Einleitung

Bisher wurde gezeigt, wie MATLAB sequentiell (d.h. in unverzweigten Strukturen) Anweisungen abarbeitet. Sollen jedoch komplizierter Sachverhalte programmiert werden, sind verzweigte Strukturen unerlässlich. Beispielsweise muss eine bestimmte Operation mehrfach durchgeführt werden, bis eine bestimmte Abbruchbedingung erfüllt ist (→ Iteration).

Das vorliegende Kapitel zeigt, wie diese sogenannten Kontrollstrukturen in MATLAB implementiert werden können.

3.1 Benutzerdefinierte Funktionen

3.1.1. Einführung

Das MATLAB-System stellt zwar eine Vielzahl vordefinierter Funktionen und Prozeduren zur Verfügung, aber für komplexere Rechenaufgaben genügt das oft nicht, es braucht weitere, auf die besonderen Bedürfnisse zugeschnittene Funktionen.

Es ist möglich, zusätzliche Funktionen und Prozeduren ein Mal zu programmieren, dann als Skript-Datei (m-File) zu speichern und fortan immer wieder zu verwenden.

Wir betrachten ein konkretes Beispiel dazu:

Beispiel 1a: Die folgende benutzerdefinierte Funktion berechnet die Nullstellen eines quadratischen Polynoms gegeben durch $y(x) = ax^2 + bx + c$.

function

```
function NS  = Quad(Koeff)

%Berechnung der Nullstellen eines Quadratischen Polynoms.
%Input: Vektor Koeff=[a b c] der Koeffizienten des Polynoms ax^2+bx+c
%Output: Vektor NS=[x1,x2] mit den beiden Nullstellen
%Beispielaufruf: A=Quad([1 3 5])

D=sqrt(Koeff(2)^2-4*Koeff(1)*Koeff(3)); %Diskriminante
NS(1)=(-Koeff(2)+D)/(2*Koeff(1));       %1. Nullstelle
NS(2)=(-Koeff(2)-D)/(2*Koeff(1));       %2. Nullstelle
```

Speichern Sie diese Funktion als **Quad.m** ab. Diese benutzerdefinierte Funktion kann nun von anderen m-Files oder im Command-Window aufgerufen werden. Zum Beispiel mit:

```
>> A=Quad([2 1 -2])
```

Hinweis: Der Pfad (Current Directory) muss mit dem Speicherort der Funktion übereinstimmen.

Und nun versuchen Sie

```
>> help Quad
```

Was stellen Sie fest?

Die erste Zeile einer benutzerdefinierten Funktion ist immer gleich aufgebaut:

```
function [out1, out2, ...] = FunktionsName(in1, in2, ...)
     %Hilfstext ...

     Anweisungen...
```

`FunktionsName` : Benutzerdefinierter Name
`in1, in2,…` : Eingaben
`out1,out2, …` : Ausgaben

Die Funktion muss als *FunktionsName*.m abgespeichert werden.

Beispiel 1b: Das vorhin aufgeführte Beispiel kann demnach auch wie folgt beschreiben werden:

```
function [NS1, NS2] = Quad2(Koeff)

%Berechnung der Nullstellen eines Quadratischen Polynoms.
%Input: Vektor Koeff=[a b c] der Koeffizienten des Polynoms ax^2+bx+c
%Output: Vektor NS=[x1,x2] mit den beiden Nullstellen
%Beispielaufruf: A=Quad([1 3 5])


D=sqrt(Koeff(2)^2-4*Koeff(1)*Koeff(3)); %Diskriminante
NS1=(-Koeff(2)+D)/(2*Koeff(1));       %1. Nullstelle
NS2=(-Koeff(2)-D)/(2*Koeff(1));       %2. Nullstelle
```

Hinweis: Ruft man nun diese Funktion im Command-Window oder in einem anderen m-File auf, so sind zwei Aufrufe möglich:

1. `a=Quad2(Koeff)` → liefert nur `a=NS1`
2. `[a,b]=Quad2(Koeff)` → liefert beide Nullstellen `a=NS1` und `b=NS2`

3.2 Kontrollstrukturen (Flow Control)

MATLAB hat verschiedene Möglichkeiten sog. Kontrollstrukturen zu definieren. Wir werden nun die wichtigsten kennenlernen.

for –Schleife

3.2.1. for-Schleife

Eine for-Schleife definiert eine Kontrollstruktur, mit der man eine, oder mehrere Anweisungen mit einer bestimmten Anzahl von Wiederholungen ausführen kann.

Genereller Aufbau:

```
for  Zähler = Startwert:Endwert
      Anweisungen;
end
```

Beispiel 2: Quadratzahlen von 1..N als Vektor a definiert

```
for  k=1:20
     a(k)=k^2;
end
```

Das Gleiche Resultat erhält man auch mit `a=(1:20).^2`

Beispiel 3: Eine Schar von Parabeln plotten

```
x=-10:0.1:10;
for  i=0:2:40
     y=x.^2+i;
     plot(x,y);
     hold on;
end
hold off
```

Beispiel 4: Definition einer Hilbertmatrix

```
k = 8
   for m = 1:k
       for n = 1:k
           A(m,n) = 1/(m+n -1);
       end
   end
   A      %Matrix A herausgeben

   %Kontrolle mit dem Befehl hilb(k)
   B = hilb(k)
```

Beispiel 5: Eine einfache Animation (gedämpfter, schwingender Balken)

```
x=0:0.01:2; %x-Vektor
t=0:0.2:30; %Zeit-Vektor
for n=1:length(t)
    y=cos(t(n))*0.1*exp(-0.1*t(n)).*x.^2;
    plot(x,y,'LineWidth',3); axis off
    axis([0 2 -1 1])
    text(0.2,-0.8,['Zeit =' num2str(t(n)) ' s']);
    pause(0.05); %0.05 sec warten
end
```

3.2.2. if … else (Auswahl)

if…else

Die if-Anweisung wird verwendet, falls man überprüfen möchte, ob eine bestimmte Bedingung erfüllt ist oder nicht.

Genereller Aufbau:

```
if    Bedingung        %Falls die Bedingung gilt → Anweisung 1
      Anweisung 1;
else                   %Sonst → Anweisung 2
      Anweisung 2;
end
```

Aufbau, falls mehrere if-Bedingungen gelten sollen:

```
if      Bedingung 1    %Falls die Bedingung 1 gilt → Anweisung 1
        Anweisung 1;
elseif Bedingung 2     %Falls die Bedingung 2 gilt → Anweisung 2
        Anweisung 2;
.
.
.
elseif Bedingung n     %Falls die Bedingung n gilt → Anweisung n
        Anweisung n;

else                   %Sonst → Anweisung 2
        Anweisung 2;
end
```

Beispiel 6: Was ist die Aufgabe der folgenden Funktion?

```
      function z=bsp(a)

      %Input a: Vektor mit zwei Einträgen (Bsp: a=[5,2])

      if    a(1)>a(2)

            z(1)=a(2);
            z(2)=a(1);

      else
            z=a;
      end
```

Antwort: _____

Beispiel 7: Wir definieren und plotten die zusammengesetzte Funktion $f(x)=\begin{cases}1\ , & x\le 0\\ e^x, & x>0\end{cases}$.

```
x=-2:0.01:2;

for k=1:length(x)
     if x(k)<=0
          f(k)=1;
     else
          f(k)=exp(x(k));
     end
end

plot(x,f);
axis([-2 2,0 5])
```

3.2.3. while … end

while…end

Die while-Schleife wird verwendet, falls eine Berechnung solange ausgeführt werden soll, bis eine gewisse Bedingung nicht mehr erfüllt ist. `while...end` wird dann gebraucht, falls man zu Beginn nicht weiss, wie oft die Schleife durchlaufen werden soll (sonst `for`…`end`)

Genereller Aufbau:

```
while Bedingung        %Solange die Bedingung gilt → Anweisung
     Anweisung;
end
```

Beispiel 8: Ganze Zahlen 1,2,3,… sollen addiert werden, solange die Summe kleiner als 1000 bleibt

```
%Zu Beginn Summe=0 und Zahl=1 setzen („initialisieren“)
Summe=0;
zahl=1;

while Summe<1000
    Summe=Summe+zahl;
    zahl=zahl+1;
end
```

3.2.4. switch … case … otherwise (Auswahl) (freiwillig)

switch .. case

Die letzte der häufig verwendeten Kontrollstrukturen ist `switch … case … otherwise`, die in der MATLAB-Hilfe ausführlich beschrieben ist.

Beispiel 10: Zuordnung von Monat zu Tage.

`monat` = "*Zahl zwischen 1 und 12 eingeben*" ;

```
switch     monat      %Eingabe Monat

      case {4, 6, 9, 11}
           tage = 30

      case {1, 3, 5, 7, 8, 10, 12}
           tage = 31

      case {2}
           tage = 28

      otherwise
           tage = 0  %Fehler

end
```

3.3 Rekursionen (freiwillig)

Rekursionen

Allgemein spricht man von ***Rekursion***, wenn ein Problem, eine Funktion oder ein Algorithmus „durch sich selbst" definiert ist. Algorithmen oder Programme bezeichnet man als ***rekursiv***, wenn sie Funktionen oder Prozeduren enthalten, die sich direkt oder indirekt selbst aufrufen.

Beispiel 11: Berechnung von n!

$$n! = \begin{cases} 1 & \text{für } n = 0 \\ \prod_{i=1}^{n} i & \text{für } n \geq 1 \end{cases} \qquad \text{oder rekursiv} \qquad n! = \begin{cases} 1 & \text{für } n = 0 \\ n \cdot (n-1)! & \text{für } n \geq 1 \end{cases}$$

```
function fakt=fakultaet(n)

      %Input eine natürlich Zahl (Bsp: n = 5)

      if n == 0;
         fakt = 1;          rekursiver Aufruf
      else
         fakt = n*fakultaet(n-1);
      end
```

Beispiel 12: Fibonacci-Zahlen rekursiv berechnet

```
function fzahl=fibo_rek(n)

     %Input eine natürlich Zahl (Bsp: n = 5)

      if n <= 2;
         fzahl = 1;
      else
         fzahl = fibo_rek(n-1) + fibo_rek(n-2);
     end
```

3.4 Operatoren in Matlab

Beziehungs-Operatoren

Die Operatoren für Beziehungen in MATLAB sind:

==	gleich (nicht verwechseln mit der Zuweisung =)
~=	nicht gleich
<	kleiner als
>	grösser als
<=	kleiner gleich als
>=	grösser gleich als

Logische Operatoren

Zudem können auch logische Operatoren eingesetzt werden, so zum Beispiel

&&	logisches UND
||	logisches ODER
~	logisches NICHT

Beispiele 13: Logisches UND

```
if a>0 && b>2        %Falls a>0 und b>2 …
     Anweisungen…
end
```

Beispiele 14: Es sollen die Elemente der Matrix

$$A=\begin{pmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\end{pmatrix}$$

gefunden werden, die grösser als 3 sind:

```
>>P=A>3

  P =
       0   0   0   1
       1   1   1   1
```

Nebst den Standartoperatoren gibt es noch weitere nützliche logische Operatoren:

`any`	= 1 wenn mindestens ein Element ungleich Null ist
`all`	= 1 wenn alle Elemente ungleich Null sind
`find`	sucht nach den Indizes der Elemente, die einer bestimmten Bedingung genügen
`exist`	Prüft die Existenz einer Variablen

Beispiel 15: Es sollen die Indizes der Elemente von x ausgegeben werden, die kleiner als 5 sind:

```
>> x = [-5 8 3 6 5 1 2]
>> find(x<5)

ans =

   1   3   6   7
```

Testataufgaben Kapitel 3:

Diese Aufgaben gehören zu den Testatbedingungen und sind bis Mittwoch, 4. Mai 2016, auf Ilias in den Ordner

> Briefkasten > Abgabe Matlab > Serie3

hochzuladen. Packen Sie alle Übungen in ein File (m-File oder pdf). Testate die zu spät hochgeladen werden, werden nicht berücksichtigt.

Übung 3.1: Generieren Sie die Fibonacci-Zahlenfolge a=[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …] mit Hilfe einer `for-end`-Schleife. Die Vorschrift für diese Folge lautet $a_k=a_{k-1}+a_{k-2}$ für k>2 mit Startwerten $a_1=a_2=1$.

Übung 3.2: Es gibt viele Möglichkeiten die Quadratwurzel $x=\sqrt{a}$ einer positiven Zahl a numerisch zu berechnen. Eine elegante Methode, die auf Newton zurückgeht, ist durch die Iteration

$$x_{k+1}=\frac{1}{2}\left(x_k+\frac{a}{x_k}\right) \quad k=1,2,3,...$$

mit Startwert $x_1=a$ gegeben.

a) Schreiben Sie die Iteration als for-Schleife und spielen Sie mit verschiedenen a und N. Vergleichen Sie die Resultate mit den exakten Wurzeln.

b) Natürlich ist es lästig, die Anzahl der Iterationsdurchgänge N von Hand vorzugeben. Wünschenswert wäre, dass die Iteration bei genügend hoher Genauigkeit selbstständig abbricht. Hohe Genauigkeit heisst in diesem Fall, dass die Differenz zweier aufeinander folgender Schritte x_k und x_{k-1} sehr klein ist.

   Die Iteration wird solange wiederholt, bis für die Differenz $|x_k - x_{k-1}| \le e$ gilt, wobei e eine durch den Benutzer definierte Schranke definiert (zBp e=0.0001).

   Hinweis: Arbeiten Sie mit der `while … end` Anweisung.

c) Wie oft muss die Iteration für $x=\sqrt{2}$ durchgeführt werden, damit der Fehler zum exakten Wert weniger als 0.1% beträgt?

Übung 3.3: Ein einfaches Ratespiel

Gegeben ist das folgende m-File

```
clear; clc;

%max. Zahl, die geschätzt werden soll
N=5;

%Erzeuge eine ganze Zufallszahl zwischen 0 und N
nRandom=floor(rand(1)*(N+1));

gameover = 0;

while (gameover == 0)

  % Der User soll die vermutete Zahl im Command-Window eingeben
  guess = input('Welche Zahl wird gesucht? ');

  %Vergleiche die Eingabe mit nRandom
  if guess==nRandom
     fprintf('Jawohl, das stimmt!\n');
     gameover = 1;
  else
     fprintf('Leider nein! Versuche es weiter!\n');
  end

end
```

a) Führen Sie das m-File aus und versuchen Sie die einzelnen Komponenten und Befehle zu verstehen. Schauen Sie die nicht bekannten Befehle in der Hilfe nach.

b) Fügen Sie einen Zähler `nTry` (für die Anzahl Versuche) ein und geben Sie bei game-over `nTry` im Command-Window aus

c) Erweitern Sie das m-File so, dass der User informiert wird, falls die angenommene Zahl zu gross resp. zu klein ist. Zusatzfrage: Mit welcher Strategie werden die Anzahl Versuche minimiert?

d) Fügen Sie Soundeffekte hinzu (siehe Hilfe für den Befehl `sound()`)

e) ….

Übung 3.4: (Zusammengesetzte Funktion) Definieren und plotten Sie die zusammengesetzte Funktion

$$f(x)=\begin{cases}0\ ,\ x\le 0\\ 1\ ,\ 0<x\le 1\\ x,\ x>1\end{cases}$$

im Intervall $-5\le x\le 5$.