Matrix Calculations Using XLispStat
by John E. Floyd
University of Toronto
May 15, 2013

This document presents an exercise in matrix calculations using the freely available statistical program XLispStat, which was written by Luke Tierney at the University of Minnesota. In doing this exercise, you will learn a few of the basics of XLispStat.

To obtain XLispStat, download the self extracting zip-file \texttt{wxls32zp.exe} from \url{http://www.economics.utoronto.ca/jfloyd/stats/wxls32zp.exe}, and place it in a directory you create for it called \texttt{xlispstat} in the Program Files directory on your MS-Windows computer. Then click on \texttt{wxls32zp.exe} and all the program files will be extracted into that directory. Finally, right-click on the \texttt{wxls32.exe} icon in the directory and drag it to your desktop to create a desk-top icon.

Before proceeding, the following group of questions is presented for which the program is used to check answers that can easily be calculated by hand.

1. Subtract the matrix
$$\begin{bmatrix} 4 & 0 & 8 \\ 6 & 0 & 2 \\ 8 & 2 & 3 \end{bmatrix}$$ from the matrix
$$\begin{bmatrix} 8 & 1 & 3 \\ 4 & 0 & 1 \\ 6 & 0 & 3 \end{bmatrix}.$$ 

2. Calculate the inner-product and outer-product of the vectors
$$\begin{bmatrix} 8 \\ 4 \\ 6 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}.$$
3. Pre-multiply and post-multiply the matrix
\[
\begin{bmatrix}
4 & 8 \\
6 & 2 \\
8 & 3
\end{bmatrix}
\] by the matrix \[
\begin{bmatrix}
8 & 1 & 3 \\
4 & 2 & 1
\end{bmatrix}
\].

4. Pre-multiply and post-multiply the matrix \[
\begin{bmatrix}
8 & 1 & 3 \\
4 & 2 & 1
\end{bmatrix}
\] by the matrix \[
\begin{bmatrix}
4 & 8 \\
6 & 2
\end{bmatrix}
\].

5. Calculate the determinants of the two matrices
\[
\begin{bmatrix}
4 & 0 & 8 \\
6 & 0 & 2 \\
8 & 2 & 3
\end{bmatrix}
\] and \[
\begin{bmatrix}
4 & 2 & 2 \\
6 & 2 & 3 \\
8 & 4 & 2
\end{bmatrix}
\]
and explain why they are singular or non-singular.

6. Put the three equations
\[
8x + y + 3z = d \\
4x + z = e \\
6x + 3z = f
\]
into matrix form and then solve by substitution to obtain the equilibrium values of \(x\), \(y\) and \(z\) in terms of the parameters \(d\), \(e\) and \(f\). Write the solution in matrix form and then extract the inverse of the original matrix that pre-multiplied the \((x, y, z)\) column-vector. Assuming that \(d = e = f = 1\), what are the equilibrium values of \(x\), \(y\) and \(z\)?
While XLispStat is a program using the programming language Lisp (which means List Processing) for statistical calculations, we can very easily use it for computational matrix programing and analysis. To begin, click on the XLispStat icon on your desktop. A screen will appear with the following text at the top.


> The > character is the program’s request for your input to tell it what to do.

Lisp commands have a very simple form. You enclose in a single set of brackets ( ) first the command or function you want the program to execute and then a series of words giving the arguments—that is, relevant information—the function requires in executing the command.

The first thing we need to do is give the XLispStat interpreter instructions to make some lists of numbers. The function we need to execute is named (not surprisingly) list and requires as arguments the group of numbers to appear in the list in the order in which you want them to appear. Also we need to give each list a name. This we do by using the function def which takes as its two arguments in this case the name of the list and the command to execute to produce the list. So let us make the following lists of numbers by entering the code

> (def LIST1 (list 4 6 8))
> (def LIST2 (list 0 0 2))
> (def LIST3 (list 8 2 3))
> (def LIST4 (list 8 4 6))
> (def LIST5 (list 1 0 0))
> (def LIST6 (list 3 1 3))

Notice that the list function is embedded in the def function.

To find out what items are in the work space we simply enter at the prompt the variables function, which takes no arguments, and to check the content of the lists we have just defined, we simply type the name of the list-object at the prompt. For example,
> (variables)
(LIST1 LIST2 LIST3 LIST4 LIST5 LIST6)
> LIST1
(4 6 8)
> LIST2
(0 0 2)
> LIST3
(8 2 3)
> LIST4
(8 4 6)
> LIST5
(1 0 0)
> LIST6
(3 1 3)
>
If you have a look at exercise 1 above, you will note that these lists are the columns of the two matrices there defined. To construct the matrices from these lists we use the bind-columns function, which takes as its arguments the names of the lists that are to form the columns. Again we have to use the def function to give these matrices names.

> (def MATA (bind-columns LIST1 LIST2 LIST3))
MATA
> (def MATB (bind-columns LIST4 LIST5 LIST6))
MATB
>
To have a look at these matrices, we use the print-matrix function, which takes as its single argument the name of the matrix to be printed.

> (print-matrix MATA)
#2a((
4 0 8
6 0 2
8 2 3
))
NIL
> (print-matrix MATB)
#2a(
  (8 1 3 )
  (4 0 1 )
  (6 0 3 )
)
NIL
>
The term **NIL** tell us that no new objects have been created in the workspace by our command.

Subtraction of **MATA** from **MATB** simply involves appropriate use of the `-` function which subtracts the second matrix listed from the first.

> (print-matrix (- MATB MATA))
#2a(
  ( 4 1 -5 )
  (-2 0 -1 )
  (-2 -2 0 )
)
NIL
>
Suppose now that we want to take the transpose of **MATA**. To do this we use the function **transpose**, feeding it the name of the matrix and using the **def** function to give the transpose the name **MATAT**.

> (def MATAT (transpose MATA))
MATAT
>
And we can have a look at this transposed matrix using the command

> (print-matrix MATAT)
#2a(
  (4 6 8 )
  (0 0 2 )
  (8 2 3 )
)
NIL
>
Another way to calculate the transpose of $\text{MATA}$ is to bind the lists $\text{LIST1}$ $\text{LIST2}$ and $\text{LIST3}$ together as rows, not columns using the $\text{bind-rows}$ function.

$$> \text{(def MATC} \ (\text{bind-rows list1 list2 list3}}))$$

$\text{MATC}$

$$> \text{(print-matrix MATC)}$$

```
#2a( 
   (4 6 8 ) 
   (0 0 2 ) 
   (8 2 3 ) 
)
```

$\text{NIL}$

$$>$$

You will recognize that $\text{MATC}$ and $\text{MATAT}$ are identical.

Question 2 asked for calculation of the inner-and outer-products of two column vectors, which can be created as 3x1 matrices using the $\text{bind-columns}$, $\text{list}$ and $\text{def}$ functions.

$$> \text{(def vec1 (bind-columns} (\text{list} \ 8 \ 4 \ 6)))$$

$\text{VEC1}$

$$> \text{(def vec2 (bind-columns} (\text{list} \ 2 \ 3 \ 3)))$$

$\text{VEC2}$

$$> \text{(print-matrix VEC1)}$$

```
#2a( 
   (8 ) 
   (4 ) 
   (6 ) 
)
```

$\text{NIL}$

$$> \text{(print-matrix VEC2)}$$

```
#2a( 
   (2 ) 
   (3 ) 
   (3 ) 
)
```

$\text{NIL}$

$$>$$

Notice that we embedded the $\text{list}$ function in the $\text{bind-columns}$ function which was then embedded in the $\text{def}$ function.
The inner-product and the outer-product can be calculate using the function \texttt{matmult}, which pre-multiplies the right-most matrix by the left-most matrix given as its two arguments.

\begin{verbatim}
> (def IPROD (matmult (transpose VEC1) VEC2))
IPROD
> (print-matrix IPROD)
#2a((
   (46.0000 )
))
NIL
> (def OPROD (matmult VEC1 (transpose VEC2)))
OPROD
> (print-matrix OPROD)
#2a((
   (16.0000 24.0000 24.0000 )
   ( 8.0000 12.0000 12.0000 )
   (12.0000 18.0000 18.0000 )
))
NIL
>
Next we construct the two matrices in Question 3, embedding alternatively the \texttt{list} and the \texttt{bind-columns} and \texttt{list} and \texttt{bind-rows} functions in the \texttt{def} function.

\begin{verbatim}
> (def MATD (bind-columns (list 4 6 8)(list 8 2 3)))
MATD
> (def MATE (bind-rows (list 8 1 3)(list 4 2 1)))
MATE
> (print-matrix MATD)
#2a((
   (4 8 )
   (6 2 )
   (8 3 )
))
NIL
>
> (print-matrix MATE)
#2a((
   (8 1 3 )
))
\end{verbatim}
Now when we pre- and post-multiply \textbf{MATD} by \textbf{MATE} we get

\begin{verbatim}
> (def MATED (matmult MATE MATD))
MATED
> (def MATDE (matmult MATD MATE))
MATDE
> (print-matrix MATED)
#2a(
    ( 62.0000  75.0000 )
    ( 36.0000  39.0000 )
)
NIL
> (print-matrix MATDE)
#2a(
    ( 64.0000  20.0000  20.0000 )
    ( 56.0000  10.0000  20.0000 )
    ( 76.0000  14.0000  27.0000 )
)
NIL
>
\end{verbatim}

Notice that the two product matrices are completely different.

Question 4 asks you to pre- and post-multiply \textbf{MATD} above by the \(2 \times 2\) matrix constructed below.

\begin{verbatim}
> (def MATF (bind-columns (list 4 6)(list 8 2)))
MATF
\end{verbatim}
> (print-matrix MATF)
#2a(
  (4  8  )
  (6  2  )
)
NIL
>
Pre-multiplication of MATF by MATD yields
> (def MATDF (matmult MATD MATF))
MATDF
> (print-matrix MATDF)
#2a(
  ( 64.0000  48.0000  )
  ( 36.0000  52.0000  )
  ( 50.0000  70.0000  )
)
NIL
>
while post-multiplication of MATF by MATD produces the error message
> (def MATFD (matmult MATF MATD))
Error: dimensions do not match
Happened in: #<Byte-Code-Closure-MATMULT: #1418c34>
>
which results, of course, because the matrices are not conformable for multiplication in that order—MATF has 2 columns while matrix MATD has 3 rows.

Question 5 asks for calculation of the determinant of MATA and the determinant of the following new matrix.
>
> (def MATG (bind-columns (list 4 6 8)(list 2 2 4)(list 2 3 2)))
MATG
> (print-matrix MATG)
#2a(
  (4  2  2  )
  (6  2  3  )
  (8  4  4  )
)

9
To calculate determinants we use the function \texttt{determinant} which takes as its sole argument the matrix for which the determinant is to be calculated. Applying this function to the two matrices above yields

\begin{verbatim}
> (def DETMATA (determinant MATA))
DETMATA
80.0
>
which is non-singular and

\begin{verbatim}
> (def DETMATG (determinant MATG))
DETMATG
0.0
>
which is singular because, as you can see, the third column is the first column multiplied by 2 and the two columns are therefore not independent.
\end{verbatim}

Finally, we need to do the computations relevant for Question 6. The three-equation system is as follows in matrix form.

\[
\begin{bmatrix}
8 & 1 & 3 \\
4 & 0 & 1 \\
6 & 0 & 3
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
=
\begin{bmatrix}
d \\
e \\
f
\end{bmatrix}
\]

Solving the system by substitution produces the following results.

\[
x = (1/2)e - (1/6)f \\
y = d - e - (2/3)f \\
z = -e + (2/3)f
\]

which can be presented in matrix form as

\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
=
\begin{bmatrix}
0.0000 & 0.5000 & -1.6666 \\
1.0000 & -1.0000 & -0.6666 \\
0.0000 & -1.0000 & 0.3333
\end{bmatrix}
\begin{bmatrix}
d \\
e \\
f
\end{bmatrix}
\]

where the square matrix is the inverse of the original one. We construct the original matrix in XLispStat using the following code.
> (def ORIGMAT (bind-columns (list 8 4 6)(list 1 0 0)(list 3 1 3)))
ORIGMAT
> (print-matrix ORIGMAT)
#2a(
  (8 1 3  
  (4 0 1  
  (6 0 3  
)
NIL
>
The inverse of this matrix is

> (def INVOMAT (inverse ORIGMAT))
INVOMAT
> (print-matrix INVOMAT)
#2a(
  (-6.938894E-18 0.50000 -0.166667  
  ( 1.00000 -1.00000 -0.666667  
  ( 2.775558E-17 -1.00000 0.666667  
)
NIL
>
which is the same as the one calculated by hand using substitution, except for the fact that XLispStat makes mathematically perfect calculations which recognize that the zero-numbers are only approximate and must therefore be presented in scientific notation. To obtain conventional decimal numbers, the decimal point of the number in the upper left corner of the matrix has to be moved 18 positions to the left, yielding the conventional decimal number \(-.00000000000000006938894\), and 17 positions to the left in case of the number in the bottom left corner, yielding \(.00000000000000002775558\).
Finally, assuming that $d = e = f = 1$, the solution of the system can be obtained in XLispStat as follows.

```lisp
> (def XYZEQ (matmult INVOMAT (bind-columns (list 1 1 1))))
XYZEQ
> XZYE
#2A((0.33333333333333337) (-0.6666666666666666) (-0.33333333333333337))
> 
You are advised that the calculations here are performed in the XLispStat batch file `matrix.lsp` and the output you will obtain from running that file is in the file `matrix.lou`. You will notice in the batch file the functions `princ` and `terpri` that are not discussed here. The `princ` function tells the program to print whatever is included as an argument. When printing text, the material must be encased in quotation marks while printing objects in the workspace requires only the name of the object. The `terpri` function, which is placed in the usual brackets without any arguments included, merely tells the interpreter to start a new line.