README: Nonparametric Demand Estimation

James Brand*

September 9, 2020

1 Description

This repository contains programs intended to make it easier for researchers to estimate demand functions nonparametrically in settings which fall in one of two categories:

1. The number of products is small (< 5)
2. The number of products is *large* but each product is a close substitute with only a small subset of other products

Accordingly, there are two procedures implemented. In case (1), one can simply apply Compiani (2020)'s nonparametric demand estimator. In case (2), the researcher needs to either specify the relevant substitutes for each product or

1.1 Implemented Model

See Compiani (2020) for details on the estimator and the model. Briefly, the model of interest is one in which there are J products (total, or in a subgroup of close substitutes) in many markets. The demand for each product j in market t can be written

$$s_{jt} = \sigma_j(p_t, \xi_{jt}) \tag{1}$$

*University of Texas at Austin. email:: jamesbrand@utexas.edu

where p_t is a vector of j prices and ξ_{jt} is an unobservable shock. The version of this model implemented here further assumes that

$$s_{jt} = \sigma_j(\delta_{jt}, p_{-jt})$$
$$\text{where } \delta_{jt} = -p_{jt} + \xi_{jt}$$

and p_{-jt} is a vector of $J-1$ prices, excluding the price of product j, and where σ_j is assumed to be monotonically increasing in δ_{jt}. Under these assumptions, we can invert the vector $\sigma \equiv (\sigma_1, \ldots, \sigma_J)$ to arrive at another J equations:

$$\delta_{jt} = \sigma_j^{-1}(s_t)$$
$$-p_{jt} = \sigma_j^{-1}(s_t) - \xi_{jt}$$

Now, the basic approach of my code is to estimate σ_j^{-1} by (potentially constrained) IV regression. From those estimates, we can use estimates of derivatives of σ^{-1} to calculate price elasticities of demand.

In cases (2), where the number of products is large, I first run a model selection procedure which is a combination of Bien et al (2013)'s "hierarchical lasso" and Bach (2008)'s "bolasso." The procedure will described in full detail in the draft of my paper on this approach, but essentially amounts to running an IV lasso regression for each product j of $-p_{jt}$ on the a polynomial in the vector market shares, constraining the polynomial coefficients such that the selected model is easily interpretable (e.g. if the quadratic term in s_{kt} is included, the linear term must be included as well. I run this procedure for multiple bootstrapped samples and take the intersection of included products, which reduces the likelihood that extraneous products will be included with very little cost in terms of excluding some relevant products. The selected products are then marked as "substitutes" for product j.

I specifically choose not to impose other constraints, in particular exchangeability, for two reasons. First, I find in practice that the best-working order for the sieve corresponding to each inverse demand function (σ_j^{-1}) depends on the available variation in the price of good j. Thus, in practice, researchers may want the order of the sieve to vary for each product. It is much more difficult to implement exchangeability with this flexibility, and I have not spent time on it. Second, some models of interest violate exchangeabiltiy explicitly. The most obvious example is one in which consumers are imperfectly attentive (see work by

Abaluck and Adams). Still, dropping exchangeability can come at a considerable (efficiency) cost, as doing so is comparable to reducing the sample size by a factor of J. In cases in which one is willing to impose this restriction, researchers should use the package released by Giovanni Compiani himself, which is very well written, easy to use, and can be found on his website. He also includes a version of his code which does not include price in the index δ, which may also be of interest to some.

1.2 Files

In this repository I have included the following files:

- *b.jl, db.jl, bern.jl, dbern.jl, fullInteraction.jl, makeConstraint.jl, solve_s_nested_flexible.jl, objective_priceIndex.jl*
- *inverse_demand.jl*
- *simulate_logit.jl*
- *price_elasticity_priceIndex.jl*
- *basic_example.jl, model_selection_example.jl, pure_model_selection.jl*

The first set of programs perform background tasks which construct the Bernstein polynomial sieves and (monotonicity) constraints, and can generally be ignored. The main estimation file is *inverse_demand.m*, which takes as inputs matrices of market shares, prices, and instruments (for market shares), and estimates inverse demand functions for all products. Each σ_j^{-1} is estimated separately. This saves a lot of memory, as estimation requires constructing and inverting some very large matrices.

The program *price_elasticity_priceIndex.m* calculates price elasticities using the results of *inverse_demand.m*. Elasticities can be calculated either at a specified vector of prices and "deltas" (the inverted index) or at the realized vector of market shares (controlled by the option *trueS*). The program produces three main outputs. The first is the vector of price elasticities requested. The second is an array where each element contains the full estimated Jacobian for the corresponding market. These estimates can be used to, for example, quickly calculate markups in a Nash-Bertrand or monopoly pricing model. The third output is the

vector of market shares implied by the supplied prices and deltas when *trueS* is set to zero. These are the market shares at which elasticities are calculated.

The file *basic_example.m* implements a very simple example to demonstrate the functionality of the estimation programs. In this file, I simulate a simple demand system using *simulate_logit.m*, estimate J demand functions, and then calculate elasticities twice. First, I set all prices except for that of product 1 to its median value. I move the price of product 1 from the 25^{th} to the 75^{th} percentile of its distribution, and calculate own-price elasticities at each of these values. Then, using the option *trueS*, I instead estimate own-price elasticities for product 1 in each simulated market. I conduct each of these operations for S simulated samples, and summarize the results of the first two exercises in figures at the end of the program.

The files *model_selection_example.jl* and *pure_model_selection.jl* demonstrate the main model selection features of the package. The latter is an example which simulates a system of demand in which each product is only a substitute for half of the products in the market. I then run the model selection procedure described above. The output of this file are summary dataframes which demonstrate the performance of this selection procedure over many simulated samples. The former file performs model selection and estimates demand nonparametrically on the selected model.

1.3 Some Comments

In practice, the order of the Bernstein polynomials is a huge degree of freedom for researchers. Setting the order too high makes estimates unstable, and too low induces significant biases. This is unavoidable for now, but users should experiment with multiple orders. I'll mention that in my simulations, low-order estimates (even when biased) perform better than parametric estimates which are incorrectly specified. It is also worth emphasizing that we are targeting the *inverse* demand function, which we then manipulate to get functionals of the demand function itself. When the actual demand function is very steep (with respect to price), the inverse demand function is shallow, and precise estimates require much more data. In smaller samples this can cause numerical issues, and estimates in these cases are often unstable. Finally, I'll note that in general I find that these nonparametric estimates generate long tails in the distribution of price elasticities (i.e. when *trueS* is zero). This is ameliorated if you allow the order of the polynomial sieve to grow with the sample size.