ISSN 1500-6050

Oslo Scientific Computing Archive

Report 1999-01

How to use Matlab in C++ programs

Elizabeth Acklam Donna Calhoun
Kent-Andre Mardal

April 26, 1999

Aims and scope: *Traditionally, scientific documentation of many of the activities in modern scientific computing, like e.g. code design and development, software guides and results of extensive computer experiments, have received minor attention, at least in journals, books and preprint series, although the the results of such activites are of fundamental importance for further progress in the field. The* Oslo Scientific Computing Archive *is a forum for documenting advances in scientific computing, with a particular emphasis on topics that are not yet covered in the established literature. These topics include design of computer codes, utilization of modern programming techniques, like object-oriented and object-based programming, user's guide to software packages, verification and reliability of computer codes, visualization techniques and examples, concurrent computing, technical discussions of computational efficiency, problem solving environments, description of mathematical or numerical methods along with a guide to software implementing the methods, results of extensive computer experiments, and review, comparison and/or evaluation of software tools for scientific computing. The archive may also contain the software along with its documentation. More traditional development and analysis of mathematical models and numerical methods are welcome, and the archive may then act as a preprint series. There is no copyright, and the authors are always free to publish the material elsewhere. All contributions are subject to a quality control.*

Oslo Scientific Computing Archive 1999-01

Title

How to use Matlab in C++ programs

Contributed by

Elizabeth Acklam
Donna Calhoun
Kent-Andre Mardal

Communicated by

H. P. Langtangen
March 20, 1999

Oslo Scientific Computing Archive is available on the World Wide Web. The format of the contributions is chosen by the authors, but is restricted to PostScript files, generated from LaTeX, and HTML files for documents with movies and text, and compressed tar-files for software. There is a special LaTeX style file and instructions for the authors. There is also a standard for the use of HTML. All documents must easily be printed in their complete form.

Contents

This report should be referenced as shown in the following BibTeX entry:

```
@techreport{OSCA1999-01,
  author = "Elizabeth Acklam and Donna Calhoun and Kent-Andre Mardal ",
  title = "How to use Matlab in C++ programs",
  type = "Oslo Scientific Computing Archive",
  note = "URL: http://www.math.uio.no/OSCA; ISSN 1500-6050",
  number = "\#{}1999-01",
  year = "April 26, 1999",
}
```

How to use Matlab in C++ programs

Elizabeth Acklam[1] **Donna Calhoun**[2]
Kent-Andre Mardal[3]

Abstract

The report describes a C++ interface to Matlab, such that simulation programs can use Matlab for array computation and visualization during execution.

1 Introduction

One of the more tedious tasks in scientific computing involves plotting the results of your computations. Computational scientists usually write their programs in some conventional language such as FORTRAN, C or C++, write out results of their computations in data files that are stored on disk, and then read these files into another package such as Matlab to manipulate and plot their data. Rarely does one make use of graphics libraries that can be called directly from their programs.

To simplify the task of plotting data created by C++ programs, we present a set of C++ classes that can be used to make direct calls to Matlab functions. While the original motivation for designing these classes was to treat Matlab as a graphics library, it will be obvious to the reader that these classes will make available all of the Matlab routines to the C++ program.

The classes presented here serve as an interface to the MATLAB routines supplied in the *Matlab Engine Library*, described in the Matlab External Interface Guide [1]. It is assumed that the interested reader will be using the array classes supplied in the Diffpack software and so all classes work directly with these arrays.

[1]Numerical Objects A.S. Email: `eac@nobjects.com`.

[2]Department of Applied Mathematics, University of Washington, Seattle, USA. Email: `calhoun@amath.washington.edu`.

[3]Numerical Objects A.S. Email: `kentm@math.uio.no`

2 The MatlabEngine class

We now assume we have some general C++ matrix and vector classes with base index 1, which may be Diffpack's `Mat(real)` and `Vec(real)` classes. First, you must include the MatlabEngine headerfile:

```
#include <MatlabEngine.h>
```

The most important member functions of the class are listed below:

```
class MatlabEngine
{
  public:
    MatlabEngine(); //constructor
   ~MatlabEngine(); //destructor

    void putMatrix(const Mat(real)& A, const String& name);
    void putVector(const Vec(real)& v, const String& name);
    void putScalar(real x, const String& name);

    Mat(real) getMatrix(const String& name);
    Vec(real) getVector(const String& name);
    real getScalar(const String& name);

    void operator(const String& s);

    int wait();
};
```

The `'put'` functions allow the user to put a matrix, vector or scalar into MATLAB. The variable `name` is what identifies the variable inside MATLAB. You can use this name in other MATLAB calls later in the program. The `'get'` functions allow you to retrieve matrices, vectors and scalars with a given name from MATLAB. The member function `operator(const String& s)` allows you to carry out any MATLAB call from your program, and finally, `wait()` waits for you to hit return before the program continues. Now we present a simple example on how to use the MatlabEngine:

```
#include <Vec_real.h>
#include <math.h>
#include <MatlabEngine.h>

int main()
{
  Vec(real) x(101), v(101);
  for (int i=1; i<=101; i++){
    x(i)=0.02*(i-1);
    v(i)=cos(5*3.1415*x(i));
  }
  MatlabEngine matlab;
  matlab.putVector(v,"v");
  matlab.putVector(x,"x");
  matlab("plot(x,v);");
  matlab.wait();

  return 0;
}
```

3 The MatlabGraphics class

The MatlabGraphics class is a subclass of the MatlabEngine class. Several of MATLAB's graphics commands are implemented as members of the class. This hides the MATLAB engine and therefore makes it easier to use. The most important member functions of the class are:

```
class MatlabGraphics : public MatlabEngine
{
  public:
    void plot(const Vec(real)& x, const Vec(real)& y);
    void plot(const Vec(real)& x, const Vec(real)& y,
              const String& s = "-");
    void plot(const Mat(real)& A, const Mat(real)& B);
    void plot(const Mat(real)& A, const Mat(real)& B,
              const String& s = "-");
    void mesh(const Mat(real)& x, const Mat(real)& y, const Mat(real)& A);
    void surf(const Mat(real)& x, const Mat(real)& y, const Mat(real)& A);

    void title(const String& s);
    void xlabel(const String& s);
    void ylabel(const String& s);
    void gtext(const String& s);
    void figure(int figNum);

    void print(const String& s);

    void hold(int c); // c should be set to MatlabGraphics::ON
                      // or MatlabGraphics::OFF
};
```

One important thing to notice is that when you plot vectors and matrices using the MatlabGraphics class, the name of the variable is not sent in to the workspace as it is in the MatlabEngine class. This means that you can not use the member functions of the MatlabGraphics class if you later wish to call your variables explicitly in MATLAB. Below is an example on how to use the MatlabGraphics class:

```
#include <Vec_real.h>
#include <math.h>
#include <MatlabEngine.h>

int main()
{
  Vec(real) x(101), v(101), w(101);
  for (int i=1; i<=101; i++){
    x(i)=0.02*(i-1);
    v(i)=cos(5*3.1415*x(i));
    w(i)=sin(5*3.1415*x(i));
  }

  MatlabGraphics graph;

  graph.putVector(v,"v");
  graph.putVector(w,"w");
  graph.putVector(x,"x");

  graph.plot(x,v,"red");
```

```
  graph.hold(MatlabGraphics::ON);
  graph.plot(x,w,"g-.");
  graph.title("Two curves made in a C++ program");
  graph.wait();

  return 0;
}
```

4 Additional Diffpack features

In addition to the more general C++ features in the classes described above, there are some member functions more specifically made for Diffpack. The MatlabEngine class has the public members

```
void putArrayGen(const ArrayGen(real)& A, String s);
```

and

```
ArrayGen(real) getArrayGen(const String& s);
```

which allows you to put and get an instance of the class `ArrayGen(real)` of any dimension into the MATLAB workspace. Multi-dimensional arrays are a new feature in Matlab 5, and therefore only 1D and 2D instances of `ArrayGen(real)` will work in Matlab 4. There is no `getArrayGen` routine supported for Matlab 4 as this would be either a matrix, a vector or a scalar.

You can also plot 2D `ArrayGen` objects directly by using the member functions of the `MatlabGraphics` class

```
  void mesh(const ArrayGen(real)\& A)
  void mesh(const Mat(real)\& A)
  void surf(const ArrayGen(real)\& A)
  void surf(const Mat(real)\& A)
```

The great advantage with the `ArrayGen` functionalities is that one can now easily plot Diffpack's finite difference field, the `FieldLattice`. An example of this is given below.

```
#include <FieldLattice.h>
#include <FieldFunc.h>
#include <TimePrm.h>
#include <math.h>
#include <MatlabEngine.h>

real field(const Ptv(real)& x, real t=DUMMY){
  real s=sin(3.1415*t/16);
  real r=sqrt( x(1)*x(1)*s*s + x(2)*x(2) ) + 0.0001;
  return sin(r)/((real) r);
}

int main()
{
  MatlabEngine matlab;
```

```
  TimePrm  tip;

  GridLattice grid;
  grid.scan("d=2 [-10,10]x[-10,10] [0:30]x[-5:25]");
  tip.scan("dt=1 t in [0.0,25.0]");
  FieldLattice f(grid,"f");

  tip.initTimeLoop();
  while (!tip.finished())
    {
      f.fill(field, tip.getTime());

      // give each ArrayGen object a name
      String name = oform("f%d" ,tip.getTimeStepNumber());
      matlab.putArrayGen(f.values(), name);
      //      f.values().save(oform("f%d.m" ,tip.getTimeStepNumber()));
      tip.increaseTime();
    }

  // Create a movie of the results
  matlab.putScalar(tip.getTimeStepNumber(), "T");
  matlab("M=moviein(T);");

  for (int i=0; i<tip.getTimeStepNumber(); i++){
    matlab(oform("M(:,%d)=getframe;", i+1));
    matlab(oform("surf(f%d);" ,i));
  }

  matlab("movie(M)");

  matlab("print sombrero");
  matlab.wait();

  return 0;
}
```

The `MatlabGraphics` class has no functionalities for plotting `FieldFE` objects directly. However, the `DrawFE` class allows you to print the field to a file in a format that MATLAB can read, and you can use the `MatlabEngine` to load the data into the MATLAB workspace. To demonstrate this we rewrite the previous example, and you should observe that only minor changes in the program are required to produce a similar movie.

```
#include <FieldFE.h>
#include <FieldFunc.h>
#include <TimePrm.h>
#include <SimRes2matlab.h>
#include <math.h>
#include <MatlabEngine.h>

real field(const Ptv(real)& x, real t=DUMMY){
  real s=sin(3.1415*t/16);
  real r=sqrt( x(1)*x(1)*s*s + x(2)*x(2) ) + 0.0001;
  return sin(r)/((real) r);
}

int main(int nargs, const char** args)
{
  initDIFFPACK(nargs, args);
```

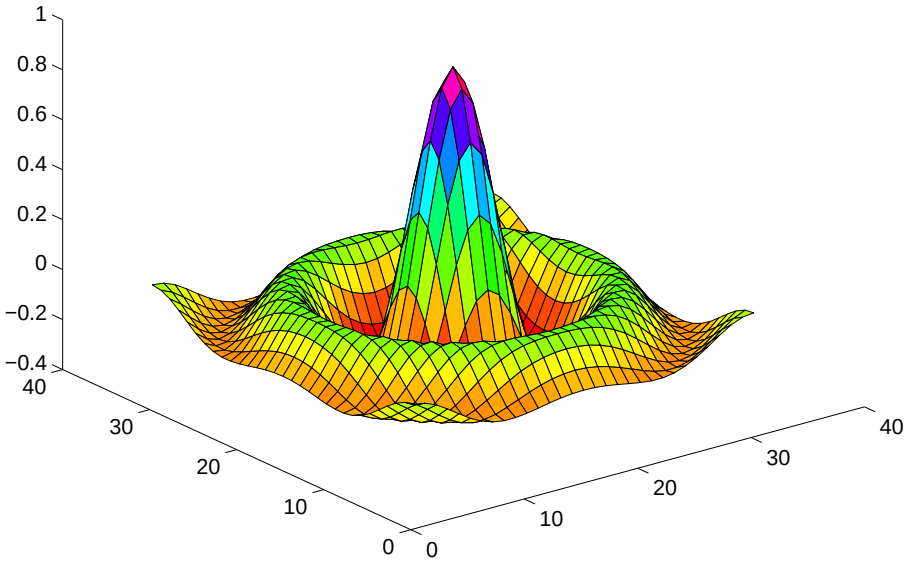

Figure 1: The plot produced by the example program.

```
TimePrm  tip;

MatlabEngine matlab;

GridFE grid;
grid.scanLattice("d=2 [-10,10]x[-10,10] [1:31]x[1:31]");
tip.scan("dt=1 t in [0.0,30.0]");
FieldFE f(grid,"f");
tip.initTimeLoop();
String script,data;
while (!tip.finished())
  {
    f.fill(field, tip.getTime());
    script = oform("f%d.m" ,tip.getTimeStepNumber());
    data  = oform("f%d.m.dat" ,tip.getTimeStepNumber());
    Os scriptfile(script,NEWFILE);
    SimRes2matlab::plotmatlabScalar(f, data ,scriptfile);
    tip.increaseTime();
  }

// Create a movie of the results

matlab.putScalar(tip.getTimeStepNumber(), "T");
matlab("M=moviein(T);");

for (int i=0; i<tip.getTimeStepNumber(); i++) {
  matlab(oform("M(:,%d)=getframe;", i+1));
  matlab(oform("f%d;" ,i));
}
```

```
  matlab("movie(M)");
  matlab.wait();


  return 0;
}
```

A How to compile and link an application with Matlab

When compiling Diffpack programs containing Matlab calls, one needs access to the Matlab header files, and for linking one needs the Matlab library files `libeng.so`, `libmat.so`, `libmi.so`, `libmx.so` and `libut.so`. The easiest way to access Matlab from Diffpack applications is to make proper links from `$NOR/ext/$MACHINE_TYPE/lib` and `$NOR/ext/include` to the Matlab include and library directories. Assume that the Matlab `libeng.so`, `libmat.so`, `libmi.so`, `libmx.so` and `libut.so` libraries is located in

```
/local/MATLAB/extern/lib/lnx86/
```

We then either make appropriate links to the official Diffpack directories for external software.

```
ln -s /local/MATLAB/extern/lib/lnx86/libmat.so \
$NOR/ext/$MACHINE_TYPE/lib/libmat.so
ln -s /local/MATLAB/extern/lib/lnx86/libeng.so \
$NOR/ext/$MACHINE_TYPE/lib/libeng.so
ln -s /local/MATLAB/extern/lib/lnx86/libmx.so \
$NOR/ext/$MACHINE_TYPE/lib/libmx.so
ln -s /local/MATLAB/extern/lib/lnx86/libmi.so \
$NOR/ext/$MACHINE_TYPE/lib/libmi.so
ln -s /local/MATLAB/extern/lib/lnx86/libut.so \
$NOR/ext/$MACHINE_TYPE/lib/libut.so

ln -s /local/MATLAB/extern/include/cmex.h   $NOR/ext/include/cmex.h
ln -s /local/MATLAB/extern/include/engine.h $NOR/ext/include/engine.h
ln -s /local/MATLAB/extern/include/fintrf.h $NOR/ext/include/fintrf.h
ln -s /local/MATLAB/extern/include/matrix.h $NOR/ext/include/matrix.h
ln -s /local/MATLAB/extern/include/mex.h    $NOR/ext/include/mex.h
ln -s /local/MATLAB/extern/include/tmwtypes.h \
$NOR/ext/include/tmwtypes.h
ln -s /local/MATLAB/extern/include/mat.h  $NOR/ext/include/mat.h
```

The only explicit task that the user must do to take advantage of Matlab in Diffpack programs is then to include

```
SYSLIBS += -lmat -leng -lmx -lmi -lut
```

in the `.cmake2` file *in each application directory that needs Matlab.*

The libraries are dynamically linked and the LD_LIBRARY_PATH variable must be set in your .cshrc or .envir file.

```
setenv LD_LIBRARY_PATH $NOR/ext/$MACHINE_TYPE/lib
```

If you do not have privilege to make links to the `$NOR/ext` directories, you can add some statements in the `.cmake2` file such that the compiler and linker find the appropriate include and library files:

```
SYSLIBS += -lmat -leng -lmx -lmi -lut
INCLUDEDIRS += /local/MATLAB/extern/include
LDPATH += /local/MATLAB/extern/lib/lnx86
```

The default implementation assumes that you are using Matlab version 5, but you can also use Matlab version 4 by adding the following lines in the `.cmake2` file

```
CXXOPTIONS += -DMATLAB4
SYSLIBS +=  -lmat
INCLUDEDIRS +=  -I/mn/lugulbanda/apps/MATLAB4.linux/extern/include
LDPATH +=-L/mn/lugulbanda/apps/MATLAB4.linux/extern/lib/lnx86
```

where we assume that Matlab version 4 is located in

```
/mn/lugulbanda/apps/MATLAB4.linux
```

(relevant for the Department of Mathematics, Oslo)

References

[1] *MATLAB External Interface Guide* The Mathworks, Inc. Natick, Massachusetts 01760, (508) 653-1415.