

Using MATLAB to Encourage Formation of Conjectures by Students

Jeffrey L. Stuart*

February 28, 1994

Abstract

Whereas MATLAB is often used in courses as a matrix calculator, we examine how it can be used as a matrix laboratory. In particular, by utilizing a variety of random matrix generators to produce large sets of structured, random matrices, and by employing simple test functions that call these generators, we show how students can be led to formulate conjectures about basic matrix functions and about some classical theorems in matrix theory.

1 Introduction

When scientific calculators first appeared in the mid 1970's, the initial focus of their use was on calculating specific values of the basic scientific functions. Scientific calculators were viewed as tools that vastly enhanced the ability of students to do calculations. The wide-spread use of calculators enabled instructors and textbook authors to worry less about constructing clever examples for which the function values worked out to "nice" answers. As users became more accustomed to the abilities of such calculators, and as the novelty of simply being able to do such computations as $\sin(\sqrt{2})$ wore off, people began to experiment with how scientific calculators could be used as a teaching tool rather than as an *idiot savant*. One typical application of scientific calculators in this latter role was in the study of limits. A representative example of this use would be the estimation of $\lim_{x \rightarrow 0} \frac{\sin x}{x}$ by evaluating the function for values of x arbitrarily close to zero. This example is particularly instructive because it provides a motivation for using radians, a system of measurement that most beginning calculus students find to be completely alien.

During the last decade, the personal computer revolution and the concomitant development of software for matrix computations has created a situation in linear algebra education that parallels the situation in trigonometry and calculus education a decade earlier. Many instructors expect their students to have access to some form of programs for performing matrix operations. As is to be expected with a new tool, the initial applications have focused on computations - products, powers, eigenvalues. Matrix computation software has liberated instructors from having to carefully craft examples that have nice eigenvalues, nice null spaces, or some other artificially nice structure. Still, the focus of most textbook applications seems to be of the form: "Here is a particular matrix A , go compute its" What we are proposing here is that with a little more effort, matrix computation software can be used as a *laboratory* rather than merely as a calculator.

*Department of Mathematics, University of Southern Mississippi, Hattiesburg, Mississippi 39406 (jstuart@whale.cc.usm.edu).

While the focus of this article is on the use of MATLAB as a matrix laboratory, and while the particular coded functions are written for use in MATLAB, most of the ideas are immediately applicable to other software packages

2 MATLAB as a Matrix Calculator

For those who are not familiar with the MATLAB software package, which is produced and distributed by the Mathworks, Inc., we will review here some of its principal features as a basic, *numerical* matrix calculator. MATLAB provides all of the standard and many highly specialized scalar functions. It provides the matrix operations of addition, subtraction, multiplication (regular, kronecker and hadamard products), division (inverses and psuedoinverses), and integer powers. In addition, MATLAB provides an extensive suite of matrix oriented functions of which we only mention a few: transpose, determinant, trace, rank, norms, condition number, characteristic polynomial, reduced row echelon form, lu factorization, cholesky factorization, qr factorization, least squares solution for singular linear systems, null space basis, eigenvalues and eigenspace bases, singular value decomposition, matrix exponential, matrix logarithm and matrix square root. MATLAB also provides a variety of predefined matrix types, and an abundance of tools for analyzing data presented in an array format and for manipulating arrays to construct new arrays. In addition to its built-in functions and families of matrices, MATLAB has a simple script language that allows users to write their own functions, called *m-files*. In short, MATLAB provides a complete numerical calculator for all of the problems that a student might encounter in a first year, graduate level, numerical matrix theory course, and those tools are presented in such a way that even first semester, undergraduate linear algebra students should be able to use the basic tools comfortably and efficiently with minimal training.

It is not our purpose to reiterate the well-developed work of others on integrating MATLAB or any other numerical matrix software into the linear algebra curriculum as a calculator. Those interested in pursuing that subject might wish to consult any of the organizers of or participants in the ATLAST program, to review the guidelines of the Linear Algebra Curriculum Study Group, or to examine the new generation of linear algebra textbooks. We would strongly encourage anyone who is not currently using software in some fashion to reconsider his or her position. While good software does not make good students, it can relieve students of the drudgery of hand calculations in the same way that the scientific calculator does, freeing them up to see a little more of the forest and a little less of each tree.

3 MATLAB as a Matrix Laboratory

Consider the manner in which student laboratory experiments are structured in a traditional laboratory science such as physics. Students arrive in the lab with some background knowledge and a detailed set of directions written by the lab instructor. Using this information, students assemble the appropriate equipment, and run the experiment, varying specified input parameters, and recording the corresponding outputs. Subsequently, the students analyze their data, and attempt to form a relationship between the input parameters and the output. Or consider how a mathematician might use MATLAB as a testbed: initially he or she has an idea about some matrix property that can be measured numerically. After selecting an appropriate population of matrices to examine, he or she performs computations on a large number of examples, and searches the output either for a pattern or for a counterexample to the initial idea. If patterns are found, they suggest

a conjecture about the relationship between the selected population and the resultant computations.

If we are to use MATLAB as a matrix laboratory for our students, it would be helpful to closely follow the model suggested above. In particular, must carefully decide which aspects of the experiment are predetermined in the detailed set of instructions: which "equipment" is used (which functions and commands), and which parameters will be varied (which populations of matrices).

In subsequent sections, we will examine several experiments, we will consider how to select appropriate populations of matrices, and we will note several pitfalls.

3.1 An Experiment

The following experiment is one that can be assigned in a first semester, linear algebra course as soon as determinants have been introduced.

"Examine the output of the following m-file function *testdet(k,n)* for various choices of k and n . Explain what the m-file does at each loop. Based on the output vectors, make a reasonable conjecture about the determinant function. Can you prove your conjecture or provide a counterexample?"

```
testdet.m
function u=testdet(k,n)
u= zeros(1,n);
for j = 1:n
u(j) = det(rand(k,k));
end
```

Independent of the choices of k or n , the vector u will be strictly nonzero and real. Thus a reasonable student conjecture might be that $\det(A) \neq 0$ for all square matrices A . Presumably, better students will realize that this is false, since $\det(0_{k \times k}) = 0$.

This experiment highlights an important aspect of designing experiments: choosing an appropriate test population of matrices. The pitfall here is that the function *rand(k,k)* generates nonsingular matrices with probability approximately one.

3.2 Populations of Test Matrices

MATLAB is supplied with a number of predefined matrices. These include the identity matrix, the rectangular matrix of zeros, the rectangular matrix of ones, and the hilbert matrices. MATLAB has functions that use specified vectors to build Toeplitz, Hankel, and circulant matrices. In addition, MATLAB has a function, *rand(m,n)*, which returns an $m \times n$ matrix, each of whose entries is a random number uniformly distributed in the interval $(0,1)$. As the following two sample m-files, *randrank(m,n)* and *randpat(m,n,k,p)* indicate, it is easy to generate test populations of random matrices that display some type of desired structure.

```
randrank.m
function A = randrank(m,n)
% RANDRANK produces an m x n random matrix with random
% rank and entries in the interval (0,1).
k = ceil((min([m n])*rand(1)));
A = rand(m,k)*rand(k,n);
```

```

randpat.m
function A = randpat(m,n,p,q)
% RANDPAT produces an m × n random sign pattern matrix with entries
% in {-1,0,1} with zeros occurring with probability p, and with ones
% occurring among the nonzero terms with probability q.
A = (-ones(m,n)).^(round((rand(m,n)) - (q - 0.5)));
B = round((rand(m,n)) - (p - 0.5));
A = A.* B;

```

3.3 Experiments

One basic format for creating a laboratory experiment is to design the experiment around one or more m-files. Such an m-file, with an appropriate name such as *testfact.m*, would require a list of input parameters to be selected by the student. The m-file would contain a loop that creates a random matrix from the appropriate population based on the input parameters and then performs the intended computation on that matrix. Finally, the function would write an output vector from which the students should observe a pattern on which to base a conjecture. At a more advanced level, the choice of random matrix generators can be deferred to the student. The instructions for the experiment would prompt the student to explicitly identify what is being determined during each loop of the experiment and what information is presented in any outputs. The student would be prompted to make some choices, either of parameter values, or of families of structured random matrices. Finally, the student would be prompted to look for patterns, and to draw conclusions in the form of conjectures or possibly counterexamples.

As an example of this approach, consider two experiments using the m-file, *testpf.m* in the next section. First, the experiment could be run using the random matrix generator *rand(m,n)*, which generates entrywise positive matrices, and then it could be run again using the random matrix generator *randpat(m,n,p,q)*, which generates signed matrices. In the former case, students might conjecture that strictly positive matrices have a real eigenvalue that dominates the magnitudes of all other eigenvalues (one of the key results of the Perron Frobenius Theorem); in the latter case, students would observe that general real matrices do not have this property. Another conjecture that this experiment might prompt is that for real matrices, eigenvalues occur in conjugate pairs.

3.4 Another Experiment

”Run the following function for several choices of k and n . Explain what is being displayed at each iteration of the loop. What do you notice about the plots?. (Pay careful attention to the scale of the vertical axis in comparison to that of the horizontal axis.) Can you make any conjectures? Once you have made your conjectures, replace *rand(k,k)* with *randpat(k,k,0.5,0.5)*, and repeat the experiment. Again comment on your observations. Do you see any similar behavior? Any new behavior?”

```

testpf.m
function v=testpf(k,n);
v = [];
for j = 1:n
e = eig(rand(k,k));
plot(real(e),imag(e),'x')
hold on

```

```
plot([0],[0], 'r+')
grid
pause
plot(abs(e), zeros(k,1), 'bo')
pause
hold off
end
```

3.5 Pitfalls

There are two types of pitfalls that await anyone who tries to use MATLAB as a student laboratory. The first type is mathematical. As was seen in the first example of an experiment, care must be exercised in selecting a random matrix generator. The basic random matrix generator, $rand(m, n)$, produces full rank matrices (probability approximately one), well-conditioned matrices (probability approximately 0.95), and in the square case, diagonalizable matrices with distinct eigenvalues (probability approximately one). Consequently, if any of these properties are undesirable for a particular experiment, it is essential to construct an alternative random matrix generator with more desirable properties.

The second type of pitfall is that students can often confuse the forest with the trees. That is, students may express their conjectures as conjectures about the output matrix rather than about the test function, and consequently have no chance of finding a proof of their conjecture. As an example, suppose that the test function computes the rank of a matrix and the rank of the matrix times its hermitian conjugate. If the output from an experiment is a $2 \times n$ matrix whose j th column is $rank(A_j)$ and $rank(A_j^* A_j)$, students may express their conjecture in terms of the output matrix, such as "The output matrix has rank one," or "The numbers in the output matrix are always less than the size of the input matrix," rather than in terms of the test function, "The rank of a matrix always equals the rank of that matrix times its hermitian conjugate." This type of pitfall should diminish as the course progresses, and students have more experience in interpreting their laboratory results.