Efficient Solution of Large Fixed Effects Problems Using R

Tom Balmat[*], Jerome P. Reiter[†]

June 30, 2018

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 explore various methods for solving large, sparse fixed effects OLS problems and propose a method, using a combination of available R packages and custom algorithms, that efficiently computes parameter estimates and standard errors without creation of the complete expanded design matrix and limiting computations to those involving non-zero level effects. 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, efficient high dimension computation, 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 $\boldsymbol{X'X}\hat{\boldsymbol{\beta}} = \boldsymbol{X'Y}$ for $\hat{\boldsymbol{\beta}}$. For small designs, where $\boldsymbol{X}(n \times p)$ 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 $10^7 \times 10^3$ and above, present special storage and computational challenges.[1] For example, creation of a 50,000,000 x 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 nearly one terabyte, of memory. Further, to generate $\boldsymbol{X'X}$ with standard R functions, that is using `t(X)%*%X`, requires an additional copy of $\boldsymbol{X}$ 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.[3] In addition to memory constraints, solution feasibility is practically limited by the processing time required to compose $\boldsymbol{X'X}$ and $\boldsymbol{X'Y}$, solve $\boldsymbol{X'X}\hat{\boldsymbol{\beta}} = \boldsymbol{X'Y}$ for parameter estimates, and compute $\hat{\boldsymbol{\sigma}}^2(\boldsymbol{X'X})^{-1}$ for (homoskedastic) standard errors. While solving large ($10^7 \times 10^3$ and above) fixed effects regression problems, and searching for optimal methods among those currently available, it was observed that certain solutions concentrate on reduced memory usage, others target efficiency in composing $\boldsymbol{X'X}$ and computing parameter estimates, yet none were identified that could be considered optimal in computing standard errors. A strategy was heuristically developed that utilizes available R packages where optimal, combined with customized algorithms to improve computational efficiency where needed. This strategy is presented along with an evaluation of the efficiency of various alternatives. Additionally, an efficient algorithm is presented for estimating robust and clustered standard errors.[4] Traditional clustered standard error estimation involves the variance equation $\text{Var}(\hat{\boldsymbol{\beta}}) = \text{diag}\left[(\boldsymbol{X'X})^{-1}\boldsymbol{X'UX}(\boldsymbol{X'X})^{-1}\right]$, where $\boldsymbol{U}$ is the $n \times n$ matrix of within cluster residual covariances, and is estimated by setting all inter-cluster errors, $(\boldsymbol{\epsilon'\epsilon})_{cluster}$, 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 $\text{Var}(\hat{\boldsymbol{\beta}})$ 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

[1] `lm()` is the base linear regression function of R (R Foundation for Statistical Computing, 2017, R Reference, `lm()`). Inspection of the program source for `lm()`, using `edit(lm())`, reveals a sequence of function calls to the R function `lm.fit()`, the C function `.Call(C_Cdqrls)`, and finally the Fortran functions `dqrls()` and `dqrdc()`. `dqrls()` and `dqrdc()` reveal that, instead of solving $\boldsymbol{X'X}\hat{\boldsymbol{\beta}} = \boldsymbol{X'Y}$ directly, the QR decomposition of $\boldsymbol{X}$ is computed, then $\boldsymbol{R}\hat{\boldsymbol{\beta}} = \boldsymbol{Q'Y}$ is solved. For program listings see (R-core, 2017b, `Cdqrls`) and (R-core, 2017a, `dqrls.f`, `dqrdc.f`). Due to storage and computation inefficiencies, computing the QR decomposition of $\boldsymbol{X}$ is generally impractical for problems of the size considered in this paper. QR decomposition is compared to alternative methods in *Solving the OLS Normal Equations*, below.

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

[3] Sparse matrix methods offer additional conservation by encoding non-zero matrix elements only. For more on sparse methods, see Koenker and Ng (2003) and Bates and Maechler (2017).

[4] Robust and clustered (heteroskedastic) 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)).

[5] There 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).

Var($\hat{\boldsymbol{\beta}}$) to achieve very reasonable solution times (minutes) for problems on the order of those mentioned above.

2 Background

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.[6] Models varied, but researchers were interested in estimates for parameters and their homoskedastic and/or heteroskedastic (clustered) standard errors. A simultaneous requirement arose from the Synthetic Data Project, also at Duke, to certify a synthetic U.S. federal human capital data set to be used in a public use verification server system.[7]

2.1 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).[8] 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.[9] 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).[10] 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.[11] The model is

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

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

[6]For more on the Human Capital Project at Duke, see Duke University Human Capital Project.
[7]For more on synthetic data and verification systems, see Reiter et al. (2009); Reiter (2003); Barrientos et al. (2017).
[8]For a general description of OPM data resources, see U.S. Office of Personnel Management (a).
[9]For more on CPDF Status records, see U.S. Office of Personnel Management (b).
[10]See OPM Guide to Data Standards, U.S. Office of Personnel Management (c).
[11]Bolton and de Figueiredo interact *sex* and *race* by fitting separate models for each sex.

log(*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 $\boldsymbol{X}$, 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 kth 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 $24,574,480 \times 1,236$. 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 $\boldsymbol{X}$ 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 $\boldsymbol{X}$ and it is this property that sparse methods exploit in efficient construction of $\boldsymbol{X'X}$ and solution of the OLS normal equations. 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 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

3 Design Matrix and Operation Density

To solve $\boldsymbol{X'X}\hat{\boldsymbol{\beta}} = \boldsymbol{X'Y}$, we need $\boldsymbol{X'X}$. Suppose that we are given a large, sparse design matrix $\boldsymbol{X}$ ($25,000,000 \times 2,000$ 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 $\boldsymbol{X'X}$ with matrix operations is to accumulate products of pairs of non-zero elements (indicated by i and j) into corresponding positions of the resulting $\boldsymbol{X'X}$ matrix.[12] Since $\boldsymbol{X'X}$ is symmetric, additional efficiency is gained by accumulating the upper triangle only, then copying its transpose to the lower triangle. Sparse methods imply limiting computations strictly

[12]The standard method in R is `Z <- t(X)%*%X`.

to those that affect the final result, in the case of matrix multiplication, products that are non-zero. The *Matrix and Operation Density* appendix establishes 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 $\boldsymbol{X}'\boldsymbol{X}$.[13] Figure 2 (*Operation Density*) of the appendix indicates that for models involving fixed effects with approximately 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. Model (1), combined with the example OPM data set, has an operation density of approximately 0.00001, making it a candidate for computational efficiency improvement.[14]

4 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 (SAS Institute, 2017, `proc GLM`), the `regress` command of Stata (Stata Corporation, 2017, `regress`), 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 $\boldsymbol{X}'\boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{X}'\boldsymbol{Y}$ for $\hat{\boldsymbol{\beta}}$.[15] The *Review of Available Solutions* appendix evaluates popular software solutions for solving large OLS regression problems.[16] Since this paper targets an R audience, solutions are limited to R functions and packages available at time of writing, including `lm()`,[17] `biglm()`,[18] `bigglm()`,[18] `biglm.big.matrix()`,[19] `SparseM.slm.fit()`,[20] `speedlm()`,[21] `lfe()`,[22] and combined functions from the `Matrix` and `Matrix Models` packages.[23] Each addresses efficiency obstacles in different 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. Each solves or approximates solution of the OLS normal equations, providing estimates for all continuous and fixed effect parameters. Some provide homoskedastic standard error estimates, some provide matrix operations for

[13] The *Matrix and Operation Density* appendix is available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).

[14] Operation density can be derived from a constructed $\boldsymbol{X}'\boldsymbol{X}$ 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

[15] Verification 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).

[16] Available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).

[17] See (R Foundation for Statistical Computing, 2017, R Reference, `lm()`).

[18] See Lumley (2015).

[19] See Emerson and Kane (2016).

[20] See Koenker and Ng (2003).

[21] See Enea et al. (2017).

[22] See Gaure (2016).

[23] These include `sparse.model.matrix()`, `lm.fit.sparse()`, and `solve()` (Bates and Maechler, 2017, 2015).

estimating standard errors using $\hat{\boldsymbol{\sigma}}^2(\boldsymbol{X}'\boldsymbol{X})^{-1}$, while few offer estimates of heteroskedastic robust or clustered standard errors. 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.

While testing available solutions for feasibility and efficiency, with a requirement to meet project deadlines, the authors developed algorithms and programs in R and C for efficiently indexing $\boldsymbol{X}$, composing $\boldsymbol{X}'\boldsymbol{X}$, solving for parameter estimates, and computing homoskedastic, robust, and clustered standard errors.[24] The custom R function `feXTX()` implements this solution and performance of it is compared to other methods in *Performance of Solutions with Simulated Data and Models* and *Performance with OPM Data and Models*, below.

An alternative to solving the OLS equations, $\boldsymbol{X}'\boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{X}'\boldsymbol{Y}$, for all $\hat{\boldsymbol{\beta}}$ is to re-specify the problem, transforming continuous independent variables to deviations from within-level means. This is referred to as *demeaned regression.* Although efficient, the method does not provide estimates for fixed effect parameters or standard errors for any parameter. It is mentioned for completeness, covering the case where only continuous variable parameter estimates are desired.[25]

5 Solving the OLS Normal Equations

It will be seen in later sections on performance that, for large fixed effects problems, the most efficient methods involve solution of $\boldsymbol{X}'\boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{X}'\boldsymbol{Y}$ for parameter estimates, $\hat{\boldsymbol{\beta}}$, and computation of $\hat{\boldsymbol{\sigma}}^2(\boldsymbol{X}'\boldsymbol{X})^{-1}$ for standard errors (as opposed to some form of approximation). We see that, in addition to $\boldsymbol{X}'\boldsymbol{X}$, $\boldsymbol{X}'\boldsymbol{Y}$ is needed but, as demonstrated in the *Efficient Indexing* and *Review of Available Solutions* appendices, it is efficiently produced with sparse indexing or matrix multiplication operations.

Of the various methods for solving $\boldsymbol{X}'\boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{X}'\boldsymbol{Y}$ for $\hat{\boldsymbol{\beta}}$ two that are well established and known for computational efficiency are QR decomposition and Cholesky decomposition.[26] [27] The efficiency of QR and Cholesky decomposition derive from a strategy of forming pairs of matrices, with at least one triangular, such their product represents the left hand matrix in the system being solved, $\boldsymbol{X}'\boldsymbol{X}$ in our case.[28] In the case of Cholesky decomposition, an upper triangular matrix $\boldsymbol{R}$ is computed such that $\boldsymbol{R}'\boldsymbol{R} = \boldsymbol{X}'\boldsymbol{X}$, then $\boldsymbol{R}'\boldsymbol{R}\hat{\boldsymbol{\beta}} = \boldsymbol{X}'\boldsymbol{Y}$ is solved in two stages: $\boldsymbol{R}'\boldsymbol{W} = \boldsymbol{X}'\boldsymbol{Y}$ for $\boldsymbol{W}$ then $\boldsymbol{R}\hat{\boldsymbol{\beta}} = \boldsymbol{W}$ for $\hat{\boldsymbol{\beta}}$. The triangular nature of R

[24] These are presented in appendices *Efficient Indexing of X in Composing $X'X$*, *An Efficient $X'X$ Indexing Algorithm in R*, *Parallel Cholesky Decomposition Algorithm*, and X^{-1} *Using Parallel Cholesky Decomposition*, available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).

[25] For more on demeaned regression, see Gormley and Matsa (2013) and Williams (2015).

[26] For 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()`.

[27] For 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()`).

[28] An alternate method, as used by `lm()`, is to compute the QR decomposition of X, such that $QR = X$, where Q is $(n \times p)$ orthogonal and R is $(p \times p)$ triangular, giving $X'X\hat{\beta} = X'Y \implies R'Q'QR\hat{\beta} = R'Q'Y \implies R'R\hat{\beta} = R'Q'Y \implies R\hat{\beta} = Q'Y$, assuming R is invertible. However, for even small fixed effects problems $(1,000,000 \times 500)$, the computational cost of computing Q and R is many times that of composing $X'X$ then computing its QR decomposition to solve $X'X\hat{\beta} = X'Y$. For large problems, as mentioned in the introduction, storage of Q $(n \times p)$ becomes prohibitive. For a comparison of the efficiency of computing the QR decomposition of X vs. that of $X'X$, see the *QR Decomposition of X vs. $X'X$* appendix, available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).

makes solution of associated equations very efficient by enabling the use of computationally simple forward and backward solving steps. In addition to facilitating solution of $\boldsymbol{X}'\boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{R}'\boldsymbol{R}\hat{\boldsymbol{\beta}} = \boldsymbol{X}'\boldsymbol{Y}$, $\boldsymbol{R}$ can be used to solve $\boldsymbol{X}'\boldsymbol{X}\boldsymbol{W} = \boldsymbol{I}$, where $\boldsymbol{W} = (\boldsymbol{X}'\boldsymbol{X})^{-1}$, the computationally significant component of the homoskedastic parameter covariance equation $\text{Cov}(\hat{\boldsymbol{\beta}}) = \sigma^2 (\boldsymbol{X}'\boldsymbol{X})^{-1}$. QR decomposition follows a similar strategy, but with matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, where $\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{X}'\boldsymbol{X}, \boldsymbol{Q}$ is orthogonal, and $\boldsymbol{R}$ is upper triangular. In practice, the computational effort required to generate $\boldsymbol{R}$ (or $\boldsymbol{Q}$ and $\boldsymbol{R}$) combined with solving two simpler systems tends to be less than in solving $\boldsymbol{X}'\boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{X}'\boldsymbol{Y}$ using direct Gaussian elimination and the resulting equations and solution tend to exhibit improved numerically stability.[29] Although both QR decomposition and Cholesky decomposition produce near mathematically equivalent results for problems of the size considered, 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 $\hat{\boldsymbol{\beta}}$ and one for $\text{Cov}(\hat{\boldsymbol{\beta}})$, than does QR decomposition. As an example, table 2 compares times to compute the QR decomposition, Cholesky decomposition, and inverse of $\boldsymbol{X}'\boldsymbol{X}$ 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()`.[30] [31] [32] The apparent four to one performance ratio gives Cholesky decomposition a significant advantage.

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

Method	6,200 level matrix	16,000 level matrix
`qr() + qr.solve()`*	10.0	172.7
`chol() + chol2inv()`	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 fixed effects regression problems, it was observed that as much as three-fourths of the entire computation cycle [composition of $\boldsymbol{X}'\boldsymbol{X}$, computation of $\boldsymbol{R}$, solution of $\hat{\boldsymbol{\beta}}$, solution of $\text{Cov}(\hat{\boldsymbol{\beta}})$] was devoted to the final three items, all involving $\boldsymbol{R}$. Although, algorithmically, $\boldsymbol{R}$ 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.[33] As part of the current project, two Cholesky related functions were developed: one to compute the decomposition $\boldsymbol{R}$, given a composed $\boldsymbol{X}'\boldsymbol{X}$, and one to compute $(\boldsymbol{X}'\boldsymbol{X})^{-1}$ using $\boldsymbol{R}$. Both are implemented in C, using the R package `Rcpp`[34], and employ parallel methods, using `OpenMP`[35]. Source listings are given in appendices *Parallel Cholesky Decomposition*

[29]See Higham (1990).

[30]Data sets were generated as described in *Performance of Solutions with Simulated Data and Models*, below.

[31]`qr.solve(`$\boldsymbol{X}$`)` computes the inverse of $\boldsymbol{X}$ (R Foundation for Statistical Computing, 2017, `qr.solve()`).

[32]`chol2inv(`$\boldsymbol{R}$`)` computes the inverse of $\boldsymbol{X} = \boldsymbol{R}'\boldsymbol{R}$ (R Foundation for Statistical Computing, 2017, `chol2inv()`).

[33]This is verified by observing processor activity during execution.

[34]See Eddelbuettel et al. (2017).

[35]Available in `Rtools` (Ripley and Murdoch, 2017).

Algorithm and *Parallel Algorithm to Compute Inverse of $\boldsymbol{X}$ Using the Cholesky Decomposition*.[36] Function `choleskyDecomp()` computes $\boldsymbol{R}$ and function `cholInvDiag()` computes $(\boldsymbol{X}'\boldsymbol{X})^{-1}$. Figure 1 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.[37] 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.

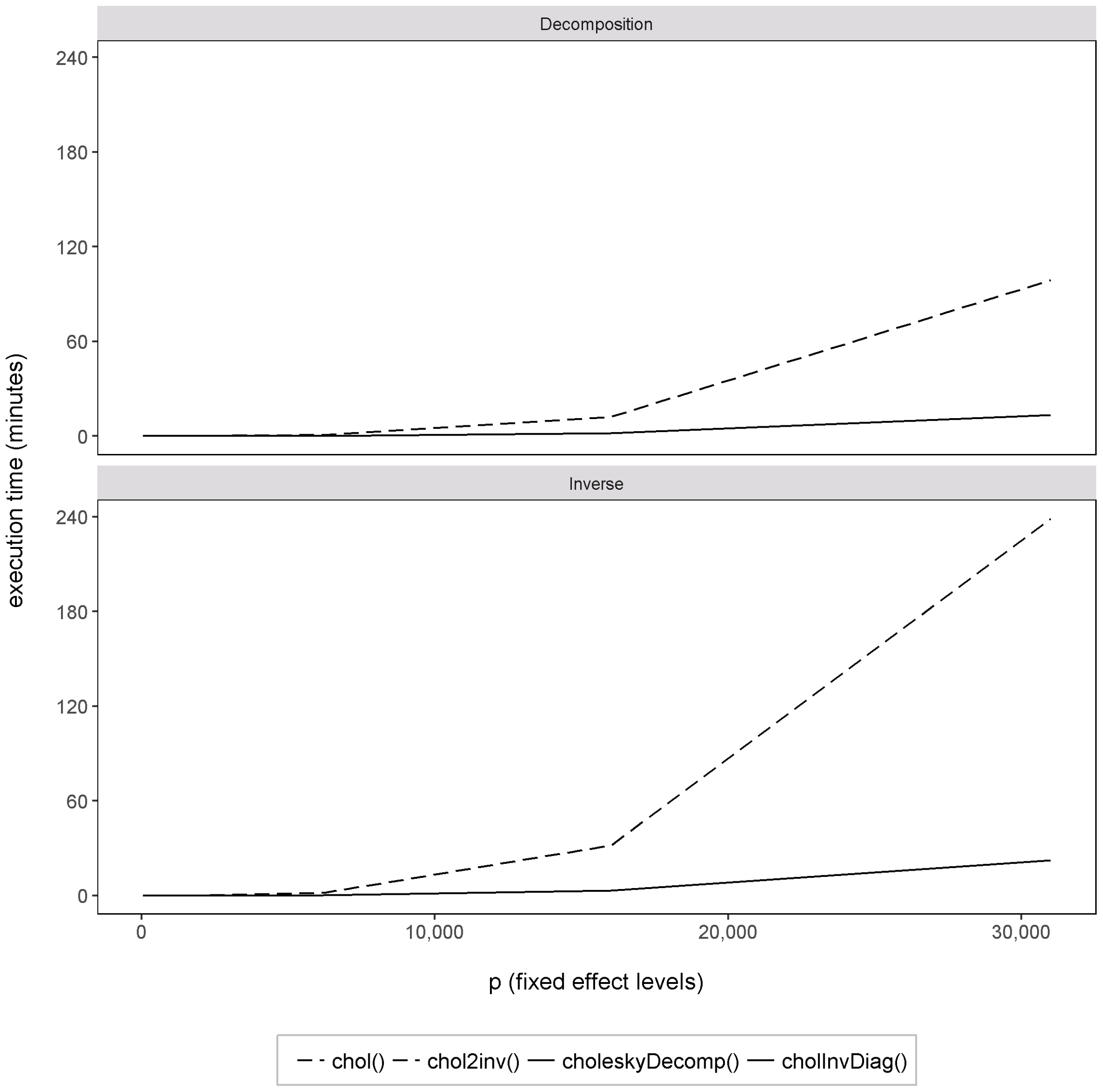

Figure 1: Comparison of execution times to compute Cholesky decomposition and $(\boldsymbol{X}'\boldsymbol{X})^{-1}$ using custom parallel algorithm and base R functions. Five independent data sets were generated for each level of p_k (number of fixed effect levels). Observation counts ranged from 10,000 (small p_k) to 25,000,000 (large p_k).

[36]Available in the om-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).
[37]Sample data sets were generated as described in *Performance of Solutions with Simulated Data and Models*, below.

To verify consistency of results between the custom parallel and base R functions, element-wise differences in computed $\boldsymbol{R}$ or $(\boldsymbol{X}'\boldsymbol{X})^{-1}$ matrices of absolute value greater than 10^{-10} were tested for using `which`(`abs`($\boldsymbol{R}_1 - \boldsymbol{R}_2$) $>$ 1e^{-10}) and `which`(`abs`($(\boldsymbol{X}'\boldsymbol{X})_1^{-1} - (\boldsymbol{X}'\boldsymbol{X})_2^{-1}$) $>$ 1e^{-10})), where $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are custom and base R Cholesky decomposition results, respectively, and $(\boldsymbol{X}'\boldsymbol{X})_1^{-1}$ and $(\boldsymbol{X}'\boldsymbol{X})_2^{-1}$ are custom and base R inverses, respectively. None were reported. *Revise this evaluation to measure the number of significant digits that agree between two corresponding values, since the current method merely reports the position of first digits in error - seemingly small deviations could actually be significant when compared values are sufficiently small.*

6 Performance of Solutions with Simulated Data and Models

We now compare the performance and results of methods presented in section 4. Since the example OPM data set mentioned in section 2.1 is not publicly available, simulated data sets with specifiable covariate relationships will be used, allowing general comparison and a ready means for the reader to test methods presented. The *Simulated Data Generation and Tests* appendix 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.[38] Y is simulated as a linear function of the X values plus normally distributed error such that

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

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.

Also in the *Simulated 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()`, `feXTX()`, `Matrix`$_1$, and `Matrix`$_2$.[39] [40] [41] Note that appropriate packages must be installed in order to execute related model fitting functions. Figure 2 plots mean execution time by method using five independently simulated data sets for each of parameter sets 1 through 9. The significant increase in execution time with respect to problem size for methods 1 through 6 indicates a lack of readiness for these current and popular solution methods to solve large fixed effects problems. In addition to execution efficiency, the

[38] The appendix is available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).

[39] *Fitting* consists of generating coefficient and (homoskedastic) standard error estimates for all continuous and fixed effect independent variables.

[40] `feXTX()` employs efficient indexing and the custom parallel Cholesky algorithms `choleskyDecomp()` and `cholInvDiag()`, as earlier presented, along with base R functions `forwardsolve()` and `backsolve()`.

[41] `Matrix`$_1$ employs `sparse.model.matrix()`, `lm.fit.sparse(method="cholesky")`, and `solve()` from the `Matrix` and `Matrix Models` packages. `Matrix`$_2$ employs `sparse.model.matrix()` from the `Matrix Models` package, base R functions `forwardsolve()` and `backsolve()`, and custom parallel Cholesky algorithms `choleskyDecomp()` and `cholInvDiag()`.

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

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).

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

Method	Memory Used (Gb)
`lm()`	19.3
`biglm()`	19.3
`bigglm()`	19.4
`biglm.big.matrix()`	0.80*
`SparseM.slm.fit()`	20.3
`speedlm()`	28.8
`lfe()`	10.4
`feXTX()`	1.0
$\texttt{Matrix}_1$	1.0
$\texttt{Matrix}_2$	1.0

* backing file created

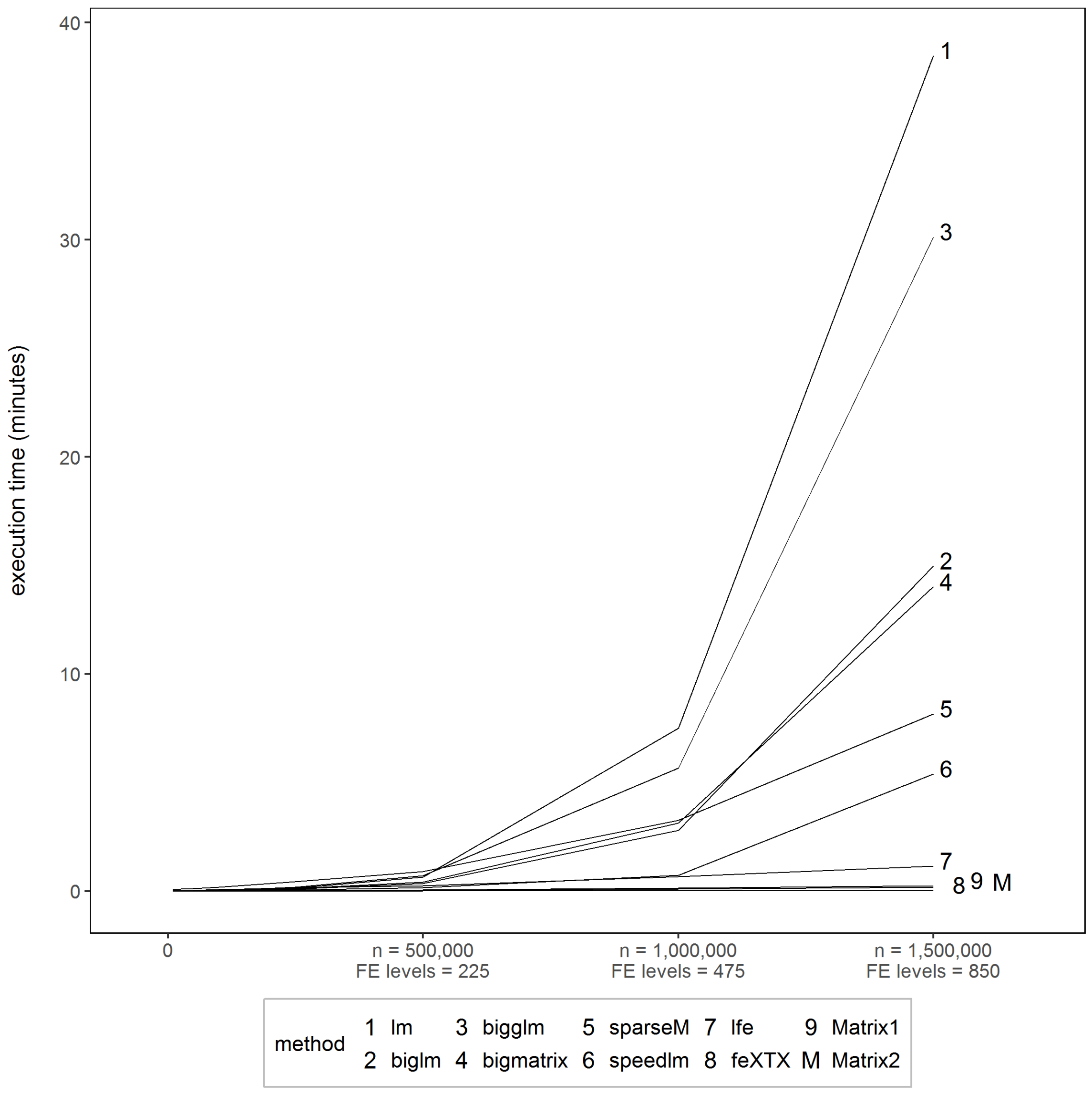

Figure 2: Mean execution time to solve graduated fixed effects problems by method. Execution time is the differential of "elapsed" time as reported by `proc.time()`. All processing was performed on a single dedicated server.

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.[42] *Revise this evaluation to measure the number of significant digits that agree between two corresponding values, since the current method merely reports the position of first digits in error - seemingly small deviations could actually be significant when compared values are sufficiently small.* There does seem to be consensus, with all methods except `lfe()` having all deviations of parameter estimates less than 10^{-7}.[43] [44]

Table 5: Comparison of Parameter Estimates, Evaluated Solutions vs. `feXTX()`, Simulated Data Set 9

Method	$0 < 10^{-12}$	$10^{-12} < 10^{-10}$	$10^{-10} < 10^{-7}$	$10^{-7} < 10^{-5}$	$> 10^{-5}$
`lm()`	384	464	2	0	0
`biglm()`	341	507	2	0	0
`bigglm()`	341	507	2	0	0
`bigmatrix()`	341	507	2	0	0
`sparseM()`	0	101	749	0	0
`speedlm()`	0	101	749	0	0
`lfe()`	0	0	3	0	847
`Matrix`$_1$	0	101	749	0	0
`Matrix`$_2$	0	101	749	0	0

In terms of execution and memory efficiency, `lfe()`, `feXTX()`, `Matrix`$_1$, and `Matrix`$_2$ 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()`, `feXTX()`, `Matrix`$_1$, and `Matrix`$_2$, we see in figure 3 a continued divergence of execution time, with `feXTX()` and `Matrix`$_2$ exhibiting near linear increase with respect to problem size, executing the largest problem, with 25,000,000 observations and 31,000 fixed effect levels, in one-twelfth to less than one-half the time required by `lfe()` and `Matrix`$_1$, respectively. Memory requirements to fit model (2) using parameter set 13 were ??? Gb for `lfe()`, ??? Gb for `feXTX()`, ??? Gb for `Matrix`$_1$, and ??? Gb for `Matrix`$_2$. An important consideration is that the `Matrix`$_2$ method is a hybrid solution, constructing $\boldsymbol{X'X}$ using sparse functions from the `Matrix` package, while computing parameter estimates and standard errors using functions from `feXTX()`. Figure (*include figure*) compares `feXTX()` and `Matrix`$_2$ execution times for data sets

[42] The smallest magnitude of all `feXTX()` parameter estimates, for this simulated data set, was 0.5. Absolute deviations from this value of less than 10^{-7} should be considered insignificant. Comparison functions are available in the *Simulated Data* appendix, available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).

[43] Although `feXTX()` and `Matrix`$_2$ use identical Cholesky and forward/back solve functions, the difference in estimates is traced to a difference in $X'X$ construction, `feXTX()` using indicator columns for counts and sums of indexed columns, `Matrix`$_2$ executing traditional matrix multiplication.

[44] By default, `lfe()` does not estimate an intercept and fixed effect reference levels are not specifiable when the faster "kaczmarz" method is used (the method appears to force highest frequency levels as reference). This causes differences in remaining estimates. Differences in effects between non-reference level and desired reference level could be computed from supplied estimates, but this was not pursued. The alternative "cg" method permits reference level specification but, in trials conducted with data and models presented, is many times slower than "kaczmarz." For more on this, see (Gaure, 2016, pg. 26).

generated using parameter sets 1 through 13, plus an additional test set consisting of 50,000,000 observations and 10,000, 20,000, and 50,000 fixed effects levels for X_3, X_4, and X_5, respectively. It is seen that `feXTX()` outperforms `Matrix`$_2$ for data sets 1 through 12 and both methods have approximately equivalent performance for data set 13, but for the additional, larger data set, `Matrix`$_2$ outperforms `feXTX()`. Although nuanced, these results offer a mix of performance characteristics that can be employed to maximize overall efficiency. In developing synthetic data verification measures, the authors have conducted simulation studies using fixed effects models similar to model (1) fit to partitions of data, resulting in over 10,000 individual models being fit per simulation. On this order, choosing a solution method that is optimal for the data sets employed can lead to significant overall savings of time. (*cite VM paper*). Another important consideration is that `feXTX()` provides fixed effect indicator columns useful in computing robust and clustered standard errors as presented in section 8. It is likely that compatible indicator columns can be extracted from a sparse matrix generated by functions of the `Matrix` package, but that is beyond the scope of our discussion.

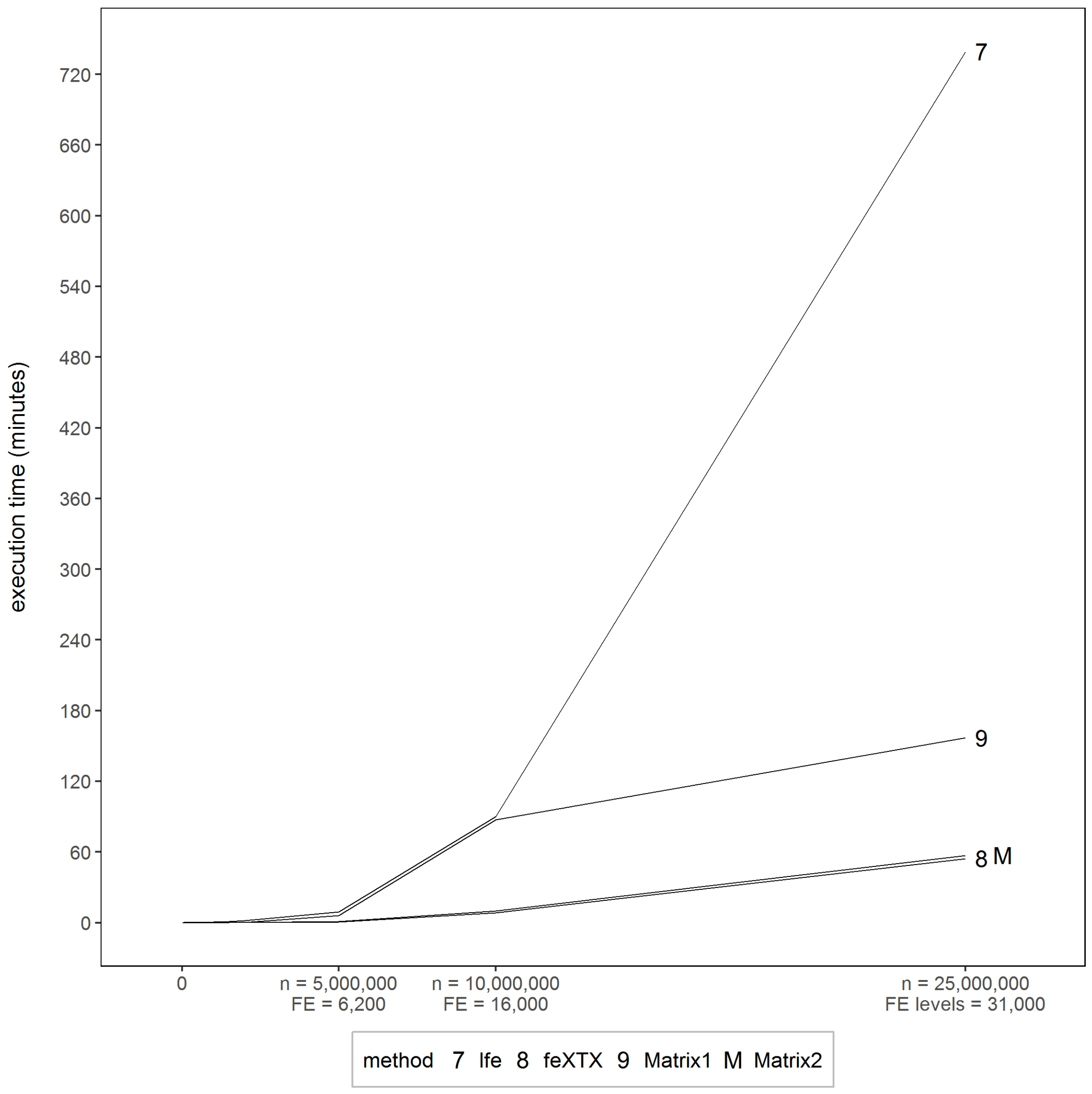

Figure 3: Mean execution time to solve graduated fixed effects problems, including high dimension effects. Limited to `lfe()`, `feXTX()`, `Matrix`$_1$, and `Matrix`$_2$ methods. Execution time is the differential of "elapsed" time as reported by `proc.time()`. All processing was performed on a single dedicated server.

7 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.[45] Due to obstacles (errors during execution) and lengthy execution times with larger models, these solutions are not considered here. As previously stated, the example OPM data set contains 24,574,480 observations and two moderate dimension fixed effects (bureau

[45]Available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).

and occupation, with 409 and 791 levels, respectively). A fully expanded design matrix would have dimension $24,574,480 \times 1,236$ and relatively low density (matrix density of 0.0006, operation density of 0.00001). Problem size, density, and the fact that actual area research was delayed due to difficulties obtaining parameter and standard error estimates to hypothesized models in reasonable time, make the data set and model (1) good candidates for analysis. Execution times and memory requirements to fit model (1) and compute homoskedastic standard errors, using the example OPM data, are listed in table 6.[46] [47]

Table 6: Execution Time and Memory Required to Fit Model (1) to OPM Data

Method	Execution Time (min)	Memory used (Gb)
`lfe()`	34.8	128*
`feXTX()`	2.6	6
`Matrix`$_1$	6.0	8
`Matrix`$_2$	3.8	8

* 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

As with the simulated data sets and models, the `feXTX()` and `Matrix`$_2$ methods significantly outperform `lfe()` and `Matrix`$_1$, with no penalty in memory usage. As previously mentioned, `Matrix`$_2$ consists of sparse matrix functions (importantly `sparse.model.matrix()`) from the `Matrix` and `Matrix Models` packages combined with custom Cholesky decomposition and inverse functions from `feXTX()`. While searching for efficient solutions to large computational problems, it is important to observe and measure performance of individual components and to develop methods for combining optimal elements into an overall optimal solution. Table 7 lists, by method, absolute deviations of coefficient estimates to those computed by `feXTX()`. With a minimum estimate of 0.0003 (absolute value, computed by `feXTX()`), deviations of less than 10^{-7} indicate at least three significant digits of precision, making deviations substantively insignificant.[48] As previously mentioned, restrictions on reference level specification cause significant deviations in certain estimates computed by `lfe()`.[49]

[46] Although the OPM data are not available for public release, the appendix *Measuring Performance of Fixed Effect Solutions Using OPM Data*, available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository), contains the R instructions and function calls used to generated this table.

[47] For `feXTX()` execution, *sex* $\times$ *race* interaction columns were constructed and declared as continuous variables, since it has been observed that, for interactions of low dimension fixed effects (*sex* has two levels, *race* has five), this approach is generally more efficient than having the indexing algorithm compose interaction columns. Often, with experimentation, observation, and measurement, algorithm features or alternate model specification can be employed for efficiency gains.

[48] This has been confirmed in discussions with members of the Human Capital project at Duke.

[49] On inspection, it is observed that `lfe()` chooses highest frequency levels as reference levels. By coincidence, the Human Capital models also use highest frequency levels for Bureau and Occupation. This enables `lfe()` to produce estimates that are nearer to those of `feXTX()`, `Matrix`$_1$, and `Matrix`$_2$ with models fit to the OPM data than with models fit to simulated data.

Table 7: Comparison of Parameter Estimates, Evaluated Solutions vs. `feXTX()`, OPM Data

Method	$0 < 10^{-12}$	$10^{-12} < 10^{-10}$	$10^{-10} < 10^{-7}$	$10^{-7} < 10^{-5}$	$> 10^{-5}$
`lfe()`	2	145	1,022	27	36
`Matrix`$_1$	0	1	1,231	0	0
`Matrix`$_2$	0	1	1,231	0	0

8 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 analytical equation for $\mathrm{Cov}(\hat{\boldsymbol{\beta}})$ derived from the OLS normal equations:

$$\mathrm{Cov}(\hat{\boldsymbol{\beta}}) = \left(\boldsymbol{X}'\boldsymbol{X}\right)^{-1} \boldsymbol{X}'\boldsymbol{u}\boldsymbol{u}'\boldsymbol{X} \left(\boldsymbol{X}'\boldsymbol{X}\right)^{-1} \tag{3}$$

where $\boldsymbol{uu'}$ 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).[50] Note that $\boldsymbol{uu'}$ 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, $\boldsymbol{X}(24,574,480 \times p)$. Standard matrix operations involving objects of this size are computationally impractical, in both time and memory requirements. However, sparse indexing methods similar to those used by `feXTX()` to compose $\boldsymbol{X'X}$ can be used to compute $\mathrm{Cov}(\hat{\boldsymbol{\beta}})$ very efficiently.[51] In fact, having constructed sparse indices to fixed effect columns of $\boldsymbol{X}$ along with $\left(\boldsymbol{X}'\boldsymbol{X}\right)^{-1}$, we simply proceed with construction of $\boldsymbol{U}$ (which is efficiently accomplished in parallel), then multiplication of matrices in equation (3) referencing only non-zero elements of $\boldsymbol{X}$ (which is also done in parallel). Note that, since the standard errors of $\hat{\boldsymbol{\beta}}$ are generally of interest, only $\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \mathrm{diag}\left[\mathrm{Cov}(\hat{\boldsymbol{\beta}})\right]$ is computed.

Robust standard error algorithm outline:

- Compute robust, uncorrelated, heteroskedastic (independent, non-identically-distributed) errors
- Estimate robust variances, square of $\mathrm{SE}(\hat{\boldsymbol{\beta}})$, as $\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \mathrm{diag}\left[(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{uu}'\boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\right]$, where $\boldsymbol{u}$ is the vector of observation response to predicted value errors, $\boldsymbol{u} = \boldsymbol{Y} - \hat{\boldsymbol{Y}}$
- This derives from the expression for parameter variances from the normal OLS equations

[50] For more on heteroskedasticity and inter-cluster independence, see Esarey and Menger (2017).

[51] These are presented in appendices *Efficient Indexing of X in Composing $X'X$* and *An Efficient $X'X$ Indexing Algorithm in R*, available in the on-line repository (Duke University Synthetic Data Project, Fixed effects solution git repository).

- $\boldsymbol{uu}'$ 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 $\boldsymbol{X'uu'X}$ 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 $\text{Var}(\hat{\boldsymbol{\beta}}) = \text{diag}\left[(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{uu'}\boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\right]$, where $\boldsymbol{uu'}$ is the var-cov matrix of observation to estimate errors, $\boldsymbol{uu'} = \text{Cov}(\boldsymbol{Y} - \hat{\boldsymbol{Y}})$
- This is the expression for $\text{Var}(\hat{\boldsymbol{\beta}})$ from ordinary least squares estimates of beta on the assumption that $\boldsymbol{X}$ is non-stochastic and $\boldsymbol{uu'}$ 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
- $\boldsymbol{uu'}$ is altered such that all off-diagonal inter-group elements, that is $\boldsymbol{uu'_{ij}}$ where indices i and j belong to different groups, are set to 0
- The modified (inter-group independent error) $\boldsymbol{uu'}$ has the form:

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

  where $\boldsymbol{uu'}_{group_i}$ 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 $\boldsymbol{uu'}$, $\boldsymbol{X'uu'X}$ is the sum (over all groups i) of $\boldsymbol{X}'_{group_i}\boldsymbol{uu'}_{group_1}\boldsymbol{X}_{group_i}$ where $\boldsymbol{X}_{group_i}$ and $\boldsymbol{uu}'_{group_i}$ are the rows of $\boldsymbol{X}$ and sub-matrix of $\boldsymbol{uu}'$ corresponding to group i [to see this, imagine multiplying $\boldsymbol{X}'$ by $\boldsymbol{uu}'$ and consider the effect of $\boldsymbol{uu}'$ 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 $\boldsymbol{X}$ (each row belongs to a single group) gives a $p \times p$ sum of individual group products]
- Note that $\boldsymbol{X}'_{group_i}\boldsymbol{u}_{group_i} = [\boldsymbol{u}'_{group_i}\boldsymbol{X}_{group_i}]'$, making $\boldsymbol{X}'_{group_i}\boldsymbol{uu}'_{group_i}\boldsymbol{X}_{group_i}$ symmetric
- Also, $\boldsymbol{u}'_{group_i}\boldsymbol{X}_{group_i}$ is a $1 \times p$ vector, where p is the column dimension of $\boldsymbol{X}$ (the design matrix)

- For each group ID, construct $\boldsymbol{v} = \boldsymbol{u}'_{group_i}\boldsymbol{X}_{group_i}$ then use it to compute $\boldsymbol{v}'\boldsymbol{v} = \boldsymbol{X}'_{group_i}\boldsymbol{u}\boldsymbol{u}'_{group_i}\boldsymbol{X}_{group_i}$

8.1 Evaluation of Robust Standard Errors

Support for robust and clustered standard errors (RCSE) differs among various regression solutions available for R. Although the `Matrix` package lacks an RCSE option, its sparse matrix operations should support evaluation of equation (3). Unfortunately, the authors have not had success with this (*continue and provide details on errors observed*). `lfe()` offers an RCSE feature, but estimates differ from those of other solutions, unless estimable functions for an intercept and non-reference level fixed effects are specified. `feXTX()` computes RCSE by direct evaluation of equation (3) using sparse indicator vectors composed during $\boldsymbol{X}'\boldsymbol{X}$ construction along with specifiable fixed effect reference levels. Computational efficiency and numerical accuracy of `feXTX()` RCSE results will be compared to those of the `regress()` command of Stata, which is known for computational efficiency and accuracy, using a common set of fixed effect reference levels.[52] Table 8 lists execution times to fit model (1) to the OPM data described in section 2.1 and to compute robust and clustered standard errors using `lfe()`, `feXTX()`, and Stata (version 13.1 of Stata/MP). Estimable functions are not specified for `lfe()` and results for it are included for execution time comparison only. All processing was executed on a single, dedicated 24 core Windows 7 server. The approximate 2 to 1 performance ratio of `feXTX()` to `lfe()` and Stata indicates an advantage to computing heteroskedastic RCSE by solving the analytical variance equation (3) 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. (*Revise difference in values analysis to include minimum number of common significant digits*)

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)
`lfe()`*	—	51.9	128
`feXTX()`**	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

[52] For more on the Stata regress() command, see (Stata Corporation, 2017)

9 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 (parallel R, under Windows; an `Rcpp/OpenMP` implementation would enable shared memory)
- Parallelization of interaction vector-products
- Implement a hybrid solution where $\boldsymbol{X}'\boldsymbol{X}$ is constructed and normal equations are solved using sparse methods from the `Matrix` package, followed by homoskedastic, robust, and clustered standard error estimation using custom parallel Cholesky and heteroskedastic SE methods demonstrated above
- Implement a parallel algorithm for forward and back solving of OLS system of equations using computed Cholesky decomposition

10 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 effectively 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 $\boldsymbol{X'X}$ using efficient alternatives to vector multiplication
- Solution of parameter estimates from the OLS normal equations ($\boldsymbol{X'X}\hat{\boldsymbol{\beta}} = \boldsymbol{X'Y}$) using a custom, parallelized Cholesky decomposition algorithm implemented in C
- Solution of homoskedastic parameter standard errors from the analytical equation $\mathrm{Cov}(\hat{\boldsymbol{\beta}}) = \hat{\sigma}^2(\boldsymbol{X'X})^{-1}$, 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 $\mathrm{Cov}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{X'X})^{-1}\boldsymbol{X'uu'X}(\boldsymbol{X'X})^{-1}$, using sparse fixed effect index columns produced during construction of $\boldsymbol{X'X}$

Finally, algorithm efficiency was compared to that of eight solutions and 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 taken from actual research in U.S federal pay practices. In all cases, the custom algorithms presented out-performed the remaining methods, 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 Burea of Economic Research Working Paper 23534, 2017.

Bates, D. and Maechler, M. R Matrix Models Package, 2015. URL `https://cran.r-project.org/web/packages/MatrixModels/MatrixModels.pdf`.

Bates, D. and Maechler, M. R Matrix Package, 2017. URL `https://cran.r-project.org/web/packages/Matrix/Matrix.pdf`.

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 Burea 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.

Duke University Human Capital Project. URL `https://ssri.duke.edu/projects/human-capital-career-dynamics-and-organizational-performance-us-federal`

Duke University Synthetic Data Project. Fixed effects solution git repository. URL `https://github.com/DukeSynthProj/FixedEffectsSolutionResources`.

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Russell, N., Bates, D., and Chambers, J. R Rcpp Package, 2017. URL `https://cran.r-project.org/web/packages/Rcpp/Rcpp.pdf`.

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`.

Gormley, T. A.
and Matsa, D. A. Common Errors: How to (and Not to) Control for Unobserved Heterogeneity, 2013. URL `http://portal.idc.ac.il/en/main/research/caesareacenter/annualsummit/documents/gormley-matsa.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`.

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`.

Lumley, T. R biglm Package, 2015. URL
https://cran.r-project.org/web/packages/biglm/biglm.pdf.

R-core. R Parallel Package, 2016. URL
https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf.

R-core. R Project Source, 2017a. URL https://svn.r-project.org/R/trunk/src/appl.

R-core. R Project Source, lm.c function, 2017b. URL
https://svn.r-project.org/R/trunk/src/library/stats/src/lm.c.

R Foundation for Statistical Computing. R Reference, 2017. URL
https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf.

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.

Ripley, B. and Murdoch, D. Rtools, 2017. URL https://cran.r-project.org/bin/windows/Rtools/.

SAS Institute. Proc GLM, 2017. URL
https://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#glm_toc.htm.

Stata Corporation. regress Command, 2017. URL http://www.stata.com/manuals13/rregress.pdf.

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/a57ee5611ca84a8936fdfc29a3f1f19dc1e9.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.