lfe: Linear Group Fixed Effects

by Simen Gaure

Abstract Linear models with fixed effects and many dummy variables are common in some fields. Such models are straightforward to estimate unless the factors have *too many* levels. The R package **lfe** solves this problem by implementing a generalization of the **within transformation** to multiple factors, tailored for large problems.

Introduction

A typical linear model looks like:

```
> y ~ x1+x2+x3 + f1+f2+f3
```

where `f1,f2,f3` are arbitrary factors, and `x1,x2,x3` are other covariates. Coefficients may easily be estimated by `lm()`:

```
> lm(y ~ x1+x2+x3 + f1+f2+f3)
```

However, in some applications, in particular in econometrics, the factors have too many levels and are still needed as fixed effects, as in Abowd et al. (1999). The use of fixed effects as opposed to random effects is typically necessitated by the possibility that the factors and the other regressors are correlated. They study log wage (`logwage`) as an outcome, where fixed effects for individuals (`id`) and firms (`firm`) are included, with some ordinary covariates (exemplified by `x`), i.e. a model of the type:

```
> logwage ~ x + id + firm
```

where `id` is a factor with one level for each employee, and `firm` is a factor with one level for each firm. Employees change firm now and then, so it is not a nested model. The model is used to account for arbitrarily distributed time constant individual and firm hetereogeneity, and is used to study the correlation between the firm effect and the employee effect. There are also similar cases with 3 factors, e.g. in Torres et al. (2013), where job title is also considered. They have a dataset of 27 million observations, with 5.5 million, 568,000, and 96,000 levels in the three factors. In other applications the factors are primarily used as controls, as in Markussen and Røed (2012), where a variety of models are used, controlling for various combinations of interactions between factors. These datasets are typically sourced from large public registries, and can contain tens of millions of observations, with hundreds of thousands or millions of factor levels. This far exceeds the capabilities of `lm()`, and can also be too demanding for the sparse methods in package **Matrix** (Bates and Maechler, 2013).

When estimating such models, the case with a single factor is special. It can be estimated by using the within groups transformation, where the mean of the groups are subtracted from the covariates, resulting in a system without the factor, via the Frisch-Waugh-Lovell theorem. See e.g. Wooldridge (2002, Section 10.5). The coefficients for the factor levels can easily be recovered as the group means of the residuals. This estimation method can be found in package **plm** (Croissant and Millo, 2008). The method can also be used with more than one factor, by using the within transformation to project out the factor with the highest number of levels, coding the others as dummy variables. However, if all of the factors have many levels this can still result in a too large system which is non-sparse. Moreover, having such sets of dummies in datasets which are not balanced may lead to non-trivial identification problems. We will illustrate how to use the package **lfe** (Gaure, 2013b) to solve some of these problems. For clarity we construct quite small datasets by drawing random covariates, rather than to refer to large real datasets which are not publicly available.

Enter lfe

The package **lfe** is designed to handle the above estimation problem. It also contains some methods for solving identification problems, though not completely in all cases. Since **lfe** handles quite ordinary linear models which conceptually could be handled by `lm()`, we do not go into detail about the feasibility of fixed effect linear models as such, only the problems which are more or less peculiar to models with a large number of dummies.

The short story is that coefficients for `x1`, `x2`, and `x3` in the above model can be estimated by:

```
> library(lfe)
> est <- felm(y ~ x1+x2+x3 + G(f1)+G(f2)+G(f3))
```

whereas the coefficients for the factor levels of `f1`, `f2`, and `f3` can be retrieved by:

```
> alpha <- getfe(est)
```

A longer story follows, the theoretical part of which is mainly based on Gaure (2013a).

We write our model in matrix form

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

where D is a matrix of dummies coding the factor levels, X is a matrix with the other covariates, y is the response vector, and ϵ is a normally distributed error term. Performing an ordinary least squares regression (OLS) on this system yields the OLS estimates $\hat{\beta}$ for the X covariates and $\hat{\alpha}$ for the factor levels. The Frisch-Waugh-Lovell theorem states that if P is the projection onto the orthogonal complement of the range of D, then the projected system

$$Py = PX\beta + P\epsilon,$$

yields the same $\hat{\beta}$ when estimated with OLS. Moreover, the matrix $(X^tPX)^{-1}$ which is used to find the covariance matrix of $\hat{\beta}$, is identical to the $\hat{\beta}$-part of the corresponding matrix in the full system (1). Similarly, the residuals are identical. The projected system does not contain the dummies for the factor levels, and may therefore be manageable with conventional methods.

Unfortunately, P is an $n \times n$ matrix, where n is the number of observations, so it is impractical to compute P when $n \approx 10^7$. However, it is easier to compute Px for a vector x, i.e. y and the columns of X. In the special case when D encodes a single factor, the transformation $x \mapsto Px$ is the within transformation, i.e. centring of the groups on their means. It is shown in Gaure (2013a) that when there is more than one factor, say $e > 1$ factors, we may consider the centring transformation for each of them, a projection P_i for each $i = 1 \dots e$, and compute Px as

$$Px = \lim_{m\to\infty} \left((P_1 P_2 \cdots P_e)^m \, x \right).$$

This approach is known as the *method of alternating projections* and is based on a result by Halperin (1962, Theorem 1). The procedure can easily be implemented with an R function taking as input a vector `x` and the list of factors `flist`:

```
> demean <- function(x, flist) {
+   cx <- x; oldx <- x - 1
+   while(sqrt(sum((cx - oldx) ^ 2)) >= 1e-8) {
+     oldx <- cx
+     for(f in flist) cx <- cx - ave(cx, f)
+   }
+   return(cx)
+ }
```

This algorithm was also arrived at by Guimarães and Portugal (2010, p. 637) as a technical simplification of their iterated estimation approach to the same problem.

For efficiency reasons, this linear transformation has been written in C, made threaded to centre vectors in parallel, and is available as the function `demeanlist()` in **lfe**, though the function `felm()` wraps it in an `lm()` like function.

We create a simple example to illustrate the usage. We have 100,000 observations of a covariate `x`, and two factors `f1`, `f2`, each with 10,000 randomly drawn levels. We create an outcome variable `y` and estimate the `x` coefficient. The `G()` syntax is used to specify which factors should be projected out of the system, a similar syntax as for the `Error()` term in `aov()`. The `G()` is not an R function in itself, though it translates to `as.factor()` inside `felm()` after the `G()` terms have been removed from the model for special handling.

```
> library(lfe)
> set.seed(42)
> x <- rnorm(100000)
> f1 <- sample(10000, length(x), replace=TRUE)
> f2 <- sample(10000, length(x), replace=TRUE)
> y <- 2.13*x + cos(f1) + log(f2+1) + rnorm(length(x), sd=0.5)
> est <- felm(y ~ x + G(f1) + G(f2))
> summary(est)

Call:
   felm(formula = y ~ x + G(f1) + G(f2))

Residuals:
```

```
       Min         1Q     Median         3Q        Max
-1.9531308 -0.3018539 -0.0003573  0.3007738  2.2052754

Coefficients:
  Estimate Std. Error t value Pr(>|t|)
x 2.130889   0.001768    1205   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5013 on 80000 degrees of freedom
Multiple R-squared: 0.9683   Adjusted R-squared: 0.9603
F-statistic: 122.1 on 19999 and 80000 DF, p-value: < 2.2e-16
```

The result of `felm()` is a 'felm' object which is quite similar to an 'lm' object. However, it is not fully compatible with it, so some methods for 'lm' objects may work on a 'felm' object, others may not. In an earlier version, the 'felm' object inherited from the class 'lm', but there are important differences in the structure, so this inheritance has been removed. In particular there is no qr-decomposition in the 'felm' object, so methods for 'lm' objects which depend on the qr-decomposition, such as `anova()`, can not be used. Also, the 'felm'-object does not contain a copy of the dataset. This has been removed to conserve memory for very large datasets. Simple extractors like `coef()` and `residuals()` work. Extracting the covariance matrix with `vcov()` also works, via a separate S3 method, albeit only for the $\hat{\beta}$s. Also, a `summary()` S3-method, and an accompanying `print()` method have been included. It is possible to try 'lm'-methods explicitly as in: `getS3method('vcov','lm')(est)`, though the author does not guarantee any success with this approach.

The careful reader has noticed that the behaviour of `summary()` on a 'felm' object with respect to degrees of freedom and R^2 is the same as that of on an 'lm' object when including an intercept. There is no explicit intercept in the result of `felm()`, but the factor structure includes one implicitly.

The coefficients for the factor levels

If the factors are used as controls only, the above procedure is all we have to worry about, but if we also need the group coefficients $\hat{\alpha}$, what econometricians often refer to as *the* fixed effects, we can solve the equation

$$D\hat{\alpha} = (I - P)(y - X\hat{\beta}), \tag{2}$$

for $\hat{\alpha}$. The right hand side is easily computed when we have $\hat{\beta}$, PX and Py. The equation is solved by the Kaczmarz method (Kaczmarz, 1937), as described in Gaure (2013a). The method is available as a function `kaczmarz()`, but the wrapper `getfe()` is more useful. A solution of equation (2) is not unique, the matrix D dummy-encodes *all* the factor levels, hence there will be multicollinearities (unless there is only one factor), both the obvious ones, but there can also be spurious ones. The multicollinearities are resolved by applying an estimable function. **lfe** contains a function `efactory()` for creating estimable functions, but users can also supply their own.

`getfe()` returns a 'data.frame' with some information in addition to the coefficients. The reason for this is that the intended use of **lfe** is for factors with so many levels that they probably anyway must be analyzed according to the researcher's needs prior to being presented. We continue the example:

```
> alpha <- getfe(est)
> nrow(alpha)

[1] 20000

> alpha[9998:10003,]

             effect obs comp fe   idx
f1.9998  -0.2431720   9    1 f1  9998
f1.9999  -0.9733257   5    1 f1  9999
f1.10000 -0.8456289   9    1 f1 10000
f2.1      0.4800013   9    1 f2     1
f2.2      1.4868744  14    1 f2     2
f2.3      1.5002583  11    1 f2     3
```

The `'obs'` column is the number of observations of this level. The `'fe'` and `'idx'` columns are `factors` containing the same information as the row name, but split into the name of the factor and the *level* for convenience. The `'comp'` column is important in that it is used for identification purposes, and here is a main deviation from how `lm()` treats identification in the presence of factors. The

default method when using `lm()` is taken from the global `'contrasts'` option which defaults to using treatment contrasts for each factor. It introduces a reference level for each factor, this is equivalent to forcing the coefficient for the reference level to zero, and the other coefficients are only meaningful when compared to this zero coefficient. It is also possible to specify different constraints for the `lm()` coefficients, such as forcing the sum of the coefficients to zero, or an arbitrary one via the `'contrasts'` option, or the `'contrasts'` argument to `lm()`, but typically a single constraint is used for each factor.

Identification when the factors have many levels can be a more complicated matter. The standard example in the econometrics literature is the one found in Abowd et al. (1999), elaborated in Abowd et al. (2002). In this case there are two factors, one for employees and one for firms. It may happen that one set of employees move between one set of firms, whereas another disjoint set of employees move between some other firms. There are no movements between these *mobility groups*, hence coefficients from different groups can not be compared. A useful construction for analysis of this problem is the undirected, bipartite graph which has the factor levels as vertices, and an edge between levels that occur in the same observation. The mobility groups are then the connected components of this graph. It is shown in Abowd et al. (2002, Appendix 1) that for identification of the coefficients, it is sufficient to introduce a single reference level in each of the disjoint mobility groups, either a firm or an employee. This was previously established by Eccleston and Hedayat (1974) in a different context, and there is another argument in Gaure (2013a) using spectral graph theory. The `'comp'` column in the return value from `getfe()` when using estimable functions from `efactory()` is a factor enumerating these groups, i.e. the connected components. `efactory()` chooses by default the level with the highest number of observations as a reference level, and sets the coefficient to 0. When interpreting the coefficients, they should never be compared between the components; a coefficient is only meaningful relative to the reference level in the component it belongs to. For this reason, one may choose to restrict attention to the largest component only, the one with `comp == 1`, if this is feasible for the problem at hand.

To the author's knowledge, the identification problem when there are more than two factors has not been solved in general. `efactory()`'s behaviour with more than two factors is to assume that the connected components of the two first factors are sufficient for identification, and a single reference is used in each of the remaining factors. **lfe** contains a probabilistic test for estimability in this case, the one described in Gaure (2013a, Remark 6.2), and `getfe()` will issue a warning if it finds an identification problem. The test is also available as the function `is.estimable()`. In theory, the test can fail only on a set of measure zero. Specifically, the test uses the fact that an estimable function must evaluate to the same value on all solutions of equation (2). Different solutions can be obtained by running the Kaczmarz algorithm with different initial vectors. By starting with the zero vector we obtain the solution with least Euclidean norm. The test works by comparing the value of the candidate estimable function on the least norm solution and a solution obtained by drawing the initial vector η at random. The test will only fail to find an identification problem if η happens to lie in a particular, but unknown, subspace of positive codimension, i.e. in a set of Lebesgue measure zero. However, due to numerical inaccuracies, finite arithmetic, and bad luck, the test may in practice fail to find an identification problem even if there is one, with some very small positive probability. It does however not report identification problems when there are none. If there are identification problems which are unaccounted for, then some, or all of the resulting coefficients are non-estimable, i.e. they are meaningless.

Sometimes, an identification problem can be alleviated by changing the order of the `G()` terms in the formula supplied to `felm()`. To illustrate, consider a situation where we add a third factor to the example in the Introduction, `nkids`, the number of an individual's offspring, with only 5 levels. If our formula contains `G(firm) + G(nkids) + G(id)`, it is likely that the graph associated with the two first factors, `firm` and `nkids`, is connected, and there can be many mobility groups which are unaccounted for. `getfe()` issues a warning, the returned coefficients are non-estimable; there is no correct indication of which mobility groups the coefficients belong to, and therefore, we do not know which coefficients can be meaningfully compared. If we change the model specification to `G(id) + G(firm) + G(nkids)`, or `G(id) + G(firm) + nkids`, the mobility groups are found, and accounted for.

Another approach to the identification problem is found in Carneiro et al. (2012). They restrict attention to a subset of the dataset in which, by construction, all elementary contrasts, i.e. differences between coefficients, are estimable, as described by Weeks and Williams (1964). Such a partitioning of the dataset can be found by constructing a graph with the observations as vertices, and with an edge between two observations if they differ in at most a single factor. The partitions of the dataset correspond to connected components of the graph. They can be found by the function `compfactor(...,WW=TRUE)`, which returns a factor enumerating the partitions. It can be used to select a subset of the dataset prior to calling `felm()`. When there are only two factors, this partitioning structure coincides with the mobility groups discussed above. In some datasets, like the one in Torres et al. (2013), the largest partition comprises almost the entire dataset, but if it does not, such a selection may introduce bias in the coefficients. An extreme example:

```
> set.seed(42)
> f1 <- factor(sample(50, 1000, replace=TRUE))
> f2 <- factor(sample(50, 1000, replace=TRUE))
> f3 <- factor(sample(50, 1000, replace=TRUE))
> ww <- compfactor(list(f1,f2,f3), WW=TRUE)
> head(table(ww))

ww
 1  2  3  4  5  6
29 20 19 16 14 14
```

The largest partition has 29 observations, fewer than the number of levels, even though far more estimable functions exist. This can be seen by computing the rank deficiency of the matrix D with the function `rankMatrix()` from package **Matrix**. Indeed, it is sufficient with two references in the 3 factors:

```
> D <- t(do.call('rBind', lapply(list(f1,f2,f3), as, 'sparseMatrix')))
> ncol(D) - as.integer(rankMatrix(D))

[1] 2
```

The standard method of `efactory()`, to analyze the connected components of the first two factors only, works well in this particular case. It will find a reference among the two first factors, and a reference in the last, and differences between coefficients within each factor will be estimable:

```
> x <- rnorm(1000)
> y <- 3.14*x + log(1:50)[f1] + cos(1:50)[f2] + exp(sqrt(1:50))[f3] + rnorm(1000, sd=0.5)
> est <- felm(y ~ x + G(f1) + G(f2) + G(f3))
> coef(est)

       x
3.139781

> is.estimable(efactory(est), est$fe)

[1] TRUE
```

This can happen because the construction in Weeks and Williams (1964, Theorem 1) does not find a maximal subset, only a subset which is guaranteed to have the desired property.

Yet another algorithm for finding estimable functions is described by Godolphin and Godolphin (2001), but **lfe** does not implement it.

Specifying an estimable function

The function `efactory()` in **lfe** is by default called by `getfe()` to create an estimable function for a factor structure. The default is to use reference levels as described above, some other estimable functions are also available. However, researchers may have other needs, so it is possible to supply a user written estimable function to `getfe()`. We describe this interface with an example, for a small dataset. The function takes as an argument a vector of length the sum of the number of levels of the factors, i.e. a solution to equation (2), applies an estimable function of choice, and returns the result. It is not necessary to include a full set of estimable functions; if so desired, our function may return just a single difference between two specific factor levels, or even some nonlinear transformation thereof. It should, however, evaluate to the same value on all the different solutions of equation (2).

We make a small dataset with 100 observations, a covariate `x` and 3 factors with a handful of levels.

```
> set.seed(42)
> x <- rnorm(100)
> f1 <- factor(sample(4, 100, replace=TRUE))
> f2 <- factor(sample(5, 100, replace=TRUE))
> f3 <- factor(sample(6, 100, replace=TRUE))
> e1 <- sin(1:4)[f1] + 0.02*((1:5)^2)[f2]  + 0.17*((1:6)^3)[f3] + rnorm(100)
> y <-  2.5*x + (e1-mean(e1))
> est <- felm(y ~ x + G(f1) + G(f2) + G(f3))
```

Then we create an estimable function which is the same as the one provided by `lm()` when using treatment contrasts. The `addnames` argument is there because in the event that the estimable function

is used in bootstrapping, adding names to very long vectors would only contribute to exercising R's memory management; hence names should only be added when required. It is also possible to add an `attribute` called `'extra'`, which is a named list of vectors of the same length as the names, for adding additional information to the result of `getfe()`. This `attribute` is added by the estimable functions returned by `efactory()` to provide the `'obs'`, `'fe'`, `'comp'` and `'idx'` columns, but the content is not used for anything inside **lfe**.

```
> ef <- function(gamma, addnames) {
+   ref1 <- gamma[1]  # first level of f1
+   ref2 <- gamma[5]  # first level of f2
+   ref3 <- gamma[10] # first level of f3
+   # put the intercept in the first coordinate
+   icpt <- ref1 + ref2 + ref3
+   # subtract the references for each factor
+   # unlike the efactory() functions, we omit the zero coefficients.
+   result <- c(icpt, gamma[2:4]-ref1, gamma[6:9]-ref2, gamma[11:15]-ref3)
+   if(addnames) {
+     names(result) <- c('(Intercept)',
+                        paste('f1', levels(f1)[2:4], sep=''),
+                        paste('f2', levels(f2)[2:5], sep=''),
+                        paste('f3', levels(f3)[2:6], sep=''))
+     attr(result, 'extra') <- list(fe=factor(
+                                     c('icpt', rep('f1',3),
+                                       rep('f2',4), rep('f3',5))),
+                                   idx=factor(c(1, 2:4, 2:5, 2:6)))
+   }
+   result
+ }
```

We now check that our function is estimable:

```
> is.estimable(ef, list(f1,f2,f3))

[1] TRUE
```

The estimable function is supplied to `getfe()` via the argument `ef`. In this example we also request bootstrapped standard errors with the arguments `se` and `bN`, we elaborate on this in the next section.

```
> getfe(est, ef=ef, se=TRUE, bN=1000)

                 effect   fe idx        se
(Intercept) -10.9016327 icpt   1 0.3077378
f12          -0.1265879   f1   2 0.2356162
f13          -0.7541019   f1   3 0.2896058
f14          -1.7409436   f1   4 0.2776542
f22           0.4611797   f2   2 0.3012931
f23           0.6852553   f2   3 0.2898361
f24           0.8467309   f2   4 0.3232411
f25           0.5886517   f2   5 0.2841049
f32           1.0898551   f3   2 0.3364884
f33           4.3490898   f3   3 0.3058420
f34          10.7505266   f3   4 0.3377505
f35          21.3832700   f3   5 0.3649107
f36          36.7369397   f3   6 0.3059049
```

`getfe()` performs no automatic addition of columns like `'idx'` or `'comp'`, they have to be provided by the estimable function. The reason being that `getfe()` does not know the structure imposed by the user written function. If bootstrapped standard errors are requested via the `se` argument, the `'se'` column is added also for user written functions. If you are certain that your function is estimable, you can add an attribute to it, `attr(ef,'verified') <- TRUE`, this causes `getfe()` to skip the estimability test, and may save significant amounts of time with datasets for which convergence is slow. The functions produced by `efactory()` for one and two factors have this attribute set.

Standard errors

The standard errors for $\hat{\beta}$ are easily computed, and are identical to the the standard errors from `lm()` if we were to run it with all the dummies. However, there is a minor difference. The degrees of freedom

depends on the rank of the matrix D, or D^tD, or the trace of P. The rank is easily computed for the cases with one or two factors, but requires a much more time consuming approach for 3 or more factors. The default approach of `felm()` is to assume that the column rank deficiency of D is $e-1$ when there are $e > 2$ factors. This may result in a too high value for the rank, hence a too low value for the degrees of freedom, which will yield too high standard errors. In most real cases the author has witnessed, the error is negligible. `felm()` has an argument `exactDOF` which may be set to `TRUE` to activate a more accurate computation of the rank. It does a sparse pivoted Cholesky factorization of $D^tD + \epsilon I$ for some small ϵ and finds the rank deficiency by counting small pivots. This is not a perfect method, as noted by Higham (1990), but it seems to work well for the matrices the author has encountered, and it is much faster than `rankMatrix()` from package **Matrix**. To use `rankMatrix()` instead, one may specify `exactDOF='rM'`. If, for some reason, one happens to know the degrees of freedom in advance, they can be specified as a numeric, like `exactDOF=12536819`.

The standard errors for $\hat{\alpha}$, the coefficients for the factor levels, can be estimated by bootstrapping. This requires resampling the residuals with replacement, and do the whole estimation over and over again, an operation which can be very time consuming. It can be requested, as in the above example, by the `se` argument to `getfe()`, the number of samples can be specified in the `bN` argument. The common practice of resampling observations rather than residuals is not useful for this kind of model, it results in a different factor structure, hence possibly a different set of identified coefficients in each sample.

Instrumental variables

lfe supports instrumental variables estimation through 2SLS, i.e. two step OLS (Wooldridge, 2002, Chapter 5).

Here is an example. We first create some covariates:

```
> set.seed(276709)
> x <- rnorm(10000)
> x2 <- rnorm(length(x))
> x3 <- rnorm(length(x))
```

Then we create some factors, and their effects:

```
> id <- factor(sample(2000, length(x), replace=TRUE))
> firm <- factor(sample(1300, length(x), replace=TRUE))
> id.eff <- rnorm(nlevels(id))
> firm.eff <- rnorm(nlevels(firm))
```

and a normally distributed error term `u`, and an outcome `y`:

```
> u <- rnorm(length(x))
> y <- x + 0.5*x2 + id.eff[id] + firm.eff[firm] + u
```

We create a covariate `Q` which is correlated with the other covariates, the instrument `x3`, and the error term, and add it to the outcome:

```
> Q <- 0.3*x3 + x + 0.2*x2 + 0.5*id.eff[id] + 0.7*u + rnorm(length(x), sd=0.3)
> y <- y + 0.9*Q
```

Estimation is now carried out by specifying the instrumented variable equation with the `iv` argument of `felm`. Only the instrument variable, in this case `x3`, is needed, the other covariates are added by `felm()`.

```
> ivest <- felm(y ~ x + x2 + G(id) + G(firm) + Q, iv=Q ~ x3)
> summary(ivest)

Call:
   felm(formula = y ~ x + x2 + G(id) + G(firm) + Q, iv = Q ~ x3)

Residuals:
      Min        1Q    Median        3Q       Max
-6.036142 -0.903260  0.000759  0.913758  4.971614

Coefficients:
         Estimate Std. Error t value Pr(>|t|)
x         0.94963    0.03975   23.89   <2e-16 ***
```

```
x2        0.49567    0.01449   34.20   <2e-16 ***
`Q(fit)`  0.94297    0.03816   24.71   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.668 on 6717 degrees of freedom
Multiple R-squared: 0.8112   Adjusted R-squared: 0.7189
F-statistic: 8.791 on 3282 and 6717 DF, p-value: < 2.2e-16
```

`felm()` supports more than one instrumented variable, in this case the `iv` argument must be a `list` of `formulas`, one for each instrumented variable, with all the instruments on the right hand side.

The author has been made aware that both the `G()` syntax and the handling of instrumental variables equations could be done in a smoother fashion with multi-part `formulas`. Such a syntax is not presently available in **lfe**.

Estimation time

It is a fact of life that both `felm()` and `getfe()` may take quite a while to complete. What is more unfortunate is that time to completion does not only depend on the size of the dataset, but also on its structure. In particular, the dependencies between the factors, beyond the pure identification problems, have a huge impact. The rate of convergence for the general method of alternating projections has been analyzed by Aronszajn (1950); Kayalar and Weinert (1988); Gearhart and Koshy (1989); Deutsch and Hundal (1997); Bauschke et al. (2003); Badea et al. (2012), among others. Their results are in terms of the cosine c of generalized angles between the subspaces corresponding to the projections P_i. For two factors, the convergence rate in operator norm is known to be

$$\|(P_1P_2)^n - P\| = c^{2n-1}, \tag{3}$$

where $0 \leq c < 1$. The convergence is linear in the case with two factors, but the rate depends heavily on the structure of the factors. With 3 or more factors, the convergence rate depends on both the structure of the factors and their order in the iterations, in a quite complicated way (Deutsch and Hundal, 1997, Theorem 2.7). The theoretical convergence results are also valid for the Kaczmarz method used by `getfe()`, as this is a special case of the method of alternating projections (Deutsch and Hundal, 1997, Section 4). Though, for our Kaczmarz step the number of projections is the number of observations, not the number of factors.

We do not offer general results which relate intuitive properties of the factors to the convergence rate, but here are some small examples to illustrate the complexity.

In our first example the factors `f1` and `f2` are independent:

```
> set.seed(54)
> x <- rnorm(100000)
> f1 <- sample(10000, length(x), replace=TRUE)
> f2 <- sample(300, length(x), replace=TRUE)
> y <- x + cos(f1) + log(f2+1) + rnorm(length(x), sd=0.5)
```

We time the estimation:

```
> system.time(est <- felm(y ~ x + G(f1) + G(f2)))

   user  system elapsed
  2.420   0.008   1.955

> system.time(alpha <- getfe(est))

   user  system elapsed
  0.256   0.008   0.263
```

This is a quite fast example, in general it is the author's experience that datasets with this kind of full independence between the factors converge quite fast. Convergence is equally fast with 10,000 levels in the second factor as well.

But then we introduce a dependence which makes convergence slow. We let the second factor be closely related to the first, with some stochasticity, but we do not increase the number of levels. In this example, the second factor `f3` can only have 5 different values for each value of `f1`.

```
> f3 <- (f1 + sample(5, length(x), replace=TRUE)) %% 300
> y <- x + cos(f1) + log(f3+1) + rnorm(length(x), sd=0.5)
> system.time(est <- felm(y ~ x + G(f1) + G(f3)))

   user  system elapsed
 34.624   0.000  18.804

> system.time(alpha <- getfe(est))

   user  system elapsed
  3.868   0.008   3.880
```

Execution time increases by an order of magnitude, even though the size of the dataset is the same as before. In this example, with only 300 levels in the second factor, we may as well encode it as ordinary dummies, reducing our model to the classical within groups estimator, with 300 covariates:

```
> system.time(est <- felm(y ~ x + G(f1) + factor(f3)))

   user  system elapsed
 10.340   0.832   5.379

> length(coef(est))

[1] 300

> system.time(alpha <- getfe(est))

   user  system elapsed
  0.192   0.000   0.192
```

This is far from being the whole story. A "small" change to the second factor brings the execution time back down. We still have only 5 different values of the second factor `f4` for each value of `f1`, but they are spread irregularly apart:

```
> f4 <- (f1 + sample(5, length(x), replace=TRUE)^3) %% 300
> y <- x + cos(f1) + log(f4+1) + rnorm(length(x), sd=0.5)
> system.time(est <- felm(y ~ x + G(f1) + G(f4)))

   user  system elapsed
  2.564   0.000   2.081

> system.time(alpha <- getfe(est))

   user  system elapsed
  0.240   0.004   0.244
```

To better appreciate the complexity, we create two more examples which are similar to the previous one, but the randomly drawn numbers are spread regularly apart. The first one results in slow convergence:

```
> f5 <- (f1 + sample(seq(1,197,49), length(x), replace=TRUE)) %% 300
> y <- x + cos(f1) + log(f5+1) + rnorm(length(x), sd=0.5)
> system.time(est <- felm(y ~ x + G(f1) + G(f5)))

   user  system elapsed
 32.636   0.000  17.368

> system.time(alpha <- getfe(est))

   user  system elapsed
  3.972   0.000   3.975
```

The second one converges fast:

```
> f6 <- (f1 + sample(seq(1,201,50), length(x), replace=TRUE)) %% 300
> y <- x + cos(f1) + log(f6+1) + rnorm(length(x), sd=0.5)
> system.time(est <- felm(y ~ x + G(f1) + G(f6)))

   user  system elapsed
  2.548   0.000   2.073
```

```
> system.time(alpha <- getfe(est))

   user  system elapsed
  0.244   0.000   0.244
```

By comparing the "user" and "elapsed" times for `felm()` in the above examples, it can be inferred that most of the time in the fast examples is spent in bookkeeping outside the centring, otherwise "user" time would be close to twice the "elapsed" time, since the actual centring of `y` and `x` is done in parallel. This means that the actual centring convergence rate differences are larger than the reported differences in execution time. For a careful analysis of centring times only, the centring process should be timed directly, as mentioned in Gaure (2013a, top of p. 17).

The author has not succeeded in finding general intuitive guidelines for the convergence rate. However, with two factors, a relevant concept seems to be the bipartite graph where the vertices are factor levels and there is an edge between levels which are observed together. It is the connected components of this graph which determines estimability. The graph can be analyzed with the tools in package **igraph** (Csardi and Nepusz, 2006) as follows:

```
> library(igraph)
> mkgraph <- function(flist) {
+   graph.adjacency(tcrossprod(do.call('rBind',
+                                      lapply(flist, as, 'sparseMatrix')))>0,
+                   'undirected', diag=FALSE)
+ }
```

We make a list of the associated graphs:

```
> glist <- lapply(list(f2,f3,f4,f5,f6),
+                 function(f) mkgraph(lapply(list(f1,f), factor)))
```

The graphs, except for the last, are connected:

```
> sapply(glist, no.clusters)

[1]  1  1  1  1 50
```

The average degrees, i.e. the number of edges in the graph, show no convincing signs of being strongly related to the estimation time:

```
> sapply(glist, function(g) mean(degree(g)))

[1] 19.100301  8.404311  8.410719  8.400039  8.423925
```

A property of the graph which shows some promise of correlating with the convergence rate is the *diameter*. To get some intuituion of what this is, consider the example with firms and employees. The associated graph has as vertices the firms and employees, and there is an edge between a firm and an employee if the employee has worked for the firm. A *path* between e.g. two employees *Roy* and *Floyd* can be constructed by finding a firm that *Roy* has worked for, then another employee *Warshall* of this firm, then another firm for *Warshall*, zigzagging between firms and their employees until we reach the employee *Floyd*. For each pair of vertices (i.e. firms and employees) in a mobility group, there is a shortest path. The length of the longest of the shortest paths among all vertex pairs is the graph's diameter.

In these particular examples, it turns out that in the cases with fast convergence (the first, third, and fifth), the shortest paths between pairs of vertices are typically much shorter than in the slowly converging cases. We do not compute the diameter as a typical fast algorithm for this, the *Roy-Warshall-Floyd* algorithm, has complexity of order $O(n^3)$, where n is the number of vertices, i.e. 10300 in our example. Below, for simplicity, we exclude the last graph, it is disconnected, hence of infinite diameter, though the diameters of its (comparably small) components are relevant. A component of this graph would typically contain only about $300/50 = 6$ different levels of the second factor `f6`, so its diameter can't possibly be very large.

```
> for(gr in glist[1:4])
+   print(fivenum(shortest.paths(gr, v=sample(V(gr),10), to=sample(V(gr),10))))

[1]  2  2  4  4  4
[1]  2 20 39 62 76
[1]  2  4  6  6  8
[1]  2 18 40 58 76
```

Also, the many variants of factor structures in Markussen and Røed (2012) (described in Gaure (2013a)) and ongoing subsequent works, use factors which are interactions of other factors, in such a way that there are a few collinearities by design. This is not a conceptual problem when the factors are used as controls, but manually removing these known collinearities by merging some levels into a common "reference" level, yields performance improvements of up to two orders of magnitude (Markussen, 2013), a phenomenon which was *not* noted in Gaure (2013a). Incidentally, this also leads to shorter paths via the common reference level.

This suggests that, loosely speaking, datasets with six degrees of separation converge fast, whereas less well connected datasets converge slower. However, we cannot rule out other structural differences which may have an impact on the convergence rate; there is no such thing as "larger diameter, ceteris paribus" for graphs. It is possible that a diameter path could be used to construct a lower bound for the c in equation (3), but the author has not found a proof for such an assertion.

The convergence rate with more than two factors is more complicated, the theoretical results of Deutsch and Hundal (1997) are less complete, the rate is not a function of the cosines of *pairs* of subspaces, and even the order of the projections matters. In cases where only some of the factors have many levels, the remaining factors may be specified without `G()`, treating them as ordinary dummy-encoded covariates.

lfe utilizes two acceleration techniques for the alternating projections methods. In `demeanlist()`, which is used by `felm()` to centre the covariates, the line search method in Bauschke et al. (2003, Theorem 3.16) is used, even though their *Example 3.24* shows that in some special cases with more than two factors, this method is in fact slower. For the Kaczmarz method, random shuffling of the equations is used, as suggested by Gaure (2013a).

Parallelism

`felm()` is thread parallelized over the vectors to be centred, i.e. all variables in the model except those enclosed in `G()`. The number of processors is fetched with `getOptions('lfe.threads')` which is initialized upon loading of **lfe** from the environment variable `LFE_THREADS` or `OMP_NUM_THREADS`, or otherwise from an heuristic snatched from package **multicore** (Urbanek, 2011). It may of course be set manually after loading the package. There is no benefit from specifying more threads than there are variables to centre. `getfe()` is not parallelized as such, but bootstrapping with `getfe(...,se=TRUE)` is.

Elsewhere

lfe is similar in function, but not in method, to the Stata (StataCorp., 2013) modules A2reg (Ouazad, 2008) and felsdvreg (Cornelißen, 2008). It is the same algorithm as in the Stata module reg2hdfe (Guimarães, 2009), described by Guimarães and Portugal (2010) in terms of Gauss-Seidel iterations.

Summary

The package **lfe** contains methods for estimating ordinary least square models with multiple factors with too many levels for conventional methods like `lm()`. Such models may exhibit non-trivial identification problems. These are satisfactorily solved for the two factor case, and the package also includes some partial solutions for the case with more factors. Instrumental variable regression via the two step OLS is also supported. Convergence rate can be an issue for some datasets, and the present paper suggests some intuitive structural reasons for this.

The method employed by **lfe** is known as the *method of alternating projections*, a somewhat old and well studied method which to the author's knowledge has not been applied to the particular problem of linear regression with dummy variables before.

Acknowledgements

I wish to thank the associate editor and two anonymous reviewers for helpful comments, as well as Bjørn-Helge Mevik for fruitful discussion.

Bibliography

J. M. Abowd, F. Kramarz, and D. N. Margolis. High wage workers and high wage firms. *Econometrica*, 67(2):251–333, 1999. doi: 10.1111/1468-0262.00020. URL http://www.jstor.org/stable/2999586. [p104, 107]

J. M. Abowd, R. H. Creecy, and F. Kramarz. Computing person and firm effects using linked longitudinal employer-employee data. Longitudinal Employer-Household Dynamics Technical Papers 2002-06, Center for Economic Studies, U.S. Census Bureau, 2002. URL http://ideas.repec.org/p/cen/tpaper/2002-06.html. [p107]

L. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950. URL http://www.ams.org/journals/tran/1950-068-03/S0002-9947-1950-0051437-7. [p111]

C. Badea, S. Grivaux, and V. Müller. The rate of convergence in the method of alternating projections. *St. Petersburg Mathematical Journal*, 23:413–434, 2012. URL http://www.ams.org/journals/spmj/2012-23-03/S1061-0022-2012-01202-1. preprint in arXiv:1006.2047. [p111]

D. Bates and M. Maechler. *Matrix: Sparse and dense matrix classes and methods*, 2013. URL http://CRAN.R-project.org/package=Matrix. R package version 1.0-15. [p104]

H. H. Bauschke, F. Deutsch, H. Hundal, and S.-H. Park. Accelerating the convergence of the method of alternating projections. *Transactions of the American Mathematical Society*, 355(9):3433–3461, 2003. URL http://www.ams.org/journals/tran/2003-355-09/S0002-9947-03-03136-2. [p111, 114]

A. Carneiro, P. Guimarães, and P. Portugal. Real wages and the business cycle: Accounting for worker and firm heterogeneity. *American Economic Journal: Macroeconomics*, 4(2):133–152, Apr. 2012. doi: doi:10.1257/mac.4.2.133. URL http://www.ingentaconnect.com/content/aea/aejma/2012/00000004/00000002/art00005. [p107]

T. Cornelißen. The Stata command felsdvreg to fit a linear model with two high-dimensional fixed effects. *Stata Journal*, 8(2):170–189, 2008. URL http://www.stata-journal.com/article.html?article=st0143. [p114]

Y. Croissant and G. Millo. Panel data econometrics in R: The plm package. *Journal of Statistical Software*, 27(2), 2008. URL http://www.jstatsoft.org/v27/i02/. [p104]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.sf.net. [p113]

F. Deutsch and H. Hundal. The rate of convergence for the method of alternating projections, II. *Journal of Mathematical Analysis and Applications*, 205(2):381–405, 1997. ISSN 0022-247X. doi: 10.1006/jmaa.1997.5202. URL http://www.sciencedirect.com/science/article/pii/S0022247X97952021. [p111, 114]

J. A. Eccleston and A. Hedayat. On the theory of connected designs: Characterization and optimality. *The Annals of Statistics*, 2(6):1238–1255, 1974. URL http://www.jstor.org/stable/2958341. [p107]

S. Gaure. OLS with multiple high dimensional category variables. *Computational Statistics & Data Analysis*, 66:8–18, 2013a. ISSN 0167-9473. doi: http://dx.doi.org/10.1016/j.csda.2013.03.024. URL http://www.sciencedirect.com/science/article/pii/S0167947313001266. [p105, 106, 107, 113, 114]

S. Gaure. *lfe: Linear group fixed effects*, 2013b. URL http://CRAN.R-project.org/package=lfe. R package version 1.5-1107. [p104]

W. B. Gearhart and M. Koshy. Acceleration schemes for the method of alternating projections. *Journal of Computational and Applied Mathematics*, 26(3):235–249, 1989. ISSN 0377-0427. doi: 10.1016/0377-0427(89)90296-3. URL http://www.sciencedirect.com/science/article/pii/0377042789902963. [p111]

J. D. Godolphin and E. J. Godolphin. On the connectivity of row-column designs. *Utilitas Mathematica*, 60:51–65, 2001. [p108]

P. Guimarães. REG2HDFE: Stata module to estimate a linear regression model with two high dimensional fixed effects. Statistical Software Components, Boston College Department of Economics, 2009. URL http://ideas.repec.org/c/boc/bocode/s457101.html. [p114]

P. Guimarães and P. Portugal. A simple feasible procedure to fit models with high-dimensional fixed effects. *Stata Journal*, 10(4):628–649, 2010. URL http://www.stata-journal.com/article.html?article=st0212. [p105, 114]

I. Halperin. The product of projection operators. *Acta Scientiarum Mathematicarum (Szeged)*, 23(1-2):96–99, 1962. URL http://acta.fyx.hu/acta/showCustomerArticle.action?id=7164&dataObjectType=article. [p105]

N. J. Higham. Analysis of the Cholesky decomposition of a semi-definite matrix. In M. G. Cox and S. J. Hammarling, editors, *Reliable Numerical Computation*, chapter 9, pages 161–185. Oxford University Press, 1990. eprint: http://eprints.ma.man.ac.uk/1193. [p110]

A. Kaczmarz. Angenäherte Auflösung von Systemen linearer Gleichungen. *Bulletin International de l'Academie Polonaise des Sciences et des Lettres. Classe de Sciences Mathématiques et Naturelles.*, 35: 355–357, 1937. Série A, Sciences Mathématiques. [p106]

S. Kayalar and H. L. Weinert. Error bounds for the method of alternating projections. *Mathematics of Control, Signals and Systems*, 1:43–59, 1988. doi: 10.1007/BF02551235. URL http://link.springer.com/article/10.1007%2FBF02551235. [p111]

S. Markussen. Personal Communication, 2013. [p114]

S. Markussen and K. Røed. Social insurance networks. IZA Discussion Papers 6446, Institute for the Study of Labor (IZA), 2012. URL http://ideas.repec.org/p/iza/izadps/dp6446.html. [p104, 114]

A. Ouazad. A2REG: Stata module to estimate models with two fixed effects. Statistical Software Components, Boston College Department of Economics, 2008. URL http://econpapers.repec.org/RePEc:boc:bocode:s456942. [p114]

StataCorp. *Stata Data Analysis Statistical Software: Release 13*. StataCorp LP, College Station, TX, 2013. URL http://www.stata.com/. [p114]

S. M. Torres, P. Portugal, J. T. Addison, and P. Guimarães. The sources of wage variation: A three-way high-dimensional fixed effects regression model. IZA Discussion Paper 7276, Institute for the Study of Labor (IZA), 2013. URL http://ssrn.com/abstract=2238309. [p104, 107]

S. Urbanek. *multicore: Parallel processing of R code on machines with multiple cores or CPUs*, 2011. URL http://CRAN.R-project.org/package=multicore. R package version 0.1-7. [p114]

D. L. Weeks and D. R. Williams. A note on the determination of connectedness in an N-way cross classification. *Technometrics*, 6(3):319–324, 1964. doi: 10.1080/00401706.1964.10490188. URL http://www.tandfonline.com/doi/abs/10.1080/00401706.1964.10490188. [p107, 108]

J. M. Wooldridge. *Econometric Analysis of Cross Section and Panel Data*. The MIT Press, 2002. [p104, 110]

Simen Gaure
Ragnar Frisch Centre for Economic Research
Oslo, Norway
Simen.Gaure@frisch.uio.no