Estimating Price Elasticity Matrices

Maximilian Schaller Stephen Boyd

April 12, 2026

Abstract

The relationship between demand and prices of a set of products can be modeled as a linear mapping from logarithmic price changes to logarithmic changes in demand. We consider the problem of estimating the coefficient matrix of this mapping, the elasticity matrix, based on observed data consisting of real-valued prices and integer-valued demands. We regularize the estimation problem by imposing a factor model structure, *i.e.*, that the elasticity matrix is diagonal plus low-rank, similar to factor models used for financial returns. Maximizing the likelihood of observations of this model is a bi-convex problem, meaning that there is a partition of the variables in which it is convex in each set when the other is fixed. We propose and compare three methods for finding a locally optimal estimate. The first is based on alternating maximization, and involves solving a sequence of convex problems. The second method exploits efficient gradient computations in a gradient ascent method. The final method is to use a general purpose nonlinear programming method. While all methods give the same result on numerical examples, the gradient ascent method is substantially faster, due to its efficient gradient evaluations. We report the likelihood with different hyper-parameters for synthetic and real-world data, with similar results. For synthetic data, we also report the realized profit when using the elasticity estimate for optimal pricing, which is maximized for the same set of hyper-parameters that also maximizes the likelihood. This paper is accompanied by easy to use open source Python code for fitting elasticity matrices to observed data, using our three numerical methods.

Contents

1 Introduction

1.1 Price elasticity of the demand

Already in the 1960s, the relationship between demand and prices of multiple products has been analyzed, in terms of the Rotterdam model [Bar64, The65], which is a local first-order approximation of utility-maximizing demand systems [AV18, MCWG95, §3.D]. In particular, the Rotterdam model relates local changes in log-demand linearly to local changes in log-prices. The coefficient matrix is referred to as the *elasticity matrix* or *Slutsky substitution matrix* [MCWG95, §2.F], and is constant. The diagonal entries of such a matrix are typically referred to as *self-elasticities* and the off-diagonal entries are typically referred to as *cross-elasticities*. The Translog model [CJL75] from the 1970s and the PIGL and PIGLOG architectures [Mue75], including the widely used almost ideal demand system (AIDS) [DM80a] and LA-AIDS [DM80b] from the 1980s, compute budget shares as linear functions of log-price changes. One can convert those coefficients to demand elasticities, which depend on the budget shares, and therefore on the prices themselves. The exact affine Stone index (EASI) [LP09] model from the 2000s is a generalization of AIDS, and shares the property of non-constant elasticity.

The above models generally result in full-rank elasticity matrices, potentially rendering the estimation of such for large product portfolios computationally expensive, or intractable. The nested logit [Ber94] and nested constant elasticity of substitution (CES) [BW06] models from the 1990s and 2000s, respectively, impose structure on the demand model, by grouping products into nests. Within a nest, cross-elasticities are larger, and across nests, cross-elasticities are smaller. This structure corresponds to a low-rank plus diagonal elasticity matrix. The low-rank part can be constructed by summing a rank-1 matrix per nest, plus the own-price effects. Recent work on low-rank cross-price effects [FL25] imposes similar structure, by using *aggregators*, which are functions that map information from the set of prices to a scalar, which is then mapped to the demand for all products.

While these approaches share the idea of low-rank cross-elasticity, motivated from first principles, we propose a generic low-rank plus diagonal structure for identifying elasticity matrices in a data-driven way.

1.2 Our contribution

We provide a simple and efficient method for fitting low-rank plus diagonal elasticity matrices to data, that can be used for, *e.g.*, optimal pricing [Wil93, GVR94, FLSL16, SB25].

A low-rank plus diagonal elasticity matrix $E \in \mathbf{R}^{n\times n}$ (for n goods) is one of the form

$$E = BC^T + \mathbf{diag}(s),$$

where the factors $B, C \in \mathbf{R}^{n\times r}$ have rank $r < n$, and $s \in \mathbf{R}^n$. As described above, there are several interpretations of this structure [BW06, FL25]. We suggest a general, statistical interpretation. Each column of C is a principal direction in the space of log-price changes. The columns of B attribute price movements along these directions, respectively, to changes

in the principle directions of log-demand. The part of the self-elasticities that is not captured by the low-rank part is captured in s.

The same structure is widely used to model the (symmetric, positive definite) covariance of financial returns [JOP+23, LP20a, LP20b, FF92, FF93]. In that case, B and C (which are equal in this case) are the *factor loadings*, *i.e.*, mappings between n financial returns and r statistical factors [LP20a, LP20b] or fundamental factors [FF92, FF93] (that explain the returns), times a Cholesky root of the factor covariance. The diagonal part s (which is positive in this case) then corresponds to the idiosyncratic risk of the the individual financial instruments.

We explore several methods for fitting such elasticity matrices to data, where the demand is measured in counts of goods sold, *i.e.*, is integer-valued, and prices are real-valued. For typical measurement intervals of, say, large online retailers, counts can be large (*e.g.*, thousands or more), so that one could reasonably approximate demand as real-valued, but we do not make this assumption.

One method we use is *alternating maximization* [JNS13, UHZB16], where in each iteration one of the factors B and C of the low-rank part of the matrix is held constant, which yields a convex optimization problem that can be effectively solved. The other method is a gradient ascent method with a simple step size schedule, where we make heavy use of fast gradient computations that exploit the specific problem structure. We compare the two methods to a generic nonlinear programming solver. We find that all methods give the same result on our numerical examples, so none produces better fits. Our gradient ascent method is, however, much faster than the other two methods, due to its efficient gradient computations.

We evaluate the elasticity estimates statistically in terms of likelihood under a simple Poisson model. In synthetic examples we also evaluate the results in terms of realized profit when used within the product pricing problem from [SB25], which is a natural end use of an elasticity model. It is re-assuring that the model hyper-parameters (rank and the regularization factor for the size of the low-rank part) are the same when chosen using these two evaluation methods. For real data from the Dominick's Finer Foods (DFF) dataset [Jam97], the likelihood behaves similarly as a function of the hyper-parameters.

Code and data to reproduce the results of this paper, as well as to fit general elasticity matrices with different structure, is available at

`https://github.com/cvxgrp/elasticity-estimation`.

1.3 Outline

In §2 we introduce a simple Poisson model for the demand given prices of multiple products, and describe how it relates to a log-linear demand model, where a price elasticity matrix maps changes in log-prices to changes in log-demand. In §3 we describe how the maximum likelihood estimate for such an elasticity matrix can be obtained via convex optimization, and how the estimation problem becomes nonconvex when a low-rank plus diagonal structure is imposed on the elasticity matrix. In §4 we present three numerical methods for fitting a

low-rank plus diagonal elasticity matrix to data. We define evaluation metrics in §5 before reporting experimental results in §6, for synthetic and real demand data.

2 Price elasticity matrices

We follow the notation from [SB25]. For a collection of n goods (products, services, etc.), the demand d_i is the (integer) quantity of product i sold at price p_i in a given time interval. The respective vectors of demand and price are $d \in \mathbf{Z}_+^n$ and $p \in \mathbf{R}_+^n$.

2.1 Poisson model of the demand

We model the demand, which takes on integer values, as an IID Poisson distribution, [Fel91] (which is often used in demand modeling [ST06, BHH12]),

$$\mathbf{Prob}(d_i = k) = \frac{\lambda_i(p)^k e^{-\lambda_i(p)}}{k!}, \quad k = 0, 1, \ldots, \quad i = 1, \ldots, n, \tag{1}$$

where the rates $\lambda_i \in \mathbf{R}_{++}$ (equal to $\mathbf{E}\, d_i$) are functions of the price vector p. In particular, we use the Poisson regression model [CWA09, HH14] with n features

$$\log \lambda_i = \alpha_i + \sum_{j=1}^{n} \beta_{ij} f_j(p_j), \quad i = 1, \ldots, n, \tag{2}$$

where $\alpha_i \in \mathbf{R}$ and $\beta_{ij} \in \mathbf{R}$ are model parameters and $f_j : \mathbf{R}_+ \to \mathbf{R}$ is a function of the jth price. This is sometimes referred to as a generalized linear model [HP17] with a logarithmic link function. Here, the jth feature $f_j(p_j)$ depends on the jth price p_j.

2.2 Price elasticity of the demand

We consider (positive) nominal prices $p_1^{\text{nom}}, \ldots, p_n^{\text{nom}}$, which are typically the prices at which the products (or similar reference products, respectively) have been sold in the past. We assume that they are known. We denote the demand for product i at its nominal price p_i^{nom} as d_i^{nom}, which is not known, in general, and real-valued as it represents the respective Poisson rate.

In the model (2), we may interpret the offset α_i as $\log d_i^{\text{nom}}$ and set

$$f_j(p_j) = \log(p_j / p_j^{\text{nom}}),$$

so we can re-write (2) as

$$\log(\mathbf{E}\, d_i / d_i^{\text{nom}}) = \sum_{j=1}^{n} \beta_{ij} \log(p_j / p_j^{\text{nom}}).$$

We see that it is natural to view the relationship between demand and prices in terms of the logarithmic fractional (expected) demand and price changes

$$\delta_i = \log(\mathbf{E}\, d_i/d_i^{\rm nom}), \quad \pi_i = \log(p_i/p_i^{\rm nom}), \quad i = 1, \ldots, n,$$

respectively. We let $\delta, \pi \in \mathbf{R}^n$ denote the vectors of demand and price changes, respectively. Using matrix notation, we arrive at

$$\delta = E\pi,$$

where $E \in \mathbf{R}^{n\times n}$ is the elasticity matrix as introduced in §1.2, with $E_{ij} = \beta_{ij}$, the elasticity of the demand for product i with respect to a changed price of product j.

3 Estimating elasticity matrices

3.1 Maximum likelihood estimate

We consider a sample of N observations $d^{(j)} \in \mathbf{Z}_+^n, p^{(j)} \in \mathbf{R}_+^n$ drawn IID from (1). We denote the respective price changes from nominal by $\pi^{(j)} = \log(p^{(j)}/p^{\rm nom})$, and the rates according to (2) by $\lambda^{(j)}$. The log-likelihood takes the form

$$\sum_{j=1}^{N}\sum_{i=1}^{n}(d_i^{(j)} \log \lambda_i^{(j)} - \lambda_i^{(j)}),$$

where we dropped constant terms, and the dependence on the model parameters is implicit via the Poisson regression model (2).

We define $y_i^{(j)} = \log \lambda_i^{(j)}$ and

$$\tilde{E} = \left[\begin{array}{cc} E & \log d^{\rm nom} \end{array}\right], \quad \tilde{\pi}^{(j)} = (\pi^{(j)}, 1), \quad j = 1, \ldots, N,$$

where log is taken elementwise, so $y^{(j)} = \tilde{E}\tilde{\pi}^{(j)}$. We arrange the observed sales and price changes as

$$D = \left[\begin{array}{ccc} d^{(1)} & \cdots & d^{(N)} \end{array}\right], \quad \tilde{\Pi} = \left[\begin{array}{ccc} \tilde{\pi}^{(1)} & \cdots & \tilde{\pi}^{(N)} \end{array}\right].$$

To obtain the maximum likelihood estimate (MLE) for E and $d^{\rm nom}$, we solve

$$\text{maximize} \quad f(\tilde{E}) = \tfrac{1}{N}\textstyle\sum_{j=1}^{N}\sum_{i=1}^{n}(D \circ \tilde{E}\tilde{\Pi} - \exp(\tilde{E}\tilde{\Pi}))_{ij} \tag{3}$$

where $\tilde{E} \in \mathbf{R}^{n\times n+1}$ is the variable and exp is taken elementwise, and $\circ$ denotes the Hadamard or elementwise product. We extract the MLE for E and $d^{\rm nom}$ from the solution $\tilde{E}^\star$. This is a convex optimization problem, as the first part of the objective as affine, and the second is concave [BV04].

3.2 Low-rank plus diagonal elasticity

In typical consumer data, the squared number of goods exceeds the number of data points, so estimating a full-rank n by n elasticity matrix is not tractable. Instead, we consider the low-rank plus diagonal structure introduced in §1.2,

$$E = BC^T + \mathbf{diag}(s), \tag{4}$$

where $B, C \in \mathbf{R}^{n\times r}$ and $s \in \mathbf{R}^n$, so the number of estimated parameters is $2nr+n = n(2r+1)$. When $r \ll n$, this is much less than the n^2 parameters of a full-rank elasticity matrix. Imposing this structure on the elasticity matrix makes parameter estimation tractable, and more importantly improves the out-of-sample prediction of demand from prices.

Low-rank plus diagonal MLE. We add (4) as a constraint and a quadratic regularizer to the MLE problem (3), and solve

$$\begin{array}{ll} \text{maximize} & \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{n}(D \circ \tilde{E}\tilde{\Pi} - \exp(\tilde{E}\tilde{\Pi}))_{ij} - \frac{\lambda}{2}(\|B\|_F^2 + \|C\|_F^2) \\ \text{subject to} & \tilde{E}_{1:n} = BC^T + \mathbf{diag}(s). \end{array} \tag{5}$$

Here, the variables are $\tilde{E} \in \mathbf{R}^{n\times n+1}$ (the subscript $1:n$ denotes the first n columns), $B, C \in \mathbf{R}^{n\times r}$, and $s \in \mathbf{R}^n$. The value of $\lambda > 0$ controls the regularization strength. In fact, this regularizer has an equivalent effect to the dual spectral norm penalty $\lambda\|M\|_*$ (the sum of singular values), when imposing $E = M + \mathbf{diag}(s)$ with $\mathbf{rank}(M) \leq r$ [UHZB16]. Therefore, we can view r as an upper bound on the rank of the resulting low-rank part, and λ as a tuning knob for the resulting rank.

This is not a convex optimization problem, since the equality constraint involves the nonlinear expression BC^T. However, when either B or C are fixed, the problem becomes convex [BV04], which we will exploit.

4 Fitting methods

We describe three methods for solving problem (5). In §4.1, we describe alternating maximization, where we alternate between fixing B or C, and solving a respective problem that is convex in the remaining variables. In §4.2, we derive an expression for the gradient of the objective of problem (5) with respect to B, C, s, and $\log d^{\text{nom}}$, which can be evaluated efficiently, in a vectorized way. The third method is described in §4.3 and applies a general nonlinear programming solver to problem (5) as is, with variables $\tilde{E}$, B, C, and s.

4.1 Alternating maximization

Problem (4) becomes convex when fixing either B or C, since then the constraint becomes an affine equality constraint (and the maximized objective was concave already) [BV04]. Therefore, we consider an alternating maximization scheme where we alternate between computing solutions to variants of (5) with B or C fixed, respectively.

We initialize $\hat{C}$ randomly, away from zero. Then, we take the variant of (5) with C declared a parameter, assign $\hat{C}$ to C and obtain a solution with a convex optimization solver, from which we take $\hat{B}$. Next, we take the variant of (5) with B declared a parameter, assign $\hat{B}$ to B, solve the problem, and obtain $\hat{C}$ from the solution. We repeat this procedure until the objective value converges.

This alternating maximization scheme is an ascent method. Since each sub-problem is a convex maximization problem, the respective optimal value can be computed effectively, and will be at least as high as the value at the initialization (from the previous iteration).

CVXPY specification. Figure 1 shows how the procedure is implemented in the domain-specific modeling language for convex optimization CVXPY [DB16, AVDB18]. In lines 4–22, the two respective problems are modeled with CVXPY. In lines 10 and 11, `Parameter` objects are declared to allow for efficient updates to the respective problems when the values of B and C change. In lines 25–30, five iterations of alternating maximization are run.

4.2 Gradient ascent

We run a simple gradient ascent method, where we take steps along the gradients of the objective in (5), denoted by $g(X)$, with respect to $X = \begin{bmatrix} B & C & s & \log d^{\text{nom}} \end{bmatrix} \in \mathbf{R}^{n\times 2r+2}$. We use a simple line search to guarantee that the algorithm is an ascent method. If the likelihood is improved with a gradient step of the current step size (denoted by α), we use it for the current iteration and increase it by a constant factor $\gamma > 1$ for the next iteration. Otherwise, we repeatedly shrink the step size by a constant factor $\eta > 1$ until the likelihood is improved. The method is given in algorithm 1.

Algorithm 1 Gradient ascent

Initialize X^0, α^0, $k = 0$
repeat
  $\hat{X} = X^k + \alpha^k \nabla g(X)$ ▷ tentative update
  if $g(\hat{X}) > g(X^k)$ **then**
    $X^{k+1} = \hat{X}$, $\alpha^{k+1} = \gamma\alpha^k$ ▷ accept update and increase step size
  else
    $\alpha^k \leftarrow \alpha^k/\eta$, go to step 3 ▷ shrink step size and re-evaluate
  end if
  $k \leftarrow k + 1$
until $\|\nabla g(X^k)\|_F \leq \epsilon^{\text{rel}}\|X^k\|_F + \epsilon^{\text{abs}}$

Initialization. We draw the entries of B^0 and C^0 IID from $\mathcal{N}(0, 1/(n\sqrt{r}))$, set $s^0 = -\mathbf{1}$, and $(\log d^{\text{nom}})^0 = 0$. The algorithm is not particularly dependent on the initial step size α^0 and the line search parameters γ and η. Reasonable choices are, *e.g.*, $\alpha^0 = 1.0$, $\gamma = 1.2$, and $\eta = 1.5$.

```
import cvxpy as cp

# variables
Etilde = cp.Variable((n, n + 1))
B = cp.Variable((n, r))
C = cp.Variable((n, r))
s = cp.Variable(n)

# parameters
Bp = cp.Parameter((n, r))
Cp = cp.Parameter((n, r))

# objectives and constraints
f = cp.sum(cp.multiply(D, Etilde @ Pitilde) - cp.exp(Etilde @ Pitilde)) / N
obj_over_B = cp.Maximize(f - (lam / 2) * cp.sum_squares(B))
obj_over_C = cp.Maximize(f - (lam / 2) * cp.sum_squares(C))
constr_over_B = [Etilde[:, :-1] == B @ Cp.T + cp.diag(s)]
constr_over_C = [Etilde[:, :-1] == Bp @ C.T + cp.diag(s)]

# problems
prob_over_B = cp.Problem(obj_over_B, constr_over_B)
prob_over_C = cp.Problem(obj_over_C, constr_over_C)

# initialize and solve
Cp.value = ...
for _ in range(5):
    prob_over_B.solve()
    Bp.value = B.value
    prob_over_C.solve()
    Cp.value = C.value
```

Figure 1: Alternating maximization with CVXPY. The dimensions `n`, `r`, `N`, and the data `D`, `Pitilde`, `lam` are given.

Gradient computation. The gradient of the objective in (5) with respect to $\tilde{E}$ is

$$\Delta = (1/N)(D - \exp(\tilde{E}\tilde{\Pi}))\tilde{\Pi}^T = (1/N)(D\tilde{\Pi}^T - \exp(\tilde{E}\tilde{\Pi})\tilde{\Pi}^T).$$

The second expression is faster to compute for changing values of $\tilde{E}$ when $N > n$, since $D\tilde{\Pi}^T$ needs to be computed only once ahead of time. We apply the chain rule to obtain the gradient with respect to X,

$$\nabla g(X) = \left[\begin{array}{cccc} \Delta_{1:n}C - \lambda B & \Delta_{1:n}^T B - \lambda C & \mathbf{diag}(\Delta_{1:n}) & \Delta_{n+1} \end{array} \right],$$

where subscript $n+1$ denotes column $n+1$. We can compute $\nabla g(X)$ very fast, as it involves mostly BLAS level 3 computations [DDCHD90].

Stopping criterion. We stop the algorithm as soon as the termination criterion

$$\|\nabla g(X^k)\|_F \leq \epsilon^{\text{rel}}\|X^k\|_F + \epsilon^{\text{abs}}$$

with $\epsilon^{\text{rel}}, \epsilon^{\text{abs}} > 0$ is met. Typical values for ϵ^{rel} and ϵ^{abs} range between 10^{-6} and 10^{-2}.

Python specification. Figure 2 shows a Python implementation of algorithm 1. The variable, gradient, and step size are initialized in lines 3–6, and the actual algorithm is implemented via two simple loops in lines 8–17.

4.3 Nonlinear programming

One can view the constraint in problem (5) as an instance of general nonlinear (and differentiable) constraints and apply nonlinear programming (NLP) methods, which are local methods as they use gradients or (approximate) Hessians [Ber99, LN89, NW06, KT13]. Well-known NLP solver implementations are the open-source IPOPT [WB06] and the proprietary KNITRO [BNW06].

CVXPY specification. Figure 3 shows how problem (5) is solved via the CVXPY NLP interface [CZNB25]. The nonconvex problem is modeled in lines 4–13 with CVXPY. After variable initialization in line 15, the problem is solved in line 16, where `nlp=True` informs CVXPY that the problem should be treated as an NLP.

5 Evaluation

There are several ways to assess the quality of an elasticity estimate. We focus on two different types of metrics: the first statistical, based on out-of-sample log-likelihood, and the second based on a typical end-use, realized profit after optimally pricing the products using the elasticity model. In the first, we compute the log-likehood of the observed demand under the Poisson model with the estimated price elasticity matrix (and nominal demand).

```
import numpy.linalg as la

X = ...
grad_X = grad(X)
g_values = [g(X)]
alpha = 1.0

while la.norm(grad_X) > eps_rel * la.norm(X) + eps_abs:
    while True:
        X_hat = X + alpha * grad_X
        if g(X_hat) > g_values[-1]:
            X = X_hat
            grad_X = grad(X)
            g_values.append(g(X))
            alpha *= 1.2
            break
        alpha /= 1.5
```

Figure 2: Gradient ascent in Python. The initial value of `X`, the Python functions `g` and `grad`, and the tolerances `eps_rel` and `eps_abs` are given.

```
import cvxpy as cp

# variables
Etilde = cp.Variable((n, n + 1))
B = cp.Variable((n, r))
C = cp.Variable((n, r))
s = cp.Variable(n)

# objective, constraint, and problem
f = cp.sum(cp.multiply(D, Etilde @ Pitilde) - cp.exp(Etilde @ Pitilde)) / N
obj = cp.Maximize(f - (lam / 2) * (cp.sum_squares(B) + cp.sum_squares(C)))
constr = [Etilde[:, :-1] == B @ C.T + cp.diag(s)]
prob = cp.Problem(obj, constr)

# solve problem
Etilde.value, B.value, C.value, s.value = ...
prob.solve(nlp=True)
```

Figure 3: Nonlinear programming with CVXPY. The dimensions `n`, `r`, `N`, and the data `D`, `Pitilde`, `lam` are given.

This provides a direct measure of how well the estimator captures the statistical structure of the demand. In the second, we solve a profit maximization problem given the elasticity estimate, and record the realized profit, given a simulated true demand function.

These metrics offer complementary insights. Log-likelihood quantifies how well the estimator aligns with the observed pattern of price and demand changes. However, it relies on the assumption that the demand follows a Poisson distribution, which may or may not hold in practice. Assessing the elasticity estimate in terms of profit does not require such a distributional assumption and instead evaluates the economic usefulness of the estimator by measuring its profit impact on pricing decisions. At the same time, the profit after pricing also depends on design choices, such as limits on price changes, making attribution less direct, not to mention that we compute it only in simulation. Fortunately we will see that these two different evaluations of proposed models lead to very similar conclusions.

5.1 Log-likelihood

The average log-likelihood is

$$\mathcal{L}(\tilde{E}; D, \tilde{\Pi}) = \frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{n} (D \circ \tilde{E}\tilde{\Pi} - \exp(\tilde{E}\tilde{\Pi}) - \log(D!))_{ij} \tag{6}$$

where log and factorial are taken elementwise and we consider data where D contains positive integers (counts of goods sold).

We perform K-fold cross-validation. We split the data $D, \tilde{\Pi}$ into K equally sized column slices, or folds,

$$D = \left[\begin{array}{ccc} D^{(1)} & \cdots & D^{(K)} \end{array} \right], \quad \tilde{\Pi} = \left[\begin{array}{ccc} \tilde{\Pi}^{(1)} & \cdots & \tilde{\Pi}^{(K)} \end{array} \right].$$

We then fit K models $\tilde{E}^{(1)}, \ldots, \tilde{E}^{(K)}$, leaving out one fold each time, for which we then compute the respective log-likelihood. We report the average log-likelihood over all folds,

$$\frac{1}{K} \sum_{k=1}^{K} \mathcal{L}(\tilde{E}^{(k)}; D^{(k)}, \tilde{\Pi}^{(k)}). \tag{7}$$

5.2 Pricing performance

For a given estimate $\tilde{E}$, we extract E and d^{nom}, from which we compute the nominal revenues $r^{\mathrm{nom}} = d^{\mathrm{nom}} \circ p^{\mathrm{nom}}$ and nominal costs $\kappa^{\mathrm{nom}} = d^{\mathrm{nom}} \circ c$, given the nominal prices p^{nom} and the vector of cost per unit c. We then solve the (nonconvex) profit maximization problem from [SB25, §2.4]

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} \left(r_i^{\mathrm{nom}} \exp(\delta_i + \pi_i) - \kappa_i^{\mathrm{nom}} \exp(\delta_i) \right) \\ \text{subject to} & \delta = E\pi, \quad \pi^{\min} \preceq \pi \preceq \pi^{\max}, \end{array} \tag{8}$$

where $\delta, \pi \in \mathbf{R}^n$ are the variables and $\pi^{\min}, \pi^{\max} \in \mathbf{R}^n$ are given limits on the price changes.

Again, we perform K-fold cross-validation. We solve problem (8) K times, with the data taken from $\tilde{E}^{(1)}, \ldots, \tilde{E}^{(K)}$, respectively. From the respective solution $\pi^{(k)}$, we simulate the profit

$$\mathcal{P}(\pi^{(k)}) = \sum_{i=1}^{n} \left(r^{\text{nom,syn}} \circ \exp((E^{\text{syn}} + I)\pi^{(k)}) - \kappa^{\text{nom,syn}} \circ \exp(E^{\text{syn}}\pi^{(k)}) \right)_i,$$

where superscript syn denotes synthetic data, *i.e.*, we take a synthetic elasticity matrix E^{syn} and nominal demand $d^{\text{nom,syn}}$ (see §6.1), and set $r^{\text{nom,syn}} = d^{\text{nom,syn}} \circ p^{\text{nom}}$, $\kappa^{\text{nom,syn}} = d^{\text{nom,syn}} \circ c$. We report the average over all folds,

$$\frac{1}{K} \sum_{k=1}^{K} \mathcal{P}(\pi^{(k)}). \tag{9}$$

6 Results

We fit elasticity matrices by applying alternating maximization, gradient ascent, and nonlinear programming, as described in §4. In gradient ascent, we use $\gamma = 1.2$ and $\eta = 1.5$ for the line search and termination tolerances $\epsilon^{\text{rel}} = \epsilon^{\text{abs}} = 10^{-3}$. We use the same termination tolerances for alternating maximization and nonlinear programming. The results are not particularly dependent on the tolerance value. To solve the convex subproblems of alternating maximization, we use the convex optimization solver MOSEK [MOS25]. For nonlinear programming, we use the open-source NLP solver IPOPT [WB06]. We interface with both solvers via CVXPY and use their respective default settings. We run the experiments on an Apple M1 Pro.

6.1 Synthetic data

Data set. We generate the entries of $p^{(1)}, \ldots, p^{(N)}$ uniformly and IID from $[1, 2]$ and set

$$\log p^{\text{nom}} = (1/N) \sum_{j=1}^{N} \log p^{(j)},$$

where log is taken elementwise. Then, we compute the log price changes

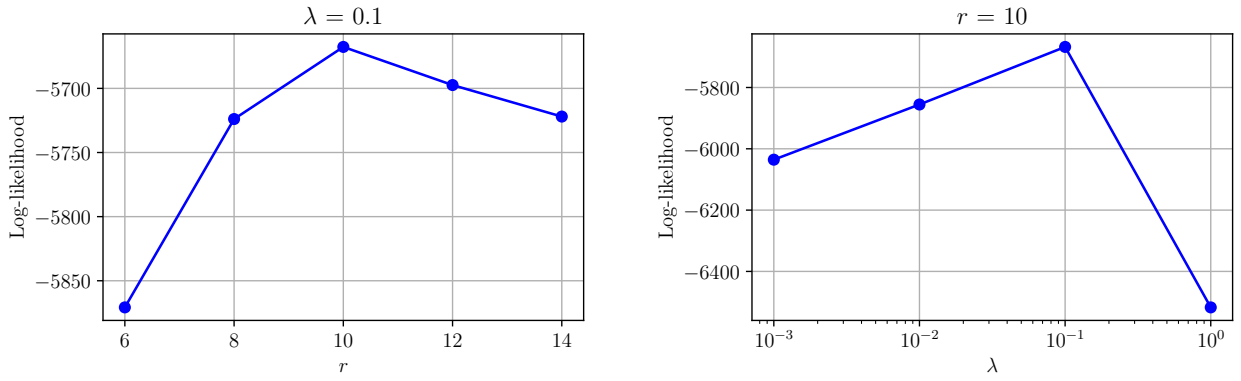
$$\pi^{(j)} = \log p^{(j)} - \log p^{\text{nom}}, \quad j = 1, \ldots, N$$

and assemble them in $\tilde{\Pi}$ as described in §3.1.

We construct a synthetic elasticity matrix

$$E^{\text{syn}} = B^{\text{syn}}(C^{\text{syn}})^T + \mathbf{diag}(s^{\text{syn}}),$$

where the entries of $B^{\text{syn}}, C^{\text{syn}} \in \mathbf{R}^{n \times r^{\text{syn}}}$ are generated IID from $\mathcal{N}(0, 0.1)$, and the entries of s^{syn} are generated uniformly and IID from $[-5, -1]$.



(a) Log-likelihood for varying r at $\lambda = 0.1$.

(b) Log-likelihood for varying λ at $r = 10$.

Figure 4: Log-likelihood for varying r and λ, with synthetic data.

Ultimately, we set $d^{\text{nom,syn}} = \mathbf{1}$ so $\log d^{\text{nom,syn}} = 0$ and the Poisson rates are

$$\lambda^{(j)} = \exp(E^{\text{syn}}\pi^{(j)}), \quad j = 1, \ldots, N.$$

We then generate the columns $d^{(j)}$ of D from Poisson distributions with respective rates $\lambda^{(j)}$. We generate the entries of the cost vector c IID from $[0.8, 1.2]$.

Results. For $n = 100$, $N = 200$, and $r^{\text{syn}} = 10$, we compare the time it takes to solve problem (5) where $r = 10$ and $\lambda = 0.1$, with alternating maximization (AM), gradient ascent, and nonlinear programming (NLP), as described in §4. All three methods converge to the same objective value of -27.1 (up to the relative tolerance). Table 1 shows the respective solve times. Gradient ascent is two to three orders of magnitude faster than NLP and AM,

AM	**Gradient ascent**	**NLP**
230 s	0.202 s	49.2 s

Table 1: Solve times with AM, gradient ascent, and NLP.

respectively, due to its use of efficient (vectorized) gradient computations. The other two mehods have much overhead, *e.g.*, many exponential cones when using AM [MOS25]. We use gradient ascent for the remaining analyses.

We cross-validate our elasticity estimator obtained by solving problem (5) for

$$r \in \{6, 8, 10, 12, 14\}, \quad \lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}.$$

We report the cross-validated log-likelihood (7) for $K = 5$ folds. We obtain the highest log-likelihood at $r = 10$ (equaling r^{syn}, as expected) and $\lambda = 0.1$. Figure 4 shows the log-likelihood over r and λ, respectively. For the optimal configuration, figure 5 compares the elasticity estimate $E^{\star}$ to the true (synthetic) elasticity matrix E^{syn}. We observe that our

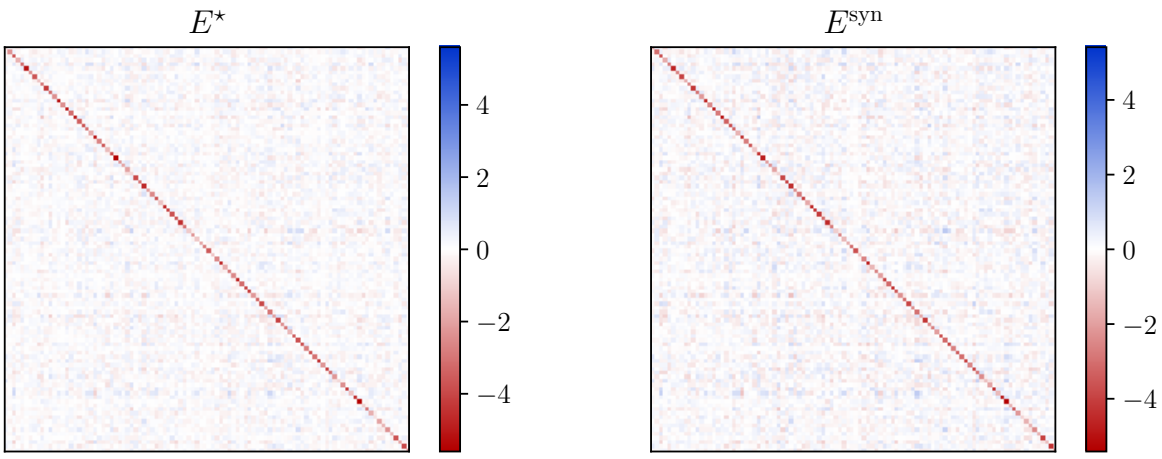


Figure 5: Estimated elasticity matrix $E^\star$ for $\lambda = 0.1$ and $r = 10$ and true (synthetic) elasticity matrix E^{syn}.

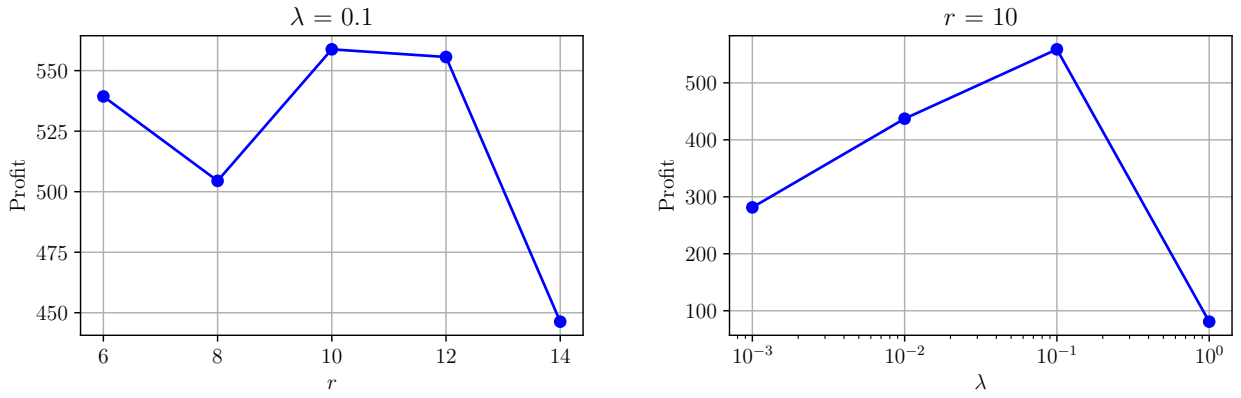
method recovers the true elasticity matrix, up to slightly shrunk cross-elasticities due to regularization.

We also compute the cross-validated pricing performance (9), *i.e.*, the simulated profit after solving the optimal pricing problem (8) with our elasticity estimate. We take $\pi^{\min} = \log(0.8)\mathbf{1}$ and $\pi^{\max} = \log(1.2)\mathbf{1}$ in problem (8), *i.e.*, we allow price changes between $\pm 20\%$. Consistent with the log-likelihood, the highest pricing performance is obtained for $r = 10$ and $\lambda = 0.1$, as shown in figure 6. The maximum cross-validated profit is 558, compared to a mean of 662 when solving problem (8) with the true elasticity matrix (and true nominal demand).

6.2 Real data

Data set. We use the Dominick's Finer Foods (DFF) dataset provided by the Kilts Center for Marketing at the University of Chicago Booth School of Business [Jam97]. The dataset contains scanner data from purchases at DFF stores in the Chicago area from the 1990s, aggregated to weekly time intervals. Unlike more recent datasets, the DFF dataset is publicly available to anyone.

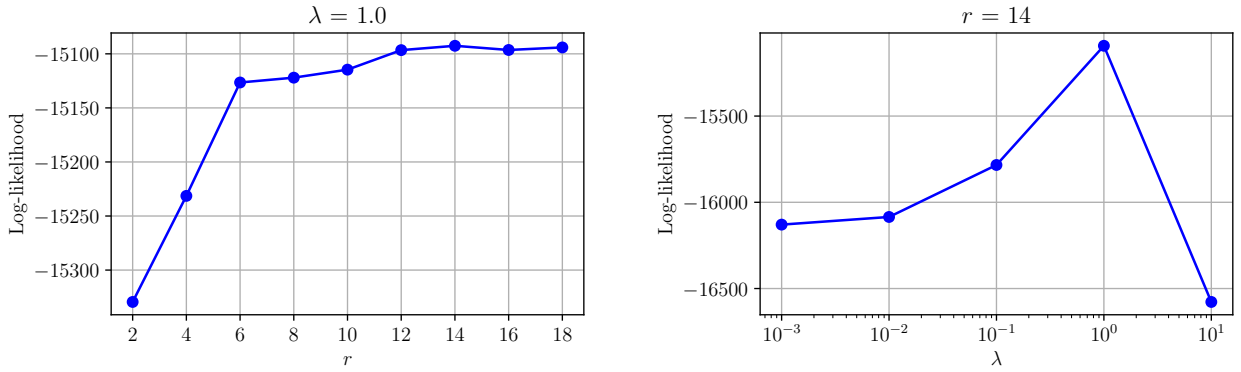
We consider $n = 20$ of the most frequently sold beverages, of which $N = 360$ complete data points (out of 400) are available; we only consider weeks where all 20 beverages have been sold, and therefore have a known price. (In scanner data, the price during a certain period is unknown when the product is not sold in that period.) We accept this selection bias as we demonstrate our numerical fitting method. We take the nominal log-prices as the average of the log-prices.



(a) Simulated profit for varying r at $\lambda = 0.1$.

(b) Simulated profit for varying λ at $r = 10$.

Figure 6: Pricing performance for varying r and λ, with synthetic data.



(a) Log-likelihood for varying r at $\lambda = 1.0$.

(b) Log-likelihood for varying λ at $r = 14$.

Figure 7: Log-likelihood for varying r and λ, with real data.

Results. We consider $r \in \{2, 4, \ldots, 18\}$ and $\lambda \in \{10^{-3}, 10^{-2}, \ldots, 10^{1}\}$. The cross-validated log-likelihood (7) is shown for $K = 5$ folds in figure 7. We obtain the highest log-likelihood at $r = 14$ and $\lambda = 1.0$. The median relative error of the predicted (mean) demand with respect to the true demand, over all products and time steps, is 34% (cross-validated over the same five folds). The median relative error due to the (predicted) Poisson noise is 12%, *i.e.*, about one third of the overall error.

Acknowledgments

The authors thank Alexander Thebelt, Marco Tacke, and Lutz Gruber for inspiring this work.

References

[AV18] M. Armstrong and J. Vickers. Multiproduct pricing made simple. *Journal of Political Economy*, 126(4):1444–1471, 2018.

[AVDB18] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[Bar64] A. Barten. Consumer demand functions under conditions of almost additive preferences. *Econometrica*, 32(1):1–38, 1964.

[Ber94] S. Berry. Estimating discrete-choice models of product differentiation. *The Rand Journal of Economics*, 25(2):242–262, 1994.

[Ber99] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, Nashua, 1999.

[BHH12] M. Burda, M. Harding, and J. Hausman. A Poisson mixture model of discrete choice. *Journal of Econometrics*, 166(2):184–203, 2012.

[BNW06] R. Byrd, J. Nocedal, and R. Waltz. KNITRO: An integrated package for nonlinear optimization. In *Large-Scale Nonlinear Optimization*, pages 35–59. Springer, Boston, 2006.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 2004.

[BW06] C. Broda and D. Weinstein. Globalization and the gains from variety. *The Quarterly Journal of Economics*, 121(2):541–585, 2006.

[CJL75] L. Christensen, D. Jorgenson, and L. Lau. Transcendental logarithmic utility functions. *The American Economic Review*, 65(3):367–383, 1975.

[CWA09] S. Coxe, S. West, and L. Aiken. The analysis of count data: A gentle introduction to Poisson regression and its alternatives. *Journal of Personality Assessment*, 91(2):121–136, 2009.

[CZNB25] D. Cederberg, W. Zhang, P. Nobel, and S. Boyd. Disciplined nonlinear programming, 2025. Working paper, available at https://stanford.edu/~boyd/papers/dnlp.html.

[DB16] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DDCHD90] J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17, 1990.

[DM80a] A. Deaton and J. Muellbauer. An almost ideal demand system. *The American Economic Review*, 70(3):312–326, 1980.

[DM80b] A. Deaton and J. Muellbauer. *Economics and Consumer Behavior*. Cambridge University Press, Cambridge, 1980.

[Fel91] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. John Wiley & Sons, New York, 1991.

[FF92] E. Fama and K. French. The cross-section of expected stock returns. *The Journal of Finance*, 47(2):427–465, 1992.

[FF93] E. Fama and K. French. Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1):3–56, 1993.

[FL25] T. Fally and E. Ligon. Consumer demand with price aggregators and low-rank cross-price effects, 2025. Working paper, available at https://fally.are.berkeley.edu/Papers/Aggregators.pdf.

[FLSL16] K. Ferreira, B. Lee, and D. Simchi-Levi. Analytics for an online retailer: Demand forecasting and price optimization. *Manufacturing & Service Operations Management*, 18(1):69–88, 2016.

[GVR94] G. Gallego and G. Van Ryzin. Optimal dynamic pricing of inventories with stochastic demand over finite horizons. *Management Science*, 40(8):999–1020, 1994.

[HH14] M. Hayat and M. Higgins. Understanding Poisson regression. *Journal of Nursing Education*, 53(4):207–215, 2014.

[HP17] T. Hastie and D. Pregibon. Generalized linear models. In *Statistical Models in S*, pages 195–247. Routledge, New York, 2017.

[Jam97] James M. Kilts Center for Marketing, University of Chicago Booth School of Business. Dominick's Finer Foods dataset, 1997. Available at https://www.chicagobooth.edu/research/kilts/research-data/dominicks.

[JNS13] P. Jain, P. Netrapalli, and S. Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, pages 665–674, 2013.

[JOP+23] K. Johansson, M. Ogut, M. Pelger, T. Schmelzer, and S. Boyd. A simple method for predicting covariance matrices of financial returns. *Foundations and Trends in Econometrics*, 12(4):324–407, 2023.

[KT13] H. Kuhn and A. Tucker. Nonlinear programming. In *Traces and Emergence of Nonlinear Programming*, pages 247–258. Springer, Basel, 2013.

[LN89] D. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[LP09] A. Lewbel and K. Pendakur. Tricks with Hicks: The EASI demand system. *American Economic Review*, 99(3):827–863, 2009.

[LP20a] M. Lettau and M. Pelger. Estimating latent asset-pricing factors. *Journal of Econometrics*, 218(1):1–31, 2020.

[LP20b] M. Lettau and M. Pelger. Factors that fit the time series and cross-section of stock returns. *The Review of Financial Studies*, 33(5):2274–2325, 2020.

[MCWG95] A. Mas-Colell, M. Whinston, and J. Green. *Microeconomic Theory*, volume 1. Oxford University Press, New York, 1995.

[MOS25] MOSEK ApS. The mosek optimizer api manual. version 11.0, 2025. Available at https://docs.mosek.com/latest/capi/index.html.

[Mue75] J. Muellbauer. Aggregation, income distribution and consumer demand. *The Review of Economic Studies*, 42(4):525–543, 1975.

[NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, 2006.

[SB25] M. Schaller and S. Boyd. A note on optimal product pricing. *arXiv preprint arXiv:2511.06156*, 2025.

[ST06] J. Silva and S. Tenreyro. The log of gravity. *The Review of Economics and Statistics*, 88(4):641–658, 2006.

[The65] H. Theil. The information approach to demand analysis. *Econometrica*, 33(1):67–87, 1965.

[UHZB16] M. Udell, C. Horn, R. Zadeh, and S. Boyd. Generalized low rank models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.

[WB06] A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[Wil93] R. Wilson. *Nonlinear Pricing*. Oxford University Press, New York, 1993.