SAL

SAL

Overview

This guide contains tips and pointers for compiling TOMLAB using MathWorks' Matlab Compiler. Observe that compilation is not possible with a demo license or standard license. A special standalone license is needed.

A few modifications may be needed to TOMLAB before it compiles well with MathWorks' MCC. Traditionally, customers have been compiling QP solver solutions like TOMLAB /Xpress, where only a few m-files are used. Nevertheless, TOMLAB in its entirety will compile fine with a few adjustments.

Filenames

MCC 3.0 can experience problems when m-files with mixed-case names are called through *feval*. Therefore it is recommended to write all routines using lowercase filenames only (or patch MCC 3.0, see Section 7). If a program compiles fine but complains about something similar to:

```
ERROR: Reference to unknown function 'qp_f' from nargin in stand-alone mode.
```

the following actions will help:

- Make sure that the all such files are named in the call to MCC.
- Make sure that all references to them are correct. Use a search utility to locate these filenames in the m-files used.

Using the profiler

The Matlab profiler is a very useful tool when preparing for compilation. It should be used to check which files are being used by your application.

```
profile on
...
...
profile report
```

Making a minimal copy

When an application runs fine in Matlab mode and one wants to compile it, it is highly recommended to make a minimal TOMLAB copy in which only the m-files and dll's being called are included. The profiler will show which files to add.

The files should be copied to a single directory, and the application checked so that it runs without *tomlab/startup.m* being executed (make sure that the Matlab paths for TOMLAB are not cached). Do not forget the standalone tomlablic.dll and any other dll's. Continue to fill this directory with TOMLAB m-files and dll's until the standalone application is working.

One can then compile, naming all the dll's the program is using, as well as the m-files called through *feval*. It is usually a bit of a trial-and-error process before everything is running smoothly.

The main advantage with creating a minimal TOMLAB is that the final executable size is brought down. For example, on one user case a 4MB executable was reduced to about 700kB when only the essential files were compiled.

It is recommended not to use the universal driver routine *tomRun* (or the optimization toolbox interface) when compiling as it references many unnecessary files. Replace the following code:

```
Prob = conAssign( ... );
...
...
Result = tomRun('snopt', Prob, 0);
```

with:

```
Prob = conAssign( ... );
...
...
Prob = ProbCheck(Prob, 'snopt'); Result = snoptTL(Prob);
```

A Simple Example

To compile the following code, in *test.m*.

```
% ------------------------------------------------------------------
function f = test

probfile = 'con_prob';
probNumber = 10;
ask = [];
Prob = [];

Prob = probInit(probfile, probNumber, ask, Prob);
Result = conSolve(Prob);
PrintResult(Result);

f = Result.f_k;
% ------------------------------------------------------------------
```

with a TOMLAB /SOL license (which means that QPOPT is chosen as the QP subsolver in conSolve), the profiler and a little trial-and-error will show that the following m-files are needed:

BoundInit.m	LineSearch.m	SOLGet.m
ComputeQR.m	LinearConstr.m	SOLSet.m
CreateProbQP.m	NonlinConstr.m	StateDef.m
DefPar.m	PrintResult.m	checkType.m
GetSolver.m	ProbCheck.m	conProbSet.m
LagMult.m	ProbDef.m	conSolve.m
LineParamDef.m	ProbGate.m	conVarDef.m
LineParamSet.m	ResultDef.m	con_H.m
con_c.m	defblbu.m	nlp_dc.m
con_d2c.m	endSolve.m	nlp_f.m
con_dc.m	globalGet.m	nlp_g.m

con_f.m	globalSave.m	optParamDef.m
con_fm.m	iniSolve.m	optParamSet.m
con_g.m	mFiles.m	probInit.m
con_gm.m	nlp_H.m	qpProbSet.m
con_prob.m	nlp_c.m	qpVarDef.m
qp_Hess.m	test.m	
qp_f.m	tomSolve.m	
qp_fX.m	tomlabVersion.m	
qp_g.m	xnargin.m	
qp_gX.m	xprint.m	
qp_H.m	xprinte.m	
qp_prob.m	xxx_prob.m	
qpoptTL.m		

and these dll's:

qpopt.dll tomlablic.dll tomsol.dll

The compile command to execute is:

```
mcc -m test.m qpopt.dll tomsol.dll tomlablic.dll ...
:con_prob con_f con_g con_H con_c con_dc con_d2c qp_f qp_g
:qp_H
```

Notice how a number of m-files must be named here. If not, upon execution there will be errors like:

```
ERROR: Reference to unknown function 'qp_f' from nargin in stand-alone mode.
```

Still, there may be warnings about missing functions, however this is not a problem if one has checked which m-files are needed for the application to work. An alternative is to compile all TOMLAB m-files, but thereby increasing both size and compile time.

The example should run fine:

```
>> ! test.exe

===== * * * ========================================================================= * * *
TOMLAB SOL - stand-alone test license
=============================================================================================
Problem: con_prob - 10: Schittkowski 66. Eckhardt f_k          0.518163274278335420
                                     User given f(x_*)         0.518163274099999960
                                     sum(\|constr|)            0.000000000003534950
                                     f(x_k) + sum(|constr|)    0.518163274281870370
                                     f(x_0)                    0.579999999999999960



Solver: conSolve. EXIT=0. INFORM=1.
Schittkowski SQP algorithm, analytic Hessian
Optimal solution found
```

```
Iteration points are close

FuncEv 66 GradEv 66 ConstrEv 66 Iter 43
CPU time: 0.375000 sec. Elapsed time: 0.375000 sec.
Starting vector x:
x_0: 0.000000 1.050000 2.900000
Optimal vector x:
x_k: 0.184116 1.202155 3.327280
User given stationary point x_* (1): - Minimum point
x_*: 0.184126 1.202168 3.327322
Lagrange multipliers v. Vector length 5:
v_k: 0.000000e+000 0.000000e+000 0.000000e+000 6.654713e-001 2.000045e-001
Gradient g_k:
g_k: -8.000000e-001 0.000000e+000 2.000000e-001
cErr -6.852297e-013 -2.849720e-012
Projected gradient gPr:
gPr: -1.006538e-006 -1.210015e-006 -4.026060e-006
   NonLLow 1 NonLLow 2
v: 6.654706e-001 2.000040e-001

=== * * * ================================================= * * *
```

6 Potential problems

A few problems may occur when compiling TOMLAB. For example TOMLAB /MAD cannot be compiled and needs to be replaced with dummy files.

6.1 P-code

It is not possible to compile p-coded files if they are present.

The following messages may appear when the example above is compiled:

```
error LNK2001: unresolved external symbol _mlfNGetinternalderivs.
error LNK2001: unresolved external symbol _mlfNGetvalue.
error LNK2001: unresolved external symbol _mlfNGetderivs.
```

The problem is resolved by copying the files in *tomlab/mad/@dummy* to *tomlab/mad* and removing everything else in the latter.

6.2 Mex files

If using a more recent version of the Matlab Compiler the following error or similar may occur:

```
"C:\PROGRAM FILES\MATLAB\R2006A\BIN\MEX.PL: Error: mbuild cannot
link to 'npsol.dll' directly. Instead, you must link to the file
'npsol.lib' which corresponds to 'npsol.dll'."
```

The problem is resolved by renaming files with extension *dll* to *mexw32*. It is also possible to update to the most recent version of TOMLAB if using version Matlab R2007a or above.

The following error may also be displayed:

```
ERROR: Function 'help' is not implemented in standalone mode.

EXITING
```

This means that a TOMLAB dll (mex) used by application has not been named in the mcc call.

6.3 startup.m

startup.m may be executed when running mcc if still in path. If so rename the file to *startuptomlab.m.*

6.4 Shared dlls

The shared dlls included with TOMLAB are located in *tomlab/shared*. The location of this folder needs to be in the environment variable `PATH` for deployed units.

Please contact support@tomopt.com [1] for all other problems.

7 Calling TOMLAB from C/C++

This section describes how to create a standalone shared library with TOMLAB functionality using the MCC compiler.

The guide covers MCC 3 as well as MCC 4. If using MCC 3, make sure to patch it to R13SP1.

7.1 Creating a stand alone shared library

The procedure to create a standalone shared library of TOMLAB files is identical to creating a standalone application, except for the commands to compile and link the files. As described earlier, copy all needed m-files and mex-files to a single directory. If needed, create new m-files as an interface between TOMLAB and C/C++. These can be used as entry points to the shared library (although, all m-files can be used as entry points).

A suitable entry point could be an m-file taking F, c, A, b_L, b_U, x_L and x_U matrices as input, generating a QP Prob struct out of this, and calling a suitable solver. Only the results of interest, for example the optimal vector x and objective function value, could be returned.

After all MATLAB files have been put in a single directory, compile them into a shared library using the following command:

```
> mcc -B csharedlib:<lib-name> <m-files> <mex-files>
```

This creates a shared library: `<lib-name>` with extension depending on platform (.dll on Windows, .so on Linux). If the client to use the shared library is written in C++ one can build a C++ shared library supporting exception handling and C++ classes for handling the MATLAB arrays. The command to compile a C++ shared library is:

```
> mcc -B cpplib:<library name> <m-files> <mex-files>
```

For more detailed information about this, refer to: http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/ [2]

7.2 Calling the shared library from C/C++

The new shared library now contains entry points for all files compiled with MCC; mex-files as well as m-files. Each m-file has two library functions that may call it:

```
mlxFunctionname
mlfFunctionname
```

Notice that the first character in the function name is always upper case. The rest of the characters are always lower case, regardless of the case of the original function.

The difference between the two functions prefixed with mlx and mlf is the way the arguments are passed. The mlx-function expects the arguments like this:

```
extern void mlxSolveqp(int nlhs, mxArray \*plhs\[\], int nrhs, mxArray \*prhs\[\]);
```

just like a mex-file (Refer to the MATLAB documentation for more details and information on this.)

The mlf-function is perhaps more user friendly, as it looks more like the original function call in MATLAB. Observe that the arguments to the mlf-functions differ between MCC 3 and MCC 4. The MATLAB function:

```
function \[x, f\] = solveqp(F, c, A, b_L, b_U, x_L, x_U)
```

is declared in the standalone library as follows when using MCC 4:

```
extern void mlfSolveqp(int nargout, mxArray\*\* x, mxArray\*\* f, mxArray\* F
                       , mxArray\* c, mxArray\* A, mxArray\* b_L
                       , mxArray\* b_U, mxArray\* x_L, mxArray\* x_U);
```

A MATLAB function without left hand side arguments do not have the nargout and pointer to pointer arguments. The same MATLAB function is declared like this when using MCC 3:

```
extern mxArray \* mlfSolveqp(mxArray \* \* f,
                             mxArray \* F,
                             mxArray \* c,
                             mxArray \* A,
                             mxArray \* b_L,
                             mxArray \* b_U,
                             mxArray \* x_L,
                             mxArray \* x_U);
```

The return value is the first left hand side parameter; the MATLAB x in this case.

In order to call these function in the shared library from a client application, there are some tasks that must be completed before anything else. First of all, one has to include a header file in the application generated by the MCC compiler when compiling the library. The name of the file is the name of the library plus the extension .h. Each time the shared library is recompiled a new header file will be generated. This header file includes all necessary function declarations needed when calling a shared library, setting up input data and reading output data. No other header files need to be included in order to use the shared library.

Before using any of the MATLAB API functions, a function `mclInitializeApplication()` has to be called. This only applies to MCC 4 though. In MCC 3 there is no `mclInitializeApplication()`-function.

Before calling any of the functions in the shared library, an initialization function has to be called. The name of this function is: `<lib-name>Initialize()`, where `<lib-name>` is the name of the library (including the `lib` prefix on Linux).

When these two (or one in MCC 3) functions have been called, all m- and mex-files are callable from within the application using the prefixes mlf or mlx. At this point the application could for example read measurement data from a device or read input data from a user interface, then formulate an optimization problem and call TOMLAB through the shared library. It is recommended to write m-files for compilation with the shared library to act as an interface between the application and TOMLAB as it is usually easier to handle MATLAB data in the MATLAB language than in C.

When all calls to the shared library are done, a termination function should be called: <lib-name>Terminate() where `<lib-name>` is the name of the library (including the lib prefix on Linux). When the MATLAB API functions are not needed anymore, call `mclTerminateApplication()`. This only applies to MCC 4. To compile the application, use the mbuild utility: On Windows:

```
> mbuild <source code files> <lib-name>.lib
```

where `<source code files>` are the source code files for the application, and `<lib- name>` is the name of the shared library generated by MCC earlier. On Linux:

```
> mbuild <source code files> <lib-name> -L. -I.
```

where `<source code files>` are the source code files for the application, and `<lib-name>` is the shared library generated by MCC earlier *without* the lib-prefix and *without* the extension.

7.3 TOMLAB /CPLEX QP solver example

The TOMLAB distribution includes an example running TOMLAB /CPLEX for a QP problem from a separate C application. It is available in the TOMLAB distribution in the /examples/sal/sallib directory.

In summary, the example was constructed like this:

1. An m-file called *solveqp.m* was created to take simple input in the form of matrices for F, c, A, b L, b U, x L and x U. This m-file creates a QP Prob structure from these matrices and runs the TOMLAB /CPLEX solver through *tomRun* and returns the optimal vector x and objective function value.
2. The profiler in MATLAB was turned on and the solveqp function was called with some QP data. When a solution was obtained, the profile report command displayed the m- and mex-files used during the execution.
3. The files were copied to a project directory for the new shared library.
4. In *xnargin.m*, some changes were made in order to have more descriptive error messages when failing to find a specific function. See *xnargin.m* for instructions on what to change.
5. The library was built using the MCC compiler. (See the make file for the example for details).
6. A simple client application was written creating input data for the *solveqp.m* function, calling it and writing the results to the screen. (See the Appendix A for example code).
7. The application was built using the mbuild command. (See the make file of the example for details).
8. The application was run, and there were some error messages reporting missing files.
9. The missing files were added to the command building the library, and the library was rebuilt.
10. Step 8 and 9 were repeated a few of times until everything was working smoothly.

Source files for the simple client application (*application.c*) and the interface m-file (*solveqp.m*) are available in Appendix A of this document.

8 Calling TOMLAB from Microsoft Excel

This section contains a simple description of how to call the TOMLAB solvers from Microsoft Excel using the Matlab Builder for Excel. Matlab Builder for Excel is an extension to the Matlab Compiler.

8.1 Creating a stand alone Excel Add-In

The first step is to make a selection of TOMLAB m-files and mex-files needed in the stand alone program and to create a suitable entry point. See the beginning of section 7.1. Do not forget to include tomlablic.dll. After all Matlab files have been put in a single directory, compile them into an Excel Add-In using the following command:

```
> mcc -B cexcel:<addin_name>,<addin_name>,<version_number> <m-file(s)>
```

where `<addin_name>` is the name of the Add-In to create, and `<version_number>` is the version number of the Add-In, for example: 1.0.

(Notice: This is tested using MCC 4 only.)

This creates several new files, most of which are only used temporarily during the compilation by the Matlab Compiler. The important files are:

```
<addin_name>_<version_number>.dll
<addin_name>.ctf
<addin_name>.bas
```

The `.bas`-file is just an example of Excel Macro code in Visual Basic. This should be imported to the Excel workbook from where TOMLAB is called, but it often has to be modified in order to be useable. It is good as a template to start from though.

The `.dll`- and `.ctf`-files are the needed runtime files. They have to be in the same directory as the Excel workbook when making the TOMLAB calls.

8.2 Calling the Add-In from Excel

To be able to call the Add-In, one must create a link between the Add-In and Excel. This is done through the Visual Basic Editor in Excel.

In Excel, start the Visual Basic Editor: `Tools -> Macro -> Visual Basic Editor`

Then, in the Visual Basic Editor, import the .bas-file created during the compilation by Matlab Builder for Excel: `File -> Import File...`

All m-files mentioned in the compilation command corresponds to one Visual Basic function. A function can be called from an Excel worksheet by writing something like this in a cell:

```
=function_to_call(argument1, argument2, ...)
```

This could be suitable if the chosen entry point returns a scalar value. The arguments may be ranges of cells, which is an analogy for matrices in Matlab. The cell is then filled with the first element in the first returned matrix of function_to_call. In most cases this does not satisfy the needs, and one has to customize the Visual Basic code.

One is often interested in the solution vector x, and perhaps to call TOMLAB from Excel via a macro. Then the function should be rewritten as a subroutine, setting the variables/matrices to be sent to TOMLAB to ranges of the Excel worksheet. This includes both input variables/matrices and output variables/matrices. When the subroutine has been written it can be called from the menu: `Tools -> Macro -> Macros...`

8.3 TOMLAB /CPLEX LP example

The TOMLAB distribution includes an example running TOMLAB /CPLEX for an LP problem from an Excel workbook. It is available in the TOMLAB distribution in the /examples/sal/excel directory.

In summary, the example was constructed like this:

1. An m-file called *solvelp.m* was created to take simple input in the form of matrices for c, A, x_L, x_U, b_L and b_U. This m-file simply passes the matrices to cplex.m from where the CPLEX solver is called.
2. The profiler in Matlab was turned on and the solvelp function was called with some LP data. When CPLEX returned, the profile report command displayed the m- and mex-files used during the execution.
3. These files were copied to a dedicated project directory for the new Excel example.
4. The Excel Add-In was built using the MCC compiler.
5. Excel was started and the Visual Basic Editor was opened. The example `.bas`-file was imported. The Visual Basic `solvelp` function was replaced by a `solvelp` subroutine.
6. A problem example was set up in the Excel worksheet. A push button was added and assigned the solvelp macro.
7. The workbook was saved as lpsolver.xls.
8. The macro was run by clicking on the pushbutton. No error messages appeared, and the example was done.
9. In case an error message appears, it would likely be about missing m-functions. The missing files would need to be added and the project recompiled.

9 Calling TOMLAB using MATLAB Engine

This section describes an example of calling TOMLAB using the MATLAB Engine from a stand alone application written i C.

9.1 The MATLAB Engine and TOMLAB

A stand alone application can utilize the MATLAB Engine API to literally start a MATLAB session and run commands in it as if it was a normal MATLAB session. The application can also create MATLAB data structures and put to the workspace of the MATLAB Engine workspace.

One can use the MATLAB Engine to call TOMLAB if one wants to be able to use the TOMLAB functionality as if it was run from MATLAB directly. For example, one advantage of the MATLAB Engine is when upgrading TOMLAB. Then your applications will use the upgraded version of TOMLAB without the need of recompiling the application. A drawback with the MATLAB Engine is that a MATLAB session requires much memory and takes more time to start than a compiled stand alone TOMLAB solution.

Calling TOMLAB and MATLAB using the MATLAB Engine doesn't require a TOMLAB stand alone license (TOMLAB SAL) or a MATLAB compiler license.

9.2 Compiling the example

In the TOMLAB distribution in directory: /examples/sal/engine there is an example of how to call TOMLAB using the MATLAB Engine.

On Windows make sure you have set the `PATH` environment properly according to the section: `Building the example -> Setting paths` in the README file in `/examples/sal/engine`. If you are using Microsoft Visual Studio C++ then call exmake.bat. If you are using another compiler then you could need to change the name of MATLAB compiler options file in exmake.bat. Some compilers need special treatment with floating point exception handling. See the MATLAB help for more information on this. The command line for compiling this stand alone application using the MATLAB Engine looks like this:

```
> mex -f <engine options file> tomeng.c
```

where <engine options file> is the options file to use for the compiler used by mex. In case of Microsoft Studio Visual C++ it is: <matlabroot>\bin\win32\mexopts\msvc50engmatopts.bat

On Unix make sure dynamic library path is set according to the section: Building the example -> Setting paths in the README file in /examples/sal/engine. Then call exmake.sh to compile the application. The command line for compiling a stand alone application under Unix looks exactly as it does for Windows, but the <engine options file> is: <matlabroot>/bin/engopts.sh instead.

9.3 Running the example

The example application sets up a QP problem by passing data to the MATLAB Engine workspace from the C part of the application. Then it uses a TOMLAB assign routine to create a TOMLAB representation of the problem and solves it using an arbitrary solver, chosen by the user.

The executable created when compiling the application should be called with two arguments:

```
> tomeng <tomlab directory> <TOMLAB QP solver name>
```

Assume TOMLAB is installed in c:\tomlab on a Windows system and /usr/local/tomlab on a Unix system and we want to use snopt to solve the problem, then the execution of tomeng would look like this respectively:

```
> tomeng c:\tomlab snopt
```

or

```
> ./tomeng /usr/local/tomlab snopt
```

If any of the steps during program execution fails it probably is because of an invalid TOMLAB installation at the chosen TOMLAB directory, an invalid TOMLAB license or a bad solver choice. The source code of this example is situated in /examples/sal/engine/tomeng.c.

A Example source files

Source code for the simple client application. Notice that the preprocessor flag MCC4 has to be defined when compiling this file for use with MCC 4.

Source code for application.c

Source code for the interface function *solveqp.m* :

References

[1] mailto:support@tomopt.com

[2] http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/

Article Sources and Contributors

SAL *Source*: http://tomwiki.com/index.php?oldid=2604 *Contributors*: Elias