ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG
INSTITUT FÜR INFORMATIK
Lehrstuhl für Mustererkennung und Bildverarbeitung
Prof. Dr.-Ing. Hans Burkhardt

Georges Köhler Allee
Geb. 052, Zi 01-029
D-79085 Freiburg
Tel. (0761) 203-8260

Einführung in MATLAB

Für Algorithmen in digitaler Bildverarbeitung

SS2007

Maja Temerinac
temerina@informatik.uni-freiburg.de

Inhaltsangabe

1.Einführung

1.1. Was ist Matlab?

MATLAB® ist eine Programmiersprache, die sich besonders für technische Berechnungen eignet. Sie integriert Berechnungen, Visualisierung und Programmierung in eine einfach zu bedienende Programmierumgebung. Probleme und Lösungen werden mit Hilfe gängiger mathematischer Formeln ausgedrückt. Typische Anwendungen sind:

- *Mathematische Berechnungen*
- *Entwicklung von Algorithmen*
- *Modellierung und Simulation*
- *Analyse, Erkundung, und Visualisierung von Daten*
- *Wissenschaftliche und technische Graphen*
- *Entwicklung von Applikation, die in eine GUI eingebettet sind*

MATLAB ist ein interaktives System, dessen Grundelement ein Array ist, der keine Dimensionsdeklaration braucht. Dies erlaubt uns insbesondere Aufgaben zu lösen, die sich mit Hilfe von Vektoren bzw. Matrizen formulieren lassen.
Der Name MATLAB steht für ***matrix laboratory***. Mehr Information über MATLAB, MathWorks und Erwerb von Lizenzen finden sich unter:
www.mathworks.com
MathWorks bietet kostenlose Lizenzen für Studenten an. Mehr Informationen hierzu gibt es auch unter obiger Adresse.

.

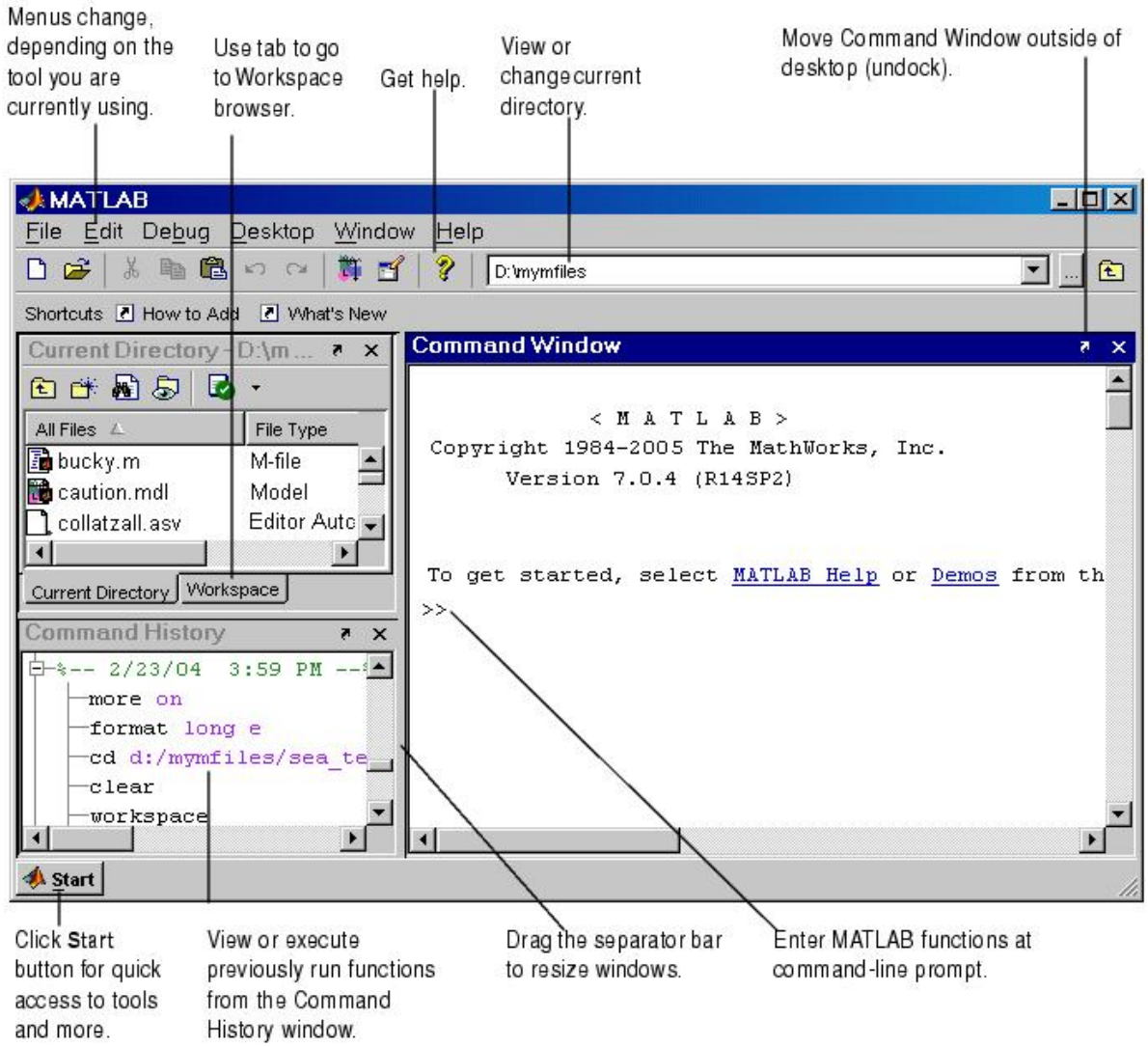
1.2. Das MatlabSystem

Der Matlab Desktop beinhaltet 3 wesentliche Fenster:

- Command Window (rechts) um die Befehle direkt einzugeben.
- Command History (links unten) um die bereits eingegebenen Befehle anzuzeigen
- Workspace (links oben) um die geladenen Variablen zu sehen

Außerdem ist es wichtig zu wissen in welchem Ordner man sich gerade befindet (oben rechts).
Die Hilfe zu Matlab ist sehr ausführlich und eignet sich sowohl zum nachschlagen (Index) als auch zum lernen (viele gut lesbare Artikel und Demos mit Code zum selber experimentieren).

Hier ist ein Überblick des MatLab Desktops und der Desktop tools:



MatLab Command Window

Im Command Window kann man direkt Daten eingeben und verändern, Funktionen und m-Dateien ausführen. In Kapitel 1.4 steht mehr über m-Dateien.

Versuche z.B. folgende Zeilen in den Command Window einzugeben.

(Das Symbol ¶ steht für "enter"):

```
A = [1, 1, 1; 2, 2, 2; 3, 3, 3] ¶
B = [2, 2, 2; 4, 4, 4; 6, 6, 6] ¶
A + B ¶
```

In diesem Beispiel, wurden zuerst zwei Matrizen *A* und *B* definiert mit der Größe 3x3. Dann rechnet MatLab die Summe der beiden aus, *A+B*. Wenn man ein Semikolon nach jeder Zeile eingibt, so bekommt man keine Ausgabe im Command Window, obwohl Matlab die Berechnung durchgeführt hat. Versuche dieselbe Eingabe mit Semikolon. Was kann man beobachten?

Versuche nun folgende Zeile einzugeben:

```
C = A + B;
```

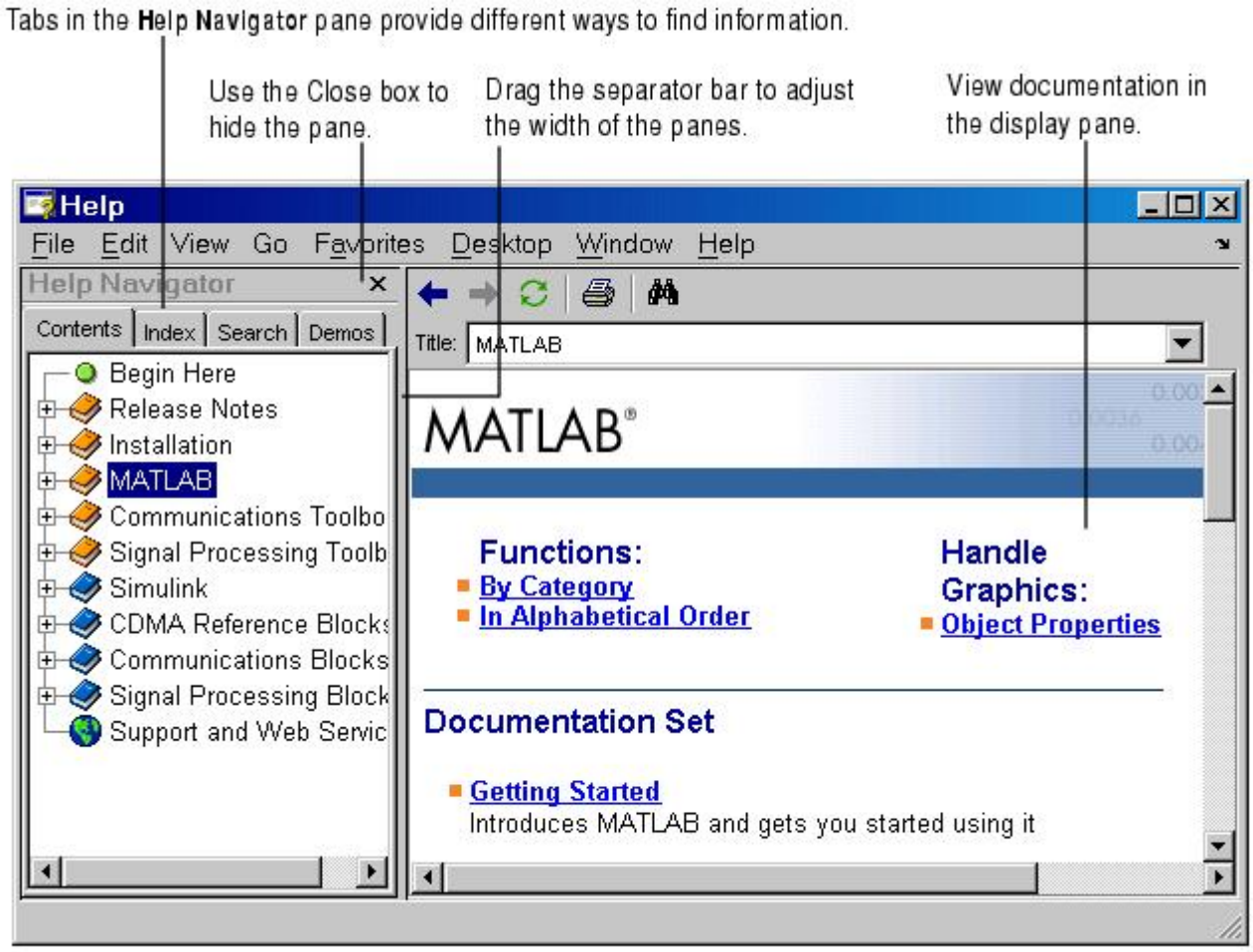
Und dann:

```
C
```

MatLab hat das Ergebnis der Addition von A und B in der Matrix C abgespeichert. Besonders bemerkenswert ist, dass man den Variablentyp gar nicht angeben muss (wie bei Java oder C++). Intern werden alle Zahlen in MatLab in “long precision” Format gespeichert. Der IEEE Standard definiert “long precision”von 10^-308 bis 10^308.
Alle Variablen in MatLab werden als Matrizen gespeichert. Wenn man z.B. eine Variable *myVariable* mit dem Wert 5.37 definiert, so speichert MatLab eine 1x1 Matrix unter *myVariable* ab. Man kann beliebig die Dimension der Variablen verändern.

MatLab Help Browser

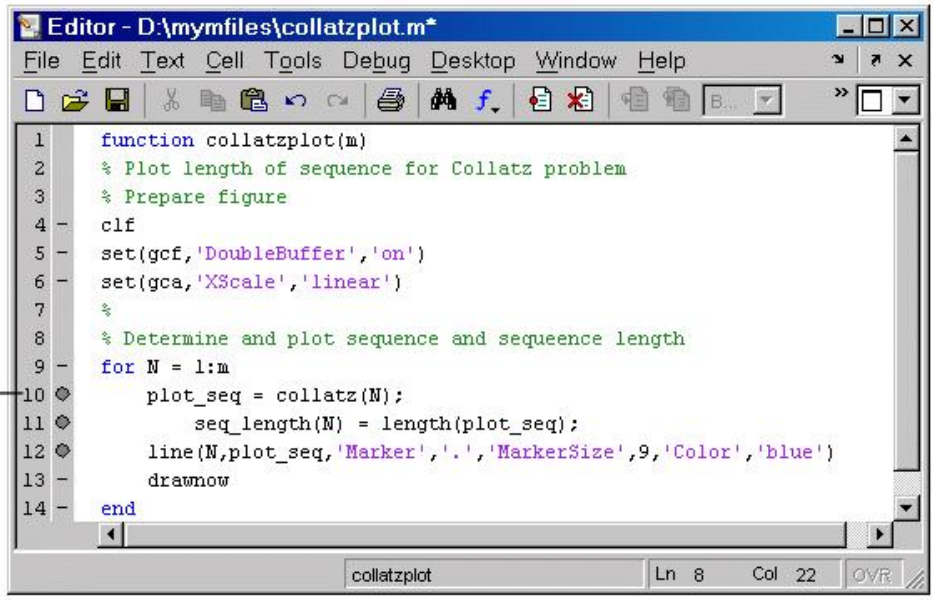
Hier ist ein Überblick über den Matlab Help Browser. Hier kann man alles über MatLab finden – Tutorials, alphabetische Auflistung der Funktionen und viele Beispiele. Viele dieser Dokumente sind auch als PDF Dateien vorhanden und können ausgedruckt werden.



1.3. m-Dateien und Programme

Außer im Command Window, wo man die Daten direkt eingibt und ausführt, kann man den Programmcode in einer Datei mit der Endung *.m* abspeichern. Man kann m-Dateien im MatLab Editor bearbeiten:

When breakpoints are gray, they are not valid. In this example, it is because the file has not been saved since changes were made to it. Save the file to make the breakpoints valid (red).



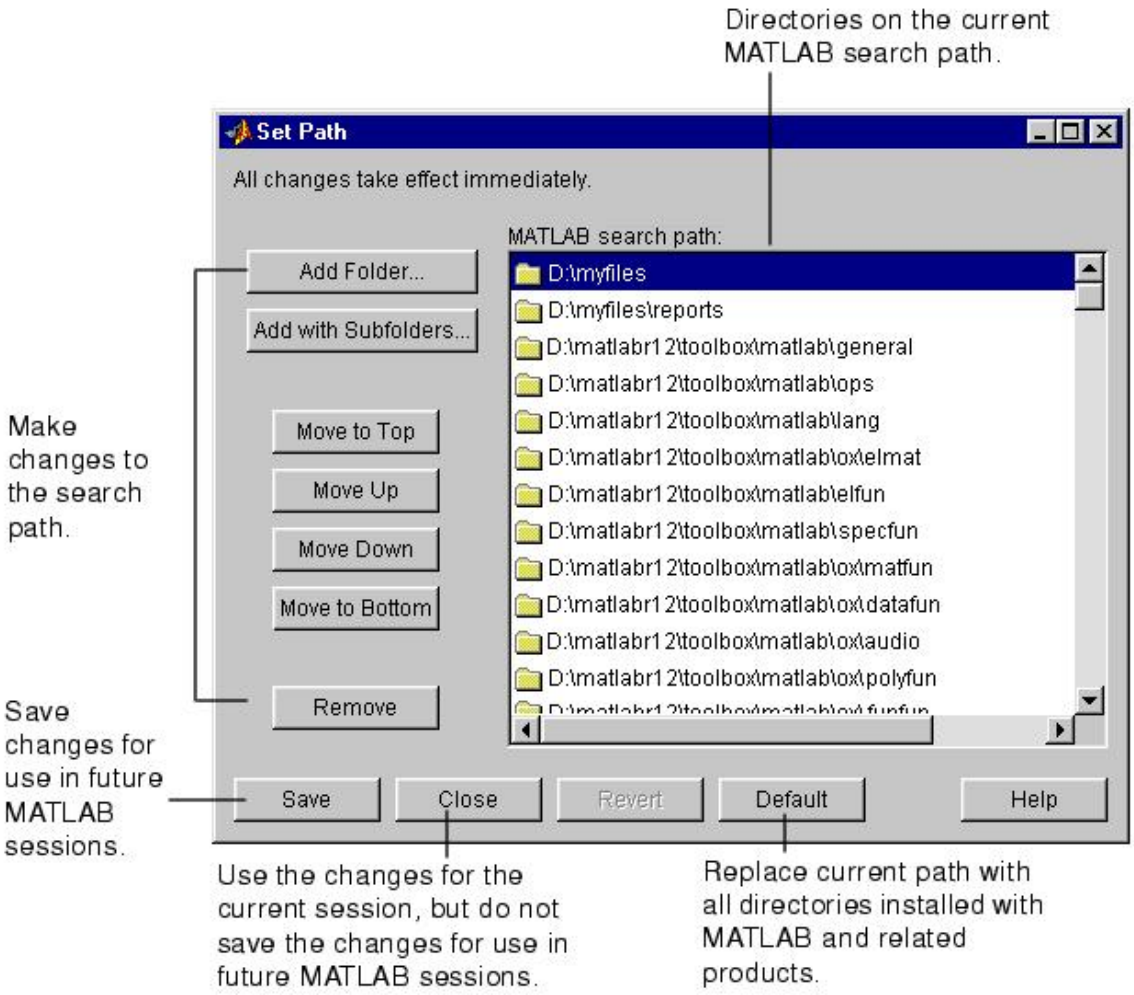
Ein m-File definiert eine Funktion (siehe obiges Beispiel). Hier ist ein kleines Beispiel einer einfachen Funktion, die keine Ein- oder Ausgabeparameter besitzt:

```
% this is a comment
% first, define the name of function
function calculateIt()


% here comes the code of the function
% if you put semicolons after the lines, MatLab won't show you any feedback
(echo) in the command window
A = [1, 1, 1; 2, 2, 2; 3, 3, 3];
B = [2, 2, 2; 4, 4, 4; 6, 6, 6];
A + B

% you don't have to explicitly end the function.
```

Öffne den MatLab Editor (klicke nur auf das Symbol für leeres Blatt) und kopiere obigen Code rein. Speichere die Datei unter dem Namen calculateIt.m in einen Ordner, indem Du auch die anderen Testbeispiele speichern möchtest.
Nun kannst Du Deine erste m-Datei ausführen. Dafür musst Du noch den richtigen Pfad in MatLab setzen.
MatLab hat eine Liste von Ordnern (wie die run-path in DOS oder classpath in Java), wo es nach den m-Dateien sucht. Öffne den *Path Editor* mit *File ->Set Path*. Nun erscheint folgendes Fenster:



Klick auf *Add Folder* und finde den Ordner, in dem Du die erste m-Datei abgespeichert hast. Jedes mal, wenn Du eine Datei in einen anderen Ordner abspeicherst, musst Du den Path in MatLab neu setzen bevor Du die Funktion ausführen kannst
Nun sind wir bereit zum testen! Tippe in die Kommandozeile:

```
calculateIt
```

Du siehst, was Matlab berechnet in der Kommandozeile.

Nun ein kleines Beispiel derselben Funktion nun mit Ein-und Ausgabeparametern.

```
% this is a comment
% first, define the return variable (where you store the variable to be
returned), the name of function and the parameters
% all of the names are your own creations
function myResult = calculateIt(i)


% here comes the code of the function
% if you put semicolons after the lines, MatLab won't show you any feedback
(echo) in the command window
A = [1, 1, 1; 2, 2, 2; 3, 3, 3];
B = [2, 2, 2; 4, 4, 4; 6, 6, 6];


% store the result in the return variable
myResult = (A + B) * i


% you don't have to explicitly end the function.
```

Experimentiere mit dem alten Code! Versuche mehr Eingabeparameter zu setzen oder lösche das Semikolon am Ende. Wie kannst Du die Anzahl der Ausgabeparameter verändern?

Achtung: Der Parameter i muss eine 1x1 Matrix (eine einfache Variable) oder eine 3x3 Matrix sein. Wenn i = 4 ist, so wird MatLab eine elementweise Multiplikation mit 4 durchführen. Wenn wir mit einer 3x3 Matrix multiplizieren, so findet eine Matrixmultiplikation statt. Wenn Du explizit willst, dass Matlab eine elementweise Multiplikation durchführt, so musst Du einen Punkt vor der Operation schreiben.
Zum Beispiel:
A * B wird eine Matrixmultiplikation durchführen mit dem Ergebnis:

```
    12    12    12
    24    24    24
    36    36    36
```

A .* B (achte auf den Punkt vor dem Mal) ist die elementweise Multiplikation mit dem Ergebnis:

```
     2     2     2
     8     8     8
    18    18    18
```

Wichtig: Funktionen ohne Parameter ruft man ohne Klammern auf, Funktionen mit Parametern werden mit der Klammer aufgerufen. Daher muss die Funktion *calculateIt(i)* folgenderweise aufgerufen werden:

```
calculateIt(4)
```

2.Vektoren und Matrizen

Wir haben bereits viele Matrizen gesehen. Nun wollen wir ein paar Funktionen kennen lernen, um mit Matrizen arbeiten zu können.
Die Funktion *magic* gibt ein magisches Quadrat zurück. Wir wollen herausfinden warum sie magisch ist!
Tippe:

```
mag = magic(4)
```

in die Kommandozeile. Du siehst:

```
>> mag = magic(4)
mag =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>>
```

Indem Du nur mag in die Kommandozeile eingibst kannst Du Dir ganze Matrix *mag* anschauen. Wenn Du aber nur einige der Elemente dieser Matrix anschauen möchtest, so musst Du Klammern und den Doppelpunkt-Operator verwenden.
Tippe z.B.:

```
mag(1,1)
```

Wie Du siehst, gibt Dir MatLab nur das erste Element in der ersten Zeile in mag zurück. Der erste Index ist die gewünschte Zeile, der zweite ist die gewünschte Spalte (verwechsle diese nicht!). Versuche andere Elemente auszugeben.
Du kannst auch mehrer Elemente auf einmal mit Hilfe von einem Doppelpunkt ausgeben:

```
mag(1:3,1)
```

gibt das erste Element der Zeilen 1 bis 3 aus.

```
mag(:,1:2)
```

gibt alle Elemente der ersten beiden Spalten aus.
Versuche alle Elemente der ersten beiden Zeilen auszugeben und speichere diese in einer neuen Matrix *smallMag* ab.

Einige andere Funktionen zum ausprobieren

*size(mag)*liefert die Größe der Matrix zurück. *size(mag,n)* liefert die Größe in der n-ten Dimension.
mag' liefert die transponierte Matrix – d.h., die Matrix mit vertauschten Spalten und Zeilen.
sum(mag) liefert die Summe der Spalten in einer Matrix.
diag(mag) liefert die diagonalen Elemente einer Matrix.
ones(n,m) liefert eine Matrix, deren Elemente alle Eins sind und welche die Dimension n x m besitzt.
zeros(n,m) ist das gleiche wie ones, nur dass nun alle Elemente Null sind.
fliplr(mag) dreht eine Matrix von links nach rechts.

Ändern der Dimension einer Matrix

Man kann die Dimension einer Matrix beliebig verändern. Erinnere Dich, wie wir eine einfache Matrix definiert haben:

```
A = [1,1,1;2,2,2;3,3,3]
```

Man kann nun die Matrix A verwenden, um eine größere Matrix B zu basteln:

```
B = [A, A]
```

Man kann aber auch folgendermaßen eine Matrix definieren, die A enthält:

```
C = [A; [3,4,5]]
```

Man muss sehr vorsichtig sein bei der Anzahl der Elemente pro Zeile und Spalte. MatLab füllt nicht automatisch die fehlenden Elemente auf oder ignoriert die überflüssigen Elemente.

Wichtig: Versichere Dich diesen Teil des Handbuches gut zu verstehen. Teste alle Funktionen und Operation hier und experimentiere mit ihnen. Kannst Du beliebige Matrizen definieren und editieren. Was ist das magische an einem magischen Quadrat?

3.Kontrollstrukturen

3.1. *if,else,end*

Hier ist ein Beispiel für *if*:

```
if (a > 6) & (b< 3)
      % do something
```

Und für *else*:

```
else
      % do something else
end
```

Diese Kontrollstruktur berechnet den booleschen Ausdruck nach dem *if* Schlüsselwort und falls der Ausdruck wahr ist, so wird der Block nach dem *if* Schlüsselwort ausgeführt, wenn er falsch ist, dann wird der Block nach dem *else* Schlüsselwort (falls er vorhanden ist) ausgeführt. Der *if* Ausdruck muss immer mit dem Schlüsselwort *end* schließen.

3.2. *switch*

Hier ist ein Beispiel für switch:

```
switch (rem(n,4)==0) + (rem(n,2)==0)
    case 0
       M = ones(n)
    case 1
       M = zeros(n)
    case 2
       M = magic(n)
    otherwise
       error('This is impossible')
 end
```

In diesem Fall berechnet MatLab den Wert des Ausdrucks nach dem *switch* Schlüsselwort und vergleicht ihn mit allen Werten, die nach dem *case* Schlüsselworte stehen. Falls keiner der Werte gleich dem Wert nach dem *switch* Schlüsselwort ist, so wird der Block nach dem *otherwise* Schlüsselwort ausgeführt.

Wichtig: Anders als bei der Programmiersprache C, führt Matlab's *switch* nur den ersten wahren *case* Ausdruck aus und stoppt dann (C würde alle folgenden *case* Ausdrücke auch ausführen). Also, benötigt man bei MatLab keinen *break* Ausdruck.

3.3. *for*

Hier ist ein Beispiel für *for*:

```
for i=1:n
      for j=1:m
            A(i,j) = A(i,j) / B(j,i)
      end
end
```

Die *for* Schleife führt den inneren Ausdruck so oft aus wie die Kontrollvariablen (in diesem Fall *i* und *j*) es angeben. Man sollte *for* Schleifen in MatLab möglichst vermeiden, da sie sehr langsam sind. In vielen Fällen kann man die *for* Schleife durch eine Matrixoperation ersetzen. Matrixoperationen sind in MatLab viel schneller!

3.4. *while*

Hier ist ein Beispiel für *while*:

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
   x = (a+b)/2;
   fx = x^3-2*x-5;
   if sign(fx) == sign(fa)
      a = x; fa = fx;
   else
      b = x; fb = fx;
   end
end
```

Die *while* Schleife führt den inneren Ausdruck durch, falls der logische Ausdruck nach dem *while* Schlüsselwort wahr ist. Wenn dieser Ausdruck falsch ist, so macht das Programm nach dem *end* der *while* Schleife weiter. Wird der Ausdruck nie falsch, so haben wir eine endlose Schleife generiert. Diese sind in der Regel zu vermeiden!
Das Ergebnis des *while* Ausdrucks ist eine Wurzel des Polynoms *x3 - 2x - 5*, nämlich:

```
x =
   2.09455148154233
```

3.5. *continue, break and return*

continue zwingt die aktive Schleife die Ausführung des Befehls der nächsten Iteration derselben Schleife zu überlasen.

break führt dazu, dass die Schleife stoppt. Die Befehle nach der Schleife werden ausgeführt.

return führt dazu, dass die ganze Funktion stoppt.

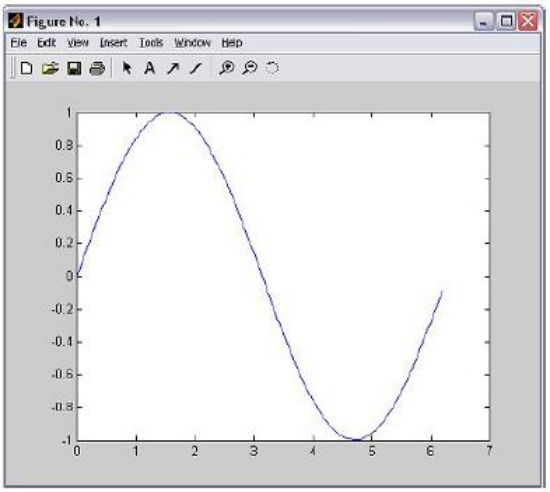
4.Graphische Darstellung mit Matlab

4.1. *plot*

Graphische Darstellung von Funktionen ist mit Matlab sehr einfach. Hier ist ein Beispiel für die Funktion Sinus:

```
x = 0:0.1:2*pi;
y = sin(x);
plot(x,y)
```

Und hier ist das (graphische) Ergebnis:



Die Zeichnung (*plot*) wird in einer *figure* dargestellt.

4.2. Formatieren von graphischen Darstellungen

Mit folgendem Befehl kann man die X- oder Y-Achse labeln:

```
xlabel('X Axis')
ylabel('Sine Function')
```

Mit dem *text* Befehl kann man überall auf dem Bild einen Text hinzufügen:

```
text(0, 1, 'Please note the smoothness of the curve')
```

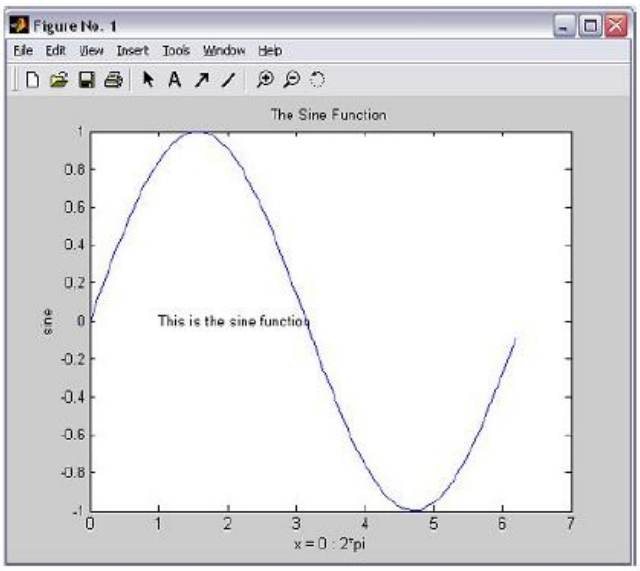
Der Befehl *title* gibt der *figure* einen Namen:

```
title('The Sine Function')
```

Und hier ist ein Beispiel mit allen graphischen Funktionen, die wir kennen:

```
x = 0:0.1:2*pi;
y = sin(x);
plot(x,y)
title('The Sine Function')
xlabel('x = 0 : 2*pi')
ylabel('sine')
text(0,1,'This is the sine function')
```

Das Ergebnis sieht dann so aus:

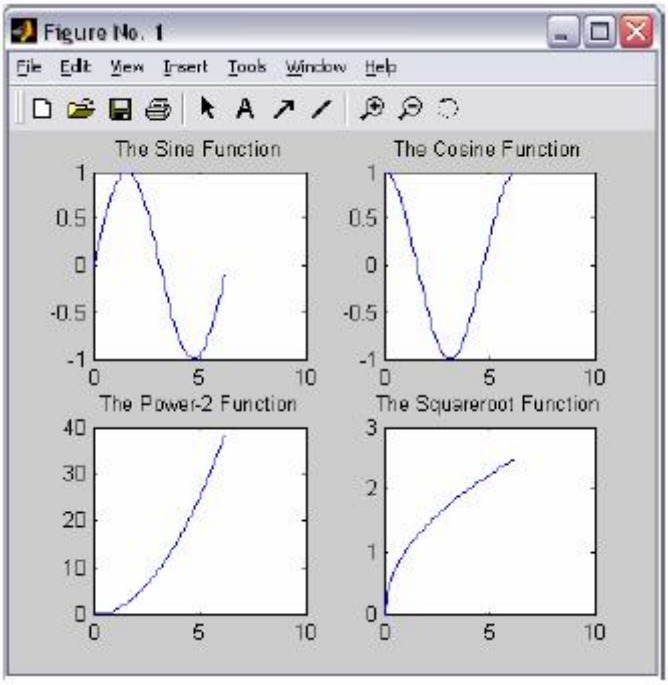


4.3. *subplot*

Man kann mehrere graphische Darstellungen in einem *figure* anzeigen mit dem Befehl *subplot*:

```
% define the data
x = 0:0.1:2*pi;
y1 = sin(x);
y2 = cos(x);
y3 = x .* x;
y4 = sqrt(x);
% subplot the plot into 2x2 matrix, activate the first subplot
subplot(2,2,1)
plot(x,y1)
title('The Sine Function')
% subplot the plot into 2x2 matrix, activate the second subplot
subplot(2,2,2)
plot(x,y2)
title('The Cosine Function')
% subplot the plot into 2x2 matrix, activate the third subplot
subplot(2,2,3)
plot(x,y3)
title('The Power-2 Function')
% subplot the plot into 2x2 matrix, activate the fourth subplot
subplot(2,2,4)
plot(x,y4)
title('The Squareroot Function')
```

Und so sieht dann das Ergebnis aus:



4.4. *figure*

Man kann auch mehrere Figure auf einmal öffnen:

```
figure(n)
```

Hier ist n die Nummer der Figure und MatLab generiert diese Figure (falls sie nicht schon vorhanden ist) und aktiviert diese. Anschließend kannst Du wieder *plot* oder *subplot* verwenden, um Deine Daten anzuzeigen.

5.Bilder

Für die inhaltsbasierte Bildsuche müssen wir wissen, wie man auf Bilddaten zugreifen kann.

5.1. Lesen und schreiben von Bilddaten

MatLab kann mit folgenden Bildformaten umgehen: .jpeg, .tiff, .bmp, .png, .hdf, .pcx, .xwd.
Wir werden das JPEG, GIF und das BMP Bildformat benutzen.
Um ein Bild einlesen zu können benötigst Du den *imread* Befehl:

```
myImage = imread('quadrate.gif');
```

Die Bildinformation der Datei *quadrate.gif* wird nun in der lokalen Variable *myImage* abgespeichert.
Falls Du ein Bild schreiben möchtest, so kannst Du den Befehl *imwrite* verwenden, um ein BMP Bild zu erzeugen:

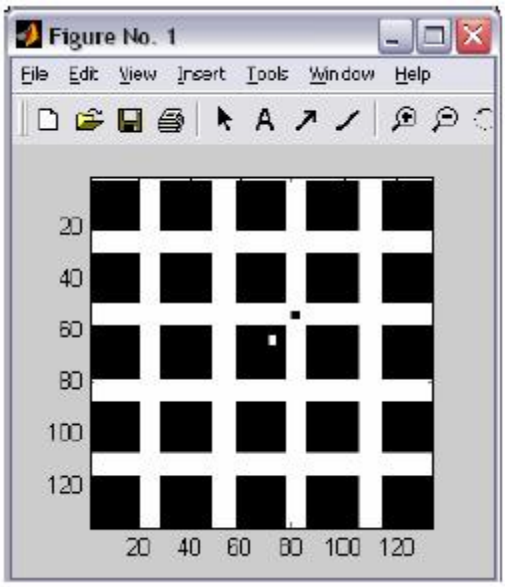
```
imwrite(myImage, 'myImage.bmp', 'bmp')
```

5.2. Graphische Darstellung von Bilddaten

Der Befehl *image* zeichnet die Bilddaten. Hier ist ein Beispiel für das Lesen und das Zeichnen eines kleinen GIF-Bildes.

```
i = imread('quadrateNew.gif');
image(i)
colormap(gray(2))
axis image
```

Und hier ist das Ergebnis:



Wir haben den *image* Befehl statt dem *plot* Befehl benutzt. Der Befehl *axis image* bewirkt, dass nicht das ganze Bild mit der Zeichnung ausgefüllt wird, sondern dass das Bild einheitlich reskaliert und in die Zeichnung eingefügt wird. Was passiert wenn Du *axis image* weglässt?
Die *colormap* Funktion legt fest welche Colormap verwendet wird (D.h. welche Farben im Bild vorkommen). Im obigen Fall haben wir ein Grauwertbild mit nur zwei möglichen Werten – 0 und 1 (also ein binäres Bild). Daher setzen wir die Colormap auf den Wert *gray(2)*. Für normale Grauwertbilder müssen wir den Wert auf *gray(256)* setzen.
Führe die obigen Befehle nacheinander aus und beobachte was passiert!

5.3. Bildformate in Matlab

MatLab liest Bilder im *uint8* Format. Das ist ein Integer Format mit geringer Genauigkeit. Matlab kennt auch die Formate *uint16* und *double*. Manche Operationen sind nur für das *double* Format definiert. Also muss man die Bilddaten in das *double* Format umwandeln, um Operatoren wie +, - , * benutzen zu können.
Hier ist ein Beispiel:

```
img = imread('quadrate.gif');
img = double(img) + 1;
```

Dieser Code wandelt *uint8* Daten in *double* Daten um. Die zusätzlich 1 kommt von dem Offset die alle im Matlab gespeicherten Daten haben. Man muss eine 1 immer dann addieren, wenn man *uint8* oder *uint16* Daten in *double* Daten konvertiert. Andererseits muss man 1 abziehen, wenn man *double* Daten in *integer* Daten umwandelt.

5.4. Manipulation von Bildern

Nachdem Du die Bilddaten in das *double* Format umgewandelt hast, kannst Du die Daten beliebig manipulieren. Du kannst so z.B. auf das Pixel (i,j) Deines Bildes zugreifen:

```
img(i,j) = 1;
```

Für ein RGB Bild werden R, G und B Daten als dritte Dimension des Pixels img(i,j) abgespeichert. Also gilt:

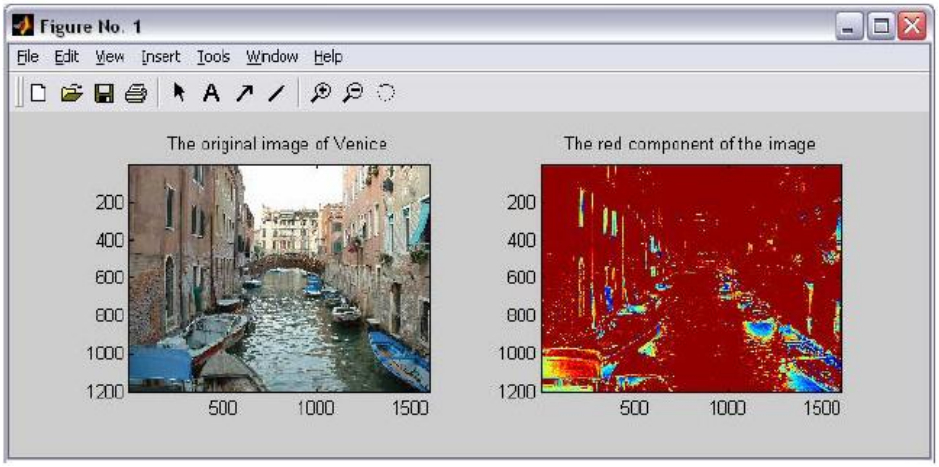
```
img(i,j,1) is the red component
img(i,j,2) is the green component
img(i,j,3) is the blue component
```

So kannst Du z.B. nur die rote Farbe zeichnen:

```
% read the image data
img = imread('venice.jpg')
% plot first the original image into the left subplot
figure(1)
subplot(1,2,1)
```

```
image(img)
title('The original image of Venice')
axis image
% then the red component into the right subplot
subplot(1,2,2)
image(img(:,:,1))
title('The red component of the image')
axis image
```

Das sieht dann so aus:



6.Häufig verwendet Befehle

Einige besondere Funktionen:

pi	3.14159265..
i	imaginäre Einheit (Wurzel aus -1)
j	gleich i
eps	Ungefähr 2^-52, sehr kleine Fließkommazahl
realmin	Kleinste Fließkommazahl, 2^-1022
realmax	Größte Fließkommazahl, 2^1022
NaN	Keine Zahl (undefiniert)

Einfache mathematische Funktionen:

cos	Cosinus Funktion
sin	Sinus Funktion
round	Rundet die Zahl mathematisch
abs	Betrag
sqrt	Wurzel
log	Logarithmus
mod	Modulo Operation
sign	Signum Operation

Zeichenbefehle:

plot	Zeichnet 2D oder 3D Daten
text	Zeichnet Text
xlabel,ylabel,zlabel	Labelt die Achsen
title	Zeigt den Titel der Zeichnung
subplot	Fügt mehrere Plots zusammen
figure	Zeigt oder erzeugt ein Figure

Bildmanipulation:

image	Zeichnet Bilddaten
imread	Liest Bilddaten
imwrite	Schreibt Bilddaten
imfinfo	Zeigt Information über das Bild

7.Übungen

Aufgabe 1

Erforsche die Eigenschaften eines magischen Quadrats der Dimension 6x6. Definiere eine magische Matrix. Dann bilde die Summe der Zeilen und die Summe der Spalten. Bilde die Summe der Diagonalen. Vertausche nun die zwei mittleren Spalten.
Handelt es sich dann noch immer um ein magisches Quadrat?
Im Mittelalter hat man geglaubt, dass das magische Quadrat magische Eigenschaften besitzt. Albrecht Dürer, ein deutscher Maler und Hobbymathematiker, hat das magische Quadrat in seinem Gemälde von 1514 benutzt.

Aufgabe 2

Lade die Daten des Grauwertbildes *gull.gif* aus dem *examples* Ordner. Zeichne es in figure 1 und gebe der Zeichnung eine Überschrift. Skaliere das Bild relativ zum Original.
Dann bilde das Inverse des Bildes und zeichne es in figure 2. Gebe diesem auch eine Überschrift.
Dann versuche aus dem Grauwertbild ein binäres Bild zu machen (*thresholding*).
Thresholding ist für die Bildverarbeitung sehr wichtig und es funktioniert so:

1. Wähle einen Threshold, z.B. 60
2. Alle Pixelwerte, die größer als 60 sind werden auf 1 gesetzt, alle anderen auf 0.

Schreib nun den ganzen Code (laden der Bilddaten, thresholding und zeichnen des Ergebnisses) in eine m-Datei mit zwei Parametern – einen für den Namen des Bildes und einen für den threshold. Das ganze funktioniert natürlich nur bei Grauwertbildern! Wenn Du Dir nicht sicher bist, was für ein Format Dein Bild hat, so verwende den Befehl *imfinfo* mit dem Namen Deines Bildes. Experimentiere mit verschiedenen Werten für den threshold.