Einführung in Matlab
Teil I

Glad Mihai

Universität Greifswald
Funktionelle Bildgebung
Januar 2012

Inhaltsverzeichnis

1 Was ist Matlab?

Laut Wikipedia ist Matlab:

> "eine kommerzielle Software zur Lösung mathematischer Probleme und zur grafischen Darstellung der Ergebnisse. Matlab ist primär für numerische Berechnungen mithilfe von Matrizen ausgelegt, woher sich auch der Name ableitet: MATrix LABoratory."

MATLAB ist ein numerisches Rechenprogramm, das Anwendung in Industrie, Forschung und Lehre findet. Es kann Daten erfassen, bearbeiten und darstellen, ist sehr flexibel und hat eine vielzahl von eingebauten Funktionen für Bereiche von Mathematik und Statistik bis zu Computational Finance oder Biologie.

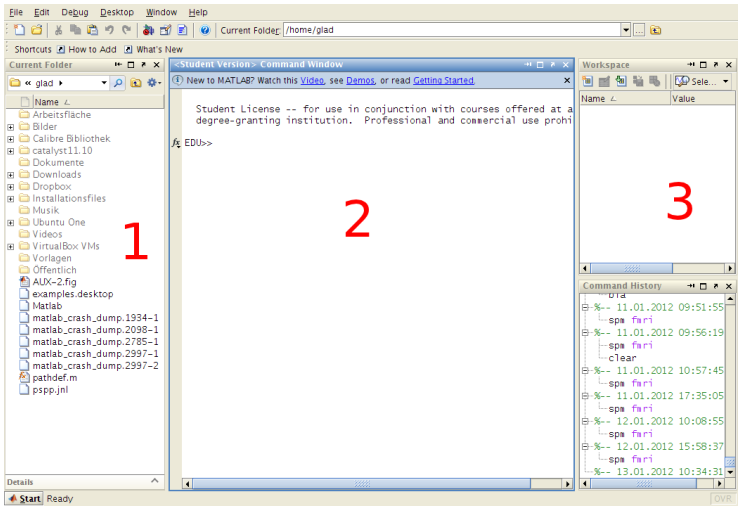

Abbildung 1: Benutzeroberfläche von Matlab. (1) Aktueller Ordner, (2) Kommandofenster, (3) Workspace.

Als Programmiersprache ist Matlab übersichtlich, einfach strukturiert und besitzt ganz viele vorprogrammierte Befehle.

2 Benutzeroberfläche

Nachdem man Matlab startet, bekommt man eine graphische Oberfläche. Das Wichtigste ist das Kommandofenster (Abbildung 1 (2)). Hier gibt man die Befehle ein. Außerdem ist der aktuelle Ordner angezeigt (Abbildung 1 (1)) und die im Speicher vorhandenen Variablen (Abbildung 1 (3), Workspace). Man kann sich die Oberfläche individuell einrichten. In der Mitte oben ist der *Current Folder* (aktuelle Ordner) angezeigt. Hier kann man den Ordner, in dem man arbeiten möchte, wählen.

3 Eingabe

Im Kommandofenster werden direkt nach dem Doppelpfeil (`>>` oder `EDU>>` für die Studentenversion) die Befehle eingegeben. Diese Stelle im Kommandofenster wird auch Prompt genannt.

3.1 Matlab als Taschenrechner

Einfache Rechenaufgaben kann Matlab mit den einfachen mathematischen Operatoren ($+, -, *, /$, etc.) ausführen. Jeder Eingegebene Befehl wird mit der `Return`-Taste ausgeführt:

```
EDU>> 2+3
ans =
     5
```

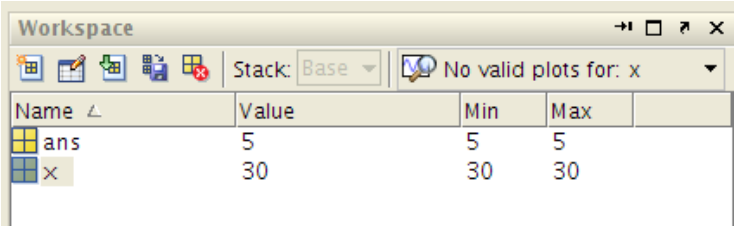

Abbildung 2: Variablen im Workspace

Matlab benutzt die Variable `ans` um das Ergebnis zu speichern,, falls keine andere Variable angegeben wird. Man kann eine Variable direkt benennen, und mit dem Semikolon ‘`;`’ wird die Ausgabe im Kommandofenster unterdrückt:

```
EDU>> x = 2*15;
```

Variablen dürfen keine Sonderzeichen außer dem Unterstrich enthalten und dürfen nur mit Buchstaben anfangen. Beide Variablen (`ans`, `x`) sind im Workspacebereich zu finden. Hier kann man eine Variable auswählen und mittels Doppelklick öffnen.

Eine Variable kann man aufrufen und man kann damit weiter rechnen:

```
EDU>> x
x =
    30
EDU>> y = x/1.5
y =
    20
```

Matlab achtet auf Groß- und Kleinbuchstaben (case sensitive):

```
EDU>> X = x^2 + y - 200
X =
   720
```

In einer Zeile können mehrere Operationen eingegeben werden. Diese können entweder mit Semikolon (Unterdrückung der Ausgabe) oder Komma (keine Ausgabenunterdrückung) getrennt werden:

```
EDU>> a = (3*x+2)^3; b = X/x, c = a - X
b =
    24


c =
      777968
```

Im Workspace erscheinen alle im Speicher vorhandenen Variablen, auch wenn deren Ausgabe im Kommandofenster unterdrückt wurden. Mit dem Befehl `who` werden alle Variablen angezeigt:

```
EDU>> who
Your variables are:
X    a    ans  b    c    x    y
```

Übungen

1. Was gibt Matlab nach Verarbeitung der folgenden Anweisungen aus? Erst überlegen, dann überprüfen:

```
m = 23; n = -20
(m^2+2)/2
a = b - m
a = pi
b = 2pi
```

2. Welche Variablen akzeptiert Matlab?

```
greifswald=1, stralsund+ruegen=2, 3_lubmin=3, atomkraft_666=0
```

3.2 Dateien speichern, laden; Navigation

Daten können mit dem Befehl `save`, gefolgt vom gewünschten Dateinamen und den Variablen in denen die Daten enthalten sind gespeichert werden:

```
EDU>> save datei X a
```

Hier wird die Datei mit dem Namen `datei.mat` erzeugt, in der die Variablen `X` und `a` gespeichert werden. Lässt man die Variablen nach dem Dateinamen weg, so wird der gesamte Workspace in der Datei gespeichert. Um bestimmte Variablen aus dem Arbeitsspeicher zu löschen benutzt man den Befehl `clear` gefolgt von der zu löschenden Variable:

```
EDU>> clear X
EDU>> who
Your variables are:
a    ans  b    c    x    y
```

Alle Variablen im Workspace können mit dem Befehl `clear` gelöscht werden. `.mat` Dateien können auch eingelesen werden:

```
EDU>> clear
EDU>> load datei
EDU>> who
Your variables are:
X  a
```

Kommandozeilennavigation in Matlab folgt der Unix-Syntax. Das aktuelle Verzeichnis wird mit `pwd` (print working directory) angegeben:

```
EDU>> pwd
ans =
/home/glad
```

Mit '`cd ..`' gelangt man in ein Verzeichnis höher, und mit `ls` wird der Inhalt eines Verzeichnisses angezeigt. Um in ein Verzeichnis zu wechseln benutzt man `cd` *Verzeichnis*.

Um Kommentare einzufügen die von Matlab nicht interpretiert werden, muss man ein Prozent `%` Zeichen setzen. Alles was danach kommt wird von Matlab ignoriert:

```
EDU>> Antwort_1 = X*a   % multipliziere X mit a
Antwort_1 =
   560655360
```

Übungen

1. Was gibt Matlab aus?

   ```
   Antwort_1 = X*a   multipliziere X mit a
   ```

2. Navigiere mit Matlab in das `C:\` Verzeichnis und zeige dessen Inhalt. Navigiere danach zurück in den Arbeitsordner. Tip: nach `cd` *TAB* drücken.

4 Vektoren und Matrizen

Alle Variablen in Matlab werden als Matrizen interpretiert. Eine Skalare wird als 1x1 Matrix dargestellt. Matrizen werden in eckigen Klammern geschrieben. Zeileneinträge werden durch Leerzeichen oder Kommas getrennt, und jede Zeile wird durch ein Semikolon getrennt. Z.B. wird folgende Matrix:

$$A = \begin{pmatrix} 1 & 5 & -20 \\ -23 & 0 & 9 \end{pmatrix}$$

folgendermaßen in Matlab eingegeben:

```
EDU>> A = [1 5 -20; -23 0 9]
A =
     1     5   -20
   -23     0     9
```

So geht es auch: `>> A = [1, 5, -20; -23, 0, 9]`. Die Dimension, d.h. die Anzahl der Zeilen und Spalten einer Matrix erhält man durch:

```
EDU>> size(A)
ans =
     2     3
```

Um die Länge der Matrix anzuzeigen benutzt man `length()`.
Ein Vektor wird als $n \times 1$ oder $1 \times n$ Matrix angegeben:

```
EDU>> a = [2; 3; 18], b = [99 -98 10001], c = 666
a =
     2
     3
    18

b =
          99         -98       10001

c =
   666
```

`whos` verschafft uns einen besseren Überblick über die Variablen mit Name, Größe der Matrix, Speichergröße und Klasse:

```
EDU>> whos
  Name      Size             Bytes  Class     Attributes

  a         3x1                 24  double
  b         1x3                 24  double
  c         1x1                  8  double
```

Meistens arbeitet man mit einer großen Datenmenge, und ein Vektor oder eine Matrix enthält tausende von Einträgen. Wenn man alle Einträge in einer Matrix der selben Rechenoperation unterwerfen möchte, z.B. die Wurzel ziehen, ist die Syntax besonders einfach.

```
EDU>> a = [1 3 5 7 9];
EDU>> b = sqrt(a)
b =
    1.0000    1.7321    2.2361    2.6458    3.0000
```

Ganz nützlich ist folgende Abkürzung, wenn man einen Vektor mit Werten die den gleichen Abstand voneinander haben erzeugen möchte: `a = 1:2:9`, d.h. fange mit 1 an und zähle 2 zum nächsten Eintrag dazu, usw. bis die Grenze von 9 erreicht wird.

Will man jedes Element in der Matrix `a` quadrieren so stoßt man auf ein Problem:

```
EDU>> d = a^2
??? Error using ==> mpower
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
```

Um elementenweise zu quadrieren muss man einen Punkt vor dem Operator setzen:

```
EDU>> d = a.^2
d =
     1     9    25    49    81
```

Dies gilt auch für Matrizen:

```
EDU>> A = [a;d]
A =
     1     3     5     7     9
     1     9    25    49    81

EDU>> A.^2
ans =
           1           9          25          49          81
           1          81         625        2401        6561
```

Vektoren und Matrizen kann man mit einem Apostroph transponieren:

```
EDU>> a_trans = a'
a_trans =
     1
     3
     5
     7
     9

EDU>> A_trans = A'
A_trans =
     1     1
     3     9
     5    25
     7    49
     9    81
```

Eine Matrix kann mit einem Skalar addiert oder multipliziert werden:

```
EDU>> a + 4  % Vektor + Skalar
ans =
     5     7     9    11    13

EDU>> a * 4  % Vektor * Skalar
ans =
     4    12    20    28    36
```

Um Matrizen oder Vektoren zu addieren müssen diese die gleiche Größe oder *Dimension* haben:

```
EDU>> a + [1 2 3]
??? Error using ==> plus
Matrix dimensions must agree.

EDU>> size(a)
ans =
     1     5
```

Diese Operation geht nicht, da die Dimension von $a = 1 \times 5$ und die Dimension von dem anderen Vektor 1×3 ist.

Um zwei Matrizen miteinander zu multiplizieren, müssen die inneren Matrix Dimensionen übereinstimmen (z.B. eine 2×3 mit einer 3×2 Matrix kann man multiplizieren, hier ist die innere Dimension 3):

```
EDU>> B = [34 25; 66 -12]
B =
    34    25
    66   -12

EDU>> A*B
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

```
EDU>> C = [55 66; 88 -99; 23 32; -36 -69; 64 21]
C =
    55    66
    88   -99
    23    32
   -36   -69
    64    21

EDU>> A*C
ans =
         758        -365
        4842       -1705
```

Um auf einen bestimmten Wert aus einer Matrix zuzugreifen, muss man die Indizes angeben. Will man den zweiten Wert aus dem Vektor a herausnehmen:

```
EDU>> a(2)
ans =
     3
```

Für Matrizen ist eine n-dimensionale Koordinate notwendig, z.B. will man den Wert aus der mittleren Zeile und Spalte einer 3×3 Matrix heraussuchen, so gibt man folgendes ein:

```
EDU>> e = [1 2 3; 4 5 6; 7 8 9];
EDU>> e(2,2)
ans =
     5
```

Möchte man die ganze erste Zeile herausnehmen:

```
e(1,:)
ans =
     1     2     3
```

Übersetzt: Suche in der ersten Zeile alle Einträge aus.

Wichtig! In Matlab fängt die Nummerierung der Einträge bei 1 an, und nicht bei 0 wie in anderen Programmiersprachen wie C, oder Java.

```
EDU>> e(0)
??? Subscript indices must either be real positive integers or
logicals.
```

Übungen

1. Mit welchem Befehl kann man alle Einträge aus der zweiten Spalte der Matrix `e` herausnehmen? In der dritten Spalte den zweiten und dritten Eintrag?

2. Erzeuge einen Vektor `q` aus der Matrix `e`: Zeile 1, den 1. Eintrag, Zeile 2 den 2. Eintrag und Zeile 3 den 3. Eintrag.

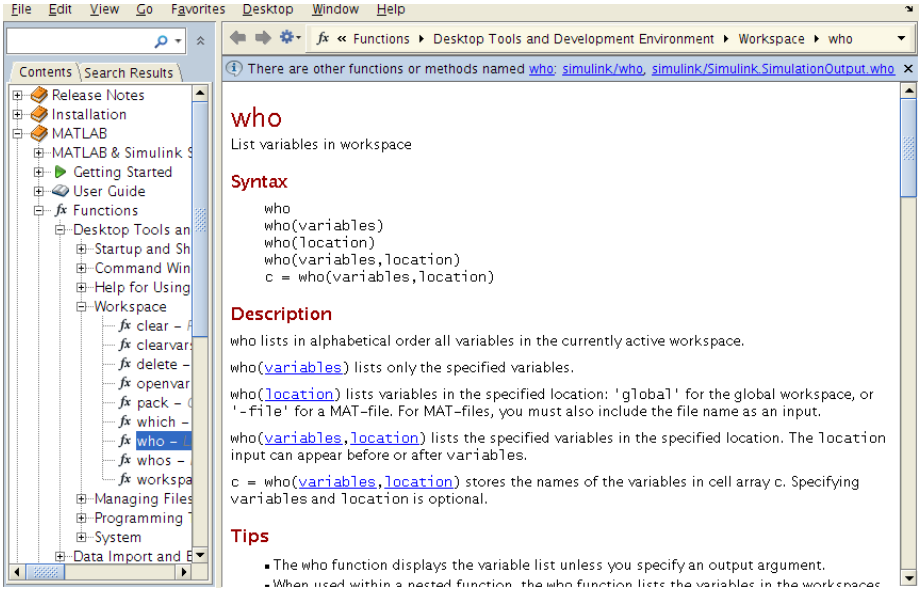

Abbildung 3: Matlab Helpdesk bietet eine ausführliche Hilfe über Matlabfunktionen.

5 Matlab Hilfe

Die Matlabhilfe enthält eine ausführliche Erklärung der Matlabfunktionen, zusammen mit leicht verständlichen Beispielen. Man kann in der Kommandozeile die Kurzbeschreibung eines Befehls mit `help` gefolgt mit dem gewünschten Befehlsnamen aufrufen, z.B. `help who`. Eine ausführliche hilfe bekommt man durch den Helpdesk: `doc who`. Ein neues Fenster wird direkt bei dem Befehlseintrag geöffnet (Abbildung 3). Helpdesk kann man direkt mit dem Befehl `helpdesk` aufrufen.

Übung

1. Berechne den natürlichen Logarithmus eines Vektors der Einträge von 0 bis 10 in einser Schritten enthält.

6 Grafiken

Matlab kann Ergebnisse schnell und flexibel veranschaulichen. Mit dem Befehl `plot(x,y)` wird ein Grafikfenster erzeugt. In diesem Grafikfenster wird die Grafik geplottet. Jedes Grafikfenster bekommt einen eigenen Namen. Um mehrere Grafiken in verschiedene Fenster zu erzeugen, verwendet man `figure(`*Ganzzahl*`)`.

Als Beispiel plottet man den Exponenten des Vektors x. In einem zweiten Fenster plottet man das Quadrat von x:

```
EDU>> x = [1:.05:3];
EDU>> figure(1),plot(x,exp(x))
EDU>> figure(2),plot(x,x.^2)
```

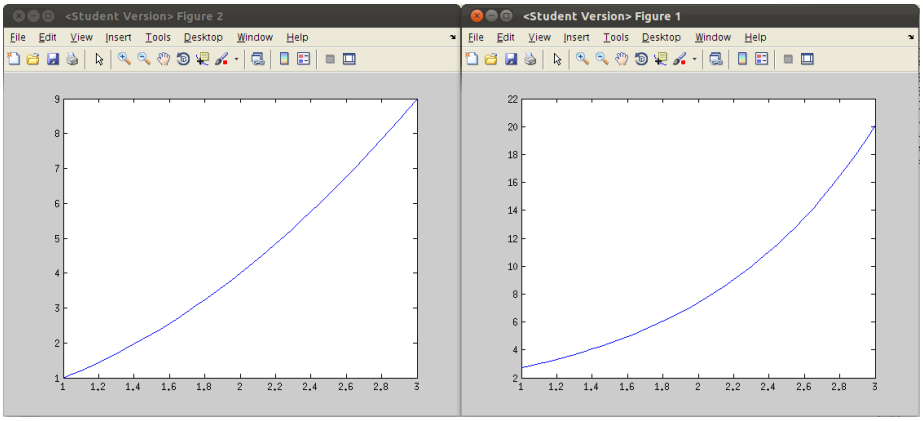

Abbildung 4: Plots von $y = x^2$ *und* $y = e^x$.

Jetzt haben wir zwei Grafikfenster (Figure 1, Figure 2), mit jeweils zwei unterschiedlichen Plots. Die x-Achse ist bei beiden gleich, jedoch ist die y-Achse verschieden (siehe Abbildung 4).

Eine Beschriftung des Plots und der Achsen ist hier sehr hilfreich:

```
EDU>> figure(1),title('y = exp(x)'),xlabel('x'),ylabel('exp(x)')
EDU>> figure(2),title('y = x^2'),xlabel('x'),ylabel('x^2')
```

Beide Grafiken kann man auch in einem Fenster mit Hilfe von `subplot(m,n,p)` plotten, siehe Abbildung 5.

```
EDU>> figure(3),subplot(2,1,1),plot(x,exp(x)),subplot(2,1,2),plot(x,x.^2)
```

Es ist sehr aufwendig jeden Befehl direkt im Kommandofenster einzugeben. Dafür kann man `m`-Files benutzen. Um so ein `m`-File zu erzeugen kann man entweder *Strg+N* drücken, oder den Befehl `edit` in der Kommandozeile eingeben. Der Editor ist in Abbildung 6 dargestellt.

Gebt folgende Befehle im Editor ein und speichert die Datei:

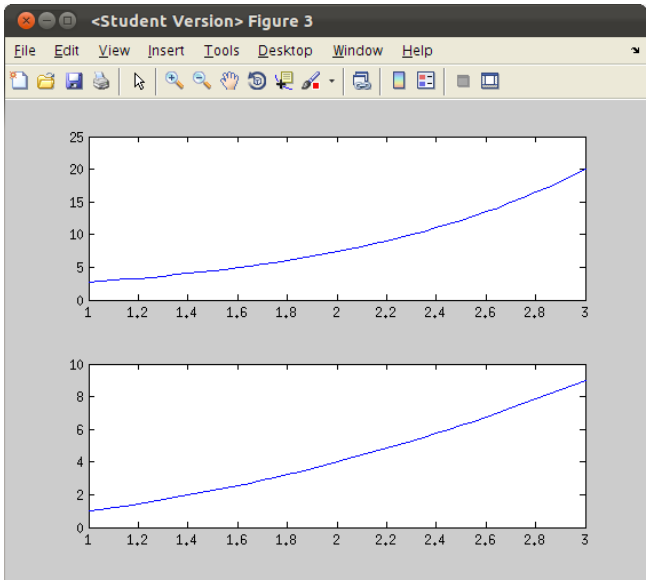

Abbildung 5: Mehrere Grafiken ein einem Fenster.

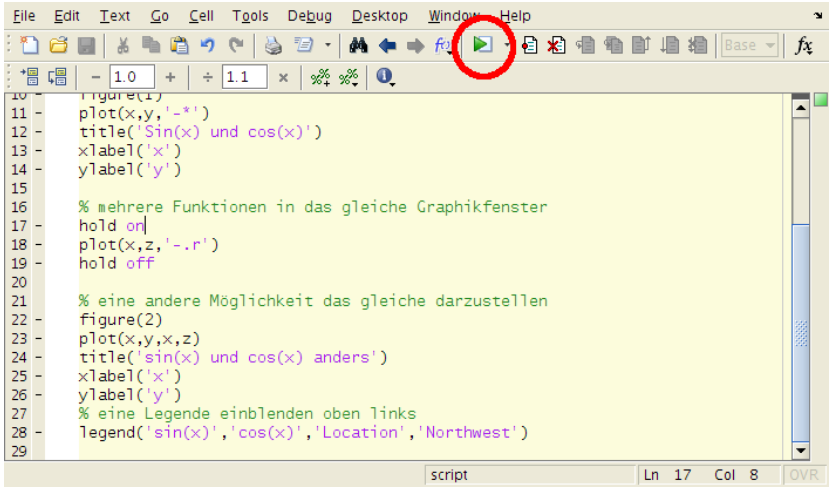

Abbildung 6: M-file Editor. Mit dem Run Knopf (roter Kreis) oder mit der F5 Taste kann man das m-File ausführen.

```
%% M-file Beispiel mit Grafiken

clear   % Workspace befreien

%% Variablen definieren
x = linspace(-pi,pi,30);
y = sin(x);
z = cos(x);

%% plotten
figure(1)
plot(x,y,'-*')
title('Sin(x) und cos(x)')
xlabel('x')
ylabel('y')

% mehrere Funktionen in das gleiche Grafikfenster
hold on
plot(x,z,'-.r')
hold off

% eine andere Möglichkeit das gleiche darzustellen
figure(2)
plot(x,y,x,z)
title('sin(x) und cos(x) anders')
xlabel('x')
ylabel('y')
% eine Legende einblenden oben links
legend('sin(x)','cos(x)','Location','Northwest')
```

Die Datei speichern. Um das `m`-File auszuführen kann man entweder den *Run* Knopf klicken oder die *F5* Taste benutzen. Möchte man nur bestimme Zeilen ausführen, so markiert man diese mit der Maus und drückt die *F9* Taste.

7 Kontrollstrukturen

Schleifen und Verzweigungen sind wichtige Bestandteile einer Programmier- bzw. Skriptsprache. Hier werden `for`- und `while`-Schleifen sowie die `if-else`-Verzweigung diskutiert.

7.1 Die `for` Schleife

Die `for`-Schleife ist für wiederholte Berechnungen geeignet. Eine `for`-Schleife wird durch folgende Syntax beschrieben:

```
for Index = Werte
   Anweisungen
end
```

Index bezieht sich auf einen Zähler. Z.B.: `index = 1:n`, die Schleife wird `n`-Mal durchgeführt.

Als Beispiel haben wir folgende Berechnung:

$$a_{x,y} = \frac{2 \cdot x + 3}{y}$$

für $x = 1 \ldots n$ und $y = 1 \ldots n$. Mit einer `for`-Schleife kann man es folgendermaßen als `for_schleife.m` programmieren:

```
%% For-Schleife zur Berechnung von a = (2x+3)/y

anzahl = 20;        % Anzahl Berechnungen

% Matrix initialisieren, damit die Schleife effizienter durchläuft
a = zeros(anzahl,anzahl);

%% for-Schleife für den Aufbau der Matrix
for x = 1:anzahl
    for y = 1:anzahl
        a(x,y) = (2*x+3)/y;
    end
end
```

Es ist zwar sehr einfach mit Schleifen zu arbeiten, jedoch nicht effizient. Matlab bevorzugt Berechnungen mit Vektoren oder Matrizen, da es diese parallel ausführen kann. Um dies zu verdeutlichen führen wir die gleiche Berechnung ohne `for`-Schleife mit Hilfe von Matrixoperationen durch. Mit den Befehlen `tic` und `toc` wird die Dauer der Ausführung angezeigt.

Damit die Matrix Multiplikation zugänglicher ist, wird die Umwandlung kurz erklärt. Die Berechnung in der Schleife erzeugt folgende Matrix für $x = y = [1, 2, 3]$:

$$B = \begin{pmatrix} \frac{1\cdot 2+3}{1} & \frac{1\cdot 2+3}{2} & \frac{1\cdot 2+3}{3} \\ \frac{2\cdot 2+3}{1} & \frac{2\cdot 2+3}{2} & \frac{2\cdot 2+3}{3} \\ \frac{3\cdot 2+3}{1} & \frac{3\cdot 2+3}{2} & \frac{3\cdot 2+3}{3} \end{pmatrix}$$

Wenn man jetzt alle blauen Zahlen in eine Matrix steckt, diese mit dem Skalar 2 multipliziert und 3 dazu addiert, so haben wir die Operation im Zähler des Bruchs:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix} \cdot 2 + 3 = \begin{pmatrix} 1 \cdot 2 + 3 & 1 \cdot 2 + 3 & 1 \cdot 2 + 3 \\ 2 \cdot 2 + 3 & 2 \cdot 2 + 3 & 2 \cdot 2 + 3 \\ 3 \cdot 2 + 3 & 3 \cdot 2 + 3 & 3 \cdot 2 + 3 \end{pmatrix}.$$

Nun wird elementenweise geteilt, d.h. jedes Element aus der Matrix A wird durch das zugehörige Element aus der Matrix Y geteilt. Deren Inhalt stellen die roten Zahlen dar (Vorsicht, diese Notation ist Matlabspezifisch! In der linearen Algebra gibt es sie nicht.):

$$B = A./\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}.$$

So sehen die Anweisungen in Matlab aus:

```
% Vergleich von for-Schleife und Matrix Multiplikation

clear
clc
%% Variablen
% für Schleife
x = [1:2000];
y = [1:2000];
% für Matrixmultiplikation
X = repmat(x',1,length(x));
Y = repmat(y,length(y),1);

%% Schleife
tic
for i = 1:length(x)
    for j = 1:length(y)
        A(i,j) = (x(i)*2+3)/y(j);
    end
end
toc

%% Matrix Multiplikation
tic
B = (X*2+3)./Y;
toc
```

7.2 Die `while` Schleife

'While' heisst auf englisch solange, und wird hier genau so interpretiert. Solange der Ausdruck nach dem `while` Befehl wahr ist, wird die Schleife ausgeführt.

```
while Ausdruck
   Anweisungen
end
```

Als Beispiel soll der Wert von x angezeigt werden, angefangen von Null, solange $x < 10$ ist. In jedem Durchlauf wird eine 1 zu x addiert.

```
%% While-Schleife: Beispiel
% Der Wert von x wird angezeigt solange x kleiner als 10 ist.

x = 0;
while x < 10
    sprintf('x hat den Wert %d',x)
    x = x + 1;
end
```

7.3 Die `if-else` Verzweigung

Mithilfe der `if-else` Verzweigung können Fallunterscheidungen durchgeführt werden. Folgende Syntax wird benutzt:

`if` *logischer Ausdruck*
    Anweisungen
`elseif` *logischer Ausdruck*
    Anweisungen
`else`
    Anweisungen
`end`

Wenn der erste Ausdruck wahr ist, so wird die Anweisung darunter ausgeführt. Wenn er falsch ist so wird der nächste Ausdruck auf Richtigkeit überprüft und die Anweisung darunter im positiven Fall durchgeführt. Sind die ersten beiden Ausdrücke falsch, so wird nur die letzte Anweisung unter dem Punkt `else` durchgeführt. Man kann beliebig viele `elseif` Zweige benutzen. Fehlen `elseif` und `else` so wird der Ausdruck zur `if`-Verzweigung.

```
%% Beispiel einer if-else Schleife
%{
Es wird nach einer Zahl gefragt. Diese Zahl wird dann eingeordnet
als groesser als 100, 50 oder 10.
%}

% Frage den Benutzer nach einer Zahl
zahl = input('Bitte geben Sie eine positive Zahl ein: ');

%% else-if Schleife
if zahl>100
    disp('Die von Ihnen gewaehlte Zahl ist groesser als 100.');
elseif zahl>50
    disp('Die von Ihnen gewaehlte Zahl ist groesser als 50.');
else
    disp('Die von Ihnen gewaehlte Zahl ist groesser als 10.');
end
```

Was stimmt mit dem Programm nicht?