`fereg`: Fast high-dimensional **f**ixed-**e**ffects **reg**ressions in Stata

Johannes F. Schmieder*

November 2026

Abstract

Regressions with many fixed effects are central to applied research, but fitting them to large datasets can be slow. In this article, I present the commands `fereg` and `feppml`, which fit linear and Poisson pseudo-maximum-likelihood regressions with high-dimensional fixed effects. Both commands use familiar Stata syntax and pass the computation to a compiled plugin. The plugin chooses among projection, Krylov, and multigrid solvers according to the structure of the fixed effects, checks every result before posting it, and runs on several processor cores. The commands support multiple and interacted fixed effects, group-specific slopes, regression weights, robust and multiway clustered standard errors, saved fixed effects, and prediction. Two simulated examples illustrate their use. In simulated designs with up to ten million observations, `fereg` was 5–62 times faster than `reghdfe` and `feppml` 5–87 times faster than `ppmlhdfe`. Against packages in other languages, the commands were faster than `fixest` (R) in 43 of 44 comparable configurations and than `PyFixest` (Python) in 37 of 45, and competitive with `FixedEffectModels.jl` (Julia). On administrative data with 1.2 million workers and 117,000 firms, `fereg` and `feppml` were at least 2.2 times faster than every other command at every number of cores.

Keywords: high-dimensional fixed effects, AKM, Poisson pseudo-maximum likelihood, iterative solver, parallel computation

*Department of Economics, Boston University, Boston, Massachusetts, United States. Email: johannes@bu.edu. I used OpenAI's Codex and Anthropic's Claude to assist with software development, testing, and the preparation and revision of this manuscript. I take full responsibility for the code, analysis, and text, including any remaining errors.

1 Introduction

How long should a regression take when the dataset contains millions of observations and hundreds of thousands of fixed effects? For a researcher working with linked employer–employee records, trade flows, or a long panel of firms, the answer shapes the whole project. A single regression may be manageable, but a typical analysis requires many outcomes, alternative specifications, and sample restrictions. When each fit takes several minutes, examining the data and revising the analysis become expensive.

In this article, I present `fereg`, a package for fast fixed-effects estimation in Stata. Its two estimation commands are `fereg` for ordinary and weighted least squares and `feppml` for Poisson pseudo-maximum likelihood (PPML). Both follow the usual Stata arrangement of an outcome, covariates, an `absorb()` option, and a `vce()` option. Results are posted in `e()`, so a researcher can store estimates, inspect the estimation sample, test hypotheses, and obtain predictions within the same session used to prepare the data.

The statistical models are the familiar ones; the contribution is computational. Three design choices make `fereg` fast and reliable. First, the complete fit runs in a single call to a compiled plugin written in Rust, a systems programming language. Sample construction, absorption of the fixed effects, estimation, and the covariance matrix are computed together in native code, which avoids repeated transfers of large arrays between Stata and the numerical routines and allows the work to be spread across processor cores. The data must still be read from Stata once, a cost that packages in other languages do not pay (section 6.1). Second, the plugin chooses a solver according to the structure of the absorbed effects. Besides accelerated alternating projections, it offers preconditioned Krylov methods and, for worker–firm designs, a combinatorial multigrid solver that exploits the graph formed by workers and firms. For PPML, each iteration solves for the fixed effects and the coefficient update jointly, and a multigrid preconditioner can be reused as the weights change. Third, every result is checked independently before it is posted: the command verifies that the residuals are orthogonal to the absorbed effects at the requested tolerance and returns an error if they are not. For PPML, the iterations use checks adapted to the joint update, and the standard errors come from a final fit that passes the full check.

These choices combine established numerical methods with a Stata interface and automatic choices suited to different designs. Eliminating one of two fixed effects follows the reduction described by Somaini and Wolak (2016). Combinatorial multigrid builds on Koutis et al. (2011) and a Rust port (Schmieder, 2026) of the reference solver by Koutis and Miller (2026). LSMR and its pairwise preconditioner come from the `within` library (Fischer and Schröder, 2026b), which is also used by `PyFixest`. Fischer and Schröder (2026a) describe and evaluate this preconditioner, whose local approximate factorizations build on Gao et al. (2026). These are building blocks of `fereg`, rather than new algorithms introduced in this article.

The gains can be large. In a simulated worker–firm panel with one million observations and four processor cores, `fereg` fit a linear model in 0.82 seconds, compared with 14.7 seconds for `reghdfe`, and `feppml` fit a PPML model in 5.72 seconds, compared with 91.9 seconds for `ppmlhdfe`. The

advantage depends on the structure of the fixed effects, but `fereg` was faster than `reghdfe` and `ppmlhdfe` in every design and option in which the results matched, and in the worker–firm designs even on a single core. On administrative data with five million observations and more than a million workers, `fereg` estimated a model with worker and firm effects in 10.9 seconds on one core. Section 7 reports the details.

The natural alternatives depend on the reader's current workflow. Within Stata, `reghdfe` provides a broad and widely used interface for linear models with multiple fixed effects, building on a computational literature on absorption (Guimarães and Portugal, 2010; Gaure, 2013; Correia, 2016). Earlier Stata commands for two high-dimensional effects include `felsdvreg` (Cornelissen, 2008), `a2reg` (Ouazad, 2008), and `reg2hdfe` (Guimarães, 2015). For PPML, `ppmlhdfe` combines high-dimensional absorption with iteratively reweighted least squares and separation detection (Correia et al., 2020, 2019). Since Stata 19, the official `areg` and `xtreg, fe` commands also absorb several categorical variables (StataCorp, 2025b,a,c). Outside Stata, the R package `fixest` and the Python package `PyFixest` provide fast linear and Poisson fixed-effects estimation (Bergé, 2018; Bergé and McDermott, 2026; PyFixest developers, 2026), and the Julia package `FixedEffectModels.jl` fast linear estimation (Gomez, 2026). A goal of `fereg` is to make Stata competitive with these implementations, so that computation time is not a reason to move an established workflow to another language. Section 7 therefore compares `fereg` with all of these implementations.

Section 2 sets out the models, the estimation sample, and the available weights and standard errors. Section 3 documents the syntax, options, and stored results. Section 4 works through two simulated examples, a wage regression with worker and firm effects and a PPML model of bilateral trade. Section 5 explains the methods that absorb the fixed effects and how `fereg` chooses among them, section 6 describes the implementation, and section 7 reports timings. Appendix A collects the estimation formulas, and an online appendix states the exact numerical rules of the absorption methods.

2 Models and estimation

2.1 Linear regression with many fixed effects

Consider the model

$$y_i = x_i'\beta + \sum_{\ell=1}^{L} \alpha_{\ell,g_\ell(i)} + \varepsilon_i, \tag{1}$$

where x_i contains the covariates of interest and $g_\ell(i)$ identifies observation i's group in fixed-effect dimension ℓ. In a wage equation, two dimensions may identify the worker and the employer, so that worker effects capture persistent differences across workers and firm effects capture persistent differences across employers (Abowd et al., 1999). Other dimensions could identify a location, an industry, or a calendar period.

With positive regression weights w_i, `fereg` minimizes

$$\sum_{i \in \mathcal{S}} w_i \left(y_i - x_i'\beta - \sum_{\ell=1}^{L} \alpha_{\ell, g_\ell(i)} \right)^2 \tag{2}$$

over the coefficients and absorbed effects on the estimation sample $\mathcal{S}$. Setting every weight to one gives ordinary least squares (OLS). The number of groups can run into the millions even when x_i has only a few entries, so creating an indicator variable for every group is not a practical way to fit the model.

Absorption instead removes the part of the outcome and of each covariate that the fixed effects explain. By the Frisch–Waugh–Lovell theorem (Frisch and Waugh, 1933; Lovell, 1963), regressing the outcome residual on the covariate residuals gives the same slope coefficients as including the full set of indicators. With one categorical effect, the transformation is subtraction of weighted group means. With several effects, subtracting each set of means once does not complete the transformation. Except in small problems that admit a direct solution, an iterative calculation is needed; section 5 describes how `fereg` performs it.

Absorption does not create variation that the specification has removed. A covariate that is constant within worker has no separate coefficient in a model with worker effects, and a common linear trend is redundant once a full set of year effects is included. `fereg` reports such covariates as omitted.

An absorbed term can also let a continuous slope vary across groups. For example, firm-specific linear trends add $\gamma_j t_i$ alongside firm intercepts. This is more restrictive than a separate effect for every firm–year cell, which allows an unrestricted time path for each firm.

2.2 Poisson pseudo-maximum likelihood

For a nonnegative outcome, `feppml` fits the conditional mean

$$\mathbb{E}(y_i \mid x_i, g_1(i), \ldots, g_L(i)) = \mu_i, \qquad \mu_i = \exp\left(x_i'\beta + \sum_{\ell=1}^{L} \alpha_{\ell, g_\ell(i)} + o_i \right), \tag{3}$$

where o_i is an optional known offset. With prior weights q_i, the estimator maximizes the Poisson pseudo-log-likelihood $\sum_{i \in \mathcal{S}} q_i (y_i \log \mu_i - \mu_i)$. Because no logarithm of the outcome is taken, observations with $y_i = 0$ contribute to estimation.

Consistency requires a correctly specified conditional mean, not a Poisson distribution: the outcome need not be an integer, and its conditional variance need not equal its mean (Santos Silva and Tenreyro, 2006). For this reason `feppml` always reports robust or clustered standard errors. A regression of $\log y_i$ estimates a different conditional relationship and discards zero outcomes.

If a covariate is in logarithms, its coefficient is an elasticity of the conditional mean. For a binary covariate with coefficient b, switching from zero to one changes the conditional mean by $100\{\exp(b) - 1\}$ percent, holding the other terms fixed.

An exposure enters through $o_i = \log e_i$ with $e_i > 0$. When modeling event counts over observation windows of unequal length, for example, an exposure makes the expected count proportional to the length of the window. The `exposure()` option takes e_i; the `offset()` option takes o_i directly.

2.3 The estimation sample

Both commands apply `if` and `in` restrictions and then exclude observations with missing values in any required variable. Required variables include the fixed-effect and cluster identifiers as well as the outcome, covariates, weights, and any offset or exposure. Adding a cluster variable with missing values can therefore change the sample even when the regression equation is unchanged. As usual, `e(sample)` marks the observations used.

By default, singleton observations are removed iteratively. A singleton is the only observation in its group of some absorbed effect. Removing it can create a singleton in another dimension, so the removal is repeated until no singletons remain. Singletons do not affect the slope estimates, but they inflate the number of observations and can overstate the precision implied by clustered standard errors (Correia, 2015). The `keepsingletons` option retains them. Frequency weights count as literal copies for this purpose.

PPML raises the further issue of separation. In some samples no finite coefficient vector maximizes the pseudo-likelihood: it approaches its supremum as some coefficients diverge and fitted means approach zero. A group of an absorbed effect in which every outcome is zero is the simplest case; more general cases involve combinations of effects and covariates (Correia et al., 2019). By default, `feppml` detects separated observations with three methods, `separation(fe simplex ir)`, and removes them before fitting. A removal can create new singletons, which are then removed as well. Zero outcomes by themselves are never a reason to drop an observation; in the trade example below, every zero flow is retained.

2.4 Weights and standard errors

`fereg` accepts analytic, frequency, and probability weights; `feppml` accepts frequency and probability weights. Analytic weights describe relative precision, frequency weights represent repeated observations, and probability weights are inverse sampling probabilities. These choices can yield the same point estimates but differ in sample counts and covariance calculations.

For unweighted or analytically weighted OLS, the default covariance matrix assumes independent, homoskedastic (IID) errors. `vce(robust)` requests the heteroskedasticity-robust estimator with the usual HC1 small-sample correction, and probability weights select it by default. `feppml` uses robust standard errors unless clustering is requested. Both commands accept `vce(cluster` *`clustervars`*`)` with one or more cluster dimensions; several dimensions are combined with the inclusion–exclusion formula of Cameron et al. (2011).

Fixed effects and clustering answer different questions. Firm effects remove a firm-specific intercept; clustering by firm allows residuals to be correlated within a firm. Including the first does not remove the need for the second. Following `reghdfe`, `fereg` does not count absorbed effects

nested within clusters toward the absorbed degrees of freedom, because the cluster correction already accounts for them. With two categorical effects the count of absorbed degrees of freedom is exact; with more, the default count is an approximation, and `e(dof_table)` reports whether it was exact. The exact degrees-of-freedom and small-sample conventions appear in appendices A.1 and A.2.

3 The `fereg` and `feppml` commands

3.1 Syntax

`fereg` *depvar* [*indepvars*] [`if`] [`in`] [*weight*] [`, absorb(`*absvars*`) noconstant keepsingletons vce(`*vcetype*`) dof(`*policy*`) residuals(`*newvar*`) solver(`*method*`) preconditioner(`*type*`) memory(`*plan*`) poolsize(`#`) threads(`#`) deterministic tolerance(`#`) iterate(`#`) timeit`]

`feppml` *depvar* [*indepvars*] [`if`] [`in`] [*weight*] [`, absorb(`*absvars*`) keepsingletons offset(`*varname*`) exposure(`*varname*`) separation(`*methods*`) vce(`*vcetype*`) residuals(`*newvar*`) residualtype(`*type*`) xb(`*newvar*`) d(`*newvar*`) eta(`*newvar*`) mu(`*newvar*`) dropreason(`*newvar*`) `*execution_options*]

`fereg, selftest` [`verbose threads(#)`]

`fereg` accepts `aweights`, `fweights`, and `pweights`; `feppml` accepts `fweights` and `pweights`. The covariate list may contain continuous variables, numeric factor variables, interactions among factor variables, and time-series operators applied to continuous variables. It may be empty when the model contains only absorbed effects. The *execution_options* of `feppml` are those of `fereg` except `preconditioner()`.

Terms in `absorb()` are separated by spaces and follow the grammar in table 1. Categorical sources may be numeric or fixed-width string variables.

Table 1: Terms accepted by `absorb()`

Term	Meaning
`worker firm`	Separate worker and firm intercept effects.
`firm#year`	One intercept effect for each observed firm–year cell.
`firm##c.trend`	Firm intercepts and firm-specific slopes on `trend`.
`firm#c.trend`	Firm-specific slopes on `trend`, without firm intercepts.
`worker##c.(t1 t2)`	Worker intercepts and two worker-specific slopes.
`fe_firm=firm`	Firm effects, saved in the new variable `fe_firm`.

3.2 Options

Model.

`absorb(`*absvars*`)`

specifies the fixed effects and absorbed slopes (table 1). Writing *newvar*`=` before a term saves that term's normalized contribution in *newvar*; for a term with slopes, the slope contributions are saved in *newvar*`_Slope1`, *newvar*`_Slope2`, and so on.

`noconstant`

(`fereg` only) suppresses the normalized intercept that `fereg` otherwise reports alongside the absorbed effects.

`keepsingletons`

retains singleton observations, which are otherwise removed iteratively.

`offset(`*varname*`)`, `exposure(`*varname*`)`

(`feppml` only) add a known term to the linear predictor, either directly or as the logarithm of a positive exposure. The two options cannot be combined.

`separation(`*methods*`)`

(`feppml` only) selects the separation checks. The default, `fe simplex ir`, removes groups in which every outcome is zero, then applies a linear-programming check and an iterated rectifier check (Correia et al., 2019). `mu` adds a check based on fitted means. `separation(none)` disables the checks.

Standard errors.

`vce(`*vcetype*`)`

specifies the covariance estimator. *vcetype* may be `unadjusted` (IID; `fereg` only, the default without probability weights), `robust`, or `cluster` *clustervars*. Several cluster variables give multiway clustering, up to ten dimensions for `fereg` and five for `feppml`. A cluster term may be an interaction such as `state#year`.

`dof(`*policy*`)`

(`fereg` only) sets how redundant absorbed groups are counted. The default, `pairwise`, matches the default of `reghdfe`. The alternatives are `firstpair`, `continuous`, and `none` (appendix A).

Saved variables.

`residuals(`*newvar*`)`

saves the residuals. After `fereg`, this option is also required for postestimation predictions that include the absorbed effects. After `feppml`, `residualtype()` selects the `response` (default), `score`, `pearson`, `working`, `deviance`, or `anscombe` residual.

`xb()`, `d()`, `eta()`, `mu()`

(`feppml` only) save the finite-covariate part of the linear predictor including any offset, the total absorbed contribution, the complete linear predictor, and the fitted mean.

`dropreason(`*newvar*`)`

(`feppml` only) records why each observation was excluded, for instance a missing value, a singleton, or separation.

Execution.

`solver(`*method*`)`

selects the algorithm used to absorb the fixed effects. The default, `auto`, chooses among the methods described in section 5. For `fereg`, *method* may also be `map`, `it`, `squarem`, `lsmr`, or `cmg`; for `feppml`, it may be `map`, `lsmr`, or `cmg`.

`preconditioner(`*type*`)`

(`fereg` only) selects the preconditioner for LSMR: `auto` (default), `off`, `diagonal`, or `schwarz`.

`memory(auto|fast|low)`

selects a memory plan. The default chooses a plan that fits in available memory; `low` processes covariates in blocks of `poolsize(#)` columns (default 10).

`threads(#)`

sets the number of native threads. The default, `threads(0)`, lets the plugin choose.

`deterministic`

fixes the partition of work across threads and the order in which partial results are combined, for a given machine architecture and thread count. It may be slower.

`tolerance(#)`, `iterate(#)`

set the convergence tolerance (default 10^{-8}) and the iteration limit (16,000 solver iterations for `fereg`; 10,000 outer iterations for `feppml`).

`timeit`

displays timing, memory, and thread diagnostics.

3.3 Postestimation

Table 2 lists the statistics available from `predict`. Predictions are produced only within the estimation sample. Because the absorbed effects are not stored in `e()`, any prediction that includes them relies on a variable saved during estimation: the residuals after `fereg`, or one of `d()`, `eta()`, or `mu()` after `feppml`. If `d()` was saved, predictions combine it with the current values of the covariates; otherwise the saved linear predictor or mean is used as a fixed snapshot. Typing `fereg` or `feppml` without a variable list redisplays the last results, and `level()` and Stata's other display options can be changed there. Table 3 summarizes how the commands work with other Stata commands.

Table 2: Statistics available from `predict`

Command	Statistic	Result
`fereg`	`xb` (default)	Covariate part of the prediction, including the intercept.
	`xbd`	`xb` plus the absorbed contribution; requires `residuals()`.
	`d`	Absorbed contribution; requires `residuals()`.
	`residuals`	Residual; requires `residuals()`.
`feppml`	`mu` (default)	Fitted conditional mean.
	`eta`, `xbd`	Complete linear predictor.
	`xb`	Covariate part of the linear predictor, including any offset.
	`d`	Absorbed contribution; requires a saved `d()`.
	`response`, `pearson`, `working`, `deviance`, `anscombe`, `score`	Residuals of the stated type.

Notes: After `feppml` with absorbed effects, all statistics except `xb` require a saved `d()`, `eta()`, or `mu()` variable.

Table 3: Using `fereg` and `feppml` with other Stata commands

Commands	Use
`test`, `lincom`, `nlcom`	Work as usual with `e(b)` and `e(V)`.
`margins`	Works after both commands. After `feppml` with absorbed effects, the default prediction requires a saved `d()`, `eta()`, or `mu()`; otherwise specify `predict(xb)`.
`estimates store`, `table`, `replay`; `esttab`	Work as usual.
`estat summarize`, `ic`, `vce`	Work as usual.
`bootstrap:`, `jackknife:`, `statsby:`	Refit the model on each replication or group. Do not save variables with `residuals()`, `d()`, or named effects inside these prefixes, because each refit would try to create them again.
`by:`	Not allowed; use `statsby:` or `if`.
`svy:`, `suest`	Not supported.

3.4 Stored results

Both commands store the results in table 4 in `e()`. Results marked (O) are stored only by `fereg` and results marked (P) only by `feppml`.

Table 4: Stored results

Result	Meaning
Scalars	
`e(N)`	number of observations (sum of frequency weights)
`e(N_full)`	observations before singleton removal
`e(N_physical)`	physical rows retained (P)
`e(num_singletons)`	singleton observations removed
`e(separated)`	separated observations removed (P)
`e(rank)`, `e(df_m)`	rank of the model; model degrees of freedom
`e(df_r)`	residual degrees of freedom (O)
`e(df_resid)`	residual degrees of freedom (P)
`e(df_a)`	absorbed degrees of freedom
`e(N_hdfe)`	number of absorbed terms
`e(N_clust)`, `e(N_clustervars)`	clusters in the smallest dimension; number of cluster dimensions
`e(rss)`, `e(rmse)`, `e(F)`	residual sum of squares, root mean squared error, model F statistic (O)
`e(r2)`, `e(r2_a)`, `e(r2_within)`	R^2, adjusted R^2, and within R^2 (O)
`e(ll)`, `e(ll_0)`	log (pseudo)likelihood of the full and the null model
`e(deviance)`, `e(r2_p)`, `e(chi2)`	deviance, pseudo-R^2, and model Wald χ^2 (P)
`e(ic)`, `e(converged)`	iterations of the absorption method (O) or of the outer iterations (P); 1 if converged
Macros	
`e(cmd)`, `e(cmdline)`	`fereg` or `feppml`; command as typed
`e(depvar)`, `e(absvars)`	dependent variable; absorbed terms
`e(vce)`, `e(vcetype)`, `e(clustvar)`	covariance estimator and cluster variables
`e(wtype)`, `e(wexp)`	weight type and expression
`e(dofmethod)`	degrees-of-freedom policy (O)
`e(separation)`	separation methods (P)
`e(predict)`	program used to implement `predict`
`e(fereg_version)`	release of the package that produced the results
Matrices	
`e(b)`, `e(V)`	coefficient vector and variance–covariance matrix
`e(dof_table)`	number of groups, redundancies, and rank by absorbed term
Functions	
`e(sample)`	marks estimation sample

Named diagnostics show what the automatic choice actually did. After `fereg`, `e(fereg_solver_used)` names the absorption method and `e(fereg_solver_fallback)` indicates whether a fallback

was used; `feppml` stores the corresponding `e(feppml_solver_*)` results. For `feppml`, the method requested, the method used in the ordinary iterations, and the method used in the final fit can differ; the stored results name each one. Other results beginning with `e(fereg_)` or `e(feppml_)` record timing, memory, and thread counts; `ereturn list` displays the complete set.

3.5 Installation and requirements

Both commands require Stata 15 or later. The GitHub package includes native plugins for Windows x86-64, Linux x86-64, and macOS (Intel and Apple Silicon); no compiler, Rust installation, or Python installation is needed. Install the current package, version 0.5.0, directly from the repository's `main` branch:

```
net install fereg, ///
    from("https://raw.githubusercontent.com/johannes-schmieder/fereg/main/") replace
fereg, selftest
```

Alternatively, users of the community-contributed `github` installer can run:

```
net install github, from("https://haghish.github.io/github/")
github install johannes-schmieder/fereg
fereg, selftest
```

Both routes install `fereg`, `feppml`, their help files, and the native plugin. The self-test checks compatibility of the ado-files and plugin and runs a short diagnostic calculation; `fereg, selftest verbose` also reports the host and loaded plugin. Exit Stata before an update and install in a fresh session so that the previous plugin is unloaded. Because `main` changes during development, retain the installed package and its version/source information with replication materials; `e(fereg_version)` records the release with every set of results. Source code, corresponding-source archives, and installation details are available at https://github.com/johannes-schmieder/fereg. `fereg` is distributed under the MIT license. Copyright and license notices for third-party dependencies accompany the source distribution.

3.6 Limitations

The commands do not estimate instrumental-variables or GMM models, models with group-level outcomes and individual effects, generalized linear models other than PPML, or HAC and Driscoll–Kraay covariance matrices. Among the covariates, interactions between factor and continuous variables and between continuous variables are not yet supported. Such terms can be created with `generate` before estimation, as for the age polynomial in section 7.6, and group-specific slopes can be absorbed instead. When `absorb()` contains an interaction or a slope, base-level overrides such as `ib2.region` are not available. String variables that identify groups must be fixed-width rather than `strL`.

4 Examples

Both examples use simulated data so that they can be run without access to confidential records. The data-generating code and every command in this section are in the do-file that accompanies the article, which also checks the identities stated below. The output was produced with version 0.5.0 of the GitHub main-branch package on November 3, 2026; the do-file fixes both the random-number seeds and the sort seed, and the replication folder records the installed plugin and source identities.

4.1 A wage regression with worker and firm effects

The first dataset follows 200 workers for six periods at 40 firms, giving 1,200 worker–period observations. Each worker changes firm every period, so worker and firm effects are well connected. The log wage depends on a training indicator, an experience measure that varies within workers, worker and firm effects, and an independent disturbance. The data-generating coefficients are 0.08 on training and 0.04 on experience.

The basic regression absorbs worker and firm effects and clusters the standard errors by firm:

```
. fereg log_wage training experience, absorb(worker firm) ///
>     vce(cluster firm)
Fixed-effect solver: cg   Accuracy check: passed (tolerance 1e-08)

HDFE Linear regression                            Number of obs   =      1,200
Absorbing 2 HDFE groups                           F(   2,     39) =     310.83
Statistics robust to heteroskedasticity           Prob > F        =     0.0000
                                                  R-squared       =     0.9376
                                                  Adj R-squared   =     0.9219
                                                  Within R-sq.    =     0.3832
Number of clusters (firm)            =         40 Root MSE        =     0.1223

                                (Std. err. adjusted for 40 clusters in firm)
------------------------------------------------------------------------------
             |               Robust
    log_wage | Coefficient  std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
    training |   .0844506   .0080279    10.52   0.000     .0682125    .1006886
  experience |    .041101   .0019159    21.45   0.000     .0372258    .0449763
       _cons |    2.03768   .0098715   206.42   0.000     2.017713    2.057647
------------------------------------------------------------------------------

Absorbed degrees of freedom:
-----------------------------------------------------+
 Absorbed FE | Categories  - Redundant  = Num. Coefs |
-------------+---------------------------------------|
      worker |       200           1         199     |
        firm |        40          40           0  *  |
-----------------------------------------------------+
* = FE nested within cluster; treated as redundant for DoF computation
```

The line above the table names the method that absorbed the fixed effects, here conjugate gradients on the reduced system of section 5.4, and confirms that the absorbed variables passed the accuracy

check (6) at the default tolerance. The header reports the retained sample, model test, fit statistics, and cluster count. The absorbed-degrees-of-freedom table marks the firm effects as nested within the firm clusters and treats them as redundant for the absorbed degrees-of-freedom calculation. The training coefficient is 0.0845 and the experience coefficient is 0.0411, close to their generating values. In a log-wage specification, the training coefficient corresponds to a difference of 8.81 percent in the exponentiated fitted log wage. The 240 worker and firm effects are absorbed rather than listed, while the two slopes and the intercept are posted in `e(b)` and `e(V)` and are available through `_b[]` and `_se[]`. All 1,200 observations are retained. In applications the number can fall when missing values and singletons are removed, so `count if e(sample)` is worth checking before results are compared across specifications.

4.2 Saving effects and obtaining fitted values

Suppose the next step is to study residual wages or the estimated firm effects. These quantities must be requested at estimation:

```
fereg log_wage training experience, ///
    absorb(worker_fe=worker firm_fe=firm) ///
    vce(cluster firm) residuals(wage_residual)
predict double fitted_log_wage, xbd
predict double absorbed_part, d
predict double covariate_part, xb
```

Naming a term inside `absorb()` saves that effect, and `residuals()` retains what `predict` needs to reconstruct the absorbed part. Here `fitted_log_wage` includes covariates and absorbed effects, `covariate_part` contains the covariate prediction including the intercept, and their difference is `absorbed_part`. Within the estimation sample, the log wage equals the fitted log wage plus the residual. Note that `predict, xb` alone does not reconstruct log wages when the model contains absorbed effects.

Individual effects are identified only up to a normalization. Adding a constant to all worker effects and subtracting it from all firm effects in the same connected set leaves the fit unchanged. `fereg` applies the normalization described in appendix A.4; the sum of the absorbed effects is the uniquely identified object. The variance of the saved effects also includes estimation noise, so it overstates the variance of the true effects.

4.3 Changing the specification

Common extensions use familiar Stata notation:

```
fereg log_wage training experience i.year, absorb(worker firm) ///
    vce(cluster worker firm)
estimates store wage_year

generate double precision_weight = .5 + mod(worker,7)/7
fereg log_wage training experience [aw=precision_weight], ///
    absorb(worker firm) vce(cluster firm)
estimates store wage_weighted

fereg log_wage training experience, absorb(worker firm#year) ///
    vce(cluster firm)
estimates store wage_firm_year

generate double trend = year - 2015
fereg log_wage training experience, ///
    absorb(worker firm##c.trend) vce(cluster firm)
estimates store wage_trends
```

The first specification adds year indicators and clusters by both worker and firm. The second uses analytic weights; the weight here only illustrates the syntax. The third replaces the firm intercept with an unrestricted firm–year effect, written `firm#year`, without generating a separate group identifier. The fourth gives each firm a linear trend in addition to its intercept. The double hash in `firm##c.trend` includes both terms, whereas `firm#c.trend` would include only the firm-specific slopes. Absorbed slopes, like absorbed intercepts, are not reported in `e(b)`.

4.4 A PPML model of bilateral trade

The second dataset contains 20 countries observed for four years. Excluding domestic pairs gives 1,520 directed country-pair–year observations. Trade is drawn from a Poisson distribution whose mean depends on log distance, a common-border indicator, a trade-agreement indicator, and exporter–year and importer–year effects:

$$\mu_{odt} = \exp\left(\beta_1 \log \text{distance}_{od} + \beta_2 \text{border}_{od} + \beta_3 \text{agreement}_{odt} + \alpha_{ot} + \delta_{dt}\right). \tag{4}$$

The generating values are $\beta_1 = -0.55$, $\beta_2 = 0.30$, and $\beta_3 = 0.40$. The outcome is drawn from a Poisson distribution only for convenience; `feppml` requires only a correctly specified conditional mean (section 2). The variable `pair` identifies the directed country pair, and standard errors are clustered by pair:

```
. feppml trade log_distance border agreement, ///
>     absorb(exporter#year importer#year) ///
>     vce(cluster pair) d(trade_effects)
(PPML estimator converged in 5 iterations)

HDFE PPML regression                              Number of obs   =      1,520
Absorbing 2 HDFE groups                           Residual df     =        379
Statistics robust to heteroskedasticity           Wald chi2(3)    =     904.03
Deviance             =    1582.1074               Prob > chi2     =     0.0000
Log pseudolikelihood = -2270.455416               Pseudo R2       =     0.2425

Number of clusters (pair) =        380

                                (Std. err. adjusted for 380 clusters in pair)
------------------------------------------------------------------------------
             |               Robust
       trade | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
log_distance |  -.5664018   .0206168   -27.47   0.000    -.6068101   -.5259936
      border |   .2101274   .0728495     2.88   0.004     .0673451    .3529098
   agreement |   .5064423   .0589481     8.59   0.000     .3909061    .6219784
       _cons |          0  (omitted)
------------------------------------------------------------------------------

Absorbed degrees of freedom:
-----------------------------------------------------+
  Absorbed FE | Categories  - Redundant  = Num. Coefs |
--------------+--------------------------------------|
exporter#year |        80           1          79     |
importer#year |        80           4          76     |
-----------------------------------------------------+
```

The header reports convergence, the Wald test, deviance, log pseudolikelihood, and pseudo-R^2. The constant lies in the absorbed span: `feppml` fixes its reported coefficient at zero and includes the overall level in the absorbed contribution. Its degree-of-freedom count assigns one direction to this structural constant, leaving 155 in the absorbed-effects table. The cluster-adjusted residual degrees of freedom are $380 - 1 = 379$.

The estimated distance elasticity is -0.5664: a one percent increase in distance is associated with a 0.57 percent lower expected trade flow, holding the other terms fixed. The agreement coefficient of 0.5064 corresponds to a 65.94 percent higher expected flow. The border coefficient, 0.2101, differs from its generating value of 0.30 by 1.2 standard errors, ordinary sampling variation in a sample of this size. The input contains 390 zero flows, and all of them remain in the estimation sample: an observed zero is compatible with a positive conditional mean. In other data, separated observations would be removed and reported in `e(separated)`.

Clustering by directed pair allows dependence over time within a pair. It does not allow dependence across pairs with the same exporter or the same importer. If such dependence matters, the cluster variables must reflect it, for instance through multiway clustering by exporter and importer.

4.5 Predicted trade

Because the fit saved `d()`, the absorbed effects are available to `predict`:

```
predict double expected_trade, mu
predict double log_mean, eta
predict double trade_residual, response
summarize trade expected_trade if e(sample)
```

The variable `expected_trade` is the predicted conditional mean, `log_mean` is the complete linear predictor, and `trade_residual` is the observed outcome minus the predicted mean. The summary shows:

```
. summarize trade expected_trade if e(sample)

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
       trade |      1,520    1.845395    2.056264          0         23
expected_t~e |      1,520    1.845395    1.601739   .1956786   28.22882
```

The sample means of observed and predicted trade agree, both 1.8454, as the first-order conditions for the absorbed effects imply. Saving `d()` is preferable when the covariates may later change, because predictions then combine the saved absorbed contribution with the current covariates. Saving only `eta()` or `mu()` provides a snapshot of the fit. Predictions are confined to the estimation sample and cannot be produced for groups that were not observed.

4.6 A practical workflow

For routine work, start with the model, the sample restrictions, and the standard errors, and leave the computational options at their defaults. Inspect the retained sample and any omitted covariates before interpreting estimates, and request the residuals, effects, or fitted means that later steps will need, because stored estimates do not recreate them. If a regression is slow, `timeit` shows where the time is spent. `threads()` limits the number of cores used, which is useful on a shared server, and `memory(low)` reduces working memory for fits close to the machine's capacity. Neither option changes the model being estimated.

5 How `fereg` absorbs the fixed effects

Nearly all of the computing time in a regression with many fixed effects goes into one step: removing the fixed effects from the outcome and from every covariate. No single method is best for this step in every design. With one fixed effect, subtracting group means is exact and takes a single pass over the data. Worker–firm models have a graph structure that a multigrid method can exploit, and models with three or more effects call for yet another approach. The default, `solver(auto)`, therefore inspects the absorbed terms and chooses a method for each type of design, and this section

explains those methods. The choice affects how long a regression takes, not what it estimates: whichever method runs, its result must pass the same accuracy check before `fereg` reports it.

Figure 1 traces the decisions that `fereg` makes. They depend on the structure of the absorbed terms, on available memory, and on how quickly the iterations converge, but never on timing measurements. Section 5.1 describes the calculation that every method performs and the check that ends it. The subsections that follow take the designs in turn, from a single fixed effect to Poisson regression. Section 5.8 closes with the methods that can be requested instead through `solver()`. The online appendix lists every method with its iteration target and fallback (online table B.1) and states the exact numerical rules (online appendix B).

5.1 The calculation behind every fit

Collect the absorbed terms of (1) in a matrix D with one column for each group of each fixed effect. In a model with worker and firm effects, row i of D contains a one in the column of observation i's worker, a one in the column of its firm, and zeros elsewhere. An absorbed slope contributes columns that contain the slope variable instead of ones. Let W be the diagonal matrix of regression weights w_i; in an unweighted regression, W is the identity matrix. For each variable v in the regression, that is, for the outcome and for every covariate, `fereg` computes the residual from a weighted regression of v on D:

$$\tilde{v} = v - D\hat{\alpha}_v, \qquad \text{where} \qquad D'WD\,\hat{\alpha}_v = D'Wv. \tag{5}$$

The equation on the right collects the normal equations of this regression. By the Frisch–Waugh–Lovell theorem (section 2), a regression of the transformed outcome $\tilde{y}$ on the transformed covariates then gives the slope coefficients of the full model. Every linear absorption method computes these residuals. PPML computes them in its final weighted fit and solves a closely related joint system during its ordinary iterations (section 5.7).

For categorical effects, $D'WD$ has one row and one column for each group, so in a worker–firm dataset it can have millions of rows. Its entries are simple counts. A diagonal entry is the size of a group: the number of observations of a worker or of a firm, or the sum of their weights in a weighted regression. An off-diagonal entry counts the observations that two groups have in common, such as the number of periods that a worker spent at a firm. Most of these counts are zero, because each worker is observed at only a few firms. `fereg` never forms D or $D'WD$. It stores each observation's group identifiers and computes products such as $D'Wv$ as sums within groups.

The normal equations do not determine $\hat{\alpha}_v$ uniquely. Adding a constant to every worker effect and subtracting it from every firm effect leaves $D\hat{\alpha}_v$ unchanged, and one such redundancy arises in each connected set of workers and firms (appendix A.2). The residual $\tilde{v}$, however, is unique, and all methods below aim to compute it.

Exact residuals average to zero within every group: for each worker and each firm, the weighted mean of $\tilde{v}$ is zero. An iterative method instead produces an approximate residual r whose group

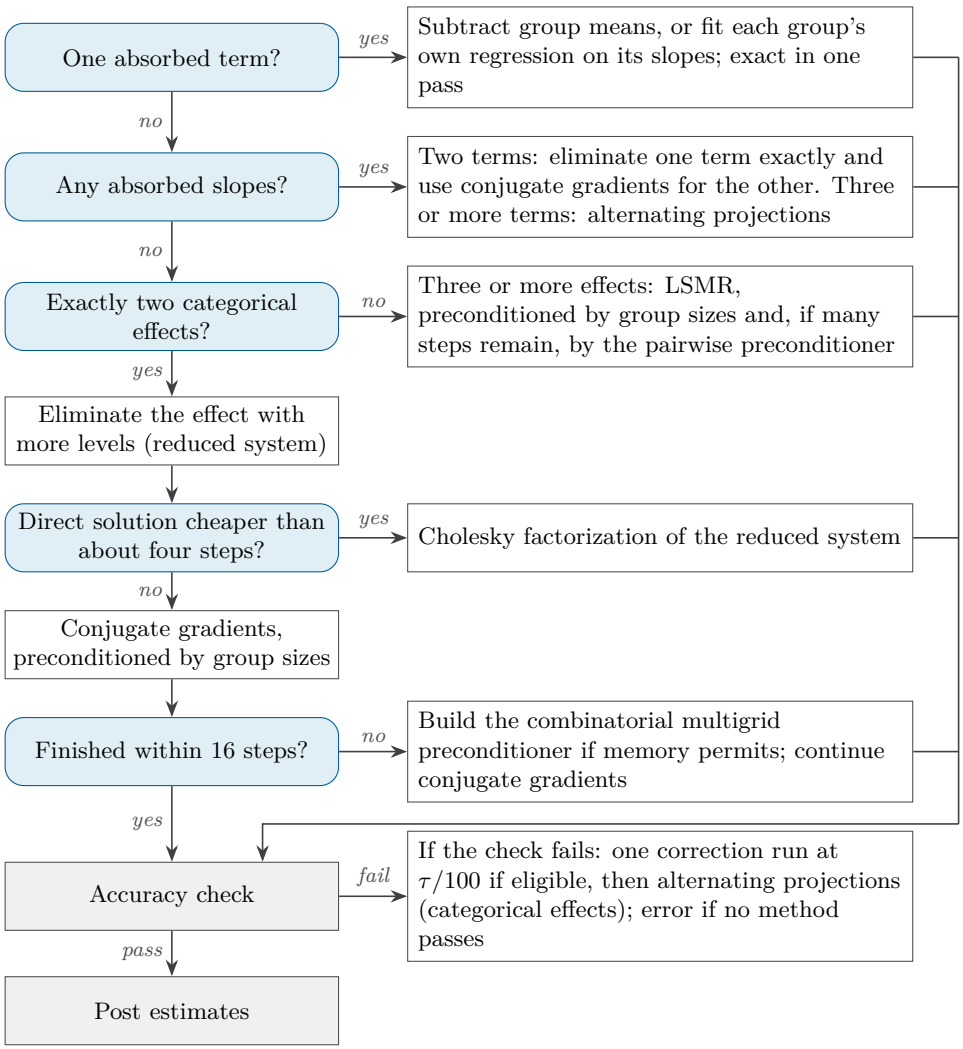


Figure 1: Decisions made by `solver(auto)` in `fereg`. Each path ends with the accuracy check (6); the command posts results only if the check passes.

means are small but not exactly zero. `fereg` accepts r only if, for every group g of every absorbed effect,

$$\frac{\sqrt{n_g}\,|\bar{r}_g|}{s} \leq \tau, \tag{6}$$

where $\bar{r}_g$ is the weighted mean of r in group g, n_g is the sum of the weights in the group (its observation count with unit weights), s is the larger of the weighted lengths of v and r, which makes the check independent of the units of v, and τ is the tolerance set by `tolerance()` (default 10^{-8}). In words, the average residual in every group must be negligible, and larger groups are held to a stricter standard. For an absorbed slope, the check also requires the residual to be uncorrelated with the slope variable within each group. `fereg` computes the check directly from the data and the weights, outside the method that produced r. The check is therefore the same for every method, and a method's own report that it has converged is never enough.[1] If no method passes the check, the command stops with an error and does not post estimates.

Evaluating (6) requires a full pass over the data, so an iterative method instead tracks its progress with an internal measure that follows the check only approximately. Each method therefore stops at a smaller target, such as $\tau/2$ or $\tau/10$, so that its result usually passes the check, and one correction is allowed if it does not (online appendix B.2). These margins affect only when a method stops, not the standard it must meet: every result is accepted or rejected by (6) at τ.

Saved results are the one case in which the standard itself is stricter. When the command saves residuals or fixed effects, or when later predictions need the absorbed effects, `fereg` computes the final residuals with 10^{-14} in place of τ (or with τ if it is smaller). These variables leave the regression and may be used for other purposes, such as a variance decomposition of the estimated worker and firm effects, so their accuracy should not depend on how accurately the slope coefficients needed to be computed.

The check bounds the remaining group means of the residuals, not the error in the coefficients directly, and two features limit what it guarantees. Because it is scaled by the larger of the lengths of v and r, a covariate that the fixed effects explain almost entirely, such as tenure with worker effects, is held to a standard that is loose relative to its small residual. And because it bounds every group separately, the remaining group means taken together can be larger than any one of them. When the firms are poorly connected, two methods that both pass the check can therefore give statistics that differ on a scale larger than τ. A smaller `tolerance()` narrows such differences, and comparing estimates at the default and a tenfold smaller tolerance is a useful sensitivity check for difficult designs and for covariates with little variation within groups. In practice, the default leaves a wide margin for inference: in three test datasets, including the largest connected set of the

[1] Two small examples illustrate the value of an independent check (online appendix B.9). Rescaling a covariate in an identified two-effect OLS model makes default `fixest` return 1.992225 where the exact coefficient in common units is 2, without a convergence warning; `fereg` and `reghdfe` recover 2. In a positive-outcome PPML example, `ppmlhdfe` and `fixest` report convergence although their exponentiated coefficients are 5.75 and 6.20 times the known exact value; `feppml` recovers the optimum. The inaccurate solutions fail independent moment or score checks, and stricter tolerances largely repair them.

administrative panel of section 7.6, no coefficient fitted at the default tolerance differed from a fit at 10^{-14} by more than about 1.3×10^{-10} of its standard error (online appendix C.3).

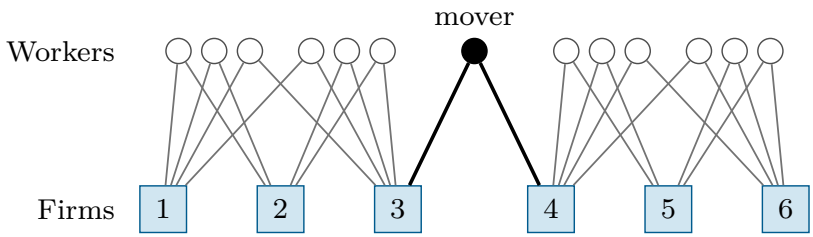
5.2 One fixed effect: subtracting group means

The simplest case, a model with a single categorical effect, needs no iteration at all. Its matrix $D'WD$ is diagonal, and the normal equations separate into one equation per group. Each group's effect is the weighted mean of v in that group, so $\tilde{v}$ is v minus its group mean, and one pass over the data computes all group means. An absorbed slope, as in `firm##c.trend`, is handled in the same way: `fereg` fits a separate weighted regression of v on a constant and the trend within each firm and keeps the residuals. Under `solver(auto)`, every model with a single absorbed term uses this exact calculation.

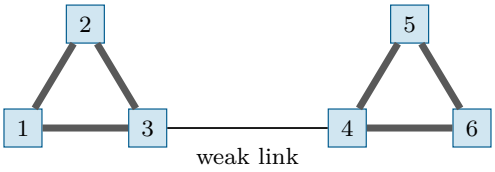
5.3 Alternating projections and why they can be slow

With a second fixed effect, the shortcut no longer works: subtracting worker means and then firm means does not finish the job, because subtracting the firm means reintroduces small differences in the worker means. The method of alternating projections repeats the two steps until the residual stops changing (Guimarães and Portugal, 2010; Gaure, 2013; Correia, 2016). `fereg` sweeps through the effects in order and then back in reverse order, which makes each round symmetric. A sweep is cheap, one pass over the data for each effect, but the number of sweeps can be very large. `fereg` uses this classic method in three roles: as a fallback, as the main method for some models with absorbed slopes, and as a brief first attempt in easy designs with three effects. Its weakness explains why the most common designs need something better.

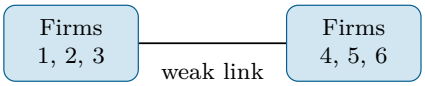
The number of sweeps depends on how well the data connect the groups. Figure 2a draws a worker–firm panel as a graph: each worker and each firm is a vertex, and a line joins a worker to every firm at which the worker is observed. Workers who change employers link firms with each other. In the figure, two clusters of firms are joined by a single mover. Suppose that the remaining error is a difference in level between the clusters, so that the current firm effects on the left are all too high relative to those on the right. Within each cluster, workers and firms are consistent with each other, so subtracting their means changes little. Only the mover's observations reveal the discrepancy, and each sweep removes only a small part of it. Low mobility, in which large groups of firms are linked by few movers, is common in linked employer–employee data, and it is exactly the case in which alternating projections can need thousands of sweeps. Fischer and Schröder (2026a) make this argument precise by relating the speed of alternating projections to a measure of how well the worker–firm graph is connected. With three or more effects, the difficulty compounds, because each step uses values of the other effects that the current sweep has not yet updated.



(a) Workers and firms



(b) Firm graph after eliminating the workers



(c) A coarse level of multigrid

Figure 2: How the connections in the data shape the computation. (a) A worker–firm panel as a graph. Squares are firms, circles are workers, and a line joins a worker to each firm at which the worker is observed. One worker (filled) moves between two clusters of firms. (b) The graph behind the reduced system (7): firms are linked when they share a worker, and thicker lines indicate more shared workers. Only the mover links the two clusters. (c) Multigrid merges each cluster of strongly linked firms into one vertex, so a difference in level between the clusters becomes a difference between two vertices.

5.4 Two fixed effects: a smaller system for the firm effects

The most common design in applied work has two sets of categorical fixed effects, such as worker and firm effects. For this case `fereg` combines three ideas. It removes one set of effects exactly, solves the much smaller system that remains by conjugate gradients, and, when the data are poorly connected, speeds up the iterations with a multigrid method that corrects errors across weakly linked clusters of firms.

Eliminating the worker effects. Split $D = [D_1\ D_2]$ into the columns for workers (D_1) and for firms (D_2), and split $\hat{\alpha}_v$ into worker effects a_1 and firm effects a_2. Three blocks of $D'WD$ enter the normal equations: the diagonal matrices $N_1 = D_1'WD_1$ and $N_2 = D_2'WD_2$, which hold the sizes of the workers and of the firms, and the worker-by-firm table $C = D_1'WD_2$, whose entry C_{jk} counts the observations of worker j at firm k. Given the firm effects, each worker's effect is the weighted mean of $v - D_2a_2$ over that worker's observations. Substituting this expression into the equations for the firms leaves a system in the firm effects alone:

$$S\,a_2 = b, \qquad S = N_2 - C'N_1^{-1}C, \qquad b = D_2'Wv - C'N_1^{-1}D_1'Wv. \tag{7}$$

The matrix S has one row for each firm, instead of one row for each worker and each firm.

This is the block elimination used by Somaini and Wolak (2016) to estimate two-way fixed-effects models. Their algorithm also eliminates the larger effect, leaving the smaller system to solve. It forms and inverts that reduced matrix explicitly, after imposing a normalization, and reuses the

result across variables and specifications with the same sample, weights, and fixed effects. This can be economical even when the initial inversion is costly. In worker–firm data, however, the smaller effect can still have hundreds of thousands of groups, making a dense inverse too large to store. `fereg` uses the same reduction but changes how it solves (7): except in small problems (see below), it works iteratively with products of S and vectors, computed from the data without forming S. Iterative solutions have precedents as well: Abowd et al. (2002) solved the normal equations of the full worker–firm model by preconditioned conjugate gradients, and Mittag (2019) solves the same reduced system by conjugate gradients or a Cholesky factorization. What distinguishes `fereg` here is its preconditioner, which exploits the graph structure described next.

The reduced matrix S has a useful interpretation. Its off-diagonal entry S_{kl} is negative when firms k and l have employed a common worker and zero otherwise, and its size depends on how many observations the shared workers have at the two firms. The entries in each row of S sum to zero. A matrix with these properties is called the Laplacian of a weighted graph. Here the graph is the one in figure 2b: firms are vertices, and two firms are linked if they share a worker. Within each connected set of firms, adding the same constant to every firm effect leaves Sa_2 unchanged; this is the redundancy noted in section 5.1. `fereg` treats each connected set separately, and the redundancy does not affect the residuals $\tilde{v}$.

Conjugate gradients. `fereg` solves (7) with the method of conjugate gradients (Hestenes and Stiefel, 1952). Like alternating projections, conjugate gradients improves an approximate solution step by step, and each step needs only the product of S with a vector, which `fereg` computes in two passes over the data without forming S. Unlike alternating projections, each step moves in a new direction, chosen so that it does not undo the progress of earlier steps. As a result, conjugate gradients usually needs far fewer steps, especially when the firms are poorly connected (Saad, 2003, ch. 6).

A preconditioner reduces the number of steps further. A preconditioner is an approximation M of S that is easy to solve. At each step, conjugate gradients works with M^{-1} applied to the remaining discrepancy $b - Sa_2$, rather than with the discrepancy itself, and the closer M is to S, the fewer steps are needed. `fereg` starts with the simplest choice, $M = N_2$, which rescales each firm's equation by the size of the firm. This preconditioner costs almost nothing and suffices when the firms are well connected. Conjugate gradients stops when its internal measure of accuracy reaches $\tau/2$, and `fereg` then reconstructs the residuals $\tilde{v}$ and applies the check (6) at τ.

Combinatorial multigrid for poorly connected data. If conjugate gradients has not finished after 16 steps, `fereg` attempts to build a stronger preconditioner, subject to available memory, and continues from the current solution. The new preconditioner is combinatorial multigrid (Koutis et al., 2011), a method designed for graph Laplacians such as S. It merges firms that are strongly linked to each other into clusters and links the clusters that share workers, which gives a smaller, coarser graph (figure 2c). It repeats this step until the graph is small enough to solve directly. Each application of the preconditioner passes the remaining discrepancy down through the coarser graphs,

solves the problem on the smallest graph, and passes the corrections back up, refining them at each level. The coarse levels handle exactly the errors that slow down the simpler methods. On the coarse graph in figure 2c, a difference in level between two weakly linked clusters of firms becomes a difference between two vertices, which the coarse solve can correct directly. Multigrid can therefore reduce sensitivity to how often workers change employers. The multigrid code in `fereg` is a Rust port (Schmieder, 2026) of the reference implementation by Koutis and Miller (2026).

Building the multigrid preconditioner takes time, so `fereg` waits for 16 steps, and designs that finish sooner avoid this cost. The preconditioner is built from a sparser version of the firm graph that keeps the links between firms on average (Kyng and Sachdeva, 2016; Gao et al., 2026); online appendix B.3 describes the construction. The approximation affects only how quickly the iterations converge, not the system (7) that they solve or the check.

When both effects have few groups, `fereg` solves (7) directly by Cholesky factorization if that is cheaper than iterating. If the residuals fail the check, `fereg` runs one correction and then, if necessary, alternating projections; if those also fail, the command stops with an error. Online appendix B.3 gives the exact rules.

5.5 Three or more fixed effects: preconditioned LSMR

Many models add a third set of effects, such as worker, firm, and year effects, or the exporter-by-year, importer-by-year, and country-pair effects of a gravity model of trade. Eliminating one set of effects then no longer leaves a single graph problem, so `fereg` switches to a different iterative method, LSMR (Fong and Saunders, 2011). LSMR is designed for least-squares problems such as the regression of v on D. Like conjugate gradients, it improves the solution step by step in directions that do not undo earlier progress. It works with D directly, through products such as Dz and $D'Wv$, rather than with the normal equations, which makes it less sensitive to rounding errors.

As with two effects, the preconditioner is what makes LSMR fast. Alternating projections update one effect at a time and ignore how the effects are linked, which is why they slow down. The pairwise preconditioner is instead built from pairs of effects; in the language of numerical analysis, it is an additive Schwarz preconditioner (Toselli and Widlund, 2005). Fischer and Schröder (2026a) describe its construction and compare it with alternating projections and other preconditioners. For each pair of effects (workers and firms, workers and years, and firms and years), and for each connected set of groups within that pair, it prepares an approximate solver for the two-effect problem of section 5.4. Each of these solvers eliminates the effect with more groups, using the same sparser graphs as above, and then factors the remaining graph Laplacian approximately (Gao et al., 2026); small remaining systems are factored exactly. Each application of the preconditioner solves all pair problems independently, which can be done in parallel, and adds up their corrections, with weights that keep groups belonging to several pairs from being counted more than once. The pair problems capture how each pair of effects is linked, which is where slow convergence originates.

Building the pairwise preconditioner is the expensive step, and many designs do not need it. Fischer and Schröder find that it speeds up convergence substantially when the effects are poorly

connected, but that building it can take most of the time when they are well connected. `fereg` therefore starts LSMR with the simplest preconditioner, the diagonal of $D'WD$, which rescales each group by its size, and builds the pairwise preconditioner only when the recent rate of progress projects many more steps. With exactly three unweighted effects that are well mixed, a few sweeps of alternating projections may already pass the check, and LSMR is then skipped.

`fereg` prepares LSMR once per regression and reuses it, including any pairwise preconditioner, for the outcome, every covariate, and the final residuals; this reuse saves the most time in models with many covariates. If the result fails the check after one correction, or if the workspace for LSMR does not fit in memory, `fereg` falls back to alternating projections (online appendix B.4).

For LSMR, `fereg` calls the `within` library (Fischer and Schröder, 2026b), developed by Kristof Schröder at the appliedAI Institute for Europe with contributors to the `PyFixest` project. `PyFixest` has offered it as an LSMR backend since version 0.50 (PyFixest developers, 2026). The library supplies the LSMR iterations, the pairwise preconditioner, its approximate local factorizations, and an adaptive strategy that starts with a diagonal preconditioner. `fereg` supplies the model data, chooses when to switch to the pairwise preconditioner, and checks the returned residuals independently.

5.6 Absorbed slopes

Group-specific slopes, such as firm-specific trends combined with worker effects, change the structure of the problem. A term with slopes gives each group a small regression of its own, with an intercept and one coefficient for each slope variable, and `fereg` solves these group regressions exactly whenever it updates the term. With exactly two absorbed terms, `fereg` applies the idea of section 5.4 to the coefficients of the two terms: it eliminates the term with more coefficients exactly and solves for the coefficients of the other term by conjugate gradients. If this result does not pass the check, the command stops with an error. When the workspace for this method does not fit in memory, or when the model has three or more absorbed terms, `fereg` uses alternating projections, in which each step is a set of exact group regressions. `solver(auto)` does not use multigrid or LSMR for models with absorbed slopes.

5.7 Poisson regression

Poisson regression multiplies the cost of absorption, because `feppml` must remove the fixed effects anew in every iteration of its fitting algorithm. `feppml` fits the model by iteratively reweighted least squares (appendix A.3). Each iteration is a weighted linear regression of a working response on the covariates and the absorbed effects, with weights $q_i\mu_i$ that change from one iteration to the next, and a fit can take dozens of iterations. Applying the methods above to the working response and to every covariate in every iteration would multiply the cost of a linear regression by the number of iterations. Three features keep the total cost down.

First, `feppml` solves each iteration's regression in one piece. To take the next step, it needs only the update of the coefficients, not the transformed covariates themselves. With two or more

absorbed terms and at least one covariate, `feppml` therefore eliminates the term with the most coefficients exactly, as in sections 5.4 and 5.6, and then solves for the remaining fixed effects and the coefficient update together by conjugate gradients. The preconditioner rescales the fixed effects by their group sizes under the current weights and treats the coefficients exactly, using the cross-product matrix of the covariates after the eliminated effect has been removed. A single iterative solve thus replaces separate absorptions of the working response and of every covariate.

Second, `feppml` matches the precision of each absorption to the progress of the fit: it starts with a loose tolerance while the weights are still far from their final values and tightens it over the first iterations. Each joint update is checked against the original data without absorbing every covariate separately: the reconstructed residual must average to nearly zero within every group and be nearly orthogonal to the covariates.

Third, once the fit appears to have converged and the fitted values have stopped moving appreciably, `feppml` solves the weighted regression once more at the strictest tolerance, this time absorbing every covariate, and confirms that the estimates do not move; if they do, it resumes iterating. At this stage each reconstructed absorption passes (6) at the strict tolerance. Standard errors use only this final, checked fit. The final tolerance is well below `tolerance()` because the convergence criteria of the outer iterations are computed from the absorbed variables: an absorption error of the same size as τ could make the fit appear converged, or not, for the wrong reason.

With two categorical effects, the most common case, `feppml` also uses combinatorial multigrid. It would help here for the same reason as in linear regression, but building a new multigrid preconditioner in every iteration would cost more than it saves. `feppml` instead builds one when an iteration's solve still runs after 64 steps, and then keeps it. The weights change from one iteration to the next, but the graph of figure 2b, which records which firms share workers, does not. A preconditioner built under earlier weights can remain useful, and `feppml` adapts it to each new iteration by rescaling it with the current group sizes. The system that conjugate gradients solves and the check always use the current weights.

The remaining cases follow the linear methods: one absorbed term uses the exact calculation of section 5.2, and a model without covariates, or an iteration whose joint solve does not fit in memory or fails its check, absorbs every variable separately with the methods above. If that also fails, the command stops with an error (online appendix B.7).

5.8 Other methods available through `solver()`

The automatic choice can be overridden with `solver()` (section 3). The alternatives are useful for comparing results across algorithms and for designs in which a particular method is known to work well. Their results must pass the same check. In `fereg`, an explicitly requested method stops with an error if its result fails the check, rather than falling back on alternating projections as `solver(auto)` does; online appendix B.8 lists the exceptions.

`solver(map)` runs alternating projections with symmetric sweeps, as described in section 5.3. `solver(it)` and `solver(squarem)` accelerate these sweeps by extrapolation: after a few sweeps,

they estimate where the sequence of residuals is heading and jump ahead, using the methods of Irons and Tuck (1969) and Varadhan and Roland (2008), respectively. `solver(lsmr)` applies LSMR to any supported design, including absorbed slopes, with the preconditioner selected by `preconditioner()`: `diagonal`, `schwarz` for the pairwise preconditioner, `off`, or the default `auto`. `solver(cmg)` treats the two largest categorical effects as a worker–firm graph and solves for all of their effects together with multigrid-preconditioned conjugate gradients, rather than eliminating one of them first. `feppml` accepts `solver(map)`, `solver(lsmr)`, and `solver(cmg)`, with the requirements stated in online appendix B.8.

6 Implementation

This section describes how `fereg` carries out the calculations of section 5: which parts run in Stata and which in compiled code, how the work is spread across processor cores, how memory is managed, and what `fereg` checks before it posts results.

6.1 Native computation and parallelism

Stata parses the command, evaluates `if`, `in`, factor variables, and time-series operators, and displays the results. A single call to the plugin then does everything else: it builds the final sample, encodes the fixed effects and clusters, removes singletons, absorbs the effects, estimates the coefficients and covariance matrix, runs the checks, and writes the requested variables. Keeping this work in one call avoids repeated transfers of large arrays between Stata and the numerical code and lets the group encodings and workspaces be reused within a fit.

One transfer remains: loading the data. Stata's plugin interface returns one value of one variable per call, so the plugin copies the model's variables into its own memory once at the start of the command. Parallel copying from Stata has precedent in the `julia` package of Roodman (2025), which uses a C++ plugin to transfer data across columns concurrently. `fereg` also reads numeric variables in parallel: several threads request values from Stata at once while Stata's main thread waits, and each thread writes only to the plugin's own memory. Checking the variables, reading string variables, and posting the results stay on Stata's main thread. Stata does not document that its plugin interface may be called from several threads, so `fereg` reads in parallel only on combinations of operating system, Stata version, edition, and interface on which this has been tested; elsewhere it reads the data serially. In the four-core linear worker–firm comparison of section 7, loading took about 11 nanoseconds per value. R, Python, and Julia pass arrays already in their own memory to compiled code directly or by a fast copy. Their command times therefore omit this Stata-to-plugin transfer; the other Stata commands also begin with data in Stata's memory.

Reuse matters especially for PPML. Iteratively reweighted least squares solves a sequence of weighted linear problems as the fitted means change (Stammann, 2018; Correia et al., 2020). The group memberships and their encodings stay fixed across iterations, so only the weighted sums must be recomputed.

Native threads divide the observations and the variables among processor cores. Their benefit depends on the amount of independent work, the size of the data, and memory bandwidth. Small regressions finish before parallelism helps much. Larger fits benefit from several cores, but not in proportion, because parts of the calculation remain sequential and threads compete for the same memory system.

6.2 Numerical libraries and reproducibility

The plugin uses the `within` and `schwarz-precond` libraries for LSMR and pairwise preconditioning, including local approximate Cholesky solvers (Fischer and Schröder, 2026b; Gao et al., 2026), and a Rust port (Schmieder, 2026) of Koutis and Miller's MATLAB/C implementation of combinatorial multigrid (Koutis et al., 2011; Koutis and Miller, 2026). `fereg` uses a fork of `within` version 0.4.0, fixed at one revision and adapted so that it works on the plugin's data without copying them, can resume and cancel its iterations, stays within a memory budget, and combines partial results in a reproducible order. The corresponding-source archive records the dependency versions used to build the plugin.

Reproducing a numerical run requires the same data, observation order, software build, options, thread count, and any random seeds used to generate the data. The graph-sampling choices of the default solver use fixed seeds. `deterministic` fixes native work partitions and reduction order for a given platform and thread count. This supports repeated runs in the same environment; it does not promise bitwise equality across thread counts, processor architectures, or software versions. Replication materials should record these settings as well as the statistical model.

6.3 Working within available memory

Retaining transformed variables and solver workspaces avoids repeated calculation at the cost of memory. `memory(auto)` estimates the memory needed by a fast and a low-memory plan before allocating and chooses one that fits. The low-memory plan processes covariates in blocks of `poolsize()` columns and can use temporary scratch files, which are removed when the command finishes. Smaller blocks use less memory but require more passes over the data. Memory use still grows with the number of observations, because the dataset, the group identifiers, and any saved variables must be held in memory, and more threads require more workspace. Before building an optional preconditioner, such as multigrid or the pairwise preconditioner, `fereg` checks separately whether its workspace fits in memory. If it does not, the automatic solver skips the preconditioner and continues with a method that fits. The result must still pass the same accuracy check. If no method that fits in memory can complete the fit, the command returns an error.

6.4 Checks before results are posted

Besides the accuracy check of section 5.1, the commands check scores, the inverse cross-product matrix, fit statistics, and the covariance matrix for finiteness and consistency. `feppml` additionally checks the objective, final weights, and saved outputs.

Long native calculations check for Stata's Break request. On an error or Break, the command rolls back its pending output and preserves the data and previous estimation results. Temporary scratch files are removed, and settings changed for the duration of the call are restored. Results and requested variables are published only after the required checks pass.

7 Computational performance

This section compares `fereg` and `feppml` with the Stata commands `reghdfe`, `ppmlhdfe`, and `areg`, with the R package `fixest`, the Python package `PyFixest`, and, for linear models, the Julia package `FixedEffectModels.jl`. Three findings stand out. The gains over the Stata fixed-effect commands are large across the tested fixed-effect structures, options, core counts, data sizes, and processors. `fereg` is competitive with the fastest implementations in other languages, which are faster in some designs, including smaller sparse designs with three effects. And its advantage is largest in the worker–firm designs for which it was built, including a large administrative panel.

7.1 Design and measurement

The measurements ran on November 2–3, 2026 on dedicated compute nodes of Boston University's Shared Computing Cluster, using version 0.5.0 of `fereg`, `reghdfe` 6.14.1 (Correia, 2026), `ppmlhdfe` 2.3.4, the `areg` command of Stata/MP 19 (licensed for four cores), `fixest` 0.14.2, a development version of `PyFixest` whose default solver is the LSMR method of the `within` library (Fischer and Schröder, 2026b), which `fereg` also uses for designs with three or more effects, and `FixedEffectModels.jl` 2.1.0 (Gomez, 2026). Every command used its default options and tolerances. Because default tolerances differ between packages, `fixest`, `PyFixest`, and `FixedEffectModels.jl` also ran with strict tolerances. Each comparison ran all commands on the same node and the same processor cores. Online appendix C.1 lists the hardware, software versions, thread settings, and tolerances.

Each configuration ran five times in fresh processes, or three times above one million observations. The reported time is the median elapsed time of the estimation command, including the standard errors and any saved fixed effects, but not loading the dataset into memory. For the Stata commands, the timed command also reads the data from Stata's memory, whereas the other languages start from data already held in their own memory (section 6.1); for `fereg` this took 0.20 seconds, or 25% of its time, in the linear worker–firm design with one million observations. A command was stopped after one hour (`reghdfe` and `ppmlhdfe`) or ten minutes (all others) and, in a series of growing datasets, not run on the larger ones. Memory is the largest resident set size of the estimating process during the command, as recorded by the operating system; it includes the data held in memory.

Ratios divide a comparator's median time by `fereg`'s on the same node; absolute times also reflect differences between nodes.

A speed comparison is meaningful only if the commands deliver the same results. Each comparator's results were therefore compared with `fereg`'s: the same sample, coefficients within a relative tolerance of 10^{-7}, covariance matrices within 10^{-6}, the same objective, and saved effects and predictions that satisfy the estimating equations to within 10^{-7}. `fereg`'s own results were checked against independent dense-matrix references and were identical across repeated runs. Packages count the degrees of freedom absorbed by the fixed effects differently: `FixedEffectModels.jl` counts every group, and `fixest` and `PyFixest` ignore disconnected parts of sparse worker–firm graphs. Such results are treated as matching when everything else agrees and the standard errors differ only by the implied small-sample factor, here always less than one percent. Where a comparator missed a check, its time is still reported but marked, and the notes give the reason.

The simulated data follow the worker–firm structure of Abowd et al. (1999). Each worker is observed for five periods and moves to a randomly chosen firm between periods with probability 0.2, so that one million observations contain 200,000 workers and 10,000 firms. The model has worker and firm effects and 16 continuous controls. Variants change the structure of the fixed effects: a single worker effect; two or three effects formed by a dense Cartesian design; and sparse designs with two effects, or three with an added 32-group effect, in which a mobility of 0.02 splits the worker–firm graph into hundreds of disconnected pieces. Labor markets are less well mixed than this design. A clustered-mobility variant therefore draws firm sizes from a heavy-tailed Pareto distribution and lets workers move with probability 0.1, keeping 95% of moves within one of 50 regions, so that the firms form weakly linked clusters. The PPML outcomes of these designs are positive, so no observations are separated. A gravity design adds the leading application of PPML, with many zeros: trade flows between 201 countries over 25 years, with exporter-year, importer-year, and country-pair effects. Of its one million flows, 47% are zero; all flows of 3% of the exporter-years are zero, and one control equals one only for some zero flows, so that 30,000 observations are separated by the fixed effects and 10,000 by the control. Datasets range from 100,000 to ten million observations.

7.2 Comparison with Stata commands

`fereg` and `feppml` were faster than the Stata commands in every configuration with matching results. Table 5 reports the worker–firm design with one million observations and four cores. In the linear model, `fereg` took 0.82 seconds, `reghdfe` 14.7 seconds, about 18.0 times as long, and `areg` 9.0 seconds, 11.0 times as long. For PPML, `feppml` took 5.72 seconds and `ppmlhdfe` 91.9 seconds, 16.1 times as long. Robust or clustered standard errors and saved fixed effects give similar ratios in the worker–firm design. Across all configurations of the study with matching results, `reghdfe` took between 5 and 62 times as long as `fereg`, `ppmlhdfe` between 5 and 87 times, and `areg` between 2 and 108 times. The largest ratios arise when the worker–firm graph is sparse or clustered and when the data are large (section 7.5). In the clustered-mobility design, for example, `reghdfe` took 44.3 and `ppmlhdfe` 46.9 times as long as `fereg` and `feppml`. On the separate Granite

Rapids node, `reghdfe` and `ppmlhdfe` took 15.4 and 21.1 times as long as `fereg` and `feppml` in the baseline worker–firm models (online table C.1). These are comparisons within each host; differences between hosts also reflect their memory systems and ordinary system activity.

In the worker–firm design, the shorter running time comes with similar or lower memory use: with one million observations, `fereg` used 0.62 GiB at its peak, `reghdfe` 0.76 GiB, and `areg` 1.07 GiB. For PPML, `feppml` used 1.30 GiB and `ppmlhdfe` 1.27 GiB. Both figures include the data that Stata holds in memory. The one design in which `fereg` needed clearly more memory than `reghdfe` is the sparse design with three effects: 1.3 GiB against 0.9 GiB with one million observations.

7.3 Fixed effects, options, and other languages

Figure 3 shows, for each design and option, how long each comparator took relative to `fereg`; values above one mean that `fereg` was faster. Hollow markers denote comparators whose results differ from `fereg`'s by more than the tolerances; their times are still informative, but they did not compute the same quantities. This happens for four reasons. The default tolerance of `FixedEffectModels.jl` is too loose for the accuracy check in the sparse and clustered designs, and that of `fixest` in the sparse two-effect linear design. With saved effects, the saved predictions of every comparator at its default tolerance missed the accuracy check; with strict tolerances, those of `fixest` and, for PPML, of `PyFixest` met it. In the gravity design, `fixest` and `PyFixest` keep the observations that a control separates (see below). And with clustered standard errors, `areg` does not discount the worker effects nested within the worker clusters.

Table 6 summarizes the comparisons with the packages in other languages over all configurations with matching results. `fixest` at its default tolerances was slower than `fereg` in every such configuration except the PPML worker–firm model with four controls. `PyFixest` took 2.7 times as long as `fereg` in the linear worker–firm design and 2.0 times for PPML, but was faster in several dense and three-effect designs, including the sparse linear size series below ten million observations; with strict tolerances, its advantage remained in some three-effect designs. `FixedEffectModels.jl` took 2.0 times as long as `fereg` in the worker–firm design, but was faster with one effect (0.71) and with three dense effects (0.84). Its default tolerance missed the accuracy check in sparse designs; with strict tolerances it matched, taking 7.1 to 9.6 times as long as `fereg` in the two sparse four-core designs.

The gravity design shows how the commands treat separation. `feppml` and `ppmlhdfe` dropped the same 4% of the observations, those separated by the fixed effects and those separated by the control, and `ppmlhdfe` took 5.1 times as long as `feppml`. `fixest` and `PyFixest` drop only the observations separated by the fixed effects. They keep those separated by the control and report a large negative coefficient for it, which grows with a stricter tolerance; their samples therefore differ from `feppml`'s, although their other coefficients agree. With these differences, they took 0.82 and 0.89 times as long as `feppml`.

The requested output matters more for some commands than for others. Saving the worker and firm effects added 31% to `fereg`'s time in the linear model and 8% in PPML, because the effects are

Table 5: Time and memory against Stata commands

Inference and output	Median seconds (times slower than `fereg`)			Peak memory (GiB)		
	`fereg`	`reghdfe/ppmlhdfe`	`areg`	`fereg`	`reghdfe/ppmlhdfe`	`areg`
Linear regression						
IID	0.82	14.72 (18.0×)	9.01 (11.0×)	0.62	0.76	1.07
Robust	0.93	14.92 (16.1×)	9.50 (10.3×)	0.63	0.76	1.07
Clustered by worker	0.92	15.15 (16.5×)	9.65 (10.5×)[a]	0.64	0.76	1.08
IID, saved effects	1.07	16.62 (15.5×)[b]	—	0.70	0.76	—
Poisson regression (PPML)						
Robust	5.72	91.90 (16.1×)	—	1.30	1.27	—
Clustered by worker	5.79	91.02 (15.7×)	—	1.32	1.27	—
Robust, saved effects	6.15	94.29 (15.3×)[b]	—	1.30	1.27	—

Notes: Simulated worker–firm panel with one million observations, 16 controls, 20% mobility, and worker and firm effects; four cores for every command, each fixed-effect model on one node. Medians of five fresh-process runs; parentheses give the comparator's time divided by `fereg`'s. Every command starts from the data in Stata's memory, and every time includes reading them. `areg` cannot save individual fixed effects (—). Peak memory is the largest resident set size of the estimating process during the command, including the data held in memory. [a] Does not discount fixed effects nested within clusters, so standard errors differ. [b] Saved fixed effects or predictions missed the 10^{-7} accuracy check.

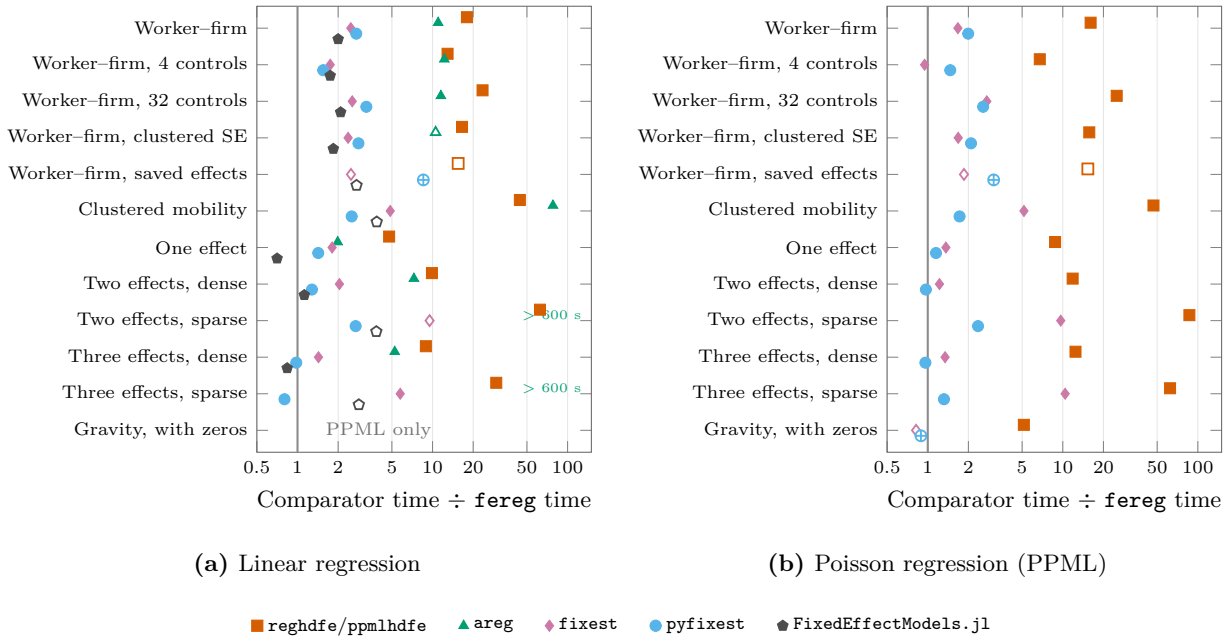


Figure 3: Running time relative to `fereg` by design and option. One million observations, four cores, and IID (linear) or robust (PPML) standard errors unless noted; worker–firm rows vary the number of controls (16 by default), the standard errors, and saved effects. Values above one mean that `fereg` was faster; the horizontal axis is logarithmic. Hollow markers: the comparator's results differ by more than the tolerances (see text). `areg` cannot save individual effects. `areg` and `FixedEffectModels.jl` were run for linear models only.

Table 6: Packages in other languages: time relative to `fereg`

		Linear regression		Poisson regression (PPML)	
Package	Tolerance	`fereg` faster	Time ratio	`fereg` faster	Time ratio
`fixest`	default	24 of 24	1.4–6.9	19 of 20	0.95–10.4
	strict	25 of 25	1.7–38.6	7 of 7	1.4–18.6
`PyFixest`	default	19 of 25	0.62–4.8	18 of 20	0.96–3.1
	strict	22 of 25	0.75–6.7	9 of 9	1.6–3.6
`FixedEffectModels.jl`	default	15 of 17	0.71–2.7	—	—
	strict	23 of 25	0.71–10.2	—	—

Notes: Simulated configurations (online table C.4) in which the package's results match `fereg`'s, exactly or up to the degrees-of-freedom convention. "`fereg` faster" counts the configurations in which the package took longer than `fereg`. The time ratio divides the package's median time by `fereg`'s; values below one mean that the package was faster. `FixedEffectModels.jl` was run for linear models only (—).

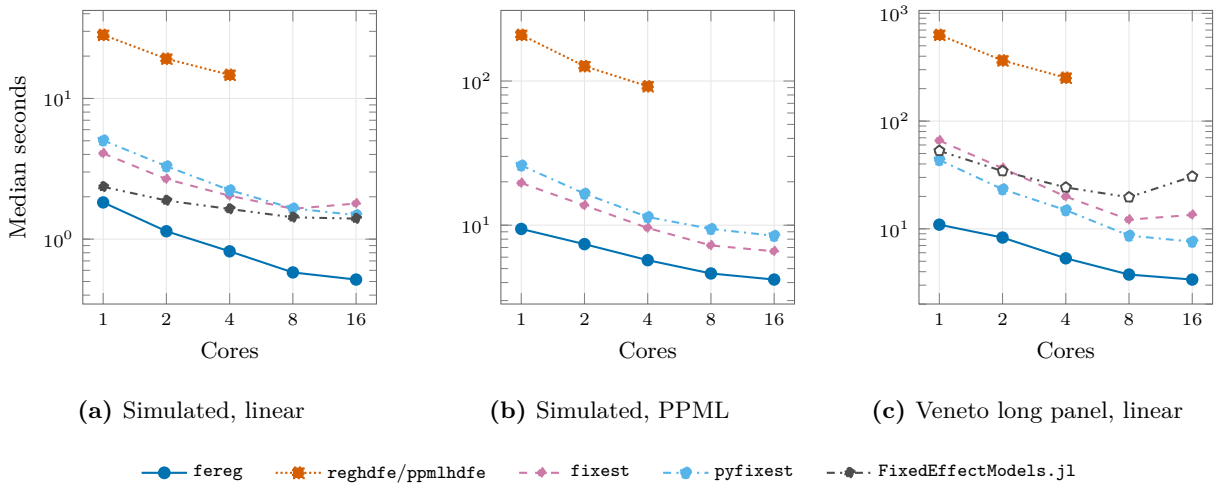
recovered from the solution already computed. `PyFixest` recovers them in a separate step, which multiplied its time by 4.1 in the linear model and 1.7 in PPML, and at its default tolerance its saved predictions did not meet the accuracy check. More controls make every command slower. In the linear model, `fereg`'s time grew from 0.49 seconds with 4 controls to 1.29 with 32. `reghdfe` took 12.9 times as long as `fereg` with 4 controls and 23.5 times with 32, and `fixest` 1.7 and 2.5 times. With 32 controls, 0.32 of `fereg`'s 1.29 seconds went to reading the data from Stata's memory, which the timings of the other languages do not include (section 6.1). For PPML, the ratio for `ppmlhdfe` grew from 6.8 to 25.1 as the number of controls rose from 4 to 32. This is the pattern expected from section 5.7: the ordinary iterations of `feppml` update the coefficients jointly with the fixed effects instead of absorbing every covariate.

7.4 Scaling with processor cores

Figure 4 shows how running time changes with the number of cores. In the simulated linear model, `fereg`'s time falls from 1.82 seconds with one core to 0.52 with sixteen, and for PPML from 9.42 to 4.20 seconds: speedups of 3.5 and 2.2, respectively. Most of the gain is reached with four to eight cores. Reading the data from Stata also runs in parallel (section 6.1): it took 0.31 seconds with one core and 0.04 with sixteen.[2] The remainder of the native computation became 3.5 times faster.

The advantage over the Stata commands is not a consequence of parallelism: with a single core `reghdfe` took 15.5 times as long as `fereg` and `ppmlhdfe` 22.2 times. `fixest` and `PyFixest` took between 1.6 and 3.5 times as long as `fereg` at every number of cores, while `FixedEffectModels.jl` gained less from additional cores: it took 1.29 times as long as `fereg` with one core and 2.71 times with sixteen.

[2] A separate test on an Apple M2 Ultra Mac isolates the effect of parallel reading: it holds estimation at 20 threads and switches only between serial and parallel reading of the data. With 20 million observations, 50 controls, and one fixed effect, median loading time fell from 4.60 to 0.96 seconds, a speedup of 4.8, and whole-command time from 10.66 to 6.98 seconds. These are medians of six alternating pairs after one warmup run of each reading method, using the same binary and data already in Stata's memory. Coefficients and covariance matrices agree exactly between the two methods and pass independent checks against `areg`. The Mac has 16 performance cores and eight efficiency cores, and the measurements ran alongside ordinary desktop activity; they concern this Mac and specification.



(a) Simulated, linear **(b)** Simulated, PPML **(c)** Veneto long panel, linear

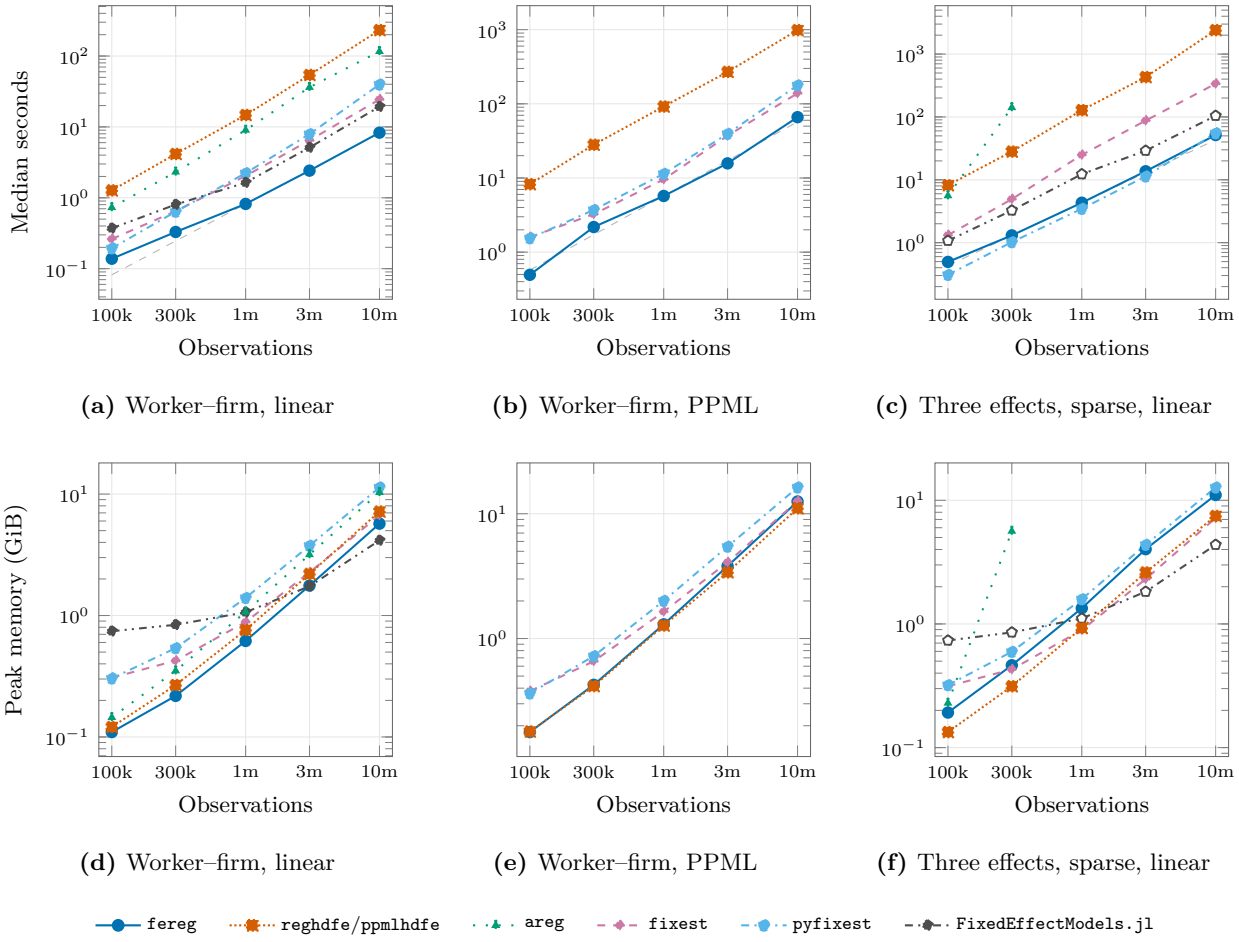
Figure 4: Running time and processor cores. Panels (a) and (b): simulated worker–firm design with one million observations; panel (c): the largest connected set of the Veneto panel (table 7, panel A). Medians of five (a, b) or three (c) runs; both axes are logarithmic. The Stata license allowed at most four cores. Hollow markers: results differ by more than the tolerances. `FixedEffectModels.jl` was run for linear models only.

7.5 Scaling with the number of observations

Figure 5 varies the number of observations from 100,000 to ten million; the numbers of workers and firms grow in proportion. In the linear worker–firm design, `fereg`'s time grows roughly in proportion to the data, from 0.14 seconds to 8.3 seconds. The Stata commands fall further behind as the data grow: `reghdfe` took 9 times as long as `fereg` with 100,000 observations and 28 times with ten million, and `areg` 5 and 14 times. `PyFixest`'s ratio grows from 1.4 to 4.8 and that of `fixest` from 1.9 to 2.9, while that of `FixedEffectModels.jl` changes little (2.7 to 2.3). For PPML, `feppml` took 66.1 seconds with ten million observations; `ppmlhdfe` finished in 986 seconds, within its one-hour limit.

Memory also grows in proportion to the data. With ten million observations, `fereg` needed 5.7 GiB in the linear model, against 7.2 GiB for `reghdfe`, 10.3 GiB for `areg`, 6.7 GiB for `fixest`, 11.3 GiB for `PyFixest`, and 4.2 GiB for `FixedEffectModels.jl`. In PPML, `feppml` needed 12.5 GiB, `fixest` 12.6 GiB, and `PyFixest` 16.3 GiB.

The sparse design with three effects has a different time–memory tradeoff. Through three million observations, `PyFixest` was the fastest command, taking between 0.62 and 0.83 times as long as `fereg`. At three million observations, `fereg` needed 4.0 GiB of memory, more than `reghdfe`, `fixest`, and `FixedEffectModels.jl` (at most 2.6 GiB) but less than `PyFixest` (4.4 GiB). At ten million, all three `fereg` repetitions completed, with a median of 51.6 seconds, compared with 55 seconds for `PyFixest`, a 6.5% lower median. The results agree up to the degrees-of-freedom convention. Peak memory was 11.04 GiB for `fereg` and 12.71 GiB for `PyFixest`, against 7.19 GiB for `fixest` and 4.38 GiB for `FixedEffectModels.jl` with strict tolerance. In this design, `fereg` trades memory for speed against these two commands.



(a) Worker–firm, linear **(b)** Worker–firm, PPML **(c)** Three effects, sparse, linear

(d) Worker–firm, linear **(e)** Worker–firm, PPML **(f)** Three effects, sparse, linear

Figure 5: Running time (top) and peak memory (bottom) by number of observations. Simulated designs with four cores and IID (linear) or robust (PPML) standard errors; each series ran on one node. The dashed gray line grows in proportion to the data from `fereg`'s time with one million observations. Lines end where a command reached its limit: one hour for `reghdfe`/`ppmlhdfe`, ten minutes for the other commands. Hollow markers: results differ by more than the tolerances. `areg` and `FixedEffectModels.jl` were run for linear models only.

7.6 Worker and firm effects in administrative data

The final comparison uses administrative social security records on private-sector employment in Veneto, the data used by Kline et al. (2020) to estimate worker and firm effects. Both panels of table 7 use the largest connected set for 1996–2001, after dropping singletons: 5,509,415 worker-years, 1,182,935 workers and 117,208 firms. Both include worker and firm effects and seven controls (polynomial age terms and year indicators). Panel A estimates OLS for log daily wages with IID standard errors. Panel B estimates PPML for the same daily wages in levels with robust standard errors. Each panel ran on a separate Skylake node.

OLS took 10.9 seconds with one core and 3.4 with sixteen; PPML took 61.1 and 21.5 seconds. Every completed comparator took longer at the same core count. `PyFixest` took 4.0 and 2.2 times as long in OLS, and 4.7 and 2.2 times in PPML. Within its one-hour limit, `reghdfe` finished at one core and took 58 times as long as `fereg`. At four cores, `reghdfe` and `ppmlhdfe` took 47 and 35 times as long as `fereg` and `feppml`.

The accuracy qualifications differ by model. At their default tolerances, `FixedEffectModels.jl` missed the OLS coefficient and objective checks, and `fixest` missed the PPML covariance check at every core count. With strict tolerances, `FixedEffectModels.jl` matched up to the degrees-of-freedom convention and `PyFixest` matched in both models. `fixest` with strict tolerances passed a separate PPML accuracy test on a short extract of the panel and matched `feppml` in the timed runs it completed, but reached the 600-second limit at one and two cores. The table reports the default timings even where the results differ; its notes report the strict-tolerance comparisons and timeouts. Peak memory for `fereg` across both models was at most 4.7 GiB; the OLS Julia runs used 1.5–1.9 GiB. Memory and time comparisons between the OLS and PPML panels reflect different estimators, outcome scales, and hosts.

These comparisons describe specific designs, on Broadwell, Skylake, and Granite Rapids hardware and with three to five runs per configuration, and are not a universal ranking.

Table 7: OLS and PPML in the Veneto largest connected set

Cores	fereg	PyFixest	FixedEffectModels.jl	fixest	reghdfe/ppmlhdfe
Panel A. OLS: log daily wages					
1	10.93	43.54 (4.0×)	52.92 (4.8×)[a]	65.93 (6.0×)	631 (57.8×)
2	8.30	23.32 (2.8×)	34.33 (4.1×)[a]	36.55 (4.4×)	365 (44.0×)
4	5.33	14.91 (2.8×)	24.22 (4.5×)[a]	19.92 (3.7×)	253 (47.4×)
8	3.76	8.66 (2.3×)	19.63 (5.2×)[a]	12.16 (3.2×)	—
16	3.38	7.60 (2.2×)	30.59 (9.0×)[a]	13.48 (4.0×)	—
Panel B. PPML: daily wages in levels					
1	61.09	289 (4.7×)	—	369 (6.0×)[a]	2015 (33.0×)
2	39.37	157 (4.0×)	—	202 (5.1×)[a]	1336 (33.9×)
4	27.64	89.87 (3.3×)	—	122 (4.4×)[a]	966 (34.9×)
8	21.70	54.86 (2.5×)	—	76.40 (3.5×)[a]	—
16	21.52	47.03 (2.2×)	—	80.42 (3.7×)[a]	—

Notes: Both panels use the largest connected set without singletons: 5,509,415 observations, 1,182,935 workers and 117,208 firms, with worker and firm effects and 7 controls. Panel A uses log daily wages and IID standard errors; panel B uses the same outcome in levels and robust standard errors. Medians of three fresh-process runs in seconds; parentheses divide the comparator's time by `fereg`'s. Each panel ran on a separate 28-core Xeon Gold 6132 node. `fereg`'s times include reading Stata's memory; other languages start with their own arrays (section 6.1). Stata was licensed for four cores; `FixedEffectModels.jl` ran only OLS (—). Strict tolerances (not shown) give matching completed fits, with time ratios `fixest` (OLS) 6.7–13.1, `PyFixest` (OLS) 2.8–5.4, `FixedEffectModels.jl` (OLS) 11.5–25.0, `fixest` (PPML) 11.2–14.4, `PyFixest` (PPML) 3.2–7.8. With strict tolerances, `fixest` passes a separate PPML accuracy test on a short extract of the panel but reaches the 600-second limit at one and two cores. [a] Coefficients or covariance entries differ by more than the tolerance.

8 Conclusions

`fereg` and `feppml` bring fast linear and Poisson estimation with high-dimensional fixed effects into a familiar Stata workflow. Researchers can absorb several and interacted effects and group-specific slopes, choose weights and multiway clustered standard errors, retain the effects and predictions needed for later analysis, and use standard postestimation commands. The two examples can be reproduced from the accompanying do-file.

The speed comes from solvers matched to the structure of the absorbed effects, including combinatorial multigrid for worker–firm graphs, and from compiled code that runs the whole fit in one call across several processor cores. Every result is checked numerically before it is posted. Across the simulated designs, `fereg` and `feppml` were many times faster than `reghdfe`, `areg`, and `ppmlhdfe`, mostly with similar or lower memory use, and the advantage remained on a single core and grew with the size of the data. They were also faster than `fixest` in 43 of 44 comparable configurations. `PyFixest` and `FixedEffectModels.jl` were faster in some designs, `PyFixest` in particular in smaller sparse designs with three effects, but `fereg` was faster than both in the worker–firm designs, including one with clustered mobility, and at every number of cores on the administrative panel.

Faster estimation makes it easier to work through alternative specifications and larger datasets without leaving Stata. The statistical choices remain the researcher's: which effects belong in the

model, what variation identifies the coefficients, which observations enter the sample, and what dependence the standard errors allow.

Acknowledgments

I thank the developers of `reghdfe`, `ppmlhdfe`, and `PyFixest`, whose implementations provided valuable reference results during development and validation. `fereg` directly uses the `within` numerical library, developed by Kristof Schröder and its contributors, and a Rust port of Koutis and Miller's CMG implementation. Their contributions, and the approximate-factorization methods on which the preconditioners build, are credited in sections 5 and 6. The comparison commands themselves are not runtime dependencies of `fereg`.

References

John M. Abowd, Francis Kramarz, and David N. Margolis. High wage workers and high wage firms. *Econometrica*, 67(2):251–333, 1999. doi: 10.1111/1468-0262.00020.

John M. Abowd, Robert H. Creecy, and Francis Kramarz. Computing person and firm effects using linked longitudinal employer–employee data. Longitudinal Employer–Household Dynamics Technical Paper TP-2002-06, U.S. Census Bureau, 2002.

Laurent Bergé. Efficient estimation of maximum likelihood models with multiple fixed-effects: The R package FENmlm. CREA Discussion Paper 2018-13, Center for Research in Economic Analysis, University of Luxembourg, 2018.

Laurent Bergé and Grant McDermott. *Fast Fixed-Effects Estimation: Short Introduction*. fixest documentation, 2026. URL https://lrberge.github.io/fixest/articles/fixest_walkthrough.html. Version 0.14.2; accessed October 7, 2026.

A. Colin Cameron, Jonah B. Gelbach, and Douglas L. Miller. Robust inference with multiway clustering. *Journal of Business & Economic Statistics*, 29(2):238–249, 2011. doi: 10.1198/jbes.2010.07136.

Thomas Cornelissen. The Stata command felsdvreg to fit a linear model with two high-dimensional fixed effects. *Stata Journal*, 8(2):170–189, 2008. doi: 10.1177/1536867X0800800202.

Sergio Correia. Singletons, cluster-robust standard errors and fixed effects: A bad mix. Technical note, Duke University, 2015. URL http://scorreia.com/research/singletons.pdf.

Sergio Correia. A feasible estimator for linear models with multi-way fixed effects. Working paper, Duke University, 2016. URL https://www.scorreia.com/research/hdfe.pdf.

Sergio Correia. reghdfe 6.14.1 help file. Version 6.14.1, GitHub commit 29b2b20, 2026. URL https://github.com/sergiocorreia/reghdfe/blob/29b2b203f534413369d675d0c0f674d61291f536/src/reghdfe.sthlp. Accessed August 30, 2026.

Sergio Correia, Paulo Guimarães, and Thomas Zylkin. Verifying the existence of maximum likelihood estimates for generalized linear models. *arXiv preprint arXiv:1903.01633*, 2019. doi: 10.48550/arXiv.1903.01633.

Sergio Correia, Paulo Guimarães, and Thomas Zylkin. Fast poisson estimation with high-dimensional fixed effects. *Stata Journal*, 20(1):95–115, 2020. doi: 10.1177/1536867X20909691.

Alexander Fischer and Kristof Schröder. Graph preconditioning for high-dimensional fixed effects regression. Working paper, draft of June 2026, GitHub commit bd77e87, 2026a. URL https://github.com/py-econometrics/within-paper. Accessed October 28, 2026.

Alexander Fischer and Kristof Schröder. within: Fast solvers for high-dimensional fixed effects. Rust crate and Python package, version 0.4.0, 2026b. URL https://github.com/py-econometrics/within. appliedAI Institute for Europe; accessed October 26, 2026.

David Chin-Lung Fong and Michael A. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5):2950–2971, 2011. doi: 10.1137/10079687X.

Ragnar Frisch and Frederick V. Waugh. Partial time regressions as compared with individual trends. *Econometrica*, 1(4):387–401, 1933. doi: 10.2307/1907330.

Yuan Gao, Rasmus Kyng, and Daniel A. Spielman. AC(k): Robust solution of Laplacian equations by randomized approximate Cholesky factorization. *SIAM Journal on Scientific Computing*, 48(3):A1643–A1684, 2026. doi: 10.1137/24M1673577.

Simen Gaure. OLS with multiple high dimensional category variables. *Computational Statistics & Data Analysis*, 66:8–18, 2013. doi: 10.1016/j.csda.2013.03.024.

Matthieu Gomez. FixedEffectModels.jl: Fast estimation of linear models with high-dimensional categorical variables. Julia package, version 2.1.0, 2026. URL https://github.com/FixedEffects/FixedEffectModels.jl. Accessed October 27, 2026.

Paulo Guimarães and Pedro Portugal. A simple feasible procedure to fit models with high-dimensional fixed effects. *Stata Journal*, 10(4):628–649, 2010. doi: 10.1177/1536867X1101000406.

Paulo Guimarães. REG2HDFE: Stata module to estimate a linear regression model with two high dimensional fixed effects. Statistical Software Components, Boston College Department of Economics, 2015. Distribution date 28 March 2015.

Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952. doi: 10.6028/jres.049.044.

Bruce M. Irons and Robert C. Tuck. A version of the Aitken accelerator for computer iteration. *International Journal for Numerical Methods in Engineering*, 1(3):275–277, 1969. doi: 10.1002/nme.1620010306.

Patrick Kline, Raffaele Saggio, and Mikkel Sølvsten. Leave-out estimation of variance components. *Econometrica*, 88(5):1859–1898, 2020. doi: 10.3982/ECTA16410.

Ioannis Koutis and Gary L. Miller. CMG: Combinatorial multigrid solver. MATLAB and C code, GitHub commit 19752fc, 2026. URL https://github.com/ikoutis/cmg-solver. Accessed October 26, 2026.

Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115(12):1638–1646, 2011. doi: 10.1016/j.cviu.2011.05.013.

Rasmus Kyng and Sushant Sachdeva. Approximate Gaussian elimination for Laplacians—fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582, 2016. doi: 10.1109/FOCS.2016.68.

Michael C. Lovell. Seasonal adjustment of economic time series and multiple regression analysis. *Journal of the American Statistical Association*, 58(304):993–1010, 1963. doi: 10.1080/01621459.1963.10480682.

Nikolas Mittag. A simple method to estimate large fixed effects models applied to wage determinants. *Labour Economics*, 61:101766, 2019. doi: 10.1016/j.labeco.2019.101766.

Amine Ouazad. A2REG: Stata module to estimate models with two fixed effects. Statistical Software Components, Boston College Department of Economics, 2008. Distribution date 11 June 2008.

PyFixest developers. *PyFixest: Fixed-Effects Estimation in Python*. PyFixest documentation, 2026. URL https://pyfixest.org/. See the feols and fepois function references; accessed October 7, 2026.

David Roodman. Julia as a universal platform for statistical software development. *Stata Journal*, 25(2): 255–284, 2025. doi: 10.1177/1536867X251341105.

Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 2003.

J. M. C. Santos Silva and Silvana Tenreyro. The log of gravity. *The Review of Economics and Statistics*, 88 (4):641–658, 2006. doi: 10.1162/rest.88.4.641.

Johannes F. Schmieder. CMG: Combinatorial multigrid in Rust. Rust crate, GitHub commit 5c5b5b2, 2026. URL https://github.com/johannes-schmieder/CMG. Accessed October 29, 2026.

Paulo Somaini and Frank A. Wolak. An algorithm to estimate the two-way fixed effects model. *Journal of Econometric Methods*, 5(1):143–152, 2016. doi: 10.1515/jem-2014-0008.

Amrei Stammann. Fast and feasible estimation of generalized linear models with high-dimensional k-way fixed effects. arXiv preprint arXiv:1707.01815, 2018.

StataCorp. *Stata 19 Base Reference Manual: areg*. Stata Press, College Station, TX, 2025a. URL https://www.stata.com/manuals/rareg.pdf.

StataCorp. High-dimensional fixed effects in stata 19. Stata 19 feature overview, 2025b. URL https://www.stata.com/new-in-stata/high-dimensional-fixed-effects/. Accessed August 30, 2026.

StataCorp. *Stata 19 Longitudinal-Data/Panel-Data Reference Manual: xtreg*. Stata Press, College Station, TX, 2025c. URL https://www.stata.com/manuals/xtxtreg.pdf.

Andrea Toselli and Olof B. Widlund. *Domain Decomposition Methods—Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer, Berlin, 2005. doi: 10.1007/b137868.

Ravi Varadhan and Christophe Roland. Simple and globally convergent methods for accelerating the convergence of any EM algorithm. *Scandinavian Journal of Statistics*, 35(2):335–353, 2008. doi: 10.1111/j.1467-9469.2007.00585.x.

About the author

Johannes F. Schmieder is a professor of economics at Boston University. His research is in labor economics, including unemployment insurance, wage determination, and the analysis of linked employer–employee data.

A Estimation formulas

A.1 Linear model

On the estimation sample, write the model as $y = X\beta + D\alpha + \varepsilon$, where D represents all absorbed terms and $W = \text{diag}(w_i)$. The weighted residual-maker is

$$M_D = I - D(D'WD)^{-}D'W,$$

where the generalized inverse accounts for redundant effects; the projection itself is unique. With $\widetilde{y} = M_D y$ and $\widetilde{X} = M_D X$, the coefficients on the linearly independent columns of $\widetilde{X}$ are

$$\widehat{\beta} = (\widetilde{X}'W\widetilde{X})^{-1}\widetilde{X}'W\widetilde{y}.$$

The residual $u = y - X\widehat{\beta} - D\widehat{\alpha}$ removes both the covariate contribution and the fitted absorbed sum. An absorbed slope adds columns to D that contain the slope variable rather than indicators.

Let N be the number of observations (the sum of frequency weights), and let K and A count the identified covariates and the absorbed degrees of freedom defined below, with the intercept counted only once. For unweighted OLS, let $B = (\widetilde{X}'\widetilde{X})^{-1}$. The IID covariance matrix is $\widehat{\sigma}^2 B$ with $\widehat{\sigma}^2 = u'u/(N - K - A)$. The robust covariance matrix is

$$\widehat{V}_{\text{HC1}} = \frac{N}{N - K - A} B \left(\sum_i \widetilde{x}_i \widetilde{x}_i' u_i^2 \right) B.$$

For one-way clustering, the sum of observation scores is replaced by $\sum_g s_g s_g'$ with $s_g = \sum_{i \in g} \widetilde{x}_i u_i$, and the correction becomes

$$\frac{N-1}{N-K-A} \cdot \frac{G}{G-1},$$

where G is the number of clusters. Absorbed groups nested within the clusters are not counted in A, and in that case the denominator $N - K - A$ is reduced by one. With several cluster dimensions, the middle term combines the score sums for every nonempty intersection of the dimensions by inclusion and exclusion (Cameron et al., 2011), G is the smallest number of clusters across dimensions, and negative eigenvalues of the resulting matrix are set to zero. With clustering, the reported residual degrees of freedom are $\min(N - K - A, G - 1)$. Analytic weights enter B and the scores in the usual way. Frequency weights multiply each observation's contribution to the robust middle term by its weight, not by its square, because they represent repeated observations.

A.2 Absorbed degrees of freedom

The absorbed degrees of freedom A equal the number of absorbed groups minus the number that are redundant. With one effect, no group is redundant. With two categorical effects, one group of the second effect is redundant in each connected set of the bipartite graph formed by the two effects, so $A = J_1 + J_2 - \kappa$ exactly, where J_1 and J_2 count the groups and κ counts the connected sets.

With more effects, the default `dof(pairwise)` applies the same count of connected sets to each pair of effects, as the default of `reghdfe` does. The count is then an approximation rather than exact. `dof(firstpair)` adjusts only the first pair, `dof(continuous)` adjusts only for redundant absorbed slopes, and `dof(none)` counts every absorbed coefficient. The matrix `e(dof_table)` reports the number of groups, the redundancies, and whether the count is exact for each absorbed term.

A.3 Poisson pseudo-maximum likelihood

Let q_i be the prior weight and o_i the offset. At iteration k, with linear predictor $\eta_i^{(k)}$ and mean $\mu_i^{(k)} = \exp(\eta_i^{(k)})$, the working response and weight are

$$z_i^{(k)} = \eta_i^{(k)} + \frac{y_i - \mu_i^{(k)}}{\mu_i^{(k)}}, \qquad h_i^{(k)} = q_i \mu_i^{(k)}.$$

A weighted regression of $z_i^{(k)} - o_i$ on the covariates and absorbed effects, with weights $h_i^{(k)}$, gives the next linear predictor. The implementation controls the step length and declares convergence after two consecutive iterations that satisfy both a deviance criterion and a score criterion at the requested tolerance. Because the weights change at every iteration, each iteration requires a new weighted absorption (Stammann, 2018; Correia et al., 2020).

Inference uses the final fit. After absorbing the effects from the covariates with the final weights $q_i\widehat{\mu}_i$, the bread is $\widehat{B} = (\sum_i q_i\widehat{\mu}_i\widetilde{x}_i\widetilde{x}_i')^{-1}$ and the observation score is $q_i(y_i - \widehat{\mu}_i)\widetilde{x}_i$. The robust covariance matrix is the sandwich formed with these terms and the correction $N/(N-1)$. Clustered covariance matrices sum the scores within clusters, combine several dimensions by inclusion and exclusion, and apply the correction $G/(G-1)$ with G the smallest number of clusters. Inference is based on the normal distribution. The residual degrees of freedom reported in `e(df_resid)` are N minus the rank of the covariates and of the absorbed effects, capped at $G-1$ with clustering.

A.4 Saved fixed effects

The fit identifies the total absorbed contribution, but individual effects require a normalization. Within each connected set, the command centers each categorical intercept effect after the first, in the order declared in `absorb()`, and adds the removed mean to the first effect, so that the total contribution is unchanged. The centering uses the regression weights for OLS and the final weights $q_i\widehat{\mu}_i$ for PPML. A saved slope term contains the slope coefficient multiplied by its variable, so that saved terms add up to the total absorbed contribution. These conventions should be taken into account when comparing saved effects across commands.