Eine kleine Anleitung zum Programmieren mit MATLab

Kathrin Smetana

April 9, 2010

Dies ist eine (stetig erweiterbare) Liste der Dinge, die man beachten sollte, wenn man mit MATLab programmieren, sprich M-Files schreiben, möchte.

1 Organisatorisches

1. Der Name des M-Files darf keine Sonderzeichen, sprich Zeichen, die entweder bereits MATLab-Operatoren sind (“+”, “(”, “)”, “-”), oder Umlaute (MATLab basiert auf der englischen Sprache) enthalten. Möchte man mehrere Wörter verwenden und diese trennen, so kann man dies mit den Zeichen “_ ”oder “.”tun. Ein Beispiel: Dies_wird_ein.korrekt.bezeichnetes.MFile

2. Ein M-File kann ausgeführt werden, indem man im Editor oben auf das weiße Blatt mit dem grünen Pfeil klickt, oder indem man im Command Window den Namen des Programms eingibt und Enter drückt.

3. Will man im Hauptprogramm selbstgeschriebene Funktionen aufrufen (Kapitel 4.2 im Skript), so sollte man auf folgende Dinge achten:

   (a) Die Funktion (im Skript im Programm `Rechteck.m` wäre zum Beispiel `Diagonale_Rechteck.m` eine Funktion) muss im selben Ordner gespeichert werden, da MATLab sonst die Funktion nicht findet.

   (b) Der Name der Funktion, welcher in der Definition gewählt wird (`function d =` $\underbrace{\text{Diagonale\_Rechteck}}_{\text{Funktionsname}}$ $\underbrace{(a,b)}_{\text{Argumente}}$), der Name des M-Files der Funktion `Diagonale_Rechteck.m` und der Funktionsaufruf im Hauptprogramm `d = Diagonale_Rechteck(a,b)` müssen alle gleich sein (kurz: überall steht `Diagonale_Rechteck`). Das sollte bei jeder Funktion nachgeprüft werden. Ein kleiner Tipp: hat man seine Funktion geschrieben und will diese speichern, schlägt MATLab automatisch den richtigen Namen vor.

   (c) Sowohl bei der Definition der Funktion, als auch beim Funktionsaufruf bitte die Argumente (hier `a` und `b`) nicht vergessen. Bei der Benennung des M-Files dagegen auf keinen Fall die Argumente dazu schreiben, sondern lediglich den Funktionsnamen verwenden.

   (d) Wenn man mit solchen Funktionen arbeitet kann es hilfreich sein, wenn man sich an mathematischen Funktionen orientiert. Ein Beispiel: `function y=f(x)`. Der Umgang geht dann völlig analog. Man würde ja nun beispielsweise auch nicht `y = f` schreiben oder eine Funktion mit `sin(x)` bezeichnen.

   (e) Es wird immer das Hauptprogramm und nie die Funktion ausgeführt.

2 Syntaxfehler vermeiden/beheben

1. Rechts oben neben der Scrollbar findet man im MATLab-Editor ein kleines buntes Quadrat. Dieses ist entweder

   - **grün** (= alles in bester Ordnung)
   - **orange** (= Warning. Das Programm hat keine Fehler, lässt sich aber eventuell noch verbessern. MATLab gibt beispielsweise eine Warning aus, wenn ein Semikolon fehlt und damit die Matrix, Funktion,.. im Command Window ausgegeben wird, oder wenn in einer Schleife eine Matrix definiert wird, die vorher noch nicht initialisiert wurde.
   - **rot** (= Syntaxfehler)

   Die Stelle welche das Warning oder den Syntaxfehler hervorruft markiert MATLab einerseits mit einem Strich neben der Scrollbar (wenn man neben den Strich scrolled, ist die Warning/ der Fehler in dem aktuellen Fenster zu sehen) und andererseits mit einer Welle unter dem Zeichen, welches die Warning/den Fehler hervorruft. Um Warnings oder Syntaxfehler zu beheben geht man mit der Maus auf die entsprechende Stelle im Code (also auf die Welle). MATLab gibt dann an wie der Fehler zu beheben bzw. der Code zu verbessern ist. Wenn man in dieser Anzeige auf `FIX` klickt, behebt MATLab den Fehler/die Warning selbstständig.

2. Schreibt man geschachtelte Schleifen oder Entscheidungen, so empfiehlt es sich die Anweisungen bzw. die neue Schleife/Entscheidung der Übersichtlichkeit wegen einzurücken.

3 Das Programm hat keine Syntaxfehler, funktioniert aber nicht, sondern liefert Fehler im Command Window

Zunächst sollte geprüft werden, ob alle Punkte unter Kapitel 1: Organisatorisches beachtet wurden. Die Fehlermeldung, dass eine Variable unbekannt ist, wird zum Beispiel unter anderem dadurch verursacht, dass nicht das Hauptprogramm sondern die Funktion ausgeführt wurde. Ist dies geschehen und es gibt trotzdem noch Fehler, so muss man mit Hilfe der Fehlermeldung die MATLab im Command Window ausgibt den Fehler finden. Geben wir zum Beispiel im Programm `Ableitung` (S. 43 im Skript) nach dem Befehl `f = @(x) sin(x);` den Befehl `plot(f)` ein und führen das Programm aus, so gibt MATLab im Command Window den folgenden Fehler aus:

??? Error using ==> plot
Conversion to double from function_handle is not possible.

Error in ==> Ableitung at 13
plot(f);

Die unteren beiden Zeilen geben an, wo wir einen Fehler gemacht haben. Die oberen beiden Zeilen zeigen welcher Fehler gemacht wurde. In diesem Fall haben wir den `plot`-Befehl falsch verwendet. Um den Fehler zu korrigieren verwenden wir die Hilfe und stellen fest, dass der `plot`-Befehl nur Vektoren akzeptiert. Daher die Meldung (frei übersetzt), dass MATLab das Funktion Handle nicht in einen Vektor umwandeln konnte. Daher muss nun entweder die Funktion vektorisiert werden (Intervall splitten und die Funktion in den Gitterpunkten auswerten -> S. 58 im Skript) oder man verwendet den Befehl `ezplot`. Kurz gesagt, sollte man bei der Fehlersuche also wie folgt vorgehen:

1. Wo wurde der Fehler gemacht (hier: Zeile 13, `plot`-Befehl)

2. Herausfinden was falsch gemacht wurde (hier: Befehl falsch verwendet)
3. Genaue Fehlerquelle finden (hier: es wurde kein Vektor sondern eine Funktion verwendet)
4. Fehler korrigieren (hier: mit Hilfe der von Literatur oder MATLab-Hilfe den richtigen Befehl finden)

4 Wie vermeide ich Fehler?

Generell sollte jede Programmzeile die man schreibt nochmals durchgelesen werden bevor man weiterschreibt. Ferner sollte das Programm so bald es ausführbar ist getestet werden, ob es das richtige tut (=keine Fehler produziert und richtige Ergebnisse liefert). Ist dies der Fall fährt man fort. Nach jedem noch so kleinen zusätzlichen Programmcode testet man wieder. Beim Programm `Ableitung.m` sollte man zunächst testen, ob die anonyme Funktion richtig definiert wurde. Dazu führt man das Programm aus. Funktioniert das alles, so verwendet man den `ezplot`-Befehl um zu schauen, ob auch die richtige Funktion definiert wurde (z.B. `x^2 + 5` statt `x^2 + 0.5`). Die Verwendung des `ezplot`-Befehls hat außerdem den Vorteil, dass man so weiter testen kann, ob die Funktion korrekt definiert wurde. Es kann nämlich passieren, dass erst die Verwendung von Funktionen, Matrizen,.. Fehler produzieren, welche bei der reinen Definition derselbigen noch nicht zu Tage getreten sind. Ein Beispiel: Wir schreiben ein kleines M-File mit den beiden Zeilen

```
f = @(x) x^2;
g = @(x) f + 3;
```

Führt man dieses Programm aus, so gibt MATLab keinen Fehler aus. Testen wir nun das Programm mit `ezplot`, schreiben also

```
f = @(x) x^2;
g = @(x) f + 3;
ezplot(g);
```

so gibt MATLab eine Fehlermeldung aus. Dies liegt nicht an der falschen Verwendung von `ezplot` (was man leicht nachprüfen kann, indem man `f` plottet), sondern daran, dass die anonyme Funktion `g` nicht korrekt definiert wurde. Wir fahren nun mit dem Test unseres Programms `Ableitung` fort. Als nächstes lässt man sich den Vektor `x` ausgeben, um die Unterteilung zu überprüfen. Besonders bei der Verwendung von Formeln (hier in `numAbleitung` der Differenzenquotient) oder ähnlichem, sollte nochmals ganz genau geprüft werden, ob alles richtig geschrieben wurde. Zum Schluss lässt man sich zur Kontrolle die Ableitung plotten. Besonders Programmieranfänger sollten beim Programmieren Schritt für Schritt vorgehen und nicht den Code auf einmal runterschreiben und dann testen.

5 Das Programm produziert zwar keine Fehler aber die Ergebnisse machen keinen Sinn

Um den Fehler zu finden lässt man sich, wie in Kapitel 4 beschrieben, Zwischenergebnisse (Matrizen, Werte von Variablen,..) ausgeben und plottet die verwendeten Funktionen. So kann man feststellen bis zu welcher Stelle das Programm richtig rechnet und ab wann Fehler auftreten. Auf diese Weise lässt sich der Fehler lokalisieren. Hat man dies getan, so geht man wie in Kapitel 3 vor um den Fehler zu beheben.