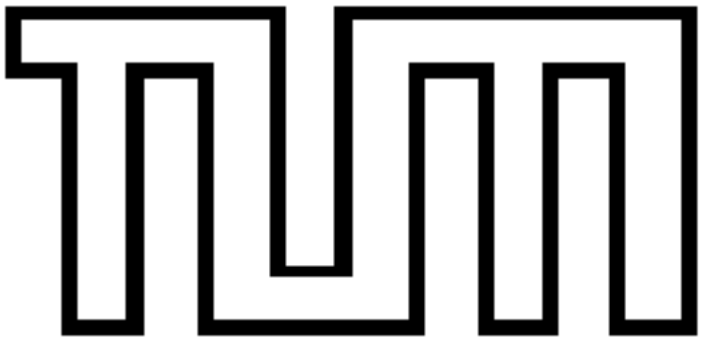

Technische Universität München

Interdisziplinäres Projekt

Entwurf eines Interface zwischen MATLAB und LAPACK

Bearbeiter: Goran Jovic, Thomas Woitsch

Aufgabensteller und Betreuer: Prof. Dr. Folkmar Bornemann

Inhaltsverzeichnis

1 Einleitung

Das Programmpaket MATLAB von der Firma „The Mathwork“ hat sich in den letzten Jahren in Wirtschaft und Lehre zu einer Standardplattform entwickelt, die numerische Berechnungen, erweiterte Grafik- und Visualisierungsmöglichkeiten und eine höhere Programmiersprache miteinander verbindet.
Das Projekt LAPACK (Linear Algebra Package) ist eine Bibliothek von Fortran77 Routinen, das mit Hilfe von BLAS1, BLAS2 und BLAS3 (Basic Linear Algebra Subprograms) viele Probleme der linearen Alegbra auf verschiedenen Plattformen effizient löst.
Nun bietet MATLAB eine Standardschnittstelle (API) für C-Programme an, sogenannte Mex-Files. Über diese liese sich die Funktionalität von LAPACK mit optimiertem BLAS, etwa aus der SUN Performance Library, für MATLAB nutzbar machen.
Eine solche Schnittstelle zwischen MATLAB und LAPACK soll in diesem Interdisziplinären Projekt entworfen und für die wichtigsten Routinen realisiert werden.

2 Aufgabenstellung

Wie in der Einleitung erwähnt soll in diesem Projekt eine Schnittstelle zwischen MATLAB und LAPACK entwickelt und für einige wichtige Funktionen implementiert werden. Die Schnittstelle soll folgende Eingenschaften haben:

- als Schnittstelle zwischen MATLAB und LAPACK dient das C-API von MATLAB (mex-Files)
- es wird das LAPACK mit optimiertem BLAS aus der SUN Performance Library verwendet
- Verstecken aller den Benutzer von übergeordneter Ebene nicht interessierender Details, etwa Workspace-Berechnungen und Eingabe der Matrixdimensionen
- an die Matlab-Syntax und -Philosophie angepasste Aufrufe
- Benutzerhilfe für die implementierten LAPACK Funktionen

Mit dieser Schnittstelle soll es also möglich sein, LAPACK Routinen aus MATLAB heraus aufrufen zu können, und somit die Vorteile von MATLAB und LAPACK zu vereinen.

3 Realisierung

In diesem Abschnitt wird auf die Entwicklung und Realisierung des Projekts im Detail eingegangen.

3.1 C-API von MATLAB (mex-Files)

Als Programmierschnittstelle zwischen MATLAB und LAPACK dient das C-API von MATLAB, die sogenannten mex-Files. Mit Hilfe der mex-Files können eigene C-Routinen aus MATLAB heraus aufgerufen werden, als wären sie schon direkt in MATLAB integriert. Mex-Files sind dynamisch gelinkte Routinen, die der MATLAB Interpreter automatisch laden und ausführen kann. Unter Solaris kennzeichnet die Endung *.mexsol* ein mex-File, während die Benutzerhilfe für ein solches File in einem m-File abgelegt wird. In MATLAB werden alle mex-Files wie gewöhnliche m-Funktionen aufgerufen, also mit dem Namen der Datei.
Ein mex-File hat folgenden Aufbau:

```
#include "mex.h"

void mexFunction(
    int nlhs, mxArray *plhs_m[],
    int nrhs, const mxArray *prhs_m[])
{

   /* C code ... */

}
```

Der Eintrittspunkt jedes mex-Files ist die *mexFunction*. Für jeden Aufruf enthält *nlhs* die Anzahl der Ausgabeparameter und *nrhs* die Anzahl der Eingabeparameter. Alle MATLAB Variablen, wie Skalare, Vektoren, Matrizen, Strings, usw., werden als MATLAB Arrays abgespeichert. In C wird das MATLAB Array als Datentyp *mxArray* deklariert. Das *mxArray* enthält unter anderem Informationen über Typ und Dimension der gespeicherten Variablen. Über die Paramter *plhs* und *prhs* werden die eigentlichen Aus- und Eingabeparameter zwischen MATLAB und *mexFunction* ausgetauscht.
Die Schnittstelle zwischen MATLAB und LAPACK ist so realisiert, dass für jede zu implementierende LAPACK Routine ein gleichnamiges mex-File existiert. In diesem mex-File wird in der *mexFunction* der entsprechende Aufruf der LAPACK Routine aus der SUN Performance Library durchgeführt. Da in MATLAB die Ein- und Ausgabeparameter der *mexFunction mxArrays* sind, und die SUN Performance Library diesen Datentyp nicht unterstützt, muss vor und nach jedem Aufruf einer LAPACK Routine eine Datentypkonvertierung erfolgen. Die Benutzerhilfe für jede implementierte LAPACK Routine befindet sich in einem gleichnamigen m-File.

3.2 Compilieren von mex-Files

Die in C implementierten mex-Files müssen vor einem Aufruf in MATLAB mittels *mex «programmname».c* compiliert und gelinkt werden. Da der Befehl *mex* eine Konfigurationsdatei *mexopts.sh* benötigt, muß zunächst mittels *mex -setup* eine solche Datei erzeugt werden. *mexopts.sh* enthält Informationen darüber, welcher Compiler und Linker mit welchen Optionen genutzt und welche Bibliotheken gelinkt werden sollen.
Der folgende Ausschnitt zeigt den Teil der *mexopts.sh* Datei, der für dieses Projekt eingesetzt wurde:

```
#----------------------------------------------------------
        sol2)   # gcc version 2.7.2.f.1
#----------------------------------------------------------
    CC='gcc'
    CFLAGS="-fPIC -I /opt/SUNWspro/SC4.2/include/cc"
    CLIBS="-L$GCC_LIBDIR -lgcc -lsunperf -lSUNWPro_lic -lF77
           -lM77 -lsunmath -lm"
    COPTIMFLAGS='-O -DNDEBUG'
    CDEBUGFLAGS='-g'
#
    FC='f77'
    FFLAGS='-G'
    FLIBS=''
    FOPTIMFLAGS='-O'
    FDEBUGFLAGS='-g'
#
    LD='/usr/ccs/bin/ld'
    LDFLAGS="-G -M $MATLAB/extern/lib/sol2/$MAPFILE"
    LDOPTIMFLAGS=''
    LDDEBUGFLAGS=''
#----------------------------------------------------------
```

Wie zu sehen ist, wird *gcc* als Compiler und *ld* als Linker eingesetzt. Es müssen die Bibliotheken *gcc, sunperf, SUNWPro_lic, F77, sunmath* und *m* gelinkt werden.
Nach dem Compilieren und Linken mittels *mex* muss das erzeugte mex-File noch mit der SUN Performance Library lizensiert werden. Dies geschieht mit dem Befehl */opt/SUNWspro/SC4.2/bin/lic_lib_auth «programmname».mexsol sunperf.*

3.3 Automatisierung

In diesem Projekt sollen die Schnittstellen für die LAPACK Funktionen mit den Datentypen DOUBLE und DOUBLE COMPLEX mit folgenden Matrixtypen implementiert werden:

- GE general
- GT general tridiagonal
- HE Hermitian
- PO symmetric or Hermitian positive definit
- PT symmetric or Hermitian positive definit tridiagonal
- ST symmetric tridiagonal
- SY symmetric

Da es sich dabei um ca. 140 Routinen handelt und die Implementierung jeder einzelnen Schnittstelle per Hand ziemlich langwierig und fehleranfällig ist, bietet sich an, diesen Prozess teilweise zu automatisieren.
Um eine Automatisierung einführen zu können, müssen alle benötigten Informationen einer LAPACK Routine in einheitlicher und digitaler Form vorliegen. Diese Eigenschaft besitzen die unformatierten Manpages.
Aus diesem Grund wird in diesem Projekt ein Generator erstellt, der eine semiautomatische Generierung der mex-Files für jede einzelne LAPACK Routine mit Hilfe der Manpage (FORTRAN77 Syntax) durchführt. Die erzeugten mex-Files müssen eventuell nach der Generierung per Hand nachbearbeitet werden. Ebenso wie bei den mex-Files werden auch die m-Files, die die Benutzerhilfe enthalten, mit einem seperaten Generator semiautomatisch erzeugt.

3.3.1 Generierung der mex-Files

Eine unformatierte Manpage hat, z.B. hier für *dgebak*, folgenden Aufbau:

```
.SH NAME
dgebak - form the right or left eigenvectors of a real general
matrix by backward transformation on the computed
eigenvectors of the balanced matrix output by DGEBAL
.SH SYNOPSIS
SUBROUTINE DGEBAK(JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV, INFO )
...
.SH PURPOSE
DGEBAK forms the right or left eigenvectors of a real general
matrix by backward transformation on the computed
eigenvectors of the balanced matrix output by DGEBAL.
.SH ARGUMENTS
JOB     (input) CHARACTER*1
```

```
...
SIDE    (input) CHARACTER*1
...
N       (input) INTEGER
...
ILO     (input) INTEGER
...
IHI     (input) INTEGER
...
SCALE   (input) DOUBLE PRECISION array, dimension (N)
...
M       (input) INTEGER
...
V       (input/output) DOUBLE PRECISION array, dimension (LDV,M)
...
LDV     (input) INTEGER
...
INFO    (output) INTEGER
...
```

Alle im Projekt verwendeten unformatierten Manpages befinden sich im Verzeichnis */opt/SUNWspro/man/man3p* und haben die Endung *.3p*.

Der in diesem Projekt entwickelte Generator *make_mex* läuft in folgenden Schritten ab:

1. Dem Programm wird die Manpage der zu generierenden LAPACK Routine als Dateiname übergeben. Die Datei wird dann für das spätere Parsen vollständig in den Speicher geladen.

2. Anschließend wird ein mex-File mit dem entsprechendem Namen der LAPACK Routine im aktuellen Verzeichnis erzeugt und das Grundgerüst eines mex-Files (siehe 3.1) hineingeschrieben.

3. Aus der Datei *sunperf.h*, die sich im gleichen Verzeichnis wie *make_mex* befinden muß werden anhand der C Aufrufsyntax der LAPACK Routine die benötigten Variablen im mex-File deklariert.

4. Im folgenden Schritt wird die im Speicher liegende Manpage geparst. Dabei wird zuerst nach *.SH SYNOPSIS* gesucht, um aus der darauffolgenden FORTRAN77 Syntaxbeschreibung die einzelnen Variablen auszulesen. Ab *.SH ARGUMENTS* wird jede einzelne dieser Variablen abgearbeitet. Man unterscheidet die Variablen nach Eingabetyp (input, output, input/ouput, input or output, workspace) und nach Datentyp (double, complex, character, integer, arrays, functions). Je nach Eingabe- und Datentyp wird im mex-File ein entsprechender C-Code für die Datentypkonvertierung zwischen MATLABs *mxArray* Datentyp und den in der SUN Performance Library verwendeten C Datentypen eingetragen. Falls es sich bei den

Datentyp um ein Array handelt, wird die Dimension des Arrays mitausgelesen. Je nach Eingabetyp werden Variablen vor dem MATLAB Benutzer versteckt.
Folgende Auflistung zeigt, welcher C-Code für welchen FORTRAN77 Datentypen während des Parsens der Manpage in das mex-File geschrieben wird. *in* ist ein Zähler für die Ein- und *out* für die Ausgabeparamenter. x ist der Name des Parameters. Die Dimension einer Matrix steht in *dim*, die Anzahl der Spalten in *dim1* und die Anzahl der Zeilen in *dim2*.

- (input) CHARACTER*1, CHARACTER :

  ```
  x=mxArrayToString(prhs_m[in])[0];
  ```

- (input) INTEGER :

  ```
  x=(int)mxGetScalar(prhs_m[in]);
  ```

  Falls es sich bei dem Parameter x um eine Variable handelt, die die Anzahl der Spalten oder Zeilen einer Matrix oder die Leading Dimension eines Arrays angibt, wird diese Variable vor dem MATLAB Benutzer versteckt, d.h. dieser Parameter erscheint nicht in der Eingabeparameterliste der mex-Funktion.
  In diesem Fall muß mittels der mx-Befehle *mxGetM* oder *mxGetN* die Dimension in die Variable x eingelesen werden. Wenn es sich bei x um die Leading Dimension eines Arrays handelt, wird x einem vom Benutzer eingegebener Wert zugewiesen.

- (input) DOUBLE PRECISION :

  ```
  x=mxGetScalar(prhs_m[in]);
  ```

- (input) INTEGER array :

  ```
  x=mxGetData(prhs_m[in]);
  ```

- (input) DOUBLE PRECISION array :

  ```
  x=mxGetPr(prhs_m[in]);
  ```

- (input) LOGICAL FUNCTION :

  ```
  select_name=malloc(256*sizeof(char));
  select_name=mxArrayToString(prhs_m[in]);
  ```

  select_name enthält den Namen einer MATLAB Funktion, die dann über eine Dummy-Funktion im mex-File mittels *mexCallMATLAB* aufgerufen werden kann (siehe z. B. dgees.c) . Der LAPACK Routine wird dann als Parameter die Dummy-Funktion übergeben.
  Diese Dummy-Funktion wird nicht vom Generator erzeugt und muß manuell eingefügt werden.

- (input) COMPLEX*16 array :

  ```
  x=malloc(mxGetM(prhs_m[in])*mxGetN(prhs_m[in])*
    sizeof(doublecomplex));
  ```

```
for (i=0; i<mxGetM(prhs_m[in])*mxGetN(prhs_m[in]); i++)
{
  x[i].r=mxGetPr(prhs_m[in])[i];
  x[i].i=mxGetPi(prhs_m[in])[i];
}
```

- (input/output) INTEGER [array]:

```
plhs_m[out]=mxDuplicateArray(prhs_m[in]);
x=mxGetData(plhs_m[out]);
```

- (input/output) DOUBLE PRECISION [array] :

```
plhs_m[out]=mxDuplicateArray(prhs_m[in]);
x=mxGetPr(plhs_m[out]);
```

- (input/output) COMPLEX*16 [array] :

```
plhs_m[out]=mxDuplicateArray(prhs_m[in]);
x=malloc(mxGetM(prhs_m[in])*mxGetN(prhs_m[in])*
  sizeof(doublecomplex));
for (i=0; i<mxGetM(prhs_m[in])*mxGetN(prhs_m[in]); i++)
{
  x[i].r=mxGetPr(prhs_m[in])[i];
  x[i].i=mxGetPi(prhs_m[in])[i];
}
```

  Nach dem Aufruf der LAPACK Routine muß *plhs_m[out]* mit den Werten aus x gefüllt werden. Dies geschieht folgendermaßen:

```
for (i=0; i<mxGetM(plhs_m[out])*mxGetN(plhs_m[out]); i++)
{
  mxGetPr(prhs_m[in])[i]=x[i].r;
  mxGetPi(prhs_m[in])[i]=x[i].i;
}
```

- (output) INTEGER :

```
x=malloc(sizeof(int));
x[0]=1;
plhs_m[out]=mxCreateNumericArray(1,x,mxINT32_CLASS,mxREAL);
x=mxGetData(plhs_m[out]);
```

- (output) DOUBLE PRECISION :

```
plhs_m[out]=mxCreateDoubleMatrix(1,1,mxREAL);
x=mxGetPr(plhs_m[out]);
```

- (output) INTEGER array:

```
x=malloc(dim*sizeof(int));
x[0]=dim1;
x[1]=dim2;
plhs_m[out]=mxCreateNumericArray(1,x,mxINT32_CLASS,mxREAL);
```

```
free(x);
x=mxGetData(plhs_m[out]);
```

- (output) DOUBLE PRECISION array:

  ```
  plhs_m[out]=mxCreateDoubleMatrix(dim1,dim2,mxREAL);
  x=mxGetPr(plhs_m[out]);
  ```

- (output) COMPLEX*16 array:

  ```
  plhs_m[out]=mxCreateDoubleMatrix(dim1,dim2,mxCOMPLEX);
  x=malloc(mxGetM(plhs_m[out])*mxGetN(plhs_m[out])*
    sizeof(doublecomplex));
  ```

  Nach dem Aufruf der LAPACK Routine muß *plhs_m[out]* mit den Werten aus x gefüllt werden. Dies geschieht folgendermaßen:

  ```
  for (i=0; i<mxGetM(plhs_m[out])*mxGetN(plhs_m[out]); i++)
  {
    mxGetPr(prhs_m[in])[i]=x[i].r;
    mxGetPi(prhs_m[in])[i]=x[i].i;
  }
  ```

- (input or output) [INTEGER] [DOUBLE PRECISION] [COMPLEX*16] [array]
  Bei diesen Datentypen handelt es sich, je nach einer Bedingung, entweder um einen Eingabe- oder Ausgabetyp.
  Der Generator erzeugt in diesem Fall immer einen Eingabetyp, so daß das mex-File manuell nachbearbeitet werden muß, um die nötige Fallunterscheidung einzufügen.

- (workspace[/output]) INTEGER

  ```
  iwork=malloc((liwork)*sizeof(int));
  allocate_int_workspace(iwork,liwork);
  ```

  Alle Workspacetypen werden eigentlich in der SUN Performace Library automatisch allokiert. Aus diesem Grund besitzen alle LAPACK Routinen aus dieser Library im Gegensatz zu den FORTRAN77 Routinen keine Workspace-Argumente. Lediglich die INTEGER Workspaces müssen manuell im mex-File allokiert werden, da sonst MATLAB aufgrund eines Speicherschutzfehlers abstürzt.
  Wurden alle Datentypen geparst, wird der LAPACK Aufruf und nach ihm alle nötigen Speicherfreigabebefehle in das mex-File geschrieben.

- Falls eine manuelle Nachbearbeitung des mex-Files nötig ist, wird eine entsprechende Textmeldung in die Datei geschrieben, so daß diese nicht fehlerfrei kompiliert werden kann.
  Zum Schluß wird das mex-file geschlossen und der Generator beendet.

  WICHTIG: Es können nur Manpages für die Generierung von mex-Files benutzt werden, die die in Kapitel 3.3.1 aufgezeigte Struktur aufweisen.

Auf die Bedienung von *make_mex* während dem Parsen wird in Kapitel 4 näher eingegangen.

3.3.2 Generierung der m-Files

Der Ablauf des Generators *make_m* für die Benutzerhilfe hat folgendes Aussehen:

1. Laden der Manpage der ausgewählten LAPACK Routine in den Speicher. Der Name der zu bearbeitenden Manpage wird beim Aufruf des Generators übergeben.

2. Erzeugen des m-files mit dem Namen der LAPACK Routine in dem angegebenen Verzeichnis, dessen Pfad ebenfalls beim Aufruf des Generators übergeben wird

3. den erklärenden Text zur LAPACK Routine, der zwischen *.SH PURPOSE* und *.SH ARGUMENTS* steht (s.o.), in das m-File schreiben. Dabei werden in diesem Schritt und in den folgenden alle Manpage bedingten Formatierungszeichen entfernt.

4. ab *.SH SYNOPSIS* werden die Funktionsparameter ausgelesen und der Ein/Ausgabetyp, mit Hilfe der Parameterbeschreibung ab *.SH ARGUMENTS*, bestimmt. Mit diesen Informationen werden dann die Funktionsparameter mit dem Funktionsnamen zu einem MATLAB Funktionsaufruf zusammengesetzt und in das m-File geschrieben. Während des Konkatenierens wird darauf geachtet, daß versteckte Paramter im Funktionsaufruf nicht in Erscheinung treten.

5. zum Schluß werden die Parameterbeschreibungen ab *.SH ARGUMENTS* in das m-File kopiert. Dabei werden die FORTRAN77 Datentypbeschreibungen auf MATLAB Datentypbeschreibungen umgewandelt. Die Parameterbeschreibungen von versteckten Parametern werden dabei nicht in das m-File eingetragen.

Die Generatorbedienung wird Kapitel 4 näher erklärt.
WICHTIG: Es können nur Manpages für die Generierung von m-Files benutzt werden, die die in Kapitel 3.3.1 aufgezeigte Struktur aufweisen.

3.4 Bedienung der Generatoren

Wie im Kapitel 3.3 erwähnt, werden die mex-Files und die m-Files semiautomatisch erzeugt, d.h. es ist eine Steuerung während des Generierens und eventuell eine Nachbearbeitung der erzeugten Dateien von Nöten. Die Vorgehensweise wird in diesem Kapitel genauer erklärt.

3.4.1 Bedienung von *make_mex*

Mit *make_mex dateiname* wird der Generator mit der zu bearbeitenden Manpage aufgerufen.
Es erscheint die folgende Meldung (hier z.B. für die Routine *dgebak*):

```
++++++++++ PERFORMING dgebak ++++++++++
```

Datentypen, die automatisch geparst werden können, werden in einer Zeile ausgegeben. Falls ein Argument nicht automatisch erzeugt werden kann, wird der Teil der Manpage für dieses Argument ausgegeben und eine entsprechende Eingabeaufforderung angezeigt, so daß der Benutzer die entsprechende Information selbst eingeben muß. Dabei wird auf Groß- oder Kleinschreibung nicht geachtet. Die Manpageausgaben müssen genau gelesen werden, da eventuell Fallunterscheidungen vorkommen können, die später maunell im mex-File implementiert werden müssen.
Im Kapitel 3.3.1 sind alle vom Generator unterstützten Datentypen und der für das mex-File entsprechende C-Code zu finden, so daß eine Nachbearbeitung ohne größere Schwierigkeiten erfolgen kann. Bei folgenden Fällen muß nachgearbeitet werden:

1. *(input or output)* Datentypen:
   Kommt ein solcher Datentyp vor, so erzeugt der Generator automatisch einen *(input)* Datentyp. Somit muß das erzeugte mex-File mit einer entsprechenden IF-Anweisung ergänzt werden (Bedingung für die IF-Anweisung siehe Manpage der LAPACK Routine). Als Beispiel siehe Routine *dgesvx*.

2. *LOGICAL FUNCTION* Datentypen:
   Hier muß eine Dummyfunktion in das mex-File eingetragen werden, über die ein m-File in MATLAB aufgerufen wird. Als Implementierungsbeispiel einer solchen Dummyfunktion siehe *dgees.c*.

3. Parameter, deren Wert von einer Bedingung abhängt:
   Analog zu einem *(input or output)* Datentypen muß hier ebenfalls eine IF-Anweisung nachträglich implementiert werden. Dies liegt daran, daß während des Parsens einem solchen Parameter nur ein bestimmter Wert zugewiesen werden kann. Als Beispiel siehe Parameter *LDVL* der Routine *dgeev*.

Falls das mex-File erfolgreich generiert werden konnte, wird die folgende Meldung angezeigt:

```
++++++++++ SUCCESSFULLY DONE ++++++++++
```

Falls diese Meldung nach der Beendigung des Generators nicht erscheint, ist ein Fehler während des Parsens aufgetreten. Der Grund hierfür ist ein abweichendes Format der Manpage, welches der Generator nicht unterstützt.
Das fertig generierte mex-File sollte nochmals mit den Daten der Manpage aus Sicherheitsgründen verifiziert werden. Nach einer eventuellen Nachbearbeitung kann das mex-File kompiliert werden (siehe Kapitel 3.2).

3.4.2 Bedienung von *make_m*

Es gibt zwei Arten des Aufrufs um m-Files aus LAPACK Manpages zu erzeugen:

1. Der Aufruf von *make_m «manpage» .3p «verzeichnis»* .
   Es wird direkt das Programm *make_m* gestartet mit der Übergabe des Namens der LAPACK Manpage aus der heraus die Benutzerhilfe generiert wird und dem Verzeichnis wo das m-File anzulegen ist. Der Generator bestätigt durch Ausgabe von *Make «routinenname».m .....*, daß die Bearbeitung des m-Files eingeleitet wird. Wurde das m-File erfolgreich generiert, wird dies durch *done.* angezeigt.
   Gibt der Generator den Hinweis *New kind of parameter: ...* aus, so hat er einen Ein/Ausgabetyp nicht erkannt, generiert aber trotzdem zu Ende. Man sollte dieses m-File anschließend genau kontrollieren. Im Fehlerfall wird mit *«routinenname».m was not done.* dem Nutzer mitgeteilt, daß das m-File nicht generiert werden kann und der Vorgang deswegen abgebrochen wird. Der Grund liegt meist am abweichenden Format der LAPACK Routinen Manpage. Ein spezieller Fehler am Format der Manpage wird erkannt und mit der Meldung *ERROR: In «routinenname».m the I/O-Parameters can't be determined!!* angezeigt. Dabei handelt es sich um eine Manpage bei der die Ein/Ausgabetypen der Parameter nicht angegeben werden.
   Kann ein m-File aus oben genannten Gründen nicht erstellt werden, wird der Inhalt der Manpage, von Formatierungszeichen befreit, einfach kopiert und im Zielverzeichnis in «routinenname».m* abgelegt. So wird einem die Möglichkeit gegeben selbst das m-File zu konstruieren.

2. Der Aufruf des Scripts *gen_mfiles* ist vorteilhaft bei der Generierung mehrerer m-Files.
   Nach der Eingabe von *gen_mfiles*, wird man zunächst aufgefordert das Quellverzeichnis anzugeben, in dem die LAPACK Manpages stehen. Wurde diese Frage beantwortet, wird man aufgefordert das Zielverzeichnis anzugeben, wo die generierten m-Files liegen sollen. Die Informationen bei der Generierung werden für jedes m-File in der gleichen Struktur und Bedeutung angezeigt.

Es ist zu beachten, daß mit dem Generator nur die Grobstruktur des m-Files erstellt wird. Veränderungen an Texten, die den Inhalt betreffen, werden nicht durchgeführt. Dies betrifft vorallem die Beschreibung derjenigen Parameter, die als Eingabe und zugleich als Ausgabe dienen. In der LAPACK Routine in FORTRAN77 werden meist die Eingabedaten mit den Ausgabedaten überschrieben, was aber im Funktionsaufruf über MATLAB nicht mehr vorkommt. Z.B.:
Orginal Parameterbeschreibung in der Manpage:

```
.SH ARGUMENTS
...

V       (input/output) DOUBLE PRECISION array, dimension (LDV,M)
```

```
            On  entry,  the matrix of right or left eigenvectors
            to be transformed, as returned by DHSEIN or  DTREVC.
            On  exit, V is overwritten by the transformed eigen-
            vectors.
...
```

Parameterbschreibung in der MATLAB Benutzerhilfe wie sie nach der händischen Manipulation aussehen könnte:

```
%ARGUMENTS
...

%V/OV       (input/output) DOUBLE MATRIX
%V is the matrix of right or left eigenvectors to be
%transformed, as returned by DHSEIN or DTREVC.
%OV contains the transformed eigenvectors.

...
```

Das oben beschriebene Beispiel ist auch meist der Hauptgrund, das generierte m-File nochmals nachzubearbeiten.

4 Verzeichnisstruktur

Im folgenden werden die Verzeichnisse angegeben, in denen sich die Dateien des Projekts befinden.

- m- und mex-Files: */usr/applic/packages/lapackmex/lib*
  Die Umgebungsvariable MATLABPATH zeigt auf das obige Verzeichnis. Dadurch werden die Routinen von MATLAB automatisch erkannt.
- Quelldateien der m- und mex-Files: */home/bornemann/m3/appdir/lapackmex/mexsrc*
- make_mex: */home/bornemann/m3/appdir/lapackmex/make_mex*
- make_m: */home/bornemann/m3/appdir/lapackmex/make_m*
- Dokumentation: */home/bornemann/m3/appdir/lapackmex/doc*