Schleifen

Stand: 25.9.2014

A) WHILE-SCHLEIFEN

Schleifen sind ein Programmierkonzept, mit dem man eine Abfolge von Befehlen in Abhängigkeit von einem Kriterium mehrfach wiederholen kann. Insofern gibt es eine starke Ähnlichkeit zu Fallunterscheidungen (*if*-Abfragen): die Befehle werden nur dann ausgeführt, wenn ein Kriterium erfüllt ist. Im Gegensatz zu Fallunterscheidungen erfolgt diese Ausführung bei Schleifen jedoch normalerweise nicht nur einmal sondern wird solange wiederholt, bis die Bedingung nicht mehr erfüllt ist. (Beim Kochrezept ist das der Unterschied zwischen "je nach Geschmack Zucker oder Salz beifügen" und "Rühren bis sich alle Klumpen aufgelöst haben".)

Die allgemeine Syntax einer while-Schleife (also einer Schleife, deren Befehle so lange immer wieder ausgeführt werden, bis eine Bedingung nicht mehr erfüllt ist) lautet

- *Initialisierung Test-Variable*
- *while Bedingung*
  Befehle
  Aenderung der Test-Variablen
- *end*

Wichtig zu beachten ist dabei

- Die Bedingung muss von (mindestens) einer Test-Variablen abhängen
- Dieser Test-Variablen muss vor Beginn der Schleife ein Wert zugewiesen werden
- Der Wert dieser Variablen muss sich innerhalb der Schleife ändern können
- Achtung! Wenn die Bedingung immer wahr bleibt wird die Schleife unendlich oft ausgeführt. Es kommt zu einer sogenannten Endlosschleife.
- Endlosschleifen (oder auch andere Programme, die einem zu lange dauern) kann man in Matlab mit `ctrl-c` abbrechen.

T4A1) Ein Beispiel für eine while-Schleife könnte folgendes Programm sein (Download: [guesses.m])

```
function number=guesses
% function number=guesses
% Der Computer wählt zufällig eine ganze Zahl zwischen 1
und 20 aus. Der
```

```
% Benutzer versucht, dieses zu erraten.
% Rueckgabewert:
% number ist die Anzahl Versuche, die der Benutzer
benoetigt hat.
secret=ceil(rand*20);
% Zufaellige ganze Zahl zwischen 1 und 20
guessed=false;
% Testvariable initialisieren
number=0;
% Anzahl der Versuche mit 0 initialisieren
while guessed==false
% Solange die Zahl nicht geraten ist
     user_guess=input('Bitte geben Sie eine ganze Zahl
zwischen 1 und 20 ein:');
     %neue Zahl vom Benutzer einlesen
     number=number+1;   % Zaehler hochsetzen
     if user_guess==secret
     % Wenn der Benutzer die Zahl erraten hat
          guessed=true;
          % Testvariable entsprechend aktualisieren
     end
end
disp(['Sie haben ',num2str(number),' Versuche gebraucht',
number]);
```

- Probieren Sie das Programm aus.
- Verfolgen Sie im *debugging*-Modus, wie sich die Variablen verändern, während das Programm läuft.
- Erweitern Sie es so, dass der Benutzer gewarnt wird, wenn er eine unsinnige Eingabe macht.

T4A2) Warum geht folgende Schleife schief?

```
a=1; b=10;
while a<b
     b=2*b
end
sprintf('a=%g, b=%g',a,b)
```

(Denken Sie an *ctrl-c*, wenn Sie diese Schleife ausprobieren...)

T4A3) Schreiben Sie eine Funktion, die eine Zahl als Eingabeargument bekommt und solange Zufallszahlen auf dem Bildschirm ausgibt, bis die gezogene Zufallszahl größer als die vom Benutzer der Funktion übergebene Zahl ist.

- Benutzen Sie die Funktion *rand(1)*, *um* eine Zufallszahl zwischen 0 und 1 zu erzeugen.
- Denken Sie daran, eine Abfrage zu machen, ob die Eingabe des Benutzers sinnvoll ist.

B) FOR-SCHLEIFEN

Prinzipiell reichen while-Schleifen für alle Anwendungen. Häufig weiss man jedoch von Anfang an wie viele Schritte erledigt werden müssen. In diesen Fällen ist es praktischer, eine Zählschleife zu benutzen. Diese wird in Matlab durch das Schlüsselwort *for* gestartet und benutzt in der Regel den Doppelpunktoperator, um den zu betrachtenden Zahlenbereich festzulegen.

Die allgemeine Syntax einer *for*-Schleife lautet:

```
for zaehler=anfangswert:schrittweite:endwert
      Befehle
end
```

Für die Variable, die in der *for*-Schleife als Zähler verwendet wird, gibt es ein Paar Regeln:

- Der Zähler kann innerhalb der Schleife verwendet werden (das ist sogar sehr oft der Fall).
- Der Wert des Zählers darf innerhalb der Schleife jedoch nicht verändert werden.
- Im Prinzip können beliebige Vektoren für den Zähler benutzt werden, insbesondere auch durch den Doppelpunkt mit anderer Schrittweite erzeugte Vektoren (z.B. *a=7:0.1:8*) oder auch Vektoren, deren Elemente explizit angegeben werden (z.B. `wert=[0 1 2 4 8]`)

Häufig benutzt man geschachtelte *for*-Schleifen, z.B. um jedes Element einer Matrix `M` einzeln zu bearbeiten. Solche Programme sehen normalerweise so aus:

```
Initialisiere M
[zeilen,spalten]=size(M)
for z=1:zeilen
      for s=1:spalten
          Befehle
      end
end
```

T4B1) Probieren Sie folgendes Programm aus:

```
aktuell=1;
for z=1:1:10
     aktuell=aktuell*z;
     sprintf('aktueller Wert: %i ',aktuell)
end
```

- Was berechnet dieses Programm?

- Verfolgen Sie im *debugging*-Modus, wie sich die Variablen verändern, während das Programm läuft.
- Übertragen Sie das Programm in eine *while*-Schleife.

T4B2) Der (uns ja schon lange bekannte) Doppelpunkt-Operator ist eine implizite *for*-Schleife. Schreiben Sie mit Hilfe von *for*-Schleifen neue Versionen der alten Aufgaben:

a) eine Funktion, die die Werte auf der Diagonalen einer quadratischen Matrix zurückgibt

b) Erzeugen Sie ein 8*8 Schachbrett aus 0 und 1 und sehen es mit *imagesc* an.

*) Erzeugen Sie spaßeshalber auch ein 25*25 Schachbrett und sehen es an.

T4B3) Was ist der Unterschied zwischen dem Doppelpunktoperator und *for*-Schleifen? Fällt Ihnen ein Beispiel ein, das man mit einer *for*-Schleife lösen kann, aber nicht mit einem Doppelpunktoperator? Gibt es auch umgekehrte Beispiele?

T4B4) Schreiben Sie eine Funktion, die eine Matrix und einen Schwellwert als Eingabeargumente bekommt und den Wert zurückgibt, wie viele der Matrixelemente größer als dieser Schwellwert sind.

- a) Benutzen Sie zwei geschachtelte *for*-Schleifen
- b) Benutzen Sie nur eine `for`-Schleife
- c) Benutzen Sie eine *while*-Schleife
- *d) Finden Sie einen Weg, diese Aufgabe ohne Schleife zu lösen?

T4B5) Schleifen können einem auch helfen, wenn man mehr Daten grafisch darstellen möchte, als man sinnvoll in einer Abbildung unterbringen kann. Um mehrere Abbildungen zu erzeugen, gibt es zwei Möglichkeiten in Matlab:

- Mehrere Grafikfenster:
  - *figure* öffnet ein zusätzliches Grafikfenster
  - *figure(1)* macht die Abbildung mit der Nummer `1`zum aktiven Fenster, in dem der nächste plot-Befehl Daten darstellt
  - *clf* löscht den Inhalt des aktuellen Fensters
  - *close(2)* schließt das Fenster mit der Nummer `2`
  - *close all* schließt alle Grafikfenster
- Unter-Abbildungen im gleichen Grafikfenster:
  - *subplot(zeilenzahl,spaltenzahl,aktiv)* erzeugt man eine "Matrix" aus kleinen Abbildungen. Z.B. erzeugt

    - *subplot(2,3,5)* mit 2 Zeilen und 3 Spalten, wobei das 5. Unter-Fenster aktiv ist.
    - Achtung! Aus mir unverständlichen Gründen werden die Abbildungsfenster nicht wie bei der linearen Indizierung von Matrizen entlang der Spalten, sondern entlang der Zeilen durchgezählt!
    - Im aktiven Unter-Fenster diesem kann der *plot*-Befehl (und alle dazugehörigen Formatierungen und Beschriftungen) normal verwendet werden.
    - Um ein anderes Unter-Fenster zu aktivieren, wird wiederum der Befehl *subplot* mit der entsprechenden Angabe für die aktuelle Nummer genutzt (z.B.*subplot(2,3,1)*).
- a) Schreiben Sie ein Programm, das für eine Variable Anzahl `N` jeweils ein Grafikfenster und plotten Sie für den Vektor *x=-5:0.1:5* die Kurve der jeweiligen Funktion*x.^N* hinein.
- *b) Schreiben Sie ein Programm, das mit `subplot`mehrere Graphen in Unter-Abbildungen darstellt.
    - Geplottet wird die Funktion *x.^N* für den jeweiligen Vektor *x=[-x_grenze:0.1:x_grenze]*
    - Für `N` werden die Werte *N=[1, 2, 3, 4]* und für den Vektor `x` die Grenzwerte `x_grenze=[5, 10, 20]`verwendet.
    - Jede der 12 Kombinationsmöglichkeiten dieser beiden Parameter soll in einer eigenen Unter-Abbildung dargestellt werden.
    - Sortieren Sie die Abbildungen so, dass
        - Parabeln mit gleichem Definitionsbereich (also gleichem Wert von *x_grenze*) jeweils nebeneinander in der gleichen "Zeile" der Abbildungsfenster dargestellt sind
        - Parabeln mit gleichem Exponenten *N* jeweils in der gleichen "Spalte" untereinander angeordnet sind.

T4B6) Schleifen haben nicht nur Vorteile... in Matlab haben sie den massiven Nachteil, dass sie Programme sehr langsam machen können (insbesondere geschachtelte Schleifen.) Deshalb ist es immer eleganter, für ein Problem eine Lösung mit Matrixoperationen zu finden als mit Schleifen. Wir probieren das hier mal aus. Die Befehlskombination *tic Befehle toc* gibt auf dem Bildschirm aus, wie viel Zeit zwischen *tic* und *toc*, also beim Abarbeiten der Befehle vergangen ist. Benutzen Sie *tic* und *toc*, um zu untersuchen, wie schnell sich eine 1000 x 1000 Matrix erzeugen lässt, deren Elemente alle den Wert `7` haben. Erzeugen Sie diese Matrix:

- a) Mit dem Befehl *ones*

- b) Mit einer Schleife
- c) Mit zwei geschachtelten Schleifen
- d) Gibt es einen Unterschied, wenn man `for` oder `while` benutzt?

Arbeiten Sie mit einer der erzeugten Matrizen weiter und bestimmen Sie die Zeit, um auf jedes der Elemente den Wert 10 zu addieren:

- e) Mit Matrixaddition
- f) Mit einer Schleife
- g) Mit zwei geschachtelten Schleifen
- h) Gibt es einen Unterschied, wenn man for oder while benutzt?
-

T4B7) Die letzte Aufgabe ist eine gute Anwendung, um den Matlab Profiler einzuführen. Dieses Matlab Tool untersucht ein Programm daraufhin, welche Zeilen am meisten CPU Zeit verbrauchen. Man startet den Profiler entweder durch Klicken auf `desktop -> profiler` oder man nutzt die Befehle `profile` um die Analyse zu starten und `profile viewer`, um ein Fenster zu öffnen, in dem die Resultate gezeigt werden. z.B.

```
clear all
profile on
T4B6a_jutta
profile viewer
```

Probieren Sie den Profiler für Ihre Programme aus T4B6 aus.

C) BEISPIEL: POPULATIONSWACHSTUM

Die Erklärung der Entwicklung von Populationsgrößen hat eine lange Tradition in den Biowissenschaften. Hier wollen wir zunächst das einfachste Beispiel betrachten, das lineare Modell für exponentielles Wachstum:

Nehmen wir an, wir haben in unserem Labor eine Population von `P` Fliegen und zählen jeden Tag, wie groß `P` ist. Wenn wir das eine Weile lang machen, werden wir bestimmte Regelmäßigkeiten feststellen, z.B. könnte es sein, dass jeden Tag 20% der Fliegen neu geschlüpft sind und 10% der Fliegen sterben. Wir nennen diese Zahlen `f=0.2` (Fertilität) und `m=0.1` (Mortalität). (Natürlich gelten solche Regeln nur im Mittel - mit statistischen Schwankungen rund um diese Mittelwerte beschäftigen wir uns nächste Woche.)

Nun können wir mit Hilfe dieser Regeln voraussagen, wie die Population wachsen wird. Betrachten wir die Population P_t am Tag t, dann wird diese Anzahl sich bis zum nächsten Tag ändern um

$dP_t = f*P_t - m*P_t = (f-m)*P_t$.

Also ergibt sich als Abhängigkeit für die Populationsgröße P_{t+1}am Tag

t+1 von der Populationsgröße am vorhergehenden Tag *t*:

$P_{t+1} = P_t + dP_t = P_t + (f-m)*P_t = (1+f-m)*P_t$

T4C1) Nehmen wir an, die Population mit `f=0.2` (Fertilität) und `m=0.1` (Mortalität) hat am Tag `t=1` eine Größe von `P=500`.

- Wie viele Tiere sind es am nächsten Tag?
- Rechnen sie die Populationsgröße für einige weitere Tage aus und schreiben Sie diese in einen Vektor *P*, so dass das *t*-te Element `P(t)` des Vektors den Wert P_t am Tag *t* enthält.

T4C2) Schreiben Sie ein Programm, das für die vorgegebenen Werte die Populationsgröße der nächsten `T=100` Tage ausrechnet und in einen Vektor schreibt.

- Lassen Sie den Vektor graphisch ausgeben. Denken Sie daran, die Achsen zu beschriften.
- Verallgemeinern Sie Ihr Programm so, dass sie damit einfach verschiedene Werte für die Modell-Parameter f*, m, P(1)* und *T* ausprobieren können.
- Probieren Sie für einige verschiedene Mortalitäts- und Fertilitäts-Raten aus, wie die Population sich entwickelt. Was passiert z.B. bei *f<m*? Wie wirken sich *P(1)* und *T* auf die Entwicklung der Population aus?
- Machen Sie aus Ihrem Programm eine Funktion, mit der der Benutzer die Modell-Parameter übergeben kann und den Vektor der Populationsgröße als Rückgabe erhält.

*T4C3) Schreiben Sie ein Programm das statt einer festen Anzahl von Tagen einen Schwellenwert als Parameter bekommt und so lange die Populationsgröße des nächsten Tages ausrechnet, bis der Schwellenwert erreicht wird.

- Überlegen Sie sich, welche Probleme bei der Benutzung ungeeigneter Schwellenwerte auftreten können und wie man diese umgehen kann.

**T4C4) Man kann die Populationsentwicklung mit einer Funktion komplett ohne Schleife berechnen. Wie könnte das gehen?

D) HAUSAUFGABE:

AUFGABEN ZUM SCHLEIFEN-ÜBEN:

T4H1) Erzeugen Sie noch einmal eine Multiplikationstabelle für das kleine Einmaleins - diesmal mit Schleifen.

*T4H2) Schreiben Sie eine Funktion, die in einem beliebigen Vektor die Reihenfolge der Elemente umdreht. Z.B. aus [19 7 30] wird [30 7 19].

T4H3) Programmieren Sie ein Ratespiel, in dem ein Matlabprogramm mit so wenigen ja/nein Fragen wie möglich eine ganze Zahl zwischen 1 und 100 errät, die Sie sich ausdenken. Lassen Sie am Schluss die Anzahl der benötigten Fragen ausgeben.

*T4H4) Schreiben Sie eine Funktion mit den beiden Eingabeargumenten Zeilen und Spalten. Diese Funktion soll eine mit zweistelligen Dezimalzahlen gefüllte Matrix zurückgeben, die gerade dem Indexpaar des jeweiligen Eintrags entsprechen, also z.B.

11	12	13	14
21	22	23	24

Erweitern Sie Ihre Funktion so, dass sie meckert, wenn der Benutzer eine Spaltenzahl >9 eingibt (denn dann wird die Sache sehr hässlich.)

*T4H5) Etwas für Grafik-Interessierte: Man kann mit Matlab auch wunderschön farbige Flächen und geometrische Figuren erzeugen.

Machen wir zunächst mal einen Kreis:
```
degrad = pi/180;
w=0:1:360;  si=sin(w*degrad);  co=cos(w*degrad);
plot(co,si)
```

Wenn man das ausprobiert, sieht dieser Kreis ziemlich nach einem Ei aus. `axis equal` macht die Achsen quadratisch, so dass es schöner anzuschauen ist. Einen geschlossenen Linienzug wie diesen Kreis kann man mit dem Befehl `fill(x,y,farbdef)` mit einer Farbe ausfüllen.

- Probieren Sie, den Kreis mit einer hübschen Farbe zu füllen.
- Zeichnen sie mehrere verschieden große Kreise ineinander, die mit verschiedenen Farben gefüllt sind.
- Verschieben sie das Zentrum der Kreise gegeneinander, so dass ein dreidimensionaler Eindruck entsteht. (Wahrscheinlich sieht man das am besten bei sehr vielen nicht gefüllten Kreisen
- Wie zeichnet man ein Quadrat? Probieren Sie das Erzeugen eines dreidimensionalen Eindrucks ebenfalls mit Quadraten verschiedener Größe aus.
-

INTERESSANTERE AUFGABEN:

**T4H6) Um ein N-Eck mit zu zeichnen, kann man sich daran

orientieren, wie in T4H5 ein Kreis gezeichnet wird. Dieser ist nämlich genaugenommen ein 360-Eck.

- Zeichnen Sie in gleicher Weise ein 3-Eck, ein 7-Eck und ein 25-Eck.
- Probieren Sie aus, wie man die Größe und die Richtung dieser Figuren ändert.
- Man muss das Programm nur geringfügig verändern, um einen in einem Zug gezeichneten fünfzackigen Stern zu zeichnen. Wie?
- a) Zeichnen Sie einen hübschen Sternenhimmel mit verschieden großen, bunt gefüllten Sternen mit verschieden vielen Ecken vor dunkelblauem Hintergrund.
- b) Verschönern Sie Ihren Sternenhimmel dadurch, dass er den ganzen Bildschirm füllt. (Nutzen Sie die Hilfe, um rauszubekommen, wie das geht.)
- c) Zeichnen Sie Ihre Sterne an zufällige Orte auf den Himmel (eine normalverteilte Zufallszahl erzeugt man mit `x=randn(1)`).
- ***d) Stellen Sie dabei sicher, dass die verschiedenen Sterne sich nicht überlagern.

** T4H7) (Wird später weiterverwendet) Sie haben die Aufgabe, jeweils 10 männliche und 10 weibliche Kohlmeisen, Amseln und Rotkehlchen zu fangen, zu beringen und ihr Gewicht zu bestimmen.

- Schreiben Sie eine Funktion, das diese Arbeit für Sie erledigt und eine Tabelle zurückgibt, die für die 60 Tiere jeweils die Nummer des Ringes (aufsteigend von 1 bis 60 nach Fangreihenfolge), die Vogelart, das Geschlecht und das Gewicht enthält. Nutzen Sie dabei jeweils einen Zahlencode (z.B. Kohlmeise=1, Amsel=2, Rotkehlchen=3), der im Kommentar des Programms genau dokumentiert ist.
- Die Vögel "fangen" Sie mit folgender Funktion[vogelfang.m], die mit `[Art,Weibchen,Gewicht]=vogelfang` aufgerufen wird und Ihnen zufällige Vögel ins Netz flattern lässt. (Diese Funktion müssen Sie jetzt noch nicht verstehen sondern nur benutzen, wir besprechen sie am nächsten Kurstag).
- Rückgabewerte:
  `Art` ist eine Zeichenkette,
  `Weibchen` ist ein Wahrheitswert zur Angabe des Geschlechts,
  `Gewicht` ist eine Zahl (in Gramm)
- Wenn Sie einen Vogel fangen, den Sie für Ihre Statistik nicht brauchen, lassen Sie ihn frei, ohne seine Daten aufzunehmen.