Wima 1 - Praktikum (Woche 7)

Lernziele

In diesem Praktikum sollen Sie üben und lernen:

- Einbinden von C-Code in MATLAB
- interne Datenspeicherung von MATLAB
- Umgang mit dem Compiler `mex`

Am Anfang geben wir Ihnen einen kurzen Überblick über die benötigten MATLAB-Anweisungen.

Beantworten Sie danach bitte erst einige Fragen bzw. einige off-line Aufgaben, bevor Sie sich einloggen!

Mex-Schnittstelle

MATLAB erlaubt das Einbinden von C-Code, dies ermöglicht häufig einen Geschwindigkeitsgewinn oder existierende C-Funktionen/Bibliotheken können verwendet werden. Um das Einbinden zu erlauben, müssen Schnittstellen-Funktionen in C geschrieben werden. Anhand einfacher Beispiele wollen wir die Funktionsweise von Schnittstellenfunktionen erläutern.

`Hello World!`-Beispiel

Betrachten wir als erstes die folgende mex-Datei `ex07x01.c`:

```
1  #include "mex.h"      /* beachte "..", nicht <..> */
2
3  void mexFunction(int nlhs, mxArray* plhs[],
4                   int nrhs, const mxArray* prhs[])
5  {
6    if (nlhs>0 || nrhs>0)
7      mexPrintf("Diese Funktion hat keine Parameter!\n");
8    else
9      mexPrintf("Hello world, again!\n");
10   return;
11 }
```

Wenn Sie den mex-Compiler das erste Mal verwenden sollten, müssen Sie vorab den zugrundeliegenden Compiler auswählen. Geben Sie hierzu `mex -setup` an der Kommandozeile ein und wählen Sie z.B. `lcc` oder `gcc`, wenn vorhanden, aus. Ist der mex-Compiler nun initialisiert, erzeugen Sie mit dem Befehl `mex ex07x01.c` eine MATLAB-Funktion `ex07x01`. Wenn Sie diese am MATLAB-Prompt aufrufen, erhalten Sie die Ausgabe

```
≫ ex07x01(1)
Diese Funktion hat keine Parameter!
≫ ex07x01
Hello world, again!
≫ a = ex07x01
Diese Funktion hat keine Parameter!
One or more output arguments not assigned during call to "ex07x01".
```

Was kann man aus dem Beispiel festhalten:

- Es wird **immer** eine Header-Datei `mex.h` mit dem Befehl `#include "mex.c"` am Anfang der Datei eingebunden.
- Die `main`-Funktion entfällt.
- Die eigentliche Schnittstellen-Funktion heißt `mexFunction` und hat die Form

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

  - `nlhs` = Number Left-Hand Side = Anzahl Output-Parameter
  - `plhs` = Pointer Left-Hand Side = Array Output-Parameter
    (Man beachte, dass dieser Speicherplatz allokiert werden muss!)
  - `nrhs` = Number Right-Hand Side = Anzahl Input-Parameter
  - `prhs` = Pointer Right-Hand Side = Array Input-Parameter

- Möchte man Sachen auf der Shell ausgeben, so ist die C-Anweisung `printf` durch `mexPrintf` zu ersetzen.

Eine Tabelle mit den wichtigsten Funktionen zur Erstellung einer `mex-datei` ist im Folgenden gegeben.

`mex`-**Funktion**	**Bedeutung**
mxCreateDoubleScalar	erzeugt einen Skalar
mxCreateDoubleMatrix(m,n,flag)	erzeugt eine Matrix der Dimension $m \times n$ `flag=mxREAL` reelle Matrix `flag=mxCOMPLEX` komplexe Matrix
mxCreateString	erzeugt einen String
mxGetM	liefert die Zeilendimension
mxGetN	liefert die Spaltendimension
mxGetPr	liefert einen Pointer auf den Realteil
mxGetPi	liefert einen Pointer auf den Imaginärteil
mexErrMsgTxt	Gibt eine Fehlermeldung aus und kehrt zum MATLAB-Prompt zurück
mxIsDouble	prüft, ob Pointer auf double Fließkommazahl zeigt
mxIsComplex	prüft, ob Pointer auf komplexe Zahl zeigt, d.h. es gibt Real- und Imaginärteil
mexPrintf	ermöglicht Ausgabe im Befehlsfenster
mxMalloc	statt malloc
mxFree	statt free
mxRealloc	statt realloc die MATLAB-Versionen zur Speicherverwaltung überlassen die Speicherverwaltung MATLAB, so dass bei Programmabbruch mit `Ctrl-C` auch der Speicher fehlerfrei freigegeben wird.

Ein Beispiel mit Ein-und Ausgabeparametern

Die folgende `mex`-Funktion erzeugt eine $m \times n$-Matrix $A = (a_{ij})$ mit $a_{ij} = i + j$

```
1  #include "mex.h"
2  void mexFunction(int nlhs, mxArray* plhs[],
3       int nrhs, const mxArray* prhs[])
4  {
5    int i, j, M, N, mrows, ncols;
6    double *mpointer, *npointer, *A;
7
8    if (nrhs != 2)
9      mexErrMsgTxt("Two inputs required!");
10   else if (nlhs > 1)
11     mexErrMsgTxt("Too many output arguments!");
12   /* The inputs must be noncomplex scalar doubles.*/
13   for (i=0; i<2; ++i)
14   {
15     mrows = mxGetM(prhs[i]);
16     ncols = mxGetN(prhs[i]);
17     if( !mxIsDouble(prhs[i]) || mxIsComplex(prhs[i]) ||
18         !(mrows==1 && ncols==1) ) {
19       mexErrMsgTxt("Input must be noncomplex scalar double.");
20     }
21   }
22   /* Assign pointers to each input. */
23   mpointer = mxGetPr(prhs[0]);
24   npointer = mxGetPr(prhs[1]);
25   M = (int) mpointer[0];
26   N = (int) npointer[0];
27   /* Create and fill matrix A */
28   plhs[0] = mxCreateDoubleMatrix(M,N,mxREAL);
29   A = mxGetPr(plhs[0]);
30   for (i=0; i<M; ++i) {
31     for (j=0; j<N; ++j) {
32      A[i+j*M] = i + j + 2;
33     }
34   }
35 }
```

- Zeile 1: Einbinden der Header-Datei `mex.h`
- Zeile 2-3: Parameterliste der `mex`-Schnittstellenfunktion (Zeilen bleiben unverändert!)

- Zeile 5-6: Typvereinbarungen
- Zeile 8-11: Prüfen (während der Laufzeit), ob die Anzahl der Eingabe- und Ausgabeparameter den Vorgaben entspricht. Hier soll die Dimension der $m \times n$-Matrix als erster und zweiter Parameter m und n übergeben werden. Die Matrix A soll dann an die Standardvariable `ans` oder eine vorgegebene Variable zurückgegeben werden, daher ist die Anzahl der Rückgabewerte 0 und 1 zugelassen, d.h. erst `nlhs` > 1 führt zu einem Fehler.
- Zeile 13-21: Der Pointer `prhs` zeigt auf die Eingabeparameter, der Pointer auf den ersten Eingabeparameter ist `prhs[0]`. Man beachte, dass C mit Null anfängt zu indizieren. Die Anzahl der Zeilen des $k+1$-ten Eingabeparameters, kann man mit `mxGetM(prhs[k])` auslesen. Analog liefert `mxGetN(prhs[k])` die Anzahl der Spalten des $k+1$-ten Eingabeparameters. Mit `mxIsDouble` und `mxIsComplex` kann man überprüfen, ob es sich um eine rein reelle oder komplexe Zahl handelt. Es gibt keine Funktion, die überprüft, dass das Argument ganzzahlig ist.
- Zeile 25-26: Die Eingabewerte werden in ganzzahlige Werte überführt.
- Zeile 28: Erzeugen der reellwertigen Rückgabematrix der Dimension $M \times N$ als erster Rückgabewert, d.h. `plhs[0]`. Auch hier beginnt die Indizierung wieder mit Null.
- Zeile 29: Zuweisen der Adresse der Rückgabematrix an die Variable A, d.h. nun weiss das C-Programm wo es hinschreiben kann.
- Zeile 30-34: Hier wird die $M \times N$-Matrix $A = (a_{ij})$ mit $a_{ij} = i+j$ erzeugt. Man beachte, die Matrix ist in C als Vektor angelegt, wobei die Einträge spaltenweise durchlaufen werden. a_{11} ist somit der erste Eintrag, dann kommen a_{21}, a_{31}, $\ldots a_{M1}$, a_{12} bis a_{MN}. Beachtet man nun noch, dass die Indizierung in C bei Null beginnt so erhält man z.B. im Falle einer 2×3-Matrix die folgende Zuordung von Matrixeintrag a_{ij} und Adresse im C-Vektor:

$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$
a_{11}	a_{21}	a_{12}	a_{22}	a_{13}	a_{23}

oder im Falle einer $M \times N$-Matrix

$A[0]$	$A[1]$	$\ldots$	$A[M-1]$	$A[M]$	$\ldots$	$A[j-1+M(k-1)]$	$\ldots$	$A[MN-1]$
a_{11}	a_{21}	$\ldots$	a_{M1}	a_{12}	$\ldots$	a_{jk}	$\ldots$	a_{MN}

Lässt man nun die Indizes j und k nicht von 1 bis M bzw. N sondern von 0 bis $M-1$ bzw. $N-1$ laufen, so ergibt sich die folgende Zuweisung, dass $A[j+kM]$ dem Matrixkoeffizienten $a_{1+j,1+k}$ entspricht. Damit erklärt sich, dass in Zeile 35 für i, j dem Matrixkoeffizienten $a_{1+j,1+k}$ der Wert $i+j+2$ zugewiesen wird.

```c
#include "mex.h"
void timestwo(double *y, double *x)
{
  y[0] = 2.0*x[0];
}

void mexFunction( int nlhs, mxArray* plhs[],
                  int nrhs, const mxArray* prhs[] )
{
  double *x,*y;
  int mrows,ncols;

  /* Check for proper number of arguments. */
  if(nrhs!=1) {
    mexErrMsgTxt( "One input required.");
  } else if(nlhs>1) {
    mexErrMsgTxt( "Too many output arguments.");
  }

  /* The input must be a noncomplex scalar double.*/
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
      !(mrows==1 && ncols==1) ) {
    mexErrMsgTxt("Input must be a noncomplex scalar double.");
  }

  /* Create matrix for the return argument. */
  plhs[0] = mxCreateDoubleMatrix(mrows, ncols, mxREAL);

  /* Assign pointers to each input and output. */
  x = mxGetPr(prhs[0]);
  y = mxGetPr(plhs[0]);

  /* Call the timestwo subroutine. */
  timestwo(y,x);
}
```

In der letzten mex-Datei ist nun die „eigentliche“ C-Funktion außerhalb der `mexFunction` definiert.

- Zeile 2-5: Es wird der Wert eines Skalars in der Funktion `timestwo` verdoppelt. Man beachte, dass hier die Variablen mit „call by reference“ übergeben werden, d.h. die Adresse wird übergeben, wo die Variablen abgelegt sind und nicht ihr expliziter Wert. Der erste Wert in dem Datenfeld `x` wird mit `x[0]` ausgelesen und mit `y[0]= 2*x[0]` dem ersten Datenfeld von `y` verdoppelt zugewiesen.
- Zeile 32-36: Man mache sich klar, was sich hinter diesen Zeilen verbirgt, bzw. was hier genau geschieht.

Als letztes wollen wir nun die eigentliche C-Funktion und die mex-Schnittstelle in zwei Dateien speichern, denn später haben Sie vielleicht eine feste C-Bibliothek, die Sie von MATLAB getrennt pflegen wollen. Es folgen `timestwo.c` und `ex07x04.c`.

```
1 void timestwo(double *y, double *x)
2 {
3   y[0] = 2.0*x[0];
4 }
```

```
1 #include "mex.h"
2
3 void timestwo(double *y, double *x);
4
5 void mexFunction( int nlhs, mxArray* plhs[],
6                   int nrhs, const mxArray* prhs[] )
7 {
8   double *x,*y;
9   int mrows,ncols;
10
11  /* Parameter checks as above, rows 13.-26. */
12
13  /* Create matrix for the return argument. */
14  plhs[0] = mxCreateDoubleMatrix(mrows, ncols, mxREAL);
15
16  /* Assign pointers to each input and output. */
17  x = mxGetPr(prhs[0]);
18  y = mxGetPr(plhs[0]);
19
20  /* Call the timestwo subroutine. */
21  timestwo(y,x);
22 }
```

- Zeile 3: Funktionsdeklaration von `timestwo`. Man beachte, es wird nur der Funktionskopf gefolgt von einem Semikolon angegeben.

Mit

```
≫ mex -c timestwo.c
≫ mex ex07x04.c timestwo.o
```

erzeugen Sie zuerst mit der Option `-c` einen Objektcode `timestwo.o` und mit `mex ex07x04.c timestwo.o` kompilieren Sie zuerst `ex07x04.c` und linken es mit `timestwo.o` zu einer in MATLAB ausführbaren Funktion. Ein einfacher Aufruf liefert

```
≫ ex07x04(3)

ans =

     6
```

Wenn man nun `help ex07x04` eingibt, erhält man keine Hilfe bzw. Dokumentation. (Haben Sie ja auch noch nicht angelegt!!!) Hierzu legen wir eine zusätzliche Datei `ex07x04.m` an.

```
1 % timestwo function whose output is two times its input.
2 %    This MATLAB file illustrates how to construct an MATLAB
3 %    mex-function that computes an output value based upon
4 %    its input.  The output of this function is two times
5 %    the input value:
6 %
7 %      y = 2 * u;
8
9 % erzeugt zu Ostern vom Hasen
```

```
≫ help ex07x04
   timestwo function whose output is two times its input.
     This MATLAB file illustrates how to construct an MATLAB
     mex-function that computes an output value based upon
     its input.  The output of this function is two times
     the input value:

       y = 2 * u;
```

Offline Aktivitäten

Übereinstimmen

Schreiben Sie vor jeden Begriff auf der linken Seite den passenden Buchstaben der Beschreibung, die am besten mit der aus der rechten Spalte übereinstimmt.

_____	1. `;`	a.	Steht am Ende einer Prototypendeklaration
_____	2. `mxGetN`	b.	Anzahl Output-Parameter
_____	3. `nlhs`	c.	Geschwindigkeitsgewinn
_____	4. Pseudocode	d.	Pointer Right-Hand Side
_____	5. `mex.h`	e.	Headerdatei
_____	6. `prhs`	f.	Ermöglicht Ausgabe im MATLAB-Befehlsfenster.
_____	7. `mxGetM`	g.	liefert Zeilendimension
_____	8. `mexPrintf`	h.	liefert die Spaltendimension
_____	9. C-Code in MATLAB	i.	Compiler Option erzeugt Objektcode
_____	10. `-c`	j.	Veranschaulichung eines Algorithmus.

Ihre Antwort:

Füllen Sie die Lücken aus

Ergänzen Sie die folgenden Sätze.

11. Mit `#include` werden in C _____ Dateien eingebunden.
12. Mit _____ erzeugt man in C eine Matrix die an MATLAB zurückgegeben werden kann.
13. Standardmäßig wird in einer Schnittstellenfunktion die Headerdatei _____ eingebunden.
14. Die Dokumentation zu einer `mex`-Funktion wird in eine separate _____ -Datei geschrieben.
15. Beim Übersetzen von C-Code in eine ausführbare `mex`-Funktion wird der Code _____ und gelinkt.
16. Mit der Option _____ kann man beim `mex`-Compiler den zugrundeliegenden Compiler auswählen.
17. Die Eingabeparameter sollten am Anfang einer Schnittstellenfunktion immer auf ihre _____, ihre Größe und ihren Typ hin überprüft werden.
18. In C gibt es _____ und call-by-value.
19. Die `main`-Funktion gibt es bei einer Schnittstellenfunktion _____.
20. Die Rückgabeparameter sollten am Anfang einer Schnittstellenfunktion auf ihre _____ überprüft werden.

Ihre Antwort:

Kurz und knapp

Geben Sie bitte eine kurze Antwort zu jeder der folgenden Fragen. Ihre Antwort sollte so kurz und präzise wie möglich sein; versuchen Sie es mit zwei bis drei Sätzen.

21. Erläutern Sie kurz den Vorteil C-Code in MATLAB einbinden zu können.

    Ihre Antwort:

22. Warum ist die Typüberprüfung am Anfang einer Schnittstellenfunktion wichtig?

    Ihre Antwort:

Korrigieren Sie den Code

Für jedes der folgenden Codesegmente sollen Sie feststellen, ob ein Fehler enthalten ist. Falls ein Fehler vorliegt, markieren Sie diesen und spezifizieren Sie, ob es sich dabei um einen Semantik- oder Syntaxfehler handelt. Schreiben Sie die korrigierten Anweisungen jeweils in jeden dafür vorgesehenen Bereich unter der Problemstellung. Falls das Segment keinen Fehler enthält, schreiben Sie einfach „kein Fehler“. [Bemerkung: Es kann sein, dass ein Programm mehrere Fehler enthält.]

23. Das folgende C-Segment soll kontrollieren, ob der erste Eingabeparameter ein reellwertiger Skalar vom Typ double ist.

```
1   /* The first input must be a noncomplex scalar double.*/
2   mrows = mxGetM(prhs[1]);
3   ncols = mxGetN(prhs[1]);
4   if( !mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
5       !(mrows==1 && ncols==1) ) {
6     mexErrMsgTxt("First input must be a double scalar.");
7   }
```

Ihre Antwort:

24. Die folgende C-Funktion soll $\alpha x + y$ der Variablen y zuweisen. Hierbei seien x, $y \in \mathbb{R}^n$ und $\alpha \in \mathbb{R}$. Die Variablen x und y sollen mittels call-by-reference übergeben werden, der Skalar α durch call-by-value.

```
1 void daxpy(double *y, double *x, double alpha, int n)
2 {
3   int k;
4   for (k=0; k<=n; k++) {
5     y[k] += alpha*x[k];
6   }
7 }
```

Ihre Antwort:

Praktikumsaufgabe - mex-Schnittstelle

Lesen Sie die Aufgabenstellung, studieren Sie dann die vorgegebenen Programmzeilen. Ersetzen Sie dann die %% Kommentare im vorgegebenen Code durch Matlab-Anweisungen und führen Sie das Programm aus.

---

25. Schreiben Sie ein mex-File `axpy.c`, welches die folgende C-Routine in MATLAB einbindet. Die C-Routine berechnet zu gegebenen Vektoren x, $y \in \mathbb{R}^n$ und Skalar α den Vektor $z \in \mathbb{R}^n$.

```
1 void axpy(double *x, double *y, double *z,
2           double alpha, int n)
3 {
4   int k;
5   for (k=0; k<n; k++) {
6     z[k] = alpha * x[k] + y[k];
7   }
8 }
```

Ein einfacher Test sollte das folgende liefern.

```
≫ axpy(2,[1;2;3],[1;1;1])

ans =

     3
     5
     7
```

26. Schreiben Sie ein mex-File `rankone.c`, welche zu gegebenen Vektoren $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ die Rang-1-Matrix $x \cdot y^T$ zurückgibt. Ein einfacher Test sollte das folgende liefern.

```
≫ rankone([1;2],[1;1;1])

ans =

     1     1     1
     2     2     2
```