**Modern dynamic programming**

For scientific computing, Matlab and Julia are both productive and fast (vectorization + just-in-time compiler)… We can get there from Python with numpy + numba + jax (which also allows GPU computing)! Note that numerical code is not as clean in Python as in Julia, but there are plenty of pros of course. What about econometrics libraries? For now, should rather go to R/STATA.

What about current trends?

* Parallelization: CPU frequency growth is slowing, hence multi-core processors + GPU/TPUs are becoming more important for pieces of code with compute-intensive functions
* Distributed computing…

NB: Possible to use Google Colab instead of local Python setup, especially for GPU

**History lesson**

Big data = information overload compared to available intellectual and technical infrastructure to process it (hence the relevance of such notion across history)

Also on the theory side…

Simulations in between… Mostly useful to know how long the long run is (once parameters are estimated)? Hence, resulting computational standards… Epistemological resistances and institutional contexts also matter, not just computational efficiency

Cf. also debates about structural vs. reduced-form, etc.

Questions about lack of criteria to assess models, also software, etc. (on ABM for instance)

**Performance tricks for Dynamic Programming**

Check part 2 on optimal stopping (precursor to dynamic programming, with only 2 choices, cf. Bellman equations)

Recursive preferences are a more fancy form than (linear) Markov chains

Value function iteration (VFI) is the basic way of solving Bellman equations, from a Bellman operator to a v-greedy policy.

Alternatively, let us consider an “abstract” dynamic program: feasible correspondence defines feasible state-action pairs, and set of feasible policies. A recursive decision process (RDP) consists of candidate value functions and a (valid) value aggregator. Standard Bellman structure works, but what about state-dependent discounting, Epstein-Zin time preferences, risk-sensitive preferences, ambiguity aversion, shortest path problems, etc. ?

Lifetime value is just defined as a fixed point of the abstract dynamic problem: this is a sigma-value function. Conveniently, this will fall under Newton lemma with a policy (Bellman) operator?

See contraction mapping (from value to value in graph)… Upper envelope is the Bellman operator from taking the maximum across policy operators…

See Howard policy iteration as an alternative to VFI algorithm: we take a smaller number of larger steps, making it easier to parallelize. Also note that it is always globally convergent to the exact optimum policy, under finite sets (not necessarily the case for VFI, see conditions on Newton algorithm)!

Optimistic policy iteration is basically a convex combination of VFI and Howard policy iteration with respect to step size (from m=1 to inf).

NB: also note that, due to overhead, we do not want to parallelize small tasks, only big ones: meaning only on the outer loop, even if very small (except for extreme binary cases for instance)!

A final note on JAX: everything needs to be vectorized and cannot be written in loops! A quick fix is to write in loops first, and then optimize the code. GPU also makes it super faster! If we run on Google Colab, parallel computing is fused in one big task, so no server connection issues!

JAX is way faster than Numba! But Numba works with loops (and GPU could be used with cuda low-level interface)…