

# Einführung in Matlab

## Einheit 7

Jochen Schulz

Georg-August Universität Göttingen 

4. September 2009

1 Schnittstelle zu C

2 Mex-Dateien

3 C-Programme mit MATLAB

- MATLAB läßt sich mit anderen Programmiersprachen kombinieren.
- Die Verknüpfung geschieht über sogenannte *Schnittstellen*.
- Es existieren Schnittstellen zu C, Fortran und Java.
- Über diese Schnittstellen werden in der Regel Kommandos und insbesondere Daten übermittelt.

# Warum Schnittstellen zu C?

- Große bereits existierende C-Programme können von MATLAB aus gestartet werden, ohne dass sie als *m*-Files neugeschrieben werden müssen.
- Bottleneck Berechnungen (in der Regel Schleifen), die in MATLAB nicht schnell genug laufen, können aus Effizienzgründen in C neu programmiert werden.
- Man kann aus C-Programmen heraus, den großen Befehlsumfang von MATLAB nutzen (einfaches Erstellen von Grafiken).

Es gibt 2 Möglichkeiten, MATLAB mit C zu verbinden.

- Das Einbinden von C-Funktionen in MATLAB. Dies geschieht durch die sogenannten mex-Dateien. Sie bestehen aus 2 Teilen: der eigentlichen C-Funktion und einer Schnittstellen-Routine zwischen C und MATLAB.
- Das Starten eines MATLAB-Fensters aus einem C-Programm heraus. Hier bindet man passende Bibliotheken ein: die sogenannte Engine.

1 Schnittstelle zu C

2 **Mex-Dateien**

3 C-Programme mit MATLAB

# Erstellen von Mex-Funktionen

- Um eine Mex-Datei `mex_beispiel.c` ausführbar zu machen, kompiliere man es durch

```
mex mex_beispiel.c
```

- Die Befehlseingabe kann sowohl im Command Window von MATLAB als auch in einem beliebigen terminal (unter Linux) erfolgen (Pfad beachten).
- Die Funktion kann dann in MATLAB aufgerufen werden als sei sie ein normales m-File.

# Erstellen von Mex-Funktionen

- Mex-Dateien verhalten sich genau wie m-Files oder built-in Funktionen.
- Mex-Dateien sind plattform-abhängig.
- Die Plattform ist an der Endung zu erkennen: `mexaxp` (Alpha), `mexglx` (Linux), `mexsol` (Solaris), `dll` (Windows).



# Optionen von mex

- Auswählen des Default-Compilers durch

```
mex -setup
```

- Es ist auch möglich, von Fall zu Fall verschiedene Compiler zu benutzen. Aufruf:

```
mex filename -f optionsfile
```

Beispiele: lccengmatops.bat (MATLAB-Compiler, Windows),  
gccopts.sh (gcc, Linux)

- Durch `mex -help` erhält man weitere Informationen zum Aufruf von mex.

# Linken mehrere Files

Beim Erzeugen von mex-Routinen ist es möglich verschiedene Objekt- und Bibliothek-Files zu kombinieren. Das Beispiel erzeugt unter Windows

```
mex circle.c square.obj rectangle.c  
    shapes.lib
```

die ausführbare Datei `circle.dll` . Benutzen von Befehlen wie `make` ist möglich. Dateien werden am Ende durch `mex` zusammengebunden.

# Aufbau von mex-Files

Sie bestehen aus 2 Teilen: Einer Gateway Routine und der eigentlichen C-Funktion. Aufruf der Gateway-Funktion:

```
void mexFunction(  
    int nlhs, mxArray *plhs[],  
    int nrhs, const mxArray *prhs[])
```

- Benutzen Sie `mxCreate` Funktionen, um die MATLAB Arrays für die Output Argumente zu erzeugen. Setzen Sie `plhs [0],[1], ..` auf die Zeiger von den neuerzeugten MATLAB-Arrays.
- Benutzen Sie die `mxGet` Funktionen, um die Daten von `prhs [0],[1], ..` zu lesen.
- Aufruf der C-Unterroutine mit den Input- und Output-Zeigern als Funktionsparameter.

- Aufruf MATLAB:  $[C,D]=\text{func\_beispiel}(A,B)$ .
- Startet `func_beispiel.c`:

```
const mxArray *B,*A;  
A = prhs[0];  
B = prhs[1];  
  
mxArray *C,*D;  
C = plhs[0];  
D = plhs[1];
```

- Ergebnis MATLAB: `plhs[0]` wird in  $C$  geschrieben, `plhs[1]` wird in  $D$  geschrieben.

# Klassifizierung von Funktionen

Es gibt drei verschiedene Klassen von Funktionen im Zusammenhang mit der Schnittstelle.

- **mex-Funktionen:**

Mex-Routinen interagieren mit der MATLAB Umgebung.

Beispielsweise interpretiert `mexEvalString` einen String im MATLAB Workspace.

- **mx-Funktionen:**

Menge von Funktionen mit denen man MATLAB Arrays erzeugen und manipulieren kann.

- **Engine Funktionen:**

Menge von Funktionen, die das Arbeiten mit der MATLAB-Engine steuern.

# Größter gemeinsamer Teiler (ggT)

Berechnung des ggT von natürlichen Zahlen  $a$  und  $b$  mit Hilfe des euklidischen Algorithmus

Idee: Es gilt  $\text{ggT}(a, b) = \text{ggT}(a, b - a)$  für  $a < b$ .

Algorithmus:

Wiederhole, bis  $a = b$

- Ist  $a > b$ , so  $a = a - b$ .
- Ist  $a < b$ , so  $b = b - a$

```
function a = ggt(a,b)
%-----
% ggt berechnet den groessten gemeinsamen Teiler (ggT)
%      zweier natuerlichen Zahlen a und b
%      INPUT:  natuerliche Zahlen a
%               b
%
%      OUTPUT: ggT
%
% Gerd Rapin      11.11.2003
%-----
while (a ~= b)
    if (a > b)
        a = a-b;
    else
        b = b-a;
    end
end
end
```



## C-File: ggt.c (Teil I)

```
/*  
 *  
 *      ggt.c  
 *  
 */  
#include "mex.h"  
  
void ggt( double    result [], double a, double b);  
  
void mexFunction( int nlhs, mxArray *plhs [],  
                  int nrhs, const mxArray *prhs [])  
{  
    double *a,*b,*result;  
  
    /* Erzeuge Matrix fuer das Rueckgabe-Argument. */  
    plhs[0] = mxCreateDoubleMatrix(1,1, mxREAL);
```

## C-File: ggt.c (Teil II)

```
/* Die pointer fuer die Variablen setzen */
a = mxGetPr(prhs[0]);
b = mxGetPr(prhs[1]);
result = mxGetPr(plhs[0]);
/* Aufruf der ggt Routine */
ggt ( result , *a, *b );
}

void ggt( double result [], double a, double b)
{
    while (a!=b)
    {
        if (a>b)
            a = a - b;
        else
            b = b - a;
    }
    result[0] = a;
}
```

- Einbinden der Header Datei

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

- Definieren eines Zeigers  $x$  auf ein Objekt vom MATLAB-Typ `mxArray`.

```
mxArray *x = NULL;
```

Zeiger vom Typ `mxArray` dienen zur abstrakten Zuweisung von MATLAB-Datenstrukturen.

# Erzeugen von Rückgabe-strukturen

- Definieren von double-Matrizen

```
mxArray *mxCreateDoubleMatrix(int m,  
int n, mxComplexity Flag);
```

$m$  ist die Anzahl von Zeilen,  $n$  die Anzahl von Spalten und Flag ist entweder *mxREAL* or *mxCOMPLEX*.

- Definieren eines double-Skalars mit Wert value

```
mxArray *mxCreateDoubleScalar  
(double value);
```

mxCreateCellArray	Array für Cell-Arrays
mxCreateCharArray	Array von Characters
mxCreateString	String
mxCreateSparse	Sparse Matrix
mxCreateLogicalMatrix	Array für Logicals

# Zugriff auf mxArray

```
double *mxGetPr(mxArray *array_ptr)
```

Rückgabeargument ist ein Zeiger, der auf das erste (reelle) Element des Arrays `*array_ptr` zeigt. Für imaginäre Elemente `mxGetPi`.

Es findet keine automatische Überprüfung der Ein- und Ausgabeparameter statt. 2 Möglichkeiten:

- (a) Man versieht die Gateway-Routine mit entsprechenden Abfragen.
- (b) Man kapselt die mex-Routine durch eine MATLAB-Routine, die die Parameter überprüft.

## C-File: ggt\_2.c (Auszug)

```
/* Ueberpruefen der Anzahl von Argumenten */
if(nrhs!=2) {
    mexErrMsgTxt("Genau zwei Input-Variablen erforderlich.");
} else if(nlhs!=1) {
    mexErrMsgTxt("Falsche Anzahl an Output-argumente");
}
/* Input Variablen muessen nichtnegative Double sein.*/
mrows = mxGetM(prhs[0]);
ncols = mxGetN(prhs[0]);
if( !mxIsDouble(prhs[0]) || !(mrows==1 && ncols==1) ) {
    mexErrMsgTxt("Erster Eingabeparameter muss
        ein reelles Skalar sein.");
}
mrows = mxGetM(prhs[1]);
ncols = mxGetN(prhs[1]);
if( !mxIsDouble(prhs[1]) || !(mrows==1 && ncols==1) ) {
    mexErrMsgTxt("Zweiter Eingabeparameter muss
        ein reelles Skalar sein.");
}
```



- Abfragen der Zeilen- bzw. Spaltenanzahl eines mxArray:

```
int mxGetM( const mxArray *array_zeiger )  
int mxGetN( const mxArray *array_zeiger )
```

- Fehlermeldung:

```
mexErrMsgTxt( 'Fehler .... ' )
```

erzeugt eine entsprechende Fehlermeldung in MATLAB und beendet die Routine.

```
bool mxIsDouble( mxArray *array_zeiger );
```

ist wahr (1), falls durch \*array\_zeiger eine Matrix mit double oder float repräsentiert werden, sonst ist der Rückgabewert 0.

Weitere Abfragen:

mxIsComplex, mxIsChar, mxIsInf, mxInt64, mxInt32, mxIsNaN.

# Mandelbrot-Menge

Die Mandelbrot-Menge ist die Menge von Punkten  $c \in \mathbb{C}$  bei denen die Folge  $(z_n)_n$ , die durch

$$z_0 := c, \quad z_{n+1} = z_n^2 + c, \quad n \in \mathbb{N}$$

definiert ist, beschränkt ist.

- Gilt  $|z_n| \geq 2$ , so wird die Folge divergieren.
- Wir benutzen dies als Abbruchkriterium.

Mit  $z = z_1 + iz_2$ ,  $x = x_1 + ix_2$  und  $c = c_1 + ic_2$  ergibt sich aus

$$\begin{aligned} z_1 + iz_2 &= z = x^2 + c = (x_1 + ix_2)^2 + (c_1 + ic_2) \\ &= [x_1^2 - x_2^2 + c_1] + i[2x_1x_2 + c_2] \end{aligned}$$

die Formel

$$z_1 = x_1^2 - x_2^2 + c_1, \quad z_2 = 2x_1x_2 + c_2.$$

# Programm - MATLAB

```
MAX_IT = 150;
x1 = linspace(-2.1,0.6,601);
y1 = linspace(-1.1,1.1,401);
C = zeros(length(y1),length(x1));
for i = 1:length(x1)
    for j = 1:length(y1)
        % Berechnen der Folge
        m = 0; a = x1(i); b = y1(j);
        x = a; y = b;
        while (sqrt(x^2+y^2)<2 & m<MAX_IT)
            t = x; x = a+x^2-y^2;
            y = b+2*t*y; m = m+1;
        end;
        C(j,i) = m;
    end
end
C = (1/MAX_IT) * C;
contourf(x1,y1,C,20);
```

## Routine - C (Teil I)

```
#include "mex.h"
#include <math.h>

void mandel_c( double result [], double x1 [], double y1 [],
               int x1_laenge, int y1_laenge );

void mexFunction( int nlhs, mxArray *plhs [],
                  int nrhs, const mxArray *prhs [] )
{
    double *a,*b,*result;
    int acols, bcols;

    acols = mxGetN(prhs[0]);
    bcols = mxGetN(prhs[1]);
```

## Routine - C (Teil II)

```
printf("\n Erzeuge (%d x %d)-Matrix \n \n", bcols, acols);

/* Erzeuge Matrix fuer das Rueckgabe-Argument. */
plhs[0] = mxCreateDoubleMatrix(bcols, acols, mxREAL);

/* Die pointer fuer die Variablen setzen. */
a = mxGetPr(prhs[0]);
b = mxGetPr(prhs[1]);
result = mxGetPr(plhs[0]);

/* Aufruf der Routine */
mandel_c ( result, a, b, acols, bcols);
}
```

## Routine - C (Teil III)

```
void mandel_c( double result [], double x1 [], double y1 [],
    int x1_laenge, int y1_laenge)
{
    int MAX_IT = 150;
    int i, j;
    double m, a, b, x, y, t;
    for (i = 0; i < x1_laenge; i++)
    {
        for (j = 0; j < y1_laenge; j++)
        {
            /* Berechnen der Folge */
            ...
            while (sqrt(x*x+y*y)<2 & m<MAX_IT)
            {
                ...
            }
            result[i*y1_laenge+j] = m;
        }
    }
}
```



# Aufruf aus MATLAB

```
MAX_IT = 150;

tic;
x1 = linspace(-2.1,0.6,201);
y1 = linspace(-1.1,1.1,401);

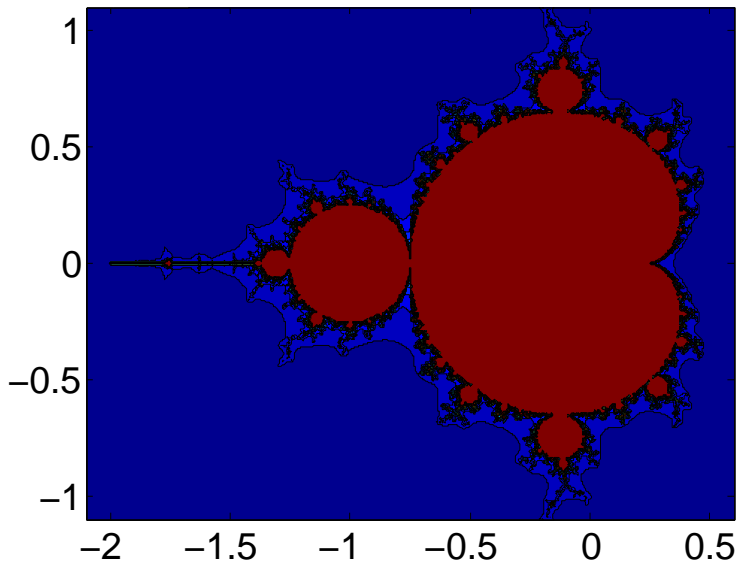
C = mandel_c(x1,y1);

disp('Benoetigte Zeit')
toc;

C = (1/MAX_IT) * C;

% Plotten der Funktion
contourf(x1,y1,C,20);
```

# Die Mandelbrotmenge



1 Schnittstelle zu C

2 Mex-Dateien

3 C-Programme mit MATLAB

# Kompilieren unter Linux

Befehle und Pfade gelten für MATLAB 7 (R2009a).

- Einbinden der Shared Libraries (nur Linux)

- Bourne Shell

```
LD_LIBRARY_PATH=/usr/local/matlab2009a/bin/glnx86/:\  
/usr/local/matlab2009a/sys/os/glnx86:$LD_LIBRARY_PATH  
export LD_LIBRARY_PATH
```

- C-Shell

```
setenv LD_LIBRARY_PATH \  
$MATLAB/bin/glnx86/:$LD_LIBRARY_PATH
```

- Kompilieren von test.c

```
/usr/local/matlab2009a/bin/mex -f \  
/usr/local/matlab2009a/bin/engopts.sh test.c
```

- Aufruf in MATLAB von

```
mex -f lccengmatopts.bat datei.c
```

kompiliert die Datei datei.c.

- Starten durch Ausführen von datei.exe.
- Alternativ ein DOS-Fenster öffnen, ins richtige Verzeichnis wechseln und dort das Programm durch Eingabe von datei starten.

# Erstes Beispiel

- Das C Programm öffnet ein MATLAB Fenster.
- Dort wird eine Hilbert-Matrix erzeugt.
- Die Eigenwerte der Matrix werden berechnet.
- Die Eigenwerte werden grafisch veranschaulicht.

```
#include <stdio.h>
#include "engine.h"

main(int argc, char* argv[])
{
    Engine *ep;
    mxArray *x_m = NULL;

    double n=10;

    printf("\n Open MATLAB engine...\n");
    ep = engOpen(NULL);

    x_m = mxCreateDoubleMatrix(1, 1, mxREAL);
    *mxGetPr(x_m) = n;
```

## hilbert1.c (Forts.)

```
engPutVariable(ep, "x_m", x_m);  
engEvalString(ep, "a=hilb(x_m)");  
engEvalString(ep, "semilogy(eig(a), '*' )");  
  
printf("Please press Return \n");  
fgetc(stdin);  
engClose(ep);  
}
```



# Umgang mit der MATLAB-Engine

- Einbinden der Bibliothek

```
#include "engine.h"
```

- Anlegen eines Handles für die MATLAB-Engine

```
Engine *ep;
```

- Öffnen der MATLAB-Engine

```
engOpen(NULL)
```

- Schliessen der MATLAB-Engine:

```
engClose(ep);
```

- Benenne die Variable 'name' in MATLAB. Die Variable wird mit den Daten werte versehen.

```
engPutVariable(ep, "name", werte);
```

- Ausführen von MATLAB-Befehlen:

```
engEvalString(ep, "Befehl");
```

Startet das Kommando Befehl in der MATLAB-Engine ep.

## Zweites Beispiel

- Das C Programm öffnet ein MATLAB Fenster.
- Dort wird eine Hilbert-Matrix erzeugt.
- Die Eigenwerte der Matrix werden berechnet.
- Die Eigenwerte werden an das C-Programm zurückgegeben und dort ausgegeben.

# hilbert2.c (Teil I)

```
#include <stdio.h>
#include "engine.h"

main(int argc, char* argv[])
{
    Engine *ep;
    mxArray *x_m = NULL;
    mxArray *d = NULL, *le = NULL;
    double *Dreal;
    double laenge;
    int i;

    double n=10;

    printf("\n Open MATLAB engine...\n");
    ep = engOpen(NULL);
```

## hilbert2.c (Teil II)

```
x_m = mxCreateDoubleMatrix(1, 1, mxREAL);
*mxGetPr(x_m) = n;

engPutVariable(ep, "x_m", x_m);
engEvalString(ep, "d=eig(hilb(x_m))");
engEvalString(ep, "le=length(d)");

d = engGetVariable(ep, "d");
le = engGetVariable(ep, "le");
```

## hilbert2.c (Teil III)

```
Dreal = mxGetPr(d);  
laenge = *mxGetPr(le);  
  
engClose(ep);  
  
for (i=0; i<laenge; i++)  
{  
    printf ( "%d. Eigenwert %g \n" , i+1,Dreal[i] );  
}  
  
mxDestroyArray(x_m);  
mxDestroyArray(d);  
}
```

- Kopieren der Variable  $d$  aus dem MATLAB-Workspace in das C-Programm.

```
mxArray *engGetVariable(ep, "d");
```

- Freigeben des Speichers (Wichtig!)

```
mxDestroyArray(mxArray *x_m);
```

- `plot_poisson.c` plottet die Funktion  $f(x, y) = \sin(4\pi x) \sin(2\pi y)$  auf  $[0, 1] \times [0, 1]$ .
- In C betrachten wir das Gitter, das durch Zerlegen der x- und der y-Richtung in 50 Punkten entsteht.
- Dann berechnen wir die Funktionswerte in C.
- Das Gitter und die berechneten Vektoren werden an MATLAB übergeben.
- Dort wird die Lösung mit Hilfe des Befehls `surf` gezeichnet.



# plot-poisson.c (Teil I)

```
/* _____  
-      plot_poisson.c  
_____*/  
  
#include <stdio.h>  
#include <math.h>  
#include <string.h>  
#include "engine.h"  
  
#define MAX_ORDER 50  
  
/* Function for plotting data on cartesian grids */  
int plot_graph(  
    double *x,    /* vector of x-values */  
    double *y,    /* vector of y-values */  
    double *z,    /* value at (x[i],y[j]) (row-wise) */  
    int x_n,      /* size of array x */  
    int y_n);    /* size of array y */
```

## plot-poisson.c (Teil II)

```
/*-----  
-               main program               -  
-----*/  
  
main(int argc, char* argv[])  
{  
    double x[MAX_ORDER];  
    double y[MAX_ORDER];  
    double z[MAX_ORDER*MAX_ORDER];  
    int x_n,y_n;  
    int i,j;  
    x_n = 50;  
    y_n = 50;
```

## plot-poisson.c (Teil III)

```
for (i=0; i<x_n; i++)
{
    x[i] = (double) i/(x_n-1);
}
for (i = 0; i<y_n; i++)
{
    y[i] = (double) i/(y_n-1);
}
for (i = 0; i<x_n; i++)
{
    for (j = 0; j<y_n; j++)
    {
        *(z+i+j*x_n) = sin(4*3.14159*x[i])*sin(2*3.141459*y[j]);
    }
}
if (plot_graph(x,y,z,x_n,y_n)==0)
    printf("\n Ungueltiger Aufruf von 'plot_graph' \n");
return 0;
}
```

## plot-poisson.c (Teil IV)

```
/*———— function 'plot_graph' —————*/
int plot_graph(double *x, double *y, double *z, int x_n, int y_n)
{
    Engine *ep;
    mxArray *x_m = NULL;
    mxArray *y_m = NULL;
    mxArray *z_m = NULL;
    int i, j;
    if ((x_n == 0) || (y_n == 0))
        return (int) 0;
    printf("\n Open MATLAB engine...\n");
    /*———— Start MATLAB engine */
    if (!(ep = engOpen("\0"))) {
        fprintf(stderr, "\n Can't start MATLAB engine\n");
        return EXIT_FAILURE;
    }
    printf("Create MATLAB arrays...\n");
    x_m = mxCreateDoubleMatrix(1, x_n, mxREAL);
    y_m = mxCreateDoubleMatrix(1, y_n, mxREAL);
    z_m = mxCreateDoubleMatrix(x_n, y_n, mxREAL);
}
```

## plot-poisson.c (Teil V)

```
printf("Copy entries into MATLAB ...\n");
memcpy((void *)mxGetPr(x_m), (void *) x, x_n*sizeof(double));
memcpy((void *)mxGetPr(y_m), (void *) y, y_n*sizeof(double));
memcpy((void *)mxGetPr(z_m), (void *) z, x_n*y_n*sizeof(double));
engPutVariable(ep, "x_m", x_m);
engPutVariable(ep, "y_m", y_m);
engPutVariable(ep, "z_m", z_m);
printf("Execute MATLAB commands ...\n");
engEvalString(ep, "[Y,X]=meshgrid(y_m,x_m)");
engEvalString(ep, "surf(X,Y,z_m)");
engEvalString(ep, "xlabel('x')");
engEvalString(ep, "ylabel('y')");
printf("Please press Return \n");
fgetc(stdin);
printf("\n Close MATLAB engine... \n");
engClose(ep);
return (int) 1;
}
```