Übungen zur „Einführung in die wissenschaftliche Programmierung“

MATLAB Einführung Teil I

Modifiziertes Exzerpt aus:

- Christian Karpfinger, Boris von Loesch: MATLAB – Eine Einführung, 14. Oktober 2013
- https://www-m11.ma.tum.de/fileadmin/w00bnb/www/people/karpfinger/MATLAB-Tutorial.pdf

Gekürzt und modifiziert:

- L.Billmann, 15.Mai 2017

Inhaltsverzeichnis

Einleitung

MATLAB ist eine mächtige Anwendung für Mathematiker und Ingenieure, die von dem Unternehmen The MathWorks, Inc., kurz MathWorks, entwickelt und vertrieben wird. Vielfältigste Funktionen zur Lösung numerischer Probleme, der Visualisierung von Daten und die Erweiterbarkeit des Basispakets über Toolboxen machen MATLAB zu einem guten Werkzeug für Studenten der Mathematik, Natur- und Ingenieurwissenschaften. MATLAB enthält eine komfortable IDE (integrated development environment) die den Programmierer bei seiner Arbeit unterstützt. In ihr sind unter anderem enthalten

- eine interaktive Codeeingabe,
- eine ausführliche Hilfe,
- ein Quellcode Editor mit Syntax Highlighting,
- ein Debugger, um Programme schrittweise durchlaufen zu lassen,
- ein Profiler zum Erkennen von Geschwindigkeitsengpässsen.

Im Gegensatz zu anderen Programmiersprachen wie C, Fortran oder Java muss ein MATLAB Code vor der Ausführung nicht kompiliert werden, sondern wird direkt interpretiert (seit MATLAB R6.5 wird der Code während der Ausführung durch einen Just-In-Time Kompilierer übersetzt um eine höhere Performance im Vergleich zur direkten Ausführung zu erhalten). Auch müssen Variablen vor ihrer ersten Verwendung nicht deklariert werden. Deshalb ist MATLAB in die Kategorie der Skriptsprachen, wie z.B. Python, Tcl oder Perl, einzuordnen.
Das Besondere an MATLAB ist die Verwendung von mehrdimensionalen Feldern (auch mit komplexen Zahlen) wie Vektoren und Matrizen als Standardtypen und eine umfangreiche mathematische Bibliothek. Damit ist es möglich, Algorithmen in einer mathematischen Syntax unter Verwendung von Standard-komponenten wie z.B. einer LR-Zerlegung schnell zu implementieren (Prototyping). Der Vorwurf, dass Programme in MATLAB im Vergleich zu Fortran oder C viel langsamer sind, ist im Allgemeinen nicht berechtigt, wenn man bei der Programmierung gewisse Prinzipien befolgt. Dies liegt daran, dass MAT-LAB intern sehr schnelle Routinen z.B. für das Matrix-Vektor-Produkt oder die Fouriertransformation verwendet; wie andere Numerikpakete verwendet MATLAB für die Lineare Algebra die BLAS und LA-PACK Bibliotheken.

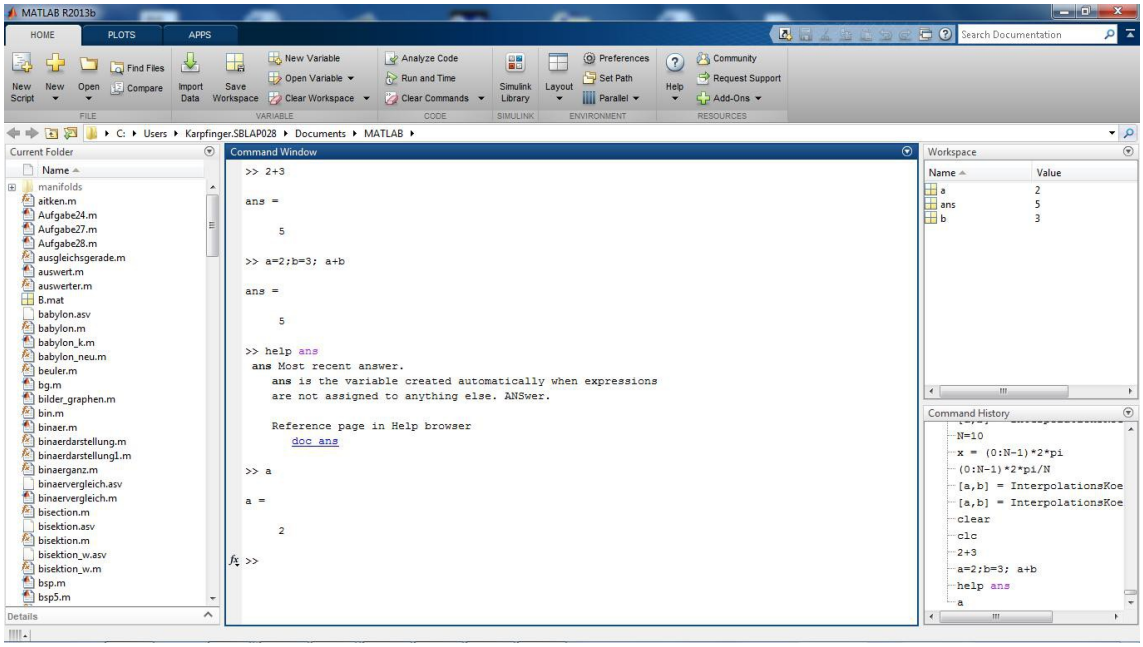
Diese Einführung soll einen Einblick in den Aufbau und die grundlegenden Funktionen von MATLAB geben. Weiterführende Informationen findet man in der umfangreichen eingebauten Hilfe, in den vielen MATLAB Bücher und natürlich auch im Internet.

Kapitel 1

Erste Schritte

1.1 Die Oberfläche

Beim Start von MATLAB hat man vier Fenster:



- Das **Command Window**: Hier werden die Befehle und Variablenzuweisungen eingegeben. Testen Sie 2+3 oder a=2; b=3; a+b oder auch help ans, jeweils mit <return> abschließen.
- Das **Current Folder**: Das ist das aktuelle Verzeichnis, standardmäßig ist das MATLAB unter EigeneDateien. Beim ersten Arbeiten mit MATLAB ist das Verzeichnis üblicherweise leer. Hier findet man Dateien mit den Endungen .m (sogenannte *M*-Files) bzw. .mat (abgespeicherte Variable oder Parameter).
- Der **Workspace**: Hier sind die momentan benutzen Variablen und Parameter zu sehen.
- Die **Command History**: Hier werden die ausgeführten Eingaben im Command Window aufge-listet. Durch Doppelklick eines dieser Befehle wird dieser erneut ausgeführt (Alternative: Cursor-↑ und Cursor-↓).

1.2 Grundrechenarten

Zu den Grundrechenarten zählen wir Addition +, Substraktion -, Multiplikation *, Division / und Potenzbildung ^.
Variablen müssen dabei nicht gesondert deklariert werden. Sie werden durch eine Wertzuweisung erzeugt, und ihre Werte können anderen Variablen übertragen werden.

Das Ergebnis eines Befehls wird standardmäßig direkt ausgegeben. Dies kann unterdrückt werden durch Abschließen des Befehls mit einem Semikolon. Wird das Ergebnis eines Befehls nicht in einer Variable gespeichert, wird es automatisch in die Variable ans geschrieben.

Mehrere Befehle können durch Kommata (mit Ausgabe der Ergebnisse) oder Semikolon (ohne Ausgabe der Ergebnisse) getrennt in eine Zeile geschrieben werden. Hierbei wird nur das letzte Resultat in ans gespeichert.

```
>> c=1
c =
    1

>> x=3*c-4;
>> x
x =
    -1

>> r=x^3
r =
    -1

>>r+c
ans =
    0
```

Bemerkung. Komplexe Zahlen werden durch

<Realanteil>+<Imaginäranteil>i

eingegeben, z. B. 2+0.5i. Aus diesem Grund sollte die imaginäre Einheit i nicht durch eine Variable überschrieben werden, da es sonst Probleme mit komplexen Zahlen geben kann.

MATLAB verwendet automatisch komplexe Zahlen, wenn dies nötig ist. So wird z.B. $\sqrt{(-1)}$ richtig als komplexe Zahl interpretiert. In vielen anderen Programmiersprachen würde dieser Aufruf eine Fehlermeldung erzeugen.

1.3 Einfache Funktionen

Wir unterscheiden etwas eigenwillig und sehr grob *einfache* und *komplizierte* Funktionen: Einfache Funktionen sind dabei solche, die man schnell mal im Command Window auf Zahlen oder Vektoren anwenden kann, komplizierte Funktionen sind solche, die man besser separat in einer sogenannten .m-Datei erklärt, um sie erst dann im Command Window aufzurufen.
Einfache Funktionen sind die Sinusfunktion sin, Kosinusfunktion cos, Exponentialfunktion exp, Logarithmusfunktion log, Wurzelfunktion sqrt .... Die Anwendung dieser Funktionen auf Zahlen erfolgt mit runden Klammern etwa im Command Window:

```
>> sin(pi/2)
ans =
    1

>> a=sin(pi/4);
>> a
a =
    0.7071

>> sqrt(2)
ans =
    1.4142
```

Weitere einfache Funktionen, die man im Command Window erklären und anwenden kann, sind Polynom-funktionen oder Komposita einfacher Funktionen ... Solche Funktionen kann man im Command Window problemlos als sogenannte *anonyme Funktionen* erklären. Im folgenden Beispiel erklären wir eine solche anonyme Funktion f und wenden sie auf verschiedene Zahlen an:

```
>> f = @(x) x^2 + sin(x^2+pi/2)
f =
    @(x)x^2+sin(x^2+pi/2)

>> f(0)
ans =
    1

>> f(pi)
ans =
    8.9669
```

Wir können auch Funktionen in mehreren Vartiablen auf diese Weise erklären, etwa eine Funktion g in den Variablen x und y:

```
>> g=@(x,y) x^2 -y^2
g =
    @(x,y)x^2-y^2

>> g(1,2)
ans =
    -3

>> g(2,1)
ans =
    3
```

MATLAB unterscheidet zwischen Groß- und Kleinschreibung. So kann der Befehl sin(pi/2) richtig interpretiert werden, die Eingabe von Sin(pi/2) gibt jedoch eine Fehlermeldung zurück:

??? Undefined function or method 'Sin' for input arguments of type 'double'.

1.4 Nützliche Kleinigkeiten

Bevor wir nun mit Vektoren und Matrizen weitermachen, halten wir kurz inne und stellen einige wenige, aber sehr nützliche Kleinigkeiten zusammen:

- clc: damit leert man das Command Window.
- clear: damit entfernt man alle Variablen im Workspace. Natürlich können auch einzelne Variable gelöscht werden. So entfernt etwa clear a die Variable a.
- Die Tastenkombination Strg+C bricht MATLAB (meistens) ab, falls Sie z. B. eine Endlosschleife erzeugt haben.
- help: damit ruft man die Hilfe auf, z. B. help sin oder help clear.
- Falls die help-Hilfe nicht ausreichend ist, so greife man auf doc zurück, z. B. doc sin.
- Zahlenformate in MATLAB: Mit format kann das Zahlenformat geändert werden:
  - format short – das ist standardmäßig voreingestellt, z. B. 0.3212.
  - format long – z. B. 0.321234276512387.
  - format rat – MATLAB rechnet mit rationalen Zahlen, z. B. e = 1457/536.

1.5 Vektoren bzw. Matrizen erzeugen

Wir fassen Spaltenvektoren als einspaltige Matrizen und Zeilenvektoren als einzeilige Matrizen auf. Somit können wir uns auf Matrizen beschränken.
Matrizen können auf mehrere Arten erzeugt werden. Viele Typen von Matrizen können direkt über eine MATLAB-Funktion generiert werden. Die Nullmatrix, die Einheitsmatrix und Einsmatrizen können über die Funktionen zeros, eye und ones erzeugt werden. Alle

haben die gleiche Syntax. Zum Beispiel erzeugt zeros(m,n) oder zeros([m,n]) eine m×n Nullmatrix, während zeros(n) eine n×n Nullmatrix erzeugt.

```
>> zeros(2)
ans =
    0  0
    0  0

>> ones(2,3)
ans =
    1  1  1
    1  1  1

>> eye(3,2)
ans =
    1  0
    0  1
    0  0
```

Mit rand erzeugt man Matrizen mit Pseudozufallszahlen als Einträge. Die Syntax ist die gleiche wie bei eye. Ohne Argument gibt die Funktion eine einzelne Zufallszahl zurück.

```
>> rand
ans =
    0.9501

>> rand(3);
ans=
    0.2311  0.8913  0.0185
    0.6068  0.7621  0.8214
    0.4860  0.4565  0.2835
```

Matrizen können explizit über die Klammernotation (square bracket notation) erzeugt werden. Zum Beispiel kann eine 3x3-Matrix mit den ersten neun Primzahlen mit folgendem Befehl erzeugt werden:

```
>> A = [2  3  5
        7 11 13
        17 19 23]
A =
    2   3   5
    7   11  13
    17  19  23
```

Das Ende einer Zeile kann über ein Semikolon anstatt eines Zeilenumbruchs angegeben werden. Ein kürzere Variante des letzten Beispiels ist also:

```
>> A = [2 3 5; 7 11 13; 17 19 23]
```

Innerhalb einer Zeile können einzelne Elemente über ein Leerzeichen oder Komma getrennt werden. In ersterem Fall sollte beachtet werden, dass bei Angabe von Vorzeichen

für die einzelnen Einträge kein Leerzeichen zwischen Vorzeichen und Element gelassen werden darf. MATLAB interpretiert das Vorzeichen sonst als Plus oder Minus Operator.

```
>> v = [-1 2 -3 4]
v =
    -1   2   3   4

>> w = [-1,2,-3,4]
w =
    -1   2   -3   4

>> x = [-1 2 - 3 4]
x =
    -1   -1   4
```

Häufig sehr praktisch ist das Erzeugen von Matrizen durch Angabe von Blöcken, anstatt der einzelnen Elemente. Sei z.B. B=[1 2; 3 4]:

```
>> C = [B zeros(2); ones(2) eye(2)]
C =
    1   2   0   0
    3   4   0   0
    1   1   1   0
    1   1   0   1
```

1.5.1 Spezielle Matrizen

Über fünfzig spezielle und berühmte Matrizen können mit dem Befehl gallery erzeugt werden, diese sind teilweise dünn besetzt. Mehr über diese speziellen Matrizen erfährt man in der Hilfe mit doc gallery.

1.5.2 Doppelpunkt Operator

Der Doppelpunkt Operator ist einer der wichtigsten Operatoren in MATLAB und findet in vielen Fällen Verwendung. Mit seiner Hilfe können spezielle Zeilenvektoren erzeugt werden, die z.B. bei der Indizierung in for Schleifen oder beim Plotten verwendet werden. Dabei wird ausgehend von einer Zahl solange eine Einheit addiert und in dem Vektor gespeichert, bis ein vorgegebenes Ende erreicht oder überschritten wurde. Die allgemeine Syntax ist

<Start>:<Ende> oder <Start>:<Increment>:<Ende>.

Ein paar Beispiele sollten die Verwendung deutlich machen:

```
>> j=1:5
j =
    1  2     3     4     5

>> X=1.2:0.2:2
X =
    1.2000  1.4000     1.6000     1.8000     2.0000

>> X=1:-0.3:0
X =
    1.0000  0.7000     0.4000     0.1000
```

Dem Doppelpunkt Operator verwandt ist die Funktion linspace, die als Eingabe neben Start und Ende die Anzahl der zu erzeugenden Punkte verlangt anstatt des Abstandes. linspace(a,b,n) erzeugt n Punkte gleichen Abstands zwischen a und b. Der Standardwert für n ist 100.

```
>> linspace(-1,1,9)
ans =
    Columns 1 through 6
    -1.0000  -0.7500  -0.5000  -0.2500      0  0.2500

    Columns 7 through 9
    0.5000  0.7500  1.0000
```

1.6 Indizierung und Doppelpunktnotation

Die Indizierung von Feldern erfolgt in MATLAB durch runde Klammern und startet beim Index 1. Bei Matrizen steht der erste Index für die Zeilen-, der zweite für die Spaltennummer des Elements.

```
>> A=rand(2,2)
A =
    0.6557  0.8491
    0.0357  0.9340

>> A(1,1)
ans =
    0.6557

>> A(2,1)
ans =
    0.0357
```

Es gibt bei MATLAB keine nicht positiven Indizes, der Versuch wird durch eine Fehlermeldung bestraft:

```
>> A(0,0)
??? Attempted to access A(0,0); index must be a positive integer or
    logical.
```

Ein Vorteil von MATLAB gegenüber anderen Programmiersprachen ist, dass nicht nur auf die einzelnen Elemente eines Feldes zugegriffen werden kann, sondern auf beliebige Teilblöcke. Dafür ersetzt man einfach beim Indizieren die Zahlen durch Vektoren. Ist A eine n × m-Matrix und r und c Vektoren mit positiven ganzzahligen Elementen kleiner gleich n (bzw. m). Dann ist B=A(r,c) eine Matrix mit Einträgen bij =A(r(i),c(j)).

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

>> A([1,2],3)
ans =
    3
    6

>> A([2,3],[2,3])
ans =
    5  6
    8  9
```

Besonders häufig wird in diesen Fällen der Doppelpunktoperator verwendet. Die Teilmatrix bestehend aus der Schnittmenge der Zeilen p bis q und den Spalten r bis s wird mit A(p:q,r:s) zurückgegeben. Ein Sonderfall ist ein einzelner Doppelpunkt, dieser wählt sämtliche Zeilen oder Spalten; A(:,j) bezeichnet also die j-te Spalte, und A(i,:) die i-te Zeile von A. Das Schlüsselwort end steht für den letzten Index in der angegebenen Dimension; A(end,:) bezeichnet also die letzte Zeile von A.

```
>> A(1:3,2:end)
ans =
    2  3
    5  6
    8  9

>> A(1,:)
ans =
    1  2  3

>> A(:,end)
ans =
    3
    6
    9
```

1.7 Operatoren und Funktionen

Um ein Feld zu transponieren, stellt MATLAB den Operator ' zur Verfügung.

```
>> A=[1 , 2; 1 , 2]
A =
    1  2
    1  2

>> A'
ans =
    1  1
    2  2
```

1.7.0 Rechenoperatoren

MATLAB unterstützt das Rechnen mit Vektoren und Matrizen. So können zwei Felder gleicher Dimensionen einfach durch + addiert und mit - subtrahiert werden:

```
>> x=1:3;
>> y=2:4;
>> x+y
ans =
    3  5  7

>> eye(2)-ones(2)
ans =
     0  -1
    -1   0
```

MATLAB interpretiert den Multiplikationsoperator * als Matrixprodukt oder als Multiplikation mit einem Skalar. Bei ersterem muss die Anzahl der Spalten des ersten Arguments gleich der Anzahl der Zeilen des zweiten Argumentes sein. Daneben gibt es noch den elementweisen Multiplikationsoperator **.***.

```
>> x=1:3;
>> y=2:4;
>> x*y
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> x*y'
ans =
    20

>> x.*y
ans =
    2   6   12

>> A=[1,2,3;4,5,6;7,8,9];
>> A*y
ans =
    20
    47
    74

>> 2*A
ans =
    2    4    6
    8   10   12
   14   16   18
```

Auch das Potenzieren ^ wird im Sinne des Matrixprodukt interpretiert, analog zur Multiplikation gibt es auch die „gepunktete“ Version .^.
Die Division gibt es in zwei Ausführungen: den Slash / und den Backslash \. Beide entsprechem dem (ggf. approximativen) Lösen eines linearen Gleichungssystems. B/A steht ungefähr für BA^{-1}, A\B für $A^{-1}B$).

```
>> A=[1 0.5;0.5 1/3];
>> b=[1;2];
>> A\b
ans =
    -8
    18
```

Natürlich gibt es auch wieder das elementweise dividieren ./, dieses muss auch eingesetzt werden, wenn ein Skalar durch einen Vektor geteilt wird.

```
>> x=1:3;
>> 3/x
??? Error using ==> mrdivide Matrix dimensions must agree.

>> 3./x
ans =
    3.0000  1.5000  1.0000

>> x./x
ans =
    1  1  1
```

1.7.2 Basisfunktionen

Um Eigenschaften wie die Länge oder Dimensionen von Feldern abzufragen, gibt es in MATLAB eine Hand voll Funktionen:

- length Gibt die Länge eines Vektors zurück, bei einer Matrix wird die größere der beiden Dimensionen zurückgegeben.
- size Gibt die Dimensionen einer Matrix zurück.
- numel Gibt die Anzahl der Elemente einer Matrix zurück.

```
>> B=[1,2;2,3;4,5]
B =
    1  2
    2  3
    4  5

>> length(B)
ans =
    3

>> size(B)
ans =
    3  2

>> numel(B)
ans =
    6
```

1.7.3 Funktionen

MATLAB verfügt über unzählige Funktionen die Vektoren und Matrizen als Eingabe erwarten. Grundsätzlich verändert eine Funktion in MATLAB kein Eingabeargument (Eingabeargumente werden als Wert und nicht als Referenz übergeben). Viele dieser Methoden kann man einer von drei Gruppen zuordnen:

- Skalare Funktionen
- Vektorfunktionen
- Matrixfunktionen

Skalare Funktionen sind solche, die komponentenweise wirken. Beispiele sind sin, cos, exp und factorial (Fakultät). Übergibt man diesen Funktionen ein mehrdimensionales Feld, hat die Rückgabe die gleiche Dimensionen wie das Eingabeargument und die Funktion wurde auf jeden Eintrag des Feldes angewendet.

```
>> x=linspace(-pi/2,pi/2,5)
x =
    -1.5708  -0.7854     0   0.7854  1.5708

>> sin(x)
ans =
    -1.0000  -0.7071     0  0.7071  1.0000
```

Vektorwertige Funktionen operieren auf Vektoren und geben ein Skalar oder einen Vektor zurück. Beispiele sind max, sum, prod mit skalarem Rückgabewert, diff (Differenz jeweils aufeinander folgender Elemente), cumsum oder sort geben einen Vektor zurück. Übergibt man diesem Typ von Funktion eine Matrix, wird die Funktion auf jede Spalte angewendet und die Rückgabe wieder in eine Spalte geschrieben.

```
>> x=1:4;
>> sum(x)
ans =
    10

>> max(x)
ans =
    4

>> diff(x)
ans =
    1  1  1
```

1.8 Matrizen manipulieren

Es gibt mehrere Befehle für die Manipulation von Matrizen. Die Funktion reshape ändert die Dimensionen einer Matrix: reshape(A,m,n) erzeugt eine m × n Matrix, deren Elemente spaltenweise aus A entnommen werden.

```
>> A = [1 4 9; 16 25 36], B = reshape(A,3,2)
A =
     1   4   9
    16  25  36

B =
     1  25
    16   9
     4  36
```

Die Notation [] steht für eine leere 0 × 0-Matrix. Weist man einer Zeile oder Spalte einer Matrix den Wert [] zu, so wird sie aus der Matrix gelöscht.

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

>> A(2,:)=[]
A =
     8   1   6
     4   9   2
```

In diesem Beispiel würde der gleiche Effekt durch A = A([1 3],:) erzielt werden. Die leere Matrix ist auch als Platzhalter in Argumentlisten nützlich.
An einen vorhandenen Vektor können einfach weitere Elemente angehängt werden, indem man einem Index der größer als die Länge des Vektors ist, einen Wert zuweist. Der Vektor wird daraufhin automatisch bis zu dem entsprechenden Index verlängert und mit Nullen gefüllt.

```
>> x=1:3;
>> x(6)=9
x =
     1   2   3   0   0   9
```

Auf dieselbe Weise können neue Zeilen und Spalten an eine Matrix angehängt werden:

```
>> A=eye(2)
A =
     1   0
     0   1
>> A(3,:)=[1,1]
A =
     1   0
     0   1
     1   1
>> A(:,3)=2
A =
     1   0   2
     0   1   2
     1   1   2

>> A=eye(2)
A =
     1   0
     0   1
>> A(3,3)=2
A =
     1   0   0
     0   1   0
     0   0   2
```