ulm university universität ulm

WiMa-Praktikum (Matlab 2/9)

Einführung in LaTeX und Matlab

Prof. Dr. Stefan Funken, Dipl.-Ing. Christoph Erath
15. Mai 2009

m-Files Teil 1: Skripte (Wiederholung)

- Anweisungen können in Textdateien mit Endung `.m` geschrieben werden.
- Die Anweisungen werden im aktuellen Workspace ausgeführt und können auf die im Workspace definierten Variablen zugreifen.
- Skripten können keine Argumente übergeben werden.
- Variablen, die im Skript angelegt oder gelöscht werden, bleiben im Workspace bzw. werden aus dem Workspace gelöscht.

shift.m

```
n=1;
clear y;
y=x+n;
clear x;
```

```
>> x=1:5
x =
     1     2     3     4     5
>> who

Your variables are:

x

>> shift
>> who

Your variables are:

n  y

>> y
y =
     2     3     4     5     6
```

m-Files Teil 2: Funktionen

- Für komplexere Funktionen können Funktions-m-Files verwendet werden. Hier werden, wie in Skripten, Anweisungen in einer Textdatei gespeichert, die der Reihe nach abgearbeitet werden.
- Um die Datei als Funktion zu kennzeichnen, wird die Funktion durch

  `function <Ausgabeargumente> = <Funktionsname>(<Eingabeargumente>)`

  eingeleitet. Hierbei sollte `<Funktionsname>` mit dem Dateinamen übereinstimmen! (Konflikt in zukünftigen versionen!) Falls der Funktionsname und der Dateiname nicht übereinstimmen wird der Dateiname verwendet.
- Einfaches Beispiel: Addieren zweier Zahlen.

addieren.m

```
function c = addieren(x1, x2)
%ADDIEREN Berechnet die Summe zweier Zahlen x1 und x2

c=x1+x2;
```

```
>> c = addieren(57,104)
c =
   161
```

Funktions-Beispiel

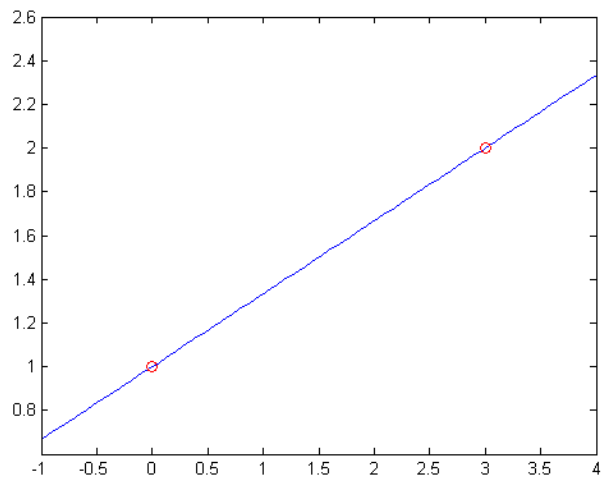
- Beispiel: Aufstellen einer linearen Funktion $ax + b$ durch 2 Punkte $(x1, y1)$ und $(x2, y2)$. Dabei gilt für die Steigungskomponenten in a: $a_i = \frac{y2_i - y1_i}{x2_i - x1_i}$ und den y-Achsenabschnitt b: $b_i = y1_i - a_i x1_i$

gerade.m

```
function [a, b] = gerade(x1, x2 ,y1, y2)
%GERADE Berechnet die Koeffizienten
% der linearen Funktion ax+b durch die
% Punkte (x1, y1) und (x2, y2)

a=(y2-y1)./(x2-x1);
b=y1-a.*x1;
```

```
>> [c1,c2]=gerade(0, 3, 1, 2)
c1 =
    0.3333
c2 =
     1
>> x=linspace(-1,4);
>> y=c1.*x + c2;
>> plot(x,y,[0,3],[1,2],'or')
```



Funktionsaufruf

- `function out1 = gerade(in1)` 1 Eingabe und 1 Rückgabeparameter, geht auch so

  `function [out1] = gerade(in1)`
- `function [out1,out2] = gerade(in1)` 1 Eingabe und 2 Rückgabeparameter
- `function [out1,out2,out3] = gerade(in1,in2,in3,in4)` 4 Eingabe und 3 Rückgabeparameter

- `function gerade(in1)` 1 Eingabe und kein Rückgabeparameter, z.B. bei einem Plot-Befehl

- `function out1 = gerade` kein Eingabe und 1 Rückgabeparameter, z.B. Zeitmessung
- `function [out1,out2] = gerade` kein Eingabe und 2 Rückgabeparameter, z.B. Zeitmessung, Minuten & Stunden

Funktions-Workspace

- Im Gegensatz zu Skripten wird eine Funktion in einem eigenen Workspace ausgeführt.
- In diesem Workspace werden Variablen unabhängig vom Matlab Workspace angelegt und gelöscht. Nach dem Beenden der Funktion werden die Variablen des Funktionsworkspace gelöscht.
- Soll auf eine Variable des Matlab Workspace zugegriffen werden, die nicht als Argument übergeben wird, so muss diese mit `global <Variablenname>` im Matlab-Workspace und im Funktions-Workspace sichtbar gemacht werden.
- Variablen, die nach beenden der Funktion ihren Wert behalten sollen, müssen mit `persistent <Variablenname>` deklariert werden.

modulfunc.m

```
function y = bspfunc(x)

global letzterAufruf
persistent funcnt

if isempty(funcnt)
    funcnt=1;
else
    funcnt=funcnt+1;
end
disp('Funktionsaufrufe:')
disp(funcnt);
letzterAufruf=datestr(now, 13);
y = 2*x;
```

```
>> bspfunc(3)
Funktionsaufrufe:
     1
ans =
     6
>> bspfunc(4)
Funktionsaufrufe:
     2
ans =
     8
>> disp(letzterAufruf)
12:57:09
>> who
Your variables are:
ans             letzterAufruf
```

m-File Funktionen: Kommentare

- Mit % kann ein einzeiliger Kommentar und mit %{ }% ein Kommentarblock definiert werden. Kommentare mit % beginnen beim Kommentarzeichen und gehen bis zum Ende der Zeile. Bei der Definition eines Kommentarblocks müssen die %{ }% Zeichen jeweils in einer eigenen Zeile stehen.
- Die ersten zusammenhängenden Kommentarzeilen vor der ersten Anweisung werden beim Aufruf von `help <Funktionsname>` angezeigt.
- Die erste Kommentarzeile dieses Blocks wird beim Aufruf von `lookfor` durchsucht.

eratosthenes.m

```
function x = eratosthenes(n)

%ERATOSTHENES Berechnet alle Primzahlen bis zu einer oberen Schranke
%
% INPUT : n  Obere Schranke der Primzahlen
% OUTPUT: x  Vektor der Primzahlen kleiner oder gleich n
%
% Die Funktion berechnet Primzahlen mit dem Sieb des Eratosthenes

...
```

```
>> lookfor eratosthenes
ERATOSTHENES Berechnet alle Primzahlen bis zu einer oberen Schranke
```

m-File Funktionen: Ein- und Ausgabeargumente

- Eingabeparameter werden mit call-by-value übergeben, d.h. Änderungen der Eingabevariablen in der Funktion ändert die beim Aufruf benutzte Variable nicht.

increase.m

```
function increase(n,m)
n=n+m
```

```
>> n=2;
>> m=3;
>> increase(n,m)
n =
     5
>> n
n =
     2
```

if Anweisung

- Syntax:

```
if <Bedingung>
    <Anweisung>
elseif <Bedingung>
    <Anweisung>
else
    <Anweisung>
end
```

- Der `else` Block und der `elseif` Block ist optional und kann weggelassen werden;
- Die `if` Anweisung kann beliebig viele `elseif` Blöcke enthalten;

ifbsp.m

```
x=rand(2,1)
abstand=norm(x)
disp('Der Punkt liegt...');

if(abstand>1)
    disp('...ausserhalb...');
elseif(abstand<1)
    disp('...im Innern...');
else
    disp('...auf dem Rand...');
end
disp('des Einheitskreises');
```

```
>> ifbsp
x =
    0.7060
    0.0318
abstand =
    0.7068
Der Punkt liegt
...im Innern...
des Einheitskreises
```

switch Anweisung

- Syntax:

```
switch <Ausdruck>
    case Wert
        <Anweisung>
    case {Wert1, Wert2, ...}
        <Anweisung>
    otherwise
        <Anweisung>
end
```

- Der Ausdruck wird von oben nach unten mit den Werten verglichen und die Anweisungen der ersten Übereinstimmung ausgeführt. Spätere Übereinstimmungen werden ignoriert.
- Falls es keine Übereinstimmung gibt werden die Anweisungen des `otherwise` Blocks ausgeführt.

forbsp.m

```
n=mod(floor(rand(1)*10), 9)+1

switch n
    case {1,4,9}
        disp('ist Quadratzahl');
    case {2,3,5,7}
        disp('ist Primzahl');
    case {6}
        disp('hat 2 Primfaktoren');
    otherwise
        disp('ist Kubikzahl');
end
```

```
>> switchbsp
n =
     2
ist Primzahl
```

for Schleife

- Syntax:

```
for <Variable>=<Matrix>
        <Anweisung>
end
```

- In der for Schleife wird der Variablen nacheinander die Spalten der Matrix zugewiesen und die Anweisungen ausgeführt.
- In einer for Schleife kann mit `continue` zur nächsten Zuweisung gesprungen und mit `break` der Schleifendurchlauf beendet werden.

forbsp.m

```
% Berechet Fibonacci Zahlen

n=6;
f=[0, 1];

for i=2:n
    f=[f, f(i)+f(i-1)];
end

disp(f);
```

```
>> forbsp
     0
     1
     1
     2
     3
     5
     8
```

while Schleife

- Syntax:

```
while <Ausdruck>
        <Anweisung>
end
```

- Durch `break` bzw. `continue` kann wieder die Schleife beendet bzw. zur Überprüfung des Ausdrucks gesprungen werden.

whilebsp.m

```
% Berechnet Naeherung von e

e=1;
n=1;

while abs(e-exp(1))>0.1
    e=e+1/factorial(n)
    n=n+1
end
```

```
>> whilebsp
e =
     2
n =
     2
e =
    2.5000
n =
     3
e =
    2.6667
n =
     4
```

for Schleife

- In Schleifen (`for`, `while`) kann mit `continue` zum Anfang der Schleife gesprungen werden; mit `break` der Schleifendurchlauf beendet werden.

```
% Beispiel für continue
% summiert Kehrwert zufälliger Zahlen
sum = 0;
for k = 1:10
  tmp = rand;
  if tmp == 0
    continue
  end
  sum = sum + 1/tmp;
end
sum
```

```
% Beispiel für break
% beendet wenn Zufallszahl < 1/2 ist
sum = 0;
while 1
  tmp = rand;
  if tmp < 0.5
    break
  end
  sum = sum + tmp;
end
sum
```

for Schleife vs. while Schleife

- Sind beiden Schleifen äquivalent?

```
% Beispiel für for-Schleife
sum = 0;
for k = 1:10
  sum = sum + k;
end
sum
```

```
% Beispiel für while-Schleife
sum = 0;
k = 1;
while k <= 10
  sum = sum + tmp;
  k = k+1;
end
sum
```

for Schleife vs. while Schleife

- Sind beiden Schleifen äquivalent?

```
% Beispiel für for-Schleife
sum = 0;
s = [1 3 5 9];
for k = s
  sum = sum + k;
end
sum
```

```
% Beispiel für while-Schleife
sum = 0;
k = 1;
s = [1 3 5 9];
while k <= length(s)
  sum = sum + s(k);
  k = k+1;
end
sum
```

Weitere Funktionen zur Ablaufsteuerung

Weitere Funktionen zur Kontrolle des Ablaufs eines Skriptes sind:

- `pause`: Wartet eine angegeben Zeitspanne bis zum Ausführen des nächsten Befehls;
- `keyboard`: Wechselt in einen Benutzermodus, in dem zusätzliche Befehle über die Tastatur eingegeben werden können. Der Modus wird durch Eingabe von `RETURN` beendet;
- `input`: Wartet auf ein Tastatureingabe des Benutzers;
- `ginput`: Wartet auf Mauseingaben in einem Graphikfenster;
- `return`: Beenden des Programmablaufs.