

Article

Some Computational Aspects of Feasible GLS Estimation of Large Panels in R

Giovanni Millo 

Dipartimento di Scienze Economiche, Aziendali, Matematiche e Statistiche (DEAMS), University of Trieste,
34127 Trieste, Italy; giovanni.millo@deams.units.it

Abstract

Econometric estimation of panel data models by feasible generalized least squares (FGLS) provides an example of how conceptually simple problems may run into computational bottlenecks. I address the main computational tasks of FGLS within the R system for statistical computing, comparing different tools from the point of view of computational efficiency. I concentrate on estimating two models: the popular “random effects” with two error components and the less restrictive “general GLS” specification, which does not fit into the standard computational framework usually employed for the former. I compare the standard solution (partial time demeaning) with two alternative strategies, based respectively on algebraic properties and on object-oriented programming. I show how, while naive implementations become infeasible with large datasets, both list operators and object-oriented matrix routines available in the R environment make the problem tractable for most practically relevant sample sizes on any machine. I conclude by briefly discussing the parallelization of critical tasks.

Keywords: panel; FGLS; computing; object orientation; parallelization

MSC: 62J05; 62H12; 65F30; 65C20; 62P20

1. Introduction

Feasible generalized least squares (FGLS) estimation plays a central role in the econometric analysis of panel data models. While the underlying theory is well-established, its practical implementation can become computationally demanding when applied to large datasets, particularly in settings where the cross-sectional dimension is large relative to the time dimension. In such cases, operations involving the covariance matrix of the error term quickly become the main computational bottleneck.

This issue is especially relevant for modern microeconomic applications, where datasets often consist of thousands or tens of thousands of individual units observed over a relatively small number of time periods. In these contexts, more flexible specifications of the error structure, such as general GLS (GGLS), which allows for unrestricted within-unit covariance, are theoretically appealing but are often regarded as impractical due to their computational cost.

Standard approaches to random effects estimation exploit model-specific transformations, such as quasi-time demeaning, to reduce the problem to ordinary least squares. While computationally efficient, these methods are not applicable to more general covariance structures.



Academic Editor: Baccouch Mahboub

Received: 16 April 2026

Revised: 8 June 2026

Accepted: 10 June 2026

Published: 17 June 2026

Copyright: © 2026 by the author.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](https://creativecommons.org/licenses/by/4.0/)

[Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

This paper revisits the computational problem of FGLS estimation from a practical and implementation-oriented perspective. Focusing on the R environment for statistical computing [1], I show that a combination of modern programming techniques (vectorization, functional programming constructs, and object-oriented matrix representations) makes it possible to implement both standard and general GLS estimators efficiently, even for large panel datasets.

1.1. Research Context

In this sense, my contribution follows in the line of research of Gaure [2,3] and Bergé et al. [4] on efficient computing of panel estimators, but with a different focus and a more modest aim. The above papers in fact concentrate on efficient estimation in settings including a large amount of fixed effects (potentially in the millions) across multiple levels, achieving optimization through precompiled routines in C++, while I address random effects estimation employing user-level functionality.

Related recent advances in the field also include work on weighted average least squares [5] with implementations in Stata [6], and on generalizing the error covariance matrix to including cross-sectional correlations, managing the resulting large number of correlations via banding and thresholding [7]. An implementation of this last method would substantially extend the feasible general GLS estimator considered here, and is left for future work.

1.2. Empirical Relevance

The computational results presented in this paper also have broader methodological implications for applied panel data research. In practice, the choice between alternative panel estimators is often influenced not only by statistical considerations, but also by computational feasibility. Because standard random effects estimation can be implemented through highly efficient transformations, applied researchers have traditionally preferred relatively restrictive covariance structures, especially when working with large datasets. The results of this study suggest that such preferences may partly reflect computational constraints rather than purely econometric considerations.

More efficient implementations of feasible GLS procedures reduce the practical cost of adopting richer covariance specifications, making it increasingly feasible to estimate models allowing for more general forms of within-unit heteroskedasticity and serial correlation. This may encourage a broader use of flexible GLS-based approaches in empirical applications where restrictive random effects assumptions are questionable. More generally, the findings highlight how advances in computational methods and software infrastructure can influence empirical modeling practices by expanding the range of estimators that are practically usable in large-scale applications.

These considerations are particularly relevant in modern microeconomic applications based on large longitudinal datasets, such as household and labor-market panel surveys, firm-level administrative databases, or repeated health and educational observations, where the cross-sectional dimension may reach tens or hundreds of thousands of units while the temporal dimension remains comparatively short. In such contexts, computational constraints have traditionally favored simpler covariance specifications even when richer dependence structures may be empirically more appropriate. More efficient feasible GLS implementations therefore widen the set of practically usable modeling strategies for applied researchers working with large-scale panel data.

1.3. Contributions

The contribution of this paper is computational and methodological, in the sense that it clarifies how different implementation strategies affect the practical feasibility of FGLS estimation in large panels. The main contributions can be summarized as follows:

1.3.1. Unified Computational Comparison of FGLS Strategies

The paper provides a systematic comparison of three alternative approaches to computing FGLS estimators: (i) quasi-time demeaning, (ii) block-wise GLS based on algebraic decomposition, and (iii) full-matrix methods exploiting object-oriented matrix classes. These approaches are evaluated within a common framework and applied to both random effects and general GLS specifications.

1.3.2. Demonstration of Efficient Implementations for General GLS

It is shown that general GLS estimation can be implemented efficiently in R using block-wise formulations combined with optimized list-based operations. This significantly lowers the practical barrier to using more flexible covariance structures in applied work.

1.3.3. Assessment of Programming Paradigms for Numerical Performance

The paper highlights the performance implications of different programming styles in R, contrasting traditional loop-based implementations with functional programming constructs (e.g., `lapply`, `tapply`) and vectorized operations. The results show that high-level language features, when properly used, can yield substantial performance gains without resorting to low-level programming.

1.3.4. Evaluation of Object-Oriented Matrix Representations

By exploiting specialized matrix classes (e.g., block-diagonal symmetric matrices), the paper demonstrates that “naive” formulations based on full covariance matrices can remain computationally viable, thanks to optimized storage, algebraic routines, empirical benchmarking and practical guidance. Through extensive simulation experiments covering a range of panel dimensions, the paper provides practical evidence on the relative performance of alternative implementations. The results offer guidance on the choice of computational strategy depending on the structure and size of the dataset.

1.3.5. Discussion of Parallelization Potential

The paper briefly examines the extent to which FGLS computations can benefit from parallel execution, identifying the components of the algorithm that are most amenable to parallelization and assessing the resulting performance gains.

Overall, the paper shows that the computational constraints traditionally associated with FGLS estimation are, to a large extent, implementation-dependent. By leveraging modern features of the R environment, it is possible to make a broader class of panel data models tractable in practice.

1.4. Limitations

This piece of work is deliberately confined to computing solutions within the reach of an advanced user of R who, in the true spirit of the R Project, evolves into programming one’s own methods: “useRs become programmers”. As such, it does not touch the most advanced and powerful paths to performance enhancement in R (or, in the case of parallelization, it explores only the easiest solutions).

1.4.1. Parallelization

R offers several parallelization options that vary in abstraction level and platform support. The base *parallel* package (included with R) provides low-level tools like `mclapply()` (based on forking, Unix/macOS only) and `textttparLapply()` (using PSOCK clusters, cross-platform), enabling multicore and distributed execution.

Higher-level frameworks such as *foreach* (with backends like *doParallel*, *doSNOW*, or *doFuture*) allow expressive parallel loops with flexible execution engines. *RcppParallel* enables parallelism in C++ code for performance-critical tasks. Overall, R's parallel infrastructure spans from simple drop-in replacements for **apply* functions to scalable, backend-agnostic systems suited for clusters and cloud environments.

Low-level parallelization in R can be achieved by linking it against a multithreaded BLAS/LAPACK implementation, which transparently accelerates linear algebra operations (e.g., matrix multiplication, decompositions) without changing R code. Common choices include OpenBLAS, Intel MKL, and Apple's Accelerate framework, all of which exploit SIMD instructions and multithreading to speed up numerical routines used by functions like `%*%`, `solve()`, and `svd()`. Installation typically involves replacing the default reference BLAS with a high-performance alternative (via system packages or tools like update-alternatives on Linux, or custom builds on macOS/Windows). Care must be taken to avoid oversubscription when combined with higher-level parallel tools. This approach is especially effective for compute-heavy workloads dominated by dense linear algebra, offering substantial speedups with minimal code changes; on the converse, it cannot be expected to deliver substantial gains for smaller-scale matrix operations, the overhead often crowding out the performance increase.

In the spirit of this paper, which is to explore simple, user-level ways to enhance computational performance, I only address the (almost transparent) parallelization of list operations through substituting `lapply()` with `mclapply()`.

1.4.2. Precompiled Code

Importantly, I do not address the most promising, and most advanced, performance-enhancing solution: linking to precompiled routines. In fact, enhancing performance in R often involves offloading computationally intensive tasks to precompiled code written in lower-level languages such as C, C++, or Fortran. Through interfaces like `.C()`, `.Call()`, or the modern *Rcpp* package [8], R can seamlessly call optimized native routines, significantly reducing execution time for loops, numerical methods, or large-scale simulations. This approach leverages the speed and memory efficiency of compiled languages while retaining R's flexibility for high-level analysis, making it especially effective for performance-critical sections of statistical workflows.

The aim of the paper is more modestly to suggest efficient user-level computational strategies, making use of readily available infrastructure.

2. Methodology

Let us briefly sketch the econometric problem. Panel data regressions draw on data with double subscripting, usually one for individuals (groups, firms, countries. . .) and the other for time.

$$y_{it} = X_{it}\beta + v_{it} \quad (1)$$

The assumption that v_{it} be independent and identically distributed (*iid* across individuals *and* time periods) is generally considered too simplistic. The individual error components model is the workhorse of panel data econometrics.

2.1. The Random Effects Model

In the standard one-way error components model, the disturbance term may be written as

$$v_{it} = c_i + u_{it} \quad (2)$$

where c_i denotes the (time-invariant) individual-specific effect and u_{it} the idiosyncratic error. Observations regarding the same individual share the same c_i effect; thus, the relative errors are autocorrelated. Ordinary least squares estimates for β in model (Equation (1)) are therefore inefficient, though consistent.

The random effects model is characterized by the assumption that the individual specific effect c_i is random and uncorrelated both with the explanatory variables and with the idiosyncratic error. Under this specification and under homoskedasticity in both c_i and u_{it} and no serial correlation in u_{it} , the variance–covariance matrix of the errors becomes

$$V = \sigma_c^2(I_N \otimes J_T) + \sigma_u^2(I_N \otimes I_T) \quad (3)$$

i.e., it is block-diagonal with

$$V = I_N \otimes \Omega \quad (4)$$

where

$$\Omega = \begin{bmatrix} \sigma_u^2 + \sigma_c^2 & \sigma_c^2 & \dots & \sigma_c^2 \\ \sigma_c^2 & \sigma_u^2 + \sigma_c^2 & \dots & \vdots \\ \dots & \dots & \ddots & \sigma_c^2 \\ \sigma_c^2 & \dots & \dots & \sigma_u^2 + \sigma_c^2 \end{bmatrix} \quad (5)$$

2.2. The Estimators

By Aitken's theorem (see, e.g., [9] (Section 10.5)), generalized least squares (GLS) is the efficient estimator for the model parameters if Ω is known. The estimator is then

$$\hat{\beta}_{GLS} = (X'V^{-1}X)^{-1}(X'V^{-1}y) \quad (6)$$

Various feasible GLS procedures exist, drawing on consistent estimators of Ω which are then plugged into V by (4) and hence into (6).

2.2.1. FGLS Estimators for the RE Model

Several estimation procedures have been proposed for the variances of the two error components in Equation (5):

- Wallace and Hussein use the residuals from the pooled model $\hat{v}_{OLS} = \tilde{v}$ to estimate both σ_v^2 and $\sigma_c^2 = T \sum_t \bar{v}_i^2 / n$; hence $\sigma_c^2 = (\sigma_v^2 - \sigma_u^2) / T$;
- Amemiya uses the LSDV residuals as above;
- Swamy and Arora estimate σ_v^2 from the between residuals $\hat{v}_{BE}$ and σ_u^2 from the within residuals $\hat{v}_{FE}$;
- Nerlove estimates σ_c^2 on the basis of the dummy coefficients in the LSDV regression, and σ_u^2 from the within residuals; then, $\sigma_v^2 = T\sigma_c^2 + \sigma_u^2$;
- Wooldridge uses pooled residuals $\hat{v}_{OLS}$ and estimates total error variance as $\sigma_v^2 = \sum_i \sum_t u^2 / (nT - K)$ and σ_c^2 as average of nondiagonal error products.

In the following, we will employ the Wooldridge solution as the base case. In any case, being all based on OLS, none of the components' estimation procedures poses substantial computational problems in comparison to the GLS steps that follow them.

2.2.2. General GLS Estimation

The above, assuming both (4) and (5), is the standard approach to random effect panels and has long been implemented in most econometric software packages. In “large N” panels, a less restrictive approach is possible, termed *general GLS* estimator [10] (Section 10.4.3), which allows for arbitrary *intragroup* heteroskedasticity and serial correlation of errors, i.e., inside the Ω covariance blocks, provided that these remain the same for every individual. Formally, it assumes (4), while the only requisites for Ω are to be symmetric and positive definite:

$$\Omega(ij) = \Omega(ji) = \sigma_{ij} \quad (7)$$

This is sometimes referred to as the Parks estimator [11] and is, as observed by Driscoll and Kraay [12], a variant of the SUR estimator by Zellner [13]: see e.g., Greene [9], where the same estimator is presented in the context of pooled time series, with fixed N and “large” T.

This estimator is much less popular than RE-FGLS, but still has some less-known implementations in software packages. It is implemented in Eviews, where it is called *Cross-section SUR*, and in Stata (through the *xtgls* command); in R, it can be performed through the *pggls* function within the “plm” package [14]. Programming it by oneself in a high-level interpreted language proves difficult from the point of view of performance, as we will see in the following, unless one is able to outsource heavy tasks to precompiled routines or unless computationally efficient tools are available.

2.2.3. Comparison and Practical Relevance

From the perspective of applied empirical research, the standard random effects (REs) specification remains the dominant feasible GLS approach because it offers a convenient compromise between statistical efficiency, interpretability, and computational simplicity. By reducing the covariance structure to two variance components, RE estimation can exploit transformations such as quasi-time demeaning, making estimation feasible even for very large datasets through ordinary least squares routines [15]. This computational convenience explains its widespread implementation in econometric software and its extensive use in microeconomic and longitudinal applications.

General GLS (GGLS) specifications, by contrast, allow for unrestricted within-unit covariance structures and are therefore potentially more realistic in settings characterized by heteroskedasticity and serial correlation of unknown form [9,10]. Such flexibility may be especially relevant in short panels with a large cross-sectional dimension, where restrictive error-component assumptions can become empirically questionable. In fact, the GGLS estimator requires $N \gg T$, as the number of “free” variance parameters to be estimated with NT data points is $T(T+1)/2$. This is appropriate, for example, in micropanel data like, e.g., household income surveys, where N is in the thousands but T is typically quite short.

Lastly, both estimators assume away any correlation between cross-sectional units. Recent work [7] has strived towards generalizing the error covariance matrix to including cross-sectional correlations; while relaxing the cross-units incorrelation hypothesis, such efforts nevertheless require different constraints to keep the number of covariance parameters manageable. In particular, Bai et al. [7] achieve limiting the number of correlation parameters to be estimated through banding and thresholding techniques.

2.3. Implementation

I now turn to the practical issue of implementing functions performing estimation of β according to the two above approaches in the R language. As observed before, the random effects model is a special case of GLS where the covariance matrix of the composite errors

is block-diagonal, and thus fully characterized by $\Omega_i, i = 1, \dots, N$. The first step in FGLS is estimation of $\hat{\Omega}$.

2.3.1. Random Effects Variance Components Estimation

In the standard case, two parameters characterize Ω : σ_c and σ_u . Various estimators for the variance of the two error components have been proposed in the literature (see [15]). Following, again, Wooldridge [10], a suitable estimator for σ_c^2 is the average of the empirical covariances in the upper (or lower) triangle, diagonal excluded, of the empirical Ω blocks given by the outer products of pooled OLS residuals $\hat{v}_{it}^{OLS}$ for any given i :

$$\hat{\sigma}_c^2 = \frac{1}{[NT(T-1)/2 - K]} \sum_{i=1}^N \sum_{t=1}^T \sum_{s=t+1}^T \hat{v}_{it}^{OLS} \hat{v}_{is}^{OLS} \quad (8)$$

σ_v^2 may be estimated from the residual variance of the pooled OLS regression and thus, by independence, the variance of the idiosyncratic error is $\sigma_u^2 = \sigma_v^2 - \sigma_c^2$.

I estimate the error variance components based on response variable y , model matrix X and group indices vector ind , employing the *ragged array* structure for applying a function individual-wise over the time dimension. Hence, I extract the number of groups N , K the number of regressors, and the total number of observations. Next, I estimate the pooled model by OLS in order to extract residuals and calculate their variance.

2.3.2. General GLS Covariance Estimation

In the general GLS case there are, in Equation (7), $T(T+1)/2$ covariances to be estimated by averaging over all the N empirical Ω blocks given by the outer product of the pooled OLS residuals vectors $\hat{v}_i^{OLS} = (\hat{v}_{i1}^{OLS}, \dots, \hat{v}_{iT}^{OLS})$:

$$\hat{\Omega} = \frac{1}{N} \sum_{i=1}^N \hat{v}_i^{OLS} \hat{v}_i^{OLS'} \quad (9)$$

Starting again from the pooled model residuals, estimation of $\hat{\Omega}$ is straightforward, making a tri-dimensional array of N empirical $\Omega_i = \hat{v}_i^{OLS} \hat{v}_i^{OLS'}$ matrices and averaging over the third dimension, *preserving* the first two.

2.3.3. Feasible GLS Step

Once $\hat{\Omega}$ and hence, from (4), the covariance matrix of errors $\hat{V}$ have been consistently estimated, model parameters can be estimated by feasible GLS as

$$\hat{\beta}_{FGLS} = (X' \hat{V}^{-1} X)^{-1} X' \hat{V}^{-1} y$$

which is the feasible counterpart of (6). In theory, this is a straightforward matrix operation. A plain translation in R would make use of Kronecker products:

$$\hat{V} = I_N \otimes \hat{\Omega} \quad (10)$$

making use of `solve(A,B)` for $A^{-1}B$ and `crossprod(A,B)` for $A'B$, as suggested in [16], for the sake of speed and stability.

In practice, plain computation of $\hat{\beta}$ has long been intractable even for moderately sized problems because of the need to invert the $NT \times NT$ $\hat{V}$ matrix; on the other hand, this is not needed, given the special block-diagonal structure of V . By the properties of Kronecker

products and more generally of block-diagonal matrices, as long as (4) holds, (6) may be written as

$$\hat{\beta}_{FGLS} = \left(\sum_{i=1}^N X_i' \Omega^{-1} X_i \right)^{-1} \sum_{i=1}^N X_i' \Omega^{-1} y_i \quad (11)$$

reducing the maximum dimensions of the matrices to be inverted to $T \times T$.

For the special case of the RE structure, a more specific transformation of (6) has been devised to reduce it to the OLS problem on transformed data:

$$\hat{\beta}_{FGLS} = (\tilde{X}' \tilde{X})^{-1} \tilde{X}' \tilde{y} \quad (12)$$

further reducing the maximum size of inversions to $K \times K$, and thus making use of efficient and readily available software routines. I review both in the following.

2.3.4. Summary of Computational Challenges

From a computational perspective, (either kind of) feasible GLS estimation poses challenges that grow rapidly with panel size and with the flexibility of the assumed covariance structure. Naive implementations based on explicit construction and inversion of the full covariance matrix quickly become impractical because both memory usage and computational cost scale poorly with the product NT . While the standard random effects model admits efficient transformations that reduce the problem to ordinary least squares, more general GLS specifications require handling repeated blockwise matrix operations and covariance estimation. Consequently, practical feasibility depends critically on exploiting the structure of the covariance matrix, avoiding unnecessary large-scale matrix inversions, and relying on efficient programming paradigms such as vectorization, list-based operations, and structured matrix representations.

In typical applications, memory requirements are typically not an issue provided that one does at least handle the blockwise structure of the covariance properly, i.e., RAM usage can be challenging only in naive implementations. Otherwise, as shown here, CPU usage will be the limiting factor. In turn, and unlike many implementations of maximum likelihood estimators (see e.g., [17]), numerical stability is hardly going to be an issue in the present context, which relies solely on matrix algebra operators, for the sample sizes considered here.

3. Computational Strategies for the GLS Step

Some different computational strategies are available for calculating $\hat{\beta}_{GLS}$ in plain R. The most widely used one is very convenient but less general, and in fact does not allow general GLS estimation. The latter two work with both estimators of Ω as well, slight differences arising only in the unbalanced case.

3.1. Quasi-Time Demeaning

The established solution to RE-FGLS estimation goes under the name of *quasi-time demeaning*, or λ -*demeaning*, because it is based on performing OLS on a transformation of the original data obtained subtracting a fraction λ of the time-mean of each individual, λ being in turn a function of the estimated variance components σ_c^2 and σ_u^2 .

It can be shown that, as far as GLS estimation of β is concerned, the model in Equation (1) with RE errors, as in Equations (4) and (5), is equivalent to

$$y_{it} - \lambda \bar{y}_i = (\mathbf{x}_{it} - \lambda \bar{\mathbf{x}}_i) \beta + (v_{it} - \lambda \bar{v}_i) \quad (13)$$

where $\lambda = 1 - [\sigma_u^2 / (\sigma_u^2 + T\sigma_c^2)]^{1/2}$, $\bar{y}$ and $\bar{\mathbf{x}}$ denote time-means of y and $\mathbf{x}$, and the disturbance $v_{it} - \lambda \bar{v}_i$ is homoskedastic and serially uncorrelated. Thus, the feasible RE estimate

for β may be obtained estimating $\hat{\lambda}$ and running an OLS regression on the transformed data (see [15] or [10]). This strategy, termed *quasi-time demeaning*, is very convenient computationally. No inversion of large matrices is needed any more: once data have been transformed, OLS estimation of model (13) may be carried out through fast precompiled least squares routines that are available in any statistical software package. In the case of R, a computational strategy based on quasi-time demeaning has the advantage of making use of the highly optimized `lm.fit()` function. It is convenient, for this purpose, to define a quasi-demeaning auxiliary function to be *applied* columnwise to X and y . The *ragged array* structure proves very useful in this context as a natural tool for calculating group means. The latter are replicated over every group by subscripting with repeated values. Estimation of the demeaned model (13) and of the covariance matrix of parameters follow.

Unfortunately, as observed, this well-known computational shortcut, which is used for RE estimation in the majority of econometric software including the “plm” R package [14], is specific to the RE structure and does not fit the general GLS case.

3.2. Block-GLS (Algebraic Simplification Approach)

As observed above, plain linear algebra suggests a general solution for the GLS problem. Not setting any bounds on the structure of Ω , the expression in Equation (11) allows for standard RE as well as general GLS estimation as defined by (4), (7) and (6), yet it exchanges inversion of the full $NT \times NT$ error covariance matrix for that of Ω , which is $T \times T$. Standard random effects as well as general GLS estimates for Ω can be plugged in, much in the same way: from an econometric viewpoint, the formulation in Equation (11) is appropriate as long as the errors are independent across individual units.

3.2.1. Plain Matrix Operations

The natural way to code the block-GLS procedure is by subsetting the relevant matrices and vectors through *for* loops, creating tri-dimensional arrays where the third dimension is the individual unit of observation: respectively, X_i and y_i for $i = 1, \dots, N$. The Ω submatrix is then calculated in the usual way, inverted (once for all) and employed in “blockwise” GLS calculations to obtain stacked “piles” of $A_i = X_i' \Omega^{-1} X_i$ and $B_i = X_i' \Omega^{-1} y_i$ submatrices, one for each individual. The summations in Equation (11) are then performed, *apply*-ing the `sum()` function over the individual dimension i , preserving the other two (t and either k or 1), obtaining $A = \sum_i A_i$ and $B = \sum_i B_i$. The usual expressions for $\hat{\beta}_{GLS} = A^{-1}B$ and $V(\hat{\beta}_{GLS}) = A^{-1}$ follow as in Equation (11); this is now an operation of much smaller scope, involving only $K \times K$ matrix inversions.

3.2.2. List Operations

“List operators” are known to be much faster than *for* loops in R, mainly because they are implemented in optimized internal (C-level) code, which reduces the overhead of repeated interpretation that occurs in R’s high-level loop constructs. Each iteration of a *for* loop is handled by the R interpreter, adding overhead, whereas `lapply()` delegates the iteration to faster, compiled routines. Additionally, *apply* functions encourage a functional programming style that avoids dynamically growing objects inside loops—a common source of inefficiency due to repeated memory reallocation. They also make it easier for R to optimize memory usage and sometimes enable better use of vectorization under the hood. While modern R has improved loop performance, `lapply()` and similar functions still tend to be more concise, less error-prone, and often faster in practice, especially for large or complex data operations.

Writing the block-GLS procedure using list operators requires some adaptation but in the end becomes quite natural as well. For example, the “ragged array” structure of panel data (vectors with a grouping index attached) simplifies the making of n empirical variance

submatrices $\hat{\Omega}_i = \hat{v}_i \hat{v}_i'$ in one elegant line, *tapply*-ing the outer product function to each residuals vector $\hat{v}_i$. The averaging over elements in the upper triangle of Ω is then applied listwise, yielding $\hat{\sigma}_\mu^2$.

The GLS step is also rewritten, preparing a list of N $(X_i; y_i)$ couplets that will hold the submatrices of each individual unit: the former as a *data.frame* object, the latter as a vector. Subsetting the “ragged array” y can be done through the familiar *tapply()* operator, while for X , I make use of the *split()* function. *Map()*-ping them together produces the final result needed.

After calculating $\hat{\Omega}$ as above (and its inverse $\hat{\Omega}^{-1}$ once for all, also as above), two specialized GLS “list operators” for the operations $A_i = X_i' \Omega^{-1} X_i$ and $B_i = X_i' \Omega^{-1} y_i$ can be defined and seamlessly applied.

Transforming the lists back to three-dimensional arrays, respectively $T \times K \times N$ and $T \times 1 \times N$, makes it possible to proceed further, obtaining $A = \sum_i A_i$ and $B = \sum_i B_i$ as in the previous case, obtaining $\hat{\beta}_{GLS} = A^{-1}B$ and $V(\hat{\beta}_{GLS}) = A^{-1}$.

3.2.3. Task Parallelization

The problem outlined above of computing the feasible GLS panel estimator can be easily parallelized on the basis of the block-by-block inversion strategy. This belongs in fact to a class of problems known as *embarrassingly parallel*, that is to say, in a nutshell, that it can easily be divided into many self-contained smallest tasks which do not need to communicate between them until the results of each are rounded up by the main procedure. This is evidently the case of the separate matrix inversions needed for the computation of (11), which may be distributed across processors within either a CPU or a computing network.

List operators allow for the simplest parallelization solution: once performance-critical snippets have been written in this fashion, exploiting parallel operation is as simple as substituting *lapply()* with *mclapply()* from the *parallel* package. In the following, I will compare the performance of executing the list-based implementation serially (on just one processor) or distributing the performance-critical GLS operations across multiple processors.

3.3. Object-Oriented Approach, Using the *bdsmatrix* Class

The R environment has general facilities for fast matrix computation based on object orientation: particular types of matrices (symmetric, sparse etc.) are assigned the relevant class and the additional information on structure is used in the computations, sometimes with dramatic effects on performance: see [16] and the accompanying documentation to packages *Matrix* [18] and *SparseM* [19], and a usage case in a related field of panel data econometrics—spatial panels—in [17] (Section 5.1.3). In our case, some linear algebra routines originally available in the R package *kinship* [20] and now in package *bdsmatrix* [21] which exploit the particular block-diagonal and symmetric structure of $\hat{V}$ make it possible to implement a fast and reliable full-matrix solution to problems of any practically relevant size.

The $\hat{V}$ matrix is constructed as an object of class *bdsmatrix* by supplying “the vector of sizes for the matrices on the diagonal and the contents of the diagonal blocks, strung out as a vector” [20]. The peculiar properties of this matrix class are used for efficiently storing the object in memory and then by ad hoc versions of the *solve()* and *crossprod()* methods, dramatically reducing computing times and memory usage. In the following code (balanced case), the same $\hat{\Omega}$ covariance block is replicated N times along the diagonal. The resulting matrix is then used “the naive way” as in (6) to compute $\hat{\beta}$. Despite being the easiest to program, these solutions prove very speedy and stable. This is the approach currently used for the GGLS estimator in the “plm” R package Croissant and Millo [14].

4. Computing Timings and Comparison

In the following, I compare the computational speed of the above implementations on typical examples of “large” panels, i.e., where $N \gg T$, as could arise in microeconomic datasets. Performance tests have been conducted on an AMD Ryzen 7700 with 32 GB RAM, on the Linux operating system (Ubuntu Unity 7.7), simulating two sets of samples: one combining a quite large cross-sectional sample ($N = 10,000, 20,000, 30,000, 50,000, 100,000$) with a short temporal one ($T = 5, 10$), the second combining a cross-sectional sample size a tenth of the former ($N = 1000, 2000, 3000, 5000, 10,000$) with a much larger temporal one ($T = 30, 100$). The first choice in particular is typical of microeconometric panel data.

Results are reported in Tables 1–8: in particular, Tables 1, 2 and 3, 4 are dedicated to the standard RE estimator applied, respectively, to the wider and shorter ($T = 5, 10$), and to the longer panel ($T = 30, 100$). Tables 5, 6 and, respectively, Tables 7, 8 report the results for the GGLS, in the same order.

4.1. Simulation Design

Data for the timing exercise are generated according to the standard RE model described in Equations (1) and (2), with two regressors. In matrix form,

$$\begin{aligned} \mathbf{y} &= \beta \mathbf{X} + \mathbf{v} \\ \mathbf{v} &= (\mathbf{c} \otimes \mathbf{I}_T) + \mathbf{u} \end{aligned} \quad (14)$$

where $\mathbf{y}$, $\mathbf{v}$ and $\mathbf{u}$ are $NT \times 1$ vectors, $\mathbf{c}$ is $N \times 1$, $\mathbf{X}$ is a $NT \times 3$ design matrix and $\mathbf{I}_T$ is an identity matrix of dimension T . The regressors are generated as $x_1 \sim \text{Unif}[-0.5, 0.5]$, $x_2 \sim N(0, 1)$; the individual effects $\mathbf{c} \sim N(0, 1)$; and the idiosyncratic errors $\mathbf{u} \sim N(0, 1)$. In accordance with the RE specification, regressors and individual effects are drawn independently.

Given that, on the one hand, RE can be seen as a special case of GGLS, so that the latter is still an appropriate estimator for RE, and that, on the other hand, computing times can depend on the specific structure of the data, in order to facilitate meaningful comparisons, the same design is used for both RE and GGLS exercises.

Table 1. Timings in seconds, $N \times 5$. AMD Ryzen 7700, 32 GB RAM. Top to bottom, object-oriented solution (bdsmatrix); standard RE (quasi-time demeaning); block-GLS, using submatrices; block-GLS, using list operators, serial; idem, parallelized on eight cores. For comparison (italics): production implementation in the R package *plm*; Stata SE (serial execution), Stata MP (parallel execution on two cores) and Gretl.

	10,000	20,000	30,000	50,000	100,000
bdsmatrix	0.97	2.90	7.84	25.41	110.05
quasi-t.d.	0.88	3.15	8.17	26.34	101.80
bGLS.matrix	1.98	8.64	23.91	65.24	298.81
bGLS.list	0.78	1.65	2.75	4.51	9.98
bGLS.listx8	0.79	1.28	1.89	2.85	5.65
<i>R plm</i>	0.16	0.43	0.52	1.39	2.83
<i>Stata SE</i>	0.22	0.40	0.51	0.78	1.57
<i>Stata MP (x2)</i>	0.16	0.20	0.30	0.40	0.82
<i>Gretl</i>	0.01	0.02	0.03	0.05	0.12

Table 2. Timings in seconds, $N \times 10$. See previous Table for details.

	10,000	20,000	30,000	50,000	100,000
bdsmatrix	1.84	8.15	18.99	56.08	251.82
quasi-t.d.	1.81	8.13	18.12	44.90	196.76
bGLS.matrix	4.84	22.63	50.34	147.89	569.56
bGLS.list	0.86	1.86	2.91	5.13	11.90
bGLS.listx8	0.58	1.14	1.85	3.05	8.19
<i>R plm</i>	0.22	0.53	0.95	2.50	4.83
<i>Stata SE</i>	0.32	0.57	0.78	1.24	2.56
<i>Stata MP (x2)</i>	0.19	0.34	0.45	0.72	1.47
<i>Gretl</i>	0.02	0.04	0.07	0.12	0.26

Table 3. Timings in seconds, $N \times 30$. See previous Table for details.

	1000	2000	3000	5000	10,000
bdsmatrix	0.10	0.29	0.65	1.80	6.97
quasi-t.d.	0.10	0.25	0.55	1.45	5.14
bGLS.matrix	0.16	0.59	1.34	3.93	16.42
bGLS.list	0.09	0.18	0.29	0.53	1.15
bGLS.listx8	0.25	0.33	0.51	0.64	1.10
<i>R plm</i>	0.08	0.14	0.19	0.33	0.93
<i>Stata SE</i>	0.10	0.17	0.25	0.35	0.65
<i>Stata MP (x2)</i>	0.09	0.12	0.18	0.23	0.47
<i>Gretl</i>	0.01	0.01	0.02	0.03	0.06

Table 4. Timings in seconds, $N \times 100$. See previous Table for details.

	1000	2000	3000	5000	10,000
bdsmatrix	0.86	2.17	3.77	8.20	26.57
quasi-t.d.	0.33	0.82	1.72	4.44	15.09
bGLS.matrix	0.56	1.90	4.04	10.80	48.90
bGLS.list	0.20	0.42	0.64	1.20	2.87
bGLS.listx8	0.37	0.58	0.74	1.42	2.72
<i>R plm</i>	0.20	0.48	0.89	1.96	3.77
<i>Stata SE</i>	0.25	0.46	0.61	0.96	1.97
<i>Stata MP (x2)</i>	0.17	0.32	0.41	0.66	1.34
<i>Gretl</i>	0.02	0.04	0.06	0.10	0.23

Table 5. Timings in seconds, $N \times 5$. AMD Ryzen 7700, 32 GB RAM. Top to bottom, object-oriented solution (bdsmatrix); block-GLS, using submatrices; block-GLS, using list operators, serial; idem, parallelized on eight cores; production implementation in the package *plm*.

	10,000	20,000	30,000	50,000	100,000
bdsmatrix	0.91	3.85	8.60	21.36	85.43
bGLS.matrix	2.60	11.10	24.07	62.35	259.44
bGLS.list	0.80	3.29	4.28	7.45	12.81
bGLS.listx8	0.94	2.04	3.41	5.65	8.76
<i>R pgls</i>	0.11	0.19	0.30	0.53	1.17

Table 6. Timings in seconds, $N \times 10$. See previous Table for details.

	10,000	20,000	30,000	50,000	100,000
bdsmatrix	2.11	6.05	15.52	43.64	189.19
bGLS.matrix	4.14	16.66	43.07	125.09	572.35
bGLS.list	0.77	1.73	4.57	9.78	17.84
bGLS.listx8	2.32	3.43	5.42	7.90	15.19
<i>R pggls</i>	0.16	0.33	0.53	1.06	3.46

Table 7. Timings in seconds, $N \times 30$. See previous Table for details.

	1000	2000	3000	5000	10,000
bdsmatrix	0.13	0.36	0.71	1.93	5.93
bGLS.matrix	0.20	0.58	1.46	3.83	14.30
bGLS.list	0.20	0.67	1.01	1.95	3.88
bGLS.listx8	0.37	0.86	0.73	1.06	3.38
<i>R pggls</i>	0.06	0.12	0.16	0.28	1.14

Table 8. Timings in seconds, $N \times 100$. See previous Table for details.

	1000	2000	3000	5000	10,000
bdsmatrix	0.96	2.21	3.57	6.88	20.39
bGLS.matrix	0.71	2.28	4.69	12.54	45.02
bGLS.list	3.00	8.33	14.67	28.58	56.96
bGLS.listx8	2.77	7.03	8.60	12.84	36.41
<i>R pggls</i>	0.49	1.17	2.06	3.07	5.91

4.2. Discussion

In the following, I first compare the performance of the above-described proof-of-concept estimator functions between them. Then, I set their computing times next to those of production-quality, highly optimized routines from the R environment (*plm*), from a leading commercial software (Stata) and from a popular open-source project (Gretl). This latter step is possible only for RE, because GGLS as defined in Equation (7) is not implemented in either Stata19 or Gretl.

Comparison of Computational Strategies

Relying on advanced infrastructure features for matrix algebra makes the bdsmatrix strategy competitive with (plain implementations of) quasi-time demeaning; both are feasible in little time at least until data sample size remains moderate. The block-by-block inversion strategy, though in principle computationally far less demanding in terms of matrix inversions with respect to bdsmatrix (the matrix to be inverted is $T \times T$ instead of $NT \times NT$), is definitely slower if implemented the natural way, through matrix subsetting and for loops. Resorting to list operators, though, dramatically improves computing times, in particular for the most relevant case: the larger, shorter panel dataset with $N = 10,000$ to 100,000.

The list operator-based solution keeps its advantage in the potentially less relevant case of the “longish micropanel” with $N = 1000$ to 10,000 and $T = 30$ or 100, being still one order of magnitude faster than the bdsmatrix solution.

The gains from parallelization are not substantial, never reaching a factor of 2. The reason is perhaps in the limited performance drag of the operations that have been parallelized. For example, operations where each $\hat{\Omega}_i$ submatrix has to be processed separately—as may happen in the calculation of sandwich estimators (see, e.g., [22])—are a more promising candidate for optimization along these lines.

GGLS estimation confirms the previous results up to the “longish” panel case, where in the more extreme dimension case $T = 100$ —less likely to be encountered in practice—the list-based block-GLS solution is beaten by bdsmatrix. The way the computing times of RE and GGLS estimators scale with the time dimension of the panel is depicted in Figures 1 and 2.

Lastly, RAM usage is generally modest, although of course it is slightly increased under shared resource usage in the parallelized case, and never a limiting factor for today’s standards.

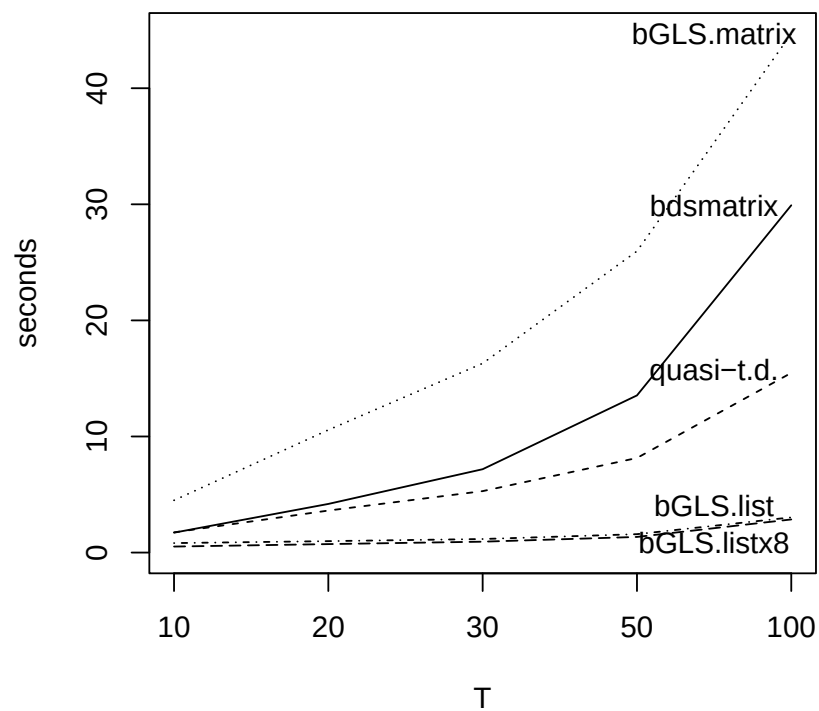


Figure 1. Scaling of computing times with T , $N = 10,000$, RE.

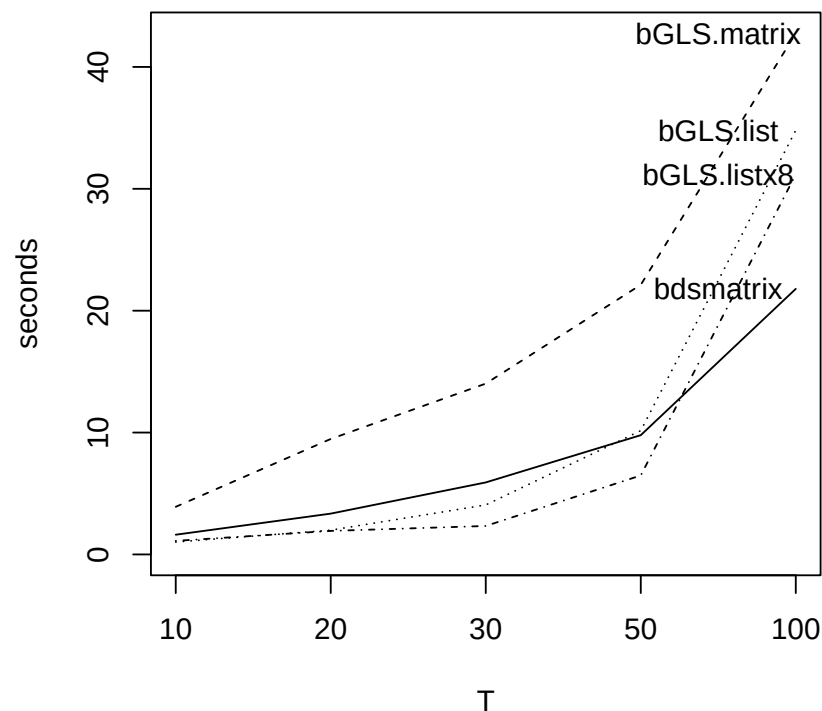


Figure 2. Scaling of computing times with T , $N = 10,000$, GGLS.

4.3. Comparison with Other Software

I compare the RE timings with the production version of the *plm* package for R [14] and with those from two different econometric software suites: Stata, arguably the world standard in the econometric analysis of panel data, and Gretl, a popular—and notoriously efficient—open-source econometric library [23].

I consider two editions of Stata: Stata 19 SE (standard edition, running serially) and Stata 19 MP (multiprocessor edition, in my case running on two cores), and the current stable version for Linux Ubuntu of Gretl (Version 2023c).

Stata is in principle an interpreted language, but most commands are indeed calling precompiled routines, as happens with many performance-critical components in R as well. In particular, the *xtreg* command for standard panel model estimation employed here is arguably one of the most popular, and therefore precompiled and optimized. In turn, Gretl, implemented in C, is compiled and well-known within the econometric community for its speed, accuracy and stability [24]. Lastly, the *plm* implementation is based on quasi-time demeaning, and written in plain R, although it obviously uses the precompiled R routine for OLS estimation accessible through the *lm.fit* function. One must keep in mind that this production version is highly optimized with respect to the relatively crude implementations I consider here, which are meant mostly for comparing between them the efficiency of the estimation step, given that a thorough optimization of all the procedure from interface to computing is out of the scope of this note.

Both non-R programs have their own scripting languages, including timing features. I employ scripts for importing the simulated datasets from R, converted into Stata's native *.dta* format, setting the panel features as necessary, and performing estimation. Obviously, only the estimation step is timed.

Impressively, the *plm* versions are only slightly less efficient than Stata. List operator-based versions of the R code, although marginally slower, remain within the same order of magnitude of Stata. The very efficient Gretl software is instead one order of magnitude faster. All things considered, the “proof of concept” versions presented here fare surprisingly well when based on list operations, while plain versions of the block-GLS and even of the quasi-time demeaning approaches are far less efficient.

As far as GGLS is regarded, as observed, only the comparison with the *plm* implementation (function *pggls*) is possible. Based on a combination of *bdsmatrix* and list operations, *pggls* is up to one order of magnitude faster than the list-based proof of concept, almost two with respect to the *bds-matrix* one, demonstrating the importance of fine-tuning optimization. Simple list operations again show substantial gains with respect to the object-oriented matrix algebra alternative, but it is the combination of the two that makes the production code really efficient.

Parallelization

The improvement from employing the multiprocessor version of Stata (MP) in place of Stata SE seems to scale linearly across all experiments, at least in the two-core version available to me. The speedup factor is consistently between 1.5 and 2, showing widespread code optimization along the whole procedure. By contrast, the comparatively disappointing result of R employing eight cores was to be expected, given the admittedly crude level of optimization of the implementation I consider here, which parallelizes only the single, although performance-critical, GLS step. As observed above, settings where every individual diagonal block in the covariance matrix has to be inverted separately are more likely to yield substantial gains.

Lastly, as observed above, there is the possibility of exploiting parallelization on a deeper level by installing on the system a BLAS optimized for parallel operation. A comparison along these lines is left for future work.

5. Conclusions

This paper has examined the computational aspects of feasible generalized least squares (FGLS) estimation for panel data models, with particular attention to large datasets characterized by a dominant cross-sectional dimension. While the econometric theory underlying both random effects and general GLS estimators is well-established, their practical applicability depends critically on how the associated computations are implemented.

The results show that computational feasibility is not an inherent limitation of the estimators themselves, but rather a consequence of the chosen implementation strategy. Naïve approaches based on full-matrix operations or explicit looping quickly become impractical as sample size grows. In contrast, block-wise formulations exploiting the structure of the covariance matrix, combined with vectorized and list-based operations, allow substantial reductions in computational cost. In particular, implementations based on functional programming constructs in R prove to be both efficient and scalable, especially in the empirically relevant case of large (N) and small (T).

From a practical standpoint, the findings suggest a clear hierarchy of methods. Block-GLS approaches implemented via list operators offer the best mix of generality and performance. Object-oriented matrix representations provide a viable alternative, combining ease of implementation with competitive speed, particularly in moderately sized problems. The standard quasi-time demeaning solution is still a viable solution for standard random effects models, but is inherently limited to that specific structure; and it is, respectively, outperformed and matched by the above more general alternatives.

The analysis also shows that parallelization can be applied naturally to block-wise computations, although the performance gains are moderate in the scenarios considered. This suggests that further improvements are more likely to arise from algorithmic and data-structure optimizations than from parallel execution alone, at least for typical panel dimensions.

Overall, the paper highlights that modern high-level programming environments provide sufficient tools to make a broader class of panel data estimators operational in practice. This expands the set of feasible modeling choices available to applied researchers, particularly in contexts where more flexible error structures are theoretically justified but have traditionally been avoided for computational reasons.

Future research could extend this analysis in several directions, including the study of memory usage and numerical stability in very large-scale settings, the integration with sparse matrix techniques, and the application of these methods to real-world datasets with complex patterns of unbalancedness and missing data.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The author declares no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

OLS	Ordinary least squares
GLS	Generalized least squares
FGLS	Feasible generalized least squares
GGLS	General generalized least squares
bGLS	Block-GLS (computational strategy)

References

1. R Core Team. *R: A Language and Environment for Statistical Computing*; R Foundation for Statistical Computing: Vienna, Austria, 2024.
2. Gaure, S. OLS with multiple high dimensional category variables. *Comput. Stat. Data Anal.* **2013**, *66*, 8–18. [\[CrossRef\]](#)
3. Gaure, S. lfe: Linear Group Fixed Effects. *R J.* **2013**, *5*, 104. [\[CrossRef\]](#)
4. Bergé, L.R.; Butts, K.; McDermott, G. Fast and user-friendly econometrics estimations: The R package fixest. *arXiv* **2026**, arXiv:2601.21749.
5. Magnus, J.R.; De Luca, G. Weighted-average least squares (WALS): A survey. *J. Econ. Surv.* **2016**, *30*, 117–148.
6. De Luca, G.; Magnus, J.R. Bayesian model averaging and weighted-average least squares: Equivariance, stability, and numerical issues. *Stata J.* **2011**, *11*, 518–544. [\[CrossRef\]](#)
7. Bai, J.; Choi, S.H.; Liao, Y. Feasible generalized least squares for panel data with cross-sectional and serial correlations. *Empir. Econ.* **2021**, *60*, 309–326.
8. Eddelbuettel, D.; Balamuta, J.J. Extending R with C++: A Brief Introduction to Rcpp. *Am. Stat.* **2018**, *72*, 28–36. [\[CrossRef\]](#)
9. Greene, W.H. *Econometric Analysis*; Prentice Hall: Hoboken, NJ, USA, 2003.
10. Wooldridge, J.M. *Econometric Analysis of Cross Section and Panel Data*; MIT Press: Cambridge, MA, USA, 2010.
11. Parks, R. Efficient Estimation of a System of Regression Equations When Disturbances Are Both Serially and Contemporaneously Correlated. *J. Am. Stat. Assoc.* **1967**, *62*, 500–509. [\[CrossRef\]](#)
12. Driscoll, J.C.; Kraay, A.C. Consistent Covariance Matrix Estimation with Spatially-Dependent Panel Data. *Rev. Econ. Stat.* **1998**, *80*, 549–560. [\[CrossRef\]](#)
13. Zellner, A. An Efficient Method of Estimating Seemingly Unrelated Regressions and Tests of Aggregation Bias. *J. Am. Stat. Assoc.* **1962**, *57*, 500–509. [\[CrossRef\]](#)
14. Croissant, Y.; Millo, G. *Panel Data Econometrics with R*; Wiley Online Library: Hoboken, NJ, USA, 2019.
15. Baltagi, B.H. *Econometric Analysis of Panel Data*; John Wiley and Sons: Hoboken, NJ, USA, 1995.
16. Bates, D. Least Squares Calculations in R. *R News* **2004**, *4*, 17–20.
17. Millo, G. Maximum likelihood estimation of spatially and serially correlated panels with random effects. *Comput. Stat. Data Anal.* **2014**, *71*, 914–933. [\[CrossRef\]](#)
18. Bates, D.; Maechler, M. *Matrix: A Matrix Package for R*, R package version 0.995-5; R Foundation for Statistical Computing: Vienna, Austria, 2006.
19. Koenker, R.; Ng, P. *SparseM: Sparse Linear Algebra*, R package version 0.68; R Foundation for Statistical Computing: Vienna, Austria, 2003.
20. Atkinson, B.; Therneau, T. *kinship: Mixed-Effects Cox Models, Sparse Matrices, and Modeling Data from Large Pedigrees*, R package version 1.1.0-11; R Foundation for Statistical Computing: Vienna, Austria, 2006.
21. Therneau, T. *bdsmatrix: Routines for Block Diagonal Symmetric Matrices*, R package version 1.3-7; R Foundation for Statistical Computing: Vienna, Austria, 2024.
22. Millo, G. Robust standard error estimators for panel models: A unifying approach. *J. Stat. Softw.* **2017**, *82*, 1–27. [\[CrossRef\]](#)
23. Baiocchi, G.; Distaso, W. GRETL: Econometric software for the GNU generation. *J. Appl. Econom.* **2003**, *18*, 105–110. [\[CrossRef\]](#)
24. Yalta, A.T.; Yalta, A.Y. GRETL 1.6. 0 and its numerical accuracy. *J. Appl. Econom.* **2007**, *22*, 849–854. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.