README: Nonparametric Demand Estimation

James Brand*

April 19, 2020

1 Details

Let me first offer a disclaimer that I am by no means an expert programmer. I have relied on the Matlab function *eval* for some parts of the code, which honestly make the code in the main estimation files somewhat difficult to read. There are surely ways to make this code more efficient (some of which I am working on). Thankfully, there should be little to no need for a user to modify these programs, as I have tried to comment the example file such that all of the features of the code are obvious. My hope is that, for researchers studying demand in settings with only a few products, these programs lower the cost of trying nonparametric methods. I have an older version of these files written in R, and I hope to update and post them soon.

1.1 Implemented Model

The code I have included herein implements the nonparametric demand estimation procedure introduced by Giovanni Compiani's job market paper. The package is designed for estimating demand when both the number of products and the number of product characteristics are small, and many markets with the same choice set are observed. This package only estimates the version of his model in which price is included in the index. I allow the option to include another (exogenous) covariate x in the index as well. I also allow other covariates to be included in the demand function. Giovanni, in his paper, discusses a number

---

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

of constraints on the inverse demand function which can reduce the number of parameters to be estimated. In this package I have only implemented the constraint that all demand functions are monotonic in the index.

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 in this setting, 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 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, which may also be of interest to some.

1.2 Files

I use CVX instead of *fmincon* as the minimizer in estimation. In order to run the programs herein, you must first install CVX from http://cvxr.com/cvx/doc/install.html. In this zip file I have included the following files:

- *b.m, db.m, bern.m, dbern.m, fullInteraction.m, makeConstraint.m, solve_s_nested_flexible.m, objective_priceIndex.m*
- *inverse_demand.m*
- *simulate_logit.m*
- *price_elasticity_priceIndex.m*
- *example1.m*

The first set of programs should essentially be ignored, as they perform background tasks which construct the Bernstein polynomial sieves and (monotonicity) constraints. The main estimation file is *inverse_demand.m*, which 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. Doing so separately by demand function shrinks the size of these matrices and lets us store only one in memory at a time. This makes a big difference with large datasets or large sieves. One could pool the estimation into a single step if they wished to impose relationships between parameters of different inverse demand functions, but without exchangeability this problem can become infeasible very quickly as the number of products increases.

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 outputs worth discussing here. The first is the vector of price elasticities requested. The second is a cell matrix, where each row is a cell containing the 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 *example1.m* implements a very simple example to demonstrate the functionality of the estimation programs. In this file, I simulate an extremely simple demand system using *simulate_logit.m*, estimate demand, and then calculate elasticities three times. 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. Next, I perform the same operation for cross-price elasticities. Finally, 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.

I plan to update this file soon with details for a couple of model selection approaches, which can in principle allow a researcher to use these methods in markets of more reasonable size (think 10-20, though larger may be possible in some cases).

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. The program *price_elasticity_priceIndex.m* will report the number of markets which create any numerical warnings/errors or NaNs as the number of "bad markets." 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). Essentially, at the tails of the price distribution, price elasticities are often over-estimated because the curvature of demand (second derivatives) are imprecise at the tails. These estimates are still often better (on average) than misspecified parametric estimates, but it is good to know this is a potential concern.