Simulating Reforms to Taxation of Social Security Benefits

From Paycheck to Payout: Wealth and Income in Retirement

Ben Ogorek
Presenting On Behalf of Max Ghenis, Nikhil Woodruff, and Pavel Makarchuk
PolicyEngine

118th Annual Conference on Taxation, 2025

November 7, 2025

Motivation: A Social Security Policy Reform

Universal 85% Inclusion

Current System (since 1984):

- 0%, 50%, or 85% of benefits taxable
- Depends on "provisional income" (AGI + half of SS benefits + tax-exempt interest)
- Thresholds: $25k/$32k (single/joint) and $34k/$44k

Proposed Reform:

- **85% of all benefits taxable**, regardless of income
- Eliminates threshold system entirely

Python Implementation in PolicyEngine

```
def tax_85_percent_ss():
    """Tax 85% of Social Security benefits for all recipients."""
    return {
        # Set combined income fraction to 1.0 (instead of 0.5)
        "gov.irs.social_security.taxability.combined_income_ss_fraction": {
            "2026-01-01.2100-12-31": 1.0
        },
        # Set all base thresholds to 0
        "gov.irs.social_security.taxability.threshold.base.main.JOINT": {
            "2026-01-01.2100-12-31": 0
        },
        "gov.irs.social_security.taxability.threshold.base.main.SINGLE": {
            "2026-01-01.2100-12-31": 0
        },
        # ... (all other filing statuses)

        # Set all adjusted base thresholds to 0
        "gov.irs.social_security.taxability.threshold.adjusted_base.main.JOINT": {
            "2026-01-01.2100-12-31": 0
        },
        "gov.irs.social_security.taxability.threshold.adjusted_base.main.SINGLE": {
            "2026-01-01.2100-12-31": 0
        },
        # ... (all other filing statuses)
    }
```

Explore the Code Yourself

PolicyEngine is Open Source

https://github.com/PolicyEngine/policyengine-us

/

**policyengine_us/
parameters/**

All tax code parameters in YAML
Tax rates, thresholds, credits,
deductions, elasticities

/

**policyengine_us/
variables/**

Income, tax, benefit calculations
The computational logic that
implements the tax code

/

**policyengine_us/
reforms/**

Policy reform definitions
Pre-built reforms you can use or
customize

Simulated Reform Revenue Impacts of Universal 85% Inclusion

1	reform_name	year	baseline_revenue	reform_revenue	revenue_impact
290	option2	2089	2378.84	2599.59	220.76
291	option2	2090	2342.49	2570.58	228.09
292	option2	2091	2306.42	2542.31	235.89
293	option2	2092	2270.5	2514.61	244.11
294	option2	2093	2237.92	2490.43	252.51
295	option2	2094	2200.47	2461.99	261.52
296	option2	2095	2163.72	2434.5	270.78
297	option2	2096	2130.31	2410.62	280.3
298	option2	2097	2096.87	2387.12	290.25
299	option2	2098	2061.18	2362.03	300.85
300	option2	2099	2030.24	2341.91	311.66
301	option2	2100	1999.04	2322.23	323.19

This particular (very much in beta) simulation run:

- Incorporates **labor supply elasticities** (dynamic behavioral response)
- Reflects **demographic changes** over 75-year horizon
- Could be extended to do distributional analysis with income, age, etc.

The Rest of the Presentation is about Showing How We Got Here

Behavioral Responses in PolicyEngine: Labor Supply Elasticities

Two Key Parameters in PolicyEngine:

Income Elasticity

- `gov.simulation.labor_supply_responses.elasticities.income`
- Default: 0 (no response)
- CBO estimate: -0.04

Substitution Elasticity

- `gov.simulation.labor_supply_responses.elasticities.substitution`
- Default: 0 (no response)
- CBO estimate: 0.27

How Elasticities Flow Through the Engine

The Complete Calculation Chain

1. **Elasticity parameters** (YAML files: base.yaml, by_position_and_decile.yaml)
↓
2. **Read and apply adjustments** (income_elasticity, substitution_elasticity variables)
↓
3. **Multiply by earnings/wage changes** (income_elasticity_lsr, substitution_elasticity_lsr)
↓
4. **Sum the two effects** (labor_supply_behavioral_response)
↓
5. **Allocate by employment share** (employment_income_behavioral_response)
↓
6. **Add to base income** (employment_income)

```
# employment_income.py
class employment_income(Variable):
    adds = [
        "employment_income_before_lsr",
        "employment_income_behavioral_response",  # <-- Contains the LSR
    ]
```

Extension: Age Heterogeneity of Labor Supply Elasticity

Empirical finding: Workers 65+ are significantly more responsive to economic incentives

Evidence from Literature:

Study	**Age**	$65+/<65$ **Ratio**
French (2005)	60 vs 40	3.0–3.25×
Blau & Shvydko (2011)	62–69	2.0–3.0×
Gustman & Steinmeier (2009)	65–69	2.0–2.5×

PolicyEngine Default (But Tunable) Implementation:

$$\varepsilon_{\text{age}\geq 65} = 2.0 \times \varepsilon_{\text{age}<65}$$

Age Multiplier Implementation

How PolicyEngine Applies the Age-Based Elasticity Multiplier in Code

Income Elasticity (income_elasticity.py):

```
age = person("age", period.this_year)
age_multiplier = where(
    age >= p.age_threshold,                 # <-- PARAMETER from YAML (65)
    p.age_multiplier_over_threshold,        # <-- PARAMETER from YAML (2.0)
    1.0,                                    # No multiplier for under threshold
)

return base_elasticity * age_multiplier
```

Substitution Elasticity (substitution_elasticity.py):

```
age = person("age", period.this_year)
age_multiplier = where(
    age >= p.age_threshold,                 # <-- PARAMETER from YAML (65)
    p.age_multiplier_over_threshold,        # <-- PARAMETER from YAML (2.0)
    1.0,                                    # No multiplier for under threshold
)

return base_elasticity * age_multiplier
```

Result: Same simple logic for both elasticities. Workers 65+ get 2× multiplier; workers under 65 get 1× (no change).

A PolicyEngine Test Case That Sets Elasticity Parameters

```yaml
# PolicyEngine US Parameter Settings (YAML)

# Income elasticity parameters
gov.simulation.labor_supply_responses.elasticities.income:
  all: -0.04                            # CBO central estimate
  age_multiplier_over_threshold: 2.0   # 65+ workers 2x more responsive
  age_threshold: 65                     # Age cutoff for multiplier

# Substitution elasticity parameters
gov.simulation.labor_supply_responses.elasticities.substitution:
  all: 0.27                             # CBO central estimate
  age_multiplier_over_threshold: 2.0   # 65+ workers 2x more responsive
  age_threshold: 65                     # Age cutoff for multiplier

# Result:
#   - Workers under 65: income_elasticity = -0.04,  substitution_elasticity = 0.27
#   - Workers 65+:      income_elasticity = -0.08,  substitution_elasticity = 0.54
```

Default: All elasticities = 0 (static analysis). Must explicitly enable for dynamic modeling.

References embedded in parameters: French (2005), CBO Working Papers 2012-12 & 2012-13

Source Data

Survey Dataset

- CPS ASEC (March Supplement)
- ~180k individuals
- Demographics + basic income

Administrative Dataset

- IRS Statistics of Income
- ~207k tax filers, 120k with demographics
- Actual Tax filings

Data Fusion Strategy

Use Quantile Regression Forests to impute conditional income distributions from IRS PUF onto CPS households

Data Fusion: Quantile Regression Forests

Problem: Simple mean imputation loses distributional information

Solution: Quantile Regression Forests (QRF)

1. Train QRF on matched administrative-survey data
2. Generate full conditional distributions for each household
3. Preserve complex demographic-income relationships

$$\hat{F}_{Y|X}(y|x) = \sum_{i=1}^{n} w_i(x) \cdot \mathbf{1}\{Y_i \leq y\}$$

Where $w_i(x)$ are forest-derived weights based on similarity

Why We Want to Impute Income (Sometimes)

CPS Issue: Rank Proximity Swapping (RPS)

Post-2011: Income values swapped among similar-ranked households

$$(\mathbf{I}_j^{\text{CPS}}, \mathbf{D}_j) = (I_{\pi(j)}^{\text{true}}, \mathbf{D}_j^{\text{true}})$$

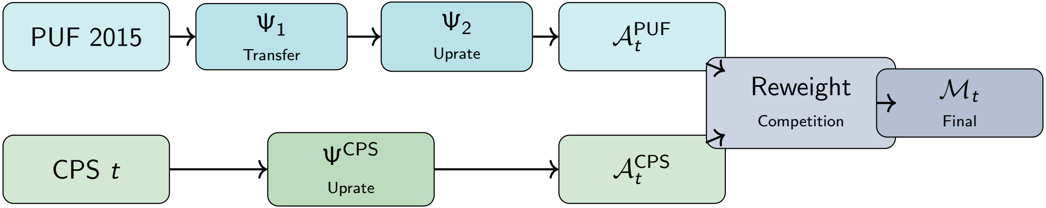
Impact: Destroys joint distribution

$$\text{Cov}(I^{\text{CPS}}, X^{\text{CPS}}) \neq \text{Cov}(I^{\text{true}}, X^{\text{true}})$$

Why This Matters for Tax Microsimulation

- 65-year-old household gets 35-year-old's income
- Tax function is nonlinear: $T = \mathcal{T}(\mathbf{I}, \mathbf{D})$
- Wrong demographics $\rightarrow$ biased tax calculations
- Upper tail especially affected

Dual-Source Data Fusion Pipeline



PUF Track: Admin data foundation

- Ψ_1: Transfer CPS patterns for missing variables *while preserving PUF income*
- Ψ_2: Uprate 2015 $\rightarrow t$ via uprating factors

CPS Track: Ψ^{CPS} Uprate 2023 $\rightarrow t$ via uprating factors

Reweighting Competition:

- PUF dominates upper tail (no RPS contamination)
- CPS contributes rich demographics
- Optimal weights via constrained optimization

$$\mathcal{M}_t = \text{Reweight}(\mathcal{A}_t^{\text{PUF}} \cup \mathcal{A}_t^{\text{CPS}})$$

Dual-Source Competitive Reweighting

Final Dataset Construction: $\mathcal{M}_t = \text{Reweight}(\mathcal{A}_t^{\text{PUF}} \cup \mathcal{A}_t^{\text{CPS}})$

Use PyTorch to Approximately Solve:

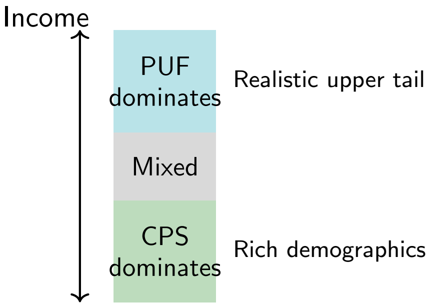
$$\mathbf{t} = \mathbf{X}\mathbf{w}$$

- $\mathbf{t}$: Calibration targets
- $\mathbf{X}$: Household characteristics matrix
- $\mathbf{w}$: Household weights

Calibration Target Categories:

- IRS SOI statistics
- Benefit program participation rates
- Demographic distributions

Competition Outcome:



Projecting to 2100 requires modeling both economic and demographic evolution

Economic Evolution

- Wage growth
- Inflation rates
- Population growth (for uprating of weights)

Source: CBO or SSA Trustees projections

Demographic Transformation

- Baby boom aging
- Declining birth rates
- Increasing longevity

Source: SSA Trustees Report

Working age	57.5% $\rightarrow$ 53.2%
Elderly (65+)	18.6% $\rightarrow$ 31.0%

$\longrightarrow$ **Fiscal pressure**

After the Economic Evolution Based on Factors: Reweighting (Again!)

We'll use a more traditional calibration that minimizes distance to the "original weights" (our reweighted weights from before) while satisfying the following auxiliary constraints:

- **Age-specific targets**: Single-year ages 0-85+ through 2100
- **SS benefit totals**: Match SSA Trustees Report Table VI.G9

GREG is the **Generalized Regression Estimator** calibrator.

- Raking could have been used, if it wasn't for the continuous SS Benefit variable.
- Now we want our calibrated weights to not move too far, so we use a traditional calibration procedure that
- minimizes a distance from the new weights to the original weights subject to the auxiliary constraints.

Thank You!

Ben.Ogorek@PolicyEngine.org
Max@PolicyEngine.org
Nikhil@PolicyEngine.org
Pavel@PolicyEngine.org

Resources:

`https://policyengine.org`