Efficient Solution of Large Fixed Effects Problems Using R

Appendix: Review of Available Solutions

Tom Balmat*
Jerome P. Reiter†

---------- DRAFT 6/20/2017 ----------

Following is a review of several popular functions, packages, and techniques implemented in R that are specifically designed for or are suitable for solving fixed effects OLS problems. Use of data and models from actual, current research attempts to measure and contrast the performance of the various solutions in, not simply a realistic setting, but one which has presented computational challenges. Deviations in numerical estimate values beyond the fourth decimal digit (in scientific notation) are considered insignificant. As a general rule, reasonable execution time is measured in minutes, and many implementations achieve this with example problems and in solving model 1. In some cases, execution times of four or more hours was observed, prompting termination prior to completion. These conditions are reported. All processing was executed using version 3.2.1 of R x64 on a dedicated 24 core MS Windows 7 server with 64 Gb of RAM supplied by the Social Science Research Institute at Duke University (cite SSRI). Individual package versions are listed as appropriate.

Solution: `lm()` and `glm()` Functions

* Social Science Research Institute, Duke University, Durham, NC 27708 (thomas.balmat@duke.edu)
† Department of Statistical Science, Duke University, Durham, NC 27708 (jerry@stat.duke.edu);

`lm()` (R 2017 `lm`) and `glm()` (R 2017 `glm`) are the standard linear and generalized linear model fitting functions in R and are included in the base distribution. `glm()` includes methods for iterated weighted least squares and logistic regression models and is not considered here. According to the R documentation, `lm()` calls the function `lm.fit()`* which, in turn, calls internal QR decomposition routines (R 2017 `lm.fit`)†. Effectively, `lm()` takes a rather straightforward linear algebraic approach to solving an OLS problem. However, it requires the expanded design matrix (fixed effects expanded to binary indicator columns) to exist completely in memory. Recall that our sample data set generates a design matrix of dimension 24,574,480 x 1,236, requiring approximately 230 Gb for each of X' and X. Respecifying the OPM variables *sex*, *race*, *bureau*, *occupation*, and *year* as factors with

```
X[,"Sex"] <- relevel(factor(X[,"Sex"]), ref="M")
X[,"Race"] <- relevel(factor(X[,"Race"]), ref="E")
X[,"BureauID"] <- relevel(factor(X[,"BureauID"]), ref="114009000")
X[,"Occupation"] <- relevel(factor(X[,"Occupation"]), ref="303")
X[,"FY"] <- relevel(factor(X[,"FY"]), ref="1988")
gc()‡
```

consumes no additional memory because columns are simply recoded, not expanded. However, executing a call to `lm()`, specifying model 1 as follows:

```
> # report R version
> sessionInfo()$R.version$version.string
[1] "R version 3.2.1 (2015-06-18)"
```

* This is also verified by inspecting the source code of `lm()` with the R command edit(lm)
† Although cryptic, `edit(lm.fit)` reveals the instruction `z <- .Call(C_Cdqrls, x, y, tol, FALSE)`, which is a call to the C based QR linear system solution function
‡ `gc()`, or garbage collector, frees unused allocated memory and is necessary due to R's recreation of objects on reassignment (cite a ref)

```
> # attempt to fit model using lm
> m <- lm(lnBasicPay ~ Sex + Race + Sex*Race + Age + I(Age**2) +
EducationYears + BureauID + Occupation + FY)
```

results in an insufficient memory message, "`Error: cannot allocate vector of size 225.6 Gb`", and the OLS problem is not solved.

There exists a function, `update()`, that can be used to modify a model and its estimates. A useful feature would be the ability to develop a model in stages, introducing a within memory capacity subset of observations in each stage such that parameter estimates and standard errors are current given all prior stages. Once the final stage is executed, the complete model is available. However, the following script indicates that additional observations are not included in the updated model (it appears that the "updated" model, containing only 500,000 observations is simply the result of `lm()` using the final data specification):

```
> # fit model
> t <- proc.time()
> m <- lm(lnBasicPay ~ Sex + Race + Sex*Race + Age + I(Age**2) +
EducationYears + FY + BureauID+Occupation, data=X[1:1000000,])
> proc.time()-t
   user  system elapsed
 557.81    2.57  560.32
> # report number of observations
> length(m$fitted.values)
[1] 1000000
> # update model
> m <- update(m, lnBasicPay ~ Sex + Race + Sex*Race + Age + I(Age**2) +
EducationYears + FY + BureauID+Occupation, data=X[1000001:1500000,])
```

```
> # report number of observations
> length(m$fitted.values)
[1] 500000
```

Further, if composite development were possible, the execution time of nearly 10 minutes* for a 1,000,000 observation subset projects to a total execution time of approximately 6 hours for our problem, which is inefficient and prohibitive.

Solution: `biglm` Package

The `biglm` package, available from the comprehensive R archive network (CRAN 2017 `biglm` package), solves linear and generalized linear models in what the authors term "bounded memory" such that memory requirements are less than p^2, where p is the number of variables in the model which, presumably, includes all dependent, independent, and expanded indicator variables (R 2017 `biglm` documentation). From cited references, the method of solution appears to be Algorithm AS274 (Applied Statistics (1992) Vol.41, No. 2), **which is a collection of algorithms optimizing solution of a linear system of equations and Cholesky factorization**, but it is not clear exactly which sub-algorithms are employed. The units of the p^2 memory upper bound (Mb, Gb, etc.) are not specified, but fitting our model to 1,000,000 observations, after conversion of fixed effects to factors as in the `lm()` example, increases memory usage by approximately 20 Gb. The `biglm()` variant of `update()` includes a `moredata` option that revises a model by adding additional observations, making it possible to fit a model to data sets of arbitrary size. The following script uses `biglm()` to fit the OPM pay disparity model to the

* Numerical portions of performance results are given as numbers for ease of comparison

first 1,000,000 observations in the data set then uses `update()` specifying the second set of 1,000,000 observations:

```
> library(biglm)
> report biglm version
> sessionInfo()$otherPkgs$biglm$Version
[1] "0.9-1"
> # fit model
> # note that OPM fixed effects have been converted to factors in X
> t <- proc.time()
> m <- biglm(lnBasicPay ~ Sex + Race + Sex*Race + Age + I(Age**2) +
EducationYears + FY + BureauID + Occupation, data=X[1:1000000,])

> proc.time()-t
   user  system elapsed
1086.30    7.97 1093.95
> # report number of observations
> m$n
[1] 1000000
> # update model
> t <- proc.time()
> m <- update(m, moredata=X[1000001:2000000,])
> proc.time()-t
   user  system elapsed
1096.00    8.91 1104.64
> # report number of observations
> m$n
[1] 2000000
```

The increase from an initial n = 1,000,000 to n = 2,000,000 observations indicates that our model was updated, but at an expense of approximately 18 minutes per 1,000,000 observations, giving

an expected execution time for the entire data set of over 7 hours, which is excessive. An additional consideration when using `update()` is that, although factors are permitted, levels must be the same across all subsets (R 2017 `biglm` documentation). This involves additional preparation prior to calling `update()`, guaranteeing that each subset contains observations representing each level of each fixed effect.

The `biglm` package includes a function, `bigglm()`, for solving generalized linear models that features a `chunksize` option that automates data subsetting update of the model (R 2017 `biglm` documentation). The following script solves model 1, using `bigglm()` with a `chunksize` of 1,000,000 (observations):

```
> t <- proc.time()
> m <- bigglm(lnBasicPay ~ Sex + Race + Sex*Race + Age + I(Age**2) +
EducationYears + FY + BureauID + Occupation, data=X, chunksize=1000000)
> proc.time()-t
    user   system  elapsed
49288.78   364.18 49640.92
```

As with `biglm()`, memory usage was approximately 20 Gb throughout. Total execution time was over 13 hours, which is inefficient and prohibitive.

p^4 memory for Huber/White sandwich matrix

Solution: `bigmemory` and `biganalytics` Packages

The `bigmemory` and `biganalytics` packages, available from the comprehensive R archive network (CRAN 2017 `bigmemory` package; CRAN 2017 `biganalytics` package), contains

functions for creating what it calls "massive matrices" using shared memory and memory mapped files (R 2017 `bigmemory` documentation) and fitting linear models from data that are too large to fit in memory (R 2017 `biganalytics` documentation). The term "massive" is subjective, of course, and data sets as small as dimension 100,000,000 x 5, for which `biglm()` appears targeted, can be considered large and frustrating (Emerson, Kane YYYY). For comparison, a design of 100,000,000 x 5 (random continuous variables) was solved using `lm()` on our test server (combined with a random 100,000,000 element dependent vector), which required approximately 10 minutes of compute time and 10 Gb of memory. Although involving fewer observations, the OPM pay disparity data set and model involve over 20 times the number of expanded variables, making it considerably larger and, as has been demonstrated, not a candidate for `lm()`. Construction of a `big.matrix` is efficient, requiring less than 2 minutes, as seen here:

```
> library(bigmemory)
> report bigmemory version
> sessionInfo()$otherPkgs$bigmemory$Version
[1] "4.5.8"
> library(biganalytics)
> sessionInfo()$otherPkgs$biganalytics$Version
[1] "1.1.12"
> # convert OPM columns to factors
> t <- proc.time()
> X <- as.big.matrix(
         data.frame(
           "Age"=opm[,"Age"],
           "AgeSq"=opm[,"Age"]*X[,"Age"],
           "EducationYears"=opm[,"EducationYears"],
```

```
            "Sex"=relevel(factor(opm[,"Sex"]), "M"),
            "Race"=relevel(factor(opm[,"Race"]), "E"),
            "FY"=relevel(factor(opm[,"FY"]), "1988"),
            "BureauID"=relevel(factor(opm[,"BureauID"]), "114009000"),
            "Occupation"=relevel(factor(opm[,"Occupation"]), "303"),
            "lnBasicPay"=opm[,"lnBasicPay"]))
Warning message:
In as.big.matrix(data.frame(Age = opm[, "Age"], AgeSq = opm[ :
  Coercing data.frame to matrix via factor level numberings.
> proc.time()-t
   user  system elapsed
  98.44   16.93  115.71
> # report dimensions of big matrix
> dim(X)
[1] 24574480       13
```

Fixed effects are converted to numeric factor levels (note the coercion warning – it is assumed that factor levels correspond to those generated by `factor()`), but indicator columns are not generated, as indicated by a column count of 13. The following script uses `biglm.big.matrix()` and the previously generated `big.matrix X` to solve the OPM fixed effects problem:

```
> t <- proc.time()
> m <- biglm.big.matrix(lnBasicPay ~ Sex + Race + Sex*Race + Age +
AgeSq + EducationYears + FY + BureauID + Occupation, data=X,
fc=c("Sex", "Race", "FY", "BureauID", "Occupation"))
> proc.time()-t
    user   system  elapsed
24141.69   188.98 24324.01
```

Note that the `fc` parameter specifies factors (fixed effects in our model). Since all columns of `X` are numeric, failure to declare factors with `fc` results in solution of a simple 13 continuous variable problem, which is inappropriate. Total execution time is nearly 7 hours, which is excessive. According to `biganalytics` documentation `biglm.big.matrix()` is a wrapper for `biglm()`, enabling it to operate on "massive data" stored in a `big.matrix` object (R 2017 `bigmemory` documentation), so completions times comparable to those of `biglm` are, perhaps, expected. Additionally, `as.big.matrix()` creates what is called a "file-backed" matrix, where an external operating system file is created (in C:\ProgramData\boost_interprocess on our test server) to contain the actual matrix contents. This file is deleted upon removal of the corresponding R `big.matrix` object (possibly as part of normal termination of R), but must be manually deleted if R terminates abnormally. The `bigmemory` package supports explicit creation of a file-backed big matrix of arbitrary size. The idea of creating a file-backed matrix followed by programmatic expansion of factors into indicator columns was attempted, but file system space sufficient to contain a 24,574,480 `X` 1,236 matrix did not exist on our test server, and the idea was abandoned.

An interesting observation is that, using `biglm.big.matrix()` with subsets of varying size, computation time appears to be proportional to the number of observations in the subset, as opposed to being exponential or requiring increasing time per observation as the number of observations increases. Figure AS-1 plots computation time against observation count for randomly selected subsets of size 1,000; 10,000; 100,000; 1,000,000; 10,000,000; and 25,000,000 and indicates a linear association of approximately 16 minutes execution time per 1,000,000 observations, independent of the number of observations. Note that both axes have

been converted to $\log_{10}$ scale, presenting small and large subsets in a single plot, while maintaining linear scale of axes. **WHAT IS THE METHOD OF SOLUTION? IS LINEARITY OF ORDER A FEATURE OF THIS? X COL DIM IS ROUGHLY CONSTANT, MAKING ROW AND COL MULTIPLIES CONSTANT, ALTHOUGH INCREASED SET SIZE => INCREASED VECTOR SIZE.**

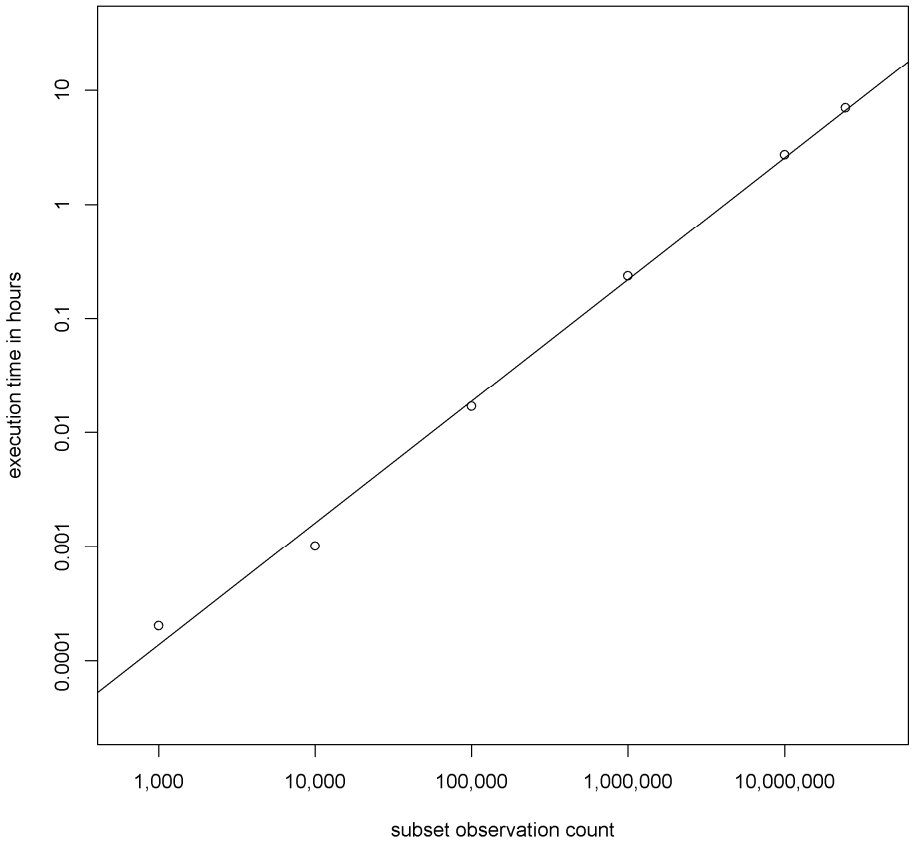

Figure AS-1. `biglm.big.matrix()` solution time vs. observation count

It should be noted that the `bigmemory` package has many useful functions for analyzing `big.matrix` objects, such as `apply()` variants and various aggregation equivalents, but, unfortunately, they do not help us solve our large fixed effects OLS problem.

Solution: lfe

Solution: sparseM

Solution: speedlm

Solution: fastLm

Solution: Demean Predictor Variables within Fixed Effects Levels

Solution: Sparse Matrix Functions from Matrix an Matrix Models Packages

Available Solutions Appendix References

CRAN 2017 biglm package, https://cran.r-project.org/web/packages/biglm/index.html

R 2017 biglm documentation, https://cran.r-project.org/web/packages/biglm/biglm.pdf

Algorithm AS274, http://www.jstor.org/stable/2347583?seq=1#page_scan_tab_contents

Algorithm AS274 Applied Statistics (1992) Vol.41, No. 2

CRAN 2017 bigmemory package, https://cran.r-project.org/web/packages/bigmemory/index.html

R 2017 bigmemory documentation, https://cran.r-project.org/web/packages/bigmemory/bigmemory.pdf

CRAN 2017 biganalytics package, https://cran.r-project.org/web/packages/biganalytics/index.html

R 2017 biganalytics documentation, https://cran.r-project.org/web/packages/biganalytics/biganalytics.pdf