Matthew Brewster

Study of free alternative numerical computation packages

Comment [MM1]:
Comments/typos from the reviewer:

1) Specify the relation between h and N (page 5).
2) Describe the abbreviation and the use of function such us ``cg, pcg, eig, spec''
(e.g., ``eig'' and `spec'' are functions for computing the eigenvalues of a square matrix...).
3) On page 5, line 8 (from the top) ``we get obtain equations''
4) Page 7, line 6 (from the bottom) ``this shows that are code is working correctly''.

There are additional comments below.

Abstract:

Matlab is the most popular commercial package for numerical computations in mathematics, statistics, the sciences, engineering, and other fields. Octave, FreeMat, and Scilab are free numerical computational packages that have many of the same features as Matlab. They are available for free download under the Linux, Windows, and Mac OS X operating systems. We investigate whether these packages are viable alternatives to Matlab for uses in research and teaching. We compare the results on the cluster tara in the UMBC High Performance Computing Facility. Based on our tests, we conclude that Octave is the best viable alternative to Matlab because it was not only fully compatible with Matlab in our tests, but it also performed very well in the complex test cases, out-performing FreeMat and Scilab by a considerable margin.

Introduction:

There are several numerical computational packages that serve as educational tools and are also available for commercial use. Matlab (www.mathworks.com) is the most widely used such package. Matlab is being used in many courses at UMBC from wide range of disciplines, but many of the assignments in the courses only access basic features of the software and require limited computer power, so a free alternative that can be used on a laptop or home computer would be of great value to students. The focus of this study is to introduce three additional free numerical computational packages: Octave (www.octave.org), FreeMat (www.freemat.org), and Scilab (www.scilab.org), and provide information on which package is most compatible to Matlab users. To evaluate Octave, FreeMat, and Scilab, a comparative approach is used based on a Matlab user's perspective. To achieve this task, we perform some basic and some complex studies on Matlab, Octave, FreeMat, and Scilab. The basic

studies include basic operations solving systems of linear equations, computing the eigenvalues and eigenvectors of a matrix, and two-dimensional plotting. The complex studies include direct and iterative solutions of a large sparse system of linear equations resulting from finite difference discretization of an elliptic test problem. The more complex test case is thus designed to give a feel for a research problem. Clearly, the use of alternatives assumes that the user's needs are limited to the basic functionalities of Matlab itself. Matlab does have a very rich set of toolboxes for a large variety of applications or for certain areas with more sophisticated algorithms. If the use of one of them is profitable or integral to the research, the other packages are likely not viable alternatives.

More details of this work on comparing Matlab and its alternatives on a research computer at UMBC are available in the HPCF tech. report [Coman et al. 2012], which also considered other packages such as R and IDL, which are not intended to be similar to Matlab and not necessarily free. This paper specifically focuses on free packages with a high degree of compatibility to Matlab and organizes the results for a direct comparison. Earlier work including [Brewster and Gobbert 2011] used a matrix-free implementation of the conjugate gradient method that makes the results less directly applicable for most users. Additionally, other work compared the software packages in a home computer setting [Sharma 2010; Sharma and Gobbert 2010], which makes it less reproducible for others at UMBC.

The computations for this study are performed using Matlab R2012a, Octave 3.6.2, FreeMat v4.0, and Scilab-5.3.1 under the Linux operating system Redhat Enterprise Linux 5. To have a central reference point, the cluster tara in the UMBC High Performance Computing Facility was used to carry out the computations, which is available to all interested users at UMBC for research. The cluster tara has 86 nodes, each with two quad-core Intel Nehalem processors (2.66 GHz, 8MB cache) and 24GB of memory, but only one node – equivalent to a desktop computer – is used. To learn more about the tara cluster in the High Performance Computing Facility (HPCF) visit http://www.umbc.edu/hpcf/.

Basic Operations Test

In this section we perform the basic operations test using Matlab, Octave, FreeMat, and Scilab. One of the basic functionalities of these software packages is the capability to solve a system of linear equations by Gaussian elimination. For example we will consider the following a system of linear system of equations

$$-u_2 + u_3 = 3,$$
$$u_1 - u_2 - u_3 = 0,$$
$$-u_1 - u_3 = -3,$$

where the solution to this system is $(1, -1, 2)$. In Matlab to solve this system let us express this linear system as a single matrix equation

$$Au = b,$$

where A is a square matrix consisting of the coefficients of the unknowns, u is the vector of unknowns, and b is the right-hand side vector. For the particular system we have

$$A = \begin{bmatrix} 0 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}, \quad \text{b} = \begin{bmatrix} 3 \\ 0 \\ -3 \end{bmatrix}$$

To find a solution *u* for this system in Matlab, the matrix A and vector b are entered using the commands

```
A = [0 -1 1; 1 -1 -1; -1 0 -1]
b = [3; 0; -3]
```

Now use the backslash operator \ to call Gaussian elimination to solve the linear system and find the vector u by calling `u = A\b` to solve this system. The resulting vector which is assigned to u is

$$u = \begin{matrix} 1 \\ -1 \\ 2 \end{matrix}$$

The backslash operator works identically for all of the packages to produce a solution to the linear system given and is an example of excellent compatibility of all packages.

We also investigate some other basic operations in these numerical computation packages and examine the commands needed the execute them [Brewster and Gobbert 2011; Coman et al. 2012]. The command **eig** has the same functionality in Octave and FreeMat as in Matlab for computing eigenvalues and eigenvectors, whereas Scilab uses the equivalent command **spec** to compute them. Plotting is another important feature we analyze by an m-file containing the two-dimensional plot function along with some common annotations commands. Once again, Octave and FreeMat use the exact commands for plotting and similar for annotating as Matlab whereas Scilab requires a few changes. For instance in Scilab, the number π is defined using **%pi** instead of simply **pi** as in Matlab, and the command **grid** from Matlab is replaced with **xgrid**. To overcome these conversions, we find that we can use the Matlab to Scilab translator, which takes care of these command differences for the most part. For instance, the translator is unable to convert the **xlim** command from Matlab to Scilab. To rectify this, we must manually specify the axis boundaries in Scilab using additional commands in **Plot2d**. This issue brings out a major concern that despite the existence of the translator, there are some functions that require manual conversion.

Complex Test Problem

This section tests the performance of the software packages using a classical complex test problem [Demmel 1997; Watkins 2010] from partial differential equations that puts a strain on the code both in terms of execution speed and memory consumption. The Poisson problem with homogeneous Dirichlet boundary conditions is given as

$$-\Delta u = f \text{ in } \Omega ,$$

$$u = 0 \text{ on } \partial\Omega.$$

We consider this problem on the two dimensional unit square $\Omega = (0,1) \times (0,1) \subset \mathbb{R}^2$ where the function f is given by

$$f(x,y) = -2\pi^2 \cos(2\pi x) \sin^2(\pi y) - 2\pi^2 \sin^2(\pi x) \cos(2\pi y).$$

The test problem solution is designed to admit a closed-form solution as the true solution

$$u(x,y) = \sin^2(\pi x)\sin^2(\pi y).$$

By applying the second-order finite difference approximation we get obtain equations that can be organized into a linear system of dimension N^2

$$Au = b$$

with system matrix A that is symmetric positive definite. The theory of the finite difference method [Braess 2007; Iserles 2009] tells us that the norm of the error $u - u_h$ behaves like

$$\|u - u_h\|_{\infty(\Omega)} \leq C h^2$$

as the mesh width h tends to zero, $h \to 0$. We can use this theoretical result to predict how the norm of the error is expected to behave, as the mesh width decreases for finer and finer meshes: Whenever the mesh width is halved by a refinement of the mesh, the ratio of errors on consecutively refined meshes approaches 4.

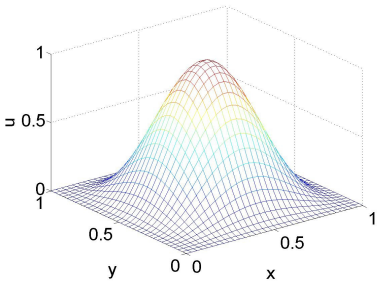
To create the matrix A, we use the observation that it is given by a sum of two Kronecker products [Demmel 1997]: Namely, A can be interpreted as the sum

$$A = \begin{bmatrix} T & & & \\ & T & & \\ & & \ddots & \\ & & & T \\ & & & & T \end{bmatrix} + \begin{bmatrix} 2I & -I & & & \\ -I & 2I & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & 2I & -I \\ & & & -I & 2I \end{bmatrix} \in \mathbb{R}^{N^2 \times N^2},$$

where

$$T = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{N \times N}$$

and I is the $N \times N$ Identity matrix, and each of the matrices in the sum can be computed by Kronecker products involving T and I, so that $A = I \otimes T + T \otimes I$. To store the matrix A efficiently, all packages provide a sparse storage mode, in which only the non-zero entries are stored.



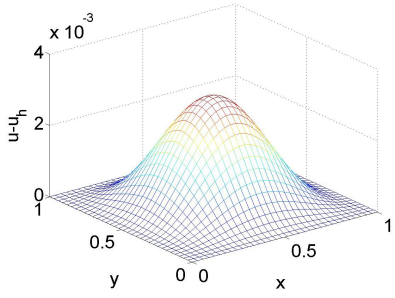


Figure 1: Numerical solution. **Figure 2:** Numerical error.

Comment [MM2]: Can you plot the ratio of numerical error to numerical solution? This would give you visually a feeling of how the relative error behaves in your grid.

Figure 1 shows the mesh plot of the numerical solution vs. (x,y), while **Figure 2** shows the error at each mesh point, which is computed by subtracting the numerical solution from the analytical solution; notice the scale of the vertical axis. We can see that the maximum error is attained at the center of the domain in the x-y-plane and has a value of approximately 3×10^{-3}.

We now want to solve the Poisson problem on finer meshes with mesh resolutions $N = 2^v$ for $v = 1, 2, 3 \ldots, 13$ in order to obtain a more precise solution. The results of this are summarized in **Table 1**, which lists the mesh size of the discretization $N \times N$, the dimension of the linear system N^2, the norm of the finite difference error $\|u - u_h\|$, and the ratio of the error norms when doubling the mesh resolution (i.e., practically speaking from one row of the table to the next).

Table 1: Convergence of finite difference error.

$N \times N$	N^2	$\|u - u_h\|$	Ratio
32×32	1,024	3.0128e-3	N/A
64×64	4,096	7.7811e-4	3.8719
128×128	16,384	1.9765e-4	3.9368
256×256	65,536	4.9797e-5	3.9690
512×512	262,144	1.2494e-5	3.9856
$1{,}024 \times 1{,}024$	1,048,576	3.1266e-6	3.9961
$2{,}048 \times 2{,}048$	4,194,304	7.8019e-7	4.0075
$4{,}096 \times 4{,}096$	16,777,216	1.9353e-7	4.0313
$8{,}192 \times 8{,}192$	67,108,864	4.6797e-8	4.1355

In the first row for the resolution 32×32, we note first that the norm of the finite difference error of 3.0128×10^{-3} confirms the approximate observation from the plot in **Figure 2**. The entries for the norm of the finite difference error in **Table 1** show that the error is converging toward zero and the ratio of the consecutively refined meshes is approaching 4. Because the ratio and finite difference error are behaving correctly in agreement with the finite difference theory, this shows that are code is working correctly.

The previous table focused on the numerical results of the solution to the Poisson problem. We now turn to comparing the performance of the four software packages in obtaining these results. **Table 2** lists the mesh resolution N, the dimension of the linear system N^2, and the observed wall clock time in hours:minutes:seconds for the different software packages, when solving the problem with Gaussian

elimination. The notation O.M. for an entry indicates that the code ran out of memory in that case and could not solve the problem. For each mesh resolution, we carefully checked the numerical results and found them to be identical for all packages in all cases, for which a solution was obtained. It is thus appropriate to focus on comparing the performance of the packages.

Table 2: Performance of Gaussian elimination.

Comment [MM3]: Can you plot on a single graph the log(time) vs log(N)?

$N \times N$	N^2	Matlab	Octave	FreeMat	Scilab
32×32	1,024	< 00:00:01	<00:00:01	<00:00:01	<00:00:01
64×64	4,096	<00:00:01	<00:00:01	<00:00:01	<00:00:01
128×128	16,384	<00:00:01	<00:00:01	<00:00:01	00:00:11
256×256	65,536	<00:00:01	<00:00:01	00:00:04	00:03:19
512×512	262,144	00:00:01	00:00:02	00:00:28	00:39:04
$1{,}024 \times 1{,}024$	1,048,576	00:00:05	00:00:16	00:03:15	09:09:32
$2{,}048 \times 2{,}048$	4,194,304	00:00:23	00:01:57	00:14:29	O.M.
$4{,}096 \times 4{,}096$	16,777,216	00:01:50	00:15:37	O.M.	O.M.
$8{,}192 \times 8{,}192$	67,108,864	O.M.	O.M.	O.M.	O.M.

By looking at **Table 2**, it can be concluded that the Gaussian elimination method built into the backslash operator successfully solves the problem up to a mesh resolution of $4{,}096 \times 4{,}096$ in both Matlab and Octave. While the Gaussian elimination method built into the backslash operator in FreeMat successfully solves the problem up to a mesh resolution of $2{,}048 \times 2{,}048$, in Scilab, it is only able to solve up to a mesh resolution of $1{,}024 \times 1{,}024$. The wall clock results show that Matlab was faster than Octave, FreeMat, and Scilab. Octave was faster and was able to solve a larger mesh resolution than both FreeMat and Scilab. Scilab was the slowest and could not solve the same mesh resolution as the other packages.

We see that none of the packages considered were able to solve the problem on an $8{,}192 \times 8{,}192$ mesh. The need to solve larger systems leads us to another method known as conjugate gradient

method to solve the linear system. This iterative method is an alternative to using Gaussian elimination to solve a linear system with a symmetric positive definite system matrix, such as the given matrix. We use the zero vector as the initial guess and a tolerance of 10^{-6} on the relative residual of the iterates. **Table 3** lists the mesh resolution $N \times N$, the dimension of the linear system N^2, the number of iterations taken by the iteration method to converge (#iter), and the observed wall clock times in hours:minutes:seconds for the different software packages. For each mesh resolution, we again carefully compared the numerical results and found them to be equivalent among the packages as well as equivalent to the results obtained by Gaussian elimination in all cases, where a solution was obtained.

Table 3: Performance of the conjugate gradient method.

Comment [MM4]: Can you plot on a single graph the log(time) vs log(N)?

$N \times N$	N^2	#iter	Matlab	Octave	FreeMat	Scilab
32×32	1,024	48	<00:00:01	<00:00:01	<00:00:01	<00:00:01
64×64	4,096	96	<00:00:01	<00:00:01	00:00:02	<00:00:01
120×128	16,384	192	<00:00:01	<00:00:01	00:00:17	<00:00:01
256×256	65,536	387	00:00:02	00:00:02	00:02:29	00:00:02
512×512	262,144	783	00:00:12	00:00:14	00:21:16	00:00:22
$1{,}024 \times 1{,}024$	1,048,576	1,581	00:01:34	00:01:56	02:59:08	00:03:19
$2{,}048 \times 2{,}048$	4,194,304	3,192	00:12:42	00:17:50	E.T.R	00:26:57
$4{,}096 \times 4{,}096$	16,777,216	6,452	01:41:10	02:34:29	E.T.R.	O.M.
$8{,}192 \times 8{,}192$	67,108,864	13,033	13:43:55	20:01:27	E.T.R.	O.M.

Table 3 shows that conjugate gradient method is indeed able to solve for mesh resolutions as large, or larger, than those solved Gaussian elimination for each package. The sparse matrix storage implementation of the conjugate gradient method allows us to solve a mesh resolution up to $8{,}192 \times 8{,}192$ for Matlab and Octave. Scilab is able to solve the system for a resolution up to $4{,}096 \times 4{,}096$. In FreeMat, we wrote our own **cg** function because it does not have a built in **pcg** function and we were able to solve the system for a resolution of $2{,}048 \times 2{,}048$ within a reasonable

amount of time; the notation E.T.R. indicates excessive time requirements of over 5 days for that case. The wall clock times show that Matlab was faster than Octave, but Octave was faster than both FreeMat and Scilab. They also show that FreeMat was slower than Octave, Matlab, and Scilab and was not able to solve as large of a system before the time required to solve the problem became excessively long. Scilab performed better than FreeMat, but it ran out of memory in fact for the finest resolution attempted. We see that only Matlab and Octave were in fact able to obtain the solution on all desired meshes.

Conclusions:

We tested the four software packages Matlab, Octave, FreeMat, and Scilab for two criteria: usability and performance.

The software package's usability was determined by comparing the syntax and functions to Matlab. We will consider a package more usable, the more similar its syntax is to Matlab. Octave was determined to be the most usable because the commands and syntax it used were compatible with Matlab for all of our tests. Scilab exhibited the most differences in both syntax and commands. For example instead of using the **eig** function like Matlab, Octave, and FreeMat to compute eigenvalues Scilab uses a function called **spec**.

To test the performance of the software packages we used the Gaussian elimination and conjugate gradient methods to solve the Poisson equation. The results from **Table 2** reveal that Matlab performed the best when solving the system via Gaussian elimination. Octave performed the best out of all the free software packages tested and was able to solve the same size systems as Matlab. Scilab's backslash operator was much slower, in compared to Matlab and Octave and was the least powerful. The results from **Table 3** reveal that Matlab, Octave, and Scilab were all able to solve the system comparably fast, but Scilab was not able to solve as large a system as Octave or Matlab. FreeMat was

the weakest and could not solve the system for mesh resolutions larger mesh resolutions without requiring an excessive amount of time.

In summary, FreeMat and Scilab are far less compatible with Matlab in usability and performance. However, Matlab and Octave appear fully compatible in their syntax and availability of commands, and our tests demonstrate that Octave is slower only in one test, but when absolute run times are considered, this becomes only a problem for very large problems. Additionally, Octave is of interest, since it is known to work with a free distributed-memory parallel extension pMATLAB [Kepner 2009]. The complex test problem used here is a classical test problem also for parallel computing [Raim and Gobbert 2010; Sharma and Gobbert 2009]. In the future we hope to continue testing Octave and its parallel extensions to further analyze its performance and capabilities.

Acknowledgments

The author acknowledges financial support from the Department of Mathematics and Statistics at UMBC. The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (www.umbc.edu/hpcf). The facility is supported by the U.S. National Science Foundation, with additional substantial support from UMBC.

References:

Dietrich Braess. *Finite Elements*. Cambridge University Press, third edition, 2007.

Matthew Brewster and Matthias K. Gobbert. A comparative evaluation of Matlab, Octave, FreeMat, and Scilab on tara. Technical Report HPCF-2011-10, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2011. www.umbc.edu/hpcf.

Ecaterina Coman, Matthew W. Brewster, Sai K. Popuri, Andrew M. Raim, and Matthias K. Gobbert.

A comparative evaluation of Matlab, Octave, FreeMat, Scilab, R, and IDL on tara. Technical Report HPCF-2012-15, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2012. www.umbc.edu/hpcf.

James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, second edition, 2009.

Jeremy Kepner. *Parallel MATLAB for Multicore and Multinode Computers*. SIAM, 2009.

Andrew M. Raim and Matthias K. Gobbert. Parallel performance studies for an elliptic test problem on the cluster tara. Technical Report HPCF-2010-2, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2010. www.umbc.edu/hpcf.

Neeraj Sharma. A *comparative study of several numerical computational packages*. M.S. thesis, Department of Mathematics and Statistics, University of Maryland, Baltimore County, 2010.

Neeraj Sharma and Matthias K. Gobbert. Performance studies for multithreading in Matlab with usage instructions on hpc. Technical Report HPCF-2009-1, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2009. www.umbc.edu/hpcf.

Neeraj Sharma and Matthias K. Gobbert. A comparative evaluation of Matlab, Octave, FreeMat, and Scilab for research and teaching. Technical Report HPCF-2010-7, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2010. www.umbc.edu/hpcf.

David S. Watkins. *Fundamentals of Matrix Computations*. Wiley, third edition, 2010.