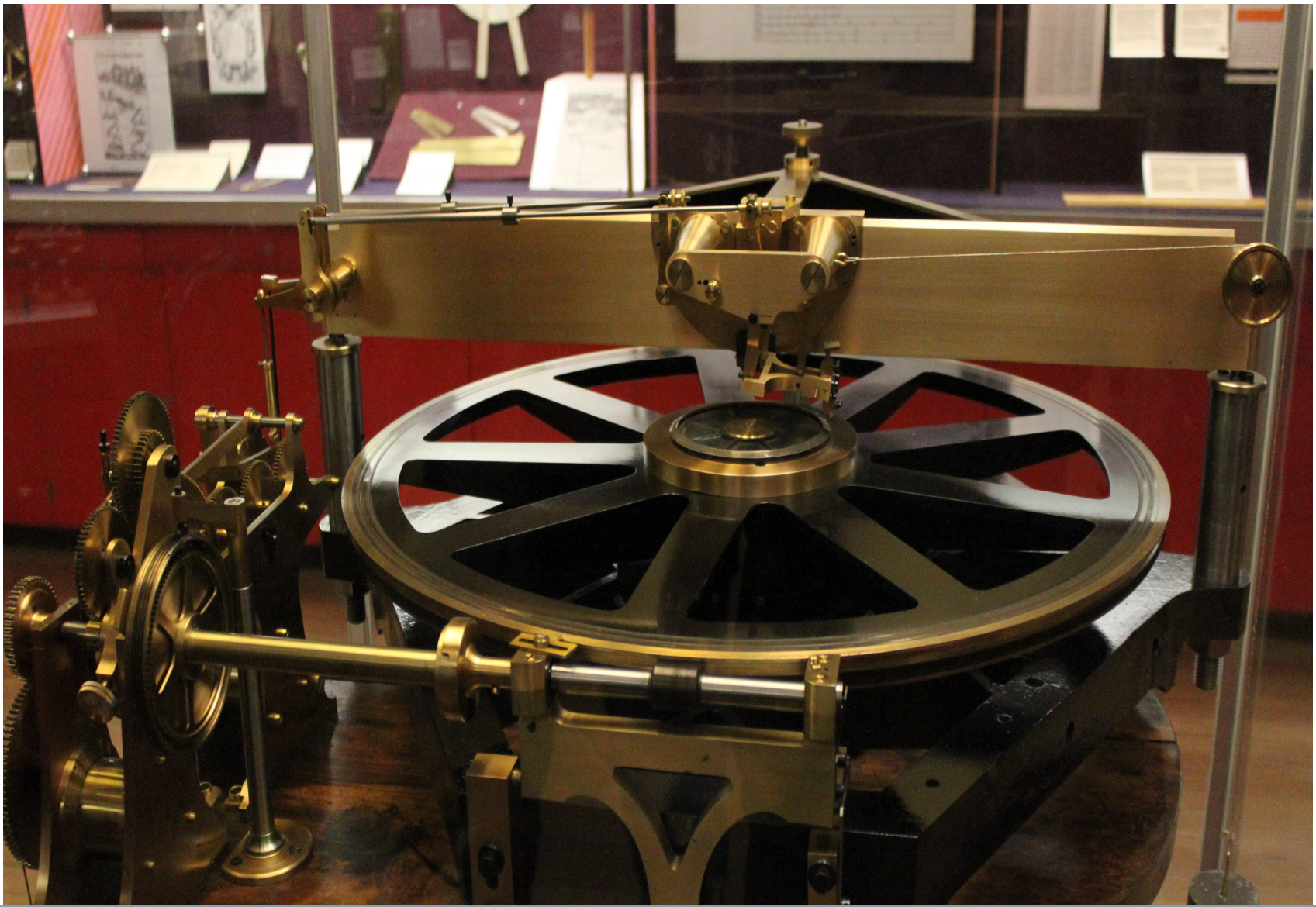
Eine Einführung zum numerischen Programmieren mit Matlab

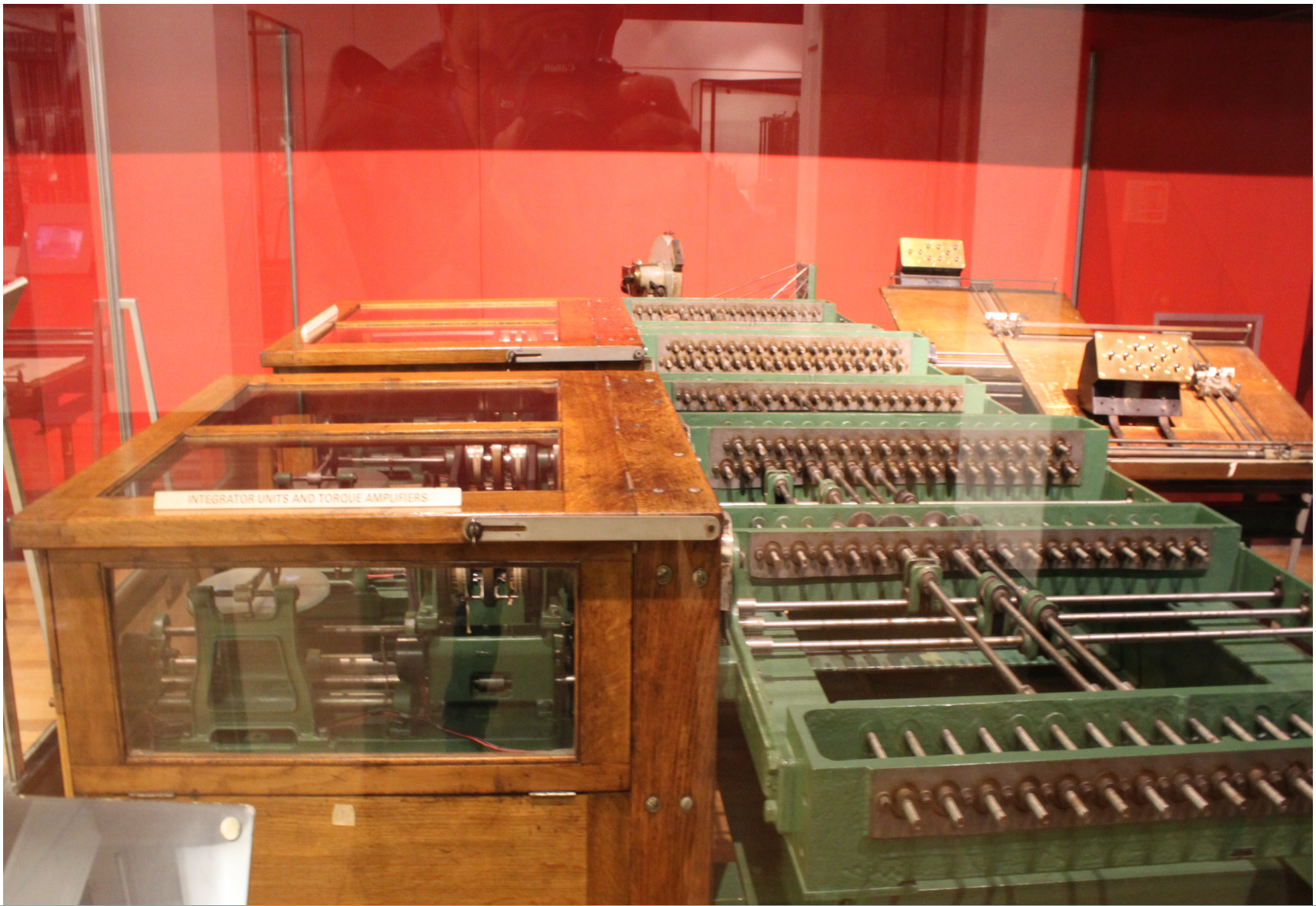
Bastian Groß

Universität Trier

Divisionsmaschine



Differenzieren und Integrieren



Grundrechenarten

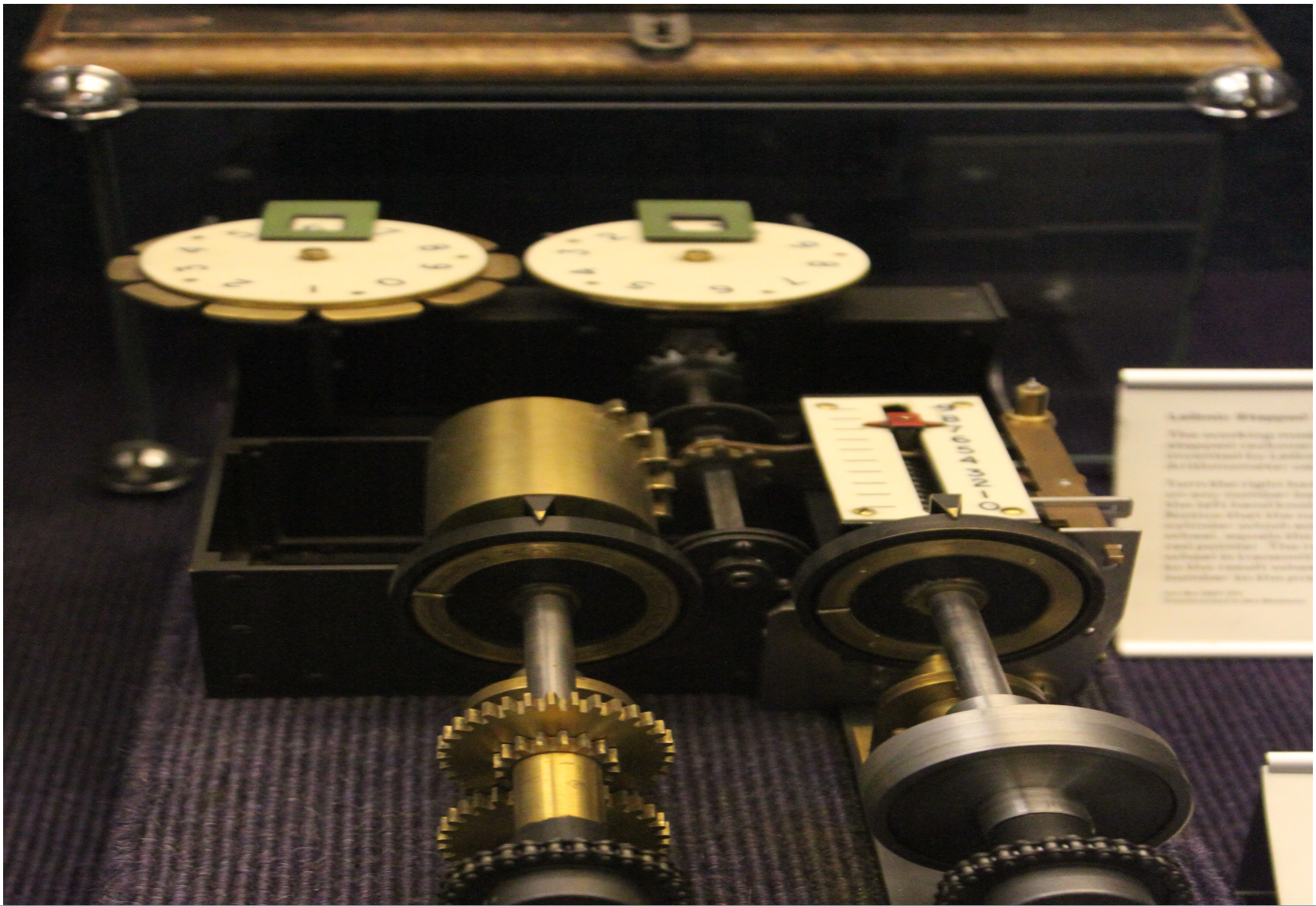
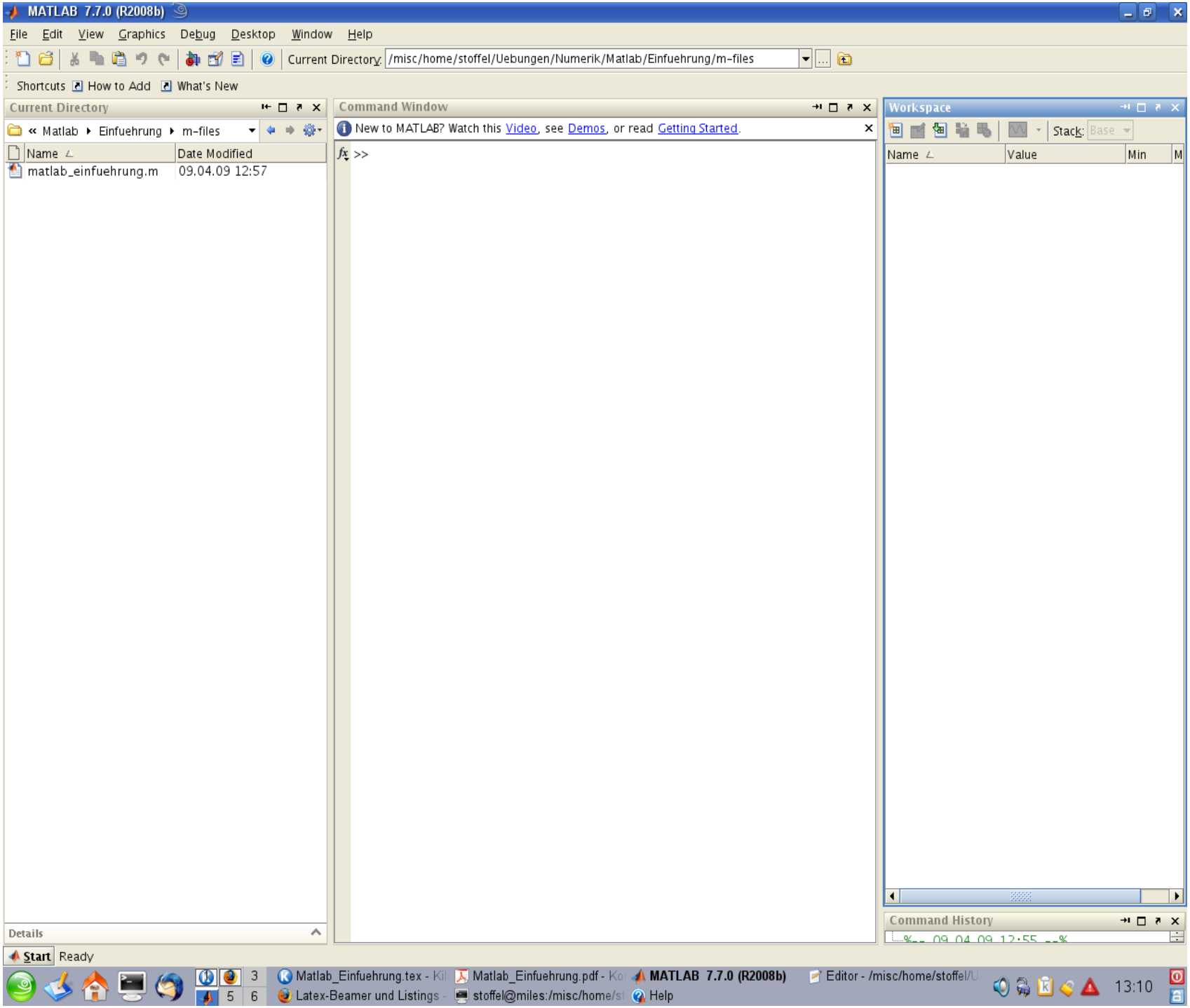


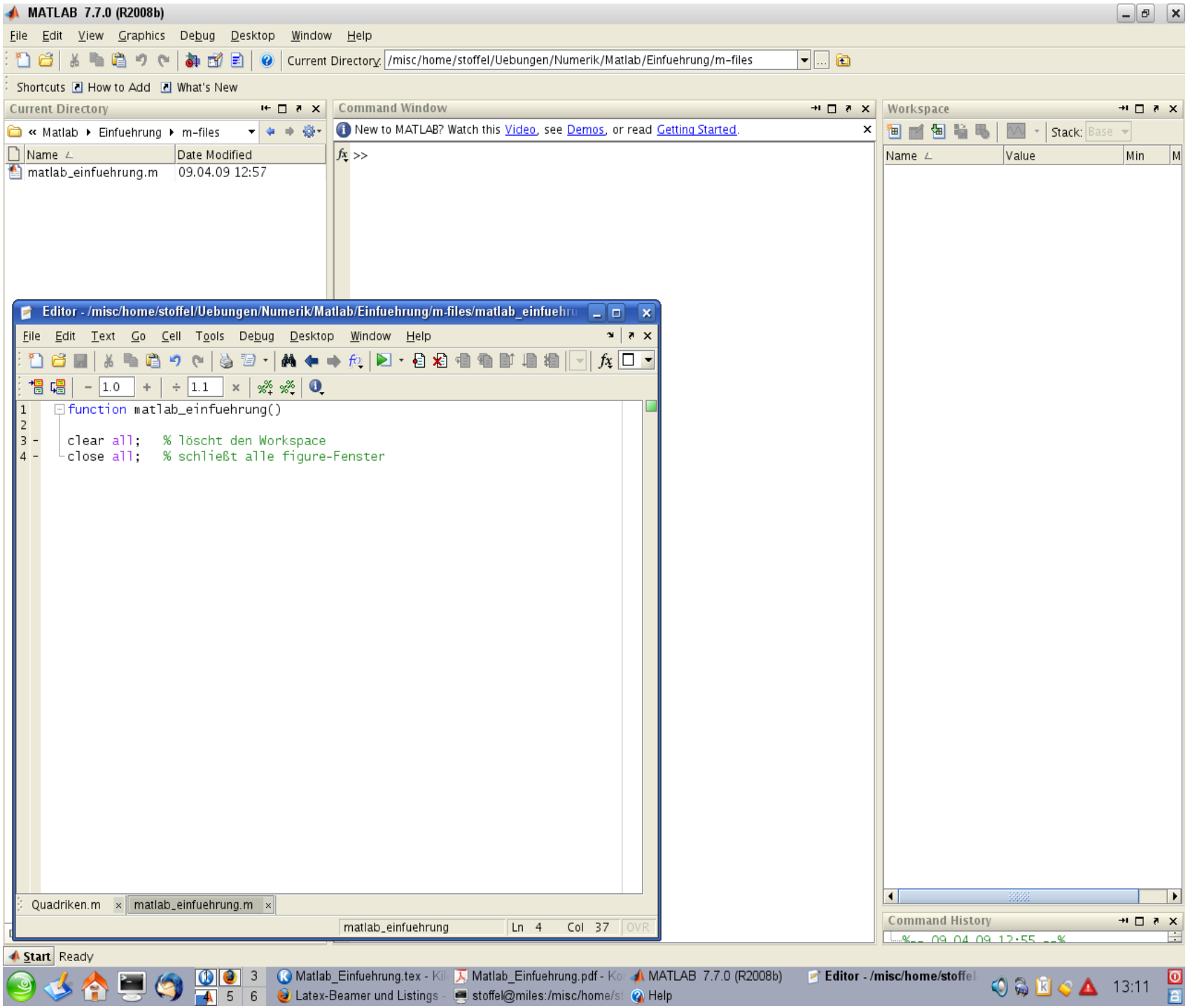
Table of contents

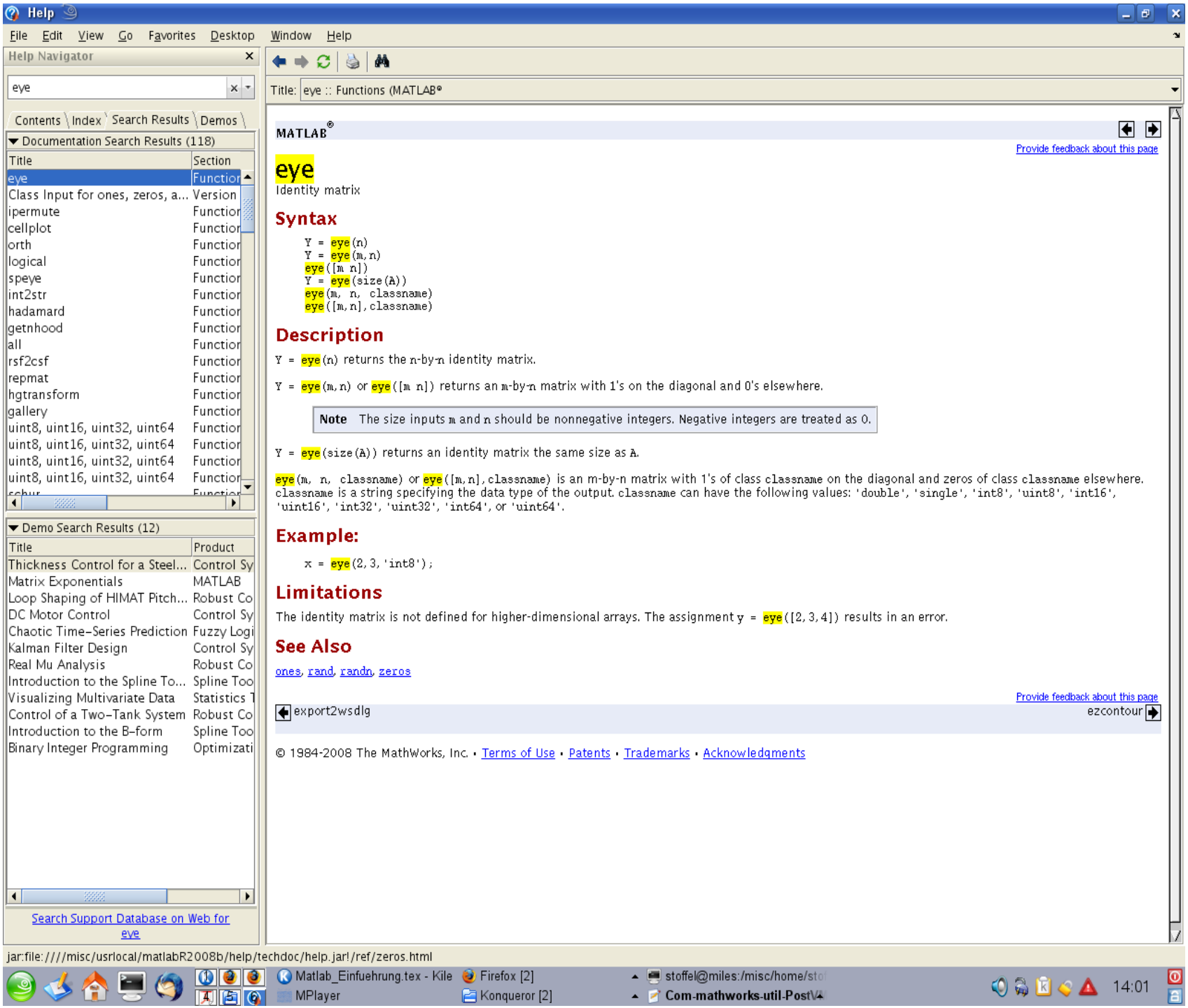
Table of contents

Erste Schritte mit Matlab

- Matlab starten:
  - Linux: Konsole öffnen und *matlab* eingeben
  - Windows: über Programmme auswählen und starten
- in das gewünschte Verzeichnis wechseln
- im Editor arbeiten
- Programm als `function` schreiben
- Programm unter dem `function`-Name als m-file speichern:
  dateiname.m (meist automatisch)
  Name an Programmzweck orientieren (z.B. Eigenwertberechnung)
  Vorsicht: Keine Doppelbezeichnungen (z.B. *plot.m* als Programmname)
- alles Unnötige schliessen bzw. löschen:
  `clear all, close all`
- Mit % kann im Programm kommentiert werden


MATLAB 7.7.0 (R2008b)
File Edit View Graphics Debug Desktop Window Help
Current Directory: /misc/home/stoffel/Uebungen/Numerik/Matlab/Einfuehrung/m-files
Shortcuts How to Add What's New
Current Directory
« Matlab ▸ Einfuehrung ▸ m-files
Name Date Modified
matlab_einfuehrung.m 09.04.09 12:57
Details
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
fx >>
Workspace
Stack: Base
Name Value Min
Command History
Start Ready
Matlab_Einfuehrung.tex - Kil
Matlab_Einfuehrung.pdf - Ko
MATLAB 7.7.0 (R2008b)
Editor - /misc/home/stoffel/U
Latex-Beamer und Listings -
stoffel@miles:/misc/home/st
Help
13:10


MATLAB 7.7.0 (R2008b)
File Edit View Graphics Debug Desktop Window Help
Current Directory: /misc/home/stoffel/Uebungen/Numerik/Matlab/Einfuehrung/m-files
Shortcuts How to Add What's New
Current Directory
« Matlab ▸ Einfuehrung ▸ m-files
Name Date Modified
matlab_einfuehrung.m 09.04.09 12:57
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
>>
Workspace
Stack: Base
Name Value Min
Editor - /misc/home/stoffel/Uebungen/Numerik/Matlab/Einfuehrung/m-files/matlab_einfuehrung.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
1 function matlab_einfuehrung()
2
3 - clear all; % löscht den Workspace
4 - close all; % schließt alle figure-Fenster
Quadriken.m
matlab_einfuehrung.m
matlab_einfuehrung Ln 4 Col 37 OVR
Command History
Start Ready
Matlab_Einfuehrung.tex - Kil
Matlab_Einfuehrung.pdf - Kor
MATLAB 7.7.0 (R2008b)
Editor - /misc/home/stoffel
Latex-Beamer und Listings -
stoffel@miles:/misc/home/st
Help
13:11


Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
eye
Contents Index Search Results Demos
Documentation Search Results (118)
Title Section
eye Function
Class Input for ones, zeros, a... Version
ipermute Function
cellplot Function
orth Function
logical Function
speye Function
int2str Function
hadamard Function
getnhood Function
all Function
rsf2csf Function
repmat Function
hgtransform Function
gallery Function
uint8, uint16, uint32, uint64 Function
uint8, uint16, uint32, uint64 Function
uint8, uint16, uint32, uint64 Function
uint8, uint16, uint32, uint64 Function
Demo Search Results (12)
Title Product
Thickness Control for a Steel... Control Sy
Matrix Exponentials MATLAB
Loop Shaping of HIMAT Pitch... Robust Co
DC Motor Control Control Sy
Chaotic Time-Series Prediction Fuzzy Logi
Kalman Filter Design Control Sy
Real Mu Analysis Robust Co
Introduction to the Spline To... Spline Too
Visualizing Multivariate Data Statistics T
Control of a Two-Tank System Robust Co
Introduction to the B-form Spline Too
Binary Integer Programming Optimizati
Search Support Database on Web for eye
Title: eye :: Functions (MATLAB®
MATLAB®
Provide feedback about this page
eye
Identity matrix
Syntax
Y = eye(n)
Y = eye(m,n)
eye([m n])
Y = eye(size(A))
eye(m, n, classname)
eye([m,n],classname)
Description
Y = eye(n) returns the n-by-n identity matrix.
Y = eye(m,n) or eye([m n]) returns an m-by-n matrix with 1's on the diagonal and 0's elsewhere.
Note The size inputs m and n should be nonnegative integers. Negative integers are treated as 0.
Y = eye(size(A)) returns an identity matrix the same size as A.
eye(m, n, classname) or eye([m,n],classname) is an m-by-n matrix with 1's of class classname on the diagonal and zeros of class classname elsewhere. classname is a string specifying the data type of the output. classname can have the following values: 'double', 'single', 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int64', or 'uint64'.
Example:
x = eye(2,3,'int8');
Limitations
The identity matrix is not defined for higher-dimensional arrays. The assignment y = eye([2,3,4]) results in an error.
See Also
ones, rand, randn, zeros
Provide feedback about this page
export2wsdlg
ezcontour
© 1984-2008 The MathWorks, Inc. • Terms of Use • Patents • Trademarks • Acknowledgments
jar:file:////misc/usrlocal/matlabR2008b/help/techdoc/help.jar!/ref/zeros.html
Matlab_Einfuehrung.tex - Kile
Firefox [2]
MPlayer
Konqueror [2]
stoffel@miles:/misc/home/sto
Com-mathworks-util-Post\A
14:01

Variablen, Vektoren, Matrizen

- Variablen können direkt ohne Speicherallocation Werte zugeordnet werden
- Variable: `x = 5`
- Vektoren und Matrizen:
  Leerzeichen oder Komma $\widehat{=}$ nächste Spalte, Semikolon $\widehat{=}$ nächste Zeile
  Zeilenvektor $v = (1, 2, 3)$: `v = [1 2 3]` oder `v = [1,2,3]`
  Spaltenvektor $v = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$: `v = [1;2;3]`
  Matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$: `A = [1 2 3; 4 5 6]`

Variablen, Vektoren, Matrizen

- Variablen können direkt ohne Speicherallocation Werte zugeordnet werden
- Variable: `x = 5`
- Vektoren und Matrizen:
  Leerzeichen oder Komma $\widehat{=}$ nächste Spalte, Semikolon $\widehat{=}$ nächste Zeile
  Zeilenvektor $v = (1, 2, 3)$: `v = [1 2 3]` oder `v = [1,2,3]`
  Spaltenvektor $v = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$: `v = [1;2;3]`
  Matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$: `A = [1 2 3; 4 5 6]`

Variablen, Vektoren, Matrizen

- Variablen können direkt ohne Speicherallocation Werte zugeordnet werden
- Variable: `x = 5`
- Vektoren und Matrizen:
  Leerzeichen oder Komma $\widehat{=}$ nächste Spalte, Semikolon $\widehat{=}$ nächste Zeile
  Zeilenvektor $v = (1, 2, 3)$: `v = [1 2 3]` oder `v = [1,2,3]`
  Spaltenvektor $v = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$: `v = [1;2;3]`
  Matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$: `A = [1 2 3; 4 5 6]`

Variablen, Vektoren, Matrizen

- besondere Vektoren/Matrizen
  - `ones(m,n)`: Vektor/Matrix der Dimension $m \times n$ mit nur Einsen
  - `zeros(m,n)`: Vektor/Matrix der Dimension $m \times n$ mit nur Nullen
  - `eye(m,n)`: Vektor/Matrix der Dimension $m \times n$ mit Einsen auf den Diagonalelementen, sonst Nullen
- Zugriff auf Elemente
  - `v(3)`: der dritte Eintrag des Vektors v, also v_3
  - `v(2:4)`: die Einträge 2 bis 4 des Vektors v, also v_2, v_3, v_4
  - `A(2,3)`: das 2,3-Element der Matrix $A = [a_{ij}]$, also a_{23}
  - `A(:,1)`: die erste Spalte von A
  - `A(3,:)`: die dritte Zeile von A
  - `A(2:3,2:4)`: Teilmatrix von A

Variablen, Vektoren, Matrizen

- besondere Vektoren/Matrizen
  - `ones(m,n)`: Vektor/Matrix der Dimension $m \times n$ mit nur Einsen
  - `zeros(m,n)`: Vektor/Matrix der Dimension $m \times n$ mit nur Nullen
  - `eye(m,n)`: Vektor/Matrix der Dimension $m \times n$ mit Einsen auf den Diagonalelementen, sonst Nullen
- Zugriff auf Elemente
  - `v(3)`: der dritte Eintrag des Vektors v, also v_3
  - `v(2:4)`: die Einträge 2 bis 4 des Vektors v, also v_2, v_3, v_4
  - `A(2,3)`: das 2,3-Element der Matrix $A = [a_{ij}]$, also a_{23}
  - `A(:,1)`: die erste Spalte von A
  - `A(3,:)`: die dritte Zeile von A
  - `A(2:3,2:4)`: Teilmatrix von A

Vektoren, Matrizen

- Vektor/Matrixoperationen (siehe auch: `help arith, help matfun`)
  - `+`: Matrix-Addition (auf Dimension achten)
  - `*`: Matrix-Multiplikation (auf Dimension achten)
  - `'`: Transponieren
  - `\` bzw. `/`: Left bzw. Right-Devision: `x = A\b` l"ost $Ax = b$ bzw. analog
  - `inv(A)`: Inverse von A (bei hohen Dimensionen nicht zu empfehlen)
  - `[m,n] = size(A)`: bestimmen der Dimension von A
  - `[V,D] = eig(A)`: bestimmen der Eigenvektoren und zugehörigen Eigenwerte von A als Orthogonal- und Diagonalmatrix
  - `det(A)`: bestimmen der Determinanten von A

Komponentenweise Operationen

- `.*`, `./` oder `.^`: Elementweise Matrix-Operationen
- Matlab Funktionen können einfach auf den ganzen Vektor/die ganze Matrix angewandt werden: z.B. `sin(A)`, `cos(A)`, `exp(A)`, `log(A)` bestimmen der Funktionswerte von Einträgen der Matrix A und geben wiederum diese als Matrix aus

Komponentenweise Operationen sind sehr ntzlich in Matlab.

Komponentenweise Operationen

- `.*`, `./` oder `.^`: Elementweise Matrix-Operationen
- Matlab Funktionen können einfach auf den ganzen Vektor/die ganze Matrix angewandt werden: z.B. `sin(A)`, `cos(A)`, `exp(A)`, `log(A)` bestimmen der Funktionswerte von Einträgen der Matrix A und geben wiederum diese als Matrix aus

Komponentenweise Operationen sind sehr ntzlich in Matlab.

Ausgabe

- Anweisungen, die nicht mit einem Semikolon abgeschlossen werden, werden im Kommandofenster ausgegeben.
- `disp('Text');` $\rightarrow$ gibt den Text im Kommandofenster aus.
- `fprintf('Text_1 %1.6f Text_2 %2.3e Text_3\n',a,b);`$\rightarrow$ gibt den angegebenen Text mit den Variablen `a` und `b` im Kommandofenster aus. Dabei sind `%1.6f` bzw. `%2.2e` die Platzhalter mit entsprechenden Format für `a` bzw. `b`. `\n` bewirkt einen Zeilenumbruch.
  Die Ausgabe lautet also (mit `a` $= 2.2$ und `b` $= 0.00123$):
  `Text_1 2.200000 Text_2 1.230e-03 Text_3`

Ausgabe

- Anweisungen, die nicht mit einem Semikolon abgeschlossen werden, werden im Kommandofenster ausgegeben.
- `disp('Text');` $\rightarrow$ gibt den Text im Kommandofenster aus.
- `fprintf('Text_1 %1.6f Text_2 %2.3e Text_3\n',a,b);`$\rightarrow$ gibt den angegebenen Text mit den Variablen `a` und `b` im Kommandofenster aus. Dabei sind `%1.6f` bzw. `%2.2e` die Platzhalter mit entsprechenden Format für `a` bzw. `b`. `\n` bewirkt einen Zeilenumbruch.
  Die Ausgabe lautet also (mit `a` $= 2.2$ und `b` $= 0.00123$):
  `Text_1 2.200000 Text_2 1.230e-03 Text_3`

Ausgabe

- in Datei schreiben:

```
fid = fopen('Dateiname.txt','w');
 ... Anweisungen ...
 fprintf(fid,'Text_1 %1.6f Text_2 %2.3e Text_3\n',a,b
 ... Anweisungen ...
fclose(fid);
```

Durch diese Anweisungen wird der Text in die Datei *Dateiname.txt* geschrieben. Dabei wird einmal am Anfang die Datei mit entsprechenden Rechten geöffnet. Dazwischen kann in die Datei geschrieben werden. Am Ende wird einmal die Datei geschlossen.

Grafiken

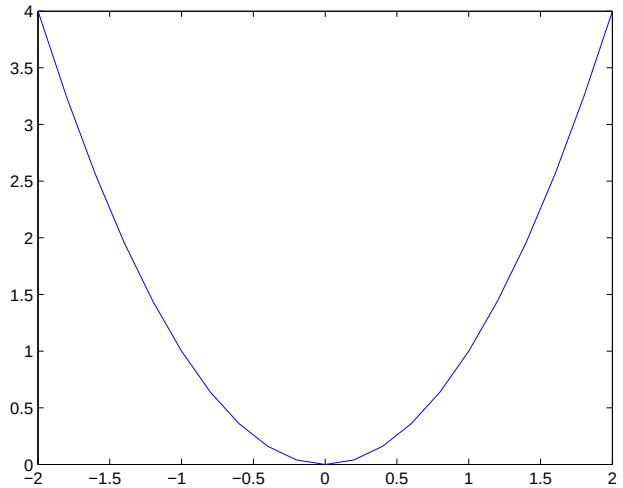
Mit den Befehlen `plot, plot3, surf, contour` etc. lassen sich Grafiken zeichnen. Weitere hilfreiche Befehle für Grafiken sind `meshgrid, surfc, isosurface`.
Das folgende Beispiel zeichnet die Funktion $y = x^2$ im Intervall $[-2, 2]$ mit Stützstellenweite 0.2, d.h es wird der Vektor x gegen den Vektor y geplottet, also die Punkte: $(x(1), y(1)); (x(2), y(2));$ usw.

```
x = [-2:0.2:2];
y = x.^2;
plot(x,y);
```

Grafiken

Mit den Befehlen `plot, plot3, surf, contour` etc. lassen sich Grafiken zeichnen. Weitere hilfreiche Befehle für Grafiken sind `meshgrid, surfc, isosurface`.
Das folgende Beispiel zeichnet die Funktion $y = x^2$ im Intervall $[-2, 2]$ mit Stützstellenweite 0.2, d.h es wird der Vektor x gegen den Vektor y geplottet, also die Punkte: $(x(1), y(1)); (x(2), y(2));$ usw.

```
x = [-2:0.2:2];
y = x.^2;
plot(x,y);
```



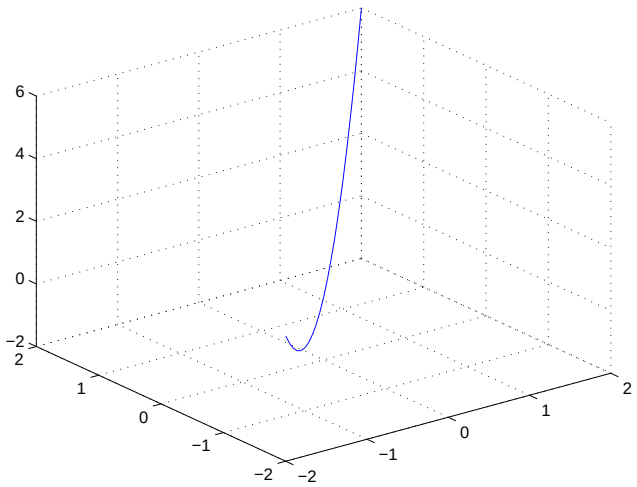
Grafiken

```
x = [-2:.2:2];
y = [-2:.2:2];
z = x.^2+y;
plot3(x,y,z)
grid on
```

```
x = [-2:.2:2];
y = [-2:.2:2];
[X,Y] = meshgrid(x,y);
Z = X.^2+Y;
surf(X,Y,Z);
```

Grafiken

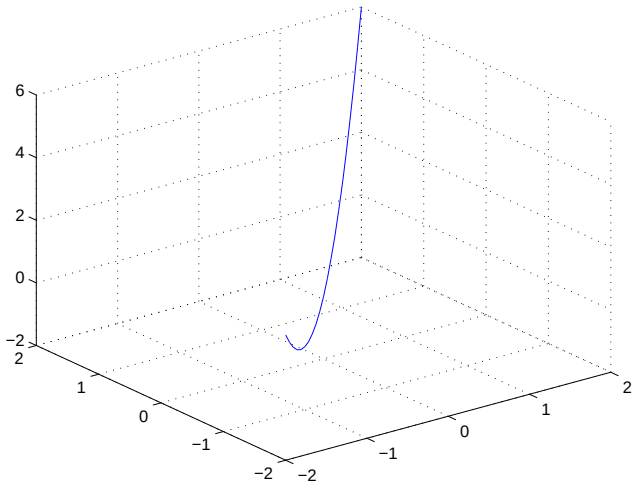
```
x = [-2:.2:2];
y = [-2:.2:2];
z = x.^2+y;
plot3(x,y,z)
grid on
```



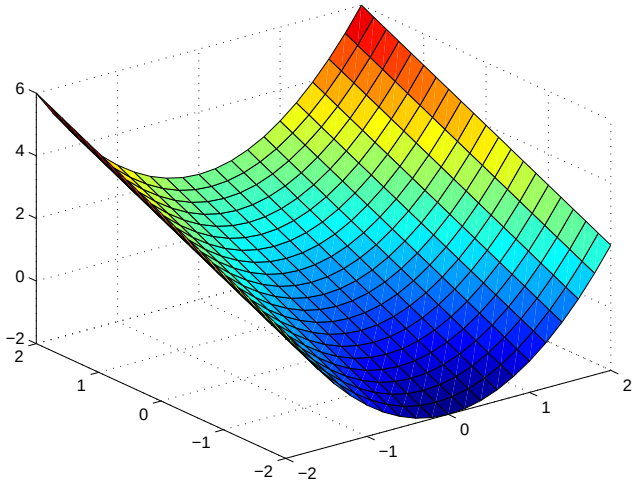

```
x = [-2:.2:2];
y = [-2:.2:2];
[X,Y] = meshgrid(x,y);
Z = X.^2+Y;
surf(X,Y,Z);
```

Grafiken

```
x = [-2:.2:2];
y = [-2:.2:2];
z = x.^2+y;
plot3(x,y,z)
grid on
```




```
x = [-2:.2:2];
y = [-2:.2:2];
[X,Y] = meshgrid(x,y
Z = X.^2+Y;
surf(X,Y,Z);
```



for-Schleife

Bei einer for-Schleife wird eine Gruppe von Anweisungen (Block) mit einer bestimmten Anzahl von Wiederholungen ausgeführt. Dabei wird die Zählvariable häufig in den Anweisungen integriert. (Vorsicht: eventuell lange Laufzeiten)

Beispiele:

```
for i = 1:100
  x(i) = 1;
end
```

In dieser Schleife wird dem i-ten Eintrag des Vektors x der Wert 1 zugeordnet. Die Schleife bewirkt das selbe wie `x = ones(1,100)`.

for-Schleife

```
for i = 1:100
  x(i) = i;
end
```

In dieser Schleife wird dem i-ten Eintrag des Vektors x der Wert i zugeordnet. Die Schleife erzeugt den Vektor $x = (1, 2, 3, \ldots, 99, 100)$.

```
x = 0;
for i = [2,3,5,10]
  x = x+i;
end
```

In diese Schleife wird zu der Variablen x, die mit 0 initialisiert ist, nacheinander die Werte 2,3,5,10 addiert. Das Endergebnis ist $x = 20$.

while-Schleife

Bei einer while-Schleife wird ein Block von Anweisungen so oft wiederholt bis die Abbruchbedingung erfüllt ist. Dabei ist das Kriterium eine logische (boolsche) Bedingung (wahr oder falsch).
Beispiel:

```
x = 0;
while x < 100
  x = x+1;
end
```

Bei dieser Schleife wird zu x solange 1 addiert, bis x größer gleich 100 ist, also 100 Wiederholungen.

Andere logische Bedingungen sind: `>`, `>=`, `<=`, `==`.
Zwei wichtige Überlegungen bei einer while-Schleife:

- Wird das Eintrittskriterium der while-Schleife erfüllt, d.h. wird überhaupt in die Schleife reingegangen?
- Wenn man in der while-Schleife ist, kommt man auch wieder raus, d.h. wird das Abbruchkriterium irgendwann erfüllt?

if-else

Bei einem if-else-Konstrukt werden logische Bedingungen überprüft und entsprechende Anweisungen ausgeführt.
Beispiel:

```
if x < 0
 Betragx = -x;
elseif x > 0
 Betragx = x;
else
 Betragx = 0;
end
```

Dieses if-else Konstrukt berechnet umständlich den Betrag von x.

Bei mehreren logischen Bedingungen oder bei Fallunterscheidungen eignet sich oft der Befehl `switch...case`.

functions

Funktionen werden definiert, um Anweisungsblöcke, die häufiger oder mit verschiedenen Werten benutzt werden, nur einmal zu programmieren. Einmal geschrieben, können die Funktionen mit ihren Funktionsnamen in dem eigentlichen Programm immer wieder aufgerufen werden (Vorsicht bei Doppelbenennung). Funktionen werden benutzt, um Programmabschnitte zu entkoppeln.
Beispiel:

```
A = [1 2; 3 4];
b = [1;1];
loesung = Funktionsname(A,b);
%-------------------------------
function [x] = Funktionsname(A,b)
  x = A\b;
```

Dieses Funktion löst das lineare Gleichungssystem $Ax = b$.

functions

Funktionen können als Unterprogramme in einem Programm integriert werden. Dazu definiert man diese hinter die Anweisungen des eigentlichen Programms, also ganz am Ende der Datei. Ebenso können Funktionen auch extern als m-file gespeichert werden und mit entsprechendem Funktionsname aufgerufen werden. Hierbei ist zu beachten, dass die Funktion im selben Verzeichnis wie das aufrufende Programm gespeichert ist.

cputime

Zum Messen der Programmlaufzeit sind zwei verschiedene Ansätze möglich. Zuerst wollen wir sehen wie `cputime` funktioniert
Beispiel:

```
t = cputime;
---Anweisung---
Time = cputime - t;
```

Dieses Funktion ergibt mit der Variable `Time` die Computerlaufzeit fr die Anweisung.

tic-toc

Ein weiterer Ansatz ist der Matlab Befehl `tic; toc;`.
Beispiel:

```
tic;
---Anweisung---
toc;
```

Dieses Funktion gibt die Computerlaufzeit für die Anweisung als `Elapsed time is xxxx.xxxx seconds.` aus.

Komponentenweise Matrixfunktionen

Matlab bedeutet MATrix LABoratory. Diese Programmiersprache ist darauf spezialisiert, Matrizen und damit auch Vektoren schnell und effizient zu berechnen. Daher sollte man, wann auch immer möglich auf Schleifen (`for`, `if`, `while`, `case`) verzichten und diese versuchen vektorweise zu programmieren. Wie effizent das sein kann werden wir auf der nsten Folie an einem einfachen Beispiel sehen. Dafür sind folgende komponentenweise Matrixfunktionen enorm wichtig:

Beispiel

Dieses Beispiel berechnet den `sin` für einen Vektor `A` der die ganzen Zahlen zwischen `-100` und `100` enthält.
Zuerst berechnen wir dies mittels der `for`-Schleife und lassen uns zusätzlich die Computerlaufzeit ausgeben. Beispiel:

```
A=[-100:1:100];
tic;
for i=1:1:200
    B(i)=sin(A(i));
end
toc;

Elapsed time is 0.004290 seconds.
```

Dieses Berechnung braucht die Computerlaufzeit für die Anweisung von `Elapsed time is 0.004290 seconds.`.

Beispiel

DNun vergleichen wir diese Zeit mit der Computerlaufzeit fr die vektorweise Berechnung.
Jetzt berechnen wir dies mittels der vektorweisen Eingabe. Beispiel:

```
A=[-100:1:100];
tic;
B=sin(A);
toc;

Elapsed time is 0.000070 seconds.
```

Dieses Berechnung braucht die Computerlaufzeit für die Anweisung von `Elapsed time is 0.000070 seconds.`.
Also ist -wie erwartet- die vektorweise Berechnung deutlich schneller!

profiler

Ein weiterer Ansatz zur Ausgabe der Computerlaufzeit ist der Matlab Befehl `profile`.
Anweisungen fr die `profile`-Umgebung:

```
profile on;
profile off;
profile clear;
profile report;
```

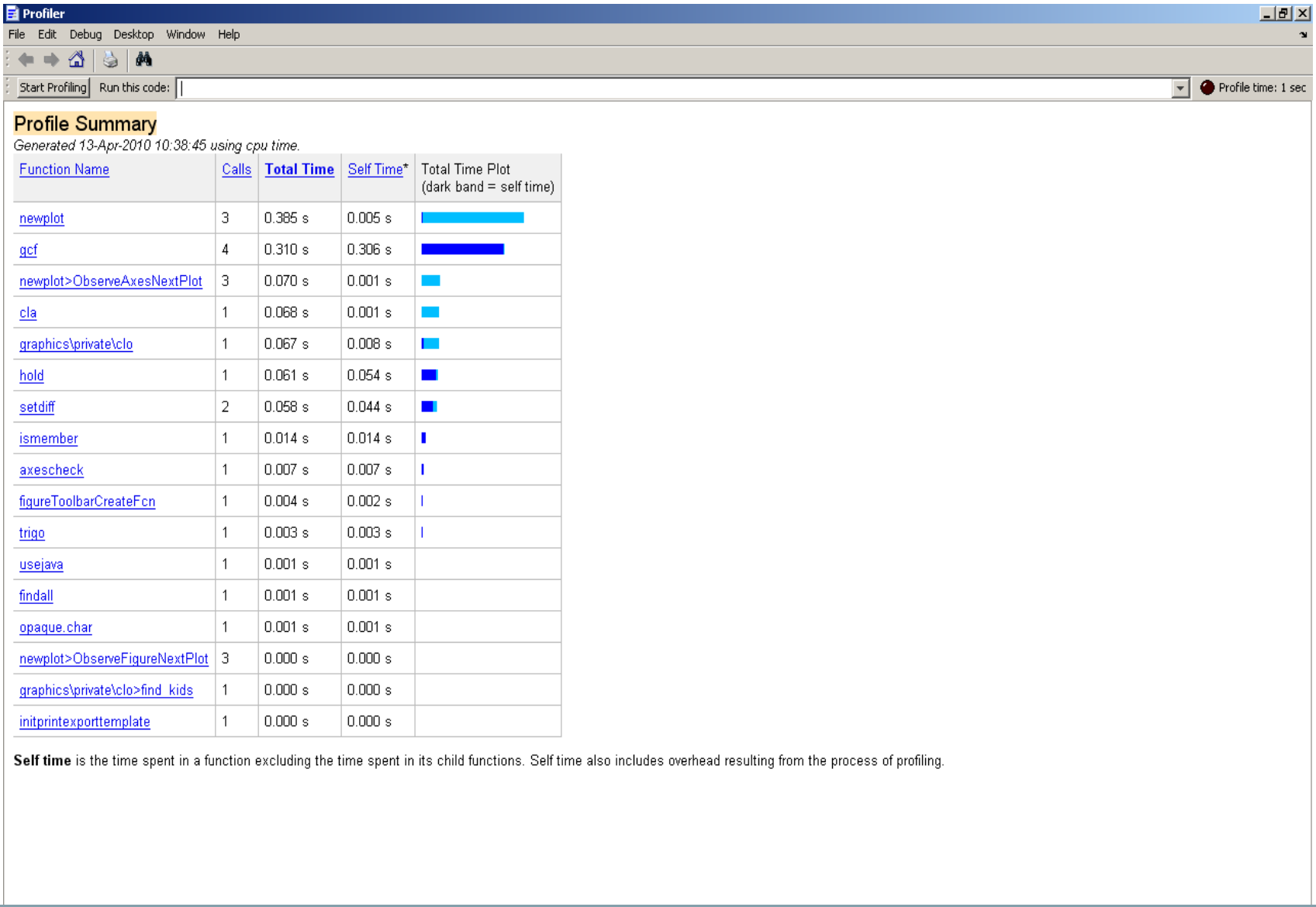
Die `profile` Umgebung bietet einen detailierten Bericht über die Laufzeiten und die Anzahl der Funktionsaufrfe.

profiler

Beispiel fr die profile-Umgebung:

```
function [sinx, cosx, fx] = trigo(x)
sinx = sin(x);
cosx = cos(x);
fx   = (cos(x.^2).*sin(x))./exp(-x);
%-------------------------------------
profile on; profile clear;
x=pi*[-5:0.01:5];
[sinx, cosx, fx] = trigo(x);
plot(x,sinx,'b-'); hold on;
plot(x,cosx,'r.-');
plot(x,fx,'g-*');
profile report;
```

profiler


Profiler
File Edit Debug Desktop Window Help
Start Profiling
Run this code:
Profile time: 1 sec
Profile Summary
Generated 13-Apr-2010 10:38:45 using cpu time.
Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time)
newplot | 3 | 0.385 s | 0.005 s
gcf | 4 | 0.310 s | 0.306 s
newplot>ObserveAxesNextPlot | 3 | 0.070 s | 0.001 s
cla | 1 | 0.068 s | 0.001 s
graphics\private\clo | 1 | 0.067 s | 0.008 s
hold | 1 | 0.061 s | 0.054 s
setdiff | 2 | 0.058 s | 0.044 s
ismember | 1 | 0.014 s | 0.014 s
axescheck | 1 | 0.007 s | 0.007 s
figureToolbarCreateFcn | 1 | 0.004 s | 0.002 s
trigo | 1 | 0.003 s | 0.003 s
usejava | 1 | 0.001 s | 0.001 s
findall | 1 | 0.001 s | 0.001 s
opaque.char | 1 | 0.001 s | 0.001 s
newplot>ObserveFigureNextPlot | 3 | 0.000 s | 0.000 s
graphics\private\clo>find_kids | 1 | 0.000 s | 0.000 s
initprintexporttemplate | 1 | 0.000 s | 0.000 s
Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

Zusammenfassung

- Programmieren ist nicht schlimm, sondern sehr hilfreich!!!
- Matlab-Hilfe bzw. Internet hilft bei vielen Problemen!!!
- Learning by Doing!!!

Zusammenfassung

- Programmieren ist nicht schlimm, sondern sehr hilfreich!!!
- Matlab-Hilfe bzw. Internet hilft bei vielen Problemen!!!
- Learning by Doing!!!
- Spaß haben!

Zusammenfassung

Weitere Informationen:
www.mathematik.uni-trier.de/~gross/
grossb@uni-trier.de

-wird fortgesetzt-