

An Efficient Algorithm for Solving Large Fixed Effects Problems

Tom Balmat*, Jerome P. Reiter†

July 27, 2017

Abstract

The standard method for solving Ordinary Least Squares (OLS) regression problems in R is to use `lm()` and, for small problems, involving fewer than 1,000,000 observations and 25 independent variables, this is efficient. But for larger problems, involving tens of millions of observations and thousands of independent variables, execution of `lm` becomes computationally impractical due to system memory requirements and execution time. Large fixed effects regression problems, involving effects with thousands of levels, present a special performance opportunity because of the large proportion of entries in the expanded design matrix (fixed effect levels translated from single columns into dichotomous indicator columns, one for each level) that are zero. For many problems, the proportion of expanded design matrix entries that are zero is above 0.995, which would be considered sparse. In this paper, we present an efficient method for solving a large, sparse fixed effects OLS problem without creation of the expanded design matrix and avoiding computations involving zero-level effects. This leads to minimal memory usage and optimal execution. A feature, often desired in social science applications, is to estimate parameter standard errors clustered about a key identifier, such as employee ID, and large problems, with ID counts in the millions, present significant computational challenges. We present a sparse matrix indexing algorithm that produces clustered standard error estimates that, for large fixed effects problems, is many times more efficient than standard sandwich matrix operations.

Keywords: fixed effects least squares solution, high dimension models, sparse matrix methods, parallel computing, clustered standard error estimation

The authors would like to thank Alex Bolton, Andres (Felipe) Barrientos, and John de Figueiredo for sharing data and models from their research and to the IT staff at Duke University’s Social Science Research Institute for high performance computing support.

*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)

1 Introduction

The classic method of estimating Ordinary Least Squares (OLS) model parameters is to solve $\mathbf{X}'\mathbf{X}\hat{\beta} = \mathbf{X}'\mathbf{Y}$ for $\hat{\beta}$. For small designs, where $\mathbf{X}(n \times p)$ involves n and p on the order of 10^4 and 10^1 , respectively, this system is efficiently solved in R using the `lm()` function, while larger designs, where $n \times p$ is on the order of $10^7 \times 10^3$, present special storage and computational challenges.¹ For example, creation of a 50,000,000 \times 2,000 numeric design matrix in R, which encodes in double floating point format (eight bytes per cell), requires $10^{11} \times 8$ bytes, or approximately one terabyte, of memory. Further, to generate $\mathbf{X}'\mathbf{X}$ with standard R functions, that is using `t(X)%*%X`, requires an additional copy of \mathbf{X} for the transpose, for a total memory requirement of approximately two terabytes (note that standard R functions require objects to be completely contained in on-line memory).² Physical memory constraints are easily overcome by using R x64 (the 64 bit version with increased memory addressing) installed on a system with on-line memory sufficient for the problem to be solved. In addition to memory constraints, solution feasibility is practically limited by the processing time required to compose $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}'\mathbf{Y}$ and finally to solve the corresponding OLS equation. For even large designs, on the order of $10^7 \times 10^3$, construction of $\mathbf{X}'\mathbf{Y}$ and solution of $\mathbf{X}'\mathbf{X}\hat{\beta} = \mathbf{X}'\mathbf{Y}$, using QR decomposition through standard R functions `qr()` and `qr.coef()`, are accomplished in minutes, leaving the majority of processing time to the construction of $\mathbf{X}'\mathbf{X}$. *(The story changes a bit with fixed effects of dimension above 30,000, where Cholesky decomposition gives a significant performance advantage over QR, but that seems a bit too much detail for the intro).* Given system memory sufficient to store \mathbf{X}' , \mathbf{X} , and necessary intermediate computational values, the time required to execute necessary row and column products can be significant (our $10^7 \times 10^3$ example involves order 10^6 row-column products, each consisting of order 10^7 cell products and additions, for a total operation count of order 10^{13}). This paper presents a method for constructing $\mathbf{X}'\mathbf{X}$ that is very efficient in a fixed effects OLS model environment, particularly in a sparse, high observation count (order 10^7 and above) and high dimension fixed effects (levels of order 10^2 and above) setting, achieving solution times measured in minutes.³ Additionally, an efficient algorithm is presented for estimating robust, clustered standard errors.⁴ Traditional clustered standard error estimation involves the variance equation $\text{Var}(\hat{\beta}) = \text{diag}[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{U}\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}]$, where \mathbf{U} is the $n \times n$ matrix of within cluster residual covariances, and is estimated by setting all inter-cluster errors, $(\epsilon'\epsilon)_{\text{cluster}}$, to 0 (on the assumption of inter-cluster independence).⁵ For observation counts, n , on the order of 10^7 and fixed effects on the order of 10^3 the $\text{Var}(\hat{\beta})$ equation involves operations on the order of 10^{19} , which is generally impracticable using standard linear algebra operations. The algorithm presented uses a sparse indexing method along with piece-wise summing of cluster row-column products in solving $\text{Var}(\hat{\beta})$ to achieve solution

¹`lm()` is the base linear regression function of R (R Foundation for Statistical Computing, 2017, R Reference, `lm()`)

²The additional copy can be confirmed by monitoring total memory usage while executing the R command `sum(t(X)%*%X)`

³For more on sparse matrix principles and methods, see Koenker and Ng (2003) and Bates and Maechler (2017)

⁴Robust and clustered standard errors are popular in social science models, where influential independent variables are correlated within groups or subjects, but not explicitly included (generally not available) in the model (for more, see Cameron and Miller (2015) and King (2015))

⁵There exist methods of estimating robust standard errors that avoid direct evaluation of the above matrix equation; a popular one is known as the ‘‘Huber Sandwich Estimator’’ (Freedman, 2006; Zeileis, 2006; Esarey and Menger, 2017)

times measured in tens of minutes for problems on the order of those mentioned above.

2 Data Sets, Models, and Research

It is said that the social sciences are in the midst of a “data revolution,” that “an explosion of information is upon [it],” and that new methods are emerging to “leverage the capacity to collect and analyze data with an unprecedented breadth and scale” (Connelly et al., 2016; Liu and Guo, 2016; Lazer et al., 2009). However, many recently developed methods, such as machine learning and text analysis, appear to be a direct response to the availability of and targeted to large, unstructured on-line data sets (Bloomberg, 2016; Economist, 2015). For research in the social sciences using large structured data sets, such as the Current Population Survey (U.S. Census Bureau, CPS), GDP forecasts (U.S. Bureau of Economic Analysis, GDP), and various socioeconomic data sets made available by the U.S. government (U.S. General Services Administration, Data.gov), least squares regression, which estimates the expected value of a dependent variable given combinations of levels of a set of co-varying independent variables, remains a popular model fitting method. Since the time of Legendre and Gauss, additional, specialized methods, such as quantile and logistic regression, have been developed and the original method has become known as ordinary least squares (OLS) regression.⁶ A common feature in social science research is to employ models with categorical independent variables, such as gender, ethnicity, region, industry, and employment status. When all categorical variable levels of interest are represented in the data, they are termed fixed effects and the model becomes a fixed effects OLS model.⁷ Because fixed effects are not necessarily ordinal or even numerical, parameter estimates for individual levels provide an estimate of the difference in mean response between the group of observations for a given level and those for a corresponding reference level, for instance difference in mean annual wages between women and men, where men are the reference category. Also, because the fixed effects OLS model produces response estimates with minimum sum of squared deviations to within-level mean, when the effect of level itself is not being studied, a fixed effect can be considered a control variable, where the response is adjusted for mean effect of individual levels prior to estimating model parameters of interest. An example of this would be a model to measure disparity in pay by gender (one fixed effect), while controlling for geographic region, occupation, experience category, and race (controlling effects). Interesting examples of models that use categorical variables for control or study include the Salk polio vaccine studies where individual family fixed effects were modeled to account for confounding of sibling health and hygiene practices (Juskewitch et al., 2010) and a study of youth traffic fatalities with respect to the minimum drinking age imposed by state law, where differences in fatality rates were measured by a variety of categorical demographic classifiers, such as sex, race, urbanicity, and affluence (O’Malley and Wagenaar, 1990).

⁶Both Legendre and Gauss are credited with, and claimed discovery of, the method of least squares (Stigler, 1986)

⁷This is distinct from fixed effects in a *panel data* sense, where a fixed effect controls for time invariant effects (Williams, 2015)

3 Value of Efficient Methods in Research

In the *Performance of Solutions with Simulated Data and Models* section below, eight methods of solving fixed effects problems are evaluated. For large problems, only two demonstrate efficiency sufficient to be considered practical. When applied to the largest problem considered, one of the efficient solutions outperforms the other by a factor of eight, solving the problem in ninety minutes as opposed to twelve hours. This difference in performance presents an opportunity to solve five problems per workday compared to two problems in three days. Alternatively, five researchers have an opportunity to solve one problem per day compared to two researchers in three days, enabling exploration of a greater variety of ideas, data sets, and verification plans, translating to an overall increase in research output. Additionally, higher efficiency should reduce operating costs due to increased utilization of organizational information technology and computation resources.⁸

4 Motivation for Sparse Matrix Methods

Consider the trivial example of composing $\mathbf{X}'\mathbf{X}$, where \mathbf{X} is a $1,000,000 \times 2,000$ matrix with all elements equal to 0. Of course, the result is a $2,000 \times 2,000$ matrix of zeros since all row and column products are zero, and we instantly translate the all element zero case to an all zero result. But R makes no such inference. It naively checks data types and dimensions then begins the task of summing the products of billions of row and column elements, each known by us to be zero. In trials, creation of $\mathbf{X}'\mathbf{X}$ from a $10,000 \times 2,000$ matrix \mathbf{X} with

```
X <- matrix(data=0, nrow=20000000, ncol=2000)
Z <- t(X)%*%X
```

requires over one hour to complete. A simple alternative solution to this trivial case is

```
Z <- matrix(data=rep(0, 4000000), ncol=2000)
```

which involves no math operations and executes immediately.

Suppose that we are given a large, sparse matrix \mathbf{X} ($25,000,000 \times 2,000$ with a high proportion of elements equal to 0) along with a list of elements (row i , column j) that are non-zero. An efficient alternative to generating $\mathbf{X}'\mathbf{X}$ with matrix operations is to accumulate products of pairs of non-zero elements into corresponding positions of the resulting $\mathbf{X}'\mathbf{X}$ matrix.⁹ Since $\mathbf{X}'\mathbf{X}$ is symmetric, additional efficiency is gained by accumulating the upper triangle only, then copying its transpose to the lower triangle.

⁸For an interesting related discussion regarding efficient computation and cost of motion, see Ramkumar et al. (2016)

⁹The standard matrix approach in R is `Z <- t(X)%*%X`

5 Example Data Set and Model

The U.S. Office of Personnel Management (OPM) maintains employment and human capital records for over one million non-DOD U.S. federal employees in what it calls the central personnel data file (CPDF).¹⁰ CPDF “Status” records for fiscal years 1988 through 2011 were supplied by OPM to the Human Capital Project at Duke University in response to a Freedom of Information Act request.¹¹ Over 28,000,000 records were provided, each describing the human capital profile of active federal employees as of September 30 of the corresponding fiscal year; there exists one record per employee per year. Important data elements are: sex, race, age, education level, bureau employed in, occupation, and fiscal year of observation (terminology is from OPMs Guide to Data Standards).¹² Current research using these data have identified systematic changes in federal employee pay over time, disparities in federal employee pay by gender, and association of elections, political ideology to turnover among federal employees (Bolton and de Figueiredo, 2017; Bolton et al., 2016). For illustration, we will use a fixed effects model (similar to those used by Bolton and de Figueiredo) that measures disparity in pay by employee declared gender and race. The model is

$$y = \beta_0 + \beta_{sex} + \beta_{race} + \beta_{sex,race} \times sex \times race + \beta_{age^2} \times age^2 + \beta_{ed} \times ed_{years} + \beta_{bureau} + \beta_{occ} + \beta_{year} \quad (1)$$

where y is the logarithm of *basic pay* (OPM variable for pay) and the β ’s are linear effects of an employees *sex*, *race*, *sex* and *race* interaction, *age*, square of *age*, *education* (years beyond HS), *bureau* (agency employed in), *occupation*, and *fiscal year*. *Sex*, *race*, *bureau*, *occupation*, and *year* are discrete fixed effects while *age* and *education* are continuous. Fixed effects parameter estimates measure the difference in expected value of $\log(\text{basic pay})$ between a given level and a reference level. Table 1 lists reference levels for each effect. For this model, observations are limited to full time employees with non-zero *basic pay* values and valid, non-empty values in each independent variable, giving a total included record count of 24,574,480. In constructing the design matrix \mathbf{X} , each fixed effect C_j is expanded into $p_j - 1$ indicator columns, one for each non-reference level of C_j , where p_j is the number of distinct levels of C_j . Element (row) i of indicator column $X_{j,k}$ corresponding to level k of fixed effect C_j contains a 1 if observation i is encoded with the k th level of C_j and contains a 0 otherwise. Table 1 lists the number of non-reference levels by effect. Including columns for the constant, continuous, and interaction variables gives a design matrix of dimension 24,574,480 x 1,228. Levels within a fixed effect are mutually exclusive since an observation can be encoded for a single level only, so that each row of the design matrix \mathbf{X} has at most one column for each fixed effect coded as a 1, with the remainder necessarily containing 0. Large p_j values cause overall low density (proportion of 1 entries) in \mathbf{X} and it is this property that we exploit in efficient construction of $\mathbf{X}'\mathbf{X}$. Table 1 also lists mean density by fixed effect (average density of columns associated with each effect). Overall design matrix density in the OPM data set relative to model 1 is 0.006, making it quite sparse and a good candidate for algorithmic and

¹⁰For a general description of OPM data resources, see U.S. Office of Personnel Management (a)

¹¹For more on CPDF Status records, see U.S. Office of Personnel Management (b)

¹²See OPM Guide to Data Standards, U.S. Office of Personnel Management (c)

solution comparison.

Table 1: Pay Disparity Fixed Effect Reference Levels, Number of Non-ref Levels, and Density

| Effect | Reference Level | Non-Ref Levels | Density |
|------------|---|----------------|---------|
| Sex | M (Male)* | 1 | 0.4800 |
| Race | E (White)* | 4 | 0.0821 |
| Bureau | 114009000 (Veterans Administration)** | 409 | 0.0020 |
| Occupation | 303 (Miscellaneous Clerk and Assistant)** | 791 | 0.0012 |
| Year | 1988*** | 23 | 0.0420 |

* customary reference levels for pay equity research

** highest marginal frequency

*** first year in study data

6 Review of Available Solutions

Many statistical software programs recognize fixed effects, or categorical variables, in solving regression problems. Popular solutions such as `proc glm` from SAS (cite), the `regress` command of Stata (cite), and the `lm()` function of R (R Foundation for Statistical Computing, 2017, `lm()`) implement a common method of expanding categorical variables into columns of the design matrix, one for each level, eliminating the column corresponding to a user declared or system selected reference level to avoid linear dependence, and finally solving the normal equations $\mathbf{X}'\mathbf{X}\hat{\beta} = \mathbf{X}'\mathbf{Y}$ for $\hat{\beta}$.¹³ The *Review of Available Solutions* appendix evaluates popular software solutions for solving large OLS regression problems.¹⁴ Since this paper targets an R audience, solutions are limited to R functions and packages available at time of writing, including `lm()`, `biglm()`,¹⁵ `bigglm()`,¹⁶ `biglm.big.matrix()`,¹⁷ `SparseM.slm.fit()`,¹⁸ `speedlm()`,¹⁹ and `lfe()`.²⁰ Each addresses efficiency obstacles in a variety of ways, some targeting memory constraints only, others addressing both efficient computation and use of memory. Some employ parallel processing methods, others create external operating system files when a design matrix exceeds on-line memory capacity. Example data sets and models are employed to illustrate performance features of various solutions. Additionally, model 1 is fit, when possible, using the example OPM data set described above. Included in the Solutions appendix is a problem re-specification alternative commonly known as *demeaning*, where continuous independent variables

¹³Verification of fixed effect expansion into binary columns with that of the reference level omitted can be accomplished with the `model.matrix()` function in R (R Foundation for Statistical Computing, 2017, R Reference, `model.matrix()`) and the `base` option of the `regress` command in Stata (Stata Corporation, 2017, `regress`).

¹⁴Available in the on-line repository (Duke University Synthetic Data Project, 2017, `feXTX()` git Repo)

¹⁵See Lumley (2015)

¹⁶See Lumley (2015)

¹⁷See Emerson and Kane (2016)

¹⁸See Koenker and Ng (2003)

¹⁹See Enea et al. (2017)

²⁰See Gaure (2016)

are transformed to deviations from within-level means. The technique has advantages and penalties (cite) and is included for completeness, covering the simple case where, say, only continuous variable parameter estimates are desired, without estimates of standard errors. Demeaning can transform a complex, high dimension, computationally intensive fixed effects structure into a simple, low p -dimension OLS problem, easily computed with `lm()`, even with 100,000,000 or more observations.

7 Design Matrix and Operation Density

The concept of design matrix density has been introduced along with a suggested relationship to efficient solution methods. More specifically, a low density design matrix presents an opportunity to limit computations strictly to those that affect the final result, in the case of matrix multiplication, products that are non-zero. The idea is to identify matrix cells, or elements, that contain non-zero entries and operate solely on those. If the proportion of non-zero entries is low (a low density matrix) then the ratio of effective operations to total operations is high and computation is efficiently executed. Consider a design matrix of dimension $n \times p$, such that p consists of p_x columns for continuous variables (including a constant) and p_k columns, one for each level of a single fixed effect. Note that the fixed effect actually has $p_k + 1$ levels, but one "reference" level is omitted to avoid linear dependence of the fixed effect and constant columns. Effect levels are mutually exclusive within observation since, at most, one level applies. Also, reference level observations have all fixed effect columns coded as 0. Therefore, within the np_k fixed effect cells, the maximum number of cells coded as 1 is $n - 1$ (at least one reference level observation must exist), giving a density upper bound within the fixed effect columns of $D_{M_k} \leq \frac{n-1}{np_k} < \frac{n}{np_k} = \frac{1}{p_k}$. Letting d_j be the density of the j th continuous column, overall design matrix density has an upper bound of $D_M \leq \frac{n[\sum_{j=1}^{p_x} d_j] + n - 1}{np}$. Assuming high density in the continuous columns (near 1.0), this becomes

$$D_M \leq \frac{np_x + n - 1}{np} < \frac{p_x + 1}{p} = \frac{p_x + 1}{p_x + p_k}.$$

It follows that density decreases with increasing fixed effect levels. In fact, the rate of change per additional level (assuming high density continuous columns) of the upper bound of D_M is

$$\begin{aligned} \Delta D_M &= \frac{np_x + n}{n(p+1)} - \frac{np_x + n}{np} = \frac{p_x + 1}{p+1} - \frac{p_x + 1}{p} = -\frac{p_x + 1}{p(p+1)} \propto -\frac{1}{p^2} \quad (\text{approximate}) \\ &= -\frac{1}{(p_x + p_k)^2}. \end{aligned}$$

It is interesting to note that the upper bounds for both D_M and ΔD_M depend solely on p which, for high dimension problems, is dominated by p_k , making density predominantly a function of the number of fixed effect levels. The upper panel of figure 1 plots total design matrix density against number of fixed effect levels from 1 through 500 (less the reference level) for a model with 1, 2, 5, and 10 continuous variables

(plus a constant term) and a single fixed effect. It is seen that density rapidly decreases as the number of levels increases, presenting efficiency opportunities for all but the lowest dimension problems. In a multiple fixed effects problem, with m effects and $p_K = \sum_{i=1}^m p_k i$ fixed effects columns, density is further reduced since np_K cells are populated with a maximum of $m(n-1)$ non-zero values. Given m fixed effects, an upper bound on matrix density is

$$D_M \leq \frac{np_x + m(n-1)}{n(p_x + p_k)} < \frac{p_x + m}{p_x + p_k} = \frac{p_x + m}{p}$$

The lower panel of figure 1 shows, for models with 5 continuous variables and 1, 2, 5, or 10 fixed effects, significant reduction in density as the combined number of levels, p_k , increases.

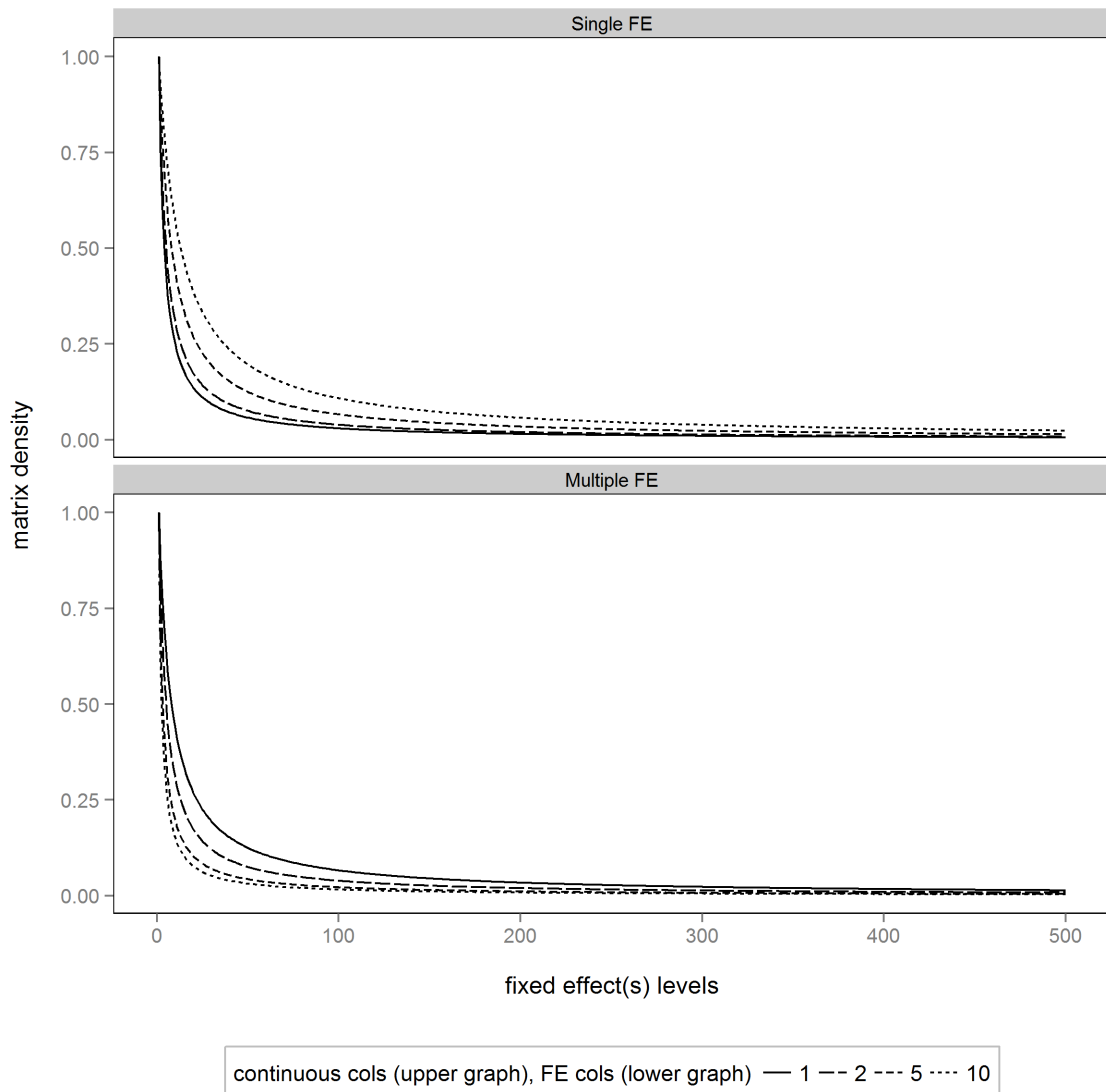


Figure 1: Design Matrix Density vs. Increase In Levels of Fixed Effects. Upper panel, one to ten continuous columns with a single fixed effect. Lower panel, a single continuous column with one to ten fixed effects with common dimension (number of levels).

Design matrix density measures the proportion of elements that contribute information to and cannot be eliminated from the $\mathbf{X}'\mathbf{X}$ operation. Figure 1 indicates that, for even low dimension fixed effects, the proportion of essential matrix elements is relatively small, typically less than 0.01 for problems in the class we consider. However, design matrix density merely characterizes an opportunity for efficient computation. More important than matrix density itself is what we will call operation density, the proportion of standard matrix operations (sums of products of row and column elements) that contribute to the resulting $\mathbf{X}'\mathbf{X}$ product. Recall that, given $\mathbf{X}(n \times p)$, the matrix product $\mathbf{X}'\mathbf{X}$ has $p \times p$ cells, the ij^{th} being the product $\mathbf{X}'_i\mathbf{X}_j$, which involves the addition of n element products, for a total of $n + n - 1$ basic operations (n

multiplication and $n - 1$ addition operations). The total number of basic operations to produce $\mathbf{X}'\mathbf{X}$, using standard matrix methods, is then $(2n - 1)p^2$. Consider a model with p_x continuous variables, each having (matrix) density of 1.0, and a single fixed effect with p_k levels that are uniformly distributed throughout observations (giving fixed effect indicator column density of $d_k = \frac{1}{p_k}$). The number of operations for products of continuous columns and continuous columns is $(2n - 1)p_x^2$, for products of continuous and fixed effect indicator columns is $(2nd_k - 1)p_x p_k = (2n - p_k)p_x$, and for products of fixed effect and fixed effect columns is $(2nd_k^2 - 1)p_k^2 = 2n - p_k^2$. Operation density for this model is then

$$\begin{aligned} D_O &= \frac{(2n - 1)p_x^2 + (2n - p_k)p_x + 2n - p_k^2}{(2n - 1)p^2} \\ &\leq \frac{(2n - 1)p_x^2 + (2n - 1)p_x + 2n - 1}{(2n - 1)p^2} \\ &= \frac{p_x^2 + p_x + 1}{p^2} \propto \frac{1}{p^2}. \end{aligned}$$

The rate of change, per additional level of the fixed effect, of the upper bound of D_O is

$$\begin{aligned} \Delta D_O &= (p_x^2 + p_x + 1) \left[\frac{1}{(p + 1)^2} - \frac{1}{p^2} \right] \\ &= -(p_x^2 + p_x + 1) \frac{2p + 1}{p^2(p + 1)^2} \propto -\frac{1}{p^3} \quad (\text{approximate}). \end{aligned}$$

Figure 2 shows the relationship of operation density to fixed effect levels. The upper panel plots ΔD_O against fixed effect levels for a model with 1, 2, 5, or 10 continuous variables and one fixed effect. The lower panel plots ΔD_O against fixed effect levels for a model with 5 continuous variables and 1, 2, 5, or 10 fixed effects, each with dimension as indicated on the x -axis. A rapid decrease in ΔD_O is apparent even when fixed effects are of low dimension, further evidence of an opportunity of computational efficiency by operating solely on non-zero cells of \mathbf{X} . It should be noted that an increase in fixed effect levels does not increase the number of non-zero matrix elements or operations, but does increase the number of standard $\mathbf{X}'\mathbf{X}$ operations, hence the relative decrease in ΔD_O . Another important feature of ΔD_O is that since, within a given fixed effect, indicator columns are orthogonal, the p_k^2 elements of $\mathbf{X}'\mathbf{X}$ corresponding to it are necessarily diagonal ($p_k \times p_k$). This means that, for example, a model with four continuous variables (plus a constant) and one fixed effect with 96 levels has $952 - 95 = 8,930$ of 10,000 total $\mathbf{X}'\mathbf{X}$ elements known in advance to be zero. Any operations involving elements from columns i and j within a fixed effect, where $i \neq j$, are unnecessary. Also, since a fixed effect is represented in \mathbf{X} as a series of indicator columns, one for each level, the p_k diagonal elements of $\mathbf{X}'\mathbf{X}$ corresponding to an effect are simply $\mathbf{X}'_j \mathbf{X}_j = n_j$, where j is the indicator column index and n_j is the observation count for the corresponding fixed effect and level. Thus, multiplication operations within fixed effects can be substituted with simple counts.

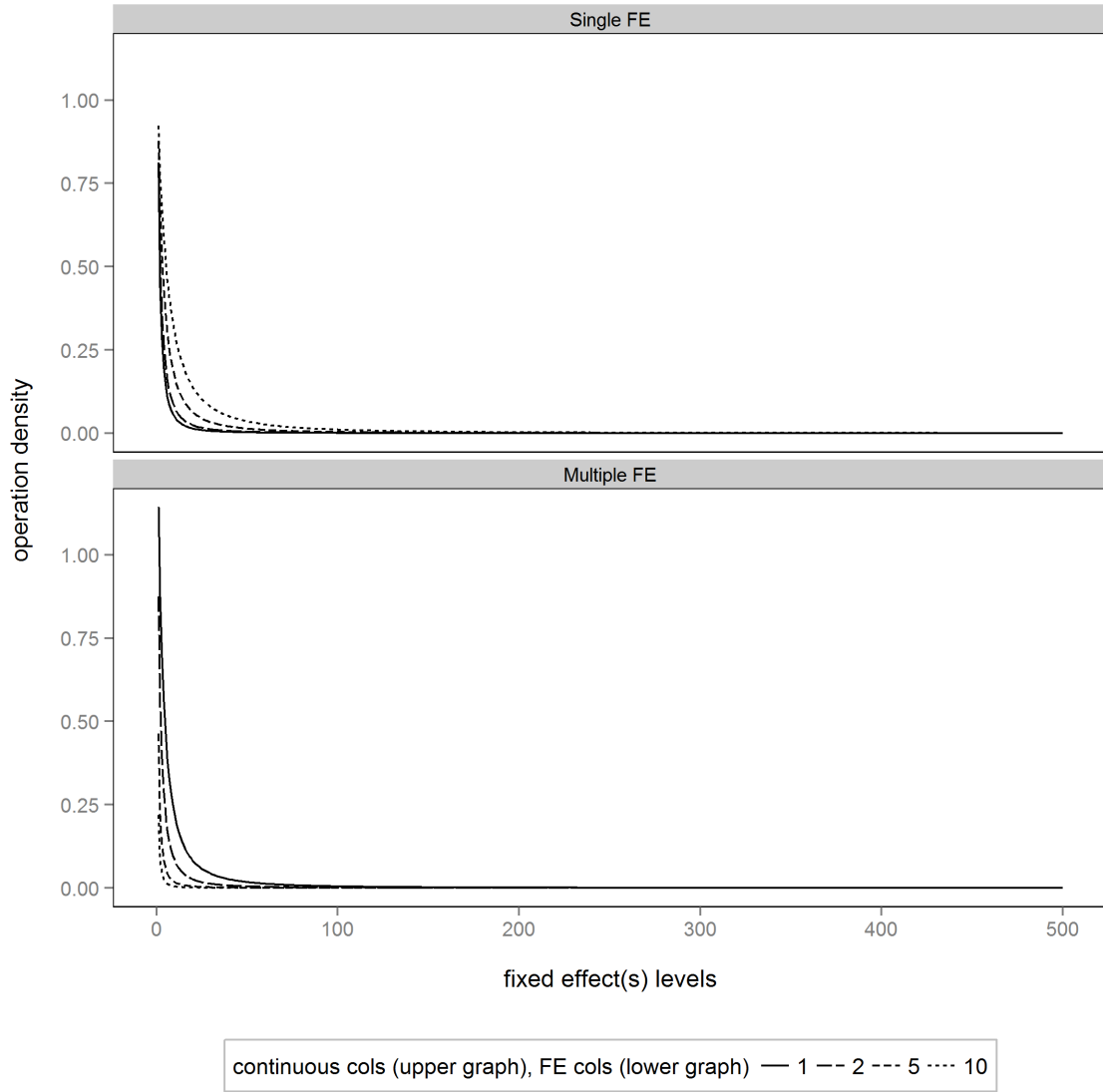


Figure 2: Operation Density vs. Increase in Fixed Effects Levels. Upper panel, one to ten continuous columns with a single fixed effect. Lower panel, a single continuous column with one to ten fixed effects with common dimension (number of levels).

We have established that matrix density is strongly influenced by fixed effect dimension, that operation density is a result of matrix density, and that theoretical measures of operation density indicate an opportunity, for sufficiently sparse problems, to eliminate a high proportion of standard matrix operations in producing $\mathbf{X}'\mathbf{X}$. Analysis of figure 2 indicates that for models involving fixed effects with 50 or more levels, the proportion of effective operations (those that must be executed) approaches zero. Models with fixed effects in this range are very common (50 or more occupations, states, counties, schools, etc.), making implementation of a solution method both relevant and appealing. The example OPM data set and model, with three continuous and five fixed effects variables, has an operation density of approximately 0.00001,

making it a candidate for computational efficiency improvement.²¹

8 Method: Efficient Indexing

When a design matrix, \mathbf{X} , exhibits low operation density, we know that relatively few operations are actually necessary to produce $\mathbf{X}'\mathbf{X}$, or more correctly stated, relatively few elements of \mathbf{X} need to be operated on. The initial challenge, then, is to develop an efficient method of expanding fixed effects into a representation of design matrix indicator columns that index strictly non-zero effect levels in the design. The ideal solution to this is:

- efficient, solving in minutes, problems on the order of the example OPM federal pay disparity model, and
- general, using base R functions, making it readily portable and robust

In R, a simple function for general purpose indexing is `which()`.²² `which()` is generally easy to implement: to generate a vector of indices, positions within a column \mathbf{X} that contain a particular value a , we simply execute `which(X==a)`. By executing `which()` within `lapply()`, once for each unique non-reference level in a fixed effect column, we generate a list of p_k variable length vectors, x_1, \dots, x_{p_k} , one for each fixed effect level, where x_j contains the indices (observation row positions) corresponding to observations coded with effect level j .²³ The density of x_j is $\frac{n_j}{n}$, where n_j is the number of observations coded with level j and n is the total number of observations. The x_j vectors indicate strictly non-zero, effective matrix elements and are the sole ones to be operated on. One drawback to using the `lapply(which())` method is that `which()` is invoked independently for each of the p_k fixed effect levels, requiring a complete scan of a fixed effect column for each level. This is computationally expensive with large p_k . Additional efficiency is gained by executing `lapply()` in parallel, instructing multiple worker processes (cores) to index blocks of levels simultaneously.²⁴ Specifying n_c cores reduces execution time to slightly greater than $\frac{1}{n_c}$, but incurs, in a MS Windows sockets environment, additional cost in time and memory to export entire fixed effect vectors to each core. An alternative is to `order()` observation indices by fixed effect level, identify level transition positions, using `match()`, and parse all indices between transition points, giving sets of non-zero observation indices by effect level.²⁵ For large n , the default implementation of `order()` is relatively inefficient, but its inefficiencies are compensated for by the speed of `match()`. An improvement of `order()` is to convert a fixed effect column to numeric, using `factor()`, then order the resulting indices using `sort.list(method="quick")`.²⁶
²⁷ Since all indices are needed by `order()` and `sort.list()` simultaneously, the `order()`, `sort.list()`,

²¹Operation density can be derived from a constructed $\mathbf{X}'\mathbf{X}$ matrix using the fixed effect indicator product entries, since they report the number of intersecting non-zero elements and indicate the number of pseudo-multiplication operations performed

²²See R Foundation for Statistical Computing (2017), `which()`

²³See R Foundation for Statistical Computing (2017), `lapply()`

²⁴Using `parLapply()` from the `snow` (Tierney et al., 2016) or `parallel` (R-core, 2016) packages

²⁵See R Foundation for Statistical Computing (2017), `order()`, `match()`

²⁶See R Foundation for Statistical Computing (2017), `factor()`

²⁷`sort.list()` is a variant of `order()` (R Foundation for Statistical Computing, 2017, `sort.list()`)

`match()` methods are not good candidates for parallelization. So, which method is most efficient? The answer depends on n , p_k , and n_c . Figure 3 shows simulated computation times, in minutes, to index $n = 25,000,000$ observations in $p_k = 10, 50, 100, 250, 500, 1,000$, and $5,000, 100,000$, and $1,000,000$ random, uniformly distributed fixed effect levels using $n_c = 16$ cores (for `parLapply(which())` method). Results are mean elapsed time (as reported by `proc.time()`) of ten iterations executed on a dedicated server. The clear winner is to convert a fixed effect to a factor then use `sort.list()` and `match()` to parse indices. The important lesson in this exercise is to measure computation time of critical elements of an algorithm and to test alternative methods that exploit high performance features of readily available functions. Although, for indexing, `which()` may come to mind first, it is generally not an efficient method for the problems of the size we consider and, although conversion to (numeric) factors and use of `sort.list()` and `match()` are not as intuitive, they remain base R functions and perform very well, eliminating minutes to hours from overall computation time by comparison.

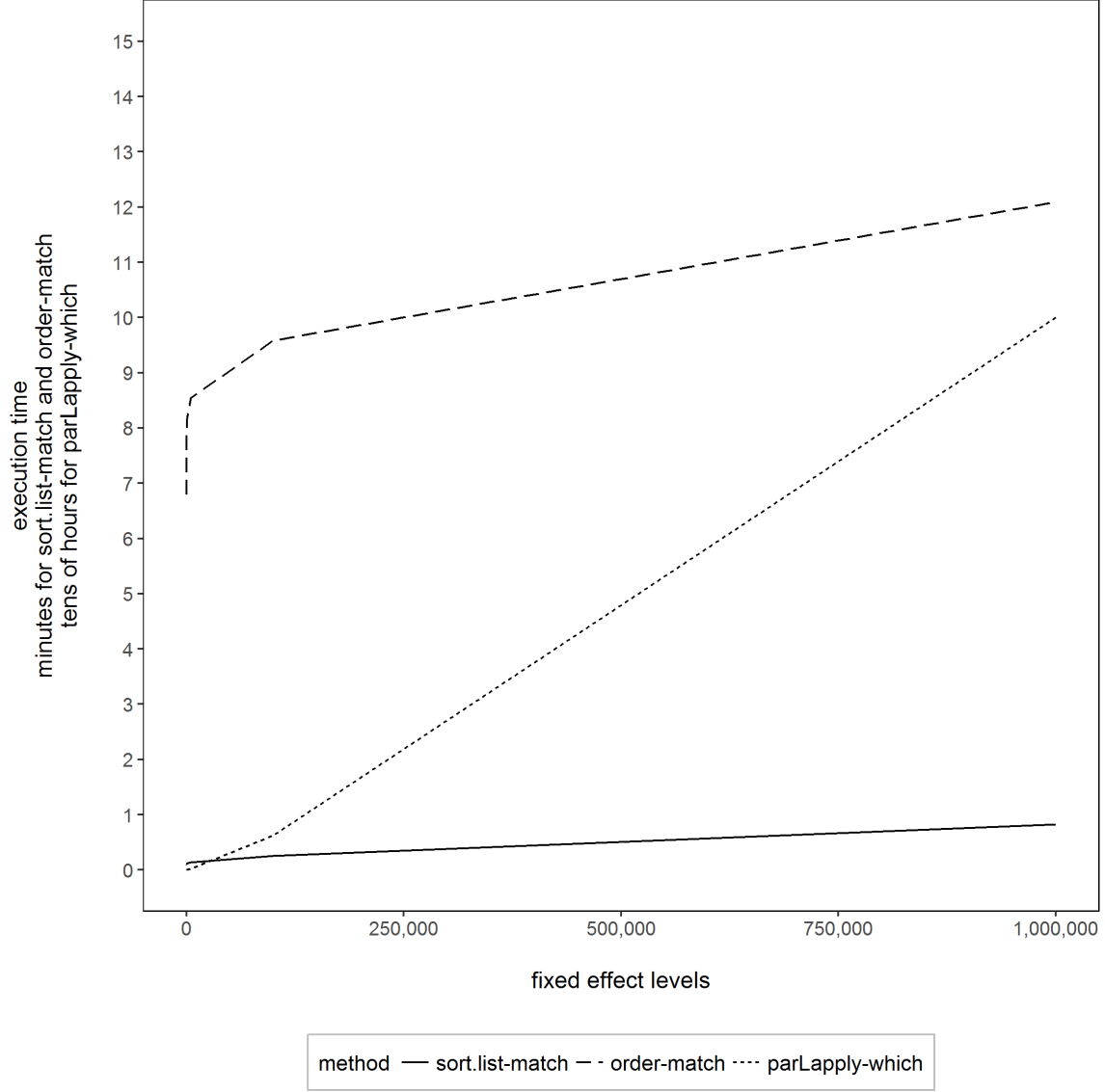


Figure 3: Three Indexing Methods: Comparison of Time to Index a Single High Dimension Fixed Effect Column. `sort.list-match` and `order-match` measured in minutes; due to lengthy execution time, `parallel-apply-which` is measured in hours divided by 10.

Equipped with minimal sets of fixed effects observation indices, those that must be included in matrix operations, we now proceed with $\mathbf{X}'\mathbf{X}$ construction. First, knowing that $\mathbf{X}'\mathbf{X}$ is symmetric, we will construct the diagonal and upper triangle only, then copy the transpose of the upper triangle to the lower triangle. Ignoring (for simplicity, although the algorithm supports) interactions, there are four types of “products” that must be accommodated:

- the transpose of the constant (1) vector “multiplied” by continuous and fixed effects columns
- the transpose of a continuous column “multiplied” by a continuous column
- the transpose of a continuous column “multiplied” by a fixed effect column

- the transpose of fixed effect column “multiplied” by a fixed effect column

Multiplied is quoted to indicate that a result identical to that of standard matrix multiplication operations will be produced, but using where possible, fewer and simpler operations. Each of the four “product” types uses a distinct combination of operations:

- Row 1 (constant times continuous and fixed effects columns) consists of sums of corresponding columns. Column 1 is simply n , the total number of observations. For continuous columns, $\text{sum}(\mathbf{X}_j)$ is used. For fixed effects columns, $\text{length}(\mathbf{X}_{k_i})$ is used, where \mathbf{X}_{k_i} is the previously constructed non-zero index vector for the i^{th} level of the k^{th} fixed effect.
- Products involving two continuous columns are composed with $\text{sum}(\mathbf{X}_i * \mathbf{X}_j)$, which is equivalent to the standard matrix operation $\mathbf{X}_i' \mathbf{X}_j$.
- Products of continuous and fixed effects columns equate to the sum of continuous \mathbf{X}_j elements corresponding to non-zero positions of fixed effects and use $\text{sum}(\mathbf{X}_j[\mathbf{X}_{k_i}])$, where \mathbf{X}_{k_i} is the previously composed index vector for the i^{th} level of the k^{th} fixed effect.
- Products of fixed effects with fixed effects have three types:
 - On-diagonal products are simply the number of observations coded with the corresponding level (since the product of a fixed effect indicator variable by itself is the sum of n_{k_i} products of 1.0 and 1.0, where n_{k_i} is the number of observations coded with level i of fixed effect k).
 - Off-diagonal products of columns from within a single fixed effect are zero, due to orthogonality.
 - Off-diagonal products of columns from distinct fixed effects use the length of the intersection of corresponding index columns, equating to the count (sum of products of 1.0 and 1.0) of positions where each index is non-zero. Specifically, $(\mathbf{X}'\mathbf{X})_{k1_i, k2_j} = \text{length}(\text{intersect}(\mathbf{X}_{k1_i}, \mathbf{X}_{k2_j}))$, where \mathbf{X}_{k1_i} is the index vector for the i^{th} level of fixed effect 1 and \mathbf{X}_{k2_j} is the index vector for the j^{th} level of fixed effect 2.

9 Solving the OLS Normal Equations

With $\mathbf{X}'\mathbf{X}$ constructed (after copying the transpose of the constructed upper triangle to the lower triangle²⁸), we now turn to solving the OLS normal equations, $\mathbf{X}'\mathbf{X}\hat{\beta} = \mathbf{X}'\mathbf{Y}$ for $\hat{\beta}$. We see that, in addition to $\mathbf{X}'\mathbf{X}$, $\mathbf{X}'\mathbf{Y}$ is need and it should be noted that a complete expanded \mathbf{X} matrix (consisting of columns for continuous and fixed effect indicators) is not available to us. However, $\mathbf{X}'\mathbf{Y}$ is simply a $p \times 1$ vector with p_x positions consisting of inner-products of continuous \mathbf{X} columns and \mathbf{Y} , followed by p_k fixed effect positions, one for each non-reference level of each effect and containing the sum of \mathbf{Y} elements corresponding to the associated

²⁸with $\mathbf{X}'\mathbf{X} \leftarrow \mathbf{X}'\mathbf{X} + \text{t}(\mathbf{X}'\mathbf{X}) - \text{diag}(\text{x}=\text{diag}(\mathbf{X}'\mathbf{X}), \text{nrow}=\text{p})$; note that $\mathbf{X}'\mathbf{X}$ is initially $0_{p \times p}$; also note the subtraction of a redundant diagonal

effect and level. $\mathbf{X}'\mathbf{Y}$ positions corresponding to fixed effects are efficiently constructed using the non-zero indicators and indexing methods developed in composing $\mathbf{X}'\mathbf{X}$.

Of the various methods for solving $\mathbf{X}'\mathbf{X}\hat{\beta} = \mathbf{X}'\mathbf{Y}$ for $\hat{\beta}$ two that are well established and known for computational efficiency are QR decomposition and Cholesky decomposition.^{29 30} The efficiency of QR and Cholesky decomposition derive from a strategy of forming pairs of triangular matrices such their product represents the left hand matrix in the system being solved, $\mathbf{X}'\mathbf{X}$ in our case. In the case of Cholesky decomposition, an upper triangular matrix \mathbf{R} is computed such that $\mathbf{R}'\mathbf{R} = \mathbf{X}'\mathbf{X}$, then $\mathbf{R}'\mathbf{R}\hat{\beta} = \mathbf{X}'\mathbf{Y}$ is solved in two stages: $\mathbf{R}'\mathbf{W} = \mathbf{X}'\mathbf{Y}$ for \mathbf{W} then $\mathbf{R}\hat{\beta} = \mathbf{W}$ for $\hat{\beta}$. In addition to facilitating solution of $\mathbf{X}'\mathbf{X}\hat{\beta} = \mathbf{X}'\mathbf{Y}$, \mathbf{R} can be used to solve $\mathbf{X}'\mathbf{X}\mathbf{W} = \mathbf{I}$, where $\mathbf{W} = (\mathbf{X}'\mathbf{X})^{-1}$, the computationally significant component of the homoskedastic parameter covariance equation $\text{Cov}(\hat{\beta}) = \sigma^2 (\mathbf{X}'\mathbf{X})^{-1}$. QR decomposition follows a similar strategy, but with matrices \mathbf{Q} and \mathbf{R} , where $\mathbf{Q}\mathbf{R} = \mathbf{X}'\mathbf{X}$, \mathbf{Q} is orthogonal, and \mathbf{R} is upper triangular. In practice, the computational effort required to generate \mathbf{R} (or \mathbf{Q} and \mathbf{R}) combined with solving two simpler systems tends to be less than in solving $\mathbf{X}'\mathbf{X}\hat{\beta} = \mathbf{X}'\mathbf{Y}$ using direct Gaussian elimination and the resulting equations and solution tend to exhibit improved numerical stability.³¹ Although both QR decomposition and Cholesky decomposition produce near mathematically equivalent results, it has been the authors' experience that, for large fixed effect regression problems, Cholesky decomposition requires less time to solve both systems of equations, one for $\hat{\beta}$ and one for $\text{Cov}(\hat{\beta})$, than does QR decomposition. As an example, table 2 compares times to compute the QR decomposition, Cholesky decomposition, and inverse of $\mathbf{X}'\mathbf{X}$ from two design matrices (one of size 5,000,000 \times 6,200; the other 10,000,000 \times 16,000) using the standard R functions `qr()`, `qr.solve()`, `chol()`, and `chol2inv()`.^{32 33} The apparent four to one performance ratio gives Cholesky decomposition a significant advantage.

Table 2: Time (in minutes) to Compute Decomposition and Inverse of $\mathbf{X}'\mathbf{X}$

| Method | 6,200 matrix | 16,000 matrix |
|---|--------------|---------------|
| <code>qr()</code> + <code>qr.solve()</code> * | 10.0 | 172.7 |
| <code>chol()</code> + <code>chol2inv()</code> | 2.5 | 44.1 |

* computed using the slightly more efficient LAPACK option, as opposed to the default LINPACK

We will restrict our discussion to Cholesky decomposition. While solving various large regression problems, it was observed that as much as three-fourths of the entire computation cycle [composition of $\mathbf{X}'\mathbf{X}$

²⁹For more on QR decomposition, see Wikipedia (2017b) and Vandenberghe (2017). QR decomposition is implemented by the R function `qr()` (R Foundation for Statistical Computing, 2017, `qr()`) and is the default algorithm used by `lm()`.

³⁰For more on Cholesky decomposition, see Wikipedia (2017a) and Heath (2013). Cholesky decomposition is implemented by the R function `chol()` (R Foundation for Statistical Computing, 2017, `chol()`).

³¹See Higham (1990)

³²`qr.solve(X)` computes the inverse of \mathbf{X} (R Foundation for Statistical Computing, 2017, `qr.solve()`)

³³`chol2inv(R)` computes the inverse of $\mathbf{X} = \mathbf{R}'\mathbf{R}$ (R Foundation for Statistical Computing, 2017, `chol2inv()`)

(using efficient indexing as presented), computation of \mathbf{R} , solution of $\hat{\boldsymbol{\beta}}$, solution of $\text{Cov}(\hat{\boldsymbol{\beta}})$] was devoted to the final three items, all involving \mathbf{R} . Although, algorithmically, \mathbf{R} can be computed in simultaneous parallel operations, the base R functions `chol()` and `chol2inv()` are implemented as a sequence of row and column operations, with a single row or column being computed before subsequent rows or columns are begun. As part of the current project, two Cholesky related functions were developed: one to compute the decomposition \mathbf{R} , given a composed $\mathbf{X}'\mathbf{X}$, and one to compute $(\mathbf{X}'\mathbf{X})^{-1}$ using \mathbf{R} . Both are implemented in the C language (using package `Rcpp`) and employ parallel methods (using `OpenMP`). Source listings are in appendices *Parallel Cholesky Decomposition Algorithm* and *Parallel Algorithm to Compute Inverse of \mathbf{X} Using the Cholesky Decomposition*. Function `choleskyDecomp()` computes \mathbf{R} and function `cholInvDiag()` computes $(\mathbf{X}'\mathbf{X})^{-1}$. Figure 4 compares the performance of the custom parallel algorithms to that of their base R counterparts, solid lines indicate execution times of custom functions, dashed lines for standard R functions. With performance of approximately eight times that of base R functions, equating to a savings of over five hours for the largest problem tested, the benefit of custom, parallel algorithm implementation is significant.

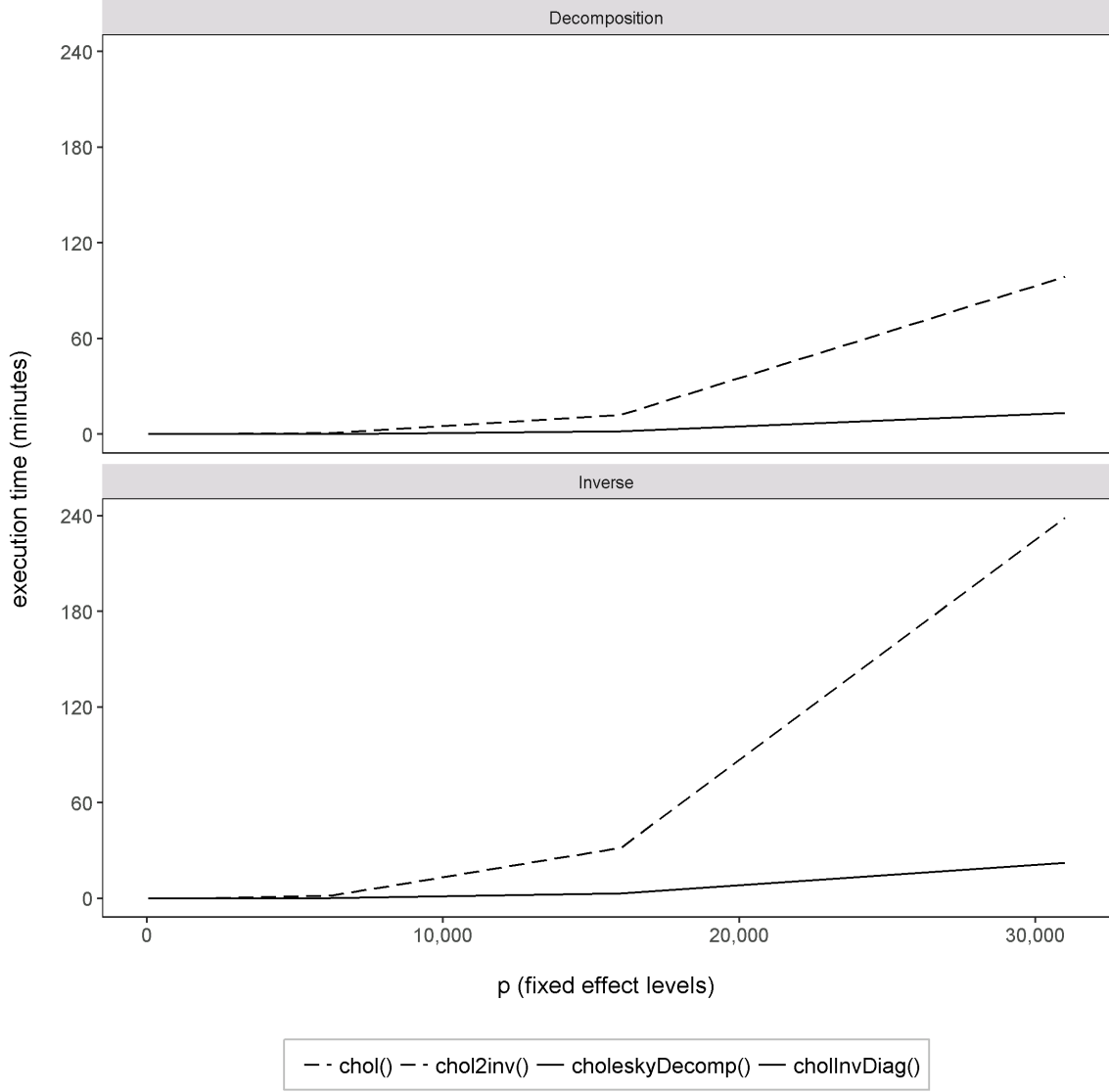


Figure 4: Comparison of Execution Times to Compute Cholesky Decomposition and $(\mathbf{X}'\mathbf{X})^{-1}$ Using Custom Parallel Algorithm and Base R Functions. Five independent data sets were generated for each level of p_k (number of fixed effect levels). Observation counts ranged from 10,000 (small p_k) to 25,000,000 (large p_k).

To verify consistency of results between the custom parallel and base R functions, element-wise differences in computed \mathbf{R} or $(\mathbf{X}'\mathbf{X})^{-1}$ matrices of absolute value greater than 10^{-10} were tested for using `which(abs($\mathbf{R}_1 - \mathbf{R}_2$) > 1e $^{-10}$)` and `which(abs(($\mathbf{X}'\mathbf{X}$) $_1^{-1} - (\mathbf{X}'\mathbf{X})_2^{-1}$) > 1e $^{-10}$)`, where \mathbf{R}_1 and \mathbf{R}_2 are custom and base R Cholesky decomposition results, respectively, and $(\mathbf{X}'\mathbf{X})_1^{-1}$ and $(\mathbf{X}'\mathbf{X})_2^{-1}$ are custom and base R inverses, respectively. None were reported.

10 Implementation: feXTX(), Parallel Cholesky Decomposition

While conducting research into U.S. federal employee pay disparity, the Human Capital Project at Duke University developed a requirement for fitting high dimension fixed effects OLS models to a large set of historical federal employee human capital data, described under *Example Data Set*, above. A simultaneous requirement arose from the Synthetic Data Project at Duke, to certify a synthetic U.S. federal human capital data set to be used in a public use verification server system.³⁴ After various attempts to use the available solutions previously described, with limited success, the teams decided to develop a solution tailored to solving general large fixed effects problems while taking advantage of the performance and efficiency opportunities mentioned so far in this paper. The result is an implementation of the $\mathbf{X}'\mathbf{X}$ construction and Cholesky decomposition methods presented in previous sections that the teams call `feXTX()`. With the exception of parallel `apply()` and the custom Cholesky decomposition functions, `feXTX()` uses base R functions. Parallel functions are taken from the `parallel` package (`snow` has also been tested).³⁵

`feXTX()` takes the following parameters:

- **data**: source data frame containing observations (vectors must be named)
- **Y**: the (character string) name of the dependent variable vector with this name must appear in data
- **contX**: vector of (character string) names of continuous variables (corresponding columns must exist in data); specify `NULL` if none
- **fixedX**: vector of (character string) names of fixed effect variables (corresponding columns must exist in data); must contain at least one column name
- **refLevel**: vector of (character string) fixed effect reference levels in order of **fixedX**
- **interactionX**: two column array of interactions, each row an interaction; specify `NULL` if none
- **estBetaVar**: `"stdOLS"` for homoskedastic variances, `"robust"` for robust variances, `"cluster"` for clustered variances, or `"none"` for none
- **robustVarID**: (character string) column name in data containing values about which to cluster variances (vector with this name must appear in data)
- **nCoreXTX**: number of parallel cores to be used in beta solution (for the example pay disparity model, a value of 12 has proven to give good results benefits of increasing cores is offset by the requirement to export data to them in a Windows environment)
- **nCoreVar**: number of parallel cores to be used in clustered variance estimation (core configuration for `stdOLS` variances is controlled by the `cholInvDiag()` function, the robust variance method uses data and cores created during the $\mathbf{X}'\mathbf{X}$ construction step); for clustered variance estimates, a value of 4

³⁴For more on synthetic data and verification systems, see Reiter et al. (2009); Reiter (2003); Barrientos et al. (2017)

³⁵Parallel functions `parApply()` and `parLapply()` are employed (R-core, 2016, `parApply()`, `parLapply()`)

(perhaps less) is optimal here, since clustered variance computations require significant export of data (Windows), while the computations are relatively efficient within each parallel process

- **solMethod:**
 - "QR" (default) for QR decomposition
 - "Chol" for Cholesky decomposition using base R `chol()` and `chol2inv()` functions
 - "Chol-Parallel" for Cholesky decomposition using custom parallel `choleskyDecomp()` and `cholInvDiag()` functions

The value returned by `feXTX()` is a list with the following members:

- **status:** (character string) completion status (" " for success, otherwise error message)
- **Y:** (character string) echo of input **Y** parameter
- **contX:** (character string) echo of input **contX** parameter
- **fixedX:** (character string) echo of input **fixedX** parameter
- **refLevel:** (character string vector) echo of input **refLevel** parameter
- **interactionX:** (two column array or NULL) echo of input **interactionX** parameter
- **nX:** (numeric scalar) number of rows in data
- **beta:** (numeric vector) OLS parameter estimates for $Y \sim b0 + contX + indicator(fixedX|not\ ref) + interactionX$
- **vBeta:** (numeric vector) parameter estimate variances, as instructed in **estBetaVar**
- **estBetaVar:** (character string) echo of input **estBetaVar** parameter
- **XTX:** (matrix) $\mathbf{X}'\mathbf{X}$ matrix
- **dimXTX:** (numeric scalar) column (and row) dimension of $\mathbf{X}'\mathbf{X}$
- **rankXTX:** (numeric scalar) rank of $\mathbf{X}'\mathbf{X}$ as reported by `qr()` or `chol()`
- **XTY:** (numeric vector) resulting $\mathbf{X}'\mathbf{Y}$
- **yEst:** (numeric vector) estimated $\hat{\mathbf{Y}}$ values
- **yVar:** (numeric scalar) variance of observed errors, $\frac{\sum(\mathbf{Y}-\hat{\mathbf{Y}})^2}{df}$
- **time:** matrix of execution time, row 1 for compute, row 2 for memory export (Windows)

A complete program listing of `feXTX()` is in the *R feXTX() Program Listing* appendix.³⁶ Program listings for `choleskyDecomp()` and `cholInvDiag()` are in appendices *Parallel Cholesky Decomposition Algorithm* and *Parallel Algorithm to Compute Inverse of Matrix X Using the Cholesky Decomposition*, respectively.³⁷

11 Performance of Solutions with Simulated Data and Models

We now compare the performance and results of `feXTX()` with other solutions presented. Before using the example OPM data set, since it is not available to the public, and to give the reader a ready means of testing various methods, simulated data sets with specifiable covariate relationships will be generated. The *Sample Data Generation* appendix³⁸ contains an R script that produces a data frame, `feDat`, with one dependent (continuous numeric) vector Y and independent vectors X_1 , X_2 , X_3 , X_4 , and X_5 , where X_1 and X_2 are continuous numeric vectors and X_3 , X_4 , and X_5 are categorical (character) fixed effects. Y is simulated as a linear function of the X values plus normally distributed error such that

$$Y_i = \beta_0 + \beta_1 X_{1_i} + \beta_2 X_{2_i} + \beta_{3_i} + \beta_{4_i} + \beta_{5_i} + \epsilon \quad (2)$$

where β_{3_i} , β_{4_i} , and β_{5_i} are the effects corresponding to levels of X_3 , X_4 , and X_5 of observation i . Values for the β 's and ϵ are available in the appendix. Table 3 lists test parameters (number of observations and number of levels for X_3 , X_4 , and X_5) used to generate simulated data sets for evaluation of execution times.

Table 3: Simulated Data Set Parameters for Execution Time Evaluation

| Parameter Set | n (Observations) | Levels X_3 | Levels X_4 | Levels X_5 |
|---------------|------------------|--------------|--------------|--------------|
| 1 | 10,000 | 10 | 15 | 20 |
| 2 | 25,000 | 10 | 20 | 40 |
| 3 | 50,000 | 15 | 30 | 50 |
| 4 | 100,000 | 15 | 40 | 60 |
| 5 | 150,000 | 25 | 40 | 60 |
| 6 | 250,000 | 25 | 50 | 75 |
| 7 | 500,000 | 50 | 75 | 100 |
| 8 | 1,000,000 | 75 | 150 | 250 |
| 9 | 1,500,000 | 100 | 250 | 500 |
| 10 | 2,000,000 | 100 | 500 | 1,000 |
| 11 | 5,000,000 | 200 | 1,000 | 5,000 |
| 12 | 10,000,000 | 1,000 | 5,000 | 10,000 |
| 13 | 25,000,000 | 1,000 | 5,000 | 25,000 |

³⁶Available in the on-line repository (Duke University Synthetic Data Project, 2017, `feXTX()` git Repo)

³⁷Available in the on-line repository (Duke University Synthetic Data Project, 2017, `feXTX()` git Repo)

³⁸Available in the on-line repository (Duke University Synthetic Data Project, 2017, `feXTX()` git Repo)

Also in the *Sample Data* appendix are functions for fitting model 2 to simulated data, using `lm()`, `biglm()`, `bigglm()`, `biglm.big.matrix()`, `biglm.big.matrix()`, `SparseM.slm.fit()`, `speedlm()`, `lfe()`, and `feXTX()`. Note that appropriate packages must be installed in order to execute related model fitting functions. Figure 5 plots mean execution time by method using five independently simulated data sets for parameter sets 1 through 9. In addition to execution efficiency, the total on-line memory used by an algorithm is important, since it is generally limited and shared by multiple concurrent users and processes. As a quick assessment of memory required by each method, table 4 lists maximum differential of memory in use (from baseline prior to execution, as reported by Windows task manager) while the algorithms were executed using a single instance of data generated by parameter set 9 (the highest dimension set that all methods were executed with).

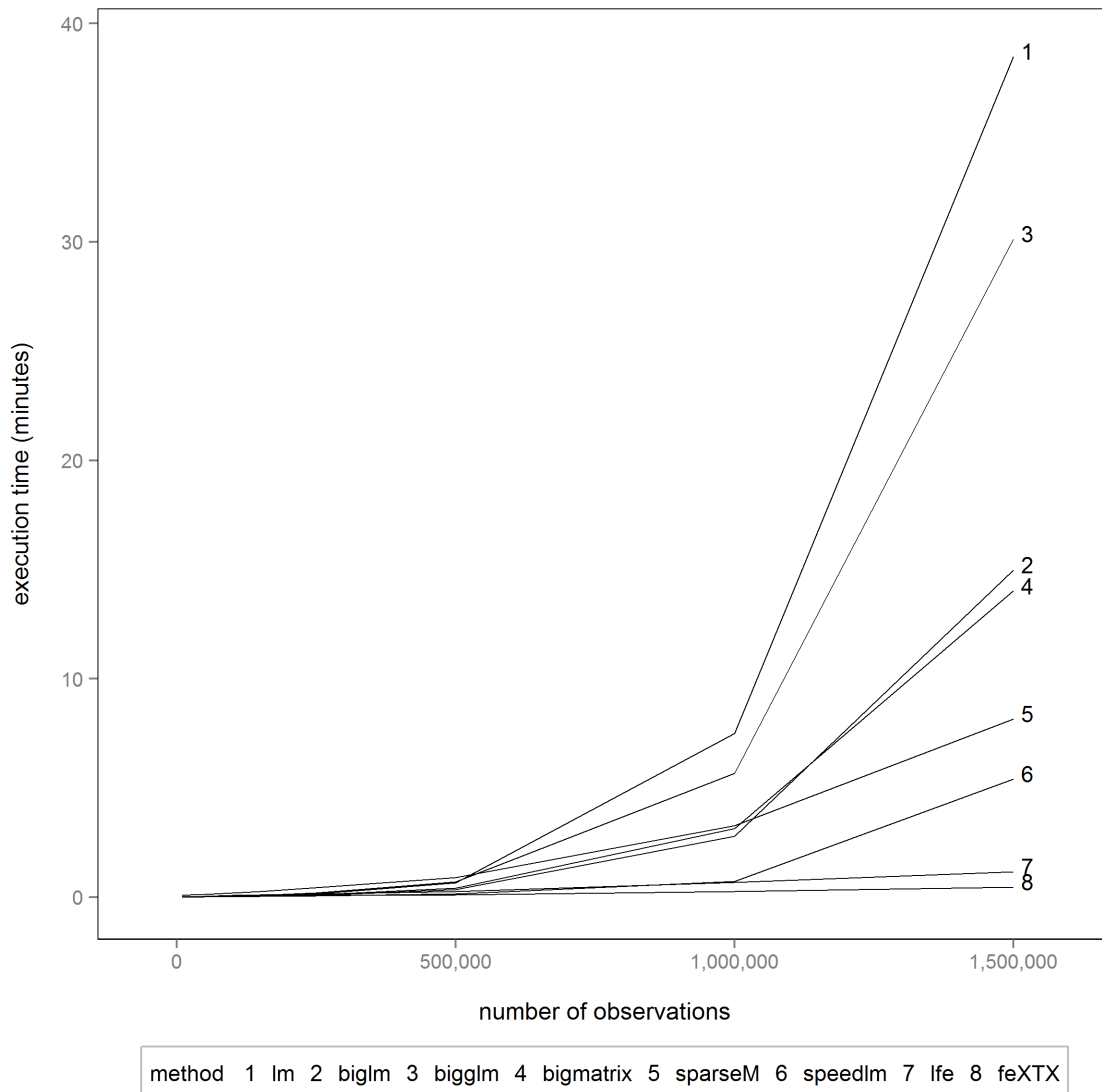


Figure 5: Mean Execution Time to Solve Graduated Fixed Effects Problems by Method. Execution time is the differential of “elapsed” time as reported by `proc.time()`. All processing was performed on a single dedicated server.

Efficient execution is important, but accuracy of results is of the utmost importance. Considering the computational challenges of fitting high dimension models to large data sets, obtaining an authoritative set of parameter estimates for objective comparison is problematic. As an alternative, we might take the approach of comparing estimates to a consensus of those from established methods. If no major deviations are observed then we can state that, with data sets and models tested, our method introduces no error beyond that observed in methods currently in use. Table 5 shows, by method, the number of parameter estimates, using simulation parameter set 9, that differ with those generated by `feXTX()`, grouped by absolute value ranges listed in the column headers. There does seem to be consensus, with all methods except `lfe()` having

Table 4: Maximum On-line Memory Used While Fitting Simulation Parameter Set 9

| Method | Memory Used (Gb) |
|---------------------------------|------------------|
| <code>lm()</code> | 19.3 |
| <code>biglm()</code> | 19.3 |
| <code>bigglm()</code> | 19.4 |
| <code>biglm.big.matrix()</code> | 0.80* |
| <code>SparseM.slm.fit()</code> | 20.3 |
| <code>speedlm()</code> | 28.8 |
| <code>lfe()</code> | 10.4 |
| <code>feXTX()</code> | 0.50 |

* backing file created

all deviations of parameter estimates less than 10^{-7} . By default, `lfe()` does not estimate an intercept and fixed effect reference levels are not omitted, causing significant differences in estimated values. Alternate parameter estimates can be requested, but only by constructing estimable function specifications, which was not pursued.

Table 5: Comparison of Parameter Estimates, Evaluated Solutions vs. `feXTX()`

| Method | $0 < 10^{-12}$ | $10^{-12} < 10^{-10}$ | $10^{-10} < 10^{-7}$ | $10^{-7} < 10^{-5}$ | $> 10^{-5}$ |
|--------------------------|----------------|-----------------------|----------------------|---------------------|-------------|
| <code>lm()</code> | 369 | 478 | 3 | 0 | 0 |
| <code>biglm()</code> | 339 | 508 | 3 | 0 | 0 |
| <code>bigglm()</code> | 339 | 508 | 3 | 0 | 0 |
| <code>bigmatrix()</code> | 339 | 508 | 3 | 0 | 0 |
| <code>sparseM()</code> | 0 | 2 | 848 | 0 | 0 |
| <code>speedlm()</code> | 0 | 2 | 848 | 0 | 0 |
| <code>lfe()</code> | 0 | 0 | 2 | 1 | 847 |

In terms of execution and memory efficiency, `lfe()` and `feXTX()` appear to significantly outperform the other solutions and maintain efficiency throughout the range of smaller data sets. Continuing with more demanding data sets (parameter sets 10, 11, 12, and 13) and limiting our study to `lfe()` and `feXTX()`, we see in figure 6 a continued divergence of execution time, with `lfe()` exhibiting exponential increase with respect to problem size (as observed with the other methods and smaller problems), while `feXTX()` remains nearly linear, executing the largest problem, with 25,000,000 observations and 31,000 fixed effects levels, in 1/8th the time required by `lfe()`. Memory requirements to fit model 2 using parameters set 13 were ??? Gb and ??? Gb for `lfe()` and `feXTX()`, respectively.

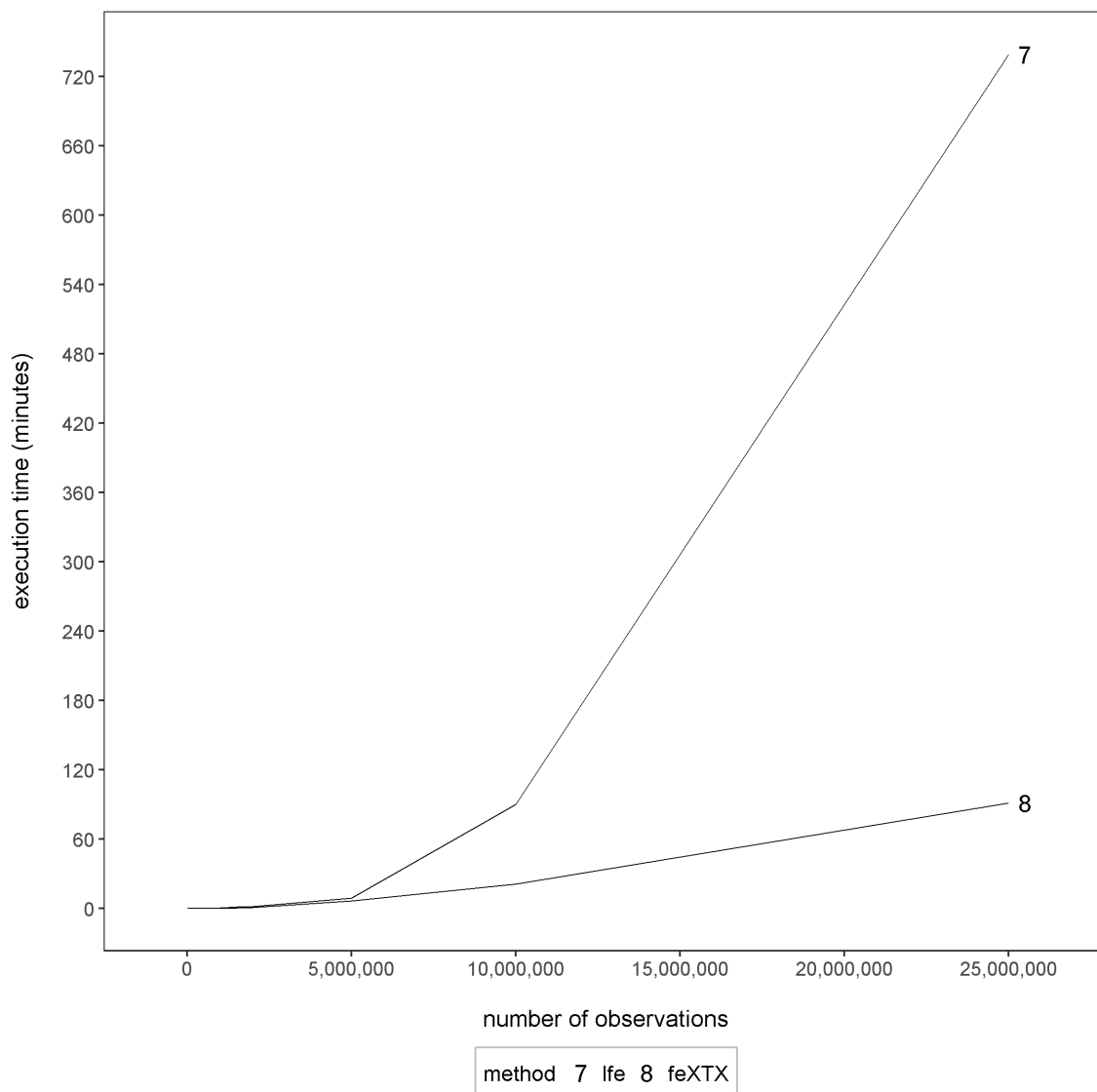


Figure 6: Mean Execution Time to Solve Graduated Fixed Effects Problems. Higher dimension effects, for `lfe()` and `feXTX()` methods. Execution time is the differential of “elapsed” time as reported by `proc.time()`. All processing was performed on a single dedicated server.

12 Performance with OPM Data and Models

However compelling the case for efficiency is made using simulated data, ultimately we must assess utility with actual data and models. Performance and issues regarding execution of `lm()`, `biglm()`, `bigglm()`, `biglm.big.matrix()`, `SparseM.slm.fit()`, and `speedlm()` using the OPM data and models are discussed in the *Review of Available Solutions* appendix.³⁹ Due to obstacles and limitations, these solutions are not considered here. As previously stated, the example OPM data set contains 24,574,480 observations and two

³⁹Available in the on-line repository (Duke University Synthetic Data Project, 2017, `feXTX()` git Repo)

moderate dimension fixed effects (bureau and occupation, with 409 and 791 levels, respectively). With our understanding of matrix and operation density, it is of interest to measure algorithm performance using a somewhat large, higher dimension data set and an actual model. Before fitting model 1 we will use

$$y = \beta_0 + \beta_{sex} + \beta_{race} + \beta_{age^2} \times age^2 + \beta_{ed} \times ed_{years} + \beta_{bureau} + \beta_{occ} + \beta_{year} \quad (3)$$

which is identical to model 1 without the $sex \times race$ interaction term (this model appears in Bolton and de Figueiredo, 2016 - cite). Table 6 lists execution times and memory requirements to fit model 3 using `lfe()` and `feXTX()`.

Table 6: Execution Time and Memory Required to Fit Model 3 to OPM Data

| Method | Execution Time (min) | Memory used (Gb) |
|----------------------|----------------------|------------------|
| <code>lfe()</code> | 39.5 | 128* |
| <code>feXTX()</code> | 9.6 | 15.3 |

* exceeds memory available on 64 Gb test server; execution accomplished on alternate dedicated server with 256 Gb of memory; “non-estimable function” warnings observed on alternate server

Finally, execution times and memory requirements to fit model ?? to the example OPM data are listed in table 7. Although efficient in a sparse context, the `feXTX()` indexing algorithm for interactions involving fixed effect columns is somewhat complex and adds inordinately to execution time.⁴⁰ An alternative to specifying interactions in the function call to `feXTX()`, when low dimension fixed effects are involved (*sex* and *race* have 1 and 4 non-reference levels, respectively), is to construct indicator columns for each intersection of interacting levels and include these in the input data frame as continuous variables.⁴¹ Table 7 lists execution time to fit model ?? using `lfe()` (with interactions in model specification the preferred method) and `feXTX()` (with interactions expanded as described).

Table 7: Execution Time and Memory Required to Fit Model 1 to OPM Data

| Method | Execution Time (min) | Memory used (Gb) |
|----------------------|----------------------|------------------|
| <code>lfe()</code> | 40.5 | 128* |
| <code>feXTX()</code> | 13.2 | 20 |

* exceeds memory available on 64 Gb test server; execution accomplished on alternate dedicated server with 256 Gb of memory; “non-estimable function” warnings observed on alternate server

⁴⁰Ironically, density of fixed effect interactions is expected to be low, $d_1 \times d_2$ for two interacting levels of distinct fixed effects, but the algorithm involves exhaustive, nested calls to `apply()` functions. At present, interaction elements of $\mathbf{X}'\mathbf{X}$ are processed sequentially; a future release will compute them in parallel.

⁴¹Approximately two seconds of compute time is required to construct *sex* and *race* interaction vectors

Again, we observe a favorable difference in execution and memory efficiency when comparing `feXTX()` to `lfe()`, which itself has been observed to be reasonably efficient.

13 Robust and Clustered Standard Errors

To compensate for heteroskedasticity of within-group errors, analysts often report parameter estimates with respect to robust or clustered standard errors (Cameron and Miller, 2015). `feXTX()` computes robust and clustered standard errors using the fundamental equation for $\text{Cov}(\hat{\beta})$ derived from the OLS normal equations:

$$\text{Cov}(\hat{\beta}) = (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{U}\mathbf{X} (\mathbf{X}'\mathbf{X})^{-1} \quad (4)$$

where \mathbf{U} is the $n \times n$ matrix of within cluster residual covariances, and is estimated by setting all inter-cluster errors, $(\epsilon'\epsilon)_{cluster}$, to 0 (on the assumption of inter-cluster independence (Esarey and Menger, 2017)). Note that \mathbf{U} is $n \times n$ which, for the problems we consider, can be quite large, $24,574,480 \times 24,574,480$ in our example OPM data set. Further, it is involved in operations with another large matrix, \mathbf{X} ($24,574,480 \times p$). Standard matrix operations involving objects of this size are computationally impractical, in both time and memory requirements. However, indexing methods similar to those used in composing $\mathbf{X}'\mathbf{X}$ can be used to compute $\text{Cov}(\hat{\beta})$ very efficiently. In fact, having constructed sparse indices to fixed effect columns of \mathbf{X} along with $(\mathbf{X}'\mathbf{X})^{-1}$, we simply proceed with construction of \mathbf{U} (which is efficiently accomplished in parallel), then multiplication of matrices in equation 4 referencing only non-zero elements of \mathbf{X} (which is also done in parallel). Note that, since the standard errors of $\hat{\beta}$ are generally of interest, only $\text{Var}(\hat{\beta}) = \text{diag} [\text{Cov}(\hat{\beta})]$ is computed.

Robust standard error algorithm outline:

- Compute robust, uncorrelated, heteroskedastic (independent, non-identically-distributed) errors
- Estimate robust variances, square of $\text{SE}(\hat{\beta})$, as $\text{Var}(\hat{\beta}) = \text{diag} [(\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{u}\mathbf{u}'\mathbf{X} (\mathbf{X}'\mathbf{X})^{-1}]$, where \mathbf{u} is the vector of observation response to predicted value errors, $\mathbf{u} = \mathbf{Y} - \hat{\mathbf{Y}}$
- This derives from the expression for parameter variances from the normal OLS equations
- $\mathbf{u}\mathbf{u}'$ forms an $n \times n$ diagonal ϵ variance-covariance matrix with all off-diagonal elements set to 0, which asserts the assumption of independent (uncorrelated) errors
- But, since the diagonal elements are expected to be non-constant, this method models parameter standard errors in a heteroskedastic (independent, but not identically distributed) error setting
- Since $\mathbf{X}'\mathbf{u}\mathbf{u}'\mathbf{X}$ is symmetric, construct upper triangle only then copy transpose to lower triangle

Clustered standard error algorithm outline:

- Compute heteroskedastic, correlated within cluster, independent (uncorrelated) between cluster, non-identically-distributed (heteroskedastic) errors
- Estimate clustered standard errors using $\text{Var}(\hat{\beta}) = \text{diag}[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{uu}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}]$, where \mathbf{uu}' is the var-cov matrix of observation to estimate errors, $\mathbf{uu}' = \text{Cov}(\mathbf{Y} - \hat{\mathbf{Y}})$
- This is the expression for $\text{Var}(\hat{\beta})$ from ordinary least squares estimates of beta on the assumption that \mathbf{X} is non-stochastic and \mathbf{uu}' is an estimate of error covariance and, further, that covariance observed within clusters (groups) estimates that in the population while covariance between groups is 0
- \mathbf{uu}' is altered such that all off-diagonal inter-group elements, that is \mathbf{uu}'_{ij} where indices i and j belong to different groups, are set to 0
- The modified (inter-group independent error) \mathbf{uu}' has the form:

$$\begin{bmatrix} \mathbf{uu}'_{group_1} & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{uu}'_{group_2} & 0 & \cdots & 0 \\ 0 & 0 & \mathbf{uu}'_{group_3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \mathbf{uu}'_{group_k} \end{bmatrix}$$

where \mathbf{uu}'_{group_i} is the $n_{group_i} \times n_{group_i}$ sub-matrix of the ϵ covariance matrix corresponding to group i

- Note that with this construction of \mathbf{uu}' , $\mathbf{X}'\mathbf{uu}'\mathbf{X}$ is the sum (over all groups i) of $\mathbf{X}'_{group_i}\mathbf{uu}'_{group_i}\mathbf{X}_{group_i}$ where \mathbf{X}_{group_i} and \mathbf{uu}'_{group_i} are the rows of \mathbf{X} and sub-matrix of \mathbf{uu}' corresponding to group i [to see this, imagine multiplying mtX by \mathbf{uu}' and consider the effect of \mathbf{uu}' elements of rows that do not correspond to a particular group - each resulting column corresponds to products from a single group - then multiplying that by \mathbf{X} (each row belongs to a single group) gives a $p \times p$ sum of individual group products]
- Note that $\mathbf{X}'_{group_i}\mathbf{u}_{group_i} = [\mathbf{u}'_{group_i}\mathbf{X}_{group_i}]'$, making $\mathbf{X}'_{group_i}\mathbf{uu}'_{group_i}\mathbf{X}_{group_i}$ symmetric
- Also, $\mathbf{u}'_{group_i}\mathbf{X}_{group_i}$ is a $1 \times p$ vector, where p is the column dimension of \mathbf{X} (the design matrix)
- For each group ID, construct $\mathbf{v} = \mathbf{u}'_{group_i}\mathbf{X}_{group_i}$ then use it to compute $\mathbf{v}'\mathbf{v} = \mathbf{X}'_{group_i}\mathbf{uu}'_{group_i}\mathbf{X}_{group_i}$

Because `lfe()` requires specification of estimable functions to compute estimates for an intercept and non-reference level fixed effects, which is not pursued, comparison of robust and clustered standard errors is made to those computed by Stata, which is known for computational efficiency and accuracy. Table 8 lists execution times to fit model 1 to the OPM data and to compute robust and clustered standard errors using `lfe()`, `feXTX()`, and Stata (Stata/MP version 13.1). `lfe()` is included for execution time comparison. All processing was executed on a single, dedicated, 24 core Windows 7 server. The approximate 2 to 1

performance ratio of `feXTX()` to both `lfe()` and Stata indicates an advantage to computing heteroskedastic robust and clustered standard errors by solving the analytical variance equation (eq 4) using efficient indexing methods. The largest (absolute value) pair-wise deviation in Stata and `feXTX()` computed standard errors was less than 10^{-5} , lending credence to accurate analytical equation evaluation.

Table 8: Execution Time (in minutes) to Fit Model 1 to OPM Data and Compute Robust and Clustered Standard Errors

| Method | Robust SE Time (min) | Clustered SE Time (min) | Memory used (Gb) |
|-------------------------|----------------------|-------------------------|------------------|
| <code>lfe()</code> * | — | 51.9 | 128 |
| <code>feXTX()</code> ** | 13.2 | 13.2 | 20/38 |
| Stata*** | 51.5 | 55.5 | <20 |

* bootstrap estimates computed; robust standard errors require use of ancillary bootstrap estimation functions which was not pursued; executed on alternate server with additional required memory
message received: "chol.default() - matrix is either rank-deficient or indefinite"

message received: "warning; non-estimable function"

** SE's result from evaluation of analytical standard error equation

*** Using Stata/MP Version 13.1

14 Further Development

While developing `feXTX()` and associated algorithms, several opportunities for improvement of performance or utility were identified but, due to practical limitations of time and striving to hold a course of completing primary features, they are withheld for future development. Among them are:

- Improved sub-setting and transmission of fixed effect vectors to parallel cores transmit only levels to be operated on by a given core
- Elimination of fixed effect level index vector export (however, passing large lists as parallel function parameters is also expensive)
- Parallelization of interaction vector-products
- Encode composed $\mathbf{X}'\mathbf{X}$ matrix in `SparseM` format, since it is itself sparse (this has been experimented with; normal equation solution is efficient, but construction of sparse $\mathbf{X}'\mathbf{X}$ is not; also, robust/standard errors are not immediately available)
- Implement parallel algorithm for forward and back solving of OLS system of equations using computed Cholesky decomposition

15 Conclusion

As large data sets become increasingly available and available data become increasingly complex, new methods are being developed to model systems and identify patterns using both deep and broad statistical probing. At the same time, traditional modeling methods, with tried and true histories, remain relevant and in use, and with data sets from the “small data” period of their origin, familiar methods remain computationally practical. However, with larger data sets, computation of analytical equations from traditional models may be of such order to make solution impractical without introducing estimation or convergence methods. In this paper, we have demonstrated that the traditional, analytical normal equations of ordinary least squares (OLS) in a high dimension fixed effects setting can be efficiently solved by employing a combination of efficient indexing of sparse vectors, parallel computation, and targeted C programming. A methodology was presented that 1.) identifies sources of computational inefficiency in standard OLS functions as implemented in R; 2.) evaluates the efficiency and accuracy of existing computational solutions; 3.) develops alternative strategies and algorithms to efficiently compute results that are mathematically equivalent to those produced by standard functions; and 4.) adapts solutions beyond basic estimates of model parameters to computationally intensive estimates of parameter variance. Efficiency methods and algorithms targeting large data sets and models presented in the paper include:

- Analysis and measurement of design matrix and operation density
- Alternative representation of sparse fixed effect indicator columns using compact index vectors
- Construction of $\mathbf{X}'\mathbf{X}$ using efficient alternatives to vector multiplication
- Solution of parameter estimates from the OLS normal equations ($\mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}'\mathbf{Y}$) using a custom, parallelized Cholesky decomposition algorithm implemented in C
- Solution of homoskedastic parameter standard errors from the analytical equation $\text{Cov}(\hat{\boldsymbol{\beta}}) = \hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}$, using a custom, parallelized function, implemented in C, that computes the inverse of a matrix from its Cholesky decomposition
- Solution of heteroskedastic, robust and clustered, parameter standard errors from the analytical equation $\text{Cov}(\hat{\boldsymbol{\beta}}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{uu}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$, using sparse fixed effect index columns produced during construction of $\mathbf{X}'\mathbf{X}$

Finally, algorithm efficiency was compared to that of seven solutions or packages available for R using small to large simulated data sets. Performance of the most efficient solutions was evaluated using a moderate sized data set and models from the authors’ research. In all cases, the methods and algorithms presented performed with considerably greater efficiency than other solutions evaluated, eliminating hours to days of compute time. Accuracy of results was confirmed using a consensus of results from all solutions and, in some cases, by making comparisons to estimates produced by alternative software (Stata). Although restricted

to a limited family of problems (fixed effects OLS), it is hoped that the general approach of inefficiency isolation, experimentation, algorithm development, performance comparison, and result certification demonstrated here may be useful to readers when facing computationally challenging problems encountered in their research.

References

- Barrientos, A. F., Bolton, A., Balmat, T., Reiter, J. P., de Figueiredo, J. M., Machanavajjhala, A., Chen, Y., Kneifel, C., and DeLong, M. A Framework for Sharing Confidential Research Data, Applied to Investigating Differential Pay by Race in the U. S. Government. Tech. rep., National Bureau of Economic Research Working Paper 23534, 2017.
- Bates, D. and Maechler, M. R Matrix Package, 2017. URL <https://cran.r-project.org/web/packages/Matrix/Matrix.pdf>.
- Bloomberg. Rise of artificial intelligence and machine learning, 2016. URL <https://www.bloomberg.com/professional/blog/rise-of-artificial-intelligence-and-machine-learning>.
- Bolton, A. and de Figueiredo, J. M. Measuring and explaining the gender wage gap in the U.S. federal government. Tech. rep., Duke University Law School, 2017.
- Bolton, A., de Figueiredo, J. M., and Lewis, D. E. Elections, ideology, and turnover in the U.S. federal government. Tech. rep., National Bureau of Economic Research Working Paper 22932, 2016.
- Cameron, A. C. and Miller, D. L. A Practitioners Guide to Cluster-Robust Inference. *Journal of Human Resources*, 50(2):317–332, 2015.
- clustered errors. a. 2017. URL b.
- Connelly, R., Playford, C. J., Gayle, V., and Dibben, C. The role of administrative data in the big data revolution in social science research. *Social Science Research*, 59:1–12, 2016.
- Duke University Synthetic Data Project. feXTX() git repository, 2017. URL <https://github.com/DukeSynthProj/SyntheticDataXTXResources>.
- Economist. Rise of the machines, 2015. URL <https://www.economist.com/news/briefing/21650526-artificial-intelligence-scares-peopleexcessively-so->
- Emerson, J. W. and Kane, M. J. R biganalytics Package, 2016. URL <https://cran.r-project.org/web/packages/biganalytics/biganalytics.pdf>.
- Enea, M., Meiri, R., , and Kalimi, T. R speedlm Package, 2017. URL <https://cran.r-project.org/web/packages/speedglm/speedglm.pdf>.
- Esarey, J. and Menger, A. Practical and Effective Approaches to Dealing with Clustered Data, 2017. URL <http://jee3.web.rice.edu/cluster-paper.pdf>.
- Freedman, D. A. On the so-called "huber sandwich estimator" and "robust standard errors". *The American Statistician*, 60(4):299–302, 2006.

- Gaure, S. R lfe Package, 2016. URL <https://cran.r-project.org/web/packages/lfe/lfe.pdf>.
- Heath, M. T. Parallel Numerical Algorithms, 2013. URL https://courses.engr.illinois.edu/cs554/fa2013/notes/07_cholesky.pdf.
- Higham, N. J. Analysis of the Cholesky Decomposition of a Semi-definite Matrix. 1990. URL http://eprints.ma.man.ac.uk/1193/01/covered/MIMS_ep2008_56.pdf.
- inter-cluster errors. a. 2017. URL b.
- Juskewitch, J. E., Tapia, C. J., , and Windebank, A. J. Lessons from the Salk Polio Vaccine: Methods for and Risks of Rapid Translation, 2010. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2928990/#b15>.
- King, G. How Robust Standard Errors Expose Methodological Problems They Do Not Fix, and What to Do About It. *Political Analysis*, 23:159–179, 2015.
- Koenker, R. and Ng, P. SparseM: A Sparse Matrix Package for R, 2003. URL <http://www.econ.uiuc.edu/~roger/research/sparse/SparseM.pdf>.
- Lazer, I., Pentland, A., Adamic, L., Aral, S., Barabasi, A. L., Brewer, D., Christakis, N., Contractor, N., Fowler, J., Gutmann, M., Jebara, T., King, G., Macy, M., Roy, D., and Alstyne, M. V. Computational Social Science. *Science*, 323(5915):721–722, 2009.
- Liu, I. and Guo, I. Opportunities and challenges of big data for the social sciences: The case of genomics data. *Social Science Research*, 59:13–22, 2016.
- Lumley, T. R biglm Package, 2015. URL <https://cran.r-project.org/web/packages/biglm/biglm.pdf>.
- O’Malley, I. and Wagenaar, I. Effects of minimum drinking age laws on alcohol use, related behaviors and traffic crash involvement among American youth: 1976-1987. *Journal of Studies on Alcohol and Drugs*, 52(5):478–491, 1990.
- R-core. R Parallel Package, 2016. URL <https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf>.
- R Foundation for Statistical Computing. R Reference, 2017. URL <https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf>.
- Ramkumar, P., Acuna, D. E., Berniker, M., Grafton, S. T., Turner, R. S., and Kording, K. P. Chunking as the result of an efficiency computation trade-off. *Nature Communications*, 7:12176, 2016.
- Reiter, J. P. Model diagnostics for remote access regression servers. *Statistics and Computing*, 13:371–380, 2003.

- Reiter, J. P., Oganian, A., and Karr, A. F. Verification servers: Enabling analysts to assess the quality of inferences from public use data. *Computational Statistics and Data Analysis*, 53:1475–1482, 2009.
- Stata Corporation. regress Command, 2017. URL <http://www.stata.com/manuals13/rregress.pdf>.
- Stigler, S. M. *The History of Statistics*. Belknap Harvard, Cambridge, Ma., 1986.
- Tierney, L., Rossini, A. J., Li, N., and Sevcikova, H. R snow Package, 2016. URL <https://cran.r-project.org/web/packages/snow/snow.pdf>.
- U.S. Bureau of Economic Analysis. Gross Domestic Product. URL <https://www.bea.gov/national/index.htm>.
- U.S. Census Bureau. Current Population Survey. URL <https://www.census.gov/programs-surveys/cps.html>.
- U.S. General Services Administration. Data.gov. URL <https://www.data.gov>.
- U.S. Office of Personnel Management. Data, Analysis, and Documentation, a. URL <https://www.opm.gov/policy-data-oversight/data-analysis-documentation/>.
- U.S. Office of Personnel Management. FedScope, b. URL https://www.fedscope.opm.gov/datadefn/aehri_sdm.asp.
- U.S. Office of Personnel Management. Guide to Data Standards, c. URL <https://catalog.data.gov/dataset/guide-to-data-standards-gds>.
- Vandenberghe, L. QR Factorization, 2017. URL <http://www.seas.ucla.edu/~vandenbe/133A/lectures/qr.pdf>.
- Wikipedia. Cholesky decomposition, 2017a. URL <https://pdfs.semanticscholar.org/f229/a57ee5611ca84a8936fd9c29a3f1f19dc1e9.pdf>.
- Wikipedia. QR Decomposition, 2017b. URL https://en.wikipedia.org/wiki/QR_decomposition.
- Williams, R. Panel Data: Very Brief Overview, 2015. URL <https://www3.nd.edu/~rwilliam/stats2/Panel.pdf>.
- Zeileis, A. Object-oriented computation of sandwich estimators. *Journal of Statistical Software*, 16(9), 2006.