Institut für Physikalische Chemie

Übungen zum Methodenkurs „Programmierkonzepte in der Physikalischen Chemie“ im WS 2013/2014

Dr. Till Biskup

— Lösungen zum Aufgabenblatt 2 vom 08.01.2014 —

Aufgabe 2—1

Sie finden die Einstellungen zur Tastaturbelegung unter „Preferences“ → „Keyboard“ → „Shortcuts“. Je nach Betriebssystem heißen die Einstellungen unterschiedlich. Sie wollen mit hoher Sicherheit nicht das „Emacs Default Set“ verwenden.

Aufgabe 2—2

Nachfolgend finden Sie den von Fehlern und Warnungen bereinigten Quellcode. Vergleichen Sie ihn mit dem ursprünglichen Quellcode von der Internetseite[1] des Methodenkurses.

Ein paar Anmerkungen zum Quellcode in seiner bereinigten Form:

- Wie Sie sehen, wurde zusätzlich noch die Einrückung automatisch vereinheitlicht. Sie finden diese Funktion im Matlab-Editor im Menü „Text“ → „Smart Indent“.
- In Zeile drei (Zeile zwei im originalen Quellcode) befand sich ein Fehler, der nicht vom Editor selbst bemerkt wurde, sondern erst durch die versuchte Ausführung auf der Matlab-Kommandozeile. Hier fehlte an einer Stelle der Punkt vor dem Multiplikationszeichen. Im vorliegenden Fall wollen Sie elementweise multiplizieren, da es sich bei „`x`“ um einen Vektor handelt. Hierfür benötigen Sie als Multiplikator „`.*`“ statt einfach nur „`*`“.
- Es wurden minimale Kommentare in den Quellcode eingefügt und die einzelnen, komplett unabhängigen Codeteile optisch durch eine Leerzeile voneinander getrennt.
- Wie Sie sehen können gibt es Befehle, die man sinnvoller Weise mit einem Semikolon abschließt, um die Ausgabe zu unterdrücken (z.B. die Definition der Vektoren in den ersten Zeilen), Befehle, bei denen es egal ist, ob sie mit Semikolon abgeschlossen werden (z.B. der Befehl `plot`), und Befehle, wo Ihnen (zumindest im hier vorliegenden Kontext) daran gelegen ist, die Ausgabe nicht zu unterdrücken und deshalb den Befehl gerade *nicht* mit einem Semikolon abzuschließen (Befehl `toc`) im obigen Beispiel.
- Einen Aspekt, den der Matlab-Editor nicht oder nur in seltenen Fällen bemängelt, sind mehrere durch Semikolon getrennte Definitionen oder Befehle in einer Zeile. Trotzdem erhöht es immens die Übersichtlichkeit, wenn Sie hier die entsprechenden Definitionen jeweils in einer eigenen Zeile vornehmen.

[1] `http://till-biskup.de/de/lehre/programmierkonzepte/`

Listing 1: Fehlerbereinigter Quellcode

```
% First plot: Some trigonometric functions
x=1:0.1:40*pi;
y=sin(5*x).*sin(0.1*x).*cos(2*x);
plot(x,y)
xlabel('time');
ylabel('intensity');

% Second plot: Random numbers
tic;
figure();
maxvalue = 100;
set(gca,'XLim',[0 maxvalue]);
set(gca,'YLim',[0 1]);
hold on;
plot([0 maxvalue],[.5 .5])
for k=1:maxvalue
    value = rand;
    if value<0.5
        disp('Lower half');
    else
        disp('Upper half');
    end
    plot(k,value,'rx');
    pause(0.02);
end
hold off
toc

% Third plot: Lissajous
tic;
t = 0:0.01:2*pi;
x = 4 * cos(3*t);
y = 4 * cos(4*t+pi/2);
figure;
hold on;
for k=1:length(x)
    plot(x(k),y(k),'r.');
    pause(0.001);
end
toc
```

Aufgabe 2—3

(Grafik-)Objekte haben in Matlab Eigenschaften, die Sie über die Befehle `get` und `set` abfragen und setzen können. Die Richtung einer Achse ist in der Eigenschaft `XDir` bzw. `YDir` abgelegt, ein Wert von `normal` bedeutet dabei die im mathematischen Kontext übliche Richtung der Achse, ein Wert von `reverse` dreht die Achse um. Entsprechend kann die Richtung einer Achse (hier die y-Achse) wie folgt abgefragt werden (beachten Sie, dass der Befehl nicht durch ein Semikolon abgeschlossen wird, um die Ausgabe der Antwort nicht zu unterdrücken):

Listing 2: Richtung einer Achse abfragen

```
get(gca,'YDir')
```

Liefert der obige Aufruf z.B. `reverse` zurück, was im Falle einer mit `image` oder `imagesc` erzeugten Grafik (für die y-Achse) der Fall ist, kann die Achse wie folgt in ihrer Richtung umgekehrt werden:

Listing 3: Richtung einer Achse setzen

```
set(gca,'YDir','normal');
```

Eine vollständige Liste aller Acheneigenschaften („axes properties“) finden Sie in der Matlab-Hilfe[2].

Aufgabe 2—4

Die Antwort liefert die Matlab-Hilfe, erreichbar durch den Aufruf von `doc str2num`. Der entscheidende Aspekt findet sich in einem hervorgehobenen Kasten, gleich zu Beginn unter der Überschrift „Description“:

> str2num uses the eval function to convert the input argument. Side effects can occur if the string contains calls to functions. Using str2double can avoid some of these side effects.
>
> `http://www.mathworks.de/de/help/matlab/ref/str2num.html`

Das Problem wird hier klar angesprochen: `str2num` evaluiert (über einen Aufruf von `eval`, für Details vergleiche die Hilfe zu diesem Befehl) den übergebenen String. Enthält dieser String Funktionsaufrufe, werden diese entsprechend ausgeführt und können zu unvorhersehbarem Verhalten führen.

Hinweis: Evaluierungen von Strings sind generell ein Sicherheitsrisiko in Software, wenn Nutzereingaben durch solche Funktionen verarbeitet werden, da sie dem versierten Nutzer Zugriff auf das Programm oder gar das ganze System geben können.

Ein weiterer Unterschied ist der Rückgabewert: Während `str2double` als Wert für eine fehlgeschlagene Konversion `NaN` („Not a Number“) zurückgibt, ist das Rückgabeargument von `str2num` leer, wenn die Konversion fehlschlägt.

[2] `http://www.mathworks.de/de/help/matlab/ref/axes_props.html`