USING MATLAB WITH EXTERNAL DATA FILES

William H. Jermann
Department of Electrical and Computer Engineering
The University of Memphis
Memphis, Tennessee 38152

Abstract

Using MATLAB with external data files can be awkward. This paper cites potential difficulties and provides remedies. An illustrative classroom example is given.

Introduction

Since the early days of computer networking, the use of more than a single resource in seeking a problem solution has been suggested. Data can be streamed from one computer to another. However, it is often more practical to write output data into a text file. Then another resource can open the text file, and continue addressing the problem. Even if the computers or resources are operating in different environments, this method is practical. Using File Transfer Protocol or FTP programs, text files can easily be transferred among different environments. [1]

MATLAB is a valuable resource for a wide range of computational problems. In many applications it is the only tool needed. Intermediate results of computations may be stored in text files. At a later date, these results may be read into a MATLAB interpreter for additional processing. However, since data can be transferred to text files, other resources may be used in seeking a solution to a particular problem.

We have observed that some MATLAB users who have no difficulty storing and retrieving data from a MATLAB environment may encounter difficulties when transferred text files between MATLAB and other resources. For example, sometimes when a large file produced by MATLAB is entered into the workspace of a text editor, it appears to be just one long line containing non-conventional characters. Also, sometimes when a matrix produced by an external resource is entered into a MATLAB environment, it appears to become a different matrix. Problems such as these are relatively easy to avoid or to solve. However, if they are not addressed, it may become impractical to transfer data between MATLAB and other resources. These potential difficulties and appropriate remedies are addressed in the subsequent section of this paper.

In a text file, there is generally a one-to-one correspondence between visible characters and their ASCII codes. From the perspective of a programmer, a text file may be viewed as a file of lines. Each line consists of a sequence of ASCII codes terminated by a new-line code. Input data is not read directly from a file, but generally read from a file input buffer that contains a line.

However, the way in which the lines of a text file are stored differs among operating systems. Figure 1 illustrates how the bytes of a two-line text file are stored in three different operating systems.

The ASCII representation for “dog” is the sequence of base-10 codes, 100, 111, and 103, while “cat” is represented as 99, 97, and 116. In a Windows environment the end-of-line

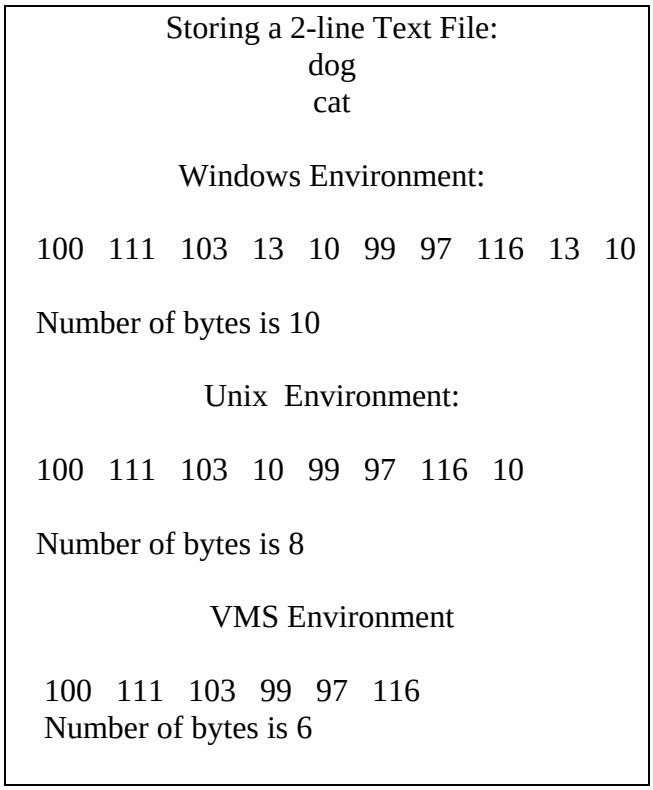

Figure 1. Storing a Two-line Text File in Different Environments

information is stored using two bytes, the carriage-return code and the line-feed code. In a Unix environment only one byte is used, and in a VMS environment no additional bytes are stored to represent end-of-line. The VMS mode of storage is possible since storage of a byte of information generally involves saving more than eight bits. One of the extra bits may be asserted when a character is the last character in a line of a text file.

In any environment, text files are generally read by retrieving characters from a file input buffer containing a line of the file. The last character of the line in the input buffer is generally the ASCII representation of the line-feed or new-line character. Thus, if the text file described in Figure 1 is opened as a text file, and if the characters are read one at a time and counted, the total count would be eight. This count is independent of the environment in which the characters are actually stored.

The information contained in Figure 1 was obtained by storing the lines of a text file, closing the file, and then opening the file as a binary file and reading the bytes. The source code for the corresponding program can be downloaded from the author's home page. Generally there is no need to know the details of text-file storage. FTP software can be used to transfer a text file between two different environments, and the modifications are generally transparent to the user.

Using Text Files With MATLAB

MATLAB users may store data in text files or read data from text files using commands that are very similar to the use of standard C functions. A major difference is that a file-read or file-write command applies to each element of a matrix rather than to just a single scalar. To open a file for writing, the following MATLAB command may be given.

```
x = fopen('filename','w')          (1)
```

This appears to be almost identical to invoking the C function "fopen", except that single quotes rather than double quotes are used with the arguments. However, this function is more closely related to a lower-level Unix function in that it returns a file descriptor rather than a pointer to a FILE-type structure. A file descriptor is an integer that generally has a value between 3 and 20 if the file is successfully opened. This file descriptor is then used to identify the file in which data is to be read from or written to.

Suppose a MATLAB user in a Windows environment creates the following square matrix:

```
a = [ 1  2          (2)
      3  4 ]
```

If an output file has been opened as shown in (1), the data may be stored in a text file in a manner similar to using the C function, fprintf.

That is,

```
fprintf(x,'%f\n',a)                (3)
```

Execution of this command causes each number in the matrix to be stored in the file. The numbers are stored "column-wise"; that is, the numbers in the first column are stored, followed by the numbers in the next column, etc. However, there is a major difference between execution of the MATLAB function and invoking the corresponding C function. Invoking the MATLAB command with the specified format string will produce a text file in Unix format, whereas invoking the corresponding C function will produce a text file in the native environment. [2]

A Windows text file produced by execution of the command given in (3) should appear as,

```
1.000000                           (4)
3.000000
2.000000
4.000000
```

However, if a file produced by this command is printed by a routine that prints each stored character, the following output will be printed:

```
1.000000                           (5)
        3.000000
                2.000000
                        4.000000
```

Furthermore, if this file is opened using Notepad, just a single line containing non-conventional characters will appear on the screen.

This difficulty is avoided if the MATLAB function that is called to store the matrix uses a format designator that explicitly stores both the return character and the new line character at the end of each line. That is,

```
fprintf(x,'%f\r\n',a)              (6)
```

If a Unix-type text file has been created in a Windows environment, there are several ways to convert it to a proper Windows text file.

1. If MATLAB is available, read the file into the MATLAB workspace, and then save it employing the proper format designator as shown in (6).

2. If the user has access to a Unix account, use FTP and send the file as a binary file to the Unix environment, and then retrieve it as an ASCII file.
NOTE: If the file is both sent and retrieved as an ASCII file, it will still be properly converted to a Windows file. When transmitted to a Unix setting, FTP just deletes the return characters, none of which exist in the source file.

3. Run a user-friendly program that converts a Unix file to a Windows file, or vice versa. Source code for such a program can be downloaded from the author's home page.

Use of the fscanf function for reading data from a text file is less sensitive since all leading white space is skipped before characters are read. However, a user should be aware that when matrix values are read into a MATLAB environment using the format string '%f", they are read by column; that is, the numbers entered are assigned to the first column of a matrix, then to the second column etc. Suppose a matrix is stored in a text file as shown below:

```
1.0     2.0
3.0     4.0
```

If the fscanf function is invoked to read this data into a 2 by 2 matrix, the matrix in the MATLAB environment will be the transpose of the above matrix.

Related Classroom Activities

A required course in both our Electrical Engineering program and our Computer Engineering program is a junior-level course, Matrix Computer Methods in Electrical

Engineering. In addition to providing students with knowledge of basic concepts of linear algebra, the course is intended to provide students with better understanding of both procedural programming techniques and object-oriented techniques.

This course includes 10 required computer assignments. Six involve procedural techniques, and four require the use of object-oriented techniques. The C programming language is used initially to develop sets of elementary matrix functions. Then a Matrix class is defined as well as two derived classes, Square and Diagonal. The functions that have already been written are recompiled in C++ and used as basic building blocks for the related class methods. The operators + and * are overloaded so that Matrix objects can be added or multiplied without explicitly invoking class methods. [3]

The Matrix class includes class methods that perform several elementary matrix operations. The Square class inherits the attributes of the Matrix class. Its class methods include a method that returns the determinant of a Square object as well as a method that returns the inverse of a nonsingular matrix. The prototype of another Square class method is shown below.

```
Square funct(ftp,char *file_name, int flag=0) ;
```
(7)

This class method returns a function of a square matrix. Recall that a function of a matrix may be defined using the MacLaurin series corresponding to the function, and that the function is specified exactly by evaluating a remainder polynomial. Further recall that a function of a matrix can be computed by evaluating the product,

$$M \text{ times } f(D) \text{ times } M^{-1} . \qquad (8)$$

In the above expression, D is a spectral matrix of the source matrix, f is a function of diagonal matrix D, and M is a modal matrix corresponding to D. Clearly the computational method specified in (8) is valid only if the source matrix has a complete set of independent eigenvectors.

Refer to the class method prototype shown in (7). The data type “ftp” has been previously defined by a type definition statement as a pointer to a function with one argument, a double, that returns the value of a double. The method “funct” computes the specified function by performing the computation shown in (8), and returns the value of the resulting matrix.

The Square class that has been defined does not include resources for finding eigenvalues or eigenvectors of a matrix. These must be obtained from some other resource, such as MATLAB, and stored in a text file. A string containing the name of this text file is carried into file_name. This text file must contain the values of the source matrix, a normalized modal matrix, and its corresponding spectral matrix. Figure 2 shows a program that uses Square objects and the “funct” class method to compute functions of a square matrix, and Figure 3 shows the results when the program is run using two different external data files.

The statement identified as Line a in Figure 2 causes the source code for all class methods to be included. A programmer who is not a novice would probably separately compile the file called matrix7.cpp, and then link with the object code for the main module. The class definitions would have to be defined in each module using the statement, #include "matrix7.h". In the statement identified as Line b, the 4 by 4 square object x invokes the class method called funct, the specified function is computed, and the resulting matrix is returned and assigned to y. It is not necessary to define the values of the source matrix prior to invoking funct since they are contained in the file called file1.txt. In addition to computing the specified function, funct assigns values of the source matrix to the object that invokes funct. In the lines identified as Line d and Line e, two other functions of the source matrix are computed and printed. The results are shown in Figure 3.

```
#include "matrix7.cpp"                                          // Line a
#include <math.h>
double what(double x)   { return 2 * x; }

int  main()
{   Square x(4); Matrix y;   printf("\n Matrix x is: ");
    y = x.funct(what,"file1.txt");                    x.print();       // Line b
    printf("\n 2 * x is:");                           y.print();       // Line c
    printf("\n exp(x) is:");  y = x.funct(exp,"file1.txt");   y.print();       // Line d
    printf("\n cos(x) is:");   y = x.funct(cos,"file1.txt");   y.print();       // Line e
    /* Find a function of a matrix that is not symmetric */
    Square z(2); y = z.funct(sqrt,"file2.txt",NOT_SYMM);              // Line f
    printf("\n Matrix z is: ");                       z.print();       // Line g
    printf("\n y = square root of matrix z is:");     y.print();       // Line h
    printf("\n y squared is"); y = y*y;                y.print();       return 0;}
```

Figure 2. A Program That Finds Functions of a Square Matrix

```
Matrix x is:

  1.000   2.500   0.000  -1.000
  2.500   0.500   1.000   2.000
  0.000   1.000   1.500  -0.500
 -1.000   2.000  -0.500   2.000

 2 * x is:

  2.000   5.000   0.000  -2.000
  5.000   1.000   2.000   4.000
  0.000   2.000   3.000  -1.000
 -2.000   4.000  -1.000   4.000

exp(x) is:

  14.845  15.197   5.608   3.286
  15.197  22.340   6.713  14.157
   5.608   6.713   7.107   0.424
   3.286  14.157   0.424  19.362

 cos(x) is:

 -0.697   0.077  -0.427  -0.034
  0.077  -0.889  -0.005   0.054
 -0.427  -0.005  -0.064   0.106
 -0.034   0.054   0.106  -0.896

 Matrix z is:

  5.000   4.000
  1.000   8.000

 y = square root of matrix z is:

  2.200   0.800
  0.200   2.800

 y squared is

  5.000   4.000
  1.000   8.000
```

Figure 3. Output of Program Shown in Figure 2

The class method called funct was originally written to find functions of symmetric matrices. It can also be used to compute functions of asymmetric matrices that have real eigenvalues and a complete set of independent eigenvectors. Refer to Line h. To find functions of an asymmetric matrix, a third argument must be carried into funct. This argument can be any non-zero integer. The statement identified as Line h returns the square root of the 2 by 2 matrix stored in "file2.txt". The last line of the program is used to demonstrate that the matrix returned by funct really is a square root of the source matrix.

MATLAB is used to produce the required text files. First, a text editor is used to store the values of a 4 by 4 matrix in file1.txt. If this is done outside a Windows environment, an FTP program can be used to transfer the ASCII file to a Windows environment. Then a MATLAB interpreter is opened, the commands shown in Figure 4 are entered, and the desired text file is produced. If the C++ program identified in Figure 2 is run in some other environment, FTP can be used to transfer this ASCII file. [4]

```
y = fopen('file1.txt','r')
 a = fscanf(y, '%f',[4,4])
 fclose(y)
 y = fopen('file1.txt','a')
 [x d] = eig(a)
 fprintf(y,'%f\r\n',x)
 fprintf(y,'%f\r\n',d)
```

Figure 4. MATLAB Commands that Produce the Required Text File

Students are given an assignment in which they must find functions of two matrices. A portion of the assignment includes using MATLAB with external data files. Since many of our students execute C++ in either a Unix or a VMS environment, the assignment also requires transporting text files between different environments.

Conclusions

We believe students benefited from activities discussed in this paper. Perhaps for the first time, they were required to use more than one computer resource to obtain a computational solution to a problem. Furthermore, we believe they developed better understanding of data file storage in various environments.

All software discussed in this paper as well as the related student assignment can be downloaded from the author's home page: www.people.memphis.edu/~wjermann/jermann.htm

References

1. Douglas E. Comer, Computer Networks And Internets, second edition, Prentice Hall, 1999.

2. Brian Kernighan and Dennis Ritchie, The C programming Language, second edition, Prentice Hall, 1989.

3. W. H. Jermann, "Reinforcing Basic Concepts With Class Definitions, Computers in Journal, Vol. XIV, No. 2, April-June,2004, pp 8-12.

4. MATLAB Reference Guide, The Math Works, Inc., Natick MA, 1992.

Biographical Information

Dr. Jermann has been teaching engineering subjects for over 40 years. He has written 3 textbooks, and has received several teaching awards.