TWMCC ⋆ Texas-Wisconsin Modeling and Control Consortium

Technical report number 2003-01

Ten Years of Octave—Recent Developments and Plans for the Future

John W. Eaton*and James B. Rawlings
Department of Chemical Engineering
University of Wisconsin-Madison
Madison, WI 53706 USA

May 2, 2003

Abstract

GNU Octave (`www.octave.org`) has been publicly available for more than ten years. During that time the scope of the project has grown from a simple interface to numerical tools intended for classroom use to a capable system with thousands of users worldwide.

This paper describes in detail some recently completed additions to Octave and provides a summary of some current and future projects.

1 Introduction

The original motivation for writing Octave was to provide software to accompany an undergraduate textbook on chemical reaction engineering. That textbook is now available, together with Octave code to compute all the figures and examples discussed in the book (Rawlings and Ekerdt, 2002)[1].

With the publication of the textbook, we have finally met the goal we originally set. In the meantime, Octave has become much more widely used than we ever imagined at the start of the project. In addition to using Octave to support our teaching[2] and research activities, Octave is used by others for teaching numerical analysis, computer programming,

*`jwe@bevo.che.wisc.edu`

[1]Code and more information is available on the web at `http://www.che.wisc.edu/~jbraw/chemreacfun`.

[2]We have used Octave to teach a several courses at UW-Madison in a classroom using a wireless network and notebook computers issued to all the students. By giving the students access to computational tools during the lecture, we were able to much more successfully integrate computing in the course.

simulation, modeling, supporting secondary school instruction, and for a wide variety of industrial and academic research applications including signal processing, statistical genetics, computer vision, econometrics, neuroscience, and computational biology.

When we first started thinking about Octave, we had just become aware of the original Fortran version of Matlab[3], and by the time coding actually began, growing number of our colleagues were beginning to use the new commercial version of Matlab as their primary computational tool. We eventually decided to implement something that would be mostly Matlab-compatible so that our colleagues could switch to using Octave without having to learn a completely new language.

Making Octave freely available has also contributed to its success, as has the growing popularity of GNU/Linux systems, but Octave's compatibility with Matlab is probably equally important. Many people become interested in Octave because they hear it is a "Matlab clone"[4] available free of charge. The problems associated with this perception and with language compatibility issues are discussed by Eaton (2001), who argued for abandoning Matlab compatibility entirely. If we were designing a new language from scratch we might take that approach. But given that the language used by Octave is already very close to Matlab and that many (most?) people using Octave have chosen it because it offers compatibility, it seems that differences in behavior only cause trouble, even if those differences can be justified on the grounds of correctness or for consistency in the language. So instead of trying to correct any perceived misfeatures of Matlab, we are actively working to remove the differences, making an increasingly large body of existing Matlab code (much of it freely available) accessible to Octave users. Ideally, it will be trivial to "port" code from Matlab to Octave.

Recent changes to enhance compatibility include the addition of cell and structure arrays (multidimensional lists and tables), a new representation for variable-length argument lists, subfunctions, function handles, and many other smaller improvements. Projects that are currently underway to improve compatibility include the addition of multidimensional numeric arrays and sparse matrices. In addition, a number of new features have been added and improvements made that are not directly related to Matlab. These include the addition of a numeric value to represent missing values that is compatible with GNU R, improved differential equation solvers, an interface to MPI (message passing interface for parallel processing), and a method for communicating with R (allowing Octave to execute R code and R to execute Octave code), and a better method for signal handling. The following sections examine these additions in more detail.

Ideas for other future projects include the addition of banded, triangular, symmetric, or other specialized matrix types, improved graphics, GUI features, and linkage to other scripting languages (possibilities include Guile, Python, Perl, Ruby, etc.) following the method used by the R-Octave interface.

[3]Matlab and Handle Graphics are registered trademarks of the MathWorks, Inc.

[4]We have never referred to it this way.

2 Recent Changes

2.1 Changes for Compatibility with Matlab

The following changes have been made not only to improve Octave's compatibility with Matlab, but also to improve its overall capabilities. In some cases (for example, cell and structure arrays and variable argument lists), Octave already had similar capabilities, but the new code offers additional features and more flexibility. In other cases (for example, subfunctions and function handles), the changes implement completely new concepts for Octave.

2.1.1 Cell and Structure Arrays

Matlab's *cell array* is a multidimensional indexed container object that can hold any other type of object.

One-dimensional lists have been available in Octave for several years. Although it is possible use them to mimic multidimensional lists (a list of lists of lists of ...), indexing them is inconvenient. The addition of cell arrays improves Octave's ability to handle heterogenous collections of objects. The list data type is still used in a number of functions, but should be considered deprecated and not used in new code.

Matlab's *structure array* is a multidimensional indexed map object in which each element of the map is an array of the same length, and each array element can hold any other type of object.

Octave has had a structure type almost from the beginning (before Matlab had anything comparable). It could be used to store arrays of objects (or lists) but there was no automatic way to enforce all structure fields to have the same number of elements.

In Octave, both of these new data types are currently limited to two dimensions, but that restriction should be eliminated once Octave fully supports multidimensional numeric arrays.

2.1.2 Variable Argument Lists

Early versions of Matlab offered no way to pass a variable number of arguments. A function that needed to approximate this had to be defined using something like

```
function f (x1, x2, x3, x4, x5, x6, x7, x8, x9)
  ...
```

and then use `nargin` to check the number of arguments that were actually supplied. Obviously, it is inconvenient to have to modify the function definition if you should happen to need more than the maximum number of arguments allowed. Octave fixed this problem by introducing a new syntax and some functions for passing a variable number of arguments that was patterned after the C language. Later, when Matlab was modified to include cell arrays, it was also changed to allow variable argument lists to be passed as a special cell array object. Recently, Octave has adopted the Matlab-compatible way of handling variable length argument lists and deprecated the original method.

This is an example of the kind of trouble that can arise due to innovations that Eaton described at DSC-2001. Even though a transition to a slightly different method can be somewhat difficult, we now believe, given Octave's strength as a MATLAB-compatible system, that in cases such as this it is better to support the MATLAB way of doing things and phase out the previous method[5].

2.1.3 Subfunctions

Originally, MATLAB only allowed functions to be defined in files, with one function per file. With newer versions, it is possible to define several functions in one file. Only the first is visible outside the file, with the scope of the remaining *subfunctions* limited to the file. For example, in a file containing

```
function fun ()
  ...
function sub1 ()
  ...
function sub2 ()
  ...
```

the functions `sub1` and `sub2` may only be called by each other or `fun`. Note that there is no token to mark the end of a function—a new `function` keyword ends the previous function definition and marks the beginning of a new subfunction.

Octave, which also allows functions to be defined on the command line and includes an `endfunction` keyword to support this feature, now also supports subfunctions, but only inside function files. We considered trying to support nesting functions to an arbitrary depth

```
 function fun ()
   ...
   function sub1 ()
     ...
     function sub2 ()
       ...
     endfunction
   endfunction
endfunction
```

but in the end decided that conflicts with the MATLAB semantics and the extra complexity were not worth the effort.

2.1.4 Function Handles

In earlier versions of Octave and MATLAB, the function `feval` could be used to call a function by name (as a character string). Recent versions of MATLAB allow functions to

[5] At least until Octave is the dominant system for interactive matrix computations.

be referenced using the syntax `@function_name`, which creates a *function handle* object that references `function_name` using the lookup rules that apply at the location where the function handle is created. This feature allows subfunctions to be passed to `feval`.

MATLAB function handles are also a multidimensional object. This is not really necessary, as a collection of function handles could be stored in a cell array, and Octave's implementation currently allows only a scalar function handle object.

2.2 Other Changes

A number of other changes have been made recently that enhance Octave's capabilities but have nothing to do with MATLAB compatibility. Some of the more notable additions are described below.

2.2.1 Missing Values Compatible with R

Octave can now represent missing numeric values in a way that is compatible with R (using a specific NaN value with the low word set to the mysterious decimal value 1954). Using this definition will make it easier for R and Octave to share data as we will be able to pass matrices between systems without having to specifically check for and handle missing values.

2.2.2 Improved Differential Equation Solvers

We now have interfaces to three additional differential equation solvers:

- `ODESSA` (Leis and Kramer, 1988): ODE solver with sensitivity analysis based on `LSODE` (Hindmarsh, 1983). `ODESSA` solves the problem

$$\dot{x}(t;p) = f(x(t),t;p) \qquad x(t=0) = x_0$$

  and simultaneously computes the first order parametric sensitivities defined by the additional set of differential equations

$$\dot{S}(t) = \frac{\partial f}{\partial x} S(t) + \frac{\partial f}{\partial p} \qquad S_{ij}(t=0) = \begin{cases} 1 & \text{if } p_j = x_i(0) \\ 0 & \text{otherwise} \end{cases}$$

- `DASPK2.0` (Brown et al., 1994): Newer version of `DASSL` with a better method for computing a consistent initial condition. `DASPK2.0` solves the problem

$$f(t, x(t), \dot{x}(t)) = 0 \qquad x(t=0) = x_0,\ \dot{x}(t=0) = \dot{x}_0$$

- `DASRT` (Brenan et al., 1989): DAE solver with root finding based on `DASSL` (Petzold, 1983). `DASRT` solves the same problem as `DASPK2.0` but terminates the integration when any of the stopping conditions $g_i(x,t) = 0;\ i = 1, \ldots, N_g$ is satisfied.

We now have two `LSODE` variants and three `DASSL` variants in Octave. It would be much better to have only one of each, but we are currently relying on code from external sources, and these have developed in a seemingly haphazard way, with changes not folded back in to the original sources.

We would also like to use `DASPK3.0` (Li and Petzold, 2003), which includes a sensitivity analysis feature, but it is not yet clear whether it will have a license that will allow it to be included with Octave.

2.2.3 Parallel Processing

Parallel processing applications based on Matlab are nothing new, but none are integrated with Matlab itself, and one of the primary stumbling blocks is that many of the implementations require one license per process, which can quickly become prohibitively expensive. Octave and other free software tools have a significant advantage here, as the in incremental cost to running N additional processes is quite low.

A survey of past efforts for Matlab is available at `http://supertech.lcs.mit.edu/~cly/survey.html`. This list is divided by method:

- **Embarrassingly parallel**—coarse granularity, usually giving you some way to send sending jobs to remote systems for execution. Not usually an effective way to split up linear algebra tasks, but a reasonable way to solve problems that just require executing the same task many times (say for different parameter values).

- **Message passing** (MPI[6], PVM[7])—MPI is *message passing interface* and is one of the emerging "standards" for parallel computations. PVM is *parallel virtual machine* and has been around longer than MPI.

- **Backend support**—typically provide replacement functions for things that can be done in parallel. Some have you call `qr_p` instead of `qr` if you want to do it in parallel. The most promising is probably Matlab*P, which uses Matlab's object-oriented features to do this more transparently.

- **Compilers**—some people have written Matlab compilers that attempt to do automatic parallelization of Matlab code. Most of these efforts are several years old now, and apparently not currently maintained. None of them are as compatible with Matlab as Octave is now, so adding the features of one of these experimental systems to Octave would have the advantage of providing a more up-to-date language implementation along with the parallel tools in a system that is widely available and actively maintained.

Octave will eventually have some combination of the above methods to support parallel tasks. A few simple interfaces to MPI have been written that allow Octave to use some basic MPI functionality, but it would be better to try to provide a more complete interface to the standard and then provide the higher level functions in Octave code. This task is

[6]`http://www-unix.mcs.anl.gov/mpi/mpich`

[7]`http://www.csm.ornl.gov/pvm/pvm_home.html`

probably not too difficult, but not trivial either, as the public interface of the MPI library is relatively large.

The Matlab*P project seems active, and we might be able to use that if Octave could support the MATLAB way of doing object-oriented programming. The development version of Matlab*P is distributed with no copyright notice, but the first version was distributed with a "don't redistribute without telling us" notice in a `README` file.

2.2.4 An Interface to R

Duncan Temple Lang has contributed code to allow Octave to call R functions, and for R to call Octave functions. For example, the function call

```
ROctave_callR ("rnorm", 10)
```

in Octave will pass the value 10 (it could be any expression) to the R function `rnorm` and return the result to Octave. Similarly, the code

```
x <- .Call ("R_OctaveCall", "syl", list (a, b, c));
```

uses the Octave function `syl` to solve the matrix equation $AX + XB + C = 0$ given appropriate definitions of `a`, `b`, and `c` as R matrices.

More complex applications are also possible, for example with R calling an Octave procedure that requires a user-supplied function, which can be written in Octave or in R[8].

This interface between R and Octave is a good start, but there are a few problems that will need to be addressed before it is ready for serious use. Probably the most important problem is that several global functions in R and Octave have the same signature, which can lead to some trouble[9]. Also, the current implementation dynamically loads R (Octave) and passes copies of values to and from R (Octave), which can be slow for large matrices. In the future, it may be possible to improve performance by passing values as references instead. We will, however, still need to make the data available in a suitable format, so unless Octave and R data formats can be made compatible, some transformation will be required.

At this point, it is useful to be able to make these inter-interpreter calls, but unless we are able to make it more efficient and less clumsy to use, it seems unlikely that it will be widely used.

2.2.5 Improved Signal Handling to Avoid Resource Leaks

Until recently, Octave handled signals by calling `longjmp` to jump directly from the signal handler back to the beginning of the main event loop. On Unix-like systems, this method works to handle interrupts immediately, but it is not guaranteed to work on all systems, and with Octave's reference counted memory management, it could lead to severe memory leaks. To overcome these problems, the signal handler no longer calls `longjmp` directly.

[8]If we provide a wrapper in Octave to call back to R using `ROctave_callR` though perhaps with a little magic it will be possible to eliminate the need for the wrapper function.

[9]This problem may be partially fixed in the development version of R. See also Section 3.1.5.

Instead, it sets a global variable and returns. This variable is then checked at a number of points throughout the code using the following macro:

```
#define OCTAVE_QUIT \
  do \
    { \
      if (octave_interrupt_state) \
        { \
          octave_interrupt_state = 0; \
          octave_throw_interrupt_exception (); \
        } \
    } \
  while (0)
```

If `octave_interrupt_state` is nonzero, a C++ exception is thrown. The exception is caught at the bottom of the main event loop, the state of the interpreter is reset to a sane state, and execution continues. The advantage of throwing a C++ exception instead of using `longjmp` is that it automatically takes care of calling destructors for any temporary objects that have been constructed up to the point of the interrupt.

This macro is now used in approximately 120 locations throughout the interpreter, primarily inside loops. Inserting most of the calls only took an hour or two. A few more were added after the initial conversion.

This new method of handling interrupts is reasonably responsive for code that is a part of Octave, and is much better for avoiding resource leaks for C++ objects if we follow the relatively simple rule that class constructors are responsible for allocating resources and class destructors are responsible for releasing those resources.

Unfortunately[10], not all of Octave is written in C++, nor can all of the source be modified to call `OCTAVE_QUIT` in all the necessary locations. Even when we have access to the source code, it may be undesirable to modify it—we don't want to take on the maintenance of every piece of code that is used with Octave, and we would like to be able to easily adopt new versions of external libraries as they are available.

To make this method of signal handling work when foreign code is involved, we wrap the call to the foreign code as shown in Figure 1. The variable `octave_interrupt_immediately` is used to tell the Octave's interrupt signal handler whether it should interrupt immediately. If it is nonzero, then it should call `siglongjmp` to return to the location specified by `saved_context`. We use `octave_jmp_buf` and `octave_save_current_context` instead of `jmp_buf` and `setjmp` to hide the details of those functions to make it easier to handle potential portability problems.

A set of macros are available to allow us to avoid having to write this mess everywhere it is needed. Instead, we use

[10] Or not. . .

```
{
  octave_jmp_buf saved_context;
  octave_save_current_context ((char *) saved_context);
  if (octave_set_current_context)
    {
      octave_restore_current_context ((char *) saved_context)          1
      octave_throw_interrupt_exception ();
    }
  else
    {
      octave_interrupt_immediately++;

      call_to_foreign_code ();   // Out of our control.

      octave_interrupt_immediately--;
      octave_restore_current_context ((char *) saved_context);          2
  }
}
```

Figure 1: Octave's new method of calling foreign code.

```
BEGIN_INTERRUPT_IMMEDIATELY_IN_FOREIGN_CODE;                  1

call_to_foreign_function ();    // Out of our control.

END_INTERRUPT_IMMEDIATELY_IN_FOREIGN_CODE;                    2
```

which is much clearer and easier to type.

Fortunately, we only have to worry about these issues when the foreign code could consume more than a few tenths of a second (we expect that should be good enough for interactive interrupts). For functions that complete fast enough that it should never matter that they are uninterruptible, we don't need to bother with all of this additional code.

2.2.6 Improved Portability

Thanks to the efforts of Mumit Khan, Paul Kienzle, Per Persson and others, it is now possible to build and run Octave with compilers other than GCC and on a wider variety of systems, including Microsoft Windows and Apple Computer's Mac OS X. As more compilers begin to implement standard conforming C++ compilers, and as GCC's support for standard C++ has improved, Octave has become less dependent on GCC. Although GCC remains our preferred compiler on most platforms, it is useful to be able to build and test Octave with vendor compilers.

Octave is now running reasonably well on Microsoft Windows systems using the Cygwin toolkit[11]. It is also possible to build a mostly working version using the MinGW[12] toolkit (Cygwin tools for building, but generating a binary that doesn't require the Cygwin library

[11]`http://www.cygwin.com`
[12]`http://www.mingw.org`

to run). Not surprisingly, POSIX system call features are not well supported with the MinGW version, and process creation (using `fork` and `exec`) is not fully functional. Various binary packages of Octave for Windows are available on the web.

The port to Apple's Mac OS X was much simpler since the core of the system is Unix based and GCC is the standard compiler. A binary package of Octave is available as a part of the Fink[13] project.

3 Future Plans

3.1 Immediate Future

Plans for the immediate Future (within the next year or so) include the following items.

3.1.1 Multidimensional Arrays

Octave should support multidimensional numerical arrays, at least as a container object with a few basic numerical operations. Although it may eventually be useful to adopt a convention for functions like `svd` to work on slices of multidimensional arrays, this does not seem to be immediately necessary.

Along with support for multidimensional numerical arrays, Octave's current cell and structure arrays and function handle objects should also be extended to allow multidimensional data.

3.1.2 Sparse Matrices

Andy Adler has provided some basic sparse matrix capabilities for Octave based on the SuperLU package from Netlib[14]. This code could become the basis for a built-in sparse matrix type for Octave. The current version of Andy's code is available as a part of Octave Forge (see below for more information about the Octave Forge project).

3.1.3 Optimization Tools

A joint project with Doug Bates will result in an NLP solver that can be used in both R and Octave. This project is still in the planning stages, but we currently intend to implement an efficient SQP algorithm for dense problems by the end of the year. We also plan to implement a robust interior-point QP solver.

3.1.4 Global Namespace Cleanup

Octave's internals currently have a number of global functions that are inconsistently named and likely to conflict with the functions from other packages, should they be linked together (as R and Octave are when using the ROctave package). For example, both R (version

[13]`http://fink.sourceforge.net`
[14]`http://www.netlib.org`

1.6.2 at least, though this seems to be fixed in the current development sources) and Octave export functions with the following (C-linkage) names:

```
    extern void warning (const char *fmt, ...);
    extern void error (const char *fmt, ...);
```

Because the signatures are the same, a crash does not immediately result if Octave code calls the `error` function from R, but it will not do precisely what was intended (Octave's `error` does more than simply printing an error message).

3.1.5 API Definition

In addition to some attempt to avoid global name conflicts, Octave will need to provide a stable, documented set of functions that can be used by external packages. Currently, all internal functions are available for use by external packages and this is a frequent source of trouble as Octave's internals continue to evolve[15]. By providing a stable interface for packages to use, we hope avoid most of these problems in the future.

3.2 Longer Term

In the longer term, we see the following projects as potentially useful improvements.

3.2.1 Package System

Paul Kienzle has been maintaining a collection of contributed functions and information about Octave for several years. It is now available on the web at `http://octave.sf.net`. One of the original motivations for for the project was to collect functions that improved Octave's capabilities, particularly in the area of MATLAB-compatibility. The external repository functions as a central repository for contributed code, where it can be evaluated and, in some cases, merged with the core Octave distribution.

Eventually, we hope to have a package system (perhaps similar to, or even sharing code with, the one used by R). Once that happens, we will be able to move some code out of the core Octave distribution and into external packages such as the Octave Forge collection.

3.2.2 Specialized Matrix Types

For memory and computational efficiency, it would be useful to include banded, triangular, symmetric, or other specialized matrix types. These should not be too hard to add given Octave's object-oriented structure, though we will probably need some method to handle the explosion in the number of new functions for mixed-type operations that will need to be defined. Some clever use of templates may help here, but I still see this problem as unsolved.

[15]Even though this may cause trouble from time to time, this is a feature we would like to preserve, at least for wizards.

3.2.3 Improved Graphics and GUI Tools

Recently, Ole Jacob Hagen has started work on a new graphics engine for Octave that aims to be compatible with MATLAB's Handle Graphics. This work is part of the Zherlock[16] project that aims to create data analysis software that will allow the user to access advanced methods in statistics, chemometrics and artificial intelligence through an easy-to-use graphical user interface.

3.2.4 Multithreading

If Octave is to have programmable GUI features, it will probably need to have some multithreading capability to allow the GUI to continue functioning while a long calculation takes place. This modification will be a challenge given that the current implementation relies on a number of global variables to maintain the state of the interpreter.

4 Summary

In the two years since DSC-2001, Octave's user group and the number of interesting and useful contributions has grown significantly. A small but active group of users participate regularly on the mailing lists, answering questions and contributing enhancements and bug fixes. New contributors arrive on the scene regularly, and we look forward to this trend continuing. The pessimism expressed two years ago now seems unwarranted.

During the same time, we have been working toward improved compatibility with MATLAB while still keeping an eye on innovation. We are convinced that Octave will continue to improve and attract a wider audience.

Acknowledgment

The authors gratefully acknowledge the financial support of the industrial members of the Texas-Wisconsin Modeling and Control Consortium and NSF through grant #CTS-0105360.

References

K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Elsevier Science Publishers, New York, 1989.

P. N. Brown, A. C. Hindmarsh, and L. R. Petzold. Using Krylov methods in the solution of large-scale differential-algebraic systems. *SIAM Journal of Scientific Computing*, 15: 1467–1488, 1994.

[16] `http://www.zherlock.org`

John W. Eaton. Octave: Past, present and future. In Kurt Hornik and Friedrich Leisch, editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15-17, 2001, Technische Universität Wien, Vienna, Austria*, 2001. URL `http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/`. ISSN 1609-395X.

Alan C. Hindmarsh. ODEPACK, a systematized collection of ODE solvers. In R. S. Stepleman, editor, *Scientific Computing*, pages 55–64, Amsterdam, 1983. North-Holland.

Jorge R. Leis and Mark A. Kramer. The simultaneous solution and sensitivity analysis of systems described by ordinary differential equations. *ACM Transactions on Mathematical Software*, 1988.

S. Li and L. R. Petzold. Software and algorithms for sensitivity analysis of large-scale differential-algebraic systems. to appear, Journal of Computational and Applied Mathematics, 2003. URL `http://www.engineering.ucsb.edu/~cse`.

L. R. Petzold. A description of DASSL: A differential-algebraic system solver. In R. S. Stepleman, editor, *Scientific Computing*, pages 65–68, Amsterdam, 1983. North-Holland.

James B. Rawlings and John G. Ekerdt. *Chemical Reactor Analysis and Design Fundamentals.* Nob Hill Publishing, Madison, WI, 2002.