Complementing Stata with Python: Optimization and Self-Differentiation in Point-Mass Models

15th Spain Stata Conference
Seville, May 22, 2025

A. Morales-Kirioukhina[1], E. Congregado[1], D. Troncoso-Ponce[2]

[1]Department of Economics, University of Huelva and CCTH
[2]Department of Economic Analysis and Political Economy, University of Seville

Motivation

- **Stata has evolved**: since version 16 (2019), it supports direct integration with Python.
- This allows running Python code from within Stata, using packages like `pandas`, `numpy`, or `scikit-learn`.
- From Stata 17+, users can also control Stata from Python using the `pystata` package.
- This bidirectional interoperability expands possibilities for advanced data science, automation, and reproducibility.
- **But**: despite this progress, Stata still faces limitations in flexibility, automation, and integration depth—particularly when customizing advanced workflows or scaling up computations.

The case study: Duration models in Stata

- Estimation of discrete-time duration models with unobserved heterogeneity.
- `hshaz` (Jenkins, 2004): implements proportional hazard model with mass-point heterogeneity à la Heckman-Singer (1984).
- `hshaz2` (Troncoso-Ponce, 2017): same model, but using algebraic gradient and Hessian (Stata's `d2` method) $\Rightarrow$ huge speed-up.
- Example: youth unemployment data from Spanish Social Security (CSWH), 2000–2013.
- Sample size: 568,042 observations (unispell), $>$1.5 million (multispell).

Model specification

- Discrete-time hazard model with time-varying covariates and unobserved heterogeneity:

$$h(t|x,\eta) = 1 - \exp(-\exp(\lambda(t) + x\beta + \eta))$$

- Heterogeneity modeled as a finite mixture: $\eta \in \{\eta_1, \eta_2, ..., \eta_P\}$.
- Estimated via maximum likelihood. The likelihood contribution of individual i:

$$L_i = \sum_{j=1}^{P} \pi_j \prod_{t=1}^{T_i} h_{ijt}^{y_{it}} (1 - h_{ijt})^{1-y_{it}}$$

- `hshaz2` allows estimation with up to 5 support points and multispell structures.
- Estimation with 2 mass-points:

$$\text{Type 1: } \eta_1 = 0, \quad \text{Type 2: } \eta_2 \text{ estimated}, \quad \pi_1 + \pi_2 = 1$$

`hshaz` vs. `hshaz2`: Analytical Efficiency

- `hshaz2` (Troncoso-Ponce, 2017) rewrites Jenkins' `hshaz` using explicit algebraic derivatives:
  - Gradient and Hessian coded analytically using Stata's `d2` method.
  - Leads to major gains in computation time.
- Code size comparison (effective lines):
  - `hshaz`: 735 lines; `hshaz_ll`: 267 lines.
  - `hshaz2`: 1,957 lines; `hshaz2_ll`: ∼25,900 lines / ∼1.35M characters.

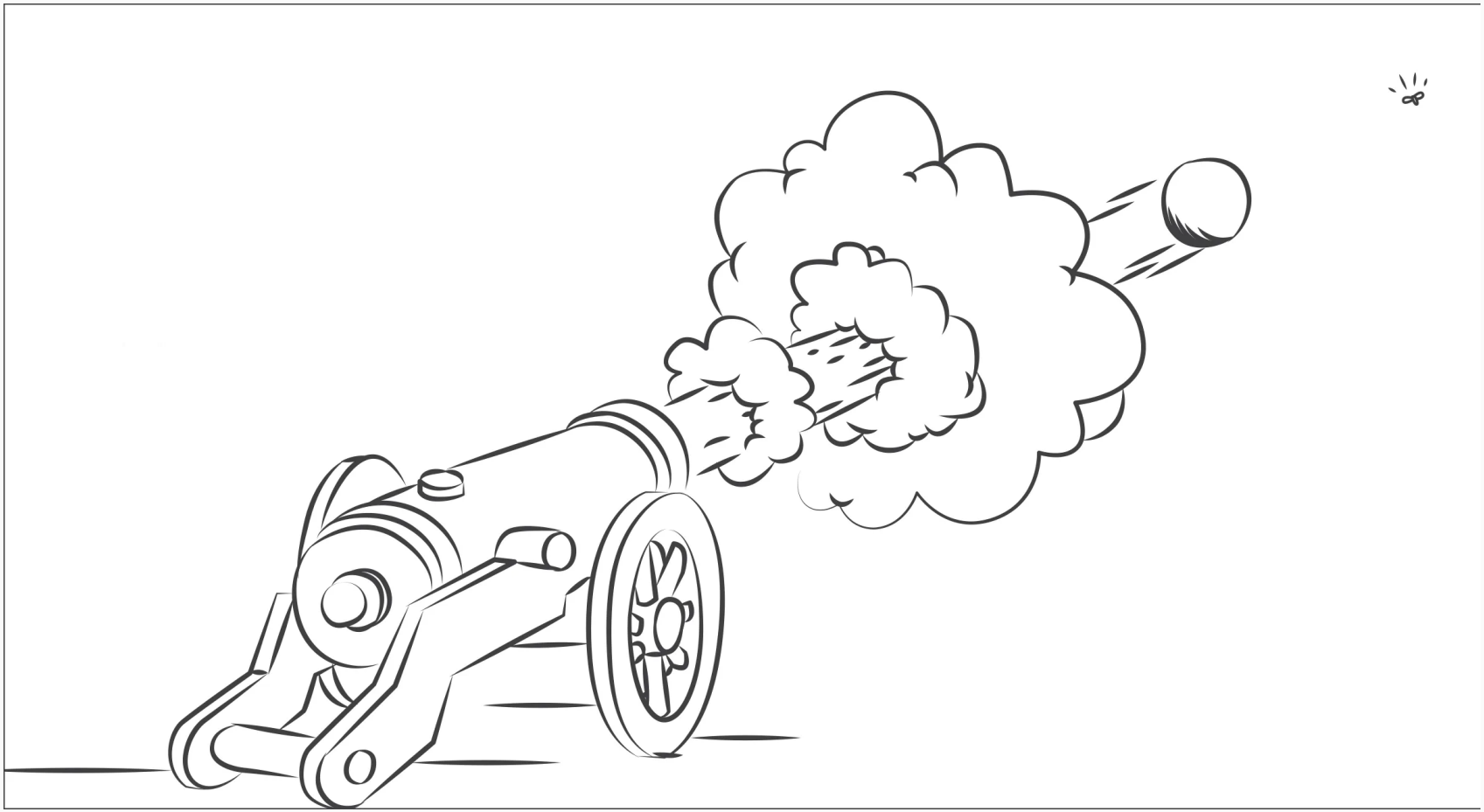
Speed gains (unispell, 568,042 obs)

Method	Running time (hh:mm:ss)
`hshaz` — Two mass-points	0:22:20
`hshaz` — Three mass-points	0:43:08
`hshaz2` — Two mass-points	0:00:42
`hshaz2` — Three mass-points	0:01:13

From Stata to Python – Why and how?

- What lies beyond Stata? Time to explore other universes.
- **Challenge 1:** Replicate `hshaz2` results in Python.
- **Challenge 2:** Beat Stata in speed and flexibility.
- First attempts with `numpy` and `scipy`: failed to converge or extremely slow.
- `statsmodels` and other R-inspired libraries: better, but inaccurate and inefficient.
- Stata proved both faster and more precise — hard to beat C-compiled routines.
- Needed a different angle: What does cutting-edge computation look like today?

“Using a sledgehammer to crack a nut”



Yes... but what if that nut is multidimensional, noisy, and moving?

JAX: Modern Optimization Beyond Deep Learning

- **Epiphany:** Tensor libraries are still calculus libraries.
- If they compute gradients for deep learning, they can optimize any differentiable function.
- Auto-differentiation with `JAX` eliminates manual derivative coding.
- Optimization with `scipy.optimize`: BFGS, L-BFGS-B, trust-constr.
- Achieved execution time: $\sim$ 30 seconds.
- Greater flexibility for future model extensions.
- Final twist: **could this be called from Stata?**

Python implementation

- Separate programs for each number of mass points, and for each optimization method: BFGS, L-BFGS-B, trust-constr.
- Also includes a generic version where number of mass points is an input argument.
- Code size: $\sim$100 lines per case (excluding comments).
- Fully modular, flexible, and replicable.
- Allows easy benchmarking across solvers and configurations.

Calling Python from Stata

- Python programs were executed via `.do files` using Stata's `python` integration.
- Requires only: `jax`, `scipy`, `statsmodels`, `pandas`.
- Execution from Stata was even faster than from the terminal.
- Full integration is feasible and efficient.

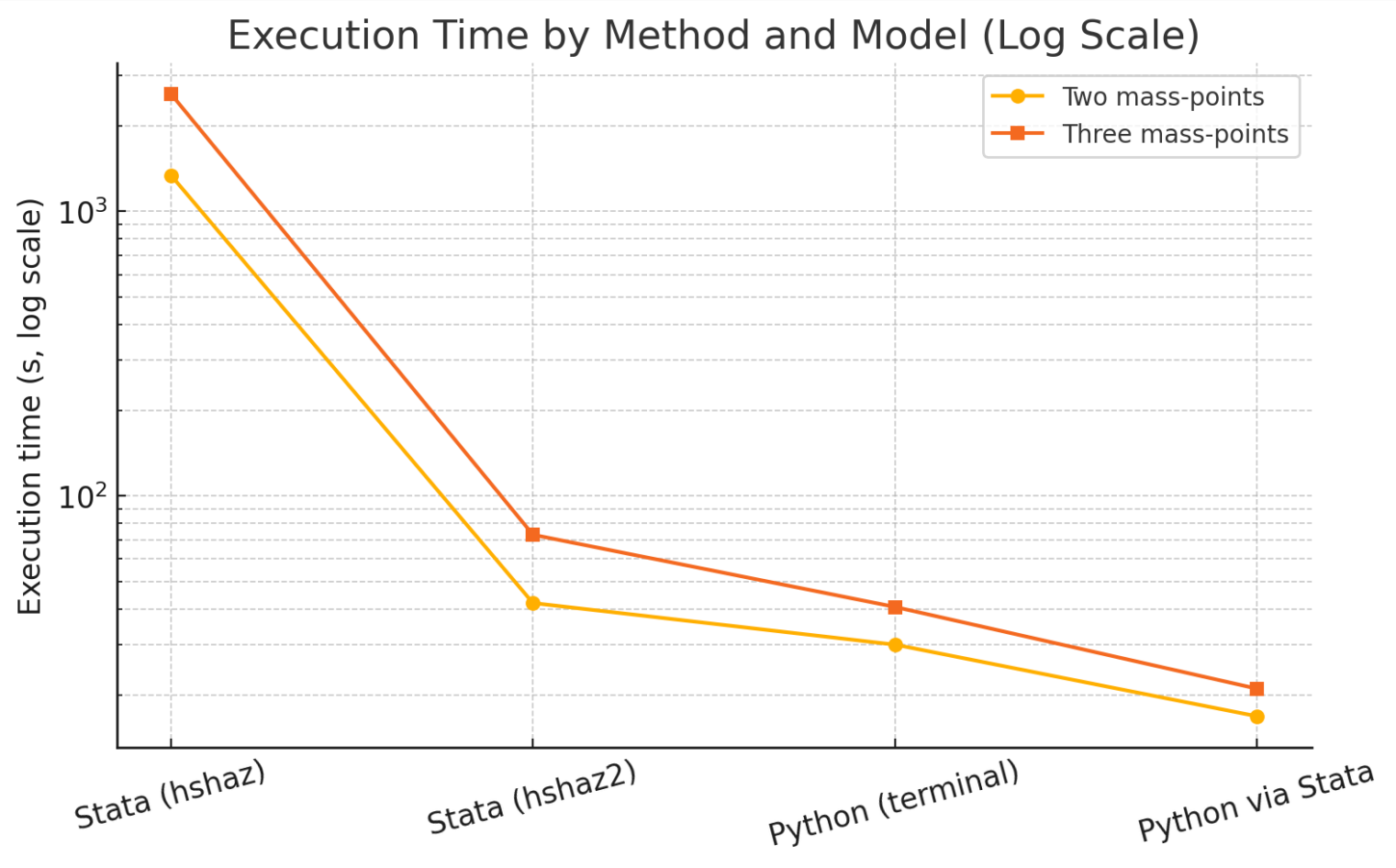
Stata is no longer an island.
The bridge to Python works — and it's fast.

Speed comparison (unispell, 568,042 obs)

Model	Execution method	Time
Two mass-points	Stata (hshaz)	22 min 20 s
	Stata (hshaz2)	42 s
	Python (terminal)	30.0 s
	Python via Stata	16.8 s
Three mass-points	Stata (hshaz)	43 min 08 s
	Stata (hshaz2)	1 min 13 s
	Python (terminal)	40.7 s
	Python via Stata	21.0 s

Execution times rounded to seconds where applicable.

Execution Time by Method and Model



Log scale reveals massive gains in performance.

Conclusions

- **Significant reduction in computation time** — from over 40 minutes to under 30 seconds.
- **Much shorter code** — ∼100 lines per version (no manual derivatives).
- **No need for Hessians** — auto-differentiation handles everything.
- **Full Stata–Python integration** — seamless execution via `python` in `.do` files.
- **Cutting-edge tools** — vectorized libraries and robust optimizers from operations research.
  - JAX, `scipy.optimize`, and multiple solver choices (e.g., BFGS, trust-constr).
  - Flexible backend: suitable for complex, nonlinear models.

Efficient, transparent, and future-proof estimation.

Thank You

Thank you for your attention!

Questions are welcome.

A. Morales-Kirioukhina — alejandro.morales@dege.uhu.es