`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 and default stopping settings, `fereg` was 6–204 times faster than `reghdfe` and `feppml` 11–103 times faster than `ppmlhdfe`. The commands were also faster than `fixest` (R) and `PyFixest` (Python) in all 53 simulated configurations and than the Julia packages `FixedEffectModels.jl` and `GLFixedEffectModels.jl` in 52 of 53, with geometric-mean time ratios between 3.0 and 7.7. On administrative data with 1.2 million workers and 117,000 firms, `fereg` and `feppml` were at least 3.4 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. With two to five effects, it eliminates the effect with the most groups exactly and solves the much smaller remaining system by preconditioned conjugate gradients, switching to a combinatorial multigrid preconditioner that exploits the graph formed by workers and firms when the data are poorly connected; with more effects, it preconditions the full system with local solves for pairs of effects. Alternating projections and LSMR remain available on request and as fallbacks. 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 fixed effect 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). The pairwise Schwarz preconditioner follows Fischer and Schröder (2026a), who describe and evaluate such preconditioners, and its local approximate factorizations build on Gao et al. (2026). The modified LSMR engine was adapted from the `within` library (Fischer and Schröder, 2026b), which is also used by `PyFixest`. These are building blocks of `fereg`, rather than new algorithms introduced in this article. For

PPML, iteratively reweighted least squares, reuse of earlier solutions, and adaptive absorption tolerances are also established features of `ppmlhdfe` (Correia et al., 2020). `fereg`'s choices concern how to solve and check these weighted problems: exact elimination of one effect, a joint update of the remaining effects and covariates, and reuse of a multigrid preconditioner for two-effect designs.

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 0.56 seconds, compared with 15.0 seconds for `reghdfe`, and `feppml` fit a PPML model in 2.52 seconds, compared with 83.8 seconds for `ppmlhdfe`. The advantage depends on the structure of the fixed effects, but `fereg` and `feppml` were faster than `reghdfe` and `ppmlhdfe` in every design, option, and number of cores tested, at default and at tighter stopping settings. On administrative data with five million observations and more than a million workers, `fereg` estimated a model with worker and firm effects in 5.5 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, 2026). 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. The Stata command `reghdfejl` provides another route to Julia's linear estimator, and `GLFixedEffectModels.jl` provides generalized linear models with high-dimensional effects (Roodman, 2026; GLFixedEffectModels.jl contributors, 2026); section 7 compares all of these implementations. Other approaches include `xhdfe`, with a shared C++ core for Stata, Python, and R; `creghdfe` in `ctools`, a C implementation for Stata; `hdfe-stream`, which uses Polars to process data in batches; and `dbreg`, which computes regressions through database backends (xhdfe contributors, 2026; ctools contributors, 2026; Tucker, 2026; dbreg contributors, 2026).

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., 2026). 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*[`, `*suboptions*]`) 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`; its `solver()` accepts fewer methods.

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., 2026). `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* `[,` *suboptions*`])`

selects the method used to absorb the fixed effects. The default, `auto`, chooses among the methods described in section 5. For `fereg`, *method* may also be `map` (alternating projections), `lsmr`, or `cg` (conjugate gradients); for `feppml`, it may be `lsmr` or `cg`. The *suboptions* name the parts of the method: `reduce(auto|full|eliminate)` keeps every effect or eliminates the effect with the most groups; `precondition(auto|none|jacobi|block|schwarz|cmg)` selects the preconditioner; and `accelerate(none|irons-tuck|squarem)` accelerates `map`. Examples are `solver(map, accelerate(irons-tuck))`, `solver(lsmr, precondition(schwarz))`, and `solver(cg, reduce(eliminate))`. A combination that the grammar does not allow returns an error. A valid request that its method cannot serve continues on another certified method, and the stored results report this fallback (section 5.8).

`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 was requested and what ran, in the words of `solver()`. `e(fe_requested)` records the request, and `e(fe_requested_method)`, `e(fe_requested_reduction)`,

`e(fe_requested_preconditioner)`, and `e(fe_requested_acceleration)` its parts. `e(fe_route)` names the route that absorbed the fixed effects, for example `cg, reduce(eliminate) precondition(jacobi)`, and `e(fe_reduction)`, `e(fe_method)`, and `e(fe_preconditioner)` its parts; `e(fe_iterations)` and `e(fe_escalations)` count iterations and switches to a stronger preconditioner. `e(fe_fallback)` is 1 when a certified fallback served the request, and `e(fe_fallback_reason)` names why: `unsupported`, `memory`, `nonconvergence`, or `uncertified`. After `fereg`, results beginning with `e(fe_residual_)` report the same for the final residuals. After `feppml`, these results describe the route that solved the iterations, and `e(fe_final_route)` the route of the final weighted fit; with `solver(lsmr)`, `e(fe_formulation)` reports whether the iterations were solved jointly for the covariates and fixed effects (`joint`), by separate projections (`fwl`), or both (`joint+fwl`), with counts in `e(fe_joint_frames)`, `e(fe_fwl_frames)`, and `e(fe_handoffs)`. The matrix `e(fe_solve)` holds iteration counts, timings, and the values of the checks. 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 release, version 0.6.0, from its release tag:

```
net install fereg, ///
    from("https://raw.githubusercontent.com/johannes-schmieder/fereg/v0.6.0/") replace
fereg, selftest
```

Replacing `v0.6.0` by `main` installs the development version. Alternatively, users of the community-contributed `github` installer can install from `main`:

```
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. The release tag fixes the package, including its native plugins; because `main` changes during development, use the tag for replication and retain the installed version 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 7.7, 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 on November 11, 2026 with version 0.6.0, installed from its release tag; 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, reduce(eliminate) precondition(jacobi)
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 first line above the table names the route that absorbed the fixed effects in the syntax of `solver()`: here conjugate gradients on the reduced system of section 5.4, after eliminating the worker effects, with the diagonal preconditioner. The second 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, simulated dataset contains 20 countries observed for four years. Excluding domestic pairs gives 1,520 directed country-pair–year observations. For this example, 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 6 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. With two to five effects, removing the effect with the most groups exactly leaves a much smaller system, and when the data are poorly connected, preconditioners that exploit the graph of workers and firms, or the links between pairs of effects, speed up its solution. The default,

`solver(auto)`, therefore inspects the absorbed terms and chooses a method for each type of design, and this section explains those methods. They are implemented in the plugin's fixed-effect library (section 6). 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.

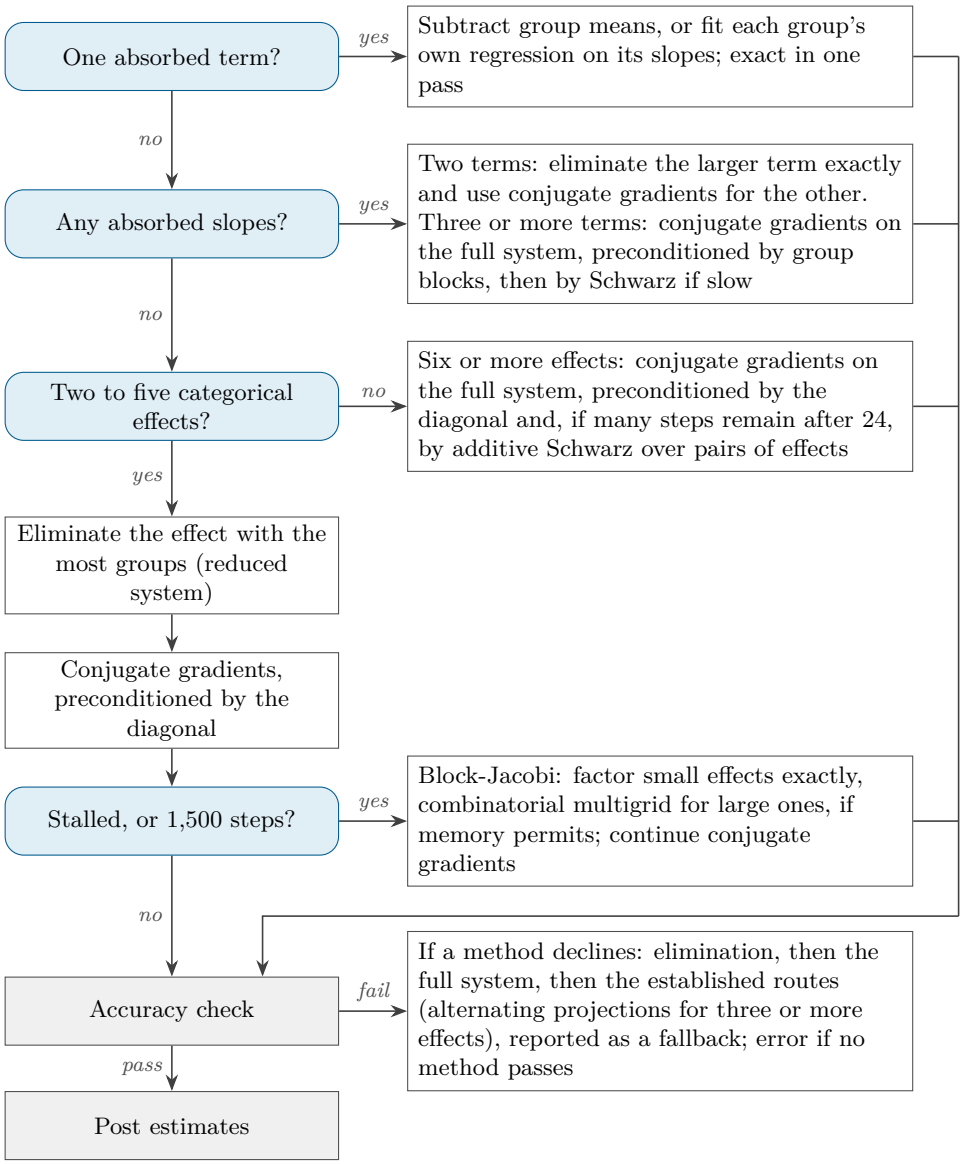


Figure 1: Decisions made by `solver(auto)` in `fereg`. Each path ends with the accuracy check (6); the command posts results only if the check passes.

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 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, usually $\tau/10$, so that its result usually passes the check; a result that does not is continued at a tighter target or handed to another method (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

[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.

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 administrative panel of section 7.7, no coefficient fitted by version 0.5.0 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.6). In the benchmarks of section 7, default fits differed from fits at 10^{-12} by at most 4.1×10^{-10} standard errors.

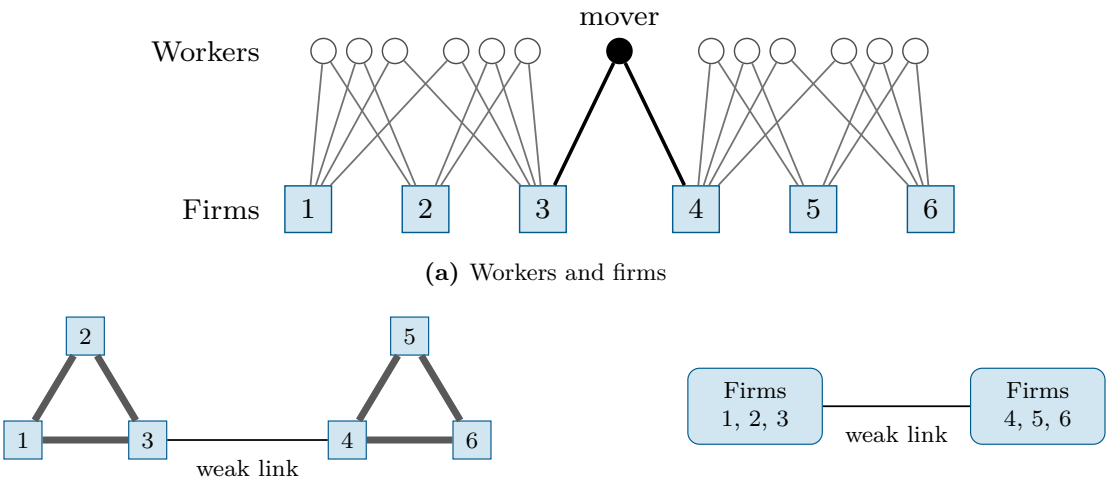
5.2 One fixed effect: subtracting group means

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 on request and as a fallback when the methods below decline. 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 to five fixed effects: eliminating the largest effect

The most common design in applied work has two sets of categorical fixed effects, such as worker and firm effects, and many add a third or fourth, such as year or industry effects. For two to five categorical effects, `fereg` combines three ideas. It removes the effect with the most groups exactly, solves the much smaller system that remains by conjugate gradients, and, when the data are poorly connected, speeds up the iterations with a stronger preconditioner, combinatorial multigrid for the graph of firms.

Eliminating the worker effects. Split $D = [D_1\ D_2]$ into the columns for workers (D_1), the effect with the most groups, and for the remaining effects (D_2), here the firms, and split $\hat{\alpha}_v$ into worker effects a_1 and remaining effects a_2. Three blocks of $D'WD$ enter the normal equations: the diagonal matrix $N_1 = D_1'WD_1$, which holds the sizes of the workers, the matrix $N_2 = D_2'WD_2$, which with firm effects alone is the diagonal matrix of firm sizes, 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. With three to five effects, a_2 stacks the coefficients of all remaining effects, and S has one block of rows for each of them.

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 solves (7) 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 how it escalates the 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. The residuals $\tilde{v}$ are unaffected by it. With more remaining effects, each diagonal block of S is such a Laplacian for one remaining effect.

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 worker by worker from the cells of the worker-by-firm table, 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). `fereg` advances up to eight variables together, so that one pass over the data serves all of them.

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, the diagonal of S, which rescales each firm's equation. This preconditioner costs almost nothing and suffices when the firms are well connected. The discrepancy $b - Sa_2$ is exactly the vector of weighted residual sums of the remaining groups, so conjugate gradients tracks the quantity that the check (6) bounds. It stops when this

discrepancy falls below $\tau/10$, relative to the length of the variable after removing its worker means. `fereg` then computes the worker effects, forms the residuals $\tilde{v}$ in one more pass, and applies the check at τ to the group means recomputed from them; the worker means are zero by construction.

Combinatorial multigrid for poorly connected data. If conjugate gradients stalls, halving its discrepancy less than once every 40 steps, or has not finished after 1,500 steps, `fereg` switches to a stronger preconditioner and continues from the current solution. The new preconditioner, block-Jacobi, treats each remaining effect's block of S as a whole. A block with at most a few hundred groups, or about a thousand when that is cheap, is factored exactly. A larger one is approximated by one cycle of 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` builds it only when the simple preconditioner stalls. Designs that converge quickly avoid this cost, and later solves for the same regression, such as the final residuals, start with the stronger preconditioner. The multigrid graph represents the links between firms exactly: a worker observed at more than five firms becomes an auxiliary vertex linked to each of them, whose elimination restores the worker's links between all pairs of these firms. The preconditioner affects only how quickly the iterations converge, not the system (7) that they solve or the check.

The elimination declines if the iterations stall even with the stronger preconditioner, if its workspace does not fit in memory, or if the reconstructed residuals fail the check even after a further pass. The variables then go to the method of section 5.5 and, if that also declines, to the established routes of earlier versions: for two effects, conjugate gradients on (7), which a structural rule may precondition with multigrid, and for more effects a short pilot of alternating projections followed by alternating projections. Such a result is reported as a certified fallback in the stored results. If no method passes the check within the iteration limit, the command stops with an error. Online appendix B.3 gives the exact rules.

5.5 Six or more fixed effects: conjugate gradients on the full system

With six or more categorical effects, `fereg` works with the normal equations (5) of all effects together, which it also uses for designs that the elimination declines. It solves them by conjugate gradients,

again for up to eight variables at once, starting from the group means of the largest effect. The matrix $D'WD$ is singular, because of the redundancies noted in section 5.1, but the equations are consistent, and conjugate gradients converges to a solution with the unique residual $\tilde{v}$.

As before, the preconditioner is what makes the method fast. `fereg` starts with the diagonal of $D'WD$, which rescales each group by its size. This simple choice ignores how the effects are linked, which is where slow convergence originates. After 24 steps, `fereg` projects from the recent rate of progress how many steps each variable still needs. If a projection exceeds 100 steps, or if the iterations stall, it switches to an additive Schwarz preconditioner (Toselli and Widlund, 2005) built from pairs of effects, if its workspace fits in memory, and continues from the current solution. Fischer and Schröder (2026a) describe this kind of graph preconditioner and compare it with alternating projections and other preconditioners. `fereg` pairs the largest effect with every other effect, so that every group belongs to at least one pair. For each pair, and for each connected set of groups within that pair, it prepares a solver for the two-effect problem of section 5.4: it eliminates the effect with more groups exactly and factors the remaining graph Laplacian, exactly when that is cheap and otherwise by approximate Cholesky factorization (Gao et al., 2026). Each application of the preconditioner solves all pair problems independently, which can be done in parallel, and adds up their corrections. The pair problems capture how each pair of effects is linked.

Building the Schwarz 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. The projection after 24 steps therefore builds it only when the simple preconditioner is predicted to be slow, and later solves for the same regression start with it. Conjugate gradients stops at a tenth of the tolerance and is accepted only if the reconstructed residuals pass the check for every group. It may use half of the limit set by `iterate()`. If it declines, the remaining budget goes to the established routes, with alternating projections for three or more effects, and the result reports a certified fallback (online appendix B.4).

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 the corresponding block of $D'WD$ for each group is a small dense matrix. `fereg` applies the methods above group by group. With exactly two absorbed terms, it eliminates the larger term exactly, group by group, and solves for the coefficients of the other term by conjugate gradients, preconditioned by the exact inverse of each of its groups' blocks. With three or more terms, it uses conjugate gradients on the full system with the same group blocks as preconditioner, switching to additive Schwarz over pairs of terms when convergence is slow. If these methods decline, alternating projections, in which each step is a set of exact group regressions, finish the fit, and the result reports a certified fallback. `solver(auto)` does not use multigrid 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. Reusing earlier solutions and adapting the absorption tolerance to the progress of IRLS are established methods used by `ppmlhdfe` (Correia et al., 2020). The following describes `feppml`'s particular solver, checks, and preconditioner choices.

First, a fresh fit with two to five categorical effects and covariates, in the fast memory plan, starts with plain iterations: each projects the working response and the covariates onto the fixed effects with the exact elimination of section 5.4, reusing the previous iteration's residuals, and solves the small normal equations of the covariates. These iterations stop once the deviance changes by less than a hundredth of the tolerance. If the last of them also passes the score check, it counts as the first of the two converged iterations that `feppml` requires (appendix A.3).

Second, `feppml` then 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, starting from the previous iteration's solution. 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.

Third, `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.

Fourth, 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 with the methods of sections 5.4 and 5.5, 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, the joint solves also use 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. The first word of `solver()` names the method, and suboptions name its parts: the reduction (`reduce()`), the preconditioner (`precondition()`), and, for alternating projections, the acceleration (`accelerate()`).

`solver(map)` runs alternating projections with symmetric sweeps, as described in section 5.3; `accelerate(irons-tuck)` and `accelerate(squarem)` extrapolate these sweeps, using the methods of Irons and Tuck (1969) and Varadhan and Roland (2008): after a few sweeps, they estimate where the sequence of residuals is heading and jump ahead. `solver(cg)` runs the conjugate gradients of sections 5.4 and 5.5, with `reduce(eliminate)` for the elimination and `reduce(full)` for the full system, and with a fixed preconditioner if one is named (`jacobi`, `block`, `schwarz`, or `none`); it also accepts absorbed slopes. `solver(lsmr)` applies LSMR (Fong and Saunders, 2011) to the full system, preconditioned by Jacobi, additive Schwarz, or nothing. LSMR is designed for least-squares problems such as the regression of v on D and works with D directly, through products such as Dz and $D'Wv$, rather than with the normal equations. `solver(cg, precondition(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(lsmr)`, which solves each iteration's regression for the covariates and fixed effects together by LSMR, and `solver(cg)`, which uses the multigrid graph, with the requirements stated in online appendix B.8.

An explicitly requested method is held to the same check. If it cannot serve the request, because the design is outside its support, its workspace does not fit in memory, it does not converge within its share of the iteration limit, or its result fails the check, the calculation continues with the next certified method within the remaining iterations. The stored results report the request, the method that ran, and the reason for the fallback (section 3).

The modified LSMR engine of these solvers comes from the `schwarz-precond` crate of the `within` library (Fischer and Schröder, 2026b), developed by Alexander Fischer, Kristof Schröder, and contributors at the appliedAI Institute for Europe and in the `PyFixest` community; `PyFixest` has offered `within` as an LSMR backend since version 0.50 (PyFixest developers, 2026). Its MIT-licensed source was adapted and moved into the plugin (section 6).

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 methods of section 5 are implemented in fedemean, the plugin's fixed-effect library. It separates three choices, which are also the words of `solver()` and of the stored results: the reduction of the problem (keep every effect or eliminate the largest), the iterative method (conjugate gradients or modified LSMR), and the preconditioner (the diagonal, block-Jacobi, additive Schwarz, or combinatorial multigrid). A projection is prepared once for a design and reused for every variable and, in PPML, across the changing weights of the iterations; a solve can start from an earlier solution, which PPML uses from one iteration to the next. Every result carries the moment check (6), recomputed from the formed residuals, and a method that cannot pass it declines and returns the variables for the next method.

fedemean contains a modified LSMR engine adapted from the MIT-licensed `schwarz-precond` crate of the `within` library (Fischer and Schröder, 2026b). Its source, adapted so that iterations can be resumed and cancelled, stay within a memory budget, and combine partial results in a reproducible order, was moved into fedemean with its tests and license notice; `fereg` no longer depends on `within`. The approximate Cholesky factorizations of the Schwarz preconditioner come from the `approx-chol` library (Gao et al., 2026), and combinatorial multigrid from a Rust port (Schmieder, 2026), version 0.3.0, of Koutis and Miller's MATLAB/C implementation (Koutis et al., 2011; Koutis and Miller, 2026). 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. Sums over observations and groups are combined from fixed chunks in a fixed order, and the random choices of the solvers, such as those of approximate Cholesky factorization, use fixed seeds, so a fit repeats exactly with the same number of threads; results agree to numerical accuracy, but not necessarily in the last digits, across thread counts and platforms. The LSMR iterations and the rank-revealing QR factorization of the covariates compute lengths with a function built from basic floating-point operations instead of the platform's mathematical library, so that their iteration counts, stopping decisions, and pivots do not depend on the platform. The QR factorization treats columns whose norms differ only by rounding as tied, so that of covariates that are exact multiples of one another, the one listed first is kept, as with `regress`. `deterministic` fixes native work partitions and reduction order for a given platform and thread count also where the default does not. 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

The default `memory(auto)` chooses a working plan based on the estimated memory available. `memory(low)` processes covariates in smaller blocks; `poolsize()` controls their size. Smaller blocks reduce memory use but can require more passes over the data and temporary scratch files, which are removed when the command finishes. The dataset still has to fit in Stata's memory, and additional threads need workspace. `timeit` reports the chosen plan, memory estimates, and scratch use. These settings help manage memory; they do not impose a hard limit on the process's total memory use.

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`, `areg`, and `reghdfejl`, with the R package `fixest`, the Python package `PyFixest`, and the Julia packages `FixedEffectModels.jl` for linear models and `GLFixedEffectModels.jl` for PPML. Three findings stand out. At default stopping settings, `fereg` and `feppml` were faster than the Stata commands in every configuration, by factors that in linear models grow with the size of the data. They were also faster than the packages in other languages in all simulated configurations but one, by geometric-mean factors between three and eight. And the gains over most comparators were largest in sparse or clustered worker–firm designs, in large datasets, and in a large administrative panel.

7.1 Design and measurement

The measurements ran on November 10, 2026 on whole 28-core compute nodes of Boston University's Shared Computing Cluster, using the pre-release build `80b6768e` of `fereg`. Its numerical runtime is identical to that of the released version 0.6.0, which differs from it only in version metadata.[2] The comparators were the same pinned `reghdfe` 6.14.1 (Correia, 2026) and `ppmlhdfe` 2.3.4 as in earlier measurements, the `areg` command of Stata/MP 19 (licensed for four cores), `reghdfejl` at commit `e22427e7`, which passes the data to FixedEffectModels.jl 1.13.3 in Julia 1.12.7 (Roodman, 2026), `fixest` 0.14.2, a development version of `PyFixest` using the `within` library, and

[2]The release is at https://github.com/johannes-schmieder/fereg/releases/tag/v0.6.0. The benchmarked plugin was built from commit `80b6768e7322c7072a8fc60b6cf5687d53cdfa6d`; the replication archive records its hash.

`FixedEffectModels.jl` 2.1.0 (Gomez, 2026) and `GLFixedEffectModels.jl` 0.5.6 (GLFixedEffectModels.jl contributors, 2026), both in Julia 1.11.7.[3] `reghdfejl` was run for linear models only. For Poisson models it calls `GLFixedEffectModels.jl`, but its tolerance option does not reach that package, so that a tighter profile would repeat the default; the PPML comparisons therefore run `GLFixedEffectModels.jl` directly.

The measurements are organized in comparison blocks. A block is one configuration, one comparator, and one predetermined repetition, run on a whole node reserved for it: `fereg` and the comparator each ran at their default and at tighter stopping settings, in fresh processes and on the same assigned cores. Every package received the same number of cores. Each configuration has five blocks per comparator, or three above one million observations and on the administrative panel; the 1,297 blocks contain 5,188 timed fits and ran on 47 nodes. A reported ratio is the median, over the blocks, of the comparator's time divided by `fereg`'s time in the same block, so that differences between nodes do not enter it. Absolute times are medians over blocks and therefore over nodes; for `fereg` they pool its fits in all blocks of a configuration.

The article reports the default stopping settings. Stopping criteria differ across packages, so equal settings need not deliver equal accuracy; the online appendix reports the tighter settings and the achieved numerical differences. Each package also used its default standard errors. Their small-sample conventions differ, and reported standard errors differ by up to 14% across packages for that reason alone (section 7.4). Online appendix C.1 lists the software versions, thread settings, and tolerances.

The reported time is the 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, and `reghdfejl`'s also passes them to Julia, whereas the other languages start from data already held in their own memory (section 6.1); for `fereg` reading took 0.20 seconds, or 36% of its time, in the linear worker–firm design with one million observations. Every command had the same limit of one hour, and every scheduled fit was attempted. Of the timed fits, 10 reached the limit, all at tighter settings: three of `reghdfe` and seven of `GLFixedEffectModels.jl`. They are reported as censored results; no fit was repeated or replaced. 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.

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

[3] The PyFixest checkout reports version 0.60.0 but is pinned to commit `0f41fffd7276f89cbd87528ef13e8c0261622932`, with `within-py` 0.4.0. Its default LSMR demeaner uses the adaptive preconditioning policy: it starts with diagonal preconditioning and can switch to Schwarz preconditioning if progress stalls; PPML can reuse the preconditioner across iterations. The replication archive identifies the full Python dependency freeze and the Julia manifests by their hashes.

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. Table 5 reports the worker–firm design with one million observations and four cores. In the linear model, `fereg` took 0.56 seconds and `reghdfe` 15.0 seconds; within the same blocks, `reghdfe` took a median 26.6 times as long as `fereg`, `areg` 12.1 times, and `reghdfejl` 11.4 times. For PPML, `feppml` took 2.52 seconds and `ppmlhdfe` 83.8 seconds, 33.1 times as long. Robust or clustered standard errors and saved fixed effects give ratios of the same order. Across all simulated configurations (table 6), `reghdfe` took between 6 and 204 times as long as `fereg` (geometric mean 31), `ppmlhdfe` between 11 and 103 times (30), `areg` between 2.2 and 17 times, and `reghdfejl` between 3.9 and 54 times. The smallest ratios arise with a single fixed effect. The largest arise when the worker–firm graph is sparse or clustered and when the data are large (section 7.6). In the clustered-mobility design, for example, `reghdfe` took 74.0 and `ppmlhdfe` 69.7 times as long as `fereg` and `feppml`.

In the linear models, the shorter running time comes with lower memory use. With one million observations, `fereg` used 0.55 GiB at its peak in the worker–firm design, `reghdfe` 0.76 GiB, `areg` 0.82 GiB, and `reghdfejl`, which also holds the data in Julia, 2.82 GiB. In every linear configuration with one million observations, `fereg`'s peak was below that of every other command, except that `areg` needed less with a single fixed effect (0.22 against 0.52 GiB). For PPML, `feppml` and `ppmlhdfe` both used about 1.27 GiB. These figures include the data that Stata holds in memory.

7.3 Fixed effects, options, and other languages

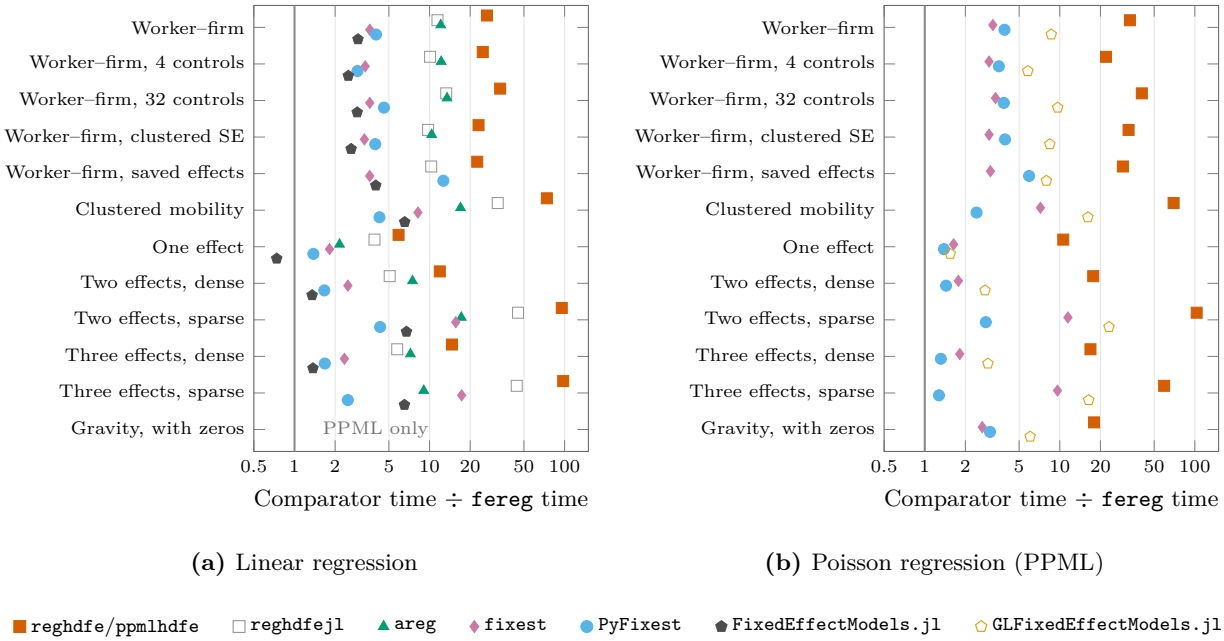
Figure 3 shows, for each design and option, how long each comparator took relative to `fereg`; values above one mean that `fereg` was faster. Table 6 summarizes the ratios over all simulated configurations. All default-setting timings are shown, whatever their numerical agreement, which is reported separately (section 7.4).

`fixest` and `PyFixest` were slower than `fereg` in all 53 simulated configurations, with geometric-mean ratios of 5.0 and 3.8 in linear models and 3.4 and 3.0 in PPML. `FixedEffectModels.jl` was faster than `fereg` in one of its 29 configurations, the model with a single fixed effect, where it took 0.74 times as long; its geometric-mean ratio was 3.6. `GLFixedEffectModels.jl` was slower in all

Table 5: Time and memory against Stata commands

Inference and output	Median seconds (times as long as `fereg`)				Peak memory (GiB)			
	`fereg`	`reghdfe`/ `ppmlhdfe`	`areg`	`reghdfejl`	`fereg`	`reghdfe`/ `ppmlhdfe`	`areg`	`reghdfejl`
Linear regression								
IID	0.56	15.0 (26.6×)	6.64 (12.1×)	6.51 (11.4×)	0.55	0.76	0.82	2.82
Robust	0.67	16.5 (14.4×)	6.82 (10.3×)	6.41 (9.5×)	0.56	0.76	0.82	2.95
Clustered by worker	0.66	15.3 (23.1×)	6.74 (10.4×)	6.49 (9.8×)	0.57	0.76	0.83	2.89
IID, saved effects	0.74	17.1 (22.5×)	—	7.91 (10.3×)	0.61	0.76	—	3.09
Poisson regression (PPML)								
Robust	2.52	83.8 (33.1×)	—	—	1.27	1.27	—	—
Clustered by worker	2.61	83.4 (32.3×)	—	—	1.29	1.27	—	—
Robust, saved effects	2.90	86.1 (29.3×)	—	—	1.27	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, default stopping settings and each package's default standard errors. Each comparator ran in five comparison blocks; a block is a whole node on which `fereg` and the comparator ran in fresh processes. Comparator times are medians over these blocks, `fereg`'s the median of its fits in all blocks of the configuration. Parentheses give the median, over the blocks, of the comparator's time divided by `fereg`'s time in the same block. Every command starts from the data in Stata's memory, and every time includes reading them; `reghdfejl`'s also includes passing them to Julia. `areg` cannot save individual fixed effects, and `areg` and `reghdfejl` fit linear models only (—). Peak memory is the median, over the blocks, of the largest resident set size of the estimating process during the command, including the data held in memory and, for `reghdfejl`, the embedded Julia session.



(a) Linear regression

(b) Poisson regression (PPML)

Figure 3: Running time relative to `fereg` by design and option, at default stopping settings. Each point is the median, over the comparison blocks, of the comparator's time divided by `fereg`'s time in the same block. 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. `areg` cannot save individual effects. `areg`, `reghdfejl`, and `FixedEffectModels.jl` were run for linear models and `GLFixedEffectModels.jl` for PPML.

Table 6: Default-setting time ratios relative to `fereg`

	Linear regression			PPML		
Package	Completed	Range	Geo. mean	Completed	Range	Geo. mean
`reghdfe`	24 of 24	5.9–204	31.0	—	—	—
`ppmlhdfe`	—	—	—	19 of 19	10.6–103	30.0
`areg`	21 of 21	2.2–17.2	9.39	—	—	—
`reghdfejl`	24 of 24	3.9–54.0	13.4	—	—	—
`fixest`	29 of 29	1.8–27.8	4.98	24 of 24	1.6–11.9	3.45
`PyFixest`	29 of 29	1.4–15.5	3.82	24 of 24	1.3–6.5	3.03
`FixedEffectModels.jl`	29 of 29	0.74–10.3	3.56	—	—	—
`GLFixedEffectModels.jl`	—	—	—	24 of 24	1.5–34.2	7.68

Notes: All simulated configurations at default stopping settings, with package-default standard errors. Completed counts configurations with at least one completed time among those scheduled for the package; a configuration's ratio is the median, over its comparison blocks, of the comparator's time divided by `fereg`'s time in the same block. Range and geometric mean are taken over the completed configurations. Values below one favor the comparator. Stopping criteria differ across packages, so equal settings need not deliver equal accuracy. Numerical deviations and tighter settings are reported in the online appendix.

24 PPML configurations, with a geometric mean of 7.7. For every comparator except `PyFixest` in PPML, the ratios were smaller in the designs with one effect or with dense effects, which the other packages also solve quickly, than in the sparse and clustered 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 retained 960,000; `ppmlhdfe` took 17.9 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 coefficient for it, so that they retained 970,000 observations. `GLFixedEffectModels.jl` does not detect separation and kept all 1,000,000. On these different samples, the coefficients of the other controls agree, and `fixest`, `PyFixest`, and `GLFixedEffectModels.jl` took 2.7, 3.0, and 6.0 times as long as `feppml`.

The requested output matters more for some commands than for others. Saving the worker and firm effects raised `fereg`'s median time by 32% in the linear model and 15% in PPML, because the effects are recovered from the solution already computed. `PyFixest` recovers them in a separate step; its median time grew by a factor of 4.1 in the linear model and 1.7 in PPML. More controls make every command slower. In the linear model, `fereg`'s time grew from 0.26 seconds with 4 controls to 0.91 with 32. `reghdfe` took 24.8 times as long as `fereg` with 4 controls and 33.3 times with 32, and `fixest` 3.3 and 3.6 times. With 32 controls, 0.29 of `fereg`'s 0.91 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 22.0 to 40.5 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 Numerical agreement

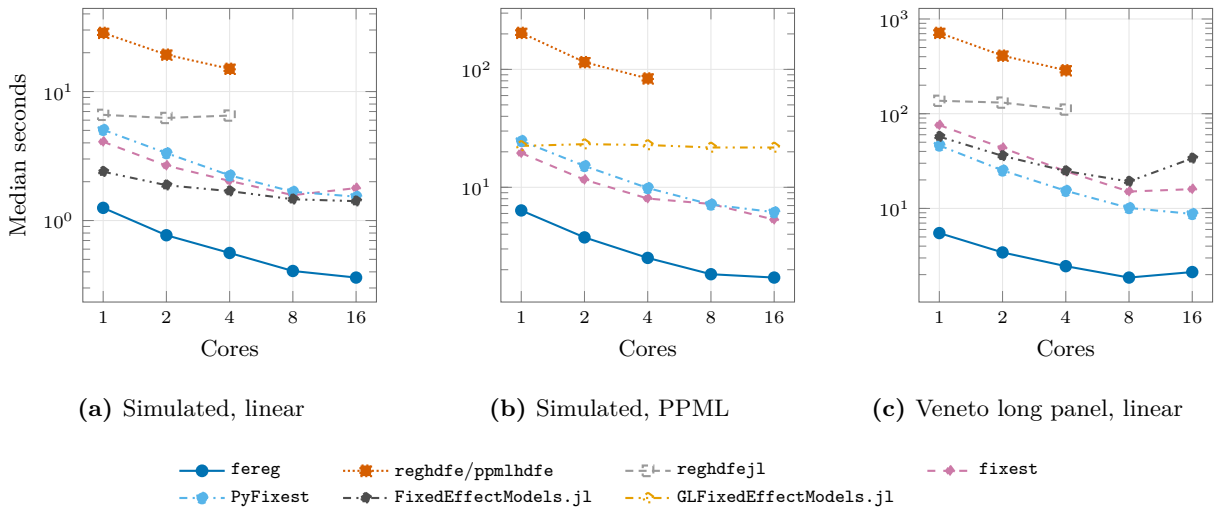
Timings and numerical agreement answer different questions, and the online appendix reports them separately (appendix C.2). Every fit is compared with the tighter `fereg` fit (10^{-12}) of the same block. Such a comparison measures agreement with that reference, not an independent guarantee of accuracy. Default `fereg` differed from it by at most 4.1×10^{-10} standard errors in any coefficient. The largest coefficient differences of the comparators were 4.7×10^{-4} standard errors for `areg` at either setting, 2.3×10^{-4} for `FixedEffectModels.jl` and 6.1×10^{-5} for `fixest` at default settings, and 9.5×10^{-7} for `GLFixedEffectModels.jl`; all others were at most 9.0×10^{-8}. Standard errors differ more, for reasons of convention rather than convergence. The robust PPML standard errors of `fixest` and `PyFixest` were up to 12.5% larger than `feppml`'s in the simulations and 14.4% on the administrative panel, the factor $N/(N-p)$ by which their covariance matrices differ when p counts the absorbed fixed effects along with the coefficients. The clustered standard errors of `areg` were up to 11.9% larger. Tighter settings leave these differences unchanged. Retained samples differ only in the gravity design. In 115 comparator fits with saved effects, mostly at default settings, the saved predictions did not pass an independent check of the model scores, and in 15 tighter `GLFixedEffectModels.jl` fits its reported log likelihood of $-\infty$ could not be reproduced from the predictions; these fits keep their timings. Every one of `fereg`'s 2,594 timed fits completed and passed the campaign's acceptance checks.

7.5 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 1.25 seconds with one core to 0.36 with sixteen, and for PPML from 6.37 to 1.72 seconds: speedups of 3.5 and 3.7, 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.[4] 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 22.6 times as long as `fereg` and `ppmlhdfe` 32.1 times. `fixest` and `PyFixest` took between 2.6 and 5.0 times as long as `fereg` at every number of cores. `FixedEffectModels.jl` gained less from additional cores: it took 1.3 times as long as `fereg` with one core and 3.7 times with sixteen, although its own time fell from 2.39 to 1.41 seconds, by a factor of 1.7. The time of `GLFixedEffectModels.jl` hardly changed with the number of cores (22.4 seconds with one core and 21.7 with sixteen), so that its ratio grew from 3.1 to 12.7; neither did that of `reghdfejl` (6.60 seconds with one core and 6.51 with four). These flat profiles reflect the pinned package

[4] A separate test with version 0.5.0 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.



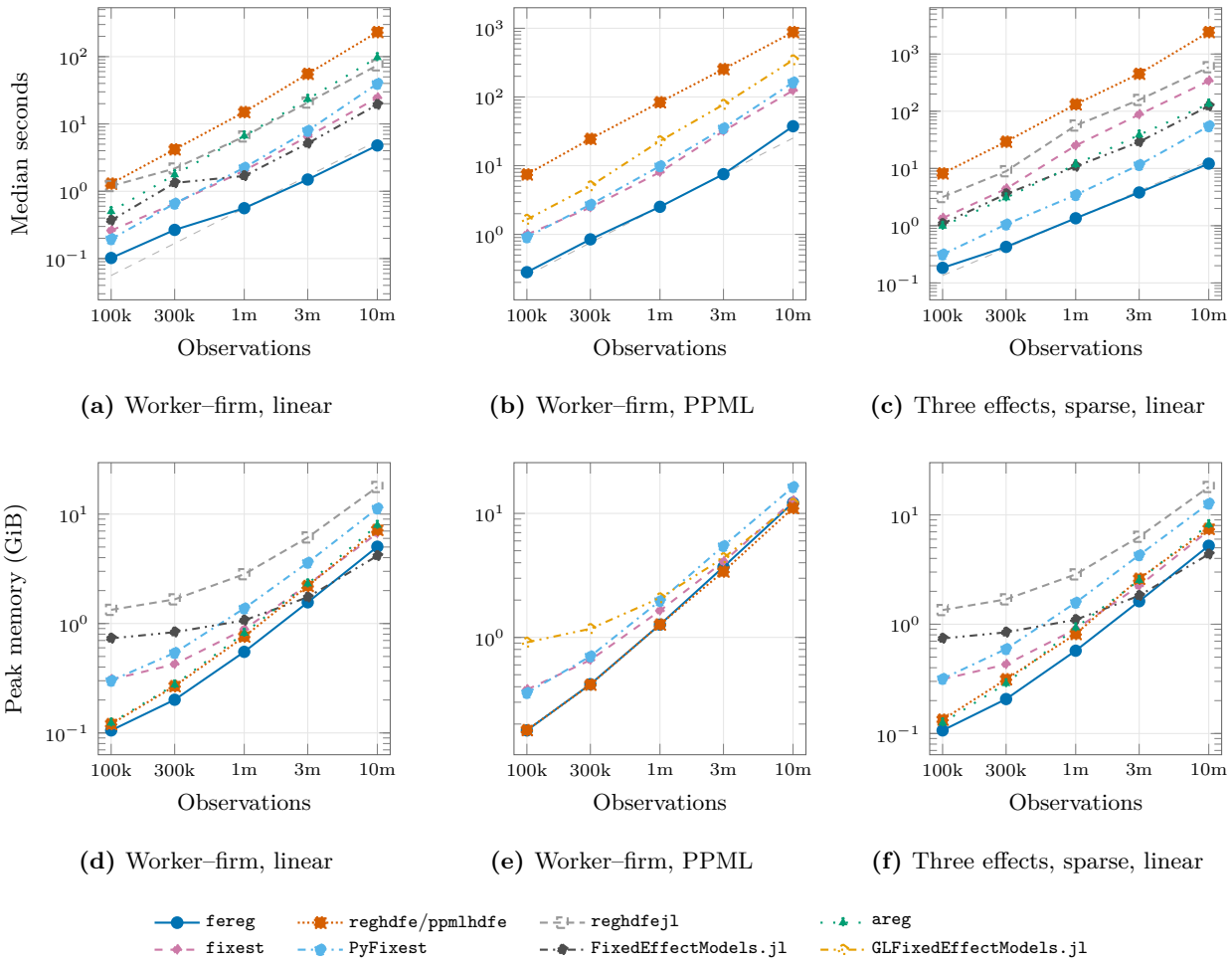
(a) Simulated, linear **(b)** Simulated, PPML **(c)** Veneto long panel, linear

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 over five (a, b) or three (c) comparison blocks per comparator; `fereg`'s medians pool its fits in all blocks. Both axes are logarithmic. The Stata license allowed at most four cores. `reghdfejl` and `FixedEffectModels.jl` were run for linear models and `GLFixedEffectModels.jl` for PPML. `GLFixedEffectModels.jl` and `reghdfejl` run on FixedEffectModels.jl 1.13.3 and FixedEffects.jl 2.7.0, whose fixed-effect solver uses one thread whatever the thread setting; their flat profiles do not reflect a setting of the comparison (section 7.5).

versions, not the thread settings, which started Julia with the assigned number of threads, checked it in every fit, and passed it to both packages. `GLFixedEffectModels.jl` 0.5.6, its current release (GLFixedEffectModels.jl contributors, 2026), requires FixedEffectModels.jl 1.x, and the environment that `reghdfejl` installs (Roodman, 2026) resolves its linear models to the same version, 1.13.3, with FixedEffects.jl 2.7.0, whose fixed-effect solver accepts a thread count but runs on one thread. The direct comparison uses `FixedEffectModels.jl` 2.1.0 with FixedEffects.jl 3.5.0, whose linear solver is threaded. The difference between `reghdfejl`'s 6.51 seconds at four cores and the 1.69 seconds of direct `FixedEffectModels.jl` consists mostly of passing the data from Stata to Julia, on one thread, and setup on the Stata side, both inside the timed command.

7.6 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 somewhat less than in proportion to the data, from 0.10 seconds to 4.8 seconds. The Stata commands fall further behind as the data grow: `reghdfe` took 12 times as long as `fereg` with 100,000 observations and 49 times with ten million, `areg` 4.3 and 16.9 times, and `reghdfejl` 11.8 and 17.6 times. `PyFixest`'s ratio grows from 1.9 to 8.5 and that of `fixest` from 2.5 to 5.1, while that of `FixedEffectModels.jl` changes little (3.7 to 4.1). For PPML with ten million observations, `feppml` took 37.5 seconds, `ppmlhdfe` 879, and `GLFixedEffectModels.jl` 345.



(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, at default stopping settings. Medians over the comparison blocks, which ran on different nodes. The dashed gray line grows in proportion to the data from `fereg`'s time with one million observations. Every command completed every size within the one-hour limit. `areg`, `reghdfejl`, and `FixedEffectModels.jl` were run for linear models and `GLFixedEffectModels.jl` for PPML.

Memory grows roughly in proportion to the data. With ten million observations, `fereg` needed 5.0 GiB in the linear model, against 7.2 GiB for `reghdfe`, 7.9 GiB for `areg`, 17.7 GiB for `reghdfejl`, 6.7 GiB for `fixest`, 11.3 GiB for `PyFixest`, and 4.2 GiB for `FixedEffectModels.jl`. In PPML, `feppml` needed 12.2 GiB, `ppmlhdfe` 11.1 GiB, `fixest` 12.6 GiB, `PyFixest` 16.3 GiB, and `GLFixedEffectModels.jl` 12.0 GiB.

The sparse design with three effects separates the commands most clearly. With ten million observations, `fereg` took 12.1 seconds and `reghdfe` 2419, about 204 times as long. `PyFixest` was the closest comparator, taking between 1.9 and 2.9 times as long as `fereg` through three million observations and 4.6 times with ten million. `fereg`'s peak memory at ten million observations, 5.2 GiB, was below that of every comparator except `FixedEffectModels.jl` (4.4 GiB); `PyFixest` needed 12.7 GiB.

7.7 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.

OLS took 5.5 seconds with one core and 1.9 with eight; PPML took 45.8 and 14.4 seconds. Sixteen cores did not help further (2.1 and 15.9 seconds; these medians pool different nodes). Every comparator took longer at every core count, at least 3.4 times as long. The closest, `PyFixest`, took 8.4 and 4.0 times as long in OLS with one and sixteen cores, and 6.8 and 3.4 times in PPML. Among the Stata commands, `reghdfe` took 130 times as long as `fereg` with one core and 116 times with four, `areg` 14–16 times, and `reghdfejl` 25–45 times; `ppmlhdfe` took 47–56 times as long as `feppml`. `GLFixedEffectModels.jl` took 42–95 times as long as `feppml`; at tighter settings, it reached the one-hour limit in seven of its 15 fits (online table C.4).

In OLS, `fereg`'s peak memory of 1.4–1.5 GiB was the lowest of all commands. In PPML, `feppml` needed 4.0–4.7 GiB, more than `fixest` and `GLFixedEffectModels.jl` and more than `ppmlhdfe` (3.2 GiB). Memory and time comparisons between the OLS and PPML panels reflect different estimators and outcome scales.

`fereg`'s own times can also be set against the version 0.5.0 measurements of November 2–3, which used the same configurations and data on other nodes. Over the 63 configurations, its default times were on average (geometric mean) 0.57 of the earlier ones, ranging from 0.24 to 1.05. This comparison crosses nodes and allocations and is descriptive; no ratio in this section divides a new time by an earlier one.

These comparisons describe specific designs, on one cluster's nodes and with three to five blocks per configuration and comparator, and are not a universal ranking.

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

Cores	`fereg`	`PyFixest`	`fixest`	Julia†	`reghdfe`/ `ppmlhdfe`	`reghdfejl`	`areg`
Panel A. OLS: log daily wages							
1	5.49	46.2 (8.4×)	75.7 (13.6×)	57.3 (10.4×)	713 (130×)	137 (24.8×)	79.6 (14.5×)
2	3.43	25.2 (7.3×)	43.5 (12.9×)	35.9 (10.4×)	410 (119×)	131 (38.8×)	53.4 (15.9×)
4	2.46	15.4 (6.3×)	24.7 (10.1×)	24.8 (10.1×)	287 (116×)	111 (45.3×)	37.8 (15.3×)
8	1.86	10.1 (5.4×)	15.1 (8.2×)	19.2 (10.4×)	—	—	—
16	2.13	8.75 (4.0×)	16.0 (6.5×)	33.8 (15.8×)	—	—	—
Panel B. PPML: daily wages in levels							
1	45.8	313 (6.8×)	411 (9.0×)	1,873 (41.9×)	2,172 (47.1×)	—	—
2	27.9	158 (5.7×)	235 (7.8×)	1,615 (57.8×)	1,400 (50.7×)	—	—
4	18.6	104 (5.5×)	142 (7.5×)	1,498 (79.0×)	1,053 (56.5×)	—	—
8	14.4	65.2 (4.5×)	89.7 (5.8×)	1,348 (95.1×)	—	—	—
16	15.9	53.8 (3.4×)	92.5 (5.8×)	1,500 (94.6×)	—	—	—

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. Default stopping settings and package-default standard errors. Each comparator ran in three comparison blocks with `fereg`; entries are median seconds, and parentheses give the median of the within-block ratios of the comparator's time to `fereg`'s. `fereg`'s median pools all its fits at that core count. `fereg`'s and the Stata commands' times include reading Stata's memory; other languages start with their own arrays (section 6.1). Stata was licensed for four cores (—). † `FixedEffectModels.jl` for OLS, `GLFixedEffectModels.jl` for PPML. `reghdfejl` and `areg` fit linear models only.

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 exact elimination of the largest effect and 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, at default stopping settings, `fereg` and `feppml` were many times faster than `reghdfe`, `areg`, `reghdfejl`, and `ppmlhdfe`, in the linear models with lower memory use; the advantage remained on a single core and, in linear models, grew with the size of the data. They were also faster than `fixest` and `PyFixest` in every simulated configuration and than the Julia packages in all but one, typically by factors of three to eight, and on the administrative panel every other command took at least 3.4 times as long at every number of cores. These comparisons use each package's default stopping rules and standard errors; the online appendix reports tighter settings and the numerical differences that accompany them.

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`'s modified LSMR engine was adapted from the MIT-licensed `schwarz-precond` crate of the `within` numerical library, developed by Alexander Fischer, Kristof Schröder, and contributors at the appliedAI Institute for Europe and in the `PyFixest` community, and the plugin uses a Rust port, version 0.3.0, 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. Fast poisson estimation with high-dimensional fixed effects. *Stata Journal*, 20(1):95–115, 2020. doi: 10.1177/1536867X20909691.

Sergio Correia, Paulo Guimarães, and Tom Zylkin. Verifying the existence of maximum likelihood estimates for generalized linear models. *Econometric Reviews*, 45(9):1270–1305, 2026. doi: 10.1080/07474938.2026.2673989.

ctools contributors. ctools: C-Accelerated Stata Commands. https://github.com/mdroste/stata-ctools, 2026. Software documentation; accessed November 6, 2026.

dbreg contributors. dbreg: Regressions on Database Backends. https://github.com/grantmcdermott/dbreg, 2026. Software documentation; accessed November 6, 2026.

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.

GLFixedEffectModels.jl contributors. GLFixedEffectModels.jl: Generalized Linear Models with High-Dimensional Fixed Effects. https://github.com/jmboehm/GLFixedEffectModels.jl, 2026. Software documentation; accessed November 6, 2026.

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.

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.

David Roodman. reghdfejl: Linear Regression with High-Dimensional Fixed Effects from Stata through Julia. https://github.com/droodman/reghdfejl, 2026. Software documentation; accessed November 6, 2026.

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, version 0.3.0 (GitHub commit 368eceb), 2026. URL https://github.com/johannes-schmieder/CMG/releases/tag/v0.3.0. Accessed November 11, 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.

Lee Tucker. hdfe-stream: Fixed-Effect Regression Using Polars Streaming. https://github.com/leetucker/hdfe-stream, 2026. Software documentation; accessed November 6, 2026.

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.

xhdfe contributors. xhdfe: Linear Regression with Multiple High-Dimensional Fixed Effects. https://github.com/reisportela/xhdfe-xfe, 2026. Software documentation; accessed November 6, 2026.

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.