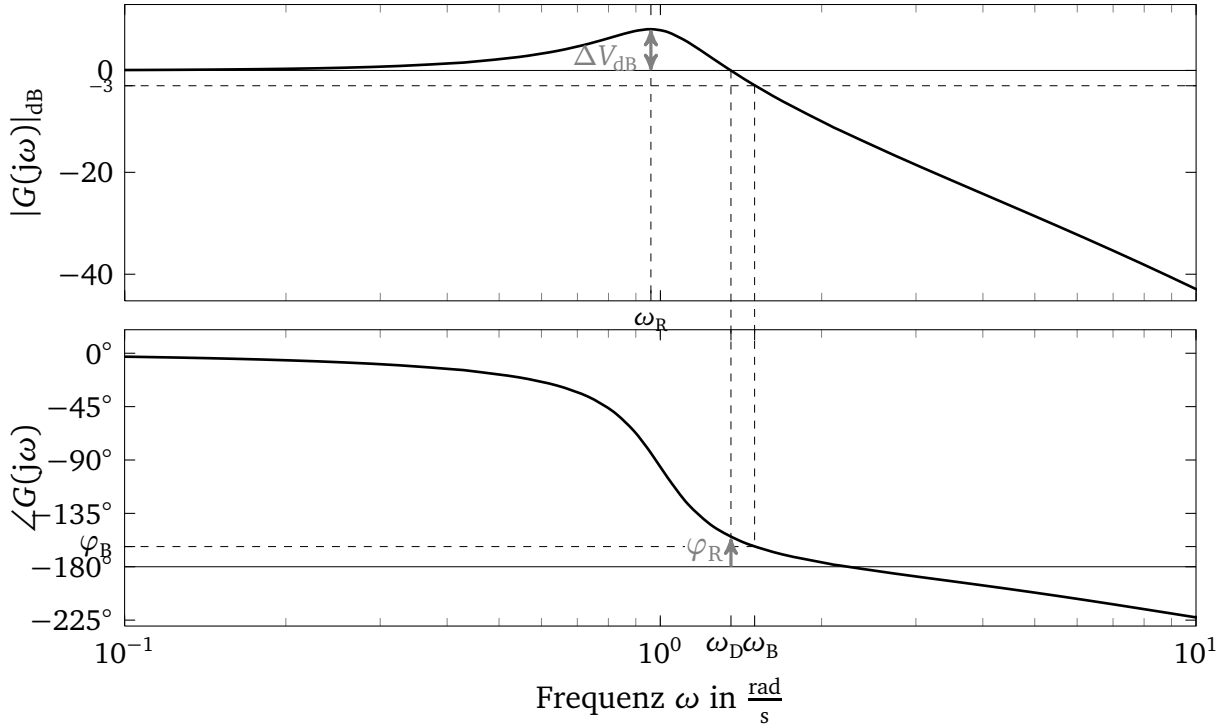
Plots mit LaTeX und pgfplots
Eine ausführliche Anleitung

TECHNISCHE UNIVERSITÄT DARMSTADT

V1.0 23.11.2012 Markus Grün

Der große Vorteil von pgfplots liegt darin, dass die Grafiken direkt in LaTeX kompiliert werden und somit die gleiche Schriftart- und größe besitzen und die Liniendicke definiert ist. Dadurch ergibt sich ein schönes Gesamtbild des Dokuments, das wirkt als wäre es aus einem Guss.



Durch das Kompilieren des Bildes direkt in LaTeX kann sich jedoch auch ein Problem ergeben, nämlich dann, wenn die Datenmenge zu groß ist. Das TeXnicCenter beschwert sich dann mit einem

```
[...] ! TeX capacity exceeded, sorry [main memory size=3000000].
```

oder ähnlich. Dies wird sehr schnell erreicht, wenn zum Beispiel viele Plots in einem Dokument vorhanden sind. Stefan Kopf und Matthias Singer haben deswegen einige Batch-Skripte erstellt, mit denen man bequem die einzelnen Grafiken externalisieren kann. Es wird von jeder einzelnen Grafik eine eigene `.eps`- und `.pdf`-Datei erstellt, die anschließend im Dokument wieder eingebunden wird. Auf diese Weise ist es nicht nur möglich, viele umfangreiche Plots in einem Dokument zu haben, es wird – und das ist bei langen Dokumenten wichtig – auch die Kompilierzeit drastisch verringert, da die Grafiken nicht mehr jedesmal neu kompiliert, sondern nur noch eingebunden werden müssen. Diese Skripte können in das TeXnicCenter integriert werden, so dass man damit sehr einfach und bequem arbeiten kann.

Inhalt der zip-Datei

- Ordner „TikZ“, enthält die Batch- und Matlabskripte
- Ordner „Handbücher“, enthält Handbücher zur Nutzung von pgfplots und TikZ.
- userimages.bmp

Einrichtung der Arbeitsumgebung

- Kopiere den Inhalt des Ordners "TikZ" nach "D:\TikZ".
- In MATLAB muss der Ordner "D:\TikZ\Matlab" in den Suchpfad aufgenommen werden (File->Set Path->Add Folder...)
- Ersetze die Datei „UserImages.bmp“ im Ordner „C:\Programme\TeXnicCenter“ durch die entsprechende Datei aus dem zip-File
- Im TEXNICCENTER über Extras->Anpassen->Tools die folgenden Befehle wie in Tabelle 1 definieren. Danach das Fenster schließen, damit die Befehle gespeichert werden.
- Unter Extras->Anpassen->Tastatur können den neu definierten Befehlen Tastaturkürzel zugewiesen werden. Die neuen Befehle befinden sich unter der Kategorie „Extras“.
- Über Extras->Anpassen->Befehle->Symbolleisten kann eine neue Symbolleiste erstellt werden. Wechselt man in den Reiter Befehle kann man per Drag&Drop die Befehle auf die neue Symbolleiste ziehen. Mit einem Rechtsklick (bei noch geöffnetem Dialogfeld) kann man das Symbol der Schaltfläche ändern. Wurde die „UserImages.bmp“ durch die neue Datei ersetzt, stehen entsprechende Symbole zur Verfügung.

Tabelle 1: Definition der Befehle

Name	TikZ erstellen
Befehl	D:\TikZ\maketikzTC.bat
Argumente	"%dc" "%tc" "%dm" "%bm"
Tastaturkürzel	F8
Name	TikZ anzeigen
Befehl	C:\Pogramme\SumatraPDF\SumatraPDF.exe
Argumente	"%bc-ext.eps"
Tastaturkürzel	F6
Name	TikZ-Quelltext anzeigen
Befehl	C:\Programme\TeXnicCenter\TeXnicCenter.exe
Argumente	/ddecmd "[goto('%dm\%s.tikz','1')]"
Tastaturkürzel	F12

Das TeXnicCenter lässt sich am bequemsten über Tastaurkürzel und Sondertasten (F-Tasten) bedienen. Die wichtigsten Tastaturkürzel für die Bedienung sind dabei:

Kürzel	Funktion
F5	Dokument in Sumatra-PDF öffnen
F7	Dokument kompilieren
F6	Bild in Sumatra-PDF öffnen
F8	Bild externalisieren
F12	Bildquellcode in TeXnicCenter öffnen
`strg+s`	Speichern

Benutzung der Skripte

Der Arbeitsablauf zum Externalisieren der Grafiken lässt sich am besten an einem Beispiel demonstrieren. Es werden dabei im Folgenden alle notwendigen Schritte aufgezeigt, um `pgfplots` effizient einzusetzen. Es empfiehlt sich, das nachfolgende Beispiel am eigenen PC Schritt für Schritt nachzuvollziehen.

Projekt erstellen

Zuerst erstellen wir im TeXnicCenter über `Datei->neu->Projekt` ein neues Projekt namens „myProject“ im Ordner „`D:\myFolder\`“. Es öffnet sich die Datei „`myProject.tex`“ und wir fügen dort folgende Zeilen ein:

```
\documentclass{scrartcl}
\input{D:/TikZ/pgfplotssetup.tex}

\begin{document}

Hello World!

\end{document}
```

Durch Drücken von „`F7`“ können wir das Dokument nun kompilieren und mit „`F5`“ im Sumatra-Reader betrachten.

Bild erstellen

Prinzipiell sollten alle Bilder in einem extra Ordner abgelegt werden, um eine übersichtliche Ordnerstruktur zu wahren. Wir erstellen also den Ordner „`D:\myFolder\Bilder`“, indem alle Bilder dieses Dokuments abgespeichert werden. Diesmal möchten wir das Bodediagramm zweier PT_1 Glieder plotten. Dazu wechseln wir zu MATLAB und tippen ins `Command Window`:

```
G1 = tf([1],[0.5  1]);
G2 = tf([1],[3.5  1]);
pgfbode(G1, G2, 'myPlot', 'D:\myFolder\Bilder');
```

Die Funktion „`pgfbode{.}`“ hat die gleichen Eigenschaften, wie die MATLAB-eigene Funktion „`bode{.}`“, mit dem Unterschied, dass eine `.tikz`-Datei erzeugt wird, die direkt in LaTeX kompiliert werden kann. Wie aus der Kurzanleitung bekannt, werden nun die Datei „`myPlot.tikz`“, sowie der Ordner „`TikZdata`“ mit den `.txt`-Dateien erstellt. Wir öffnen die `.tikz`-Datei am einfachsten, indem wir sie aus dem Explorer per Drag&Drop ins TeXnicCenter ziehen.

Die durch die MATLAB-Funktion `pgfbode(.)` erzeugte `.tikz`-Datei ist fertig kompilierbar. Wir können also durch Drücken von „`F8`“ das Bild direkt erzeugen. Im Ordner „`D:\myFolder\Bilder`“ sollten nun die Dateien

- `myPlot.tikz`
- `myPlot-ext.pdf`
- `myPlot-ext.eps`

enthalten sein. Drückt man im TeXnicCenter „`F6`“, wird die `.eps`-Datei im SUMATRA angezeigt.

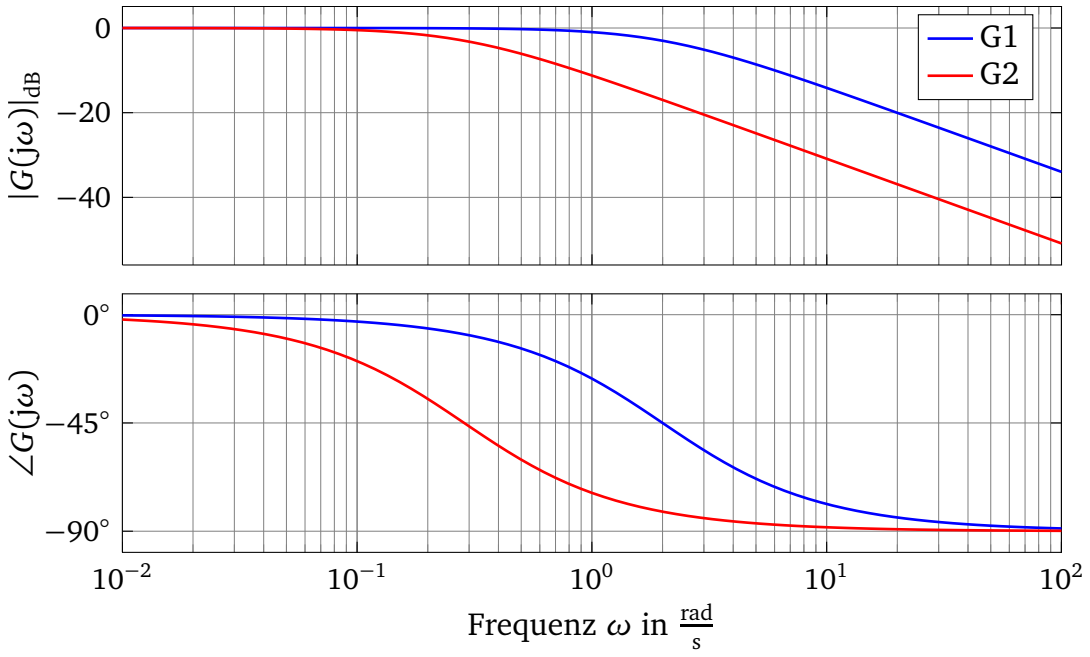


Bild einfügen

Damit das Bild auch in unserem Dokument erscheint, fügen wir in der „myProject.tex“ folgende Zeile ein:

```
\documentclass{scrartcl}
\input{D:/TikZ/pgfplotssetup.tex}

\begin{document}

Hello World!
\includegraphics{./Bilder/myPlot-ext}

\end{document}
```

Im Gegensatz zu dem Beispiel in der Kurzanleitung wird das Bild hier nicht über `\input{.}` eingebunden, sondern über `\includegrahpics{.}`. Es ist dabei besonders wichtig, dass bei `\includegraphics{.}` keine Dateiendung, sondern „`-ext`“ hintenan gefügt wird. Auf diese Weise wird die korrekte Dateiendung selbständig entsprechend des Ausgabeprofils gewählt. Mit „F7“ kompilieren wir unser Projekt und das Bodediagramm sollte nun auch in unserem Dokument dargestellt werden.

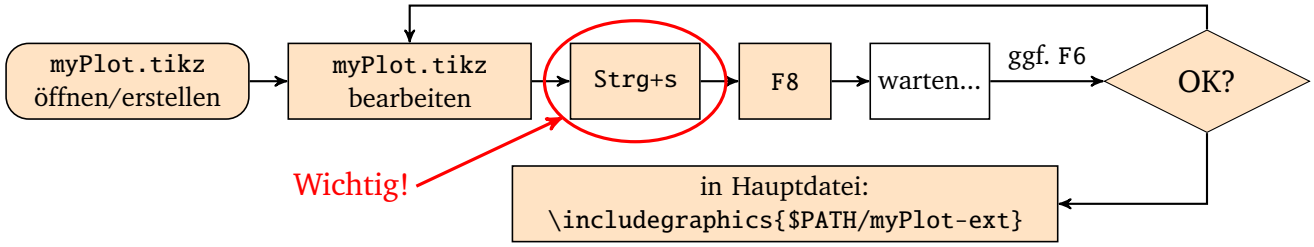
Bild ändern

Farbige Linien sollten prinzipiell vermieden werden, da sie bei einem Ausdruck auf einem Schwarz-Weiß Drucker nicht gut zu unterscheiden sind. Dies wollen wir nun ändern. Falls die `.tikz`-Datei geschlossen wurde, markieren wir den Pfad in `\includegraphics{.}`, jedoch ohne die Endung „`-ext`“ und drücken „F12“. Die Bilddatei wird jetzt wieder im TeXnicCenter geöffnet.

```
...
\includegraphics{./Bilder/myPicture-ext}
...
```

Die Linienfarbe finden wir in Zeile 12, 14, 23 und 24. `\addplot[smooth, blue,...` Dort können wir auch die Linienart ändern, zum Beispiel in gestrichelt „`dashed`“ oder in gepunktet „`dotted`“. Natürlich muss die Linienfarbe und -art sowohl im Amplitudengang, als auch im Phasengang geändert werden! Auch können wir den Eintrag in der Legende anpassen.

Bevor wir die Datei erneut externalisieren, *muss sie zwingend gespeichert werden*! Wir drücken also *erst* “`Strg+s`“ und dann „F8“ zum externalisieren des Bildes. Vergisst man die Datei zu speichern, wird die alte Datei, ohne die Änderungen kompiliert.

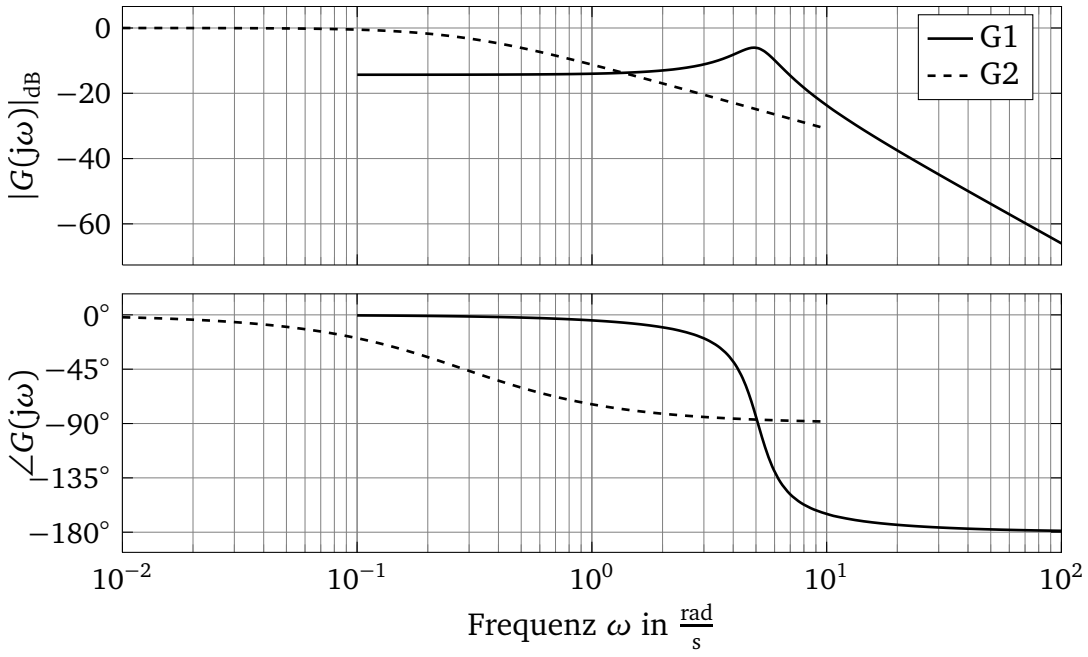


```
\renewcommand{\mywidth}{0.8\textwidth}
\renewcommand{\myheight}{50mm}

\begin{tikzpicture}
% Amplitudengang
  \begin{bodeAmpDB}[
                    enlarge x limits = false,
                    grid=both, % minor|major|both|none
                    legend pos = north east,  % [outer] south west|south east|north west|north←
                         ← east
                    legend cell align=left,   % Legendeneinträge linksbündig
                    ]
    \addplot[smooth, line width=1pt] table{./Bilder/TikZdata/myPlot/G1_mag.txt};
    \addlegendentry{G1};
    \addplot[smooth, dashed, line width=1pt] table{./Bilder/TikZdata/myPlot/G2_mag.txt};
    \addlegendentry{G2};
  \end{bodeAmpDB}

% Phasengang
  \begin{bodePhase}[
                    enlarge x limits = false,
                    grid=both, % minor|major|both|none
                    ]
    \addplot[smooth, line width=1pt] table{./Bilder/TikZdata/myPlot/G1_phase.txt};
    \addplot[smooth, dashed, line width=1pt] table{./Bilder/TikZdata/myPlot/G2_phase.txt};
  \end{bodePhase}
\end{tikzpicture}
```

Jetzt fällt uns allerdings ein, dass in *diesem* Bild statt den beiden PT_1 Gliedern lieber nur ein PT_2 Glied plotten wollten. Wir gehen also zurück zu MATLAB und tippen ein:

```
G1 = zpk([],[-1+5j -1-5j],5);
pgfbode(G1, 'myPlot', 'D:\myFolder\Bilder');
```

Aus der Kurzanleitung ist ja schon bekannt, dass die in der `.tikz`-Datei vorgenommenen Einstellungen bezüglich Linienfarbe und -art erhalten bleiben. Wenn wir jetzt das Bild aber erneut kompilieren, stellen wir fest, dass es noch nicht so ganz unseren Wünschen entspricht.



Das Bodediagramm der Übertragungsfunktion „G2“ wird immer noch geplottet, obwohl wir in pgfbode(.) nur noch G1 angegeben haben. Dafür sind die vorgenommenen Einstellungen bezüglich Linienfarbe und -art erhalten geblieben.

Hier sehen wir die Vor- und Nachteile der MATLAB-Skripte. Da die .tikz-Datei von MATLAB nicht mehr verändert wird, sobald sie einmal erstellt wurde, können natürlich auch die „\addplot(.)“ Einträge, mit denen die Wertedateien eingebunden werden, nicht mehr den neuen Gegebenheiten angepasst werden. Tritt solch ein Fall auf, dass sich die Anzahl der darzustellenden Plots ändert, oder sich die Länge der Wertedateien (drastisch) ändert, so müssen die neu hinzugekommenen Dateien, bzw. die nun überflüssigen Dateien manuell aus der .tikz-Datei entfernt werden.

Voraussetzungen und Hinweise für die Benutzung der Skripte

- Es *muss* ein Projekt im TeXnicCenter geöffnet sein.
- Die Dateiendung der Bilddatei *muss* .tikz lauten!
- Die Bilddatei darf sich nicht im gleichen Verzeichnis befinden, wie die gerade geöffnete Projektdatei (.tcp) (Bilder am besten immer in einen eigenen Ordner „Bilder“)
- Der Text \begin{document} darf keine Leerzeichen enthalten und muss direkt in der ersten Spalte beginnen.
- Die Skripte verwenden die Hauptdatei des gerade geöffneten Projektes. Mit den Einstellungen (Makros, Packages, Schriften, etc.) dieser Hauptdatei wird die Bilddatei kompiliert. Alle für die Erstellung der Grafik benötigten Makros müssen schon vor \begin{document} definiert werden, damit sie bei der Bilderstellung verfügbar sind.
- Die Bilddatei wird nicht automatisch gespeichert. Vor dem Erstellen des Bildes also *unbedingt* die Quelldatei manuell speichern!
- Die Größe des Bildes darf *nicht* über \includegraphics[scale=XXX]... geändert werden, sondern ausschließlich über die Parameter „\mywidth{XX}“ und „\myheight{XX}“ in den ersten beiden Zeilen des .tikz-Codes.

Funktionsweise der Matlabfunktionen

Jede Matlabfunktion erstellt im Ordner „.\Bilder\" eine Quelldatei des Bildes (z. B. „myPlot.tikz“) und ein oder mehrere Wertedateien (.txt-Dateien) im Ordner „.\Bilder\TikZdata\myPlot\“.

Die Quelldatei des Bildes beinhaltet den .tikz-Code. Hier werden alle Einstellungen vorgenommen, die das Aussehen des Bildes beeinflussen (Achsenbeschriftungen, Linienart und -farben, Pfeilspitzen, Grid, Markierungen, etc.) sowie die Wertedateien geladen. Die Wertedateien (.txt-Dateien) enthalten den eigentlichen „Plot“, also alle x- und y-Werte in Form einer Tabelle.

Die Matlabfunktion erstellt die Quelldatei des Bildes (.tikz-Datei) nur *einmal*. Sie wird bei einem erneuten Ausführen nicht mehr überschrieben, nur die Wertedateien werden jedesmal neu erstellt. Dies bedeutet, dass bei einem erneuten Ausführen der Matlabfunktion alle vorgenommenen Einstellungen in der .tikz-Datei erhalten bleiben und nur der eigentliche Plot neu gespeichert wird.

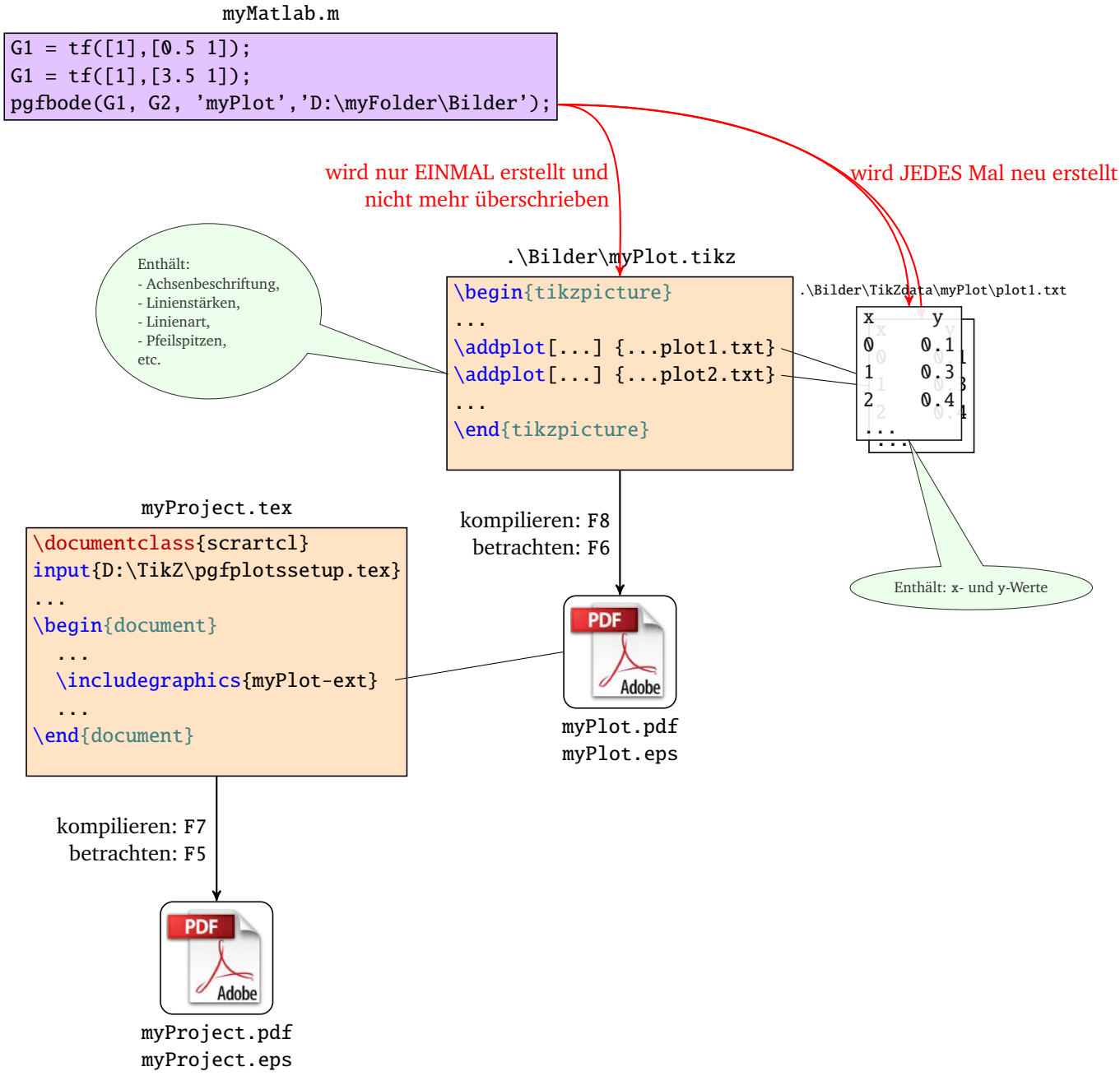


Abbildung 1: Funktionsweise der MATLAB-Skripte

Matlabfunktionen

Alle Matlabfunktionen aus dem Ordner „`c:\Programme\TikZ\Matlab`“ haben die gleiche Grundfunktionalität, wie die entsprechende Originalfunktion von MATLAB. Die Funktionsnamen sind ebenfalls identisch, jedoch mit einem vorangestellten „`pgf`“. Jeder Funktion müssen als letzte zwei Parameter der Dateiname der `.tikz`-Datei und der Zielordner übergeben werden. Alle Angaben in eckigen Klammern „`[ ]`“ sind optional.

Variable	Bedeutung
`sys`, `sys1`,...	Übertragungsfunktion, bzw. Matrix von Übertragungsfunktionen vom Typ „`tf`“, „`zpk`“ oder „`ss`“
`x`, `y`, `x1`, `y1`, ...	Vektoren oder Matrizen gleicher *Länge*
`mag`, `phase`, `freq`	Vektoren oder Matrizen gleicher *Größe* (`freq`: gleiche Länge)
`w`	Frequenzintervall in der Form von `w = {wmin, wmax}`
`name`	Name des Plots (die `.txt`-Datei wird entsprechend benannt und die Legende entsprechend gesetzt)
`filename`	Dateiname der `.tikz`-Datei
`folder`	Speicherort der `.tikz`-Datei
`[...]`	optionale Angabe, kann weggelassen werden.

pgfplot.m

`pgfplot(x,y,[name], [x2,y2,[name],...], filename, folder)`
Erzeugt einen Plot der Vektoren/Matrizen `x` und `y` (und ggf. `x2`, `y2`,...).

pgfplot3.m

`pgfplot3(x,y,z,[name], [x2,y2,z2,[name],...], filename, folder)`
Erzeugt einen dreidimensionalen Plot der Vektoren/Matrizen `x`, `y` und `z` (und ggf. `x2`, `y2`, `z2`,...).

pgfplotyy.m

`pgfplotyy(x1,y1,[name], x2,y2,[name], filename, folder)`
Erzeugt einen Plot mit zwei y-Achsen mit den Vektoren/Matrizen `x1`, `y1` und `x2`, `y2`.

pgfsemilogx.m

`pgfsemilogx(x,y,[name], [x2,y2,[name],...], filename, folder)`
Erzeugt einen Plot der Vektoren/Matrizen `x` und `y` (und ggf. `x2`, `y2`,...), wobei die Werte auf der x-Achse logarithmisch und die auf der y-Achse linear skaliert sind.

pgfsemilogy.m

`pgfsemilogy(x,y,[name], [x2,y2,[name],...], filename, folder)`
Erzeugt einen Plot der Vektoren/Matrizen `x` und `y` (und ggf. `x2`, `y2`,...), wobei die Werte auf der x-Achse linear und die auf der y-Achse logarithmisch skaliert sind.

pgfloglog.m

`pgfloglog(x,y,[name], [x2,y2,[name],...], filename, folder)`
Erzeugt einen doppelt-logarithmischen Plot der Vektoren/Matrizen `x` und `y` (und ggf. `x2`, `y2`,...).

pgfbode.m

`pgfbode(sys,[name], [sys2,[name],...], [w], filename, folder)`
`pgfbode(mag, phase, freq, [name], filename, folder)`
Erzeugt ein Bodediagramm der Übertragungsfunktion `sys` (und ggf. `sys2`,...), bzw. aus den Vektoren `mag`, `phase` und `freq`.

pgfbodeLOG.m

`pgfbodeLOG(sys,[name], [sys2,[name],...], [w], filename, folder)`
`pgfbodeLOG(mag, phase, freq, [name], filename, folder)`
Erzeugt ein Bodediagramm in doppelt-logarithmischer der Übertragungsfunktion `sys` (und ggf. `sys2`,...), bzw. aus den Vektoren `mag`, `phase` und `freq`.

pgfasymp.m

`pgfasymp(sys,[name], [sys2,[name],...], [w], filename, folder)`
Erzeugt ein Bodediagramm in der Übertragungsfunktion `sys` (und ggf. `sys2`,...). Zusätzlich werden die Asymptoten in das Bodediagramm eingezeichnet.

pgfasympLOG.m

`pgfasympLOG(sys,[name], [sys2,[name],...], [w], filename, folder)`
Erzeugt ein Bodediagramm in doppelt-logarithmischer Darstellung der Übertragungsfunktion `sys` (und ggf. `sys2`,...). Zusätzlich werden die Asymptoten in das Bodediagramm eingezeichnet.

pgfstep.m

`pgfstep(sys,[name],[sys2,[name],...], [Tend / Tvec], filename, folder)`
Erzeugt den Plot einer Sprungantwort der Übertragungsfunktion `sys` (und ggf. `sys2`,...). Mit „`Tend`“ kann optional die Simulationsdauer angegeben werden oder mit „`Tvec`“ optional ein Zeitvektor.

Matlabfunktionen im alpha-Stadium

Die folgenden Funktionen erzeugen die Plots, haben aber noch ein paar Bugs, die behoben werden müssen.

```
pgfnyquist(...)
pgfpzmap(...)
pgfrlocus(...)
```

Einstellungen für `pgfplots`

Mit `pgfplots` lässt sich praktisch alles am Plot von der Linienstärke, den Pfeilspitzen, das Gitternetz und Größe der „Ticks“ bis zur Legende verändern. Einige dieser Einstellungen sollen für alle Plots gelten (Wie zum Beispiel die Linienstärke des „Kastens“ und des Gitternetzes), andere sollen nur für den jeweiligen Plot gelten (Linienart, -stärke und -farbe des eigentlichen Plots, etc.).

Die globalen Einstellungen, die für alle Plots gelten sollen, sind in einer separaten Datei abgelegt `c:\TikZ\pgfplotssetup.tex`. Auf diese Weise kann das Erscheinungsbild *aller* Bilder des Dokuments auf einmal angepasst werden, ohne jede Datei einzeln bearbeiten zu müssen.

Alle lokalen Einstellungen sollten in der jeweiligen Bilddatei vorgenommen werden. Unten findet sich eine kurze Sammlung den wichtigsten Befehlen. Das `pgfplots`-Manual ist sehr ausführlich und bietet noch viel mehr Befehle.

Wichtige Befehle in `\addplot[...]`

Feature	Befehl

Linienart	`dashed`
	`dotted`
	`dashdotted`
	`dashdotdotted`
	`densely [dashed, dotted, dashdotted,...]`
	`loosely [dashed, dotted, dashdotted,...]`

Linienfarbe	`red`, `blue`,`green`,... (alles was das `xcolor` paket hergibt) `red!30!white` = Mix aus 30% rot und 70% weiß. noch mehr Farben ab Seite 123 im PGFPLOTS-Manual

Liniendicke	`thin`
	`semithick`
	`thick`
	`very [thin, thick]`
	`ultra [thin, thick]`
	`linewidth=1pt`
	`linewidth=2mm,...`

Plotstyle	`smooth` (geglätteter Verlauf) `sharp plot` (ungeglätteter Verlauf) `const plot` (eckiger Verlauf) mehr auf Seite 53 des PGFPLOTS-Manuals

Wichtige Befehle in der `axis[...]` Umgebung

Feature	Befehl

Achsenbeschriftung	`xlabel=...`
	`ylabel=...`
	`xlabel style={yshift=2mm}` (xlabel verschieben)
	`ylabel style={yshift=2mm}` (ylabel verschieben)

Achsenskalierung	`xmin=...`, `xmax=...` `ymin=...`, `ymax=...`

manuelle Achsunterteilung und -beschriftung	`xtick={1,2,...,10}`
	`ytick={1,2,...,10}`
	`x tick label={1,2,...}`
	`y tick label={$a$,$b$,...}`

zusätzliche Ticks	`minor x tick num=1`
	`minor y tick num=3`
	`extra x tick={3.14159}`
	`extra x tick label={$\pi$}`

Gitternetz	`grid = both / major / minor / none`
	`xmajorgrids = true / false`
	`xminorgrids = true / false`
	`ymajorgrids = true / false`
	`yminorgrids = true / false`

Legende	`legend pos = [outer] south west / north east ...` `legend cell align=left` (Text linksbündig) `legend style={font=\tiny}` (Schriftgröße) `legend style={row sep=2mm}` (Zeilenabstand)
Beschriftung an beliebiger Stelle:	`node[] at (axis cs:X,Y) {Hello World!}`

Troubleshooting

Problem	Lösungsmöglichkeit
Das Batch-Fenster öffnet und schließt sich wieder, aber es werden keine `.eps` und `.pdf`-Dateien erstellt	Lautet die Dateiendung der Bilddatei „`.tikz`“? Wenn das Beispiel aus dieser Anleitung erfolgreich erstellt externalisiert werden kann: Aller Wahrscheinlichkeit nach gibt es einen Tippfehler im `.tikz`-Code oder der Verweis auf die Quelldateien für die Plots ist fehlerhaft. Statt `\includegraphics{myPicture-ext}` im Dokument `\input{mypicture.tikz}` einbinden und die Fehlermeldung im TeXnicCenter analysieren.
Das Batch-Fenster öffnet sich, bleibt aber hängen	Vermutlich fehlt ein Semikolon als Zeilenabschluß im `.tikz`-Code.
Nach einem erneuten Ausführen des Matlab-Codes ist der Plot unvollständig, bzw. verschwunden.	Da die Bilddatei (`.tikz`) nur einmal erstellt, aber nicht mehr überschrieben wird, können zusätzliche Wertedateien (`.txt`) nicht automatisch in den Quellcode eingefügt werden. Dies muss jetzt manuell geschehen.
Ich habe ein bereits vorhandenes Bild eingefügt, aber die Schriftart/-größe stimmt nicht.	Die Bildquelldatei muss erneut kompiliert werden, wobei darauf zu achten ist, dass das korrekte Projekt im TeXnicCenter geladen laden ist.
Ich habe Änderungen im Quelltext vorgenommen aber nach dem Kompilieren sieht das Bild genauso aus wie vorher.	Vor dem Kompilieren muss die Datei *unbedingt* gespeichert werden!
Ich habe Änderungen im Quelltext vorgenommen und auch gespeichert, aber nach dem Kompilieren sieht das Bild trotzdem noch genauso aus wie vorher.	Dann ist vermutlich der `.tikz`-Code fehlerhaft. Die Datei über `\input{./Bilder/myPlot.tikz}` einbinden und die Fehlermeldungen im TeXnicCenter analysieren.

Wenn Ihr Fehler findet, schreibt mir bitte an `mgruen@iat.tu-darmstadt.de`. Danke.