MATLAB-Einführung[1]
(zu Algorithmische Mathematik und Programmieren bei Dr. Martin Lanser, WS 2019/20)

Dieser Text soll Ihnen bei der Bearbeitung der ersten Programmieraufgaben zur Seite stehen. Es ist sicherlich sinnvoll die gegebenen Beispiele einmal in Ruhe mit MATLAB nachzuvollziehen.

1 Einleitung

Bei MATLAB handelt es sich um ein interaktives Programm zur besonders effizienten Durchführung von Matrixoperationen wie z.B. dem Lösen von Gleichungssystemen.
Wenn man MATLAB gestartet hat, kann man unter **Help** den Punkt **Demos, Beispiele, Examples o.ä.** aufrufen. Dort finden sich Demonstrationen zu den Möglichkeiten, die MATLAB bietet. Weitere Informationen zu jedem Befehl kann man erhalten, indem man **help** ***Funktionsbezeichnung*** eingibt. Bei neueren MATLAB-Versionen ist es auch möglich den Funktionsnamen von MATLAB hinter dem **help** (oder auch ohne help, wenn die Funktion angewendet werden soll) ergänzen zu lassen. Dazu beginnt man mit dem Funktionsnamen und drückt die Tabulatortaste, um den Befehl von MATLAB sinnvoll ergänzen zu lassen. Bei mehreren Alternativen werden diese in einem Kästchen alphabetisch sortiert angegeben und man kann mit dem Cursor den passenden Befehl auswählen und mit der Enter-Taste (hinter dem **help**) einfügen.
"Interaktiv" bedeutet, dass MATLAB jede einzelne Eingabe auszuwerten versucht. Mehrere Anweisungen in einer Zeile können durch Trennung mit einem Komma (,) oder Semikolon (;) eingegeben werden. Ein Semikolon nach einer Anweisung verhindert zudem die Ausgabe der Zwischenergebnisse auf dem Bildschirm. Insgesamt ist es anzuraten in einem M-File neue Befehle in eine neue Zeile zu schreiben, um den Programmtext übersichtlicher zu gestalten.
Eine MATLAB-Eingabe besteht immer aus einer **Anweisung** oder auch aus **Variable = Anweisung**. Liefert die Anweisung ein Ergebnis, dem keine Variable zugeordnet ist, wählt MATLAB immer den Namen **ans** (answer).
MATLAB unterscheidet zwischen Groß- und Kleinschreibung, dass bedeutet A bzw. a sind verschiedene Variablen. Alle eingebauten MATLAB-Funktionen werden klein geschrieben.
Mit dem Befehl **quit** kann MATLAB beendet werden.

[1] Originalautor: Dr. Stefanie Vanis

2 Eingabe von Skalaren, Vektoren und Matrizen

`MATLAB` interpretiert jede Zahl sofort als Matrix, wobei Skalare und Vektoren als Sonderfälle unterschieden werden. Es ist daher immer abhängig vom Kontext bzw. der vorgenommenen Zuweisung, ob eine Variable ein Skalar, ein Vektor oder eine Matrix ist.
Die am häufigsten verwendeten Eingabemöglichkeiten in `MATLAB` sehen dabei wie folgt aus:

- Einen Skalar kann man einfach eingeben:

  $a = 1;$ ohne Semikolon erhält man eine bestätigende Ausgabe von `MATLAB`

- Einen Zeilenvektor erzeugt man, indem man die Werte in eckige Klammern setzt und entweder durch Leerzeichen oder Kommata voneinander trennt:

  $$a = [1 \quad -\mathrm{sqrt}(2) \;\; 3.5] \quad \text{oder} \quad a = [1\,,\, -\mathrm{sqrt}(2)\,,\, 3.5]$$

  erzeugt

  $$a \;=\; \begin{matrix} 1 & -1.4142 & 3.5 \end{matrix} \quad \text{(Zeilenvektor in MATLAB-Darstellung)}$$

- Einen Spaltenvektor erhält man entweder indem man einen Zeilenvektor mittels $'$ transponiert oder indem man die Einträge mit Semikola trennt

  $$a = [1 \quad -\mathrm{sqrt}(2) \;\; 3.5]' \quad \text{oder} \quad a = [1\,;\, -\mathrm{sqrt}(2)\,;\, 3.5]$$

  erzeugt

  $$a \;=\; \begin{matrix} 1 \\ -1.4142 \\ 3.5 \end{matrix}$$

- Dementsprechend erzeugt man eine Matrix, indem man sie zeilenweise eingibt und die Zeilen dann mit Semikola voneinander trennt:

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

  erzeugt

  $$a \;=\; \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

  Man könnte hier auch auf die Kommata zur Trennung der Einträge in den jeweiligen Zeilen verzichten (siehe oben).

- Zahlen, die in wissenschaftlicher Notation eingegeben werden, dürfen keinerlei Leerzeichen enthalten:

  6.12e-9 (nicht 6.12 e -9 oder ähnliches)

- Genauso können auch komplexe Zahlen verwendet werden. Für die komplexe Einheit wird dann ein **i** oder **j** benötigt und es dürfen wieder keine Leerzeichen auftauchen:

  8-2i (nicht 8 -2 i oder ähnliches)

- Matrizen und Vektoren können auch elementweise definiert werden. Dazu gibt man die Nummer des Eintrags an. Dabei ist zu beachten, dass in `MATLAB` die Zählung bei 1 beginnt, abweichend von z.B. C-Programmen, bei denen die Zählung mit 0 beginnt.
  Um also den Eintrag in der k-ten Zeile und der l-ten Spalte der Matrix A auf den Wert 2.2 zu setzten, müßte der folgende Befehl verwendet werden:

  $$A(k, l) = 2.2$$

  Bei Vektoren kann auf die Angabe der Spalte (bei Spaltenvektoren) bzw. der Zeile (bei Zeilenvektoren) verzichtet werden, wenn die Vektorstruktur vorher eindeutig vorgegeben wurde, da in einem solchen Fall nur eine Spalte bzw. Zeile vorhanden ist.

3 Funktionen zum Erzeugen von Matrizen

Es gibt bei `MATLAB` verschiedene Funktionen die bestimmte Matrizen erzeugen. Einige davon sind:

- **rand(n)** bzw. **rand(n,m)**
  erzeugt eine zufällige Matrix der Größe n×n bzw. n×m, bei der die Einträge zwischen 0 und 1 gleichverteilt sind.

- **eye(n)**
  erzeugt eine n×n-Einheitsmatrix.

- **zeros(n)** bzw. **zeros(n,m)**
  erzeugt eine n×n bzw. n×m Nullmatrix.

- **ones(n)** bzw. **ones(n,m)**
  erzeugt eine n×n bzw. n×m Matrix deren Einträge alle 1 sind.

- **diag(v)** (A = diag(v))
  erzeugt eine Matrix, deren Diagonale aus dem Vektor v besteht und die ansonsten mit Nullen aufgefüllt ist.

- **diag(A)** (w = diag(A))
  liefert einen Vektor mit den Einträgen der Diagonalen der Matrix A.

4 Untermatrizen

`MATLAB` erlaubt einen sehr effizienten und lesbaren Umgang mit den Daten sowie Matrixmanipulationen, indem auf Untermatrizen bzw. Einträge von Vektoren und Matrizen direkt zugegriffen werden kann. Dieses Vorgehen ist schneller als eine Formulierung mit Schleifen und sollte deshalb wo immer möglich benutzt werden.
Vektoren mit gleichen Abständen zwischen den Elementen lassen sich verkürzt mit

Startwert : Endwert (Schrittweite = 1) bzw.

Startwert : Schrittweite : Endwert angeben.

Der Befehl $\mathbf{v} = \mathbf{1} : \mathbf{5}$ erzeugt also:

$$v = \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix}$$

Die Zahlen müssen dabei nicht aufsteigend sortiert sein. Der Befehl $w = 1.2 : -0.2 : 0.2$ beispielsweise liefert den folgenden Zeilenvektor

$$w = \begin{matrix} 1.2 & 1.0 & 0.8 & 0.6 & 0.4 & 0.2 \end{matrix}$$

Diese Notation ermöglicht den Zugriff auf relativ beliebige "Untermatrizen" einer Matrix. Zu

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

liefert $\mathbf{A}(\mathbf{1} : \mathbf{2}, \mathbf{3})$ in einem Eintrag die beiden ersten Einträge der dritten Spalte:

$$\text{ans} = \begin{matrix} 3 \\ 6 \end{matrix}$$

$\mathbf{A}(:, \mathbf{3})$ die gesamte dritte Spalte:

$$\text{ans} = \begin{matrix} 3 \\ 6 \\ 9 \end{matrix}$$

$\mathbf{A}(:, [\mathbf{1}, \mathbf{3}])$ liefert die erste und dritte Spalte:

$$\text{ans} = \begin{matrix} 1 & 3 \\ 4 & 6 \\ 7 & 9 \end{matrix}$$

Auch für Zuweisungen ist diese Notation nützlich:
$\mathbf{X(\ :\ ,\ [2\ 4\ 5])\ =\ Y(\ :\ ,\ 1{:}3)}$ ordnet etwa den Spalten 2, 4 und 5 der Matrix X die Werte der Spalten 1 bis 3 der Matrix Y zu.
Falls $\mathbf{x}$ ein n-dimensionaler Vektor ist, so tauscht $\mathbf{x} = \mathbf{x}(\mathbf{n} : -\mathbf{1} : \mathbf{1})$ die Reihenfolge der Einträge.
Ist $\mathbf{A}$ die obige Matrix und $\mathbf{z} = [\mathbf{2}\ \ \mathbf{3}\ \ \mathbf{1}]$ ein Permutationsvektor, so vertauscht die Anweisung $\mathbf{A} = \mathbf{A}(\mathbf{z}, :)$

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

die Zeilen von A gemäß dem Permutationsvektor.

5 Rechenoperationen

Folgende Operationen sind für Matrizen (und natürlich auch Skalare) möglich:

$+$	Addition
$-$	Subtraktion
$*$	Multiplikation
ˆ	Potenzieren
\	"Linksdivision": Die Auflösung des linearen Gleichungssystems $\mathbf{A} * \mathbf{x} = \mathbf{b}$ ist durch die Operation $\mathbf{x} = \mathbf{A} \backslash \mathbf{b}$ möglich.
/	"Rechtsdivision": Die Auflösung des linearen Gleichungssystems $\mathbf{x} * \mathbf{A} = \mathbf{b}$ ist durch die Operation $\mathbf{x} = \mathbf{b}/\mathbf{A}$ möglich.

Wenn den Operatoren $*$, ˆ , \ oder / ein Punkt **.** vorangestellt wird, so arbeiten sie nicht im Sinne der entsprechenden Matrixoperatoren, sondern **eintragsweise**.
Die übliche Matrixmultiplikation ist:

$$[\mathbf{1}\ \ \mathbf{2}\ ;\ \mathbf{3}\ \ \mathbf{4}] * [\mathbf{0}\ \ \mathbf{1}\ ;\ \mathbf{1}\ \ \mathbf{2}]$$

$$\text{ans} \ = \begin{matrix} 2 & 5 \\ 4 & 11 \end{matrix}$$

Rechnet man elementweise erhält man jedoch:

$$[\mathbf{1}\ \ \mathbf{2}\ ;\ \mathbf{3}\ \ \mathbf{4}] .* [\mathbf{0}\ \ \mathbf{1}\ ;\ \mathbf{1}\ \ \mathbf{2}]$$

$$\text{ans} \ = \begin{matrix} 0 & 2 \\ 3 & 8 \end{matrix}$$

6 Skalare Funktionen

Einige `MATLAB`-Funktionen arbeiten mit Skalaren; werden sie auf Vektoren oder Matrizen angewandt, so arbeiten sie komponentenweise. Dazu gehören unter anderem:

sin, cos, tan	die üblichen Winkelfunktionen
asin, acos, atan	die Umkehrabbildungen der Winkelfunktionen
exp	natürliche Exponentialfunktion
log	natürlicher Logarithmus
rem	Divisionsrest (rem(2,3) = 2, rem(a,b) = "Rest der Division a/b")
abs	Absolutbetrag
sqrt	Quadratwurzel
sign	Signumsfunktion (für $x > 0$ ist sign(-x) = -1 und sign(x) = 1)
round	Rundung (rundet zur nächsten ganzen Zahl ab oder auf ; round(1.5) = 2 ; round(1.49) = 1)
floor	Abrundung zur nächsten ganzen Zahl
ceil	Aufrundung zur nächsten ganzen Zahl

Man kann also eine Wertetabelle der Sinusfunktion aufstellen, indem man zuerst $\mathbf{x} = \mathbf{0.0} : \mathbf{0.1} : \mathbf{2.0}$ definiert und damit $\mathbf{y} = \mathbf{sin}(\mathbf{x})$ berechnet. Mit $[\mathbf{x}\ \ \mathbf{y}]$ bekommt man die Wertetabelle ausgegeben.

7 Vektorfunktionen

Andere `MATLAB`-Funktionen arbeiten insbesondere auf Zeilen- oder Spaltenvektoren. Auf Matrizen angewandt arbeiten sie diese spaltenweise ab und geben die einzelnen Ergebnisse zusammengefaßt als Zeilenvektor aus. Zu diesen Funktionen gehören unter anderem:

max, min	Maximum und Minimum
sort	aufsteigende Sortierung des Vektors
any	gibt 1, falls irgendein Vektoreintrag ungleich 0 ist
all	gibt 1, falls alle Vektoreinträge ungleich 0 sind
length	Vektorlänge
norm	Vektornorm (norm(v) entspricht der 2-Norm, ansonsten meint norm(v,p) die bekannten p-Normen und norm(v,inf) bzw. norm(v,-inf) max- und min-Norm)

8 Matrixfunktionen

Es gibt in `MATLAB` auch einige Funktionen, die nur auf Matrizen angewendet werden können. Hier werden nur kurz die wichtigsten genannt. Genauere Informationen zu den Funktionen erhält man, wenn man den Befehl **help** ***Funktionsname*** verwendet.

′	Transposition
inv	inverse Matrix
norm	1-Norm, 2-Norm oder Maximumsnorm wählbar (norm(A,1) , norm(A,2) bzw. norm(A,inf))
cond	Kondition in der 2-Norm
rank	Rang der Matrix
eig	**eig(A)** liefert einen Vektor der Eigenwerte, **[V,D] = eig(A)** liefert in der Diagonalmatrix D die Eigenwerte und in der Matrix V zusammengefaßt die Eigenvektoren, so dass **A * V = V * D** gilt
det	Determinante
size	Format (bzw. Größe) der Matrix (wird als Zeilenvektor [Zeilenanzahl Spaltenanzahl] ausgegeben)

9 Grafiken

Zweidimensionale Graphen lassen sich leicht mit dem Befehl **plot** realisieren, z.B.

x = - 2 * pi : .01 : 2 * pi	Vektor von -2π bis $+2\pi$ mit der Schrittweite 0.01
$\mathbf{y} = \mathbf{x} .* \mathbf{sin}(\mathbf{x})$	Vektor der y-Werte ($y_i = x_i * \sin(x_i)$ für alle i)
plot(x,y)	gibt einen zweidimensionalen Plot des Grpahen

Man kann den Zwischenschritt, in dem y aufgestellt wird, auch auslassen und statt dessen
$\mathbf{plot}(\mathbf{x}, \mathbf{x} .* \mathbf{sin}(\mathbf{x}))$
schreiben. Der ausgegebene Graph ist der selbe wie oben.
Möchte man mehrere Graphen in unterschiedlichen Fenstern darstellen, so kann mit dem Befehl **figure(n)** das n-te Fenster als aktuelles gesetzt werden.
Gibt man
figure(1);
plot(x,y1);
figure(2);
plot(x,y2);
ein, so zeigt der Befehl **figure(n);** die n-te Graphik an.
Weitere Optionen des Befehls **plot** ermöglichen z.B. die Auswahl der Liniendarstellung. So wird mit **plot(x,y,'-.')** der Graph mit einer "Strich-Punkt-Linie" dargestellt. Zudem kann man mehrere Graphen mit dem Befehl **plot(x,y1,'-.',x,y2,'.',.....)** in einem Bild darstellen lassen. Dabei ist es möglich die Graphen entweder mit einer wie vorher erwähnten Ergänzung zu unterscheiden oder nur die von `MATLAB` vorgegebene farbliche Unterscheidung zu nutzen. Weitere mögliche Befehle erhält man mit **help plot**.
Eine zweidimensional Funktion $z = f(x, y)$ kann man auf zwei verschiedenen Wegen darstellen lassen.
Die erste besteht darin mit dem Befehl **meshgrid** zwei Matrizen zu erzeugen, bei denen in der ersten der erste Argumentvektor (z.B. x-Komponente) in jede Zeile eingetragen wird und in der zweiten der zweite Argumentvektor (z.B. y-Komponente) in jeder Spalte.

Auf diese Matrizen kann elementweise eine matrixwertige Funktion erzeugt werden, deren Ergebnis mit dem Befehl **mesh** dargestellt wird.
[x , y] = meshgrid(-2 : .2 : 2 , -2 : .2 : 2) erzeugt ein Gitter im Quadrat $[-2,2]^2$ mit einer Schrittweite von 0.2 in jeder Richtung
$\mathbf{z} = \mathbf{x} .* \exp(-\mathbf{x}.\hat{}\,\mathbf{2} - \mathbf{y}.\hat{}\,\mathbf{2})$ erzeugt in jedem Gitterpunkt (x_i, y_j), den Wert $z = x_i * \exp(x_i^2 - y_j^2)$
mesh(x,y,z) gibt ein dreidimensionales Bild, auf dem z über der Fläche des Quadrats dargestellt wird

Die andere Variante besteht in der Verwendung des Befehls **plot3**, über den mit **help plot3** weitere Informationen verfügbar sind.
Beschriftungen für die erzeugten Graphiken lassen sich mit **title, legend** und **text** erzeugen. Die möglichen Positionierungsparameter und die genaus Anwendung kann ebenfalls mit dem **help**-Befehl nachgelesen werden.

10 For, while, if

Grundsätzlich ist die Programmsteuerung in `MATLAB` ähnlich denen in anderen Computersprachen möglich. Dabei werden die Befehle, welche mit **for, while** oder **if** eingeleitet werden, mit einem **end** geschlossen und nicht wie z.B. in C-Programmen in geschweifte Klammern eingeschlossen.

Bei **for**-Schleifen werden Start- und Endwert durch einen Doppelpunkt voneinander getrennt. Möchte man eine andere Schrittweite als 1 zwischen die Werte der Schleife legen, so ist die genauso wie beim Erzeugen der Vektoren durch einen entsprechenden Einschub zwischen Start- und Endwert möglich. Dazu zwei Beispiele:

```
for k = 1 : 4
x(k) = k^2;
end
```

Erzeugt:

$$x = \begin{matrix} 1 & 4 & 9 & 16 \end{matrix}$$

```
for k = 4 : -1 : 1
x(k) = k^2;
end
```

Erzeugt:

$$x = \begin{matrix} 16 & 9 & 4 & 1 \end{matrix}$$

Schachtelt man Schleifen ineinander, so ist das eine Möglichkeit eine Matrix zu erzeugen

```
for m = 1 : 3
    for n = 1 : 3
        A(m,n) = m * n;
```

```
    end
end
```

Der Befehl **while** sorgt dafür, dass der eingeschlossene Befehl solange ausgeführt wird, wie die angegebene Bedingung erfüllt ist. Im folgenden Beispiel wird die kleinste ganze Zahl n gesucht, für die $2^n \geq a$ ist, und anschließend ausgegeben.

```
n = 0;
while 2^n < a
    n = n + 1;
end
n
```

Mit dem Befehl **break** lassen sich **for**- und **while**-Schleifen vorzeitig verlassen.

if ermöglicht es ggf. mit **elseif** oder **else** verschiedene Alternativen gegeneinander abzugrenzen; z.B.

```
if   k == l
    A(k,l) = 2;
elseif   abs(k-l) == 1
    A(k,l) = -1;
else
    A(k,l) = 0;
end
```

Würde man diese Befehle nun in zwei **for**-Schleifen von jeweils 1 bis 4 einbinden, so bekäme man am Ende

$$A = \begin{matrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{matrix}$$

An dieser Stelle schließt sich nun eine Liste mit den zur Verfügung stehenden Vergleichsoperatoren und logischen Verknüpfungen an.

==	gleich (nicht zu verwechseln mit "=" womit eine Zuordnung vorgenommen wird)
~=	ungleich
<	echt kleiner
<=	kleiner gleich
>	echt größer
>=	größer gleich
&	und (verknüpft Bedingungen die alle gleichzeitig erfüllt sein sollen)
|	oder (verknüpft Bedingungen von denen mindestens eine erfüllt sein muss)
~	nicht

Auf Skalare angewandt ergeben wahre Beziehungen eine 1, falsche eine 0. Verwendet man Vergleichsoperatoren bei Matrizen, so werden diese komponentenweise verglichen und es wird eine Matrix mit 0 und 1 Einträgen als Ergebnis ausgegeben. Man kann das Ergebnis eines Vergleichs auch einer Variablen zuweisen. Also ist $\mathbf{c} = \mathbf{a} > \mathbf{b}$ in `MATLAB` eine gültige Anweisung.
Bei der Anwendung von Vergleichsoperatoren bei Matrizen muss daher besonders darauf geachtet werden, dass zwei Matrizen nur dann durch den Operator $\sim=$ als ungleich erkannt werden, wenn wirklich alle Einträge nicht miteinander übereinstimmen. Ansonsten muss man mit dem Befehl $\mathbf{any}(\mathbf{any}(\mathbf{A} \sim= \mathbf{B}))$ arbeiten (hierbei überprüft das erste any jeweils die Spaltenvektoren, mit dem zweiten any kann dann kontrolliert werden, ob alle Spalten gleich sind oder nicht), um die Ungleichheit in einer Komponente ausreichen zu lassen, damit die Matrizen als ungleich definiert werden. Man kann diesem Problem ausweichen, indem man die Gleichheit der Matrizen mit **if** abfragt und in die **else**-Anweisung den gewünschten Befehl für den Fall schreibt, dass die Matrizen verschieden sind.

11 Variablenverwaltung

Eine Übersicht über alle zur Zeit belegten Variablennamen liefert der Befehl **who**; **whos** liefert darüber hinaus noch Angaben über die Variablengröße. Eine Variable kann durch **clear** ***Variablenname*** gelöscht werden. Durch **clear** ohne Zusätze werden alle Variablen gelöscht.
Wird `MATLAB` verlassen, gehen alle Variablen verloren. Mit **save** werden diese jedoch in die Datei **matlab.mat** geschrieben und stehen beim nächsten Start von `MATLAB` mit **load** wieder zur Verfügung.

12 M-Files

`MATLAB` kann auch eine Folge von Befehlen ausführen, die in anderen Dateien abgelegt ist, und dies sogar rekursiv (bedeutet, dass man von der einen Datei aus auch wieder eine weitere aufrufen kann). Diese Dateien werden "M-Files" genannt, weil sie auf "**.m**" enden müssen. Die meiste Arbeit in `MATLAB` erfordert normalerweise das Schreiben und Überarbeiten von M-Files.
Es gibt zwei Sorten von M-Files: Script-Files und Function-Files. Eine Script-File ist eine Folge von normalen `MATLAB`-Anweisungen. Sie ermöglicht es "Programme" zu schreiben. Heißt die Datei etwa **rotate.m**, veranlaßt der Befehl **rotate** `MATLAB` dazu, diese Datei abzuarbeiten. Dafür muss die Datei allerdings im aktuellen Laufwerksverzeichnis enthalten sein. Variablen in einer Script-File haben globalen Status und verändern ggf. die der Umgebung. Geschrieben werden diese Dateien mit einem Texteditor, der von `MATLAB` aus geöffnet werden kann, indem man **/File/New/M-File** auswählt.
Function-Files erlauben es `MATLAB` selbst zu "erweitern", indem beliebige eigene Funktionen geschrieben werden können, die `MATLAB` dann, wie die internen, zur Verfügung stehen. Die Variablen in einer Function-File haben

lokalen Status. Ein einfaches Beispiel für eine Function-File ist

function P = prodsqr(A,B)
% PRODSQR Produkt des Quadrates zweier Matrizen
P = A^2 * B^2;

Nach dem Schlüsselwort **function** folgen demnach der Rückgabewert (hier P, bei mehreren Rückgabewerten als Zeilenvektor in eckigen Klammern [a , b , ...]). Nach dem Gleichheitszeichen der Funktionsname und in runden Klammern dahinter die Argumente. Mit dem Prozentzeichen **%** lassen sich Kommentare einfügen. Die so gekennzeichneten Zeilen werden von `MATLAB` nicht bearbeitet. Direkt nach dem Funktionsnamen ist dies insbesondere der von **help** ausgegebene Kommentar.
Diese Datei kann in der gleichnamigen Datei **prodsqr.m** abgelegt werden. Für den Aufruf der Funktion ist der Funktionsname entscheidend. Mit dieser Definition ist z.B. der `MATLAB`-Aufruf **x = prodsqr(x,y)** möglich.

14 Funktionen oder "Function Handles"

Wenn man Funktionen übergeben möchte sind Function Handles äusserst wichtig. Ein Function Handle einer Funktion wird mit einem @ eingeleitet. Hat man z.B. das M-File

function $y = z(x)$
$y = x^2$
end

erstellt, so kann man mit

fh = @z

ein Function Handle von z erstellen. Eine Übergabe dieser 'Funktion' an eine andere Funktion, d.h. ein anderes M-File, kann nun mittels **fh** erfolgen. Eine Auswertung für $x = 3$ erfolgt einfach mittels **fh(3)**. Bei 'einfachen' Funktionen wie x^2 oder x^2+y kann man auch schneller und ohne M-File ein Function Handle erzeugen:

fh1 = @(x) x^2**;**
fh2 = @(x,y) $x^2 + y$**;**

Weitere Informationen auch unter: http://de.mathworks.com/help/matlab/matlab_prog/creating-a-function-handle.html.

15 Eigene Funktionen vektorisieren

Möchten wir eine eigene Funktion nicht nur für x=3 sondern für x=[0,1,2,3,4] auswerten, so können wir eigene Funktionen vektorisieren. Wenn wir die Funktion x*z(x), wobei z(x) aus dem vorherigen Abschnitt kommt, für den Zeilenvektor x auswerten wollen, können wir folgendes neues M-File schreiben:

```
function result = func2(x,z)

result = zeros(0,0);
for i=1:length(x)
result = [result, x(i)*z(x(i))];
end

end
```

Ein Aufruf erfolgt nicht über z sondern ein Function Handle von selbigem, d.h. **fh = @z** und **func(x,fh)**.

16 Effizienzvergleich für Algortihmen: tic & toc

Die Laufzeit einer Operation wird am einfachsten ermittelt, indem der zu untersuchenden Befehlskette **tic**, der Beginn der Zeitmessung, vorangestellt wird, und danach mit **toc** die Uhr gestoppt und die Zeit ausgegeben wird.

17 Protokoll

Man kann eine `MATLAB`-Sitzung einfach mitprotokollieren lassen, etwa, um diese später zu überarbeiten und auszudrucken. Dazu wählt man einen Dateinamen, unter dem gespeichert werden soll, zusammen mit einer brauchbaren Kennung wie etwa **.txt**, und teilt diesen `MATLAB` mit
diary *Dateiname.txt*
In diese Datei wird die ganze Sitzung (mit Ausnahme der Graphiken) ab diesem Punkt mitgeschrieben, bis **diary off** gewählt wird. Fortsetzen kann man wiederum mit **diary on**.

18 Ausgabe in Dateien und auf dem Bildschirm

Man kann mit dem Befehl **fopen** eine beliebige Datei erstellen bzw. öffnen. Je nachdem ob man in der Datei selber schreiben will oder die Datei nur lesen möchte benötigt man die Befehle **w = write** oder **r = read**. Vollständig sehen die Befehle dann beispielsweise wie folgt aus:

Ein = fopen('Eingabewerte.txt','r');
(öffnet die Datei "Eingabewerte" nur zum Auslesen von Werten)

Aus = fopen('Ausgabewerte.txt','w+');
(erzeugt eine neue Datei "Ausgabewerte.txt', in der geschrieben werden kann)

Um nun Werte oder Worte aus der Datei "Eingabewerte.txt" auszulesen kann man den Befehl **fscanf** verwenden:

Wert = fscanf(Ein, '%..',...)

"Ein" verweist hier auf die Datei, aus der etwas ausgelesen werden soll. Hinter das Prozentzeichen kommt z.B. ein **f** für eine Zahl oder ein **s** für ein Wort. An letzter Stelle (**,....**) kann man angeben, wie viele Werte ausgelesen werden sollen. Wenn man dort keine Angaben macht werden alle vorhandenen Werte ausgelesen und in der Variablen "Wert" als Vektor abgespeichert.
Um Werte in die Datei "Ausgabewerte.txt" zu schreiben. Benötigt man den Befehl **fprintf**:

fprintf(Aus, 'Heute ist schönes Wetter. \n Die Temperatur beträgt %f Grad Celsius', Temp);

Mit dieser Befehlszeile kann man den Wert der Variablen Temp an der mit **%f** bezeichneten Stelle ausgeben. **\n** erzeugt einen Zeilenumbruch in der Datei.

Wenn man eine Ausgabe auf dem Bildschirm erzeugen möchte, verwendet man den Befehl **disp**. Um hier Zahlen auszugeben, benötigt man den Operator [ ]. Zur genaueren Verwendung des Operators sollte man die Hilfe für den Befehl **disp** aufrufen und dann zu **int2str** oder **num2str** weitergehen, wo die Konvertierungen beschrieben werden.