Exploring the Main Sources of Inefficiency of Softwares for Matrix Operations

Luis Frank[*] Guillermo Frank[†]

Abstract
MINI 1.1 is a computer program designed to solve matrix operations by means of simple scripts. The program was first released in 2014 and improved since to reach the computing speed of leading softwares. Although MINI 1.0 (compiled on Xubuntu) outperformed the speed of other competing softwares, the authors noted that the computation by means of scripts instead of built-in functions turned out to be a serious limitation of MINI because it reduced the speed 5 to 10 times, but that such drawback could be partially compensated by parallel execution of the program, as preliminary results showed that parallel execution reduces the computing time almost to one tenth. The paper below presents a new version of MINI (version 1.1), this time compiled directly on Xubuntu and MS Windows, and executed through its own GUI, and compares its performance to that of other softwares when executed on parallel mode. The results show that scripts are still competitive compared to built-in functions.

Key Words: Software, Linear Algebra

1. Introduction

MINI is an open-source computer program for matrix operations created in 2014 to overcome some speed limitations of available software mainly to solve linear systems of the form $\mathbf{Ax} = \mathbf{b}$. MINI was written entirely in C language (later translated to C++ with minor adaptations) and conceived under a minimalist philosophy.[1] To that end, the built-in functions were limited to those strictly necessary to interpret simple matrix operations, and its scripting language designed to perform one operation at a time. A detailed description of MINI's features may be found in G. Frank and L. Frank (2014, 2015) which we complement with an updated command sheet in the appendix of this paper. In the referred papers, MINI was also compared with other four softwares on two operating systems (MS Windows and Xubuntu Linux) to prove the authors' conjecture that the efficiency of most scientific computation software (e.g. MATLAB, GNU Octave and Euler Math toolbox) is nowadays limited mostly by the sophistication of the command interpreter rather than the built-in numerical algorithms. The findings of such comparison were:

(a) MINI 1.0 outperformed the benchmark software when ran in parallel mode on MS Windows, and is the second best on Xubuntu Linux. A detailed diagnosis on MINI's running processes revealed that 40 to 60% of its computing time was spent in the execution of loops and the assignment of variables to elements within matrices, while less than 10% of the time was used to solve linear systems with the gaussian elimination routine.

[*]University of Buenos Aires, School of Agriculture, Av. San Martín 4453, C1417DSE Buenos Aires, Argentina.
[†]University of Buenos Aires, Physics Department, Intendente Güiraldes 2160 - Pabellón I, Ciudad Universitaria, C1428EGA, Buenos Aires, Argentina.
[1]The latest version of MINI can be downloaded from www.miniproject.org

(b) The computation by means of scripts was in general 5 to 10 times slower than by means of built-in functions, which turns out to be a serious limitation for MINI because its minimalist philosophy discourages the inclusion of complex built-in functions (for example for SVD-decomposition) into the source code.

(c) MINI's *relative* performance on the Cygwin environment was similar to that of other software running on MS Windows, which meant that there was a chance to improve MINI's computing time by compiling the source code directly on Windows.

From these results, the authors proposed to release a new version of MINI fully compilable with Linux and Windows (and hopefully with other OS), and to run it in parallel mode, in order to compensate the loss of efficiency due to the use of scripts instead of built-in functions and to keep the command interpreter simple to reduce the overall computing speed of the program.

2. Objectives

In short, the aim of this paper is to present an improved version of MINI (version 1.20), this time compiled directly on Xubuntu and MS Windows and executed through its own GUI, and to compare its performance to that of other softwares. To that end, we test MINI's performing time to decompose 10 nonsingular big matrices $\mathbf{A}$ into singular values through a standard SVD-decomposition algorithm. The chosen algorithm is an iterative procedure that decomposes $\mathbf{A}'\mathbf{A}$ into the factor matrices $\mathbf{Q}$ and $\mathbf{R}$, and then uses these matrices to find the factor matrices **U**, **S**, and **V** returned by a reduced-form of the SVD-decomposition. To guarantee a fair comparison the same algorithm will be translated to other scripting languages in order to perform exactly the same operations.

3. Algorithms and Script

3.1 The QR-decomposition algorithm

QR-decomposition of an $n \times n$ matrix $\mathbf{A}$ into the factor matrices $\mathbf{Q}$ and $\mathbf{R}$ following the modified Gram-Schmidt process with normalization during the algorithm. If $\mathbf{A}$ is a nonsingular matrix, $\mathbf{Q}$ is an orthogonal $n \times n$ matrix and $\mathbf{R}$ is an $n \times n$ upper triangular matrix. Otherwise, if $\mathbf{A}$ is not full-rank, $\mathbf{Q}$ has r orthogonal columns (and $n-r$ zero-columns) and $\mathbf{R}$ has r non-zero-rows (r is the rank of $\mathbf{A}$), and the product $\mathbf{Q}'\mathbf{Q}$ is no longer equal to $\mathbf{I}_n$ although the identity $\mathbf{A} = \mathbf{QR}$ still holds, but is not unique.

```
$\mathbf{A}_0 = \mathbf{A}$
$\mathbf{Q} = \mathbf{I}_n$    ($\mathbf{Q}$ may also be set to $\mathbf{0}_{n\times n}$)
for $j = 1, \ldots, n$
    $\mathbf{Q}_j = \mathbf{A}_j / \sqrt{\mathbf{A}_j'\mathbf{A}_j}$    (if $\mathbf{A}_j'\mathbf{A}_j = 0$, stop)
    for $k = j+1, \ldots, n$
        $\mathbf{A}_k = \mathbf{A}_k - \mathbf{Q}_j'\mathbf{A}_k\mathbf{Q}_j$
    end
end
$\mathbf{R} = \mathbf{Q}'\mathbf{A}_0$
```

□

3.2 The SVD-decomposition algorithm

SVD-decomposition of an $n \times p$ matrix $\mathbf{A}$ into the factor matrices **U**, **S**, and **V**. $\mathbf{A}$ is a full column rank matrix. Then, $\mathbf{A} = \mathbf{USV}'$, where **U** and **V** are orthogonal matrices, that is, $\mathbf{U}'\mathbf{U} = \mathbf{I}$ and $\mathbf{V}'\mathbf{V} = \mathbf{I}$.

```
$\mathbf{G} = \mathbf{A}'\mathbf{A}$
$\mathbf{G}_0 = \mathbf{G}$
$\boldsymbol{\lambda}^{\text{OLD}} = \mathbf{1}_p$
$\mathbf{V} = \mathbf{I}_p$
$\mathbf{S} = \mathbf{0}_{p\times p}$
$\mathbf{U} = \mathbf{0}_{p\times p}$

for $h = 1 \dots 100$    (eigenvalues)
    $\boldsymbol{\lambda}^{\text{NEW}} = \mathbf{0}_p$
    $\{\mathbf{Q}, \mathbf{R}\} = QR(\mathbf{G})$
    $\mathbf{G} = \mathbf{Q}'\mathbf{G}\mathbf{Q}$
    Let the diagonal elements of $\mathbf{G}$ equal $\lambda_i^{\text{NEW}}$
    $\boldsymbol{\delta} = \boldsymbol{\lambda}^{\text{NEW}} - \boldsymbol{\lambda}^{\text{OLD}}$
    if $|\boldsymbol{\delta}'\boldsymbol{\delta}| < 10^{-8}$, then stop   (optional)
    $\boldsymbol{\lambda}^{\text{OLD}} = \boldsymbol{\lambda}^{\text{NEW}}$
end
Let the diagonal elements of $\mathbf{S}$ equal $\sqrt{\lambda_i}^{\text{NEW}}$
$\mathbf{G} = \mathbf{G}_0$
for $h = 1 \dots 20$    (eigenvectors)
    $\{\mathbf{Q}, \mathbf{R}\} = QR(\mathbf{G})$
    $\mathbf{V} = \mathbf{VQ}$
    $\mathbf{G} = \mathbf{RQ}$
end
$\mathbf{B} = \mathbf{VS}'$
for $i = 1 \dots n$
    $\mathbf{U}_i' = \mathbf{B}^{-}\mathbf{A}_i'$
end
return $\{\mathbf{U}, \mathbf{S}, \mathbf{V}\}$    $\square$
```

3.3 The Script

The following script illustrates MINI's scripting language. The full rank matrix $\mathbf{A}_{n\times p}$ is the input matrix. $\mathbf{U}, \mathbf{S}, \mathbf{V}$ are output matrices that satisfy $\mathbf{A} = \mathbf{USV}'$. The script starts with $\mathbf{G} = \mathbf{A}'\mathbf{A}$, $\mathbf{U} = \mathbf{0}_{n\times p}$, $\mathbf{S} = \mathbf{0}_p$, $\mathbf{V} = \mathbf{I}_p$, $\mathbf{q}_0 = \mathbf{0}_{p\times 1}$ and $\mathbf{q}_1 = \mathbf{1}_{p\times 1}$, and invokes the function `qr()`, not shown in this paper. The lines highlighted in red show some new features of MINI 1.2. First, for programming ease numeric names for variables are allowed, so e.g. the number 1 may be called as 1. That's the meaning of `1=1` in the script below. Second, auxiliary functions may be called from any functions. However, this facility should be handled with caution as all variables are global in MINI. Third, the => conditional may be used either to define a `for` loop (as in the example) or to insert an `if` conditional sentence.

```
# Compute S and
# eigenvalues
iter=100
```

```
LOOP02
  eig=q0
  A=G
  qr(A) (returns Q, R)
  Q'
  ans*G
  ans*Q
  G=ans
  p=q1'
  p=p*q1
  1=1 (define 1)
  i=1
  LOOP03
    h=i-1
    h=h*p
    h=h+i
    G[h]
    eig[i]=ans
    i=i+1
    p*p
    ans-h
  ans=>LOOP03
  iter=iter-1
ans=>LOOP02
i=1
LOOP04
  h=i-1
  h=h*p
  h=h+i
  eig[i]
  sqrt(ans) (built-in)
  S[h]=ans
  i=i+1
  p*p
  ans-h
ans=>LOOP04  (if ans≠ 0 → LOOP04)

# Compute V and
# eigvectors
G=G0
iter=20
LOOP05
  A=G
  qr(A)
  V=V*Q
  G=R*Q
  1=1
  iter=iter-1
ans=>LOOP05
iter={}
```

```
# Compute U
# for A=USV′
A=A1
i=-1
V′
S⋆ans
ans^i
A⋆ans
U=ans
```

□

Note: the clearing of variables at the end of the script is not shown in the paper.

4. Results and Conclusions

The table below shows the time (in seconds) to decompose ten $10^3 \times 10^3$ and $10^3 \times 10^2$ matrices into singular values via built-in functions and scripts, respectively. The comparison involved 5 softwares on two OS.

	Xubuntu Linux		Windows	
	built-in	scripts	built-in	scripts
Euler Tool. 22.8	–	–	436	1,127
Octave 3.6.4	366	10,675	183	3,666
MATLAB 7.8/6.1	89	843	150	725
MINI 1.0 S	–	10,580	–	3,571
MINI 1.0 P	–	1,067	–	479
MINI 1.2 S β	–	334	–	384
MINI 1.2 P β	–	104	–	–
RLaB 2.1	–	–	239	–

The results show that the differences in the execution time of MINI 1.0 S (sequential mode) vs GNU Octave and MINI 1.0 P (parallel mode) vs MATLAB are non-significant ($\alpha = 0.05$) using Lord's U test statistic. The best performance of MINI 1.2 compared to its previous version is due to the development of its own GUI. Recall that version 1.0 runs under Windows only via the Cygwin Linux emulator. The results also show that scripts slow down significantly (according to the U statistic) the computing time of complex functions, but the slowdown may be compensated by parallel computation (P) and the optimization of the syntax of scripts. It is also worth noting that the OS does not reduce the computing time.

In short, we confirm that most of the programs of matrix operations suffer from serious inefficiencies due mainly to the command interpreter and performing environment. This hypothesis was verified, for example, MINI 1.1 (Frank L. and G. Frank 2015), in which 40 to 60% of the execution time was devoted to resolving for-loops and to the assignment of elements within matrices to variables, while less than 10% of the time was spent to solve linear systems, and is verified now in MINI 1.2, in which the execution through its own

GUI improved its runtime to reach that of other leading software. So far, the only way the authors found to compensate the mentioned inefficiencies is by parallel execution. The development of new versions of MINI will go in this direction and in that of optimizing its syntax to avoid unnecesary loops in the scripting language.

REFERENCES

Burden R and Douglas Faires L., 1998. “Análisis numérico.” 6ta. Edición. Thomson Editores.
The Cygwin Project. Cygwin DLL 2.1.0. Downloadable from https://www.cygwin.com/
Grothmann R., 2014. “Euler Math Toolbox version 2014-06-26.” Downloadable from http://euler.rene-grothmann.de/.
MathWorks 2014. “MATLAB R2014a.” http://www.mathworks.com/products/matlab/
Frank G. and L. Frank, 2014a. “Designing a Computer Program for Matrix Operations.” 2014 JSM Proceedings - Section on Statistical Computing pp.734-741.
Frank G. and L. Frank, 2014b. “The Mini Matrix Language Project.” https://sites.google.com/site/scientificmini/
Frank L. and G. Frank, 2015. “Advances in the Development of a High-Level Matrix Language.” 2015 JSM Proceedings - Section on Statistical Computing pp.1005-1015.
Olver P., 2010. Orthogonal Bases and the QR Algorithm. Univeristy of Minnesota. http://www.math.umn.edu/∼olver/aims_ / qr.pdf
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/

A. MINI Command Sheet

Instruction	**Description**
`#`	first character for doing a comment.
`a`	display variable `a`. If `a` does not exist, nothing happens.
`a={}`	clear variable `a`. If `a` does not exist, nothing happens.
`mydata={}`	delete file `mydata`. If `mydata.txt` does not exist, nothing happens.
`a=3.1415`	assign (evaluate) the value `3.1415` to variable `a`.
`1=1`	assign (evaluate) the value `1.0` to variable `1` (indeed!).
`a=mydata`	load `mydata.txt` (in matrix format) into `a`. (wrong: `mydata=mydata`. See `a=a`)
`a=a`	save variable `a` to a new file `a.txt`. Overwrites old file.
`b=a`	assign (copy) variable `a` to `b`.
`b<a`	ask if smaller. `1` `b` is smaller than `a`, else `0`.
`b==a`	ask if equal. `1` if equal, `0` if not.
`b>a`	ask if greater. `1` `b` is greater than `a`, else `0`.
`b={a}`	get size and memory position of `a` and copy it to `b` like `(rows,cols,pos)`.
`{a}=b`	create a new variable `a` with dimensions specified by `b=(rows,cols,pos)`.
`{a}=b`	if `a` exists, relocate data specified by `b=(rows,cols,pos)` to `a`. ¡dangerous!
`c=b[a]`	get the elements of `b` sepecified by (vector) `a` and copy it to `c`. (`b=b[a]` is valid)
`b[a]=c`	place data in `c` into the elements of `b` sepecified by `a`. (`b[a]=b` is valid)
`b(a)`	function `b()` applied to `a`. (`tr`, `det`, `sqrt`, `cbrt`, `exp`, `ln`, `cos`, `sin`, `acos`, `asin`, `int`)
`f()`	user defined subroutine specified in `f.min` file.
`b=a'`	transpose matrix `a`.
`c=a+b`	matrix sum.
`c=a-b`	matrix difference.
`c=a*b`	matrix multiplication or scalar by matrix multiplication.
`c=a^b`	`a` multiplied `b` times (`a` is square and `b` is an integer)
`c=a/b`	means ab^{-1}. (b^{-1} obtained by gaussian elimination)
`c=a\b`	means $a^{-1}b$. (a^{-1} obtained by gaussian elimination). Less efficient than `a/b`.
`a=>b`	if `a` is non-zero, then goto label `b`.
`seed()`	seed for random generator.
`vartab()`	list all the current variables.
`vartab(a)`	ask if variable `a` exists. `1` if true, `0` if false.
`build(a)`	link all subroutines in `a.min` and write the result back to `a.min`.