Designing a Computer Program for Matrix Operations

Guillermo Frank* Luis Frank†

Abstract
The paper presents a high-level software for solving linear systems. The program, written entirely in standard C language, operates directly on the command-line or through simple ”scripts” following a syntax similar to the classical mathematical notation. The paper also presents numerous statistical examples (e.g. parameter-estimation of linear regression-models) using large matrices. Finally, the program performance is compared against other software available for scientific computing, and conclusions are drawn about the benefits of working with each of them.

Key Words: Matrix Language, Software, Linear Algebra

1. Introduction

The matrix-languages revolutionized scientific computing since the early 80s mainly due to the simplicity of their syntax, which allows the user to perform complex algebraic operations with minimal programming skills. In such languages the basic matrix-operations (e.g. multiplication, inversion, several matrix factorization, etc.) are built-in functions written in a low-level computer language (typically C or C++) and compiled into an executable file that runs “scripts” of high-level commands. These scripts are just text files with commands that resemble the traditional matrix-handwritten notation.

Currently there are several matrix-languages available. Possibly, the best known are MATLAB, offered as a licensed software by The MathWorks Company (2014), and GNU Octave (2014), offered freely by John Eaton and a large community of developers. MATLAB was conceived under MS Windows and extended later to Linux and other operating systems, while GNU Octave was first developped on a Linux environment and then spread to MS Windows. Other free matrix-languages are Euler Math Toolbox, developed entirely by René Grothmann (2014) under MS Windows and RLaBplus developed by Marijan Kostrun (2014) under Linux after Ian Searle’s RLaB (2005) for Windows. The script syntax of these four softwares are pretty similar and transcripts from one to another is fairly straightforward.

The four softwares (and others that we omit mention for brevity) are excellent products. However, there is a feeling that they lost attractive for new users (mainly graduate students of maths, engineering and economics) for several reasons: (a) MATLAB, for example, lost its original simplicity by introducing graphical interfaces and distributing its functions in many specific “tool-boxes”; (b) GNU Octave has also lost simplicity but in another sense: its source-code has grown significantly with the contribution of numerous developers turning it virtually incomprehensible for a single person; (c) Euler Math Toolbox and RLaBplus are one-man projects, requiring a thorough knowledge of the source-code and build process in the event that they are discontinued; and, even more important, (d) some of these programs are not completely transparent because they integrate their own source-code with precompiled libraries borrowed from other projects, preventing the sight of the numerical algorithms underlying the scripting high-level commands.

*Intendente Güiraldes 2160 - Pabellón I, Ciudad Universitaria, C1428EGA, Buenos Aires, Argentina.
†University of Buenos Aires, Faculty of Agronomy, Av. San Martín 4453, C1417DSE Buenos Aires, Argentina.

2. Objectives

The disadvantages mentioned above led us to create a minimalist matrix language that would reproduce the original simplicity of the aforementioned languages and serve as a benchmark to test whether the increasing complexity of those softwares has been accompanied by losses of calculation efficiency. We called this program MINI.

3. The MINI Matrix Language

MINI is a computer program for matrix operations written entirely in (ANSI) standard C. The source-code may be divided into two parts: (i) a command interpreter and (ii) a set of functions for solving basic matrix operations. The program runs in two different modes, a command line and script-files, both of which interpret a high-level language similar (although not identical) to that of MATLAB, GNU Octave, Euler Math Toolbox or RLaBplus. So far MINI can be compiled in Linux distributions only, although we are already working on a version for Windows and Apple as well. To test MINI we suggest compiling the single file source-code with the GCC compiler for Linux writting the following options:

```
>gcc -Wall -o mini.e mini.c -lm
```

In the forthcoming subsections we describe the main features of MINI's source-code and matrix language. MINI's source-code and example scripts are downloadable from (Frank G. and L. Frank, 2014).

The Command Interpreter

To understand the role of the command interpreter recall that the commands can be entered (on the command line) in two ways:

```
>arg1=mydata1.txt
>arg2=mydata2.txt
>arg1[operation] arg2
```

or

```
>arg1=mydata1.txt
>arg2=mydata2.txt
>arg1 [assignment] arg2 [operation] arg3
```

In the first case, the variables `arg1` and `arg2` are stored in memory, the operation between them (e.g. addition, product, etc.) is "interpreted" and solved, and the result is stored in memory under the name `ans`. The second case is similar to the first one, except that `arg1` and `arg2` are now called `arg2` and `arg3`, respectively, and (for the assiggnment "=") the result is stored in memory as `arg1`, besides being stored in `ans`. To perform these steps the `[operation]` symbol has to be interpreted (or "decoded") by a specific function, which we called `code`, and then the operation has to be solved solved by another function called `solve`. The following are the C codes of both functions.

Source-code 1. Function `code` decodes the commands written in the command-line. `c` is a string that contains all the characters introduced in the commnd-line. In the transcript below we only show the case of reading a product between matrices.

```
void code(char *c,char *arg1,char *arg2,char *op){
  int  i,j;
  i=0;
  j=0;
  *op='\0';
  *arg1='\0';
  *arg2='\0';
  while (*(c+i)!='\0'){
      switch(*(c+i)){
          case '*':  {*op='*'; *(c+i)='\0'; j=i+1; i--; break;}
        }
      i++;
    }
  *(op+1)='\0';
  strcpy(arg1,c);
  strcpy(arg2,c+j);
  return;
}
```

Source-code 2. Function `solve` for solving operations. The function below shows only the case of the inner product between matrices.

```
void solve(char *arg1,char *arg2,char *op){
  switch(*op){
      case '*':   {prod(arg1,arg2); break;}
      case '\0': {show(arg1); break;}
  }
  return;
}
```

So far, the matrix operations supported by MINI are

+	summation
−	substraction
⋆	product or scalar-by-matrix product
/	inversion or scalar inversion
ˆ	discrete power

while the assignments are

=	assignment of a result to a variable
=>	logic operator (see next section)

Built-in Functions

Three major matrix-operations were included in the source-code: adition (and substraction), product of two matrices and matrix-inversion. Matrix addition and substraction are simply the element by element counterpart of the same scalar operation and need no further explanation. The product of two matrices is the sum along the columns of the elements of the i-th row of left-hand-side matrix by the corresponding elements of the j-th column of the right-hand-side matrix. This rule, however, is not an extension of the scalar by a matrix product. In order to avoid defining different symbols for the matrix inner product and the scalar product in the scripting language we introduced in the source-code two conditionals that select the product type according to the dimensions of the factors. In this way, we retained the same symbol ($*$) for all kinds of products in the same fashion as in the handwritten syntax.

The solution to a linear system is perhaps the kernel of the software because it enables matrix inversion and several matrix decompositions. According to the literaure (e.g. Burden and Faires pp. 350-362, Press et al. pp. 36-43) the most widely used method to solve linear systems is the Gauss-Jordan elimination and back-substitution algorithm. Because of its simplicity and its extensive usage among softwares we introduced this algorithm in the source-code as a built-in function. However, we are aware that the Gauss-Jordan method may not be numerically optimal under certain circumstances, e.g. when solving systems involving sparse matrices. Once the Gauss-Jordan algorithm was introduced, matrix inversion is rather straightforward as each column of the inverse matrix is the solution to the linear system defined by the matrix to be inverted and the corresponding column of the identity matrix. Then, the reader should note that in a strict sense we refer by inverse-matrix to the left-inverse of the matrix. For documentary purposes, we reproduce below the C code of the function `gauss` that solves linear systems of the type $\mathbf{Ax} = \mathbf{b}$.

Source-code 3. Function `gauss` for solving linear systems by Gaussian elimination and backsubstitution. The function returns as a by-product the determinant of the right-hand-side matrix of the system.

```
double gauss(double *a,double *b,double *x,int n) {
 int h,i,j,k;
 double det,dum,pivot,cumsum;
 det=1.0;
 for(i=0;i<n;i++) *(x+i)=0.0;
 for(k=0;k<n;k++) {                  // check for singular matrix
     if(*(a+n*k+k)==0.0) {
         for(h=k+1;h<n;h++) {
             if(*(a+n*h+k)!=0.0) {
                 for(j=0;j<n;j++) {
                     dum=(*(a+n*k+j));
                     *(a+n*k+j)=(*(a+n*h+j));
                     *(a+n*h+j)=dum;
                 }
                 dum=(*(b+k));
                 *(b+k)=(*(b+h));
                 *(b+h)=dum;
                 det*=-1.0;
                 h=n+1;
             }
         }
         if(h==n) det=0.0;
     }
     if(det) {                        // compute gaussian elimination
         for(i=k+1;i<n;i++) {
             pivot=(*(a+n*i+k))/(*(a+n*k+k));
             for(j=k;j<n;j++) *(a+n*i+j)=(*(a+n*i+j))-pivot*(*(a+n*k+j));
```

```
                *(b+i)=(*(b+i))-pivot*(*(b+k));
            }
        }
    }
    for(i=0;i<n;i++) det*=(*(a+n*i+i));    // determinant
    if(det) {                              // back sustitution
        for(k=0;k<n;k++) {
            cumsum=0.0;
            for(i=0;i<k;i++) cumsum+=(*(a+n*(n-k-1)+n-i-1))*(*(x+n-i-1));
            *(x+n-k-1)=(*(b+n-k-1)-cumsum)/(*(a+n*(n-k-1)+n-k-1));
        }
    }
    return det;
}
```

Scripting Language

As mentioned before, MINI runs both in command-line mode and in script mode, although we conceived the program to operate preferably in the later. To quickly understand the script syntax we present two illustrative examples. In the first example we show the operations between matrices, while in the second we introduce iterative cycles.

Example 1. The OLS estimator. We estimate by OLS the parameter-vector of the linear model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma^2\mathbf{I})$$

where ϵ is a vector of iid errors. There are two possible syntaxes for this computation

```
X=Xdata
y=ydata
Z=X'
A=Z*X
b=Z*y
x=A\b
x
```

or,

```
X=Xdatos
y=ydatos
i=-1
X'
ans*X
ans^i
X*ans
ans'
ans*y
ans
```

In both syntax, we first read the matrices `Xdata.txt` and `ydata.txt`. Although it is possible to omit the `.txt`, as we did in the example, MINI requires that all data are saved in this format. A simple inspection of the two scripts also reveals that MINI supports only one mathematical operation at a time, including the transposition operation. The first syntax resembles that of other matrix languages in the sense that each partial result is stored in memory as a new variable (in the example $\mathbf{Z}$, $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{x}$) by means of the equal sign. And like other languages it is also possible to update a variable referencing it to its previous value, for example, as $\mathbf{x} = \mathbf{A}\mathbf{x}$. In the last line of the first script we show the usage of the back-slash for solving linear systems by Gauss-Jordan elimination. The vector $\mathbf{x}$ is, in the example, the solution to the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$. An alternative notation would be $\mathbf{x} = \mathbf{b}/\mathbf{A}$ (where $\mathbf{x}$ and $\mathbf{b}$ are now row vectors) which is analogous to the scalar notation.

The second syntax exploits the use of the variable `ans` for holding on the result computed immediately before. This is one of the most notable differences among MINI and other matrix languages in which every result is stored in a variable defined with the equal sign. Besides, in this second syntax the OLS solution is obtained by inverting the matrix $\mathbf{X}'\mathbf{X}$ and multiplying it by $\mathbf{X}'\mathbf{y}$. This way of finding the solution is similar to that of the first syntax because the inverse $(\mathbf{X}'\mathbf{X})^{-1}$ is also obtained by Gauss-Jordan elimination and back-substitution, but solving first the linear system $\mathbf{A}\mathbf{G} = \mathbf{I}$ in $\mathbf{G}$ and then multiplying the solution by $\mathbf{X}'\mathbf{y}$.[1] Note that the second script only stores in memory the variable $i = -1$ and the last result included in `ans`. Another feature of MINI is that it does not compute elementary operations between variables and scalars directly, such as $2\mathbf{A}$ or $\mathbf{A}^{-1}$. Although this may appear as a drawback, the authors followed this criterium in order to keep the source-code small. So far, it is not possible to define matrices via the command line, a facility we intend to introduce in future versions of the software.

Example 2. Probability of a standard normal random variable. We compute the probability $P(Z < z)$ by te well-known numerical approximation given in (Wikipedia, 2014).

$$\Phi(z) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \sum_{i=0}^{n} \frac{z^{2i+1}}{\prod_{j=0}^{i} 2j+1}$$

for an arbitrary z.

```
z=1.96
e=2.71828182846
a=0.398942280401

i=-1
j=1
b=2

z^b
i*ans
ans/b
e^ans
a=a*ans

i=-10
k=j
c=j
d=0
loop
```

[1]MINI does not invert matrices following the definition.

```
    z^k
    ans/c
    d=d+ans
    k=k+b
    c=c*k
    i=i+j
i=>loop

p=j/b
a*d
p=p+ans
p
```

In the above example we use the recurrence relation defined by the => operator. This operator is indeed an `if` operator that repeats the execution of the commands written below the label on the right hand side of the symbol => (called "loop" in the example) if the variable on the left is not equal to 0, or continues with the executing of the script otherwise. The => operator appears to be similar to the typical `for`/`end` cycle of other matrix languages. However, the commands to be executed inside the => cycle are not required to be located immediately after the cycle-call as would be the case in structured languages. Instead, those commands may be located anywhere in the script in a `goto` fashion. The main advantage of this design is that it does not require to retain in memory the status of the counter variable of a traditional `for` cycle.

4. Testing MINI against other Softwares

So far we saw that MINI is a computer program that interprets scripts of commands that resemble the traditional algebraic notation. MINI is also much smaller than alternative softwares used by the scientific community. Table 1 shows the size of four softwares installed in their full version. It can be seen that MINI is about 2×10^2 times smaller than the smallest of the four other programs without requiring auxiliary files. The obvious question at this point is, wether MINI is also as powerful as alternative softwares. To answer the question we measured the computing time demanded by MINI and the other programs to invert 10 randomly generated matrices of size $10^3 \times 10^3$. The test matrices were composed of integers between 1 and 10^4 and were the same for testing all the softwares. [2] The results are shown in Table 2.

5. Final Remarks

We presented a new (minimalist) matrix language inspired in early versions of languages available to the scientific community. The development of this language was intended (a) to provide the non-specialized user with a transparent and tractable source code, and (b) to allow the user the development of his own codes in the form of scripts whose syntax reproduces handwritten matrix notation. Both objectives were fully achieved.

Comparing the speed of MINI against that of other softwares we confirmed our initial conjecture that the latter lost efficiency as they grew in versatility and that this trend seems not to have been offset by the inclusion of optimized libraries such as LAPACK. Nevertheless, it should be pointed out that the selected benchmark softwares usually perform additional numerical refinements to increase the accuracy of the results, while MINI does

[2] We used integers instead of real numbers for ease of reading of the data files.

Table 1: Size of six softwares in [KB]

	Ubuntu Linux	MS Windows
Euler Math Toolbox 22.8 (2013) [3]	–	203,285
GNU Octave 3.6.4 (2013)	355,000	107,542
MATLAB 7.8/6.1 (2009/2001)	3,200,000	1,080,000
MINI 0.0 (2014)	21	–
RLaB 2.1 (2001)	–	3,735

Table 2: Elapsed time [sec] for inverting ten matrices of size $10^3 \times 10^3$

	Ubuntu Linux	MS Windows
Euler Math Toolbox 22.8 (2013)	–	117.57
GNU Octave 3.6.4 (2013)	38.13	72.46
MATLAB 7.8/6.1 (2009/2001)	44.26	32.85
MINI 0.0 (2014)	10.35	–
RLaB 2.1 (2001)	–	44.20

not. Anyway, the results still justify the development of specific software under certain circumstances as well as their coexistence with mature soft such as MATLAB, Octave or Euler Math Toolbox. The development of MINI also allowed us to identify some syntactic rigidities generalized in other matrix languages (cfr. for example the syntax of loops in MINI with that of the other softwares) which we assume come from the early versions and could not be improved to maintain syntactic compatibility among the current version and previous versions of each softwares.

Despite the mentioned advantages, MINI is far from being fully optimized. Among the next goals we propose (i) make more flexible the definition of variables to allow the use of scalars in sums, products and powers; (iii) allow the extraction of rows, columns or blocks of matrices; and (iii) incorporate new matrix operations, such as $\mathbf{QR}$ decomposition or $\mathbf{SVD}$ decomposition.

REFERENCES

Burden R and Douglas Faires L., 1998. “Análisis numérico.” 6ta. Edición. Thomson Editores.

Grothmann R., 2014. “Euler Math Toolbox version 2014-06-26.” Downloadable from http://euler.rene-grothmann.de/.

Wikipedia, the Free Encyclopedia. “Error function. http://en.wikipedia.org/wiki/Error_function.” Accesed July 4th., 2014.

MathWorks 2014. “MATLAB R2014a.” http://www.mathworks.com/products/matlab/

Frank G. and L. Frank, 2014. “The Mini Matrix Language Project.” https://sites.google.com/a/agro.uba.ar/mini/

Press W., Teukolsky S., Vetterling W. and B. Flannery, 2002. “Numerical Recipies in C: the Art of Scientific Computation.” Second Edition. Cambridge University Press.

GNU Octave, 2014. “GNU Octave 3.8.1.” http://www.gnu.org/software/GNU Octave/

Searle I., 2005. “RLaB 2.1.05 for Windows.” http://rlab.sourceforge.net/

Kostrun M., 2014. “RLaBplus 1.0.” http://rlabplus.sourceforge.net/