HOW LFE WORKS

SIMEN GAURE

Abstract. Here is a proof for the demeaning method used in **lfe**, and a description of the methods used for solving the residual equations. As well as a toy-example. This is a preliminary version of [3]. This method, formulated as the Gauss-Seidel algorithm, was noted in [4] without the author noticing it in the first version of this paper.

1. Introduction

We assume we have an OLS model in matrix form

$$Y = X\beta + D\alpha + \epsilon \tag{1}$$

where X is a $(n \times k)$-matrix, and D is a $(n \times g)$-matrix. D is a set of dummies for e category variables. I.e. D is a block matrix, $D = \begin{bmatrix} D_1 & D_2 & \cdots & D_e \end{bmatrix}$. That is, the entries of each D_i consists of 0 and 1, with only one non-zero entry per row. These are the dummies from a single factor, one column per level. Hence, the columns of each D_i are pairwise orthogonal. Though, in general, D_i is not orthogonal to D_j for $i \neq j$.

That is, in R the model will be

```
Y ~ X1 + X2 + ... + Xk + D1 + D2 + ... + De
```

where `D1, D2, ..., De` are arbitrary factors. I.e. an entirely ordinary model which may easily be estimated by `lm`, or with sparse-versions of the same.

g is the sum of the number of levels in the factors. Now, suppose $g \approx 10^6$, indeed, assume that all the factors have many levels, so that an unmanagable number of dummies will be created when we try to estimate, even if we sweep out the largest with a within transformation.

Then, we must do the math. Let's write the model in a slightly different block matrix form, to get hold of some facts of the Frisch-Waugh-Lovell theorem:

$$Y = \begin{bmatrix} X & D \end{bmatrix} \begin{bmatrix} \beta \\ \alpha \end{bmatrix} + \epsilon$$

We get the normal equations

$$\begin{bmatrix} X & D \end{bmatrix}' \begin{bmatrix} X & D \end{bmatrix} \begin{bmatrix} \hat{\beta} \\ \hat{\alpha} \end{bmatrix} = \begin{bmatrix} X & D \end{bmatrix}' Y$$

which, when multiplied out, become

$$\begin{bmatrix} X'X & X'D \\ D'X & D'D \end{bmatrix} \begin{bmatrix} \hat{\beta} \\ \hat{\alpha} \end{bmatrix} = \begin{bmatrix} X' \\ D' \end{bmatrix} Y$$

We then write them as two rows

Date: March 25, 2011, update in May 2013.

$$X'X\hat{\beta} + X'D\hat{\alpha} = X'Y \tag{2}$$

$$D'X\hat{\beta} + D'D\hat{\alpha} = D'Y, \tag{3}$$

and assume, for a moment, that we have removed sufficient reference levels from D, so that $D'D$ is invertible. Now, multiply through equation (3) with $X'D(D'D)^{-1}$ and subtract equation (2) from (3). This removes the $\hat{\alpha}$-term from (3). We then name $P = I - D(D'D)^{-1}D'$ to get

$$X'PX\hat{\beta} = X'PY.$$

Now, note that P is a projection, i.e. $P = P' = P^2$, hence we have $X'PX = X'P'PX = (PX)'PX$ and $X'PY = X'P'PY = (PX)'PY$ which yields

$$(PX)'PX\hat{\beta} = (PX)'PY \tag{4}$$

which is the normal equation of the system

$$PY = PX\beta + P\epsilon. \tag{5}$$

That is, $\hat{\beta}$ may be estimated from system (5), with the dummies removed, taking into account the adjusted degrees of freedom when computing the covariance matrix.

Moreover, by multiplying through equation (3) with $D(D'D)^{-1}$ and noting that $D(D'D)^{-1}D' = I - P$, we get

$$(I - P)X\hat{\beta} + D\hat{\alpha} = (I - P)Y \tag{6}$$

which may be reordered as

$$Y - (X\hat{\beta} + D\hat{\alpha}) = PY - PX\hat{\beta}$$

showing that the residuals of the projected system (5) equals the residuals of the original system (1). Similarly, the $\hat{\beta}$ part of the covariance matrix of (1) can be shown to be identical to the covariance matrix of (5).

All this is well-known as the Frisch-Waugh-Lovell theorem, and is not the main point here, that's why we're still in the "Introduction"-section.

2. What **lfe** does about this

The problem is to compute the projection P, so that we may estimate $\hat{\beta}$ from (5). Whenever $e = 1$, i.e. a single factor, applying P amounts to subtracting the group-means. This is known as the within-groups transformation, or centering on the means, or *demeaning*. But, what does it look like when we have more factors?

Here's the idea behind **lfe**, from [3]:

For each of the factors, we have a demeaning projection $P_i = I - D_i(D_i'D_i)^{-1}D_i'$. This is the projection onto the orthogonal complement of the range (column space) of D_i, called $R(D_i)^\perp$. These are easy to compute, it's just to subtract the means of each level. Similarly, P is the projection on $R(D)^\perp$. This one is not yet obvious how to compute.

There is a relation between all these range-spaces:

$$R(D)^\perp = R(D_1)^\perp \cap R(D_2)^\perp \cap \cdots \cap R(D_e)^\perp.$$

To see this, consider a vector $v \in R(D)^\perp$. By definition, it's orthogonal to every column in D, hence to every column in every D_i, thus v is in the intersection on

the right hand side. Conversely, take a v which is in all the spaces $R(D_i)^\perp$. It's orthogonal to every column of every D_i, hence it's orthogonal to every column in D, so it's in $R(D)^\perp$.

This relation may be written in terms of projections:

$$P = P_1 \wedge P_2 \wedge \cdots \wedge P_e.$$

Now, there's a theorem about projections [5, Theorem 1] stating that for every vector v, we have

$$Pv = \lim_{n\to\infty} (P_1 P_2 \cdots P_e)^n v. \tag{7}$$

So, there's how to compute Pv for an arbitrary vector v, just demean it with each projection in succession, over and over, until it gives up. We do this for every column of X and Y to find PY and PX, and then we may solve $\hat{\beta}$ from (4). This procedure has been wrapped up with a threaded C-routine in the function `felm`. Thus, the `X1,X2,...,Xk` can be estimated efficiently by

```
felm(Y ~ X1 + X2 + ... + Xk + G(D1) + G(D2) + ... + G(De))
```

If there is only one factor (i.e. $e = 1$), this reduces to the within-groups model.

3. The dummies?

To find $\hat{\alpha}$, the coefficents of all the dummies, we may write (6) as

$$D\hat{\alpha} = (Y - X\hat{\beta}) - (PY - PX\hat{\beta})$$

where the right hand side is readily computed when we have completed the steps above. There will be no more than e non-zeros in each row of D. This type of sparse system lends itself to solving by the Kaczmarz method ([6]).

The Kaczmarz method may be viewed as a variant of (7), specifically for solving linear equations. (Though, historically, the Kaczmarz-method predates Halperin's more general Hilbert-space theorem by 25 years.) The idea is that in a matrix equation like

$$Dx = b$$

we may view each row of the system $\langle d_i, x\rangle = b_i$ as an equation defining a hyperplane Q_i (where d_i is the i'th row of D). The solution set of the system is the intersection of all the hyperplanes $Q = Q_1 \cap Q_2 \cap \cdots \cap Q_n$. Thus, again, if the projection onto each Q_i is easy to compute (it is $x \mapsto x + (b_i - \langle d_i, x\rangle)d_i/\|d_i\|^2$), we may essentially use (7) on these projections to find a vector in the intersection, starting from the zero-vector.

In our case, each row d_i of the matrix D has exactly e non-zero entries, which are all equal to unity. This makes the computation of the projection on each Q_i easy and fast. We don't have to care about rank-deficiency (*you* do, if you're going to interpret the results); but we do remove consecutive duplicate rows, as these are just repeated applications of the same projection, and thus contribute nothing to the result (because projections by defintion are idempotent.)

Anyway, the Kaczmarz method converges to a solution $\hat{\alpha}$. Since we use 0 as our starting point, we compute the projection of the zero-vector onto the solution space, this is, by a defining property of projections, the solution with minimal norm. We must then apply some estimable function to get interpretable coefficients, the package supplies a couple to choose from. Moreover, it's easy to get different

solutions by using different vectors as starting points. Estimable functions should evaluate to the same value for any two such solutions, this is utilized to test user-supplied functions for estimability in the function `is.estimable`.

From the Kaczmarz method we don't get any indication of the rank-deficiency. Though for $e = 2$, this can be inferred from the component-structure returned by `getfe`. The method requires little memory, and it's way faster then most other methods.

A drawback is that the Kaczmarz method is not immediately parallelizable (though there's a variant by Cimmino which is, each iteration projects the point onto each hyperplane, then the next approximation is the centroid of these projections), and it does not yield any covariance matrix or standard errors. However, it is fast, so it's possible to bootstrap the standard errors if that's desired.

A benchmark real dataset used during development contained 15 covariates, approx 20,000,000 observations, with 2,300,000 levels in one of the factors, and 270,000 in the other. Centering the covariates takes approx 2 hours (on 8 CPUs), and then computing the fixed effects by the Kaczmarz-method takes about 4 minutes (on 1 CPU). Bootstrapping the standard errors (112 times) takes about 14 hours. (It is not necessary to centre the covariates over again when bootstrapping, only the resampled residuals. These are generally faster to centre than arbitrary covariates.) This is the default method used by `getfe`.

Alternatively, one may choose a sparse Cholesky solver. That is, we have from (3) that

$$D'D\hat{\alpha} = D'(Y - X\hat{\beta}).$$

In the case $e = 1$, we have that $D'D$ is diagonal, this is the within-groups case, and $\hat{\alpha}$ is just the group-means of the residuals $Y - X\hat{\beta}$. In the general case, we have a large, but sparse, linear system. This may be solved with the methods in package **Matrix**. This procedure has been packaged in the function `getfe`.

Now, it turns out that identification, hence interpretation, of the coefficients, *may* be a complicated affair. The reason is that the matrix $D'D$ may be rank-deficient in unexpected ways. It's sometimes not sufficient to remove a reference-level in each factor. In the case $e = 2$ these difficulties are well understood and treated in [1] and [2], as well as implemented in **lfe**. For larger e, this problem is harder, **lfe** uses a pivoted Cholesky-method to find linear dependencies in $D'D$, and removes them, but the resulting interpretation of the coefficients are in general not well understood. (This, of course, applies to the Kaczmarz method as well).

4. Optimization potential

Profiling with the tool "perf" on linux, reveals that there is some potential for optimizations in both the centering process and the Kaczmarz-solver. Both suffer from memory-bandwidth limitations, leading to an "Instructions Per Cycle"-count in some cases below 0.3 (where the theoretical maximum is 2 or 4), despite being almost pure floating point operations. This depends heavily on the problem size, cache architecture of the CPU, number of cores in use, memory bandwidth and latency, and the CPU-speed. I haven't figured out a good solution for this, though I haven't given it a lot of thought.

An interesting optimization would be to use a GPU for these operations. They are quite simple, and thus quite well suited for coding in OpenCL, CUDA or similar

GPU-tools, and could possibly yield an order of magnitude speedup, though I haven't tried anything of the sort.

5. An example

First we create a couple of covariates:

```
set.seed(41)
x <- rnorm(500)
x2 <- rnorm(length(x))
x3 <- rnorm(length(x))
```

Then we create some random factors, not too many levels, just for illustration, and some effects:

```
f1 <- factor(sample(7, length(x), replace = TRUE))
f2 <- factor(sample(4, length(x), replace = TRUE))
f3 <- factor(sample(3, length(x), replace = TRUE))
eff1 <- rnorm(nlevels(f1))
eff2 <- rexp(nlevels(f2))
eff3 <- runif(nlevels(f3))
```

Then we create an outcome with some normal residuals:

```
y <- x + 0.5 * x2 + 0.25 * x3 + eff1[f1] + eff2[f2] + eff3[f3] + rnorm(length(x))
```

Now, for illustration, create a demeaning function according to (7):

```
demean <- function(v, fl) {
    Pv <- v
    oldv <- v - 1
    while (sqrt(sum((Pv - oldv)^2)) >= 1e-07) {
        oldv <- Pv
        for (f in fl) Pv <- Pv - ave(Pv, f)
    }
    Pv
}
```

and demean things

```
fl <- list(f1, f2, f3)
Py <- demean(y, fl)
Px <- demean(x, fl)
Px2 <- demean(x2, fl)
Px3 <- demean(x3, fl)
```

And then we estimate it

```
summary(lm(Py ~ Px + Px2 + Px3 - 1))
##
## Call:
## lm(formula = Py ~ Px + Px2 + Px3 - 1)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -2.7367 -0.6249 -0.0114  0.7014  3.0506
##
## Coefficients:
##     Estimate Std. Error t value Pr(>|t|)
## Px    1.0654     0.0448   23.76  < 2e-16 ***
## Px2   0.5099     0.0454   11.23  < 2e-16 ***
## Px3   0.2274     0.0435    5.23  2.5e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.991 on 497 degrees of freedom
## Multiple R-squared:  0.586,Adjusted R-squared:  0.583
## F-statistic:  234 on 3 and 497 DF,  p-value: <2e-16
```

Note that `lm` believes there are too many degrees of freedom, so the standard errors are too small.

The function `felm` in package **lfe** adjusts for the degrees of freedom, so that we get the same standard errors as if we had included all the dummies:

```
library(lfe, quietly = TRUE)
summary(est <- felm(y ~ x + x2 + x3 + G(f1) + G(f2) + G(f3)))
##
## Call:
##    felm(formula = y ~ x + x2 + x3 + G(f1) + G(f2) + G(f3))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7367 -0.6249 -0.0114  0.7014  3.0506
##
## Coefficients:
##    Estimate Std. Error t value Pr(>|t|)
## x    1.0654     0.0454   23.47  < 2e-16 ***
## x2   0.5099     0.0460   11.09  < 2e-16 ***
## x3   0.2274     0.0440    5.17  3.5e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1 on 485 degrees of freedom
## Multiple R-squared: 0.842   Adjusted R-squared: 0.838
## F-statistic:  185 on 14 and 485 DF, p-value: <2e-16
## *** Standard errors may be too high due to more than 2 groups and exactDOF=FALSE
```

We also illustrate how to fetch the group coefficients. We use an estimable function similar to the one in `lm`. (Though, a similar function is available with `ef='ref'` which is the default for `getfe`).

```
ef <- function(v, addnames) {
    r2 <- v[[8]]
```

```
    r3 <- v[[12]]
    v[1:7] <- v[1:7] + r2 + r3
    v[8:11] <- v[8:11] - r2
    v[12:14] <- v[12:14] - r3
    if (addnames)
        names(v) <- c(paste("f1", 1:7, sep = "."), paste("f2", 1:4, sep = "."),
            paste("f3", 1:3, sep = "."))
    v
}
# verify that it's estimable
is.estimable(ef, est$fe)
## [1] TRUE
getfe(est, ef = ef, se = TRUE, bN = 10000)
##         effect     se
## f1.1  3.766027 0.1538
## f1.2  2.105724 0.1611
## f1.3 -0.791654 0.1570
## f1.4  3.905124 0.1645
## f1.5  0.003446 0.1481
## f1.6  2.633193 0.1500
## f1.7  2.458959 0.1579
## f2.1  0.000000 0.0000
## f2.2  1.271989 0.1265
## f2.3  0.170457 0.1261
## f2.4  2.084170 0.1245
## f3.1  0.000000 0.0000
## f3.2 -0.157646 0.1140
## f3.3 -0.221572 0.1128
```

Here's the same estimation in `lm`, with dummies:

```
summary(lm(y ~ x + x2 + x3 + f1 + f2 + f3))
##
## Call:
## lm(formula = y ~ x + x2 + x3 + f1 + f2 + f3)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7367 -0.6249 -0.0114  0.7014  3.0506
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.7660     0.1559   24.16  < 2e-16 ***
## x             1.0654     0.0454   23.47  < 2e-16 ***
## x2            0.5099     0.0460   11.09  < 2e-16 ***
## x3            0.2274     0.0440    5.17  3.5e-07 ***
## f12          -1.6603     0.1768   -9.39  < 2e-16 ***
## f13          -4.5577     0.1734  -26.29  < 2e-16 ***
```

```
## f14           0.1391     0.1769    0.79    0.432
## f15          -3.7626     0.1668  -22.56  < 2e-16 ***
## f16          -1.1328     0.1689   -6.71  5.6e-11 ***
## f17          -1.3071     0.1747   -7.48  3.5e-13 ***
## f22           1.2720     0.1271   10.01  < 2e-16 ***
## f23           0.1705     0.1277    1.33    0.183
## f24           2.0842     0.1250   16.68  < 2e-16 ***
## f32          -0.1576     0.1129   -1.40    0.163
## f33          -0.2216     0.1131   -1.96    0.051 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1 on 485 degrees of freedom
## Multiple R-squared:  0.842,Adjusted R-squared:  0.838
## F-statistic:  185 on 14 and 485 DF,  p-value: <2e-16
```

References

1. J. M. Abowd, F. Kramarz, and D. N. Margolis, *High Wage Workers and High Wage Firms*, Econometrica **67** (1999), no. 2, 251–333.
2. M. Andrews, L. Gill, T. Schank, and R. Upward, *High wage workers and low wage firms: negative assortative matching or limited mobility bias?*, J.R. Stat. Soc.(A) **171(3)** (2008), 673–697.
3. S. Gaure, *OLS with Multiple High Dimensional Category Variables*, Computational Statistics and Data Analysis **66** (2013), 8–18.
4. P. Guimarães and P. Portugal, *A simple feasible procedure to fit models with high-dimensional fixed effects*, Stata Journal **10** (2010), no. 4, 628–649(22).
5. I. Halperin, *The Product of Projection Operators*, Acta Sci. Math. (Szeged) **23** (1962), 96–99.
6. A. Kaczmarz, *Angenäherte Auflösung von Systemen linearer Gleichungen*, Bulletin International de l'Academie Polonaise des Sciences et des Lettres **35** (1937), 355–357.

Ragnar Frisch Centre for Economic Research, Oslo, Norway