

Introduction  
oooo

Mixed Logit  
oooooooooooo

BLP  
oooooooooooo

Integrated Model  
ooooo

Conclusion  
oooo

# Understanding Demand Estimation Methods

Presenter: Liyu Zhao

February 10, 2026

1 Introduction

2 Mixed Logit

3 BLP

4 Integrated Model

5 Conclusion

# Why Demand Estimation?

Structural demand estimation is the prerequisite for any structural analysis of market power and pricing strategies.

- Calculate precise price elasticities: addressing price endogeneity ( $p$  correlated with unobserved  $\xi$ ). It is the starting point for any other analysis.
- Recover supply-side primitives: inverting the FOC allows us to recover marginal costs and markups without observing real supply-side data.
- Counterfactual analysis: merger simulation, introduction of new goods, or taxes.
- Basis for other topics: vertical relations, network effects, consumer search, and many others...

# Why Mixed Logit & BLP?

Standard Logit is insufficient due to the IIA (Independence of Irrelevant Alternatives) property.

- **The IIA Restriction:**
  - Logit implies proportional substitution (Red Bus / Blue Bus).
  - Cross-price elasticities are restricted by market shares, not product similarity.
- **The Random Coefficients Solution:**
  - Allows for flexible substitution patterns: Consumers substitute towards products with similar characteristics.
  - Captures Unobserved Heterogeneity:  $\beta_i = \bar{\beta} + \sigma\nu_i$ . Different consumers value attributes differently.

# Learning From Replication

Bridging the gap between textbook theory and computational practice:

- **Optimization:**
  - Understanding the “Inner Loop” (Contraction Mapping) vs. “Outer Loop” (GMM/MLE).
- **Computational Bottlenecks:**
  - The burden of high-dimensional integration ( $N$  consumers  $\times$   $T$  periods  $\times$   $R$  draws).
- **Modern Implementation:**
  - **Vectorization & Broadcasting:** Eliminating loops.
  - **JIT Compilation (Numba):** If loops are necessary.
  - **GPU Acceleration (PyTorch):** Parallelizing massive integral simulations.

1 Introduction

2 Mixed Logit

3 BLP

4 Integrated Model

5 Conclusion

# Utility Function and Likelihood

The utility of consumer  $i$  choosing product  $j$  is:

$$u_{ij} = x_j \beta_i + \varepsilon_{ij} = \underbrace{x_j \beta}_{\text{mean utility}} + \underbrace{\sum_k x_{jk} \sigma_k \nu_{ik}}_{\text{heterogeneity}} + \underbrace{\varepsilon_{ij}}_{\text{iid T1EV}} \quad (\text{Here, } \beta_i \sim N(\beta, \sigma^2))$$

Conditional on individual tastes  $\nu_i$ , the probability is:

$$p_{ij}(\nu_i) = \frac{\exp(x_j \beta + x_j \sigma \nu_i)}{1 + \sum_{k=1}^J \exp(x_k \beta + x_k \sigma \nu_i)}$$

We observes choices  $y_{ij}$  ( $= 1$  means  $i$  chooses  $j$ ). The likelihood function is

$$LL(\{\beta, \sigma\}) = \sum_{i=1}^N \log \int \left( \prod_{j=1}^J p_{ij}(\nu)^{y_{ij}} \right) \phi(\nu) d\nu$$

# Numerical Integration

The goal is to maximize the likelihood, but it does not have a closed form since RCs follow a normal distribution.

How to do it?

- Think of the integral as an expectation:  $E_\nu[P_{ij}(\nu)]$ .
- By the Law of Large Numbers, the sample mean converges to the expectation.

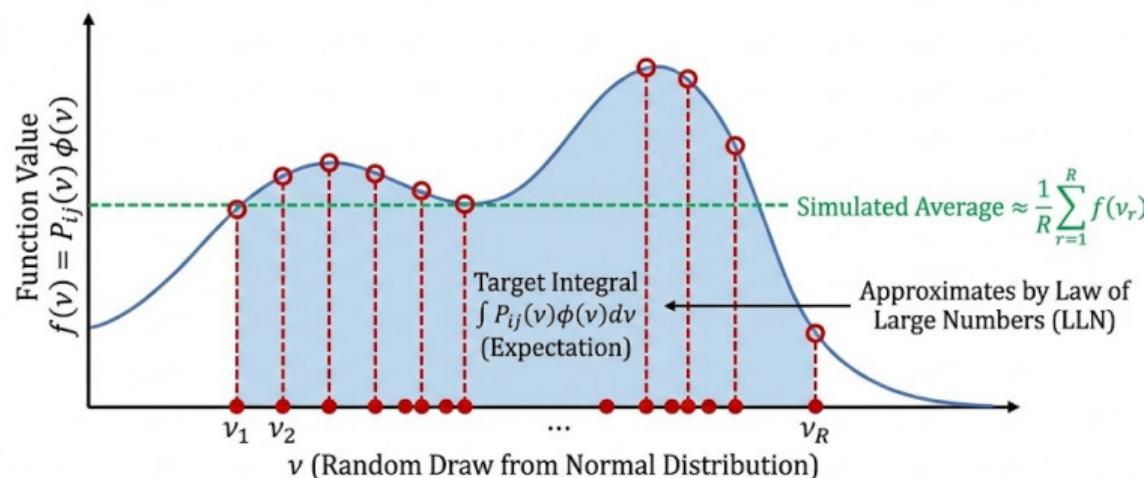
We simulate the integral:

- ① Draw  $R$  random values:  $\nu_1, \dots, \nu_R \sim N(0, 1)$ .
- ② Calculate probability for each draw:  $P_{ij}(\nu_r)$ .
- ③ Take the average:

$$SLL(\{\beta, \sigma\}) = \sum_{i=1}^N \log \left( \frac{1}{R} \sum_{r=1}^R \prod_{j=1}^J p_{ij}^{y_{ij}}(\nu_r) \right)$$

# Numerical Integration

## Numerical Integration: Monte Carlo Approximation



Concept: The integral is estimated by averaging the function values evaluated at  $R$  random draws of  $v$ . As  $R$  increases, the average converges to the true integral.



To reduce the number of draws, we can use the Halton draw which covers space more uniformly.  $\text{Halton}(R = 100) \approx \text{Random}(R = 1000)$ .

# Specify Analytical Gradient

To minimize  $f(x)$ , we need to find a step  $\Delta x$  such that  $f(x + \Delta x) < f(x)$ .

$$f(x + \Delta x) \approx f(x) + \underbrace{\nabla f(x)^T \Delta x}_{\text{Must be negative}}$$

Without  $\nabla f(x)$ , the algorithm relies on slow finite-difference guesses.

# Deriving Gradient

Consider the simplest case, no simulation. Let  $i$  choose product  $j$ . The Log-Likelihood is:

[Full derivation](#)

$$LL(\beta) = \ln P_{ij} = x_{ij}\beta - \ln \left( \sum_{k=1}^J \exp(x_{ik}\beta) \right)$$

Take the derivative w.r.t  $\beta$ :

$$\begin{aligned} \frac{\partial LL}{\partial \beta} &= x_{ij} - \frac{\partial}{\partial \beta} \ln \left( \sum_k e^{x_{ik}\beta} \right) \\ &= x_{ij} - \frac{1}{\sum_k e^{x_{ik}\beta}} \cdot \sum_k \left( e^{x_{ik}\beta} \cdot x_{ik} \right) \quad (\text{Chain Rule}) \\ &= x_{ij} - \sum_k \underbrace{\left( \frac{e^{x_{ik}\beta}}{\sum_m e^{x_{im}\beta}} \right)}_{P_{ik}(\beta)} \cdot x_{ik} \end{aligned}$$

# Deriving Gradient

The gradient has a beautiful, intuitive form:

$$\nabla_{\beta} LL = \underbrace{x_{ij}}_{\text{Observed Attribute}} - \underbrace{\sum_{k=1}^J P_{ik}(\beta)x_{ik}}_{\text{Expected Attribute } E[x]}$$

## Intuition:

- Imagine  $x$  is “Horsepower”. I chose a Ferrari ( $x_{high}$ ).
- If the model predicts I like slow cars, then  $E[x]$  is low.
- Gradient =  $x_{high} - x_{low} > 0$ .
- **Action:** Increase  $\beta_{HP}$ . The model stimulates the preference for horsepower up until Observed  $\approx$  Expected.

This is essentially the Method of Moments condition!

# The Remaining Bottleneck

Even with analytical gradients, estimation is slow. Why?

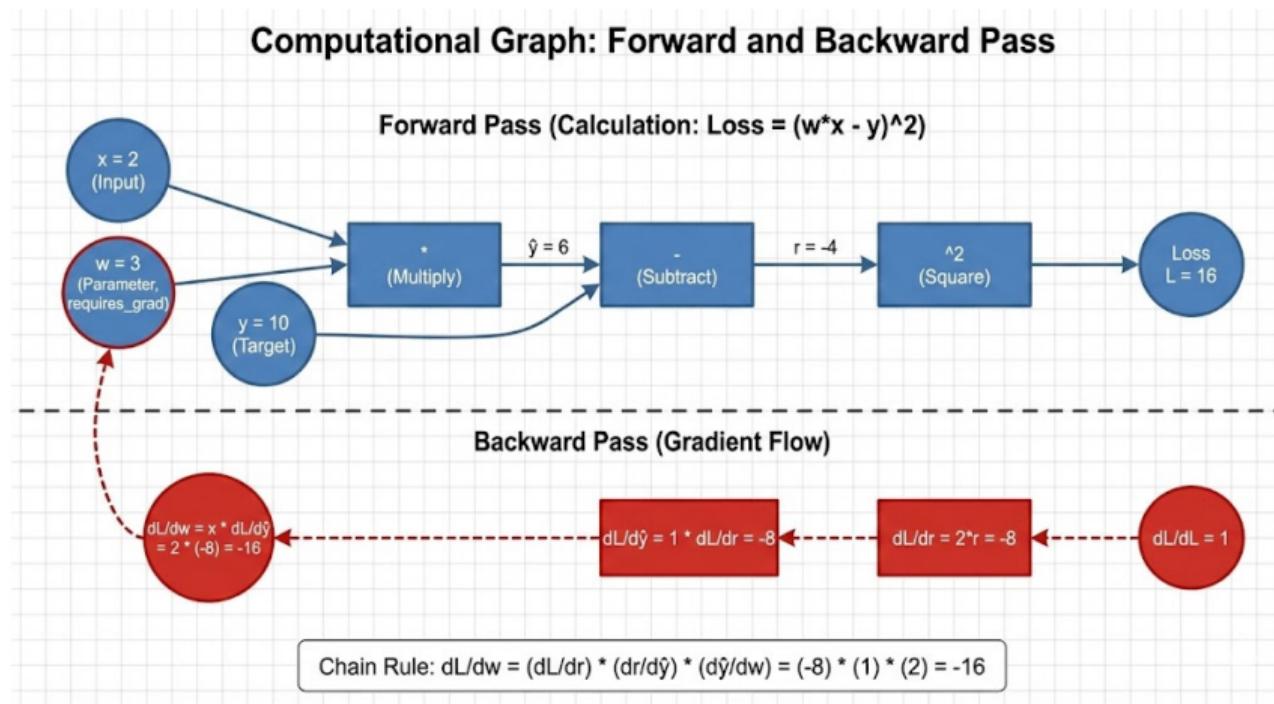
- Heavy operations
  - We perform matrix multiplication ( $X\beta$ ) and exponentiation ( $\exp$ ) for every consumer, every product, and every simulation draw.
  - Tensor Size:  $N(10^4) \times J(10) \times R(500) = 50$  Million Operations.
- The CPU limitation (sequential computing)
  - NumPy runs primarily on the CPU.
  - CPUs have few cores (8-16). They process large matrices in “chunks” (sequentially).

# Why PyTorch?

PyTorch is not just for neural networks. For structural estimation, it offers two advantages:

- Massive parallel computation (GPU Acceleration)
  - Move data from CPU to GPU.
  - GPUs have thousands of small cores.
  - We can compute utility for all  $N$  consumers and  $R$  draws simultaneously.
- Automatic Differentiation (AutoGrad)
  - No need to manually derive the complex gradient.
  - `loss.backward()` computes exact gradients using the computational graph.

# Understanding Backward Propagation



# Application: Advertising and Demand for Addictive Goods (Tuchman, 2019)

**Research Question:** Do E-cigarette ads decrease demand for traditional cigarettes?

The Model: Mixed Logit with State Dependence (Addiction)

Consumer  $i$  chooses product  $j$  at week  $t$ . Utility depends on past choices:

$$U_{ijt} = \underbrace{\alpha p_{jt} + \phi A_{mt}}_{\text{Mean Utility}} + \underbrace{\gamma \mathbb{I}(y_{it-1} = j)}_{\text{Addiction (State Dep.)}} + \underbrace{\mu_{ij}}_{\text{Heterogeneity}} + \epsilon_{ijt}$$

- Data: Nielsen Household Panel.
- Key Challenge: We must estimate the random coefficients ( $\sigma_\beta$ ) and addiction parameters ( $\gamma$ ) by integrating over thousands of consumers.

# Implementation: Manual Derivation vs. PyTorch AutoGrad

**The Task:** Estimate parameters ( $\theta$ ) using simulated maximum likelihood.

- Numpy (Manual gradient)
  - Complex math: Must derive  $\frac{\partial LL}{\partial \gamma}$  manually.
  - Runtime: approximately 1 hour 22 minutes on BlueHive.
- PyTorch (AutoGrad)
  - Zero math: Just write the utility function  $U_{ijt}$ .
  - Runtime: approximately 16 minutes on an RTX 4060 Ti

1 Introduction

2 Mixed Logit

3 BLP

4 Integrated Model

5 Conclusion

# Transition: From Micro-Data to Market-Level Data

We have seen how Mixed Logit handles heterogeneity using household panel data.

## But what if we only have Aggregate Data?

- Data: Market shares ( $s_{jt}$ ), Prices ( $p_{jt}$ ), Characteristics ( $x_{jt}$ ).
- No individual choices ( $y_{ijt}$ ) are observed.

## The “Unobserved Quality” Problem ( $\xi_{jt}$ )

$$u_{ijt} = \alpha_i p_{jt} + x_{jt} \beta_i + \underbrace{\xi_{jt}}_{\text{Unobserved Quality}} + \epsilon_{ijt}$$

- $\xi_{jt}$ : Characteristics observed by consumers and firms (e.g., style, prestige), but *unobserved* by the econometrician.
- Endogeneity: Firms set prices based on  $\xi_{jt}$  (High quality  $\rightarrow$  High price).
- Result:  $\text{Corr}(p_{jt}, \xi_{jt}) > 0$ , which yields biased price elasticities.

# The BLP Solution: GMM with Instrumental Variables

Berry, Levinsohn, and Pakes (1995) propose a solution to estimate mixed logit with aggregate data and endogenous prices.

## Key Idea: Inversion

- We cannot use MLE because  $\xi_{jt}$  is not random noise; it's correlated with price.
- Instead, we invert the market share function to recover the “Mean Utility” ( $\delta_{jt}$ ):

$$s_{jt}(\delta, \theta_2) = S_{jt}^{Data} \implies \delta_{jt}(\theta_2) = s^{-1}(S^{Data}, \theta_2)$$

Once we recover  $\delta_{jt}$ , we have a linear equation:

$$\delta_{jt} = x_{jt}\beta + \alpha p_{jt} + \xi_{jt}$$

Now we can use IVs to handle the correlation between  $p_{jt}$  and  $\xi_{jt}$ .

# Berry's Inversion (1994)

The market share (there is no RC):

$$s_{jt} = \frac{\exp(x_{jt}\beta - \alpha p_{jt} + \xi_{jt})}{1 + \sum_k \exp(x_{kt}\beta - \alpha p_{kt} + \xi_{kt})}$$

Normalize the outside utility to zero:

$$s_{0t} = \frac{1}{1 + \sum_k \exp(\delta_{kt})}$$

This implies

$$\ln \frac{s_{jt}}{s_{0t}} = x_{jt}\beta - \alpha p_{jt} + \xi_{jt}$$

**Implication:** We can now estimate this using IV Regression (2SLS/GMM) to solve the correlation between  $p_{jt}$  and  $\xi_{jt}$ .

# When There are Random Coefficients

The market share is the integral of individual probabilities over the population distribution  $F(\nu)$ :

$$s_{jt}(\delta, \theta_2) = \int \frac{\exp(\delta_{jt} + \mu_{ijt})}{1 + \sum_k \exp(\delta_{kt} + \mu_{ikt})} dF(\nu)$$

Here,  $\delta_{jt} = x_{jt}\beta - \alpha p_{jt} + \xi_{jt}$ , and  $\mu_{ijt} = \sum_k x_{jtk}(\sigma_k \nu_{ik}) - p_{jt}(\sigma_p \nu_{ip})$

Since the integral has no closed form, we approximate it by simulation (averaging over  $R$  draws):

$$s_{jt} \approx \frac{1}{R} \sum_{r=1}^R \frac{\exp(\delta_{jt} + \mu_{rjt})}{1 + \sum_k \exp(\delta_{kt} + \mu_{rkt})}$$

## Inner Loop: Contraction Mapping

Since we cannot analytically invert the integral  $s_{jt}(\delta, \theta_2)$ , we solve for  $\delta$  numerically.

From Berry (1994), we use the following iteration:

$$\delta_{jt}^{h+1} = \delta_{jt}^h + \ln(S_{jt}^{Data}) - \ln(s_{jt}(\delta^h, \theta_2))$$

- **Intuition:** If Model Share < Data Share, increase  $\delta$ .
- **Convergence:**  $T(\delta) = \delta + \ln(s) - \ln(s(\delta))$  is a contraction mapping (Berry, 1994).

Connection to Rust's NFXP: BLP is structurally identical to John Rust's Nested Fixed Point algorithm.

$$T(W(x)) = \max_d [u(x, d; \theta) + \beta E[W(x') | x, d; \theta]]$$

## The Outer Loop: Estimating Parameters via GMM

Once the contraction mapping converges ( $|\delta_{jt}^{h+1} - \delta_{jt}^h| < \text{threshold}$ ) for a given guess of non-linear parameters  $\theta_2$ , we obtain the mean utilities  $\delta_{jt}(\theta_2)$ .

- ① Recover linear parameters  $(\hat{\beta}, \hat{\alpha})$  using 2SLS and the structural error  $(\xi_{jt})$ :

$$\xi_{jt}(\theta) = \delta_{jt}(\theta_2) - (x_{jt}\hat{\beta} - \hat{\alpha}p_{jt})$$

- ② Create moment conditions: since price is endogenous ( $E[\xi p] \neq 0$ ), we use Instruments  $Z_{jt}$ :

$$\bar{g}(\theta) = \frac{1}{J \times T} \sum_{j,t} Z'_{jt} \xi_{jt}(\theta)$$

- ③ GMM: we search for parameters  $\hat{\theta}_2$  that minimize the distance of moments to zero:

$$\min_{\theta_2} J(\theta_2) = \bar{g}(\theta_2)' W \bar{g}(\theta_2) = \xi(\theta_2)' Z W Z' \xi(\theta_2)$$

# Identification

Recall the moment conditions we used:

$$E(\xi_{jt} X_{jt}) = 0, \quad E(\xi_{jt} Z_{jt}) = 0.$$

$E(\xi_{jt} X_{jt}) = 0$  are used to identify  $\beta$ . One of the  $E(\xi_{jt} Z_{jt}) = 0$  conditions (where  $Z_{jt}$  is an IV for price) is used to identify  $\alpha$ . The identification of  $\sigma$  comes from the rest of the  $E(\xi_{jt} Z_{jt}) = 0$  conditions.

## Types of instruments:

- Cost shifters
- Hausman IV: price of product  $j$  in other markets  $p_{j't}$
- BLP IV: characteristics of rival products  $x_{-j,t}$ , and other variants:
  - $\sum_{j' \neq j} (X_{jtm} - X_{j',t,m})^2$
  - $\sum_{j' \neq j} (X_{jtm} - X_{j',t,m}) (X_{jtm'} - X_{j',t,m'})$

## Identificaiton

**Question:** What variations identify random coefficients?

**Answer:** Substitution patterns - with random coefficients, consumers substitute more to a similar product.

Variations in substitution patterns:

- Substitution based on characteristic distance.
- Choice set variations.

Sometimes, there is not enough variation to identify the unobserved heterogeneity in the data. Instead, we can identify the heterogeneity correlated to demographics. It is easier to use observables to explain substitution patterns.

$$\beta_i = \bar{\beta} + \underbrace{\sum \nu_i}_{\text{unobserved}} + \underbrace{\prod D_i}_{\text{demographics}}$$

# Application: Inference and Impact of Category Captaincy (Zhu, 2024)

This paper focuses on category captaincy, which is a vertical arrangement where retailers delegate shelf placement and pricing to a leading manufacturer. However, the contract is confidential. In the data, we can just see Dannon leads in some chains, Yoplait in others. Is this due to consumer preference or shelf advantage (better display, space)?

Utility of consumer  $i$  for product  $j$  at retailer  $r$ :

$$u_{ijrmt} = \bar{u}(x_{jrmt}, \beta, \eta_{irmt}) + \xi_{jrmt} + \varepsilon_{ijrmt}$$

**Key Innovation:** Decompose unobserved quality  $\xi_{jrmt}$ :

$$\xi_{jrmt} = \xi_{jt} + \underbrace{\xi_{brt}}_{\text{Captaincy Effect}} + \Delta\xi_{jrmt}$$

# Application: Inference and Impact of Category Captaincy

**Identification:** To solve price endogeneity, the paper uses two sets of IVs (selected using the specification test (Gandhi and Houde 2019) and Lasso (Belloni et al. 2012)):

- **Cost Shifters:** cost or markup shifters, which includes input/transportation costs interacted with product characteristics, assortment variables, and demographic characteristics of a brand's main market within retailer interacted with that brand's product characteristics
- **Differentiation IVs:** characterizes the degree of differentiation of each product in a retailer market and exploits variation in household demographics across retailer markets (Gandhi & Houde, 2019).

# PyBLP

The burdens of hand-coding BLP: updating the optimal weighting matrix, deriving the analytical Jacobian, and managing numerical stability with high-precision tolerances. PyBLP handles these burdensome steps. That's why I recommend it.

Aside from the estimation of BLP, it can also do post-estimation practices: price elasticities, consumer surplus calculation, markup and margin calculation, and merger simulation.

The estimation in `pyblp` is organized around three core objects: Formulation, Integration, and Problem.

# The Core Workflow in PyBLP

1. Model Configuration: separate linear parameters ( $\beta, \alpha$ ) from non-linear random coefficients ( $\Sigma$ ).

```
X1_formulation = pyblp.Formulation(  
    '0 + prices + sugar + sodium + fat + size + # flavor + calorie +  
    organic + # sales',  
    absorb='C(fe_product_year) + C(fe_brand_retailer_year)'  
)  
  
X2_formulation = pyblp.Formulation(  
    '1 + prices + # flavor + size + sodium + sugar + fat'  
)  
  
agent_formulation = pyblp.Formulation('0 + income + kids + edu')
```

# The Core Workflow in PyBLP

## 2. Integration and Initialization:

```
problem = pyblp.Problem(  
    product_formulations=(X1_formulation, X2_formulation),  
    product_data=product_data,  
    agent_data=agent_data,  
    agent_formulation=agent_formulation  
)  
  
results = problem.solve(  
    sigma=initial_sigma,  
    sigma_bounds=(np.zeros((7, 7)), np.zeros((7, 7))),  
    pi=initial_pi, pi_bounds=(pi_lower, pi_upper),  
    optimization=pyblp.Optimization('l-bfgs-b', {'gtol': 1e-4}),  
    method='2s', se_type='clustered'  
)
```

1 Introduction

2 Mixed Logit

3 BLP

4 Integrated Model

5 Conclusion

# Revisit Tuchman (2019)

## Problem:

- Micro data alone: Sparse. Hard to estimate precise brand-market-time fixed effects ( $\delta_{jmt}$ ) needed to control for price endogeneity.
- Macro data alone: Hard to identify complex heterogeneity distributions ( $\Sigma$ ) and state dependence parameters ( $\gamma$ ).

## Solution:

- Use aggregate data to pin down mean utilities ( $\delta_{jmt}$ ).
- Use household data to identify non-linear parameters ( $\Sigma, \gamma$ ).
- Consistency Constraint: The  $\delta_{jmt}$  that rationalizes aggregate shares must be the same  $\delta_{jmt}$  entering the household likelihood.

# The Estimation Algorithm

We iterate between two steps until convergence of non-linear parameters  $\theta_2 = \{\Sigma, \gamma\}$ :

## Step 1: The Macro Step (Contraction Mapping)

- Given  $\theta_2$ , solve for  $\delta_{jmt}$  such that  $s^{pred}(\delta, \theta_2) = s^{obs}$ .
- Challenge: Standard BLP is parallel over markets ( $t$  is independent). Tuchman's model is recursive: Market shares at  $t$  depend on market shares  $t - 1$ .

## Step 2: The Micro Step (MLE)

- Treat  $\delta_{jmt}$  as data (fixed).
- Maximize  $LL(\theta_2 | \delta, \text{Household Data})$ .

## Step 3: Linear Parameters (Post-Convergence)

- Regress converged  $\delta_{jmt}$  on Prices/Ads (Border strategy).

# Step 1: Handling Recursion with Numba

$$s_{jmt} = \int \sum_k \pi_{jmt}(k) \cdot \underbrace{\Pr(y_{it-1} = k)}_{\text{Evolves over time!}} dF(\beta)$$

## Implementation Detail:

- Cannot vectorize over  $T$ . We must loop  $t = 1 \dots T$ .
- Pure python loops are too slow for the contraction mapping.
- **Solution:** `@jit(nopython=True)` compiles the recursion to machine code.

```
# Core logic inside Numba
for t in range(T):
    if t > 0:
        delta_t = delta_sol[t-1, :, :].copy() # History dependence

    # Contraction Mapping Loop for current t
    while error > 1e-9:
        # ... solve for delta_t ...

    # Update state distribution for t+1
    prob_state_lag = update_probs(probs)
```

## Step 2: GPU Acceleration with PyTorch

$$LL = \sum_i \ln \int \prod_t P_{ijt}(\delta_{jmt}, \gamma, \Sigma) dF(\nu)$$

### Implementation Detail:

- We need to broadcast the  $\delta_{jmt}$  (solved in Step 1) to thousands of households.
- This involves massive matrix operations: ( $N_{HH} \times T \times R_{draws}$ ).
- **Solution:** Move tensors to GPU.

```
# Broadcasting Macro delta to Micro observations
t_delta_i = t_delta_est[self.t_h_week, self.t_h_mkt, :].unsqueeze(2)

# Adding Heterogeneity (Sigma) and Addiction (Gamma)
t_U = t_delta_i + t_mu_matrix + t_lag_boost

# Choice Probabilities computed in parallel on GPU
t_exp_U = torch.exp(t_U)
```

# Handling Initial Conditions: The Burn-in Period

## Issue:

- At  $t = 0$ , we don't know the distribution of addicted users.
- Assuming an arbitrary starting distribution (e.g., all non-smokers) biases the early estimates.

## Solution in Code:

- Simulate the first 24 weeks without using them in the likelihood.
- Start from a steady state:

$$LL(\theta) = \sum_{t=\text{burn\_in}}^T \ln P(y_{it} | y_{it-1}, \dots)$$

1 Introduction

2 Mixed Logit

3 BLP

4 Integrated Model

5 Conclusion

# Conclusion

## Summary of Methods:

- Mixed Logit: Best for micro-data, handles heterogeneity via simulation, but computationally heavy.
- BLP: Best for aggregate-data, handles price endogeneity via inversion ( $\delta$ ) and IVs.
- Integrated Model: Combines the identification power of micro-data ( $\Sigma, \gamma$ ) with the endogeneity control of aggregate-data ( $\delta$ ).

## The Takeaway:

- Tools like PyTorch and Numba allow us to estimate complex dynamics (e.g., addiction, state dependence) that were previously intractable.
- Final Thought: A good model is not just theoretically sound, but computationally feasible.

# Deriving Gradient

Let  $P_{ij}(\nu_r)$  be the probability of the chosen alternative  $j$  for consumer  $i$  in simulation draw  $r$ .

The simulated probability is the average over draws:

$$\hat{P}_{ij} = \frac{1}{R} \sum_{r=1}^R P_{ij}(\nu_r)$$

The gradient of the Log-Likelihood with respect to  $\beta$ :

$$\frac{\partial SLL}{\partial \beta} = \sum_{i=1}^N \frac{\partial \log \hat{P}_{ij}}{\partial \beta} = \sum_{i=1}^N \frac{1}{\hat{P}_{ij}} \cdot \frac{\partial \hat{P}_{ij}}{\partial \beta}$$

Substitute the derivative of the simulated probability:

$$\frac{\partial \hat{P}_{ij}}{\partial \beta} = \frac{1}{R} \sum_{r=1}^R P_{ij}(\nu_r) \left( x_{ij} - \sum_k P_{ik}(\nu_r) x_{ik} \right)$$

# Deriving Gradient

Combining terms, the gradient has a beautiful structure:

$$\nabla_{\beta} SLL = \sum_{i=1}^N \sum_{r=1}^R \underbrace{\left[ \frac{P_{ij}(\nu_r)}{\sum_{m=1}^R P_{ij}(\nu_m)} \right]}_{\text{Weight } w_{ir}} \cdot \underbrace{(x_{ij} - \bar{x}_i(\nu_r))}_{\text{Deviation}}$$

## Interpretation:

- Deviation:  $(x_{ij} - \bar{x}_i(\nu_r))$  is the difference between the observed attribute and the predicted attribute.
- Weights  $w_{ir}$ : These are posterior probabilities of the draws (there exists bayesian updating within MLE).
- **Mechanism:** The gradient gives more weight to simulation draws ( $\nu_r$ ) that make the observed choice more likely.