# RMaxima README File

Kseniia Shumelchyk
Hans W. Borchers

2012-07-08

## 1 Introduction

The goal of the project RMaxima is to create a package that allows to use from within R the capabilities of Maxima, a Computer Algebra System much more powerful than the symbolic algebra systems available for R right now. The package should add new functionality to R for using symbolic expressions, such as equation solving, differentiation, integration, three-dimensional plotting, or multiple-precision arithmetics.

## 2 Project Description

At the beginning were explored and analyzed existing approaches to communicate with Maxima. Two ways of interaction with the Maxima were identified. First, based on TCP sockets and second based on using the mechanism of pipes. Similar API BSD sockets are available in most modern operating systems, but some separate methods are required to start the Maxima process.

In case of using unnamed pipes portability is provided automatically but will depend on additional libraries. Based on previous knowledge it was natural to look at Boost.Process which is associated with, but not officially part of, the well-known C++ Boost[1] library. Possible alternatives were, among others, *Libexecstream*[2] or *PStreams*[3].

For the development of this package unnamed pipes were chosen as a way of allowing the R console to interact with the Maxima process. After connecting to Maxima there is the challenge of interaction with the output of the Maxima console. The communication protocol via Maxima console is in principle quite simple: The user repeatedly enters expressions and Maxima produces some result and returns it as a string. This message exchange pattern may be described as 'request-response'.

To simplify the processing, Maxima output is governed by an option file (`display.lisp`). It configures Maxima output in such way that allows to iden-

[1] `http://www.boost.org`
[2] `http://libexecstream.sourceforge.net/`
[3] `http://pstreams.sourceforge.net/`

tify different parts of the output coming from Maxima. Result expressions are marked in a special way and displayed in the one-dimensional, i.e. non-graphical mode. Parsing the output is realized through regular expressions provided by the library Boost.Regex.

All this has been implemented in a demo system at the end of the first coding phase, except that the communication between R and Maxima consoles is more batch-like and not yet truly interactive.

The original project plan, taken from the project proposal, is enclosed here.

Table 1: Project Plan

| **Timeline** | **Activity** |
|---|---|
| April 23 - May 10 | Reading the documentation of Maxima and R, getting familiar with the project. |
| May 11 - May 20 | Discussing a project architecture with a mentor. |
| May 21 - June 3 | Creating a simple Maxima "communication manager" with unit tests. |
| June 4 - June 10 | Code review of the managers code; testing and bugfixes |
| June 11 - June 19 | Extending Maxima communication manager on Windows and Mac OS |
| June 20 - June 22 | Testing and bugfix |
| June 23 - July 2 | Adding ability to send commands from R project interpreter to maxima and getting the result |
| July 3 - July 4 | Code review and making changes |
| July 5 - July 8 | Testing and bugfix |

It shows that the project is about one week behind the original project plan. The intention to provide a general package that will be distributed on CRAN may not be reached in the end. But usable versions for Linux and Windows are still possible and could later be extended to full-scale packages by the community.

## 3 Project Results (1st Coding Phase)

The result of the first coding phase of the project is a demo program proving the feasibility of the approach described above. When all tools needed are installed (Boost, Boost process library, Maxima) as listed below, the demo library can be compiled as a shared library for R and then dynamically loaded into R with the `dyn.load()` function.

The demo program will automatically load the RMaxima shared library and then send some simple commands to Maxima. The result strings returned from Maxima will be displayed on the R command terminal. Here is an example, with the symbolic commands `float`, `sin + cos`, `plot2d`, and `diff` send to Maxima:

```
> source("maximaexe.R")
# Calling C function
>>> float(1/3);
0.33333333333333
# Returning to R

# Calling C function
>>> sin(%pi/2) + cos(%pi/3);
3/2
# Returning to R

# Calling C function
>>> plot2d(x^2 - x + 3, [x, -10, 10]);
""
# Returning to R

# Calling C function
>>> diff(sin(x), x);
cos(x)
# Returning to R
```

The graphics window, opened by Maxima through the `plot2d` command, will only shortly pop up and then disappear again. It has to be seen how graphical output from Maxima can be kept open until the user closes the window..

The other commands return correct results from Maxima, at the moment available only as strings. To call Maxima interactively from the R command line will be one of the next tasks.

## 4 Problems Encountered

**Maxima** – Automatic processing of Maxima console output is quite complex: it's difficult to impose a general structure on Maxima output that will be valid for all kinds of requests sent to Maxima. According to documentation of the Maxima distribution the system has a number of variables that control the allocation of significant fragments of console output, which can significantly simplify the automatic processing of Maxima output. Here are the variables we used and their meanings:

1. `prompt-prefix` - the string is printed before each input request;
2. `prompt-suffix` - the string is printed after each input request;;
3. `alt-display2d` - a function that replaces the standard Maxima in two-dimensional mode when printing expressions.

Setting these variables happens in the file `display.lisp`, written in Common Lisp (the programming language the kernel of Maxima is written in), and the path to that

file is provided with the option `.p <File Path>`. The content of this file is loaded during Maxima booting.

**Boost** – Also there were problems with using the Boost.Process library. For correctly working functions from Boost.Process required the Boost.Filesystem version 2, but latest Boost versions by default use version 3. So for using Boost.Process these things had to be fixed manually, and the fixes are included in the Boost Process files distributed with RMaxima.

The Boost Process[4] library was rejected by the Boost core team in March 2011 and has since then not been included into Boost officially. A consequence is that it must be downloaded and installed separately. That poses the problem of how to make RMaxima become a package distributed on CRAN. By the way, we were in contact with the developer of Boost.Process and he indicated he is still working on the library at times.

[Remark: As Dirk Edelbuettel told us, the CRAN platform for Linux is "Debian testing" which includes 'boostlib', the standard Boost library. For Windows he provides Uwe Ligges with files and Boost headers such that the compilation process works on that platform too. (Mac OS X ?)]

## 5 Next Steps (2nd Coding Phase)

The RMaxima GSoC project has reached its aim to construct an interprocess communication between R and another process running Maxima. Command strings can be sent from R to Maxima, and returned results will be displayd in the R command terminal.

The following goals appear as natural follow-ups for the second phase of the project:

- Polish the interprocess communication, e.g., how to start and stop the Maxima process from within R.
- Let the user send commands to Maxima interactively; shall these commands be filtered; how will a set of commands be combined; what about errors and error mesages generated by Maxima.
- How to handle plots generated by Maxima – or disallow them; these are now windows popping up and disappearing immediately.
- Extend to Windows, i.e. what is needed to generate a binary Windows version, even if this cannot be provided as a binary package through CRAN.
- Some steps in the direction of making it a package, for instance, how can the Boost process library be distributed with the package.
- Specialized R commands that will only call, e.g., integration routines in Maxima (here the approach of the Euler Math Toolbox may be a good place to look at).

[4] `http://www.highscore.de/boost/process/`

- Use Maxima results in R, i.e. use floating point numbers returned from Maxima in R calculations, or convert function expressions returned from Maxima into R functions.
- How can high-accuracy numbers returned from Maxima be utilized in R through the Rmpfr package.

Not all these task can be worked on during the second coding phase. The student developer and the mentors will have to make some decisions.

## 6 Installation

This installation procedure for RMaxima works only on Linux systems and has been tested on Ubuntu and Debian Linux. It is assumed that a newer version of R is installed as usual, and that the normal C++ development environment with 'gcc' etc. is available on the Linux system.

1. Install the Boost libraries, the preferred way is to use the Synaptic package manager, you can also try

   ```
   sudo apt-get install libboost-dev
   ```

   Make sure that the Boost libraries `boost_filesystem` and `boost_regex` are included with the installation.

2. Install the Boost process library. The file provided is `BoostProcess.tar.gz` that needs to be expanded. The subdirectory `boost/process` shall be copied to the Boost directory on the Linux system, i.e. `/usr/include/boost/process/`, the file `process.hpp` into `/usr/include/boost/`.

   As administrator rights required, for this action you could apply the `sudo nautilus` command.

   Don't touch the `libs` directory, it is meant for installing on Windows.

3. Install Maxima for the Linux system. Usually this is best be done through the Synaptic package manager, or with `sudo apt-get install maxima`. Exporting the `MAXIMA_PATH` is no longer necessary.

Now we are ready to install and run the RMaxima demo program:

1. Unpack the files included in the file `RMaxima` into a directory, would be best to call that `RMaxima` too. The directory will contain the following files:

   ```
   display.lisp    MaximaChain.cpp     MaximaChain.h
   maximaexe.cpp   maximaexe.R
   ```

2. Go to this directory (using a command terminal) and compile the RMaxima library with the following command:

   ```
   R CMD SHLIB maximaexe.cpp MaximaChain.cpp
               -lboost_filesystem -lboost_system -lboost_regex
   ```

If no error occurs, this command will, among others, generate a shared library called `rmaximaexe.so`.

3. Start R (with the `R` command) in this directory and source the R script file that calls Maxima with some fixed demo commands and returns Maxima's results:

   ```
   source("rmaximaexe.R")
   ```

This will load the shared library and display the interaction with Maxima through fixed commands, as described in section "Project Results" above.