MATTHEW **BREWSTER**

Matthew Brewster is a junior majoring in mathematics with a minor in environmental science. He hopes to continue his education studying environmental science. Matthew is a member of the Undergraduate Training Program in Biology and Mathematics (UBM) and has participated in the Interdisciplinary Program in High Performance Computing at UMBC. Matthew is also a member of several honor societies. He would like to thank his advisor, Dr. Matthias Gobbert, for his guidance, patience, and encouragement during the course of this project. He would also like to thank his friends, family, and everyone who supported him throughout his education at UMBC.

ALTERNATIVES TO THE MATHEMATICAL SOFTWARE PACKAGE MATLAB IN RESEARCH AND EDUCATION

AS A NEW FRESHMAN, I ENROLLED IN DR. NAGARAJ NEERCHAL'S First Year Seminar, Crime Busting with Mathematics and Statistics. Dr. Neerchal suggested that I contact Dr. Matthias Gobbert about research opportunities. The following summer, I applied to the Interdisciplinary Program in High Performance Computing at UMBC, and wrote a technical report on my comparative analysis of MATLAB, Octave, FreeMat, and Scilab. A year later, I took Introduction to Parallel Computing with Dr. Gobbert, and learned the basics of parallel computing and how it is applied to numerical algorithms. I also helped extend my technical report to R and IDL, along with Sai Popuri and Oana Coman. In addition to my research with Dr. Gobbert, I am involved in the UBM Program as part of the Erill-Gobbert group for bioinformatics research.

INTRODUCTION

THERE ARE SEVERAL NUMERICAL COMPUTATIONAL packages that serve as educational tools and are available for commercial use. MATLAB (www.mathworks.com) is the most widely used of these packages. MATLAB is used in many courses at UMBC and in a wide range of disciplines, but many of these courses only utilize the basic features of the software that require limited computer power. It would be valuable to students to have a free alternative with the necessary features that can be used on a laptop or home computer.

This study examines three free numerical computational packages: Octave (www.octave.org), FreeMat (www.freemat.org), and

Scilab (www.scilab.org), and analyzes the compatibility of each package with MATLAB. We use a comparative approach to the packages based on a MATLAB user's perspective. We perform both basic and complex studies on each of the four software packages. The basic studies include simple operations such as solving systems of linear equations, computing the eigenvalues and eigenvectors of a matrix, and plotting two-dimensional data. The complex studies involve direct and iterative solutions of a large sparse system of linear equations resulting from finite difference discretization of an elliptic test problem. The complex test case is designed to emulate a practical research problem. This study assumes that the needs of a typical student user are limited to the basic functionalities of MATLAB. Although MATLAB has a rich set of toolboxes for more sophisticated algorithms, this study does not consider alternatives to MATLAB for research or applications that require these features.

Similar work on comparing MATLAB and its alternatives on a research computer at UMBC are available in the HPCF Technical Report [Coman et al. 2012], which also considers 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. Previous work used a matrix-free implementation of the conjugate gradient method [Brewster and Gobbert 2011] that makes the results less directly applicable for most users. Additionally, earlier work compared the software packages in a home computer setting [Sharma 2010; Sharma and Gobbert 2010], which makes it less reproducible for users working from a school computer or laptop.

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. The same server in the UMBC High Performance Computing Facility was used to carry out all computations to maintain consistency in the results. The server, named cluster tara, has 86 nodes, each with two quad-core Intel Nehalem processors (2.66 GHz, 8 MB cache) and 24 GB of memory, but only one node – equivalent to a desktop computer – is used for this study.

BASIC OPERATIONS TEST

IN THIS SECTION, we describe 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 consider the following system of linear system of equations,

$$\begin{aligned} -u_2 + u_3 &= 3, \\ u_1 - u_2 - u_3 &= 0, \\ -u_1 \qquad - u_3 &= -3, \end{aligned}$$

where the solution to this system is $(1,-1,2)^T$. To solve this system In MATLAB, let us express the 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

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

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

$$A = [0\ -1\ 1;\ 1\ -1\ -1;\ -1\ 0\ -1]$$

$$b = [3;\ 0;\ -3].$$

The backslash operator \ invokes Gaussian elimination to solve the linear system (i.e. find the vector u) by calling $u = A \backslash b$. The resulting vector, which is assigned to u, is

$$u = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}.$$

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

We also investigate 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 using an m-file containing the two-dimensional plot function, along with some common annotations commands. Once again, Octave and FreeMat use the same commands as MATLAB for plotting, and similar commands for annotating, 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** that adds grid lines to the plot from MATLAB is replaced with **xgrid**. To overcome these conversions, the MATLAB-to-Scilab translator can be used, which largely takes care of these command differences. However, the translator is unable to convert the **xlim** command, which returns the limits of the current axes, from MATLAB to Scilab. To rectify this, the axis boundaries must be manually specified in Scilab using additional commands in **Plot2d**. This issue brings out a major concern: 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:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega\,, \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

We consider this problem on the two-dimensional unit square $\Omega = (0,1) \times (0,1) \in \mathbb{R}^2$ 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

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

Let us define a grid of mesh points with resolution $(N+2) \times (N+2)$, with mesh spacing $h = 1/(N+1)$. By applying the second-order finite difference approximation, we 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

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

and as the mesh width h tends to zero, $h \longrightarrow 0$. 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 four.

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

$$A = \begin{bmatrix} T & & & & \\ & T & & & \\ & & \ddots & & \\ & & & T & \\ & & & & T \end{bmatrix} + \begin{bmatrix} 2I & -I & & & \\ -I & 2I & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & 2I & -I \\ & & & -I & 2I \end{bmatrix} \in \mathbb{R}^{N2 \times N2}$$

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

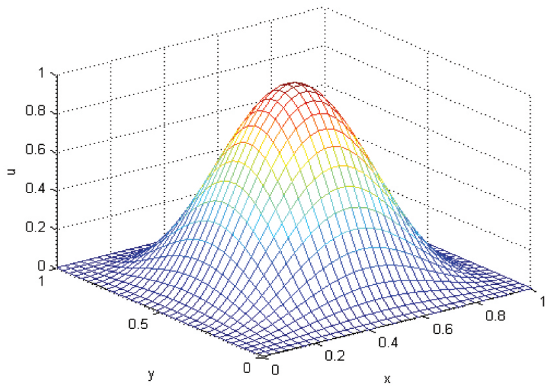


FIGURE 2: Numerical error

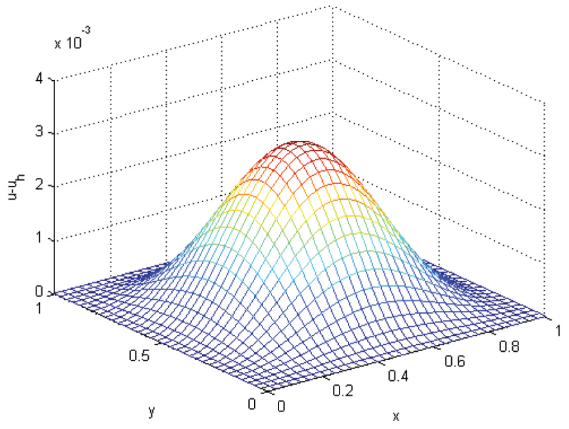


Figure 1 shows the mesh plot of the numerical solution versus *(x,y)*, while Figure 2 shows the error at each mesh point, which is computed by subtracting the numerical solution from the analytical solution. Notice the different scales for each vertical axis. The maximum error is attained at the center of the domain in the x-y plane and has a value of approximately 3×10^{-3}.

We 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 test are summarized in Table 1, which lists the mesh size of the discretization $N \times N$, the dimension of the

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
1,024×1,024	1,048,576	3.1266e-6	3.9961
2,048×2,048	4,194,304	7.8019e-7	4.0075
4,096×4,096	16,777,216	1.9353e-7	4.0313
8,192×8,192	67,108,864	4.6797e-8	4.1355

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).

In the first row, for the resolution 32×32, we note 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 four. The ratio and finite difference error behave in agreement with the finite difference theory, indicating that the code is working correctly. The previous table focused on the numerical results of the solution to the Poisson problem.

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 and could not solve the problem. For each mesh resolution, we found the numerical results to be identical for all packages, in all cases for which a solution was obtained. It is therefore appropriate to compare the performance of the packages.

TABLE 2: PERFORMANCE OF GAUSSIAN ELIMINATION

$N \times N$	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
1,024×1,024	1,048,576	00:00:05	00:00:16	00:03:15	09:09:32
2,048×2,048	4,194,304	00:00:23	00:01:57	00:14:29	O.M.
4,096×4,096	16,777,216	00:01:50	00:15:37	O.M.	O.M.
8,192×8,192	67,108,864	O.M.	O.M.	O.M.	O.M.

Table 2 shows that the Gaussian elimination method built into the backslash operator successfully solves the problem up to a mesh resolution of *4,096* × *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* × *2,048*, in Scilab it is only able to solve up to a mesh resolution of *1,024* × *1,024.* The wall clock results show that MATLAB was faster than Octave, FreeMat, and Scilab. Octave was faster than both FreeMat and Scilab, and was able to solve a larger mesh resolution. 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 with an *8,192* × *8,192* mesh. The desire 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 each software package. For each mesh resolution, we again found the numerical results 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

$N \times N$	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
128×128	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
1,024×1,024	1,048,576	1,581	00:01:34	00:01:56	02:59:08	00:03:19
2,048×2,048	4,194,304	3,192	00:12:42	00:17:50	E.T.R	00:026:57
4,096×4,096	16,777,216	6,452	00:41:10	02:34:29	E.T.R	O.M.
8,192×8,192	67,108,864	13,033	13:43:55	20:01:27	E.T.R	O.M.

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

E.T.R. indicates excessive time requirements of over five days. The wall clock times show that MATLAB was faster than Octave, but Octave was faster than both FreeMat and Scilab. The times also show that FreeMat was slower than Octave, MATLAB, and Scilab, and was not able to

FIGURE 3: Log*(N)* vs. Log(Time) for Gaussian elimination

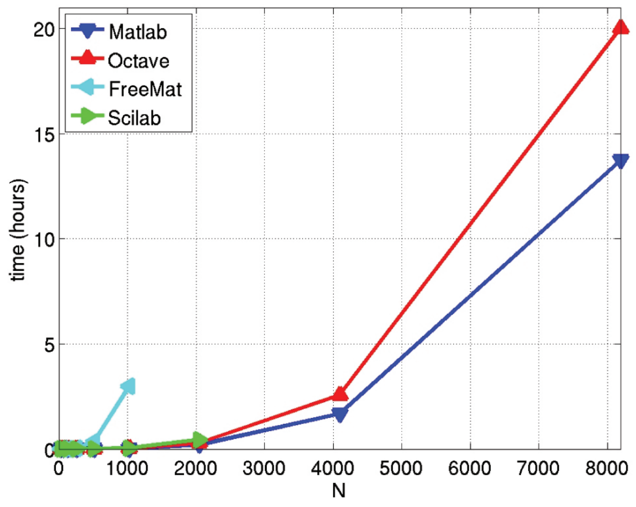
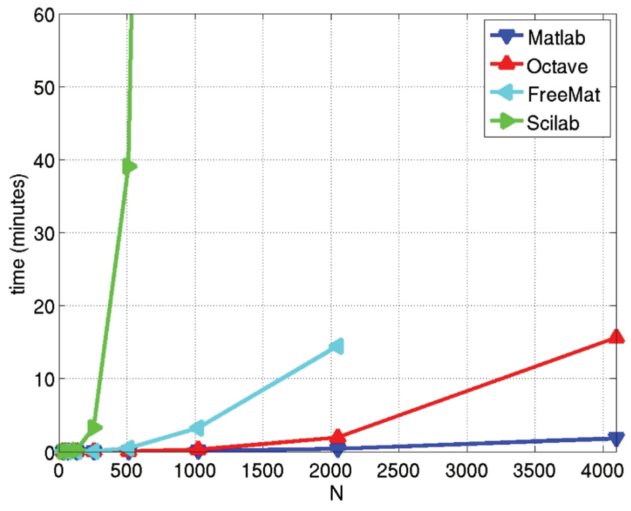


FIGURE 4: Log*(N)* vs. Log(Time) for Conjugate Gradient Method



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 for the finest resolution attempted. We see that only MATLAB and Octave were able to obtain the solution on all desired meshes. In Figures 3 and 4, log(time) is plotted against log(N) for the data reported in Tables 2 and 3, respectively, to see how the run times for each software package are affected by the mesh size for both Gaussian elimination and the conjugate gradient method. Figure 3 shows the results for Gaussian elimination. MATLAB was the fastest of all of the numerical computation packages with a time of about one hour for the largest mesh of $N \times N = 4{,}096 \times 4{,}096$. Octave was somewhat slower than MATLAB, but could solve a problem of the same size. The plots reveal that FreeMat and Scilab were much slower and not able to solve problems of the same sizes as MATLAB and Octave. In fact, Scilab was so slow that its line goes far beyond the limit of the plot already for a mesh of $1{,}024 \times 1{,}024$.

Figure 4 displays the results for the conjugate gradient method. The plot shows that MATLAB and Octave can solve the problem for the largest mesh of $N \times N = 8{,}192 \times 8{,}192$, but MATLAB is somewhat faster than Octave. FreeMat and Scilab could not solve problems on the same meshes as MATLAB and Octave, and were somewhat slower on the meshes they were able to solve. Overall, the different scales of both axes in Figures 3 and 4 highlight that the absolute run times are larger for the conjugate gradient method than for Gaussian elimination, but that the conjugate gradient method allowed for larger meshes than Gaussian elimination.

CONCLUSIONS AND FUTURE WORK

WE TESTED THE usability and performance of four software packages: MATLAB, Octave, FreeMat, and Scilab. The usability of each software package was determined by comparing its syntax and functions to MATLAB. A package is considered more usable when its syntax is similar to the syntax in MATLAB. Octave was determined to be the most usable because its commands and syntax 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 to compute eigenvalues like MATLAB, Octave, and FreeMat, Scilab uses a function called **spec**.

To test the performance of the software packages, Gaussian elimination and conjugate gradient methods were used 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 better than the other free software packages tested and was able to solve the same size systems as MATLAB. The backslash operator in Scilab was much slower than in MATLAB and Octave, and was also the least powerful. The results from Table 3 reveal that MATLAB, Octave, and Scilab were all able to solve the system with comparable speed, 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 larger mesh resolutions without requiring an excessive amount of time.

In summary, FreeMat and Scilab are far less compatible with MATLAB, and do not compare in regards to usability and performance. However, MATLAB and Octave appear fully compatible in their syntax and availability of commands. Our tests demonstrate that Octave is slower only in one test, and when absolute run times are considered, the performance difference is only an issue for very large problems.

Octave is of particular 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 National Science Foundation, with additional substantial support from UMBC.

REFERENCES

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

2. 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.

3. 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.

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

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

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

7. 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.

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

9. 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.

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

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