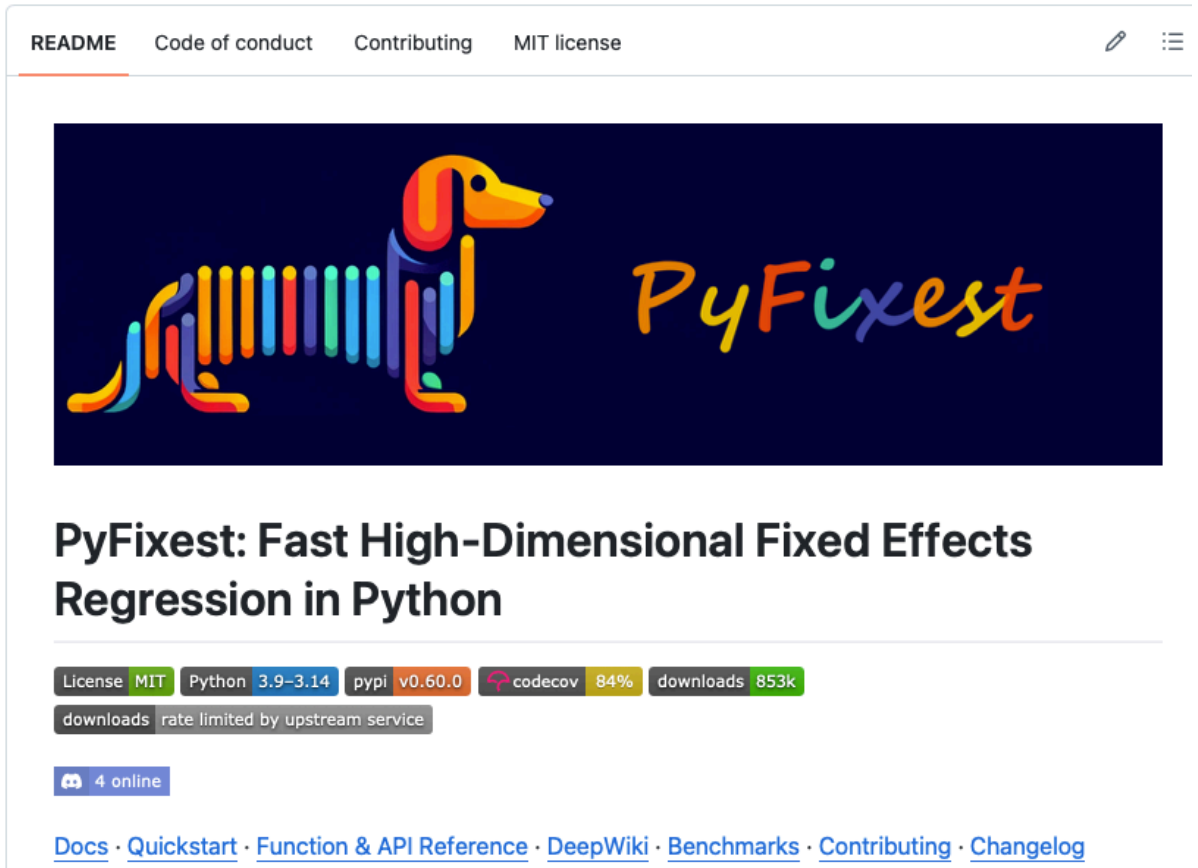
From “MUCH slower” to “Converges and runs really fast”

Mostly Harmless Fixed-Effects Regression via an Additive Schwarz-Preconditioned Krylov Solver

Alexander Fischer

Data Scientist at trivago

What I do in my free time



How it started ...

pyfixest is MUCH slower than (Stata + Julia) reghdfejl

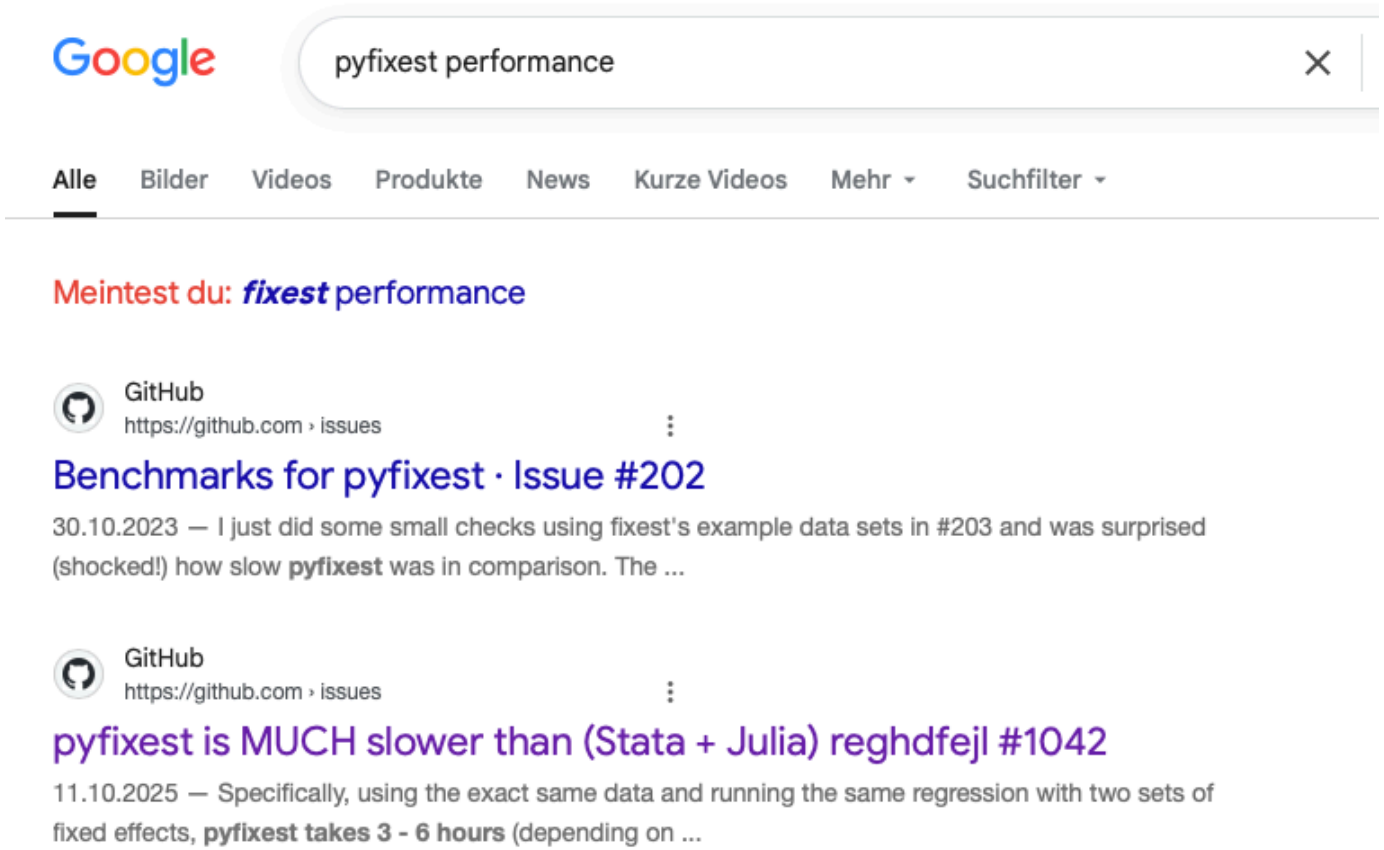
Open #1048

rhstanton opened on Oct 12, 2025

First, thanks very much for this package. It gets me a lot closer to being able to do everything in Python! However, I've noticed an enormous speed difference between pyfixest and the Stata (+ Julia) function reghdfejl. Specifically, using the exact same data and running the same regression with two sets of fixed effects, pyfixest takes 3 - 6 hours (depending on what I set the tolerance to), while reghdfejl takes something like 3 *minutes*. That's quite a difference! The results generated are very similar in both cases.

I'm using pyfixest 0.30.2 with jax 0.7.2 and cuda 13. pyfixest is definitely using the GPU (Nvidia 3090Ti), which shows 100% utilization during the entire run, has something over 18GB RAM allocated, and heats up my office nicely. reghdfejl is a Stata routine that calls Julia to do the heavy processing, and again I've set it to use the GPU.

Anyone here doing SEO?



PyFixest is for Fixed Effects

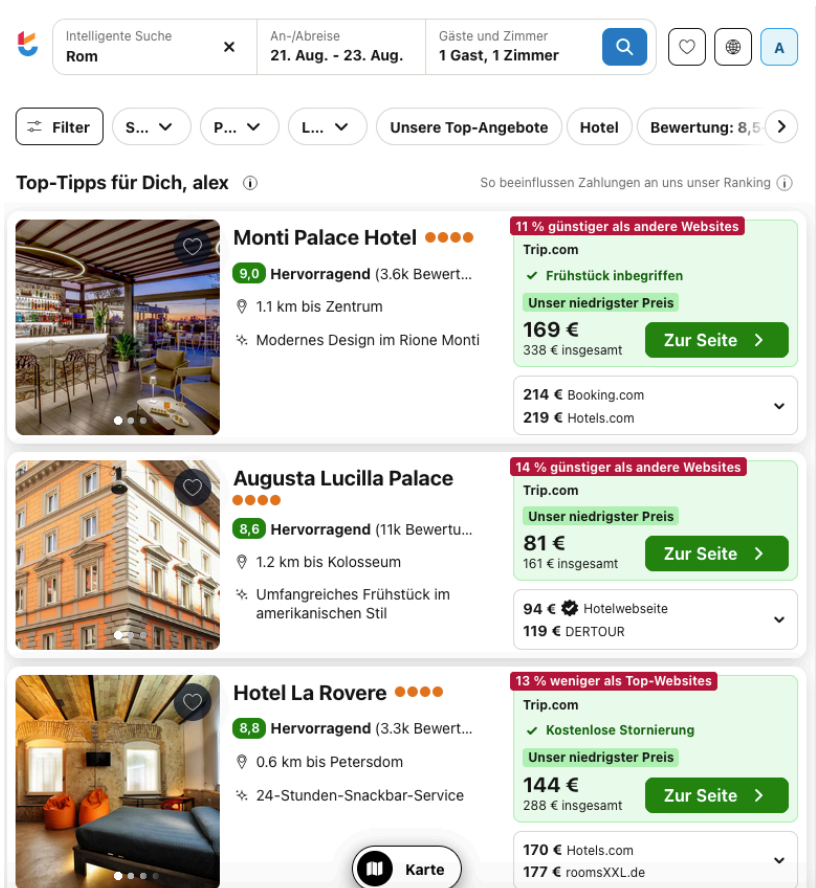
Modeling click-through rates:

$$y = X\beta + D\alpha + \varepsilon$$

A fixed effect is one learned baseline for each entity or category.

- X: rate placement and price.
- D: one-hot columns for item, rank, etc.
- α: fixed effects.

```
is_clicked ~ rate_placemenr + price
           | item_id + item_rank + ...
```



High cardinality makes the dummy matrix enormous

High cardinality means a categorical variable has thousands or millions of distinct values.

$D_{\text{partner}} \in \{0,1\}^{N \times k}$: N impressions $\times$ k item dummies

	A	B	C	D	$\cdots$
a_1	1	0	0	0	0
a_2	0	1	0	0	0
a_3	0	0	1	0	0
a_4	0	1	0	0	0
a_5	0	0	0	1	0

Naive dense normal equations do not scale

$$Z = [X \quad D] \in \mathbb{R}^{n \times k}$$

$$\hat{\theta} = (Z^\top Z)^{-1} \; Z^\top y$$

Gram matrix

$(k \times n)(n \times k) \to k \times k$

$O(nk^2)$

Matrix inversion

$(k \times k)^{-1} \to k \times k$

$O(k^3)$

Cross-product

$(k \times n)(n \times 1) \to k \times 1$

$O(nk)$

> With high-cardinality fixed effects, k is too large to form and invert this dense system.

Economists do not want to use SGD.

The economist's trick: FWL removes fixed effects first

1. Demean

$\tilde{y} = \text{demean}(y, \text{FE})$ $\tilde{X} = \text{demean}(X, \text{FE})$

→

2. Regress

$\tilde{y} = \tilde{X}\beta + \tilde{\varepsilon}$

→

3. Same $\hat{\beta}$

No fixed-effect dummies in the final regression.

> **Frisch-Waugh-Lovell:** demeaning / residualizing y and X against D gives the same coefficient $\hat{\beta}$ as the full dummy regression. Demeaning is often the core computational bottleneck.

Demeaning is OLS of μ on D

For each target $\mu \in \{y, X_1, \ldots, X_p\}$:

$$\hat{\alpha}_\mu = \arg\min_\alpha \frac{1}{2} \| D\alpha - \mu \|_2^2$$

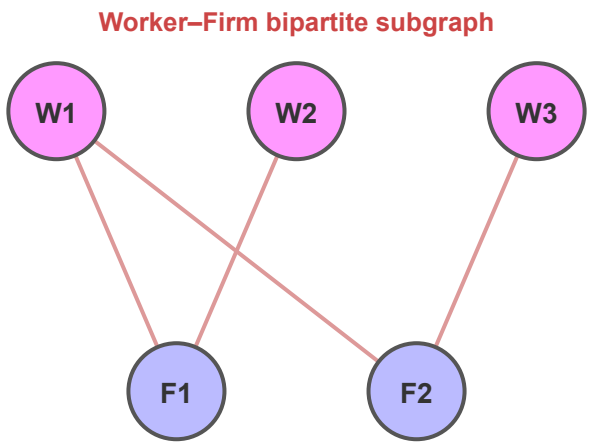
$$\hat{\alpha}_\mu = D^\top \mu, \quad G = D^\top D$$

A core question from Labour Economics: AKM

> Does a worker earn more because of their own skills or because they work at a high-paying firm?

$$\text{wage} = X\beta + \text{worker} + \text{firm} + \varepsilon$$

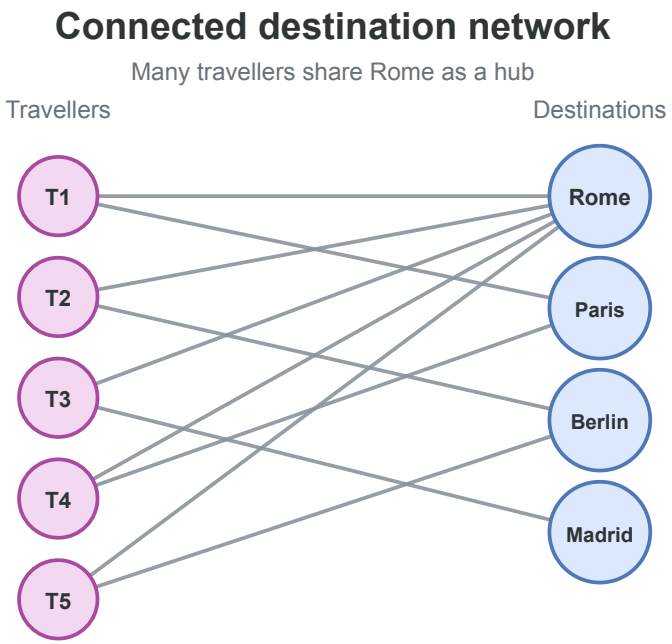
- Worker effects capture persistent differences between people.
- Firm effects capture persistent pay differences between employers.

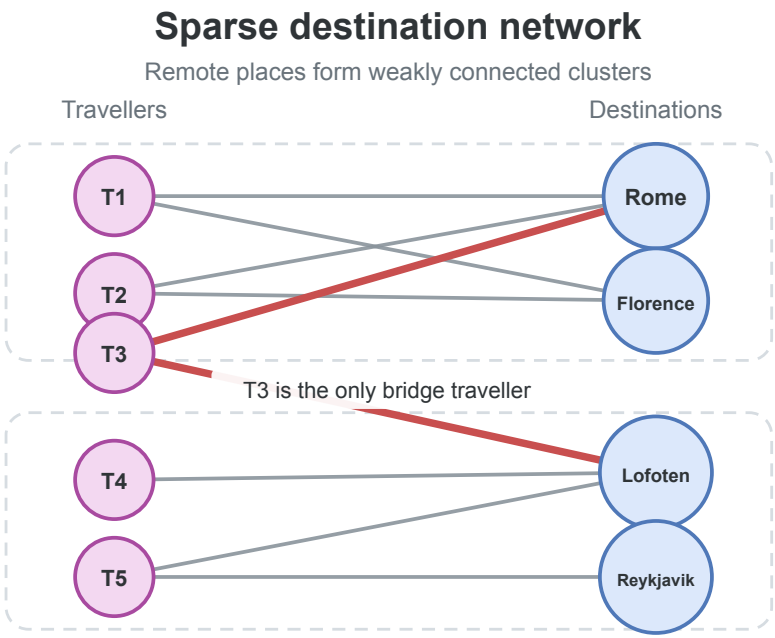


Single connected component (W1 bridges both firms)

- Movers connect firms and let us separate the two effects.
- Stayers alone cannot tell us which side explains the wage.

The same graph appears at GetYourGuide





> A traveller visiting multiple destinations is a **mover**. Rome connects the left-hand graph; on the right, one traveller is the only bridge to remote places.

The Worker-Firm Graph is Encoded in the Gramian

The same worker-firm data:

Obs	Worker	Firm	Wage
1	W1	F1	3.2
2	W1	F2	4.1
3	W2	F1	2.8
4	W2	F1	3.9
5	W3	F2	5.0
6	W3	F2	4.5

Column order:

$$(\text{W1}, \text{W2}, \text{W3}, \text{F1}, \text{F2})$$

$G = D^\top D$ becomes:

$$G = \begin{pmatrix} G_{WW} & G_{WF} \\ G_{WF}^{\text{top}} & G_{FF} \end{pmatrix} = \begin{array}{|cccc|c|}\hline \multicolumn{4}{|c|}{\multirow{4}{*}{$\begin{matrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 1 & 2 & 0 & 3 \end{matrix}$}} & \begin{matrix} 1 \\ 0 \\ 2 \\ 0 \end{matrix} \\ \hline & & & & \begin{matrix} 1 \\ 0 \\ 0 \\ 2 \end{matrix} \\ \hline \end{array}$$

The cross-tabulation block is:

$$G_{WF} = \begin{pmatrix} 1 & 1 \\ 2 & 0 \\ 0 & 2 \end{pmatrix}$$

- diagonal entries: **group counts**
- cross-factor entries: **worker-firm links**

Takin' notes: A sign flip turns the Gramian into a graph Laplacian

Flip the sign of the firm coordinates while leaving the worker coordinates unchanged:

$$T = \operatorname{diag}(I_W, -I_F) = \operatorname{diag}(1, 1, 1, -1, -1)$$

$$L = TGT = \begin{array}{|ccccc|}\hline \mathbf{2} & 0 & 0 & -1 & -1 \\ 0 & \mathbf{2} & 0 & -2 & 0 \\ 0 & 0 & \mathbf{2} & 0 & -2 \\ -1 & -2 & 0 & \mathbf{3} & 0 \\ -1 & 0 & -2 & 0 & \mathbf{3} \\ \hline \end{array}$$

positive degrees on the diagonal

negative links off the diagonal

every row sums to zero

Demeaning Workhorse: The Method of Alternating Projections (MAP)

Recall: demeaning is an OLS regression of μ on D.

Start from the full block system:

$$\begin{pmatrix} G_{WW} & G_{WF} \\ G_{WF}^{\text{top}} & G_{FF} \end{pmatrix} \begin{pmatrix} \alpha_W \\ \alpha_F \end{pmatrix} = \begin{pmatrix} D_W^\top \mu \\ D_F^\top \mu \end{pmatrix}$$

Expand by block row:

$$\begin{aligned} G_{WW}\,\alpha_W + G_{WF}\,\alpha_F &= D_W^\top \mu \\ G_{WF}^\top\,\alpha_W + G_{FF}\,\alpha_F &= D_F^\top \mu \end{aligned}$$

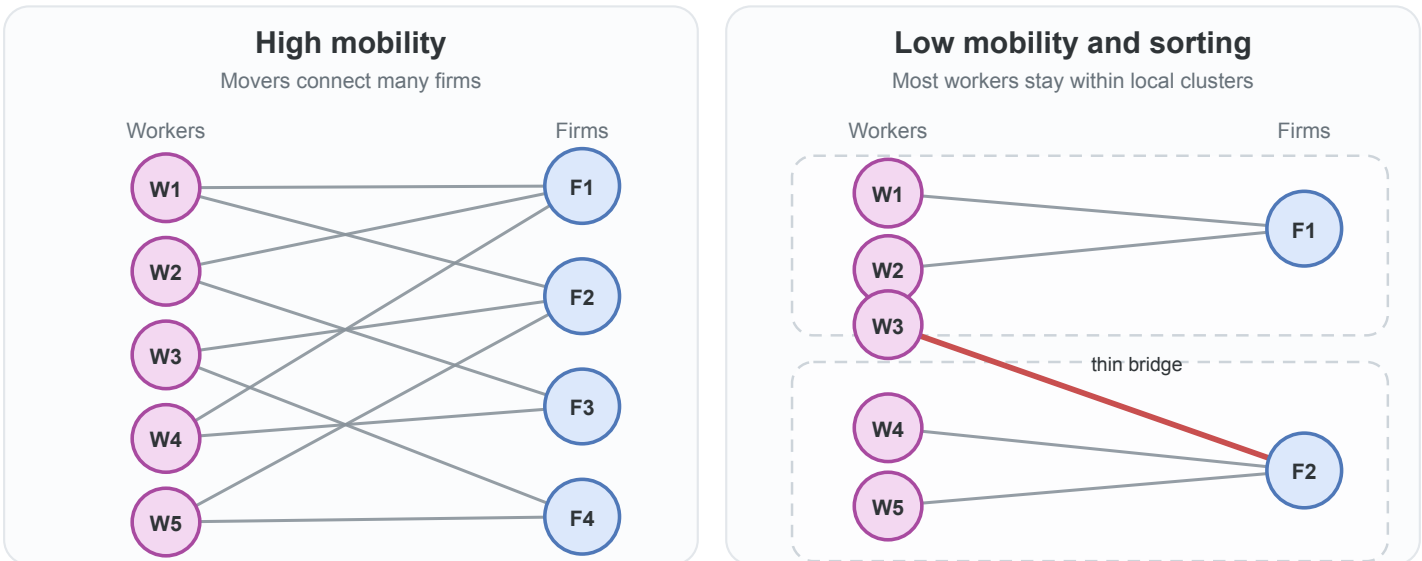
$\longrightarrow$

Rearrange one row at a time:

$$\begin{aligned} G_{WW}\,\alpha_W &= D_W^\top(\mu - D_F\alpha_F) \\ G_{FF}\,\alpha_F &= D_F^\top(\mu - D_W\alpha_W) \end{aligned}$$

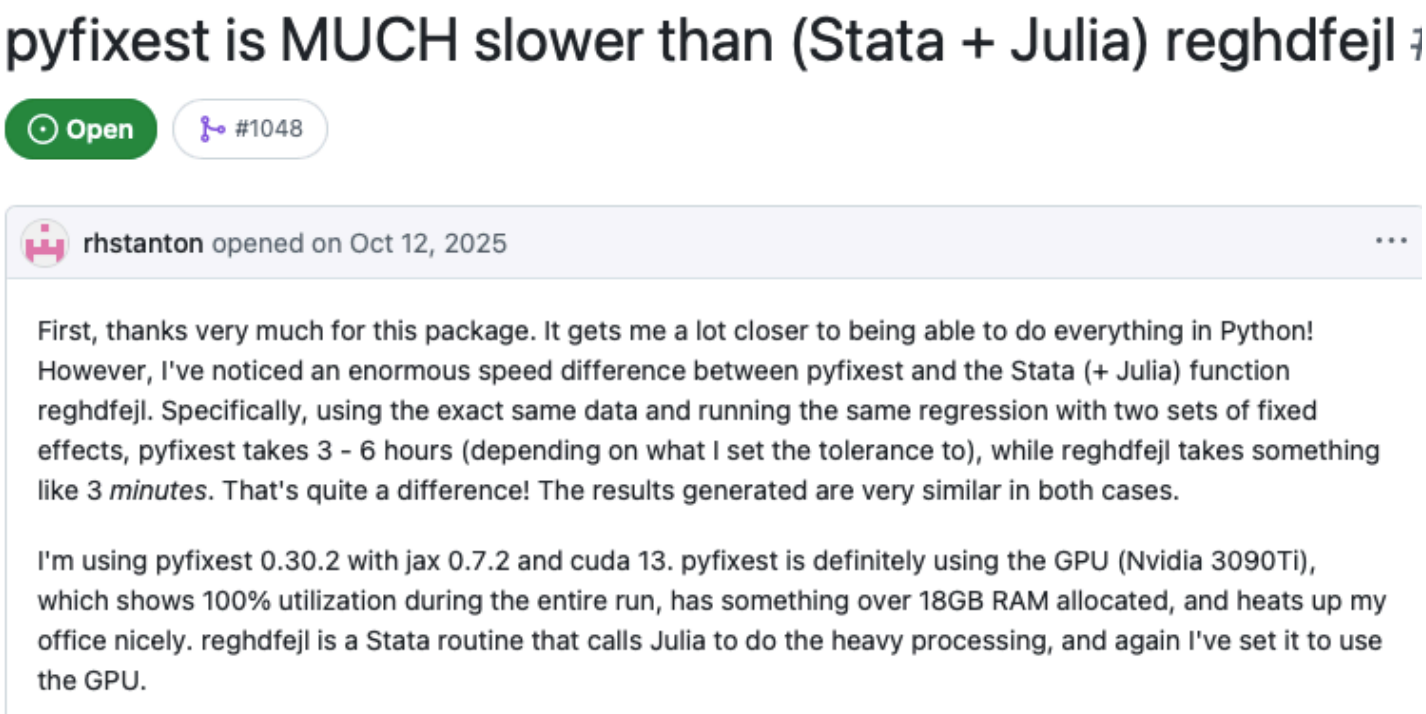
MAP: Solve for one fixed effect, holding the other constant. Use the updated coefficients in the next block solve. Repeat until convergence.

MAP's bottleneck: poorly connected graphs



When workers rarely move, a higher worker effect can be offset by a lower firm effect. MAP needs many back-and-forth updates to separate them.

Recall this?



pyfixest is MUCH slower than (Stata + Julia) reghdfejl

Open #1048

rhstanton opened on Oct 12, 2025

First, thanks very much for this package. It gets me a lot closer to being able to do everything in Python! However, I've noticed an enormous speed difference between pyfixest and the Stata (+ Julia) function reghdfejl. Specifically, using the exact same data and running the same regression with two sets of fixed effects, pyfixest takes 3 - 6 hours (depending on what I set the tolerance to), while reghdfejl takes something like 3 *minutes*. That's quite a difference! The results generated are very similar in both cases.

I'm using pyfixest 0.30.2 with jax 0.7.2 and cuda 13. pyfixest is definitely using the GPU (Nvidia 3090Ti), which shows 100% utilization during the entire run, has something over 18GB RAM allocated, and heats up my office nicely. reghdfejl is a Stata routine that calls Julia to do the heavy processing, and again I've set it to use the GPU.

Exactly what happened here.

Recall that the worker-firm links are encoded in the Gramian

Across a thin bridge, MAP can move a correction only one factor update at a time.

MAP: separate updates

$$G_{WW}\,\Delta\alpha_W = r_W$$

recompute the residual

$$G_{FF}\,\Delta\alpha_F = r_F$$

Information reaches the other factor on a later update.

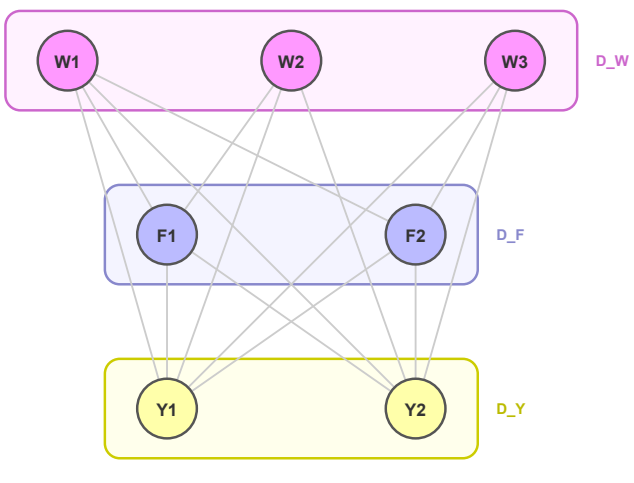
Joint factor-pair update

$$\begin{pmatrix} G_{WW} & G_{WF} \\ G_{WF}^{\top} & G_{FF} \end{pmatrix} \begin{pmatrix} \Delta\alpha_W \\ \Delta\alpha_F \end{pmatrix} = \begin{pmatrix} r_W \\ r_F \end{pmatrix}$$

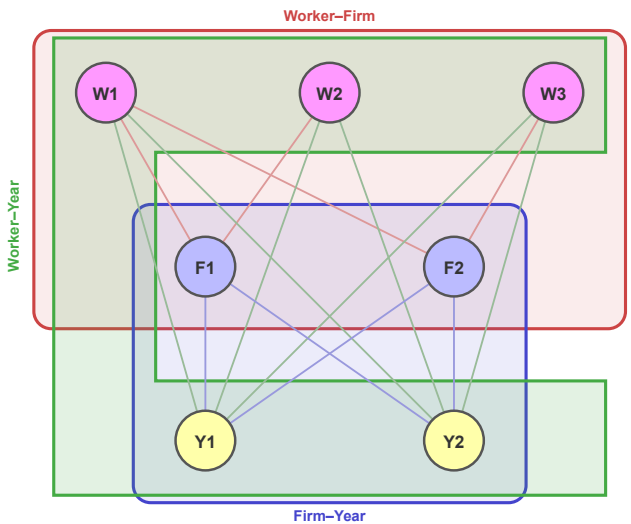
The **off-diagonal blocks** record the worker-firm links.

Pairing factors uses graph information

One block per factor



One block per factor pair



Core idea: Use the off-diagonal links somehow…

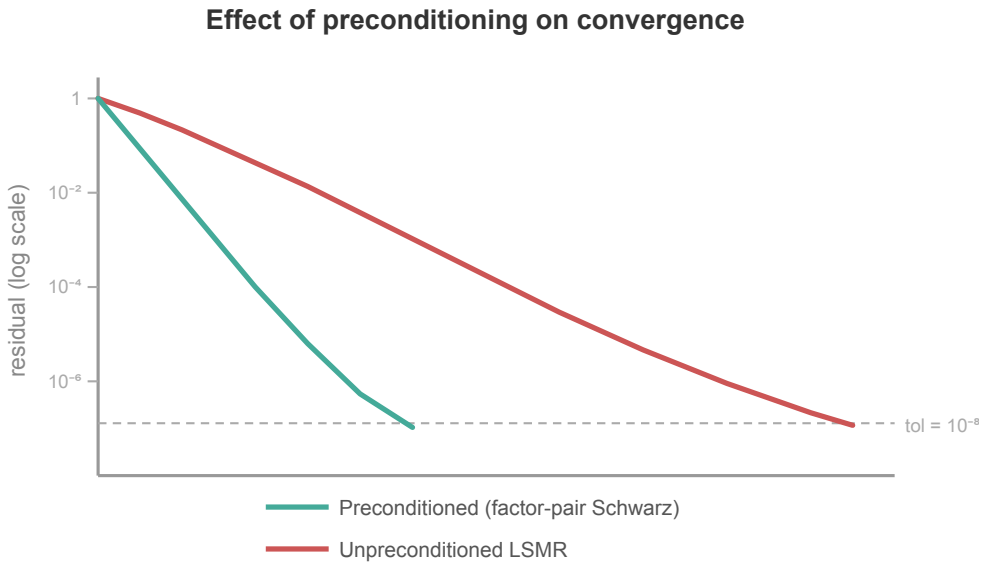
How about using a Additive Schwarz (Pre-) Conditioner...?

...for poorly conditioned systems.

Preconditioning reduces iterations without changing the solution

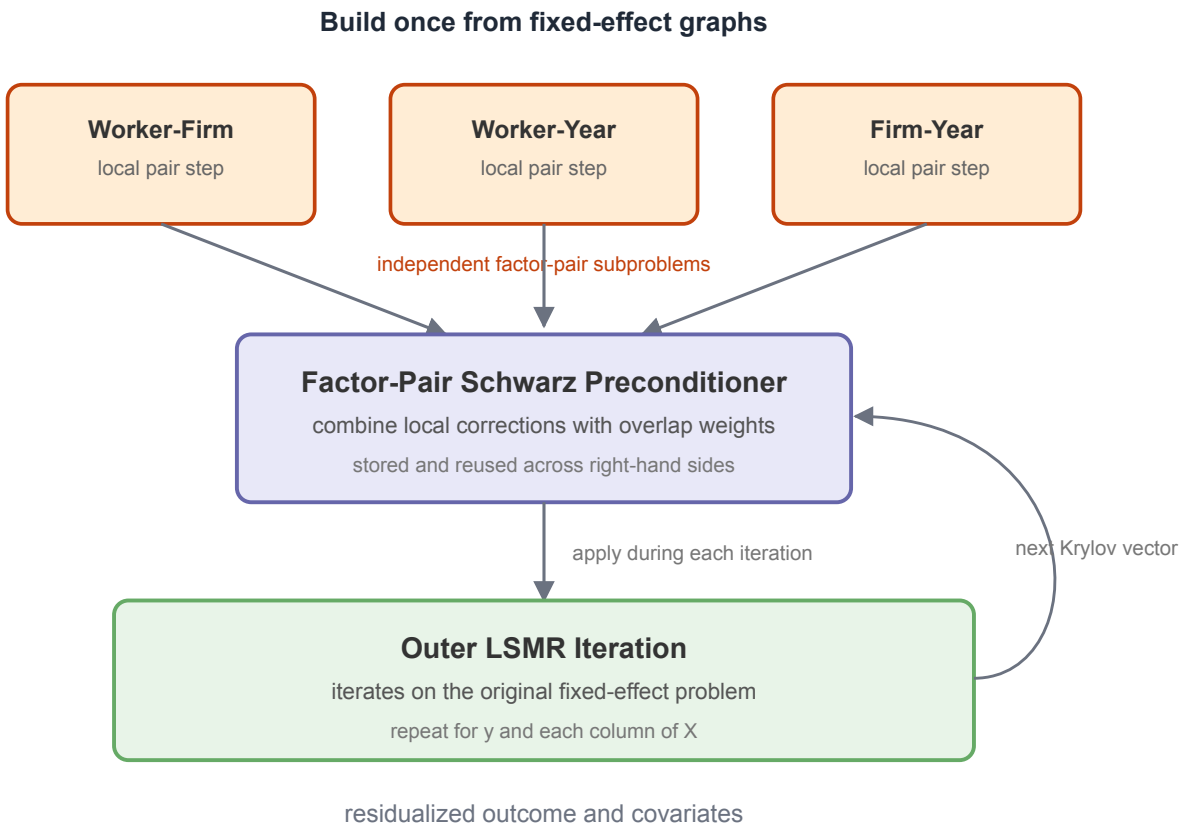
$$G\alpha = b$$

$$M^{-1}G\alpha = M^{-1}b$$



A good M^{-1} approximates G^{-1} closely enough to accelerate an iterative solver. The approximation needs to be low cost!

Factor-pair corrections precondition the global LSMR iteration



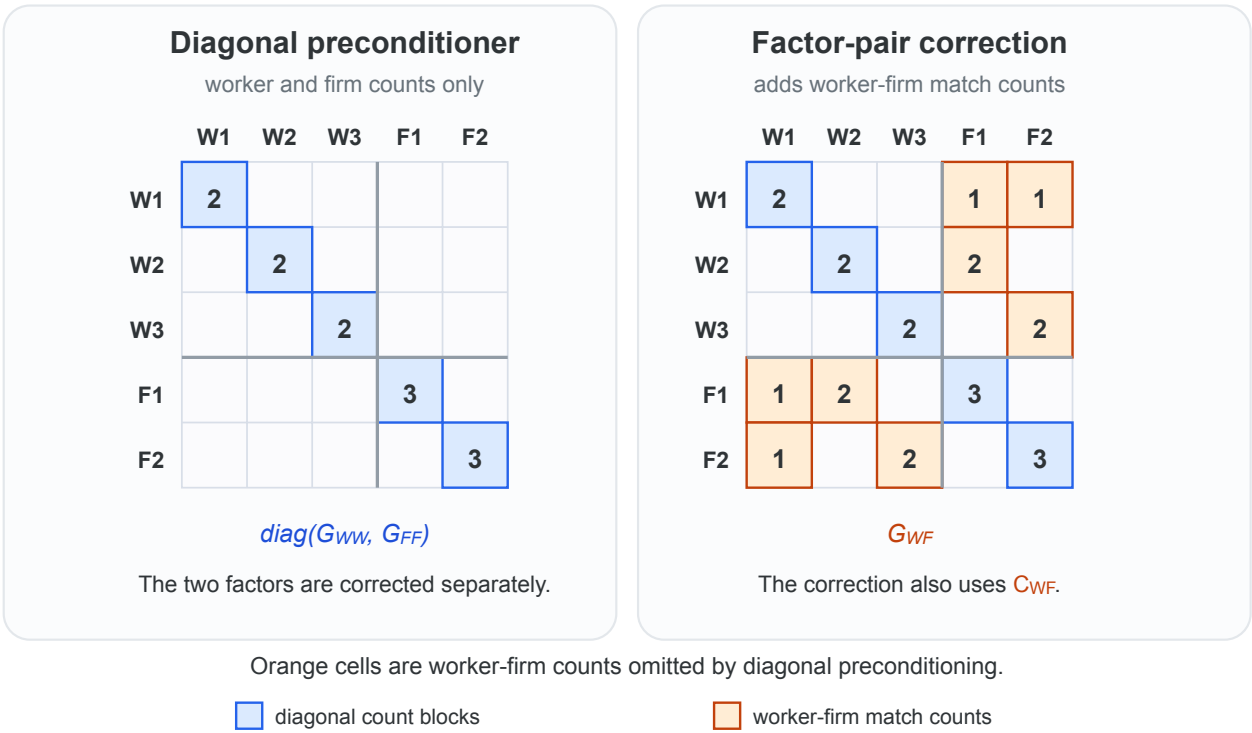
Schwarz corrects connected factor pairs

MAP applies a diagonal inverse:

$$M_{\mathrm{MAP},W}^{-1} = G_{WW}^{-1}.$$

Schwarz applies a coupled pair inverse:

$$A_{WF} \approx \begin{pmatrix} G_{WW} & G_{WF} \\ G_{WF}^{\top} & G_{FF} \end{pmatrix}^{-1}$$



Task: Approximate G_{pair}^{-1} cheaply for each pair

Pair block

$$G_{\text{pair}}$$

Exact inversion is costly

$\rightarrow$

Sign flip

$$T = \text{diag}(I_W, -I_F)$$

$$L = TG_{\text{pair}}T$$

$\rightarrow$

Laplacian

$$L$$

Sparse graph structure

$\rightarrow$

Approx. Cholesky

$$L \approx RR^\top$$

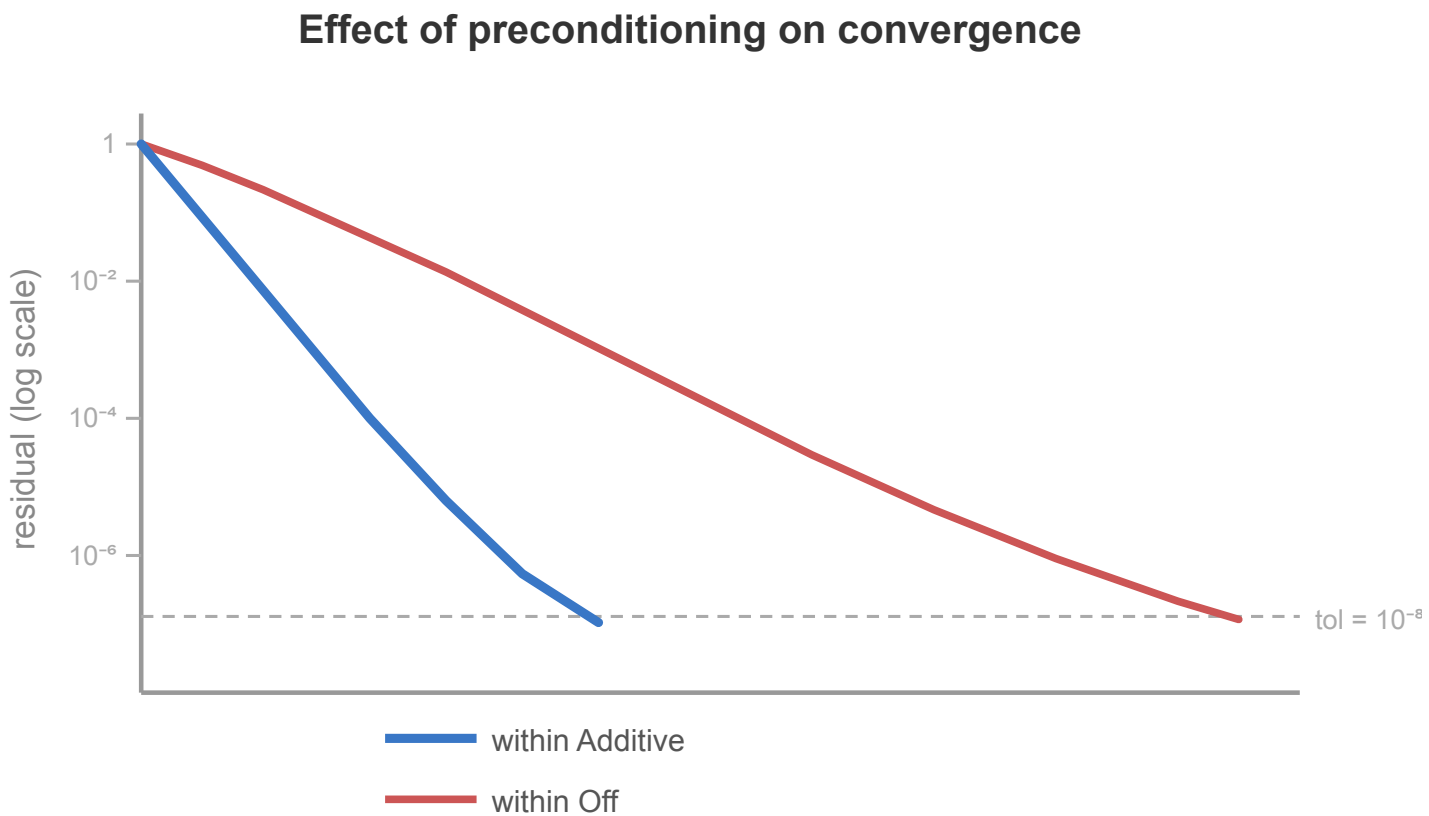
Build sparse R once

Gao, Kyng, and Spielman (2026), *AC(k): Robust Solution of Laplacian Equations by Randomized Approximate Cholesky Factorization*, SIAM J. Sci. Comput.

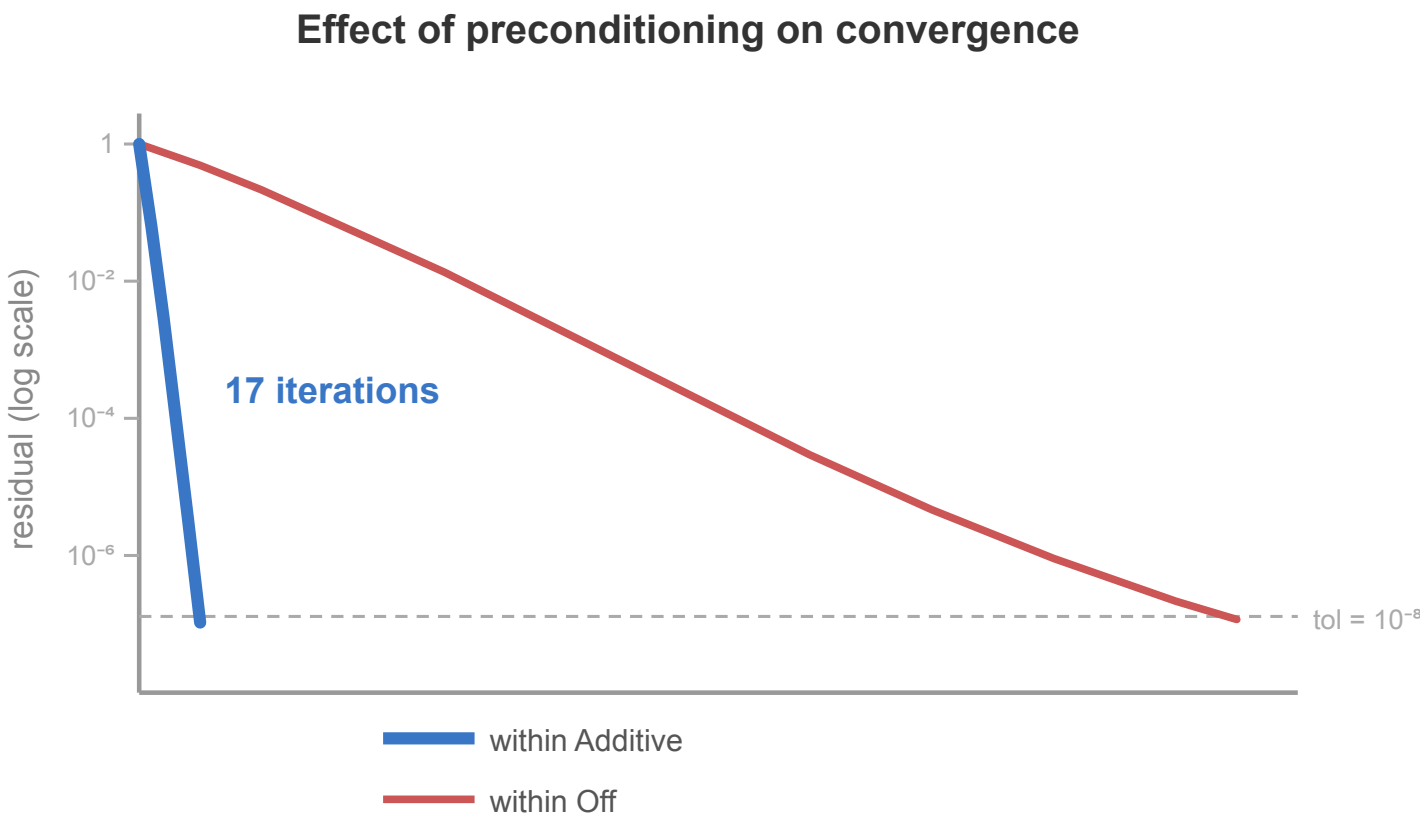
1,582 → 17

Configuration	Iterations	Median time	Converged
Off	1,582	0.101 s	yes
Additive	**17**	**0.0049 s**	yes

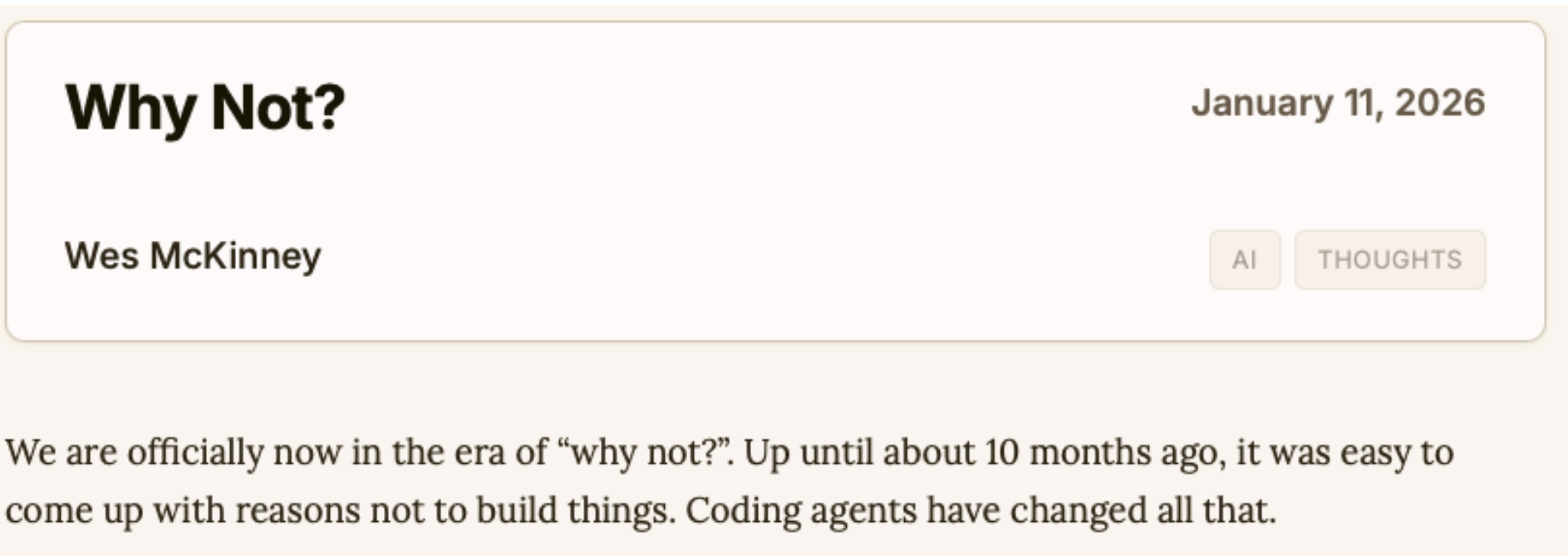
What $1,582 \to 17$ looks like



What $1,582 \to 17$ looks like



“Why not?”



Why Not?

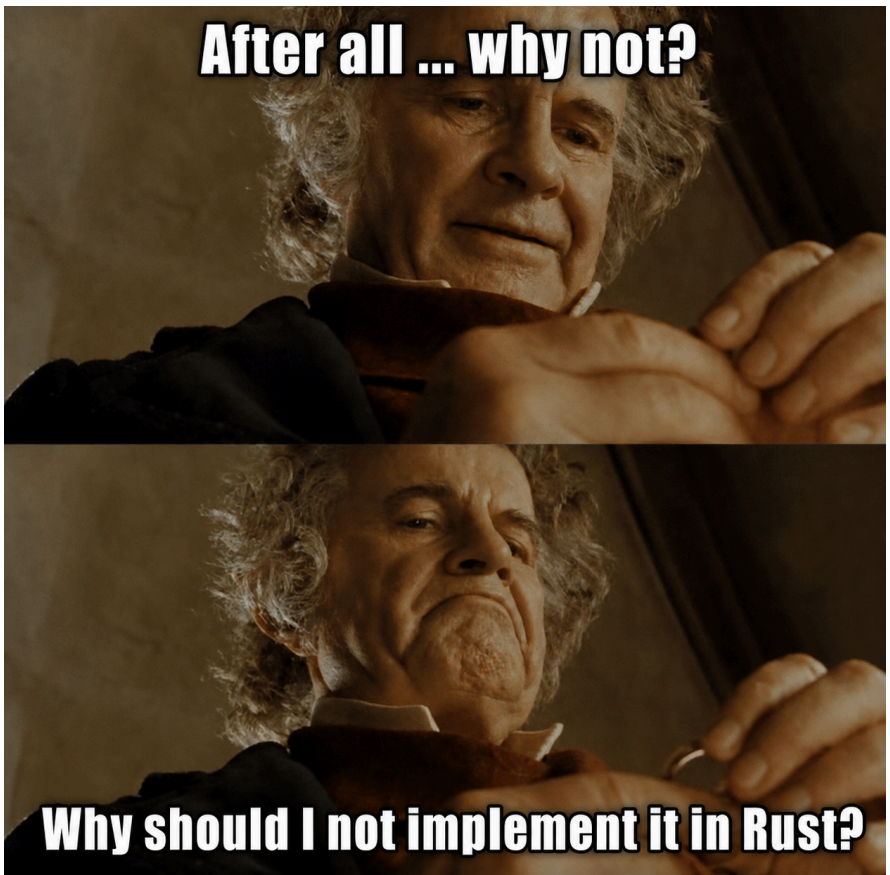
January 11, 2026

Wes McKinney

AI THOUGHTS

We are officially now in the era of “why not?”. Up until about 10 months ago, it was easy to come up with reasons not to build things. Coding agents have changed all that.

Bilbo agrees: “Why not?”



Kristof thought: why not implement it in Rust?

How we split the work

Use it directly or through PyFixest

within

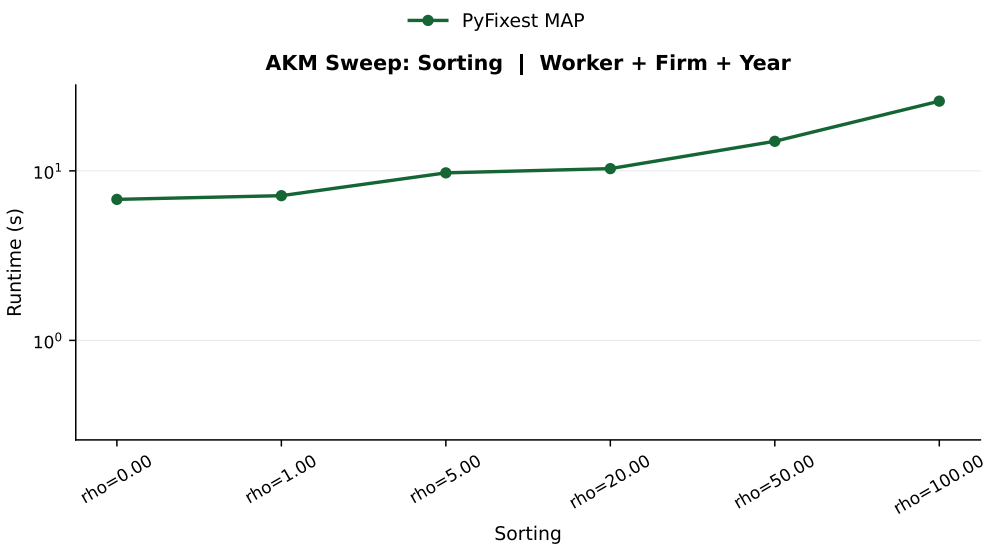
```python
from within import solve_batch
targets = np.c_[y, X]
r = solve_batch(fe, targets)
y0, X0 = r.demeaned[:, 0], r.demeaned[:, 1:]
beta = np.linalg.lstsq(X0, y0)[0]
```

PyFixest

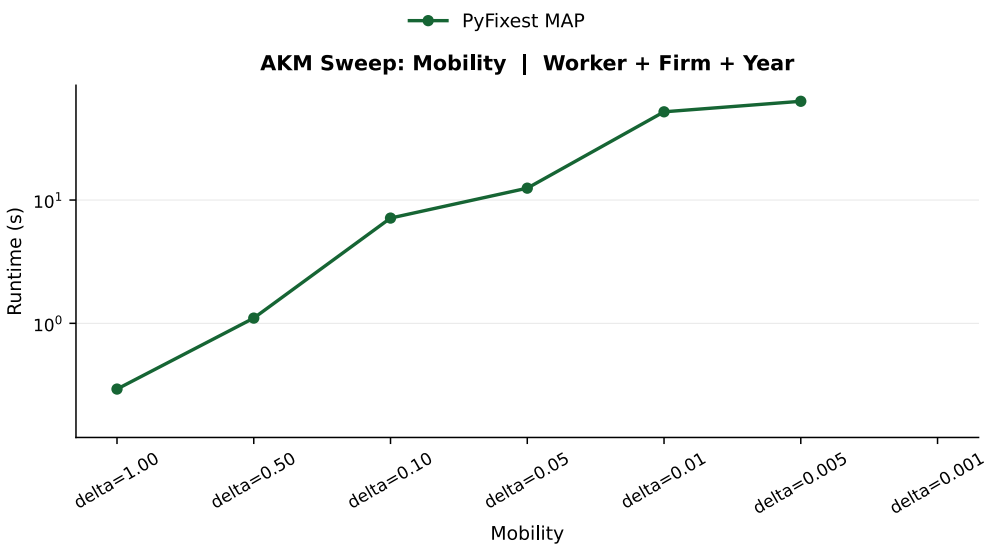
```python
import pyfixest as pf
solver = pf.LsmrDemeaner(backend="within")
ols = pf.feols("Y ~ X1 | f1 + f2", data,
demeaner=solver)
pois = pf.fepois("Y ~ X1 | f1 + f2", data,
demeaner=solver)
```

Benchmarks (1/5): PyFixest MAP

Increasing sorting

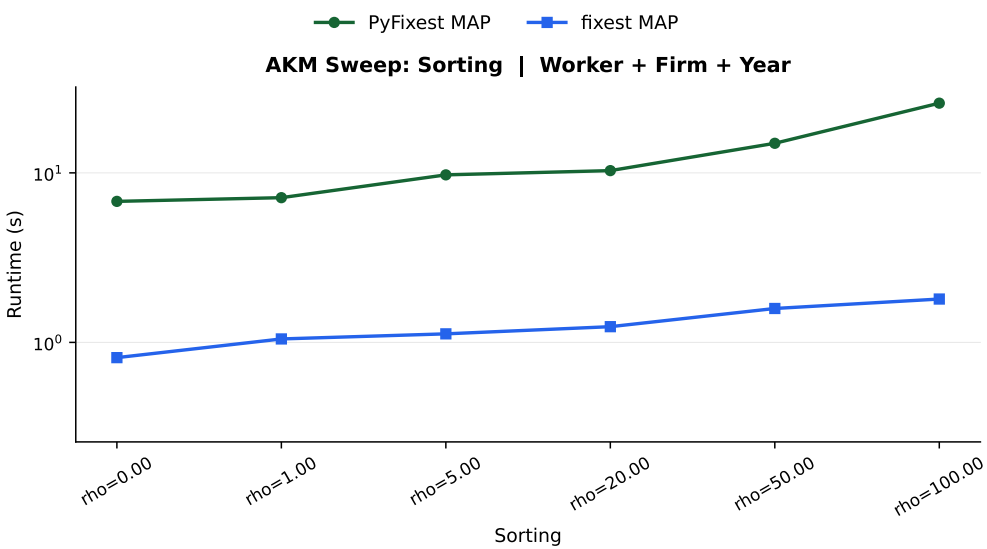


Decreasing mobility

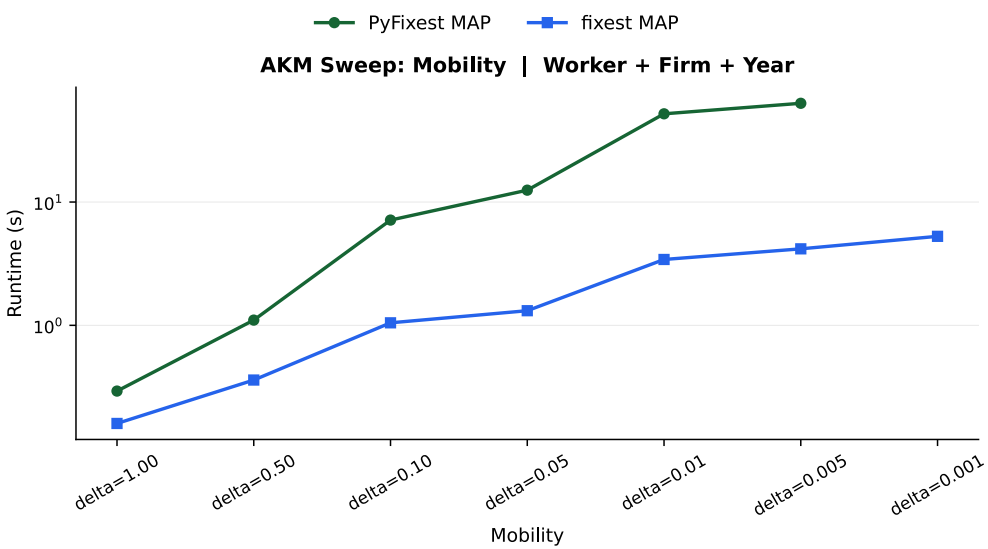


Benchmarks (2/5): + R fixest MAP

Increasing sorting

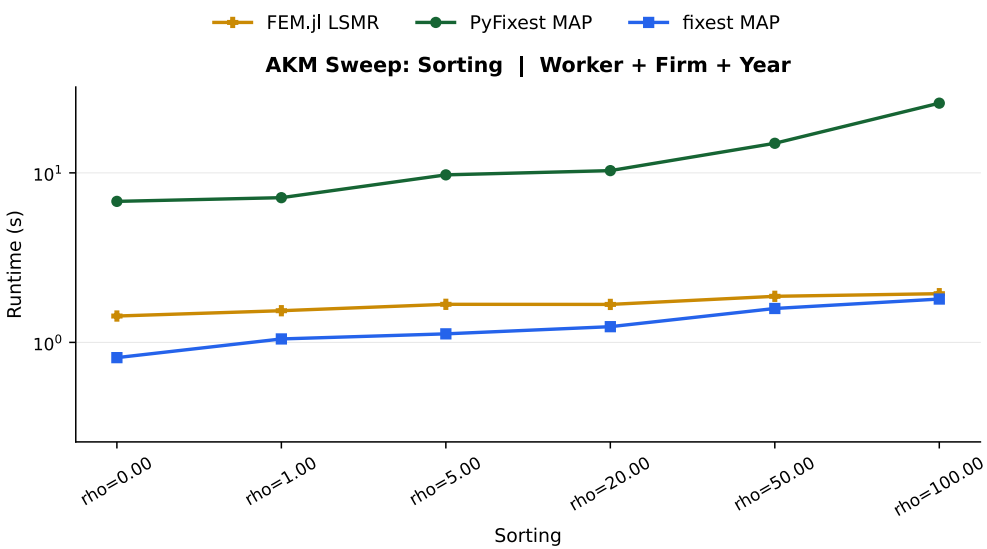


Decreasing mobility

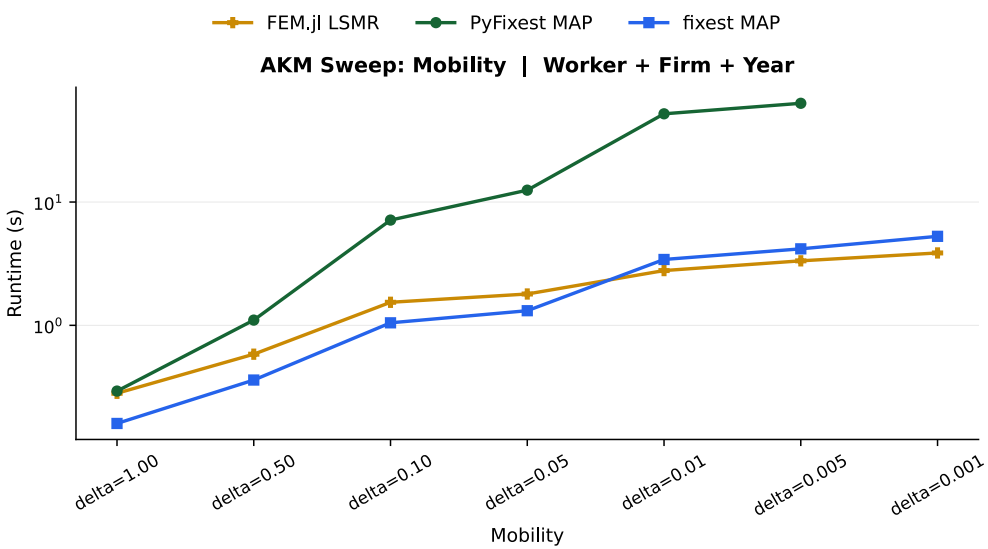


Benchmarks (3/5): + Julia LSMR

Increasing sorting

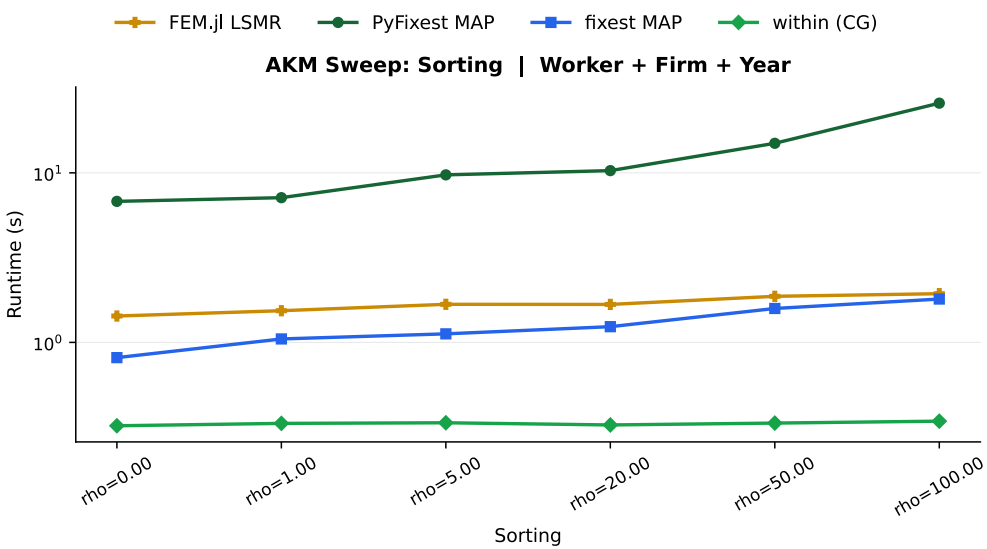


Decreasing mobility

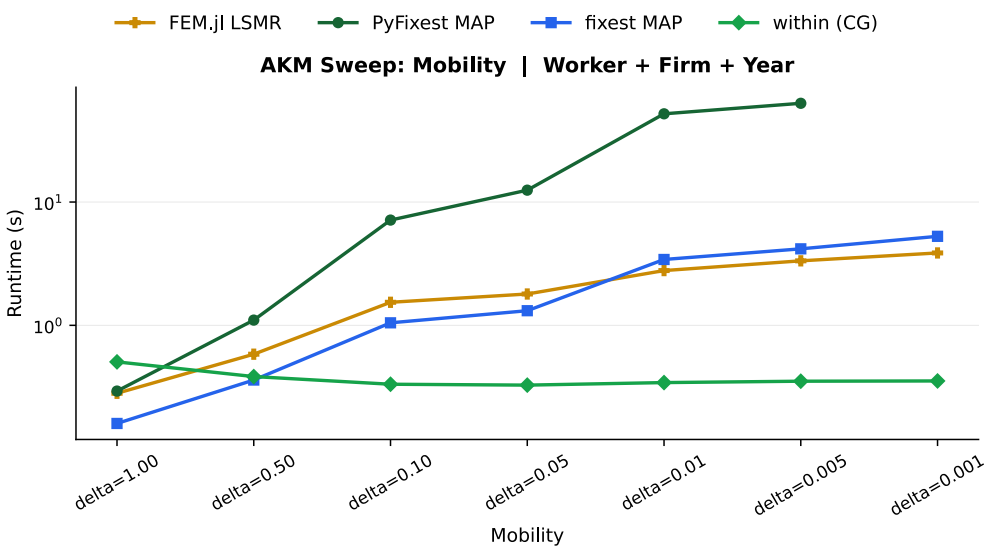


Benchmarks (4/5): + within (CG)

Increasing sorting

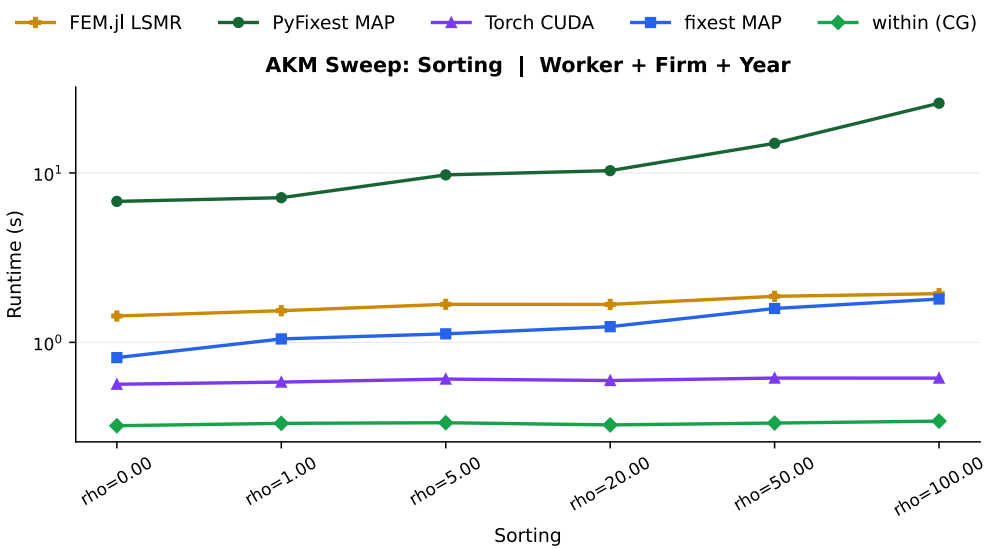


Decreasing mobility

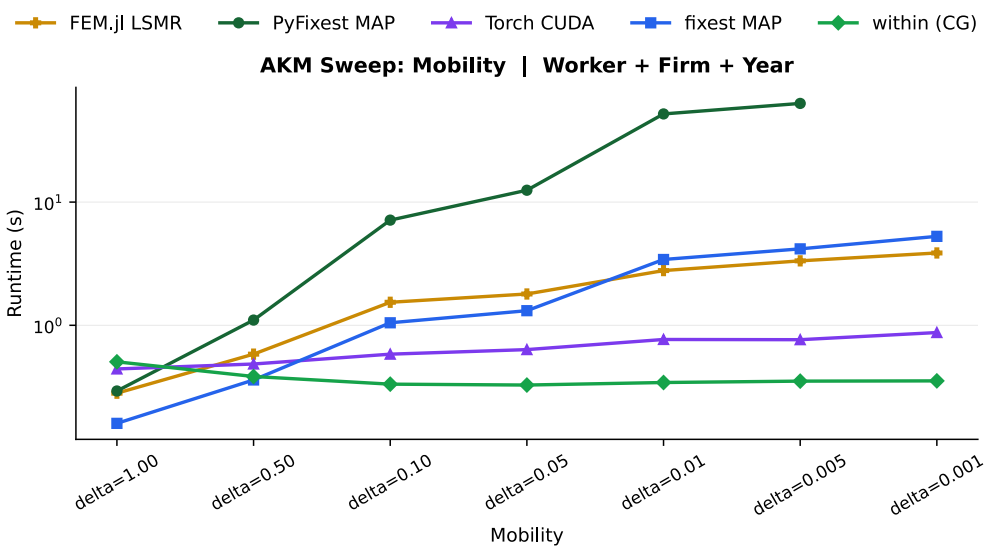


Benchmarks (5/5): + Torch (CUDA)

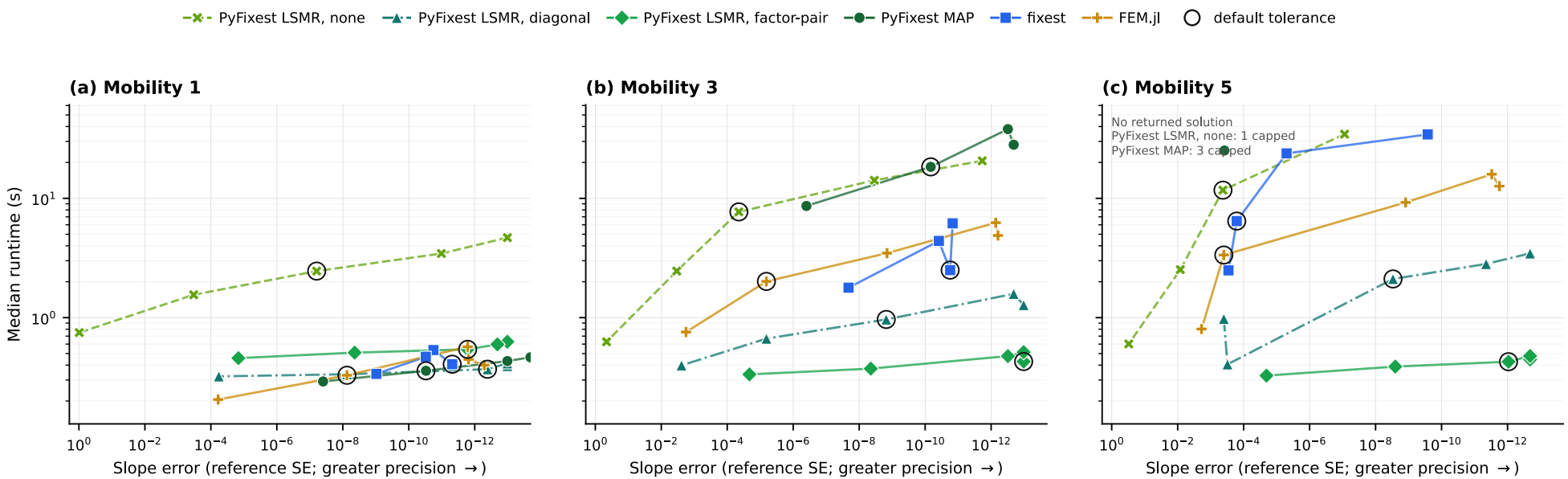
Increasing sorting



Decreasing mobility



Factor-pair preconditioning stays fast at the same accuracy



At the same coefficient error, factor-pair preconditioning remains fast as connectivity weakens.

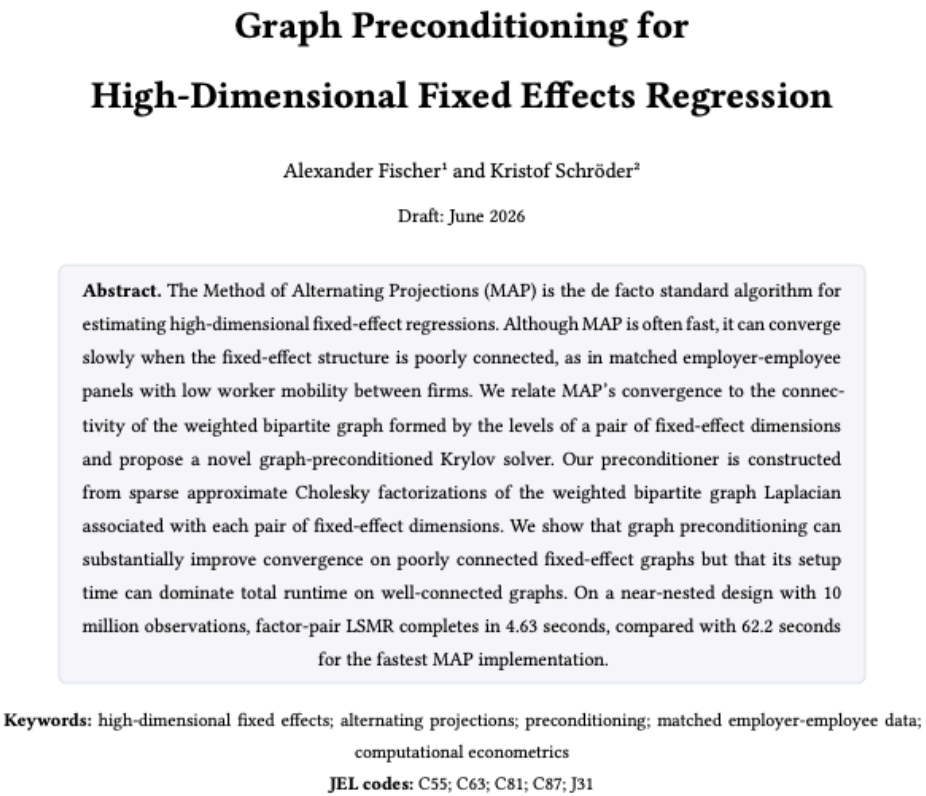
The same preconditioner can be reused for Poisson

1M observations; median of three full `fepois` calls; common 100-step IRLS cap.

Design	FE	PyFixest MAP	fixest	GLFEM.jl	PyFixest LSMR factor-pair
simple (well-connected)	3	7.95s	4.63s	5.71s	9.74s
difficult (near-nested)	3	capped (0/3)	439.3s	129.8s	**5.52s**

> The first IRLS step builds the factor-pair preconditioner; later weighted least-squares steps reuse it.

Thank you!



Graph Preconditioning for High-Dimensional Fixed Effects Regression

Alexander Fischer[1] and Kristof Schröder[2]

Draft: June 2026

Abstract. The Method of Alternating Projections (MAP) is the de facto standard algorithm for estimating high-dimensional fixed-effect regressions. Although MAP is often fast, it can converge slowly when the fixed-effect structure is poorly connected, as in matched employer-employee panels with low worker mobility between firms. We relate MAP's convergence to the connectivity of the weighted bipartite graph formed by the levels of a pair of fixed-effect dimensions and propose a novel graph-preconditioned Krylov solver. Our preconditioner is constructed from sparse approximate Cholesky factorizations of the weighted bipartite graph Laplacian associated with each pair of fixed-effect dimensions. We show that graph preconditioning can substantially improve convergence on poorly connected fixed-effect graphs but that its setup time can dominate total runtime on well-connected graphs. On a near-nested design with 10 million observations, factor-pair LSMR completes in 4.63 seconds, compared with 62.2 seconds for the fastest MAP implementation.

Keywords: high-dimensional fixed effects; alternating projections; preconditioning; matched employer-employee data; computational econometrics

JEL codes: C55; C63; C81; C87; J31

Paper

github.com/py-econometrics/within-paper

Solver

github.com/py-econometrics/within

“Converges and runs very fast!” =)