Kapitel 7

Steuerkonstrukte

7.1 Sequenz

Die einfachste Steuerstruktur ist die Aneinanderreihung von Programmteilen. Diese Programmteile sind Teile des gesamten Algorithmus und werden Teilalgorithmen oder auch Strukturblöcke genannt. Durch die Aneinanderreihung wird die zeitliche Abarbeitung von Strukturblöcken $S_1, S_2, \ldots, S_n$ in der Reihenfolge der Niederschrift festgelegt.

MATLAB Beispiel

Die zeitliche Abfolge wird durch Aneinanderreihung von Befehlen erreicht. Bei allen Zuweisungen (=) muss sichergestellt sein, dass alle Variablen auf der rechten Seite bereits bekannt sind.

```
a = 10; b = 3;
r = mod(a,b)

     1

a = 12;
r = mod(a,b)

     0
```

Ändert man den Wert einer Variablen, ändern sich nicht automatisch damit vorher berechnete Größen. Man muss, z.B. die Modulo-Division `mod` nach der Änderung von `a` wieder ausführen, damit sich auch `r` ändert.

7.2 Auswahl

In Programmen ist es häufig notwendig, Entscheidungen zu treffen, welche Strukturblöcke abgearbeitet werden sollen. Man trifft dabei mit Hilfe von logischen Ausdrücken, sogenannten Bedingungen, Entscheidungen, die den Ablauf eines Programms direkt beeinflussen.

Dafür gibt es in jeder Programmiersprache sogenannte Steueranweisungen, die die einzelnen Anweisungsblöcke einrahmen. Diese definieren den Beginn und das Ende der Steueranweisung und regeln den Ablauf innerhalb des Steuerkonstrukts.

In MATLAB gibt es für Entscheidungen zwei Strukturen, den sogenannten `IF-Block` oder die `Auswahlanweisung`, die in der Folge an konkreten Beispielen besprochen werden.

7.2.1 IF-Block

Die einfachste Form des `IF-Blocks` ist die einseitig bedingte Anweisung, die die bedingte Ausführung eines Anweisungsblocks erlaubt.

```
if Bedingung
   Anweisungsblock
end
```

Diese Form wird häufig auch in einer einzeiligen Version geschrieben:

```
if Bedingung, Anweisung; end
```

Diese einfachste Form kann durch beliebig viele `elseif Anweisungen` und maximal eine `else Anweisungen` erweitert werden. In ihrer vollständigen Form hat der `IF-Block` daher die folgende Form:

```
if Bedingung 1
   Anweisungsblock 1
elseif Bedingung 2
   Anweisungsblock 2
...
else
   Anweisungsblock n
end
```

MATLAB Beispiel

Das Programmfragment erkennt, ob eine Zahl x durch 2 und 3, bzw. nur durch 2 oder nur durch 3 oder gar nicht durch 2 und 3 teilbar ist.

Mit dem Befehl `mod(a,b)` wird die Modulodivision a/b durchgeführt. Wenn diese Null ergibt ist die Zahl a durch b teilbar.

Es wird immer nur ein Anweisungsblock ausgeführt, obwohl die Bedingungen bei mehreren erfüllt sein können.

```
if mod(x,2)==0 & mod(x,3)==0
    disp('Durch 2 und 3 teilbar!')
elseif mod(x,2)==0
    disp('Nur durch 2 teilbar!')
elseif mod(x,3)==0
    disp('Nur durch 3 teilbar!')
else
    disp('Nicht teilbar!')
end
```

Folgende wichtige Regeln gelten für `IF`-Blöcke:

- Die Bedingungen müssen logische Ausdrücke sein, mit deren Hilfe die Entscheidung getroffen wird. Es können keine logischen Felder, wie sie bei der logischen Indizierung verwendet werden, direkt die Steuerung übernehmen, da diese mehrdeutig sein können.
- Muss man logische Felder verwenden, kann man die Befehle `all(L(:))` oder `any(L(:))` anwenden, die `TRUE` ergeben, wenn alle Elemente bzw. zumindest ein Element des Feldes `TRUE` sind.
- Die direkte Anwendung der logischen Indizierung kann sehr häufig `IF`-Blöcke bei der Manipulation von Feldern ersetzen.
- Die Bedingungen werden nacheinander überprüft und es wird der erste Anweisungsblock ausgeführt, bei dem die Bedingung erfüllt ist. Danach wird das Programm am Ende des `IF`-Blocks fortgesetzt.
- Es ist erlaubt, dass sich Bedingungen überlappen. Es wird jedoch nur ein Anweisungsblock ausgeführt.
- Falls keine Bedingung erfüllt ist, wird bei Vorhandensein einer `else`-Anweisung der dort spezifizierte Block ausgeführt. Falls auch keine `else`-Anweisung vorhanden ist, wird kein Befehl ausgeführt.
- Will man für den weiteren Programmablauf sicherstellen, dass eine Variable innerhalb eines `IF`-Blocks zugewiesen wird, muss man entweder alle möglichen Fälle mit den Bedingungen abdecken, oder unbedingt die `else`-Anweisung verwenden.

- Mehrere `IF`-Blöcke können ineinander geschachtelt werden, wobei jeder mit einem `end` abgeschlossen werden muss.
- Zur besseren Lesbarkeit von Programmen sollte man die Anweisungsblöcke je nach Zugehörigkeit zu Steuerkonstrukten einrücken. Damit wird die Struktur von Programmen viel leichter ersichtlich.

7.2.2 Auswahlanweisung

Die `Auswahlanweisung` ist dem `IF`-Block sehr ähnlich und ermöglicht die Ausführung maximal eines Blocks von mehreren möglichen Anweisungsblöcken. Eine `Auswahlanweisung` hat folgende Form:

```
switch Schalter
case Selektor 1
   Anweisungsblock 1
case Selektor 2
   Anweisungsblock 2
...
otherwise
   Anweisungsblock 2
end
```

Bei der Verwendung ist Folgendes zu beachten:

- Während beim `IF-Block` mehrere Bedingungen ausgewertet werden können, richtet sich die Abarbeitung einer `Auswahlanweisung` nach dem Wert eines einzigen Ausdrucks, dem sogenannten Schalter. Dieser Schalter kann ein Skalar oder eine Zeichenkette sein. Er muss **keine logische Variable** sein.
- Die Abarbeitung erfolgt mit Hilfe der `case`-Anweisung. Die Ausführung wird durch einen Vergleich zwischen Schalter und Selektor geregelt. Stimmen die beiden überein, wird der zugehörige Auswahlblock ausgeführt und danach an das Ende der `Auswahlanweisung` gesprungen. Es wird also maximal ein Auswahlblock ausgeführt.
- Die Auswahl erfolgt naturgemäß wieder mit Skalaren oder Zeichenketten. Sollen für einen `case` mehrere Möglichkeiten erlaubt sein, kann man für den Selektor eine durch Beistriche getrennte Liste angeben. Solche Listen werden mit Hilfe geschwungener Klammern geschrieben, z.B. `{1,2,3}` oder `{'a','b','c'}`.
- Während sich die Bedingungen eines `IF-Blocks` überschneiden können, müssen die Alternativen einer `Auswahlanweisung` eindeutig sein.

- Wenn keine der von den Selektoren abgedeckten Bedingungen zutrifft, wird der Anweisungsblock einer eventuell vorhandenen `otherwise`-Anweisung ausgeführt.

MATLAB Beispiel

Dieser Programmteil zeigt an, ob die Zahl x kleiner, gleich oder größer Null ist.

Als Schalter dient dafür die Signum-Funktion `sign`.

Im zweiten Teil wird wird an Stelle des dritten Falles einfach `otherwise` verwendet, da es sonst keine Möglichkeiten mehr gibt.

```
switch sign(x)
case -1
    disp('x<0')
case 1
    disp('x>0')
case 0
    disp('x==0')
end

switch sign(x)
case -1
    disp('x<0')
case 1
    disp('x>0')
otherwise
    disp('x==0')
end
```

MATLAB Beispiel

Dieser Programmteil zeigt an, ob ein String `str` gleich einem bestimmten Buchstaben ist.

Als Schalter dient dafür einfach der String `str`.

Im dritten Fall wird eine Liste zum Vergleich herangezogen. Listen werden in MATLAB mit dem Klammerpaar `{}` umschlossen.

```
switch str
case 'a'
    disp('Fall a')
case 'b'
    disp('Fall b')
case {'c','d','e'}
    disp('Fall c, d, oder e')
otherwise
    disp('Nicht bekannt!')
end
```

MATLAB Beispiel

Dieses kleine Unterprogramm berechnet die Tage pro Monat in einem beliebigen Jahr unter Berücksichtigung der Schaltjahre ab der Kalenderreform im Jahr 1582.

```
function [tage] = tagepromonat(monat,jahr)
sj = 0;
switch jahr > 1582
case 1
    if mod(jahr,  4)==0, sj=1; end
    if mod(jahr,100)==0, sj=0; end
    if mod(jahr,400)==0, sj=1; end
end
```

An diesem Beispiel sieht man die Verschachtelung von `switch-case`- und `if`-Strukturen.

Wenn in der Zeile Platz ist, können die Keywörter und die Befehle in einer Zeile stehen. Als Trennzeichen wird dann der Beistrich verwendet.

```
switch monat
case {4,6,9,11},        tage = 30;
case {1,3,5,7,8,10,12}, tage = 31;
case 2
    switch sj
    case 0, tage = 28;
    case 1, tage = 29;
    end
otherwise
    tage = 0; error('Falscher Monat');
end
```

7.3 Wiederholung

Ein wichtiges Konstruktionsmittel in Programmiersprachen ist die Wiederholung. Sie erlaubt die wiederholte Ausführung einer Anweisungsfolge, ohne dass man gezwungen ist, die entsprechenden Anweisungen mehrmals zu schreiben. Die Anzahl der Wiederholungen wird dabei durch einen Schleifenkopf bestimmt.

In MATLAB gibt es zwei Schleifentypen, die Zählschleife `for` und die bedingte Schleife `while`. Beide Schleifentypen können darüberhinaus mit dem Befehl `break` abgebrochen werden.

Schleifenkonstrukte haben eine große Bedeutung bei allen Iterationen, wo aus bekannten Werten neue erzeugt werden. Als Beispiel soll folgende Rekursionsformel dienen:

$$a_1 = 0\ ,\ a_2 = 1\ ,\ a_k = \frac{a_{k-2} + a_{k-1}}{2} \qquad \forall k \geq 3\ . \tag{7.1}$$

In vielen anderen Fällen, hat sich durch Matrix- und Arrayoperatoren, durch die Doppelpunktnotation und durch eine Fülle von MATLAB-Befehlen die Notwendigkeit für Schleifenkonstrukte verringert. Als wichtige Regel gilt, dass allen Befehlen, die direkt auf Felder angewandt werden, gegenüber einer expliziten Programmierung mit Schleifen Vorrang zu geben ist. Bei allen internen Befehlen kann die zeitliche Optimierung durch MATLAB viel besser durchgeführt werden. Außerdem werden bei Vermeidung von Schleifen die Programme weit einfacher, kürzer und übersichtlicher. Daher sollte man sich immer die Frage stellen, ob man eine Schleife wirklich braucht oder ob man die Aufgabe nicht besser anders erledigen kann.

7.3.1 Zählschleife

In der Zählschleife wird explizit angegeben, wie oft ein Anweisungsblock ausgeführt werden soll.

```
for Schleifenindex = Feld
   Anweisungsblock
end
```

Der Schleifenindex nimmt dabei nacheinander alle Werte der Elemente eines beliebigen Feldes "Feld" an. Der Ablauf erfolgt dabei entsprechend den Regeln der linearen Feldindizierung, zuerst entlang der ersten Dimension, dann der zweiten, usw.. Für jeden Wert des Schleifenindex wird der Anweisungsblock einmal ausgeführt und danach die Schleife beendet.

Durch eine Kombination mit einer `IF-Entscheidung` kann unter Verwendung des Befehls `break` die Schleife jederzeit beendet werden.

```
for Schleifenindex = Feld
   Anweisungsblock 1
   if Bedingung, break; end
   Anweisungsblock 2
end
```

Natürlich können auch Schleifen ineinander geschachtelt werden, wobei zu jedem `for` ein `end` gehören muss. Bei geschachtelten Schleifen beendet der Befehl `break` die jeweils innere Schleife.

Der Schleifenindex hat am Ende der Abarbeitung den jeweils letzten Wert. Man kann daher überprüfen, ob es zur Ausführung des `break-Befehls` gekommen ist.

7.3.2 Die bedingte Schleife

Bei der bedingten Schleife (`while`) hängt die Anzahl der Durchläufe von einer logischen Bedingung im Schleifenkopf ab. Die Bedingung wird bei jedem Schleifendurchlauf am Beginn ausgewertet. Der Anweisungsblock wird nur dann ausgeführt, wenn die Bedingung erfüllt ist. Ist die Bedingung nicht erfüllt, wird die Schleife beendet.

```
while Bedingung
   Anweisungsblock
end
```

Ist die Bedingung schon am Anfang nicht erfüllt, wird der Anweisungsblock nie ausgeführt. Natürlich kann auch eine solche Schleife an jeder beliebigen Stelle durch eine `break`-Anweisung unterbrochen werden.

```
while Bedingung
   Anweisungsblock 1
   if Abbruchbedingung, break; end
   Anweisungsblock 2
end
```

In manchen Programmiersprachen gibt es eine sogenannte nichtabweisende Schleife (UNTIL-Schleife), die zumindest einmal durchlaufen wird. Eine solche gibt es in MATLAB nicht, man kann sie jedoch mit Hilfe einer "Endlosschleife" und eine `break`-Anweisung realisieren.

```
while 1                % immer wahr
   Anweisungsblock
   if Bedingung, break; end
end
```

Am Beispiel der "Endlosschleife" sollte auch klar werden, welche Gefahr in Schleifen steckt, wenn die Abbruchbedingungen nicht gut durchdacht sind. In solchen Fällen ist es möglich, dass Schleifen von selbst nicht mehr verlassen werden. Dies kann dann nur durch einen externen Abbruch des Programms erfolgen. Solche Fehler sollten daher vermieden werden.

MATLAB Beispiel

Hier wird die Rekursionsformel

$$a_1 = 0\ ,\ a_2 = 1\ ,\ a_k = \frac{a_{k-2} + a_{k-1}}{2} \qquad \forall k \geq 3$$

für $1 \leq k \leq n$ ausgewertet und gezeigt, wie man bei einem Limit $|a_k - a_{k-1}| < \epsilon$ die Berechnung abbricht.

Die Realisierung erfolgt einmal mit einer `for`-Schleife ohne weitere Einschränkung, einmal mit einer `for`-Schleife mit zusätzlicher Verwendung des `break`-Befehls und einmal mit einer Kombination aus `while`-Schleife und `break`-Befehl.

Wann immer es möglich ist, empfiehlt es sich, Felder vor dem Gebrauch in ihrer maximalen Größe anzulegen, damit sie nicht innerhalb der Schleife immer dynamisch vergrößert werden müssen.

Mit dem Befehl `c(k:end)=[]` wird der nicht benötigte Rest des Feldes gelöscht.

```
n = 100; limit = 1.e-6;

a = zeros(1,n); a(2) = 1;
for k = 3:n
    a(k) = (a(k-1) + a(k-2)) / 2;
end

b = zeros(1,n); b(2) = 1;
for k = 3:n
    b(k) = (b(k-1) + b(k-2)) / 2;
    if abs(b(k)-b(k-1)) < limit
        break;
    end
end
b(k+1:end) = [];

c = zeros(1,n); c(2) = 1;
k = 2;
while abs(c(k)-c(k-1)) >= limit
    k = k + 1;
    if k > n, break; end
    c(k) = (c(k-1) + c(k-2)) / 2;
end
c(k:end) = [];
```