Übungen zur „Einführung in die wissenschaftliche Programmierung“

MATLAB Einführung Teil II

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

Teil I

1 Erste Schritte
- 1.1 Die Oberfläche
- 1.2 Grundrechenarten
- 1.3 Einfache Funktionen
- 1.4 Nützliche Kleinigkeiten
- 1.5 Vektoren bzw. Matrizen erzeugen
  - 1.5.1 Spezielle Matrizen
  - 1.5.2 Doppelpunkt Operator
- 1.6 Indizierung und Doppelpunktnotation
- 1.7 Operatoren und Funktionen
  - 1.7.1 Rechenoperatoren
  - 1.7.2 Basisfunktionen
  - 1.7.3 Funktionen
- 1.8 Matrizen manipulieren

Teil II

2 Operatoren und Flusskontrolle
- 2.1 Relationale Operatoren
- 2.2 Logische Operatoren
- 2.3 Flusskontrolle

3 M-Dateien
- 3.1 Skriptdateien
- 3.2 Funktionsdateien

Kapitel 2

Operatoren und Flusskontrolle

2.1 Relationale Operatoren

Die relationalen Operatoren von MATLAB sind

==	gleich
~=	ungleich
<	weniger als
>	größer als
<=	kleiner oder gleich
>=	größer oder gleich

Tabelle 2.1: Die relationalen Operatoren in MATLAB

Beachten Sie, dass in MATLAB ein einzelnes Gleichheitszeichen = eine Zuweisung angibt und nie auf Gleichheit testet. Vergleiche zwischen Skalaren erzeugen logisch 1 wenn die Relation wahr und logisch 0 wenn sie falsch ist. MATLAB verwenden hier intern keine doubles, sondern einen anderen Datentyp (logical), dies sollte aber in den wenigsten Fällen zu Problemen führen.
Vergleiche sind auch zwischen Matrizen gleicher Dimension definiert und zwischen Matrizen und Skalaren, deren Ergebnis in beiden Fällen eine Matrix mit Nullen und Einsen ist. Für Matrix-Matrix Vergleiche werden die entsprechenden Paare von Elementen verglichen, während für Matrix-Skalar Vergleiche der Skalar mit jedem Matrixelement verglichen wird.

```
>> A = [1 2; 3 4]; B = 2*ones(2);
>> A==B
ans =
     0  1
     0  0

>> A>2
ans =
    0  0
    1  1
```

Um zu testen, ob zwei Matrizen A und B gleich sind, kann der Ausdruck isequal(A,B) verwendet werden. Die Funktion isequal is eine der vielen nützlichen logischen Funktionen, deren Namen mit is beginnt. Für eine Liste aller dieser Funktionen rufen sie in MATLAB doc is auf. Die Funktion isnan ist besonders wichtig, da der Test x == NaN immer das Ergebnis 0 (falsch) liefert, selbst wenn x = NaN! (Ein NaN ist gerade so definiert, dass er zu allem ungleich und ungeordnet ist).

2.2 Logische Operatoren

Die logischen Operatoren von MATLAB sind

&	logisches und
|	logisches oder
~	logisches nicht
xor	logisches exklusives oder
all	wahr wenn *alle Elemente* eines Vektors von Null verschieden sind
any	wahr wenn *wenigstens ein Element* eines Vektors von Null verschieden ist

Tabelle 2.2: Die logischen Operatoren in MATLAB

Wie die relationalen Operatoren produzieren &,| und ~ Matrizen von Nullen und Einsen, falls eines der Argumente eine Matrix ist. Die Funktion all gibt, wenn sie auf einen Vektor angewendet wird, 1 zurück, falls alle Elemente des Vektors verschieden von 0 sind, und 0 sonst. Die any Funktion ist ebenfalls so definiert, wobei 'irgendeiner' hier 'alle' ersetzt. Beispiele:

```
>> x = [-1 1 1]; y = [1 2 -3];
>> x>0 & y>0
ans =
    0   1   0

>> x>0 | y>0
ans =
    1   1   1

>> xor(x>0,y>0)
ans =
    1   0   1

>> any(x>0)
ans =
    1

>> all(x>0)
ans =
    0
```

Beachte, dass xor als Funktion xor(a,b) aufgerufen werden muss. Die Operatoren and, or, not und die relationalen Operatoren können ebenfalls in funktionaler Form aufgerufen werden, and(a,b) .
Die Funktionen all und any sind Vektorfunktionen (vgl. Abschnitt 1.7.3), deshalb gibt all auf Matrizen angewandt einen Zeilenvektor zurück, der die Ergebnisse von all angewendet auf die Spaltenvektoren enthält.

2.3 Flusskontrolle

MATLAB hat vier Strukturen für die Flußkontrolle: die if-Abfrage, die for-Schleife, die while-Schleife und den switch-Befehl. Die einfachste Form der if-Abfrage lautet:

```
if expression
    statements
end
```

statements werden ausgeführt, wenn die Auswertung von expression wahr ergibt (die Realteile der Elemente von expression alle verschieden von 0 sind). Der folgende Code ersetzt zum Beispiel x und y wenn x größer ist als y:

```
if x > y
  temp = y;
  y = x;
  x = temp;
end
```

Folgen auf einen if-Befehl auf der gleichen Zeile weitere Befehle, so müssen diese durch ein Komma getrennt werden, um das if-Kommando vom nächsten Befehl zu trennen.

```
>> if x > 0, sqrt(x);end
```

Befehle die nur ausgeführt werden sollen, wenn expression falsch ist, können nach else eingefügt werden.

```
>> e = exp(1);
>>if 2^e > e^2
    disp('2^e is bigger')
  else
    disp('e^2 is bigger')
  end
```

Schließlich kann mit elseif ein weiterer Test hinzugefügt werden mit (beachte, dass zwischen else und if kein Leerzeichen stehen darf):

```
if isnan(x)
  disp('Not a Number')
elseif isinf(x)
  disp('Plus or minus infinity')
else
  disp('A ''regular'' floating point number')
end
```

Bei einer if-Abfrage der Form if Bedingung 1 & Bedingung 2 wird Bedingung 2 nicht ausgewertet, wenn Bedingung 1 falsch ist (dies wird "early return" if Auswertung genannt). Dies ist nützlich, wenn die Auswertung von Bedingung 2 ansonsten einen Fehler produzieren könnte, zum Beispiel durch einen Index außerhalb des zulässigen Bereichs oder eine nicht definierte Variable.

Die for-Schleife ist einerseits eine der wichtigsten Strukturen in MATLAB, andererseits vermeiden viele Programmierer, die kurzen und schnellen Code produzieren wollen, dessen Verwendung. Die Syntax lautet:

```
for variable = ausdruck
      statements
end
```

Normalerweise ist ausdruck ein Vektor der Form i:s:j. Die Befehle werden für jedes Element von ausdruck ausgeführt, wobei variable dem entsprechenden Element von ausdruck zugeordnet ist. Zum Beispiel wird die Summe der ersten 25 Elemente der harmonischen Folge 1/i erzeugt durch:

```
>> s = 0;
>> for i=1:25, s=s+1/i; end, s
s =
    3.8160
```

Eine weitere Möglichkeit, um ausdruck zu definieren, ist, die Klammernotation zu verwenden.

```
>> for x = [pi/6 pi/4 pi/3], disp([x,sin(x)]), end
    0.5236    0.5000

    0.7854    0.7071

    1.0472    0.8660
```

Mehrere for-Schleifen können verschachtelt werden. In diesem Fall hilft einrücken, um die Lesbarkeit des Codes zu erhöhen. Der folgende Code bildet die symmetrische 5x5-Matrix $A = (a_{ij})$ mit $a_{ij} = i/j$ für j>= i:

```
>> n = 5; A = eye(n);
>> for j=2:n
     for i = 1:j-1
       A(i,j) = i/j;
       A(j,i) = i/j;
     end
end
```

Die while-Schleife hat die Form

```
while ausdruck
     statements
end
```

Die Befehle werden ausgeführt, so lange ausdruck wahr ist. Das folgende Beispiel nähert die kleinste von Null verschiedene Gleitkommazahl an:

```
x=1;
while x>0
  xmin = x;
  x = x/2;
end
xmin

xmin =
4.9407e-324
```

Eine while-Schleife kann mit dem Befehl break beendet werden, der die Kontrolle an den ersten Befehl nach dem entsprechenden end zurückgibt. Eine unendliche Schleife kann mit while true, ..., end erzeugt werden. Dies kann nützlich sein, wenn es ungünstig ist, den Abbruchtest an den Anfang der Schleife zu setzen. (Merke, dass MATLAB im Gegensatz zu manch anderen Sprachen keine ”repeat-until” Schleife hat.)

Das vorige Beispiel lässt sich damit auch folgendermaßen notieren:

```
x = 1;
while true
  xmin = x; x = x/2;
  if x == 0, break, end
end
xmin
```

Der Befehl break kann auch verwendet werden, um eine for-Schleife zu verlassen. In einer verschachtelten Schleife führt ein break zum Verlassen der Schleife auf der nächsthöheren Ebene. Der Befehl continue setzt die Ausführung sofort in der nächsten Iteration der for- oder while-Schleife fort, ohne die verbleibenden Befehle in der Schleife auszuführen.

```
for i=1:10
  if i < 5, continue, end
  disp(i)
end
```

In komplexeren Schleifen kann continue verwendet werden um umfangreiche if-Abfragen zu vermeiden.

Zuletzt bleibt noch der Kontrollbefehl switch. Er besteht aus ”switch Ausdruck”, gefolgt von einer Liste von ”case Ausdruck Befehl”, die optional mit ”otherwise Ausdruck” enden und mit einem end beendet werden. Der Ausdruck bei switch wird ausgewertet und die Befehle nach dem ersten passenden case Ausdruck werden ausgeführt. Falls keiner der Fälle passt, werden die Befehle nach otherwise ausgeführt. Das folgende Beispiel wertet die p-Norm eines Vektors x aus (also norm(x,p)):

```
switch p
  case 1
    y = sum(abs(x));
  case 2
    y = sqrt(x'*x);
  case inf
    y = max(abs(x));
  otherwise
    error('p must be 1, 2 or inf.')
end
```

Der Ausdruck nach case kann eine Liste von Werten sein, die in geschweiften Klammern eingeschlossen ist (ein Cell-Array). In diesem Fall kann der switch-Ausdruck zu jedem Element der Liste passen.

```
x = input('Enter a real number: ');
switch x
  case {inf,-inf}
    disp('Plus or minus infinity')
  case 0
    disp('Zero')
  otherwise
    disp('Nonzero and finite')
end
```

C Programmierer seien darauf hingewiesen, dass das switch-Konstrukt von MATLAB sich anders verhält als das von C: Sobald ein MATLAB case Ausdruck zur Variable passt und die Befehle ausgeführt wurden, wird die Kontrolle an den ersten Befehl nach dem switch-Block übergeben, ohne dass weitere break Befehle nötig sind.

Kapitel 3

M-Dateien

Viele nützliche Berechnungen lassen sich über die Kommandozeile von MATLAB durchführen. Nichtsdestotrotz wird man früher oder später M-Dateien schreiben müssen. Diese sind das Pendant zu Programmen, Funktionen, Subroutinen und Prozeduren in anderen Programmiersprachen. Fügt man eine Folge von Befehlen zu einer M-Datei zusammen, ergeben sich vielfältige Möglichkeiten, wie etwa

- an einem Algorithmus herum zu experimentieren, indem man eine Datei bearbeitet, anstatt eine lange Liste von Befehlen wieder und wieder zu tippen
- einen dauerhaften Beleg für ein numerisches Experiment schaffen
- nützliche Funktionen aufzubauen die zu einem späteren Zeitpunkt erneut verwendet werden können
- M-Dateien mit anderen Kollegen austauschen

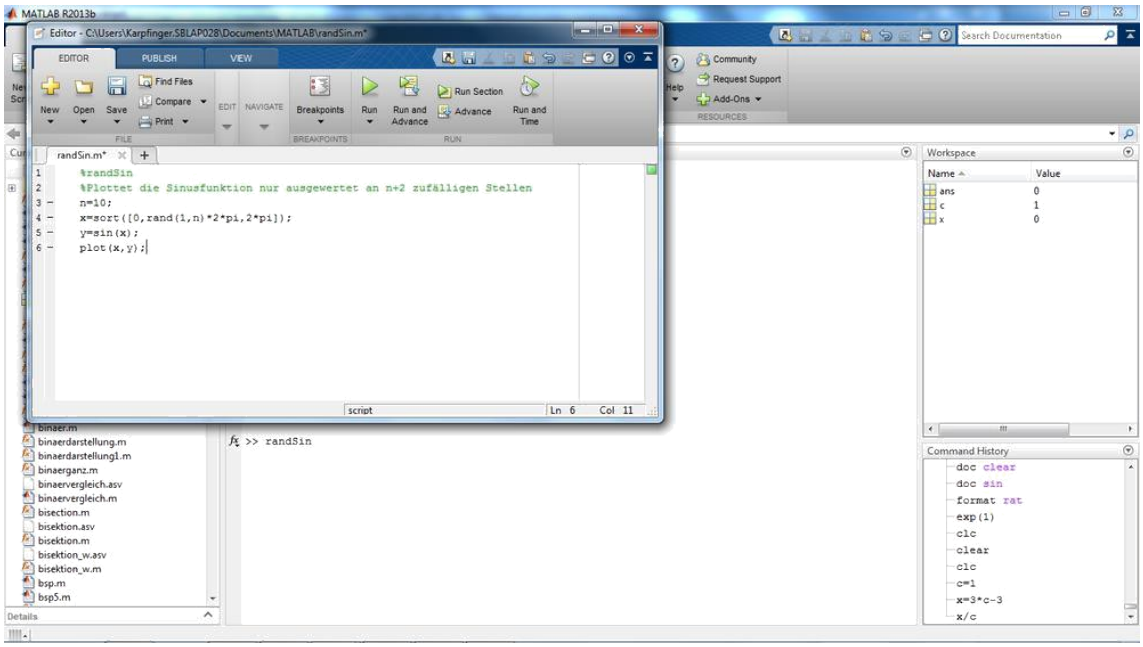
Im Internet findet man eine Vielzahl nützlicher M-Dateien, die von Benutzern geschrieben wurden. Eine M-Datei ist eine Textdatei mit der Dateiendung .m, die MATLAB Befehle enthält. Es gibt zwei Arten:

- **Skriptdateien** (oder Kommandodateien) haben keine Ein- oder Ausgabeargumente und operieren auf Variablen im Workspace.
- **Funktionsdateien** enthalten eine function Definitionszeile und akzeptieren Eingabeargumente und geben Ausgabeargumente zurück, und ihre internen Variable sind lokal auf die Funktion beschränkt (sofern sie nicht als global deklariert wurden).

3.1 Skriptdateien

Ein Skript sammelt eine Folge von Befehlen, die wiederholt verwendet werden sollen oder in Zukunft noch gebraucht werden. Ein Beispiel für ein Skript ist folgender „Sinus random plotter“:

```
% randSin
% Plottet die Sinusfunktion nur ausgewertet an n+2 zufälligen
% Stellen
n=10;
x=sort([0,rand(1,n)*2*pi,2*pi]);
y=sin(x);
plot(x,y);
```



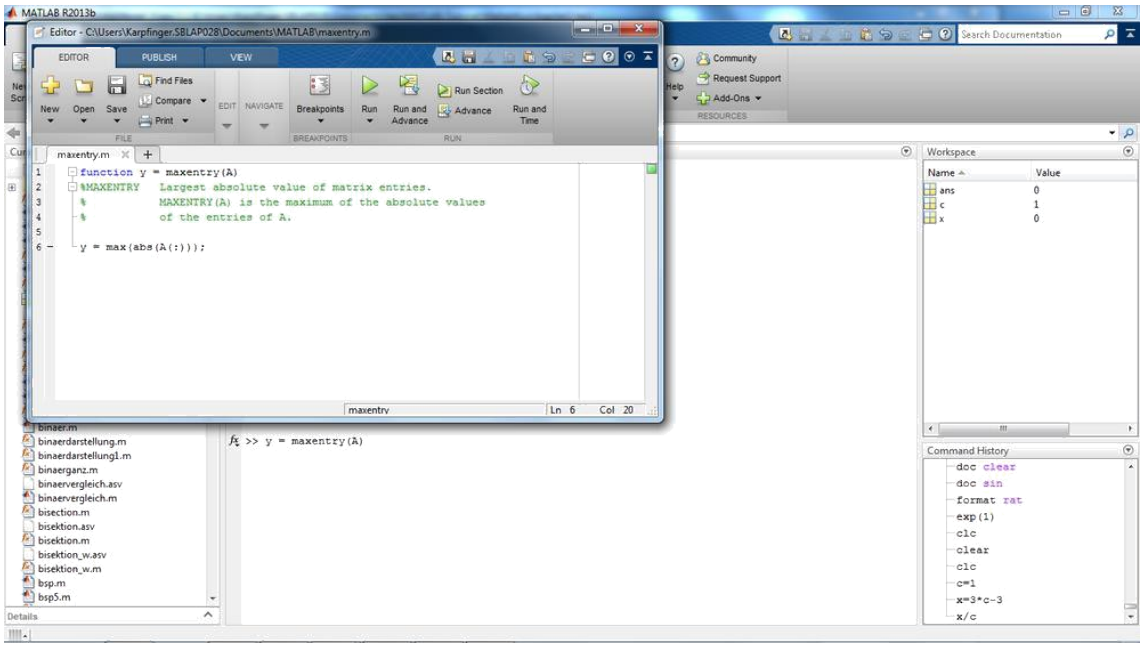
Die ersten beiden Zeilen des Skripts beginnen mit dem % Symbol und sind daher Kommentare. Sobald MATLAB auf ein % trifft ignoriert es den Rest der Zeile. Dies erlaubt das Einfügen von Text, der das Skript für Menschen leichter verständlich macht.
Angenommen dieses Skript wurde unter dem Namen randSin.m gespeichert, dann ist die Eingabe von randSin an der Kommandozeile gleichwertig zur Eingabe der einzelnen Zeilen des Skripts.

3.2 Funktionsdateien

Selbst geschriebene Funktionsdateien erweitern den Umfang von MATLAB. Sie werden auf die gleiche Weise verwendet wie die bereits existierenden MATLAB Funktionen wie sin, eye, size usw.
Hier ist ein Beispiel für eine Funktionsdatei:

```
function y = maxentry(A)
%MAXENTRY   Largest absolute value of matrix entries.
%           MAXENTRY(A) is the maximum of the absolute values
%           of the entries of A.

y = max(abs(A(:)));
```



Dieses Beispiel präsentiert mehrere Features. Die erste Zeile fängt mit dem Schlüsselwort function an, gefolgt vom Ausgabeargument y, und dem Gleichheitszeichen. Rechts von = kommt der Funktionsname maxentry, gefolgt vom Eingabeargument A in Klammern. (Im Allgemeinen kann es beliebig viele Ein-und Ausgabeargumente geben.) Der Funktionsname muss der gleiche sein wie der Name der M-Datei, die die Funktion enthält - in diesem Fall muss die Datei maxentry heißen.

Die zweite Zeile der Funktionsdatei heißt H1-Zeile (help 1). Sie sollte eine Kommentarzeile sein: Eine Zeile die mit einem %-Zeichen beginnt und danach ohne Leerzeichen dem Funktionsname in Großbuchstaben – gefolgt von einem oder mehr Leerzeichen und dann einer kurzen Beschreibung. Die Beschreibung sollte mit einem Großbuchstaben anfangen und mit einem Punkt enden und dabei auf die Worte „der“, „die“, „das“ und „ein“, „eine“ verzichten. Alle Kommandozeilen – beginnend mit der ersten bis zur ersten Nichtkommentarzeile (in der Regel eine Leerzeile, um die Lesbarkeit des Codes zu erhöhen) – werden beim Aufruf von help function_name angezeigt. Darum sollten diese Zeilen die Funktion und ihre Argumente beschreiben. Funktionenname und -argumente groß zu schreiben ist eine allgemein akzeptierte Konvention. Im Falle von maxentry sieht die Ausgabe folgendermaßen aus

```
>> help maxentry

MAXENTRY    Largest absolute value of matrix entries.
            MAXENTRY(A) is the maximum of the absolute values
            of the entries of A.
```

An dieser Stelle sei noch einmal mit Nachdruck darauf hingewiesen, dass es sich lohnt, alle Funktionsdateien auf diese Art und Weise zu dokumentieren. Oft ist es nützlich, in

Kommentaren anzugeben, wann die Funktion zuerst geschrieben wurde, und ob Veränderungen hinzugefügt wurden.
Der Befehl help funktioniert auf ähnliche Art und Weise mit Skriptdateien, er zeigt die Anfangsfolge an Kommentaren an.

Die Funktion maxentry wird wie jede andere MATLAB Funktion aufgerufen:

```
>> maxentry(1:10)
   ans =
       10

>> maxentry(magic(4))
   ans =
       16
```