

Anwendung von (Mathematica und) Matlab in der Physikalischen Chemie

5. Grundlegende Sprachkonzepte

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Sommersemester 2016

Motivation

Grundlegende Sprachkonzepte

Syntax

Operatoren

Datentypen

Entscheidungsstrukturen

Schleifen

Ausgangslage

- ▶ Wir wollen, dass uns der Computer Arbeit abnimmt.

Vorgehen

- ▶ verbale Formulierung der Problemstellung
 - Nur wenn wir konkret wissen, was wir wollen, können wir das auch einem Computer beibringen.
- ▶ Aufteilung in kleine Blöcke
 - Selten sehen wir sofort die Lösung für das ganze Problem.
 - Kleine Teilprobleme sind dagegen oft einfach lösbar.
- ▶ Umsetzung dieser Blöcke in Programme
 - Programme = Sprache, die der Computer versteht

Die „schlechte“ Nachricht

- ▶ Programmieren lernen ist wie eine Sprache lernen.
- ▶ Grundlegende Sprachkonzepte müssen bekannt sein.

Die „gute“ Nachricht

- ▶ Es gibt nur wenige grundlegende Programmierparadigma.
- ▶ Es gibt nur wenige Grundkonzepte.
- ☞ Ähnlich wie beim Erlernen einer menschlichen Sprache:
Die grundlegende Grammatik ist (oft) ähnlich.
- ☞ Die hier vorgestellten Konzepte sind recht universell...

Grammatik

- ▶ Operatoren
- ▶ Datentypen
- ▶ Entscheidungsstrukturen
- ▶ Schleifen

Syntax

- ▶ Kommentare
- ▶ Zeilenenden
- ▶ Groß- und Kleinschreibung
- ▶ Zeilenumbrüche und Leerzeilen
- ▶ Einrückungen

Syntax: Matlab-Spezifika

- ▶ **Kommentare**
 - Werden durch Prozentzeichen (%) eingeleitet
 - Alles nach dem „%“ in einer Zeile wird ignoriert.

- ▶ **Zeilenenden**
 - Befehle normalerweise mit Semikolon (;) beenden
 - Ansonsten wird der Variableninhalt ausgegeben

- ▶ **Groß- und Kleinschreibung**
 - Matlab unterscheidet zwischen Groß- und Kleinschreibung

- ▶ **Zeilenumbrüche und Leerzeilen**
 - Zeilenumbrüche in einem Befehl mit „...“
 - Leerzeilen ansonsten beliebig

Syntax: Matlab-Beispiele

Listing 1: Beispiele für die grundlegende Syntax in Matlab

```
1 % Das ist ein Kommentar
2
3 sin(2*pi) % Beispiel fuer einen Kommentar nach einem Befehl
4
5 % Hier wird das Ergebnis ausgegeben
6 sin(2*pi)
7
8 % Hier wird nichts ausgegeben
9 sin(2*pi);
10
11 % Diese beiden Befehle sind nicht identisch
12 sin(2*pi);
13 Sin(2*pi);
14
15 % Zeilenumbruch innerhalb eines Befehls
16 x = [ ...
17     1 2 3 ; ...
18     2 3 4 ; ...
19     3 4 5 ...
20     ];
```


Syntax: Einrückung

- ▶ Nicht jeder Programmiersprache ist Einrückung egal
 - Python verzichtet auf Klammern, aber erzwingt Einrückung
- ▶ Code wird viel öfter gelesen als geschrieben
 - Verständlichkeit und Lesbarkeit sind Trumpf
 - Konsistente Einrückungen sind einfach, aber sehr wirksam
- ▶ (Korrekte) Einrückung erhöht die Lesbarkeit
 - Zusammenhänge sofort erkennbar
 - Blöcke in Schleifen und Bedingungen offensichtlich
 - Inkorrekte Einrückungen sind mühsam
- ☛ Viele Editoren beherrschen automatische Einrückung (auch der Matlab-Editor)



Operator (Mathematik)

mathematische Vorschrift, durch die man aus mathematischen Objekten neue Objekte bilden kann

Operator (Informatik)

Konstrukt, das sich allgemein wie eine Funktion verhält, sich aber syntaktisch/semantisch von Funktionen unterscheidet

Operatoren – Unterscheidung nach Typen

- ▶ arithmetisch
 - $+$, $-$, $*$, $/$, $^$
- ▶ relational
 - $<$, $>$, $<=$, $>=$
 - $==$, $\sim=$
- ▶ logisch
 - $\&$, $|$, $\&\&$, $||$
- ▶ Zuweisung
 - $=$
- ▶ weitere
 - bitweise Operatoren

Operatoren – Unterscheidung nach Stelligkeit

- ▶ unär (einstellig, monadisch)
 - nur ein Operand
 - Beispiel: -1
 - Beispiele in C: $++i$, $k--$
- ▶ binär (zweistellig, dyadisch)
 - zwei Operanden
 - Beispiele: $a+b$, $c = d$
- ▶ ternär (dreistellig, tryadisch)
 - drei Operanden
 - nicht in Matlab
 - Beispiel in C: $a ? b : c$
(Bedingungsoperator: „Wenn a dann b sonst c“)

Operatorrangfolge

- ▶ Immer eine Frage der Definition
 - Einfaches Beispiel: „Punkt vor Strich“
- ▶ Für jede Programmiersprache anders
 - Einzige Chance: in der Dokumentation nachschauen
 - Matlab lehnt sich stark an die Regeln der Mathematik an.
- ▶ Nicht immer eindeutig
 - Was immer hilft: Klammern setzen
 - Beispiel aus Matlab: $-1^2 \neq (-1)^2$



Typisierung

Zuweisung eines Objekts einer Programmiersprache (zum Beispiel einer Variable) zu einem Datentyp

- ▶ Datentypen
 - Numerisch
 - Zeichen und Zeichenketten (*strings*)
 - Boolesche Ausdrücke
 - Zeiger (Referenzen)
 - Komplexe Datentypen
- ▶ Bedeutung unterschiedlicher Datentypen
 - Mit Zeichenketten kann man (meist) nicht rechnen.
 - Befehle erwarten meist bestimmte Datentypen.

Numerische Datentypen

- ▶ Zwei Unterscheidungsmöglichkeiten
 - 1 Dimension
 - 2 Präzision/Wertebereich

Dimension

- ▶ Skalar (1×1)
- ▶ Vektor ($1 \times n$, $n \times 1$)
- ▶ Matrix ($n \times m$)

Präzision/Wertebereich

- ▶ (signed) integer
- ▶ real/float/double

- ☛ Matlab rechnet normal mit Gleitkommazahlen (double).
 - Standard: ANSI/IEEE 754-1985

Numerische Datentypen

Listing 2: Numerische Datentypen unterschiedlicher Dimension in Matlab

```
1 % Skalar
2 number = 1;
3 emptyScalar = [];
4
5 % Vektoren
6 rowVector    = [1 2 3 4 5];
7 rowVector    = [1, 2, 3, 4, 5];
8 rowVector    = 1:5;
9
10 columnVector = [1; 2; 3; 4; 5];
11 columnVector = rowVector';
12
13 % Matrix
14 matrix = [ 1 2 3 ; 2 3 4 ; 3 4 5 ; 4 5 6 ];
```

- ▶ Zeilenvektor: eine Zeile/Reihe ($1 \times n$)
- ▶ Spaltenvektor: eine Spalte ($n \times 1$)

Indizierung numerischer Datentypen

- ▶ Gilt strenggenommen für alle geordneten Listen
Geordnete Liste Struktur, deren Felder über einen ganzzahligen numerischen Index adressiert werden.
- ▶ Matlab-Spezifika
 - Indices in runden Klammern, *immer* positiv, beginnen mit 1
 - Reihenfolge bei zweidimensionalen Matrizen: Reihe, Spalte
- ▶ Spezielle Indices
 - `end` – das letzte Element einer geordneten Liste
 - `:` – alle Elemente einer Dimension
- ▶ Zugriff auf Bereiche einer Dimension
 - Über eine Liste oder einen Bereich von Indices

Indizierung numerischer Datentypen

Listing 3: Indizierung geordneter Listen in Matlab

```
1 % Matrix
2 matrix = [ 1 2 3 ; 2 3 4 ; 3 4 5 ; 4 5 6 ];
3
4 % Erste Reihe der Matrix
5 firstRow = matrix(1,:);
6
7 % Zweite Spalte der Matrix
8 firstRow = matrix(:,2);
9
10 % Zweite und dritte Reihe der dritten Spalte
11 selection = matrix(2:3,3);
12 selection = matrix([2,3],3);
13
14 % Letzte Reihe der Matrix
15 lastRow = matrix(end,:);
16
17 % Erste bis vorletzte Spalte der Matrix
18 selection = matrix(:,1:end-1);
```

Genauigkeit von Gleitkommazahlen

- ▶ Grundsätzliches Problem
 - Es gibt unendlich viele rationale (und reelle) Zahlen.
 - Unendliche Genauigkeit erfordert unendlich viel Platz.
 - Begrenzter Speicher erlaubt nur endliche Genauigkeit.

- ▶ Die Situation vor 1985
 - Jede Architektur hatte ihre eigene Lösung.
 - Die Genauigkeit war rechnerabhängig.
 - Ergebnisse numerischer Rechnungen waren inkompatibel.

- ▶ Der Standard ANSI/IEEE 754-1985
 - Standard für die Repräsentation von Gleitkommazahlen
 - Zumindest für 64-bit-Repräsentation eindeutig

Zeichen und Zeichenketten

- ▶ `character`, `char`
 - Einzelnes Zeichen
- ▶ `string`
 - Zeichenkette aus einzelnen Zeichen (`char`)
- ▶ Zeichenketten in Matlab
 - Zeichenketten immer in Hochkommata ('...') eingeschlossen
 - Mehrdimensionale Strings (Array von Zeichenketten):
Reihen müssen gleiche Spaltenzahl haben
- ☞ Mehrzeilige Texte in `cell arrays` ablegen

Boolesche Ausdrücke

- ▶ Zwei Werte
 - wahr (*true*), unwahr (*false*)
- ▶ Rückgabewerte relationaler Operatoren
- ▶ Matlab
 - `true`, `false`
 - `0` gilt als `false`
 - `≠0` gilt als `true`
- ➡ Entscheidungsstrukturen
- ➡ logische Indizierung



George Boole
(1815–1864)

Komplexe Datentypen

- ▶ `cell array` (Liste)
 - Daten unterschiedlicher Typen und Größen
 - In den Feldern eines Datenfeldes (*array*) gespeichert
 - „Generalisiertes“ Datenfeld (*array*)
 - Felder numerisch (mit ganzen Zahlen) indiziert

 - ▶ `structure` (Wörterbuch)
 - Daten unterschiedlicher Typen und Größen
 - In den Feldern einer Struktur gespeichert
 - Assoziatives Datenfeld
 - Felder mit Namen (*strings*) indiziert
- ☞ Beide sind hierarchisch verschachtelbar.

Komplexe Datentypen

Geordnete Listen

#	Wert
1	0.0000
2	0.0025
3	0.0050

⋮

n-1	0.2475
n	0.2500

#	Wert
1	'Im'
2	'Anfang'
3	'war'

⋮

n-1	'die'
n	'Tat'

Assoziative Datenfelder

Schlüssel	Wert
Name	'K. Racht'
Alter	42

Adresse	Schlüssel	Wert
	Straße	'Talstraße'
	Nummer	21

Hobbies	{'...', '...'}
---------	----------------

cell arrays

Listing 4: cell arrays in Matlab

```
1 % Empty cell array
2 C = cell(0);
3
4 % Cell array of strings
5 C = {'Im', 'Anfang', 'war'};
6
7 % Cell array with different types
8 C = {'String', [1 2 3], 'String', [1 2 3; 2 3 4; 3 4 5]};
9
10 % Same cell array as above
11 C{1} = 'String';
12 C{2} = [1 2 3];
13 C{3} = 'String';
14 C{4} = [1 2 3; 2 3 4; 3 4 5];
15
16 % Accessing a field
17 foo = C{1}; % Returns a string
18 foo = C(1); % Returns a 1x1 cell
```

structures

Listing 5: structures in Matlab

```
1 % Empty structure
2 S = struct();
3
4 % Structure with some field and value
5 S.field = 'value'
6
7 % Structure with fields of different types
8 S.field1 = 'value';
9 S.field2 = pi;
10 S.field3 = [1 2 3];
11
12 % Same structure as above
13 S = struct(...
14     'field1','value',...
15     'field2',pi,...
16     'field3',[1 2 3] ...
17 );
18
19 % Accessing a field
20 foo = S.field3;
```

Konditionale Strukturen

- ▶ `if...else`
 - Testet auf bestimmte Bedingung
 - Mehrere Bedingungen über logische Operatoren verknüpft

Logische Operatoren

- ▶ Arten logischer Operatoren
 - AND (`&`), OR (`|`), EQUAL (`eq()`, `==`), NOT (`not()`, `~`)
 - Klammern zur Gruppierung logischer Ausdrücke
- ▶ „Kurzschluss-Operatoren“
 - `&&`, `||`
 - Überprüfung bricht ab, sobald die Bedingung erfüllt ist
 - Beispiel: `A && B && C` bricht nach `A` ab, wenn `A` unwahr

Listing 6: Einfachste Form einer if-Struktur in Matlab

```
1 if <condition>
2     % do something
3 end
```

- ▶ Abbruch der weiteren Abarbeitung und Rückkehr zum Aufrufer über `return`

Listing 7: if-Struktur mit Alternativzweig

```
1 if <condition>
2     % do something
3 else
4     % do something else
5 end
```

 **Tipp:** Invertierte Logik spart häufig den `else`-Zweig

Entscheidungsstrukturen – Zwei praktische Beispiele

Listing 8: Reales Beispiel einer if-Struktur in Matlab

```
1 % Compare current year
2 if str2double(datestr(now,'yyyy')) < 2016
3     disp('You''re outdated.');
```

```
4 else
5     disp('You''re in time.');
```

```
6 end
7
8 % "now"           - returns current date and time
9 % "datestring"   - formats date - here, "yyyy" means four-digit year only
10 % "str2double"  - converts string into number for comparison
```

Listing 9: Überprüfung der Zahl der Eingabeparameter

```
1 % nargin returns the number of input arguments of a function
2 if nargin ~= 2
3     return;
4 end
```

Listing 10: if-Struktur mit mehreren Alternativbedingungen

```
1 if <condition1>
2     % do something
3 elseif <condition2>
4     % do something else
5 else
6     % do something else
7 end
```

Listing 11: if-Strukturen lassen sich verschachteln

```
1 if <condition1>
2     if <additionalCondition>
3         % do something
4     else
5         % do something else
6     end
7 elseif <condition2>
8     % do whatever
9 else
10    % give up
11 end
```

Fallunterscheidungen

- ▶ `switch...case`
- ▶ Vorteile gegenüber `if...elseif...else`
 - Oft übersichtlicher
 - Gut für Unterscheidung mehrerer Fälle (>2) geeignet
- ▶ Beschränkungen von Matlab
 - Nur Skalare oder Zeichenketten (*Strings*) als Schalter
 - Keine Bedingungen in den Fällen
- ☞ Sollten immer einen `otherwise`-Zweig haben, um definierte Bedingungen zu schaffen.

Listing 12: Einfachste Form einer switch-case-Struktur in Matlab

```
1 switch switch_expression
2     case case_expression
3         statements
4     case case_expression
5         statements
6     % ...
7 end
```

Listing 13: switch-case-Struktur mit otherwise-Zweig

```
1 switch switch_expression
2     case case_expression
3         statements
4     case case_expression
5         statements
6     % ...
7     otherwise
8         statements
9 end
```

Schleifen

▶ `for`-Schleifen

- Definierte Anzahl an Schleifendurchläufen
- Typisches Einsatzgebiet:
Iterieren über die Elemente eines Vektors

▶ `while`-Schleifen

- Schleifendurchlauf, solange eine Bedingung wahr ist
- Typisches Einsatzgebiet:
Zeilenweises Einlesen einer Datei

☞ Abbruch einer Schleife über `break`

for-Schleifen

Listing 14: Einfachste Form einer for-Schleife in Matlab

```
1 for loopIndex = start : stop
2     % Do something
3 end
```

- ▶ `loopIndex` wird in jedem Durchlauf um 1 erhöht

Listing 15: for-Schleife mit angegebenem Inkrement

```
1 for loopIndex = start : increment : stop
2     % Do something
3 end
```

- ▶ `increment` kann negativ und nicht-ganzzahlig sein

for-Schleifen: Ein praktisches Beispiel

Listing 16: Iterieren über alle Elemente eines Vektors

```
1 % Define vector
2 x = 1:0.1:2*pi;
3
4 % Loop over each element of vector x
5 for k = 1 : length(x)
6     y = sin(x(k));
7 end
```

► Anmerkungen

- Als Laufvariable *nie* `i` oder `j` verwenden (komplexe Zahl)
- Als Laufvariable *nie* `1` verwenden (`1` oder `1`?)

► Eigenheiten von Matlab

- `for`-Schleifen sind langsam, lassen sich oft vermeiden

while-Schleifen

Listing 17: Einfachste Form einer while-Schleife in Matlab

```
1 while condition
2     % Do something
3 end
```

- ▶ Bedingung wird am Anfang jedes Durchlaufs überprüft.
 - Matlab kennt (anders als andere Sprachen) keine Schleifen, die die Bedingung erst am Ende überprüfen.
- ▶ Bedingung muss sich innerhalb der Schleife ändern.
 - Wird sonst zur „Endlosschleife“



...Zeit für eigene praktische Arbeit...

Vorschau: [Dokumentation](#)

- ▶ Dokumentation
- ▶ Dokumentation im Code
- ▶ Probleme von Dokumentation