Quantifying the Computational Advantage of Forward Orthogonal Deviations

Robert F. Phillips*
Department of Economics
George Washington University

August 2018

Abstract

Under suitable conditions, one-step generalized method of moments (GMM) based on the first-difference (FD) transformation is numerically equal to one-step GMM based on the forward orthogonal deviations (FOD) transformation. However, when the number of time periods (T) is not small, the FOD transformation requires less computational work. This paper shows that the computational complexity of the FD and FOD transformations increases with the number of individuals (N) linearly, but the computational complexity of the FOD transformation increases with T at the rate T^4 increases, while the computational complexity of the FD transformation increases at the rate T^6 increases. Simulations illustrate that calculations exploiting the FOD transformation are performed orders of magnitude faster than those using the FD transformation. The results in the paper indicate that, when one-step GMM based on the FD and FOD transformations are the same, Monte Carlo experiments can be conducted much faster if the FOD version of the estimator is used.

Keywords: forward orthogonal demeaning; forward orthogonal deviations; first differencing; computational complexity

1 Introduction

In a seminal paper, Arellano and Bond (1991) introduced a now well-known one-step generalized method of moments (GMM) panel data estimator. The estimator relies on first-differencing the observations — the first-difference (FD) transformation — and then applying GMM. Arellano and Bover (1995) suggested another transformation could be used. They showed that, under suitable conditions, GMM is invariant to how the data are transformed, and they introduced the forward orthogonal deviations (FOD) transformation to the panel data literature. Moreover, Arellano and Bover (1995) noted there is a computational advantage to using the FOD transformation when the number of columns in the instrument matrix is large. However, to date, there appears to be no published evidence illustrating how much of a computational advantage is conferred by using the FOD transformation.

The purpose of this paper is to fill that gap. I show how the computational complexity — the amount of computational worked required — of the FD and FOD transformations increase with the length of the time series (T) and the number of individuals (N). The results reveal that, even

---

*Address: 2115 G Street NW, Suite 340, Washington DC 20052; Telephone: 202-994-8619; Email: rphil@gwu.edu

when $\lim(T/N) = 0$,[1] computational complexity is affected much more by the size of T than by the size of N. Furthermore, the FD transformation's computational complexity increases with T at a much faster rate than the rate of increase in the FOD transformation's computational complexity. Consequently, when T is not small, the FOD transformation is computationally faster — orders of magnitude faster — than the FD transformation. A practical implication of this finding is that computationally intensive work, such as Monte Carlo simulations, can be performed much faster by relying on the FOD rather than the FD transformation.

2 The computational complexity of the FD and FOD transformations

In order to compare the computational complexity of the FD and FOD transformations, a simple case was considered — the first-order autoregressive (AR(1)) panel data model. The model is

$$y_{it} = \delta y_{i,t-1} + \eta_i + v_{it}, \tag{1}$$

with y_{i0} taken to be the first available observation. If the v_{it}s are uncorrelated, then, for instruments, one might use $z_{i1} = y_{i0}$, $\boldsymbol{z}'_{i2} = (z_{i1}, y_{i1})$, $\boldsymbol{z}'_{i3} = (\boldsymbol{z}'_{i2}, y_{i2})$, and so on up to $\boldsymbol{z}'_{i,T-1} = (\boldsymbol{z}'_{i,T-2}, y_{i,T-2})$. For this choice of instruments, one-step GMM based on the FOD transformation is numerically equivalent to one-step GMM based on the FD transformation (see, e.g., Hayakawa and Nagata, 2016).

However, although numerically the same, the two transformations are not computationally the same. To see this, consider first one-step GMM estimation of the AR(1) panel data model using the FD transformation. The first-difference transformation matrix is

$$\boldsymbol{D} = \begin{pmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 1 \end{pmatrix}.$$

The one-step GMM estimator based on this transformation can be written as follows: Let $\boldsymbol{y}_i = (y_{i1}, \ldots, y_{iT})'$, $\boldsymbol{y}_{i,-1} = (y_{i0}, \ldots, y_{i,T-1})'$, $\tilde{\boldsymbol{y}}_i = \boldsymbol{D}\boldsymbol{y}_i$, and $\tilde{\boldsymbol{y}}_{i,-1} = \boldsymbol{D}\boldsymbol{y}_{i,-1}$ $(i = 1, \ldots, N)$. Also, let $\boldsymbol{Z}_i$ denote a block-diagonal matrix with the vector $\boldsymbol{z}'_{it}$ in its tth diagonal block $(t = 1, \ldots, T-1$, $i = 1, \ldots, N)$. Moreover, let $\tilde{\boldsymbol{s}} = \sum_i \boldsymbol{Z}'_i \tilde{\boldsymbol{y}}_i$, $\tilde{\boldsymbol{s}}_{-1} = \sum_i \boldsymbol{Z}'_i \tilde{\boldsymbol{y}}_{i,-1}$, $\boldsymbol{G} = \boldsymbol{D}\boldsymbol{D}'$, and $\boldsymbol{A}_N = \sum_i \boldsymbol{Z}'_i \boldsymbol{G} \boldsymbol{Z}_i$. Finally, set $\tilde{\boldsymbol{a}} = \tilde{\boldsymbol{s}}'_{-1} \boldsymbol{A}_N^{-1}$. Then the one-step FD GMM estimator is given by

$$\widehat{\delta}_D = \frac{\tilde{\boldsymbol{a}}\tilde{\boldsymbol{s}}}{\tilde{\boldsymbol{a}}\tilde{\boldsymbol{s}}_{-1}}. \tag{2}$$

(Arellano and Bond, 1991).

For large T, the formula in (2) is computationally expensive. A measure of computational cost or work is computational complexity, which is the number of floating point operations or flops required.[2] The conclusion from counting up the number of flops required to compute $\widehat{\delta}_D$ is provided in Lemma 1.

---

[1]In this case, one-step GMM, based on all available instruments, has no asymptotic bias (Alvarez and Arellano, 2003).

[2]A flop consists of an addition, subtraction, multiplication, or division.

Lemma 1 *Given $\boldsymbol{D}$, $\boldsymbol{y}_i$, $\boldsymbol{y}_{i,-1}$ and $\boldsymbol{Z}_i$ $(i = 1, \ldots, N)$, the number of flops required by the FD formula in (2) increase with N linearly, for given T, and increase with T at the rate T^6 increases, for given N.*

Appendix A.1 provides the flop counts that verify Lemma 1.

Lemma 1 shows that, even when T is much smaller than N, it can be much more important than N in determining the amount of computational work — and hence time — it takes to obtain an estimate via the formula in (2). A substantial contribution to the amount of work required is the computation of $\boldsymbol{A}_N$ and then inverting it. Appendix A.1 shows that the number of flops required to calculate $\boldsymbol{A}_N$ is on the order of $\mathrm{O}(NT^5)$. Moreover, the work required to invert $\boldsymbol{A}_N$ increases even faster with T. Standard matrix inversion methods require on the order of another $\mathrm{O}(T^6)$ flops to compute $\boldsymbol{A}_N^{-1}$.

On the other hand, the FOD transformation does not require inverting $\boldsymbol{A}_N$. This fact makes it more efficient computationally when T is not small.

To see how much more efficient the FOD transformation is, consider again the AR(1) model in (1), and set $\ddot{\boldsymbol{y}}_i = \boldsymbol{F}\boldsymbol{y}_i$ and $\ddot{\boldsymbol{y}}_{i,-1} = \boldsymbol{F}\boldsymbol{y}_{i,-1}$, where $\boldsymbol{F}$ is the FOD transformation matrix given by

$$
\boldsymbol{F} = \operatorname{diag}\left(\left(\frac{T-1}{T}\right)^{1/2}, \left(\frac{T-2}{T-1}\right)^{1/2}, \ldots, \left(\frac{1}{2}\right)^{1/2}\right)
$$

$$
\times \begin{pmatrix}
1 & -\frac{1}{T-1} & -\frac{1}{T-1} & \cdots & -\frac{1}{T-1} & -\frac{1}{T-1} & -\frac{1}{T-1} \\
0 & 1 & -\frac{1}{T-2} & \cdots & -\frac{1}{T-2} & -\frac{1}{T-2} & -\frac{1}{T-2} \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
0 & 0 & 0 & & 1 & -\frac{1}{2} & -\frac{1}{2} \\
0 & 0 & 0 & & 1 & -1
\end{pmatrix}
$$

(see Arellano and Bover, 1995). Also, let $\ddot{y}_{it}$ and $\ddot{y}_{i,t-1}$ denote the tth entries in $\ddot{\boldsymbol{y}}_i$ and $\ddot{\boldsymbol{y}}_{i,-1}$. Then the FOD version of the one-step GMM estimator is

$$
\widehat{\delta}_F = \frac{\sum_{t=1}^{T-1} \ddot{\boldsymbol{a}}_t \ddot{\boldsymbol{s}}_t}{\sum_{t=1}^{T-1} \ddot{\boldsymbol{a}}_t \ddot{\boldsymbol{s}}_{t-1}}. \tag{3}
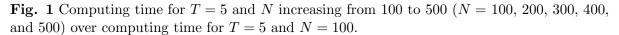$$

where $\ddot{\boldsymbol{s}}_t = \sum_i \boldsymbol{z}_{it} \ddot{y}_{it}$, $\ddot{\boldsymbol{s}}_{t-1} = \sum_i \boldsymbol{z}_{it} \ddot{y}_{i,t-1}$, $\ddot{\boldsymbol{a}}_t = \ddot{\boldsymbol{s}}_{t-1}' \boldsymbol{S}_t^{-1}$, and $\boldsymbol{S}_t = \sum_i \boldsymbol{z}_{it} \boldsymbol{z}_{it}'$ (see, e.g., Alvarez and Arellano, 2003).
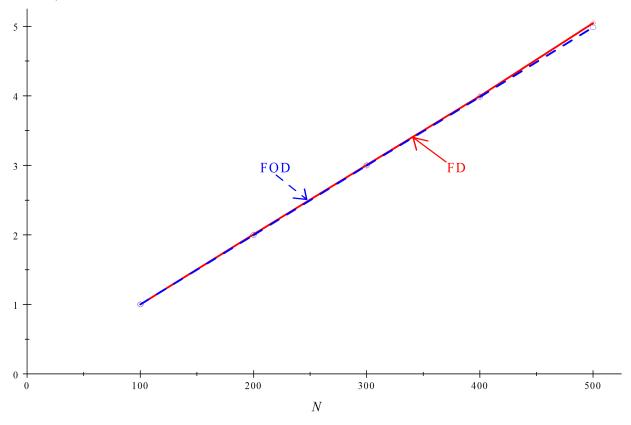
The formula in (3) replaces computing and then inverting one large matrix — the matrix $\boldsymbol{A}_N$ — with computing and inverting several smaller matrices — the matrices $\boldsymbol{S}_t$ $(t = 1, \ldots, T-1)$. The computational savings of this alternative approach are summarized in Lemma 2.

Lemma 2 *Given $\boldsymbol{F}$, $\boldsymbol{y}_i$, $\boldsymbol{y}_{i,-1}$, and $\boldsymbol{Z}_i$ $(i = 1, \ldots, N)$, the number of flops required by the FOD formula in (3) increase with N linearly, for given T, and increase with T at the rate T^4 increases, for given N.*

The flop counts are provided in Appendix A.2.

Lemmas 1 and 2 show that the computational complexity of one-step GMM based on both the FOD and FD transformations increases much faster with T than with N. But the number of flops increase with T at a much slower rate for the FOD transformation. This finding indicates that, for large T, computing time will be orders of magnitude faster for the FOD transformation than for the FD transformation. This conjecture is explored in the next section.

Fig. 1 Computing time for $T = 5$ and N increasing from 100 to 500 ($N = 100, 200, 300, 400,$ and 500) over computing time for $T = 5$ and $N = 100$.
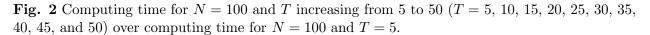

3 An illustration

In order to illustrate the reductions in computing time from using the FOD transformation rather than differencing, some experiments were conducted. For all of the experiments, observations on y_{it} were generated according to the AR(1) model
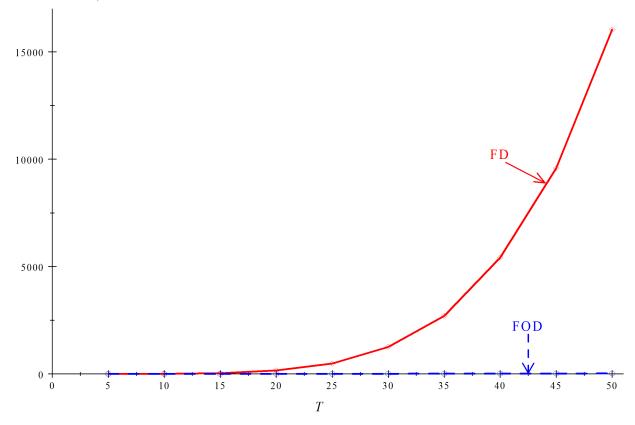
$$y_{it} = \delta y_{i,t-1} + \eta_i + v_{it}, \qquad t = -49, \ldots, T, \ i = 1, \ldots, N,$$

with $y_{i,-50} = 0$. The value for δ was fixed at 0.5 for all experiments. The error components η_i and v_{it} were generated independently as standard normal random variables. The processes were started with $t = -50$ so that, for each i, the process was essentially stationary by time $t = 0$. As for the sample sizes, T was set to either five, 10, 15, 20, 25, 30, 35, 40, 45, or 50 whereas N was either 100, 200, 300, 400, or 500. After a sample was generated, start-up observations were discarded so that estimation was based on the $T + 1$ observations $y_{i0}, \ldots, y_{iT}$ for each i ($i = 1, \ldots, N$). Finally, one-step GMM estimates were calculated both with the FD formula in (2) and the FOD formula in (3). The estimates obtained from the two formulas were identical but how long it took to compute them differed.[3]

Figure 1 plots ratios of computing times as N increases holding T fixed. In Figure 1, I plot the time required to calculate GMM estimates for 100 independent samples of size $T = 5$ and $N = X$

---

[3]All computations were performed using GAUSS. To calculate elapsed times for the FOD and FD algorithms, the GAUSS command hsec was used to calculate the number of hundredths of a second since midnight before and after calculations were executed.

Fig. 2 Computing time for $N = 100$ and T increasing from 5 to 50 ($T = 5$, 10, 15, 20, 25, 30, 35, 40, 45, and 50) over computing time for $N = 100$ and $T = 5$.

($X = 100$, 200, 300, 400, and 500) over the time required to calculate that many one-step GMM estimates for $T = 5$ and $N = 100$, first using the FD formula in (2) and then using the FOD formula in (3). The solid line shows how computing time increases with N, holding T constant, using differencing, while the dashed line shows how it increases with N using forward orthogonal deviations.

The message of the figure is clear. Because the number of computations required to compute a GMM estimate increases linearly in N, regardless of whether differencing or the FOD transformation is used, so too does computing time. For example, in Figure 1, when N is doubled from 100 to 200, computing time approximately doubles, regardless of whether estimates are computed using the differencing or the FOD transformation. When N triples from 100 to 300, computing time approximately triples, and so on.

Figure 2 plots ratios of computing times as T changes, with N held fixed at 100. Specifically, it gives the time required to calculate GMM estimates for 100 independent samples of size $N = 100$ and $T = X$ ($X = 5$, 10, 15, 20, 25, 30, 35, 40, 45, and 50) over the time required to compute that many estimates for $N = 100$ and $T = 5$. As in Figure 1, the solid curve shows how computing time increases for the FD transformation, but now as T increases with N held fixed. The dashed curve shows how computing time increases as T increases using the FOD transformation.

Figure 2 shows that computing time does not increase linearly with T. Instead, when the FD transformation is used, computing time increases much faster than linearly. For example, if the differencing formula in (2) is used to calculate one-step GMM estimates, then, for $N = 100$, it takes about 5.5 times longer to calculate an estimate, on average, when $T = 10$ than when $T = 5$. Thus,

doubling T increases computing time by 5.5 times. When T increases from 5 to 15, a three-fold increase in T, it takes about 36 times longer to calculate an estimate. Finally, if we increase T ten-fold, from 5 to 50, it takes approximately $16,038$ times longer to compute GMM estimates if differencing is used (see Figure 2).

When we use the FOD formula in (3) to calculate GMM estimates, the relative increase in computing time is much less dramatic, but it is still not linear in T. In Figure 2, I also plotted the ratios of computing times required to calculate GMM estimates using the FOD formula in (3). For the FOD transformation, how relative computing time increases with T is indicated by the dashed line in the figure. The line hugs the horizontal axis in Figure 2 because the increase in computing time with T is of a much smaller order of magnitude when the FOD transformation is used than when the FD transformation is used. Specifically, for $N = 100$, the computing time given $T = 10$ is 2.2 times what it is when $T = 5$. Thus, doubling T leads to a bit more than double the computing time. However, a ten-fold increase in computing time — when T is increased from 5 to 50 — leads to computing time taking about 24 times longer to compute an estimate when the FOD transformation is used.

Although the computational complexity of the FOD transformation is not linear in T, it increases at a much slower rate in T than is the case when differencing is used. Consequently, as T increases, using the FOD formula for calculating GMM estimates leads to significant reductions in computing time relative to using the differencing formula for computing GMM estimates. Table 1 shows how large these reductions can be.

Table 1 reports time ratios. The ratios in Table 1 are the time required to compute estimates using the FD transformation over the time required to compute those same estimates using the FOD transformation for different values of T and N. For small T, the computations based on first differences are faster. For example, for $T = 5$, it takes about half the time to calculate an estimate using the FD formula in (2) rather than the FOD formula in (3). However, because computational complexity increases in T at a slower rate when the FOD transformation is used rather than differencing, the FOD method is faster — indeed, much faster — than differencing for larger values of T. For T as small as 10, the FOD transformation is faster, and, for $T = 50$, computations based on the FOD transformation are two orders of magnitude faster — specifically, over 300 times faster — than computations exploiting the FD transformation.

Table 1: Time to compute FD GMM estimates over time to compute FOD GMM estimates.

| | | | | | T | | | | | |
	5	10	15	20	25	30	35	40	45	50
$N = 100$	0.47	1.18	4.82	15.24	33.30	65.05	106.93	167.56	235.44	316.90
$N = 200$	0.47	1.20	4.86	15.62	33.62	61.60	104.07	162.72	236.71	319.03
$N = 300$	0.47	1.18	4.92	15.89	33.64	61.70	104.19	163.38	237.45	320.49
$N = 400$	0.47	1.19	5.19	15.97	33.89	61.55	104.39	163.49	240.04	321.96
$N = 500$	0.48	1.20	5.24	15.90	33.66	61.98	104.63	163.48	239.43	321.65

Note: Each estimate is based on 100 samples.

4   Summary and concluding remarks

This paper showed that the computational complexity of one-step GMM estimation based on the FD transformation increases with N linearly but it increases with T dramatically — at the rate T^6 increases. On the other hand, when the FOD transformation is used, computational complexity still increases with N linearly, but it increases with T at the rate T^4 increases. Simulation evidence provided in Section 3 showed that the reductions in computing time from use of the FOD instead of the FD transformation are dramatic when T is not small.

The fact that estimates can be computed so much faster with the FOD transformation implies that Monte Carlo simulations, and other computationally intensive procedures, can be conducted in a fraction of the time if the FOD transformation is used rather than the FD transformation. Consequently, Monte Carlo studies using large values of T and complicated models that may be prohibitively costly for GMM estimation based on the FD transformation may be feasible for GMM based on the FOD transformation.

Appendix A: Floating point operations for one-step GMM

A floating point operation (flop) is an addition, subtraction, multiplication, or division. This appendix shows how the number of flops required to calculate $\widehat{\delta}$ via the formulas in (2) and (3) depends on N and T.

To find the number of flops required to calculate $\widehat{\delta}$, the following facts will be used repeatedly throughout this appendix. Let $\boldsymbol{B}$, $\boldsymbol{E}$, and $\boldsymbol{H}$ be $q \times r$, $q \times r$, and $r \times s$ matrices, and let d be a scalar. Then $d\boldsymbol{B}$, $\boldsymbol{B} \pm \boldsymbol{E}$, and $\boldsymbol{BH}$ consist of qr, qr, and $qs\,(2r-1)$ flops, respectively (see Hunger, 2007).

Appendix A.1: Floating point operations using differencing

To calculate $\tilde{\boldsymbol{y}}_i = \boldsymbol{D}\boldsymbol{y}_i$ $(i = 1, \ldots, N)$, a total of $N\,(T-1)\,(2T-1)$ flops are needed. After the $\tilde{\boldsymbol{y}}_i$s are calculated, the number of flops required to compute $\tilde{\boldsymbol{s}} = \sum_i \boldsymbol{Z}_i'\tilde{\boldsymbol{y}}_i = (\boldsymbol{Z}_1', \ldots \boldsymbol{Z}_N')\,(\tilde{\boldsymbol{y}}_1', \ldots, \tilde{\boldsymbol{y}}_N')'$ is $m\,[2N\,(T-1)-1]$, where $m = T(T-1)/2$ is the number of moment restrictions. Therefore, the total number of flops required to calculate $\tilde{\boldsymbol{s}}$ is $N\,(T-1)\,(2T-1) + m\,[2N\,(T-1)-1]$. Given m increases with T at a quadratic rate, the number of flops required to compute $\tilde{\boldsymbol{s}}$ is therefore of order $\mathrm{O}(NT^3)$. The same number of flops is needed to compute $\tilde{\boldsymbol{s}}_{-1}$. Hence, the number of flops needed to compute $\tilde{\boldsymbol{s}}$ and $\tilde{\boldsymbol{s}}_{-1}$ increase with N linearly, for given T, and with T at a cubic rate, for given N.

To compute $\boldsymbol{A}_N$ we must compute $\boldsymbol{G} = \boldsymbol{D}\boldsymbol{D}'$, which requires $(T-1)^2\,(2T-1)$ flops; the products $\boldsymbol{G}\boldsymbol{Z}_i$ $(i = 1, \ldots, N)$, which requires another $Nm\,(T-1)\,(2T-3)$ flops; and the products $\boldsymbol{Z}_i'\,(\boldsymbol{G}\boldsymbol{Z}_i)$ $(i = 1, \ldots, N)$, which require $Nm^2\,(2T-3)$ flops. Finally, we execute $N-1$ summations of the $m \times m$ matrices $\boldsymbol{Z}_i'\boldsymbol{G}\boldsymbol{Z}_i$ $(i = 1, \ldots, N)$ for another $(N-1)\,m^2$ flops. From this accounting, we see that $(T-1)^2\,(2T-1) + Nm\,(T-1)\,(2T-3) + Nm^2\,(2T-3) + (N-1)\,m^2$ flops are required to compute $\boldsymbol{A}_N$. Given m is quadratic in T, the number of flops required to compute $\boldsymbol{A}_N$ is of order $\mathrm{O}(NT^5)$. Hence, the number of flops increase with N linearly, for given T, but they increase with T at the rate T^5, for given N.

The number flops required to compute $\boldsymbol{A}_N^{-1}$ increases with T at the rate T^6. To see this, note that standard methods for inverting a $q \times q$ matrix require on the order of q^3 operations (see Hunger, 2007; Strang, 2003, pp. 452–455). The matrix $\boldsymbol{A}_N$ is $m \times m$, and m increases with T at the rate T^2 if all available moment restrictions are exploited. Hence, the number of flops required to invert $\boldsymbol{A}_N$ is of order $\mathrm{O}(T^6)$.

No additional calculations increase with T and N as quickly as computing $\boldsymbol{A}_N$ and its inversion. For example, after $\boldsymbol{A}_N^{-1}$ is calculated, $m(2m-1)$ flops are required to calculate $\tilde{\boldsymbol{a}} = \tilde{\boldsymbol{s}}'_{-1}\boldsymbol{A}_N^{-1}$, while computing $\tilde{\boldsymbol{a}}\tilde{\boldsymbol{s}}_{-1}$, and $\tilde{\boldsymbol{a}}\tilde{\boldsymbol{s}}$ both require $2m-1$ flops.

Appendix A.2: Floating point operations using FOD

Calculation of $\ddot{\boldsymbol{y}}_i = \boldsymbol{F}\boldsymbol{y}_i$ $(i=1,\ldots,N)$ requires $N(T-1)(2T-1)$ flops. An additional $t(2N-1)$ flops are needed to calculate $\ddot{\boldsymbol{s}}_t = (\boldsymbol{z}_{1t},\ldots,\boldsymbol{z}_{Nt})(\ddot{y}_{1t},\ldots,\ddot{y}_{Nt})'$. Therefore, calculation of all of the $\ddot{\boldsymbol{s}}_t$s $(t=1,\ldots,T-1)$ requires $\ddot{f}_1 = N(T-1)(2T-1) + (2N-1)\sum_{t=1}^{T-1} t = N(T-1)(2T-1) + (2N-1)T(T-1)/2$ flops, which is of order $O(NT^2)$. Calculation of $\ddot{\boldsymbol{s}}_{t-1}$ $(t=1,\ldots,T-1)$ requires another $\ddot{f}_2 = \ddot{f}_1$ flops.

On the other hand, computing $\boldsymbol{S}_t = (\boldsymbol{z}_{1t},\ldots,\boldsymbol{z}_{Nt})(\boldsymbol{z}_{1t},\ldots,\boldsymbol{z}_{Nt})'$ requires $t^2(2N-1)$ flops. Therefore, calculation of $\boldsymbol{S}_t$ $(t=1,\ldots,T-1)$ requires $\ddot{f}_3 = (2N-1)\sum_{t=1}^{T-1} t^2 = (2N-1)T(2T-1)(T-1)/6$ flops, which is of order $O(NT^3)$.

The matrix $\boldsymbol{S}_t$ is a $t \times t$ matrix, which requires on the order of $O(t^3)$ flops to invert. Given there are $T-1$ $\boldsymbol{S}_t$ matrices that must be inverted, the number of operations required to invert all of them is on the order of $\ddot{f}_4 = \sum_{t=1}^{T-1} t^3 = T^2(T-1)^2/4$ flops. In other words, the number of flops required to invert all of the $\boldsymbol{S}_t$ matrices is of order $O(T^4)$.

After $\boldsymbol{S}_t^{-1}$ $(t=1,\ldots,T-1)$ are computed, computing $\ddot{\boldsymbol{a}}_t = \ddot{\boldsymbol{s}}'_{t-1}\boldsymbol{S}_t^{-1}$ $(t=1,\ldots,T-1)$ requires another $\ddot{f}_5 = \sum_{t=1}^{T-1} t(2t-1) = T(T-1)(4T-5)/6$ flops, which is of order $O(T^3)$. Next, calculation of $\ddot{\boldsymbol{a}}_t\ddot{\boldsymbol{s}}_t$ $(t=1,\ldots,T-1)$ requires $\ddot{f}_6 = \sum_{t=1}^{T-1}(2t-1) = T(T-2)+1$ flops, and then summing the computed $\ddot{\boldsymbol{a}}_t\ddot{\boldsymbol{s}}_t$s — i.e., $\sum_{t=1}^{T-1} \ddot{\boldsymbol{a}}_t\ddot{\boldsymbol{s}}_t$ — is another $\ddot{f}_7 = T-2$ flops.

Hence, calculation of $\sum_{t=1}^{T-1} \ddot{\boldsymbol{a}}_t\ddot{\boldsymbol{s}}_t$ requires $\sum_{j=1}^{7} \ddot{f}_j$ flops. This work increases with N at the rate N increases, for given T, and increases with T at the rate T^4 increases, for given N.

Of course, to compute $\widehat{\delta}_F$ we must also compute $\sum_{t=1}^{T-1} \ddot{\boldsymbol{a}}_t\ddot{\boldsymbol{s}}_{t-1}$, but the $\ddot{\boldsymbol{a}}_t$s and $\ddot{\boldsymbol{s}}_{t-1}$s have already been calculated. Therefore, the remaining calculations required to compute $\widehat{\delta}_F$ are but a small part of the total number of flops required.

References

Alvarez, J., & Arellano, M. (2003). The time series and cross-section asymptotics of dynamic panel data estimators. *Econometrica*, 71(4), 1121–1159.

Arellano, M., & Bond, S. (1991). Some tests of specification for panel data: Monte Carlo evidence and an application to employment equations. *The Review of Economic Studies*, 58(2), 277–297.

Arellano, M., & Bover, O. (1995). Another look at the instrumental variable estimation of error-components models. *Journal of Econometrics*, 68(1), 29–51.

Hayakawa, K., & Nagata, S. (2016). On the behaviour of the GMM estimator in persistent dynamic panel data models with unrestricted initial conditions. *Computational Statistics and Data Analysis*, 100, 265–303.

Hunger, R. (2007). Floating point operations in matrix-vector calculus (version 1.3). Munich: Technical University of Munich, Associate Institute for Signal Processing. https://mediatum.ub.tum.de/doc/625604.

Strang, G. (2003). *Introduction to linear algebra*. Massachusetts: Wellesley-Cambridge Press.