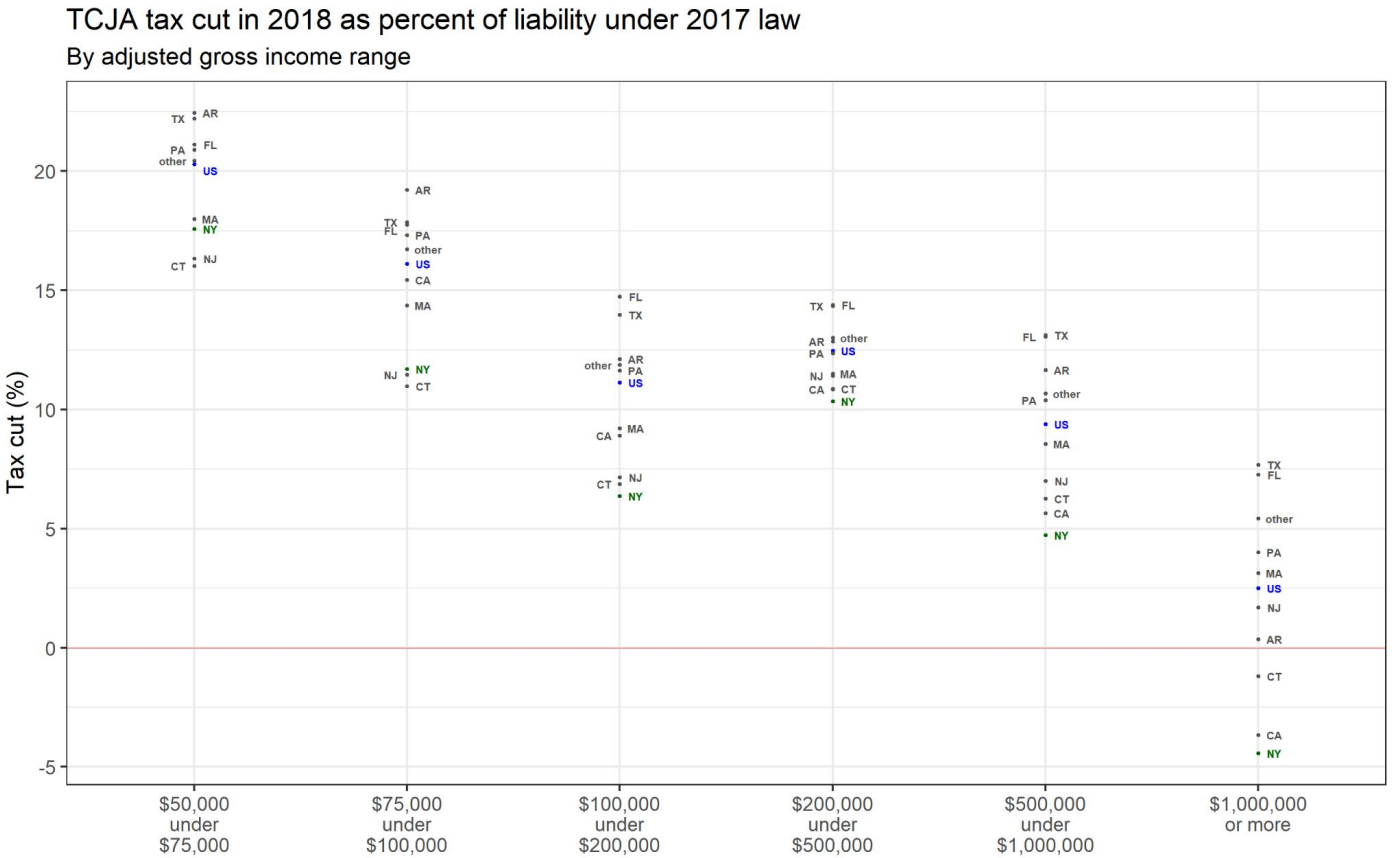
Constructing 50-state weights for the PUF that sum to national weights

-- PSL Demo Day --

Don Boyd
donboyd5@gmail.com

July 12, 2021

Motivation: Analyze impact of federal tax reforms by state



Lowest income group had very large percentage reductions and is excluded to improve readability.
Note: states below the zero line had tax increases.

Goals

Construct state weights for Tax-Calculator-ready Public Use File (puf.csv) so that:

1. Weighted summaries by state are consistent with published (or forecasted) summary data for the state, and
2. State weights on each record sum to the record's national weight, ensuring that sum of state results = national results

This talk: heavy emphasis on reproducibility, thus fairly technical. Will zoom through first 4 steps (15 mins) so we can focus on #5, the creation of state weights (30 mins).

Some slides are just for reference. I will try to cover the gist of them not the details.

In interest of speed, please hold questions until end, other than questions needed to make things clearer.

Python repositories and modules

- weighting repo
  - *microweight.py* module has *Problem* class, with two methods:
    - *reweight* -- for modifying existing weights on a file (similar to what TaxData does)
    - *geoweight* -- for finding geographic (e.g., state) weights that add to national weights
  - *make_test_problems.py* module constructs random data of arbitrary size and sparsity for any number of households and states plus targets for each state
- puf_analysis repo -- functions to work with puf and create state weights, drawing on *microweight*. *create_state_weights.py* does the work covered in this talk.

I use a branch named *psl-demo* in each repo.

All very rough: deadwood functions; incomplete and out of date documentation. WIP.

For which year(s) should we construct state weights?

1. Best data for targeting are IRS SOI Historical Table 2 (HT2), available annually from early 2000s (possibly earlier) through tax year 2018.
2. Information by state for each of 10 AGI ranges:
   a. Income and deduction amounts; some credit info
   b. # of returns with nonzero amounts for each item
   c. # of married, single returns
3. I chose 2017 over 2018 because TCJA changed meaning of HT2 dramatically in 2018. Many more people take standard deduction, no longer much data on itemized deductions. Crucial for analysis of SALT.
4. States' relative economic growth vary over time. Would like to forecast each state's share in later years, but currently keep 2017 shares going forward.

Project tasks

1. Download & parse IRS summary data for national & state targets (not shown)
2. Preliminary preparation of national data file
3. Prepare state targets
4. Revise national weights to be more-consistent with state targets
5. Apportion each record's national weight to the states
6. Extrapolate to later years and analyze reforms (TBD)

1. Download & parse IRS summary data (not shown)

GOAL: Obtain IRS data suitable for targets and structure in a computer-friendly and documented manner.

1. National: A lot of tedious work:
    a. parse IRS Excel files in self-documenting way
    b. reconcile IRS summary-data concepts to PUF concepts
    c. establish least-common-denominator AGI ranges
    d. convert to consistent units as needed
    e. programs:
        i. puf_download_national_target_files.py
        ii. puf_ONETIME_create_puf_irs_mappings_and_targets.py
    f. result: targets2017_possible.csv
2. States: A lot of tedious work:
    a. parse IRS HT2 csv file and associated documentation
    b. establish linkages between PUF concepts and HT2 concepts
    c. align HT2 variable names with PUF variable names
    d. convert to consistent units as needed
    e. calculate each state's share of national total for each HT2 concept
    f. programs:
        i. puf_download_state_HT2target_files.py
        ii. puf_ht2_shares.py
    g. result: ht2_shares.csv

2. Preliminary preparation of national data file

GOAL: 2017 national file, with tax filers only (our targets are for filers), with 2017 AGI, weighted to look like national IRS data, with just those variables needed for targeting

1. Advance latest puf.csv to 2017 using taxdata growfactors, weights, pufratios.
    a. Calculate 2017 law AGI
    b. Determine AGI bin(s) for each record, corresponding to bins used in target data
    c. Determine filer status under 2017 law; detailed IRS targeting data are for filers only
2. Get and reformat previously created targets data (ptargets)
3. Create subset needed for analysis (pufsub) - filers only, selected variables
4. Examine how close initial data are to IRS targets (baseline_national.txt)
5. Reweight *filers* in national file to more closely correspond to IRS national data for 2017.
    a. ~43 variables (amount, number positive, number negative) for each of 18 AGI ranges
    b. Exclude a few difficult variables in the lowest income ranges where PUF seems inconsistent with IRS aggregate data (deductions for medical expenses, contributions)
    c. 756 targets total for the nation as a whole
    d. ipopt used for reweighting, penalizing large changes in weights
    e. We will reweight 2 more times, in later steps.
6. Examine quality (reweighted_national.txt)

2. Prepare national data file

<<--- Run code --->>

- Examine pc.irspuf_target_map
- Examine QC report: reweighted_national.txt (c02500_nnz)

3. Prepare state target data

GOAL: Create targets for each state, for each variable of interest, for each AGI range, that add to PUF totals for the nation, for each variable and AGI range.

1. Define states to target (compstates)
2. Calculate state targets for states of interest (ht2targets)
    a. Collapse previously computed shares to states of interest and all other, by AGI range (10 ranges, not the 18 IRS ranges)
    b. Multiply national puf sum for each variable of interest by state shares; for example:
        i. if a state was 5.3% of the HT2 data's U.S. total for AGI in income range #7, then
        ii. calculate the state's AGI target for income range #7 as 5.3% of the weighted puf total of AGI in that income range
        iii. even if the puf national total is slightly different from the HT2 national total
3. Examine quality:
    a. Compare national HT2 totals to PUF totals (collapsed by HT2 income ranges) (ht2vspuf_targets_national.txt)
    b. Look for oddball state shares of HT2 national totals to get clues about which targets may be hard to hit (ht2target_analysis.txt)
4. Prepare a reformatted targets file (ht2wide) and compute potential drops (currently deadwood)

3. Prepare state target data

<<--- Run code --->>

- Examine report: ht2vspuf_targets_national.txt
- Examine: ht2target_analysis.txt
  (see other, stub 1, salaries and wages - #345, 346)

4. Prepare national PUF data for state targeting

GOAL: new national weights that hit national targets and are highly compatible with state targets.

1. Define which subset of the possible state targets we will target (22 of the 43 nationally targeted variables). Returns by marital status; most important income and deduction components (especially SALT). NOT taxable income or tax. (targvars)
2. Construct weights for each state in isolation that hit state targets (allweights2017_geo_unrestricted.csv):
    a. Do this for the 10 HT2 AGI ranges (more highly aggregated than the 18 IRS ranges). Run ipopt 510 times (51 areas x 10 AGI ranges) -- about 11 mins.
    b. Calculate sums of state weights & use them as tentative new national weights.
    c. These sums won't quite equal the existing national weight for each record and weighted national totals won't quite match target national totals for the 18 national AGI ranges (although they will be almost dead-on for the 10 HT2 AGI ranges).
3. Examine quality of national weights that are sums of unrestricted state weights (geosums_national.txt)
4. Examine state target results using the unrestricted state weights (state_comparison_wunrestricted.txt)
5. Revise (reweight) these tentative weights to get national weights that hit national targets. Do this for the 18 IRS AGI ranges.
6. Examine quality of these new national weights (state_comparison_wunrestricted.txt).
7. Update state-stub targets to reflect new slightly-revised pufsums, examine quality

TBD: Would we be truly worse off if we skipped this step?

4. Prepare national PUF data for state targeting

<<--- Run code --->>

- Examine geosums_reweighted_national.txt:
  - After (a) calculating unrestricted state weights, (b) using sums of these state weights as new national weights, and (c) reweighting the file to hit IRS targets for 18 income ranges,
  - Are the national results still close to national targets? (If so, good to go.)

5. Apportion each record's national weight to the 50 states

GOAL: weights for each record for each state that (a) sum to the record's national weight, and (b) produce state totals consistent with state targets.

1. Loop through income stubs, solve for state weights, and save results
2. Assemble full set of state weights from stub results
3. Examine quality

First, review of methods.

Review of methods

- Disclaimers
- Goals
- Possible approaches
- Details for the model approach

Technical goals for constructing state weights

- Correctness
- Robustness
  - rarely fails when problem is solvable
  - ideally works without fiddling with options -- set it and forget it
  - suboptimal results are good results, rather than total failures or incorrect results
- Fast
- Moderate memory usage, suitable for PCs

	Basic idea	Problem size**	Comments	Paper(s)
Brute force	Constrained optimization: choose weights that hit targets while minimizing differences from naive state weights*	40k recs x 50 states = solve for 2m weights 40k adding-up constraints, plus 50 states x 20 each =1,000 targets, 41k constraints	Straightforward. Imposes no functional form on weights. Compute intensive. Why penalize diff from avg?	None exactly on point. Tanton useful.
Model	Build model to forecast weights. Estimate model parameters. Predict state weights from model.	solve for 50 states x 20 targets = 1,000 parameters (<< 2m, but complex) 40k fixed effects depend on parameters	Do we like the model? Numerical challenges. Can be compute- and memory-intensive. Focus of current project.	Khitatra-kun et al., Schirm et al.
Iterative	Start w/naive weights. Choose best weights ∀ state independently. Stay near prior best in search for new weights. Pro-rata adjust weights to sum to national weights. Repeat until stopping criteria met.	40k recs x 1 state at a time, 20 targets each = solve for 40k weights at a time. Repeat 50x.	Minimal memory requirements. Minimal numerical issues. If converges, can be fast. When iterations do not converge, solution can be horrible. Used for early analysis (motivation graph).	Randrian-asolo et al.

* Naive: each state's share of national weight equals HT2 average share (e.g., ~10% for CA)
** Illustrative. Assumes one income range with 40k records, 50 states, 20 targets

Microweight implements all 3 approaches, with variants

- Brute force:
  - *method* = '*direct-ipopt'* (method is an argument name, not a class method)
- Model:
  - *method = 'poisson-...'* (many variants) (draws substantially on Khitatrakun et al.)
- Iterative:
  - *method = 'qmatrix-...'* (many variants) (based on Randrianasolo et al. R code w/adjustments)

All 3 work well in the laboratory, but...

- Made-up data (*test_microweight.py*) allows testing for:
  - Correct implementation
  - Speed
  - Memory use
  - But not robustness
- Real data throw curve balls:
  - Some (many) Jacobians (will explain shortly) cannot be inverted
  - Or give numerical difficulties
  - Some HT2 targets are zero even though PUF data have nonzero values
  - PUF data incompatible with HT2 targets in other ways, etc.
- Must attend to numerical issues and edge cases.
- Resulting state weights from different methods appear highly correlated.
- Based on testing and intuition, I use the model approach in this project.

The model approach (see Khitatrakun…)

- Treat weight for each household-state combination as coming from a poisson model with record characteristics (wages, business income, etc.) as predictors
- Makes intuitive sense - poisson often used for count data
- The model forecasts each state weight for each record using:
  - A set of state-specific parameters (1 parameter per target), 50 sets of these parameters (denoted “beta”), plus
  - A fixed effect for each household (the same regardless of which state we are forecasting for) (denoted “delta”),
  - Constrained (implicitly) so that a record’s state weights sum to its national weight

Symbolically(1): (see Khitatrakun,...) -- for reference

h: indexes households (records) (e.g., 1...40k, for one income range)
s: indexes states (e.g., 1...51)
k: indexes targets (within a state) (e.g., 1...30)
x: matrix of data, h rows, k columns (given, fixed; it is the PUF data for the problem at hand; columns might be wages, property tax deductions, ...)
$\mathbf{x}^h$: one row in this matrix (for one household; given, fixed)
X_{ks}: target for variable k for state s (given, fixed)
$\boldsymbol{\beta}$□: beta, vector of k parameters for state s (each state has its own set of parameters) - we want to solve for these
$\boldsymbol{\delta}^h$: delta for household h, its fixed effect (one delta for each household) these depend on the betas and are part of our solution
W^h: national weight for household h (given, fixed)
: weight for household h in state s (our goal - predicted by the model)
w_s^h

Symbolically(2): (see Khitatrakun,...)

Poisson model predicts the weight for one household, *h*, for one state, *s*, as:

$$w_s^h = \exp\left(\underline{\beta}_s' \underline{x}^h + \delta^h\right)$$

Our constraints require (a) sum of state weights for a household = fixed national weight, and (b) sum of household weights multiplied by data values = targets (fixed):

$$\sum_s w_s^h = W^h$$

$$\sum_h w_s^h x_k^h = X_{ks}$$

Substituting RHS of the w_s^h expression into the first constraint gives:

$$\delta^h = ln\left(\frac{W^h}{\sum_s \exp\left(\underline{\beta}_s' \ \underline{x}^h\right)}\right)$$

Note that RHS of model (1st equation) gets large fast as exponent increases. Largest double-precision is ~1.8 x 10^{308}, or ~e^{709}. Risk overflow errors (inf) if exponents get large during search. Note also that deltas depend on betas, thus connecting states to each other.

What's the objective function in this approach?

- For a given tentative set of betas for a given income range (e.g., 50 states x 20 betas = 1,000 betas):
  - predict state weights
  - calculate target values using these weights
  - compute difference between each calculated target value and desired target value (e.g, 50 states x 20 targets = 1,000 differences)
  - convert these to percentage differences (equal weight to all targets, and easy to interpret)
- Two basic approaches to objective function:
  - try to drive all differences to zero, using optimization methods to find roots (betas) of a vector-valued function (e.g., Newton's method), often with stopping criteria based partly on the l2norm of the objective function (the square root of the sum of squared differences), or
  - construct a scalar-valued objective function to minimize wrt betas -- e.g., sum of squared differences, using nonlinear least-squares optimization methods (e.g., Levenberg-Marquardt)
- *Microweight* implements both approaches, with variants. Focus here on Newton's root-finding method.

When you’re searching for something...

- In which direction should you search? (step direction)
- How far should you go in that direction before taking stock and picking a direction for the next step? (step size)

This is a “line search” approach. (An alternative approach is to define the size of a region you are comfortable searching in - a step size - and then choose a direction to search within that region. This alternative is a “trust region” approach.)

We look, iteratively, for the best betas - those that minimize differences from targets. In each iteration we look for better betas, deciding which direction to search relative to where we are now, and how far to look.

Calculating the step

- How much does the objective function change if you change one of the parameters (betas)? Generally look to the derivatives of the objective function.
- The Jacobian matrix (J) is critical: where f is the vector-valued objective function (differences from targets), and the x's are the betas in our model. Square, about 1,000 x 1000 (50 states, 20 targets) for us (1m elements):

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Newton’s method (aka Newton-Raphson)

- A “step” is the amount by which we change the betas from one iteration to the next. If we have 1,000 betas (50 states, 20 targets), then we need to adjust each beta between one iteration and the next - the step is a vector with 1,000 values.
- Newton’s method (first variant published in 1669!): A Newton step - the adjustment to the betas from the last iteration - is the difference from targets multiplied by the Jacobian inverse, all evaluated at these betas. In other words:

  step = new betas - current betas = $-\mathbf{J}^{-1}$ x **diffs** (evaluated at current betas)

- We could calculate **J** and solve for $-\mathbf{J}^{-1}$. However, multiplying by **J** gives:

  J multiplied by step = diffs (evaluated at betas)

- This is just a system of linear equations:

  $\mathbf{Ax} = \mathbf{b}$

  where **A=J**, **b=diffs** (our objective function), both evaluated at the current betas, and **x** is the unknown we want to solve for -- the next step, i.e., the amount by which we want to change the betas in this iteration.

- This opens up the dozens (at least) of methods for solving linear systems, including methods that do not require us to calculate **A** (i.e., **J**), which can be time consuming, memory intensive, and numerically unstable, but rather just need a function that can calculate the **Ax** product (a Jacobian-vector product). This can save time and memory and reduce numerical instabilities.
- In one variant I use the “lgmres” iterative method from scipy. In another, I calc Jacobian directly but calc its inverse iteratively.

Damping the step size

- Newton's method works best when close to the solution (and when certain other conditions are met). Steps can be too large or wrong when far from solution.
- Often people use a "damped" Newton method that multiplies the step by a factor in [0, 1], that changes as we approach the solution. This allows smaller steps; it can avoid stepping too far, but may converge more slowly.
- Methods for searching for the damping factor range from simple to complex.
- Popular wisdom seems to be that it doesn't make sense to spend a lot of time looking for the damping factor: better to be fast and good enough, and spend time instead searching for the next direction.
- The methods I tried include a commonly proposed approach of halving the factor until an adequate one is found, and a more-sophisticated approach known as a Wolfe linesearch, which is available in scipy. Neither worked well, at least as I implemented them.
- I ended up coding a search for the damping factor in [-1, 1] using scipy.optimize.minimize_scalar, in each Newton iteration choosing the damping factor that minimizes the l2norm of the objective function, treating the search direction as fixed. I limit the number of step-size-search iterations (with an option the user can change; default=20). It is fast and works well but there must be a reason no one else uses it.

Implementation of Newton's method

- Scale the problem to make it easier to solve
- Algebraic simplification and manipulation of objective function to reduce risk of numeric overflow or roundoff error
- jax jit for speed
- jax autodifferentiation for Jacobian (jac)
- Alternative: jacobian vector product (jvp)
- Alternative: less-accurate but fast, robust, low-memory step using a Jacobian-Free Newton-Krylov (JFNK) method, borrowed from *nonlin.py* in scipy.optimize (krylov)
    - Originally from the 1931 paper, О численном решении уравнения, которым в технических вопросах определяются частоты малых колебаний материальных систем, by Aleksei Krylov, naval engineer, known for significant contributions to theory of movement of ships in shallow water
    - paper written when he was 68 years old, method (and variants) used increasingly over last 2 decades because of low memory usage and computational cost
- Solve for step using iterative methods rather than matrix inversion
- Search for good step size using optimization rather than traditional linesearch
- Remember the best result and retrieve it at end even if later steps worsen results (rare)
- Decision rules to allow automatic switching among chosen step methods (krylov, jac, jvp) if one method stalls - intended to create set-it-and-forget-it defaults

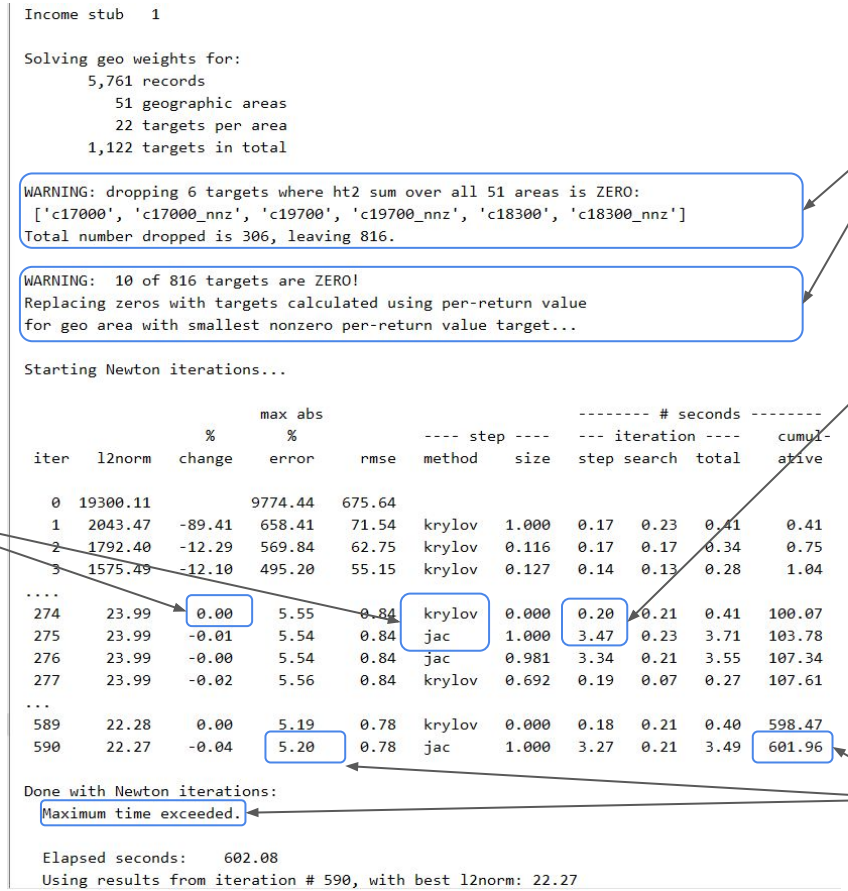
Stopping criteria

- All can be changed with options. Stop if any of the following occurs:
  - all absolute percentage differences from targets are below a threshold (default = 0.01%)
  - max iterations reached (default = 2,000)
  - max time reached (default = 20 mins)
  - no improvement in l2norm (square root of sum of squared percentage differences) for consecutive iterations using different methods; suppose methods are (krylov, jac, and jvp): if it tries krylov, then jac, then jvp, consecutively, and none improves the l2norm, it stops
- In general, we want the first stopping criterion. That usually occurs.

State weighting implementation

- What if the HT2 data say that a given variable in a given income range is zero in ALL states, yet we have PUF records in that range with values for the variable?
  - For example, HT2 says no state had contribution deductions in the “Under $1” AGI range, yet the PUF has records in this range with values.
  - No set of state weights is consistent with this. (If all state weights are zero, we’ll hit the target, but then state weights won’t sum to the nonzero national weight.)
  - Solution: Drop these targets. Occurs in the lowest and highest income ranges.
- What if HT2 data for a target for a specific variable in a specific state in a specific income range is exactly zero?
  - All optimization methods seem to have extreme difficulty with this.
  - Solution: Replace the zero target with a target based on the lowest nonzero target.
  - Example: suppose HT2 says MS AGI < $1 has zero interest income, but AL, the state with the lowest non-zero average, has $23 per-return average. Replace MS zero with: (# of MS returns in this range x $23).
  - Occurs in the lowest income range.
- Output for each AGI stub:
  - Log file (e.g., stub01_log.txt)
  - File of weights for each household and state (e.g., stub01_whs.csv)
  - Pickled file with betas (e.g., stub01_betaopt.pkl)

How to interpret solution output for an AGI stub



```
Income stub   1

Solving geo weights for:
       5,761 records
          51 geographic areas
          22 targets per area
       1,122 targets in total

WARNING: dropping 6 targets where ht2 sum over all 51 areas is ZERO:
 ['c17000', 'c17000_nnz', 'c19700', 'c19700_nnz', 'c18300', 'c18300_nnz']
Total number dropped is 306, leaving 816.

WARNING:  10 of 816 targets are ZERO!
Replacing zeros with targets calculated using per-return value
for geo area with smallest nonzero per-return value target...

Starting Newton iterations...

                          max abs                                  -------- # seconds --------
                    %       %                 ---- step ----    --- iteration ----    cumul-
 iter    l2norm   change   error     rmse    method    size    step search  total    ative

    0  19300.11           9774.44   675.64
    1   2043.47   -89.41   658.41    71.54   krylov    1.000   0.17   0.23   0.41     0.41
    2   1792.40   -12.29   569.84    62.75   krylov    0.116   0.17   0.17   0.34     0.75
    3   1575.49   -12.10   495.20    55.15   krylov    0.127   0.14   0.13   0.28     1.04
....
  274     23.99     0.00     5.55     0.84   krylov    0.000   0.20   0.21   0.41   100.07
  275     23.99    -0.01     5.54     0.84   jac       1.000   3.47   0.23   3.71   103.78
  276     23.99    -0.00     5.54     0.84   jac       0.981   3.34   0.21   3.55   107.34
  277     23.99    -0.02     5.56     0.84   krylov    0.692   0.19   0.07   0.27   107.61
...
  589     22.28     0.00     5.19     0.78   krylov    0.000   0.18   0.21   0.40   598.47
  590     22.27    -0.04     5.20     0.78   jac       1.000   3.27   0.21   3.49   601.96

Done with Newton iterations:
  Maximum time exceeded.

  Elapsed seconds:    602.08
  Using results from iteration # 590, with best l2norm: 22.27
```

Drop HT2 targets incompatible with PUF data.

Adjust implausible targets.

krylov steps are faster than jac but often less productive (latter not observed here)

krylov stalled, switch to jac

Note:
rmse = sqrt(l2norm^2 / ntargets)

For this stub, ntargets = 816.

timed out with max % error of 5.2% (limit had been set to 10 mins)

Comparison for federal AGI in New York, 2017

Table 2. Individual Income and Tax Data, by State and Size of Adjusted Gross Income, Tax Year 2017

[Money amounts are in thousands of dollars]

Item	All returns	Size of adjusted gross income									
		Under $1 [1]	$1 under $10,000	$10,000 under $25,000	$25,000 under $50,000	$50,000 under $75,000	$75,000 under $100,000	$100,000 under $200,000	$200,000 under $500,000	$500,000 under $1,000,000	$1,000,000 or more
Adjusted gross income (AGI) [5]	836,755,149	-23,906,390	6,998,061	34,317,550	78,562,853	81,527,861	73,482,730	179,003,932	129,588,367	56,532,599	220,647,586

stgroup	ht2_stub	ht2range	pufvar	ht2var	ht2	target	calcsum	d_ht2	pd_ht2	d_target	pd_target	column_description
NY	0	All income ranges	c00100	a00100	836,755,149,000	838,009,358,679	837,647,440,901	892,291,901	0.1%	-361,917,778	-0.0%	Adjusted gross income less deficit
NY	1	Under $1	c00100	a00100	-23,906,390,000	-23,981,014,664	-24,066,313,209	-159,923,209	0.7%	-85,298,545	0.4%	Adjusted gross income less deficit
NY	2	$1 under $10,000	c00100	a00100	6,998,061,000	7,033,932,476	7,099,657,358	101,596,358	1.5%	65,724,882	0.9%	Adjusted gross income less deficit
NY	3	$10,000 under $25,000	c00100	a00100	34,317,550,000	34,787,195,594	34,790,371,877	472,821,877	1.4%	3,176,282	0.0%	Adjusted gross income less deficit
NY	4	$25,000 under $50,000	c00100	a00100	78,562,853,000	78,376,559,046	78,376,238,290	-186,614,710	-0.2%	-320,756	-0.0%	Adjusted gross income less deficit
NY	5	$50,000 under $75,000	c00100	a00100	81,527,861,000	81,958,465,689	81,959,883,278	432,022,278	0.5%	1,417,588	0.0%	Adjusted gross income less deficit
NY	6	$75,000 under $100,000	c00100	a00100	73,482,730,000	73,771,540,195	73,773,339,156	290,609,156	0.4%	1,798,961	0.0%	Adjusted gross income less deficit
NY	7	$100,000 under $200,000	c00100	a00100	179,003,932,000	179,269,042,959	179,271,795,025	267,863,025	0.1%	2,752,066	0.0%	Adjusted gross income less deficit
NY	8	$200,000 under $500,000	c00100	a00100	129,588,367,000	128,977,527,948	128,979,756,198	-608,610,802	-0.5%	2,228,250	0.0%	Adjusted gross income less deficit
NY	9	$500,000 under $1,000,000	c00100	a00100	56,532,599,000	57,387,028,823	57,389,244,008	856,645,008	1.5%	2,215,186	0.0%	Adjusted gross income less deficit
NY	10	$1,000,000 or more	c00100	a00100	220,647,586,000	220,007,265,394	220,073,468,921	-574,117,079	-0.3%	66,203,527	0.0%	Adjusted gross income less deficit

The key end product

allweights2017_geo_restricted.csv

	0	1	2	3	4	5	6	7	8
0	**pid**	**ht2_stub**	**weight**	**geoweight_sum**	**AK**	**AL**	**AR**	**AZ**	**CA**
39	**41**	**2**	441.3777890652598	441.37778906526	0.5182226197375491	6.832988772475058	4.345255637650686	8.189407792729876	52.6413877190299
40	**42**	**2**	15.497248508460226	15.497248508460226	0.0278033022741181	0.1738283642118906	0.0998766392559046	0.3304869564382884	1.3899584460363958
41	**43**	**5**	125.7419283540358	125.74192835403576	0.2545771484907007	0.9632144752158788	0.6800423279200354	2.33550024076208	13.71404404565637
42	**44**	**6**	862.6882520556213	862.6882520556215	2.985556137678469	7.157175868408762	5.27474208400145	14.98330936696493	85.12104358195937
43	**45**	**3**	185.81117019311444	185.8111701931144	1.2175191032607815	1.5644966744830728	0.8778818570346031	3.463325564753067	27.568655919017697
44	**46**	**6**	37.19678297959838	37.19678297959838	0.1006512144576505	0.2454850307600301	0.1879900260928036	0.5536313582150533	3.130309669821314
45	**47**	**10**	0.7080001738363615	0.7080001738363618	1.9908713399852697e-15	4.169100573678492e-06	0.0002880961463226	0.0014838131854459	0.3986314538866398
46	**48**	**5**	1350.8414822106	1350.8414822106	2.202534294307934	16.02822712594241	10.82350450275628	25.46879444931012	101.37523154737154
47	**49**	**3**	883.6046515404622	883.6046515404619	2.501394559217774	11.1095066981965	7.457104303975691	19.52134927423439	119.73514647839288
48	**50**	**3**	1459.3631877902806	1459.3631877902812	3.339820697276898	20.767110289099843	12.818316829458	30.207192560722937	180.8193422956052
49	**51**	**3**	776.3441667929649	776.3441667929649	1.957552343757824	10.146709413891982	6.639307052599984	16.83962777807928	102.37139123984082
50	**52**	**3**	632.3441275039277	632.3441275039276	1.790101667452804	7.950423651372565	5.336613050398095	13.970286997691192	85.68743566416957
51	**53**	**2**	2724.141233041993	2724.141233041993	4.2336710369666495	46.829594447706576	28.68733449672272	48.36972184870976	256.29912186996046
52	**54**	**3**	5.5055630737037085	5.505563073703706	0.0065685779936136	0.0643198637248644	0.0414564062552635	0.1086193956318792	0.8383461913067494
53	**55**	**3**	1346.6113115159474	1346.6113115159476	3.166026239275571	16.869385194250164	11.364979149723675	29.92985002227368	182.7394252655661
54	**56**	**3**	820.5448397733517	820.5448397733517	1.7505454909749187	11.258288699374292	7.131084342716061	17.375185191163453	104.3795977529329
55	**57**	**3**	835.1770575465739	835.177057546574	1.7961265737233372	11.37102324176513	7.240711243292751	17.760426925866433	106.84120577727413
56	**58**	**3**	814.7562294714695	814.7562294714696	1.7416917975260269	11.157353517873052	7.076508098172805	17.27099876072038	103.78950255065644
57	**59**	**3**	862.4276843512159	862.427684351216	1.8696520778033912	11.651639368633656	7.458769815382795	18.41779697067692	110.9487116566246
58	**60**	**3**	813.5167960570961	813.516796057096	1.7320666496759607	11.183369452366096	7.074263658994795	17.20800545865426	103.33963212144316

5. Apportion each record’s national weight to the 50 states

<<--- Run code --->>

- Run stub 8
- Stub 4 check memory, cpu with jac then krylov
- Examine logfiles for the 10 stubs
- Examine final QC file: state_comparison_wrestricted.txt
  (see stub 2, other, salaries and wages)

One set of runs (51 areas, 22 targets, 10 income stubs)

	AGI label	# filer records	max abs error (%)	RMSE (%)	iterations	minutes
1	Under $1	5,761	6.00	0.74	2,000	11.8
2	$1 under $10,000	18,700	32.06	1.86	667	12.0
3	$10,000 under $25,000	34,354	0.07	0.01	325	12.0
4	$25,000 under $50,000	40,366	0.01	0.01	138	6.6
5	$50,000 under $75,000	25,091	0.01	0.00	125	3.3
6	$75,000 under $100,000	17,327	0.01	0.00	84	1.6
7	$100,000 under $200,000	28,545	0.01	0.00	31	1.0
8	$200,000 under $500,000	16,954	0.01	0.00	26	0.5
9	$500,000 under $1,000,000	12,081	0.01	0.00	31	0.4
10	$1,000,000 or more	28,582	5.57	0.30	358	12.0
	Total	227,761				61.3

Preliminary conclusions re state-weight creation

- Generally very good. Some large differences from targets:
  - may not concern us (e.g., the “other” geographic area), and
  - may be attributable to data incompatibilities (e.g., income stub 2).
- Untargeted variables can be off considerably (e.g., # head of household returns)
- The QC report table provides guidance about what kinds of policy analysis will be most trustworthy, and what kinds will be least trustworthy.

6. Extrapolate or adjust state weights to the future

- Currently assume state shares of record weights are constant in forecasts
- But obviously state economies grow at different rates:
  - Sometimes in short periods, e.g., Louisiana's devastating population and economic losses after Katrina
  - Sometimes in reasonably predictable ways over longer term -- e.g., fast pop growth in Utah, aging and slow growth or decline in Maine
- It makes sense, *with sufficient resources*, to forecast targets for returns, AGI, income components, etc., by state, *in a way that is consistent with TaxData* forecasts for the national economy (e.g, CBO).
- Or to directly forecast changes in shares (i.e., reduced form).
- But not today.

Next steps, loose ends, possible improvements

- Gain better understanding of data-target mismatches
- Priority weights for targets
- Nonfilers
- Targets relevant to family size and age structure (e.g., EITC, child credits)
- Forecast targets, allowing weight shares to shift over time
- Explore / improve:
  - Model approach - iterate through states, not simultaneous (TPC I think)
  - Can any scipy/jax/tensorflow solver solve the problems robustly?
  - Preconditioning
  - Stopping criteria
  - Starting point
  - Further speedups
- Lots of code cleanup and documentation

Other possible applications

- Spread state weights across substate geographic areas:
  - Congressional Districts
  - Counties

  (If suitable targeting data can be constructed.)
- Data for state tax microsimulation models

Bibliography

Fisher, Robin, and Emily Y Lin. “Re-Weighting to Produce State-Level Tax Microsimulation Estimates.” Technical Paper. United States Department of the Treasury, Office of Tax Analysis, June 2015. https://www.treasury.gov/resource-center/tax-policy/tax-analysis/Documents/TP-6.pdf.

Khitatrakun, Surachai, Gordon B T Mermin, and Norton Francis. “Incorporating State Analysis into the Tax Policy Center’s Microsimulation Model: Documentation and Methodology.” Working Paper, March 2016. https://www.taxpolicycenter.org/sites/default/files/alfresco/publication-pdfs/2000697-Incorporating-State-Analysis-into-the-TPCs-Microsimulation-Model.pdf.

Knoll, D.A., and D.E. Keyes. “Jacobian-Free Newton–Krylov Methods: A Survey of Approaches and Applications.” *Journal of Computational Physics* 193, no. 2 (January 2004): 357–97. https://doi.org/10.1016/j.jcp.2003.08.010.

Randrianasolo, Toky, and Yves Tillé. “Small Area Estimation by Splitting the Sampling Weights.” *Electronic Journal of Statistics* 7, no. 0 (2013): 1835–55. https://doi.org/10.1214/13-EJS827.

Schirm, Allen L, and Alan M. Zaslavsky. “Reweighting Households to Develop Microsimulation Estimates for States.” In *1997 Proceedings of the Section on Survey Research Methods*. Alexandria, VA, 1997. http://www.asasrms.org/Proceedings/papers/1997_051.pdf.

Schirm, Allen L, and Alan Zaslavsky. “Model-Based Microsimulation Estimates for States When State Programs Vary.” In *1998 Proceedings of American Statistical Association: Section on Survey Research Methods*, 6, 1998. http://www.asasrms.org/Proceedings/papers/1998_055.pdf.

Tanton, Robert. “A Review of Spatial Microsimulation Methods.” *International Journal of Microsimulation*, 2014, 4–25. https://microsimulation.org/IJM/V7_1/2-IJM_7_1_Tanton_.pdf.

Wächter, Andreas, and Lorenz T. Biegler. “On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming.” *Mathematical Programming* 106, no. 1 (2006): 25–57. http://www.springerlink.com/index/r31116mp70171220.pdf.