

Sommercampus 2004

MATLAB-Kurs

2. Teil: Fortgeschrittene Techniken



Gerd Brunner, **Bernard Haasdonk**, Klaus Peschke

Albert-Ludwigs-Universität Freiburg

Institut für Informatik

Lehrstuhl für Mustererkennung und Bildverarbeitung

Prof. Dr.-Ing. H. Burkhardt

Übersicht

- Debugging
- Profiling
- Geschwindigkeits-Optimierung
- Interaktion mit Betriebssystem
- Interaktion mit Programmen
- Grafik-Objekte
- GUI-Programmierung
- Aufgaben

Debugging

- Fehlertypen:
 - Syntax-Fehler: leicht durch MATLAB Interpreter detektierbar dank Fehler-Kommentar, Datei- und Zeilenangabe.
 - Laufzeit-Fehler: schwieriger zu detektieren, lokale Variablen bei Funktionsabbruch verloren.
- Programmausgaben nutzen:
 - Entfernen von Semikolon: **A = B*B;** (keine Ausgabe)
A = B*B (Ausgabe von A)
 - Einzelne Kommentare in MATLAB-Konsole ausgeben
disp(M); disp('reached A in mycode.m');
disp(['value s = ', num2str(s)]);
 - Alle Ein-Ausgaben der MATLAB-Konsole in eine Datei
diary alloutput.txt, diary on, diary off

Debugging

- Fehlertypen:
 - Syntax-Fehler: leicht durch MATLAB Interpreter detektierbar dank Fehler-Kommentar, Datei- und Zeilenangabe.
 - Laufzeit-Fehler: schwieriger zu detektieren, lokale Variablen bei Funktionsabbruch verloren.
- Programmausgaben nutzen:
 - Entfernen von Semikolon: **A = B*B;** (keine Ausgabe)
A = B*B (Ausgabe von A)
 - Einzelne Kommentare in MATLAB-Konsole ausgeben
disp(M); disp('reached A in mycode.m');
disp(['value s = ', num2str(s)]);
 - Alle Ein-Ausgaben der MATLAB-Konsole in eine Datei
diary alloutput.txt, diary on, diary off

Debugging

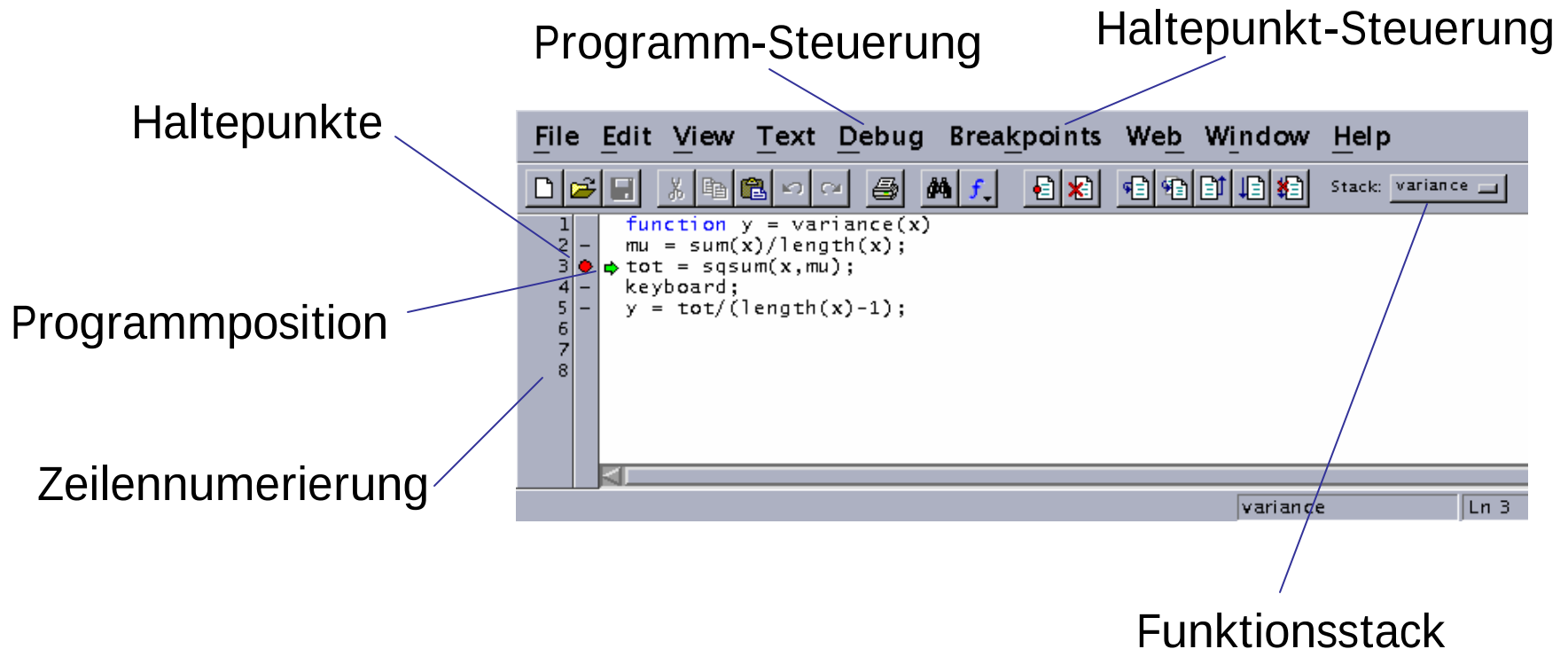
- Fehlersuche per Kommandozeile:
 - Programmstop/Debug-Modus starten durch Code-Zeile **keyboard** oder Setzen von temporären Haltepunkten(**dbstop**)
 - Debug-Mode ist erkennbar durch das Prompt **K>>**
 - Lokale Variablen können in der Kommandozeile aufgelistet (**whos**) und eingesehen/modifiziert werden
 - Fortsetzen des Programms durch Kommando **return**
- | | |
|---------------------------------------|------------------------|
| >>dbstop myfunction 3 | (Haltepunkt gesetzt) |
| >>myfunction([1 2 3 4]); | (Funktionsaufruf) |
| 3 tot = sqsum(x,mu); | (Auszuführende Zeile) |
| K>>x | (Aufruf Ausgabe von x) |
| x = | (Ausgabe von x) |
| 1 2 3 4 | |

Debugging

- Übersicht über Debug-Kommandos:
 - Haltepunkt setzen: **dbstop**
 - Haltepunkt löschen: **dbclear**
 - Allgemeiner Halt: **dbstop if warning**
 - Funktionsaufruf-Stack **dbstack**
 - Liste der Haltepunkte **dbstatus**
 - Programm fortsetzen **dbcont**
 - Schritt(e) ausführen **dbstep**
 - Funktion mit Zeilennummern **dbtype**
 - Funktionshierarchie aufwärts **dbup**
 - Funktionshierarchie abwärts **dbdown**
 - Kompletter Programmabbruch **dbquit**

Debugging

- Grafische Debugger-Oberfläche:
 - automatisches öffnen des Programm-Codes
 - Variablenübersicht im Hauptfenster
 - Debug-Mode im Hauptfenster



Profiling

- herkömmlich: „heuristisches“ Optimieren
 - willkürliches Code-Optimieren
 - „Design-basierte“ Entscheidung für zu optimierende Funktionen
 - oft suboptimal, da Code oft nur „vermeintlich relevant“: exakte Aufruf-Häufigkeit oder Laufzeit oft verschätzt.
- besser: Profiling
 - Bestimmen der exakten Häufigkeit von Funktionsaufrufen
 - Durchführen von exakten Laufzeitmessungen pro Codezeile
 - Ermitteln von „tatsächlich relevanten“ Code-Stellen

⇒ Hinweise auf Anpassung des Designs, Umprogrammierung

Profiling

- MATLAB-Profiler:
 - pro Zeile: Laufzeit, Häufigkeit
 - pro Funktion: Aufruf-Häufigkeit, parent/child-Funktionen
 - Protokollieren der Reihenfolge der Funktionsaufrufe

>>profile on -history (Beginn des Profiling)

>>myfunction1, ... (Funktionsaufrufe)

- Profile-Ergebnisse als Variable:

>>p = profile('info')

p =
 FunctionTable: [123x1 struct]
 FunctionHistory: [2*10000 double]
 Clockprecision: 3.2584e-0.8
 Name: 'MATLAB'
 ClockSpeed: 264

Profiling

- Profile-Ergebnisse als HTML: **profile report**

Summary | Function Details | Function Call History

MATLAB Profile Report: Summary

Report generated 20-Sep-2004 09:24:10

Total recorded time: 16.12 s
Number of M-functions: 79
Number of M-subfunctions: 41
Number of MEX-functions: 3
Clock precision: 0.00000003 s
Clock Speed: 264 Mhz

Function List

Name	Time	Calls	Time/call	Self time	Location
local_inv_demo	15.96000000	99.0%	16	0.997500000000	1.01000000 6.3% /home/haasdonk/matlab/local_invariants/local_inv_demo.m
xytransl_inv_c	11.55000000	71.7%	8	1.443750000000	0.80000000 5.0% /home/haasdonk/matlab/local_invariants/xytransl_inv_c.m
save_libsvm_data	8.51000000	52.8%	8	1.063750000000	8.51000000 52.8% /home/haasdonk/matlab/general/save_libsvm_data.m
load_libsvm_data	2.12000000	12.5%	8	0.265000000000	1.40000000 2.0% /home/haasdonk/matlab/general/load_libsvm_data.m

Summary | Function Details | Function Call History

MATLAB Profile Report: Function Details

[local_inv_demo](#) (/home/haasdonk/matlab/local_invariants/local_inv_demo.m)
Time: 15.96000000 s (99.0%)
Calls: 16
Self time: 1.01000000 s (99.0%)

Function:	Time	Calls	Time/call
local_inv_demo	15.96000000	16	0.997500000000

Parent functions:

local_inv_demo	11
--------------------------------	----

Child functions:

local_inv_demo	38.91000000	243.8%	11	3.537272727273
xytransl_inv_c	11.55000000	72.4%	8	1.443750000000
isshow	1.14000000	7.2%	42	0.027280952381

Summary | Function Details | Function Call History

MATLAB Profile Report: Function Call History

[local_inv_demo](#) (/home/haasdonk/matlab/local_invariants/local_inv_demo.m)

[hgload](#) (/misc/software-lin/matlab6p5/toolbox/matlab/iofun/hgload.m)

[fileparts](#) (/misc/software-lin/matlab6p5/toolbox/matlab/iofun/fileparts.m)

[ispc](#) (/misc/software-lin/matlab6p5/toolbox/matlab/general/ispc.m)

[filesep](#) (/misc/software-lin/matlab6p5/toolbox/matlab/iofun/filesep.m)

[fullfile](#) (/misc/software-lin/matlab6p5/toolbox/matlab/iofun/fullfile.m)

[filesep](#) (/misc/software-lin/matlab6p5/toolbox/matlab/iofun/filesep.m)

[cellstrmatch](#) (/misc/software-lin/matlab6p5/toolbox/matlab/strfun/cellstrmatch.m)

[nargchk](#) (/misc/software-lin/matlab6p5/toolbox/matlab/lang/nargchk.m)

[iscellstr](#) (/misc/software-lin/matlab6p5/toolbox/matlab/strfun/iscellstr.m)

- nützlich: 'self-time' und 'History'

Profiling

- Profile-Ergebnisse interaktiv:
profile viewer

File Edit Web Help				
Start Profiling Run this code: Go				
Profile Summary Generated 20-Sep-2004 09:28:02 Number of files called: 123				
Filename	File Type	Calls	Total Time	Time Plot
local_inv_demo	M-function	16	15.960 s	
xytransl_inv_c	M-function	8	11.550 s	
save_libsvm_data	M-function	8	8.510 s	
load_libsvm_data	M-function	8	2.130 s	
str2num	M-function	4668	1.460 s	
legend	M-function	48	1.380 s	
legend/make_legend	M-subfunction	32	1.230 s	
imshow	M-function	42	1.150 s	
newplot	M-function	92	0.860 s	
/misc/.../graphics/private/clo	M-function	95	0.830 s	
newplot/ObserveAxesNextPlot	M-subfunction	92	0.830 s	
str2num/protected_conversion	M-subfunction	4668	0.620 s	
legend/changedText	M-subfunction	16	0.470 s	
legend/resize_legend	M-subfunction	24	0.440 s	
hload	M-function	1	0.240 s	
textread	M-function	8	0.220 s	
allchild	M-function	138	0.190 s	

File Edit Web Help

Start Profiling

Run this code:

Find in page:

Go

▼

Profile time: 0 s

xytransl_inv_c (8 calls, 11.550 sec)

Generated 20-Sep-2004 09:31:05

M-function in file /home/haasdonk/matlab/local_invariants/xytransl_inv_c.m

(Copy to new window for comparing multiple runs)

Parents (calling functions) [table | list | hide]

local_inv_demo

Lines where the most time was spent [table | list | hide]

Line Number	Code	Calls	Total Time	% Time	Time Plot
51	save_libsvm_data(samples...	8	8.520 s	73.8%	
61	features = load_libsvm_da...	8	2.130 s	18.4%	
47	samples = [samples, resha...	1536	0.650 s	5.6%	
54	[s,w] = system(['./compute...	8	0.060 s	0.5%	
73	delete(features_file);	8	0.040 s	0.3%	
All other lines			0.150 s	1.3%	
Totals			11.550 s	100%	

Children (called functions) [table | list | hide]

Filename	File Type	Calls	Total Time	% Time	Time Plot
save_libsvm_data	M-function	8	8.510 s	73.7%	
load_libsvm_data	M-function	8	2.130 s	18.4%	
char/delete	M-function	24	0.060 s	0.5%	
num2str	M-function	32	0.050 s	0.4%	
All other time (built-ins, math, etc.)			0.800 s	6.9%	
Totals			11.550 s	100%	

```
File Edit Web Help
Find in page:  Go
Start Profiling Run this code:  Profile time: 0 sec

File listing
Color highlight code according to [Time | Number of Calls | Coverage | Acceleration | No Color]

time calls acc line
1 function [samples, labels] = load_libsvm_data(filename,silent)
2 %function [samples, labels] = load_libsvm_data(filename [,silent]
3 %
4 % loading of libsvm-data from file filename
5 % the file has to be in libsvm-train format.
6 % resulting samples are columnwise in samples.
7 % if silent is given, no output is performed.
8 % Bernard Haasdonk 10.11.2003
0.000 8 . 11 if (nargin<2)
12 silent = 0;
13 end;
14
0.000 8 . 15 maxlen = 1900000; % maximum length of whitespace separated p
16 % in file
17
0.000 8 . 18 if (nargin<1) | (isempty(filename))
19 error('no file specified!');
20 end;
21
0.000 8 . 22 if (~silent)
23 disp(['reading samplesfile in libsvm-format.']);
24 end;
0.220 8 x 25 inp = textread(filename,'%s',maxlen);
26
8 x 27 if (length(inp) == maxlen)
28 disp(['file larger than ', num2str(maxlen), ' strings, last
29 % vector will be only partially read.']);
30 end;
31
0.000 8 . 32 i = 0;
0.000 8 . 33 ns = 0;
0.001 8 . 34 samples = zeros(0,0);
0.000 8 . 35 labels = [];
36
37 % check if source file is a model-file
38 %if (isequal(inp{1},'SVM-light'));
39 % i = i+1;
40 % while fscanfz(inp{1},'%f\n'))
41 % i = i+1;
42 % end;
43 % if i>0
44 % %
45 % end;
46 % %
47 % %
48 % %
49 % %
50 % %
51 % %
52 % %
53 % %
54 % %
55 % %
56 % %
57 % %
58 % %
59 % %
60 % %
61 % %
62 % %
63 % %
64 % %
65 % %
66 % %
67 % %
68 % %
69 % %
70 % %
71 % %
72 % %
73 % %
74 % %
75 % %
76 % %
77 % %
78 % %
79 % %
80 % %
81 % %
82 % %
83 % %
84 % %
85 % %
86 % %
87 % %
88 % %
89 % %
90 % %
91 % %
92 % %
93 % %
94 % %
95 % %
96 % %
97 % %
98 % %
99 % %
100 % %
```

- nützlich: komplette Code-listings, Selektion von Zeilen
- auch sinnvolles Tool für Debugging, Code-Einarbeitung

Geschwindigkeits-Optimierung

■ Preallocation

- MATLAB erlaubt dynamisches Erweitern von Feldern. Speicherverwaltung erfordert jedoch Laufzeit.
- Falls Endgrösse eines Feldes bekannt ist, macht frühe Speicherallokation Sinn.
- Vektoren/Matrizen:

X = []; X(10,20)=0; oder X = zeros(10,20); (0-Matrix)

Y = 5 * ones(20,30); (5-Matrix)

- Cell-arrays:

C = {}; C(10)={ [] }; oder C = cell(1,10);

- Strukturen:

S = []; S.feld1 = []; S.feld2 = []; S(10).feld2 = []
oder S = struct('feld1', C, 'feld2', C)

Geschwindigkeits-Optimierung

■ Vektorisierung

- Problem von Interpreter-Sprachen: Schleifen
- Stärke von MATLAB: Operationen auf Matrizen

⇒ Umformulieren von Algorithmen, indem Schleifen durch Matrix-Operationen implementiert werden.

- elementweise Matrizen-Operationen: **+** **-** **min** **exp**
und „Punktieren“ von Matrix-Operatoren: **.*** **.^**

```
for x = 1:nx, for y = 1:ny,  
    S(x,y) = (A(y,x)+A(x,y))^2  
end;end;
```

⇓

```
S = (A+A').^2
```

Korrelationsmatrix

$$\mathbf{R} := \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \cdot \mathbf{x}_i^T$$

⇓

```
R = (X * X')/size(X,2);
```

- weitere Funktionen: **find**, **sum**, **cumsum**, **repmat**

Geschwindigkeits-Optimierung

- Optimierung bei Verwendung von komplexen Zahlen

```
z1 = a1 + i* b1;  z2 = a2 + i * b2;
```

```
z3 = z1 * z2;
```

```
a3 = real(z3)
```

```
b3 = imag(z3)
```

- **i** kann nicht optimiert werden

⇒ Definition von **i=sqrt(-1);**

- komplexe Operationen sind etwas langsamer als reelle

⇒ **a3 = a1 * a2 - b1 * b2**

b3 = a1 * b2 + a2 * b1

Interaktion mit Betriebssystem

- Shell-Kommandos ausführen:

- in MATLAB-Kommandozeile: „!“

```
>>!emacs new_function.m &
```

- im Code (keine interaktiven Befehle)

```
[stat, res] = unix('latex myfile.tex');
```

- Daten-Import/Export:

- MAT-Files (C/C++ , fortran Bibliotheken verfügbar)

```
save('allvar');      save('singlevar','dummy','L');  
load('allvar');      temp = load('singlevar');
```

- Binärdaten (**fopen**, **fread**, **fwrite**, **fclose**)
- Formatierter text (**fscanf**, **fprintf**, **save -ASCII**)

```
[name,age,hght] = textread('data.txt','%s%d%f');
```

- Bilddateitypen (**imread**, **imwrite**), ...

Interaktion mit Programmen

- MATLAB Array

- dies ist einziges Datenformat für alle möglichen Objekttypen
- Schnittstellen für dieses Datenformat ermöglicht Kombination mit anderen Programmiersprachen
- Schnittstelle für C/C++: Datentyp **mxArray**
Zugriff auf Felder via mx-Routinen

int mxGetM(mxArray *) (erste Dimension)

double * mxGetPr(mxArray *) (realteil der Daten)

- C/C++ Code: MATLAB ruft C (mex-Interfaces)

- C-File (mit bestimmter Struktur) **mymexfunc.c**
- Kompilieren mittels **mex mymexfunc.c**
dies ergibt **mymexfunc.mexglx**
- MATLAB Hilfetext in **mymexfunc.m**
- Ausführen in MATLAB via **mymexfunc**

Interaktion mit Programmen

■ Aufbau einer mex-Datei

```
#include "mex.h"
```

```
void myfunc(double *y, *x)
{
    y[0] = x[1]; y[1] = x[0];
}
```

Berechnung-Abschnitt

```
void mexFunction(
    int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[])
{
    double *y,*x;
    plhs[0]=
        mxCreateDoubleMatrix(2,1,mxREAL);
    y = mxGetPr(plhs[0]);
    x = mxGetPr(prhs[0]);
    y = myfunc(y,x);
}
```

Interface-Abschnitt

Interaktion mit Programmen

- C/C++ Code: C ruft MATLAB („engine“ programs)
 - MATLAB-Engine läuft als Hintergrund Prozess, möglicherweise auf anderem Rechner.
 - Eigene C-Programme kommunizieren mit MATLAB-Engine, stellen Berechnungsanfragen.
 - Statt grosser MATLAB-Bibliothek nur kleine „Kommunikationsbibliothek“ erforderlich.

```
#include "engine.h"                (Engine-Bibliothek einbinden)
void main() {
    Engine *ep; mxArray *a,*d;
    ep = engOpen("\0");             (Engine öffnen)
    engPutArray(ep,a);              (Variable setzen)
    engEvalString(ep,"d=A*A");      (Berechnung durch Engine)
    d = engGetArray(ep,"d");         (Variable holen)
    engClose(ep);                   (Engine schließen)
}
```

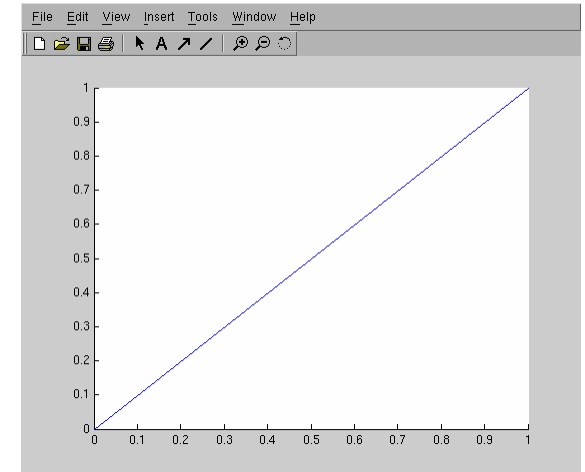
Interaktion mit Programmen

- Weitere Möglichkeiten:
 - Platform-unabhängig:
Kompilieren von MATLAB-Funktionen in standalone Anwendungen oder C/C++ Code via **mcc**
 - Fortran mex-Dateien
 - Windows: ActiveX
MATLAB als ActiveX objekt, steuerbar von jedem „Automation Controller“, z.B. MS-Office, Visual Basic/C++ Programme
 - Windows: Dynamic Data Exchange (DDE)
ermöglichen von DDE-conversations zwischen Windows Anwendungen. Unterschiedliche Unterstützung für MATLAB als client oder server.

Grafik-Objekte

- Handle Graphics
 - alle Elemente in einem Grafikfenster sind durch Handles (eindeutige Zahl) repräsentiert.
 - ein Handle **h** hat eine Liste von Property/Value-Paaren, einsehbar via **get(h)**, z.B. Type, Position
 - Setzen der Werte (falls) möglich via **set**

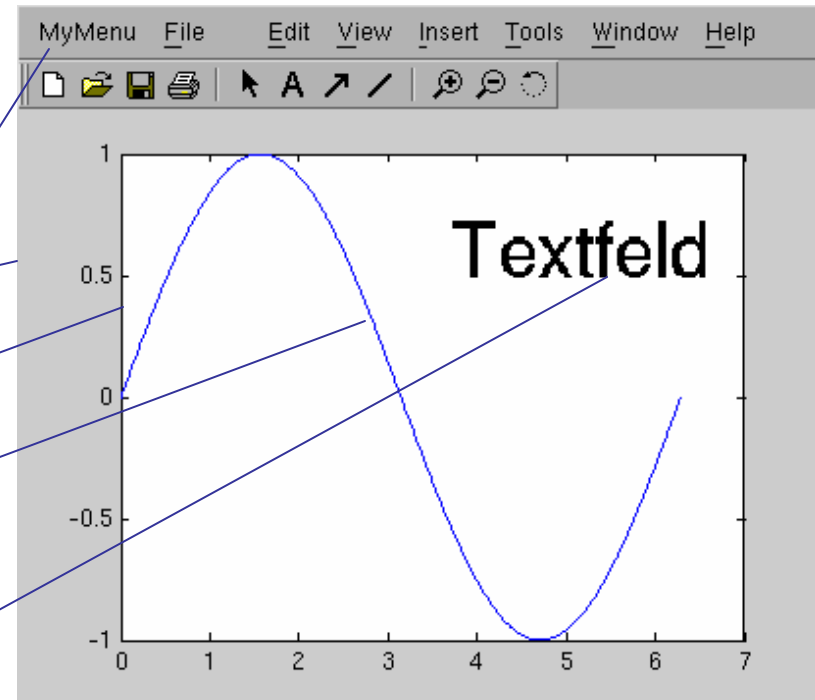
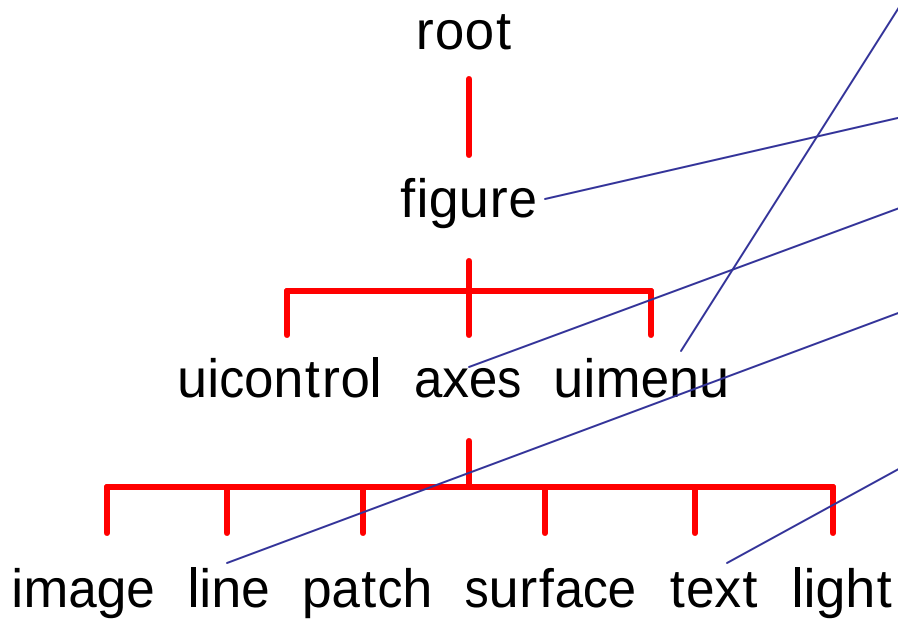
```
>>l = line([0 1],[0 1])  
l = 106.0002  
>>set(l, 'Color', [1 0 0]);
```



```
>>get(l)  
Color = [0 0 1]  
LineStyle = -  
LineWidth = [0.5]  
Xdata = [0 1]  
Ydata = [0 1]  
Parent = [101]
```

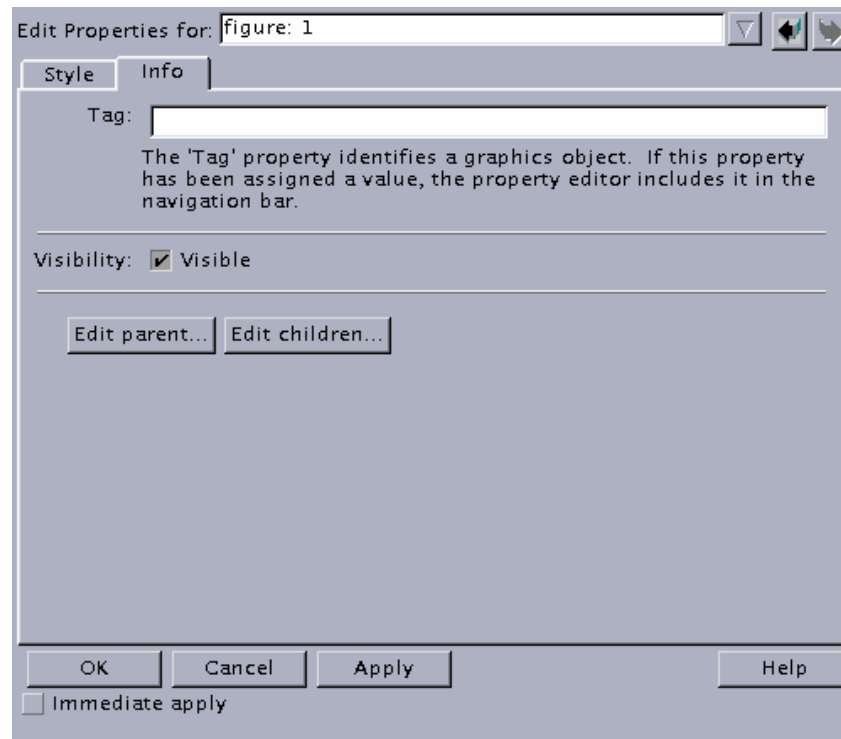
Grafik-Objekte

- Grafik Hierarchie
 - zugänglich durch Properties
 - 'Children', 'Parent'
 - Baumstruktur



Grafik-Objekte

- Property Editor
 - grafische Oberfläche zum Einsehen/Manipulieren der Objekt-Hierarchie einer Figure/Axes
 - Beispiel: aktuelle Figure editieren via **propedit(gcf)**



GUI-Programmierung

- uicontrols

- interaktive Objekte in einer Figure

pushbutton

togglebutton

radiobutton

checkbox

listbox

popupmenu

edit

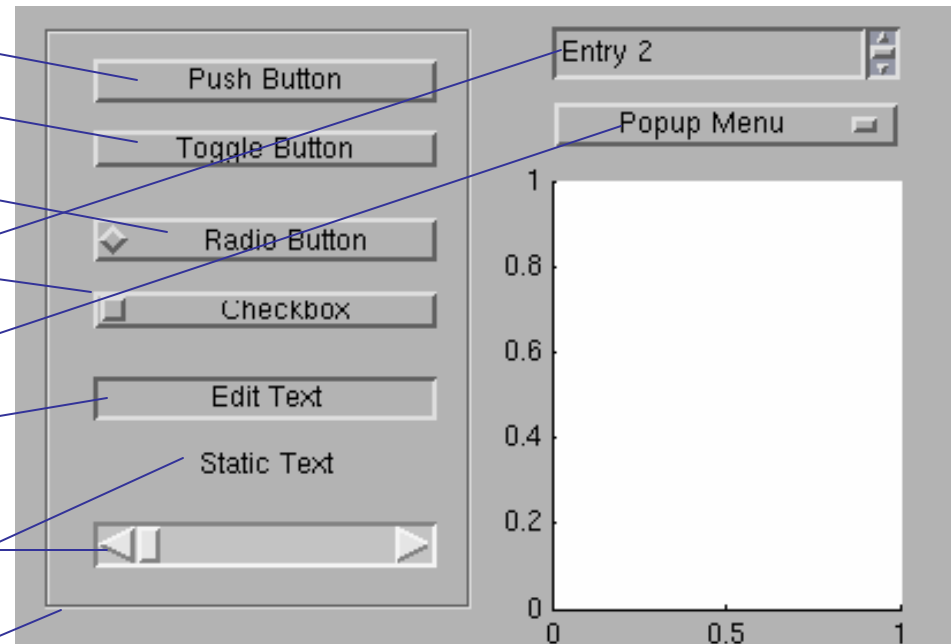
slider

- statische Objekte

text

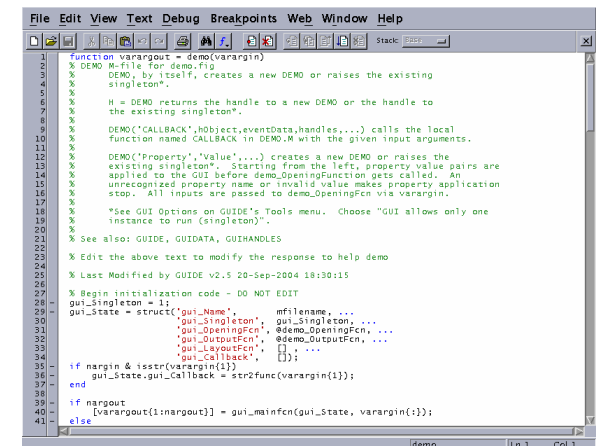
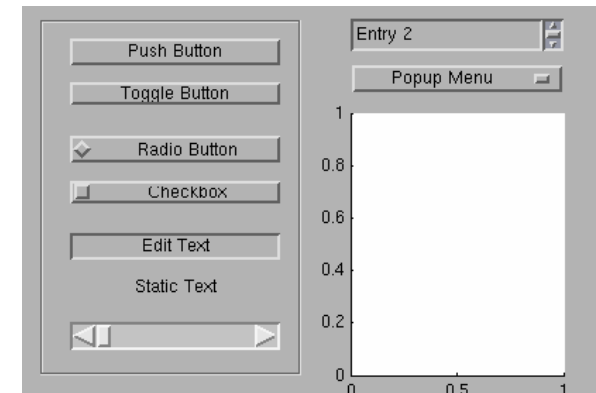
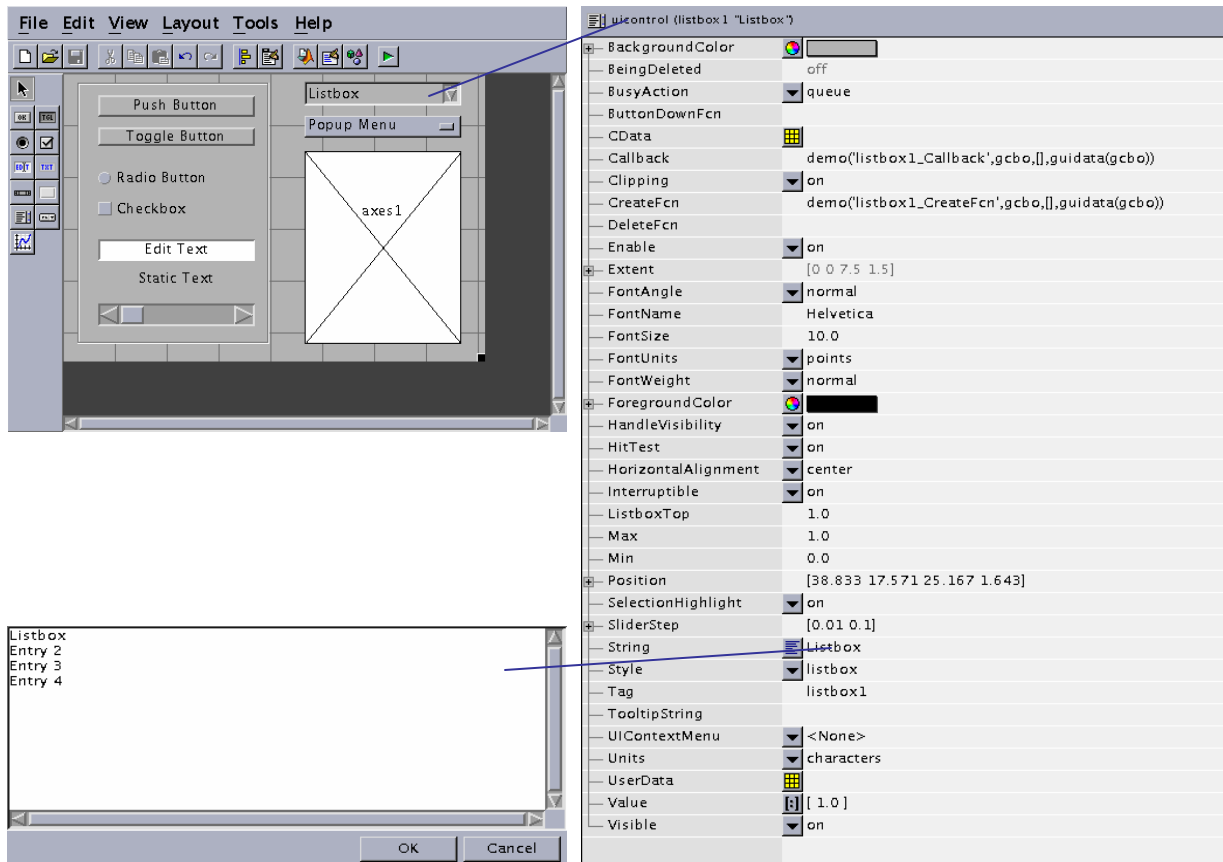
frame

- Erzeugung z.B via **`h=uicontrol(gcf,'Style','text')`**



GUI-Programmierung

- Layout der GUI mittels interaktivem Editor **guide** ⇒ **myGUI.fig** + **.m**



GUI-Programmierung

- Implementation der Callback-Funktionen in **myGUI.m**
 - Callback-Funktion eines Objekts wird bei User-Interaktion aufgerufen
 - Referenz auf aktuelle Figure via **gcbf**

function pushbutton1_Callback(...)

close(gcbf); (Schliesse Figure der GUI nach Knopfdruck)

- Referenz auf aktuelles Objekt via **gcbo**

function slider1_Callback(...)

phi = 2*pi*get(gcbo, 'Value'); (Winkel aus Slider-Wert)

- Suche das Handle eines anderen Objektes via **findobj**

st = findobj(gcbf, 'Tag', 'text1'); (statisches Textfeld)

set(st, 'String', num2str(phi)); (setze Winkel als Text)