Programmierpraktikum im WS 2010/2011

Denny Otten, V5-142, dotten@math.uni-bielefeld.de

2. Skripte und Funktionen

1 Skripte (siehe `>> help script`)

Oft ist es nötig mehrere Zeilen einzugeben um ein bestimmtes Ergebnis zu erhalten (z.B. beschriftete Skizzen). Wenn man die gleichen Zeilen mehrfach ausführen muss, ist es natürlich sinnvoll dies zu automatisieren. Die einfachste Möglichkeit, Kommandos nacheinander ablaufen zu lassen, ist, sie in eine Text-Datei mit der Endung `.m` zu schreiben (ein sogenanntes `m-file`). Solch eine Datei kannst Du natürlich mit Deinem Lieblingstexteditor oder aber am einfachsten in Matlab direkt erzeugen. Gib dazu

`>> edit meinskript` am Matlab-Prompt ein; die Dateiendung `.m` wird automatisch ergänzt.

Alternativ geht auch `File`→`New`→`Blank M-File` mit der Maus. Ein geschriebenes und gespeichertes Skript kannst Du dann starten durch

```
>> meinskript
```

Ein Skript benimmt sich so, als wären die Zeilen des Skriptes wirklich alle nacheinander am Matlab-Prompt eingegeben worden. Daher kannst Du auch Variablen Deines Arbeitsplatzes innerhalb eines Skriptes benutzen (die Möglichkeiten und Gefahren, die sich daraus ergeben werden wir aber erst später besprechen).

Aufgabe 1. *Schreibe die Befehle zum Plotten einer Funktion, sowie zur Beschriftung der Achsen und eines Titels in ein Skript und führe es aus. (Für die nötigen Befehle siehe den ersten Zettel.) Mache das selbe so effizient wie möglich für eine andere Funktion (**Hinweis:** benutze* `!cp` *und* `edit`*).*

Wichtig: Falls eine Berechnung nicht zu einem Ende kommt, so kannst Du sie in Matlab mit der Tastenkombination `Ctrl-c` abbrechen!

2 Funktionen

Die Möglichkeiten von Skripten sind sehr limitiert, da man keine Ein- und Ausgabeparameter mit übergeben kann. Außerdem ist es bei Skripten notwendig zu wissen, wie sie intern funktionieren und welche Variablen sie verändern, da ein Aufruf sonst ungeahnte Nebenwirkungen haben kann. Besser ist es eine Art "Black Box" zu haben, die bei einer gewissen Eingabe eine bestimmte, wohldefinierte Ausgabe liefert ohne sonst irgendetwas zu beeinflussen.
All diese Vorteile haben Funktionen. Sie zu erzeugen ist auf verschiedene Arten möglich.

Funktions m-files

Eine Funktion `FunktionsName` kann definiert werden, indem die Funktionsdefinition – genau wie zuvor das Skript – in einer Datei mit dem Namen `FunktionsName.m` gespeichert wird. Diese Datei muss/sollte wie folgt aussehen (siehe: `help function` sowie den Anhang am Schluss dieses Arbeitsblattes).

```
function out = FunktionsName(in)
%FUNKTIONSNAME Kurzdoku der Funktion (wird bei lookfor durchsucht)
% Ausfuehrliche Doku der Funktion (wird bei help FunktionsName angezeigt)
  Anweisung1  % Kommentar
```

```
  Anweisung2
  ...
  out = ... ; % Hier wird der Ausgabeparameter out gesetzt
end
```

bzw. mit m Ein- und n Ausgabeparametern

```
function [out1, ..., outn] = FunktionsName(in1, ..., inm)
%FUNKTIONSNAME Kurzdoku
% Ausfuehrliche Doku
  Anweisung1
  Anweisung2
  ...
  out1 = ... ;
  ...
  outn = ... ;
end
```

Der Aufruf erfolgt mit `[out1,.., outn] = FunktionsName(in1, ..., inm)`. Dabei muss sich die Datei FunktionsName.m im Matlab Suchpfad befinden: das ist das aktuelle Verzeichnis, sowie alle Verzeichnisse, die bei `path` angezeigt werden. Ob eine Funktion existiert bzw. gefunden wird, kannst Du mit `exist FunktionsName` (siehe: `help exist`) herausfinden.

Zunächst ein paar Beispiele:

```
function summe=test1(A)
%TEST1 sum of all entries of a matrix        hinter % kommen die Kommentare
% die erste Kommentarzeile ist besonders wichtig!
  [m,n]=size(A); % m=Zeilen von A, n=Spalten von A
  summe=0; % Initialisierung
  for i=1:m % i laeuft von 1 bis m
    for j=1:n % j laeuft von 1 bis n
      summe=summe+A(i,j);
    end
  end
end % dieses end ist nicht noetig, es schliesst die Funktion
```

Listing 1: `test1.m` (Unter `test1.m` abspeichern!)

Die Funktion kann nun wie jede andere Matlabfunktion benutzt werden:

```
>> help test1
>> A=rand(3,4)
>> s=test1(A)
>> type test1     gibt die Funktionsdefinition aus
```

Die erste Kommentarzeile ist besonders wichtig, sie **muss** ganz links beginnen, und **sollte** den Funktionsnamen **großgeschrieben**, sowie eine **am besten englische** Beschreibung besitzen. Diese Zeile wird mit `lookfor` durchsucht. Probiere es aus: Gib `lookfor sum` am Prompt ein! Außer der von Dir programmierten Funktion `test1` sollten noch weitere, in Matlab implementierte aufgeführt werden. Die weiteren Kommentarzeilen, die direkt folgen werden bei `help test1` angezeigt.
Funktionen können auch mehrere Variablen zurückgeben:

```
function [summe,prod,spur]=test2(A)
%TEST2 compute sum, product and trace of a matrix
% [summe,produkt,spur]=TEST2(A), falls A nicht quadratisch ist spur=0
```

```
  [m,n]=size(A);
  summe=0; prod=1; spur=0; % Initialisierung
  for i=1:m
    if (m==n) % wenn m = n, dann ...
      spur=spur+A(i,i);
    end
    for j=1:n
      summe=summe+A(i,j);
      prod=prod*A(i,j);
    end
  end
end
```

Listing 2: `test2.m` (Unter `test2.m` abspeichern!)

Der Aufruf sieht dann wie folgt aus:

```
>> help test2
>> A=[1 2;3 4]
>> test2(A)                Nur das erste Output-Argument wird ausgegeben.
>> [su,pr,sp]=test2(A);
>> su,pr,sp
>> B=[1 2 3;4 5 6]
>> [su,pr,sp]=test2(B)
```

Aufgabe 2. *Erzeuge eine Funktion die eine Matrix als Eingabe hat und die Anzahl der Elemente zurückgibt. Teste mit* `>> exist FunktionsName`*, ob Matlab die Funktion findet. Schaue Dir auch* `help exist` *an.*

Aufgabe 3. *Schaue Dir mit* `pathtool` *den aktuellen Suchpfad an und probiere, wie Du den Suchpfad damit verändern kannst. Benutze alternativ die Funktion* `addpath` *um ein Verzeichnis per Code zum Suchpfad hinzuzufügen (*`help addpath`*). Überlege, warum es nützlich ist, dass Verzeichnisse zum Suchpfad hinzugefügt werden können.*

Aufgabe 4. *Schreibe ein Skript* `MatAddSkript.m`*, dass zu der Variablen* `A` *eins hinzuaddiert (d.h. in dem Skript soll nur* `A=A+1;` *stehen). Schreibe eine Funktion* `MatAddFun.m`*, die dieselbe Aufgabe erledigt. Führe dann folgendes aus:*

```
>> clear; A= 1
>> B=MatAddFun(A)
>> A
>> MatAddSkript
>> A
```

Was fällt Dir auf?

Haupt und Unterfunktionen

Es können auch mehrere Funktionen in eine Datei geschrieben werden. Nach außen sichtbar ist aber nur die erste Funktion, die anderen dienen nur als unsichtbare Hilfsfunktionen. In der Datei `FunktionHaupt.m` kann also folgender Code stehen:

```
function out = FunktionHaupt(in)
%FUNKTIONHAUPT Nur diese nach aussen sichtbar
  N=5;
  y = FunktionHilf(N, in); % lokale Hilfsfunktion
  out = 2*y + in;
```

```
end

function x = FunktionHilf(z1, z2)
%FUNKTIONHILF von aussen unsichtbar
   x = z1.^2 + sin(z2);
end
```

Listing 3: `FunktionHaupt.m`

Verschachtelte Funktionen

Es ist auch möglich Funktionen zu verschachteln, das heißt die nächste Funktion wird noch innerhalb der Funktionsdefinition der ersten definiert. Hier *muss* jetzt die Funktion mit `end` abgeschlossen werden!

```
function out = FunktionHaupt(in)
  N=5;
  y = FunktionHilf(N, 2.4);
  out = 2*y + in;

  function x = FunktionHilf(z1, z2)
    x = z1.^2 + sin(z2);
  end
end   % das 2. end schliesst FunktionHilf ein
```

Listing 4: `FunktionHaupt.m`

Aufgabe 5. *Teste die Sichtbarkeit von Variablen in den Funktionen und Unterfunktionen, indem Du verschiedene Variablen in der Hauptfunktion und den Hilfsfunktionen definierst, diese dann jeweils veränderst und ausgibst. Du kannst Dir die aktuellen Variablen zum Beispiel mit* `who` *bzw.* `whos` *anzeigen lassen. Füge* `who` *oder* `whos` *an verschiedenen Stellen im Code ein. Welche Variablen sind jeweils sichtbar? (Du kannst Sie auch im Reiter Workspace im Hauptfenster anschauen, wenn Du das Programm Schritt für Schritt mit dem Debugger durchgehst.)*
Gibt es Unterschiede bei verschachtelten Funktionen?

Anonyme Funktionen

Anonyme Funktionen erlauben es innerhalb einer Zeile Funktionen zu definieren, die intern keine Variablen benötigen. Besonders nützlich sind anonyme Funktionen im Zusammenhang mit Funktionen, die Funktionen als Parameter benötigen (z.B. Integrieren, Differenzieren,. . .), weil es so möglich ist bei Funktionen einige Parameter festzulegen und andere variabel zu lassen.
Allgemein sieht die Definition einer anonymen Funktion wie folgt aus:
`>> FunktionsName=@(EingabeParameter) Ausdruck`
Beispielsweise lässt sich $f(x, y) = e^x - y$ so definieren:
`>> f = @ (x,y) exp(x)-y`
und um mehrere Ausgabeparameter möglich zu machen muss man die Funktion `deal` (siehe `help deal`) benutzen. Beispiel für die Verwendung von `deal`:

`>> [a,b,c]=[1,2,3]`	geht nicht, da nicht klar ist, wie die Elemente zugewiesen werden sollen
`>> [a,b,c]=deal(1,2,3)`	so geht es

Das kann man dann in der Definition einer anonymen Funktion wie folgt benutzen:

```
>> g = @(x) deal(sin(x), x^2+4);
>> [a,b] = g(1);
```

Um Parameter in anderen Funktionen festzulegen (oder einen Schnitt zu betrachten), kann man wie folgt vorgehen:

```
>> h = @(x) f(x,1)          % f(x,y)=e^x-y von oben
>> fplot(h,[-1,1])          % zeichne die Funktion h(x)=f(x,1) mit fplot
```

Sobald anonyme Funktionen definiert wurden sind sie fixiert, das heißt eine nachträgliche Änderung eines Parameters ändert die anonyme Funktion nicht mehr. Beispiel (kommentiere es):

```
>> alpha = 2;
>> r = @(x,y,z) x^2+y^2-alpha*z^2
>> r(1,1,1)
>> alpha = 0;
>> r(1,1,1)
>> r = @(x,y,z) x^2+y^2-alpha*z^2
>> r(1,1,1)
```

Aufgabe 6. *Definiere eine anonyme Funktion* `f` *für* $f(x,y) = \frac{\sin(x)}{1+x^2*y^2}$ *und zeichne sie mit* `ezmesh(f)`. *Definiere dann eine anonyme Funktion* h, *die gerade* f *auf der Diagonalen* $\{x = y\}$ *sein soll und zeichne diese mit* `ezplot(h)`.
Experimentiere mit weiteren Beispielen.

inline-Funktionen

Die letzte Möglichkeit Funktionen in Matlab zu definieren, ist mit Hilfe des Befehls `inline`. Der Befehl macht aus einem einfachen String (also gewöhnlichem Text, wie er zum Beispiel aus einer Benutzereingabe kommt) eine Funktion. Auch hier dürfen in der Funktion keine internen Variablen benutzt werden. Inline Funktionen sind Funktionsobjekte (keine Funktionshandles! (siehe unten)). Beispiel:

`>> g=inline('exp(x)-1')`	Definition der Inline Funktion
`>> g(2)`	Aufruf der Funktion
`>> h=inline('log(a*x)/(1+y^2)')`	Matlab bestimmt automatisch die Variablen
`>> h1=inline('log(a*x)/(1+y^2)','x','y','a')`	
`>> f2d=inline('[sin(x),cos(y)]')`	auch mehrere Ausgabevariablen sind möglich

Eine größere Bedeutung hat der `inline`-Befehl wegen des folgenden einfachen Skriptes, dass Du unter `interaktiv.m` speichern kannst:

```
funktext = input('Gib eine Funktion R->R ein: ','s');
f=inline(funktext);
ezplot(f);
title(funktext);
xlabel('x');
ylabel('f(x)');
```

Listing 5: `interaktiv.m`

Ferner kann man aus einer `inline`-Funktion sehr einfach eine anonyme Funktion erstellen:

`>> g = inline('sin(x)+cos(y)^2')`	Def. inline Funktion
`>> f = @(a,b) g(a,b)`	Def. anonyme Funktion, die g auswertet. Dabei muss nur die Zahl, nicht jedoch die Namen der Variablen zusammenpassen

Aufgabe 7. *Kommentiere den Code von* `interaktiv.m` *und experimentiere mit verschiedenen Eingaben. Bei was für Eingaben bekommst Du Fehlermeldungen?*

Funktionshandles – Funktionen als Übergabeparameter

Durch die Definition einer Anonymen Funktion erhält man ein **Funktionshandle**, das auf die definierte Funktion zeigt. Es können auch Funktionshandles erzeugt werden, die auf bereits existierende Funktionen zeigen:

```
>> hf = @FunktionsName; % definiert das Handle auf die Funktion FunktionsName
                        % die in FunktionsName.m definiert ist
>> ev = @(f, x) f(x)    % definiert eine anonyme Funktion ev, die eine Funktion
>> ev(hf, 4)            % f bei x auswertet
```

Man kann auch Handles auf eingebaute Matlab-Funktionen definieren, etwa `hg = @sin`. Auf diese Weise lassen sich bequem Funktionen als Parameter an andere Funktionen übergeben. Typische Beispiele für Funktionen, die Funktionen als Eingabeparameter benötigen, sind unter anderem die Differentiation und die Integration.

```
function out = auswerte(funk,x)
%AUSWERTE wertet FUNK bei X aus
% macht dasselbe wie oben ev
  out = funk(x);
end
```

Kann nur mit handles oder Funktionsobjekten umgehen

```
function out = auswerte2(funk,x)
%AUSWERTE wertet FUNK bei X aus
% macht dasselbe wie oben ev
  out = feval(funk,x);
end
```

Kann auch mit Funktionsnamen umgehen

Diese Funktionen kann man dann wie folgt benutzen

```
>> f=@test1;               % erzeugt einen Handle auf die Funktion test1 von oben
>> auswerte(f,rand(3))     % wertet mit der ersten Version aus (Handle!)
>> auswerte2(f,rand(3))    % macht das gleiche
>> auswerte('sin',5)       % kann das nicht, da 'sin' kein Handle ist
>> auswerte2('sin',5)      % dies geht
```

Aufgabe 8. *Schreibe eine Funktion* `vdiffqu`*, die eine Funktion, sowie einen Punkt x und eine Schrittweite h als Eingabe haben soll und als Ergebnis den vorwärtsgenommenen Differenzenquotienten $\frac{f(x+h)-f(x)}{h}$ haben soll.*
Experimentiere mit verschiedenen Eingaben. Was darfst Du nicht eingeben?

Aufgabe 9. *Schreibe eine Funktion* `fcnplot`*, die eine Funktion sowie ein Intervall als Eingabe hat, welche die Funktion im Intervall mit* `ezplot` *oder* `plot` *zeichnet (und den Plot eventuell noch mit Beschriftungen verschönert). (**Hinweis:** Benutze die Hilfe zu* `ezplot` *und* `plot`*.)*

Kontrollstrukturen

Matlab bietet die Standardkontrollstrukturen Verzweigung und Schleife, mit denen der Programmablauf gesteuert werden kann.

Allgemeine Syntax	Beispiel
Verzweigung mit `if` und `elseif/else` (`help if, help elseif`)	

`if Bedingung`
`  ...`
`elseif Bedingung % kann wegfallen`
`  ...`
`else  % kann wegfallen`
`  ...`
`end`
	`if a > b`
	`  min_val = b;`
	`else`
	`  min_val = a;`
	`end`

`switch-case` Verzweigung (`help switch`)	

`switch Auswahl`
`  case Ausdruck1`
`  ...`
`  case Ausdruck2`
`  ...`
`  otherwise`
`  ...`
`end`
	`switch tier`
	`  case 'hund'`
	`  disp('wau wau')`
	`  case 'katze'`
	`  disp('miau')`
	`  case 'maus'`
	`  disp('piep')`
	`  otherwise`
	`  error('Bin ich ein Tier?')`
	`end`

`for` - Schleifen (`help for`)	

`for n=Bereich`
`  ...`
`end`
	`summe = 0;  % Variable summe auf 0 setzen`
	`for n=1:10`
	`  summe = summe + n;`
	`end`

`while` - Schleifen (`help while`)	

`while Bedingung % fuehre Code aus`
`  ...  % solange`
`end  % Bedingung wahr`
	`while n < n_max`
	`  summe = summe + n;`
	`end`

Bedingungen sind logische Ausdrücke; sie haben den Wert `true` oder `false`. Diese erhält man z.Bsp. durch Anwendung eines relationalen Operators: `a op b`, wobei `op` für

`<, >, <=, >=, == (gleich), ~= (ungleich)`

steht. Mehrere Bedingungen können mit

`&& (logisches UND)` `|| (logisches ODER)` `~ (logisches NICHT)`

logisch verknüpft werden. (Bei `&&` und `||` handelt es sich um logische Operatoren, die nur auf Skalaren arbeiten!)

Aufgabe 10. *Ergänze die obigen Beispielfragmente so, dass jeweils sinnvolle Funktionen daraus werden und teste diese mit verschiedenen Eingaben.*

Aufgabe 11. *Schreibe eine Funktion* `dice`*, die einen Würfel simulieren soll. Wenn die Funktion aufgerufen wird (*`dice()`*), so soll sie also eine ganze Zufallszahl zwischen 1 und 6 liefern. Benutze dazu die Funktionen* `rand` *und* `ceil` *(siehe:* `help rand` *und* `help ceil`*).*
Schreibe jetzt eine Funktion, die als Eingabe die Anzahl der Würfe und als Ausgabe die Anzahl der gewürfelten Sechsen zurückgibt. Benutze dabei eine `for`*-Schleife, um die Würfe zu simulieren.*
Ändere das Programm danach so ab, dass es die Zahl der Würfe mit geraden Augenzahlen zählt.

Schließlich schreibe ein Programm, das als Eingabe eine natürliche Zahl n hat, und als Ergebnis liefert, wie oft es Würfeln musste bis die n-te Sechs gefallen ist.

Anhang: Regeln für die Erzeugung von Funktions m-files

1. Der Name der Funktion und der Datei sollte identisch sein. De facto wird stets die erste Funktion in der Datei bei einem Aufruf ausgeführt.

2. Der Name sollte kleingeschrieben werden, da Matlab zwischen Groß- und Kleinschreibung unterscheidet.

3. Der erste Kommentarblock nach der Funktionsdefinition wird als Hilfetext der Funktion bei `help FunktionsName` angezeigt. Er sollte (wie bei `test2` vorgemacht) stets die Funktionsdefinition (also was für Eingaben und was für Ausgaben es gibt) beinhalten, da man sich nicht immer alles merken kann und oft auch die Reihenfolge der Parameter nach einiger Zeit nicht mehr kennt.

4. Die allererste Kommentarzeile (auch H1 Zeile genannt) wird bei `lookfor` durchsucht, daher sollte sie den Funktionsnamen, sowie eine stichwortartige Beschreibung der Funktion enthalten (wenn möglich in Englisch).

5. Funktionsnamen in Hilfetexten werden in Großbuchstaben geschrieben, der Aufruf der Funktionen muss dann natürchlich die richtige Schreibweise benutzen.

6. Eine Funktion endet nach der letzten Zeile, einzige Ausnahmen sind die Befehle `error` und `return`, die ein vorzeitiges Abbrechen bzw. Beenden nach sich ziehen.

7. In Funktions m-files können mehrere Funktionen definiert werden. Alle nach der ersten Funktion sind lokal definiert und nicht direkt von Außen aufrufbar. Diese Unterfunktionen können in beliebiger Reihenfolge definiert werden.

8. Guter Stil ist es die Namen von Unterfunktionen mit `local_` beginnen lassen. Damit kann man in der Hauptfunktion klar erkennen, ob eine Unterfunktion, oder eine andere Matlab-Funktion aufgerufen wird.

9. Zum Schluss: Zu viele Kommentare sind besser als zu wenige!