VI VISUALISIEREN UND PROGRAMMIEREN IN MATLAB

S. BARTELS, 3.7.2018

1. Einordnung und Aufbau

MATLAB steht für *Matrix Laboratory* und ist eine Programmierumgebung, die viele mathematische Operationen wie das effiziente Lösen linearer Gleichungssysteme und gewöhnlicher Differentialgleichungen sowie Methoden zur Eigenwertberechnung von Matrizen bereit stellt. Darüberhinaus gibt es zahlreiche Möglichkeiten zur Visualisierung mathematischer Objekte. Im Gegensatz zu C++ müssen Programme nicht kompiliert werden und die Verwendung von Listen und Feldern ist deutlich einfacher. Insbesondere müssen Variablentypen nicht spezifiziert werden, sondern werden automatisch angepasst. Ein Nachteil ist, dass die Laufzeit von Programmen gerade bei der Verwendung von Schleifen meist länger als in kompilierten Programmiersprachen ist. Daher ist es oft sinnvoll, Probleme mit einem C++-Programm zu lösen, die Daten in einer Datei abzuspeichern und schließlich mit MATLAB die Datei auszulesen und die Daten graphisch aufzubereiten. Die Programmierumgebung besteht aus verschiedenen Teilen. Zentrale Bestandteile sind das Eingabefenster, der Editor und die Hilfefunktion, Abbildung 1 zeigt die Standardoberfläche von MATLAB.

1.1. **Eingabefenster.** Im Eingabefenster oder *Command Window* können MATLAB-Befehle direkt und interaktiv eingegeben und ausgeführt werden. Auch einige Unix-Befehle für Dateioperationen können wie in einer Konsole ausgeführt werden. Werden Befehle nicht mit einem Semikolon abgeschlossen, so wird das Ergebnis der Berechnung direkt angezeigt. Mit `clc` wird das Fenster geleert und mit `clear` die aktuellen Variablen gelöscht.

1.2. **Editor.** Im Editor werden Programme geschrieben, die dann aus dem Eingabefenster heraus gestartet werden können. Der von MATLAB bereitgestellte Editor verfügt über ein hilfreiches Syntax-Highlighting sowie Einrückfunktionen zur Verbesserung der Übersichtlichkeit. Der Editor kann über das Schaltsymbol *New Script* geöffnet werden.

1.3. **Hilfe.** Erklärungen und Beispiele zu MATLAB-Befehlen findet man über den entsprechenden Menupunkt oder über die Befehle `help` und `doc`.

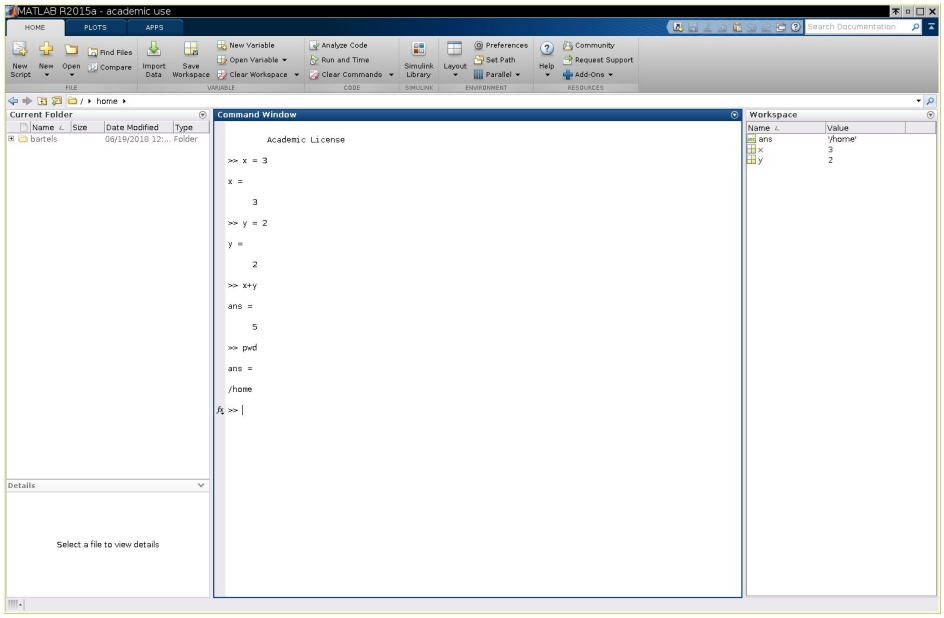

ABBILDUNG 1. Standardoberfläche von MATLAB mit dem Eingabefenester (*Command Window*) in der Mitte.

2. MATRIZEN, VEKTOREN UND LISTEN

Wichtigstes Konzept von MATLAB ist die Verwendung mehrdimensionaler Felder, die auf unterschiedliche Weisen benutzt werden. Wir verwenden die folgende Terminologie spezieller Felder:

- Unter *Matrizen* werden zweidimensionale Felder verstanden, deren Einträge meist Gleitkommazahlen sind.
- Unter *Vektoren* werden Matrizen verstanden, die nur aus einer Zeile oder einer Spalte bestehen.
- Unter *(Index-) Listen* werden Vektoren verstanden, die nur positive ganzzahlige Einträge oder nur boolesche Werte besitzen.

Man beachte, dass Vektoren und Listen spezielle Matrizen sind.

2.1. **Erzeugung von Matrizen.** Eine Matrix kann direkt durch eckige Klammern und die Einträge definiert werden, wobei Einträge einer Zeile mit Kommas und verschiedene Zeilen durch Semikolons getrennt werden. Durch den Befehl

```
A = [1.1,2.2;3.3,4.4;5.5,6.6];
```

wird eine Matrix `A` mit 3 Zeilen und 2 Spalten definiert. Das trennende Komma zwischen Zeileneinträgen kann durch ein oder mehrere Leerzeichen ersetzt werden. Spezielle Matrizen wie die Einheitsmatrix oder Matrizen, die in jedem Eintrag den Wert 0 oder 1 haben, können mit den Befehlen `eye` sowie `zeros` und `ones` definiert werden:

```
E = eye(5);
Z = zeros(2,4);
A = ones(5,3);
```

Listen und Vektoren mit gleichen Abständen zwischen den Einträgen lassen sich durch Angabe des Abstands oder der Anzahl der Einträge mit den Kommandos

```
I = i_a:incr:i_b;
X = linspace(a,b,N);
```

generieren. Durch den ersten Befehl wird eine Liste mit Einträgen erzeugt, die bei `i_a` beginnen und in Abständen von `incr` bis maximal `i_b` gehen, wobei `incr` auch negativ sein darf. Im zweiten Fall ist das Inkrement gegeben durch $(b-a)/(N-1)$. Beispielsweise werden durch die Kommandos

```
I = 1:2:9;
X = 0.0:.1:1.0;
Y = linspace(0,1,11);
```

eine Indexliste `I` mit Einträgen 1, 3, 5, 7, 9 sowie identische Zeilenvektoren `X` und `Y` mit den elf Einträgen 0.0, 0.1, 0.2, ..., 1.0 generiert.

2.2. **Rechnen mit Matrizen.** Die komponentenweise Addition und Subtraktion für Matrizen gleicher Größe lassen sich direkt angeben:

```
J = [1,2,3] + [4,5,6];
Y = [1.1;2.2] - [0.2;0.3];
```

Die Multiplikation von Matrizen entspricht der gewöhnlichen Matrixmultiplikation und muss wohldefiniert sein, das heißt die Größen der beteiligten Matrizen müssen kompatibel sein, durch

```
b = [1.0,2.0,3.0;4.0,5.0,6.0] * [1.0;2.0;3.0];
```

wird ein Vektor `b` mit den Einträgen 14.0 und 32.0 definiert. Die Indizierung von Matrizen beginnt in MATLAB mit dem Index 1, der Zugriff auf Einträge erfolgt durch Verwendung runder Klammern:

```
A = [1,2;3,4];
det_A = A(1,1)*A(2,2)-A(1,2)*A(2,1);
```

Mit zulässigen Indexlisten kann man auf Teilmatrizen zugreifen, durch

```
A = [1,2;3,4;5,6]; I = [1,2]; J = 2; B = A(I,J);
```

wird eine Matrix `B` mit zwei Zeilen und einer Spalte und den Einträgen 2 und 4 definiert. Wenn auf alle Zeilen oder Spalten zugegriffen werden soll, kann ein Doppelpunkt verwendet werden, durch

```
A = [1,2;3,4;5,6]; I = [1,2]; B = A(I,:);
```

wird eine Matrix `B` mit zwei Zeilen und zwei Spalten und den Einträgen 1, 2, 3 und 4 definiert. Die Tranposition einer Matrix erfolgt mit einem Apostroph:

```
A = [1,2;3,4;5,6]; B = A';
```

Algebraische Operationen können komponentenweise angewendet werden, um zwei Listen oder Matrizen gleicher Größe zu verknüpfen. Durch

```
I = [1,2]; J = [3,4]; K = I.*J;
```

wird eine Liste K mit den zwei Einträgen 3 und 8 erzeugt. Auf Matrizen können Funktionen angewendet werden, was in der Regel komponentenweise geschieht, beispielsweise erhält man mittels

```
X = 0.0:0.01:1.0;
Y = sin(X);
```

einen Vektor `Y` der die Werte der Sinusfunktion in den Punkten $0.0, 0.1, \dots, 1.0$ enthält. Tabelle 1 zeigt einige wichtige Kommandos zum Arbeiten mit Matrizen, Vektoren und Listen.

`[a,b,...;x,y,...]`	Definition eines Arrays
`[a,b,...],[x;y;...]`	Definition eines Zeilen- oder Spaltenvektors
`A(i,j)`, `I(j)`	Zugriff auf die Einträge eines Arrays
`a:b`, `a:step:b`	Liste von a bis b mit Schrittweite 1 oder *step*
`linspace(a,b,N)`	Äquidistante Partitionierung des Intervalls $[a, b]$
`A(i,:)`, `A(:,j)`	i-te Zeile und j-te Spalte von A
`A(I,J)`	Teilmatrix definiert durch Listen I und J
`ones(n,m)`	Array mit Einträgen 1
`zeros(n,m)`	Array mit Einträgen 0
`randn(n,m)`	Matrix mit zufällig generierten Einträgen
`A'`	Transponierte Matrix
`A+B`, `A-B`, `A*B`	Addition, Subtraktion und Produkt von Matrizen
`sort(A)`	Sortierung der Einträge eines Arrays
`sum(A,1)`, `sum(A,2)`	Spalten- und zeilenweise Summenbildung
`max(A)`, `min(A)`	Spaltenweise Extremwerte eines Arrays
`size(A)`, `length(I)`	Dimensionen eines Arrays und Länge einer Liste
`find(A)`	Finden der Nichtnulleinträge einer Matrix
`unique(I)`	Eliminieren doppelter Einträge einer Liste

Tabelle 1. Erstellung und Manipulation von Matrizen.

3. Plotten von Funktionen und Vektorfeldern

3.1. **Graphen eindimensionaler Funktionen.** Zur grafischen Darstellung eindimensionaler Funktionen eignet sich der Befehl `plot`. Dazu werden zwei Vektoren mit Argumenten und zugehörigen Funktionswerten benötigt. Die Routine erzeugt dann einen Polygonzug durch die damit definierten Punkte. Mit weiteren optionalen Angaben kann die Darstellung der Kurve beeinflusst werden:

```
X = 0:.01:1;
U = sin(X);
```

```
plot(X,U,'-r');
```

Sollen mehrere Graphen in einem Bild erscheinen, bietet sich das Kommando `hold on|off` an, welches vermeidet, dass die Grafik bei einem neuen Aufruf von `plot` gelöscht wird. Mit dem Befehl `legend` kann eine Legende erzeugt werden:

```
X = 0:.01:1;
U = sin(X); plot(X,U,'r-'); hold on;
V = cos(X); plot(X,V,'b-');
W = X.^2;   plot(X,W,'k-'); hold off;
legend('sin','cos','x^2');
```

Weitere hilfreiche Befehle zur Bearbeitung der Grafik finden sich in Tabelle 2.

3.2. **Darstellung von Kurven.** Eine Kurve, das heißt eine Abbildung $c : [a, b] \to \mathbb{R}^3$ mit einem Parameterintervall $[a, b] \subset \mathbb{R}$, lässt sich mit Punkten im Intervall $[a, b]$ und drei Vektoren, die die drei Komponenten der entsprechenden Funktionswerte enthalten, mit dem Befehl `plot3` darstellen. Für die Helix $t \mapsto (\sin(t), \cos(t), 5t)$, $t \in [0, 10\pi]$ erfolgt dies beispielsweise mit folgenden Kommandos:

```
T = [0:.01:10*pi];
C1 = sin(T); C2 = cos(T); C3 = 5*T;
plot3(C1,C2,C3);
```

Man beachte, dass die Punkte im Parameterbereich nicht für den Aufruf der Routine `plot3` benötigt werden.

3.3. **Graphen zweidimensionaler Funktionen.** Die Visualisierung von Funktionen die in zweidimensionalen Gebieten definiert sind, ist etwas aufwendiger, da ein geeignetes Gitter benötigt wird. Dies lässt sich mit dem Befehl `meshgrid` generieren:

```
[X,Y] = meshgrid(a:dx:b,c:dy:d);
```

Hier sind `X` und `Y` Matrizen, die jeweils zusammengehörende x- und y-Koordinaten enthalten, welche die Gitterpunkte $p_{ij} = (x_{ij}, y_{ij})$ im Rechteck $[a, b] \times [c, d]$ mit Abständen `dx` und `dy` definieren. Mit zugehörigen Funktionswerten $f(p_{ij})$ erlaubt der Befehl `surf` dann eine grafische Darstellung. Die Funktion $f(x, y) = \sin(x)\cos(y)$ kann so im Bereich $[0, 1] \times [\pi/2, 3\pi/2]$ beispielsweise folgendermaßen dargestellt werden:

```
[X,Y] = meshgrid(0:.1:1,pi/2:.1:3*pi/2);
U = sin(X).*cos(Y);
surf(X,Y,U);
colorbar;
```

Das Kommando `colorbar` sorgt für die Einblendung einer Farbskala.

3.4. **Darstellung von Höhenlinien und Vektorfeldern.** Wird mittels des Befehls `meshgrid` und Angabe zugehöriger Funktionswerte eine Gitterfunktion $\phi : Q \to \mathbb{R}$ auf einem Rechteck $Q = [a, b] \times [c, d]$ definiert, so erlaubt die Routine `contour` die Darstellung von Höhenlinien:

```
[X,Y] = meshgrid(0:.1:1,pi/2:.1:3*pi/2);
U = sin(X).*cos(Y);
contour(X,Y,U);
```

Zur Visualisierung von Vektorfeldern, das heißt von vektorwertigen Abbildungen $F : Q \to \mathbb{R}^d$ mit $Q \subset \mathbb{R}^d$ für $d = 2$ oder $d = 3$, eignen sich die Befehle `quiver` und `quiver3`. Dabei werden die Koordinaten von Punkten im Definitionsbereich Q sowie die Komponenten der zugehörigen Funktionswerte mit vier beziehungsweise sechs Vektoren oder Matrizen spezifiziert. Mit den Befehlen

```
[X,Y] = meshgrid(0:.1:1,0:.1:1);
F_1 = sin(Y); F_2 = cos(X);
quiver(X,Y,F_1,F_2);
```

wird beispielswise das zweidimensionale Vektorfeld $F(x, y) = [\sin(y), \cos(x)]$ im Bereich $[0, 1] \times [0, 1]$ grafisch dargestellt. Eine wichtige Klasse von Vektorfeldern sind Gradienten von Funktionen $f : Q \to \mathbb{R}$ also die Abbildung $F = \nabla f : Q \to \mathbb{R}^d$. Für eine Gitterfunktion ϕ wie oben kann ihr Gradient mit der Routine `gradient` approximativ dargestellt werden. Interessant ist beispielsweise die gemeinsame Darstellung von Höhenlinien und dem Gradientenfeld einer Funktion:

```
dx = .05; dy = .05;
[X,Y] = meshgrid(0:dx:1,0:dy:1);
U = sin(X).*cos(Y);
[F_1,F_2] = gradient(U,dx,dy);
contour(X,Y,U); hold on
quiver(X,Y,F_1,F_2); hold off
```

Dabei bestätigt sich die Tatsache, dass Gradienten orthogonal zu Höhenlinien stehen und in Richtung des steilsten Anstiegs zeigen.

3.5. **Abspeichern von Grafiken.** Einfache Grafiken wie Graphen eindimensionaler Funktionen speichert man am besten als `eps`- oder `pdf`-Datei ab, wobei das `eps`-Format für Zwecke der grafischen Nachbereitung besser geeignet ist. Aufwendige Grafiken wie Graphen zweidimensionaler Funktionen sollten speichereffizient zum Beispiel im `jpg`-Format abgespeichert werden. Durch die Befehle

```
figure(1); print -deps einfacher_plot.eps
figure(2); print -djpeg aufwaendiger_plot.jpg
```

wird die im ersten Grafikfenster dargestellten Informationen im `eps`- und die im zweiten im `jpg`-Format abgespeichert.

`disp(A), disp('txt')`	Anzeigen einer Variable und einer Zeichenkette
`input('text')`	Einlesen einer Variable
`%`	Markierung eines Kommentars
`plot(X,Y,'-*')`	Polygonzug durch Punkte $(X(k), Y(k))$ in $\mathbb{R}^2$
`hold on, hold off`	Darstellung mehrerer Objekte in einer Grafik
`mesh(X,Y,Z)`	Darstellung eines zweidimensionalen Graphen
`meshgrid`	Erzeugung eines Gitters
`axis([x1,x2,...])`	Begrenzung des dargestellten Bereichs
`xlabel`, `ylabel`	Beschriftung der Achsen
`legend`	Einfügen einer Legende
`figure(k)`	Öffnen oder Auswahl eines Grafikfensters
`clf`	Leeren des aktuellen Grafikfensters
`subplot(n,m,j)`	Darstellung mehrerer Plots in einem Fenster
`quiver`, `quiver3`	Visualisierung von Vektorfeldern
`print`	Exportieren einer Grafik

TABELLE 2. Darstellung und Bearbeitung grafischer Objekten.

4. Programmieren in Matlab

4.1. **Programmdateien.** MATLAB-Programme, die auch als M-Skripte bezeichnet werden, sind Folgen von MATLAB-Kommandos, die in einer Datei mit der Endung `.m` abgespeichert werden also beispielsweise `matlab_prog.m`. Solche Programme können aus dem Eingabefenster heraus durch Angabe des Dateinamens ohne die Endung gestartet werden, sofern man sich im entsprechenden Verzeichnis befindet. Alternativ kann ein Programm aus der Menüleiste heraus gestartet werden. Das folgende Beispielprogramm wird aus dem Eingabefenster durch Eingabe von `beispiel` gestartet:

```
% Programm beispiel.m, Start durch Eingabe "beispiel"
x = 1;
y = 2;
disp('Der Quotient von x und y ist');
disp(x/y)
```

Im Gegensatz zu C++ liefert es das Resultat 0.5. Laufende MATLAB-Programme können im Eingabefenster durch die Tastenkombination `Ctrl-C` abgebrochen werden.

4.2. **Variablen.** In MATLAB stehen die üblichen Variablentypen wie `int`, `double` und `logical` zur Verfügung. Eine Deklaration ist nicht erforderlich, diese erfolgt automatisch bei der Definition von Variablen. Darüberhinaus wird der Typ automatisch an den Wert eines Ausdrucks angepasst. In den Zeilen

```
k = 1;
k = k/2;
```

```
str = 'zeichenkette';
```

wird `k` bei Definition als Variable vom Typ `double` festgelegt und hat nach dem zweiten Kommando den Wert 0.5. Mit den arithmetischen Operationen und Vergleichen sowie konstanten Werten, die für die verschiedenen Variablen definiert sind, lassen sich Ausdrücke bilden, die dann Variablen zugewiesen werden können. Eine Auswahl der wichtigsten Operationen und konstanten Werte sowie mathematischer Grundfunktionen ist in Abbildung 3 aufgeführt.

`+,-,*,/`	Arithmetische Grundoperationen
`a==b`, `a~=b`	Logischer Test auf Gleich- oder Ungleichheit
`a<b`, `a<=b`	Logischer Vergleich zweier Variablen
`b1&&b2`, `b1||b2`, `~b`	Logisches UND und ODER sowie NEGATION
`true, false`	Logische Werte wahr und falsch
`sqrt(x)`, `x^y`	Quadratwurzel und Potenzen
`exp(x)`, `ln(x)`	Exponentialfunktion und Logarithmus
`sin(x)`, `cos(x)`, `pi`	Trigonometrische Funktionen und Konstante π
`norm(x,p)`	p-Norm eines Vektors

TABELLE 3. Grundoperationen, Vergleiche und elementare Funktionen in MATLAB.

4.3. **Kontrollstrukturen.** In MATLAB stehen die üblichen Kontrollstrukturen zur Verfügung und die Syntax einer `if`-Abfrage sowie einer `for`- und einer `while`-Schleife sind folgendermaßen definiert:

```
if b1 block1 elseif b2 block2 ... else blockn end
while b block end
for var = Z block end
```

Dabei sind `b1`, `b2` und `b` boolesche Variablen beziehungsweise Ausdrücke und `block1`, ... `blockn` sowie `block` Blöcke von Kommandos, also Folgen von Zuweisungen, Funktionsaufrufen oder Kontrollstrukturen. In der `for`-Schleife ist `Z` eine Indexliste oder ein Vektor, der von der Schleifenvariable `var` durchlaufen wird. Besonders zu beachten ist der Befehl `elseif`, der eine Schachtelung mehrerer `if`-Abfragen vermeidet. In Abbildungen 2, 3 und 4 sind entsprechende Beispielprogramme gezeigt.

4.4. **Funktionen.** In MATLAB lassen sich Funktionen definieren, deren syntaktische Struktur folgende Form besitzt:

```
function [val1,...,valm] = funktions_name(arg1,...,argn)
block
end
```

```
% M-Skript if_abfrage.m
x = input('x = ');
if x < 0
    disp('x ist kleiner als Null');
elseif x > 0
    disp('x ist groesser als Null');
else
    disp('x ist Null');
end
```

ABBILDUNG 2. MATLAB-Programm zur Bestimmung des Vorzeichens einer Zahl.

```
% M-Skript while_schleife.m
x = 1;
while 1+x > 1
    x = x/2;
end
disp(x);
```

ABBILDUNG 3. MATLAB-Programm zur Bestimmung der Differenz zwischen der Zahl 1 und der nächstgrößeren Gleitkommazahl.

```
% M-Skript for_schleife.m
J = 1:2:10;
sum = 0;
for j = J
    sum = sum+j;
end
disp(sum);
```

ABBILDUNG 4. MATLAB-Programm zur Bestimmung der Summe der ungeraden Zahlen zwischen 1 und 10.

Dabei ist das schließende `end` optional. Die Verwendung von Rekursionen ist in MATLAB zulässig. Hervorzuheben ist die Möglichkeit der Verwendung mehrerer Rückgabewerte; die zugehörigen Variablen müssen im Befehlsblock definiert werden. Funktionen werden über den Dateinamen aufgerufen, sodass es sinnvoll erscheint, als Dateinamen den Funktionsnamen mit der Endung `.m` zu verwenden. Abbildung 5 zeigt die Bestimmung der Länge eines Vektors in einer Funktion, ihr Aufruf erfolgt zum Beispiel mittels `laenge = veknorm([1,2,3])` im Eingabefenster oder einem anderen Programm.

```
% M-Skript veknorm.m
function val = veknorm(z)
val = 0.0;
for j = 1:length(z)
    val = val+z(j)^2;
end
val = sqrt(val);
% end
```

ABBILDUNG 5. Funktion zur Bestimmung der euklidischen Länge eines Vektors.

Es können mehrere Funktionen in einer Datei abgespeichert werden, wobei nur auf die zuerst aufgeführte von außen zugegriffen werden kann. Die anderen dienen dann als Unterfunktionen für die erste Funktion. Einfache Funktionen können auch in einem *Inline*-Format definiert werden, was besonders für die Definition im Eingabefenster relevant ist:

```
funkt_name = @(arg1,...,argn) ausdruck
```

Die Definition einer Funktion zur Berechnung der euklidischen Länge eines zweidimensionalen Vektors kann so beispielsweise über die Anweisung

```
f_el_2D = @(x,y) (x^2+y^2)^(1/2)
```

Der Aufruf erfolgt dann zum Beispiel mit `f(1.0,2.0)`.

5. WICHTIGSTE KONZEPTE

Wir fassen die wichtigsten Regeln, die bei dem Arbeiten mit MATLAB zu beachten sind, noch einmal zusammen:

- Variablennamen bestehen aus Buchstaben, Ziffern und Unterstrichen, wobei das erste Zeichen keine Ziffer sein darf und der Variablenname nicht nur aus einem Unterstrich bestehen darf.
- Die Verwendung des Variablennames `i` sollte vermieden werden, da dieser standardmäßig für die imaginäre Einheit vorgesehen ist.
- Es wird zwischen Groß- und Kleinbuchstaben unterschieden, das heißt beispielsweise, dass `Hausnr` und `HausNr` zwei unterschiedliche Variablen sind.
- Schlüsselwörter wie `if`, `elseif`, `else`, `while`, `true` oder `false` dürfen nicht als Variablen- oder Funktionsnamen verwendet werden.
- Werden Zuweisungen nicht mit einem Semikolon beendet, so wird der Wert des Ausdrucks auf dem Bildschirm angezeigt.
- Bei der Definition von Matrizen werden Einträge in einer Zeile mit einem Komma oder Leerzeichen und verschiedene Zeilen durch ein Semikolon getrennt.
- Die Indizierung von Listen und Matrizen beginnt mit dem Index 1.

- Variablentypen werden stets an den Typ einer Berechnung angepasst, das heißt, dass (im Gegensatz zu C++) keine Typkonvertierung und somit Rundung eines zugewiesenen Ausdrucks stattfindet.
- Mit Indexlisten kann auf Teilmatrizen zugegriffen werden.

6. Weiterführende Aspekte

6.1. **Numerische Mathematik.** Effiziente Realisierungen gewisser Standardverfahren der numerischen Mathematik sind in MATLAB verfügbar und intuitiv benutzbar. Beispielsweise erfolgt die Berechnung der Determinante, die Bestimmung von Eigenvektoren und -werten sowie das Aufstellen der Inversen einer quadratischen Matrix `A` über die folgenden Anweisungen:

```
val = det(A);
[V,D] = eig(A);
A_inv = inv(A);
```

Zur Lösung eines linearen Gleichungssystems $Ax = b$ sollte auf die Verwendung der Inversen verzichtet werden und stattdessen der *Backslash*-Operator benutzt werden:

```
x = A\b;
```

Dadurch wird ein effizienteres und stabileres Verfahren zur Lösung herangezogen. Die approximative Integration einer Funktion kann durch die Kommandos

```
f = @(x) 1./(1+x.^2);
int_f = quad(f,-1,1);
```

erfolgen. Dabei ist zu beachten, dass die zu integrierende Funktion in vektorieller Form realisiert ist, das heißt für einen Vektor als Argument einen Vektor gleicher Länge mit den entsprechenden Funktionswerten zurückgibt. Das approximative Lösen einer gewöhnlichen Differentialgleichung $y'(t) = f(t, y(t))$, $y(0) = y_0$, kann mit der MATLAB-Routine `ode45` erfolgen:

```
T = 10; y_0 = 1;
f = @(t,y) cos(2*t)*y^2;
[t_list,y_list] = ode45(f,[0,T],y_0);
plot(t_list,y_list);
```

Die Routine gibt einen Vektor von Zeitpunkten im Intervall $[0, T]$ und die zugehörigen Funktionswerte einer Näherungslösung $\widetilde{y}$ zurück, die anschließend grafisch dargestellt werden.

6.2. **Vektorisierung.** Die Verwendung von Schleifen kann in MATLAB gelegentlich zu Laufzeitproblemen führen. Sofern es möglich ist, sollte die entsprechende Berechnung mittels Vektoroperationen durchgeführt werden, was als *Vektorisierung* bezeichnet wird. Als Beispiel betrachten wir die Berechnung des Skalarprodukts zweier großer Vektoren:

```
n = 1e8; x = rand(n,1); y = rand(n,1);
tic; val = 0; for j = 1:n val = val+x(j)*y(j); end; toc
```

```
tic; val = sum(x.*y); toc
```

Die Befehle `tic` und `toc` erlauben eine Zeitmessung und es stellt sich heraus, dass die zweite Berechnung des Skalarprodukts etwa viermal schneller ist als die erste.

6.3. **Laden und Speichern von Daten.** Für das Laden und Speichern von Daten existieren zwei Möglichkeiten. Einerseits können Daten aus Textdateien, das heißt im lesbaren `ascii`-Format gelesen und abgespeichert werden, was über die Befehle

```
A = load('test_ascii.dat');
save test_ascii.dat A -ascii;
```

erfolgen kann. Dies ist besonders nützlich, wenn Daten mit anderen Programmen anderer Programmiersprachen erzeugt worden sind. Sollen in MATLAB generierte Variablen abgespeichert und später in MATLAB wieder eingeladen werden, bietet sich das MATLAB-eigene `mat`-Format an:

```
save test_mat.mat A B c d x;
load test_mat.mat;
```

Dadurch werden die gespeicherten Daten beim Laden wieder den zuvor benutzten Variablen zugewiesen.