# Gliederung einer Programmeinheit

Eine Programmeinheit besteht aus Anweisungen und Kommentaren.

- Normalerweise eine Anweisung pro Zeile
- Mehrere Anweisungen in einer Zeile sind möglich. Die Trennung erfolgt mit Kommata (oder Semikolon, s.u.).
- Eine Anweisung über mehrere Zeilen ist ebenfalls möglich. Als Fortsetzungsmarkierung werden ... verwendet.
- Man unterscheidet zwei Formen einer Anweisung:

| | |
|---|---|
| *Anweisung* | Auswertung mit Anzeige des Ergebnisses |
| *Anweisung*; | Auswertung ohne Anzeige des Ergebnisses |

- Kommentarzeilen werden mit % eingeleitet.

Anweisungen enthalten folgende Grundelemente:

| | | |
|---|---|---|
| a) | Schlüsselwörter | input, abs, while, pi |
| b) | Konstanten | 1, 3.1415, 'TEXT' |
| c) | symbolische Namen | A, a, S, Radius |
| d) | Operatoren | +,-,/,^ |
| e) | Sonderzeichen | (,=,) |

## Ganzzahl–Konstanten

- Ganze Zahlen mit (optionalem) Vorzeichen.
  Beispiel: +12, 12, -373

## Reelle Konstanten

- Festpunktdarstellung:
  Vorzeichen | ganzz. Anteil | Dezimalpunkt | gebroch. Anteil
  Beispiel: -7.12, 7.12, .1, 1.
- Gleitpunktdarstellung:
  Festpunktdarstellung | e | ganze Zahl
  Beispiel: 7.12e1, 1e1, .1e1
- Zahlen werden ausschließlich als doppelt genaue reelle (bzw. komplexe) Zahlen gemäß dem *IEEE Standard Binary Floating Point Arithmetic* dargestellt.
- 8 Byte (64 Bit) stehen zur Verfügung, und zwar 1 Bit für das Vorzeichen, 11 Bit für den Exponenten und 52 Bit für die Mantisse (normalisiert).
- Kleinste darstellbare Zahl: `realmin`$\approx 2.2 \cdot 10^{-308}$,
- Größte darstellbare Zahl: `realmax`$\approx 1.8 \cdot 10^{308}$.

Ausnahmen (Arithmetische Katastrophen):

- Eine Division durch 0 führt auf das Ergebnis `Inf` (für *infinity*).
- Eine Division von 0 durch 0 führt auf `NaN` (für *Not a Number*).

## Symbolische Namen

- Zulässige Zeichen sind
  - die 26 Großbuchstaben A-Z
  - die 26 Kleinbuchstaben a-z
  - die 10 Ziffern 0-9
  - der Unterstrich _
- Symbolische Namen müssen mit einem Buchstaben beginnen
- Groß– und Kleinschreibung wird unterschieden (A≠a)
- Symbolische Namen dürfen kein Schlüsselwort sein.
- Die signifikante Länge beträgt 19 Zeichen in älteren MATLAB-Versionen. Ab MATLAB 6 beträgt die signifikante Länge 63 Zeichen und kann mittels der Variablen namelengthmax abgefragt werden.

## Variablen

- Variablen (z.B solche, die als Werte reelle Zahlen annehmen sollen), werden NICHT explizit vereinbart, sondern stehen mit der Zuweisung/Benutzung automatisch zur Verfügung.
- Die Variablennamen müssen dabei den Bedingungen für symbolische Namen genügen.

# Arithmetische Ausdrücke

Arithmetische Ausdrücke (arithmetische Konstanten, Variablen etc.) können mit folgenden Operatoren und Klammern verknüpft werden:

| Operator | Funktion | Priorität |
|---|---|---|
| + | Addition | niedrig |
| - | Subtraktion | niedrig |
| $*$ | Multiplikation | mittel |
| / | Division | mittel |
| ^ | Exponentiation | hoch |
| ( ) | Klammer | sehr hoch |

Es sind folgende Regeln zu beachten:

- Weder zwei Operanden noch zwei Operatoren dürfen direkt nebeneinander stehen. Ausnahme: Exponentiationszeichen und Vorzeichen des Exponenten.
- Das Multiplikationszeichen $*$ muß immer angegeben werden.
- Arith. Ausdrücke, die einen nicht def. Wert besitzen, sind zulässig, aber nicht sinnvoll.

Beispiele:

- 3-9
- pi$*$Radius
- a/(-A)
- 1/3
- 1./3^2

## Die Auswertung arithmetischer Ausdrücke

- Die Auswertung eines arithmetischen Ausdrucks folgt den üblichen mathematischen Regeln.
- Die Auswertung geschachtelter Klammerausdrücke erfolgt von innen nach außen.
- Gleichrangige Operationen werden von links nach rechts ausgewertet.
- Klammerausdrücke haben stets die höchste Priorität.

## Die Wertzuweisung

Allgemeine Form: *Variable* = *arithmetischer Ausdruck*

Beispiel:

```
a = a * x / (n+1)
```

## Die Variableneingabe

Allgemeine Form: *Variable* = input('Text')

Beispiel:

```
x = input('Bitte x eingeben: ')
```

## Die Ausgabe

Einfache Form: *Variable*

Beispiel:

```
x
```

Ausgabe von Variablen ohne den Variablennamen auszugeben:
disp(*Variable*)

Beispiel:

```
disp(x)
```

## Einfache Formatierung der Ausgabe

| | |
|---|---|
| format short | (Skalierte) Festpunktdarstellung mit 5 Stellen bzw. 4 Nachkommastellen |
| format long | (Skalierte) Festpunktdarstellung mit 15 Stellen bzw. 14 Nachkommastellen |
| format short e | Gleitpunktdarstellung mit 5 Stellen bzw 4 Nachkommastellen |
| format long e | Gleitpunktdarstellung mit 15 Stellen bzw. 14 Nachkommastellen |
| format hex | Hexadezimale Ausgabe |
| format + | Nur Ausgabe des Vorzeichens (+, -, Leerzeichen) |
| format bank | „Geld“–Format (Zwei Nachkommastellen) |
| format compact | Ausgabe ohne Extra–Leerzeilen |
| format loose | Ausgabe mit Extra–Leerzeilen |
| format rat | Ausgabe mittels Approximation eines Quotienten aus zwei kleinen ganzen Zahlen |

Die Formateinstellungen bleiben solange erhalten bis sie erneut geändert werden. Die Befehle `format compact` und `format loose` sind unabhängig von den anderen Format–Befehlen.
(Hinweis: Alle Berechnungen werden in MATLAB doppelt genau durchgeführt)

## Die „while“–Schleife

Allgemeine Form:

while *Variable*
    *Anweisung*
        .
        .
        .
    *Anweisung*
end

Übliche Form:

while *Vergleichsausdruck*
    *Anweisung*
        .
        .
        .
    *Anweisung*
end

Funktionsweise: Solange die *Variable* ungleich Null ist bzw. der *Vergleichsausdruck* wahr ist, werden die Anweisungen zwischen `while` und `end` der Reihe nach ausgeführt.

# (Arithm.) Vergleichsausdrücke:

Allgemeine Form:

arithm. Ausdruck | Relationsoperator | arithm. Ausdruck

| Relationsoperator | Bedeutung |
|---|---|
| < | kleiner |
| <= | kleiner gleich |
| > | größer |
| >= | größer gleich |
| == | gleich |
| ~= | ungleich |

Die Verknüpfung logischer Ausdrücke erfolgt mittels:

| | |
|---|---|
| \| | logisches ODER |
| & | logisches UND |
| ~ | logische Negation |

# Logische Verknüpfungen

| A | B | A\|B | A&B | ~A |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | $\neq 0$ | 1 | 0 | 1 |
| $\neq 0$ | 0 | 1 | 0 | 0 |
| $\neq 0$ | $\neq 0$ | 1 | 1 | 0 |

Bemerkungen:

- & hat eine höhere Priorität als |.
- Bis MATLAB 5 haben | und & gleiche Priorität, die Auswertung erfolgt von links nach rechts.
- (Arithmetische) Vergleichsausdrücke haben höhere Priorität als | und &.
  Beispiel: (wahr/1) `0|2>1` = `0|(2>1)` $\neq$ `(0|2)>1` (unwahr/0)
- Die Negation ~ wirkt nur auch das unmittelbar folgende Argument.
  Beispiel: (unwahr/0) `~0==2` $\neq$ `~(0==2)` (wahr/1)

# Komplexe Zahlen

- Komplexe Konstanten werden in MATLAB genauso dargestellt wie in der mathematischen Literatur üblich, nämlich als Realteil+Imaginärteil$i$.
- Maschinenintern werden sowohl für den Realteil als auch für Imaginärteil doppelt genaue reelle Zahlen verwendet.

Beispiel:

```
i=5;
a=1+i
a=1+1i;
b=-2+3i;
c=a+b
c=a*b
```

## Zeichenketten–Konstanten

- Unter einer Zeichenkette (String) versteht man eine nicht leere Folge von Zeichen.
- Zeichenketten–Konstanten müssen durch Apostrophstriche begrenzt werden.
- Als Länge einer Zeichenketten–Konstanten bezeichnet die Anzahl der Zeichen (zwischen den Apostrophstrichen).
- Einen Apostrophstrich innerhalb einer Zeichenkette erhält man durch zwei Apostrophstriche.

Beispiel:

```
a='TEXT'
b='12'
c='Dies ist ein Satz!'
d='It''s'
```

## Die Eingabe von Zeichenketten

Bisher: *Variable* = input('Text')

Dies erfordert unbedingt die Eingabe der ' bei Zeichenketten.

Besser: *Stringvariable* = input('Text','s')

Beispiel:

```
s1 = input('Bitte 1. Text eingeben: ')
s2 = input('Bitte 2. Text eingeben: ', 's')
```

erfordert

```
'text1'
text2
```

# Strukturen

- Eine Struktur (ab MATLAB 5) ist ein Objekt, das aus einer Folge von benannten Komponenten mit verschiedenen Typen besteht („eine Variable mit Komponenten“).
- Die benannten Komponenten werden durch einen Punkt von der Variablen getrennt.
- Zur Erzeugung und Bearbeitung stellt MATLAB einige Funktionen zur Verfügung (siehe unten).
- Mit Strukturen kann im allgemeinen nicht gerechnet werden.
- Direkte Eingabe (und Erzeugung) ist über die Komponenten möglich.
- Strukturen haben in MATLAB keine Namen.

Beispiel:

```
s.name='Holger Grothe';
s.adresse='Schlossgartenstr. 7, 64289 Darmstadt';
s.matrikelnummer=1234567;
s.semester=inf;
s
   -->
s =
              name: 'Holger Grothe'
           adresse: 'Schlossgartenstr. 7, 64289 Darmstadt'
    matrikelnummer: 1234567
          semester: Inf
```

# Die Programmverzweigung

- **Die `if`–Anweisung**

  Allgemeine Form:

  if *Variable/Vergleichsausdruck*
  *Anweisung*
  .
  .
  .
  *Anweisung*
  elseif *Variable/Vergleichsausdruck*
  *Anweisung*
  .
  .
  .
  *Anweisung*
  elseif *Variable/Vergleichsausdruck*
  .
  .
  .
  else
  *Anweisung*
  .
  .
  .
  *Anweisung*
  end

Häufig auftretende Formen:

if *Variable/Vergleichsausdruck*
    *Anweisung*
        .
        .
        .
    *Anweisung*
end

Funktionsweise: Falls die *Variable* ungleich Null ist bzw. der *Vergleichsausdruck* wahr ist, werden die Anweisungen zwischen `if` und `end` der Reihe nach einmal ausgeführt.

if *Variable/Vergleichsausdruck*
    *Anweisung*
        .
        .
        .
    *Anweisung*
else
    *Anweisung*
        .
        .
        .
    *Anweisung*
end

Funktionsweise: Falls die *Variable* ungleich Null ist bzw. der *Vergleichsausdruck* wahr ist, werden die Anweisungen zwischen `if` und `else` der Reihe nach einmal ausgeführt, andernfalls werden die Anweisungen zwischen `else` und `end` der Reihe nach einmal ausgeführt

Bemerkungen zur Programmverzweigung:

- Der `elseif`-Block ist optional, kann aber auch mehrfach verwendet werden.
- Zu einem `elseif`-Block gehört kein eignes `end`. Ein `else if` muß dagegen mit einem eignen `end` abgeschlossen werden.
- Der `else`-Block ist ebenfalls optional und kann höchstens einmal verwendet werden.
- Der Anweisungsblock nach dem `if` ist auch optional.
- Die `if`-Anweisungen können geschachtelt werden.

- **Die `switch`–Anweisung (ab MATLAB 5)**

  Allgemeine Form:

  switch *Variable/Ausdruck*
  case wert1
  *Anweisung*
  .
  .
  .
  *Anweisung*
  case {wert2, wert3,. . ., wertN}
  *Anweisung*
  .
  .
  .
  *Anweisung*
  case . . .
  .
  .
  .
  otherwise
  *Anweisung*
  .
  .
  .
  *Anweisung*
  end

  Funktionsweise: Falls die *Variable* einen der aufgeführten Werte hat, werden die zugehörigen Anweisungen (bis zum nächsten `case`) der Reihe nach einmal ausgeführt. Andernfalls werden die Anweisungen zwischen `otherwise` und `end` der Reihe nach einmal ausgeführt.

Bemerkungen:

- Die *Variable* muß skalar oder ein String sein.
- Anstelle einer Variablen kann auch ein Ausdruck verwendet werden.
- Der `otherwise`-Block ist optional und kann höchstens einmal verwendet werden.

Beispiel:

```
method = 'bilinear';

[...]

switch method
   case {'linear','bilinear'}
        disp('Method is linear')
   case 'cubic'
        disp('Method is cubic')
   case 'nearest'
        disp('Method is nearest')
   otherwise
         disp('Unknown method.')
end
```

# Funktionen

In MATLAB steht eine große Zahl von Funktionen (ohne zusätzliche Deklaration) zur Verfügung.

| Ausgewählte trigonometrische Funktionen | |
|---|---|
| sin | Sinus |
| sinh | Hyperbelsinus |
| asin | Arcussinus |
| asinh | Inverser Hyperbelsinus (Areasinus) |
| cos | Cosinus |
| cosh | Hyperbelcosinus |
| acos | Arcuscosinus |
| acosh | Inverser Hyperbelcosinus (Areacosinus) |
| tan | Tangens |
| tanh | Hyperbeltangens |
| atan | Arcustangens |
| atanh | Inverser Hyperbeltangens (Areatangens) |
| Exponentialfunktionen | |
| exp | Exponentialfunktion $e^x$ |
| pow2 | Exponentialfunktion $2^x$ |
| log | Natürlicher Logarithmus |
| log2 | Logarithmus zur Basis 2 (Dualer Logarithmus) |
| log10 | Logarithmus zur Basis 10 (Dekadischer Logarithmus) |
| sqrt | Quadratwurzel |
| Spezielle Funktionen zu komplexen Zahlen | |
| angle | Phasenwinkel |
| conj | Konjugiert komplexe Zahl |
| imag | Imaginärteil |
| real | Realteil |

| Sonstige Funktionen | |
|---|---|
| abs | Absolutbetrag |
| fix | Abschneiden der Nachkommastellen |
| floor | Größte ganze Zahl kleiner gleich dem Argument |
| ceil | Kleinste ganze Zahl größer gleich dem Argument |
| round | Rundung zur nächsten ganzen Zahl |
| rem | Divisionrest |
| sign | Signum–Funktion |
| gcd | Größter gemeinsamer Teiler |
| lcm | Kleinstes gemeinsames Vielfaches |
| xor | exklusives Oder (log. Funktion) |
| Strukturen (ab MATLAB 5) | |
| struct | Erzeugen einer Struktur. Beispiel: `s=struct('name', 'H. Grothe', 'mnr',1234567, 'sem',1)` |
| fieldnames | Rückgabe der Namen der Komponenten einer Struktur (als Vektor von Strings). |
| getfield | Rückgabe des Wertes einer Komponente. Entspricht `struktur.name` |
| rmfield | Entfernen einer Komponente einer Struktur. |
| setfield | Setzen des Wertes einer Komponente. Entspricht `struktur.name=wert` |

Darüberhinaus gibt es noch eine Vielzahl weiterer Standardfunktionen (z.B. `cot, acoth, gamma, besselj`,...).

# Der help–Aufruf

Der Befehl `help` bzw. `help` *Topic* zeigt (englischsprachige) Hilfetexte (zu dem gewünschten Punkt) an.

Beispiel:

```
help

[...]
matlab/elfun         -  Elementary math functions.
matlab/specfun       -  Specialized math functions.
[...]

help specfun

[...]
   expint      - Exponential integral function.
   gamma       - Gamma function.
   gcd         - Greatest common divisor.
[...]

help gamma

 GAMMA  The gamma function.
        Y = GAMMA(X) evaluates the gamma function at all
        the elements of X. X must be real.
        gamma(x) = integral from 0 to inf of t^(x-1) exp(-t) dt.
        gamma(n+1) = n! = n factorial = prod(1:n).

        See also GAMMALN, GAMMAINC.
```

## Vektoren

Für die direkte Eingabe von Vektoren gilt:

- Vektoren werden in eckige Klammern eingeschlossen
- Die einzelnen Komponenten werden durch Leerzeichen oder Komma getrennt.
- Ein so erzeugter Vektor ist ein Zeilenvektor.

Beispiel:

```
[4,5 6 7]
```

entspricht

$$\begin{pmatrix} 4 & 5 & 6 & 7 \end{pmatrix}$$

## Matrizen

Für die direkte Eingabe von Matrizen gilt:

- Matrizen werden in eckige Klammern eingeschlossen
- Die einzelnen Komponenten einer Zeile werden durch Leerzeichen oder Komma getrennt.
- Die einzelnen Zeilen werden durch Semikolon oder „Newline“ getrennt.
- Matrizen können auch durch Blöcke/Untermatrizen zusammengesetzt werden.

Beispiel:

```
[1,2,3;4,5,6;7,8,9]
[1,2,3
 4 5 6;7,8 9]
[1,2,3;[4 5;7 8],[6;9]]
```

entspricht jeweils der Matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Die Zuweisung von Matrizen/Vektoren zu Variablen erfolgt wie bei gewöhnlichen Zahlen.

Allgemeine Form: *Variable* = *Matrix/Vektor*

Beispiel:

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

# Indexausdrücke

Allgemeine Form: *ug*:*sw*:*og*

Obiger Ausdruck entspricht der Zahlenfolge (dem Vektor)

$$ug, ug + sw, ug + 2 \cdot sw, \ldots, ug + k \cdot sw \leq og$$

Beispiel:

```
1:2:10    entspricht     1 3 5 7 9
```

- Wird die Schrittweite *sw* weggelassen, d.h. wird nur *ug*:*og* angegeben, so wird als Wert für die Schrittweite 1 angenommen.
- Ist bei positiver Schrittweite die Untergrenze *ug* größer als die Obergrenze *og*, so enthält die Zahlenfolge keine Elemente
- Ist bei negativer Schrittweite die Untergrenze *ug* kleiner als die Obergrenze *og*, so enthält die Zahlenfolge keine Elemente
- Für Untergrenze, Obergrenze und Schrittweite sind auch reelle Zahlen, arithmetische Ausdrücke und sogar Matrizen/Vektoren (erstes Element wird verwandt, bis MATLAB 6 wird eine Warnung ausgegeben) zulässig.
- Indexausdrücke (der :) haben niedrigere Priorität als arithmetische Operatoren.
- Indexausdrücke haben höhere Priorität als Vergleichsoperatoren (korrekt implementiert ab MATLAB 6).
- Indexausdrücke können zur Erzeugung von Vektoren (und Matrizen) benutzt werden.
  Beispiel:

  ```
  x=1:3*2.1
  y=(1:3)*2.1
  ```

  erzeugt die Vektoren $x = (1, 2, 3, 4, 5, 6)$ und $y = (2.1, 4.2, 6.3)$.

# Elemente von Matrizen und Vektoren

- Die Indizierung der Matrixelemente/Vektorelemente erfolgt von 1 an.
- Einzelne Elemente werden durch die Angabe der entsprechenden Variablen und der Indizes eingeschlossen in runden Klammer referenziert.
  Beispiel: A(2,1) bezeichnet das Element $a_{21}$ der Matrix $A$ und $x(3)$ das Element $x_3$ des Vektors $x$.
- Untermatrizen können mit Hilfe von ganzzahligen positiven Indexausdrücken referenziert werden.
- Bei der Verwendung von Indexausdrücken ist es zulässig sowohl die Untergrenze als auch die Obergrenze wegzulassen. Es werden dann als Werte 1 bzw. die entsprechende Größe der Matrix bzw. des Vektors angenommen. Der Doppelpunkt ist aber zwingend notwendig.

Beispiele:

| | |
|---|---|
| `A(:,1)` | erste Spalte von $A$ |
| `A(2,:)` | zweite Zeile von $A$ |
| `A(2:3,1:2)` | Untermatrix $\begin{pmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$ |
| `A(1:2:6,1:2)` | Untermatrix $\begin{pmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \\ a_{51} & a_{52} \end{pmatrix}$ |

# Matrixoperationen

| Operation | Symbol | Bemerkung (A,B Matrix, s skalar) |
|---|---|---|
| Addition | + | A+B, s+A (s+$a_{ij}$) möglich |
| Subtraktion | - | A-B, s-A, A-s möglich |
| Multiplikation | * | A*B, s*A (s$\cdot a_{ij}$) möglich |
| Exponentiation | ^ | A^s möglich, A quadratisch |
| Transposition | ' | A' möglich |
| X=A/B | / | „Löst“ das Gleichungssystem X*B=A |
| X=A\B | \ | „Löst“ das Gleichungssystem A*X=B |
| Elementw. Multiplikation | .* | $a_{ij} \cdot b_{ij}$ |
| Elementw. Division | ./ | $a_{ij}/b_{ij}$ |
| Elementw. Exponentiation | .^ | $a_{ij}^{b_{ij}}$ |
| Elementw. Vergleich | == | A==B, A==s möglich |
| Elementw. Vergleich | ~= | A~=B, A~=s möglich |
| Elementw. Vergleich | < | A<B, A<s möglich |
| Elementw. Vergleich | <= | A<=B, A<=s möglich |
| Elementw. Vergleich | > | A>B, A>s möglich |
| Elementw. Vergleich | >= | A>=B, A>=s möglich |
| Elementw. Funktionsanw. | $function(\cdot)$ | Beispiel: sin(A) |

| Ausgewählte Matrix-Funktionen | |
|---|---|
| sum | Summe der Elemente bzw. Spaltensummen |
| prod | Produkt der Elemente bzw. Spaltenprodukt |
| size | Größe einer Matrix (Zeilen Spalten). |
| length | Anzahl der Elemente eines Vektors. |
| norm | Matrix- oder Vektornorm |
| rank | Rang der Matrix |
| det | Determinante |
| inv | Matrixinversion |
| zeros | Matrix mit nur 0 Elementen |
| ones | Matrix mit nur 1 Elementen |
| eye | Einheitsmatrix |
| tril | Untere Dreiecksmatrix extrahieren |
| triu | Obere Dreiecksmatrix extrahieren |
| any | Testet spaltenweise auf Nicht-Null-Elemente |

# Schleifenkonstrukte

Unter einer Zählschleife versteht man eine Programmschleife, von der **vor** der Ausführung die Anzahl der Schleifendurchläufe bekannt ist. Die Umsetzung in MATLAB geschieht wie folgt:

Allgemeine Form:

for *Variable=Indexausdruck*
    *Anweisung*
        .
        .
        .
    *Anweisung*
end

Übliche Form:

for *Variable=ug:og*
    *Anweisung*
        .
        .
        .
    *Anweisung*
end

**Funktionsweise:**
Die (Lauf-)variable wird zunächst auf den Anfangswert *ug* gesetzt und die Anzahl der Schleifendurchläufe bestimmt. Ist diese Zahl grösser als 0 werden die Anweisungen in der Schleife ausgeführt. Vor jedem weiteren Schleifendurchlauf wird die Schrittweite zur Laufvariable dazu addiert.

**Bemerkungen:**

- Als Laufvariablen sind nur einfache Variablen zugelassen (also z.B. keine Matrizen bzw. Matrixelemente).
- Für die Untergrenze, die Obergrenze und die Schrittweite sind auch arith. Ausdrücke zulässig.
- Die Angabe der Schrittweite kann entfallen. Es wird dann automatisch der Wert 1 als Schrittweite verwendet.
- Ist die Anzahl der Schleifendurchläufe 0, so werden die Anweisungen in der Schleife nicht ausgeführt.
- Werden zwei Schleifen miteinander verschachtelt, so liegt die innere Schleife komplett innerhalb der äußeren Schleife.

Beispiel:

```
for k = 1:10
    vec(k) = k*3.1415
end
xvec = (1:10)*3.1415
```

Beispiel:

```
for k = 1:5
    k
    for n = 1:3
        n
        a(k,n) = input('a(k,n): ')
    end
end
```

# Höherdimensionale Felder (ab MATLAB 5)

- Felder sind nicht auf Vektoren und Matrizen beschränkt, sondern können beliebig groß sein.
- Zur Erzeugung können
  1. (erweiterte) Standardfunktionen wie `zeros` oder `ones`
  2. Indexerweiterungen
  3. die Funktionen `cat` und `repmat`

verwendet werden.

Beispiel:

```
A=[1 2 3;4 5 6];
B=[6 2 0;9 1 3];
D(:,:,1)=A;
D(:,:,2)=B;
E=ones(2,3,2);
```

Die `cat`-Funktion:

`cat(dim,A1,A2,A3,...)`

- `cat` fügt die Felder *A1, A2, A3, ...* in Dimension *dim* zusammen.
- Die Felder *A1, A2, A3, ...* müssen gleichgroß sein.
- `cat(1,A1,A2)` entspricht `[A1,A2]`
- `cat(2,A1,A2)` entspricht `[A1;A2]`

Beispiel:

```
D=cat(3,A,B)
   -->
D(:,:,1) =
     1     2     3
     4     5     6
D(:,:,2) =
     6     2     0
     9     1     3
```

Die `repmat`-Funktion:

```
repmat(A,[n1,n2,n3,...])
```

- `repmat` baut ein $n1 \times n2 \times n3 \times \ldots$ „Blockfeld“ mit dem Feld A als Element auf.
- Der Aufruf `repmat(A,n)` ist zulässig und erzeugt ein $n \times n$ „Blockfeld“.
- Der Aufruf `repmat(A,n,m)` ist zulässig und erzeugt ein $n \times m$ „Blockfeld“.

Beispiel:

```
a=[1 2;3 4];
b=repmat(a,[1,3,2])
   -->
b(:,:,1) =
     1     2     1     2     1     2
     3     4     3     4     3     4
b(:,:,2) =
     1     2     1     2     1     2
     3     4     3     4     3     4
```

Bemerkung: Ab MATLAB 5 sind auch skalare Werte bei der Zuweisung an Untermatrizen zulässig.

Beispiel:

```
a = zeros(3,5);
a(2:3,3:5) = 5*ones(2,3);
a(2:3,3:5) = 5;
```

## Felder mit beliebigen Daten (Zellenfelder, Listen)

- Neben den „numerischen“ Feldern (z.B. Vektoren und Matrizen) gibt es auch sog. Zellenfelder, die beliebige Daten enthalten können.
- Die direkte Eingabe solcher Felder (ein– oder zweidimensional) funktioniert analog zu der direkte Eingabe von Vektoren oder Matrizen. Anstelle der `[]` werden {} verwendet.
- Die Referenzierung einzelner Elemente erfolgt ebenfalls analog zu „numerischen“ Feldern. Werden `()` verwendet, so ist das Resultat wieder ein Zellenfeld. Werden dagegen {} verwendet, so erhält man als Resultat(e) einen Wert des entsprechenden Typs der Zelle(n).
- Zur Bearbeitung stellt MATLAB einige Funktionen zur Verfügung (siehe unten).
- Mit Zellenfelder kann im allgemeinen nicht gerechnet werden.

Beispiel:

```
a={'text' 1.5; eye(2), 1+1i}
   -->
a =
    'text'           [         1.5000]
    [2x2 double]     [1.0000+ 1.0000i]
```

```
a{2,1}
   -->
ans =
     1      0
     0      1
```

## Funktionen für höherdimensionale Felder und Zellenfelder

| Name | Beschreibung |
|---|---|
| Höherdimensionale Felder | |
| flipdim<br>shiftdim<br>permute | Vertauschungsoperationen. |
| ndims | Anzahl der Felddimensionen. |
| reshape | Umordnen von Feldern. Beispiel: `reshape(A,[1,6])` |
| squeeze | Entfernen von einelementigen Felddimensionen. Beispiel: `squeeze(ones(1,1,3))`. |
| Zellenfelder | |
| cell | Erzeugen eines leeren Zellenfeldes. Beispiel: `cell(3,4,2)` |
| cell2struct | Konvertierung eines Zellenfeldes in eine Struktur. |
| celldisp | Elementweise Anzeige der Zellenfelder. |
| cellplot | Graphische Strukturanzeige eines Zellenfeldes. |
| num2cell | Konvertierung eines („numerischen“) Feldes in ein Zellenfeld. |
| struct2cell | Konvertierung einer Struktur in ein Zellenfeld/eine Liste. |

# Formatierte Ausgabe

Bisher:

```
Variablenliste
```

Ausgabe mit Formatierung:

```
fprintf(1, 'Formatliste', E/A-Liste)
```

Eine Formatliste besteht aus Formatelementen (Formatbeschreibern). Man unterscheidet zwischen gewöhnlichen Zeichen (Steuerungs–Beschreiber), die in die Ausgabe kopiert werden und Umwandlungsangaben (Daten–Beschreibern). Als Elemente der E/A–Liste können Variablennamen, Feldnamen und auch Ausdrücke (z.B. arith. Ausdrücke) auftreten.

- Jede Umwandlungsangabe beginnt mit dem Zeichen % und endet mit einem Umwandlungszeichen.
- Zwischen dem % und dem Umwandlungszeichen kann folgendes angegeben werden
  - Ein Minuszeichen zur linksbündigen Ausrichtung.
  - Eine Zahl, die die minimale Ausgabebreite festlegt. Ggf. wird mit Leerzeichen aufgefüllt.
  - Ein Punkt, der die Ausgabebreite von der Genauigkeit trennt.
  - Eine Zahl, die Genauigkeit, die die maximale Anzahl von Zeichen festlegt (Zeichen eines Strings, Ziffern nach dem Dezimalpunkt, minimale Anzahl von Ziffern bei ganzen Zahlen.
  - Eines der Zeichen +, #, b, h, l, t.

| Form | Bedeutung |
|---|---|
| | **Ausgabe von ganzen Zahlen** |
| `d, i` | Die ganze Zahl wird dezimal rechtsbündig in das Ausgabefeld geschrieben. Die Angabe %*w.m*i bzw. %*w.m*d bewirkt das mindestens *m* Ziffern, ggf. führende Nullen, ausgegeben werden. %*w*.0i bzw. %*w*.0d bewirkt für den Ausgabewert Null die Ausgabe von *w* Leerzeichen. |
| `o` | Die positive ganze Zahl wird als Oktalzahl ohne Vorzeichen ausgegeben. Die Angabe %#o bewirkt die Ausgabe einer führenden Null. (→ `d, i`). |
| `x, X` | Die positive ganze Zahl wird als Hexadezimalzahl ohne Vorzeichen mit abcdef bzw. ABCDEF für 10,...,15 ausgegeben. Die Angabe %#x bzw. %#X bewirkt die Ausgabe eines führenden 0x bzw. 0X (→ `d, i`). |
| `u` | Die positive ganze Zahl wird als Dezimalzahl ausgegeben. |
| | **Ausgabe von Gleitkommazahlen** |
| `f` | in der Festpunktdarstellung<br>D.h. die Wert wird in der Festpunktdarstellung mit d Nachkommastellen (Voreinstellung: 6) in ein Ausgabefeld der Feldweite w geschrieben. |
| `e, E` | in der normalisierten Gleitpunktdarstellung<br>D.h. der Exponent wird so gewählt, daß die Mantisse zwischen 1.0 und 10.0 liegt. |
| `g, G` | In Abhängigkeit von der Größe des Wertes erfolgt die Ausgabe im „f“– oder „e“–Format. %e bzw. %E wird verwendet, wenn der Exponent kleiner als -4 oder größer gleich der Genauigkeit ist. Null und Dezimalpunkt am Schluß werden nicht ausgegeben. Die Anzahl der signikanten Stellen entspricht der Genauigkeit. |

| | |
|---|---|
| `o, x, X, u` | Mit der Zusatzangabe b (doppeltgenaue Zahl) und t (einfachgenaue Zahl) erfolgt die Ausgabe in oktaler, hexadezimaler oder dezimaler Form als ganze positive Zahl. |
| | **Ausgabe von Strings** |
| `s` | Ausgabe des Strings bzw. Ausgabe so vieler Zeichen, wie es die Genauigkeit verlangt. |
| `c` | Ausgabe eines Zeichens. |

## Beispiele:

| interner Wert | Formatangabe | Ausgabefeld |
|---|---|---|
| 12 | %4i | `  12` |
| 12 | %-4i | `12  ` |
| -12 | %4i | ` -12` |
| -12 | %2i | `-12` |
| 12 | %6.4i | `  0012` |
| -12 | %6.4i | ` -0012` |
| 12 | %o | `14` |
| 12 | %X | `C` |
| 12 | %#5x | `  0xC` |
| 12 | %#6.1o | `   014` |
| 0 | %3i | `  0` |
| 0 | %2.0i | |
| -12.3456 | %9.3f | `  -12.346` |
| -12.3456 | %9.3e | `-1.235e+01` |
| -12.3456 | %9.3g | `    -12.3` |
| -1234.56 | %9.3g | `-1.23e+03` |
| 'Text' | %s | `Text` |
| 'Text' | %2s | `Text` |
| 'Text' | %6s | `  Text` |
| 'Text' | %6.2s | `    Te` |

## Steuerungs–Beschreiber

| Form | Bedeutung |
|---|---|
| `\n` | <Newline>. Neue Zeile |
| `\r` | <Carriage return>. Positioniere am Beginn der Zeile (Wagenrücklauf) |
| `\b` | <Backspace>. Positioniere um eine Stelle nach links. |
| `\t` | <Tabulator>. Positioniere auf den nächsten Tabulator. |
| `\'` | ' |
| `\\` | \ |
| `\`*octal* | Zeichen mit dem Code der 1–, 2–, 3–, 4–stelligen Oktalzahl *octal*, die mit einer 0 beginnen muß. |
| `\a` | Alarm (Beep). |
| `\c, \f, \v` | Sonstige. |
| `'c1...cn'` | Auszugebender String. |

# Der Zusammenhang zwischen Formatliste und E/A–Liste

- Die E/A–Liste wird von links nach rechts unter Kontrolle der Liste der Formatelemente abgearbeitet.
- Die Liste der Formatelemente wird ebenfalls von links nach rechts abgearbeitet. Dabei wird jedem Element der E/A–Liste das in der Reihenfolge zugehörige Formatelement (nur Datenbeschreiber). zugeordnet. Steuerbeschreiber haben kein zugehöriges Element in der E/A–Liste. Das Formatelement steuert dabei die Datenaufbereitung.
- Sind weniger Elemente der E/A–Liste als Formatelemente vorhanden, so wird die Abarbeitung beim ersten Formatelement, zu dem kein Element der E/A-Liste vorhanden ist, abgebrochen.
- Sind mehr Elemente der E/A–Liste als Formatelemente zu ihrer Aufbereitung vorhanden, so wird (normalerweise) die Abarbeitung mit dem ersten Formatelement neu gestartet.

# Stringfunktionen

| Name | Argumente | Beschreibung |
|---|---|---|
| abs | S | Wandelt String in die entsprechenden numerischen Werte um (Zeichensatztabelle). |
| setstr | V | Wandelt Vektor von ganzzahligen numerischen Werten (0–255) in einen String um. |
| isstr | bel. | Liefert den Wert 1, falls das Argument ein String ist, andernfalls 0. |
| blanks | I | Liefert ein String, der aus $n$ Leerzeichen besteht. |
| deblank | S | Entfernt alle Nullzeichen (num. Wert 0) und alle Leerzeichen am Ende. |
| str2mat | S,S,S,... | Bildet eine Text–Matrix aus den Argument–Strings (max. 10). Die Argument–Strings werden mit Leerzeichen auf gleiche Länge gebracht. |
| eval | S | Führt einen String mit einem MATLAB–Ausdruck aus. |
| strcmp* | S,S | Vergleicht zwei Strings und liefert 1 bei Gleichheit und 0 sonst. |
| findstr | S,S | Gibt den Vektor der Indizes des Vorkommens des kürzeren Strings im längeren String aus. |
| upper | S | Wandelt alle Buchstaben des Strings in Großbuchstaben um. |
| lower | S | Wandelt alle Buchstaben des Strings in Kleinbuchstaben um. |
| isletter | S | Liefert einen Vektor aus 0 und 1 zurück. Dabei steht an der $n$-ten Stelle eine 1, falls das $n$-te Zeichen des Strings ein Buchstabe ist, ansonsten steht dort eine 0. |
| isspace | S | Analog zu isletter. Eine 1 steht für Leerzeichen, Tabulator, <Newline>, <Carriage Return> und <Formfeed>. |

| | | |
|---|---|---|
| strrep | S,S,S | Ersetzt jedes Vorkommen des zweiten Strings im ersten String mit dem dritten String. |
| strtok | S oder S,V | Gibt den Teilstring zurück, der vor dem ersten Leerzeichen bzw. vor einen der ersten Trennzeichen (zwei Argumente) steht. |
| num2str | R | Wandelt eine Zahl (Ausdruck) in einen String um. |
| int2str | I | Wandelt eine ganze Zahl (ganzzahligen Ausdruck) in einen String um. |
| str2num | S | Wandelt String in eine Zahl um (sofern möglich). |
| mat2str | M oder M,I | Wandelt eine Matrix in einen String um unter Verwendung der vollen Genauigkeit bzw. unter Verwendung von $n$ Stellen Genauigkeit. |
| sprintf | S,... | Wandelt Zahlen unter einer Formatkontrolle in einen String um ($\longrightarrow$ fprintf). |
| sscanf | S,S,... | Wandelt einen String unter einer Formatkontrolle in Zahlen um. |
| hex2num | S | Wandelt einen hexadezimalen String in eine IEEE Gleitkommazahl um. |
| hex2dec | S | Wandelt einen hexadezimalen String in eine Dezimalzahl um. |
| dec2hex | I | Wandelt eine Dezimalzahl in einen hexadezimalen String um. |
| Stringfunktionen (ab MATLAB 5) | | |
| base2dec | S,I | Konvertiert eine Zahl zur Basis `B` in eine Dezimalzahl. |
| bin2dec | S | Konvertiert eine Binärzahl in eine Dezimalzahl. |
| dec2base | I,I | Konvertiert eine Dezimalzahl in eine Zahl zur Basis `B`. |
| dec2bin | I | Konvertiert eine Dezimalzahl in eine Binärzahl. |
| strcat | S,S | Aneinanderhängen von Strings. |
| strvcat | S,S | „Vertikales“ Aneinanderhängen von Strings. |

Dabei bedeuten:

| | | |
|---|---|---|
| S | : | String (Zeichenkette). |
| V | : | Vektor aus ganzen Zahlen zwischen 0 und 255. |
| I | : | Integer (ganze Zahl). |
| R | : | Gleitkommazahl. |
| M | : | Matrix. |

* Vergleiche mit den üblichen Operatoren wie z.B. $<$ sind möglich, allerdings müssen die Strings gleiche Länge haben. Der Vergleich erfolgt zeichenweise. Als Resultat erhält meinen ein Vektor aus 0 und 1.

Bemerkung: Das Aneinanderhängen von Strings funktioniert mit Hilfe der oben erwähnten Funktionen oder analog zum Aneinanderhängen von Vektoren.

Beispiel:

```
s = 'Hello';
s = [s, ' World'];

d = 'Hallo';
d = strcat(d, ' Welt');
```

# Funktionsunterprogramme

Bisher: Standardfunktionen

Neben den Standardfunktionen gibt es noch externe Funktionen als weitere Form der Funktionsunterprogramme.

# Externe Funktionen

Diese Art von Funktionsunterprogrammen werden in selbstständigen Programmeinheiten (in einer separaten Datei) definiert.

function [*Formale Rückgabeparam.*] = *name*(*Formale Aufrufparam.*)
*Kommentarzeilen*
*Beliebige Anweisungen*
*Rückgabeparameter=Ausdruck*

Dabei gilt:

- Formale Rückgabeparameter:
  Einfache Variablen, Stringvariablen, Vektoren, Matrizen.
  Gibt es nur einen Rückgabeparameter, so können die eckigen Klammern weggelassen werden. Die Liste der Rückgabeparameter darf auch leer sein (`[] =`). In diesem Fall kann neben den eckigen Klammern auch das Gleichheitszeichen weggelassen werden.
- Formale Aufrufparameter:
  Einfache Variablen, Stringvariablen, Vektoren, Matrizen.
- Name:
  (Symbolischer) Name der Funktion. Dieser Name sollte mit dem Namen der Datei, in der die Funktion vereinbart ist, übereinstimmen.

- Kommentarzeilen:
  Diese Kommentarzeilen werden bei der `help`-Anweisung ausgegeben. Sie sollten eine Beschreibung der Funktion und der Parameter enthalten.

Der Aufruf eines Funktionsunterprogramms geschieht wie folgt:

$$\textit{(datei)name}(\textit{Aktuelle Aufrufparameter})$$

bzw.

$$[\textit{Aktuelle Rückgabeparam.}] = \textit{(datei)name}(\textit{Aktuelle Aufrufparam.})$$

Ein Funktionsaufruf bewirkt folgende Aktionen:

- Aktuelle Aufrufparameter, die in Form von Ausdrücken angegeben sind, werden berechnet.
- Die aktuellen Aufrufparameter werden den zugehörigen formalen Aufrufparametern zugeordnet.
- Die aktuellen Rückgabeparameter werden den zugehörigen formalen Rückgabeparametern zugeordnet.
- Die ausführbaren Anweisungen des Funktionsunterprogramms werden durchlaufen.
- Es folgt ein Rücksprung in die aufrufende Programmheit.

Folgende Regeln sind zu beachten:

- Die `function`-Zeile muß die erste (MATLAB-)Zeile in der Datei sein.
- Der entscheidende Name für den Aufruf ist der Name der Datei (*name*.m), nicht der Name in der `function`-Zeile.
- Die aktuellen Aufrufparameter müssen normalerweise in Anzahl, Reihenfolge, Art, und Typ mit den formalen Aufrufparametern übereinstimmen.
- Die aktuellen Aufrufparameter dürfen auch Ausdrücke/Konstanten des entsprechenden Typs sein.
- Der Datentyp (sofern eindeutig festgelegt) der formalen Aufrufparameter und der formalen Rückgabeparameter wird letztendlich durch die Anweisungen in der externen Funktion festgelegt.
- Bei der Parameterübergabe findet keine Datentypkonvertierung statt.
- Die Änderung formaler Aufrufparameter innerhalb der Funktion hat **keinerlei** Auswirkung auf die entsprechenden aktuellen Aufrufparameter.
- Variablen, die als aktueller Aufrufparameter verwendet werden, dürfen gleichzeitig auch als aktueller Rückgabeparameter verwendet werden.
- Die in dem Funktionsunterprogramm verwendeten Namen für lokale Größen haben nur dort eine (lokale) Bedeutung, d.h. die gleichen Namen dürfen auch in anderen Programmeinheiten verwendet werden.
- Expliziter bzw. impliziter rekursiver Aufruf ist gestattet.

Beispiel:
Datei `prog1.m`

```
x=input('Bitte x eingeben: ');
format long;
arctan(x)
atan(x)
```

Datei `arctan.m`

```
function fy = arctan(x)
%arctan   Inverse tangent.
%         arctan(x) is the arctangent of x.
%         See also atan.
%
epsilon = 1e-8;
y = x^2;
n = 0;
s = x;
b = -x * y;
a = b / (n+n+3);
while abs(a) / (1-y) > epsilon
  s = s + a;
  n = n + 1;
  b = -b * y;
  a = b / (n+n+3);
end
fy = s;
```

Beispiel:
Datei `sinsqr.m`

```
function y = sinsqr(x)
%sinsqr   Square of sine
%         sinsqr(x) is the square of sine of the elements of x.
%         See also sin.
%
y = sin(x).*sin(x);
```

Datei `sinquad.m`

```
function y = sinquad(x)
%sinquad  Sinusquadrat
%         sinquad(x) ist das Quadrat des Sinus von x.
%         See also sin.
%
y = sin(x)*sin(x);
```

Mögliche Aufrufe:

```
sinsqr(0.1)
sinsqr(0:0.1:1)
sinsqr(ones(2))
sinsqr('grothe')
sinquad(0.1)
```

Fehlerhafte Aufrufe:

```
sinquad(0:0.1:1)   --> Zeilenvektor * Zeilenvektor
sinquad(ones(2))   --> Falsches Ergebnis
sinquad('grothe')  --> Zeilenvektor * Zeilenvektor
```

# Variable Parameteranzahl

- Die Anzahl der aktuellen Aufrufparameter und der aktuellen Rückgabeparameter kann jeweils geringer sein als die Anzahl der formalen Aufrufparameter bzw. der formalen Rückgabeparameter.
- Die Anzahl der aktuellen Aufrufparameter steht innerhalb einer Funktion in der Variablen `nargin` zur Verfügung.
- Die Anzahl der aktuellen Rückgabeparameter steht innerhalb einer Funktion in der Variablen `nargout` zur Verfügung.
- Es können immer nur die letzten Elemente der jeweiligen Parameterliste weggelassen werden.

Beispiel:
Datei `arctan.m`

```
function z = arctan(x, y)
%arctan   Inverse tangent.
%         arctan(x) is atan(x).
%         arctan(x,y) is atan2(x,y).
%         See also atan, atan2.
%
if nargin == 1
   z = atan(x);
else
   z = atan2(x, y);
end
```

Mögliche Aufrufe:

```
arctan(0.75)
arctan(3,4)
```

Beispiel:
Datei `plusminus.m`

```
function [y, z] = plusminus(x)
%plusminus   Return x+1 and x-1.
%
y = x + 1;
if nargout == 2
   z = x - 1;
end
```

Datei `pm.m`

```
function [y, z] = pm(x)
%pm   Return x+1 and x-1.
%
y = x + 1;
z = x - 1;
```

Mögliche Aufrufe:

```
[u, v] = plusminus(4);
x = plusminus(2);
plusminus(x);
[r, s] = pm(4);
y = pm(2);
```

# Variable Argumentanzahl in Funktionen

- Der letzte formale Aufrufparameter einer Funktion kann das Schlüsselwort `varargin` sein (ab MATLAB 5).
- Der letzte formale Rückgabeparameter einer Funktion kann das Schlüsselwort `varargout` sein (ab MATLAB 5).
- `varargin` ist ein (eindimensionales) Zellenfeld/eine Liste.
- `varargout` ist ein (eindimensionales) Zellenfeld/eine Liste.

Beispiel:

```
function a = mymax(varargin)
%mymax    max. value.
a = varargin{1};
for i = 2:length(varargin)
    a = max(a, varargin{i});
end
```

Möglicher Aufruf:

```
mymax(3,5,6,4)  -->  6
```

Beispiel:

```
function [s,varargout] = mysize(x)
%mysize  size function with additional output
nout = max(nargout,1)-1;
s = size(x);
for i=1:nout
    varargout(i) = {s(i)};
end
```

Möglicher Aufruf:

```
[s,rows,cols] = mysize(ones(4,5));
  -->
s = [4 5], rows = 4, cols = 5
```

# Globale Variablen

- Globale Variablen werden mit Hilfe des Schlüsselwortes `global` gefolgt von einer Variablenliste vereinbart.
- Globale Variablen müssen in allen Programmeinheiten, in denen sie verwendet werden sollen, separat vereinbart sein.
- Globale Variablen haben Vorrang vor formalen Aufrufparametern gleichen Namens.

Beispiel:
Datei `pmd.m`

```
function [y, z] = pmd(x)
%pmd   Return x+delta and x-delta.
%
global delta;
y = x + delta;
z = x - delta;
x = x - 1;
delta = delta - 1;
```

Datei `prog.m`

```
global delta;
delta = 3.5;
y = 7;
[u, v] = pmd(y);
u,v,y,delta
```

# Die fopen–Anweisung

Die fopen–Anweisung dient dazu, externen Dateien Dateinummern zuzuordnen und gleichzeitig Zugriffseigenschaften festzulegen sowie neue Dateien anzulegen.

```
fid = fopen('Dateiname')
fid = fopen('Dateiname', 'Zugriffsart')
[fid, meldung] = fopen('Dateiname', 'Zugriffsart', 'Architektur')
fids = fopen('all')
[dateiname, zugriffsart, architektur] = fopen(fid)
```

- *Dateiname* bezeichnet den Namen der Datei.
- *Zugriffsart* kann folgendermaßen aussehen:

  | | |
  |---|---|
  | `'r'` | Die Datei wird zum Lesen geöffnet. Dies ist auch die Voreinstellung, wenn keine Angabe der *Zugriffsart* erfolgt. |
  | `'r+'` | Die Datei wird zum Lesen und Schreiben geöffnet. |
  | `'w'` | Der Inhalt einer bereits existierenden Datei wird gelöscht bzw. eine neue Datei wird angelegt und zum Schreiben geöffnet. |
  | `'w+'` | Wie `'w'` + Lesen. |
  | `'a'` | Eine bereits existierende Datei wird zum Schreiben derart geöffnet, daß an das Dateiende angefügt wird bzw. eine neue Datei wird angelegt und zum Schreiben geöffnet. |
  | `'a+'` | Wie `'a'` + Lesen. |

- Ist `fopen` erfolgreich verlaufen, so steht in der Variablen fid die Dateinummer (Kanalnummer, file identifier). Dies ist eine ganze Zahl größer 2. Die Stringvariable `meldung` ist leer.

- Ist `fopen` nicht erfolgreich verlaufen, so enthält die Variable `fid` den Wert -1 und in der Stringvariablen `meldung` steht eine Fehlermeldung.
- Das optionale Argument *Architektur* dient dazu, binäre Daten zwischen verschiedenen Rechnern auszutauschen und kann Werte wie `'native', 'vaxd', 'cray'`, etc. enthalten.
- fopen('all') gibt einer Vektor mit den Dateinummern aller aktuell geöffneten Dateien zurück.
- `[dateiname, zugriffsart, architektur] = fopen(fid)` dient zur Abfrage von Eigenschaften einer Datei mittels einer Dateinummer. Ist die Dateinummer ungültig bzw. nicht vergeben, so werden Leerstrings zurückgegben.
- Die Dateinummern 0 (Standardeingabe), 1 (Standardausgabe) und 2 (Standardfehlerkanal) stehen ohne spezielle Anweisung zur Verfügung

## Die fclose–Anweisung

Die fclose–Anweisung dient dazu, eine oder mehrere geöffnete Dateien zu schließen. Insbesondere wird dabei die Zuordnung der Datei(en) zur Dateinummer aufgehoben.

```
status = fclose(fid)
status = fclose('all')
```

- Ist die fclose–Anweisung erfolgreich verlaufen, so wird 0 zurückgegeben, andernfalls -1.

# Die fread–Anweisung

Die fread–Anweisung liest binäre Daten aus einer Datei.

```
A = fread(fid)
[A, count] = fread(fid, size, 'Genauigkeit')
```

- Die fread–Anweisung liest binäre Daten in eine Matrix ein.
- In der Variablen `count` wird die Anzahl der erfolgreich gelesen Elemente zurückgegeben.
- Das Argument `size` ist optional und kann folgende Werte annehmen:

| | |
|---|---|
| `n` | Es werden n Elemente in einen Spaltenvektor gelesen. |
| `inf` | Es werden soviele Elemente in einen Spaltenvektor eingelesen wie in der Datei enthalten sind. Dies ist die Voreinstellung. |
| `[m,n]` | Es wird spaltenweise eine $m \times n$–Matrix eingelesen. |

- Sind in der Datei weniger Elemente vorhanden als angefordert, so wird die Matrix mit 0 aufgefüllt.
- Das optionale Argument `Genauigkeit` legt den Datentyp der Elemente fest. Folgende Werte sind zulässig:

| | |
|---|---|
| `'char'` | Ein Byte/Zeichen. |
| `'schar'` | Ein Byte/Zeichen im Wertebereich -127, ..., 127. |
| `'uchar'` | Ein Byte/Zeichen im Wertebereich 0, ..., 255. |
| `'short'` | Ganze Zahl (2 Byte). |
| `'ushort'` | Positve ganze Zahl (2 Byte). |
| `'int'` | Ganze Zahl (2 oder 4 Byte). |
| `'uint'` | Positve ganze Zahl (2 oder 4 Byte). |
| `'long'` | Ganze Zahl (4 Byte). |
| `'ulong'` | Positve ganze Zahl (4 Byte). |
| `'float'` | Reelle Zahl in einfacher Genauigkeit (4 Byte). |
| `'float32'` | Reelle Zahl in einfacher Genauigkeit, maschinenunabhängig (4 Byte). |
| `'double'` | Reelle Zahl in doppelter Genauigkeit (8 Byte). |
| `'float64'` | Reelle Zahl in doppelter Genauigkeit, maschinenunabhängig (8 Byte). |
| `'intN'` | Ganze Zahl (N Bit breit). |
| `'uintN'` | Positve ganze Zahl (N Bit breit). |

## Die fwrite–Anweisung

Die fwrite–Anweisung schreibt die Elemente einer Matrix binär in eine Datei.

```
count = fwrite(fid, A, 'Genauigkeit')
```

- Die Daten werden spaltenweise geschrieben.
- In der Variablen `count` wird die Anzahl der erfolgreich geschriebenen Elemente zurückgegeben.
- Das optionale Argument `Genauigkeit` legt den Datentyp fest. (s. fread).

# Die fscanf–Anweisung

Die fscanf–Anweisung liest Daten im Textformat aus einer Datei.

```
[A, count] = fscanf(fid, 'Format', size)
[A, count] = fscanf(fid, 'Format')
```

- Die fscanf–Anweisung liest Daten aus einer Textdatei in eine Matrix ein.
- In der Variablen `count` wird die Anzahl der erfolgreich gelesen Elemente zurückgegeben.
- Das Argument `size` ist optional. Mögliche Werte: s. `fread`.
- *Format* legt das Format der Daten fest, die eingelesen werden sollen (s. auch `fprintf`). Der Formatstring kann folgendes enthalten:

| | |
|---|---|
| `%d` | Ganze Zahl in Dezimaldarstellung. |
| `%i` | Wie `%d`. Es werden zusätzlich auch Octalzahlen (führende 0) und Hexadezimalzahlen (führendes 0x bzw. 0X) akzeptiert. |
| `%u` | Positive ganze Zahl in Dezimaldarstellung. |
| `%o` | Positive ganze Zahl in Oktaldarstellung. |
| `%x`, `%X` | Positive ganze Zahl in Hexadezimaldarstellung. |
| `%e`, `%E` | Gleitkommazahl. |
| `%f` | Gleitkommazahl. |
| `%g`, `%G` | Gleitkommazahl. |
| `%s` | Zeichenkette (ohne Leerzeichen, Tabulatorzeichen, Newline-Zeichen und Wagenrücklaufzeichen). |
| `%c` | Ein Zeichen. |

Zwischen dem % und dem Umwandlungszeichen kann folgendes angegeben werden:

  - Ein * für ein Element, das nicht gespeichert werden soll.

  - Eine Zahl, die die maximale Feldbreite festlegt.
  - Eines der Zeichen h oder l.

- Leerzeichen, Tabulatorzeichen, Newline-Zeichen und Wagenrücklaufzeichen haben bei der Trennung der Eingabefelder Vorrang vor der Angabe der Feldbreite.
- Der Einlesevorgang bricht vorzeitig ab, wenn ein Formatelement nicht zu den gelesenen Daten paßt.

## Die fgetl/fgets–Anweisung

Die fgetl/fgets–Anweisung liest eine Zeile aus einer Datei.

```
line = fgetl(fid)
line = fgets(fid)
```

- Bei Verwendung der fgets–Anweisung enthält der Rückgabestring zusätzlich zum Inhalt der Zeile auch das Newline-Zeichen.
- Im Fehlerfalle (Dateiende) wird -1 zurückgegeben.

## Die fprintf–Anweisung

Die fprintf–Anweisung schreibt formatiert in eine Datei.

```
count = fprintf(fid, 'Format', E/A-Liste)
fprintf('Format', E/A-Liste)
```

- Details s. vierte Vorlesung
- In der Variablen `count` wird die Anzahl der erfolgreich geschriebenen Bytes zurückgegeben.

## Die ferror–Anweisung

Die ferror–Anweisung fragt Fehler bei der Ein– und Ausgabe ab.

```
ferror(fid)
[meldung, fehlernr] = ferror(fid, 'clear')
```

- Die ferror–Anweisung fragt Fehlermeldung und Fehlernummer der letzte Eingabe/Ausgabe–Operation auf die entsprechende Datei ab.
- Ist bei der letzten Eingabe/Ausgabe–Operation kein Fehler aufgetreten, so ist die Stringvariable `meldung` leer und die Variable `fehlernr` enthält den Wert 0. Andernfalls enthalten die beiden Variablen Fehlermeldung und die (interne) Nummer des Fehlers.
- Die Angabe von `'clear'` ist optional und bewirkt das Löschen der Fehlermeldung und der Fehlernummer.

## Die feof–Anweisung

Die feof–Anweisung fragt das Dateiende ab.

```
feof(fid)
```

- Ist das Dateiende erreicht wird der Wert 1 andernfalls der Wert 0 zurückgeben.

## Die fseek–Anweisung

Die fseek–Anweisung setzt die Dateiposition.

```
status = fseek(fid, offset, 'Startpunkt')
```

- War die Positionieraktion erfolgreich, enthält die Variable `status` den Wert 0 andernfalls den Wert -1.
- Die Dateiposition wird um `offset`-viele Bytes in Richtung Dateiende (positver Offsetwert) bzw. in Richtung Dateianfang (negativer Offsetwert) verschoben.
- *Startpunkt* legt den Bezugspunkt fest. Folgende Angaben sind möglich:

| | | |
|---|---|---|
| `'bof'` | -1 | Positionierung bzgl. des Dateianfangs. |
| `'cof'` | 0 | Positionierung bzgl. der aktuellen Dateiposition. |
| `'eof'` | 1 | Positionierung bzgl. des Dateiendes. |

## Die frewind–Anweisung

Die frewind–Anweisung positioniert auf den Dateianfang.

```
frewind(fid)
```

## Die ftell–Anweisung

Die ftell–Anweisung fragt die aktuelle Dateiposition ab.

```
position = ftell(fid)
```

- War die Abfrage erfolgreich, enthält die Variable `position` den Wert die aktuelle Dateiposition, gezählt in Bytes vom Dateianfang, andernfalls den Wert -1.

## Die load–Anweisung

Die load–Anweisung liest (MATLAB–)Variablen aus einer Datei.

```
load
load dateiname
load dateiname.extension
```

- `load` lädt alle Variablen, die in der Datei `matlab.mat` gespeichert sind.
- `load dateiname` lädt alle Variablen, die in der Datei mit dem Namen `dateiname.mat` gespeichert sind.

- `load dateiname.extension` liest aus der Datei mit dem Namen `dateiname.extension`. Diese Datei kann eine ASCII-Datei sein, die in $m$ Zeilen jeweils $n$ Werte enthält. Das Resultat des Einlesevorgangs ist eine $m \times n$–Matrix, die in der Variablen `dateiname` gespeichert ist.

## Die save–Anweisung

Die save–Anweisung speichert Variablen in einer Datei.

```
save
save dateiname
save dateiname variablen
save dateiname variablen optionen
```

- `save` speichert alle (Arbeits–)Variablen in die Datei `matlab.mat`.
- `save dateiname` speichert alle (Arbeits–)Variablen in die Datei `dateiname.mat`.
- `save dateiname variablen` speichert alle angegebenen Variablen in die Datei `dateiname.mat`.
- Die Angabe einer Option bewirkt die Speicherung im Textformat (ASCII). Als Optionen sind zulässig:

| | |
|---|---|
| `-ascii` | Ausgabe auf 8 Stellen. |
| `-double` | Ausgabe auf 16 Stellen. |
| `-tab` | Trennung der einzelnen Werte durch Tabulatorzeichen anstelle von Leerzeichen. |

## Objekte (Objekt-orientierte Programmierung)

Ab MATLAB 5 ist Objekt-orientierte Programmierung möglich (siehe `help datatypes` und `help class`). Im Rahmen dieses Kurses wird aber nicht weiter darauf eingegangen.

## Sonstige Befehle und Funktionen

| Name | Beschreibung |
|---|---|
| | Information |
| lookfor | Sucht den angegebenen Text in den jeweils ersten Zeilen aller Hilfetexte. |
| path | Setzen der Verzeichnisse, die nach M–Dateien durchsucht werden. |
| type | Programmcode einer externen MATLAB-Funktion bzw. den Inhalt einer externen Textdatei anzeigen. |
| what | Auflistung aller M–Dateien. |
| which | Vollen Pfadnamen einer (externen) MATLAB-Funktion ausgeben. |
| | Variablen |
| clear | Löschen von Objekten im Arbeitsspeicher. |
| who | Anzeige aller (Arbeits–)Variablen. |
| whos | Ausführliche Anzeige aller (Arbeits–)Variablen. |
| | Unix |
| diary | Protokoll einer MATLAB–Sitzung erstellen. |
| cd | Verzeichnis wechseln. |
| delete | Datei löschen. |
| dir | Verzeichnisinhalt anzeigen. |
| unix | Unix–Kommando ausführen. |
| | Befehlsfenster |
| clc | Inhalt des Befehlsfensters löschen. |
| more | Seitenweise Ausgabe an– bzw. ausschalten. |

| Abfrage | |
|---|---|
| all | Wahr, wenn alle Elemente eines Vektors wahr ($\neq 0$) sind. |
| any | Falsch, wenn alle Elemente eines Vektors falsch ($= 0$) sind. |
| exist | Überprüfen, ob ein MATLAB–Objekt existiert. `exist('A')` liefert:<br>0 `A` existiert nicht.<br>1 `A` ist eine (Arbeits–)Variable.<br>2 `A` ist eine M–Datei.<br>5 `A` ist eine MATLAB–Standardfunktion. |
| find | Bestimmt den Vektor der Indizes der Nicht-Null-Elemente. |
| finite | Wahr für endliche Werte ($\neq$ `NaN, Inf`). |
| isempty | Wahr für leere Objekte. |
| isieee | Wahr, falls der Rechner IEEE-Floatingdarstellung benutzt. |
| isinf | Wahr für unbeschränkte Werte ($=$ `Inf`). |
| isnan | Wahr für `NaN`–Werte. |
| isstr | Wahr für Strings. |
| iscell | Wahr für Zellenfelder. |
| isequal | Wahr, falls (beliebige) Felder identisch sind. |
| islogical | Wahr für logische Ausdrücke. |
| isnumeric | Wahr für numerische Werte. |
| isstruct | Wahr für Strukturen. |
| logical | Konvertiert numerische Werte in logische Werte. |

| Bitoperationen (ab MATLAB 5) | |
|---|---|
| bitand | Bit–weises Und. |
| bitor | Bit–weises Oder. |
| bitxor | Bit–weises Exklusiv–Oder. |
| bitcmp | Bitkomplement (bzgl. N-Bit). |
| bitshift | Bit–weiser Shift. |
| bitget | Rückgabe eines bestimmten Bits. |
| bitset | Setzen eines bestimmten Bits. |
| bitmax | Maximale ganze Gleitpunktzahl (IEEE: $2^{53} - 1$). |
| **Mengenfunktionen (ab MATLAB 5)** | |
| intersect | Schnittmenge (von zwei Vektoren). |
| ismember | Rückgabe eines Vektors, der für jedes Element der ersten Menge (Vektor) angibt, ob es in der zweiten Menge (Vektor) enthalten ist. |
| setdiff | Menge der Elemente, die in der ersten Menge aber nicht in der zweiten Menge enthalten sind. |
| setxor | Komplement der Schnittmenge bzgl. der Vereinigungsmenge. |
| union | Vereinigungsmenge. |
| unique | Elimination von Mehrfachelementen einer Menge. |
| **Zahlentheorie (ab MATLAB 5)** | |
| factor | Primzahlzerlegung. |
| isprime | Wahr für Primzahlen. |
| primes | Liste von Primzahlen. |
| nchoosek | $\binom{n}{k}$. |
| mod | Modulofunktion. |

| Programmierung | |
|---|---|
| break | Vorzeitiges Verlassen von Schleifen (`while, for`). Bei geschachtelten Schleifenkonstrukten wird nur die innerste Schleife verlassen. |
| error | Ausgabe einer Fehlermeldung und Abbruch der Funktionsausführung. |
| return | (Vorzeitige) Rückkehr in die aufrufende Programmeinheit. |
| **Interaktive Eingabe** | |
| keyboard | Interaktive Eingabemöglichkeit innerhalb von M–Dateien. |
| pause | Warten auf Tastendruck. |
| **Spezielle Größen** | |
| ans | Letzter Rückgabewert. |
| computer | Maschinentyp. |
| eps | Relative Genauigkeit von reellen Zahlen. $\left\|\frac{x - rd(x)}{x}\right\| \leq \texttt{eps} = 2^{-52} \approx 2.2 \cdot 10^{-16}.$ |
| flops | Zähler für Gleitkommaoperationen. |
| pi | $\pi$. |
| realmax | Größte darstellbare Zahl. |
| realmin | Kleinste darstellbare Zahl. |
| **Datum und Zeit** | |
| clock | Gibt die aktuelle Uhrzeit und Datum als Zeilenvektor zurück. |
| cputime | Verbrauchte CPU–Zeit in Sekunden seit dem Start von MATLAB. |
| date | Gibt das aktuelle Datum als String zurück. |
| etime | Berechnet die Differenz zwischen zwei „Zeitvektoren“ (s. `clock`). |
| tic, toc | „Stopuhr“. |

# 2D–Grafik

```
plot(X,Y)
plot(Y)
plot(X,Y,S)
handle = plot(X1,Y1,S1,X2,Y2,S2,...)
```

- `plot` erstellt einen 2D-Plot aus den Punktepaaren $(x_i, y_i)$. Die Punkte werden ggf. linear verbunden.
- Sofern nicht anderweitig festgelegt, erfolgt die Skalierung und Farbwahl automatisch.
- `X` und `Y` dürfen auch (gleichartige) Matrizen sein. Sogar die transponierte Angabe nur einer Matrix ist zulässig.
- Wird nur ein Argument angegeben, so wird der Vektor gegen seinen Index aufgetragen, d.h. $x_i = i$ angenommen. Ist `Y` allerdings komplexwertig, so wird `plot(real(Y), img(Y))` gezeichnet.
- Der dritte Parameter `S` ist ein String und kann wie folgt zusammengesetzt werden:

| | | |
|---|---|---|
| y | gelb | . Punkte mit Punkt markiert |
| m | magenta | o Punkte mit Kreis markiert |
| c | cyan | x Punkte mit x markiert |
| r | rot | + Punkte mit Pluszeichen markiert |
| g | grün | * Punkte mit Stern markiert |
| b | blau | - Durchgezogene Linie |
| w | weiß | : Gepunkte Linie |
| k | schwarz | -- Gestrichelte Linie |
| | | -. Strich–Punkt–Linie |

- `plot` gibt einen Spaltenvektor von sog. „Handles“ zurück.

- Anstelle von `plot` kann auch:

  | | |
  |---|---|
  | `loglog` | Logarithmische Skalierung der x– und y–Achse |
  | `semilogx` | Logarithmische Skalierung x–Achse |
  | `semilogy` | Logarithmische Skalierung y–Achse |

  verwendet werden.

Beispiel:

```
x=-pi:.1:pi;plot(x,tanh(x),'m--');
```

Das Erscheinungsbild der Grafik kann u.a. mit folgenden Befehlen beeinflußt werden:

| | |
|---|---|
| `axis` | `axis([`$x_{min}, x_{max}, y_{min}, y_{max}$`])` setzt „Sichtbereich“. |
| `text` | `text(`$x, y$`,'String')` fügt den Text *String* am Punkt $(x, y)$ ein. |
| `title` | `title('String')` versieht die Grafik mit der Überschrift *String*. |
| `xlabel` | `xlabel('String')` beschriftet die x–Achse mit dem Text *String*. |
| `ylabel` | `ylabel('String')` beschriftet die y–Achse mit dem Text *String*. |

`fplot('fname',limit)`

- `fplot` zeichnet die Funktion(en), die im String *fname* in MATLAB-Syntax angegeben ist(sind).
- Als formaler Aufrufparameter muß x verwendet werden.
- Der „Sichtbereich“ wird mit dem Argument *limit* gesetzt. `limit=[`$x_{min}, x_{max}, y_{min}, y_{max}$`]` bzw. `limit=[`$x_{min}, x_{max}$`]`.

- Es können auch mehrere Funktionen gleichzeitig angegeben werden, indem die Funktionen als Komponenten eines Vektors geschrieben werden.

Beispiel:

```
fplot('[tanh(x),tan(x)]',[-pi/3,pi/3]);
```

## Allgemeine Grafikbefehle

Das Grafikfenster kann u.a. mit folgenden Befehlen beeinflußt werden:

| | |
|---|---|
| `figure` | erzeugt ein leeres Grafikfenster. |
| `close` | schließt das Grafikfenster. |
| `clf` | löscht den Inhalt des Grafikfensters. |
| `hold on/off` | Inhalt des Grafikfensters für die nächste Grafik behalten/nicht behalten. |
| `delete` | Grafikobjekt mittels des „Handles“ löschen. |
| `print` | erzeugt druckbare Datei. `print -dps2 name.ps` erzeugt aus einer Grafik eine Postscript–Datei mit dem Dateiname `name.ps`, die anschließend gedruckt werden kann. |

# 3D–Grafik

Für viele der 3D–Grafikfunktionen muß zunächst ein Gitternetz erzeugt werden.

```
[X,Y] = meshgrid(x,y)
```

- `meshgrid` erzeugt aus den Vektoren $x$ und $y$ ein Gitternetz, das durch die Matrizen $X$ und $Y$ beschrieben wird.
- `meshgrid(x)` ist als abkürzende Schreibweise für `meshgrid(x,x)` zulässig.

```
mesh(X,Y,Z)
surf(X,Y,Z)
surfl(X,Y,Z)
plot3(X,Y,Z)
waterfall(X,Y,Z)
```

- Es wird das Objekt gezeichnet, daß durch die Punkte $(x_i, y_j, z_{ij})$ beschrieben wird.
- `mesh` zeichnet ein Gitternetz.
- `plot3` zeichnet eine 3D–Liniengrafik.
- `surf` zeichnet eine (farbige) Oberfläche.
- `surfl` zeichnet eine (farbige) Oberfläche mit Lichteinfall.
- `waterfall` zeichnet eine (abgeschnittene) 3D–Liniengrafik.
- Alle diese Befehle lassen auch weitere Parameterkombinationen zu

- Es wird jeweils ein Spaltenvektor von sog. „Handles“ zurückgegeben.

Beispiele:

```
[X,Y] = meshgrid(-2:.2:2, -2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
mesh(X,Y,Z);

t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t);
```

Das Erscheinungsbild der Grafik kann über die unter dem Punkt 2D–Grafik erwähnten Befehle (ggf. entsprechend erweitert) hinaus u.a. mit folgenden Befehlen beeinflußt werden:

| | |
|---|---|
| `hidden on/off` | Zeichnet nur die sichtbaren Linien/alle Linien beim Befehl `mesh` |
| `shading` | Mit den Optionen `flat, interp, faceted` kann zwischen verschiedenen Schattierungsarten umgeschaltet werden. |
| `view` | Mit Hilfe der Funktion `view` kann der Betrachtungspunkt spezifiziert werden. |

## Hinweis:

- Fast alle der hier erwähnten Grafikbefehle lassen weitere Parameterkombinationen zu (siehe Hilfetexte).
- Es gibt eine Vielzahl weiterer Grafikbefehle (siehe Hilfetexte).