

MTP-Dokumentation

Joachim Gaukel, Markus-Hermann Koch, Joachim Wipper

Version 1.1.2

Stand: 19. Dezember 2002

Vorwort

Für numerischen Berechnungen stellt die Firma MathWorks mit MATLAB [Mat99] ein effizientes Hilfsmittel bereit, das durch eine einfach zu erlernende Hochsprache sowohl interaktive als auch programmgesteuerte Berechnungen ermöglicht. Eine umfangreiche Sammlung an Grafikbefehlen lassen bei der Visualisierung der Daten kaum Wünsche offen. Hierbei entstehen qualitativ hochwertige Grafiken, welche unmittelbar zur Illustration von Texten herangezogen werden können. Dabei ist auf ein einheitliches Erscheinungsbild von Grafik und Text zu achten, was ein enges Zusammenspiel mit dem Textverarbeitungsprogramm voraussetzt. Im mathematisch wissenschaftlichen Umfeld hat sich L^AT_EX [Kop96] als Quasistandard in der Textverarbeitung etabliert.

Ziel des hier vorgestellten MTP-Programmpaketes ist es, die Grafikfähigkeit von MATLAB mit dem Satzsystem L^AT_EX zu verbinden um typographisch hochwertige Dokumente im Postscript-Format zu erstellen. Das Akronym MTP leitet sich dabei aus den Anfangsbuchstaben der involvierten Pakete – MATLAB, T_EX und Postscript – ab.

In MATLAB werden Befehle zur Beschriftung von Grafiken in L^AT_EX-Notation bereitgestellt. Anschließend übernimmt L^AT_EX die Formatierung der Beschriftung. Dabei kann der Benutzer von der hohen Qualität und der umfangreichen Sammlung von Symbolen in L^AT_EX profitieren.

Daneben stellt das MTP-Paket aber auch eine Reihe von Befehlen bereit, die das Erstellen von Grafiken in MATLAB vereinfachen beziehungsweise neue Grafikobjekte (wie zum Beispiel Pfeile) einführen. Jedes graphische Objekt besitzt in MATLAB eine umfangreiche Liste an Attributen, die leider etwas umständlich zu modifizieren sind. Durch ein Aliaskonzept wird dies wesentlich vereinfacht.

In Kapitel 1 werden zunächst die zur Installation des MTP-Paketes benötigten Schritte erläutert. Anschließend wird auf die Initialisierung des MTP-Paketes eingegangen. Kapitel 2 stellt die Grafik-, Kapitel 3 die Textbefehle des MTP-Paketes vor. Sonstige Hilfsfunktionen werden im nachfolgenden Kapitel 4 behandelt. In Kapitel 5 wird erläutert, wie die mit Hilfe des MTP-Paketes erstellten Grafiken in ein L^AT_EX-Dokument eingebunden werden können. Kapitel 6 widmet sich der Fehlerbehandlung. Der Anhang dient lediglich der Dokumentation interner Funktionen und der Festlegung einiger Programmier- und Dokumentationsrichtlinien und enthält somit keinerlei relevante Informationen für die Benutzerinnen und Benutzer des MTP-Paketes.

Einsteiger werden ausdrücklich dazu ermutigt, sofort mit den in den Kapiteln 2 und 3 beschriebenen Funktionen zu experimentieren. Hierbei ist lediglich darauf zu achten, dass in MATLAB die MTP-Funktionen zunächst mit dem Befehl `mtpinit` zu initialisieren sind.

Stuttgart, im September 2000.

Joachim Gaukel
Markus-Hermann Koch
Joachim Wipper

Inhaltsverzeichnis

1	Installation	7
1.1	Hilfsprogramme	7
1.2	Installation des MTP-Pakets	8
2	Grundlagen	9
2.1	Initialisierung der MTP-Funktionen	9
2.1.1	mtpinit	10
2.1.2	mtpinitrc	12
2.2	Das Alias-Konzept	17
3	Grafikfunktionen	20
3.1	Einführung	20
3.2	Befehlsübersicht	20
3.2.1	mtpline	21
3.2.2	mtpsegments	24
3.2.3	mtploop	26
3.2.4	mtparc	28
3.2.5	mtparea	30
3.2.6	mtpmarker	32
3.2.7	mtparrow	35
4	Textfunktionen	39
4.1	Einführung	39
4.2	Befehlsübersicht	39
4.2.1	mtptext	40
4.2.2	mptexaxes	44
4.2.3	mptitle	48
4.2.4	mtpsavetex	51
4.2.5	mtpsaveeps	53
4.2.6	mtpshow	56
5	Sonstige Funktionen	58
5.1	Einführung	58
5.2	Befehlsübersicht	58

5.2.1	<code>mtpclose</code>	59
5.2.2	<code>mtpsearch</code>	60
6	Einbinden von MTP-Grafiken in \LaTeX	62
6.1	Die MTP-Grafiktypen	62
6.1.1	<code>mtpsaveeps</code> -Grafiken	62
6.1.2	<code>mtpsavetex</code> -Grafiken	63
6.2	Grafiken einbinden	63
6.2.1	Das Paket <code>graphicx.sty</code>	64
6.2.2	<code>includegraphics</code>	64
6.2.3	Das Paket <code>mtp.sty</code>	65
6.2.4	<code>mtpincludegraphics</code>	67
7	Problembehandlung	68
7.1	Fehlerhafte EPS-Grafiken	68
7.1.1	Falsche Bounding-Boxen	68
7.1.2	Zusätzliche Linien in der Grafik	68
A	Interna	69
A.1	Interne Funktionen	69
A.1.1	<code>mtpinternalaliases</code>	70
A.1.2	<code>mtpinternalaxalignment</code>	71
A.1.3	<code>mtpinternaldvi2eps</code>	72
A.1.4	<code>mtpinternalget</code>	73
A.1.5	<code>mtpinternalmessage</code>	76
A.1.6	<code>mtpinternalplot</code>	78
A.1.7	<code>mtpinternalsave</code>	79
A.1.8	<code>mtpinternalswitch</code>	80
A.1.9	<code>mtpinternalwritevariable</code>	82
A.2	Die globalen Variablen des MTP-Pakets	83
A.3	Richtlinien für die Programmierung	84
A.4	Richtlinien zur Dokumentation	90

Kapitel 1

Installation

Die Funktionen des MTP-Pakets bilden die Schnittstelle zwischen MATLAB, T_EX und dem PostScript-Format. Sie benötigen eine Reihe von Hilfsprogrammen, die anschließend kurz vorgestellt werden. Die genannten Programme müssen, sofern sie nicht schon zur Verfügung stehen, vor dem Einsatz der MTP-Funktionen installiert werden. Näheres zur Installation ist der jeweiligen Dokumentation zu entnehmen.

1.1 Hilfsprogramme

Matlab

MATLAB dient sowohl zum Erzeugen der Grafiken sowie als Benutzerschnittstelle zum MTP-Paket. Da die MTP-Funktionen Cell- und Struct-Variablen verwenden, wird MATLAB mit einer Versionsnummer ≥ 5.0 vorausgesetzt. Nähere Informationen zu MATLAB sind im Internet unter www.mathworks.com zu finden.

L^AT_EX

Die Beschriftung der Grafiken erfolgt mit Hilfe von L^AT_EX. Dem Benutzer steht hierfür der komplette Funktionsumfang von L^AT_EX zur Verfügung. Benötigt wird L^AT_EX2 ϵ zusammen mit den Paketen `graphicx` und `psfrag`, sowie dem DVI-zu-PS-Konverter `dvips`. Diese sind Bestandteil aktueller Distributionen wie teTeX für Unix-Systeme oder miktex für Windows-Systeme. Die genannten Pakete und Distributionen können über das Comprehensive TeX Archive Network (CTAN) bezogen werden, das zum Beispiel von www.dante.de, dem Server der Deutschsprachigen Anwendervereinigung T_EX e.V., gespiegelt wird. Auf diesem Server ist auch eine sehr lesenswerte FAQ zu T_EX und L^AT_EX zu finden.

ghostscript/ghostview

Zur Erzeugung und Anzeige der beschrifteten Grafiken im PostScript-Format wird der PostScript-Interpreter `ghostscript` (`gs`) und ein darauf basierendes Anzeigeprogramm für PostScript-Dateien, wie etwa `ghostview`, `gv`, etc., benötigt. Diese Programme werden mit den meisten Unix-Systemen mitgeliefert, können aber auch unter <http://www.cs.wisc.edu/~ghost/> bezogen werden. Unter der genannten Adresse stehen auch Versionen für andere Systeme, wie zum Beispiel Windows, zur Verfügung.

1.2 Installation des MTP-Pakets

Das MTP-Paket wird als Zip-Datei mit dem Namen `mtp-rel#.zip` geliefert. Hierbei steht `#` für die aktuelle Versionsnummer des Pakets. Dieses Archiv kann mit `unzip` (Unix) oder `winzip`, etc. (Windows) entpackt werden und enthält das Hauptverzeichnis `mtp`. Darin enthalten sind die Unterverzeichnisse

matlab: enthält die MATLAB-Funktionen des MTP-Pakets,

latex: enthält die Paket-Datei `mtp.sty` und

doc: enthält die Dokumentation des MTP-Pakets im PostScript-Format.

Damit die MTP-Funktionen unter MATLAB verfügbar sind, muss das Verzeichnis `mtp/matlab` in den MATLAB-Pfad aufgenommen werden (siehe MATLAB-Dokumentation). Ebenso muss die Datei `mtp.sty` in den L^AT_EX-Suchpfad kopiert, bzw. der Suchpfad um das Verzeichnis `mtp/latex` erweitert werden.

Sofern in MATLAB bei Eingabe von `mtpinit`; `mptexaxes`; `mtpshow` keine Fehler gemeldet werden und ein durch L^AT_EX beschriftetes Koordinatensystem angezeigt wird, war die Installation erfolgreich. Wenn die zuvor genannten Hilfsprogramme installiert sind, jedoch von MTP nicht gefunden werden, kann mit Hilfe der Funktion `mtpinitrc` (siehe Seite 12) das MTP-Paket an das System angepasst werden.

Kapitel 2

Grundlagen

2.1 Initialisierung der MTP-Funktionen

Die Funktionen des MTP-Pakets müssen mit Hilfe des Befehls `mtpinit` initialisiert werden. Dies ist zugleich die entscheidende Information dieses Kapitels für jene Leserinnen und Leser, die sich einen schnellen Überblick über die Funktionalität des MTP-Pakets verschaffen wollen. Sie seien direkt auf die folgenden Kapitel verwiesen, in denen die einzelnen Funktionen des MTP-Pakets beschrieben sind. Der Rest dieses Kapitels ist erst dann von Interesse, wenn die Voreinstellungen des MTP-Pakets (wie z.B. Linienbreite, Markergröße, etc.) verändern werden sollen. Die Voreinstellungen des MTP-Pakets werden in der Funktion `mtpinit` festgelegt. Der Benutzer hat darüber hinaus die Möglichkeit diese Voreinstellungen mit Hilfe der Konfigurationsfunktion `mtpinitrc` an seine Wünsche anzupassen. Die Initialisierung erfolgt in drei Stufen:

1. `mtpinit` setzt die im zugehörigen Hilfetext genannten Defaultwerte.
2. Anschließend wird geprüft, ob die Funktion `mtpinitrc` existiert und, sofern vorhanden, ausgeführt. Dabei überschreiben die vom Benutzer in dieser Funktion festgelegten Defaultwerte die von `mtpinit` zuvor gesetzten Voreinstellungen.
3. Schließlich werden die Parameter der Funktion `mtpinit` ausgewertet und gesetzt, sofern sie nicht NaN sind. Die Defaultwerte also ggf. wieder modifiziert.

Die Funktionen `mtpinit` und `mtpinitrc` werden nun im Einzelnen erläutert.

2.1.1 mtpinit

Initialisieren des MTP-Pakets.

Syntax:

```
mtpinit
mtpinit(mtptextdistanceunit)
mtpinit(mtptextdistanceunit,LineWidth)
mtpinit(mtptextdistanceunit,LineWidth,MarkerSize)
mtpinit(mtptextdistanceunit,LineWidth,MarkerSize,ArrowShape)
```

Beschreibung: `mtpinit` initialisiert das MTP-Paket und setzt den Status `hold on`. Sofern die Funktion `mtpinitrc` existiert, überschreibt diese die von `mtpinit` gesetzten Werte. Die Parameter von `mtpinit` haben jedoch die höchste Priorität (siehe auch das auf Seite 9 beschriebene Initialisierungskonzept).

Parameter die nicht modifiziert werden sollen, können mit `NaN` übersprungen werden.

`mtpinit` initialisiert das MTP-Paket. Diese Funktion muss vor dem ersten Aufruf einer anderen MTP-Funktion ausgeführt werden. Alle `MATLAB`-Grafikfunktionen können weiterhin unabhängig vom MTP-Paket verwendet werden.

`mtpinit` setzt die globale Variable `'MTP'`, die vom Benutzer nicht modifiziert werden darf.

`mtpinit(mtptextdistanceunit)` initialisiert das MTP-Paket mit einer `TEX`-Einheit für den Abstand vom Text zu seinem Referenzpunkt, falls beim Aufruf von `mtptext` oder `mtptitle` ein solcher gewünscht wird. Default ist der String `'0.3cm'`.

`mtpinit(mtptextdistanceunit,LineWidth)` initialisiert das MTP-Paket zusätzlich mit einer gewählten Standardbreite für Linien. Default ist 0.5. Er kann durch eine beliebige positive Zahl ersetzt werden.

`mtpinit(mtptextdistanceunit,LineWidth,MarkerSize)` initialisiert das MTP-Paket zusätzlich mit einer gewählten Standardgröße für die MTP-Marker. Default ist 6. Er kann durch eine beliebige positive Zahl ersetzt werden.

`mtpinit(mtptextdistanceunit,LineWidth,MarkerSize,ArrowShape)` ermöglicht durch `ArrowShape=[ArrowLength, ArrowHeadAngle]` die Modifikation der Pfeilspitzen. `ArrowLength` ist hierbei die Länge und `ArrowHeadAngle` der Öffnungswinkel. Default ist `[16 40]`.

Beispiel: Abbildung 2.1 zeigt links einen Plot unter Verwendung der Voreinstellungen. Er wurde erzeugt durch

```
>> mtpinit
>> mtpmarker(1,1)
>> mtparrow([0 2],[0 2])
```

Für den rechten Teil der Abbildung wurden die Defaults wie folgt modifiziert:

```
>> mtpinit(NaN,8,80,[170,10])
>> mtpmarker(1,1)
>> mtparrow([0 2],[0 2])
```

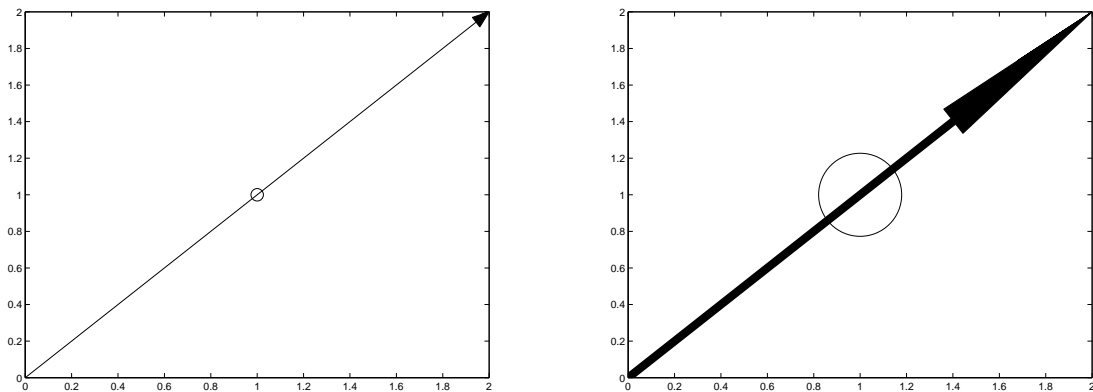


Abbildung 2.1: Rechts mit, links ohne veränderte Defaults bei `mtpinit`

Siehe auch: Abschnitt 2.1, `mtparrow`, `mtpclose`, `mtpinitrc`, `mtpline`, `mtpmarker`, `mtpstext`.

2.1.2 mtpinitrc

Konfigurationsdatei des MTP-Pakets.

Syntax:

```
mtpinitrc
```

Beschreibung: `mtpinitrc` ermöglicht es dem Benutzer die Standardeinstellungen des MTP-Pakets dauerhaft an die eigenen Wünsche und an die externen Programme anzupassen. Hierzu sind lediglich die in `mtpinitrc` festgelegten Defaultwerte zu modifizieren und die Kommentarzeichen „%“ vor den entsprechenden Zeilen zu löschen. Anschließend muss die Funktion in ein Verzeichnis im MATLAB-Pfad kopiert werden. Die Verwendung dieser Funktion ist optional. Sofern der Benutzer keine eigene `mtpinitrc`-Funktion bereitstellt, werden die im Hilfetext von `mtpinit` genannten Defaultwerte verwendet.

Im MATLAB-Verzeichnis der MTP-Funktionen findet sich das folgende Muster einer `mtpinitrc`-Funktion unter dem Namen `mtpinitrc_template.m`:

```

1  function mtpinitrc(mtptextdistanceunit,LineWidth,MarkerSize,ArrowShape)
2  % MTPINITRC  resource file of the mtp package.
3  %
4  % Function to redefine mtp default values and system dependent
5  % configuration of the mtp package.
6  % Default values of this file will overwrite the mtpinit default
7  % values, but will be overwritten by the parameters of mtpinit.
8  % See documentation for further details.
9  %
10 % See also MTPINIT.
11
12 % NOTE: Rename this file to mtpinitrc.m and place it in your Matlab path.
13
14 % Author: Joachim Gaukel (gaukel@mathematik.uni-stuttgart.de)
15 % Modified by: Joachim Wipper (wipper@mathematik.uni-stuttgart.de)
16 %              Markus-Hermann Koch (kochmn@cip.mathematik.uni-stuttgart.de)
17 % RCS-Version: $Id: mtpinitrc_template.m,v 1.22 2002/06/04 13:17:28 wipperjm Exp $
18
19 % Global variables which might be modified.
20 global MTPtemp
21
22 %% Comment-lines are preceeded by '%%', default defining lines by '%'.
23 %% Please remove the comment character of the default you want to change.
24
25 %> default values -----
26 %>> Define paramter default values for input arguments of mtpinit-----
27 %% mtptextdistanceunit: distance scale of MTPTEXT and MPTITLE
28 %%                      (LaTeX measure string)
29 %MTPtemp.NORMALSIZE.mtptextdistanceunit='0.3cm';

```

```

30
31 %% linewidth: default line width of MTPLINE, MTPLOOP and MTPSEGMENTS
32 %%             (number>0)
33 %MTPtemp.NORMALSIZE.linewidth=.5;
34
35 %% marker: default marker size of MTPMARKER (number>0)
36 %MTPtemp.NORMALSIZE.markersize=6;
37
38 %% arrow.headangle: default head angle of MTPARROW (number in [0,180])
39 %MTPtemp.NORMALSIZE.arrow.headangle=40;
40
41 %% arrow.length: default arrow tip length of MTPARROW (number>0)
42 %MTPtemp.NORMALSIZE.arrow.length=16;
43 %%<< -----
44
45 %%>> Define paramter default values for non input arguments of mtpinit--
46 %% ticklabelnulltolerance: Default tolerance factor for zero on the axis:
47 %%   mtpshow rounds #TickLabel{j} to zero if it fulfills
48 %%   #TickLabel{j}<(max(#TickLabel{j})-min(#TickLabel{j}))*ticklabelnulltol
49 %MTPtemp.NORMALSIZE.ticklabelnulltolerance=1e-6;
50
51 %% previewfontsize: default text preview fontsize of MTP text functions
52 %%             (number>0)
53 %MTPtemp.NORMALSIZE.previewfontsize=10;
54
55 %%<< -----
56 %%< -----
57
58
59 %%> Preamble for LaTeX document generated by MTPSAVEEPS -----
60 %% Define default document paperformat (a4paper, etc.),
61 %% fontsize (10, 11 or 12) and class (article, etc.) for mtpsaveeps.
62 %MTPtemp.LATEX.paperformat = 'a4paper';
63 %MTPtemp.LATEX.documentclass='article';
64 %MTPtemp.LATEX.documentfontsize=10;
65
66 %% Define default document header for mtpsaveeps.
67 %MTPtemp.LATEX.preamblestring=[
68 % '\usepackage{amsmath}' ...
69 % '\usepackage{amssymb}' ...
70 % '\usepackage[german]{babel}' ...
71 % '\usepackage[latin1]{inputenc}' ...
72 % '\usepackage{mtp}' ...
73 % '\pagestyle{empty}' ...
74 %];
75 %%< -----
76
77 %%> extern program calls -----
78 %% Some mtp functions need extern programs like latex, dvips and so on.
79 %% The following variables allow customization which programs should be
80 %% used and how the system call should look like (%s will be replaced
81 %% by a filename).

```

```

82
83 %% latex:
84 %MTPtemp.EXTERNCALL.latex='latex %s';
85
86 %% ps tools: dvips and ghostscript interpreter (without %s!)
87 %MTPtemp.EXTERNCALL.dvips='dvips'; % windows and unix systems
88 %MTPtemp.EXTERNCALL.gs='gs'; % unix systems
89 %MTPtemp.EXTERNCALL.gs='gswin32c'; % windows systems
90
91 %% psview: program to view postscript files like gv, ghostscript, etc.
92 %MTPtemp.EXTERNCALL.psview='gv %s';
93 %%< -----
94
95
96 %%> Event program calls -----
97 %% The user can give any matlab-command(s), which will be executed
98 %% if a warning or an error occurs.
99
100 %% Install beeping for error and warning (use '' to avoid this noise).
101 % MTPtemp.EVENT.error='fprintf('\a')';
102 % MTPtemp.EVENT.warning='fprintf('\a')';
103 %%< -----
104
105
106 %%> EPS generation -----
107 %% matlab eps print device: 'eps', 'epsc', 'eps2' or 'epsc2' (default)
108 %% Use 'eps' if there are unwanted lines in your plot
109 % MTPtemp.EPS.printdevice='epsc2';
110
111 %% program/algorithm to convert mtp-graphics to standalone eps-files:
112 %% 'gs', 'ps2epsi' or 'gsbb' (default)
113 %% Change if mtp fails to generate the correct bounding box.
114 % MTPtemp.EPS.converter='gsbb';
115 %%< -----
116
117
118 %%> Set input-variables of mtpinit if not NaN -----
119 %% These lines must not be altered, deleted or commented out!
120 if (~isnumeric(mtptextdistanceunit) | ~isnan(mtptextdistanceunit))
121     MTPtemp.NORMALSIZE.mtptextdistanceunit=mtptextdistanceunit;
122 end;
123 if ~isnan(LineWidth)
124     MTPtemp.NORMALSIZE.linewidth=LineWidth;
125 end;
126 if ~isnan(MarkerSize)
127     MTPtemp.NORMALSIZE.markersize=MarkerSize;
128 end;
129 if ~isnan(ArrowShape(1))
130     MTPtemp.NORMALSIZE.arrow.length=ArrowShape(1);
131 end;
132 if ~isnan(ArrowShape(2))
133     MTPtemp.NORMALSIZE.arrow.headangle=ArrowShape(2);

```

```

134 end;
135 %%< -----
136
137
138 %%> user-defined aliases -----
139 %% The following nx2-cells contain the user defined aliases.
140 %% A new alias is defined by the line:
141 %%      'Aliasname' { DEFS }
142 %% Where DEFS may be a list of previously defined aliases or/and
143 %% cells of propertyname/propertyvalue pairs. Example for mtparea:
144 %% 'BlueRed' { 'blue'                {'EdgeColor' [1 0 0]} }; or
145 %% 'BlueRed' { {'FaceColor' [0 0 1]} {'EdgeColor' [1 0 0]} };
146 %%
147 %% See mtp documentation for further details on how to define your own aliases.
148
149 %% MTPLINE/MTPLOOP aliases:
150 %MTPtemp.ALIASLIST.line={'Red'  {'Color' [1 0 0]}};
151 %      'Blue' {'Color' [0 0 1]}};
152 % };
153
154 %% MTPAREA aliases:
155 %MTPtemp.ALIASLIST.area={'Red'  {'EdgeColor' [1 0 0]}{'FaceColor' [1 0 0]}};
156 %      'Blue' {'EdgeColor' [0 0 1]}{'FaceColor' [0 0 1]}};
157 % };
158
159 %% MTPMARKER aliases:
160 %MTPtemp.ALIASLIST.marker={'Red'  {'MarkerEdgeColor' [1 0 0]}...
161 %      {'MarkerFaceColor' [1 1 1]}};
162 %      'Blue' {'MarkerEdgeColor' [0 0 1]}...
163 %      {'MarkerFaceColor' [1 1 1]}};
164 % };
165
166 %% MTPARROW aliases:
167 %MTPtemp.ALIASLIST.arrow={'Red'  {'HeadFaceColor' [1 0 0]}...
168 %      {'HeadEdgeColor' [1 0 0]}...
169 %      {'TailColor' [1 0 0]}};
170 %      'Blue' {'HeadFaceColor' [0 0 1]}...
171 %      {'HeadEdgeColor' [0 0 1]}...
172 %      {'TailColor' [0 0 1]}};
173 % };
174 %%< -----

```

`mtpinitrc` gliedert sich in die fünf für den Benutzer interessanten Abschnitte:

Defaultwerte: (Zeilen 25–56) Hier können die Defaultwerte von `mtpinit` dauerhaft verändert werden.

L^AT_EX-Umgebung: (Zeilen 59–75) Ermöglicht die Modifikation der L^AT_EX-Umgebung für `mtpsaveeps`. Benötigt werden Papierformat, Fontgröße

und eine Dokumentklasse (z.B. 'a4pa per', 12 und 'article'). Sowie im `preamblestring` das Paket `mtp`.

Externe Programme: (Zeilen 77–93) Hier erfolgt die Anpassung der externen Funktionen. Benötigt werden \LaTeX , der DVI-zu-PS-Konverter `dvips`, ein Anzeigeprogramm für Postscriptdateien (z.B. `gv`) und `ghostscript` (`gs`).

Befehle im Fall eines Fehlers oder einer Warnung: (Zeilen 96–103) Hier kann der Benutzer eine Sequenz von Befehlen angeben, die im Fall eines Fehlers bzw. einer Warnung ausgeführt werden sollen. Die vorgeschlagenen Befehle veranlassen die akustische Ausgabe eines Signaltons.

EPS-Generierung: (Zeilen 106–115) Ermöglicht die Konfiguration der Funktionen zur Generierung von EPS-Grafiken.

Benutzeraliase: (Zeilen 138–174) An dieser Stelle kann der Benutzer insbesondere für die Grafikfunktionen des MTP-Pakets eigene Aliase zur Formatierung der Grafikobjekte definieren. Die Aliase sind den einzelnen Grafikfunktionen zuzuordnen.

Die Zeilen 118-135 dürfen vom Benutzer auf keinen Fall modifiziert werden.

Siehe auch: Initialisierungskonzept (Abschnitt 2.1), Aliaskonzept (Abschnitt 2.2), `mtpinit`.

2.2 Neue Grafikoptionen definieren. Das Alias-Konzept

Ein Alias ist normalerweise ein String, der die Modifikation der Attribute eines Grafikobjekts erleichtern soll.

In MATLAB kann z.B. eine rot gefüllte und schwarz gestrichelt berandete Fläche wie folgt erzeugt werden:

```
>> H=fill([0 1 2],[2 0 1],'r');
>> set(H,'LineStyle','--','EdgeColor','k');
```

Wie man sieht, bekommt man zwar genau das gewünschte, jedoch hätte man vermutlich nicht erwartet, sich so umständlich ausdrücken zu müssen. Um das Grafikpaket von MATLAB wirklich voll ausnutzen zu können, ist es notwendig sich mit den MATLAB-Befehlen `get` und `set` vertraut zu machen. Hier ein Beispiel:

```
>> handle = plot(0:1,0:1);
```

Welchen Zustand hat das Grafikobjekt?

```
>> get(handle)
    Color = [0 0 1]
    EraseMode = normal
    ...
```

Welche Attribute können für dieses Objekt gesetzt werden?

```
>> set(handle)
    Color
    EraseMode: [ {normal} | background | xor | none ]
    ...
```

Veränderung der Linienbreite

```
>> set(handle,'LineWidth',15);
```

Vielleicht ist der Wunsch einer rot gefüllten und schwarz gestrichelt berandeten Fläche kein einmaliger, sondern es soll immer wieder ein solches Gebilde erzeugt werden.

Das Alias-Konzept ermöglicht dem Benutzer eine Menge von Objekteigenschaften zusammenzufassen und unter einem Namen (Alias) anzusprechen. Der Benutzer muss lediglich ein File entsprechend modifizieren, um in Zukunft das gewünschte in obigem Beispiel z.B. mit `mtparea(X,Y,'FredEblackDashed')` oder vielleicht etwas klarer mit `mtparea(X,Y,'Fred','Eblack','Dashed')`. Dabei steht 'F' für „Face“ bzw. 'E' für „Edge“.

Was ist zu tun? Man modifiziere die Datei `mtpinitrc.m` bzw. falls diese noch nicht angelegt wurde, benenne man zuerst die Datei `mtpinitrc_template.m`

in `mtpinitrc.m` um. Außerdem ist dafür zu sorgen, dass sich `mtpinitrc.m` im MATLAB-Pfad befindet. Siehe dazu auch Abschnitt 2.1.2. Als Vorbild dient einfach die Datei `mtpinternalaliases.m`, welche mit `>> type mtpinternalaliases` angezeigt wird. Wichtig ist dabei die Übereinstimmung der Feldnamen. `mtpinitrc.m` sieht dann wie folgt aus.

```
function mtpinitrc
...

MTP_ALIASLIST.area={
    'Fred'           {{'FaceColor' [1 0 0]}};
    'Eblack'         {{'EdgeColor' [0 0 0]}};
    'Dashed'         {{'LineStyle' '--'}};
    'FredEblackDashed' {'Fred' 'Eblack' 'Dashed'};
    'FredEgreen'     {'Fred' {'EdgeColor' [0 1 0]}};
};

MTP_ALIASLIST.marker={
    'Standard'       {'diamond' 'red'};
};
```

Damit können nach dem Aufruf von `mtpinit` die Befehle

```
>> mtparea(X,Y,'FredEblackDashed')
>> mtparea(X,Y,'Fred','Eblack','Dashed')
>> mtpmarker(X,Y,'Standard');
```

abgesetzt werden. Zu beachten ist unbedingt, dass folgende Zeichen in Aliasnamen bei der Definition nicht vorkommen dürfen: '0', '1'... , '9', '+', '-', '.', '.'. Folgende Zeichen dürfen nur eingeschränkt vorkommen: 'e', '!'. D.h. 'e' darf nicht zwischen zwei '#' (welche als Platzhalter für Zahlen stehen) auftreten, denn dann ist keine klare Unterscheidung zwischen zwei Zahlen und einer einzigen Zahl z.B. in 'bsp1.4e1' mehr möglich. '!' ist lediglich als erstes Zeichen verboten, weil ein '!' anzeigt, dass `eval` aufgerufen werden soll.

Hier die Regeln für die Definition von Aliasen.

Alias:

```
{ Aliasname Aliasdefinition }
```

Aliasname:

String, wobei '#' als Platzhalter fuer eine Zahl steht

Aliasdefinition:

```
{ Alias-Liste_opt Attributpaar-Liste_opt }
```

Alias-Liste:

Aliasname

Alias-Liste Aliasname

Attributpaar-Liste:

{ Propertyname Propertyvalue }

Attributpaar-Liste { Propertyname Propertyvalue }

Propertyvalue:

String, wobei '#1', '#2' usw. als Platzhalter fuer die erste Zahl, zweite Zahl usw. aus dem Aliasnamen stehen. Wenn '!' das erste Zeichen ist, so wird eval mit Propertyvalue(2:end) aufgerufen.

Beliebiger anderer Datentyp, der im Zusammenhang mit dem Propertyname sinnvoll ist.

Siehe auch Abschnitt A.1.4.

Kapitel 3

Grafikfunktionen

3.1 Einführung

Die Funktionen `mtpline`, `mtpsegments`, `mtploop`, `mtparea`, `mtpmarker`, `mtparrow` stellen eine komfortable Schnittstelle zur Erzeugung von MATLAB-Grafiken zur Verfügung. `mtpline` plottet einen Polygonzug und ist damit sehr ähnlich wie `plot`. `mtpsegments` plottet Liniensegmente, eignet sich also z.B. zum Plot von Verbindungsgeraden von Punktpaaren. `mtploop` plottet einen geschlossenen Polygonzug und ist ansonsten identisch mit `mtpline`. `mtparea` füllt eine Fläche mit einer anzugebenden Farbe aus. `mtpmarker` setzt Marker wie Kreise, Dreiecke oder Sterne. `mtparrow` zeichnet Pfeile.

Für alle Routinen wurde eine Reihe von nützlichen Aliasen vordefiniert. Mit Hilfe von `mtpsearch` kann sich der Benutzer leicht einen Überblick über diese verschaffen.

Ein umfassendes Alias- Konzept macht es dem Benutzer zusätzlich möglich, weitere Aliase für sich selbst zu definieren, welche dann nicht mehr von den vordefinierten zu unterscheiden sind. Außerdem lassen sich, falls gewünscht, mit diesem Konzept mehrere Aliase gleichzeitig über einen einzigen Alias ansprechen. Siehe dazu Abschnitt 2.2

3.2 Befehlsübersicht

3.2.1 mtpline

Zeichnen eines Polygonzuges, ähnlich wie `plot`.

Syntax:

```
handle=mtpline(K      [,Alias1,Alias2,...])
handle=mtpline(X,Y   [,Alias1,Alias2,...])
handle=mtpline(X,Y,Z [,Alias1,Alias2,...])
```

Beschreibung:

`handle=mtpline(K [,Alias1,Alias2,...])` erzeugt einen Polygonzug. Die Ecken des Polygonzugs sind, falls `K` einzeilig ist, $(i, K(1, i))$, zweizeilig ist, $(K(1, i), K(2, i))$, dreizeilig ist, $(K(1, i), K(2, i), K(3, i))$.

`handle=mtpline(X, Y [,Alias1,Alias2,...])` erzeugt einen Polygonzug. Die Ecken des Polygonzugs sind $(X(1, i), Y(1, i))$.

`handle=mtpline(X, Y, Z [,Alias1,Alias2,...])` erzeugt einen Polygonzug. Die Ecken des Polygonzugs sind $(X(1, i), Y(1, i), Z(1, i))$.

Optional können beliebig viele Aliase zur Formatierung angehängt werden. Die Reihenfolge und die Anzahl der optionalen Aliase spielt keine Rolle, sie werden der Reihe nach von links nach rechts abgearbeitet. Vordefinierte Aliase sind für

die Farbe: `'black'`, `'red'`, `'green'`, `'blue'`, `'yellow'`, `'magenta'`, `'cyan'`, `'white'`, `'gray#'`, `'#'` steht dabei für eine beliebige Zahl zwischen 0 und 100.

den Linientyp: `'solid'`, `'dashed'`, `'dotted'`, `'dotdashed'`,

die Größe: `'normalthick'`, `'thick'`, `'Thick'`, `'THick'`, `'THICK'`, `'THICK'`, `'THICK'`,

Dabei sind die erstgenannten jeweils der Default.

Mit `mtpsearch('mtpline')` kann eine um die vom Benutzer erweiterte Liste aller Aliase angesehen werden.

Außerdem lassen sich alle Optionen setzen, die mit Hilfe von `set` gesetzt werden könnten. Dies wird erreicht durch die Übergabe einer Cell der Form

```
{Propertyname Propertyvalue}
```

in der Aliasliste, z.B. `mtpline(1:2,1:2,{'linestyle' '--'})`.

Ein Rückgabeparameter wird von `mtpline` nur dann geliefert, sofern er von der aufrufenden Funktion gefordert wird und enthält dann das MATLAB-Handle des erzeugten Grafikobjekts. Dieses kann z.B. mit `get` und `set` bearbeitet werden.

Das hier gesagte gilt, abgesehen von den Spezialitäten für die Linie, ausnahmslos auch für die Grafikfunktionen `mtploop`, `mtparea`, `mtpmarker` und `mtparrow` und wird deshalb nicht mehr gesondert in der Beschreibung dieser erwähnt.

Beispiel: In Abbildung 3.1 (oben) ist eine Übersicht über die verschiedenen Optionen zu sehen. Die Grafik wurde durch die folgenden Befehle erzeugt.

```
>> mtpinit
>> axis([-1 6 0 5])
>> for k=1:4
>>   mtpline(0:1,k*[1 1],['gray' num2str((k-1)*25)],'THICK')
>> end;
>> Style={'solid' 'dashed' 'dotted' 'dotdashed'};
>> for k=1:length(Style)
>>   mtpline(2:3,k*[1 1],Style{k})
>> end;
>> Width={'normalthick' 'Thick' 'THICK' 'THICK'};
>> for k=1:length(Width)
>>   mtpline(4:5,k*[1 1],Width{k})
>> end;
```

Abbildung 3.1 (unten) zeigt ein typisches Beispiel für die Verwendung von `mtpline`. Die Grafik wurde durch die folgenden Befehle erzeugt.

```
>> mtpinit
>> view(3)
>> axis([-0.5 1.5 -0.5 1.5 -0.5 1.5])
>> P=[1 1 1 0
>>    1 1 0 1
>>    1 0 0 0];
>> mtpline(P(:,[1 3 4 1]));
```

Siehe auch: `axis`, `get`, `mtparea`, `mtparrow`, `mtpinit`, `mtploop`, `mtpmarker`, `mtpsegments`, `set`

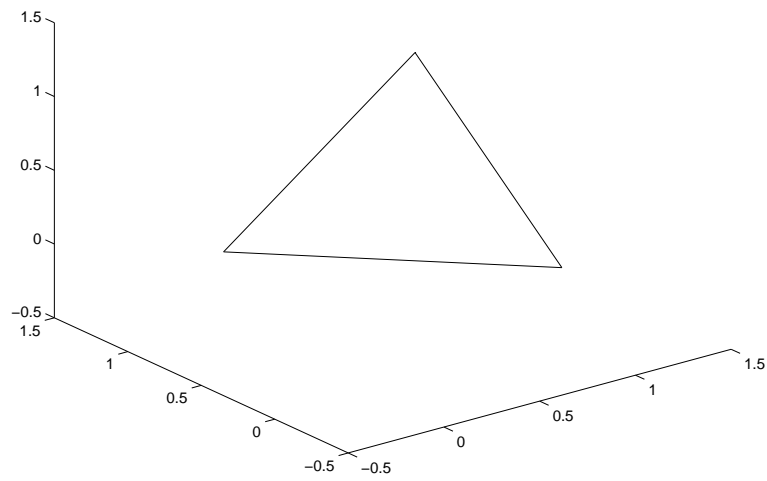
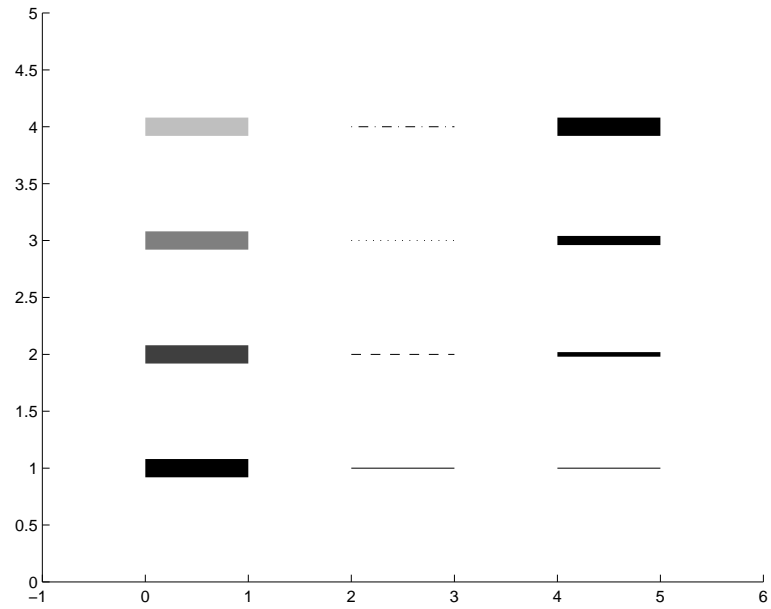


Abbildung 3.1: Mit `mtpline` erzeugte Linien

3.2.2 mtpsegments

Zeichnen von nicht zusammenhängenden Geradensegmenten.

Syntax:

```
handle=mtpsegment(K1,K2           [,Alias1,Alias2,...])
handle=mtpsegment(X1,Y1,X2,Y2     [,Alias1,Alias2,...])
handle=mtpsegment(X1,Y1,Z1,X2,Y2,Z2 [,Alias1,Alias2,...])
```

Beschreibung: Zeichnet gerade Linien zwischen dem *i*-ten Punkt des ersten Datenblocks und dem *i*-ten Punkt des zweiten Datenblocks. `mtpsegments` verwendet dieselben Aliase wie `mtpline`.

Beispiel: In Abbildung 3.3 sind Beispiele für die Verwendung von `mtpsegments` zu sehen. Oben wird ein kubisches Polynom approximiert durch ein quadratisches. Die euklidischen Abstände sind mit `mtpsegments` eingezeichnet. Die Grafik wurde mit den folgenden Befehlen erzeugt:

```
>> mtpinit
>> % Erzeugung der Vergleichsdaten
>> X=linspace(0,1);
>> P=[-3 4 -1 1];
>> mtpline(X,polyval(P,X),'solid')
>> n=length(P);
>> T=linspace(0,1,n);
>> M=vander(T);
>> PA=M(:,2:end)\polyval(P,T)';
>> mtpline(X,polyval(PA,X),'dotted')
>> % Die Vergleichsdaten sind erzeugt
>> mtpsegments([T;polyval(P,T)],[T;polyval(PA,T)],'dashed')
```

Abbildung 3.2 (unten) zeigt ein weiteres typisches Beispiel für die Verwendung von `mtpsegments`. Die Grafik wurde aufbauend auf das zweite Beispiel in der Beschreibung von `mtpline` erzeugt durch:

```
>> mtpsegments(P(:,[2 2 2]),P(:,[1 3 4]),'dashed');
```

Siehe auch: `mtpinit`, `mtpline`

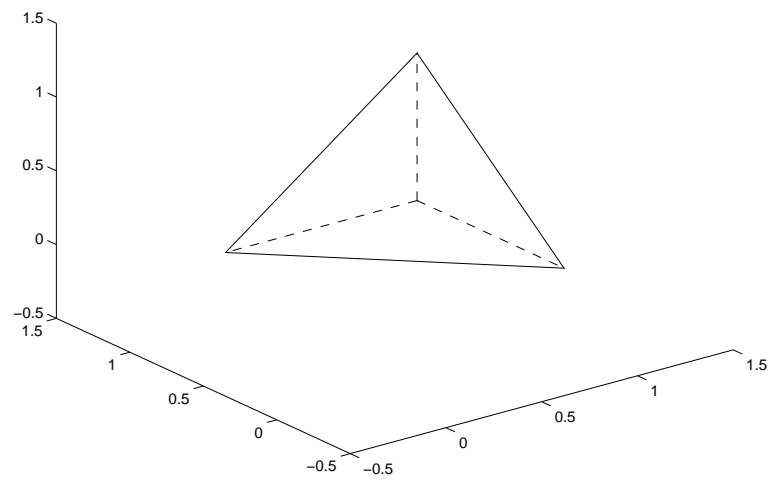
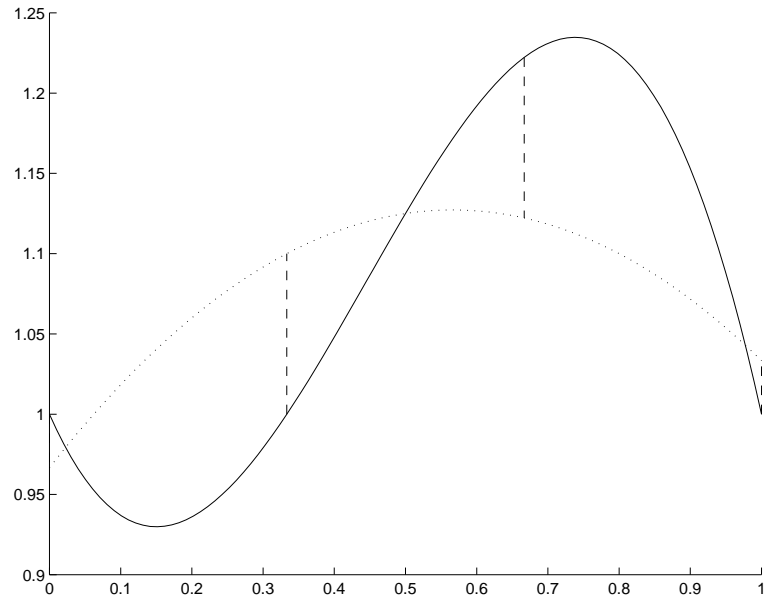


Abbildung 3.2: Mit `mtpsegments` erzeugte Liniensegmente

3.2.3 mtploop

Zeichnen eines geschlossenen Polygonzuges.

Syntax:

```
handle=mtploop(K      [,Alias1,Alias2,...])
handle=mtploop(X,Y   [,Alias1,Alias2,...])
handle=mtploop(X,Y,Z [,Alias1,Alias2,...])
```

Beschreibung: Zeichnet einen geschlossenen Polygonzug. Der einzige Unterschied zu `mtpline` besteht also darin, dass eine den Polygonzug schließende Kante eingezogen wird. Für `mtploop` gibt es keine eigenen Aliase. `mtploop` verwendet dieselben Aliase wie `mtpline`.

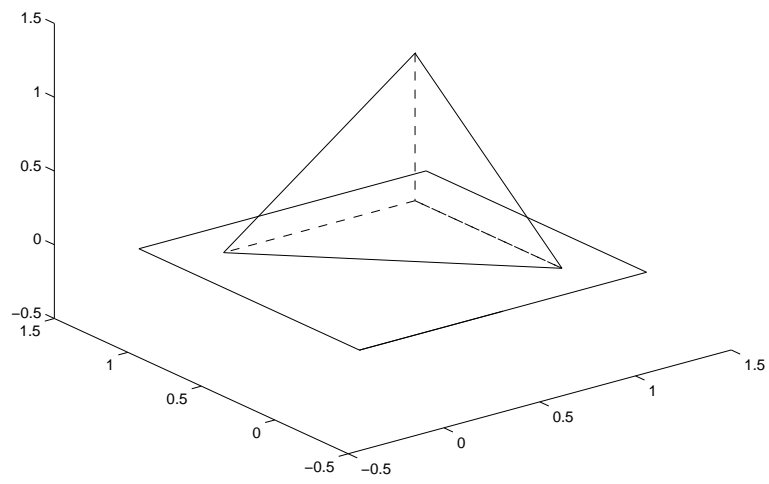
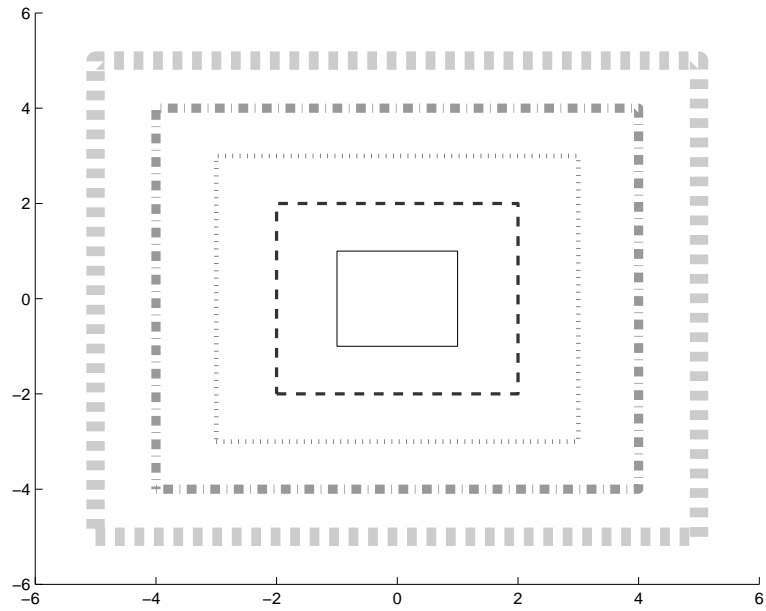
Beispiel: In Abbildung 3.3 (oben) sind Loops mit verschiedenen Optionen zu sehen. Die Grafik wurde durch die folgenden Befehle erzeugt.

```
>> mtpinit
>> axis([-6 6 -6 6]);
>> P=[-1 1 1 -1
>>    -1 -1 1 1];
>> Width={'normalthick' 'thick' 'Thick' 'THIck' 'THICK' };
>> Style={'solid' 'dashed' 'dotted' 'dotdashed' 'dashed' };
>> Color={'gray00' 'gray20' 'gray40' 'gray60' 'gray80'};
>> for k=1:5
>>    mtploop(k*P,Width{k},Style{k},Color{k});
>> end;
```

Abbildung 3.3 (unten) zeigt ein typisches Beispiel für die Verwendung von `mtploop`. Die Grafik wurde aufbauend auf das Beispiel in der Beschreibung von `mtpsegments` erzeugt durch:

```
>> F=[-0.25 1.25 1.25 -0.25
>>     -0.25 -0.25 1.25 1.25
>>     0 0 0 0 ];
>> mtploop(F);
```

Siehe auch: `mtpinit`, `mtpline`

Abbildung 3.3: Mit `mtploop` erzeugte geschlossene Polygonzüge

3.2.4 mtparc

Kreissegmente und beschriftete Winkel zeichnen.

Syntax:

```
text_coord=MTPARC(Centre,rad1,rad2,radius [,Alias1,Alias2,...])
text_coord=MTPARC(Centre,rad1,rad2,radius,text_pos)
text_coord=MTPARC(Centre,rad1,rad2,radius,text_pos, ...
                  TextString [,Alias1,Alias2,...])

[text_coord,handle]=MTPARC(...)
```

Beschreibung:

`text_coord=MTPARC(Centre,rad1,rad2,radius [,Alias1,Alias2,...])` zeichnet vom Kreis mit Mittelpunkt `Centre=[x y]` und Radius `radius` das mathematisch positiv orientierte Randsegment zwischen den in Bogenmaß gegebenen Winkel `rad1` und `rad2`. Alternativ können statt den Winkeln auch die zweielementigen Richtungsvektoren der Schenkel in `rad1` und `rad2` übergeben werden. Darüber hinaus kann als zweites Element von `radius` noch die Anzahl der Punkte bei der polygonalen Darstellung des Kreissegments angegeben werden. Voreinstellung ist 100 Punkte. `mtparc` unterstützt alle Aliase der Funktion `mtpline` zur Gestaltung des Kreisbogens.

Das erste Rückgabeargument `text_coord` enthält die Koordinaten der Beschriftungsposition zwischen dem Kreismittelpunkt und dem gezeichneten Kreissegment. Diese kann mit dem Parameter `text_pos` modifiziert werden (siehe unten). Der optionale zweite Rückgabeparameter enthält das Handle der Kreisbogenlinie.

`text_coord=MTPARC(Centre,rad1,rad2,radius,text_pos)`

Der zweielementige Vektor `text_pos` enthält die auf $[0, 1]$ normierten Positionierungsdaten für den Referenzpunkt der Beschriftung. Der erste Eintrag von `text_pos` steuert die radiale Verschiebung: 0 entspricht dem Kreismittelpunkt, 1 entspricht `radius`. Der zweite Eintrag steuert die Winkelverschiebung: 0 entspricht `rad1`, 1 entspricht `rad2`. Voreinstellung ist `text_pos=[0.75 0.5]`.

Die Koordinaten des berechneten Referenzpunktes werden in der Variablen `text_coord` zurückgegeben.

`text_coord=MTPARC(Centre,rad1,rad2,radius,text_pos,TextString)`

Setzt den in `TextString` übergebenen Text an den durch `text_pos` definierten Referenzpunkt. `mtparc` unterstützt die erweiterte Funktionalität von `mtptext` (Text drehen und ausrichten) nicht. Wird diese benötigt, so ist `mtptext` mit dem Referenzpunkt `text_coord` separat zu verwenden.

Beispiel:

MTP initialisieren und Dreieck zeichnen (Seitenlängen: 3, 4 und 5):

```
>> mtpinit
>> P=[0 5 3.2; 0 0 2.4];
>> mtploop(P)
```

Gestrichelten Thales-Halbkreis zeichnen:

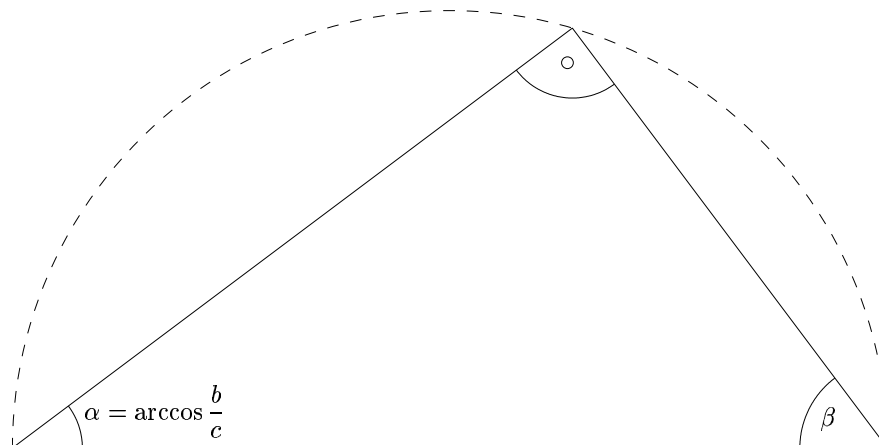
```
>> mtparc([2.5 0],0,pi,2.5,'dashed')
```

Winkel einzeichnen und beschriften:

```
>> mpos=mtparc(P(:,3),P(:,1)-P(:,3),P(:,2)-P(:,3),.4,[.5 .5]);
>> mtpmarker(mpos,'circle')
>> mtparc(P(:,1),0,atan(3/4),.4,[2.1 .4],...
>>         '$\displaystyle{\alpha=\arccos\frac{b}{c}}$')
>> mtparc(P(:,2),pi-atan(4/3),pi,.5,nan,'$\beta$')
```

Koordinatensystem anpassen:

```
>> axis equal, axis off
```



Siehe auch: mtpinit, mtpline, mtp text

3.2.5 mtparea

Zeichnen eines gefüllten Polygonzuges, ähnlich wie `fill`.

Syntax:

```
handle=mtparea(K      [,Alias1,Alias2,...])
handle=mtparea(X,Y    [,Alias1,Alias2,...])
handle=mtparea(X,Y,Z  [,Alias1,Alias2,...])
```

Beschreibung: Zeichnet einen mit einer beliebigen Farbe gefüllten Polygonzug. Die Übergabe der Eckpunkte erfolgt analog zur Übergabe bei `mtploop`, insbesondere wird ebenfalls die schließende Kante automatisch eingefügt. Vordefinierte Aliase sind für

die Farbe: 'black', 'red', 'green', 'blue', 'yellow', 'magenta',
'cyan', 'white', 'gray#', 'nocolor'

die Farbe der Berandung: 'blackedge', 'rededge', 'greenedge',
'blueedge', 'yellowedge', 'magentaedge', 'cyanedge',
'whiteedge', 'gray#edge', 'noedge'

die Dicke der Berandung: 'normalthick', 'thick', 'Thick',
'THick', 'THICK', 'THICK', 'THICK',

'#' steht dabei für eine beliebige Zahl zwischen 0 und 100. Dabei ist der Default 'gray50'. Zu beachten ist, dass die Reihenfolge der Aliase hier wichtig ist. Der Alias 'red' sorgt dafür, dass sowohl die Farbe der Fläche, als auch die Farbe des Randes rot wird. Da Aliase von links nach rechts abgearbeitet werden, muss deshalb zuerst die Farbe und dann die Farbe der Berandung angegeben werden.

Beispiel: Abbildung 3.4 (oben) zeigt Flächen mit verschiedenen Füllfarben. Die Grafik wurde durch die folgenden Befehle erzeugt:

```
>> mtpinit;
>> axis square;
>> n=17;
>> R=linspace(0,2*pi,n+1);
>> for k=1:n
>>   mtparea([0,sin(R(k:k+1));0,cos(R(k:k+1))],...
>>   ['gray' num2str(100*(k-1)/n)],'blackedge','Thick');
>> end;
```

Abbildung 3.4 (unten) zeigt eine typische Verwendung von `mtparea`. Die Grafik wurde aufbauend auf das Beispiel in der Beschreibung von `mtploop` erzeugt durch:

```
>> mtparea(P(:, [2 3 4]), 'gray75');
```

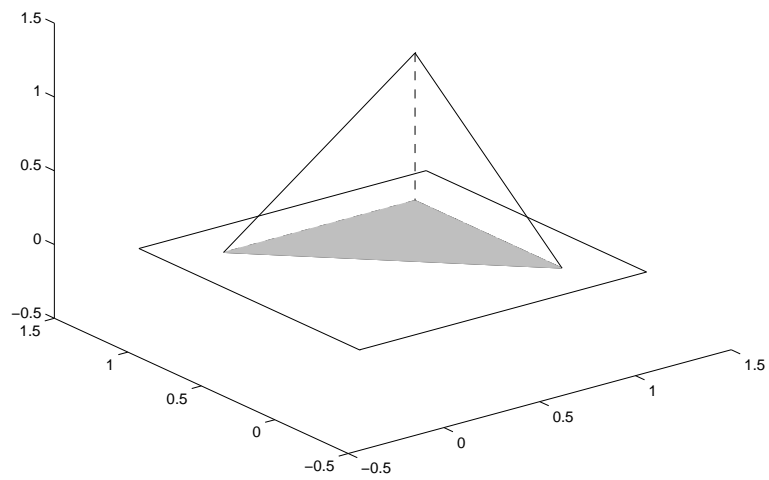
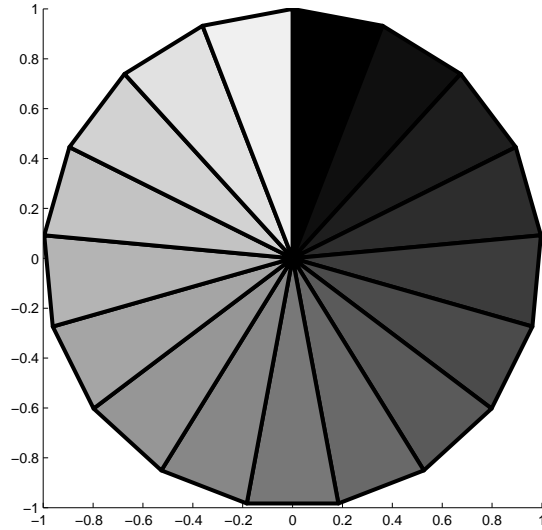


Abbildung 3.4: Mit `mtparea` erzeugte Flächen.

Siehe auch: `mtpinit`, `mtpline`

3.2.6 mtpmarker

Zeichnen von Markersymbolen, ähnlich wie `plot`.

Syntax:

```
handle=mtpmarker(K      [,Alias1,Alias2,...])
handle=mtpmarker(X,Y    [,Alias1,Alias2,...])
handle=mtpmarker(X,Y,Z  [,Alias1,Alias2,...])
```

Beschreibung: Zeichnet Marker an den gegebenen Punkten, deren Koordinaten analog zu denen von `mtpline` übergeben werden.

Arbeitet analog zu `mtpline`. Vordefinierte Aliase sind für

die Farbe: 'black', 'red', 'green', 'blue', 'yellow', 'magenta',
'cyan', 'white', 'gray#', 'transblack', ..., 'solidblack', ... '#'
steht dabei für eine beliebige Zahl zwischen 0 und 100.

das Markersymbol: 'circle', 'dot', 'plus', 'star', 'cross',
'square', 'diamond', 'triangledown', 'triangleup',
'triangleright', 'triangleleft', 'pentagram', 'hexagram'.

die Größe: 'normalsize', 'tiny', 'small', 'large', 'Large', 'LARGE',
'huge', 'Huge', 'HUGE'

Default ist 'transblack', 'circle' und 'normalsize'.

Beispiel: Abbildung 3.4 (oben) zeigt Marker mit verschiedenen Optionen. Die Grafik wurde durch die folgenden Befehle erzeugt. Die Größe der Markersymbole entspricht mit der Option 'normalsize' der Standardgröße von MATLAB. Im Allgemeinen wünscht man sich etwas größere Marker. Die Standardgröße kann in der Funktion `mtpinitrc` eingestellt werden. Siehe dazu Abschnitt 2.1.2.

```
>> mtpinit;
>> axis([0 1 0 1]);
>> Symbols={'circle' 'dot' 'plus' 'star' 'cross' 'square' ...
>> 'diamond' 'triangledown' 'triangleup' 'triangleright' ...
>> 'triangleleft' 'pentagram' 'hexagram'};
>> for k=1:length(Symbols)
>>   mtpmarker(k/(length(Symbols)+1),.75,Symbols{k},'Large');
>> end;
>> Size={'normalsize' 'tiny' 'small' 'normalsize' 'large' ...
>> 'Large' 'LARGE' 'huge' 'Huge' 'HUGE'};
>> for k=1:length(Size)
```

```
>> mtpmarker(k/(length(Size)+2),.5,Size{k});
>> end;
>> Fill={'transblack' 'black' 'solidblack'};
>> mtpline(0:1,[.25 .25]);
>> for k=1:length(Fill)
>>   mtpmarker(k/(length(Fill)+1),.25,Fill{k},'Large');
>> end;
```

Abbildung 3.4 (unten) zeigt eine typische Verwendung von `mtpmarker`. Die Grafik wurde aufbauend auf das Beispiel in der Beschreibung von `mtparea` erzeugt durch:

```
>> mtpmarker(P,'solidblack');
```

Siehe auch: `mtpinit`, `mtpline`

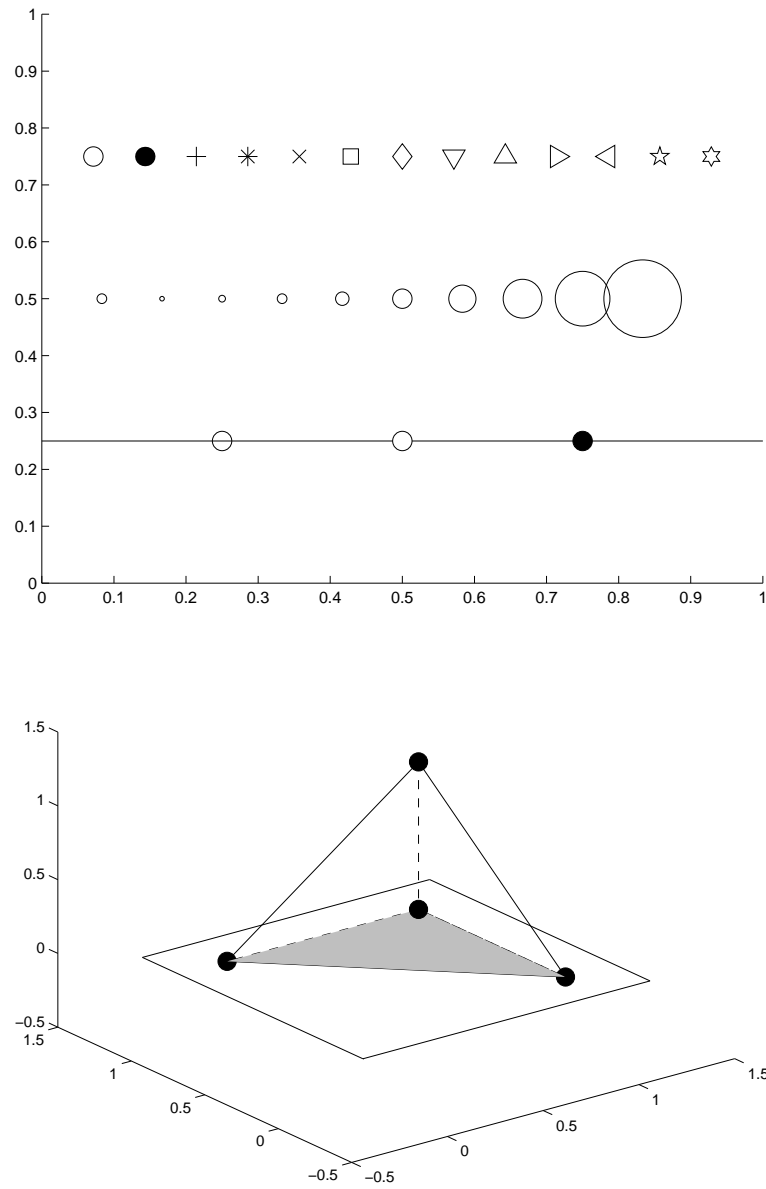


Abbildung 3.5: Mit mtpmarker erzeugte Marker.

3.2.7 mtparrow

Zeichnen eines Pfeiles.

Syntax:

```
handle=mtparrow(K      [,Alias1,Alias2,...])
handle=mtparrow(X,Y    [,Alias1,Alias2,...])
handle=mtparrow(X,Y,Z  [,Alias1,Alias2,...])
                mtparrow('redraw')
```

Beschreibung: Ein Pfeil besteht aus einem Polygonzug und einer Pfeilspitze. Die Koordinaten des Polygonzugs werden analog zu den Punkten bei `mtpline` übergeben. Die Pfeilspitze wird in der Richtung des vorletzten zum letzten Punkt gezeichnet. Sollten diese identisch sein, so wird anstatt des vorletzten der erste nichtidentische Punkt gewählt. Zu beachten ist, dass bei einem sehr fein aufgelösten Pfeil dies zu unerwünschten Effekten führen kann. Siehe auch untenstehendes Beispiel. Zurückgeliefert wird ein 2x1-Vektor von Handles. Das erste Handle ist das der Pfeilspitze, das zweite das des Polygonzuges. Zu beachten ist, dass es ohne einen vorherigen Aufruf von `axis` zu unerwünschten Effekten kommen kann. Diese können allerdings größtenteils mit dem Aufruf `mtparrow('redraw')` behoben werden.

Vordefinierte Aliase sind für

die Farbe: 'black', 'red', 'green', 'blue', 'yellow', 'magenta', 'cyan', 'white', 'gray#'. '#' steht dabei für eine beliebige Zahl zwischen 0 und 100.

die Dicke des Polygonzuges: 'normalthick', 'thick', 'Thick', 'THick', 'THICK', 'THICK'.

das Zeichnen des Polygonzuges: 'withtail', 'notail'

das Verkürzen des Polygonzuges: 'shortentail', 'shortentail#', 'noshortentail'. '#' steht dabei für eine beliebige Zahl zwischen 0 und 1.

die Gestalt der Pfeilspitze: 'noslighthead', 'slighthead'.

die Form der Pfeilspitze: 'normalanglehead', 'acutehead', 'Acutehead', 'ACUTEHEAD', 'obtusehead', 'Obtusehead', 'OBTUSEHEAD', 'DINhead', 'regularhead', 'orthogonalhead'.

die Länge der Pfeilspitze: 'normallengthhead', 'shorthead', 'Shorthead', 'SHORTHEAD', 'longhead', 'Longhead', 'LONGHEAD'.

Dabei sind die erstgenannten jeweils der Default.

Lediglich das Verkürzen des Polygonzuges bedarf einer näheren Erläuterung. Wenn ein Pfeil erzeugt werden soll, so wird zuerst der Polygonzug geplottet und anschließend die Pfeilspitze. Soll nun ein Pfeil mit der Option `'THICK'` erzeugt werden, so wäre das Ergebnis ohne die Option `'shortentail'` ein Pfeil mit einer stumpfen Spitze, da der Polygonzug unter der Spitze zum Vorschein käme. Um das zu vermeiden, werden mit der Option `'shortentail'` alle Polygonkanten in der Pfeilspitze nicht geplottet. `'shortentail'` ist deshalb Default. Bei dreidimensionalen Pfeilen deren Koordinaten nicht allesamt in einer Ebene liegen, sollte die Option durch `'noshortentail'` ausgeschaltet werden. Anstatt `'shortentail'` kann auch `'shortentail#'` benutzt werden. Dabei sollte `'#'` eine Zahl aus $[0, 1]$ sein, z.B. 0.8. Dies ist aber lediglich bei dicken Polygonzügen interessant, die nicht senkrecht in die Pfeilspitze laufen.

Es lassen sich alle Optionen setzen, die mit Hilfe von `set` gesetzt werden könnten. Dies wird erreicht durch die Übergabe einer Cell der Form

```
{PropertyName PropertyValue}
```

in der Aliasliste. Die Gestalt, die Form und die Länge der Pfeilspitze lassen sich nicht mit `set` bearbeiten. Trotzdem können diese drei Parameter direkt mit `{'HeadTipAngle' n}`, `{'HeadBaseAngle' n}`, `{'HeadLength' n}` angesprochen werden. `n` ist dabei eine positive Zahl.

Zu beachten ist, dass im Vergleich zu den anderen Funktionen, wie z.B. `mtpline` im `PropertyName` die ersten vier zusätzlichen Buchstaben entweder `'Head'` oder `'Tail'` sein müssen. Dies ist notwendig, um eine Unterscheidung der Pfeilspitze vom Polygonzug, zu ermöglichen.

`mtparrow('redraw')`: Eine Veränderung der MATLAB-Achsen führt zu einer unerwünschten Verzerrung der Pfeile. `'redraw'` führt zu einer Anpassung an die aktuellen Achsen.

Die von `mtparrow` zurückgelieferten Handles können ganz normal mit `get` und `set` bearbeitet werden. Im Vergleich zu den anderen Grafikfunktionen dürfen sie aber nicht z.B. mit `delete` gelöscht werden, da Arrowhandles intern verwaltet werden.

Beispiel: Abbildung 3.6 (links) zeigt Pfeile mit verschiedenen Optionen. Die Grafik wurde durch die folgenden Befehle erzeugt:

```
>> mtpinit
>> mtparrow([-1 -.5], [-1 -.5])
>> mtparrow([-1 -.5], [1 .5], 'red', 'slighthead')
>> mtparrow([1 .5], [1 .5], {'TailColor' [0 1 0]}, ...
>>         'regularhead', 'Thick')
```

```
>> mtparrow([1 .5],[-1 -.5], 'orthogonalhead', 'red', ...
>>           {'HeadFaceColor' [1 1 0]}, 'notail');
>> mtparrow('redraw')
% falsch zentrierte Pfeilspitze
>> X=linspace(0,2*pi);
>> mtparrow(cos(X)/2,sin(X)/2,'LONGHEAD')
```

Abbildung 3.6 (rechts) zeigt eine typische Verwendung von `mtparrow`. Die Grafik wurde aufbauend auf das Beispiel in der Beschreibung von `mtpmarker` erzeugt. Die Verwendung von `eps` ist notwendig, um den Pfeil über die Fläche zu heben, sodass dieser eindeutig sichtbar wird.

```
>> mtparrow([.5 .9;1 1;1 1]);
>> n=5;
>> X=linspace(-pi/4,0,n);
>> mtparrow([0;1;0]*ones(1,n)...
            +.7*[cos(X);sin(X),10*eps*ones(1,n)]);
```

Siehe auch: `axis`, `mtpinit`, `mtpline`

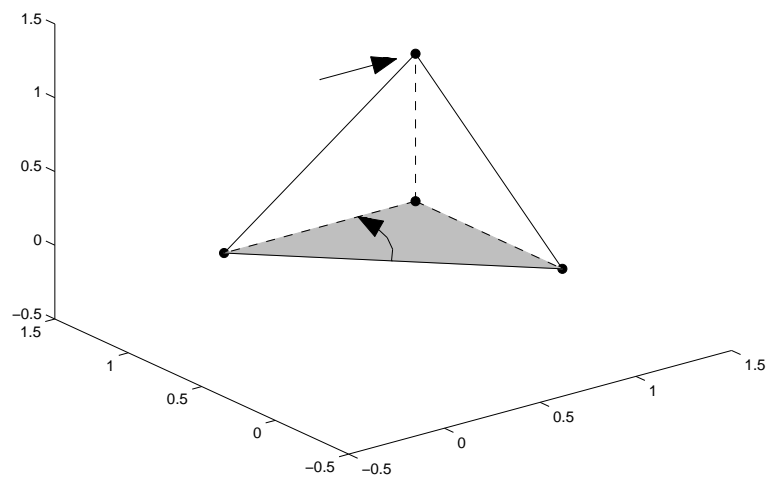
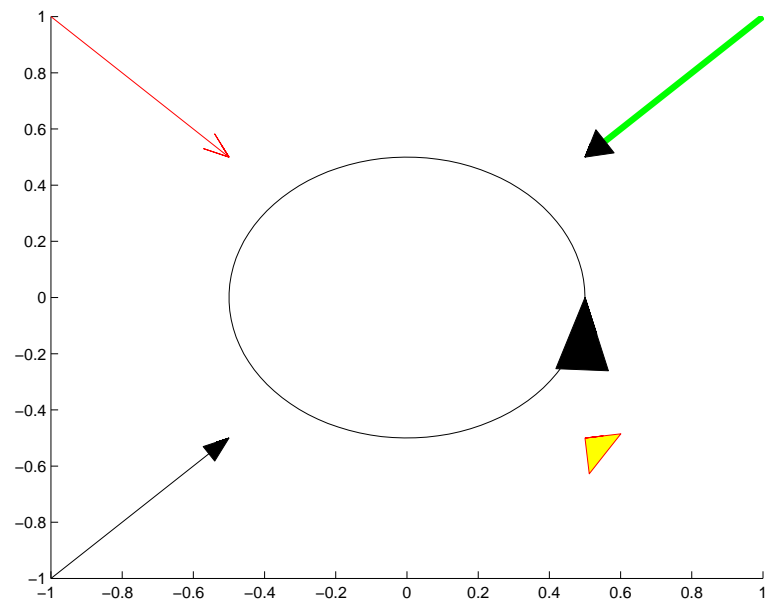


Abbildung 3.6: Mit `mtparrow` erzeugte Pfeile.

Kapitel 4

Textfunktionen

4.1 Einführung

Eine der wichtigsten Funktionen des MTP-Pakets ist die Verbindung von MATLAB und L^AT_EX. In diesem Kapitel werden die MTP-Funktionen zum Setzen von L^AT_EX-Text und deren Nachbearbeitung beschrieben. Zentrale Funktion ist hier `mptext`. Diese ermöglicht das freie Setzen von L^AT_EX-Text in einer MATLAB-Grafik. Daneben erlauben `mptexaxes` und `mptitle` die Beschriftung der Koordinatenachsen mit L^AT_EX. `mptshow` zeigt die resultierende Grafik inklusive L^AT_EX-Text an.

MTP-Grafiken, welche Text-Objekte enthalten, sind mit den Funktionen `mtpsavetex` oder `mptsaveeps` abzuspeichern. Wie die so erzeugten Grafiken in Dokumente einzubinden sind, beschreibt der Abschnitt 6.

4.2 Befehlsübersicht

4.2.1 mptext

Setzt \LaTeX -Text in die Grafik.

Syntax:

```
mptext(K,          TextString)
mptext(x,y,[z,] TextString)
mptext( ... , TextString,Alignment)
mptext( ... , TextString,Alignment,TextDistanceFactor)
mptext( ... , TextString,Alignment,TextDistanceFactor,Rotation)
```

Beschreibung: Setzt \LaTeX -Quellcode in die Grafik. Dieser wird weiterverarbeitet durch die Funktion `mtpsavetex`, `mtpsaveeps` oder `mtpshow`.

Für die Parameter `Alignment`, `TextDistanceFactor` und `Rotation` sind Defaultwerte definiert (siehe unten). Der Default wird angenommen, wenn der betreffende Parameter nicht oder auf `NaN` gesetzt wird.

Die Parameter `TextString`, `Alignment`, `TextDistanceFactor` und `Rotation` sind horizontale Cells, die jeweils soviele Werte enthalten wie Referenzpunkte vorhanden sind.

Es ist möglich einzelne Nicht-Cell-Argumente zu übergeben, die in eine Cell mit identischen Einträgen umgewandelt werden.

`mptext(K,TextString)` schreibt die \LaTeX -Strings aus `TextString` unbearbeitet an die durch die Matrix `K` gegebenen Referenzpunkte. Diese liegen, falls `K`

- einzeilig ist, bei $(i, K(i))$,
- zweizeilig ist, bei $(K(1, i), K(2, i))$,
- dreizeilig ist, bei $(K(1, i), K(2, i), K(3, i))$.

`mptext(x,y,[z,]TextString)` schreibt \LaTeX -Strings aus `TextString` an die Referenzpunkte $(x(i), y(i), z(i))$. `x`, `y` und `z` sind horizontale Vektoren, welche die x -, y - und z -Koordinaten der einzelnen Referenzpunkte enthalten. Hierbei ist `z` optional.

`mptext(...,TextString,Alignment)` legt die Textausrichtung zum Referenzpunkt fest. `mptext` versteht \LaTeX -Alignments wie

```
'rb'  'cb'  'lb'
'rB'  'cB'  'lB'
'rc'  'cc'  'lc'
'rt'  'ct'  'lt'
```

Genauer gesagt darf jeder Eintrag von `Alignment` je einen Buchstaben für die horizontale und vertikale Ausrichtung des betreffenden Textobjekts enthalten. Die Kürzel bedeuten:

Horizontal: r=right c=center l=left
 Vertikal : b=bottom B=Baseline c=center t=top

`mpttext(...,TextString,Alignment,TextDistanceFactor)` fügt einen durch den numerischen Wert `TextDistanceFactor` skalierten und durch `Alignment` orientierten Abstand zwischen Referenzpunkt und dem zugehörigen Textobjekt ein. Die Skalierung bezieht sich auf die Maßeinheit, die im \LaTeX -Dokument mit Hilfe von `\mpttextdistanceunit`, bzw. durch entsprechenden Aufruf von `mtpsaveeps` (siehe dort), definiert wird. Demzufolge bleibt der Effekt dieser Option im MATLAB-Grafikfenster unsichtbar.

`mpttext(...,TextString,Alignment,TextDistanceFactor,Rotation)` dreht den `TextString{i}` um `Rotation{i}` Grad.

Default-Werte:

```
Alignment      : 'cc'
TextDistanceFactor: 0
Rotation       : 0
```

Beispiel: Abbildung 4.1 wird von dem folgenden Programm erzeugt:

```
>> mtpinit
>> valign={'t','c','B','b'};
>> halign={'l','c','r'};
>> axis off
>> for hind=1:length(halign)
>>   for vind=1:length(valign)
>>     mtpmarker(hind,vind,'plus',{'markersize',100})
>>     alignment=[halign{hind} valign{vind}];
>>     mpttext(hind,vind,['Alg: ' alignment],alignment)
>>   end
>> end
>> mtpshow(12)
```

Die in Abbildung 4.2 linksstehende Grafik ist das Ergebnis des zweiten Beispiels. Hierbei wurde `\mpttextdistanceunit1mm` gesetzt.

```
>> mtpinit
>> axis off
>> mtpmarker(0.5,0.5,'plus')
>> for j=0:45:315
>>   mpttext(0.5,0.5,['{\scriptsize Wl: ' num2str(j) ...
>>                 ', TDF: ' num2str(j/60) '}]', 'lc', j/60, j)
>> end
>> mtpshow(12)
```

Das dritte Beispiel führt das Schaubild aus `mtparrow` fort.

```
>> mptext(P, ...
      {'\small A}' '\small B}' ...
      '\small C}' '\small D}'), ...
      {'lb' 'tc' 'lc' 'rc'},1)
>> % Schwerpunkt des Winkeldreiecks.
>> vs=[.39 .85 0]';
>> mptext(vs, '$\alpha$')
>> mptext(.7,1,0, '$\alpha$', 'cc',1)
>> mptext(.5,1,1, 'Spitze', 'rc',1)
```

Siehe auch: `mtpinit`, `mtpsaveeps`, `mtpsavetex`, `mtpshow`, `mptexaxes`,
`mptitle`

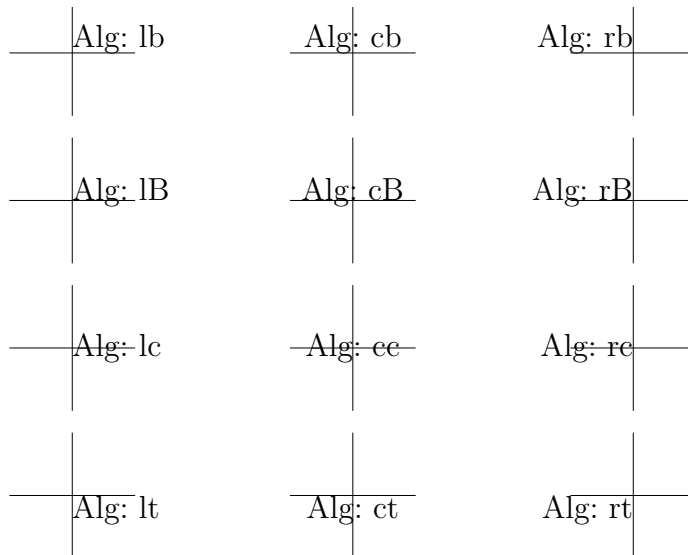


Abbildung 4.1: Alle Alignmenttypen auf einen Blick.

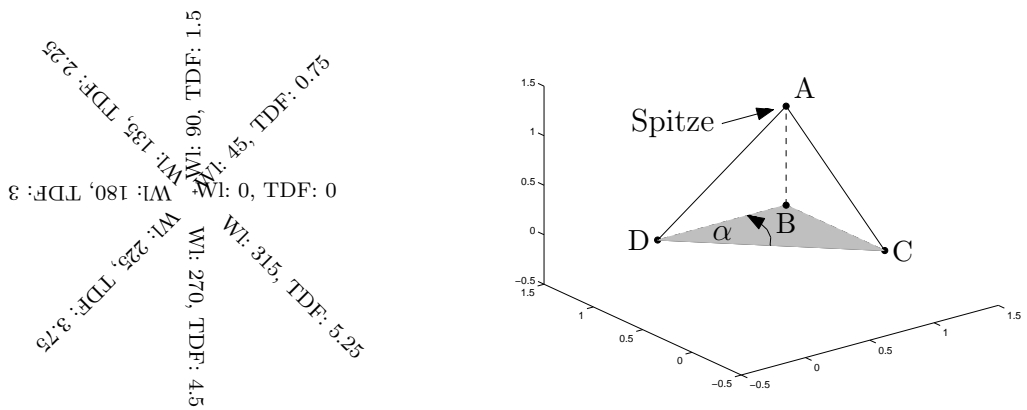


Abbildung 4.2: Mit `mtptext` erzeugte Schriftzüge.

4.2.2 mptexaxes

Achsenbeschriftung durch \LaTeX .

Syntax:

```
mptexaxes
mptexaxes(xTick)
mptexaxes(xTick,xTickLabel)
mptexaxes(xTick,xTickLabel,yTick)
mptexaxes(xTick,xTickLabel,yTick,yTickLabel)
mptexaxes(xTick,xTickLabel,yTick,yTickLabel,zTick)
mptexaxes(xTick,xTickLabel,yTick,yTickLabel,zTick,zTickLabel)
mptexaxes(...,DistanceFactor)
```

Beschreibung: Beschriftet die Achsen mit \LaTeX . Die Weiterverarbeitung erfolgt durch die Funktion `mtpsavetex`, `mtpsaveeps` oder `mtpshow`. `mptexaxes` sollte als letzte MTP-Funktion vor der Weiterverarbeitung aufgerufen werden.

Für die ersten sechs Parameter ist der Default `NaN`. Er wird angenommen, wenn der betreffende Parameter nicht gesetzt wird.

`mptexaxes` wandelt die aktuelle Achsenbeschriftung in \LaTeX um und setzt die `#limmode` properties der Achsen auf `'manual'`.

`mptexaxes(...)` setzt die gegebenen Achsenmarkierungen und Schriftzüge. Die zu beschriftenden Achsenmarkierungen, die sogenannten Ticks, können auf folgende Weisen angegeben werden:

- [...] Vektor mit streng monoton wachsenden Werten. Die Ticks werden bei den gegebenen Werten auf der Achse gesetzt. Insbesondere bewirkt die Übergabe des leeren Vektors `[]`, dass keine Ticks gesetzt werden.
- `NaN` Die momentanen Ticks werden beibehalten. Neubeschriftung ist möglich.
- `{}` Die betreffende Achse wird unmodifiziert beibehalten. Das nachfolgende Labelargument wird ignoriert.

Die zu den Ticks gehörenden Schriftzüge, die Labels, können wie folgt übergeben werden:

- [...] setzt `Label(i)` an `Tick(i)`. Beide Vektoren müssen von gleicher Länge sein.
- {...} setzt `Label{i}` an `Tick(i)`. `Tick` und `Label` müssen von gleicher Länge sein. `Label{i}` kann sein
 - NaN, um `Tick(i)` mit `Tick(i)` zu beschriften.
 - Zahl
 - String
- '...' beschriftet alle Ticks mit dem vorgegebenen String. Insbesondere ergibt die Übergabe von '' Ticks ohne Beschriftung.
- NaN verwendet den `Tick`-Vektor zur Beschriftung.

Bei der Verwendung von `DistanceFactor` wird ein durch `DistanceFactor` skaliertes Abstand zwischen der Achse und ihrer Beschriftung eingefügt. Die Skalierung bezieht sich auf die Maßeinheit, die im L^AT_EX-Dokument mit Hilfe von `\mptextdistanceunit`, bzw. durch entsprechenden Aufruf von `mtpsaveseps` (siehe dort), definiert wird. Demzufolge bleibt der Effekt dieser Option im MATLAB-Grafikfenster unsichtbar. `DistanceFactor` kann sein:

- Eine Zahl: Für alle Achsen wird derselbe Faktor verwendet.
- Ein dreielementiger Vektor: Für jede der drei Achsen wird der entsprechende Faktor verwendet.
- Ein zweielementiger Vektor: Für die dritte Achse wird der Faktor 0 verwendet.

Default-Werte:

Ticks und Labels: NaN

`DistanceFactor`: 0

Beispiel: Das Programm

```
mtpinit
t=linspace(0,2*pi);
mtpline(t,sin(t),'Thick')
mptexaxes([0:.5*pi:2*pi],[0 '$\frac{\pi}{2}$' '$\pi$' ...
'$\frac{3\pi}{2}$' '$2\pi$'],[-1:.5:1],NaN,[0 .5])
mtpshow(12)
```

erzeugt die linke obere Grafik aus Abbildung 4.3, gefolgt von dem rechten oberen Schaubild das erzeugt wurde durch

```
>> mtpinit
>> view(3)
>> grid on
>> axis([0 6*pi 0 1 -.5 .5])
>> [xx,yy]=meshgrid(linspace(0,6*pi,50),linspace(0,1,50));
>> zz=sin(xx)./(2+xx);
>> surf(xx,yy,zz)
>> mptexaxes([0:2*pi:6*pi],{0 '$2\pi$' '$4\pi$' '$6\pi$'}, ...
    [],NaN, [-.5:.25:.5], ...
    {'$-100\%$' '$-50\%$' '$0\%$' '$50\%$' '$100\%$'})
>> mtpshow(12)
```

Die linke untere Grafik ist das Ergebnis der Zeilen

```
>> surf(peaks)
>> grid off
>> mtpinit
>> mptexaxes([40*log([1:3])],'', ...
             [0:20:40],{'\tt PostScript' '\TeX' '\sc Matlab'},{ })
>> mtpshow(12)
```

Das letzte Beispiel schließlich führt die Grafik aus der Beschreibung von `mtpshow` fort:

```
>> mptexaxes([0,0.5,1],NaN,[0,0.5,1],NaN,[0,0.5,1],NaN)
```

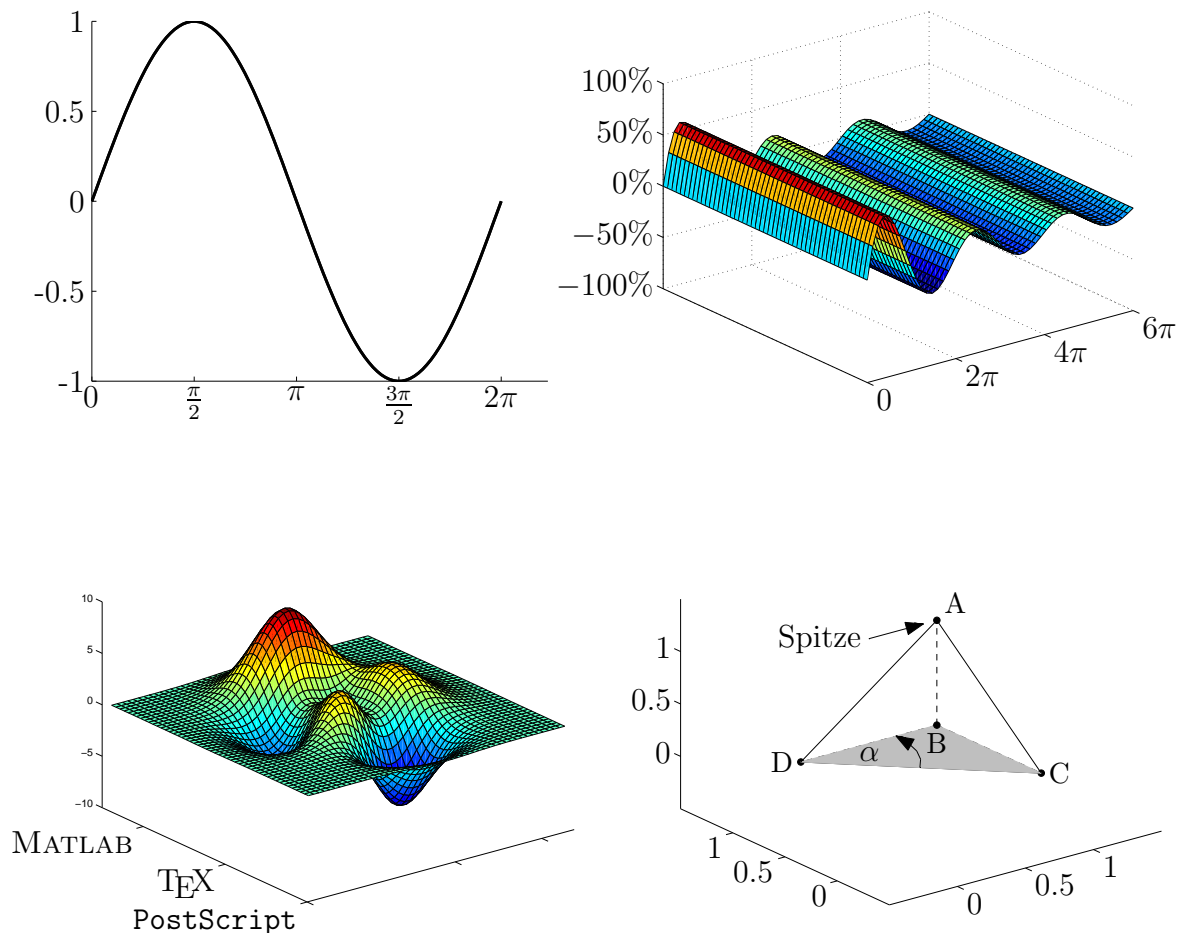


Abbildung 4.3: Anwendungsbeispiele für `mptexaxes`.

Siehe auch: `mtpsaveeps`, `mtpsavepdf`, `mtpshow`, `mtpshow`, `mtpshow`

4.2.3 mtptitle

Setzt Titel und Bezeichnung der Achsen einer Grafik mit \LaTeX .

Syntax:

```
mtptitle(Title)
mtptitle(Title,xLabel [,xDistanceFactor])
mtptitle(Title,xLabel [,xDistanceFactor], ...
         yLabel [,yDistanceFactor])
mtptitle(Title,xLabel [,xDistanceFactor], ...
         yLabel [,yDistanceFactor], ...
         zLabel [,zDistanceFactor])
```

Beschreibung: Beschriftet die Grafik, analog zu den MATLAB-Befehlen `title`, `xlabel`, `ylabel` und `zlabel`, mit \LaTeX . Die Weiterverarbeitung erfolgt durch die Funktion `mtpsavetex`, `mtpsaveeps` oder `mtpshow`.

Bei der Verwendung dieser Funktion ist zu beachten: Die oben genannten MATLAB-Befehle werden zur Beschriftung eingesetzt. Diese sorgen dafür, dass die Beschriftungen im MATLAB-Font richtig positioniert werden. Dies ist nach der Umsetzung in \LaTeX nicht mehr gewährleistet.

Eine Lösung bieten die optionalen Parameter `#DistanceFactor`, sowie die zusätzliche Verwendung von Ruleboxen zur Verschiebung des Textes.

Für alle Parameter sind Defaultwerte definiert (siehe unten). Der Default wird angenommen, wenn der betreffende Parameter nicht gesetzt wird.

`mtptitle(Title)` schreibt den \LaTeX -String `Title` an die Titelposition der Grafik. Die Übergabe von `{}` bewirkt, dass der aktuelle Titel nicht modifiziert wird.

`mtptitle(Title,...)` setzt die übergebenen `Label`. Die Übergabe von `{}` bewirkt, dass die betreffende Achse nicht modifiziert wird.

Die Zahlenwerte `#DistanceFactor` fügen einen skalierten Abstand zwischen dem Referenzpunkt und dem zugehörigen Textobjekt ein. Diese Skalierung bezieht sich auf `\mptextdistanceunit` (siehe Seite 41). Der Effekt dieser Option bleibt im MATLAB-Grafikfenster unsichtbar.

Default-Werte:

```
Title, #Label      : {}
#DistanceFactor   : 0
```

Beispiel: Das Programm

```
>> mtpinit
>> view(3)
>> [x,y]=meshgrid(-3:.5:3,-3:.1:3);
>> z =peaks(x,y);
>> ribbon(y,z)
>> mptexaxes(NaN,NaN,NaN,NaN,[-5 0 5],NaN)
>> mptitle('\large Peaks and Stripes', ...
           'x-Achse', 'y-Achse', 'z-Achse',1)
>> mtpshow(12)
```

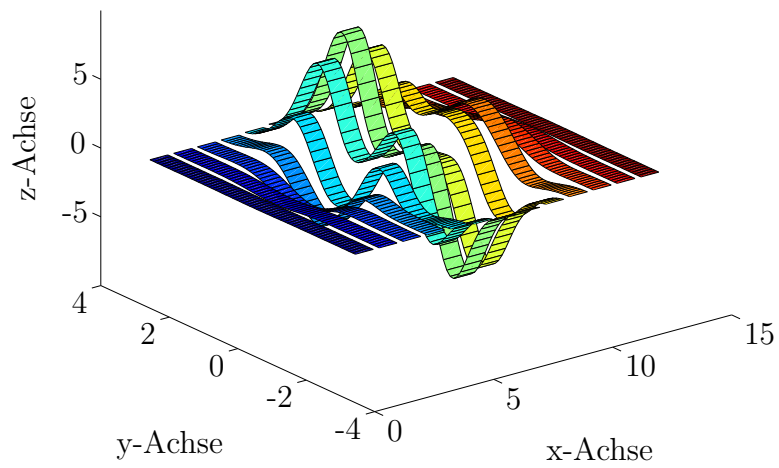
erzeugt das linke Schaubild aus Abbildung 4.4. Die Zeile

```
>> mptitle('\large MTP-Tetraeder')
```

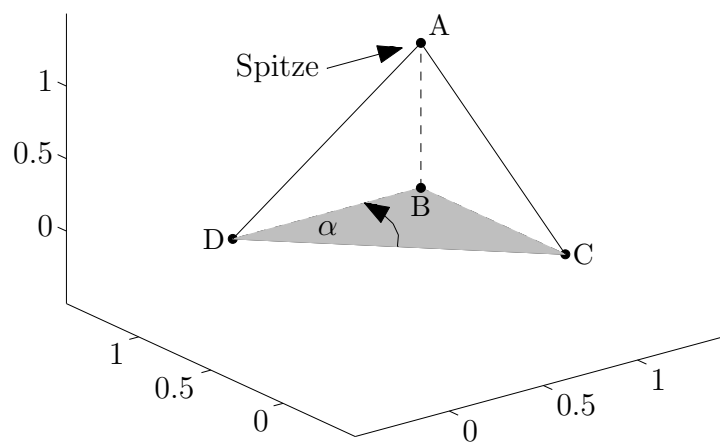
führt das Beispiel aus `mptexaxes` fort.

Siehe auch: `mptexaxes`, `mptext`

Peaks and Stripes



MTP-Tetraeder

Abbildung 4.4: Anwendungen von `mtptitle`.

4.2.4 mtpsavetex

Speichert MTP-Grafik für das Einfügen in \LaTeX .

Syntax:

```
mtpsavetex(FileName)
```

Beschreibung: `mtpsavetex(FileName)` erzeugt zwei Dateien: `'FileName.eps'`, welche die grafischen Objekte enthält und `'FileName.tex'` mit den \LaTeX -Textelementen.

Nach dem Aufruf von `mtpsavetex` kann die Grafik in ein \LaTeX -Dokument eingebunden werden, indem

1. Im Dokumentkopf der Befehl `\usepackage{mtp}` und
2. die Grafik an der gewünschten Stelle im Hauptdokument mit Hilfe der folgenden Befehle eingefügt wird:

```
{
\mptextdistanceunit#
\mtpincludegraphics[...]{FileName}
}
```

Hierbei ist für `#` die \LaTeX -Maßeinheit für den dimensionslosen Skalierungsfaktor `TextDistanceFactor` einzusetzen, welcher z.B. von `mpttext` verwendet wird. Ist `\mptextdistanceunit#` nicht angegeben, wird der Default `\mptextdistanceunit0.3cm` gesetzt, sofern er nicht durch Weglassen der geschweiften Klammern beim Einbinden einer vorigen Grafik bereits gesetzt wurde.

Die optionalen Parameter von `\mtpincludegraphics` entsprechen denen des `graphicx`-Paketes [CR99].

Beispiel: Zunächst wird das in Abbildung 4.5 dargestellte Schaubild mit den nachfolgenden Befehlen erzeugt.

```
>> mtpinit
>> mtpline([0 1;0 1])
>> mptexaxes([0 0.5 1],{0 '$\frac{1}{2}$' 1}, ...
             [0 0.5 1],{0 '$\frac{1}{2}$' 1})
>> mpttext(0,1,'$f(x)=\frac{(x-0.5)x}{x-0.5}$','t1',1)
>> mpttext(0.5,0.5,'L"ucke bei $x=0.5$','t1',1)
>> mtpmarker(0.5,0.5)
```

Dann wird

```
>> mtpsavetex('test')
```

aufgerufen, woraufhin die Dateien 'test.tex' und 'test.eps' erzeugt werden. Letztere enthält die (hier in Übergröße dargestellten) Tags und ist in Abbildung 4.6 links zu sehen.

Zuletzt wird das L^AT_EX-Zieldokument im Kopf um die Anweisung

```
\usepackage{mtp}
```

und im Hauptteil um die Zeilen

```
{
\mptextdistanceunit1mm
\mtpincludegraphics[width=0.47\linewidth]{test}
}
```

erweitert. Das Resultat ist in Abbildung 4.6 rechts dargestellt.

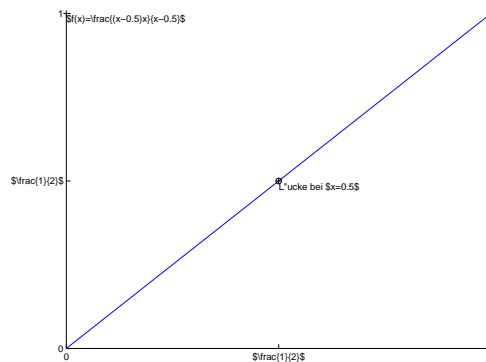


Abbildung 4.5: Das MATLAB-Grafikfenster mit previews.

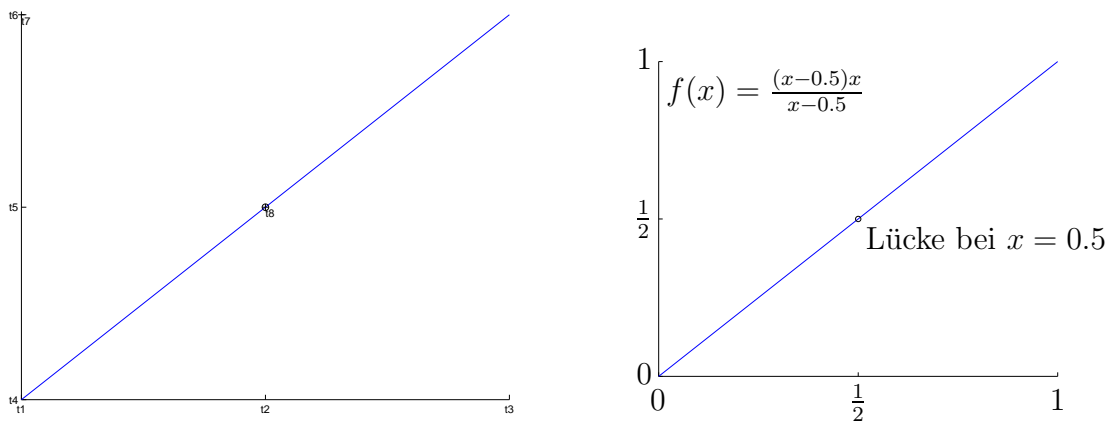


Abbildung 4.6: Tags und L^AT_EX.

Siehe auch: Abschnitt 6, `mtpsaveeps`, `mtpshow`, `mtpsexaxes`, `mtpstext`, `mtpstitle`.

4.2.5 mtpsaveeps

Speichert MTP-Grafik als L^AT_EX-beschriftete EPS-Datei

Syntax:

```
mtpsaveeps(FileName)
mtpsaveeps(FileName,FontSize)
mtpsaveeps(FileName,FontSize,Options)
mtpsaveeps(FileName,FontSize,Options,mptextdistanceunit)
```

Beschreibung: Erstellt ein L^AT_EX-Dokument und bindet die Grafik mit Hilfe von `\includegraphics` ein. Das Dokument wird kompiliert und in eine EPS-Datei umgewandelt.

Für die Parameter `FontSize`, `Options` und `mptextdistanceunit` sind Defaultwerte definiert (siehe unten). Der Default wird angenommen, wenn der betreffende Parameter nicht oder auf NaN gesetzt wird.

`mtpsaveeps(FileName)` speichert MTP-Grafik in der Datei `FileName.eps` ab, wobei die Beschriftung durch L^AT_EX erfolgt.

`mtpsaveeps(FileName,FontSize)` setzt die Dokumentschriftgröße `FontSize` für alle L^AT_EX-Elemente in der Einheit pt. `FontSize` ist als Zahl zu übergeben. Akzeptiert werden die L^AT_EX-Standardwerte 10, 11 und 12.

`mtpsaveeps(FileName,FontSize,Options)` gibt den String `Options` unverändert an den `\includegraphics`-Befehl aus dem `graphicx`-Paket [CR99] weiter, welcher beim Einbinden der Grafik verwendet wird. Dies ermöglicht beispielsweise die Skalierung der Grafik.

`mtpsaveeps(FileName,FontSize,Options,mptextdistanceunit)` definiert eine Einheit für den von `mptext` verwendeten, dimensionslosen Parameter `TextDistanceFactor`. `mptextdistanceunit` ist ein String, der ein L^AT_EX-Maß enthält.

Default-Werte:

```
FontSize           : 10
Options            : 'width=\linewidth'
mptextdistanceunit: '0.3 cm'
```

Hauptproblem bei der Erzeugung der EPS-Datei ist die Berechnung einer passenden Bounding-Box. Die interne Funktion `mtpinternaldvi2eps` übernimmt dies. Sofern die Bounding-Box nicht passt, besteht die Möglichkeit durch Setzen von

```
MTP.FLAGS.generate_alternative_eps_files=1;
```

in der Konfigurationsdatei `mtpinitrc` (siehe Seite 12) die Erzeugung alternativer EPS-Dateien einzuschalten. Erzeugt werden dann zusätzlich eine mit Hilfe von `ghostscript` erzeugte EPS-Datei (Endung `_gs.eps`) sowie eine EPS-Datei mit einer Bounding-Box im DIN A4-Format (Endung `_a4.eps`).

Beispiel: Der linke Teil von Abbildung 4.7 ist direkt nach der Ausführung des nachfolgenden Programmes dem MATLAB-Fenster entnommen worden.

```
>> mtpinit
>> mtpline([0 1;0 1])
>> mptexaxes([0 0.5 1],{0 '$\frac{1}{2}$' 1}, ...
             [0 0.5 1],{0 '$\frac{1}{2}$' 1});
>> mpttext(0,1,'$f(x)=\frac{(x-0.5)x}{x-0.5}$','t1',0.1);
>> mpttext(0.5,0.5,'L"ucke bei $x=0.5$.','t1',0.1);
>> mtpmarker(0.5,0.5);
```

Nach dem Aufruf

```
>> mpsaveeps('mtpsaveepstest',12,NaN,'2mm')
```

wird die Datei `mtpsaveepstest.eps` erzeugt, welche in Abbildung 4.7 rechts zu sehen ist.

Siehe auch: `mtpsavetex`, `mtpshow`, `mptexaxes`, `mpttext`, `mpttitle`

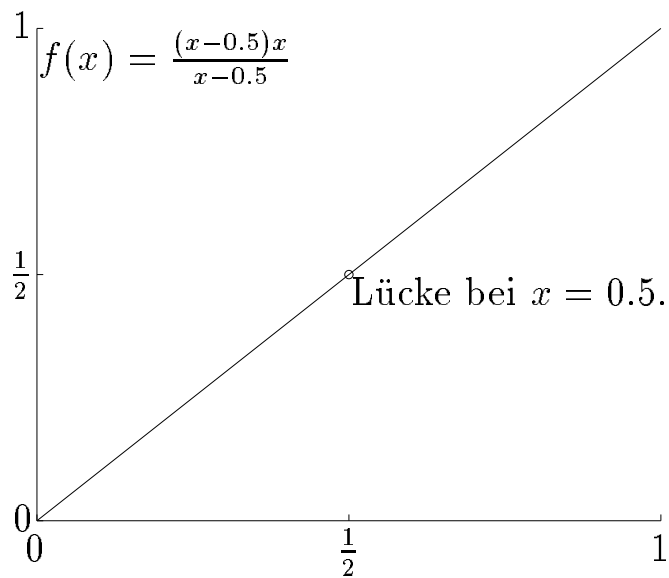
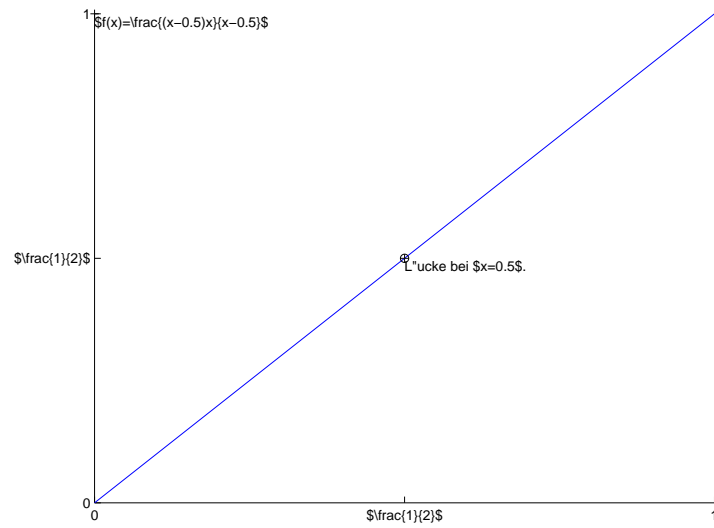


Abbildung 4.7: MATLAB-Preview und fertige EPS-Grafik.

4.2.6 mtpshow

Zeigt die vollständige Grafik inklusive \LaTeX Text.

Syntax:

```
mtpshow
mtpshow(FontSize)
mtpshow(FontSize,Options)
mtpshow(FontSize,Options,mtptextdistanceunit)
```

Beschreibung: Mit MTP erzeugte \LaTeX -Elemente können im MATLAB-Grafikfenster nicht korrekt angezeigt werden. Erst ein Aufruf von `mtpsave $\text{\textit{t}}\text{\textit{e}}\text{\textit{x}}$` oder `mtpsave $\text{\textit{e}}\text{\textit{p}}\text{\textit{s}}$` erzeugt das Gewünschte. `mtpshow` ermöglicht bereits während der Bearbeitung eine Vorschau des Ergebnisses.

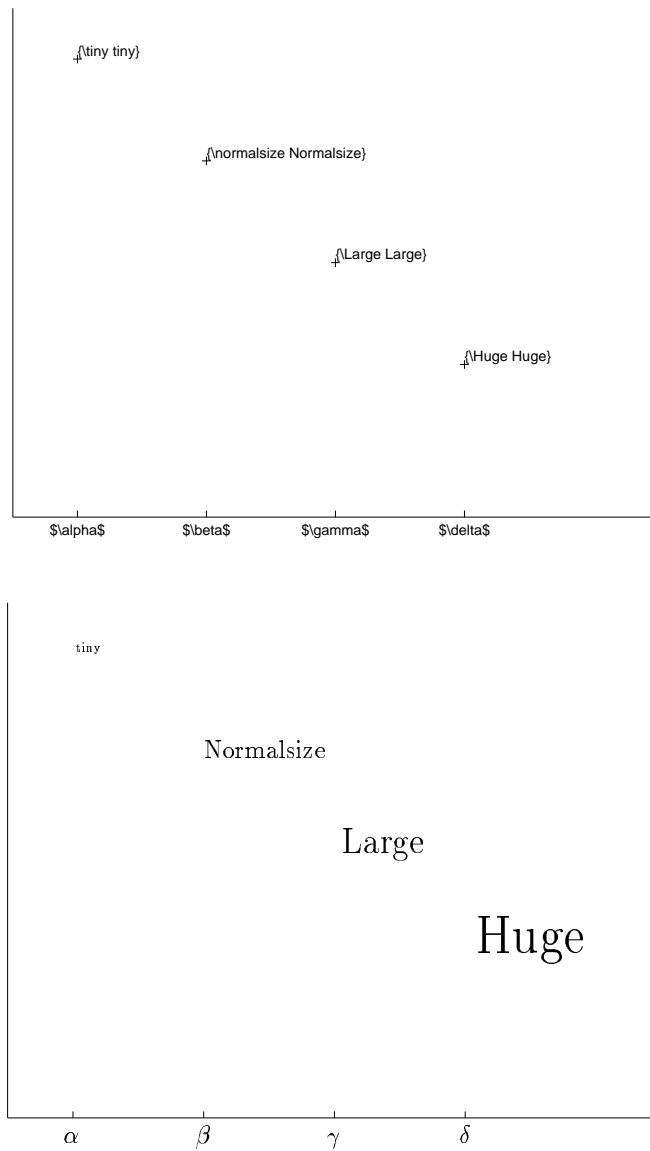
`mtpshow` erzeugt mit Hilfe von `mtpsave $\text{\textit{e}}\text{\textit{p}}\text{\textit{s}}$` die Datei `tmp_mtpshow. $\text{\textit{e}}\text{\textit{p}}\text{\textit{s}}$` und zeigt diese mit einem Postscriptviewer an. Die Datei wird beim Aufruf von `mtpclose` wieder gelöscht.

`mtpshow(...)` Alle Parameter sind wie in der Beschreibung von `mtpsave $\text{\textit{e}}\text{\textit{p}}\text{\textit{s}}$` auf Seite 53 erläutert zu verwenden.

Beispiel:

```
>> mtpinit
>> K=[0.1 0.3 0.5 0.7
      0.9 0.7 0.5 0.3]
>> mtp $\text{\textit{t}}\text{\textit{e}}\text{\textit{x}}$ axes(K(1,:), ...
    {'$\alpha$', '$\beta$', '$\gamma$', '$\delta$'}, [], {})
>> mtp $\text{\textit{t}}\text{\textit{e}}\text{\textit{x}}$ (K, {'\tiny tiny'}, '\normalsize Normalsize', ...
    '\Large Large', '\Huge Huge'}, 'lb')
>> mtpshow(12, 'width=0.8\linewidth')
```

Siehe auch: `mtpclose`, `mtpsave $\text{\textit{e}}\text{\textit{p}}\text{\textit{s}}$`

Abbildung 4.8: Anzeige im MATLAB-Fenster(oben), Anzeige mit `mtpshow`(unten).

Kapitel 5

Sonstige Funktionen

5.1 Einführung

In diesem Kapitel werden jene MTP-Funktionen beschrieben, die sich weder zu den Grafik- noch zu den Textfunktionen zuordnen lassen. Hierzu zählen `mtpclose` mit der eine MTP-Sitzung beendet wird und die Funktion `mtpsearch` mit der die Aliaslisten durchsucht und ausgegeben werden können.

5.2 Befehlsübersicht

5.2.1 mtpclose

Beendet den MATLAB-Teil einer MTP-Sitzung.

Syntax:

```
mtpclose
```

Beschreibung: `mtpclose` löscht alle MTP-relevanten globalen Variablen sowie, gegebenenfalls, die von `mtpshow` erzeugte Datei `tmp_mtpshow.eps`. Ferner wird der MATLAB-interne \TeX -Interpreter, der für die Dauer der MTP-Sitzung inaktiv war, wieder auf seinen ursprünglichen Zustand vor dem Aufruf von `mtpinit` zurückgesetzt.

Siehe auch: `mtpinit`.

5.2.2 mtpsearch

Hilfsroutine für den Benutzer, um sich einen Überblick über die definierten Aliase zu verschaffen. `mtpsearch` durchsucht die komplette Aliasliste nach Übereinstimmungen mit dem gesuchten Schlüsselwort.

Syntax:

```
mtpsearch(mtpfunction)
mtpsearch(mtpfunction,keyword)
mtpsearch(mtpfunction,keyword,type)
```

Beschreibung:

`mtpsearch(mtpfunction)` Gibt alle für die Funktion `mtpfunction` definierten Aliase auf den Bildschirm aus.

`mtpsearch(mtpfunction,keyword)` Durchsucht die komplette Aliasliste nach dem Schlüsselwort `keyword`. Entspricht dem dreimaligen Aufruf mit `type=1`, `type=2` und `type=3`.

`mtpsearch(mtpfunction,keyword,type)` Durchsucht die Aliasliste nach `keyword`. Die entsprechende Aliasdefinition wird auf den Bildschirm geschrieben falls für

`type='alias'` **oder** `type=1` das `keyword` mit dem Aliasnamen übereinstimmt. Groß- und Kleinschreibung wird unterschieden.

`type='propertyname'` **oder** `type=2` das `keyword` mit einem Propertyname übereinstimmt. Groß- und Kleinschreibung wird nicht unterschieden.

`type='propertyvalue'` **oder** `type=3` das `keyword` mit einem Propertyvalue bzw. mit einem definierenden Alias übereinstimmt. Groß- und Kleinschreibung wird unterschieden.

Beispiel:

```
>> mtpinit
>> mtpsearch('mtpmarker')
'dot' {'Marker' '.'}
...
'red' {'MarkerEdgeColor' [1 0 0]}
      {'MarkerFaceColor' [1 1 1]}
...
'r' {'red'}
...
'transred' {'MarkerEdgeColor' [1 0 0]} 'trans'}
...
'solidred' {'MarkerEdgeColor' [1 0 0]}
```

```
        {'MarkerFaceColor' [1 0 0]}}
...
'tiny' {'MarkerSize' 6}}
...
>> mtpsearch('mtpmarker','red')
'red' {'MarkerEdgeColor' [1 0 0]}
      {'MarkerFaceColor' [1 1 1]}}
Propertyname 'red' not found.
'r' {'red'}
```

```
>> mtpsearch('mtpmarker',[1 0 0],'propertyvalue')
'red' {'MarkerEdgeColor' [1 0 0]}
      {'MarkerFaceColor' [1 1 1]}}
'transred' {'MarkerEdgeColor' [1 0 0]} 'trans'}
'solidred' {'MarkerEdgeColor' [1 0 0]}
           {'MarkerFaceColor' [1 0 0]}}
```

Siehe auch: mtpinit, mtpinitrc, Abschnitt 2.2.

Kapitel 6

Einbinden von MTP-Grafiken in \LaTeX

6.1 Die MTP-Grafiktypen

Das MTP-Paket stellt mit den Funktionen `mtpsaveeps` und `mtpsavetex` zwei prinzipiell verschiedene Methoden zur Speicherung von MTP-Grafiken zur Verfügung. Die Vor- und Nachteile beider Ansätze werden im Folgenden erläutert.

6.1.1 `mtpsaveeps`-Grafiken

Die Funktion `mtpsaveeps` erzeugt eine vollständige EPS-Grafik. Diese beinhaltet insbesondere die durch \LaTeX gesetzten Beschriftungen als Bitmap-Grafiken.

Vorteile

- Grafik ist unabhängig von MTP und \LaTeX . Sie kann mit Standardpaketen, wie zum Beispiel `epsfig`, `graphicx`, etc., in \LaTeX eingebunden werden (hilfreich wenn das `mtp.sty`-Paket aufgrund fester Stilvorgaben nicht verwendet werden darf).
- Fontgröße skaliert mit der Grafikgröße.

Nachteile

- Die \LaTeX -Beschriftung im Bitmap-Format verliert je nach Skalierung an Qualität und wird bei unterschiedlicher Skalierung in x - und y -Richtung verzerrt.
- Fontgröße der Beschriftung passt je nach Skalierung nicht zur Dokument-schriftgröße.
- Fonttyp vom Dokument unabhängig.

In der Praxis überwiegen die Nachteile. Deshalb wird empfohlen, die Grafik nur dann mit `mtpsaveeps` abzuspeichern, wenn feste Stilvorgaben die Verwendung des MTP-Pakets in \LaTeX nicht erlauben.

6.1.2 `mtpsavetex`-Grafiken

Im Gegensatz zu `mtpsaveeps` trennt `mtpsavetex` die Beschriftung von der eigentlichen Grafik. Es wird eine EPS-Datei mit den Grafikobjekten und den Positionierungsdaten der Textobjekte, sowie eine \LaTeX -Datei mit den eigentlichen Textobjekten erstellt.

Vorteile

- Fontgröße und -typ werden durch die Dokumenteinstellungen bestimmt und ergeben ein einheitliches Bild von Text und Grafikbeschriftung.
- Beschriftung ist von der Skalierung der Grafik unabhängig. Insbesondere entspricht die Fontqualität der des Dokuments.
- Beschriftung darf \LaTeX -Makros des Dokuments beinhalten und kann so nachträglich modifiziert werden.

Nachteile

- Benötigt das Paket `mtp.sty`.
- Feste Position der Beschriftung führt möglicherweise zu unästhetischer Grafik bei extremer Skalierung.

Insgesamt überwiegen die Vorteile, weshalb die Verwendung von `mtpsavetex` empfohlen wird.

6.2 Grafiken einbinden

Für das Einbinden von (MTP-)Grafiken wird in \LaTeX ein Paket benötigt, welches die entsprechenden Funktionen zur Verfügung stellt. Die Wahl des Pakets hängt dabei von der verwendeten MTP-Save-Funktion ab. Prinzipiell können durch `mtpsaveeps` erzeugte Grafiken mit einem \LaTeX -Standardpaket eingebunden werden. Es wird jedoch die Verwendung des `graphicx`-Pakets [CR99] empfohlen. Bei Verwendung des Befehls `mtpsavetex` können die Grafiken nur mit dem MTP-Paket eingebunden werden. In den nächsten Abschnitten werden die genannten Pakete und ihre Befehle zum Einbinden der Grafiken beschrieben.

6.2.1 Das Paket `graphicx.sty`

Das `graphicx`-Paket [CR99] ist der derzeitige Standard für das Einbinden von (EPS-)Grafiken¹. Es stellt eine Erweiterung des ebenfalls in [CR99] dokumentierten `graphics`-Pakets dar. Da das MTP-Paket intern `graphicx` verwendet, wird der Befehl zum Einbinden von Grafiken nun ausführlich beschrieben.

6.2.2 `includegraphics`

Einbinden von EPS-Grafiken

Syntax:

```
\includegraphics[⟨key val list⟩]{⟨file⟩}
\includegraphics*[⟨key val list⟩]{⟨file⟩}
```

Beschreibung: Fügt die Grafikdatei *file* in das Dokument ein. Dabei ist der Dateiname vollständig, also inklusive Endung anzugeben. Bei der `*`-Variante wird die Grafik bei mehrspaltigen Dokumenten über alle Spalten hinweg eingefügt (sonst in der laufenden Spalte). *key val list* ist eine durch Kommas getrennte Liste mit Einträgen vom Typ **Schlüsselwort=Wert**. Hier eine Auswahl der wichtigsten Schlüsselwörter:

bb: gibt die Bounding-Box der Grafik an. Benötigt werden vier, durch Leerstellen getrennte Wert.

hiresbb: Boolescher Wert ob `%HighResBoundingBox` anstelle von `%BoundingBox` verwendet werden soll. Erlaubte Werte sind `true` und `false`.

angle: Gradzahl um welche die Grafik zu drehen ist.

origin: Ursprung für die Rotation. Erlaubt sind bis zu zwei der üblichen Alignment-Zeichen `lrcbtbB`.

width: Gewünschte Breite der Grafik.

height: Gewünschte Höhe der Grafik.

totalheight: Spezifiziert die absolute Höhe der Grafik. Ist zum Beispiel bei Rotationen sehr nützlich.

keepaspectratio: Boolescher Wert. Sofern `true` gesetzt ist, wird bei Angabe von `width` und `height` die Grafik so skaliert, dass keine der Abmessungen überschritten wird.

scale: Skaliert die Grafik um den gegebenen Faktor.

¹Da das `epsfig`-Paket nur ein Wrapper um `graphicx` darstellt und nicht alle Parameter unterstützt, wird die Verwendung von `epsfig` nicht empfohlen.

clip: Boolescher Wert, der angibt ob die Grafik auf die Bounding Box beschnitten werden soll (ermöglicht das Zoomen einer Grafik).

draft: Boolescher Wert, der angibt ob die Grafik im Entwurf-Modus dargestellt werden soll.

6.2.3 Das Paket `mtp.sty`

Grafiken, welche mit `mtpsavetex` abgespeichert wurden, müssen mit dem Paket `mtp.sty` eingebunden werden. Hierzu ist in der Präambel des Dokuments folgende Zeile einzufügen:

```
\usepackage[<options>]{mtp}
```

Da das MTP-Paket intern `graphicx` zum Einbinden der Grafiken verwendet, stellt es dieselben Optionen zur Verfügung. Erlaubt sind für *options* also Treiberoptionen (wie z.B. `dvips`), `draft`, `final`, `hiderotate`, `hidescale` und `hiresbb`.

Für die Umsetzung der Beschriftungen wird das PSfrag-Paket [GC98] verwendet. Dieses basiert auf einem zweistufigen Verfahren. Zunächst wird durch \LaTeX die Beschriftung erzeugt und anschließend bei der Umwandlung vom \LaTeX eigenen DVI-Format nach PostScript in der Grafik positioniert. Somit entsteht das fertige Dokument im Gegensatz zu \LaTeX erst durch die Verwendung eines DVI-zu-PS-Konverters. Dies hat auch zur Folge, dass in der DVI-Datei die Beschriftung noch neben und nicht in der Grafik steht (Überschrift: PSfrag replacements). Abbildung 6.1 zeigt eine Screencopy des DVI-Previews der Alignment-Tabelle der Funktion `mtp\text` (siehe Seite 40).

Mit `\mtpincludegraphics` können MTP-Grafiken in \LaTeX -Dokumente eingebunden werden. Dieser Befehl wird von dem Paket `mtp.sty` zur Verfügung gestellt.

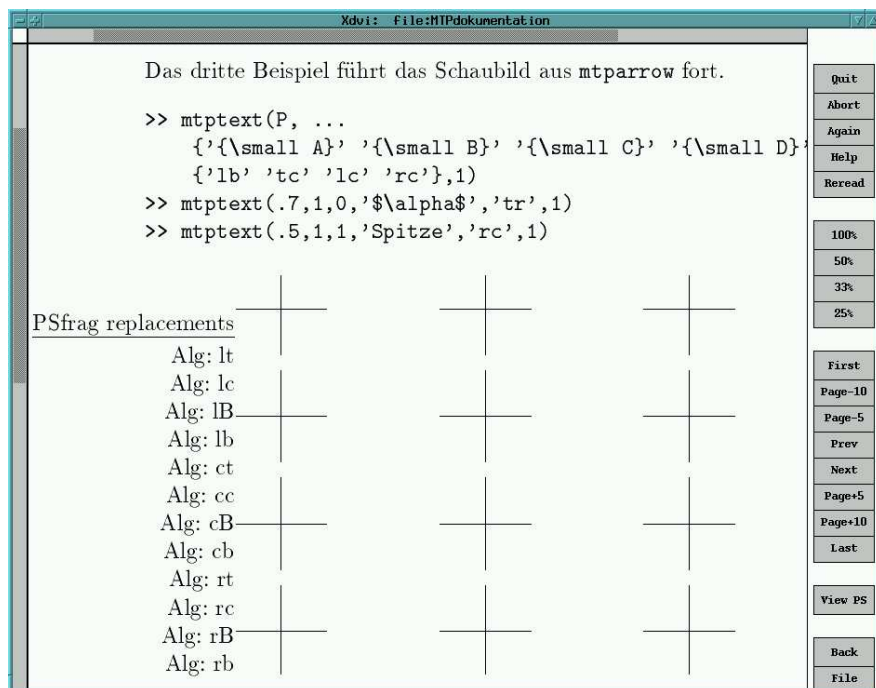


Abbildung 6.1: DVI-Preview von Textobjekten

6.2.4 mtpincludegraphics

L^AT_EX-Befehl zum Einbinden von MTP-Grafiken.

Syntax:

```
\mtpincludegraphics[⟨key val list⟩]{⟨file⟩}
```

Beschreibung: Bindet die MTP-Grafik *file* ein. Im Gegensatz zu dem Befehl `\includegraphics` des `graphicx`-Pakets darf hier jedoch keine Endung angegeben werden. Die Optionen entsprechen aber genau denen des Befehls `\includegraphics` (siehe Seite 64).

Beispiel: Ausgehend von einer mit `mtpsavetex('bild')` abgespeicherten Grafik, zeigt der nachfolgende L^AT_EX-Code ein vollständiges Dokument in das die Grafik eingebunden wird.

```
\documentclass[12pt,a4paper]{article}
\usepackage{mtp}
\begin{document}
  Eine mit MTP erzeugte Grafik:
  \begin{center}
    \mtpincludegraphics[width=.7\linewidth]{bild}
  \end{center}
  Die Breite des Grafik wurde auf 70\% der Zeilenbreite skaliert.
\end{document}
```

Kapitel 7

Problembehandlung

7.1 Fehlerhafte EPS-Grafiken

7.1.1 Falsche Bounding-Boxen

Die EPS-Treiber von MATLAB erzeugen zuweilen falsche Bounding-Boxen, die MTP zu korrigieren versucht. Hierzu stehen die folgenden drei Verfahren zur Wahl:

`gsbb`: Berechnung der korrekten Bounding-Box mit Hilfe von Ghostscript

`gs`: Konvertierung in das EPS-Format mit Hilfe von Ghostscript

`ps2epsi`: Konvertierung in das EPS-Format mit Hilfe der Funktion `ps2epsi`

Das zu verwendende Verfahren wird in der Variablen `MTPtemp.EPS.converter` in der Datei `mtpinitrc.m` festgelegt. Voreinstellung ist `gsbb`. Sollte dies nicht die gewünschten Ergebnisse liefern, kann eines der beiden anderen Verfahren gewählt werden.

7.1.2 Zusätzliche Linien in der Grafik

Die Farbfassungen der EPS-Treiber (`epsc`, `epsc2`) in MATLAB 6.1 erzeugen bei der Verwendung von `mtpmarker` ab dem zweiten Aufruf von `print` zusätzliche Linien in der Grafik. Vor dem Abspeichern der MTP-Grafik mit `mtpsaveeps` sollte daher `mtpshow` nicht aufgerufen werden, da diese Funktion wiederum `mtpsaveeps` verwendet und der zweite Aufruf eine fehlerhafte Grafik erzeugt.

Alternativ können statt der Farbfassungen auch die Monochromfassungen der EPS-Treiber (`eps`, `eps2`) verwendet werden, bei denen diese Probleme nicht auftreten. Die Wahl des EPS-Treibers erfolgt in der Datei `mtpinitrc.m` durch setzen der Variablen `MTPtemp.EPS.printdevice`.

Anhang A

Interna

A.1 Interne Funktionen

Nachfolgend werden die internen Funktionen des MTP-Pakets beschrieben. Dieser Abschnitt enthält keine für die Benutzer des MTP-Paketes relevante Funktionen, sondern dient ausschließlich der Dokumentation des MTP-Projekts.

A.1.1 mtpinternalaliases

Liefert alle Aliase zurück, die vom MTP-System bereitgestellt werden.

Syntax:

```
MTPlist=mtpinternalaliases
```

Beschreibung: `mtpinternalaliases` liefert eine Struct zurück, deren Feldnamen aus den Namen derjenigen Funktionen abgeleitet sind, die das Alias-konzept verwenden. Abgesehen von den benutzerdefinierten Aliasen ist diese Struct identisch mit der globalen Variablen `MTP.ALIASLIST`. Siehe dazu auch Abschnitt A.2.

`mtpinternalaliases` wird ausschließlich von `mtpinit` aufgerufen. Um die Aliase aber klar von der konkreten Implementierung von `mtpinit` zu trennen, wurde darauf verzichtet, die Funktion als Subfunktion zu realisieren.

Siehe auch: `mtpinit`, `mtpinitrc`, Abschnitt 2.2

A.1.2 mtpinternalaxalignment

Liefert die Alignmentstrings für `mptexaxes` und `mtptitle`.

Syntax:

```
[xAlignment,yAlignment,zAlignment] =  
    mtpinternalaxalignment(CallingFunction)
```

Beschreibung: Liefert, abhängig von den aktuellen `view`-Winkeln, für `mptexaxes` und `mtptitle` die Strings `xAlignment`, `yAlignment` und `zAlignment` zurück, welche die Textausrichtung für die Achsenbeschriftung enthalten.

Beispiel: Ausgabe der Alignments für einen 2d-Fall

```
>> mtpinit  
>> set(gca,'axislocation','top')  
>> [xAlignment,yAlignment]=...  
>>          mtpinternalaxalignment('mptexaxes')
```

```
xAlignment='cb'  
yAlignment='rc'
```

Ausgabe für einen 3d-Fall

```
>> mtpinit  
>> view(40,30)  
>> [xAlignment,yAlignment,zAlignment]=...  
>>          mtpinternalaxalignment('mtptitle')
```

```
xAlignment='rt'  
yAlignment='lt'  
zAlignment='cb'
```

Siehe auch: `mptexaxes`, `mtptitle`.

A.1.3 mtpinternaldvi2eps

Wandelt eine dvi- in eine eps-Datei um.

Syntax:

```
returncode=mtpinternaldvi2eps(filename)
```

Beschreibung: Da `dvips -E` die Bounding-Box nicht korrekt berechnen kann, wird mit Hilfe von `ghostscript` eine EPS-Datei mit der korrekten Bounding-Box erstellt (`Device=epswrite`). `ghostscript` wiederum modifiziert jedoch die Grafik, weshalb die Bounding Box der `ghostscript`-Version in die von `dvips` erzeugte EPS-Datei kopiert wird, um eine nicht modifizierte Grafik mit einer passenden Bounding Box zu erhalten.

Sofern die Konvertierung erfolgreich war, ist `returncode=1`, sonst 0.

Durch Setzen des Flags

```
MTP.FLAGS.generate_alternative_eps_files=1;
```

werden zwei zusätzliche Varianten der EPS-Datei bereitgestellt: Die durch `ghostscript` erzeugte EPS-Datei (Endung `_gs.eps`) sowie eine EPS-Datei mit einer Bounding-Box im DIN A4-Format (Endung `_a4.eps`).

Siehe auch: `mtpinternalsave`.

A.1.4 mtpinternalget

Interpretation von Aliasen.

Syntax:

```
[options,message]=mtpinternalget(toindex,aliases,whichlist)
```

Beschreibung: `mtpinternalget` interpretiert die Aliase, die als Cell-Elemente in `aliases` übergeben wurden gemäß ihrer festgelegten Bedeutung. Als maßgebliche Aliasliste wird `whichlist` selbst genommen, falls `whichlist` eine Cell ist. Falls `whichlist` ein String wie z.B. `'mtpline'` ist, so wird die entsprechende Liste aus `global MTP.ALIASLIST` verwendet. Der erste Parameter sollte bei einem externen Aufruf stets `-1` sein. `mtpinternalget` ruft sich selbst rekursiv auf. Nur in diesem Fall ist `toindex` eine positive Zahl, die den Bereich der zulässigen Aliase einschränkt. Mit ihr wird verhindert, dass Aliase in ihrer Definition andere Aliase benutzen, die in der Definition erst weiter hinten folgen. Dies ist verboten um zirkuläre Definitionen zu vermeiden.

Zurückgeliefert werden in `options` die interpretierten Aliase, bzw. im Fall eines auftretenden Fehlers `-1`. Im Fehlerfall enthält `message` die Fehlermeldung, ansonsten den leeren String.

Beispiel: Am einfachsten lässt sich die Arbeitsweise von `mtpinternalget` anhand der folgenden "vollständigen" Beispiele verstehen.

```
function MTPlist=mtpinitrc
...
MTPlist.area={
  'red'      {{'EdgeColor' [1 0 0]} {'FaceColor' [1 0 0]}};
  'Red'      {{'EdgeColor' [1 0 0]} {'FaceColor' [1 0 0]}};
  'Green'    {{'EdgeColor' [0 1 0]} {'FaceColor' [0 1 0]}};
  'Redgreen' { 'Red' {'FaceColor' [0 1 0]}};
  'Colstr#,#,#' {{'EdgeColor' '[#1 #2 #3]'}
                {'FaceColor' '[#1 #2 #3]'}};
  'Col#,#,#'  {{'EdgeColor' '![#1 #2 #3]'}
                {'FaceColor' '![#1 #2 #3]'}};
  'Demo#'    {{'Demo#1' 'Demo#1'}};
  'Norekursion' {'Rekursion'};
  'Rekursion' {'Norekursion'};
  '#         ' {'Number' '!#1' '#1'};
};
```

Klären wir zuerst die Terminologie anhand der Definition von 'Redgreen'. 'Redgreen' ist der Aliasnamen. { 'Red' {'FaceColor' [0 1 0]}} ist die Aliasdefinition. 'FaceColor' ist ein Propertyname und [0 1 0] ist ein Propertyvalue.

Zuerst muss `mtpinit` aufgerufen werden.

```
>> [P,m]=mtpinternalget(-1,{'red'},'mtparea')
P = -1
m = 'Call mtpinit first.'
```

Aliase dürfen nicht doppelt definiert werden

```
>> mtpinit
>> [P,m]=mtpinternalget(-1,{'red'},'mtparea')
P = -1
m = 'Alias "red" not unique. Perhaps internal defined
    and in mtpinitrc.'
```

Strings werden durch die in function `mtpinternalaliases` und function `mtpinitrc` festgelegte Bedeutung interpretiert:

```
>> [P,m]=mtpinternalget(-1,{'Red'},'mtparea')
P = {'EdgeColor' [1 0 0]} {'FaceColor' [1 0 0]}
m = ''
```

Leerzeichen werden ignoriert:

```
>> [P,m]=mtpinternalget(-1,{'R e d'},'mtparea')
P = {'EdgeColor' [1 0 0]} {'FaceColor' [1 0 0]}
m = ''
```

Definitionen von Aliasen durch andere Aliase sind erlaubt:

```
>> [P,m]=mtpinternalget(-1,{'Redgreen'},'mtparea')
P = {'EdgeColor' [1 0 0]} {'FaceColor' [1 0 0]}
    {'FaceColor' [0 1 0]}
m = ''
```

Zahlen können in Aliasen variabel sein:

```
>> [P,m]=mtpinternalget(-1,{'Colstr1,.532,1.4e-1'},'mtparea')
P = {'EdgeColor' '[1 0.532 0.14]'}
    {'FaceColor' '[1 0.532 0.14]'}
m = ''
```

Wie bei `eval` können Ausdrücke interpretiert werden, wenn ein `'!`' das erste Zeichen ist. Beachte: Im Vergleich zum letzten Beispiel sind die Propertyvalues nun Matrizen und keine Strings mehr.

```
>> [P,m]=mtpinternalget(-1,{'Col1,.532,1.4e-1'},'mtparea')
P = {'EdgeColor' [1 0.532 0.14]}
     {'FaceColor' [1 0.532 0.14]}
m = ''
```

Das erste Argument einer Cell in einer Aliasdefinition bleibt grundsätzlich unverändert:

```
>> [P,m]=mtpinternalget(-1,{'Demo23'},'mtparea')
P = {'Demo#1' 'Demo23'}
m = ''
```

Aliase dürfen nur durch Aliase definiert sein, die “weiter oben” bereits eingeführt wurden:

```
>> [P,m]=mtpinternalget(-1,{'Norekursion'},'mtparea')
P = -1
m = 'Alias "Rekursion" used before defined.\n
     ->Alias "Norekursion" can not be interpreted.\n'
```

Cells als Parameter werden nicht interpretiert, sondern komplett ohne Prüfung auf Konformität übernommen:

```
>> [P,m]=mtpinternalget(-1,{'EdColor' [1 0]} ...
>>                               {'FaColor' [1 0 0]}},'mtparea')
P = {'EdColor' [1 0]} {'FaColor' [1 0 0]}
m = ''
```

Siehe auch: `mtpinternalaliases`, `mtpinit`, Aliaskonzept Abschnitt 2.2

A.1.5 mtpinternalmessage

Einheitliche Ausgabe von Meldungen der MTP-Funktionen.

Syntax:

```
mtpinternalmessage(callingfunction, messagetext, messagetype)
```

Beschreibung: Der Befehl `messagetext` dient zur Vereinheitlichung aller Meldungen der MTP-Funktionen. Beim Aufruf ist in `callingfunction` der Name jener Funktion zu übergeben, die `mtpinternalmessage` aufruft. Der eigentliche Text der Meldung steht in `messagetext` und der Typ der Meldung in `messagetype`. Gültige Werte für `messagetype` sind die Schlüsselwörter `'message'`, `'error'` und `'warning'`. Die daraus erzeugte Meldung hat den folgenden Aufbau:

```
['mtp ' messagetype ' (' callingfunction '): ' messagetext]
```

`mtpinternalmessage` bricht den Meldungstext ggf. um, so dass nicht mehr als 70 Zeichen pro Zeile ausgegeben werden. Der Umbruch erfolgt dabei nach den folgenden Regeln:

- enthält `messagetext` innerhalb der ersten 70 Zeichen den Umbruchstring `'\n'`, so wird bei diesem umgebrochen
- enthalten die ersten 70 Zeichen von `messagetext` kein `'\n'`, jedoch mindestens ein Leerzeichen, so wird bei jenem Leerzeichen umgebrochen, das den größten Index ≤ 70 in `messagetext` besitzt
- enthält `messagetext` weder ein `'\n'`, noch ein Leerzeichen unter den ersten 70 Zeichen, so wird nach dem 70sten Zeichen umgebrochen

Anschließend wird `messagetext` um den ausgegebenen Teil verkürzt und nach den obigen Kriterien solange ausgegeben, bis `messagetext` leer ist. Ab der zweiten Zeile erfolgt eine Einrückung.

Warnungen werden nur dann ausgegeben, wenn die `warning`-Funktion von MATLAB nicht den Wert `off` zurückliefert.

Beispiel: Ausgabe einer Meldung mit weniger als 70 Zeichen.

```
>> mtpinternalmessage('console', 'hello world', 'message');
mtp message (console): hello world
```

Angabe einer Warnung mit einem manuellen Umbruch mittels `\n`.

```
>> mtpinternalmessage('console', ['first warning line\n' ...  
>>                               'second warning line'], 'warning');  
mtp warning (console): first warning line  
                        second warning line
```

Ausgabe einer Fehlermeldung mit mehr als 70 Zeichen. Der Umbruch erfolgt automatisch bei der letzten Leerstelle mit dem Index ≤ 70 .

```
>> mtpinternalmessage('console', ['this is a long and useless ' ...  
>>                               'error message just to demonstrate automatic ' ...  
>>                               'linebreaks'], 'error');  
mtp error (console): this is a long and useless error message just to  
                    demonstrate automatic linebreaks
```

A.1.6 `mtpinternalplot`

Plotfunktion für `mtpline`, `mtploop`, `mtparea`, `mtpmarker`, `mtparrow`.

Syntax:

```
Handle=mtpinternalplot(callingfunction,defaults,args)
```

Beschreibung: `mtpinternalplot` übernimmt die komplette Arbeit der oben genannten Funktionen, deren Namen in `callingfunction` übergeben wird. Das jeweils entsprechende Grafikobjekt wird erzeugt und die durch `defaults` und `args` übergebenen Aliase werden gesetzt. Die Daten der Koordinaten werden als numeric-Elemente der Cell `args` übergeben.

Alle oben genannten Funktionen haben einen großen Kern, der allen gemeinsam ist. Deshalb, und um die Wartung des Gesamtpakets erträglich zu halten, wurde `mtpinternalplot` geschrieben. Alle nicht-gemeinsamen Teile sind gekennzeichnet durch die Verwendung des Switch-Befehl in der stets gleichen Form: `switch callingfunction`.

A.1.7 mtpinternalsave

Speicherfunktion für `mtpsavetex` und `mtpsaveeps`.

Syntax:

```
mtpinternalsave(CallingFunction,FileName)
mtpinternalsave(CallingFunction,FileName,FontSize,Alignment,
                mptextdistanceunit)
```

Beschreibung: `mtpinternalsave` übernimmt die komplette Arbeit der oben genannten Funktionen. Die aufrufende Funktion übergibt Ihren Namen durch den String `CallingFunction`.

Die gesamte Funktionalität von `mtpsavetex` und `mtpinternalsaveeps` erschöpft sich in der korrekten Übergabe der Parameter an `mtpinternalsave`. Ist diese beendet findet der weitere Programmablauf, wie bereits in der Dokumentation von `mtpsavetex` und `mtpsaveeps` beschrieben, innerhalb von `mtpinternalsave` statt.

Siehe auch: `mtpsaveeps`, `mtpsavetex`

A.1.8 mtpinternalswitch

Schaltet von Textpreviews auf Tags und umgekehrt.

Syntax:

```
mtpinternalswitch(switchto)
mtpinternalswitch(switchto,class)
mtpinternalswitch(switchto,'mtpptext',indecex)
```

Beschreibung: Diese Funktion wird von `mtpptext`, `mtpptexaxes`, `mtpptitle`, und `mtpinternalsave` verwendet.

`mtpinternalswitch(switchto)` schaltet alle Tags zu Textpreviews um, falls `switchto` auf `'previews'` gesetzt ist. Umgekehrt werden alle Textpreviews in Tags umgewandelt, falls `switchto` als `'tags'` übergeben wird.

`mtpinternalswitch(switchto,class)` wirkt sich nur auf eine bestimmte Klasse von MTP-Objekten aus. mögliche Werte für `class` sind `'mtpptext'`, `'mtpptexaxes'` und `'mtpptitle'`.

`mtpinternalswitch(switchto,'mtpptext',indecex)` schaltet die `mtpptext`-Objekte mit den gegebenen `indecex` von `MTP.TXT` um.

Beispiel:

```
>> mtpinit
>> mtpptexaxes
>> mtpptitle('Example','x-axis','y-axis')
>> mtpptext([0.2,0.5,0.8;0.6,0.1,0.4], ...
            {'Hello' 'beautiful' 'world!'})
>> mtpinternalswitch('tags')
>> mtpinternalswitch('previews')
```

Siehe auch: `mtpptext`, `mtpptexaxes`, `mtpptitle`, `mtpinternalsave`

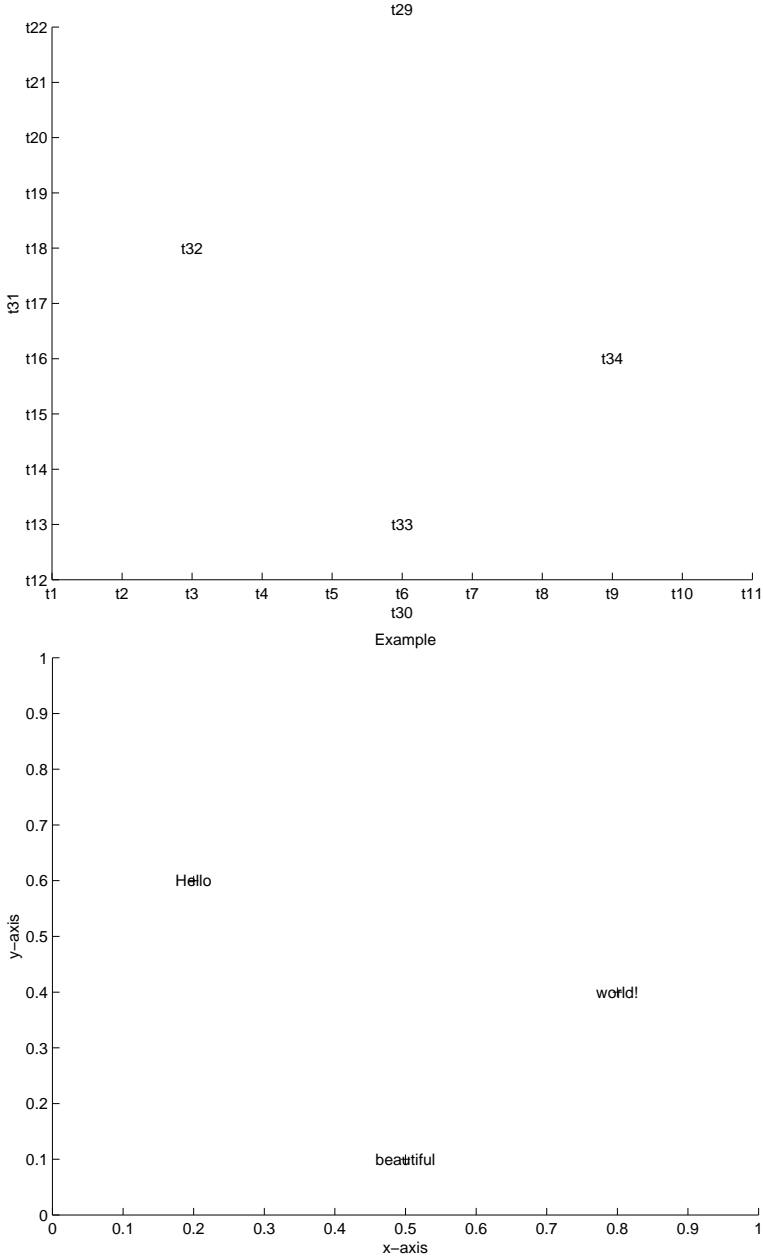


Abbildung A.1: Tags (oben) und Textpreviews (unten)

A.1.9 mtpinternalwritevariable

Umwandlung des Inhalts einer beliebigen MATLAB-Variablen in einen String.

Syntax:

```
string=mtpinternalwritevariable(variable)
string=mtpinternalwritevariable(variable,maxsize)
string=mtpinternalwritevariable(variable,maxsize,newrowmarker)
```

Beschreibung: `mtpinternalwritevariable(variable)` wandelt den Inhalt einer beliebigen MATLAB-Variablen in einen String um. Bei Matrizen und Cells werden nur die Dimensionsgrößen geschrieben.

`mtpinternalwritevariable(variable,maxsize)` schreibt bei Matrizen und Cells nur dann die Dimensionsgrößen, wenn die Zahl der Dimensionen größer als zwei ist, bzw. wenn die Zahl der Elemente größer als in `maxsize` spezifiziert ist. Im anderen Fall werden die Einträge tatsächlich ausgeschrieben. `maxsize` ist ein 2-Vektor. Der erste Eintrag bestimmt die Maximalzahl für Matrizen, der zweite Eintrag die Maximalzahl für Cells.

`mtpinternalwritevariable(variable,maxsize,newrowmarker)` bestimmt, welche Zeichen beim Ausschreiben von Matrizen bzw. Cells beim Beginn einer neuen Zeile eingefügt werden sollen. `newrowmarker` ist eine zweielementige Cell. Der erste Eintrag bestimmt die Zeilenumbruch-Sequenz für Matrizen, der zweite Eintrag die Zeilenumbruch-Sequenz für Cells.

Default-Werte:

```
maxsize      : [0,0]
newrowmarker : {';' ' ';'}'
```

Beispiel:

```
>> mtpinternalwritevariable({1 'hello'})
'1x2-cell'
>> mtpinternalwritevariable({ones(2,3) ; ones(4,6) ; ...
>>                          {1 2 3 4}},[6,3])
' {[1 1 1;1 1 1];4x6-matrix;1x4-cell}'
>> mtpinternalwritevariable({ones(2,3) ; ones(4,6) ; ...
>>                          {1 2 3 4}},[6,3],{' ; ' ';'\n'})
' {[1 1 1 ; 1 1 1];
4x6-matrix;
1x4-cell}'
```

A.2 Die globalen Variablen des MTP-Pakets

Nachfolgend werden alle globalen Variablen des MTP-Pakets aufgelistet und ihre Bedeutung sowie Struktur erläutert:

MTP.ALIASLIST: Enthält die Alias-Definitionen der einzelnen MTP-Befehle.

`MTP.ALIASLIST` ist eine Struct-Variable, deren Feldnamen aus den Namen jener MTP-Funktionen abgeleitet sind, die das Aliaskonzept verwenden. Jedes Feld enthält wiederum in einer $n \times 2$ -Cell die Definitionen der Aliase (siehe auch Abschnitt 2.2).

MTP.ARROWHANDLES: $2 \times n$ -Cell von Grafikhandles der mit Hilfe von `mtparrow` erstellten Pfeile. Jede Spalte repräsentiert einen Pfeil. In der ersten Zeile der Cell stehen die Handles der Pfeilspitzen, in der zweiten Zeile die Handles der Pfeilrumpfe.

MTP.DEFAULTTEXTINTERPRETER: In MATLAB ist standardmäßig der Textinterpreter `tex` aktiv. Da dieser \LaTeX -Befehle nur eingeschränkt interpretieren kann, wird der Textinterpreter von MTP ausgeschaltet. Zuvor wird der aktive Textinterpreter in dieser Variablen gespeichert und durch `mtpclose` wieder auf seinen ursprünglichen Wert zurückgesetzt.

MTP.EVENT: Enthält die Befehle, die im Fall eines Fehlers bzw. einer Warnung ausgeführt werden sollen. `MTP.EVENT` ist eine Struct mit den Feldern:

`error`: Code der im Fehlerfall auszuführen ist.

`warning`: Code der im Fall einer Warnung auszuführen ist.

MTP.EXTERNCALL: Enthält die Befehlssequenzen für externe Programme. Benötigt werden:

`latex`: Ein \LaTeX 2e-Compiler.

`dvips`: Konvertiert dvi- in ps-Dateien.

`psview`: Anzeigeprogramm für (e)ps-Dateien.

`gs`: Der Aladin ghostscript Interpreter.

MTP.FLAGS: Enthält Steuerflags. Sind diese $\neq 0$, so wird die genannte Aktion ausgeführt, sonst unterbleibt die Ausführung. Vorgesehen sind die folgenden Flags:

`generate_alternative_eps_files`: Steuert die Erzeugung alternativer EPS-Varianten bei `mtpsaveeps` und `mtpshow`.

MTP.LATEX: Enthält grundsätzliche, nicht durch Parameter verstellbare Dokumenteigenschaften für \LaTeX . Diese Variable wird von `mtpsaveeps` zur Erstellung der \LaTeX -Datei benötigt.

`documentclass`: enthält die Dokumentklasse.

`fontsize`: Dokumentschriftgröße für \LaTeX .

`paperformat`: enthält das Papierformat.

`preamblestring`: enthält den Teil eines \LaTeX -Dokumentkopfes, der die verwendeten Pakete und den Pagestyle festlegt.

`MTP.NORMALSIZE`: Die nachfolgend aufgeführten Felder dieser Struct-Variablen enthalten die in `mtpinit` gesetzten Standardgrößen verschiedener Grafikobjekte.

`arrow`: Struct-Variable mit Feldern für die Längen der Pfeilspitzen (`length`) und dem Öffnungswinkel der Pfeilspitzen (`headangle`).

`linewidth`: Standard-Linienbreite.

`markersize`: Standardgröße der Marker.

`mptextdistanceunit`: \LaTeX -Maßeinheit. für die Abstände des Textes zu ihren Bezugspunkten.

`previewfontsize`: Schriftgröße für die Orientierungsobjekte im MATLAB-Grafikfenster.

`MTP.SHOWFILE`: Enthält den Namen der temporären EPS-Datei, die von `mtpshow` zur Anzeige der vollständigen Grafik erstellt wird.

`MTP.TXT`: Enthält alle zur Ausgabe von \LaTeX -Text in MATLAB-Grafiken benötigte Daten. Die einzelnen Felder dieser Struct-Variablen sind:

`label`: Speichert den ursprünglich eingegebenen \LaTeX -Quellcode für `mptitle` und `mptext`.

`latex`: \LaTeX -String.

`type`: Kennzeichner von welcher MTP-Funktion der Text gesetzt wurde.

`tagnumber`: Zähler für die `psfrag`-Tags.

`texthandle`: Handle für den Previewtext von `mptext`.

`markerhandle`: Handle des Previewmarkers für ein `mptext`-Objekt.

`taghandle`: Handle für das Tag.

A.3 Richtlinien für die Programmierung

Jeder Programmierer entwickelt im Laufe der Zeit seinen individuellen Programmierstil. Dieser reflektiert die Summe seines Wissens, seiner Erfahrung und seines Könnens. Hieraus ergeben sich bei Projekten, in die mehr als ein Programmierer involviert sind, unweigerlich Konflikte zwischen den einzelnen Stilrichtungen. Um diese Reibungseffekte zu minimieren wurden für das MTP-Projekt die folgenden verbindliche Richtlinien erstellt.

Allgemeines

- Jede Zeile darf höchstens einen MATLAB-Befehl enthalten.
- Eine Zeile darf höchstens 80 Zeichen lang sein. Zeilentrennungen mittels '...' sind syntaktisch sinnvoll durchzuführen.
- Es sind möglichst "sprechende" Variablennamen (z.B. `alias` statt `a` zu verwenden).
- Der Code ist ausreichend zu kommentieren.

Aufbau des Programmkopfes

Da der Hilfetext einer MATLAB-Funktion Bestandteil des Programmkopfes ist und somit entscheidende Informationen für Benutzer und Programmierer bereitstellt, ist es wichtig, diesen zu vereinheitlichen.

Die Funktionsdatei enthält in der ersten Zeile stets die Funktionendeklaration vom Typ

```
function [out1,out2,...]=functionname(arg1,arg2,...)
```

Hierauf folgt unmittelbar der durch `help function` anzeigbare Hilfetext. Dieser besteht in der ersten Zeile aus dem Funktionsnamen und einer Kurzbeschreibung des Befehls, sowie einer darauffolgenden kommentierten Leerzeile, also

```
% MTPFUNCTION  Kurzbeschreibung.  
%
```

Im Falle interner MTP-Funktionen ist die Form

```
% MTPFUNCTION  (mtp internal function) Kurzbeschreibung.  
%
```

zu verwenden. Da der äußerst hilfreiche Befehl `lookfor` nur die Kurzbeschreibung durchsucht, ist darauf zu achten, dass diese alle relevanten Schlagwörter zur Charakterisierung der Funktion enthält.

Bei internen Funktionen schließt sich unmittelbar der Standardtextblock

```
% This is an internal mtp function. Please read the programmers  
% section of the mtp documentation for further informations about  
% this function or the user sections to learn more about the mtp  
% project.  
%  
% See also MTPINIT (and all other mtp functions)
```

an. Der nachfolgende Teil des Hilfetextes ist in diesem Fall durch eine Leerzeile abzutrennen, da er kein Bestandteil des anzuzeigenden Hilfetextes ist. Zur besseren Dokumentation besitzt der nachfolgende Teil aber genau den gleichen Aufbau wie der Hilfetext von MTP-Benutzerfunktionen.

Der Aufbau des in Englisch zu verfassenden Hilfetextes lehnt sich stark an den Stil der MATLAB-Hilfetexte an. Demzufolge sind die Namen aller Funktionen im Hilfetext in Großbuchstaben zu schreiben. Darüber hinaus hat der Hilfetext den folgenden Aufbau:

- Liste aller Aufrufmöglichkeiten der Funktion (je eine Variante pro Zeile, inkrementelle Parameterliste) gefolgt von einer Leerzeile.
- Auflistung jeder oben genannten Aufrufmöglichkeit mit erläuterndem Text. Dabei müssen nur die zusätzlichen Parameter beschrieben werden. Die Erläuterung beginnt (sofern genügend Platz vorhanden) in der gleichen Zeile wie die Aufrufvariante. Einzelne Varianten sind durch Leerzeilen zu trennen.
- Sofern Defaultwerte zur Verfügung gestellt werden, sind diese nun nach der Überschrift `Defaults:` aufzulisten. Jeder Defaultwert steht in einer eigenen Zeile und ist zwei Leerzeichen eingerückt.
- Abgetrennt durch eine Leerzeile folgen die Ausführungsbeispiele.
- Schließlich folgt die Siehe-Auch-Zeile eingeleitet durch `See also` (ohne Doppelpunkt) gefolgt von der Liste der groß geschriebenen Funktionennamen auf die an dieser Stelle verwiesen werden soll.

Hier ist (bei Funktionen für den MTP-User) der Hilfetext zu Ende, weshalb eine nicht kommentierte Leerzeile einzufügen ist um die nachfolgenden Informationen zu Autor(en) und Revision abzutrennen. Diese bestehen aus den folgenden drei Zeilen:

```
% Author: Vorname Nachname (volle E-Mail-Adresse)
% Modified by: -
% RCS-revision: $ID:$ (hier ist Id statt ID zu verwenden)
```

in der dritten Zeile ist jedoch `Id` statt `ID` zu setzen. Der daraus resultierende String `Id:` wird durch die RCS-Verwaltung mit der jeweiligen Revisionsnummer versehen (was ohne die Modifikation auch bei dieser Dokumentation geschehen würde).

Die Zeile der Revisionsinformationen stellt zugleich die einzige erlaubte Sprachgrenze dar. Alles was darüber steht *muss* Englisch sein, alle nachfolgenden Kommentare dürfen jedoch entweder komplett in Englisch oder komplett in Deutsch gehalten sein. Im englischen Text ist das Akronym MTP klein, im deutschen Text groß zu schreiben.

Zusammenfassend hier noch einmal die Musterköpfe der beiden Funktionstypen.

Achtung: Groß-/Kleinschreibung, Leerstellen und Leerzeilen sind relevant!

Musterkopf einer MTP-Benutzerfunktion

```

1  function [out1,out2,...]=mtpfunction(arg1,arg2,...)
2  %_MTPFUNCTION_Kurzbeschreibung.
3  %
4  %_MTPFUNCTION
5  %_MTPFUNCTION(Parameter1,Parameter2)
6  %_MTPFUNCTION(Parameter1,Parameter2,Parameter3)
7  %
8  %_MTPFUNCTION_Description.
9  %
10 %_MTPFUNCTION(Parameter1,Parameter2)_Description.
11 %
12 %_MTPFUNCTION(Parameter1,Parameter2,Parameter3)_Description.
13 %_Folgende_Beschreibungszeilen_werden_nicht_eingerueckt.
14 %
15 %_Defaults:
16 %_Parameter1_=0
17 %_Parameter2_=20
18 %_Parameter3_=25
19 %
20 %_Example:
21 %_codeline1
22 %_codeline2
23 %
24 %_See_also_MTPLINE,_MTPLOOP_(alphabetisch_geordnet)
25
26 %_Author:_Vorname_Nachname_(volle_E-Mail-Adresse)
27 %_Modified_by:_
28 %_RCS-revision:_$ID:$_(hier_ist_Id_statt_ID_zu_verwenden)
29

```

Musterkopf einer internen MTP-Funktion

```

1  function [out1,out2,...]=mtpfunction(arg1,arg2,...)
2  %_MTPFUNCTION_(mtp_internal_function)_Kurzbeschreibung.
3  %
4  %_This_is_an_internal_mtp_function._Please_read_the_programmers
5  %_section_of_the_mtp_documentation_for_further_information_about
6  %_this_function_or_the_user_sections_to_learn_more_about_the_mtp
7  %_project.
8  %
9  %_See_also_MTPINIT_(and_all_other_mtp_functions)

```

```

10
11  %_MTPFUNCTION
12  %_MTPFUNCTION(Parameter1,Parameter2)
13  %_MTPFUNCTION(Parameter1,Parameter2,Parameter3)
14  %
15  %_MTPFUNCTION_Description.
16  %
17  %_MTPFUNCTION(Parameter1,Parameter2)_Description.
18  %
19  %_MTPFUNCTION(Parameter1,Parameter2,Parameter3)_Description.
20  %_Folgende_Beschreibungszeilen_werden_nicht_eingerueckt.
21  %
22  %_Defaults:
23  %_Parameter1=_0
24  %_Parameter2=_20
25  %_Parameter3=_25
26  %
27  %_Example:
28  %_codeline1
29  %_codeline2
30  %
31  %_See_also_MTPLINE,_MTPLOOP.__(alphabetisch_geordnet)
32
33  %_Author:_Vorname_Nachname_(volle_E-Mail-Adresse)
34  %_Modified_by:_-
35  %_RCS-revision:_$Id:_$(ohne_Leerstellen_zwischen_$_und_$)
36

```

Gliederung des Quellcodes und Kommentare

Programme werden durch die logische Gliederung der Abläufe übersichtlicher und besser zu warten. Deshalb bietet sich die Gliederung in logische Blöcke und Unterblöcke an, deren Anfang und Ende durch Kommentarzeilen markiert werden. Diese haben eine Länge von 70 Zeichen und sind von folgender Gestalt:

```

%> Kurzbeschreibung des Blocks-----
...
%>> Kurzbeschreibung des Unterblocks-----
...
%<< Kurzbeschreibung des Unterblocks-----
...
%< Kurzbeschreibung des Blocks-----

```

Größerzeichen markieren einen beginnenden Block, Kleinerzeichen den Abschluß

eines Blocks. Die Anzahl der Größer-/Kleinerzeichen spiegelt die Gliederungsebene des Blocks wieder. So haben Unterblöcke mehr Gliederungszeichen als Hauptblöcke (welche nur ein Gliederungszeichen besitzen). Die Gliederungszeilen enthalten nach den Größer-/Kleinerzeichen genau ein Leerzeichen und dann die kurze Beschreibung des Blocks. Anschließend werden diese Kommentarzeilen zur besseren optischen Abgrenzung bis zum 70sten Zeichen mit Minuszeichen aufgefüllt. Ob die Kurzbeschreibung beim Schließen des Blocks wiederholt wird, liegt im Ermessen des Kommentators. Dies ist insbesondere dann sinnvoll, wenn der Block eine Größe erreicht, die ihn unübersichtlich macht.

Neben den Blockkommentaren, die einen ersten Eindruck vom Ablauf einer Funktion vermitteln sollen, ist der Quelltext im Detail ausreichend zu kommentieren. Klare Regeln hierfür festzulegen ist unmöglich, weshalb nur einige Richtlinien genannt werden: Kommentare ...

- müssen knapp abgefasst und prägnant sein (keine Romane, jedoch mehr als bloß Stichwörter, kurze einfache Sätze),
- müssen Konzepte und Strukturen erläutern,
- dürfen nicht aus dem Code trivial ersichtliche Anweisungen wiederholen,
- müssen vor dem zu erläuternden Code stehen und diesen sinnvoll ergänzen. (Die einzige Ausnahme dieser Regel bilden die zu Verzweigungsstrukturen wie `if`, `switch`, `while`, `for` gehörenden `end`-Zeilen. Hier darf in derselben Zeile der Kopf der Verzweigungsstruktur als Kommentar wiederholt werden.)

Hinweise zur Implementierung

Defaultwerte: Viele Parameter der MTP-Funktionen sind optional und haben Defaultwerte. Diese Defaultwerte sind im Hilfetext stets explizit anzugeben. Sofern möglich sollten alle Eingabeparameter für die ein Defaultwert existiert den Wert `NaN` als explizite Wahl dieses Defaults zulassen.

Fehlermeldungen: Fehlermeldungen sind stets mit Hilfe der internen Funktion `mtpinternalmessage` auszugeben. Die Fehlertexte müssen auch für unerfahrene Nutzer möglichst verständlich gehalten werden. Zudem muss aus ihnen hervorgehen, welche Eingabe an welcher Stelle zu dem Fehler geführt hat. Hierzu sind fehlerhafte Eingaben durch Hochkommas kenntlich zu machen und in die Fehlermeldung zu integrieren.

Internen Fehlermeldungen, wie zum Beispiel beim Aufruf interner Funktionen mit falschen oder fehlenden Parametern, ist der Text “INTERNAL ERROR:” voranzustellen.

Globale Variablen: sind grundsätzlich in `mtpinit` zu deklarieren und zu definieren. Für ihre Namen werden neben Sonderzeichen ausschließlich Großbuchstaben verwendet.

Typabfragen: Bei Abfragen und Vergleichen darf `==` bzw. `!=` nur dann verwendet werden, wenn in MATLAB für diesen Fall keine spezielle Hilfsfunktion wie `isempty`, `isnan`, `strcmp`, `isnumeric`, `ischar`, usw. vorgesehen ist.

Argumente: Da die Verwendung von `varargin` meist zu schlechter les- und wartbarem Code führt, ist dessen Verwendung nur dort erlaubt, wo nicht von festen Argumentlisten ausgegangen werden kann (z.B. bei der Verwendung von Aliasen). Bei festen Argumentlisten ist bei der Argumentkontrolle die Abfrage `exist('Variable','var')` der Verwendung von `nargin` vorzuziehen, da sie eine problemlose Umstellung der Argumentenliste in der Funktionsdeklaration ermöglicht.

A.4 Richtlinien zur Dokumentation

Neben dem Code trägt auch die Qualität der Dokumentation wesentlich zur Bedienbarkeit eines Programmes bei, weshalb jede MATLAB-Funktion des MTP-Paketes genau zu dokumentieren ist. Um die Einheitlichkeit der Dokumentation zu gewährleisten, gibt es die Dokumentationsvorlage `matlabf_template.tex`, welche entsprechend auszufüllen ist. Vorlage war auch hier der Stil der Dokumentationen von MathWorks:

Befehl `matlabfname`: Diesem Befehl ist der Name der zu dokumentierenden MATLAB-Funktion als Argument zu übergeben. Die Groß-/Kleinschreibung muss dabei der Groß-/Kleinschreibung des zugehörigen `*.m`-Dateinamens entsprechen (in der Regel also alles klein geschrieben).

Befehl `FEauthor`: Im ersten Parameterfeld ist der volle Name des Autors der Funktion einzutragen, im zweiten Feld das Erstellungsdatum der Dokumentation.

Umgebung `shortdescription`: Muss in Analogie zur Lookfor-Zeile in `*.m`-Dateien eine prägnante Kurzbeschreibung der Funktionalität der zu dokumentierenden Funktion enthalten.

Umgebung `syntax`: Hier sind die Syntaxzeilen aus dem Hilfetext des Funktionskopfes zu wiederholen. Die Groß-/Kleinschreibung orientiert sich dabei am Dateinamen der zugehörigen Funktion. Pro Zeile darf nur eine Aufrufmöglichkeit stehen (Zeilenschaltung mit `\\`).

Umgebung `longdescription`: In diesem Abschnitt werden die einzelnen Aufrufmöglichkeiten entsprechend zu ihrer Nennung im Syntaxblock dokumentiert. Sofern zuvor mehrere Aufrufmöglichkeiten genannt wurden, erfolgt die Beschreibung mit Hilfe einer `description`-Umgebung. Die Beschreibung einer Syntaxzeile wird jeweils durch ein `\item` eingeleitet (eine inkrementelle Dokumentation der Parameter ist erlaubt). In diesem Fall muss nach `\begin{longdescription}` der Befehl `\skipfirstline` eingesetzt werden.

Sofern im Syntax-Abschnitt nur eine Aufrufmöglichkeit genannt wurde, entfallen die `description`-Umgebung und der `\skipfirstline`-Befehl.

Umgebung `example`: Sollte ein möglichst instruktives und umfassendes Beispiel für die Verwendung des Befehls enthalten. Eingaben auf der MATLAB-Kommandozeile sind von den zugehörigen Ausgaben der Funktionen durch ein Voranstellen des Prompts „>>“ zu kennzeichnen. Bei Grafikfunktionen sollte auch ein entsprechendes Bild an dieser Stelle eingebunden werden. Dabei ist eine `figure`-Umgebung zu verwenden, in deren `\caption` der entsprechende Funktionsnamen zu nennen ist.

Umgebung `seealso`: Enthält Verweise auf die relevanten MTP- oder MATLAB-Funktionen (Groß-/Kleinschreibung beachten) und – sofern sinnvoll – Hinweise auf relevante Abschnitte der Dokumentation.

Literaturverzeichnis

- [CR99] D.P. Carlisle und S.P.Q. Rahtz. The graphicx Package. Technical report, 1999.
- [GC98] Michael C. Grant und David Carlisle. The PSfrag System, Version 3. Technical report, Stanford University, 1998.
- [Kop96] Helmut Kopka. *L^AT_EX (Band I – Einführung)*. Addison-Wesley, 2. Auflage, 1996.
- [Mat99] MathWorks. *Using Matlab, Version 5*. MathWorks Inc., 1999.
- [Now98] Dietrich Nowottny. PLS-Grafik und ihre Einbindung in L^AT_EX, Version 3.4. Technical report, Universität Stuttgart, 1998.