

Elementare Bildverarbeitung - MATLAB QuickGuide

Proseminar **Computer Vision**¹ SoSe 2004

Niko Rukavina (nr3@informatik.uni-ulm.de)

1. Einleitung

MATLAB steht für MATrix LABoratory. Es ist ein modular aufgebautes Softwarepaket für numerische Berechnungen mit Vektoren und Matrizen. Es wird in technischen und wissenschaftlichen Bereichen eingesetzt und bietet umfangreiche Möglichkeiten zur Visualisierung von Daten.

Dieses Dokument soll eine schnelle Einführung zum Umgang mit MATLAB bieten².

2. Auf einen Blick

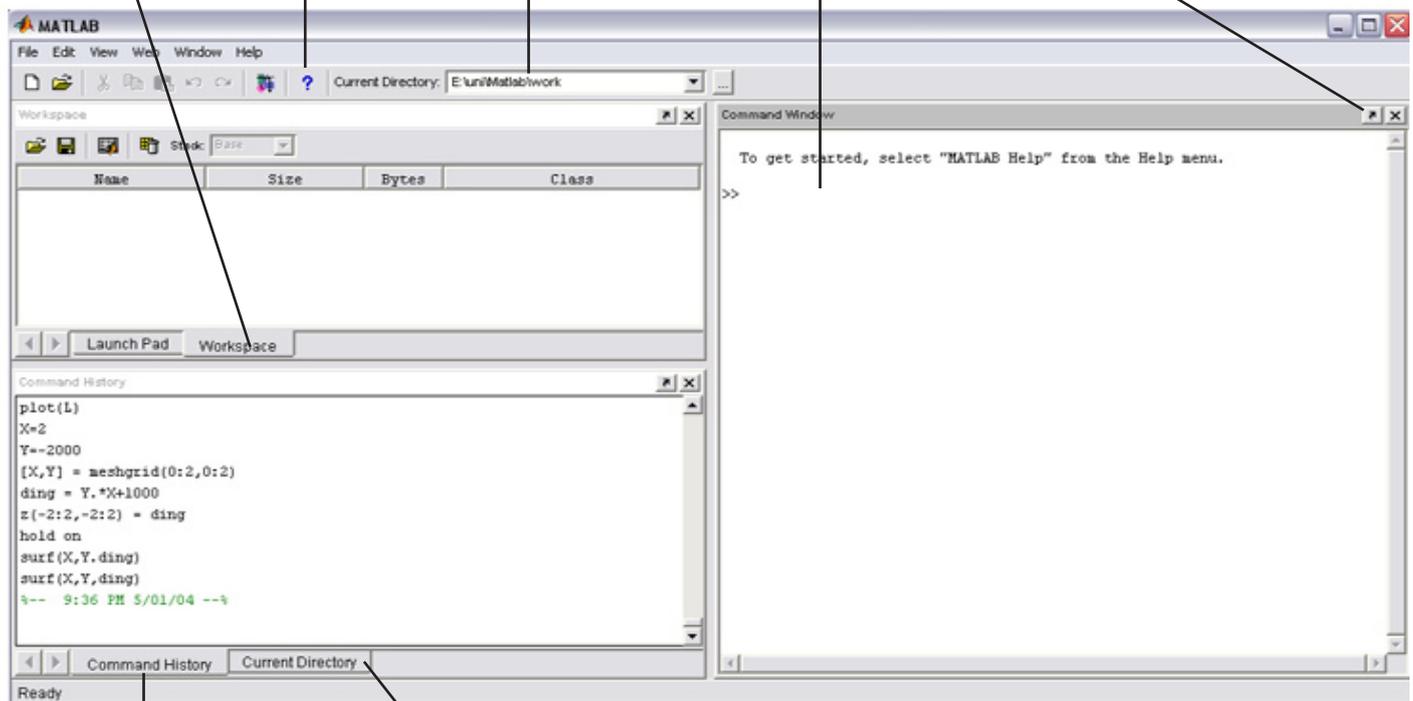
Im Workspace werden alle aktuellen Variablen angezeigt

Aufrufen der
Hilfefunktion

Wechseln des aktuellen
Verzeichnisses

Command Window: Hauptfenster zur Eingabe von MATLAB Befehlen

Fenster von
Arbeitsumgebung
abkoppeln



Zeigt alle ausgeführten Anweisungen Zeigt das aktuelle Verzeichnis

Abb. 1: MATLAB Desktop

3. Elementare Bedienung

An der Uni Ulm ist MATLAB im Linux-Pool samt einiger Toolboxen³ installiert. Zum Starten muss in einem Terminalfenster „use matlab“ gefolgt von „matlab“ eingegeben werden. Das starten kann einige Zeit in Anspruch nehmen.

Im Commandwindow (siehe Abb.1) können nun sequentiell Befehle eingegeben werden. Die Eingabeaufforderung „>>“ zeigt an das MATLAB bereit ist Befehle entgegenzunehmen. Dieses Vorgehen bezeichnet man auch als „interaktiven Modus“. Alternativ ist es auch möglich Befehle, Skripte und eigene Funktionen in sogenannten M-Files zusammenzufassen und diese dann im Stapelbetrieb abzuarbeiten (wird in diesem Vortrag nicht näher erläutert).

Besonders nützlich ist der help-Befehl. Mittels der Syntax `help <befehlname>` erhält man eine ausführliche Beschreibung zu jedem Befehl. Durch die Eingabe von `demo` bekommt man einige interaktive Demonstrationen von MATLABs Fähigkeiten zu sehen. Um alle aktiven Variablen aufzulisten schreibt man `who` und um die Elemente einer bestimmten Variable aufzulisten muss man nur ihre Bezeichnung eingeben und Enter drücken.

Die sehr ausführliche Online-Dokumentation[1] deckt alle Aspekte zum Umgang mit MATLAB ab.

1) Link zur Veranstaltungsseite: <http://www.informatik.uni-ulm.de/ni/Lehre/SS04/ProsemCV/>

2) **Konvention:** Codefragmente sind in Courier gedruckt und hellgrau hinterlegt, vom Benutzer einzugebende Teile fett gedruckt. Dunkelgrau hinterlegte Fragmente sind im Text explizit referenziert. Abkürzung von „Codefragment“ durch „Cf“.

3) Toolboxen sind Zusatzmodule, die MATLAB um zusätzliche Funktionen erweitern.

4. Die Matrix als Datentyp

MATLAB ist ursprünglich zur einfachen Benutzung mathematischer Softwarebibliotheken zur Matrixberechnung entwickelt worden.

Im wesentlichen arbeitet MATLAB daher mit einem grundlegenden Datentyp: der Matrix. Genauer gesagt handelt es sich stets um mehr-dimensionale Arrays. 1 X 1 – Matrizen werden als Skalare und Matrizen mit nur einer Zeile oder einer Spalte als Vektoren interpretiert. Dabei müssen Variablen nicht explizit deklariert werden, wie beispielsweise in Java oder C, so dass die Anpassung zwischen Variablen unterschiedlicher Dimensionierung implizit erfolgen kann.

5.1 Umgang mit Matrizen

Matrizen einzugeben und Operationen mit ihnen durchzuführen gestaltet sich in MATLAB besonders einfach. Beispiel: Es soll eine 3 X 3 Matrix erzeugt und in der Variable a gespeichert werden:

```
>> a = [ 1 2 3; 4 5 6; 7 8 9]
```

```
a =
     1     2     3
     4     5     6
     7     8     9
```

Mit Leerzeichen trennen wir verschiedene Spalten und mit einem Semikolon wird eine neue Zeile begonnen. Links im „workspace“ der Arbeitsumgebung wird für die neue Variable a eine Zeile mit Statusinformationen für diese Variable angezeigt (Abb.2).

Name	Size	Bytes	Class
 a	3x3	72	double array

Abb.2: Variablenzeile

Als Datentyp der Elemente haben wir double¹. Dies ist das Standardformat. Weitere Formate sind 16bit Integer und 8bit Integer. Es gibt noch weitere Strukturen, zum Beispiel Matrizen von Matrizen, darauf werden wir hier aber nicht weiter eingehen.

Weitere Beispiele zum Erzeugen von Vektoren:

```
>>x = [1 2 3] ergibt den Zeilenvektor x = 1 2 3
```

```
>>y = [1;2;3] ergibt den Spaltenvektor y =  $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ 
```

Es gibt viele verschiedene Möglichkeiten Vektoren und Matrizen einzugeben. Ein ausführliches Tutorial dazu findet sich in der MATLAB Einführung der FH Regensburg [2] auf Seite 8.

Auch gibt es viele Funktionen die zum automatischen Aufbau von Matrizen dienen². Eine davon ist `rand`.

Die Eingabe `rand(3)` erzeugt eine 3X3-Matrix mit Zufallswerten zwischen 0 und 1. Durch 2 Parameter erzeugt man nichtquadratische Matrizen (Cf.1):

```
>> rand(2,3)

ans =

    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
```

Cf.1: Nicht-quadratische Zufallsmatrix

`ans` steht für „answer“ und ist eine Art Puffervariable, die MATLAB erzeugt, wenn keine Ausgabevariable angegeben wird.

5.2 Der Doppelpunkt-operator³

Der Doppelpunkt ist einer der wichtigsten Operatoren. Er tritt in verschiedenen Zusammenhängen auf. Für

```
x = 1:5 bekommt man den Zeilenvektor ans = 1 2 3 4 5
```

Durch die Schreibweise `1:0.5:5` kann als Schrittgröße zwischen Anfangs und Endwert ein anderer Wert an Stelle von 1 gewählt werden:

```
ans = 1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000    4.5000    5.0000
```

Auch bei der Selektion von Zeilen und Spalten wird er benötigt.

1) Fließkommawerte

2) Mehr dazu: http://www.fi.uib.no/Fysisk/Teori/KURS/WRK/mat/chapter2_6.html#SECTION006000000000000000 [5]

3) ausführlichere Informationen zum „:“-Operator (englisch) <http://www.mathworks.com/access/helpdesk/help/techdoc/ref/colon.html> [1]

5.3 Selektion

Hier folgen einige Beispiele zur Selektion von Punkten, Vektoren und Submatrizen (Teilmatrizen) auf der Beispielmatrix aus 5.1 (siehe Cf.2). Mit einem Doppelpunkt wird angegeben das **alle** Werte aus der betreffenden Zeile bzw. Spalte zurückgegeben werden sollen. **Wichtig** zu wissen ist, dass MATLAB, wie bei der mathematischen Notation auch, beim **Index 1 beginnt** und nicht wie man es sonst vom Programmieren mit Java oder C gewohnt ist bei 0. Bei der Reihenfolge der Argumente gilt wie in der Mathematik: **Zeile vor Spalte**.

Beispielmatrix

```
a =
    1    2    3
    4    5    6
    7    8    9
```

Cf.2 Beispielmatrix

Einen Punkt abfragen

```
>>x = a(2,3)

x = 6
```

In Variable a wird der Wert von Zeile 2 und Spalte 3 abgefragt und der Variable x zugewiesen

Eine Zeile auswählen

```
>>a(3,:)

ans = 7 8 9
```

Alle Werte der Spalten in Zeile 3 der Beispielmatrix werden zurückgegeben. Das Ergebnis ist damit ein Zeilenvektor.

Spalte auswählen

```
>>a(:,1)

ans = 1
      4
      7
```

Alle Werte aus Spalte 1 werden zurückgegeben. Das Ergebnis ist ein Spaltenvektor.

Submatrix

```
>>a(1:2,1:2)

ans = 1 2
      4 5
```

Als Zeile und Spalte wird mit Hilfe des Doppelpunkt-Operators der Indexbereich 1 bis 2 angegeben. Als Ergebnis erhält man die entsprechende Teilmatrix.

Die Syntax zum löschen und ändern von Elementen, gestaltet sich ähnlich. Betrachten sie dazu folgende Beispiele:

```
a(1,1) = 9

a =
    9    2    3
    4    5    6
    7    8    9
```

Cf.3: a aus Cf.2 wird bei Zeilenindex 1 / Spaltenindex 1 der Wert 9 zugewiesen

```
a(2:3,2:3) = 1

a =
    9    2    3
    4    1    1
    7    1    1
```

Cf.4: Allen Werten von a (aus Cf.3) im Indexbereich 2 bis 3 (Zeile und Spalte) wird der Wert 1 zugewiesen

```
a(:,3) = []

a =
    9    2
    4    1
    7    1
```

Cf.5: Allen Elementen der 3. Spalte von a (aus Cf.4) wird eine leere Matrix zugewiesen, d.h. die Elemente sind leer und die Spalte fällt somit weg

```
a(:,3) = [1;2;3]

a =
    9    2    1
    4    1    2
    7    1    3
```

Cf.6: Eine Spalte deren Werte durch den Spaltenvektor auf der rechten Seite bestimmt werden, wird der Matrix a (aus Cf.5) hinzugefügt

5.4 Operatoren und Punktsyntax

MATLAB ermöglicht selbstverständlich arithmetische Operationen wie +, -, *, /. Dabei ist allerdings folgender Punkt zu beachten: Das multiplizieren zweier Matrizen a,b durch **a*b** hat eine **Matrixmultiplikation** zur Folge (Cf.7 bis 9). Will man aber die **einzelnen** Werte einer Matrix manipulieren so verwendet man die Punktsyntax. Siehe dazu Codefragment 10. Analog dazu verwendet man die Punktsyntax bei den anderen Operationen.

```
>> a = [1 2 3; 1 2 3; 1 2 3]

a =
    1    2    3
    1    2    3
    1    2    3
```

Cf.7: Matrix a

```
>> b = [1;2;4]

b =
    1
    2
    4
```

Cf.8: Matrix b

```
>> a*b

ans =
    17
    17
    17
```

Cf.9: Matrixmultiplikation

```
>> a.*a

ans =
    1    4    9
    1    4    9
    1    4    9
```

Cf.10: Punktweise Multiplikation

5.5 Funktionen und Konstanten

MATLAB bietet eine große Anzahl elementarer mathematischer Funktionen. Das selbe gilt für mathematische Konstanten und Matrixfunktionen der linearen Algebra. Dazu einige wenige Beispiele (Cf.11 bis 13). Der volle Funktionsumfang dieser Bereiche kann der Online-Dokumentation von MATLAB entnommen werden¹.

```
>> sqrt(9)

ans =
    3
```

Cf.11: Wurzelfunktion

```
>> pi

ans =
    3.1416
```

Cf.12: Konstante PI

```
>> a'

ans =
    1    4    7
    2    5    8
    3    6    9
```

Cf.13: Transponierung (Spiegelung der Werte an der Diagonalen) von a (Cf.2)

1) Der Link zu dieser Sektion: http://www.mathworks.com/access/helpdesk/help/techdoc/ref/func_b10.html#mathematics [1]

5.6 Die Funktionen „min“ und „max“

Mit den Befehlen `min(x)`, bzw. `max(x)` lässt sich das kleinste, bzw. **größte Element eines Vektors** X finden. Handelt es sich bei X jedoch um eine **Matrix**, so bekommt man einen **Zeilenvektor** zurück der die kleinsten, bzw. größten Elemente aus jeder Spalte enthält.

Es ist also **zu beachten**, dass einige Befehle unterschiedlich funktionieren **je nachdem** ob sie auf **Skalare, Vektoren** oder **Matrizen** angewendet werden.

6 Grafiksystem

MATLABs herausragendem Grafiksystem wollen wir im Folgenden besondere Aufmerksamkeit schenken. Es enthält einfach zu bedienende Funktionen zur zwei- und dreidimensionalen Darstellung von Daten aller Art¹. Wir beginnen mit einem simplen Beispiel:

6.1 Der plot-Befehl

Die Darstellung der Funktion $y = x^2$ in folgendem Beispiel soll den Umgang mit dem Befehl `plot` veranschaulichen. Zunächst definieren wir einen Vektor mit den Elementen 1 bis 10, der uns als x-Achse dienen soll:

```
>> x = 1:5;
```

Der folgende Befehl weist einem neuen Vektor `y` die punktweise quadrierten Werte von `x` zu:

```
>> y = x.^2
```

```
y = 1 4 9 16 25
```

Mit folgendem Befehl werden nun die Werte aus `y` gegen die Werte von `x` gezeichnet.

```
>> plot(x,y)
```

Mit folgenden Befehlen werden der Plot und die Achsen beschriftet. Das Ergebnis ist auf Abb.3 zu sehen.

```
>> xlabel('x')
>> ylabel('y')
>> title('y = x^2')
```

Es ist auch möglich mehrere x-y Paare in ein Grafikfenster zeichnen zu lassen, wie folgendes Beispiel zeigt (Abb.4):

```
>> x = 1:0.1:5;
>> y1=x.^2;
>> y2=x.^3;
>> y3=x.^4;
>> plot(x,y1,x,y2,x,y3)
```

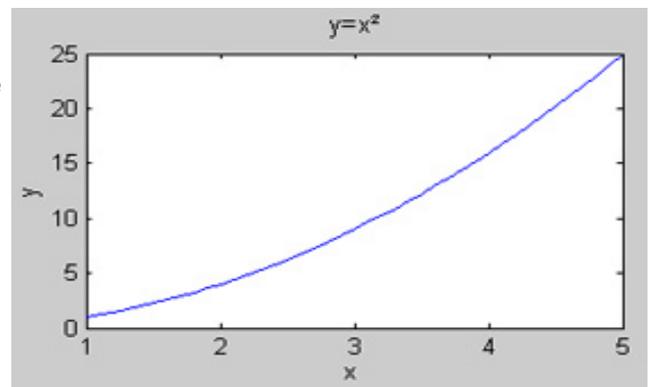


Abb.3: Plot der Funktion $y = x^2$

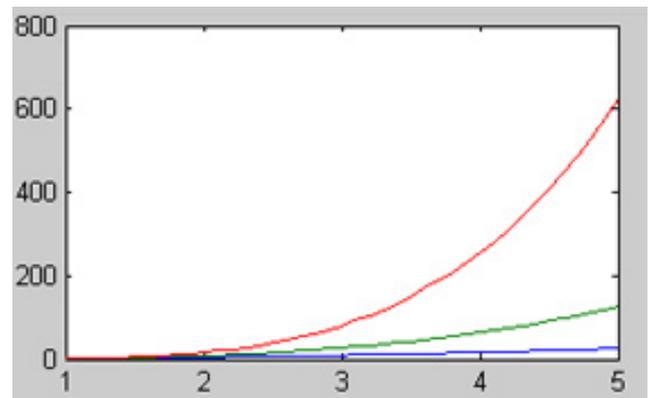


Abb.4: Plot der Funktion $y = x^2$, $y = x^3$ und $y = x^4$

6.2 3-dimensionale Funktionen, Subplots und hold

In den folgenden Beispielen werden einige Vorgehensweisen zur Darstellung von Funktionen im 3-dimensionalen Raum vorgestellt. In Beispiel 1 wird das Grundprinzip zur Darstellung überlagerter Funktionen gezeigt. Als Basis für die Darstellung von Funktionen im Raum benötigen wir Matrizen an Stelle von Vektoren. Die Funktion `meshgrid` liefert die nötigen Matrizen für unsere Zwecke (siehe Codefragment rechts). Sie erzeugt die Matrix `x`, deren Zeilen aus vertikal aneinandergereihten Kopien des im ersten Argument definierten Vektors bestehen und die Matrix `z`, deren Spalten aus horizontal aneinandergereihten Kopien des zweiten Vektors bestehen. Die Anzahl der Zeilen und Spalten ist so gewählt das `x` und `y` gleicher Gestalt sind.

```
[x,y] = meshgrid(1:2,1:3)
x = 1 2
    1 2
    1 2
y = 1 1
    2 2
    3 3
```

¹) mehr zu Visualisierung unter: <http://www.mathworks.com/access/helpdesk/help/techdoc/visualize/visualize.html> [1]

Beispiel 1

```
[x,y] = meshgrid(1:5)

x =
    1     2     3     4     5
    1     2     3     4     5
    1     2     3     4     5
    1     2     3     4     5
    1     2     3     4     5

y =
    1     1     1     1     1
    2     2     2     2     2
    3     3     3     3     3
    4     4     4     4     4
    5     5     5     5     5
```

Nun soll zunächst einmal die Funktion $z = x^2$ im Raum dargestellt werden um das Prinzip zu veranschaulichen (Abb.5):

```
>> z=x.^2;
>> mesh(z)
```

Bei den Achsenmarkierungen handelt es sich nicht um die Werte von x sondern um die Indizes der Matrix z. Die Matrix x bildet hier lediglich die Grundlage zur Berechnung von z¹. Jedem Element aus z wird ein Punkt im Raum zugeordnet. Die Funktion z verbindet all diese Punkte zu einem Gitter.

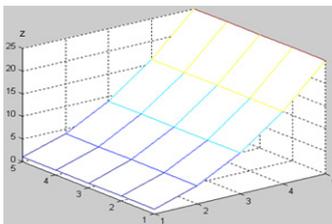


Abb.5: Plot der Funktion $z = x^2$

Um nun eine Funktion von 2 Variablen anzuzeigen, bezieht man einfach die Matrix y mit in die Berechnung von z ein (Abb.6):

```
>> z = x.^2 + y.^2

z =
     2     5    10    17    26
     5     8    13    20    29
    10    13    18    25    34
    17    20    25    32    41
    26    29    34    41    50
```

```
mesh(z)
```

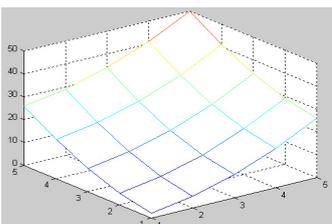


Abb.6: Plot der Funktion $z=x^2+y^2$

Beispiel 2

Jetzt das ganze, in einem etwas komplizierteren Beispiel. Diesmal wollen wir eine Sinusfunktion mit einer quadratischen Funktion überlagern, die Achsen lassen wir außerdem im negativen Bereich beginnen:

```
[x,y] = meshgrid(-2:0.1:2);
```

Zunächst sollen beide Funktionen separat angezeigt werden, dazu wird die Funktion subplot² verwendet. Folgendes Codefragment zeigt beide Funktionen im selben Fenster (Abb.7):

```
>> z = sin(x);
>> subplot(1,2,1)
>> mesh(z)

>> z = y.^2;
>> subplot(1,2,2)
>> mesh(z)
```

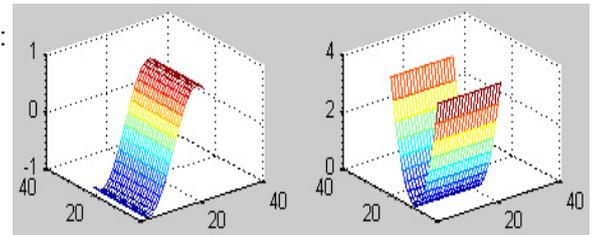


Abb.7: Unterteilung der Ausgabe mittels subplot

Die ersten 2 Argumente des subplot-befehls spalten das Grafikenfenster in eine 1x2 Matrix auf, mit dem 3. Argument wird angegeben, an welcher Position der nächste Plot stehen soll.

Um 2 Plots in ein und das selbe Achsensystem zu zeichnen wird der Befehl hold verwendet. Betrachten sie das folgende Codefragment und die zugehörige Abb.8. Zusätzlich soll für diese Ausgabe ein neues Grafikenfenster geöffnet werden, dies erreicht man mit dem Befehl figure.

```
>> figure(2)
>> z = sin(x);
>> mesh(z)

>> hold on
>> z = y.^2;
>> mesh(z)
```

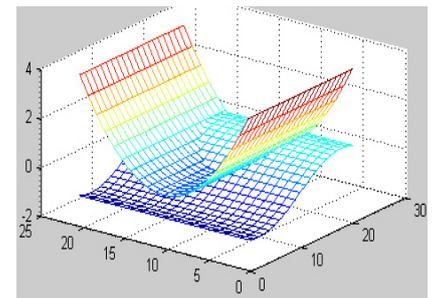


Abb.8: Mittels hold überlagerte Ausgabe

Jetzt sollen sich diese beiden Funktionen **additiv überlagern**. Es soll in das aktuelle Grafikenfenster gezeichnet werden, die hold-Funktion wird daher deaktiviert.

Statt dem Befehl mesh benutzen wir surf, der einzige Unterschied zu mesh besteht darin das eine Oberfläche zum Drahtgitter hinzu gerendert wird (Abb.9):

```
>> hold off
>> z=sin(x)+y.^2;
>> surf(x,y,z)
```

Die ersten 2 Argumente in surf sorgen dafür das als Achsenwerte die Werte der Matrizen x und y benutzt werden, wie man an den Achsenwerten in Abb.9 erkennen kann.

Diese Schreibweise mit 3 Argumenten funktioniert auch mit dem Befehl mesh.

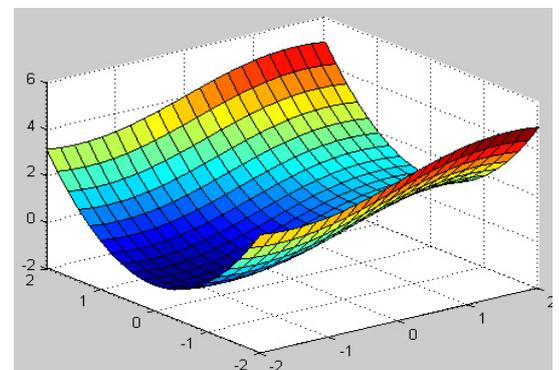


Abb.9: Plot der Funktion $y = x^2$

1) Es ist auch möglich statt der fortlaufend nummerierten Indizes die eigentlichen Werte der zu Grunde liegenden Vektoren/Matrizen anzuzeigen, wie im zweiten Beispiel gezeigt wird.

2) mehr dazu unter <http://www.mathworks.com/access/helpdesk/help/techdoc/ref/subplot.html> [1]

7.1 Bilder einlesen und anzeigen

Zunächst wählen wir den richtigen Ordner in „Current Directory“ aus, damit unser Bild auch gefunden wird. Der Befehl für das einlesen von Bildern heisst `imread`¹. Im Vortrag Lokale Operatoren[6] haben wir gesehen dass digitale Bilder nichts anderes sind, als eine 2-dimensionale Anordnung von Werten. Man kann sie also als Matrix betrachten und als solche können wir sie in MATLAB laden und wie mit jeder anderen Matrix damit arbeiten.

Für unsere Zwecke benötigen wir ein Graustufenbild. Wenn ein Graustufenbild vorliegt schreiben wir einfach:

```
>> Bild = double(imread('blume.tif'));
```

Bei „Bild“ handelt es sich um die Variable in welche die Bildinformationen gespeichert werden. Den Dateinamen schreiben wir in Hochkommata da es sich um einen String handelt. Dem geklammerten Ausdruck steht „double“ voran, weil wir die 8-bit bzw. 16-bit Werte die wir vom Bild bekommen in Fließkommawerte konvertieren wollen, um später besser damit rechnen zu können.

Wichtig: Das **Semikolon** nach der Anweisung, **verhindert** das alle Werte noch mal angezeigt werden, wie es normalerweise nach einer Zuweisung der Fall ist. Bei großen Bildern kann das nämlich recht lange dauern.

7.2 Bild anzeigen

Wenn eine Matrix als Bild angezeigt werden soll schreibt man `image(<variablenname>)`, also in unserem Fall `image(Bild)`. Ein neues „Figure“-fenster öffnet sich und zeigt das Bild an. Es kann sein, dass das Bild in Rot und Blaufarben angezeigt wird, das liegt daran das MATLAB versucht die Werte durch Falschfarben darzustellen, damit man die Unterschiede zwischen den Werten besser erkennen kann. Das ist für 3D-Plots interessant, wir wollen aber ein Graustufenbild und können dies mit der Eingabe von `colormap(gray(256))` erreichen². Solche Befehle wie `colormap` beziehen sich dann stets auf das aktuell geöffnete Grafikfenster. Ein weiterer Befehl zum anzeigen von Bildern ist `imshow`. Mit `imshow(Bild, [])` wird das Bild zusätzlich in seinem Tonwert gespreizt, d.h. der dunkelste Wert wird zu Schwarz und der hellste zu Weiß. Der Befehl `imshow` ist aber nur verfügbar wenn das Digital Image Processing Toolkit installiert wurde.

8.1 Anwendungsbeispiel 1 - Bilder als Matrizen

Nun wollen wir uns einem Beispiel zum Thema Bildbearbeitung zuwenden in dem wir einige der vorgestellten Methoden im Umgang mit Matrizen anwenden wollen und Vergleiche zu Programmen wie Photoshop ziehen.

Aus Photoshop kennen wir das Prinzip der „Maskierung“ [7]. Eine Ebenenmaske ist nichts anderes als ein Graustufenbild, welches mit dem zu maskierenden Bild verrechnet wird (Abb.10).

In diesem Beispiel soll der dunkle Bereich um die Blume aus Abbildung 10 selektiert und komplett auf die Farbe schwarz reduziert werden.



Abb.10: Masken in Photoshop



Abb.11: Das Ausgangsmotiv

Wir laden ein Graustufenbild:

```
Bild = double(imread(blume.tif));
```

Zeigen wir es mittels `image(Bild)` an sehen wir das Motiv auf Abbildung 11.

Um einen geeigneten Schwellenwert für die Selektion des dunklen Bereiches zu finden, soll ein Histogramm des Bildes angezeigt werden, wie man es auch aus Photoshop kennt. Das Histogramm zeigt an wie oft der jeweilige Wert in der Matrix vorkommt. Damit das Histogramm richtig angezeigt wird müssen wir unsere Bildmatrix zunächst in einen Vektor umwandeln, da wir als zweite Achse die Werte 0 bis 256 haben wollen.

1) Die meisten gängigen Formate wie jpg, tif etc. werden unterstützt. Um genaueres zu erfahren geben Sie „`help imread`“ ein

2) Da die Grauwerte von 0 bis 256 gehen bekommen wir somit ein ganz normales Graustufenbild, da jeder Wert dem Bereich den wir angegeben haben zugeordnet wird. Um mehr über `colormaps` zu erfahren geben Sie „`help colormap`“ ein.

Diese Umwandlung zum Vektor ist bei MATLAB in vielen Situationen hilfreich.

```
Vektor = Result(:);
```

Das Histogramm sehen wir, wenn wir nun schreiben:

```
hist(Vektor,10)
```

Mit dem 2. Argument geben wir die Anzahl der **bins** (engl. für „Behälter“) an, d.h. dass sich die Werte von 0 bis 256 auf 10 Balken im Histogramm verteilen sollen. Auf dem Ergebnis auf Abb.12 ist zu erkennen das sich viele Werte um den Bereich des Grauwertes 75 sammeln. Da der Kontrast zwischen hell und dunkel auf dem Bild recht stark ist, kann man davon ausgehen das sich die gewünschten dunklen Pixel von null bis zu eben diesem Bereich befinden.

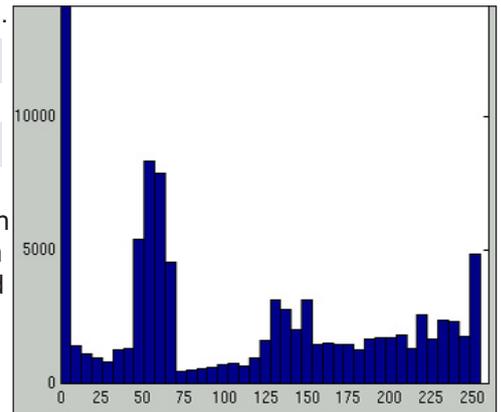


Abb.12: Histogramm

Um diese dunklen Bereiche nun auf 0 zu reduzieren speichern wir zunächst das Ergebnis folgenden Ausdrucks in einer Variable x:

```
x = Bild > 75
```



Abb.13: Matrix X, bestehend aus den Werten 0 und 1

Wir haben nun eine Matrix, die genauso groß ist wie unser Bild und für jeden Wert der im Ursprungsbild größer als 75 ist, eine 1 (da die Bedingung >75 erfüllt war) und für alle anderen eine 0 (1 entspricht „wahr“ und 0 „falsch“). Es handelt sich bei X damit um ein sog. „logisches Array“ was aber die weiteren Berechnungen damit nicht behindert. Zeigen wir diese Matrix als Bild an, bekommen wir, wie auf Abb.13 zu sehen, ein Schwarzweißbild, was Photoshopnutzer an Ebenenmasken und Alphakanäle erinnern wird (vgl. Abb.10).

Was passiert nun wenn man die Variablen Bild und X punktweise multipliziert und das Ergebnis anzeigt?

```
Result = Bild.*X;
image(Bild)
colormap(gray(256))
```

In Abb.14 sehen wir, dass die dunklen Bereiche um die Blume nun komplett schwarz sind da ja alle Grauwerte kleiner als 75 mit einer 0 aus der Matrix X multipliziert wurden. Der neue Wert ist dann 0, was in der `colormap(gray(256))` schwarz entspricht. Die Werte über dem Schwellenwert 75 wurden mit der in X korrespondierenden 1 multipliziert und blieben unverändert. Jetzt wird auch deutlich wie Ebenenoperationen in Photoshop funktionieren. Die Ebenen a und b werden punktweise nach dem gewählten Kriterium miteinander verrechnet (Abb.15).



Abb.14: Umgebender Bildbereich ist schwarz

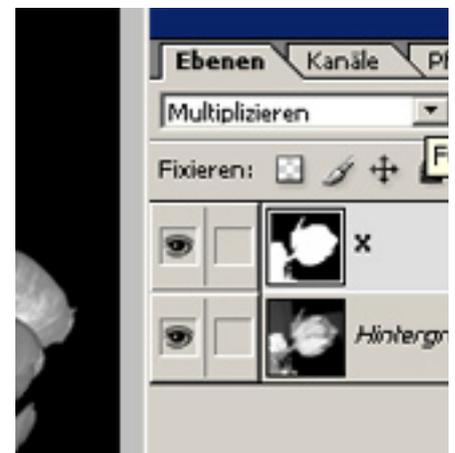


Abb.15: Ebenenberechnung in Photoshop liefert das selbe Ergebnis

Nun wollen wir mittels dem Befehl `mesh` noch einen etwas anderen Blickwinkel auf unser Bild werfen.

Die `colormap` weist den dem Wertebereich einen Farbverlauf zu, der die Darstellung der Höhenunterschiede unterstützt.

```
mesh(Result)
colormap(hot)
```

Mehr zu verschiedenen `colormaps` und deren Verwendung erfährt man durch die Eingabe von `help colormap`.

Alle Werte unter 75 wurden auf 0 reduziert was in unserer `colormap` der Farbe schwarz entspricht so kommt die ebene schwarze Fläche zu Stande. Auf Abb.16 sehen wir das Ergebnis.

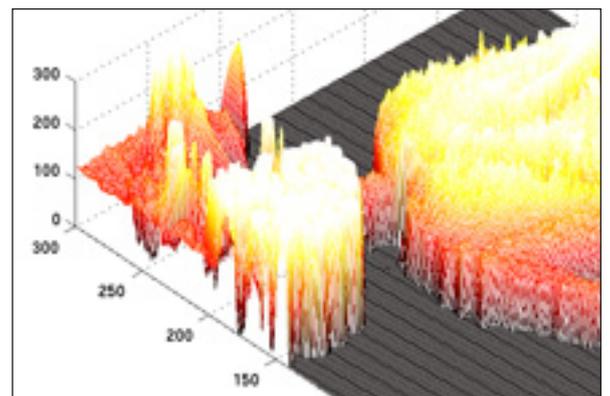


Abb.16: Abb.14 in der Mesh-Visualisierung

8.2 Anwendungsbeispiel 2 - Lineare Operationen

Aus Photoshop kennen wir lineare Filter, wie zum Beispiel den Gaußschen Weichzeichner (Abb.17) und im Vortrag „Lokale Operationen“ [6] haben wir gesehen wie sie funktionieren. Ein quadratisches Operatorfenster wird pixelweise über das Originalbild verschoben und je nach Struktur des Filterkernels bekommt das Zielpixel einen neuen Wert. Die Anwendung eines solchen Filters wollen wir nun in diesem Beispiel an Hand von MATLAB nachvollziehen. Zunächst öffnen wir wieder ein Graustufenbild (siehe Abb.18).

```
>>Bild = double(imread('haus2.tif'));
```

Dann selektieren wir zu Demonstrationszwecken einen kleineren Ausschnitt des Bildes (Abb.19). Dies entspricht etwa dem Freistellenwerkzeug in Photoshop.

```
Auswahl = Bild(100:200,100:250);
```

Jetzt brauchen wir einen Filterkernel (anderes Wort für Faltungsmaske), also nichts anderes als eine Matrix.

```
FX = [1 2 1; 2 4 2; 1 2 1]
```

Dieser Filter entspricht einem Gaußschen Weichzeichner. Man bemerke wie die Pixel außenherum schwächer berücksichtigt werden als das Zielpixel, im Gegensatz zu einem Mittelwertoperator (vgl. Vortrag „Lokale Operationen“ [6]).

Wir wollen die Luminanz (Helligkeit) des Bildes erhalten. Dazu müssen wir den Kernel normieren d.h. die Summe der Elemente muss 1 ergeben. Dazu teilen wir alle Werte durch die momentane Summe¹.

```
FX = FX ./ sum(FX(:))
```

```
FX =
[ 0.0625 0.1250 0.0625
  0.1250 0.2500 0.1250
  0.0625 0.1250 0.0625 ]
```

MATLABs Digital Image Processing Toolkit (auf unseren Poolrechnern installiert) stellt den Befehl `filter2` zur Verfügung. Seine Funktion besteht lediglich darin diesen Kernel auf das Bild anzuwenden (Siehe Vortrag „Lokale Operatoren“ [6]). Das Ergebnis sehen wir auf Abb.20.

```
Result = filter2(FX,Auswahl);
```



Abb.18: Ursprüngliches Bild



Abb.19: Bild nach Selektion



Abb.20: Bild nach dem Filtern

9. Zusammenfassung

Die in diesem Dokument vorgestellten Bereiche und Beispiele stellen natürlich nur einen kleinen Ausschnitt von MATLABs Möglichkeiten dar. Es dient primär dazu, Neueinsteigern die grundlegendsten Kenntnisse zum Umgang mit dem Programm zu vermitteln und einen Einblick in die Vielzahl der verschiedenen Anwendungsmöglichkeiten zu ermöglichen. Es wurde das elementare Arbeiten mit Matrizen und Funktionen gezeigt, einige Beispiele zur Visualisierung erläutert und Techniken zur digitalen Bildbearbeitung vorgestellt. Aufbauend darauf sind zur Vertiefung des Wissens in jede dieser Richtungen die für diesen Vortrag verwendeten Quellen zu empfehlen.

10. Quellen

- [1] Online-Dokumentation zu MATLAB (Zugriff: 27.04.2004) <http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html>
- [2] Getting Started with MATLAB (deutsche Übersetzung) von der FH Regensburg. (Zugriff: 27.04.2004) <http://homepages.fh-regensburg.de/~wah39067/Matlab/MTut2-0.pdf>
- [3] MATLAB Primer - Sehr ausführliche Publikation der Universität Florida (englisch). (Zugriff: 27.04.2004) <http://ise.stanford.edu/Matlab/matlab-primer.pdf>
- [4] MATLAB-Einführung der UNI-Hamburg (Zugriff: 27.04.2004) <http://www.rz.uni-hamburg.de/RRZ/W.Wiedl/Skripte/Matlab/index.html>
- [5] MATLAB-Einführung, Kapitel zum Aufbau von Matrizen (Zugriff 27.04.2004) http://www.fi.uib.no/Fysisk/Teori/KURS/WRK/mat/chapter2_6.htm#SECTION006000000000000000
- [6] Tobias Lübke, Proseminar Computer Vision SS 2004: Elementare Bildverarbeitung – lokale Operationen <http://www.informatik.uni-ulm.de/ni/Lehre/SS04/ProsemCV/pdfs/tluebke.pdf>
- [7] BR - Bildbearbeitung mit Adobe Photoshop (Zugriff 6.05.2004) <http://www.br-online.de/wissen-bildung/thema/alpha-bildbearbeitung/popups/br06.html>

1) `FX(:)` wandelt die Matrix FX in einen Vektor um (wie in 8.1 schon gezeigt wurde) und die Funktion „`sum`“ zählt die Elemente eines Vektors zusammen.