Using the R Statistical Programming Environment in the Teaching of a Linear Algebra Course

Scott K. Hyde
hydes@byuh.edu
Department of Mathematics
Brigham Young University – Hawaii
Laie, HI, 96762
USA

Abstract

The R statistical programming environment is an open source implementation of the S computer language. Its main functions are to analyze statistical problems, manipulate data, and produce graphics. However, because many statistical problems require the use of matrices, R is also a very powerful matrix program, which makes it ideal to use in teaching a Linear Algebra course. In addition, R is an open source program and part of the GNU project, ensuring its availability for all to download and use. This gives R a distinct advantage over programs like Matlab, Mathematica, and Maple, as they are not always available and can be cost prohibitive. R is available for many different operating systems, including UNIX, Linux, MacOS, and Windows. R is also extensible, which allows R to be changed to fit the needs of the user, either by writing new packages, or by installing additional packages from the Comprehensive R Archive Network (CRAN). The goal of this paper is to introduce the use of R in teaching a Linear Algebra course. Topics include creating vectors and matrices, extracting elements from a vector or a matrix, operations on matrices, and matrix factorizations. Instructions on the installation of R on a Windows system, as well as an example of using R in a Linear Algebra course are given.

1 Introduction

What is R? The R statistical programming environment is an excellent program with multiple functions, including analyzing statistical problems, manipulating data, mathematical computations, and producing graphics. It is an open source implementation of the S computer language, a high level language developed by John Chambers (and others) at Bell Labs [1]. Although initially written by Ross Ihaka and Robert Gentleman, who at the time were in the Department of Statistics at the University of Auckland in Auckland, New Zealand [3], many other individuals have contributed to R by writing and debugging code. R is an official part of the GNU project and is supported by the Free Software Foundation, which ensures its survival. Since

R is free, this gives it a unique advantage over programs such as Matlab, Mathematica, or Maple when the cost of the software is of primary concern to the educator or the student, and symbolic computation is not needed. R is also available for many different operating systems, including UNIX (Solaris, OpenBSD, NetBSD, etc.), Linux, MacOS, and Windows. This opens R up to virtually all people with computers. Additionally, R can also be changed to fit the needs of the user, either by writing new packages, or by installing additional packages from the Comprehensive R Archive Network (CRAN). Cheang [2] noted that the major strength of R is its powerful computing and flexible graphing capabilities.

In this paper we explore possible uses of R in teaching, learning, and doing linear algebra. First, examples of elementary topics such as creating vectors and matrices in R, along with commands for extracting elements from a vector or a matrix are given. Operations on matrices are also discussed, accompanied by factorizations of matrices. An example is given using R on a linear algebra problem, designed to demonstrate a few interesting capabilities of R. Instructions on installing R or R packages on a Windows system are provided at the end of the paper.

2 Uses of R in Linear Algebra

Commands in R are preceded by the R prompt (the greater than sign "`>`"), and are printed in a typewriter font that is slightly slanted to the right. Immediately following a command, the output is displayed in unslanted typewriter font. Comments are denoted by the `#` character preceding the comment.

2.1 Creating Vectors

The simplest way to creating a vector in R is done by using the `c()` command (short for concatenate). For example, to create the vector $\mathbf{v} = \begin{bmatrix} 1 & 3 & 2.2 \end{bmatrix}^T$, execute the commands

```
> v=c(1,3,2.2)
> v
```

```
[1] 1.0 3.0 2.2
```

A vector in R is always displayed as a row vector. However, R treats it as a column vector as well. The "`[1]`" indicates the element number of the element directly to the right of `[1]`. When vectors span multiple lines, then each subsequent line will have a similar indicator.

If the elements of the vector are consecutive in nature, then there are more concise ways to create vectors. Using the `:` command, a vector can be created from a starting point to an ending point, with a step size of one. For example, the command `3:8` creates a vector from 3 to 8, stepping one at a time.

To create a matrix in R, you use the `matrix()` command. For instance, the matrix

$$\boldsymbol{A} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \tag{1}$$

is created using either of the following commands:

```
> A=matrix(c(1,2,3,4,5,6,7,8,9),3,3)
> A=matrix(1:9,3,3)
```

2.2 Extracting Elements from a Vector or Matrix

Extracting elements is an important operation. There are several commands to do so. To extract the eighth element of a vector, one would type

```
> v=c(1,2,4,2,3,4,4,32,33,2,4)
> v[8]          #note the use of brackets [] parentheses don't work!
```

```
[1] 32
```

You can even select a sequence of elements:

```
> v[8:10]       #selects the 8th, 9th, and 10th elements.
```

```
[1] 32 33  2
```

Or a complement of a sequence of elements:

```
> v[-c(8,9,10)] #selects everything but the 8th, 9th, and 10th elements.
```

```
[1] 1 2 4 2 3 4 4 4
```

You can even select elements based on a comparison

```
> v[v>12]       #select all elements of v which are larger than 12.
```

```
[1] 32 33
```

Extracting elements from a matrix is similar, but you specify two values instead of one. For example, to extract the first column, the second row, or the element in the second row and the third column of the matrix $\boldsymbol{A}$ in equation (1), you would type the following:

```
> #extract the 1st column > #extract the 2nd row    > #extract (2,3) element
> A[,1]                   > A[2,]                   > A[2,3]

[1] 1 2 3                 [1] 2 5 8                 [1] 8
```

If you specify only one argument instead of two, then it treats the matrix as a vector (by stacking the columns). For example, the command

```
> A[c(4,7,8)]
```

```
[1] 4 7 8
```

extracts the elements in the upper triangular part of the matrix $\boldsymbol{A}$. You can also specify vectors instead. For example, can you tell what the command `A[1:2,1]` does?

```
> A[1:2,1]
```

```
[1] 1 2
```

2.3 Operations on Matrices

Common matrix operations are included in R, such as matrix addition, subtraction, or multiplication, as well as other types of operations. R performs the common addition and subtraction of matrices with the `+` and the `-` commands. However, matrix multiplication is **not** performed by the `*` command, but rather the `%*%` command. The `*` command will perform the Hadamard product [7], which performs the multiplication element-wise. An example of these commands:

```
> B=matrix(1:4,2,2)
> C=matrix(4:1,2,2)
> B
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> C
     [,1] [,2]
[1,]    4    2
[2,]    3    1

> B+C      #element-wise add
     [,1] [,2]
[1,]    5    5
[2,]    5    5

> B-C      #element-wise subtract
     [,1] [,2]
[1,]   -3    1
[2,]   -1    3

> B*C      #element-wise multiply
     [,1] [,2]
[1,]    4    6
[2,]    6    4

> B%*%C    #matrix multiply
     [,1] [,2]
[1,]   13    5
[2,]   20    8
```

Other basic operations needed for linear algebra are also possible with R, such as the determinant, the transpose, the inverse, the dimension of a matrix, augmenting matrices, or creating an identity matrix. Examples of these commands are:

```
> det(B)        #the determinant
> t(B)          #the transpose
> solve(B)      #matrix inverse of B
> sum(diag(B)) #the trace
> dim(B)        #matrix size
> cbind(B,C)    #augment column-wise
> rbind(B,C)    #augment row-wise
> diag(3)       #a 3x3 identity
```

As can be seen from the commands and operations given above, R is intuitive and quite easy to use for elementary linear algebra ideas.

One command that is lacking in the standard R package is the command to place a matrix in reduced row echelon form, or the `rref` command. However, packages can be installed that have this command. The author created the package `m343linalg`, which is a compilation of useful commands gathered from several sources. This package is provided by the author for free and can be installed by anyone using R. Instructions for installing the `m343linalg` package, which includes the `rref` feature and other important linear algebra features not in the standard R package are given in Appendix 5.2.

2.4 Matrix Factorizations

Several matrix factorizations are possible using the current core version of R, with others possible by installing new packages. The Eigen-decomposition is one of the most common decompositions in a linear algebra course. The `eigen()` command extracts both the eigenvectors and the eigenvalues. Storing the result of the `eigen()` command allows access to both the eigenvectors and the eigenvalues without recomputing them:

```
> A=matrix(1:9,3,3)
> temp=eigen(A)  #store both the eigenvalues and vectors in temp
> temp           #display everything in temp

$values
[1]  1.611684e+01 -1.116844e+00 -4.054215e-16

$vectors
           [,1]       [,2]       [,3]
[1,] -0.4645473 -0.8829060  0.4082483
[2,] -0.5707955 -0.2395204 -0.8164966
[3,] -0.6770438  0.4038651  0.4082483

> temp$values    #displays only the $values of the object temp

[1]  1.611684e+01 -1.116844e+00 -4.054215e-16
```

Another very important decomposition often covered in linear algebra is the Singular Value Decomposition. There are two different variations of this decompositions. The first variation obtains orthogonal matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, and a rectangular diagonal matrix $\boldsymbol{D}$ such that $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$. Another variation, called the full rank version, obtains an $m \times r$ matrix $\boldsymbol{U}$, an $n \times r$ matrix $\boldsymbol{V}$, and a $r \times r$ diagonal matrix $\boldsymbol{D}$, where the rank of $\boldsymbol{A}$ is r. It is called the full rank singular value decomposition since $\boldsymbol{U}$ and $\boldsymbol{V}$ have full column rank, and $\boldsymbol{D}$ is full rank. The `svd()` command produces a vector `d`, which contains the singular values, and two matrices, `u` and `v`, which contains the left and right singular vectors.

```
> svd(A)

$d
[1] 1.684810e+01 1.068370e+00 3.069525e-16

$u
           [,1]        [,2]       [,3]
[1,] -0.4796712  0.77669099  0.4082483
[2,] -0.5723678  0.07568647 -0.8164966
[3,] -0.6650644 -0.62531805  0.4082483
```

```
$v
           [,1]       [,2]       [,3]
[1,] -0.2148372 -0.8872307 -0.4082483
[2,] -0.5205874 -0.2496440  0.8164966
[3,] -0.8263375  0.3879428 -0.4082483
```

The full rank version is not automatic in R, but can be found by discarding unneeded columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ and dropping singular values that are zero. The matrix $\boldsymbol{A}$ is reconstructed below to show equality.

```
> r=sum(svd(A)$d>1e-15)  #the rank
> S=svd(A,r,r) #only keep r vectors
> U=S$u; D=diag(S$d[1:r]); V=S$v
> U
           [,1]        [,2]
[1,] -0.4796712  0.77669099
[2,] -0.5723678  0.07568647
[3,] -0.6650644 -0.62531805

> D
         [,1]     [,2]
[1,] 16.84810 0.000000
[2,]  0.00000 1.068370

> V
           [,1]       [,2]
[1,] -0.2148372 -0.8872307
[2,] -0.5205874 -0.2496440
[3,] -0.8263375  0.3879428

> U %*% D %*% t(V)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Other decompositions are built into R, such as the LU, the Cholesky, the QR, and others. Other decompositions not included in R have been programmed (or can be programmed) and included in packages for R. Several of these packages are available through the Comprehensive R Archive Network (CRAN). An example of how to install a package in R using CRAN and another not using CRAN are given in Appendix 5.2.

3 Example

A typical problem in linear algebra is solving a system of equations, $\boldsymbol{Ax} = \boldsymbol{b}$ using the inverse matrix method ($\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$). R has this capability as well as being able to solve systems of equations that have no solution or infinitely many solutions. When $\boldsymbol{A}$ is non square or singular, the system of equations $\boldsymbol{Ax} = \boldsymbol{b}$ has no solution for most choices of $\boldsymbol{b}$. Solutions will only exist when the vector $\boldsymbol{b}$ is in the column space of $\boldsymbol{A}$. However, in this case, a least squares solution may be obtained.

When students are introduced to the topic of least squares approximations in linear algebra, it may not be apparent what "solution" is being found. A graphical representation of the column space can visually motivate how the least squares solution is related to the column space of $\boldsymbol{A}$.

To find the least squares inverse, we will use the Moore Penrose generalized inverse [4; 5; 6] (also called the "pseudo inverse") to find the least squares solution to $\boldsymbol{Ax} = \boldsymbol{b}$. The solution is $\hat{\boldsymbol{x}} = \boldsymbol{A}^{+}\boldsymbol{b}$, where $\boldsymbol{A}^{+}$ denotes the Moore Penrose inverse of $\boldsymbol{A}$. Using the package `m343linalg` (installation instruction located in appendix 5.2), we can use the `pinv()` command to find the pseudo inverse of a matrix. For example, consider the inconsistent system of equations

$$\begin{array}{rcrcl} x & - & 3y & = & 5 \\ -2x & + & 6y & = & 1 \end{array}$$

Define

$$\boldsymbol{A} = \begin{bmatrix} 1 & -3 \\ -2 & 6 \end{bmatrix},\ \boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix},\ \boldsymbol{b} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

and enter the commands in R:

```
> library(m343linalg)      #load the m343linalg package
> A = matrix(c(1,-2,-3,6),2,2)
> b = c(5,1)
```

The least squares solution ($\hat{\boldsymbol{x}}$) to the system is

```
> xhat = pinv(A)%*%b
> xhat
```

```
      [,1]
[1,]  0.06
[2,] -0.18
```

The image of the least squares solution is

```
> b_colspace = A%*%xhat
> b_colspace
```

```
     [,1]
[1,]  0.6
[2,] -1.2
```

How does the image of the least squares solution ($\boldsymbol{A}\hat{\boldsymbol{x}}$) compare to that of $\boldsymbol{b}$ and to other vectors in the column space of $\boldsymbol{A}$? To illustrate this, we generate random vectors in the column space and plot $\boldsymbol{b}$ as well as $\boldsymbol{A}\hat{\boldsymbol{x}}$. In the plot below, the $\boldsymbol{b}$ vector is the triangle in the plot, vectors in the column space are dots, and the image of the least squares solution is a square. Note that the square ($\square$) is the closest vector in the column space to the triangle ($\boldsymbol{b}$). This graphical representation helps the student visualize how the least squares solution is the "closest" vector in the column space to the vector $\boldsymbol{b}$. In addition, you can use this to motivate that the square in the plot is not only the image of the least squares solution, but also the projection of $\boldsymbol{b}$ onto the column space of $\boldsymbol{A}$.

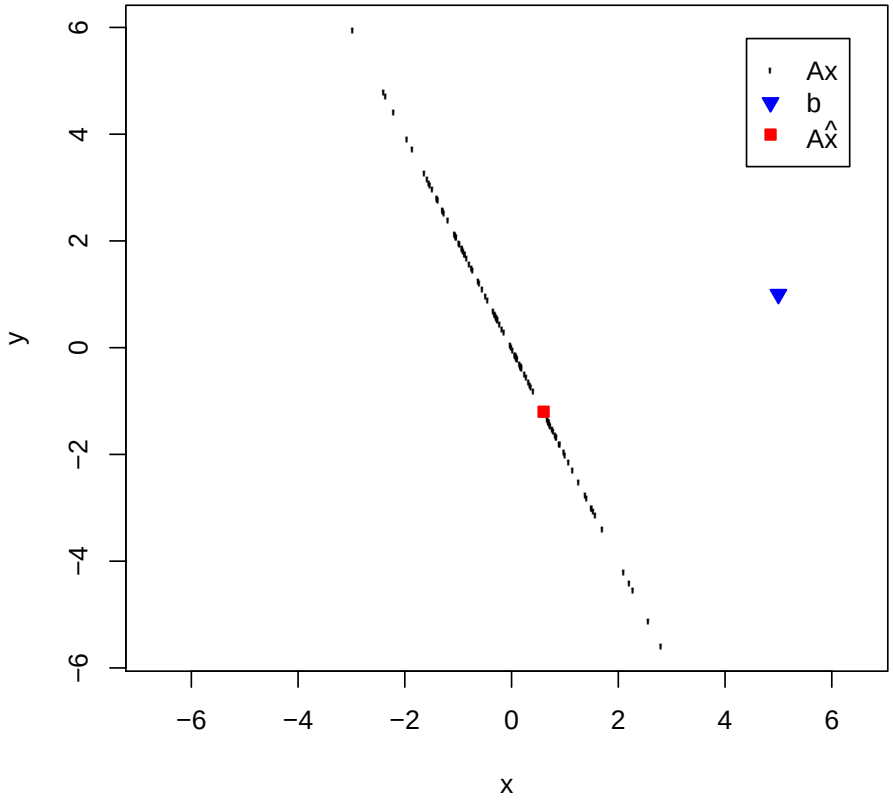

Figure 1: Plot of $\boldsymbol{b}$, random vectors $\boldsymbol{Ax}$, and the vector $\boldsymbol{A\hat{x}} = \boldsymbol{AA}^{+}\boldsymbol{b}$

4 Conclusion

R is a very powerful matrix program which can be very useful for teaching linear algebra. Learning and using R is within the grasp of undergraduate students taking linear algebra as well as for research purposes by mathematicians. Students will find that the program enhances their understanding of linear algebra. The author received several positive comments from students in his undergraduate linear algebra course. One comment was that "although R took some time to get used to, it contained many powerful tools that enabled them to understand difficult concepts and connect those concepts to other mathematical notions in other subjects such as differential equations." Another comment was that a student felt it helped them to "grasp concepts of linear algebra by allowing them to test various phenomena (the inverse of a matrix, the Gram-Schmidt orthogonalization, least squares, eigenvalues and eigenvectors) with different types of matrices easily." The programming nature of R was also commented on by the students. They were pleased that they could "create simple programming codes to test out processes which are not easily doable by hand, such as obtaining the steady state solution or

the diagonalization of a matrix." One other comment was that they were grateful for the understanding they gained by seeing the connection between orthogonal polynomials and orthogonal matrices through the use of the graphing tools of R. All of these comments serve as evidence of the helpful nature of R for learning, using, and doing linear algebra.

Since R is an open source program, it is always available to download and use free of charge. Cheang [2] states, "As a free software with additional support provided by the R Foundation, the R language provides a platform for mathematics educators and researchers to "freely explore" how technology can be applied into their teaching and research. R is an "open source" route to the development of better teaching strategies..." When the core R package is lacking in certain commands, R can be extended through the installation of packages that include new commands. For instance, the `m343linalg` package was developed from a collection of various programs for R by the author specifically for use in a linear algebra course. In this sense, R is an "evolving" language because every user is also a developer [2].

R is not only useful within a statistical setting, but also in other areas of mathematics including linear algebra. Thus, we see that R can be an invaluable tool for the educator teaching linear algebra.

5 Appendix

5.1 Installing R on a Windows System

To install R on a Windows system, follow these steps:

1. Go to `http://www.r-project.org`
2. Click on the word "CRAN" underneath the Download menu on the left.
3. Pick a mirror that is close to your location.
4. Select the operating system you use (Windows in this example).
5. Click on "base".
6. Click on the executable "R-2.7.1-win32.exe" (or similarly named R executable).
7. To install, double click on the file "R-2.7.1-win32.exe" that you downloaded.
8. A more detailed set of instructions can be found at `http://jekyll.math.byuh.edu/other/howto/R/R.shtml`, including the instructions for the installation of a couple of very nice editors of R code: Emacs and Tinn-R.

5.2 Installing an R package

To install a package for R that resides on the Comprehensive R Archive Network (CRAN), use the command `install.packages()` (or click on the Packages menu, then select "Install Package(s)"). You will then be asked to

1. Select a mirror (choose one close to you) and then click OK.

2. Select a package to install (scroll through the names until you find the package you want) and then click OK.
3. To load the package you installed, type `library(<name of package>)`, where `<name of package>` is the name of the package you installed. You can also load the package by selecting the "Packages" menu, then select "Load Packages". Select the package you want to load from the list given.

To install a package located outside of CRAN, you need to specify additional arguments to the `install.packages()` command. For example, to install the `m343linalg` package that was used for teaching Linear Algebra at Brigham Young University in Hawaii, you type the following commands:

```
> where="http://jekyll.math.byuh.edu/rlibs/"
> install.packages("m343linalg",contriburl=where)
```

To load the `m343linalg` package so that you can use the commands it provides, follow the same procedures as described above.

To get help on a particular package, select the "Help" pull down menu, then select "Html help". Your default web browser should pop up. Select the "Packages" item, and then you can select from the list of packages the one you need help on. If you know the name of the command you want help on, you can simply type `help(<command>)` at the R command prompt.

References

[1] John Chambers, *Stages in the evolution of s*, `http://cm.bell-labs.com/cm/ms/departments/sia/S/history.html`, 2000.

[2] Wai Kwong Cheang, *The use of r language in mathematics teaching and computation*, Proceedings of the 9th Asian Technology Conference in Mathematics (ATCM 2004) (T. de Alwis W. C. Yang, S. C. Chu and K. C. Ang, eds.), Singapore: National Institute of Education, December 2004, pp. 402–409.

[3] Kurt Hornik, *The R FAQ*, `http://CRAN.R-project.org/doc/FAQ/R-FAQ.html`, 2008, ISBN 3-900051-08-9.

[4] E. H. Moore, *On the reciprocal of the general algebraic matrix (abstract)*, Bulletin of the American Mathematical Society **26** (1920), 394–395.

[5] E.H. Moore, *General analysis*, Memoirs of the American Philosophical Society **1** (1935), 147–209.

[6] R. Penrose, *A generalized inverse for matrices*, Proceedings of the Cambridge Philosophical Society **51** (1955), 406–413.

[7] James R. Schott, *Matrix analysis for statistics*, John Wiley & Sons, 1997.