



HELMUT SCHMIDT  
UNIVERSITÄT

---

Universität der Bundeswehr Hamburg

# **MATLAB - Grundlagen für Ingenieurwissenschaften**

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>2</b>
1.1	Was ist MATLAB? . . . . .	2
1.2	Anwendungsgebiete in den Ingenieurwissenschaften . . . . .	2
1.3	Die Benutzeroberfläche . . . . .	2
<b>2</b>	<b>Grundlegende Operationen</b>	<b>5</b>
2.1	Variablendeklaration . . . . .	5
2.2	Mathematische Grundoperationen . . . . .	6
2.3	Komplexe Zahlen . . . . .	7
2.4	Beispielaufgaben . . . . .	8
<b>3</b>	<b>Vektoren und Matrizen</b>	<b>9</b>
3.1	Erstellen von Vektoren und Matrizen . . . . .	9
3.2	Zugriff auf Elemente und Indizierung . . . . .	10
3.3	Matrixoperationen . . . . .	11
3.4	nützliche MATLAB Funktionen . . . . .	12
3.5	Beispielaufgaben . . . . .	12
<b>4</b>	<b>Programmiergrundlagen</b>	<b>14</b>
4.1	Skripte . . . . .	14
4.2	Funktionen . . . . .	15
4.3	Schleifen und Bedingungen . . . . .	15
4.4	Beispielaufgaben . . . . .	17
<b>5</b>	<b>Arbeiten mit Dateien und Daten</b>	<b>19</b>
5.1	Speichern und Laden von Daten . . . . .	19
5.2	Importieren von Messdaten . . . . .	19
5.3	Analyse und Verarbeitung von Daten . . . . .	20
<b>6</b>	<b>Visualisierung von Daten</b>	<b>21</b>
6.1	Einfache Diagramme . . . . .	21
6.2	Mehrere Kurven in einem Diagramm . . . . .	21
6.3	Mehrere Diagramme in einer Übersicht . . . . .	21
6.4	Grafische Anpassungen . . . . .	21
<b>7</b>	<b>Anhang</b>	<b>22</b>
7.1	Dokumentation in MATLAB . . . . .	22
7.2	Übersicht wichtiger MATLAB Befehle . . . . .	22

# 1 Einführung

## 1.1 Was ist MATLAB?

MATLAB ist die Abkürzung für MATrix LABoratory. Zudem ist es ein interaktives, integriertes System zur Berechnung, Visualisierung oder Programmierung mathematischer Problemstellungen. Es bietet eine einfache Skriptsprache welche auf die Verarbeitung von Matrizen ausgelegt ist.

## 1.2 Anwendungsgebiete in den Ingenieurwissenschaften

MATLAB bietet in vielen Ingenieurwissenschaftlichen Betätigungsfeldern weitreichende Vorteile.

- Signalverarbeitung
- Regelungstechnik
- FEM-Simulation
- Schaltungsanalyse
- Bildverarbeitung
- Datenanalyse

## 1.3 Die Benutzeroberfläche

### Command Window

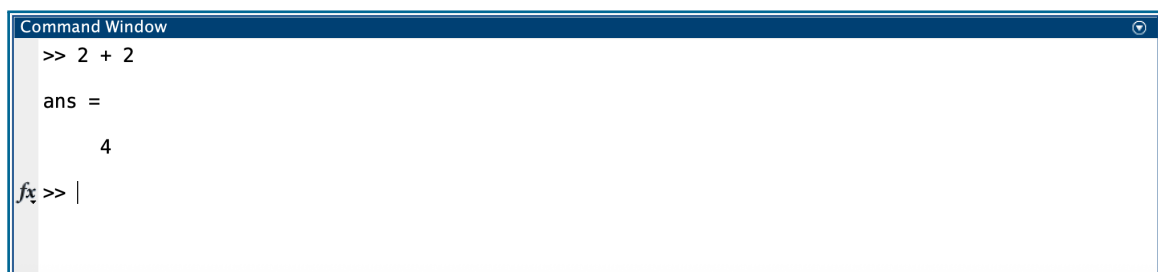
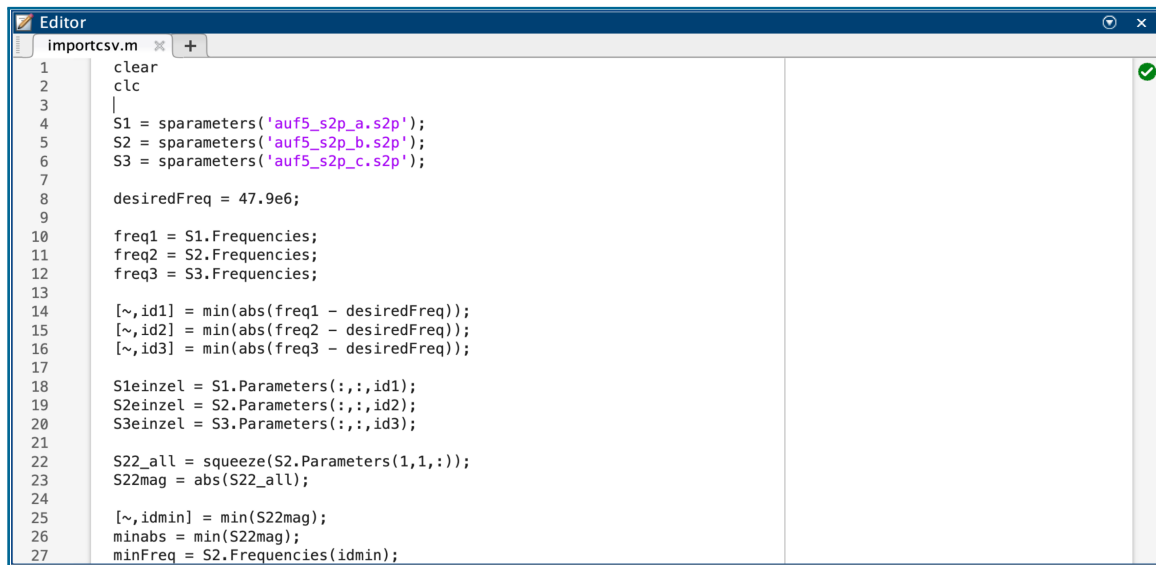


Abbildung 1: Command Window in MATLAB

Im Command Window können Befehle direkt eingegeben werden. Da Ergebnisse von Berechnungen unverzüglich angezeigt werden, können hier einzelne Befehle idealerweise getestet werden.

## Editor

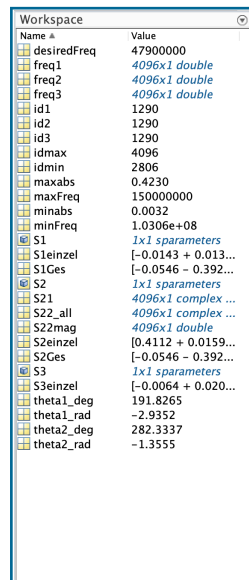


```
1 clear
2 clc
3 |
4 S1 = sparameters('auf5_s2p_a.s2p');
5 S2 = sparameters('auf5_s2p_b.s2p');
6 S3 = sparameters('auf5_s2p_c.s2p');
7
8 desiredFreq = 47.9e6;
9
10 freq1 = S1.Frequencies;
11 freq2 = S2.Frequencies;
12 freq3 = S3.Frequencies;
13
14 [~,id1] = min(abs(freq1 - desiredFreq));
15 [~,id2] = min(abs(freq2 - desiredFreq));
16 [~,id3] = min(abs(freq3 - desiredFreq));
17
18 S1einzel = S1.Parameters(:,id1);
19 S2einzel = S2.Parameters(:,id2);
20 S3einzel = S3.Parameters(:,id3);
21
22 S22_all = squeeze(S2.Parameters(1,1,:));
23 S22mag = abs(S22_all);
24
25 [~,idmin] = min(S22mag);
26 minabs = min(S22mag);
27 minFreq = S2.Frequencies(idmin);
```

Abbildung 2: Editor in MATLAB

Im Editor können komplette Skripte und Funktionen geschrieben, gespeichert und ausgeführt werden. Er unterstützt das Debugging mittels Breakpoints und Schritt-für-Schritt Ausführung.

## Workspace



Name	Value
desiredFreq	47900000
freq1	4096x1 double
freq2	4096x1 double
freq3	4096x1 double
id1	1290
id2	1290
id3	1290
idmax	4096
idmin	2806
maxabs	0.4230
maxFreq	150000000
minabs	0.0032
minFreq	1.0306e+08
S1	1x1 sparameters
S1einzel	[-0.0143 + 0.013...
S1Ges	[-0.0546 - 0.392...
S2	1x1 sparameters
S21	4096x1 complex ...
S22_all	4096x1 complex ...
S22mag	4096x1 double
S2einzel	[0.4112 + 0.0159...
S2Ges	[-0.0546 - 0.392...
S3	1x1 sparameters
S3einzel	[-0.0064 + 0.020...
theta1_deg	191.8265
theta1_rad	-2.9352
theta2_deg	282.3337
theta2_rad	-1.3555

Abbildung 3: Workspace in MATLAB

Im Workspace werden alle aktuellen Variablen inklusive ihres Inhalts angezeigt. Weiterhin ist es möglich diese Variablen hier manuell anzupassen oder zu löschen.

## Current Folder

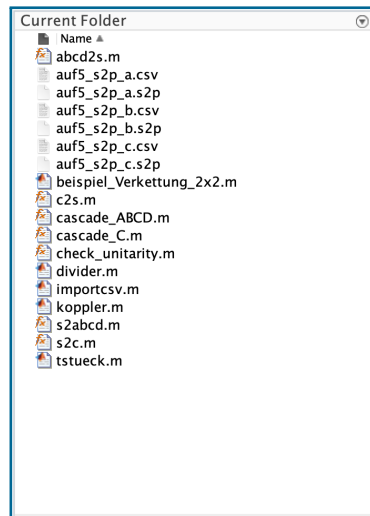


Abbildung 4: Current Folder in MATLAB

Im Current Folder findet man alle Dateien des Projektordners. Diese können durch Doppelklick oder das Ziehen in den Editor geöffnet und bearbeitet werden.

## 2 Grundlegende Operationen

### 2.1 Variablendeklaration

#### Einfache Wertzuweisung

```
a = 3;
```

Der Variable `a` wird der Wert `3` zugewiesen.

Eine Zuweisung ohne ein Semikolon am Ende der Zeile bewirkt eine direkte Rückgabe des Variablenwertes.

#### Fließkommazahl

```
a = 4.5;
```

Der Variable `a` wird der Wert `4.5` zugewiesen. Als Trennzeichen in MATLAB wird der Punkt an Stelle eines Kommas verwendet.

#### Zeichenkette

```
name = "Peter";
```

Der Variable `name` wird der String `Peter` zugewiesen.

#### Logischer Wert

```
isValid = true;
```

Der Variable `isValid` wird der boolsche Wert `true` zugewiesen.

#### Automatische Typzuweisung

```
a = pi;
```

Der Variable `a` wird die, in MATLAB vordefinierte Variable  $\pi$  zugewiesen.

Neben `pi` gibt es weitere vordefinierte Variablen. Diesen kann zwar ebenfalls ein selbst definierter Wert zugewiesen werden, jedoch ist dies nicht empfehlenswert.

Variable	Bedeutung	Wert
<code>inf</code>	Unendlich	$\frac{1}{0}$ ergibt <code>inf</code>
<code>i</code>	Imaginäre Einheit	$\sqrt{-1}$
<code>j</code>	Alternative imaginäre Einheit	$\sqrt{-1}$
<code>NaN</code>	"Not a Number ungültiger Wert"	$\frac{0}{0}$ ergibt <code>NaN</code>
<code>ans</code>	Ergebnis der letzten berechneten Zeile	z.B. <code>ans = 42</code>
<code>true</code> / <code>false</code>	Boolsche Werte	<code>1</code> bzw. <code>0</code>

## 2.2 Mathematische Grundoperationen

### Addition

```
c = a + b;
```

In der Variable **c** wird die Summe aus **a** und **b** gespeichert.

### Subtraktion

```
c = a - b;
```

In der Variable **c** wird die Differenz aus **a** und **b** gespeichert.

### Multiplikation

```
c = a * b;
```

In der Variable **c** wird das Produkt aus **a** und **b** gespeichert.

### Division

```
c = a / b;
```

In der Variable **c** wird der Quotient aus **a** und **b** gespeichert.

### Abrunden

```
c = floor(a / b);
```

In der Variable **c** wird das abgerundete Ergebnis der Division von **a** und **b** gespeichert.

### Aufrunden

```
c = ceil(a / b);
```

In der Variable **c** wird das aufgerundete Ergebnis der Division von **a** und **b** gespeichert.

### Modulo

```
c = mod(a, b);
```

In der Variable **c** wird der Rest der Division von **a** und **b** gespeichert.

### Potenzieren

```
c = a ^ 2;
```

In der Variable **c** wird das Ergebnis der zweiten Potenz von **a** gespeichert.

### Wurzeln

```
c = sqrt(a);
```

In der Variable **c** wird die Wurzel von **a** gespeichert.

### Betrag

```
c = abs(-a);
```

In der Variable `c` wird der Betrag von `-a` gespeichert.

## 2.3 Komplexe Zahlen

### Definition der komplexen Zahl

```
z = 2 + 3*i;
```

Erzeugt die komplexe Zahl `z = 2 + 3i`.

Wie unter 2.1 beschrieben, kann `j` analog zu `i` verwendet werden.

### Real- und Imaginärteil

```
re = real(z);  
im = imag(z);
```

`real()` gibt den Realteil von `z` zurück und `imag()` den Imaginärteil.

### Betrag

```
r = abs(z);
```

Berechnet den Betrag von `z`, also  $\sqrt{\text{Im}(z)^2 + \text{Re}(z)^2}$ .

### Winkel

```
phi = angle(z);
```

Gibt den Winkel von `z` im Bogenmaß zurück.

### Konjugation

```
z_conj = conj(z);
```

Gibt das konjugiert Komplexe der Variable `z` also  $z^* = \text{Re}(z) - i \cdot \text{Im}(z)$  zurück.

### Darstellung in Polarform

```
r = abs(z);  
phi = angle(z);  
z_polar = r * exp(1i*phi);
```

Gibt die komplexe Zahl `z` in Polarform zurück. `exp(1i * phi)` steht für  $e^{i\phi}$



## 2.4 Beispielaufgaben

### Aufgabe 1

Gegeben sei die Funktion  $f(x) = x^2 + 4x + 5$ . Berechnen Sie die komplexen Nullstellen der Funktion und lassen Sie sich jeweils Betrag und Phase ausgeben.

#### Lösung 1

```
p = 4;
q = 5;

x1 = -p/2 + sqrt((p/2)^2 - q);
x2 = -p/2 - sqrt((p/2)^2 - q);

r1 = abs(x1);
r2 = abs(x2);

phi1 = angle(x1);
phi2 = angle(x2);
```

### Aufgabe 2

Eine elektrische Schaltung besteht aus einem Widerstand mit  $R = 10\Omega$  einer Spule mit  $L = 0,05H$  und einem Kondensator mit  $C = 100\mu F$ . Die Reihenschaltung der drei Elemente wird bei einer Frequenz von  $f = 50Hz$  betrieben. Berechnen Sie die Gesamtimpedanz  $Z$  dieser Schaltung.

#### Lösung 2

```
R = 10;
L = 0.05;
C = 100e-6;
f = 50;
omega = 2 * pi * f;

Z_R = R;
Z_L = 1j * omega * L;
Z_C = 1 / (1j * omega * C);

Z_Gesamt = Z_R + Z_L + Z_C;
```

## 3 Vektoren und Matrizen

### 3.1 Erstellen von Vektoren und Matrizen

Zeilenvektor	
<code>v = [1 2 3 4];</code>	Erzeugt einen Zeilenvektor mit den angegebenen Werten. Statt der Trennung durch ein Leerzeichen können ebenfalls Kommata verwendet werden.
Spaltenvektor	
<code>v = [1;2;3;4];</code>	Erzeugt einen Spaltenvektor mit den angegebenen Werten.
Doppelpunktoperator I	
<code>v = x1:x2;</code>	Erzeugt einen Zeilenvektor von <code>x1</code> bis <code>x2</code> in ganzzahligen Schritten.
Doppelpunktoperator II	
<code>v = x1:step:x2;</code>	Erzeugt einen Zeilenvektor von <code>x1</code> bis <code>x2</code> in konstanten Schritten von <code>step</code> .
linspace	
<code>v = linspace(x1,x2,n);</code>	Erzeugt einen Zeilenvektor von <code>x1</code> bis <code>x2</code> mit <code>n</code> gleichmäßig verteilten Werten.
Matrizen	
<code>A = [1 2 3; 4 5 6; 7 8 9];</code>	Elemente einer Zeile der Matrix werden wie bei den Vektoren mit Leerzeichen oder Komma getrennt. Ein Zeilenumbruch erfolgt durch Eingabe eines Semikolon.
0-Matrix	
<code>A = zeros(m,n);</code>	Erzeugt eine 0-Matrix der Größe <code>m x n</code> . <code>ones()</code> funktioniert analog zu <code>zeros()</code> nur mit einsen.
Einheitsmatrix	
<code>A = eye(n);</code>	Erzeugt die Einheitsmatrix der Größe <code>n x n</code> .

## 3.2 Zugriff auf Elemente und Indizierung

### Einfache Indizierung

```
V = [10 20 30 40];  
V(2)
```

Gibt den zweiten Wert des Vektors also **20** zurück.

### Indizierung in Matrizen

```
A = [1 2 3; 4 5 6; 7 8 9];  
A(2,3)
```

Gibt den dritten Wert der zweiten Zeile also **6** zurück.

### Doppelpunktoperator

```
A = [1 2 3; 4 5 6; 7 8 9];  
A(:,3)
```

Gibt alle Werte der dritten Spalte als Spaltenvektor zurück.

### End-Schlüsselwort

```
A(end);  
A(end-1);  
A(:,end);
```

Letztes Element  
Vorletztes Element  
Letzte Spalte

### Logische Indizierung

```
A = [1 2 3; 4 5 6; 7 8 9];  
A(A>5)
```

Gibt den Spaltenvektor mit den Werten **7**, **8**, **6**, **9** zurück. MATLAB prüft jedes Element der Matrix und gibt diejenigen zurück, die größer als **5** sind. Dabei wird die Matrix spaltenweise (spaltenweise Linearindizierung) durchlaufen.

### Ändern von Werten

```
V = [1 2 3 4];  
V(3) = 7
```

Ersetzt den dritten Wert des Vektor durch 7.

### 3.3 Matrixoperationen

#### Elementweise Operationen

```
A = [1 2; 3 4];  
B = [5 6; 7 8];
```

```
C = A + B;  
D = A - B;  
E = A .* B;  
F = A ./ B;  
G = A .^ 2;
```

Für Elementweise Operationen muss ein Punkt vor dem Operator genutzt werden. Bei Addition und Subtraktion ist dies jedoch irrelevant.

#### Matrixmultiplikation

```
A = [1 2; 3 4];  
B = [5; 6];  
  
C = A * B;
```

Für die klassische Matrixmultiplikation gelten die allgemeinen Regeln aus der linearen Algebra. Beim Operator muss hier auf den Punkt verzichtet werden.

#### Transposition

```
A = [1 2; 3 4];  
B = A'
```

Für Matrizen mit komplexen Zahlen erzeugt das Hochkomma die adjungierte Matrix. Für reine Transposition wird `.'` verwendet. Bei reellen Matrizen können beide Versionen analog verwendet werden.

#### Lösen linearer Gleichungssysteme der Form $Ax = b$

```
A = [1 2; 3 4];  
b = [5; 6];  
x = A \ b;
```

Zum Lösen von linearen Gleichungssystemen wird der Backslash verwendet.

### 3.4 nützliche MATLAB Funktionen

Funktion	Rückgabe
<code>max(A)</code>	Höchster Wert der Matrix
<code>min(A)</code>	Kleinsten Wert der Matrix
<code>sum(A)</code>	Summe der einzelnen Spalten
<code>mean(A)</code>	Mittelwert der einzelnen Spalten
<code>length(A)</code>	Zeilenanzahl der Matrix
<code>numel(A)</code>	Spaltenanzahl der Matrix
<code>size(A)</code>	Zeilen- und Spaltenanzahl der Matrix
<code>det(A)</code>	Determinante der Matrix
<code>rank(A)</code>	Rang der Matrix
<code>trace(A)</code>	Summe der Diagonalelemente
<code>eig(A)</code>	Eigenwerte der Matrix
<code>flipud(A)</code>	vertikale Spiegelung der Matrix
<code>fliplr(A)</code>	horizontale Spiegelung der Matrix

### 3.5 Beispielaufgaben

#### Aufgabe 1

Gegeben seien folgende Matrizen:  $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$  und  $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ .

1. Berechnen Sie die Summe der beiden Matrizen
2. Berechnen Sie das elementweise Quadrat von A.
3. Bestimmen Sie die transponierte von B.

#### Lösung 1

```
A = [1 2; 3 4];
```

```
B = [5 6; 7 8];
```

```
C = A + B;
```

```
D = A .^ 2;
```

```
E = B';
```

## Aufgabe 2

Gegeben sei die Matrix  $A = \begin{pmatrix} 2 & 5 & 8 \\ 7 & 9 & 3 \\ 6 & 1 & 0 \end{pmatrix}$

1. Geben Sie das 2. Element der ersten Zeile aus.
2. Ersetzen Sie das 2. Element der dritte Spalte durch 1
3. Ermitteln Sie alle Einträge, die größer als 6 sind.

### Lösung 2

```
A = [2 5 8; 7 9 3; 6 1 0];
```

```
a = A(1,2);
```

```
A(2,3) = 1;
```

```
B = A(A>6);
```

## Aufgabe 3

Gegeben sei folgendes lineares Gleichungssystem:

$$2x + 3y - z = 1$$

$$4x + 1y + 1z = 9$$

$$-2x + 5y + 2z = 2$$

1. Lösen Sie das Gleichungssystem.
2. Überprüfen Sie das Ergebnis durch Rückeinsetzen.

### Lösung 3

```
A = [2 3 -1; 4 1 1; -2 5 2];
```

```
b = [1; 9; 2];
```

```
x = A \ b;
```

```
test = A * x;
```

## 4 Programmiergrundlagen

### 4.1 Skripte

Ein Skript ist eine Sammlung von MATLAB Befehlen, die in einer Datei gemeinsam abgespeichert und ausgeführt werden können. Die Dateierweiterung eines solchen Skripts ist

`.m`

#### Vorgehensweise

- Rechtsklick im Current Folder → New → Script
- Eingeben der gewünschten MATLAB Befehle im Editor analog zur Verwendung im Command Window
- Speichern des Skriptes
- Ausführen durch Eingabe des Dateinamens ohne Dateierweiterung im Command Window oder den Run Button in der Navigationsleiste bei geöffnetem Editor.

#### Kommentare

Mittels des `%` Zeichens kann ein Kommentar eingefügt werden, das beim Ausführen des Skriptes nicht beachtet wird.

#### Skriptbeispiel

```
% Erzeuge 3x3 Matrix
A = [1 2 3; 4 5 6; 7 8 9];

% Erste Spalte der Matrix als Spaltenvektor
b = A(:,1);

% Transponiert b zu Zeilenvektor
c = b';
```

#### Vorteile von Skripten gegenüber der Eingabe im Command Window

- Skripte können beliebig oft ausgeführt werden, ohne die Befehle jedes mal erneut eingeben zu müssen.
- Rechenweg bleibt komplett dokumentiert und kann einfacher überprüft und angepasst werden.
- Komplexe Abläufe lassen sich klar gliedern und durch Kommentare strukturieren.

## 4.2 Funktionen

Funktionen werden ebenfalls in einem `.m`-File gespeichert, enthalten im Gegensatz zu einem einfachen Skript jedoch Ein- und Ausgabeargumente und werden mit einem `end` beendet.

```
function [Ausgabe] = Funktionsname(Eingabe)
```

Die Verwendung von eckigen Klammern ist lediglich bei Verwendung mehrerer Ausgabeargumente notwendig, kann jedoch auch bei nur einem Argument verwendet werden. Der Funktionsname sollte dem Dateinamen entsprechen, um die Funktion auch in anderen Skripten ausführen zu können.

### Funktionsbeispiel

```
function A = Flaecheninhalt(r)

A = pi * r^2;
end
```

Die Funktion kann nun durch Eingabe im Command Window oder innerhalb eines Skriptes mit beispielsweise

```
Flaecheninhalt(3)
```

aufgerufen werden und gibt somit den Flächeninhalt eines Kreises mit dem Radius 3 zurück.

## 4.3 Schleifen und Bedingungen

Schleifen ermöglichen das mehrfache Ausführen eines Codeblocks, was jedoch immer an eine Bedingung geknüpft ist.

### for-Schleife

Der Codeblock wird für eine definierte Anzahl an Durchläufen ausgeführt. Auch eine Schleife muss immer mit einem `end` beendet werden.

### Beispiel for-Schleife

```
summe = 0;
for i = 1:10
    summe = summe + i;
end
```

Diese Schleife berechnet die Summe der Zahlen 1 bis 10.



## while-Schleife

Der Codeblock wird so lange ausgeführt, bis eine gegebene Bedingung nicht mehr erfüllt wird.

### Beispiel while-Schleife

```
n = 0;
zahl = 2;

while zahl < 1000
    zahl = zahl^2;
    n = n+1;
end
```

In diesem Fall wird geprüft, wie oft man eine Zahl quadrieren kann, bevor sie den Wert 1000 überschreitet.

## if-Bedingungen

**if**-Bedingungen sind in der Programmierumgebung wichtig, um auf unterschiedliche Arten von Eingabewerten zu reagieren. Je nach Ergebnis einer logischen Prüfung, wird nur ein Teil des bestehenden Codes ausgeführt.

### Beispiel if-Bedingung

```
T = 80;

if T>100
    disp('Dampf');
elseif T > 0
    disp('fluessig');
else
    disp('gefroren')
end
```

## Logische Operatoren

Operator	Bedeutung
<b>==</b>	gleich
<b>~=</b>	ungleich
<b>&lt;</b>	kleiner als
<b>&gt;</b>	größer als
<b>&lt;=</b>	kleiner gleich
<b>&gt;=</b>	größer gleich

## 4.4 Beispielaufgaben

### Aufgabe 1

Gegeben seien die beiden Vektoren

$$a = \begin{pmatrix} 2 & 4 & 7 \end{pmatrix}$$

$$b = \begin{pmatrix} 1 & 6 & 3 \end{pmatrix}$$

Führen Sie folgende Aufgaben innerhalb eines Skriptdes durch.

1. Addieren Sie die beiden Vektoren.
2. Multiplizieren Sie die beiden Vektoren elementweise.
3. Lassen Sie sich beide Ergebnisse mittels `disp()` ausgeben.

### Lösung 1

#### vectorAddMul.m

```
a = [2 4 7];  
b = [1 , 6 , 3]  
c = a + b;  
d = a .* b;  
disp(c);  
disp(d);
```

Ausführen des Skriptes mittels Command Window Eingabe.

```
vectorAddMul
```

### Aufgabe 2

Schreiben Sie eine Funktion `nullstellen(a,b,c)` zur Berechnung von Nullstellen einer beliebigen quadratischen Funktion. Nutzen Sie hierfür die Mitternachtsformel:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Lassen Sie sich anschließend die Nullstellen der folgenden Funktion ausgeben.

$$f(x) = (1 + i)x^2 - (5 + i)x + 10$$

### Lösung 2

#### nullstellen.m

```
function [x1, x2] = nullstellen(a,b,c)  
    x1 = (-b+sqrt((b^2)-4*a*c))/(2*a);  
    x2 = (-b-sqrt((b^2)-4*a*c))/(2*a);  
end
```

Aufruf der Funktion mittels der Command Window Eingabe

```
[a,b] = nullstellen(1 +1*i, -5 - 1*i, 10)
```

### Aufgabe 3

Schreiben Sie eine MATLAB Funktion `flaeche(a,b,c,xStart,xEnd)` welche mittels numerischer Integration und der Rechteckregel das bestimmte Integral einer quadratischen Funktion der Form

$$f(x) = ax^2 + bx + c$$

approximieren kann. Zudem soll die Funktion nur dann das Integral berechnen, wenn der Startwert kleiner als der Endwert ist.

$$A = \int_{xStart}^{xEnd} f(x)dx \approx \Delta x \cdot \sum_{i=0}^{n-1} f(x_i)$$

Berechne anschließend

$$\int_0^3 3x^2 - 2x + 8dx$$

### Lösung 3

#### flaeche.m

```
function A = flaeche(a, b, c, xStart, xEnd)
    if xEnd > xStart
        n = 100000;
        dx = (xEnd - xStart) / n;
        x = xStart : dx : xEnd - dx;
        f = a*x.^2 + b*x + c;
        A = sum(f) * dx;
    else
        disp('Endwert_muss_hoher_als_Startwert_sein');
    end
end
```

Aufruf der Funktion mittels der Command Window Eingabe

```
A = flaeche(3,-2,8,0,3)
```

## 5 Arbeiten mit Dateien und Daten

### 5.1 Speichern und Laden von Daten

#### alle Variablen speichern

```
save('dateiname.mat');
```

Speichert alle Variablen des Workspaces in einer `.mat` Datei.

#### einzelne Variablen speichern

```
save('dateiname.mat',  
    'variable');
```

Speichert eine ausgewählte Variable in einer `.mat` Datei.

#### alle Variablen laden

```
load('dateiname.mat');
```

Lädt alle Variablen einer `.mat` Datei in den Workspace.

#### einzelne Variablen laden

```
load('dateiname.mat',  
    'variable');
```

Lädt die ausgewählte Variable einer `.mat` Datei in den Workspace.

Alternativ zum Laden mittels `load()` kann dies auch über einen Doppelklick auf die `.mat` Datei im Current Folder ausgeführt werden.

#### Speichern als .csv Datei

```
dlmwrite('dateiname.csv',  
    Matrixname);
```

Speichert eine ausgewählte Matrix als `.csv` Datei.

#### Speichern als ASCII

```
save('dateiname.txt',  
    variable, '-ascii');
```

Speichert eine ausgewählte Variable im ASCII-Format in einer Textdatei. (Zum Austausch mit anderen Systemen)

### 5.2 Importieren von Messdaten

#### Import von Tabellen

```
readtable('dateiname');
```

Importiert eine Tabelle mit Headerzeilen und gibt ein `table` - Objekt zurück.

### Import von Matrizen

```
readmatrix('dateiname');
```

Importiert eine rein numerische Tabelle, ignoriert Headerzeilen und gibt eine `double` - Matrix zurück.

## 5.3 Analyse und Verarbeitung von Daten

### Mittelwert

```
mean(daten);
```

Gibt den Spaltenweisen Mittelwert einer Datenmatrix oder den Mittelwert eines Spaltenvektors zurück.

### Standardabweichung

```
std(daten);
```

Gibt die spaltenweise Standardabweichung einer Datenmatrix oder die Standardabweichung eines Datenvektors zurück.

## 6 Visualisierung von Daten

### 6.1 Einfache Diagramme

### 6.2 Mehrere Kurven in einem Diagramm

### 6.3 Mehrere Diagramme in einer Übersicht

### 6.4 Grafische Anpassungen

## 7 Anhang

### 7.1 Dokumentation in MATLAB

### 7.2 Übersicht wichtiger MATLAB Befehle