

Anwendung von (Mathematica und) Matlab in der Physikalischen Chemie

4. Skripte und Funktionen

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2017/2018

Motivation

Editor

Was muss ein guter Editor können?

Funktionen des Matlab-Editors

Refactoring

Programme

Unterschied zwischen Skripten und Funktionen

Vor- und Nachteile von Skripten und Funktionen

Funktionen

Schnittstellen von Funktionen

Kontext von Variablen

Zugriff auf Funktionen

- ▶ Bislang alles auf der interaktiven Kommandozeile
 - Großartig für „Rapid Prototyping“
 - Gut für erste Schritte mit Matlab
 - Ungeschickt für größere Auswertungen

- ▶ Problem
 - Sobald wir Matlab beenden, ist alles weg.
 - Die Befehlshistorie ist auch eher mühsam...

- ▶ Lösung
 - Befehlsfolgen in eine Datei schreiben: [Programm](#)

- ☞ Wir brauchen einen [Editor](#)
 - Was macht einen guten Editor aus?
 - Was davon erfüllt der Matlab-Editor?

Was muss ein guter Editor können?

- ▶ Syntaxhervorhebung („Syntax highlighting“)
 - ▶ automatische Codeeintrückung
 - ▶ Zusammenfalten von Codeteilen („Code folding“)
 - ▶ automatische Codevervollständigung
 - ▶ Codeüberprüfung während der Eingabe
 - ▶ Hilfe aus dem Editor heraus erreichbar
 - ▶ Refaktorisierung („Refactoring“)
- ☛ Der Matlab-Editor unterstützt die meisten Kriterien mittlerweile recht gut (bis auf Refactoring).

Syntaxhervorhebung

- ▶ Unterschiedliche Farben u.a. für
 - Befehle/Variablen
 - Zeichenketten
 - Kommentare
- ▶ Erhöht die Übersichtlichkeit und Lesbarkeit von Code.

Automatische Codevervollständigung

- ▶ Durch die Tabulatortaste
- ▶ Vervollständigt Befehle/Variablen
- ▶ Hilfreich zur Vermeidung von Tippfehlern

Hilfe aus dem Editor aufrufen

- ▶ Zwei Möglichkeiten
 - Rechtsklick mit der Maus auf einen Befehl
 - Cursor im Befehl platzieren und **F1** drücken



Codeeintrückung: Allgemeine Aspekte

- ▶ Automatische Codeeintrückung erhöht die Lesbarkeit.
- ▶ Beginn und Ende von Schleifen sind einfach erkennbar.
- ☛ Saubere Codeeintrückung ist nicht optional.

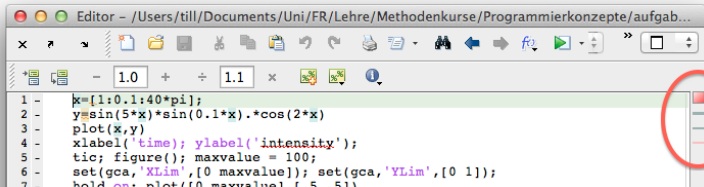
Codeeintrückung im Matlab-Editor

- ▶ Der Editor beherrscht automatische Codeeintrückung.
- ▶ Codebereiche können im Nachhinein automatisch eingerückt werden.

Codeüberprüfung im Matlab-Editor

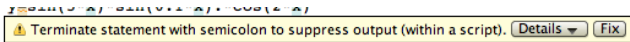
- ▶ Der Matlab-Editor zeigt drei Zustände an.
(alles in Ordnung, Warnungen, Fehler)
- ▶ Für Warnungen und Fehler können zusätzliche Hinweise angezeigt werden.
- ▶ Häufig wird für Warnungen und Fehler eine automatische Behebung angeboten („Autofix“).
- ▶ Warnungen können ignoriert/abgeschaltet werden
(im Einzelfall sinnvoll).
- ☛ Warnungen und Fehler sollten *in jedem Fall* ernst genommen und deren Ursache behoben werden.

Codeüberprüfung im Matlab-Editor



```
1 - x=[1:0.1:40*pi];
2 - y=sin(5*x)*sin(0.1*x).*cos(2*x)
3 - plot(x,y)
4 - xlabel('time'); ylabel('intensity');
5 - tic; figure(); maxvalue = 100;
6 - set(gca,'XLim',[0 maxvalue]); set(gca,'YLim',[0 1]);
7 - hold on; plot(0,maxvalue,1.5 .5)
```

Autofix im Matlab-Editor



Refactoring (noun)

a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

- ▶ Beispiele für Refactoring:
 - Umbenennung von Variablen
 - Umbenennung von Funktionen (inkl. der Aufrufe)
 - Auslagerung von Codeblöcken in Funktionen
- ☛ Matlab beherrscht nur Variablenumbenennung (in *einem* Dokument)

Listing 1: Beispiel für ein (reales) Matlab-Programm

```
1 % Setup of basic spin system
2 Sys.S = 1;           % Triplet
3 Sys.g = 2.002;      % g_iso
4 Sys.lw = [0 1];    % Only Lorentzian linewidth (in mT)
5
6 % Setting some experimental parameters
7 Exp.Range = [280 410];
8 Exp.mwFreq = 9.7;
9 Exp.Harmonic = 0;
10
11 % ZFS parameters and populations
12 Sys.D = [1300 70]; % D and E values (in MHz)
13 Exp.Temperature = [0 0.1 0.9]; % ZF populations (see note on top)
14
15 [x1,y1] = pepper(Sys,Exp);
16 figure(101);
17 plot(x1,y1);
18 set(gca,'XLim',Exp.Range);
19 title('\it D > 0, \it E > 0');
```

- ▶ **Reine Text-Dateien**
 - Keine Sonderzeichen, nur ASCII-7-bit-Zeichen
 - Matlab beherrscht kein Unicode
 - Grundsätzlich ist jeder Text-Editor geeignet

- ▶ **Dateiendung: .m**

- ▶ **Benennung**
 - Nur Buchstaben, Zahlen und „_“ sind erlaubt.
 - Name muss mit einem Buchstaben beginnen.

- ▶ **Zugriff auf Programme in Matlab**
 - Durch den Matlab-Suchpfad bestimmt
 - Alternativ: lokales Verzeichnis
 - Zur Rangfolge vgl. die Matlab-Hilfe.

Skript

- ▶ Liste von Befehlen
- ▶ Alle Variablen sind global zugreifbar
- ▶ Name beliebig (aber eindeutig)

Funktion

- ▶ Zusammengehörender Codeblock mit einer Aufgabe
- ▶ Parameterübergabe beim Aufruf
- ▶ Variablen lokal für die Funktion
- ▶ Funktions- und Dateiname müssen übereinstimmen

Skript

- ✓ Nachvollziehbarkeit dank intrinsischer Dokumentation
- ✓ Reproduzierbarkeit: Daten und Skript immer beisammen.
- ✗ Schwierige Versionierung und Wiederverwendbarkeit
- ✗ Alle Variablen auf der Kommandozeile

Funktion

- ✓ Modularität und Wiederverwendbarkeit
- ✓ Alle Variablen lokal
- ✗ Nachvollziehbarkeit und Reproduzierbarkeit nur mit zusätzlichem Aufwand erreichbar

Befehle und Funktionen

- ▶ Befehle sind Funktionen
- ▶ (Selbst geschriebene) Funktionen sind Befehle

Möglichkeiten des Aufrufs

- 1 Kommandozeile
 - Befehl wird direkt eingetippt und ausgeführt
- 2 Skript
 - Liste von Befehlen in einer Datei
- 3 Funktion selbst schreiben
 - Grund: „Ich will etwas tun, was Matlab nicht kann.“
 - Spracherweiterung mithilfe vorhandener Befehle

Funktionen in Matlab

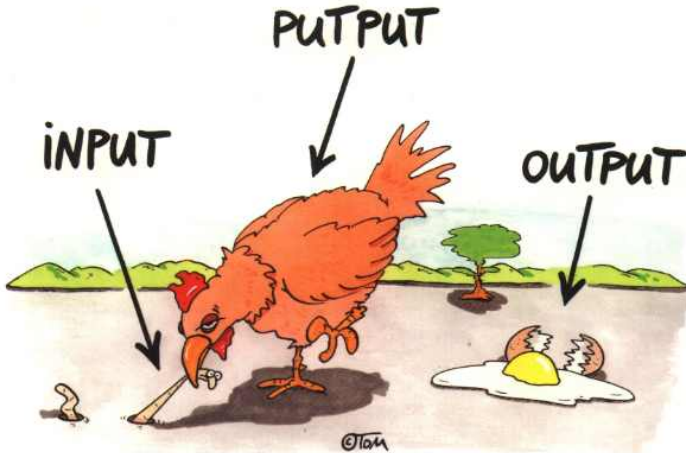
- ▶ Funktionsname und Dateiname müssen übereinstimmen.
- ▶ Nur eine Funktion pro Datei (Ausnahme: Unterfunktionen)

Benennung von Funktionen in Matlab

- ▶ Matlab unterscheidet zwischen Groß- und Kleinschreibung.
 - ▶ Funktionsnamen müssen mit einem Buchstaben beginnen.
 - ▶ Sonderzeichen sind nicht erlaubt. (Ausnahme: „_“)
-
- ☞ Tipp: Sprechende Namen erhöhen die Lesbarkeit
 - ☞ Tipp: Präfix zur Vermeidung von Doppelungen

Funktionen

Klar definierte Schnittstellen



Thomas Körner, alias ©TOM

Listing 2: Funktionsdeklaration in Matlab

```
function [out1,out2] = myFunction(in1,in2)
```

Funktionsdeklaration in Matlab

- 1 Schlüsselwort „function“
- 2 Liste der Rückgabeparameter (output)
- 3 Funktionsname
- 4 Liste der Übergabeparameter (input)

☞ Es gibt Funktionen *ohne* Parameter.

Möglicher Kontext einer Variablen

- lokal nur für die jeweilige Funktion „sichtbar“
- global für alle Funktionen „sichtbar“

Konsequenzen

- ▶ Unterschiedliche Funktionen können Variablen mit dem gleichen Namen verwenden.
- ▶ Eine Funktion kennt nur globale, ihr übergebene oder in ihr definierte Variablen.
- ☛ Skripte haben (im Gegensatz zu Funktionen) Zugriff auf alle Variablen im Matlab-„Workspace“.

Zugriff auf Funktionen

- ▶ Grundlegende Unterscheidung
 - Eingebaute Befehle
 - Zusätzliche Funktionen
- ▶ Wo Matlab nach Funktionen sucht
 - Aktuelles Verzeichnis
 - Pfaddefinition über `pathdef`
- ▶ Vordefinierte Rangfolge bei Namenskonflikten
 - Details in der Matlab-Hilfe
- ▶ Welche Funktion wird aufgerufen?
 - `which`-Befehl gibt Auskunft.

Unterfunktionen

- ▶ Matlab erlaubt Unterfunktionen in Funktionen
 - Nur aus der Hauptfunktion aufrufbar
- ▶ Zwei Arten von Unterfunktionen
 - Innerhalb der Hauptfunktion definiert
 - Unterhalb der Hauptfunktion definiert
- ▶ Vorteile
 - Können den Code sehr übersichtlich/lesbar gestalten.
 - Geeignet für besondere Aufgaben innerhalb einer Funktion.
- ▶ Nachteile
 - Unterfunktion nur aus der Hauptfunktion aufrufbar.
 - Kann schnell zu Code-Doppelungen führen.

Unterfunktion innerhalb der Hauptfunktion

Listing 3: Unterfunktion innerhalb der Hauptfunktion

```
function [...] = mainFunction(...)  
    function [...] = subFunction(...)  
    end  
end
```

- ▶ Die Unterfunktion hat Zugriff auf alle Variablen der Hauptfunktion.
- ▶ Veränderung der Variablen in der Unterfunktion wirkt auf die Hauptfunktion zurück.

Unterfunktion außerhalb der Hauptfunktion

Listing 4: Unterfunktion außerhalb der Hauptfunktion

```
function [...] = mainFunction(...)  
  
end  
  
function [...] = subFunction(...)  
  
end
```

- ▶ Die Unterfunktion hat *keinen* Zugriff auf die Variablen der Hauptfunktion.
- ▶ Parameterübergabe wie bei normalen Funktionen nur über die Schnittstelle.

Das `private`-Verzeichnis

- ▶ Besonderheit des `private`-Verzeichnisses
 - Alle Funktionen in diesem Verzeichnis sind nur von Funktionen im Verzeichnis direkt darüber aufrufbar.
 - Funktionen in diesem Verzeichnis erscheinen nicht im Matlab-Suchpfad.
- ▶ Vorteile
 - Kann Code-Duplizierung vermeiden.
 - Mehrere Funktionen können auf dieselben Funktionen im `private`-Verzeichnis zugreifen.
- ▶ Nachteile
 - Zugriff nur von Funktionen aus möglich, die direkt oberhalb des jeweiligen `private`-Verzeichnisses liegen.

Stabilität von Schnittstellen

- ▶ Schnittstellen sind das, was der Anwender von einer Funktion „sieht“ und was er verwendet.
 - Innerhalb der Funktion kann man (fast) alles ändern.
 - Schnittstellen sollten möglichst früh definiert werden.
 - Nachdenken zahlt sich aus: vorausschauend planen.

- ▶ Änderungen der Schnittstellen nur in begründeten Fällen
 - Inkompatible Änderungen führen zu Fehlern in Code, der auf älteren Versionen der Funktion basiert.
 - Wann immer möglich abwärtskompatibel programmieren
 - **Wichtig:** Dokumentation inkompatibler Änderungen

- ☞ Greift voraus auf die Entwicklung eigener Programme.



...Zeit für eigene praktische Arbeit...

Vorschau: [Grundlegende Sprachkonzepte](#)

- ▶ Syntax
- ▶ Operatoren
- ▶ Datentypen
- ▶ Entscheidungsstrukturen
- ▶ Schleifen