

# Mathematik am Computer

## 6. Übung: Matlab, Teil III

Marcus Grote und Helmut Harbrecht

Universität Basel

30. Oktober – 2. November 2023

- 1 Matlab als Programmiersprache
  - Steuerung
  - Logische Ausdrücke
  - Speicherverwaltung und Vektorisierung

# Verzweigung: if-elseif-else-end

```
if n == 1
    Befehle1
elseif n == 2
    Befehle2
elseif n < 1
    Befehle3
    ⋮
elseif n == 'A'
    BefehleN
else
    BefehleAlt
end
```

**elseif** und **else** mit den darauf folgenden Befehlen sind optional.

# Verzweigung: if-elseif-else-end

```
if n == 1  
    Befehle  
end
```

Falls  $n = 1$  gilt, so werden *Befehle* ausgeführt, sonst keine.

```
if n == 1  
    Befehle  
else  
    BefehleAlt  
end
```

Falls  $n = 1$  gilt, so werden *Befehle* ausgeführt, sonst *BefehleAlt*.

# Verzweigung: if-elseif-else-end

```
if n == 1
    Befehle1
elseif n == 2
    Befehle2
elseif n == 15
    Befehle3
else
    BefehleAlt
end
```

- Falls  $n = 1$  ist, so werden *Befehle1* ausgeführt.
- Falls  $n = 1$  falsch ist und  $n = 2$  gilt, so werden *Befehle2* ausgeführt.
- Falls sowohl  $n = 1$  und  $n = 2$  falsch sind und  $n = 15$  gilt, werden *Befehle3* ausgeführt.
- In allen anderen Fällen werden *BefehleAlt* ausgeführt.

# Logische Ausdrücke

- Aussagen wie “ $A$  ist grösser als 0” ( $A > 0$ ) oder “ $B$  ist nicht gleich 5” ( $B \neq 5$ ).
- Werden in Matlab mit einem Vektor oder Matrix dargestellt vom Typ `logical`, deren Einträge 1 oder 0 bzw. `true` oder `false` sind.
- Zahlen und Vektoren oder Matrizen werden untereinander verglichen mit `==` (gleich), `~=` (nicht gleich), `<`, `>`, `<=`, `>=`
- Logische Ausdrücke lassen sich mit sog. Junktoren zu komplexen Aussagen verbinden, diese sind: `&` bzw. `&&` für *und*, `|` bzw. `||` für *oder* und `xor` für *entweder-oder*.  
Beispiel: `(x >= 10) & (x <= 20)` und  
`(x >= 10) && (x <= 20)`
- Eine logische Matrix lässt sich mit `~` (nicht) verneinen, d.h. aus *wahr* wird *falsch* und umgekehrt.

# Logische Ausdrücke

- Vergleichen wir zwei Matrizen oder Vektoren **A** und **B**, müssen sie gleich gross sein, da die jeweils korrespondierenden Einträge  $a_{ij}$  und  $b_{ij}$  miteinander verglichen werden. Das Resultat ist dann eine gleich grosse Matrix mit Einträgen 1 oder 0.
- Es können Matrizen mit einer einzigen Zahl verglichen werden. Matlab vergleicht dann jeden Eintrag der Matrix einzeln mit dieser einen Zahl.
- Beispiel:

```
» x = 1:9; x >= 5
```

```
ans =
```

```
0 0 0 0 1 1 1 1 1
```

# Logische Ausdrücke

Quadrieren aller negativen Zahlen in einem Vektor:

```
» x = -4:4;  
» my_indices = x < 0  
  
my_indices =  
    1x9 logical array  
    1 1 1 1 0 0 0 0 0  
  
» x(my_indices) = x(my_indices).^2  
  
x =  
    16  9  4  1  0  1  2  3  4
```



# Speicherverwaltung

- › Wenn auf ein Matriceintrag *zugegriffen* wird, etwa durch  $A(2, 3:6)$ , so muss die Matrix  $A$  mindestens die Größe  $2 \times 6$  besitzen und entsprechend *vorher* definiert werden, z.B. als  $A = \text{zeros}(2, 6)$ .
- › Wenn  $A(2, 3:6)$  ein Wert *zugewiesen* wird, ohne die Matrix vorher zu definieren oder falls sie zu klein ist, so wird sie automatisch als Nullmatrix der notwendigen Größe angelegt oder entsprechend vergrößert. Das kann zu Fehlern führen und die Laufzeit der Programme verlängern.

›››› **Lösung:** Initialisiere alle Variablen die gebraucht werden!  
Also, Anfangswerte für Variablen setzen bevor gerechnet wird.

- Speicher nicht mehr benötigter Variablen freigeben durch:

`clear` oder `clear+Variablename`

- `clc` löscht alle vorherigen Eingaben und Ausgaben im Command Window.

# Initialisieren

Es ist sinnvoll, Vektoren und Matrizen, die in Schleifen (in Matlab: `for`- und `while`-Schleifen) verwendet werden, vor Beginn der Schleife zu initialisieren.

```
n=1e8; v=[];  
tic  
for i=1:n  
    v(i) = i;  
end  
toc
```

**Ausgabe:**

```
Elapsed time is  
7.615093 seconds.
```

V  
E  
R  
S  
U  
S

```
n=1e8; v=zeros(1,n);  
tic  
for i=1:n  
    v(i) = i;  
end  
toc
```

**Ausgabe:**

```
Elapsed time is  
0.244577 seconds.
```

**Vorteile der Initialisierung:**

Performance-Verbesserung, Speichereffizienz, Fehlervermeidung, Lesbarkeit des Codes, usw.

# Matrix-Vektor-Schreibweise

Es gibt (mindestens) zwei Möglichkeiten, den Vektor

$$v = ( 1 \quad -2 \quad 3 \quad -4 \quad \dots \quad \dots \quad 999 \quad -1000 )$$

zu erzeugen:

```
v = 1:1000;  
for i = 1:500  
    v(2*i) = -v(2*i);  
end
```

```
v = 1:1000;  
v(2:2:1000) = -v(2:2:1000);
```

Die zweite Möglichkeit verwendet statt **Schleifen** die **vektorielle** Rechnung. Das ist viel schneller!

Matlab rechnet schneller, wenn alle Matrix- bzw. Vektor-Operationen möglichst **ohne** for-Schleifen umgesetzt werden.