

Matlab

Prof. Dr. Klaus Höllig

Institut für Mathematischen Methoden in den Ingenieurwissenschaften,
Numerik und Geometrische Modellierung

Skripten

Speicherung von Matlab-Befehlen in Textdatei `Dateiname.m`

Ausführung:

```
>> Dateiname
```

Ablaufsteuerung:

<code>pause</code>	<i>Unterbrechung der Programmausführung</i>
<code>echo on, echo off</code>	<i>Anzeige der Befehle ein bzw. ausschalten</i>

Kommentare:

```
% Text  
%{ mehrzeiliger Text %}
```

Beispiel

```
% Gauss-Elimination
>> echo on
>> A = [2 6 4; 1 5 9; 3 7 8]
A = 2    6    4
     1    5    9
     3    7    8
>> pause
>> A(2:3,:) = A(2:3,:) - A(2:3)*A(1,)/A(1,1)
A = 2    6    4
     0    2    7
     0   -2    2
>> pause
>> A(3,2:3) = A(3,2:3) + A(2,2:3)
A = 2    6    4
     0    2    7
     0    0    9
>> echo off
```

if-Abfrage

Syntax:

```
if logischer Ausdruck  
    Befehle  
elseif logischer Ausdruck  
    Befehle  
else  
    Befehle  
end
```

Indikatorfunktionen:

```
isempty, isstr, ischar, isinf, isnan, isfinite.
```

Beispiel

Signum s einer Zahl x

$$s(x) = \begin{cases} 1 & \text{für } x > 0, \\ 0 & \text{für } x = 0, \\ -1 & \text{für } x < 0 \end{cases}$$

Berechnung in MATLAB

```
if x>0
    s=1;
elseif x<0
    s=-1;
else
    s=0;
end
```

alternativ

```
s=(x>0)-(x<0);
```

switch-Anweisung

Syntax:

```
switch Ausdruck
  case Wert
    Befehle
  case {Wert1,Wert2,...,Wertn}
    Befehle
  otherwise
    Befehle
end
```

Wert von *Ausdruck*: skalare Größe oder Zeichenkette

switch-Anweisung

Switch-Anweisung zur Ausgabe von Informationen über eine Zahl n :

```
switch n
  case {1,4,9}
    fprintf('%d ist eine Quadratzahl\n',n);
  case {2,3,5,7}
    fprintf('%d ist eine Primzahl\n',n);
  case 6
    fprintf('%d hat zwei Primfaktoren: 2 und 3\n',n);
  case 8
    fprintf('%d ist eine Kubikzahl\n',n);
  case {1,7}
    % Dieser Zweig wird nie erreicht, da die Fälle
    % 1 und 7 bereits zuvor definiert wurden
  otherwise
    disp('n muss natürliche Zahl zwischen 1 und 9 sein.');
```

end

for-Schleife

Syntax:

```
for Variable = Matrix/Cell/Feld  
    Befehle  
end
```

Variable durchläuft erste Spalte von *Matrix/Cell/Feld*

Unterbrechung der Befehlssequenz

```
break      Abbruch der Schleife  
continue  nächste Iteration
```

for-Schleife

Monte-Carlo-Schätzung der Kreiszahl $\pi \approx 3.1416$

- wähle zufällig n Punkte $(p_1, p_2) \in [0, 1)^2$
- bestimme die Anzahl z der Punkte mit $p_1^2 + p_2^2 < 1$
- $\pi/4 \approx z/n$

Vergleich der Laufzeiten unterschiedlicher Implementierungen
`tic` (Start einer Stoppuhr) `toc` (Anhalten der Stoppuhr)

for-Schleife

Implementierung mit einer for-Schleife über die Anzahl der Tests:

```
tic
n=10^6;
z=0;
for k=1:n
    p=rand(2,1);
    z=z+(p(1).^2+p(2).^2<1);
end
pi=4*z/n
toc
```

Ausgabe:

```
pi =
    3.1432
Elapsed time is 18.367863 seconds.
```

for-Schleife

Implementierung mit einer for-Schleife über die Spalten einer Zufallspunktematrix:

```
tic
n=10^6;
z=0;
for p=rand(2,n)
    z=z+(p(1).^2+p(2).^2<1);
end
pi=4*z/n
toc
```

Ausgabe:

```
pi =
    3.1453
Elapsed time is 11.304552 seconds.
```

for-Schleife

Implementierung ohne eine for-Schleife:

```
tic
n=10^6;
P=rand(2,n);
z=sum(P(1,:).^2+P(2,:).^2<1);
pi=4*z/n
toc
```

Ausgabe:

```
pi =
    3.1403
Elapsed time is 0.388576 seconds.
```

while-Schleife

Syntax:

```
while logischer Ausdruck  
    Befehle  
end
```

Unterbrechung der Befehlssequenz:

```
break    Abbruch der Schleife  
continue nächste Iteration
```

while-Schleife

Pfaffs Approximation von π durch $6 * 2^n$ -Ecke

```
% Startwerte:  
% dreifache Kantenlängen des ein- und umbeschriebenen 6-Ecks  
n=0; a=3; b=2*sqrt(3);  
  
% Iteration von Pfaff  
while b-a > 1e-4  
    n=n+1;  
    if n > 100  
        disp('Abbruch nach 100 Iterationen'); break;  
    end  
    b= 2*a*b/(a+b); a=sqrt(a*b);  
end
```

Resultat

$$a = 3.141584 < \pi < 3.141610 = b$$

Funktionen

gespeichert in Datei *Funktionsname.m*

```
function [Rückgabevariable,...]=Funktionsname(Parameter,...)  
    % Kurzbeschreibung für Stichwortsuche  
    % ... Beschreibung ...  
  
    ... Befehle  
end  
lokale Funktionen
```

Verlassen der Funktion durch `return`

Funktionen

Beispiel: Binomialkoeffizient

```
function c = binomial(n,k)
% BINOMIAL binomial coefficient
% c = binomial(n,k)
% n,k: nonnegative integers with k <= n
% c: n choose k

if k < 0 | k > n | n < 0
    disp('invalid input'); return;
end

c = 1;
for m = 1:k
    c = c * (n+1-m) / m;
end
```

Funktionen

Aufruf:

```
>> c = binomial(5,2)
```

```
>> c
```

```
10
```

```
>> binomial(3,4)
```

```
>> invalid input
```

```
>> binomial(4,3)
```

```
>> ans =
```

```
4
```

Funktionen

Darstellung der Kommentare im Funktionskopf mittels `lookfor` und `help`:

```
>> lookfor binomial  
BINOMIAL binomial coefficient
```

```
>> help binomial  
BINOMIAL binomial coefficient  
c = binomial(n,k)  
n,k: nonnegative integers with k <= n  
c: n choose k
```

Funktionen

Unterfunktion:

```
function c = binomial(n,k)
c = product(n)/(product(n-k)*product(k));
end
function p = product(m)
p = 1;
for k = 2:m, p = p*k; end
end
```

Rekursion:

```
function c = binomial(n,k)
if n == 0 | k == 0 | k == n
    c = 1;
else
    c = binomial(n-1,k-1) + binomial(n-1,k);
end
```

Funktionen

Newton-Verfahren:

```
function x = newton(f, df, x)
% Newtonverfahren zur Bestimmung einer Nullstelle
% einer Funktion f mit Ableitung df nahe bei x
max_iter = 100; tol = 1.0e-10;
for k=1:max_iter;
    fx = f(x); dfx = df(x);
    if abs(fx) < tol
        display('Nullstelle bestimmt'); return;
    elseif abs(dfx) < tol
        display('waagrechte Tangente'); return;
    end;
    x = x - fx/dfx;
end;
display('keine Konvergenz');
```

Funktionen

Übergabe der Funktion und ihrer Ableitung als function-handles
Beispiel:

```
>> f = @(x) x^3-x; df = @(x) 3*x^2-1;
>> x = newton(f, df, 4)
x =
    1.0000
```

Funktionen

kurze Programmvariante bei gesicherter Konvergenz

...

```
while abs(f(x)) > eps
    x = x - f(x)/df(x);
end
```

...

Ein- und Ausgabeparameter von Funktionen

Funktion zur Kontrolle der Ein- und Ausgabeparameter:

<code>nargin</code>	<i>Anzahl der Eingabeparameter</i>
<code>nargout</code>	<i>Anzahl der Ausgabeparameter</i>
<code>exist</code>	<i>prüft, ob eine Variable existiert</i>
<code>varargin</code>	<i>Eingabeparameterliste unbestimmter Länge (cell-array)</i>
<code>varargout</code>	<i>Ausgabeparameterliste unbestimmter Länge</i>
<code>nargchk</code>	<i>prüfen der Eingabeparameteranzahl</i>
<code>nargoutchk</code>	<i>prüfen der Ausgabeparameteranzahl</i>

Übergabe von Funktionen als:

<code>string</code>	<i>Name der m-Datei, Aufruf mit feval</i>
<code>functionhandle</code>	

Ein- und Ausgabeparameter von Funktionen

```
function [Vol, Ob] = zylinder(h, R, r)
% Volumen und Oberfläche (optional)
% eines Hohl- oder Vollzylinders
% mit Höhe h, Aussenradius R und Innenradius r (optional)

if nargin < 2
    error('zu wenige Parameter')
end
if nargin == 2
    r = 0
elseif r >= R
    error('Aussenradius nicht größer als Innenradius')
end
Vol = pi*h*(R^2 - r^2);
if nargin == 2
    Ob = 2*pi*(h*(r+R) + R^2-r^2);
end
```

Ein- und Ausgabeparameter von Funktionen

```
function [x,fx] = steffensen(f,x,tol,max_it)
% STEFFENSEN zero of f(x) by Steffensen's method
% function [x,fx] = steffensen(f,x,tol,max_it)
% f: function handle
% x: approximation of a zero, updated
% tol: bound for |fx|, fx = f(x), for termination
% max_it: maximal number of iterations

% set defaults
if nargin < 4
    max_it = 100;
    if nargin < 3
        tol = 1e-10;
        if nargin < 2 disp('too few arguments'), return end
    end
end
end
```

Ein- und Ausgabeparameter von Funktionen

```
for n = 1:max_it
    fx = f(x);
    % Abbruch bei erreichter Genauigkeit
    if abs(fx) < tol
        return
    end
    d = f(x+fx)-fx;
    if abs(d) < eps*fx
        disp('nearly zero denominator');
        return
    end
    % Iterationsschritt
    x = x - fx*fx/d;
end
disp('no convergence');

end
```

Ein- und Ausgabeparameter von Funktionen

```
>> x = steffensen(@cos,0)
x = 1.5708
```

```
>> fct = @(x) x^2-2
```

```
>> [x,fx] = steffensen(fct,1,1.0e-20,100)
no convergence
x = 1.4142
fx = 4.4405 e-016
```

Datenpfad

Verzeichnisbefehle:

<code>pwd</code>	<i>Ausgabe des aktuellen Arbeitsverzeichnisses</i>
<code>cd</code>	<i>Wechseln des Arbeitsverzeichnisses</i>
<code>dir</code>	<i>Ausgabe des Verzeichnisinhalts</i>
<code>rmdir</code>	<i>Verzeichnis löschen</i>
<code>mkdir</code>	<i>Verzeichnis erstellen</i>

Pfadbefehle:

<code>path</code>	<i>Ausgabe bzw. Durchsuchen des MATLAB-Pfads</i>
<code>addpath</code>	<i>Verzeichnis in den Pfad aufnehmen</i>
<code>rmpath</code>	<i>Verzeichnis aus dem Pfad löschen</i>
<code>savepath</code>	<i>aktuellen Pfad speichern</i>
<code>pathtool</code>	<i>Interaktive Bearbeitung des Pfads</i>
<code>which</code>	<i>Angabe des Pfads zu einer Funktion</i>

Befehle zur Benutzerinteraktion

<code>input</code>	<i>Eingabeaufforderung an den Benutzer</i>
<code>inputdlg</code>	<i>Eingabefeld in einem Dialogfenster</i>
<code>keyboard</code>	<i>zweitweilig Übergabe der Kontrolle an den Benutzer</i>
<code>uigetfile</code>	<i>Standarddialog zur Auswahl einer Eingabedatei</i>
<code>uigetdir</code>	<i>Standarddialog zur Auswahl eines Verzeichnisses</i>
<code>uiputfile</code>	<i>Standarddialog zur Auswahl einer Ausgabedatei</i>
<code>msgbox</code>	<i>Meldungsfenster</i>
<code>errordlg</code>	<i>Dialogfenster für Fehlermeldungen</i>
<code>helpdlg</code>	<i>Dialogfenster für Hilfestellungen</i>
<code>questdlg</code>	<i>Dialogfenster für Abfragen</i>
<code>warndlg</code>	<i>Dialogfenster für Warnmeldungen</i>
<code>ginput</code>	<i>graphische Eingaben mit der Maus</i>

Befehle zur Benutzerinteraktion

Beispiel: input-Werte

```
>> A = input('2x2-Matrix eingeben: ')
```

```
2x2-Matrix eingeben: [1 2; 3 4]
```

```
A =
```

```
1 2
```

```
3 4
```

Beispiel: input-Zeichenkette

```
>> fct = input('Funktionsnamen eingeben: ');
```

```
Funktionsnamen eingeben: 'sin'
```

```
fct =
```

```
sin
```

Befehle zur Benutzerinteraktion

```
function ellipse
% zeichnet eine Ellipse

% Grafikfenster öffnen
clf; hold on;
axis([-10 10 -10 10]);

% Eingabe von Mittelpunkt und Halbachsenlängen
[x,y] = ginput(1);
plot(x, y, 'ro');
H = inputdlg({'a:', 'b:'});
a = str2num(H{1}); b = str2num(H{2});

% Zeichnen der Ellipse
t = linspace(0,2*pi);
plot(x+a*cos(t), y+b*sin(t));
```