Cumulative Estimation

Damian Clarke Nicolás Paris Torres Benjamín Villena-Roldán

January 14, 2025

Abstract

We document simple procedures which allow for a wide class of econometric estimators to be implemented cumulatively, where, in the limit, estimators can be produced without ever storing more than a single line of data in a computer's memory. This result is useful in understanding the mechanics of many common regression models. These procedures can be used to speed up the computation of estimates obtained via OLS, IV, Ridge regression, LASSO, Elastic Net, and Non-linear models including probit and logit, with all common modes of inference. This has implications for estimation and inference with 'big data', where memory constraints may imply that working with all data at once is particularly costly. We additionally show that even with moderately sized datasets, this method can reduce computation time compared with traditional estimation routines.

Keywords: Big data, estimation, inference, regression, matrix inversion.
JEL codes: C55, C61, C87.

Affiliations: Clarke: University of Exeter, University of Chile, IZA, MIPP, and CAGE. Contact: dclarke@fen.uchile.cl. Paris Torres: University of Chile. Contact: nparis@fen.uchile.cl. Villena-Roldán: Universidad Andrés Bello, MIPP, and LM^{2}C^2. Contact: benjamin.villena@unab.cl.

Acknowledgements: We thank Richard Blundell, Colin Cameron, Samuel P. Engle, Sebastian Kripfganz, James MacKinnon, Jeffrey Wooldridge, Ignacia Mercadal, and Juan Velásquez as well as to participants in the 2023 PUC Alumni Workshop for their feedback and suggestions, and are grateful to Iván Gutierrez Martínez for excellent research assistance. Clarke and Villena-Roldán acknowledge the ANID Millenium Institute for Research in Market Imperfections and Public Policy, ICM IS130002 for financial and institutional support. Villena-Roldán also acknowledges the financial support of the ANID Nucleus Research Centre LM^2C^2 Labor Market Mismatch: Causes and Consequences, NCS2022_045.

1 Introduction

In this paper we formalise procedures for the cumulative estimation of a broad class of regression models, where cumulative estimation refers to estimation in a block-by-block, or line-by-line fashion. We show that many common linear and non-linear estimators can be implemented in a cumulative way, in the limit requiring that no more than a single line of data be stored in a computer's working memory. We document both how this holds for OLS – a result noted in an early computational literature (Brown, Houthakker, and Prais 1953), but also go well beyond this, documenting the fact that similar procedures hold for a range of common estimation methods including IV and 2SLS, LASSO, Elastic Net, as well as non-linear models. We also note that such procedures can be adapted quite simply to calculate common variance-covariance estimators for these models, including heteroscedasticity-robust and cluster-robust variants. We document that such procedures can also be particularly useful for cross-validation and bootstrap-based procedures.

We argue that, beyond the econometric interest of formalising such methods, these procedures also offer a range of practical benefits. When data is so large that it escapes the working memory of a computer, the methods formalised here show how estimation can proceed, with processing time simply scaling linearly with the number of observations. However, even where data is not too large to fit in a computer's working memory, we show that this result may offer a speed-up over standard commercial regression implementations, where in practice processing times tend to not scale linearly with observations due to memory paging, e.g., the storage of blocks (pages) of large data in the hard drive to make computation feasible. While an alternative solution to these issues is to simply gain access to super-computers or large server clusters, this solution may not be feasible for all researchers. The methods documented here can thus be viewed as one way to democratise access to econometric tools. What's more, as highlighted by Varian (2014), Abraham, Jarmin, Moyer, and Shapiro (2022) (among others) in an era of massive datasets, the consideration and documentation of the full breadth of tools to work with very large datasets is important.

While there are previous explorations of cumulative estimation, such as the specific OLS limit case mentioned in Brown, Houthakker, and Prais (1953), a comprehensive framework and rigorous

analysis are absent from the literature. This paper addresses this gap by providing a formal definition of cumulative estimation and establishing its broad applicability for a variety of econometric methods in estimation and inference.

Furthermore, we observe a disconnect between theoretical understanding and practical implementation. Although anecdotal evidence suggests that cumulative estimation techniques are implicitly employed by some practitioners, particularly within the computational architecture of software like SAS, to the best of our knowledge, these methods are not acknowledged or documented explicitly. Even specialized resources such as SAS computational manuals, which emphasize data handling efficiencies through sequential processing (e.g., Jordan (2018)), do not formally recognize or explain cumulative estimation.

This work contributes to the literature by bridging this divide. We offer a unifying theoretical framework for cumulative estimation, demonstrating its potential to enhance both computational efficiency and statistical inference in a wide range of econometric applications. This contribution is timely given the growing challenges posed by large datasets and complex models in modern empirical research.

This work also makes clear that cumulative estimation offers a mathematically equivalent way of implementing a broad class of regression-based estimators without the need to process data in a single instance or even on a single machine, provided that a small number of auxiliary quantities are calculated cumulatively. In this sense, these results can be viewed as relevant for use with 'data silos' in which data is physically separate and must be processed in blocks, as the methods documented here provide exact solutions to estimation and inference procedures where data cannot be pooled. Recent discussions of such settings include Karim, Webb, Austin, and Strumpf (2024) who define procedures for approximation the estimation of difference-in-differences (DID) models based on disjoint datasets *without* pooling data.[1] This work is also related to alternative fast algorithms for arriving to statistical approximations for regression estimates, such as through the use of stochastic gradient descent defined as early as Robbins and Monro (1951). However, our work is

[1] As a matter of fact, Karim, Webb, Austin, and Strumpf (2024) propose methods which exactly replicate DID estimation in certain cases, while closely approximating estimates in others.

distinct from these approaches in that estimates are mathematically equivalent rather than statistical approximations.

This paper is structured as follows. In Section 2 we define the cumulative least squares procedure, showing its equivalence to standard estimation. We begin by showing how this estimator works in cases where estimation proceeds by OLS assuming homoscedasticity, and then document how it holds in a broad range of other estimation and inference procedures. Section 3 discusses the nature of the cumulative procedure, and considerations of optimal block sizes for estimation. In Section 4 we provide a number of illustrations of the performance of these methods compared to commonly-used commercial alternatives. This includes controlled tests where sample sizes and covariate numbers are varied and computational efficiency is compared, as well as an applied example based on a sample of census data and demographic surveys and models with a large number of fixed effects. In Section 5 we provide some additional discussion and conclusions.

2 Cumulative Least Squares

2.1 Cumulative Ordinary Least Squares

Suppose we wish to run a regression of a dependent variable y on a set of K covariates $x_1, x_2, ..., x_K$, using a series of $i = 1, \ldots, N$ observations. Thus, data can be viewed as a matrix or database of size $N \times K$ independent variables which we will denote X, as well as a $N \times 1$ vector y for the dependent variable. Throughout this paper, we will adopt the notation that matrices are written as uppercase italics, vectors are written as lowercase italics, and scalars are defined as required. Suppose also that computing the regression with all the data in memory is either infeasible or undesired due to memory constraints. The data can be partitioned row-wise in J arbitrarily defined portions, where each portion, or block, is labeled as j, and consists of $N_j = N/J$ observations.[2] The blocks are mutually exclusive and cover all observations such that $\sum_{j=1}^{J} N_j = N$. We use the notation X^j to denote block j of size N_j of the independent variables, and similarly, y^j is used to denote block j of the dependent variable.

[2]To fix ideas, we will consider that N_j is common across all blocks. In Section 3 we will discuss optimal choices of N_j, not requiring that this number be equivalent across blocks.

Consider the OLS estimator of the parameter $\widehat{\beta}$. The standard OLS estimator can be written as follows:

$$\begin{aligned}
\widehat{\beta}_{OLS} &= (X'X)^{-1}X'y \\
&\equiv \left(\begin{pmatrix} X^{1\prime} & X^{2\prime} & \cdots & X^{J\prime} \end{pmatrix} \begin{pmatrix} X^1 \\ X^2 \\ \vdots \\ X^J \end{pmatrix} \right)^{-1} \begin{pmatrix} X^{1\prime} & X^{2\prime} & \cdots & X^{J\prime} \end{pmatrix} \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^J \end{pmatrix} \qquad (1) \\
&= \left(X^{1\prime}X^1 + X^{2\prime}X^2 + \cdots + X^{J\prime}X^J\right)^{-1}\left(X^{1\prime}y^1 + X^{2\prime}y^2 + \cdots + X^{J\prime}y^J\right) \qquad (2)
\end{aligned}$$

where in (1) the $K \times N$ matrix X' is re-expressed (identically) as a series of horizontally concatenated $K \times N_j$ matrices, and the $N \times K$ matrix X is similarly re-expressed as a series of vertically concatenated $N_j \times K$ matrices. The $N \times 1$ vector y is also re-written as a series of vertically stacked sub-vectors of dimension $N_j \times 1$. Based on the properties of matrix multiplication, it can easily be seen that elements from each sub-matrix or vector will be interacted only with themselves, and no products are required across blocks. The re-expressed version of $\widehat{\beta}_{OLS}$ in (2) makes clear that $X'X$ can thus be re-written as the summation over the series of J matrices $X^{j\prime}X^j$ which are each of dimension $K \times K$, and a similar procedure can be followed for $X'y$.

This suggests that a cumulative procedure can be followed, as laid out formally in Algorithm 1 below. Specifically, for ease of notation denote $X^{j\prime}X^j \equiv \Sigma_j$ and $X^{j\prime}y^j \equiv \Upsilon_j$. Define as $\Sigma_{1\sim j}$ the summation $\Sigma_1 + \ldots + \Sigma_j$, and $\Upsilon_{1\sim j} = \Upsilon_1 + \ldots + \Upsilon_j$. Then, to estimate (2), initially a single block of data can considered, and the quantities Σ_1 and Υ_1 calculated. In the following step, a new block of data can be consulted, the quantities Σ_2 and Υ_2 calculated, and the preceding quantities summed to provide $\Sigma_{1\sim 2}$ and $\Upsilon_{1\sim 2}$. In following steps, accumulated quantities $\Sigma_{1\sim j-1}$ and $\Upsilon_{1\sim j-1}$ are received at the beginning of each stage, the Σ_j and Υ_j are calculated, and the step ends with $\Sigma_{1\sim j}$ and $\Upsilon_{1\sim j}$. A key element of this procedure is that in each stage, only a single block of data of size $N_j \times (K+1)$ needs to be read into memory, with the results stored in a single accumulated matrix and vector $\Sigma_{1\sim j}$ and $\Upsilon_{1\sim j}$. As Σ and Υ are of dimensions $K \times K$ and $K \times 1$ respectively, this makes clear that we

simply need to keep track of small matrices in an ongoing fashion, and never house more than N_j observations in memory at a single time, where N_j can be an arbitrarily small value, even 1.[3] The OLS estimate $\widehat{\beta}_{OLS}$ is *only* calculated by matrix inversion (or alternative procedures such as Gauss-Jordan elimination) once the full matrices $\Sigma_{1\sim J} \equiv X'X$ and $\Upsilon_{1\sim J} \equiv X'y$ are calculated, implying that potentially costly matrix inversions are not required at every step.

The above process allows for point estimates to be recovered from independent partitions of the database. What's more, inference on regression parameters can be conducted in a similar partition-wise manner. Assuming homoscedasticity (alternative inference procedures are considered in Section 2.4), the well-known formula for the variance of OLS regression parameters is $\widehat{V}(\widehat{\beta}_{OLS}) = \widehat{\sigma}_u^2(X'X)^{-1}$. The quantity $(X'X)$ is already accumulated as laid out above. The second element of the variance is $\widehat{\sigma}_u^2 \equiv \widehat{u}'\widehat{u}/(N-K)$, where the regression residuals $\widehat{u} = y - X\widehat{\beta}_{OLS} = (I - X(X'X)^{-1}X')y = M_X y$, with M_X being the annihilator matrix, an idempotent matrix. Hence:

$$\widehat{u}'\widehat{u} = y'M_X y = y'y - y'X(X'X)^{-1}X'y, \tag{3}$$

which consists of three separate elements: $X'X$, $X'y$ and its transpose, and $y'y$. The first two of these elements are already calculated iteratively in the estimation of point estimates as $\Sigma_{1\sim J}$ and $\Upsilon_{1\sim J}$. The only additional element required to calculate $\widehat{\sigma}_u^2$ is thus $y'y$, which can similarly be calculated in a cumulative manner in the same fashion as $X'X$ or $X'y$ in (2): $y'y = (y^{1\prime}y^1 + y^{2\prime}y^2 + \cdots + y^{J\prime}y^J)$. As above, we will refer to $y^{j\prime}y^j \equiv \Psi_j$, and $\Psi_{1\sim j} = \Psi_1 + \ldots + \Psi_j$. Hence, calculating $y'y$ occurs iteratively, where at each step the accumulated $\Psi_{1\sim j-1}$ is the starting point, an additional block of y of dimension $N_j \times 1$ is loaded, and the step ends with $\Psi_{1\sim j} = \Psi_{1\sim j-1} + \Psi_j$.[4]

Formally, the entire estimation process to arrive at exact OLS point estimates and standard errors is laid out in Algorithm 1. Note that given the information calculated in Algorithm 1, other standard regression statistics can be generated following estimation, including t-tests for each regression pa-

[3] This particular limit case where $N_j = 1$ and estimation occurs via OLS to generate the matrix $\Sigma_{1\sim J}$ is mentioned in Brown, Houthakker, and Prais (1953).

[4] In Appendix D we note that this result can be documented in an alternative way, where rather than accumulating matrices Σ, Υ, and Ψ at each step, the estimate $\widehat{\beta}_{1\sim j}$ is directly updated. This result is based on the matrix inverse lemma (Woodbury 1950). However, this algorithm is less efficient than the cumulative procedure described here.

rameter against arbitrary null hypotheses, global F-tests of regressions, and R^2 or adjusted R^2 measures. For example, in order to compute the R^2, we can use the residual sum of squares (RSS), $\widehat{u}'\widehat{u}$ calculated above, and additionally require the total sum of squares (TSS), given that $R^2 = 1 - \frac{RSS}{TSS}$. The TSS is simply:

$$TSS = \sum_{i=1}^{N} (y_i - \overline{y})^2 = \sum_{i=1}^{N} y_i^2 - N \left(\frac{1}{N} \sum_{i=1}^{N} y_i \right)^2 = \sum_{i=1}^{N} y_i^2 - \frac{1}{N} \left(\sum_{i=1}^{N} y_i \right)^2,$$

and both y_i^2 and y_i can be summed iteratively, with the only addition to the statistics already laid out above being the cumulative sum of y squared, and grand mean $\overline{y}$.

Algorithm 1: Cumulative ordinary least squares

Inputs: Database consisting of (y,X), block size b.
Result: Point estimate $\widehat{\beta}_{OLS}$ and variance-covariance matrix $\widehat{V}(\widehat{\beta}_{OLS})$.

1. Set $i = 1$ and $j = b$. Load into memory partition of data covering y, X in observations i to j. ;
2. Calculate Σ_1, Υ_1, and Ψ_i ;
3. If observations i to j contain end of file, set $e = 1$, otherwise, set $e = 0$;

while $e \neq 1$ **do**

4. Replace $i = i + b$ and $j = j + b$. Load into memory partition of data covering y, X in observations i to j. ;
5. Calculate Σ_j, Υ_j, and Ψ_j. ;
6. Calculate $\Sigma_{1\sim j} = \Sigma_{1\sim j-1} + \Sigma_j$, $\Upsilon_{1\sim j} = \Upsilon_{1\sim j-1} + \Upsilon_j$, and $\Psi_{1\sim j} = \Psi_{1\sim j-1} + \Psi_j$;
7. If observations i to j contain end of file, set $e = 1$. ;

end

8. Calculate

$$\widehat{\beta}_{OLS} = (\Sigma_{1\sim J})^{-1} \Upsilon_{1\sim J} \qquad \text{and} \qquad \widehat{V}\left(\widehat{\beta}_{OLS}\right) = \widehat{\sigma}^2 (\Sigma_{1\sim J})^{-1},$$

where $\widehat{\sigma}^2 = \left[\Psi_{1\sim J} - \Upsilon'_{1\sim J}(\Sigma_{1\sim J})^{-1}\Upsilon_{1\sim J}\right]/(N-K)$.

2.2 Alternative Estimators

While the previous implementation allows for the generation of exact equivalents to OLS estimates and their standard errors (and derived statistics), this cumulative procedure can be applied far more widely. Indeed, the procedure can be used for *any* estimator which can be expressed as a

sum of squares-based procedure, where the relevant database level processing requires sums over observation-level products. We document how the cumulative process works in a range of estimators below. We then document that a similar logic can be used to arrive to estimates which are based upon other techniques such as maximum likelihood.

Weighted Least Squares A simple extension to the procedure noted in section 2.1 is weighted least squares, where some diagonal weight matrix W is incorporated, such that the estimator is defined as:

$$\widehat{\beta}_{WLS} = (X'WX)^{-1}X'Wy = \left(X^{1\prime}W^1X^1 + \cdots + X^{J\prime}W^JX^J\right)^{-1}\left(X^{1\prime}W^1y^1 + \cdots + X^{J\prime}W^Jy^J\right). \quad (4)$$

Here, it follows that an identical updating procedure can be implemented to that laid out in the case of OLS, however additionally, a variable w contains the weight associated with each observation. In this case, the cumulative estimation procedure consists of holding in memory a single block of data (y_j, w_j, X_j) and generating matrix W_j, an $N_j \times N_j$ matrix with elements w_j on the principal diagonal. In the limit, if $N_j = 1$, the matrix W_j consists simply of the scalar w_j. Then, elements $X^{j\prime}W^jX^j$ and $X^{j\prime}W^jy^j$ are calculated, and summed cumulatively, before in a final step the WLS estimator is calculated by matrix inversion or similar.

Instrumental Variables and Two-Stage Least Squares Estimators Both instrumental variables (IV) and Two-Stage Least Squares (2SLS) estimators can be similarly estimated in cumulative form. To see this, note that the IV estimator in a linear model is $\widehat{\beta}_{IV} = (X'Z)^{-1}Z'y$ and the 2SLS estimator in a linear model is: $\widehat{\beta}_{2SLS} = \left(X'Z\left(Z'Z\right)^{-1}Z'X\right)^{-1}\left(X'Z\left(Z'Z\right)^{-1}Z'y\right)$, where Z refers to an $N \times L$ dimensional vector of exogenous variables, with $L \geq K$, and in the case of IV, $L = K$. Thus, both $\widehat{\beta}_{IV}$ and $\widehat{\beta}_{2SLS}$ can be generated cumulatively following a similar procedure to (1), however in the case of $\widehat{\beta}_{IV}$ $X'X$ is substituted for $X'Z$, and $X'y$ is substituted for $Z'y$. In the case of 2SLS, an additional quantity $Z'Z$ must be calculated, though identically to $X'X$, this simply requires cross-products on all variables Z within each observation i, and as in (2), $Z'Z \equiv (Z^{1\prime}Z^1 + Z^{2\prime}Z^2 + \ldots + Z^{J\prime}Z^J)$. Once again, estimation can proceed in this case in a cumulative fashion, where in each block the quantities $X^{j\prime}Z^j$, $Z^{j\prime}Z^j$ and $Z^{j\prime}y^j$ are calculated, summed cumulatively, and ultimately,

the quantity $\widehat{\beta}_{2SLS}$ is calculated by matrix inversion and multiplication, or other standard procedures such as QR decomposition or single value decomposition.

Ridge, LASSO and Elastic Net Frequently, in cases where big data is used in economic models, practitioners wish to perform some sort of regularisation. Fortunately, these cumulative procedures cross-over seamlessly to regularised models such as Ridge, LASSO and Elastic Net. Additionally, in each case, the process of accumulation is such that work with the full dataset of dimension $N \times K$ can be viewed as a first data processing step, and the selection of tuning parameters can be conducted as a second step, without ever returning to full data.

To see this, we first document the case of the Ridge regression, which, given its use of the ℓ^2 norm for shrinkage, is particularly simple expositionally. In the case of the Ridge regression, parameters are estimated as follows:

$$\widehat{\beta}_{Ridge} = \text{argmin}_{\beta} \left\{ \sum_{i=1}^{N} (y_i - X_i'\beta)^2 + \lambda \sum_{j=1}^{K} \beta_j^2 \right\},$$

where λ is a scalar tuning parameter determining the degree of shrinkage. This can be equivalently written as:

$$\widehat{\beta}_{Ridge} = (X'X + \lambda I)^{-1} X'y \tag{5}$$

where I is an identity matrix of size K. Note that following the notation above, solving for $\widehat{\beta}_{Ridge}$ requires the quantity $\Sigma_{1\sim J}$, which we have documented can be calculated in a cumulative fashion, $\Upsilon_{1\sim J}$, which we have also documented, can be calculated cumulatively; and an additional factor λI, which is independent of the number of observations. Hence, estimation in the case of Ridge is identical to that documented in OLS in section 2.1, with the only difference being that after accumulating $\Sigma_{1\sim J}$ and $\Upsilon_{1\sim J}$, but prior to solving for $\widehat{\beta}_{Ridge}$, an additional $K \times K$ matrix is added to $\Sigma_{1\sim J}$. This is laid out formally in Algorithm 2.

Similar procedures can be conducted in the case LASSO and Elastic net, where first data can be accumulated to form $\Sigma_{1\sim J}$ and $\Upsilon_{1\sim J}$ and then, conditional on having processed the data of dimension

Algorithm 2: Cumulative least squares for Ridge regression

Inputs: Database consisting of (y,X), block size b.
Result: Point estimate $\widehat{\beta}_{Ridge}$.

1. Set $i = 1$ and $j = b$. Load into memory partition of data covering y, X in observations $i$to j. ;
2. Calculate Σ_1 and Υ_1. ;
3. If observations i to j contain end of file, set $e = 1$, otherwise, set $e = 0$;

while $e \neq 1$ **do**

4. Replace $i = i + b$ and $j = j + b$. Load into memory partition of data covering y, X in observations i to j. ;
5. Calculate Σ_j and Υ_j. ;
6. Calculate $\Sigma_{1\sim j} = \Sigma_{1\sim j-1} + \Sigma_j$ and $\Upsilon_{1\sim j} = \Upsilon_{1\sim j-1} + \Upsilon_j$. ;
7. If observations i to j contain end of file, set $e = 1$. ;

end

8. Select tuning parameter λ. Then calculate

$$\widehat{\beta}_{Ridge} = (\Sigma_{1\sim J} + \lambda I)^{-1}\Upsilon_{1\sim J}.$$

$N \times K$ to a level of $K \times K$ (or $K \times 1$), and selecting a tuning parameter[5], estimates are calculated without ever returning to data at a level of $N \times K$ (or even $N_J \times K$). To see this, note that the Lasso and Elastic net equivalents of (5) are:

$$\begin{aligned}
\widehat{\beta}_{Lasso} &= \text{argmin}_\beta \left\{ \sum_{i=1}^{N} (y_i - X_i'\beta)^2 + \lambda ||\beta_j||_1 \right\}, \\
\widehat{\beta}_{ElasticNet} &= \text{argmin}_\beta \left\{ \sum_{i=1}^{N} (y_i - X_i'\beta)^2 + \lambda_1 ||\beta_j||_1 + \frac{\lambda_2}{2} ||\beta_j||_2^2 \right\},
\end{aligned}$$

where $||\cdot||_p$ refers to the ℓ^p norm, and in the case of the Elastic net, λ_1 and λ_2 refer to the strength of the Lasso and Ridge penalties respectively.

Although the lack of the exclusive ℓ^2 norm in Lasso and Ridge does not admit a simple least-squares solution as in (5), they nevertheless can both be simply resolved using cumulative procedures and a single (accumulatory) pass through N dimensional data. Specifically, this can be implemented via coordinate descent, a standard way of computing parameters in Lasso and Elastic Net (Fu 1998).

[5] We note below that our procedures can similarly be used very efficiently for k-fold cross validation; see Section 2.3.

To see this, note that for a specific parameter β_j, the coordinate descent algorithm for estimation can be written for Lasso as:

$$\widehat{\beta}_j^{\text{new}} = \text{sign}\left(\widehat{\beta}_j^{\text{old}}\right) \max\left(|z_j| - \frac{\lambda}{N}, 0\right)$$

where $\widehat{\beta}_j^{\text{old}}$ is the value of $\widehat{\beta}_j$ at the previous iteration, z_j is the j^{th} element of the vector $z = X'(y - X\widehat{\beta}^{\text{old}})$ and sign$(\cdot)$ returns the sign of the argument. Noting that z can be re-expressed as $X'y - X'X\widehat{\beta}^{\text{old}}$ makes clear that the vector of parameters β_j can be estimated by first using the cumulative procedure laid out previously, and then working with $X'y$ and $X'X$ in coordinate descent, without ever returning to the original data. A similar procedure can be used for Elastic Net given that in this case successive iterations of coordinate descent can be calculated as:

$$\widehat{\beta}_j^{\text{new}} = \frac{z_j}{1+\lambda_2}\left(\text{sign}\left(\widehat{\beta}_j^{\text{old}}\right) \max\left(|z_j| - \frac{\lambda_1}{1+\lambda_2}, 0\right)\right),$$

where z_j is the j^{th} element of $z = X'(y - X\widehat{\beta}^{\text{old}}) + \widehat{\beta}^{\text{old}} \cdot \lambda_2$, and λ_1 and λ_2 are Lasso and Ridge regularisation parameters. Again, given that z can be expressed as $X'y - X'X\widehat{\beta}^{\text{old}} + \widehat{\beta}^{\text{old}} \cdot \lambda_2$, estimation can proceed by, firstly, accumulating $\Sigma_{1\sim J}$ and $\Upsilon_{1\sim J}$ in a block-by-block or line-by-line fashion, and then implementing coordinate descent with, at most, matrices of dimension $K \times K$.

Binary Choice Models via Iteratively Reweighted Least Squares The previously defined estimators can be implemented in a single cumulative step, potentially offering substantial speed-ups compared to traditional estimators in cases where both cumulative and standard estimators are feasible, but where limits are close to met when all observations are housed in working memory (further discussion on relative performance of cumulative and naive procedures are provided in the following sections). In the case of Binary Choice Models such as probit and logit models, cumulative procedures can similarly be implemented which exactly replicate non-cumulative procedures while at the same time never housing more than a small number of observations in memory. However in these cases it is not possible to implement these estimators as single shot processes, but rather multiple passes through the N rows of data must be conducted. Thus, while these procedures provide feasible implementations of estimators when the entire dataset cannot be held in a computer's working

memory, they are unlikely to be as fast as standard procedures when memory is not a limiting factor.

Nevertheless, to see that cumulative procedures can also be implemented in non-linear models, one alternative is to use Iteratively Reweighted Least Squares (IRLS). IRLS allows for the estimation of the parameters in non-linear models in a step-wise fashion, where at each step the updated parameter estimates are based on a weighted least squares problem (Nelder and Wedderburn 1972; Green 1984). Based on this, cumulative procedures can be used to conduct least squares estimators in each iteration. Specifically, the IRLS procedure for binary outcome models consists of iteratively solving the following equation until $\widehat{\beta}^{\text{old}}$ and $\widehat{\beta}^{\text{new}}$ converge:

$$\begin{aligned} \widehat{\beta}^{\text{new}} &= (X'WX)^{-1}X'WZ \quad \text{where} \quad Z = X\widehat{\beta}^{\text{old}} + W^{-1}(y-p) \\ &= \widehat{\beta}^{\text{old}} + (X'WX)^{-1}X'(y-p) \end{aligned} \tag{6}$$

Here y is an $N \times 1$ vector of outcome variables, and p is predicted value for each unit $p_i(x_i, \widehat{\beta}^{\text{old}})$ based on the $1 \times K$ vector of individual-level realisations x_i, such that $y - p$ represents prediction residuals. W is an $N \times N$ diagonal weight matrix with diagonal elements consisting of $p_i(x_i, \widehat{\beta}^{\text{old}})(1 - p_i(x_i, \widehat{\beta}^{\text{old}}))$. In the case of probit models, for example $p(x_i, \beta) \equiv \phi(x_i\beta)$, while in the case of logit models, $p(x_i, \beta) = \ln(x_i\beta/(1-x_i\beta))$. The quantity in (6) consists of some starting value $\widehat{\beta}^{\text{old}}$ which is taken as an input (in the first iteration, $\widehat{\beta}^{\text{old}} = 0$), and a second component which can be calculated cumulatively in a block-wise fashion following (4). Thus one can estimate non-linear models where in each step a cumulative procedure is performed, and a solution is reached when the second term in (6) converges to 0.

Maximum Likelihood and other M-Estimators The use of cumulative procedures like those described above can similarly be employed to with other classes of M-estimators where estimation is based on iterative optimisation procedures, provided that observations are assumed to be independently sampled.[6] To see this, consider maximum likelihood estimation implemented using the

[6] In cases where sampling is not assumed to be independent, generalisations of this procedure could be followed, but likelihood functions, and hence blocks in the data in cumulative procedures, would need to permit this dependence. We discuss one such case where sampling is not assumed to be independent in Section 2.4 below.

Newton-Raphson method. Estimation occurs iteratively, where at each stage the Hessian and Score matrices are evaluated based on the current iteration of β. Specifically, estimation occurs as follows:

$$\widehat{\beta}^{\text{new}} = \widehat{\beta}^{\text{old}} - \left[\frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta'}\right]^{-1}_{\beta=\widehat{\beta}^{\text{old}}} \left[\frac{\partial \ell(\beta)}{\partial\beta}\right]_{\beta=\widehat{\beta}^{\text{old}}}, \tag{7}$$

with the ML solution occurring when this equation converges.

When observations are independent, the Hessian and score matrices in ML are written as summations over observations i. For example, in the case of the logit regression:

$$\frac{\partial \ell(\beta)}{\partial\beta} = \sum_{i=1}^{N}\Big[y_i F(-x_i\beta) - (1-y_i)F(x_i\beta)\Big]x_i' \qquad \frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta'} = -\sum_{i=1}^{N} f(x_i\beta)x_i'x_i,$$

where $F(\cdot)$ and $f(\cdot)$ are the logit cdf and pdf respectively.[7] This suggests a cumulative procedure can be employed where a block of arbitrary size N_j can be read into memory and the Hessian and Score matrix can be calculated for this block j based on the values $\beta = \widehat{\beta}^{old}$. The summation for each matrix can be stored, and then a subsequent block of size N_j can be read in, the Hessian and Score matrices can be calculated and added to the previous values. This process can be updated cumulatively until the end of the data is reached. Finally, a new value for $\widehat{\beta}$ can be calculated as in (7), either providing the ML estimate if convergence has occurred, otherwise data will be read again, and another iteration of (7) calculated. In this case, as noted previously with IRLS, this procedure is feasible when large databases cannot be read into memory in their entirety, but is unlikely to be as fast as a standard ML procedures if the entire data *can* be stored in memory.

[7] Similar examples can be easily provided for other common models estimated via ML. In the case of the probit regression, these functions are written as summations over i of the following form:

$$\begin{aligned}
\frac{\partial \ell(\beta)}{\partial\beta} &= \sum_{i=1}^{N}\left[y_i\frac{\phi(x_i\beta)}{\Phi(x_i\beta)} - (1-y_i)\frac{\phi(x_i\beta)}{1-\Phi(x_i\beta)}\right]x_i' \\
\frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta'} &= -\sum_{i=1}^{N}\phi(x_i\beta)\left[y_i\frac{\phi(x_i\beta)+x_i\beta\Phi(x_i\beta)}{\Phi(x_i\beta)^2} - (1-y_i)\frac{\phi(x_i\beta)-x_i\beta(1-\Phi(x_i\beta))}{[1-\Phi(x_i\beta)]^2}\right]x_i'x_i.
\end{aligned}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the normal pdf and cdf respectively.

2.3 Grouped Estimation Procedures, Fixed Effect Estimators, Heterogeneity, and Cross-Validation

In the previous section, results were shown based on arbitrary divisions of the data into mutually exclusive blocks. All of the previous results hold if rather than groups of data being based on positions, groups of data are based on some particular indicator. Consider a variable G capturing membership in some particular group, with group levels $g \in \mathcal{G}$. Using notation X_g and y_g to indicate realisations of X and y respectively for observations where $G = g$, it is well known that the OLS estimate $\widehat{\beta}_{OLS}$ can be generated over groups as:

$$\widehat{\beta}_{OLS} = \left(\sum_{g \in \mathcal{G}} X_g' X_g \right)^{-1} \left(\sum_{g \in \mathcal{G}} X_g' y_g \right) \tag{8}$$

What's more, as was the case previously, quantities $X_g' X_g$ and $X_g' y_g$ can be built up cumulatively from arbitrarily small portions of data.

In practice, this is simply a group-level generalisation of the procedure described in Algorithm 1. As in Section 2.1, consider data broken down into J row-wise partitions, with each block denoted j and consisting of N_j observations. For a particular group $g \in \mathcal{G}$ define $X_g^{j\prime} X_g^j \equiv \Sigma_j^g$, and similarly, $X_g^{j\prime} y_g^j \equiv \Upsilon_j^g$. If no observations for group g are present in block j, Σ_j^g is simply defined to be a null matrix $O_{K,K}$, and Υ_j^g a null vector $O_{K,1}$. As previously, $\Sigma_{1\sim j}^g$ refers to the summation $\Sigma_1^g + \ldots \Sigma_j^g$, and $\Upsilon_{1\sim j}^g = \Upsilon_1^g + \ldots + \Upsilon_j^g$. A group-level generalisation of Algorithm 1 is described in Algorithm 3 below.

Heterogeneity An immediate implication of this group-level cumulative procedure is that instead of generating a single $K \times K$ matrix $\Sigma_{1\sim J}$ and $K \times 1$ vector $\Upsilon_{1\sim J}$, N_G versions of these matrices will be generated, where N_G refers to the distinct number of groups. Estimation of overall OLS parameters can then occur following (8), or any other grouped level estimator can similarly be generated. However, given that group level statistics $\Sigma_{1\sim J}^g$ and $\Upsilon_{1\sim J}^g$ are also generated, identical models for any sub-samples can then be generated nearly instantaneously, without ever returning to individual

Algorithm 3: Grouped cumulative ordinary least squares

Inputs: Database consisting of (y,X,G), block size b.
Result: Point estimate $\widehat{\beta}_{OLS}$ and variance-covariance matrix $\widehat{V}(\widehat{\beta}_{OLS})$. Aggregates $\Sigma^g_{1\sim J}$ and $\Upsilon^g_{1\sim J}$ $\forall g \in \mathcal{G}$.

1. Set $i = 1$ and $j = b$. Load into memory partition of data covering y, X, G in observations i to j. ;
for $g \in \mathcal{G}^1$ **do**
  2. Calculate Σ^g_1, Υ^g_1, and Ψ^g_i ;
3. If observations i to j contain end of file, set $e = 1$, otherwise, set $e = 0$;
while $e \neq 1$ **do**
  4. Replace $i = i + b$ and $j = j + b$. Load into memory partition of data covering y, X, G in observations i to j. ;
  for $g \in \mathcal{G}^j$ **do**
    5. Calculate Σ^g_j, Υ^g_j, and Ψ^g_j. ;
    6. If $\exists\ \Sigma^g_{1\sim j-1}$ calculate $\Sigma^g_{1\sim j} = \Sigma^g_{1\sim j-1} + \Sigma^g_j$, $\Upsilon^g_{1\sim j} = \Upsilon^g_{1\sim j-1} + \Upsilon^g_j$, and $\Psi^g_{1\sim j} = \Psi^g_{1\sim j-1} + \Psi^g_j$, otherwise, initialise $\Sigma^g_{1\sim j} = \Sigma^g_j$, $\Upsilon^g_{1\sim j} = \Upsilon^g_j$, and $\Psi^g_{1\sim j} = \Psi^g_j$;
  7. If observations i to j contain end of file, set $e = 1$. ;
end

8. Calculate $\Sigma_{1\sim J} = \sum_{g\in\mathcal{G}} \Sigma^g_{1\sim J}$, $\Upsilon_{1\sim J} = \sum_{g\in\mathcal{G}} \Upsilon^g_{1\sim J}$, and $\Psi_{1\sim J} = \sum_{g\in\mathcal{G}} \Psi^g_{1\sim J}$. Then:

$$\widehat{\beta}_{OLS} = (\Sigma_{1\sim J})^{-1}\Upsilon_{1\sim J} \qquad \text{and} \qquad \widehat{V}(\widehat{\beta}_{OLS}) = \widehat{\sigma}^2\Sigma^{-1}_{1\sim J},$$

where $\widehat{\sigma}^2 = \left[\Psi_{1\sim J} - \Upsilon'_{1\sim J}(\Sigma_{1\sim J})^{-1}\Upsilon_{1\sim J}\right]/(N-K)$.

level data. This includes estimates for each specific group g, but also for aggregated groups, such as groups of states or groups of countries. In Section 2.4 we will return to show that this also offers substantial benefits for inference in cases of (blocked) bootstrap procedures.

Fixed Effect Estimators We can similarly use these group-level procedures to generate fixed-effect estimators, again without ever returning to individual level data. To see how fixed effect estimators can also be estimated in a cumulative fashion, we will now double-index as Y_{gt} an observation t within group g (this can be thought of, for example, as a case where observations are repeated within g across time periods denoted t). We are interested in estimating the parameter vector on some independent variables X_{gt}, while controlling for time-invariant group fixed effects μ_g. The fixed effect estimator can be generated from an OLS regression on within transformed data.

Specifically, this consists of estimating:

$$\begin{aligned} y_{gt} - \bar{y}_g &= (X_{gt} - \bar{X}_g)\beta_{FE} + (\mu_g - \bar{\mu}_g) + u_{gt} - \bar{u}_g \\ \dot{y}_{gt} &= \dot{X}_{gt}\beta_{FE} + \dot{u}_{gt} \end{aligned}$$

where $\dot{y}_{gt}$ denotes the within transformation of y, $\bar{y}_g$ refers to group-level means, and similarly for other variables. The term u_{gt} is a time-varying stochastic error. The fixed effect estimator is then written as below:

$$\begin{aligned} \widehat{\beta}_{FE} &= \left(\sum_{g\in\mathcal{G}}\sum_{t=1}^{T}\dot{X}'_{gt}\dot{X}_{gt}\right)^{-1}\left(\sum_{g\in\mathcal{G}}\sum_{t=1}^{T}\dot{X}_{gt}\dot{y}_{gt}\right) \\ &= \sum_{g\in\mathcal{G}}\sum_{t=1}^{T}\left(X'_{gt}X_{gt} - \bar{X}'_g\bar{X}_g\right)^{-1}\sum_{g\in\mathcal{G}}\sum_{t=1}^{T}(X'_{gt}y_{gt} - \bar{X}'_g\bar{y}_g). \end{aligned} \tag{9}$$

The key insight in (9) is that $\sum_{g\in\mathcal{G}}\sum_{t=1}^{T}(X_{gt} - \bar{X}_g)'(X_{gt} - \bar{X}_g) = \sum_{g\in\mathcal{G}}\sum_{t=1}^{T}(X'_{gt}X_{gt} - \bar{X}'_g\bar{X}_g)$. To see why this is the case, note that we can write $\bar{X}_g$ as: M_gX_g, where $M_g = I_g - 1_g(1'_g1_g)^{-1}1'_g$ is a group-specific demeaning operator, and 1_g a matrix which indicates membership to group g as a column of ones when the observation belongs to the group, and 0s otherwise. Note also that M_g is an idempotent matrix. Then, $\dot{X}'_{gt}\dot{X}_{gt} = (X_{gt} - \bar{X}_g)'(X_{gt} - \bar{X}_g) = [(I - M_g)X_{gt}]'(I - M_g)X_{gt}$, and the matrix $(I - M_g)$ is symmetric and idempotent. Thus, the preceding quantity can be written as $X'_{gt}(I - M_g)X_{gt} = X'_{gt}X_{gt} - \bar{X}'_g\bar{X}_g$ as required. Also note, that the $K \times K$ matrix $\bar{X}'_g\bar{X}_g$ can be generated from an underlying $K \times 1$ vector of group level means and the number of observations in each group. Specifically, refer to a group level vector of variable means as $\bar{x}_g \equiv (\bar{x}_{1g}\ \bar{x}_{2g}\ \cdots\ \bar{x}_{Kg})$. Then $\bar{X}'_g\bar{X}_g = N_g \times \bar{x}'_g\bar{x}_g$, where N_g is the number of observations in group g. Identical logic holds to show that $\dot{X}_{gt}\dot{y}_{gt} = (X'_{gt}y_{gt} - \bar{X}'_g\bar{y}_g)$, and $\bar{X}'_g\bar{y}_g$ can be generated from group level averages $\bar{x}_g$, a $K \times 1$ vector, and scalar $\bar{y}_g$.

Given this, implementing fixed effect models using grouped data generated in a cumulative fashion is a straightforward extension of Algorithm 3. For ease of exposition we define $\dot{X}'_g\dot{X}_g \equiv \sum_{t=1}^{T}\dot{X}'_{gt}\dot{X}_{gt}$, and $\dot{X}'_g\dot{y}_g \equiv \sum_{t=1}^{T}\dot{X}'_{gt}\dot{y}_{gt}$. From (8), elements X'_gX_g and X'_gy_g have already been

calculated cumulatively for all g. The remaining step is to calculate $\bar{X}_g'\bar{X}_g$ and $\bar{X}_g'\bar{y}_g$ which only requires group-level variable means. From this, $K \times K$ matrices $\dot{X}_g'\dot{X}_g$ and $K \times 1$ vector $\dot{X}_g'\dot{y}_g$ can be generated, and the fixed effect estimator (9) can be calculated as:

$$\widehat{\beta}_{FE} = \left(\sum_{g\in\mathcal{G}} \dot{X}_g'\dot{X}_g\right)^{-1}\left(\sum_{g\in\mathcal{G}} \dot{X}_g\dot{y}_g\right)$$

Similar cumulative procedures can be followed for two-way fixed effect models using the double within-transformation (Baltagi 2001; Wooldridge 2021). For example for balanced panels over group g and time t, two way transformations $\ddot{X}_{gt} = X_{gt} - \bar{X}_g - \bar{X}_t + \bar{X}$ and $\ddot{y}_{gt} = y_{gt} - \bar{y}_g - \bar{y}_t + \bar{y}$ can be calculated, and similar procedures followed as in the fixed effect case.[8]

Returning Fixed Effects Generally, when fixed effect models are implemented, the interest is in estimating the coefficients and standard errors on time-varying variables, and hence a fixed effect estimator like (9) is appropriate. However, in cases where estimates and standard errors on fixed effects themselves are also desired, cumulative least squares procedures offer a particularly efficient way to generate these estimates.

To see this, note that in the case of mutually exclusive fixed effects, we can write:

$$X'X = \begin{pmatrix} \sum_{i=1}^N x_{1i}x_{1i} & \cdots & \sum_{i=1}^N x_{1i}x_{Ki} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^N x_{Ki}x_{1i} & \cdots & \sum_{i=1}^N x_{Ki}x_{Ki} & 0 & \cdots & 0 \\ \bar{x}_{1,g_1} & \cdots & \bar{x}_{K,g_1} & N_{g_1} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \bar{x}_{1,g_K} & \cdots & \bar{x}_{K,g_K} & 0 & \cdots & N_{g_K} \end{pmatrix} \qquad X'y = \begin{pmatrix} \sum_{i=1}^N x_{1i}y_i \\ \vdots \\ \sum_{i=1}^N x_{Ki}y_i \\ \bar{y}_{g_1} \\ \vdots \\ \bar{y}_{g_K} \end{pmatrix},$$

[8]This result follows from the case of single demeaning. However, here both group and time fixed effects need to be removed. Noting that we can now define the double-demeaning operation as $M_{gt} = [I_{gt} - 1_g(1_g'1_g)^{-1}1_g' - 1_t(1_t'1)^{-1}1' + 1(1'1)^{-1}1']$, and hence write $\ddot{X}$ as $M_{gt}X_{gt}$, then $\ddot{X}'\ddot{X} = X_{gt}'M_{gt}'M_{gt}X_{gt}$. However, M_{gt} is idempotent, and so $\ddot{X}'\ddot{X} = (X_{gt}'X_{gt} - \bar{X}_g'\bar{X}_g - \bar{X}_g'\bar{X}_g + \bar{X}'\bar{X})$. This then suggests a simple and feasible process for concentrating out two-way (or higher order) fixed effects by grouping aggregates $\Sigma_{1\sim J}$ and $\Upsilon_{1\sim J}$ over g and t, and calculating group-specific, time-specific, and overall means, which can then be used to estimate $\widehat{\beta}_{FE}$ after processing all data.

where here we assume data is ordered such that first time-varying variables are included in X, and then group fixed effects. In this case, the resulting matrix $X'X$ simply consists of the $K \times K$ matrix $\Sigma_{1\sim J}$ in the top-left corner (where here K refers to time-varying variables, a $N_G \times K$ matrix of group means in the bottom left corner, the matrix O_{K,N_G} in the top right-hand corner, and a $N_G \times N_G$ diagonal matrix containing the number of observations in each group on the main diagonal. Similarly, $X'y$ simply consists of the vector $\Upsilon_{1\sim J}$ in positions 1 to K, and then N_G group-level means below. In this case, the only required information beyond elements already stored in standard cumulative procedures ($\Sigma_{1\sim J}$ and $\Upsilon_{1\sim J}$), are group level means and observation numbers, which can be trivially estimated cumulatively. This thus suggests that fixed effect estimators can be estimated directly and efficiently *including all fixed effects* in a sequential procedure.

Cross-Validation In Section 2.2 we noted that cumulative procedures could be used for models such as Ridge, LASSO and elastic net, where tuning parameters are chosen. Often, such tuning parameters are chosen through k-fold cross-validation (see, *eg* Wu and Wang (2020)). We showed previously that the tuning parameter λ in these models can be chosen after accumulating matrices $X'X$ and $X'y$ (see for example the case of Ridge regression in (5)). If we follow Algorithm 3, where the group variable is simply a discrete uniform random variable taking values between 1 and k, resulting matrices $X_1'X_1, \cdots, X_K'X_K$, and $X_1'y_1, \cdots, X_K'y_K$, can be used for k-fold cross validation in an efficient way.

To see this, note that cross validation consists of a procedure where for a tuning parameter λ, a specific group g is held out, and coefficients $\widehat{\beta}_{-g,Ridge}(\lambda)$ estimated using the remaining groups. Within group g, the Mean Squared Error associated with this parameter is then calculated as $MSE = \frac{1}{N_g}||y_g - X_g\widehat{\beta}_{-g,Ridge}(\lambda)||^2$. A similar procedure is then conducted for each of the N_G groups, and the MSE associated with λ is calculated as the sum of the group-specific MSEs. Note that this quantity $\frac{1}{N_g}||y_g - X_g\widehat{\beta}_{-g,Ridge}(\lambda)||^2$ can be rewritten as:

$$\begin{aligned}
&\frac{1}{N_g}\left[(y_g - X_g\widehat{\beta}_{-g,Ridge}(\lambda))'(y_g - X_g\widehat{\beta}_{-g,Ridge}(\lambda))\right] \\
&= y_g'y_g - 2\widehat{\beta}_{-g,Ridge}(\lambda)'X_g'y_g + \widehat{\beta}_{-g,Ridge}(\lambda)'X_g'X_g\widehat{\beta}_{-g,Ridge}(\lambda)
\end{aligned}$$

Each of the quantities $X_g'y_g$, $X_g'X_g$ and $y_g'y_g$ are already calculated in a cumulative fashion, implying that the MSE for a given lambda can be calculated entirely from cumulatively calculated aggregates, and MSE-optimal tuning parameters chosen as the value of λ which minimises this MSE.

2.4 Alternative Inference Procedures

In Section 2.1 we documented that inference could be conducted in a cumulative fashion in the same way as point estimates, and this required no other special procedures, apart from the accumulation of $\Psi_{1\sim J}$, which is needed to calculate the variance-covariance matrix but not parameter estimates. This can all be done in a single pass through blocks of the data. However, this relies on a homoscedasticity assumption. Here we discuss how inference can proceed in alternative settings.

2.4.1 Heteroscedasticity Robust Standard Errors

In cases where heteroscedasticity-robust standard errors are desired, the well-known heteroscedasticity-robust estimator can be implemented cumulatively. The HC1 variance estimator for OLS is written as:

$$\widehat{V}(\widehat{\beta}_{OLS})_{HC1} = \frac{N}{N-K}(X'X)^{-1}\left[\sum_{i=1}^{N}\widehat{u}_i^2 x_i'x_i\right](X'X)^{-1}$$

From Section 2.1, we already know that $X'X = \Sigma_{1\sim J}$ can be generated cumulatively. Similarly, both K and N can be read trivially from data. If $\widehat{u}_i^2$ is known, the central component $\sum_{i=1}^{N}\widehat{u}_i^2 x_i'x_i$ could be calculated cumulatively: this value, which we will refer to as Ω could be initialised as a null matrix $O_{K,K}$, and in each block of the dataset when an observation i is read in, the quantity $\widehat{u}_i^2 x_i'x_i$ calculated, and added to all previous values, as laid out in the Algorithm 4 below. As above, we will define $\Omega_j \equiv \sum_{i\in j}\widehat{u}_i^2 x_i'x_i$, and $\Omega_{1\sim j} \equiv \Omega_1 + \cdots \Omega_j$.

The issue here however is that when the data is first loaded in blocks, we cannot calculate $\widehat{u}_i = (y_i - X_i\widehat{\beta}_{OLS})$, as this requires $\widehat{\beta}_{OLS}$, which is not known until an entire pass through the data has been completed. Thus, while heteroskedasticity robust estimates can be calculated in a cumulative fashion, this requires the data be read in a cumulative fashion a second time. In particular, first Algorithm 1 should be run to calculate $\widehat{\beta}_{OLS}$, and then Algorithm 4 be run with $\widehat{\beta}_{OLS}$ as an input. However, apart from having to return to read the data, there is no particular memory restriction which

implies that this procedure will not be feasible. The only addition is a single accumulated $K \times K$ matrix $\Omega_{1\sim J}$. Similar procedures can be conducted for IV and other estimators.

Algorithm 4: Cumulative Estimation of Heteroscedasticity-Robust Variance-Covariance Matrix

Inputs: Database consisting of (y,X), block size b. Point estimate $\widehat{\beta}_{OLS}$, and $\Sigma_{1\sim J}$ from Algorithm 1.
Result: Variance-covariance matrix $\widehat{V}(\widehat{\beta}_{OLS})_{HC1}$.

1. Set $i = 1$ and $j = b$. Load into memory partition of data covering y, X in observations i to j ;
2. Calculate Ω_1 ;
3. If observations i to j contain end of file, set $e = 1$, otherwise, set $e = 0$;
while $e \neq 1$ **do**
    4. Replace $i = i + b$ and $j = j + b$. Load into memory partition of data covering y, X in observations i to j. ;
    5. Calculate Ω_j. ;
    6. Calculate $\Omega_{1\sim j} = \Omega_{1\sim j-1} + \Omega_j$. ;
    7. If observations i to j contain end of file, set $e = 1$. ;
end

8. Calculate: $\widehat{V}(\widehat{\beta}_{OLS})_{HC1} = \frac{N}{N-K}\Sigma_{1\sim J}^{-1}\Omega_{1\sim J}\Sigma_{1\sim J}^{-1}$.

2.4.2 Cluster-Robust Variance Covariance Matrix

Similarly, in the case of standard closed-form cluster-robust variance-covariance estimators, a second pass through of the data is required to calculate cumulative standard errors.[9] In this case, slightly more information must be stored, namely an additional vector of size K for each of the N_G groups over which clustering occurs, but unless both K and N_G are exceedingly large, this should not generate a problem for the feasibility of these procedures. Perhaps somewhat surprisingly, while clustered variance-covariance matrices account for dependence among observations, it is never necessary for data for an entire cluster to be housed in a computer's working memory in order to cluster standard errors by group.

To see this, note that the standard cluster-robust variance-covariance estimator is written as fol-

[9] In Section 2.4.3 we lay out an extremely efficient clustered bootstrap procedure in which it is not necessary to return to individual-level data.

lows.

$$\widehat{V}(\widehat{\beta}_{OLS})_c = \frac{N-1}{N-K}\frac{N_G}{N_G-1}(X'X)^{-1}\left[\sum_{g\in\mathcal{G}} X_g'(\widehat{u}_g\widehat{u}_g')X_g\right](X'X)^{-1}$$

As previously g refers to groups over which clustered standard errors are desired, and N_G refers to the total number of groups. As in the case of the HC1 estimator, observation, group, and covariate quantities can be easily read in a cumulative fashion from data, and $X'X$ is similarly calculated cumulatively. However, here we additionally require the quantity $\Omega_g \equiv \sum_{g\in\mathcal{G}} X_g'(\widehat{u}_g\widehat{u}_g')X_g$. For expositional clarity, note that $\sum_{g\in\mathcal{G}} X_g'(\widehat{u}_g\widehat{u}_g')X_g = \sum_{g\in\mathcal{G}}(X_g'\widehat{u}_g)(\widehat{u}_g'X_g)$. Matrix X_g' is an $K \times N_g$ matrix, while $\widehat{u}_g$ is an $N_g \times 1$ vector of regression residuals. Thus, the matrix $X_g'\widehat{u}_g$ is a $K \times 1$ vector, while its transpose $\widehat{u}_g'X_g$ is $1 \times K$. Additionally, note that $X_g'\widehat{u}_g$ is generated by multiplying the observations *of each observation* with its own residual, and so can be generated cumulatively. Thus, as previously, if data is arbitrarily divided into J blocks, the quantity $X_g'\widehat{u}_g = X_g^{1\prime}\widehat{u}_g^1 + \ldots + X_g^{J\prime}\widehat{u}_g^J$ can be generated cumulatively by first calculating $X_g^{j\prime}\widehat{u}_g^j$ for each group present within each block j, then summing over all j, and finally using this quantity to calculate the overall quantity Ω_g.[10] This procedure is laid out formally in Algorithm 5 below, where as before, $X_g'\widehat{u}_{g,1\sim j}$ refers to the summation of $X_g^{1\prime}\widehat{u}_g^1 + \cdots + X_g^{j\prime}\widehat{u}_g^j$.

2.4.3 An Efficient Bootstrap Algorithm for Clustering

While the clustered procedure described in the previous sub-section is feasible and permits for the exact calculation of analytic cluster-robust variance covariance matrices, it requires opening the data two times: the first to calculate the parameter estimates, and the second to calculate the standard errors which requires residuals $\widehat{u}_g$. However, given the results from Section 2.3, if one wishes to generate a clustered standard error by bootstrapping, this can be done in a single pass through the data, and additionally bootstrap replicates can be conducted extremely quickly, and indeed orders of magnitude more quickly than in standard clustered bootstrap procedures. To see why, note that the parameter estimate of interest can be generated as in (8). Also note that from Algorithm 3, that cumulative procedures are used to generate $K \times K$ matrices for each group $\Sigma_{1\sim J}^g$, as well as

[10] It is important to note that this procedure requires working with the $K \times 1$ vector $X_g^{j\prime}\widehat{u}_g^j$ at each step. It is *not* possible to calculate Ω_g at each step, but rather, we must accumulate $X_g^{j\prime}\widehat{u}_g^j$ and only then calculate Ω_g.

Algorithm 5: Cumulative Estimation of Cluster-Robust Variance-Covariance Matrix

Inputs: Database consisting of (X, G), block size b. Point estimate $\widehat{\beta}_{OLS}$, and $\Sigma^g_{1\sim J}$ from Algorithm 3.
Result: Variance-covariance matrix $\widehat{V}(\widehat{\beta}_{OLS})_C$.

1. Set $i = 1$ and $j = b$. Load into memory partition of data covering G, X in observations i to j ;
for $g \in \mathcal{G}^1$ **do**
  2. Calculate $X^{1\prime}_g \widehat{u}^1_g$;
3. If observations i to j contain end of file, set $e = 1$, otherwise, set $e = 0$;
while $e \neq 1$ **do**
  4. Replace $i = i + b$ and $j = j + b$. Load into memory partition of data covering X, G in observations i to j. ;
  for $g \in \mathcal{G}^j$ **do**
    5. Calculate $X^{j\prime}_g \widehat{u}^j_g$. ;
    6. If $\exists\ X'_g \widehat{u}_{g,1\sim j-1}$ calculate $X'_g \widehat{u}_{g,1\sim j} = X'_g \widehat{u}_{g,1\sim j-1} + X^{j\prime}_g \widehat{u}^j_g$, otherwise initialise $X'_g \widehat{u}_{g,1\sim j} = X^{j\prime}_g \widehat{u}^j_g$. ;
  7. If observations i to j contain end of file, set $e = 1$. ;
end

8. Calculate

$$\widehat{V}(\widehat{\beta}_{OLS})_c = \frac{N-1}{N-K}\frac{N_G}{N_G-1}(\Sigma_{1\sim J})^{-1}\left[\sum_{g\in\mathcal{G}}(X'_g\widehat{u}_{g,1\sim J})(X'_g\widehat{u}_{g,1\sim j})'\right](\Sigma_{1\sim J})^{-1}$$

.

group-specific $K \times 1$ vectors $\Upsilon^g_{1\sim J}$.

This implies that we can generate resampled versions of (8) by simply resampling with replacement N_G pairs of matrices $\Sigma^g_{1\sim J}$, $\Upsilon^g_{1\sim J}$, and calculating a resampled estimator $\widehat{\beta}^{b*}$ as follows:

$$\widehat{\beta}^* = \left(\sum_{g^*\in\mathcal{G}^*}\Sigma^{g*}_{1\sim J}\right)^{-1}\left(\sum_{g^*\in\mathcal{G}^*}\Upsilon^{g*}_{1\sim J}\right), \tag{10}$$

where $\Sigma^{g*}_{1\sim J}$ refers to resampled matrix $\Sigma^g_{1\sim J}$, and similarly for $\Upsilon^g_{1\sim J}$. When clusters are large, such as individuals within states of countries, resampling aggregated matrices to form bootstrap resamples $\widehat{\beta}^*$ will be orders of magnitudes faster than resampling clusters of data. This suggests a potentially substantially faster bootstrap estimate for the cluster robust variance for the parameter vector $\widehat{\beta}$. This

consists of generating a large number B of resampled estimates (10), which can be used to calculate the bootstrap CRVE for $\widehat{\beta}$ as: $V(\widehat{\beta})_{CRVE} = V(\widehat{\beta}^*) = \frac{1}{B}\sum_{b=1}^{B}(\widehat{\beta}_b^* - E[\widehat{\beta}_b^*])^2$.

3 Optimal Implementation

Whether implementing cumulative or standard algorithms, identical calculations are required to be made, given that cross products are required between each element of X and between X and y for each observation i. Indeed, cumulative algorithms require *strictly more* calculations than standard algorithms. To see this, consider the case of OLS. To calculate coefficients in OLS, X' must be multiplied with X, implying computational time of order $\mathcal{O}(NK^2)$. Additionally, X' must be multiplied with y, implying computational time of order $\mathcal{O}(NK)$. Finally, resolving the linear system $X'X\beta = X'y$ involves time $\mathcal{O}(K^3)$ via Gauss-Jordan elimination. In the case of cumulative algorithms, identical procedures are required, and additionally, at each step two $K \times K$ matrices Σ_j and $\Sigma_{1\sim j-1}$ must be summed, which is of computational time $\mathcal{O}(K^2)$, and similarly, two $K \times 1$ vectors Σ_j and $\Sigma_{1\sim j-1}$ must be summed, involving time $\mathcal{O}(K)$. In general, $N >> K$, implying that $\mathcal{O}(NK^2)$ will dominate in both cases.

Nevertheless, if all computational procedures scale linearly in the number of observations, no gains will be made by implementing cumulative routines in place of their standard counterparts. However, computational routines clearly do not scale linearly with sample size indefinitely. To see this, it is sufficient to consider two cases: one where N is such that observations can be housed in a computer's working memory, and another where N exceeds the capacity of a computer's memory. In the prior case, the calculation time will be finite, while in the latter case calculation will be impossible, and hence time will be infinite. In this section we will discuss the optimal implementation where optimality refers to the block size which minimises calculation time. Given that in the limit cumulative procedures simply revert to standard OLS estimation if a block size of N is chosen, we consider only the optimal choice of block size for cumulative procedures. We return to these issues empirically in Section 4.

As above, the entire cumulative algorithm for OLS requires a number of well-defined steps. In total, matrix multiplication between X' and X is $\mathcal{O}(NK^2)$, and between X' and Y is $\mathcal{O}(NK)$.

Final resolution of the parameters is $\mathcal{O}(K^3)$. Additionally, within each block j a series of element-by-element summations must occur to accumulate $\Sigma_{1\sim j}$ and $\Upsilon_{1\sim j}$. In each step these are are of order $\mathcal{O}(K^2)$ and $\mathcal{O}(K)$ respectively. Given that there are J such blocks, and in the first block it is not necessary to accumulate $\Sigma_{1\sim 1}$ and $\Upsilon_{1\sim 1}$, these calculations are of computational time $\mathcal{O}(K^2(J-1))$ and $\mathcal{O}(K(J-1))$. Thus, total computational time of the algorithm is of the order:

$$\mathcal{O}(NK^2) + \mathcal{O}(NK) + \mathcal{O}(K^3) + \mathcal{O}(K^2(J-1)) + \mathcal{O}(K(J-1)). \tag{11}$$

Here it is clear that if a single block is chosen, and hence $J = 1$, then $\mathcal{O}(K^2(J-1))+\mathcal{O}(K(J-1)) = 0$ and the cumulative algorithm collapses to OLS.

To consider the optimal block size, we will consider separately three elements of (11). A first element, corresponding to the first two terms in (11) and denoted $L(N, K)$ is the procedure of loading data and multiplying matrices required to arrive to $X'X$ and $X'y$. We write this function as $L(N, K) = l(NK^2 + NK)$. A second element, corresponding to the third term in (11), consists of generating estimates $\widehat{\beta}$ once provided with $X'X$ and $X'y$, and is written as $S(K) = s(K^3)$. And finally, an accumulation procedure, denoted $C(J, K) = c(K^2(J-1) + K(J-1))$, consisting of the final two terms where matrices are summed in a cumulative fashion.

Note that given that $N = JN_j$, the first term can be re-expressed as s $L(J, K) = l(JN_jK^2 + JN_jK)$. For the sake of simplicity, given that N_j is determined by J, below we omit N_j terms as implicit in $l(\cdot)$. For a given K, The total time to compute the cumulative least squares algorithm can thus be written as:

$$T_c(J; K) = L(J, K) + S(K) + C(J, K).$$

Hence, the optimal number of partitions of data J should solve the problem:

$$\min_J \Big(L(J, K) + S(K) + C(J, K) \Big) \text{ subject to } 0 < J \leq N.$$

For an interior solution, this suggests that optimal number of blocks considered should satisfy the

following first order condition:

$$\begin{aligned} &\frac{\partial L(J,K)}{\partial J} + \frac{\partial C(J,K)}{\partial J} = 0 \\ \Rightarrow &\frac{\partial L(J,K)}{\partial J} = -\frac{\partial C(J,K)}{\partial J} \end{aligned} \tag{12}$$

Note that here given that regardless of the block size chosen, the same final matrix inversion is required, for a given K optimality does not depend on $S(\cdot)$, as reflected in (12). This suggests the logical conclusion that an optimal block size should be chosen which equates the marginal cost of summing an additional set of matrices across blocks, $\partial C(J,K)/\partial J$, with the marginal benefit coming from loading smaller partitions of the data into memory to calculate $X'X$ and $X'Y$.

Understanding the optimal block size for conducting cumulative least squares thus requires understanding the nature of functions $C(J,K)$ and $L(J,K)$. The precise nature of these two functions is likely highly dependent upon a particular computational environment (both software and hardware), nevertheless, we can suggest a number of key conjectures. Firstly, it is clear that for a given K, $C(J,K)$ will, abstracting from other elements, be linear in J. To see this, note that for each additional block, we simply require the summation of an additional identically sized $K \times K$ and $K \times 1$ matrix. Thus, moving from j to $j+1$ requires adding one set of summations, while moving from $j+1$ to $j+2$ requires adding an identical set of summations, and so calculation time will scale linearly in block sizes. Secondly, for a given K, $L(J,K)$ seems unlikely to be linear in J. Rather, this value is highly dependent on a particular computational environment. Note that in general, when a computer's RAM usage is high a number of internal processes such as paging occur such that loading data into memory becomes increasingly slow as the size of a database increases. Thus, when a sample approaches the limit of a computer's RAM, the marginal benefit of increasing the number of blocks of data is high given that it avoids substantial slowdowns inherent in computational architecture. However, if a computer's RAM usage is low, the marginal benefit of increasing the number of blocks approaches zero, given that no such slowdown in data loading occurs, and the total computation time $L(J,K)$ is independent of J. Thus, at very high values of J, for example where J approaches the total number of observations, the marginal benefit of increasing $L(J,K)$ is

likely essentially zero given that no memory slowdown occurs owing to the storage of large amounts of data in memory. However, at high low values of J, if data is large enough to result in memory slowdowns, the benefit of increasing J is substantial. On the other hand, the costs of increasing J, $C(J, K)$ are constant in J.

This suggests a number of general results. Firstly, if one is working with large datasets and memory is not unlimited, it is likely the case that smaller blocks of data should be preferred given that memory slowdowns can be avoided. If data does not fit in memory, this argument holds with certainty, given that $\partial L(J, K)/\partial J|_{J_{min}} = \infty$, where J_{min} refers to the point at which it becomes feasible to hold data in memory. However, if RAM limits are binding with N, the optimal solution is likely not to increase the number of blocks to the maximum theoretical limit ($J = N$), given that at low block sizes no memory slowdown will be observed, but a constant cost increase is observed in terms of sums across blocks. What's more, these results suggest that there is no gain from varying the block size across the sample, but rather that a single value of N_J should be chosen as that which satisfies (12). Finally, as the number of covariates increases, it seems likely that fewer blocks should be preferred, given that the cost of adding marginal blocks increases in K. Precise optima will vary across computers and configurations, and are thus specific to particular contexts. In the following section we will document specific examples which point to a Goldilocks principle of choosing blocks neither too big nor too small, and, fortunately, suggest that computation time is quite flat over a large range of blocks, provided that extreme situations are not encountered.

4 Illustrations

In this section we document two examples to illustrate the performance of cumulative procedures in practice. A first example is based on simulated data where we maintain fixed computational resources and vary key parameters of the data (namely the number of observations and the number of variables). And a second example is based on real data, where we document the performance of cumulative versus standard estimation procedures in a range of computational environments and with various methods of estimation.

4.1 Simulated Data

To demonstrate the relative performance of the cumulative algorithm compared with a standard regression implementation, we test the time to complete calculations under controlled conditions. Specifically, we compare the time it takes to run an OLS regression using cumulative and standard estimation routines based on the same data. We conduct these tests on a server with 1GB of dedicated RAM and no outside processes running to ensure comparability across estimation times.[11] We consider a range of observation numbers and independent variables, and, in the case of the cumulative algorithm, also document times under a range of block sizes. In each case, the time completed consists of identical procedures: namely, in the case of the cumulative algorithm it is the time to import all blocks of data, calculate the necessary block-specific quantities, and finally return the regression estimates, standard errors and R-squared. And in the case of a 'standard' regression implementation, the time simply refers to the time to open the data from the disk and estimate the OLS regression using canned software. Both routines automatically remove variables which are mutlicolinear.

The test procedure thus consists of generation of data of the following general form:

$$y = X\beta + u,$$

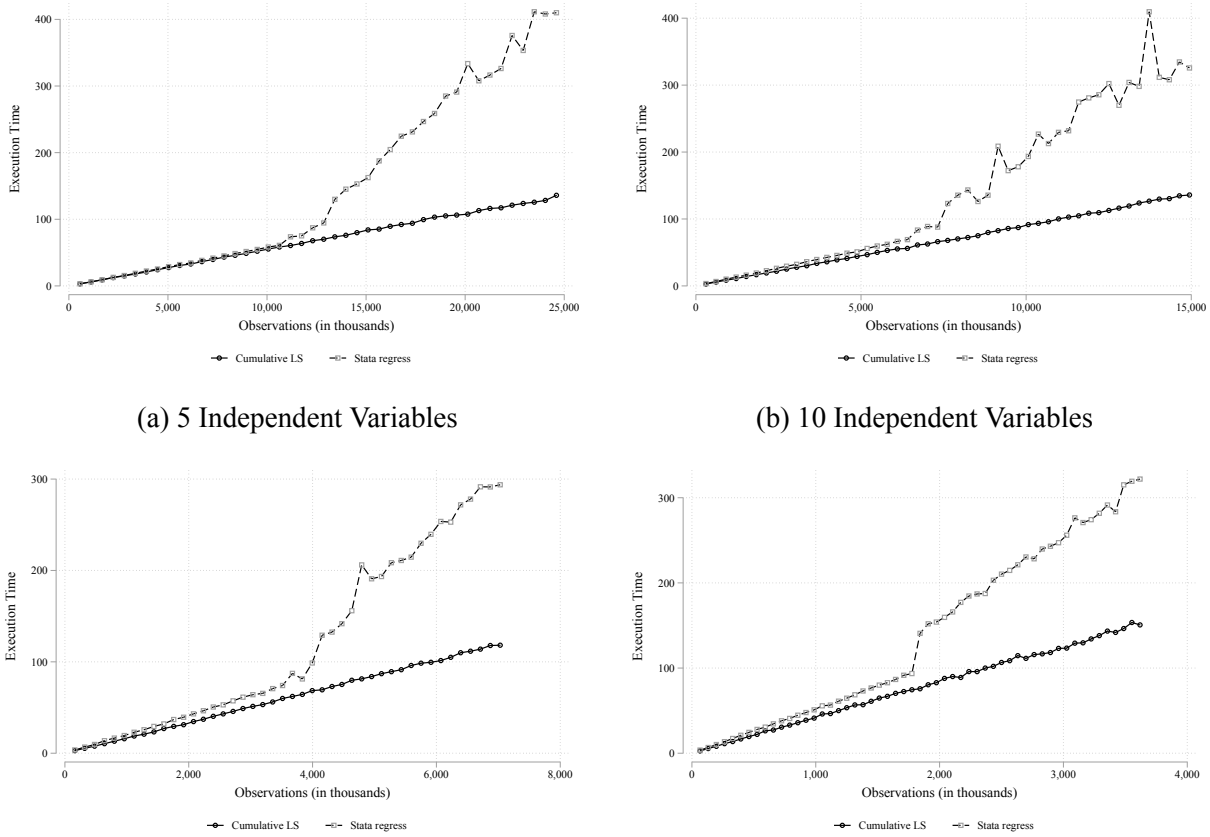
where X is an $N \times K$ matrix of simulated data consisting of a constant and $K - 1$ uniformly distributed variables, $u \sim \mathcal{N}(0, 3)$ is a simulated $N \times 1$ error term, and β is a $K \times 1$ vector of parameters. Here we consider processing times varying K, N and the block size, N_j, where in each case X and y are treated as inputs, u as unobservable, and β as a vector of parameters to estimate.

Tests are conducted using a recent version of Stata (specifically, Stata version 16), where the cumulative algorithm is written principally in Stata's matrix language Mata. Regression is conducted using Stata's native "regress" command. Initially, a single core version of Stata is used (Stata SE), however relative performance is shown to follow qualitatively similar patterns when a multiple processor version of Stata is used (Stata MP). In Section 4.2 we consider a range of alternative estimation

[11] This is a commercially available Virtual Private Server with a 4 core CPU running a Linux-based operating system. All data is stored on the server on a solid state drive.

procedures and models.

Figure 1: Sample size and execution time of commercial routine versus cumulative method



(a) 5 Independent Variables

(b) 10 Independent Variables

(c) 20 Independent Variables

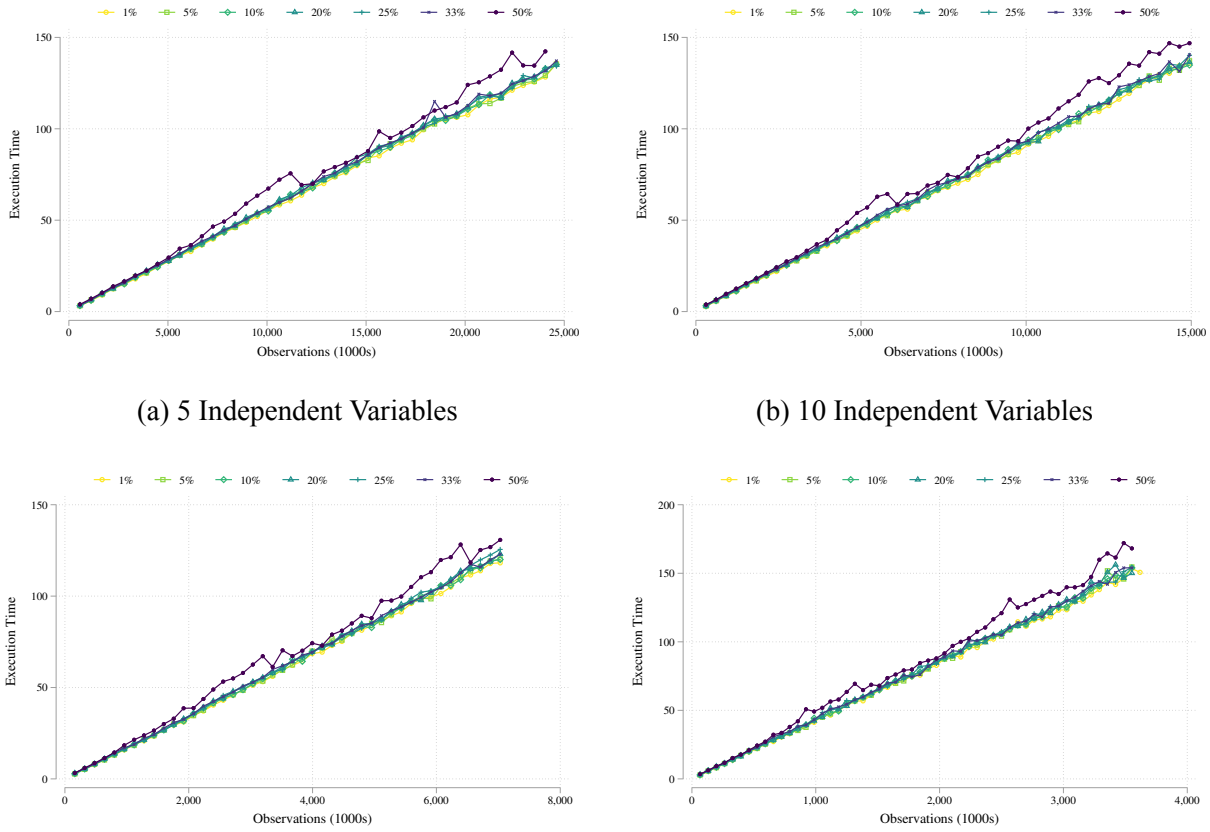
(d) 50 Independent Variables

Notes: All times refer to the computation time of reading data into memory and estimating an ordinary least squares regression. Tests are all conducted on a system with 1GB of RAM, with no other processes running. In each case, tests are conducted up to the point at which there is insufficient RAM to open the data, this precluding the estimation of standard regression models. Beyond this point, it is still feasible to estimate parameters using Cumulative Least Squares.

Processing times for estimation of cumulative algorithms versus standard regression software are documented in Figure 1. Each panel presents processing times for a particular number of simulated independent variables, ranging from 5 (panel (a)), to 50 (panel (d)). Processing time in seconds is documented on the vertical axis of each plot, and the total number of observations in thousands is documented on the horizontal axis. Times for standard regression software are presented as hollow squares with dashed lines, while times for cumulative algorithms are presented as hollow circles connected by a solid black line. Each point refers to a specific simulated dataset and the time it takes to estimate parameters, standard errors, and other regression statistics with this data. In this Figure,

in each case where cumulative algorithms are used, the block size is arbitrarily chosen to contain 10% of the total number of observations.

Figure 2: Execution Time of Cumulative Least Squares by Block Size



(a) 5 Independent Variables

(b) 10 Independent Variables

(c) 20 Independent Variables

(d) 50 Independent Variables

Notes: All times refer to the computation time of reading data into memory and estimating a cumulative least squares regression. Tests are all conducted on a system with 1GB of RAM, with no other processes running. Block sizes as a proportion of the total observation numbers are indicated in the figure legend.

Across all panels we observe, unsurprisingly, that as the total number of observations grows for a fixed K, processing time increases. For cumulative algorithms, this processing time increases approximately linearly. For example, in the case where K=5, regressions with 5, 10, 15 and 20 million observations take approximately 30, 60, 90 and 120 seconds to run. This is observed in all panels. Similar linear behaviour is observed in standard regression software when the observation numbers are moderate compared to the total RAM available. What's more, where N is relatively

small, the processing times of standard software and cumulative algorithms are similar.[12] However, the linear relationship breaks down and processing times for standard regression estimation become considerably slower from the time that the total number of observations approaches around 50% of the memory capacity of the computer.[13] This implies that at relatively small numbers of observations compared to a computer's available memory, the processing time of cumulative procedures is similar to that of standard non-cumulative procedures, however cumulative procedures then rapidly become 2 to 3 times fasted than non-cumulative counterparts. At some point, when the number of observations grows beyond the capacity of the RAM, non-cumulative procedures become infeasible to estimate, while the processing time of cumulative procedures continue to scale linearly indefinitely. If similar tests are run using multiple processor versions of software, similar patterns are observed (Appendix Figure A1).

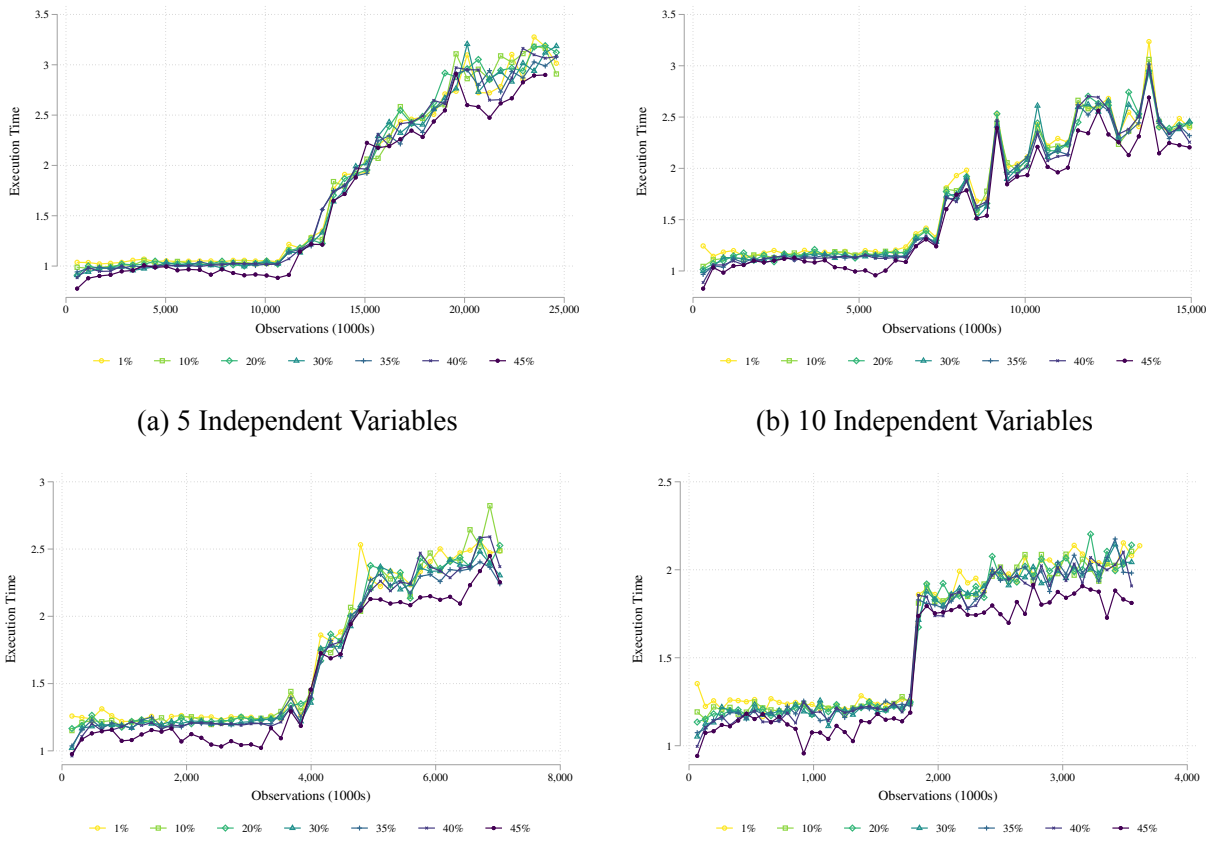
Results from Figure 1 are based on a block size N_j which is arbitrarily chosen as $N_j = N/10$. If data is very large, blocks of this size will also imply that individual blocks of data cannot fit in memory. In Figure 2 we document processing times of cumulative least squares procedures where the block size is varied from 1% of data up to 50% of data (using sizes of greater than 50% of data is not sensible, as one block will be larger than the other).[14] Once again, we document times across a range of values for K (panels), and N (horizontal axes). Each point refers to the time for a single regression. We observe that across all cases examined, in general smaller block sizes are marginally faster.

[12] As documented in Section 3, cumulative algorithms will actually conduct very slightly more calculations given the final combination of matrices to accumulate over Σ, Υ and Ψ. Thus, if coding implementations were exactly equal, and memory access was costless we would expect that at small values of N cumulative algorithms would be very slightly slower than OLS regression. In fact, we see that there are marginal differences in the other direction, with cumulative procedures being very slightly faster than standard regression at low N. This likely owes to non-estimation based procedures included in standard software such as syntax processing, or differences in the way data files are loaded from disk. Nevertheless, the fact that such variations are small suggests that differences in standard and cumulative implementations observed where N is large do not, in general, owe to code efficiency, but rather differences in the procedure itself.

[13] In principle, a computer with 1GB of RAM contains $1 \times 10^9 \times \frac{1024^3}{1000^3}$ bytes of memory. A given line of data in our simulations consists of K double precision variables, which each occupy 8 bytes of memory ($K-1$ independent variables and the dependent variable). Thus, one can calculate the theoretical maximum number of observations which could be held in memory as $\frac{1 \times 10^9}{K \times 8} \times \frac{1024^3}{1000^3}$. In Figure 1 we observe that the kink in processing times for Stata's regress command occurs at around 12,000,000 observations, or around 44% of the computer's theoretical maximum observations.

[14] These are essentially profiles of a surface where the block size is varied continuously. The entire surface is plotted in Appendix Figure A2.

Figure 3: Relative Time of Cumulative Least Squares Versus Standard Implementation by Block Size



(a) 5 Independent Variables

(b) 10 Independent Variables

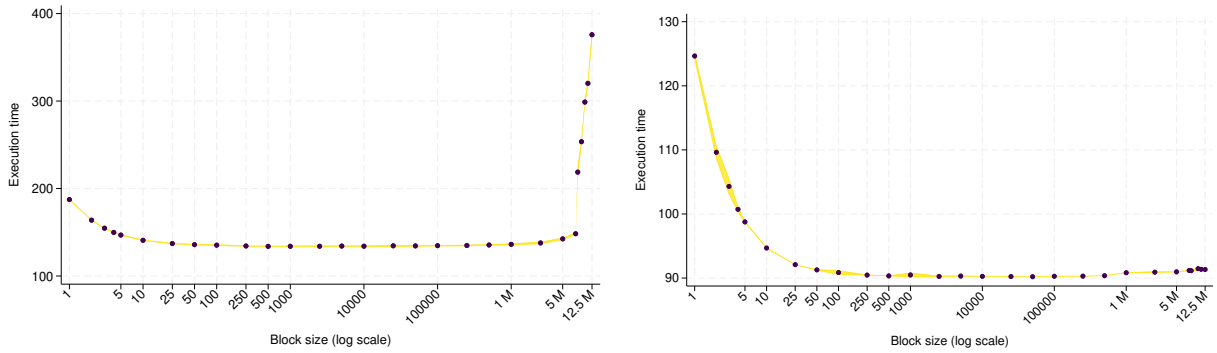
(c) 20 Independent Variables

(d) 50 Independent Variables

Notes: Each point presents the ratio of computation times of Stata's native regression command to cumulative least squares. Values less than 1 imply non-cumulative procedures are faster than cumulative procedures, and vice versa for values greater than 1. All times refer to the computation time of reading data into memory and estimating a cumulative least squares regression. Tests are all conducted on a system with 1GB of RAM, with no other processes running. Block sizes as a proportion of the total observation numbers are indicated in the figure legend.

Figure 3 documents the ratio of computation times from Stata's native regress command compared to cumulative least squares, where cumulative least squares is implemented with the same range of block sizes displayed in Figure 2. Values of less than 1 imply that standard (non-cumulative) estimation procedures are faster than cumulative procedures, while values greater than 1 imply that cumulative procedures are faster than non-cumulative procedures. In line with the substantial increase in non-cumulative procedures documented in Figure 1, we observe a sharp improvement in the ratio at around 50% of the theoretical maximum memory. In this particular implementation, when the smallest block size is used (1% of N), the ratio is consistently greater than 1.

Figure 4: Optimal Block Size and Available Memory



(a) Memory Limits Binding

(b) No Binding Memory Limits

Notes: Total time taken to estimate a regression with 5 independent variables and 25 million observations is documented. Varying block sizes are used, as plotted on the horizontal axes, and total time is plotted on the vertical axis. The left-hand panel is estimated on a server with 1GB of RAM, 4 cores, and an Intel i7 processor. The right-hand panel is estimated on a PC with 32GB of RAM, 8 cores and Intel i7 processors. In each case, estimates are generated 50 times for each block size and average times are plotted as circles. The 95% confidence interval of these estimation times are plotted in yellow.

These results may suggest that the optimal procedure is thus to choose a block size as small as possible. All results in this paper hold for block sizes as small as $N_j = 1$, and even in cases where $N_j = N/100$, the block size considered exceeds 1. In Figure 4 we consider execution times for a particular simulated dataset (N=25,000,000, $K = 5$), however here allowing block sizes to fall to their smallest possible value. A logarithmic scale is used on the horizontal scale allowing black sizes to vary from 1 to 12.5 million observations (50% of the total observations). Panel (a) uses an identical 1GB server as that used in tests above, while panel (b) documents the same times on a computer where memory limits do not bind. In this case we observe that the optimal block size *is not* the smallest possible size ($N = 1$), but rather follows the Goldilocks principle laid out in Section 3. Clearly, block sizes that are so large that memory constraints begin to bind with N_j observations should be avoided, however, a very small block size is also sub-optimal, given that this requires the accumulation of many $X^{j\prime}X^j$ and $X^{j\prime}y^j$ matrices. Where memory limits do not bind sharply (panel (b)) one may wish to work with slightly larger block sizes to avoid sub-optimal behaviour observed with very small block sizes, as provided extreme regions are avoided, the practical choice of block appears to be of second order importance.

4.2 An Empirical Example

We document the performance of cumulative algorithms and their non-cumulative counterparts on a real empirical example. This empirical example is based on a large sample of microdata, following Aaronson et al. (2020). Aaronson et al. estimate the impact of fertility on mother's labour supply using data over 2 centuries from censuses and demographic surveys. We follow Aaronson et al. (2020) in downloading data from IPUMS and the Demographic and Health Surveys resulting in 51,449,770 observations covering 106 countries, with observations drawn from 434 country by year cells. Data covers years 1787-2015, and measures women's labour force participation, total fertility, and a number of other mother-level covariates. In Appendix C we provide summary statistics as well as a graph documenting the years covered in data and a graph documenting the countries covered and the number of observations in each (Figures A3 and A4).

This example is well-suited to our setting because it allows us to document the relative performance of a number of different estimation and inference procedures. Specifically, two models are considered, and these are estimated in a number of ways. A first model is simple (weighted) ordinary least squares, where each woman's labour for participation measure is regressed on her total fertility. We estimate:

$$\text{Participation}_{ict} = \beta_0 + \beta_1 \text{Fertility}_{ict} + X'_{ict}\beta + \phi_{c\times t} + \varepsilon_{ict}, \tag{13}$$

for individual i in country c observed in year t, where country by year fixed effects are indicated as ϕ, and covariates X_{ict} are those indicated by Aaronson et al.; namely each women's age, age at first birth, and first born child's sex. Second, we estimate an IV model, where in the first stage, a measure of fertility (specifically whether a woman has a third child) is regressed on an indicator of a woman having second birth twins, and then in a second stage labour force participation is regressed on instrumented fertility:

$$\begin{aligned} \text{Fertility 3}_{ict} &= \pi_0 + \pi_1 \text{Twin 2}_{ict} + X'_{ict}\Pi + \phi_{c\times t} + \nu_{ict} \\ \text{Participation}_{ict} &= \gamma_0 + \gamma_1 \widehat{\text{Fertility 3}}_{ict} + X'_{ict}\Gamma + \phi_{c\times t} + \eta_{ict}. \end{aligned} \tag{14}$$

All other details follow those laid out in (13), and replicate models proposed by Aaronson et al. (2020).[15] This IV strategy follows a long tradition, starting with Rosenzweig and Wolpin (1980), of seeking to draw conditionally exogenous variation in fertility owing twin births (see Bhalotra and Clarke (2023) for a recent overview).

In this context, we are interested in documenting the processing times of IV and OLS estimation of coefficients and standard errors with a number of specific estimation procedures. This includes IV and OLS models where fixed effects are directly estimated as well as estimation by fixed effects estimators where fixed effects are concentrated out resulting in estimates only of coefficients on time-varying variables. We also consider a number of alternative inference procedures; namely, firstly assuming homoscedasticity, then clustering standard errors by country$\times$year, both analytically, and with a clustered bootstrap. We report the processing time of cumulative algorithms written by us to implement the procedures we lay out above compared to commercially produced (non-cumulative) algorithms written in Stata (version 18). We also consider alternative non-commercial (non-cumulative) algorithms which implement potentially more efficient fixed effect procedures, namely a more rapid implementation of the within transformation described by Gaure (2013), Guimarães and Portugal (2010), Correia (2015), implemented by Correia (2015). All processing times are measured in minutes and include the time of reading data, estimating the regression and producing output, and are estimated under controlled conditions on a a server with fixed characteristics which are varied across tests. Each procedure is estimated 10 times, with average processing times reported.

Results are displayed in Table 1. Each cell provides the average processing time for a particular estimation procedure in a particular computational environment. Estimation procedures are listed in rows, and computational resources are listed in columns. In Panel A we document times corresponding to OLS (13), and in Panel B we document times corresponding to IV (14). Within panels, we first present processing times for cumulative algorithms, and below this, standard regression or IV regression implementations. We replicate each process for a range of computational systems. These are all commercially available dedicated private servers with virtual memory limits (RAM)

[15] All results are replicated exactly.

listed in column headers. Although each column principally changes the quantity of RAM available on the server, a number of other more minor changes may occur on the server when changing configurations. For this reason, in Panel C we provide a system benchmark which shows the server's performance on a standard numerical test.[16]

Considering the behaviour of cumulative algorithms in OLS, we see that irrespective of the computer's memory, the processing time is very similar, ranging from an average of 4.0 to 4.2 minutes when estimation is conducted by within-transformed variables as in (9). This stability across systems is precisely the value of cumulative regression procedures. Here, whether one has available a system with substantial memory (32 GB) or very little memory (1 GB), there is no change in performance. The case of standard regression is of course different. In systems with small memory capacity it is simply infeasible to load data and estimate parameters. These cells are shaded in gray. Initially, when loading data into a system with 2 GB of memory is viable, the standard implementation proves markedly slower than its cumulative counterpart (13.67 minutes compared to 4.1 minutes). Yet, this instance is particularly noteworthy, revealing an overflow in the standard procedure at the threshold of memory usage limits. Moving to larger memory capacities reduces the time of processing of within transformed models slightly (to around 8.3 to 8.6 minutes), but given the efficiency with which cumulative routines can conduct fixed effect procedures (Section 2.3), standard implementations do not approach the performance of cumulative algorithms. Alternative routines for dealing with fixed effects are observed to be slightly faster when they are feasible to be estimated, as observed when within-transformed models from cumulative algorithms are compared with hdfe implementations. However, such models are infeasible in a range of cases where lower memory limits are binding, and similarly cannot return full parameter vectors.

When we wish to report full parameter vectors ("Full Estimation"), cumulative algorithms outperform comparison estimators. Across system types, cumulative procedures require between 3.9 to 4.2 minutes for estimation, while similar procedures in non-cumulative models require around 10 minutes. It is noteworthy that in the case of cumulative algorithms, full estimation including fixed

[16] Specifically, the system benchmark consists of counting the number of times that the computer is capable of calculating all the primes up to 10,000 in a 10 second span. In this case, higher values of the system benchmark imply that the computer is faster.

Table 1: Execution Time Across Models in a Labour Market Participation Example: OLS and IV

	Memory Constraints				
	1 GB	2 GB	8 GB	16 GB	32 GB
Panel A: OLS					
Cumulative Procedures					
Within-transformation	4.060	4.135	4.111	4.236	4.108
Within, Clustered	7.979	8.053	7.994	8.174	7.929
Within, Clustered Bootstrap	4.086	4.133	4.151	4.209	4.084
Clustered Bootstrap (s)	3.684	3.745	3.798	3.815	3.715
Full Estimation	4.016	4.010	4.046	4.109	3.951
Standard Procedures					
Within-transformation (areg)	–	13.677	8.669	8.290	8.497
Within, Clustered	–	–	12.034	11.029	11.333
Full Estimation	–	16.474	10.147	10.103	10.054
Within-transformation (hdfe)	–	–	–	3.605	3.476
Within, Clustered (hdfe)	–	–	–	3.794	3.554
Panel B: IV					
Cumulative Procedures					
Within-transformation	4.600	4.608	4.609	4.698	4.589
Within, Clustered	8.575	8.653	8.724	8.825	8.636
Within, Clustered Bootstrap	4.428	4.449	4.429	4.539	4.383
Within, Clustered Bootstrap (s)	4.267	4.293	4.278	4.376	4.271
Full Estimation	4.362	4.379	4.350	4.448	4.329
Standard Procedures					
Within-transformation (xtivreg)	–	–	–	–	6.629
Within, Clustered	–	–	–	–	15.958
Full Estimation	–	–	–	–	226.181
Within-transformation (hdfe)	–	–	–	–	5.996
Within, Clustered (hdfe)	–	–	–	–	8.597
Panel C: System Benchmark	768.1	764.9	769.9	772.7	778.5

Notes: Each cell reports average processing time in minutes of a particular estimation procedure based on the memory constraints listed in column headers. All averages are taken over 10 estimations. Cells are coloured gray when estimation cannot occur due to memory limits. All methods are estimated in Stata 18 SE. For clustered bootstrap errors, (s) stands for "sorted", implying that the database is sorted by clusters prior to processing. System benchmark is a standard test of processor capacity run on the operating system to provide a baseline comparison of server performance across columns.

effects is marginally faster than within-transformed versions, and this owes to the fact that within transformations require conducting group-level analyses to store matrices Σ and Υ for each group, while models to return full fixed effects do not.

Turning to inference, we observe that when calculating clustered standard errors analytically, the processing time of cumulative algorithms approximately doubles. This owes to the procedure laid out in Section 2.4 where the calculation of standard errors requires estimates of $\widehat{u}$, and hence requires reading data two times. However, one particular advantage of fixed effect estimation is that if clustered standard errors are desired, running block bootstraps is virtually costless. In Table 1 we run 500 bootstraps following the procedure laid out in Section 2.4.3, seeing that this adds nearly no processing time (row 1 compared to row 3 of Table 1). We do not display the comparison for standard clustered bootstrap processing times in the table in a *non-cumulative* process, simply because this procedure increases linearly in the number of bootstraps, and is many orders of magnitude slower than cumulative procedures. In general, all group-level processing in Table 1 occurs when data is sorted in any order, however if data is indeed sorted by group prior to processing, estimation time further falls given that group-level processing of data can occur more rapidly (row 4 of panel A).

In the case of IV models, results are even more stark given that non-cumulative calculations require housing larger matrices in memory, and make processing infeasible at a broader range of memory restrictions. Indeed, in this case, even with data with only approximately 50 million observations, processing in non-cumulative routines becomes feasible at 32 GB of memory, but not before. We again see a number of key take-aways from Panel B of Table 1. These are, firstly, that cumulative routines open up feasibility where previously estimation could not occur. Secondly, even where processing can occur, cumulative algorithms are generally faster than non-cumulative procedures. In some cases, presumably where data approaches the memory limits of the computer, comparisons suggest that even where feasible, non-cumulative methods may be considerably slower than cumulative counterparts. This is especially clear in the comparison of IV models where all fixed effects are estimated (226 minutes) to a similar procedure in cumulative 2SLS (4.3 minutes).

It is important to note that this example should be conceived as simply illustrative of the fact that,

along with the conceptual interest of cumulative procedures, they do have practical implications too. Our implementations are unlikely to compare to commercial implementations in terms of the efficiency of every internal process, so could be conceived as lower bounds of the performance in this particular setting. In other languages and on other specific computational environments results will also vary. If instead of conducting these tests in single threaded versions of Stata we conduct them in multiple processor environments, results are observed to be similar in nature (Appendix Table A1). What's more, while this data has a reasonable number of observations (around 50 million), estimation is feasible with as little as 8GB of memory depending on the estimation procedure. However, in cases with much larger data, such memory limits will be far more binding, suggesting that the feasibility benefit of cumulative algorithms may be more important.

5 Discussion and Conclusions

In this paper we show that regressions can be estimated row-wise, without ever requiring all of the information from a dataset at a given moment of time. In the limit, we show that regressions can be estimated in a simple fashion where only a single line of data is read at a time and then forgotten, with only a small number of low-dimensional matrices needing to be updated over time. This fact appears to have been documented in very early computational work in economics (Brown, Houthakker, and Prais 1953), however only for a very specific variant of the problem.

Despite the ubiquity of procedures which work in a column-wise fashion in econometrics—based on results known since the seminal papers of Frisch and Waugh (1933), Lovell (1963)—to our knowledge these row-wise results have received scarce attention in the extent corpus of theoretical literature. We show that these results hold for a broad class of regression models including OLS, IV, fixed effect, and regularised regression models such as Ridge and LASSO, and that the logic of these results holds for both least squares and other M-class estimators such as probit and logit models. These results are not approximations, providing exact calculations of regression coefficients, and can similarly generate standard errors and other regression statistics such as goodness of fit measures exactly. Turning to inference, we show that these methods apply for both homoscedastic error assumptions, as well as for heteroscedasticity- and cluster-robust variance estimators. We additionally document

that certain bootstrap procedures and the definition of tuning parameters can be substantially more efficient with applications of the results from this paper.

As well as the theoretical interest of understanding the mechanics of frequently used regression models, these results have a large number of practical uses. Furthermore, we observe a disconnect between theoretical understanding and practical implementation, which this work aims to bridge. By offering a unifying theoretical framework for cumulative estimation, we demonstrate its potential to enhance computational efficiency and statistical inference across a wide range of econometric applications. In both simulated and real data, we show that these results imply that models which cannot be estimated on certain computers using standard commercial implementations of regression software can be estimated using the same programs but with our algorithms. What's more, even in cases where a computer's memory does not limit data from being opened, we show that our algorithms can at times offer non-trivial speedups over standard software. In some sense, these results could be cast as democratizing the processing of big data via regression, as, provided that a sufficiently large hard disk is available, one could process extremely large datasets with very low memory requirements. Indeed, there is no reason why these results could not be applied to data stored remotely or on the web, implying that it would not be necessary to have access to expensive supercomputers or High Performance Computing (HPC) settings to process data of any size. Sherry and Thompson (2021) assert that algorithmic progress has contributed more to computing performance than the increase in processor speed. We hope our methods are a stride in that direction.

The results in this paper are likely lower bounds for the true performance of these algorithms. We have implemented the results in this paper in a high-level matrix processing language, and comparisons are made to programs written largely in faster low-level languages. What's more, the procedures we use here process all data in a naive sequential fashion. The results from this study make clear that regression is an embarrassingly parallelisable task, and so if data is stored or broken down into various chunks in different files, processing times likely also scale approximately linearly in parallel processes. Per-core memory becomes a bottleneck even in HPC environments, which our techniques could alleviate by saving time and energy. All told, this paper suggests that cumulative estimation procedures can be broadly applied in a wide range of common regression models in

economics, and formally defines estimation and inference in such settings.

References

Aaronson, D., R. Dehejia, A. Jordan, C. Pop-Eleches, C. Samii, and K. Schulze (2020, 08). The Effect of Fertility on Mothers' Labor Supply over the Last Two Centuries. *The Economic Journal 131*(633), 1–32.

Abraham, K. G., R. S. Jarmin, B. Moyer, and M. D. Shapiro (2022). *Big Data for Twenty-First-Century Economic Statistics*. Studies in Income and Wealth. University of Chicago Press. Conference held March 15-16, 2019.

Baltagi, B. H. (2001). *Econometric Analysis of Panel Data* (2nd ed.). Chichester: John Wiley and Sons.

Bhalotra, S. and D. Clarke (2023). *Analysis of Twins*, pp. 1–37. Springer International Publishing.

Brown, J. A. C., H. S. Houthakker, and S. J. Prais (1953). Electronic Computation in Economic Statistics. *Journal of the American Statistical Association 48*(263), 414–428.

Correia, S. (2015, March). HDFE: Stata module to partial out variables with respect to a set of fixed effects. Statistical Software Components, Boston College Department of Economics.

Frisch, R. and F. V. Waugh (1933). Partial Time Regressions as Compared with Individual Trends. *Econometrica 1*(4), 387–401.

Fu, W. (1998). Penalized Regressions: The Bridge vs. the Lasso. *Journal of Computational and Graphical Statistics 7*(3), 397–416.

Gaure, S. (2013). OLS with multiple high dimensional category variables. *Computational Statistics & Data Analysis 66*, 8–18.

Green, P. J. (1984). Iteratively Reweighted Least Squares for Maximum Likelihood Estimation, and some Robust and Resistant Alternatives. *Journal of the Royal Statistical Society. Series B (Methodological) 46*(2), 149–192.

Guimarães, P. and P. Portugal (2010, December). A simple feasible procedure to fit models with high-dimensional fixed effects. *Stata Journal 10*(4), 628–649.

Hager, W. W. (1989, 6). Updating the Inverse of a Matrix. *SIAM Review 31*(2), 221–239.

Jordan, M. L. (2018). Working with Big Data in SAS®. Technical Report SAS2160-2018, SAS Institute Inc.

Karim, S., M. D. Webb, N. Austin, and E. Strumpf (2024). Difference-in-differences with unpoolable data.

Lovell, M. C. (1963). Seasonal Adjustment of Economic Time Series and Multiple Regression Analysis. *Journal of the American Statistical Association 58*(304), 993–1010.

Miller, K. S. (1981, 3). On the Inverse of the Sum of Matrices. *Mathematics Magazine 54*(2), 67–72.

Nelder, J. A. and R. W. M. Wedderburn (1972). Generalized linear models. *Journal of the Royal Statistical Society. Series A (General) 135*(3), 370–384.

Robbins, H. and S. Monro (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics 22*(3), 400 – 407.

Rosenzweig, M. R. and K. I. Wolpin (1980). Testing the Quantity-Quality Fertility Model: The Use of Twins as a Natural Experiment. *Econometrica 48*(1), 227–240.

Sherry, Y. and N. C. Thompson (2021). How fast do algorithms improve? [point of view]. *Proceedings of the IEEE 109*(11), 1768–1777.

Varian, H. R. (2014, May). Big Data: New Tricks for Econometrics. *Journal of Economic Perspectives 28*(2), 3–28.

Woodbury, M. A. (1950). Inverting modified matrices. Memorandum Report 42, Statistical Research Group, Princeton University, Princeton, NJ.

Wooldridge, J. M. (2021). Two-Way Fixed Effects, the Two-Way Mundlak Regression, and Difference-in-Differences Estimators. Technical report, SSRN.

Wu, Y. and L. Wang (2020). A Survey of Tuning Parameter Selection for High-Dimensional Regression. *Annual Review of Statistics and Its Application 7*(1), 209–226.

Appendices for: Cumulative Estimation

Damian Clarke, Nicolás Paris Torres & Benjamín Villena-Roldán

Not for print.

A A Simple Visualisation in Matrix Form

Consider a simple illustration of the generation of $X'X$ based on a case where X is a matrix consisting of $N = 4$ observations and $K = 3$ independent variables. For ease of visualisation, we will denote X as follows:

$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \\ x_{4,1} & x_{4,2} & x_{4,3} \end{pmatrix}$$

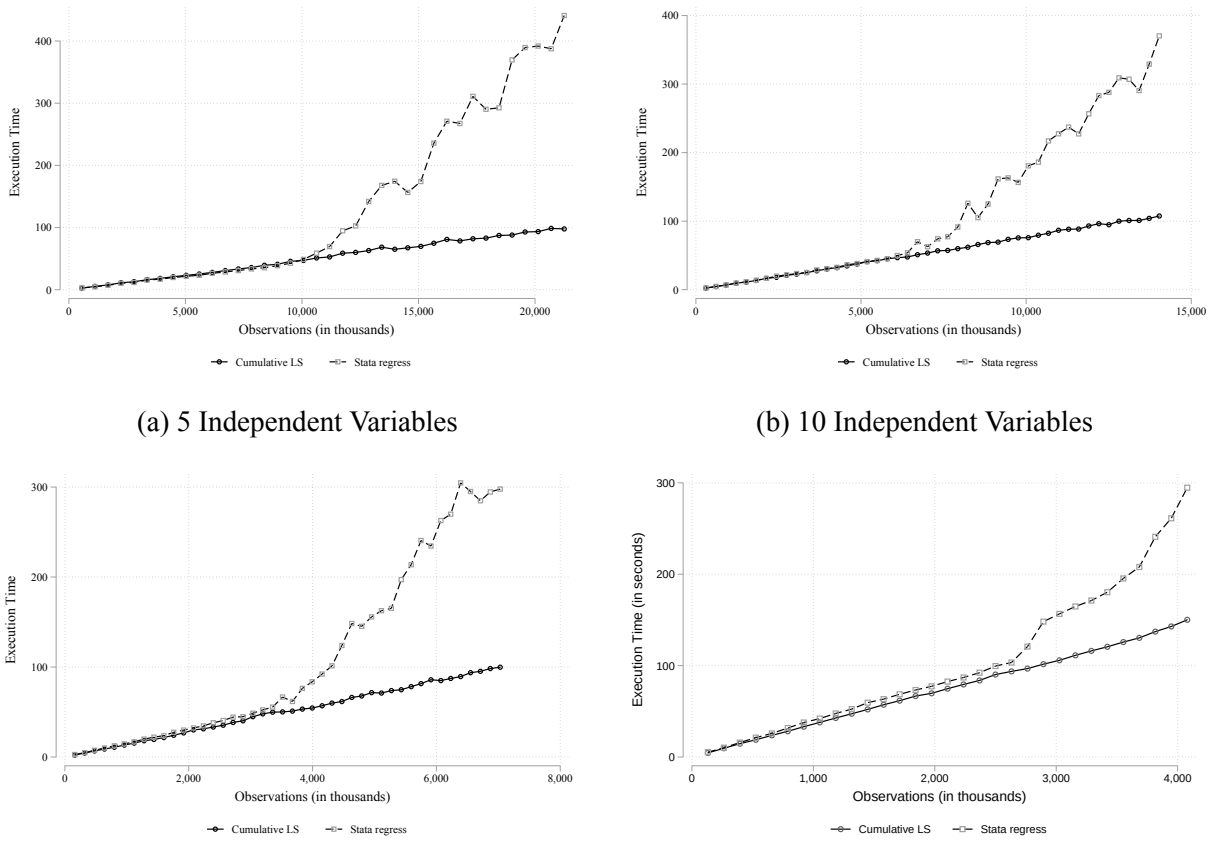
where all realisations of independent variables for observation $i = 1$ are coloured in blue, and for observation $i = 4$ are coloured in red. Now note that $X'X$ can be written in extensive form as below:

$$\begin{aligned} X'X \equiv & \begin{pmatrix} x_{1,1} & x_{2,1} & x_{3,1} & x_{4,1} \\ x_{1,2} & x_{2,2} & x_{3,2} & x_{4,2} \\ x_{1,3} & x_{2,3} & x_{3,3} & x_{4,3} \end{pmatrix} \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \\ x_{4,1} & x_{4,2} & x_{4,3} \end{pmatrix} \\ = & \begin{pmatrix} x_{1,1}^2 + x_{2,1}^2 + x_{3,1}^2 + x_{4,1}^2 & x_{1,1}x_{1,2} + x_{2,1}x_{2,2} + x_{3,1}x_{3,2} + x_{4,1}x_{4,2} & x_{1,1}x_{1,3} + x_{2,1}x_{2,3} + x_{3,1}x_{3,3} + x_{4,1}x_{4,3} \\ x_{1,2}x_{1,1} + x_{2,2}x_{2,1} + x_{3,2}x_{3,1} + x_{4,2}x_{4,1} & x_{1,2}^2 + x_{2,2}^2 + x_{3,2}^2 + x_{4,2}^2 & x_{1,3}x_{1,1} + x_{2,3}x_{2,1} + x_{3,3}x_{3,1} + x_{4,3}x_{4,1} \\ x_{1,3}x_{1,1} + x_{2,3}x_{2,1} + x_{3,3}x_{3,1} + x_{4,3}x_{4,1} & x_{1,3}x_{1,2} + x_{2,3}x_{2,2} + x_{3,3}x_{3,2} + x_{4,3}x_{4,2} & x_{1,3}^2 + x_{2,3}^2 + x_{3,3}^2 + x_{4,3}^2 \end{pmatrix} \\ = & \begin{pmatrix} x_{1,1}^2 & x_{1,1}x_{1,2} & x_{1,1}x_{1,3} \\ x_{1,2}x_{1,1} & x_{1,3}x_{1,1} & x_{1,3}x_{1,1} \\ x_{1,3}x_{1,1} & x_{1,3}x_{1,2} & x_{1,3}^2 \end{pmatrix} + \begin{pmatrix} x_{2,1}^2 & x_{2,1}x_{2,2} & x_{2,1}x_{2,3} \\ x_{2,2}x_{2,1} & x_{2,2}^2 & x_{2,3}x_{2,1} \\ x_{2,3}x_{2,1} & x_{2,3}x_{2,2} & x_{2,3}^2 \end{pmatrix} + \cdots + \begin{pmatrix} x_{4,1}^2 & x_{4,1}x_{4,2} & x_{4,1}x_{4,3} \\ x_{4,2}x_{4,1} & x_{4,2}^2 & x_{4,3}x_{4,1} \\ x_{4,3}x_{4,1} & x_{4,3}x_{4,2} & x_{4,3}^2 \end{pmatrix}. \end{aligned} \tag{15}$$

The key takeaway here is that one can simply arrive to the $K \times K$ matrix $X'X$ by calculating 4 $K \times K$ matrices based on cross-products for each observation (ie working an observation at a time), and finally summing across these matrices (as in (15)).

B Appendix Figures and Tables

Figure A1: Sample size and execution time of standard versus cumulative (Multiple Cores)



(a) 5 Independent Variables

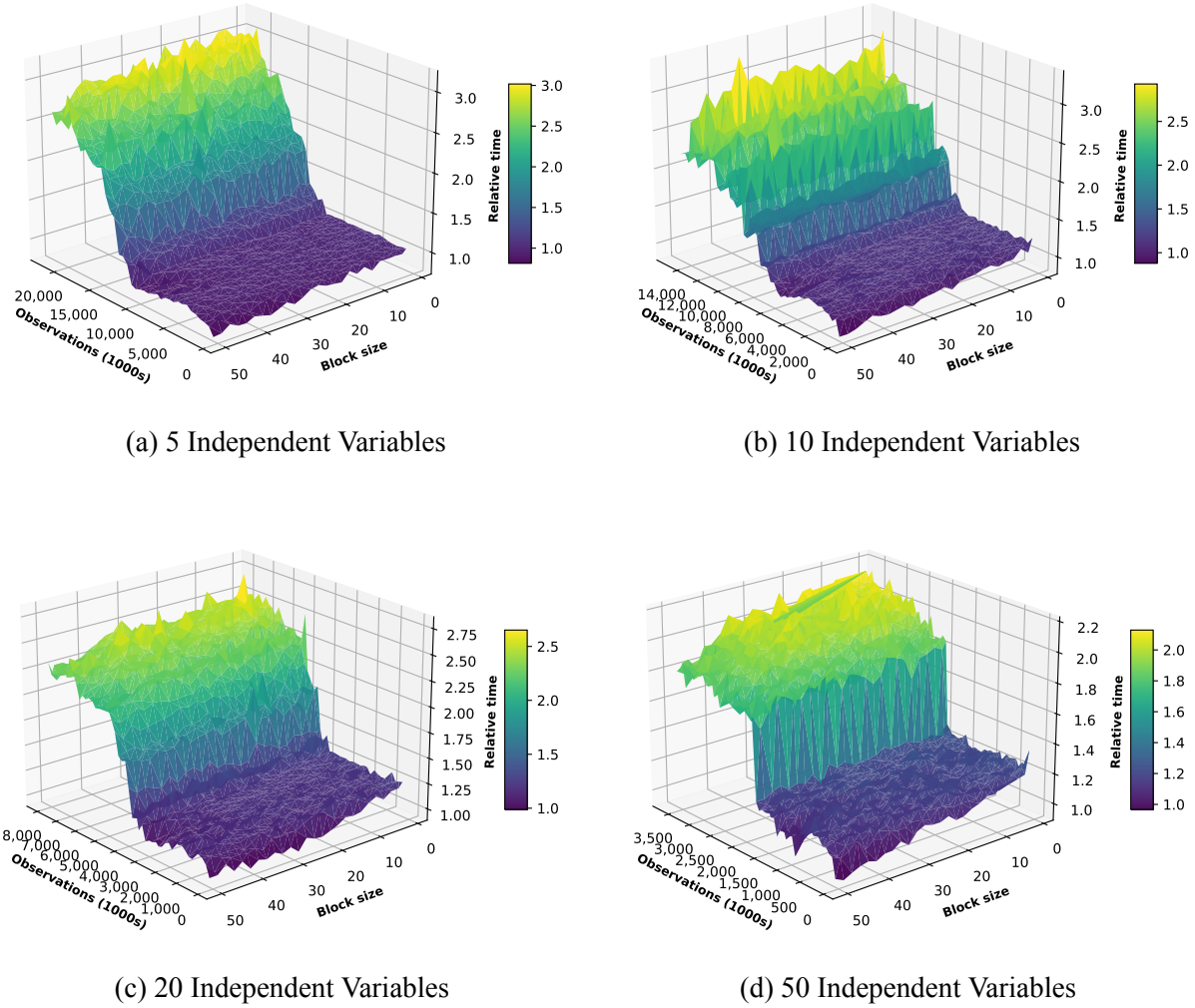
(b) 10 Independent Variables

(c) 20 Independent Variables

(d) 50 Independent Variables

Notes: All times refer to the computation time of reading data into memory and estimating an ordinary least squares regression. Tests are all conducted on a system with 1GB of RAM, with no other processes running, using Stata MP (2 cores). In each case, tests are conducted up to the point at which there is insufficient RAM to open the data, thus precluding the estimation of standard regression models. Beyond this point, it is still feasible to estimate parameters using Cumulative Least Squares.

Figure A2: Relative Performance of OLS to CLS by Block Size



(a) 5 Independent Variables

(b) 10 Independent Variables

(c) 20 Independent Variables

(d) 50 Independent Variables

Notes: Surface plots present the ratio of computation times of Stata's native regression command to cumulative least squares. Values less than 1 imply non-cumulative procedures are faster than cumulative procedures, and vice versa for values greater than 1. All times refer to the computational time of reading data into memory and estimating a cumulative least squares regression. Tests are all conducted on a system with 1GB of RAM, with no other processes running. Block sizes indicated on the horizontal axis refer to the proportion of the full dataset (*eg* 50 refers to 2 blocks each covering 50% of the data, 1 refers to 100 blocks each covering 1% of the data).

Table A1: Execution Time Across Models in a Labour Market Participation Example (MP): OLS and IV

	Memory Constraints				
	1 GB	2 GB	8 GB	16 GB	32 GB
Panel A: OLS					
Cumulative Procedures					
Within-transformation	2.973	3.153	3.266	3.091	3.303
Within, Clustered	5.735	6.003	6.315	5.973	6.438
Within, Clustered Bootstrap	3.070	3.002	3.218	3.124	3.342
Clustered Bootstrap (s)	2.584	2.760	2.930	2.639	3.131
Full Estimation	2.843	2.895	3.181	2.960	3.185
Standard Procedures					
Within-transformation (areg)	–	–	4.511	4.187	3.846
Within, Clustered	–	–	5.368	5.257	4.532
Full Estimation	–	–	4.500	4.143	3.630
Within-transformation (hdfe)	–	–	–	3.397	3.177
Within, Clustered (hdfe)	–	–	–	3.537	3.304
Panel B: IV					
Cumulative Procedures					
Within-transformation	3.371	3.435	3.501	3.316	3.651
Within, Clustered	6.351	6.498	6.798	6.374	6.950
Within, Clustered Bootstrap	3.242	3.312	3.479	3.306	3.468
Winthin, Clustered Bootstrap (s)	3.036	3.077	3.192	3.276	3.585
Full Estimation	2.957	3.003	3.142	2.981	3.538
Standard Procedures					
Within-transformation (xtivreg)	–	–	–	–	4.968
Within, Clustered	–	–	–	–	10.916
Full Estimation	–	–	–	–	45.350
Within-transformation (hdfe)	–	–	–	–	4.834
Within, Clustered (hdfe)	–	–	–	–	6.597
Panel C: System Benchmark	866.2	852.7	856.4	842.4	843.1

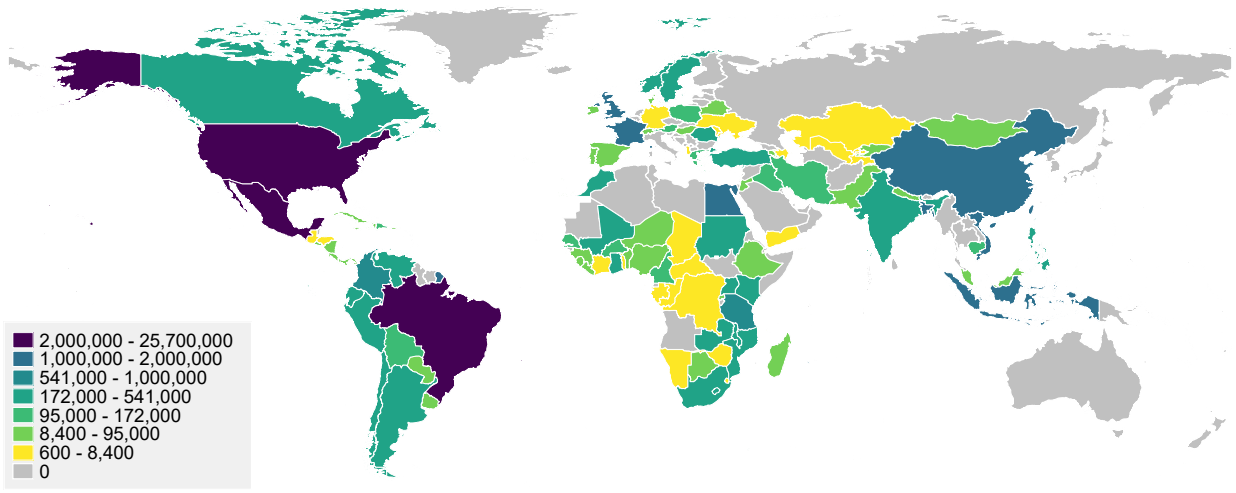
Notes: Each cell reports average processing time in minutes of a particular estimation procedure based on the memory constraints listed in column headers. All averages are taken over 10 estimations. Cells are coloured gray when estimation cannot occur due to memory limits. All methods are estimated in Stata 18 MP. For clustered bootstrap errors, (s) stands for "sorted", implying that the database is sorted by clusters prior to processing. System benchmark is a standard test of processor capacity run on the operating system to provide a baseline comparison of server performance across columns.

C Data Appendix

We collate original data from IPUMS and the Demographic and Health Survey (DHS) repository using all census data and DHS waves described in Aaronson, Dehejia, Jordan, Pop-Eleches, Samii, and Schulze (2020). This results in 51,449,770 observations drawn from 434 census files for 106 countries covering years 1787 to 2015. The geographical coverage of the data is described in Figure A3, and the temporal coverage is described in Figure A4.

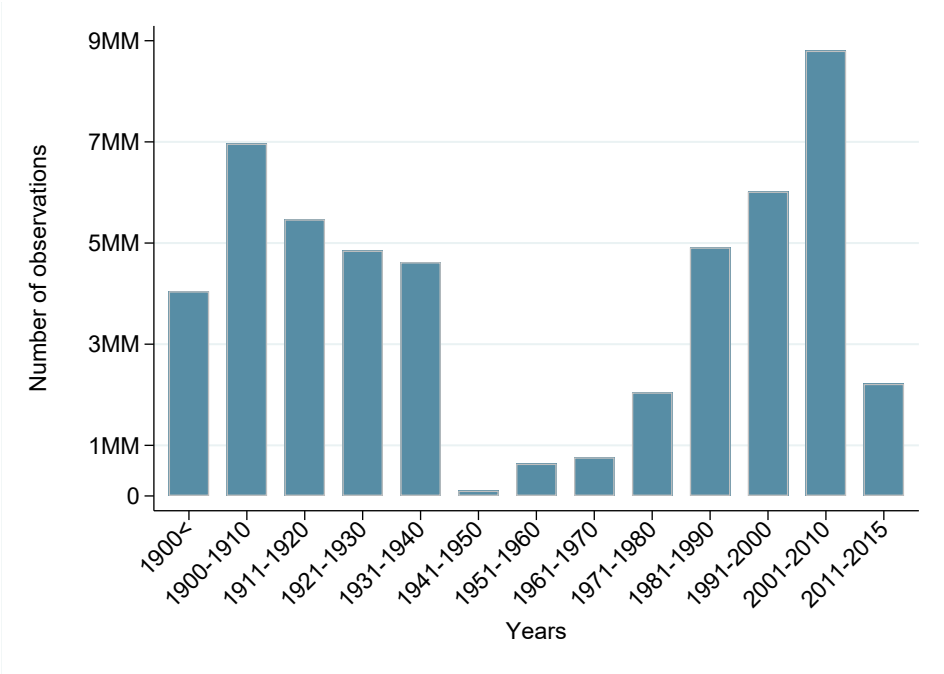
We follow replication materials of Aaronson et al. (2020) to generate all variables, and replicate their results exactly. We follow their inclusion criteria of working with women aged 21 to 35 who have at least two children, all of whom are 17 or younger. As described in Aaronson et al. (2020) families are excluded where information is missing on child gender or mother's age, and mothers are not included in the sample if they live in group quarters or give birth before the age of 15. Summary statistics of all data following the processes described in Aaronson et al. (2020) are included below in Table A2.

Figure A3: Number of Observations by Country



Notes: Values plotted refer to the total frequency of observations used in estimating samples. These are pooled across all years. Countries coloured grey have no available microdata on IPUMS or DHS.

Figure A4: Number of Observations by Year



Notes: Values refer to the total frequency of observations used in estimating samples. Years refer to the year the data was collected.

Table A2: Summary Statistics – Principal Variables

VARIABLES	Obs	Mean	Std. Dev.	Min	Max
Labour Force Participation	51,449,770	0.263	0.441	0	1
Covariables					
Fertility	51,449,770	0.525	0.499	0	1
Gender of first child	51,449,770	0.508	0.499	0	1
Mother Age	51,449,770	29.43	3.859	21	35
Mother age at first birth	51,449,770	21.04	3.300	15	35
Weights	51,449,770	1	0.390	0.0007	341.66
Years	51,449,770	1954.95	44.169	1787	2015
Instrument					
Twin	51,449,770	0.0105	0.102	0	1

Notes: Summary statistics are displayed based on all observations and data cleaning procedures described in Aaronson et al. (2020). Sample consists of women giving birth at the age of 15 or above, and ages 21 to 35 at the time of data collection. All selection criteria follows Aaronson et al. (2020).

D An Updating Estimation Procedure

Throughout the paper we work with a cumulative procedure in which matrices $X'X$ and $X'y$ are accumulated in a step-wise fashion, and estimates $\widehat{\beta}_{OLS}$ (or similar for other estimation methodologies) are generated only after data is read in its entirety. Alternative procedures can be used in which iterations occur over sequential iterations of $\widehat{\beta}_{OLS}$ estimates themselves, though these are less efficient than the cumulative procedures laid out in the body of the paper.

To see this, we will use identical notation to that laid out in the paper. Suppose we wish to estimate a regression between a dependent variable y and a set of K covariates $X_1, X_2, ..., X_K$. The database that can be partitioned into J samples. The whole sample size of the database is N, but computing an OLS regression with all the data is unfeasible due to memory constrains. As an alternative procedure, we could run a regression with the sample $j = 1$ and update the result with the other $J - 1$ samples. To fix ideas, suppose we have a samples 1 and 2 and we want to compute an OLS estimator for the whole sample denoted by the subindex $1 \sim 2$. Hence,

$$\begin{aligned}\hat{\beta}_{1\sim 2} &= \left(\begin{pmatrix} X_1' & X_2' \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \right)^{-1} \begin{pmatrix} X_1' & X_2' \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = (X_1'X_1 + X_2'X_2)^{-1}(X_1'y_1 + X_2'y_2) \\ &= (\Sigma_1 + \Sigma_2)^{-1}(\Upsilon_1 + \Upsilon_2)\end{aligned}$$

where $X_j'X_j \equiv \Sigma_j$ and $X_j'y_j \equiv \Upsilon_j$. The challenge is to compute $\hat{\beta}_{1\sim 2}$ using estimates from the two samples separately, so that we can avoid storing very large databases in memory. Trivially, as laid out in the body of the paper, this can be done cumulatively. Alternatively, we can make use of a result of the inverse of the sum of two matrices by Miller (1981). A more general perspective in this theory, including application to linear least squares is Hager (1989). The application of this result can be proven easily.

Lemma 1 *The inverse of the sum of two matrices can be obtained as*

$$(\Sigma_1 + \Sigma_2)^{-1} \equiv \Sigma_{1\sim 2}^{-1} = \Sigma_1^{-1} - \Sigma_1^{-1}\Sigma_2(I + \Sigma_1^{-1}\Sigma_2)^{-1}\Sigma_1^{-1}$$

Proof. Follows as a direct application of Woodbury (1950) matrix inverse lemma ■

Using Lemma 1, we can iterate on $\widehat{\beta}_{OLS}$, without requiring that all of the elements stored in

cumulative procedures are accumulated across iterations. We define the factor

$$\Omega_{1\sim 2} \equiv I_K - \Sigma_1^{-1}\Sigma_2 \left(I_K + \Sigma_1^{-1}\Sigma_2\right)^{-1},$$

where I_K stands for an identity matrix of dimension K, the number of regressors in the model. Hence, the joint Σ matrix can be expressed as

$$\Sigma_{1\sim 2}^{-1} = \Omega_{1\sim 2}\Sigma_1^{-1}$$

Therefore, the joint OLS estimator becomes

$$\widehat{\beta}_{1\sim 2} = \Omega_{1\sim 2}\left(\widehat{\beta}_1 + \Sigma_1^{-1}\Upsilon_2\right). \tag{16}$$

This suggests an iterative procedure for estimating OLS parameters generalising the above result to J blocks, rather than 2 blocks. This is defined as Updated Ordinary Least Squares in Algorithm 6.

Algorithm 6: Updated Ordinary Least Squares

Inputs: Database consisting of (y,X), block size b.
Result: Point estimate $\widehat{\beta}_{OLS}$ (and potentially point estimates updated at each step, $\widehat{\beta}_{1\sim j,OLS}$).

1. Set $i = 1$ and $j = b$. Load into memory partition of data covering y, X in observations $i - j$. ;
2. Calculate Σ_1 and $\widehat{\beta}_1$;
3. If observations i-j contain end of file, set $e = 1$, otherwise, set $e = 0$;

while $e \neq 1$ **do**

4. Replace $i = i + b$ and $j = j + b$. Load into memory partition of data covering y, X in observations $i - j$. ;
5. Calculate Σ_j and Υ_j. ;
6. Calculate $\Omega_{1\sim j} = I_K - \Sigma_{j-1}^{-1}\Sigma_j \left(I_K + \Sigma_{j-1}^{-1}\Sigma_j\right)^{-1}$ and $\widehat{\beta}_{1\sim j} = \Omega_{1\sim j}\left(\widehat{\beta}_{j-1} + \Sigma_{j-1}^{-1}\Upsilon_j\right)$;
7. If observations i-j contain end of file, set $e = 1$, otherwise, set $e = 0$;

end

While this procedure allows for the calculation of updated values of $\widehat{\beta}_{1\sim j}$ at each step, and additionally avoids the need of passing $\Upsilon_{1\sim j}$ across steps, each iteration involves one matrix inversion of size K in step 6. Indeed, for any given quantity of variables K and block size b, the cumulative algorithm strictly dominates the updating algorithm in terms of total computations (and hence computation time). This owes to the fact that the same calculations of order $\mathcal{O}(NK^2) + O(NK)$ discussed in Section 3 are required in calculating Σ_j and Υ_j as inputs for (16), strictly *more* elements

are required to be summed in iterating on $\Omega_{1\sim j}$ instead of $\Sigma_{1\sim j}$ and $\Upsilon_{1\sim j}$, and additionally, a matrix inversion is required at each step in calculating (16). For this reason, we focus on cumulative least squares algorithms throughout this paper.