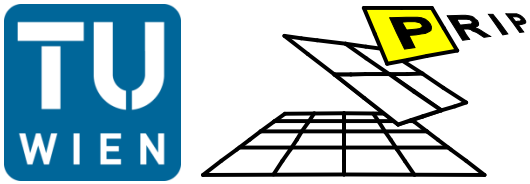
MATLAB Einführung



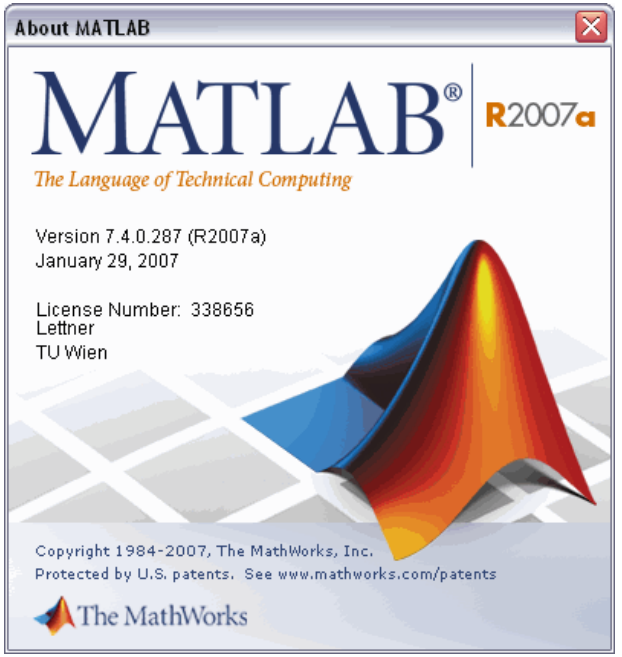
Überblick

- **Was ist MATLAB?**
- **Matrizen und Vektoren**
- **Datentypen und Programmsteuerung**
- **Skriptprogramme und Funktionen**
- **Bilder in MATLAB**
- **MATLAB-optimiert programmieren**
- **Visualisierung**

MATLAB I

- MATLAB ist eine Entwicklungsumgebung für technische Berechnungen. Es integriert Berechnungen, Visualisierung und Programmierung in einer einfachen Umgebung.

- Anwendungsbereiche:
  - **Technische Berechnungen**
  - **Entwicklung v. Steuerungen u. Regelungen**
  - **Nachrichtentechnik**
  - **Bildverarbeitung**
  - **Messtechnik und Testen**
  - **Finanzdatenmodellierung und -analyse**



MATLAB II

- Sehr kompakter Programmierstil
  - **Kurze Entwicklungszeit**
  - **Mächtige Befehle (Manipulation vieler Daten mit einem einzigen Befehl)**
  - **Keine Variablen Deklaration**

- Verwendung
  - **Interaktive Eingabe von Befehlen**
  - **Skriptprogramme (Batch-Dateien)**
  - **MEX-Funktionen (Einbinden von C-Code)**

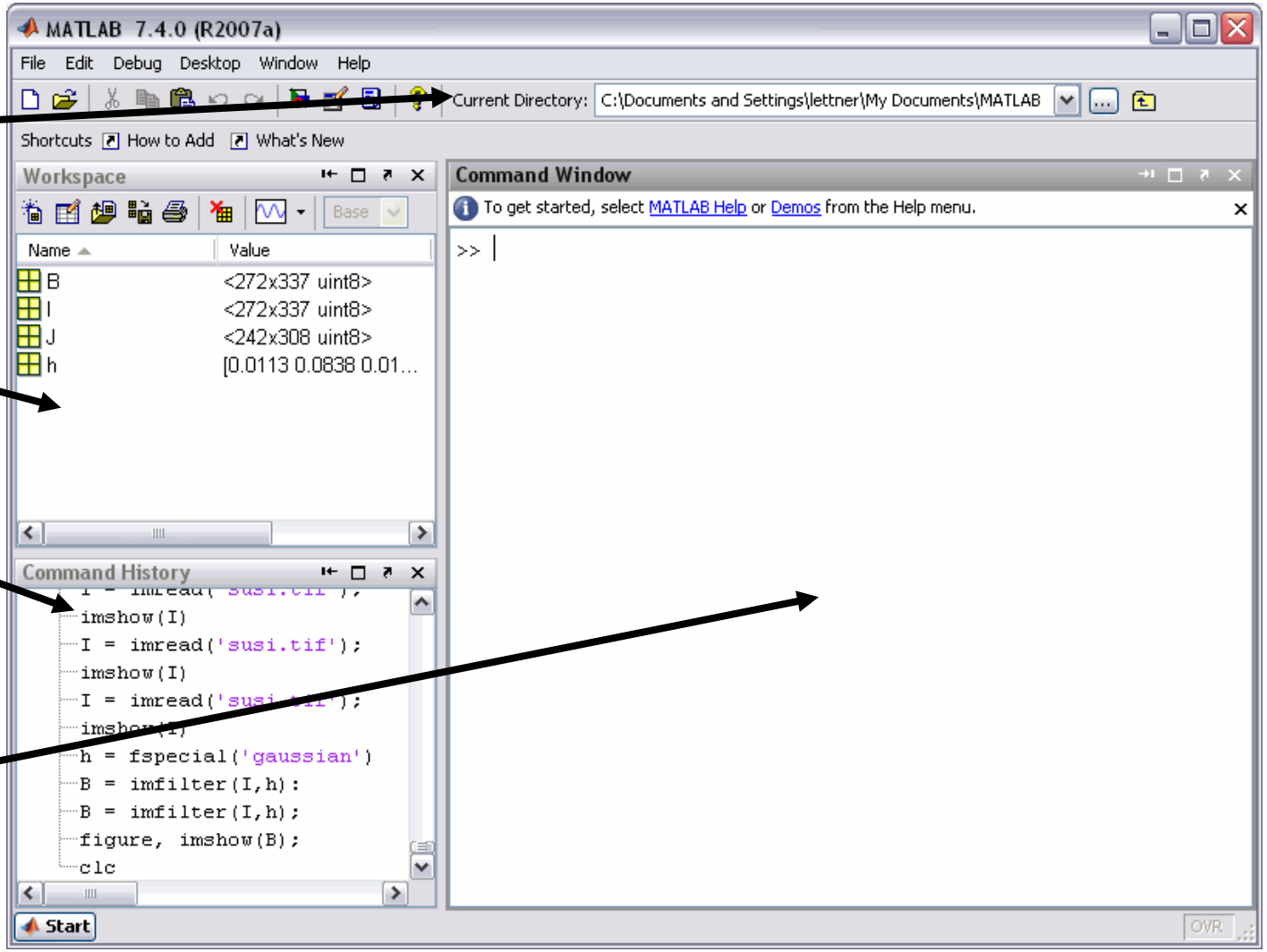
MATLAB III

Working Directory

Workspace

Command History

Command Window



Matrizen I

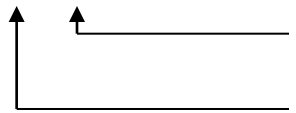
- **Erzeugen von Matrizen:**

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

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

- **Null- Bzw. Einsmatrix:**

```
>> B = zeros(2,4)
>> C = ones(1,3)
```



$$B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

- **Diagonalmatrizen:**

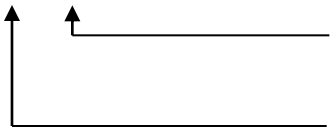
```
>> D = diag(1:3)
```

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Matrizen II

- **Indizes beginnen bei 1**
- **Zugriff auf die Elemente:**

```
>> A = zeros(2,5);
>> A(2,3) = 5;
```



- **Beim Überschreiten der Indexgrenzen wird die Matrix angepasst:**

```
>> A(3,1)=2
```

- **Matrixbereiche ansprechen:**

```
>> A(1:2,1:2) = ones(2,2);
>> A(1:end,[2 4]) = 9;
```

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 5 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 0 & 9 & 0 & 9 & 0 \\ 0 & 9 & 5 & 9 & 0 \\ 2 & 9 & 0 & 9 & 0 \end{pmatrix}$$

Vektoren

- **Zahlenreihen:**

  von:schrittweite:bis* bzw. *von:bis

```
>> D = 1:5;
>> E = 5:-0.5:4
```

- **Vektoren transponieren :**

```
>> D1=D';
```

- **Vektoren erzeugen:**

```
>> linspace ( a, b, n );
```

  Erzeugt Vektor mit *n* Elementen, von *a* bis *b*.

- **Elementweiser Zugriff: A(index)=wert;**

```
>> D(3)=100;
```

$$D = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

$$E = \begin{pmatrix} 5 & 4{,}5 & 4 \end{pmatrix}$$

$$D1 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$$

$$D = \begin{pmatrix} 1 & 2 & 100 & 4 & 5 \end{pmatrix}$$

Datentypen (Klassen)

Name	Description
`double`	Double-precision, floating-point numbers in the approximate range -10^{308} to 10^{308} (8 bytes per element).
`uint8`	Unsigned 8-bit integers in the range $[0, 255]$ (1 byte per element).
`uint16`	Unsigned 16-bit integers in the range $[0, 65535]$ (2 bytes per element).
`uint32`	Unsigned 32-bit integers in the range $[0, 4294967295]$ (4 bytes per element).
`int8`	Signed 8-bit integers in the range $[-128, 127]$ (1 byte per element).
`int16`	Signed 16-bit integers in the range $[-32768, 32767]$ (2 bytes per element).
`int32`	Signed 32-bit integers in the range $[-2147483648, 2147483647]$ (4 bytes per element).
`single`	Single-precision floating-point numbers with values in the approximate range -10^{38} to 10^{38} (4 bytes per element).
`char`	Characters (2 bytes per element).
`logical`	Values are 0 or 1 (1 byte per element).

Programmfluss I

Statement	Description
`if`	`if`, together with `else` and `elseif`, executes a group of statements based on a specified logical condition.
`for`	Executes a group of statements a fixed (specified) number of times.
`while`	Executes a group of statements an indefinite number of times, based on a specified logical condition.
`break`	Terminates execution of a `for` or `while` loop.
`continue`	Passes control to the next iteration of a `for` or `while` loop, skipping any remaining statements in the body of the loop.
`switch`	`switch`, together with `case` and `otherwise`, executes different groups of statements, depending on a specified value or string.
`return`	Causes execution to return to the invoking function.
`try...catch`	Changes flow control if an error is detected during execution.

Programmfluss II

```
for index = start : increment : end
        statements
end
```

```
while expression
        statements
end
```

```
if expression1
        statements1
elseif expression2
        statements2
else
        statements3
end
```

Skriptprogramme

- In Skriptprogrammen können alle Befehle verwendet werden, die auch interaktiv zur Verfügung stehen.

- Skriptprogramme sind ASCII-Dateien (*.m)
- Aufruf in MATLAB:
  `>> Programmname`
- Kommentare: `%`
- Nach jedem Befehl ein optionales Semikolon.
  Ohne Semikolon wird das Ergebnis ausgegeben.

Funktionen

- **Aufruf mit Argumenten, Returnwerte als Ergebnis**

- **Definition :** *function [outputs] = name (inputs)*

```
function [s, p] = sumprod ( f, g )
```

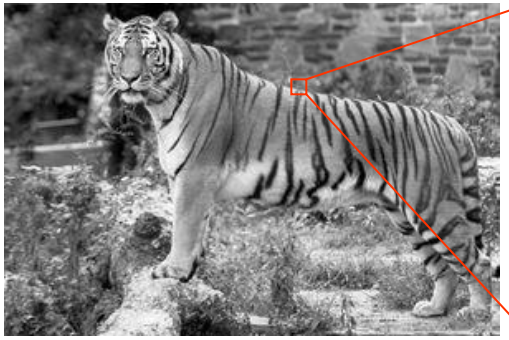
- **Aufruf:**

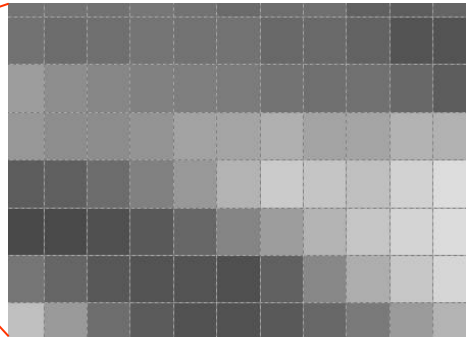
```
>> [s, p] = sumprod (f, g );
```

Digitale Bilder

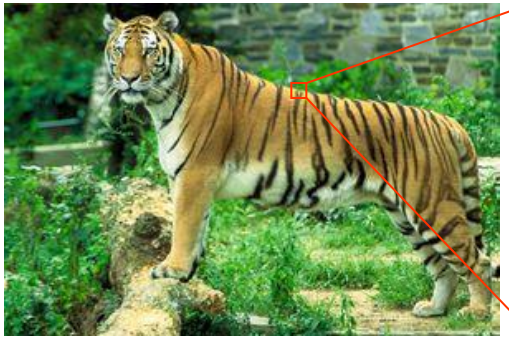
- **2D Funktion, *f(x,y)* :** **- *x, y* : Koordinaten**
  - *f* : Amplitude, Intensität

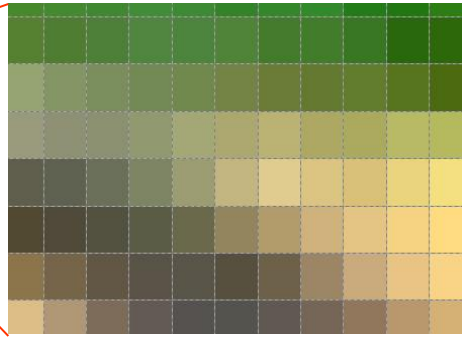
Bild *Detail* *Interne Darstellung*

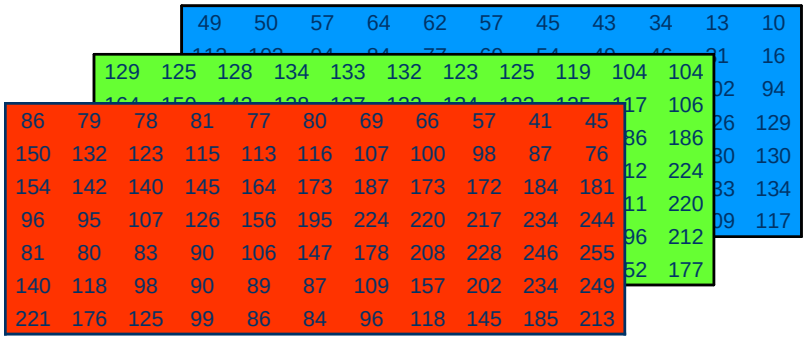




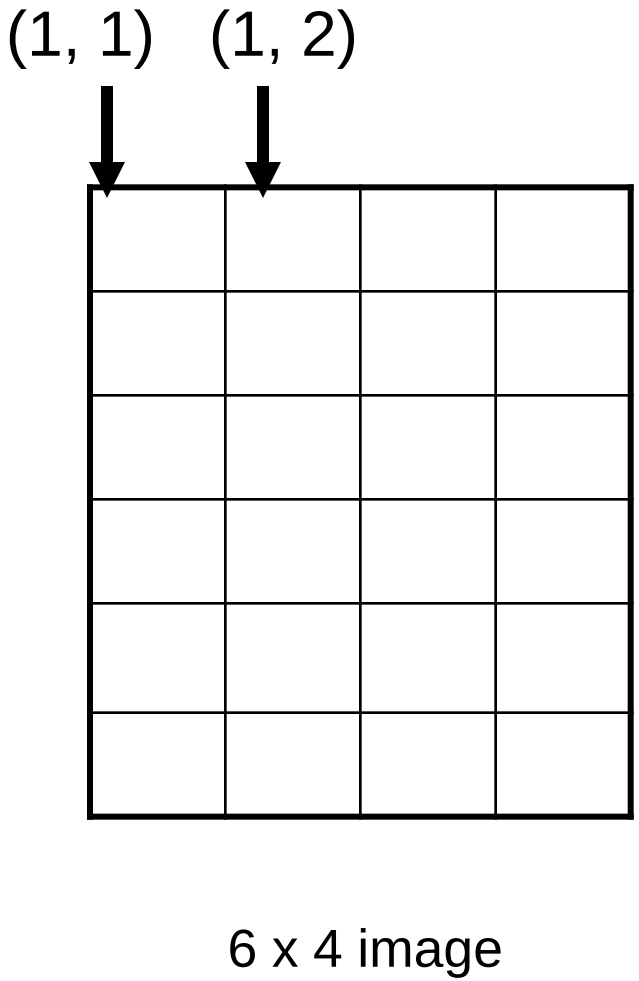
107	103	105	110	108	108	98	98	91	75	76
154	139	131	125	123	120	111	107	108	98	87
151	141	140	146	161	163	174	162	162	176	174
93	95	108	127	152	180	203	196	192	209	219
73	74	80	89	102	133	157	181	199	212	220
119	103	88	84	85	80	98	137	175	200	214
193	155	111	92	83	83	90	105	124	157	181







Koordinaten, Richtlinien



6 x 4 image

- ***m* Zeilen and *n* Spalten**
- **MATLAB Befehl für # Zeilen, # Spalten**

```
>> [m, n] = size( f );
```

- **Intensität**
  - [0, 255], [0, 65535]...
- **Binärbild**
  - 0, 1

Bilder Einlesen und Anzeigen

- ***imread ('filename'):* Öffnet ein Bild und speichert es in einer Matrix**

```
>> I = imread('pout.tif');
>> imshow ( I, G );
```

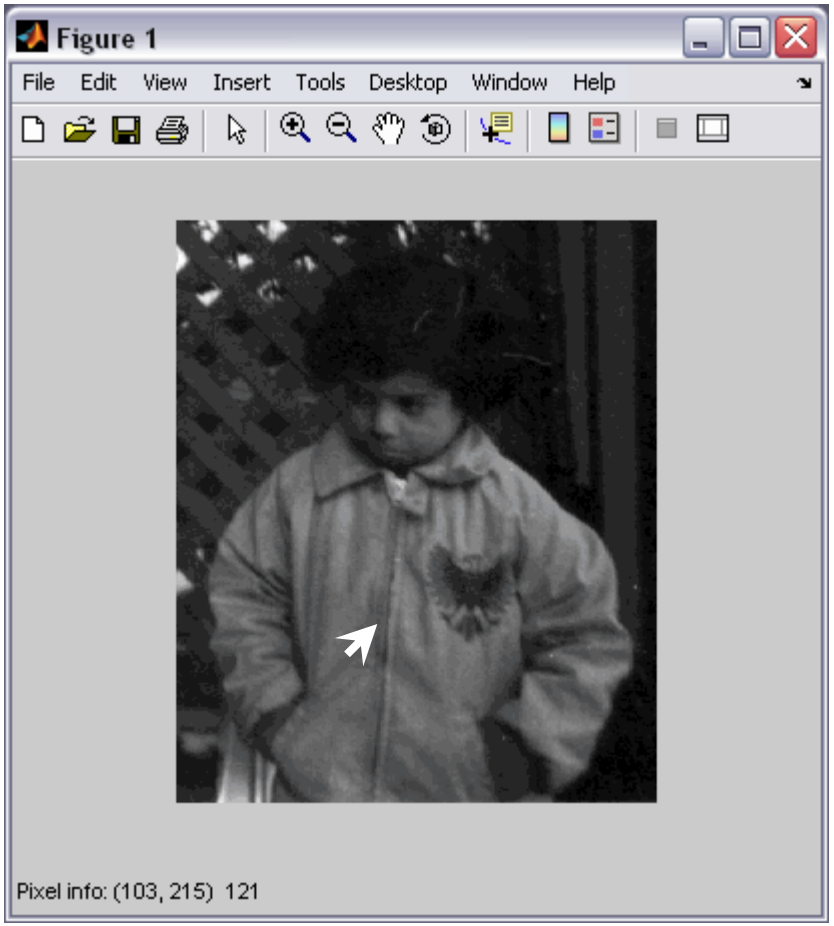
f : image array

G : # Intensitätswerte

```
>> imshow ( f, [low high] );
>> imshow ( f, [] );
```

- **Anzeigen der Werte einzelner Pixel**

```
>> impixelinfo
```



- **Vektor/Matrix statt for/while Operationen:**
- ***Beispiel:***

  f(x) = a sin (x / 2*PI), für x = 0, 1, 2, .., M-1

```
>> for x = 1:M
>>   f = a * sin ( ( x-1) / ( 2 * pi ) );
>> end
```

- **Besser:**

```
>> x = 0:M-1;
>> f = a * sin ( x / ( 2 * pi ) );
```

Vektorisierung II

- **‘.’ Unterscheidet zwischen Matrix vs. Array Operationen:**
  - *A* * *B* : Matrixmultiplikation
  - *A* .* *B* : Arraymultiplikation

- **Beispiel**

```
>> A = [1 2; 3 4]
>> B = [4 5 6; 7 8 9]
>> A*B
   ans =
        18    21    24
        40    47    54
>> A.*A
   ans =
         1     4
         9    16
```

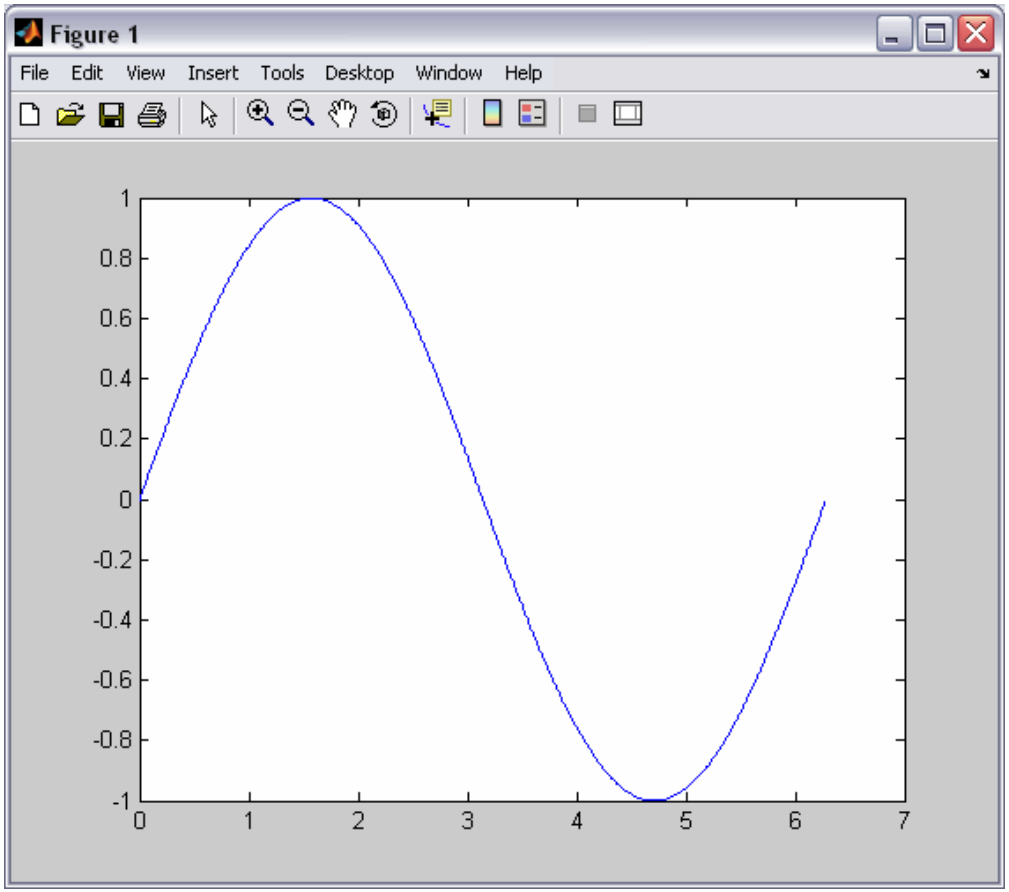
Visualisierung 2D

- **2D Plots**

```
>> t = 0:0.01:2*pi;
>> y = sin(t);
>> plot(t,y);
```

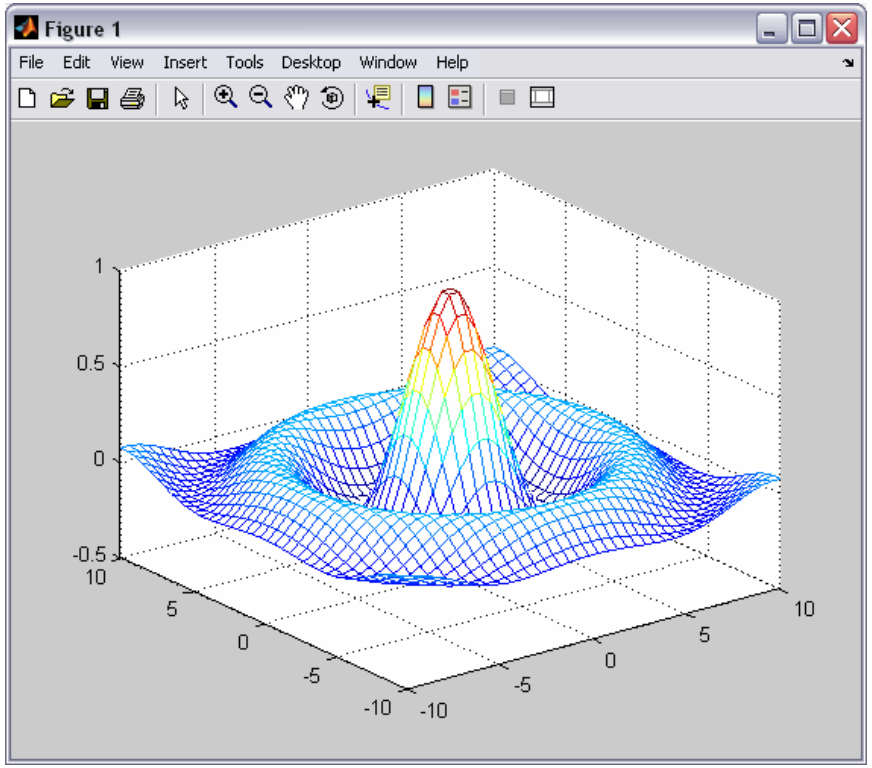
- **Achsenbeschriftung:**
  xlabel('...')* und *ylabel('...')

- **Linienstil und Farbe:**
  plot(x,y,'format');



Visualisierung 3D

```
>> x = linspace(-10,10,40);
>> y = linspace(-10,10,40);
>> [X,Y] = meshgrid(x,y);
>> R = sqrt(X.^2+Y.^2)+0.1;
>> Z = sin(R)./R;
>> mesh(X,Y,Z)
```



Beispiel 1

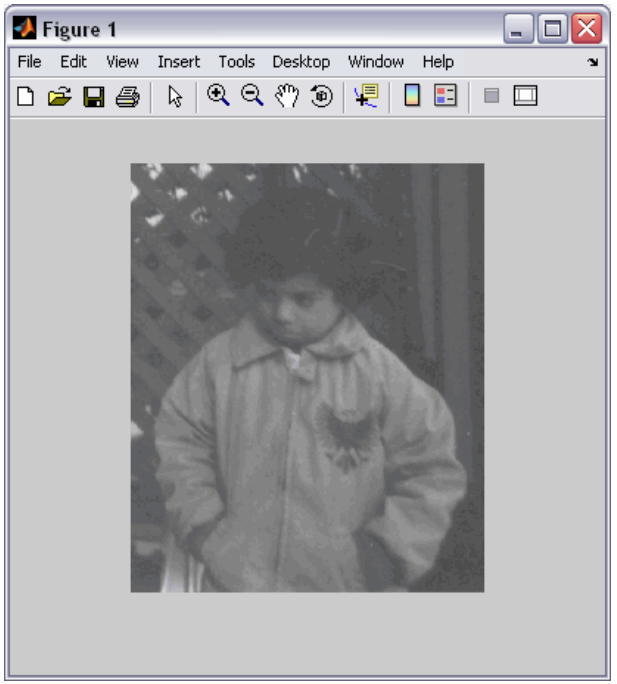
- **Erzeugen Sie einen Vektor t mit N Komponenten, die von 0 bis 4*pi linear ansteigen (N = 4, 8, 128).**
  Für N = 4 soll das Ergebnis
  [0.00000 4.1888 8.3776 12.5664]
  lauten.

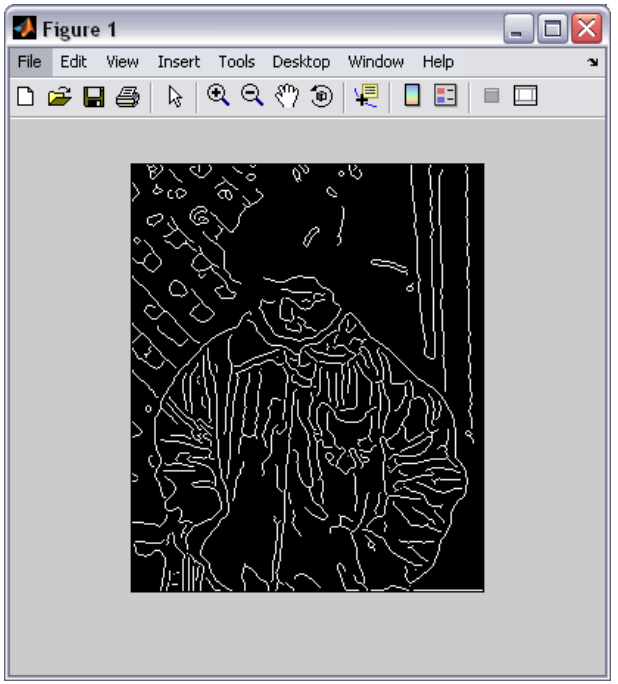
- Lösung 1
  ```
  >> N = [0:4/3*pi :4*pi]
  ```
- Lösung 2
  ```
  >>  linspace(0,4*pi, 4)
  ```

Beispiel 2

- **Suchen Sie ein beliebiges Grauwertbild *I* und erzeugen Sie ein Kantenbild mit dem Befehl *edge(I,'canny');*.**

```
>> I = imread('pout.tif');
>> BW = edge(I,'canny');
>> imshow(BW);
```





Fragen?

```
>> helpdesk
```