
`grid` [`on` | `off`]	Gitternetz ein | aus
`zoom` [`on` | `off`]	Zoomfunktion ein | aus

Grafik: Beschriftung

`xlabel(`*string*`)`	Beschriftung der x-Achse
`ylabel(`*string*`)`	Beschriftung der y-Achse
`zlabel(`*string*`)`	Beschriftung der z-Achse
`title(`*string*`)`	Überschrift erzeugen
`text(`*x*, *y*, *string*`)`	Text an Koordinate (*x*, *y*) platzieren
`legend(`*string*1, ... [, `'Location'`, ...]`)`	Legende erzeugen

3.5.3 Plot-Befehle für zweidimensionale Grafiken

Der Befehl `plot(`*xwerte*, *ywerte* ... [, *plotstil*]`)` zeichnet die als Wertepaare (*xwerte*, *ywerte*) gegebenen Punkte. Diese werden standardmäßig mit einer blauen Linie verbunden. Werden als Argument mehrere x/y-Vektoren abwechselnd übergeben, erhält man unabhängige Linien. Entfällt *xwerte*, werden die Werte *ywerte* über ihrem Index geplottet; enthält *ywerte* dabei komplexe Zahlen, wird der Imaginärteil über dem Realteil dargestellt.

[7] Die in Kap. 3.2 beschriebenen Sonderzeichen `\n`, `''` sowie `\\` gelten hier jedoch nicht.

Der Befehl `stairs` verwendet dieselbe Syntax, erzeugt aber eine treppenförmige Linie (z.B. für abgetastete Signale, siehe Beispiel in Kap. 10.3.2). Die Befehle `bar` und `stem` erzeugen Balkendiagramme (siehe Spektren in Kap. 6.2). Die Befehle `semilogx` und `semilogy` besitzen die gleiche Syntax wie `plot` mit dem Unterschied, dass die x-Achse bzw. y-Achse logarithmisch skaliert wird. `loglog` plottet beide Achsen logarithmisch. Der Befehl `polar` plottet Daten in Polarkoordinaten.

Der Parameter *plotstil* ist ein String, der Farbe, Linien- und ggf. Punkttyp eines Plots bestimmt. Aus allen drei Kategorien kann *plotstil* je ein Element enthalten (muss aber nicht). So steht z.B. `'g-.'` für eine grüne gestrichelte Linie (— · —); `'ro-'` steht für eine rote durchgezogene Linie, bei der jeder Datenpunkt zusätzlich mit einem Kreis markiert ist. Darüber hinaus können optional Eigenschaft/Werte-Paare angegeben werden. So erzeugt z.B. `plot(x,y,'r','LineWidth',2.0)` einen Plot mit dicker roter Linie. Weitere Infos über `doc plot` oder in Kap. 3.7.

Farben

`k`	schwarz	`r`	rot
`b`	blau	`m`	magenta
`c`	cyan	`y`	gelb
`g`	grün	`w`	weiß

Punkte

`.`	Punkte
`o`	Kreise
`*`	Sterne
`+, x`	Kreuze

Linien

`-`	durchgezogen
`--`	gestrichelt
`-.`	gestrichelt
`:`	gepunktet

Standardmäßig löscht jeder neue `plot`-Befehl zunächst alle vorhandenen Objekte der aktuellen Figure (bzw. des Subplots). Dies wird mit dem Befehl `hold on` nach dem ersten `plot`-Befehl verhindert. Das folgende Beispiel erzeugt Abb. 3.4:

```
figure                                              % Erzeugt neue Figure
subplot (121)                                       % Linker Subplot
    plot (-5:0.1:5, cos ((-5:0.1:5)*pi), 'k:')      % schwarz, gepunktet
    hold on                                         % alten Plot beibehalten
    fplot ('2*sin(pi*x)/(pi*x)', [-5 5], 'k--')     % schwarz, gestrichelt
subplot (122)                                       % Rechter Subplot
    t = (0:20)*0.9*pi;
    plot (cos (t), sin (t))                         % Standard: blaue Linie
```

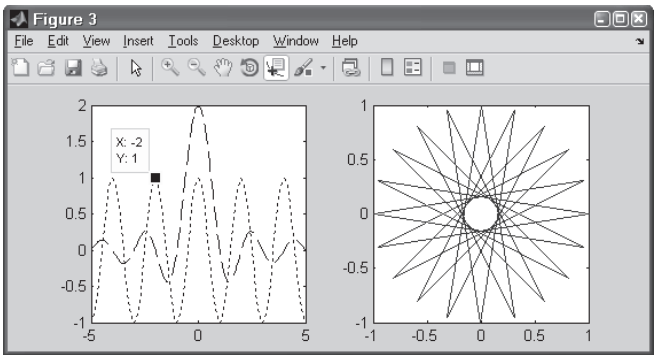

Abb. 3.4: MATLAB-*Figure mit zwei Subplots und über das Menü Tools aktiviertem Data Cursor (links)*

Der verwendete Befehl `fplot (`*funktion*, *bereich* [, *plotstil*]`)` dient der einfachen Darstellung expliziter Funktionen, die als String angegeben werden. Mit erweiterter Funktionalität steht darüber hinaus `ezplot (`*funktion*1, *funktion*2 [, *bereich*]`)` zur Verfügung, um auch implizite Funktionen und Parameterkurven zu plotten (siehe Abb. 3.5).

```
>> ezplot ('x^2 - y^2 - 2')
>> ezplot ('sin(3*t) * cos(t) / (t+pi)', ...
          'sin(3*t) * sin(t) / (t+pi)', [0, 4*pi])
```

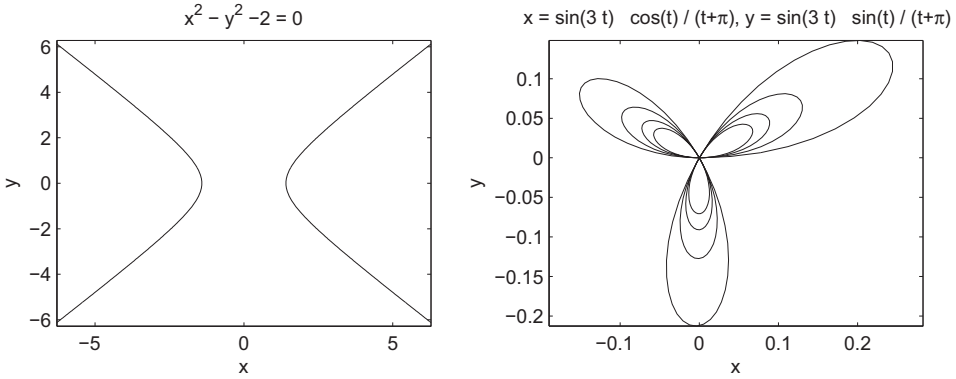

Abb. 3.5: *Implizite Funktion (links) und Parameterkurve (rechts) mit dem* `ezplot`*-Befehl*

Grafik: 2D Plot-Befehle

`plot (`[*xwerte*,] *ywerte* ... [, *plotstil*]`)`	Plot, linear
`stairs (`[*xwerte*,] *ywerte* ... [, *plotstil*]`)`	Plot, linear treppenförmig
`bar (...), stem (...)`	Plot, linear, Balken
`loglog (`*xwerte*, *ywerte* ... [, *plotstil*]`)`	Plot, beide Achsen logarithmisch
`semilogx (`*xwerte*, *ywerte* ... [, *plotstil*]`)`	Plot, x-Achse logarithmisch
`semilogy (`*xwerte*, *ywerte* ... [, *plotstil*]`)`	Plot, y-Achse logarithmisch
`polar (`*winkel*, *radius* ... [, *plotstil*]`)`	Plot, in Polarkoordinaten
`fplot (`*funktion*, *bereich*`)`	Plot, explizite Funktion
`ezplot (`*funktion* (*x*, *y*) [, *bereich*]`)`	Plot, implizite Funktion
`ezplot (`*funktion*1, *funktion*2 [, *bereich*]`)`	Plot, Parameterkurve
`hold` [`on` | `off`]	Vorhandene Objekte halten

Beispiel: Frequenzgang Tiefpass und Hochpass

Im Folgenden findet sich ein (zugegebenermaßen kompliziertes) Beispiel, das Abb. 3.6 erzeugt. Dabei wird der Frequenzgang eines Tiefpasses (auch PT_1-Glied) und eines Hochpasses (auch DT_1-Glied) mit der Zeitkonstanten T im Laplace-Bereich ($s = j\omega$) berechnet und ausgegeben. Die Eckfrequenz liegt jeweils bei $1/T = 20\,rad\,s^{-1} = 3.18\,Hz$.

$$H_{TP}(j\,\omega) = \frac{1}{1 + j\,\omega\,T} \qquad H_{HP}(j\,\omega) = 1 - H_{TP} = \frac{j\,\omega\,T}{1 + j\,\omega\,T}$$

Hinweis für Interessierte: Mit `set` wird zunächst die Achsenteilung geändert und anschließend die Beschriftung der Achsen; `gca` liefert das Handle des jeweiligen Subplots, um die Parameter `ytick` und `yticklabel` zu ändern.

```
T                = 0.05;                              % Zeitkonstante [s]
omega            = logspace (0, 3, 100);              % Frequenzvektor [rad/s]

frequenzgang_TP  = (T*j*omega + 1).^(-1);             % Frequenzgang Tiefpass
frequenzgang_HP  = 1 - frequenzgang_TP;               % Frequenzgang Hochpass

figure (3)                                            % Erzeugt Figure 3
clf                                                   % Löscht ggf. alte Plots

subplot (121)                                         % Linker Subplot: Amplitude
    loglog (omega, abs (frequenzgang_TP), '-')        % Amplitudengang Tiefpass
    hold on
    loglog (omega, abs (frequenzgang_HP), '--')       % Amplitudengang Hochpass
    loglog ([1 1]/T, [0.01 10], ':')                  % Linie Zeitkonstante
    text (1.1/T, 0.02, 'Zeitkonstante')               % Text Zeitkonstante
    title ('Übertragungsfunktionen')                  % Titel linker Subplot
    xlabel ('\omega [rad s^-^1]')                     % Achsenbeschriftung X
    ylabel ('Amplitude [dB]')                         % Achsenbeschriftung Y
    legend ('Tiefpass', 'Hochpass')                   % Legende an Standardposition
    xlim ([1 1e3])                                    % Skalierung x-Achse
    skala = -40:20:40;                                % Gewünschte Achsteilung
    set (gca, 'ytick', 10.^(skala/20))                % Setzt Achsteilung (dB)
    set (gca, 'yticklabel', skala)                    % Setzt Beschriftung

subplot (122)                                         % Rechter Subplot: Phase
    semilogx (omega, angle (frequenzgang_TP) *180/pi, '-')  % Phasengang TP
    hold on
    semilogx (omega, angle (frequenzgang_HP) *180/pi, '--') % Phasengang HP
    grid on                                           % Aktiviert Gitterlinien
    xlabel ('\omega [rad s^-^1]')                     % Achsenbeschriftung X
    ylabel ('Phase [Grad]')                           % Achsenbeschriftung Y
    axis ([1 1e3 -90 90])                             % Skalierung x- und y-Achse
    skala = -90:30:90;                                % Gewünschte Achsteilung
    set (gca, 'ytick', skala)                         % Setzt Achsteilung (Grad)
    set (gca, 'yticklabel', skala)                    % Setzt Beschriftung
```

Hinweis für immer noch Interessierte: Der Frequenzgang kann auch mit den in Kap. 5 behandelten Befehlen `tf` und `bode` dargestellt werden:

```
>> tiefpass = tf ([1], [0.05 1]);
>> hochpass = 1 - tiefpass;
>> bode (tiefpass, 'b-', hochpass, 'r--')
```

Mittels des Befehls `tf` wird die Übertragungsfunktion im Laplace-Bereich, bestehend aus Zähler- und Nennerpolynom, generiert. Mit dem Befehl `bode` wird automatisch ein Bodediagramm erzeugt.

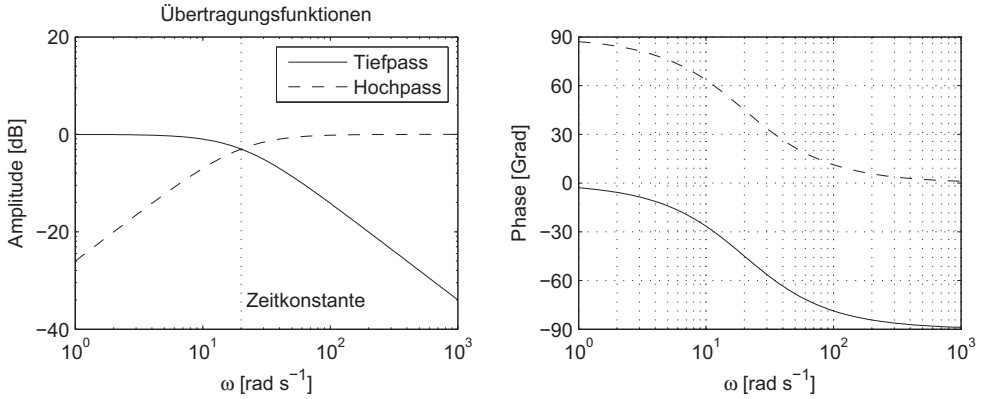

Abb. 3.6: *Beispiel: Frequenz- und Phasengang eines Tiefpasses und eines Hochpasses*

3.5.4 Plot-Befehle für dreidimensionale Grafiken

Zur Darstellung mehrdimensionaler Zusammenhänge eignen sich 3D-Plots. Im Folgenden werden einige der dafür verfügbaren Befehle beschrieben. Die Befehle zur Beschriftung und zur Achsen-Skalierung leiten sich dabei von denen bei 2D-Plots ab.

Der Befehl `plot3` entspricht `plot`, lediglich um einen dritten Datenvektor für die z-Achse erweitert. Folgende Befehle erzeugen den dreidimensionalen Plot in Abb. 3.7:

```
>> phi = (0:100) / 100 * 2*pi;
>> plot3 (-sin (2*phi), cos (3*phi), 1.5*phi, 'k.-')
>> grid on
```

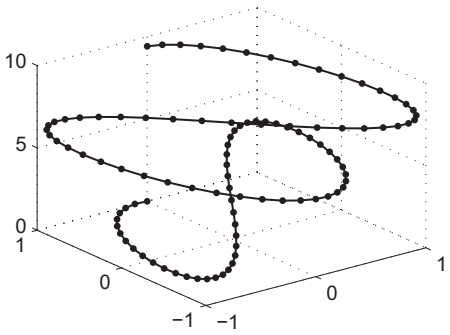

Abb. 3.7: *Dreidimensionaler Plot*

Zur Darstellung zweidimensionaler Funktionen als Flächen im Raum dient der Befehl `surf` (*xwerte*, *ywerte*, *zwerte* ... [, *farbe*]). Sind *xwerte*, *ywerte* und *zwerte* Matrizen gleicher Zeilen- und Spaltenzahl, werden die beschriebenen Punkte geplottet und die dazwischen liegenden Flächen ausgefüllt. Liegen alle Punkte in einem gleichmäßigen Raster bezüglich der x- und y-Achse, können *xwerte* und *ywerte* auch Vektoren sein. In diesem Fall werden die Einträge von *xwerte* auf die Spalten und *ywerte* auf die Zeilen der Matrix *zwerte* bezogen.

Die Befehle `mesh` und `waterfall` besitzen dieselbe Syntax wie `surf`, erzeugen aber ein Gitter ohne ausgefüllte Flächen bzw. ein Wasserfall-Diagramm. Dagegen erzeugt `contour` einen Plot der „Höhenlinien“ (Linien gleicher *zwerte*).

Optional kann eine weitere Matrix *farbe* zur Festlegung der Farbe angegeben werden. Jedes Element von *farbe* entspricht einem Element von *zwerte*. Die Farb-Werte werden als Indizes für eine Farbtabelle verwendet, die über `colormap (`*name*`)` geändert werden kann. Der größte und kleinste Eintrag von *farbe* wird automatisch auf den größten bzw. kleinsten Datenwert skaliert. Ohne Angabe einer Farb-Matrix wird *farbe* = *zwerte* angenommen. Die Skalierung der Farbe kann mit `caxis (`*farbe_min*, *farbe_max*`)` festgelegt werden. Infos über vordefinierte Farbtabellen sind über `help graph3d` zu erhalten.

3.5.5 Perspektive

Die Perspektive von 3D-Plots kann mit `view (`*horizontal*, *vertikal*`)` durch Vorgabe des horizontalen und vertikalen Winkels (in Grad) verändert werden. Standard ist $(-37.5°, 30°)$. Verschiedene Möglichkeiten sind in Abb. 3.8 zu sehen. Die Perspektive kann auch interaktiv durch Ziehen mit der Maus verändert werden, nachdem der Befehl `rotate3d on` eingegeben oder der Button  im Figure-Fenster aktiviert wurde. Für große Grafiken sollte man dabei über das Kontextmenü die Option *Rotate Options/Plot Box Rotate* wählen, damit die Grafik nicht während des Drehens neu gezeichnet wird.

Der Befehl `box on` erzeugt einen Rahmen (*Box*) um die 3D-Grafik. Weitergehende Einstellmöglichkeiten für Perspektive und Beleuchtung bietet das Figure-Menü *View/Camera Toolbar* und die Befehlsübersicht mit `help graph3d`.

Grafik: 3D Plot-Befehle

`[`*X*, *Y*`] = meshgrid (`*xvektor*, *yvektor*`)`	Koordinatenmatrizen
`plot3 (`*xwerte*, *ywerte*, *zwerte* ... [, *plotstil*]`)`	3D-Plot, Punkte/Linien
`surf (`*xwerte*, *ywerte*, *zwerte* ... [, *farbe*]`)`	3D-Plot, Fläche
`mesh (`*xwerte*, *ywerte*, *zwerte* ... [, *farbe*]`)`	3D-Plot, Gitter
`waterfall (`*xwerte*, *ywerte*, *zwerte* ... [...]`)`	3D-Plot, Wasserfall
`contour (`*xwerte*, *ywerte*, *zwerte* ... [...]`)`	2D-Plot, Höhenlinien
`box [on | off]`	Box einblenden
`rotate3d [on | off]`	Interaktives Drehen
`view (`*horizontal*, *vertikal*`)`	Perspektive ändern
`zlabel (`*string*`)`	Beschriftung der z-Achse
Farben einstellen	
`colormap (`*name*`)`	Wahl der Farbtabelle
`caxis (`*farbe_min*, *farbe_max*`)`	Skalierung der Farbe

Beispiel

Im folgenden Beispiel erzeugt der Befehl `meshgrid` zunächst aus den Vektoren x und y die Matrizen X und Y, deren Zeilen bzw. Spalten den Vektoren x bzw. y entsprechen. Daraus berechnet sich dann die Matrix Z:

```
x      = 0:0.05:2;
y      = -1:0.2:1;
[X, Y] = meshgrid (x, y);        % Erzeugt Matrizen über Bereich von x, y
Z      = (Y+1) .* cos (2*X.^2) + (Y-1) .* sin (2*X.^2) / 5;
```

Die so erhaltenen Daten werden nun mittels verschiedener Befehle und Einstellungen geplottet (siehe Abb. 3.8):

```
subplot (221)
    surf (X, Y, Z)
    view (-40, 30)
    title ('surf (X, Y, Z); view (-40, 30)')
subplot (222)
    mesh (X, Y, Z)
    view (-20, 30)
    title ('mesh (X, Y, Z); view (-20, 30); grid off; box on')
    grid off
    box on
subplot (223)
    waterfall (X, Y, Z)
    view (-20, 10)
    title ('waterfall (X, Y, Z); view (-20, 10)')
subplot (224)
    contour (X, Y, Z)
    title ('contour (X, Y, Z)')
```

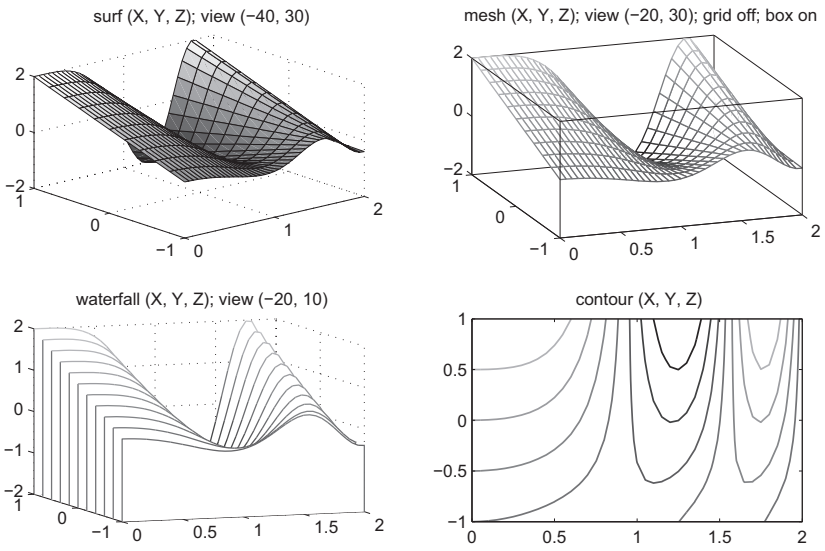

Abb. 3.8: *Beispiel: Verschiedene Darstellungen dreidimensionaler Daten*

Zum **interaktiven Plotten** eignen sich zudem der *Workspace Browser* sowie auch der *Variable Editor*. Für eine ausgewählte Variable können nach Anklicken des Pfeiles in plot( ) geeignete Plotbefehle direkt ausgeführt werden (siehe Abb. 3.9). Wird dagegen der Miniaturplot angeklickt, wendet MATLAB direkt den dort gezeigten (zuletzt verwendeten) Befehl an.

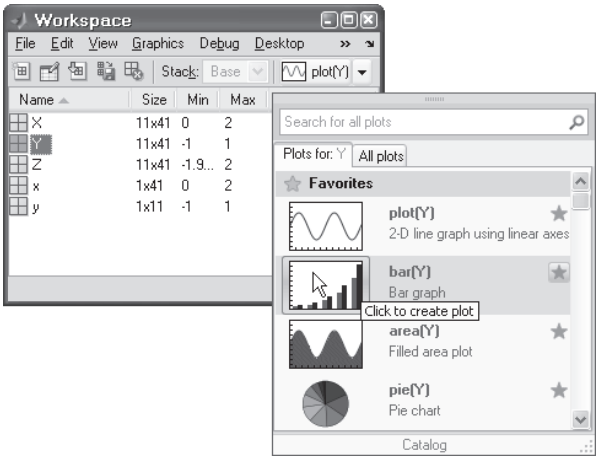

Abb. 3.9: *Interaktives Plotten aus dem* `Workspace Browser`

3.5.6 Importieren, Exportieren und Drucken von Grafiken

Zum Einlesen von Pixel-Grafiken bietet MATLAB den Befehl `imread` an. Mit *variable* `= imread ('`*datei*`', '`*format*`')` wird eine Grafik-Datei mit dem angegebenen Format[8] als Variable eingelesen, wobei *variable* bei Graustufen-Bildern eine zweidimensionale Matrix, bei Farbbildern ein dreidimensionales Array ist.

Der Befehl `image (`*variable*`)` gibt die so eingelesene Grafik in einer Figure aus. So erzeugen die folgenden Befehlszeilen die Figure in Abb. 3.10, die das im jpg-Format vorliegende Bild `foto.jpg` enthält:

```
>> foto_variable = imread ('foto.jpg', 'jpeg');
>> image (foto_variable)
```

Das Drucken und Exportieren von Grafiken aus MATLAB geschieht über den Befehl `print`. Mit `print -f`*nummer* wird die Figure *nummer* auf dem Standard-Drucker ausgegeben. Mit `print -f`*nummer* `-d`*option* `'`*datei*`'` erfolgt die Ausgabe in Dateien verschiedener Grafikformate.[9]

Über das Menü *File/Print Preview* können Lage und Größe der Figure für den Ausdruck sowie weitere Druckoptionen (z.B. Farbraum, Linien, Schrift, etc.) eingestellt werden.

```
>> print -f1;                % Drucken von Figure 1 auf Standard-Drucker
>> print -f1 -dmeta 'bild';  % Speichern als Windows-Metafile (bild.emf)
>> print -f1 -depsc 'bild';  % Speichern als Farb-Postscript  (bild.eps)
```

[8] Mögliche Formate: `'bmp'`, `'ico'`, `'jpg'`/`'jpeg'`, `'pcx'`, `'tif'`/`'tiff'`, siehe auch `doc imread`.
[9] Mögliche Formate: `bmp`, `meta` (emf), `eps`, `jpeg`, `pcx24b` (pcx), `pdf`, `tiff`, siehe auch `doc print`.

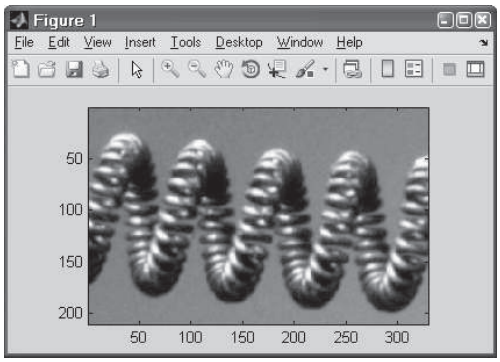

Abb. 3.10: *Mit* `imread` *und* `image` *eingebundene Grafik*

MATLAB bietet auch die Speicherung in einem eigenen Figure-Format an; dieses eignet sich insbesondere für spätere Weiterverarbeitung einer Figure in MATLAB. Gespeichert wird entweder über das Figure-Menü *File/Save As* oder mit dem Befehl `saveas(`*handle*`,'`*datei*`'[,'`*format*`'])`.[10)]

Erweiterte Ausgabeoptionen können zudem über das Menü *File/Export Setup* eingestellt werden.

Grafik: Importieren, Exportieren und Drucken

`print -f`*nummer*	Figure auf Standarddrucker drucken
`print -f`*nummer* `-d`*format* `'`*datei*`'`	Figure in Datei speichern
`saveas(`*handle*`,'`*datei*`','fig')`	Figure als MATLAB-Figure speichern
variable `= imread('`*datei*`','`*format*`')`	Pixel-Grafik in Matrix einlesen
`image(`*variable*`)`	Pixel-Grafik in Figure plotten

3.6 Grafische Benutzeroberfläche (GUI)

Zur komfortablen Bedienung eigener Programme lassen sich in MATLAB grafische Benutzerschnittstellen, so genannte *GUIs*[11)] erstellen. GUIs sind interaktive Figures in Form einer Dialogbox, die auch Text- und Grafikausgabe enthalten können. Als **Beispiel** dient die Erstellung eines **Datums-Rechners** zur Bestimmung des Wochentags für ein beliebiges Datum (siehe Abb. 3.11).

Abb. 3.11: *Beispiel: Datums-Rechner als fertiges GUI*

[10)] Mögliche Formate: `'fig'` (speichert Figure als binäres Fig-File), `'m'` (erzeugt ein Fig-File und ein M-File, das das Fig-File aufruft). Des Weiteren sind auch einige Formate wie bei `print` zulässig.
[11)] GUI = Graphical User Interface, GUIDE = GUI Design Editor