Programmierung in MATLAB

Interaktiver Modus: Eingabe der Anweisungen zeilenweise
$\rightarrow$ Unzweckmäßig für die Erstellung von Algorithmen

Lösung: m–files (MATLAB–Programme)

Man unterscheidet zwischen:

- **script–files**

- **function–files**

script–files

Ein `script-file` ist eine Folge von MATLAB–Anweisungen die sukzessiv abgearbeitet wird.

Beispiel:

```
s1=sin(linspace(0,4*pi,10));
s2=sin(linspace(0,4*pi,20));
s3=sin(linspace(0,4*pi,100));
subplot(311);plot(s1)
subplot(312);plot(s2)
subplot(313);plot(s3)
```

Ausgabe:

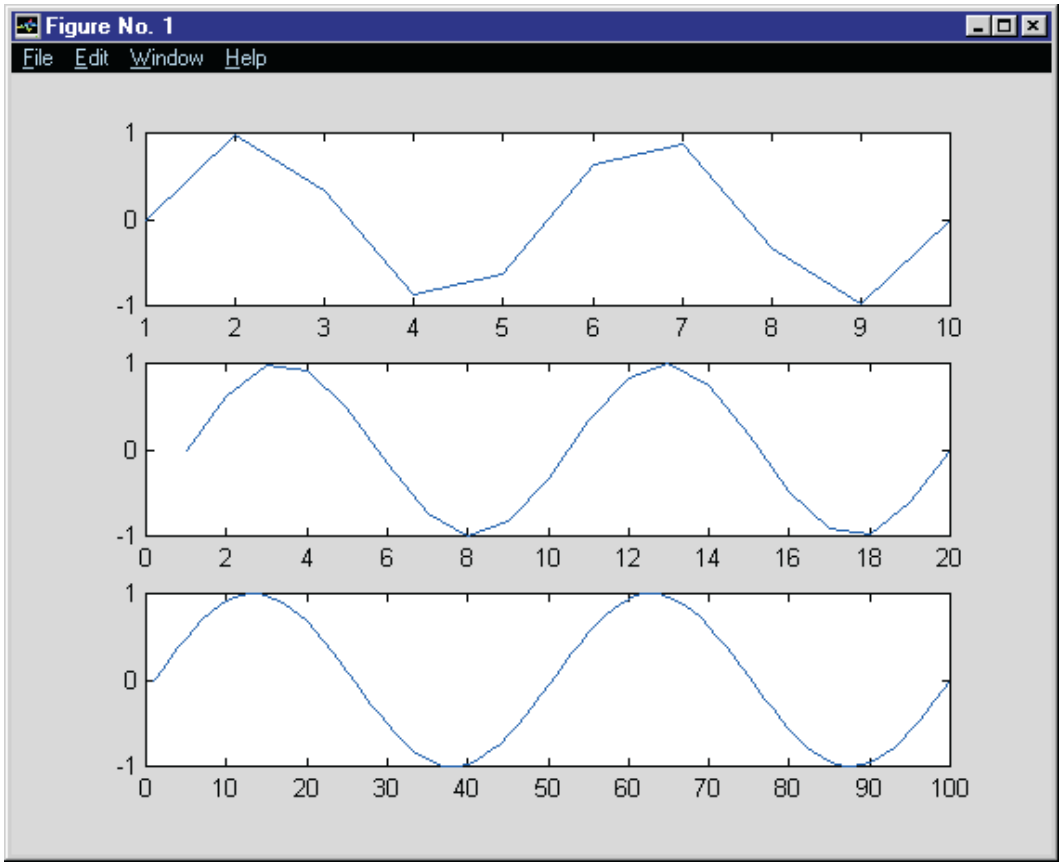
Figure No. 1
File Edit Window Help
1
0
-1
1 2 3 4 5 6 7 8 9 10
1
0
-1
0 2 4 6 8 10 12 14 16 18 20
1
0
-1
0 10 20 30 40 50 60 70 80 90 100

function–files

Die Funktionalität von MATLAB kann durch `function-files` erweitert werden.

Rumpf eines `function-files`:

```
function [out_1,...,out_n]=Funktionsname(in_1,...,in_m)
        Folge von Anweisungen
```

Beispiel:

```
function [c,nc] = test(a,b)
c=a+b;
nc=norm(c);
```

Elegantere Version:

```
function [c,nc] = test(a,b)
%Diese Funktion berechnet c=a+b und die euklidsche
%Norm von c
%Syntax: [c,norm_von_c] = test(a,b)
%Es werden jeweils 2 Eingabe- und Ausgabeargumente verlangt
if (nargin~=2 | nargout~=2)
    help test
else
c=a+b;
nc=norm(c);
end
```

Zu beachten ist:

- Name des `m-files` und Funktionsname müssen übereinstimmen.
- Das Wort `function` muß in der ersten Zeile stehen.
- Die Kommentarzeilen nach der ersten Zeile dienen als Hilfstext. Weiterhin wird die erste Kommentarzeile mit dem `lookfor`–Befehl analysiert.
- In einem `function-file` können `script-files` aufgerufen werden. Dieses `script-file` wird dann nur im *Function-Workspace* ausgewertet.
- Mit `nargin` und `nargout` kann die Anzahl der Eingabe– und Ausgabeargumente festgestellt werden.
- Eine Inter-Funktionskommunikation ist über `globale` Variablen möglich.

Steuerstrukturen

Steuerstrukturen dienen zur Kontrolle des Programmflusses.

Möglichkeiten den sequentiellen Ablauf eines Programms zu beeinflussen:

- **Schleifen (iterative Anweisungen)**
  `for`, `while`

- **Verzweigungen (bedingte Anweisungen)**
  `if`, `case`

for–Schleifen

Bei for–Schleifen kann festgelegt werden, wie oft eine Folge von Anweisungen ausgeführt werden soll.

Allgemeine Syntax:

```
for Variable = Matrix
       Folge von Anweisungen
end
```

Beispiel:

```
M=[1 2; 3 4];
for k=M
   e=k
end
```

Allgemeine Anwendung:

```
for Variable = von : Schrittweite : bis
       Folge von Anweisungen
end
```

Beispiel:

```
for k=1:1:5
   e(k)=k/10;
end
e
```

Wird dieses file gestartet, so erhält man die folgende Ausgabe:

```
e =
    0.1000    0.2000    0.3000    0.4000    0.5000
```

Es ist zu beachten ist, daß die Variable nach jedem Durchlauf neu gesetzt wird (keine <= Abbruchbedingung durch Skalarvergleich)

```
M=[1 2; 3 4];
for k=1:1:2
   k
   k=100
end
```

Ausgabe:

```
k =   1
k = 100
k =   2
k = 100
```

while–Schleifen

Bei while–Schleifen wird die Anzahl der Wiederholungen einer Folge von Anweisungen durch Vergleichsoperatoren, Vergleichsfunktionen und logische Bedingungen kontrolliert.

Vergleichsoperatoren:

<	kleiner
<=	kleiner gleich
>	größer
>=	größer gleich
==	gleich
~=	ungleich

Vergleichsfunktionen:

`any`	WAHR, wenn irgend ein Element ungleich 0 ist
`all`	WAHR, wenn alle Elemente ungleich 0 sind
`find`	Findet Indizes von Elementen ungleich 0

Logische Bedingungen sind:

`&`	UND
`|`	ODER
`~`	NICHT

Eine while–Schleife besitzt grundsätzlich die folgende Struktur:

```
while Ausdruck
   Folge von Anweisungen
end
```

Abweisende Schleife: Der Ausdruck wird ausgewertet bevor die Schleife ausgeführt wird.

Beispiel:

```
k=0;
while k<=5
   k=k+1;
end
k
```

Als Ausgabe erhält man `k=6`. Erst wenn `k` den Wert 6 annimmt, ist der Ausdruck `k<=5` FALSCH und die Schleife wird beendet, d.h. die Anweisungen nicht mehr ausgeführt.

Abbruch von Schleifen:

- **break**
- **return**

Allgemeines zu Schleifen in MATLAB

MATLAB ist in der Verarbeitung von Schleifen nicht sehr effizient. Daher sollte folgendes beachtet werden:

- **Vektorisierung:** Wenn es möglich ist: Schleifen vermeiden und durch Matrix– bzw. Vektoroperationen ersetzen.

- **Vorinitialisierung:** Wenn in Schleifen Vektoren bzw. Matrizen erzeugt werden, so sollten diese zuvor initialisiert werden. (Geeignet: `zeros`)

Beispiel:

```
a=[1,2,3,4,5];
b=[6,7,8,9,10];

c=zeros(1,length(a));

for k=1:length(a)
   c(k)=a(k)+b(k);
end
c
```

Obige Operation kann einfach durch `c=a+b` ersetzt werden

if–Anweisung

Die if–Anweisung wertet einen logischen Ausdruck aus und läßt eine Gruppe von Anweisungen ausführen, wenn er WAHR ist. Die allgemeine (mehrseitige) Syntax ist:

```
if Ausdruck1
   Folge von Anweisungen
elseif Ausdruck2
   Folge von Anweisungen
elseif Ausdruck3
   Folge von Anweisungen
   .
else
   Folge von Anweisungen
end
```

```
for k=1:5
   if k==1
      sprintf('k= %i, %s',k,'Text1') %Anweisung 1
   elseif k==4
      sprintf('k= %i, %s',k,'Text2') %Anweisung 2
   else
      sprintf('k= %i, %s',k,'Text3') %Anweisung 3
   end
end
```

Ausgabe:

```
ans = k= 1, Text1
ans = k= 2, Text3
ans = k= 3, Text3
ans = k= 4, Text2
ans = k= 5, Text3
```

switch–Anweisung

Folge von Anweisungen wird entsprechend dem Wert einer Variablen ausgeführt. Allgemeine Syntax:

```
switch Variable
   Folge von Anweisungen
case Konstante1
   Folge von Anweisungen
case Konstante2
   Folge von Anweisungen
   .
   .
   .
otherwise
    Folge von Anweisungen
end
```

Beispiel:

```
switch var
  case 2
     disp('Fall 1: var ist gleich 2')
  case {4,5}
     disp('Fall 2: var ist gleich 4 oder var ist gleich 5')
  otherwise
     disp('Weder Fall 1 noch Fall 2')
end
```

Zu beachten ist, daß die switch–Anweisung mehrere Bedingungen in einem case–Fall behandeln kann, wenn die Bedingungen in einer sog. Zelle (durch die geschweiften Klammern erkennbar) zusammengefaßt wird.

Einlesen und Speichern von Daten

In MATLAB können Daten in binärem MATLAB– und lesbaren ASCII-Format eingelesen und gespeichert werden.

Man unterscheidet zwischen:

- **High–Level Befehlen:** `load`, `save`
- **Low–Level Befehlen:** `fprintf`, `fscanf`, `fwrite`, `fread`

High–Level Befehle: Beispiel:

```
>> a=[1,2,3;4,5,6;7,8,9];
>> save a
>> clear
>> load a
>> a
a =
     1     2     3
     4     5     6
     7     8     9
```

Ausführlich hieße es `save a.mat -mat` bzw. `load a.dat-mat`. MATLAB verwendet das Binärformat (mit dem Suffix mat) jedoch standardmäßig. Für ASCII wird das Suffix ascii verwendet.

Low–Level Befehle: Beispiel:

```
a=[1 2 3;4 5 6;7 8 9];
file=fopen('matrix.dat','w');
[z,s]=size(a);
for m=1:z
   for n=1:s
      fprintf(file,'%g ',a(m,n));
   end
   fprintf(file,'\n');
end
fclose(file);
clear a;
file=fopen('matrix.dat','r');
a=fscanf(file,'%g',[z,s]);
a
```