WEB APPENDICES

A: Kernel Degree Selection

As described in the body of the paper, not all parameters of the Matérn kernel are consistently estimable. Hence, we follow common practice and fix the degree parameter ν to a half-integer value, or to ∞, which corresponds to the squared exponential kernel. By fixing ν to a half-integer, the functional form of the kernel reduces to a product of a polynomial term and an exponential term, which facilitates computation, compared to the original Bessel function formulation. Specifically, we consider the values $\nu = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}$. This follows the advice of Rasmussen and Williams (2005), who argue that values of $\nu > \frac{5}{2}$ are difficult to distinguish from ∞, given typical data sizes. In the GPDH settings, this lack of distinction is even more the case, as the GPs are being specified several levels away from the data, as governing the parameters of a latent utility (or the mean of those parameters).

The ν parameter controls the level of differentiability of the function draws, as illustrated in Figure 1. Thus, if a less smooth process is desired, or theorized a priori, the researcher may choose to use a lower value ν (e.g., $\nu = 1/2$). Alternatively, cross-validation may be used to set the value of ν. In our Application I, we find little difference both in terms of fit and prediction across different values of the degree parameter. As an example, we plot a comparison of fit and forecasting accuracy across ν values in Figure 2 for the Peanut Butter category. The value $\nu = 3/2$ does marginally better in forecasting tasks, and imposes less stringent assumptions on smoothness, assuming only once differentiable function draws. Hence, we use $\nu = 3/2$ as our primary specification throughout the paper.

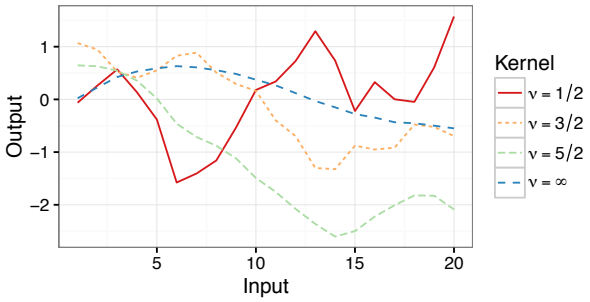


Figure 1: The impact of the choice of the degree parameter on the level of differentiability or "smoothness" of the function draws.

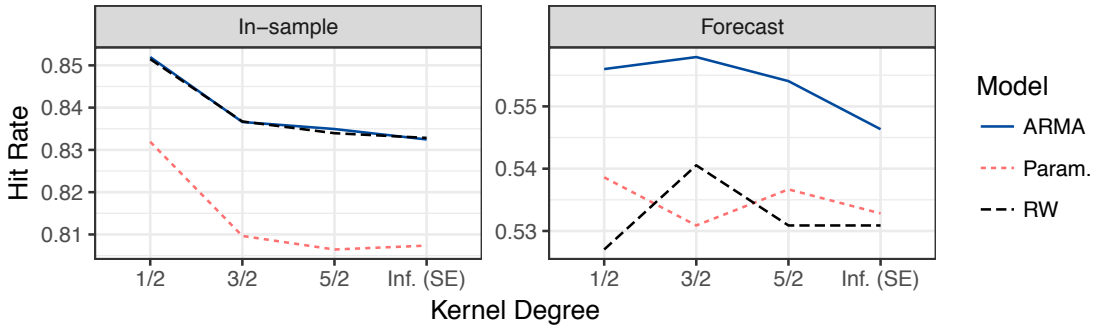


Figure 2: A comparison of the fit and forecasting ability of the GPDH-logit model across different values of the kernel degree parameter ν, across all non-GP mean models, on the peanut butter data. We limit consideration to non-GP mean models because of our assumption that, when using a GP mean, the degree parameter is the same as in the GPDH kernel.

B: Estimation Details

In this section, we give the explicit forms of the densities used in Equations 14 and 17 for estimating the models in our two applications.

Application 1. The joint density for the full model given in Equation 14 is reproduced here:

$$p(y, \beta, \mu, \alpha, \phi|X) = \prod_{m=1}^{M} p(y_m|X_m, \{\beta_{i_m p}(t_m)\}_{p=1}^{P}) \times \prod_{i=1}^{I}\prod_{p=1}^{P} p(\beta_{ip}(t)|\mu_p(t), \boldsymbol{\phi}_p)\, p(\mu_p(t)|\boldsymbol{\alpha}_p)\, p(\boldsymbol{\phi}_p)\, p(\boldsymbol{\alpha}_p).$$

The individual-level model is a multinomial logit such that

$$p(y_m = j|X_m, \{\beta_{i_m p}(t_m)\}_{p=1}^{P}) = \frac{\exp\left(\sum_{p=1}^{P} \beta_{i_m p}(t_m)\, x_{i_m pjt}\right)}{\sum_{\ell=1}^{J} \exp\left(\sum_{p=1}^{P} \beta_{i_m p}(t_m)\, x_{i_m p\ell t}\right)}.$$

The GP heterogeneity specification is given by

$$p(\beta_{ip}(\boldsymbol{t})|\mu_p(t), \boldsymbol{\phi}_p) = \text{MVN}(\mu_p(\boldsymbol{t}), K(\boldsymbol{t}, \boldsymbol{t}; \boldsymbol{\phi}_p)),$$

where $\boldsymbol{t} = \{1, 2, \ldots, T\}$ represents the vector of time points on which the GP is defined. The probabilistic representation for $p(\mu_p(t)|\boldsymbol{\alpha}_p)$ depends upon the specific mean-model used. For the ARMA(1) specification, we have

$$p(\mu_p(t)|\boldsymbol{\alpha}_p) = N(\alpha_{0p} + \alpha_{1p}\, \mu_{pt-1} + \alpha_{2p}\, \zeta_{pt-1}, \tau_p^2).$$

Finally, $p(\boldsymbol{\phi}_p)$ represents the PC prior, such that

$$p(\eta, \kappa) = \frac{1}{2}\lambda_1 \lambda_2 \kappa^{-1/2} \exp(-\lambda_1\sqrt{\kappa} - \lambda_2 \eta); \quad \lambda_1 = -\log \alpha_\rho \sqrt{\frac{\rho_0}{\sqrt{8\nu}}}, \quad \lambda_2 = \frac{\log \alpha_\eta}{\eta_0}.$$

The last term, $p(\boldsymbol{\alpha}_p)$, represents the prior over the parameters in the specific mean-model used. We chose appropriate diffuse priors for these parameters.

Application 2. The joint density is given by

$$p(w|\boldsymbol{\beta},\boldsymbol{\mu},\boldsymbol{\nu}) = \prod_{i=1}^{N}\prod_{t=t_i^{\min}}^{T}\prod_{m=1}^{M_{it}} p(w_{itm}\,|\,\boldsymbol{\beta}_i(t),\boldsymbol{\nu}) \prod_{d=1}^{D-1} p(u_{id}(t)\,|\,\mu_d(t),\kappa_d,\eta_d)\times$$
$$p(\mu_d(t)|\boldsymbol{\kappa}_{0d},\boldsymbol{\eta}_{0d})p(\eta_d,\kappa_d,\boldsymbol{\eta}_{0d},\boldsymbol{\kappa}_{0d})p(\boldsymbol{\nu}_d).$$

In the above, $p(w_{itm} = v\,|\,\boldsymbol{\beta}_i(t),\boldsymbol{\nu}) = \sum_{d=1}^{D}\nu_{dv}\beta_{id}(t)$, where $\boldsymbol{\beta}_i(t) = (\beta_{i1}(t),\ldots,\beta_{iD}(t))$ and $\boldsymbol{\nu} = (\boldsymbol{\nu}_1,\ldots,\boldsymbol{\nu}_D)$. The GPDH term is given by

$$p(u_{id}(\boldsymbol{t})\,|\,\mu_d(t),\kappa_d,\eta_d) = \text{MVN}(\mu_d(\boldsymbol{t}), K_d(\boldsymbol{t},\boldsymbol{t},\eta_d,\kappa_d)),$$

where $\boldsymbol{t} = \{1,2,\ldots,T\}$ represents the vector of time points on which the GP is defined. The GP mean-model can be represented as

$$p(\mu_d(\boldsymbol{t}),\boldsymbol{\eta}_{0d},\boldsymbol{\kappa}_{0d}) = \text{MVN}(\mathbf{0}, K_{Long}(\boldsymbol{t},\boldsymbol{t}) + K_{Short}(\boldsymbol{t},\boldsymbol{t}) + K_{Per}(\boldsymbol{t},\boldsymbol{t})).$$

We used independent PC priors for each of the hyperparametes of the different GPs. The last term is given by $p(\boldsymbol{\nu}_d) = \text{Dirichlet}(\alpha)$.

C: Additional Choice Modeling Simulations

The simulations in the body of the paper assumed for simplicity that each individual purchased the same number of times. Here, we consider the more realistic case where individuals differ in the number of observations, with some spending often, and others very infrequently. Specifically, we vary four aspects of the data-generating process: (1) the number of time periods in the data; (2) the number of individuals in the data; (3) the minimum number of purchases needed for an individual to be included in the data; and (4), the variance of the number of purchases per person. We then study how the results of GPDH differ, in terms of fit, insights, and computation time.

To investigate the performance of GPDH as a function of all of these inputs, we ran a series of simulations, varying four aspects of the data generating process: the number of people ($N = 100, 200, 400$), the number of time periods ($T = 20, 40, 60$), the minimum number of spends per person ($m_{\text{min}} = 1, 3, 5$), and the variance of the number of spends per person over the entire time window. We simulated the number of spends, m_i, as $m_i = m_{\text{min}} + a_i$, $a_i \sim \text{Round}[\text{Gamma}(1, s)]$, where s is the scale parameter of the gamma, and varied $s = 2, 10, 20$. As before, we assume that the true data generating process is a GPDH multinomial logit with a GP mean model, and that there are three brands and a price variable.

Shrinkage Estimator. Just like all Bayesian approaches to modeling heterogeneity, GPDH can be viewed as a shrinkage estimator, wherein individuals' parameter trajectories are shrunk toward the mean trajectory. For consumers with very few purchases, their estimated trajectories mirror the mean function, in terms of both shape and magnitude, with large credible intervals. For consumers with many purchases, their trajectories are estimated to be closer to the true, data-generating trajectories. We illustrate this in Figure 3, for a simulation with 200 people, and a minimum number of purchases of 3. Because of these shrinkage properties, GPDH can still perform well, even when the number of purchases per person is small. In Table 1, we show that, across all levels of simulated data sparsity, GPDH achieves superior in-sample hit rates, compared

to the FO assumption.[1]

Computational Complexity. In the GP literature, it is well established that the computation time for GP-based models scales at $O(T^3)$, where T is the number of unique inputs (Rasmussen and Williams, 2005). In our applications, the number of inputs is always fixed at the number of months in the data. In GPDH, there are two additional aspects of the data size, besides the number of inputs, which may affect scalability: the number of individuals, and the number of observations per individual.

Across these simulations, we found that m_{min} and s did not have a consistent effect on computation time. We plot the effect of N and T in Figure 4. When the number of time periods is small ($T = 20$), we see there is little difference in the computation time with respect to N. However, for larger number of time periods, the increase in computation time from increasing N becomes pronounced. Holding N fixed, we see that the increase in computation time from changing T is non-linear, consistent with the $O(T^3)$ scaling of GPs generally.

[1]The full set of fit statistics is available from the authors upon request.

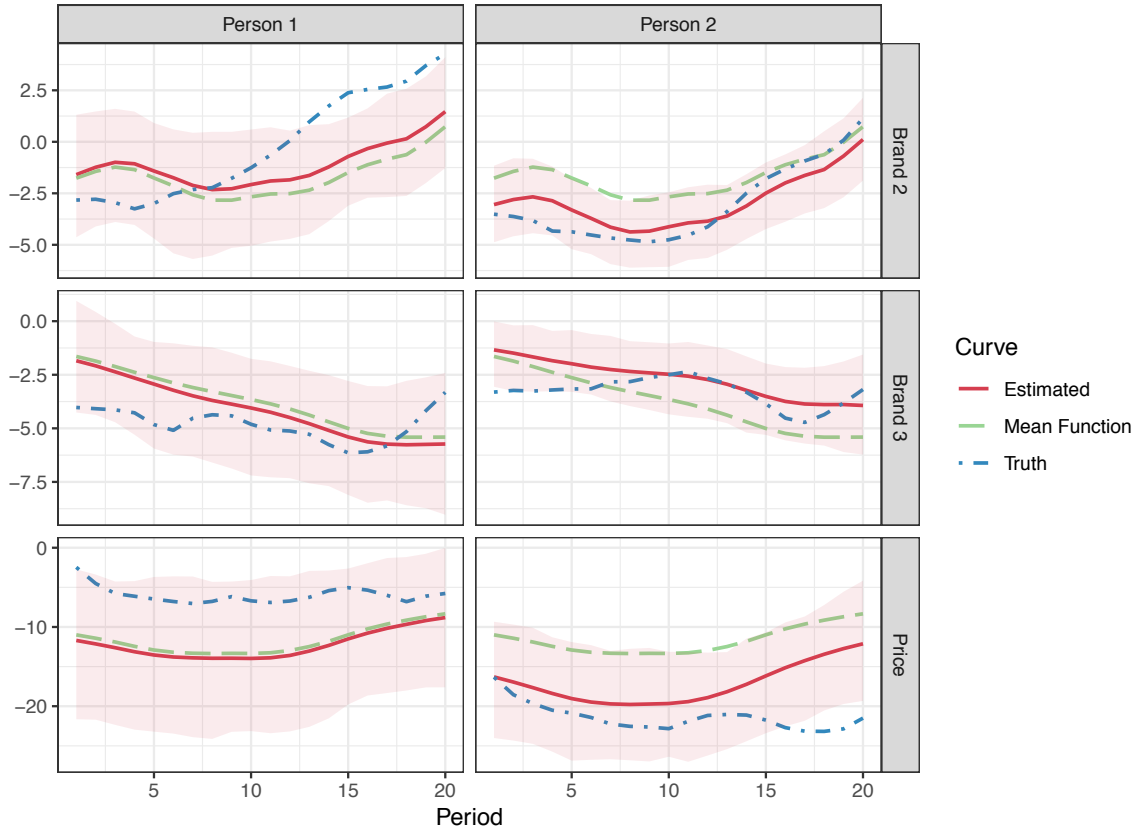


Figure 3: Shrinkage properties of GPDH: at left is a person with only 3 spends; at right is a person with 60 spends. We see that Person 1's estimated curves closely follow the mean function, while Person 2's estimated curves recover the truth, with some shrinkage toward the estimated mean function. The shaded bands are 95% credible intervals around the estimated individual-level trajectory.

Min Spends ($m_{\min}$)	1			3			5			Overall
Spend Variance (s)	2	10	20	2	10	20	2	10	20	
FO	.865	.849	.849	.870	.856	.842	.860	.848	.842	.853
GPDH	.896	.901	.902	.915	.911	.899	.900	.902	.891	.902

Table 1: Hit rate as a function of data sparsity, comparing the GPDH and FO specifications.

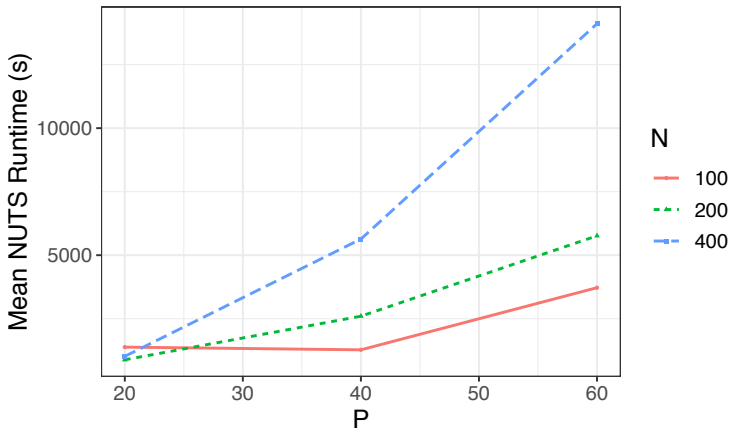


Figure 4: Average computation time, in seconds, for estimating GPDH on simulated data as a function of the number of people in the data N (the line colors and patterns), and the number of time periods T in the data (the x-axis). These results are averaged across m_{min} and s.

D: Full Fit Statistics (Application 1)

In this section, we present more fit statistics. As a whole, all fit statistics imply that GPDH significantly outperforms statistic heterogeneity, given the same mean model. In the main body of the paper, we presented several representative fit statistics in Figure 7. In this appendix, we also plot in Figure 5 the basic hit rates (accuracy) across specifications and data settings, and in Figure 6 the Watanabe-Akaike Information Criterion (WAIC), which is a Bayesian measure that measures model fit, penalizing for model complexity. We see that this measure again supports the idea that dynamic heterogeneity, as captured through GPDH, better describes the data, even taking into account the added complexity of the model. Interestingly, we find little difference in fit across mean models, except for a noted decrease in fit for the restrictive parametric model.

We also include here the full set of fit statistics, averaged across mean models, for all categories and heterogeneity specifications, in Table 2. Those statistics are based on the following counts, for a given brand b: True positives (TP_b) = the number of observations where the model predicted the consumer would choose brand b, and the consumer chose brand b ; False positives (FP_b) = the number of observations where the model predicted the consumer would choose brand b, but the consumer did not choose brand b; True negatives (TN_b) = the number of observations where the model did not predict the consumer would choose brand b, and the consumer did not choose brand b; and False negatives (FN_b) = the number of observations where the model did not predict the consumer would choose brand b, but the consumer chose brand b. From these, we compute the following statistics:

- Precision (Prec) - also called the hit rate, equal to $TP_b/(TP_b + FP_b)$,
- Sensitivity (Sens) - also called recall or the true positive rate, equal to $TP_b/(TP_b + FN_b)$,
- Specificity (Spec) - also called selectivity or the true negative rate, equal to $TN_b/(TN_b + FN_b)$,
- F1 - the harmonic mean of recall (Sensitivity) and precision.

Finally, we average these across brands in the following ways:

- Macro average: the average of each of the above rates. Intuitively, this aggregation treats all classes equally, ignoring potential class imbalance.
- Micro average: this aggregation computes the above statistics by summing over b at each step. Intuitively, this takes into account class imbalance, at the risk of showing good performance when one class dominates.
- Max: the max over b. Intuitively, this is the statistic for the class that was easiest to predict.
- Min: the min over b. Intuitively, this is the statistic for the class that was most difficult to predict.

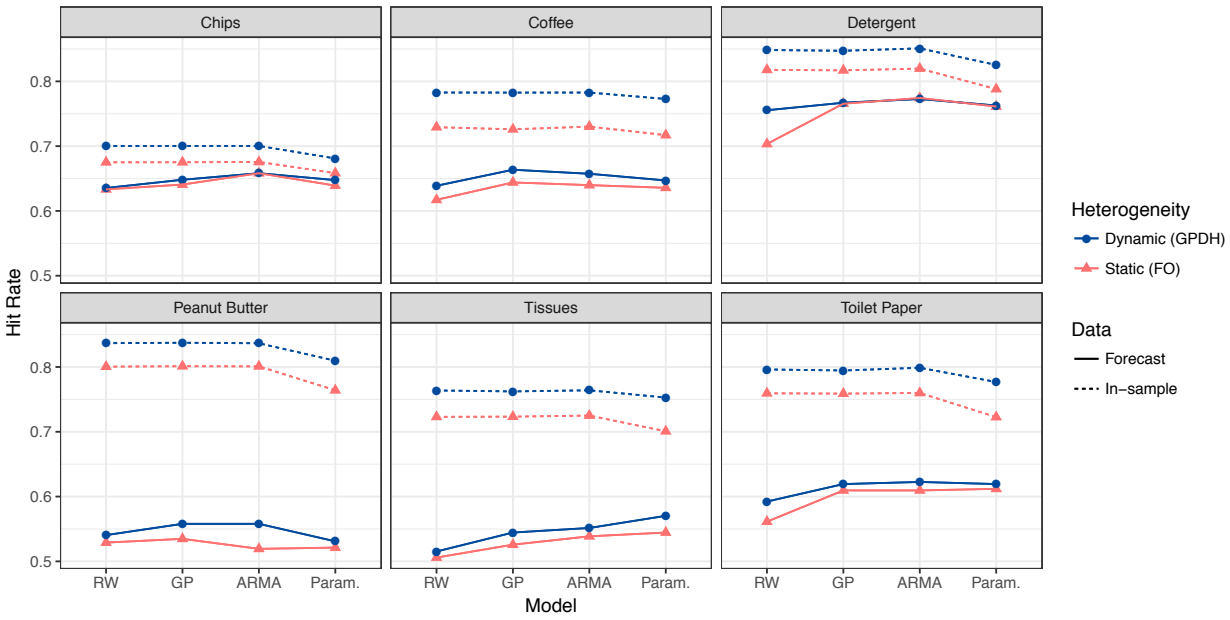


Figure 5: Hit rates (accuracy) across model specifications and data (in-sample and forecast), analogous to Figure 7

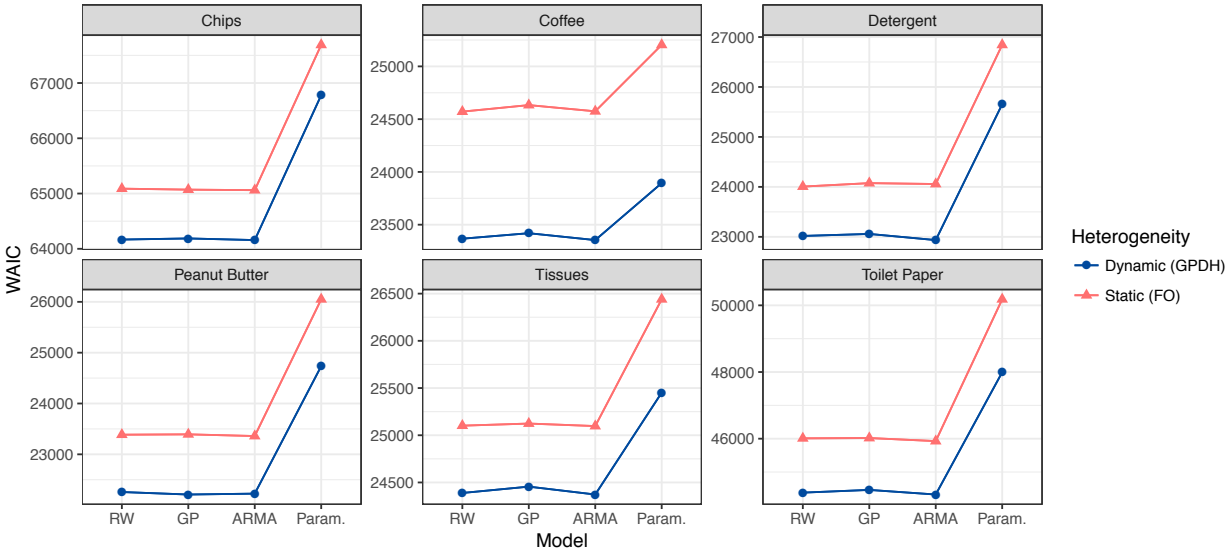


Figure 6: WAIC across model specifications. Lower indicates better fit, taking into account model complexity.

In-sample

		Macro			Micro			Max			Min		
Category	Heterogeneity	Prec	Sens	Spec	Prec	Sens	Spec	Prec	Sens	Spec	Prec	Sens	Spec
Chips	GPDH	.695	.629	.882	.695	.695	.898	.709	.832	.978	.665	.484	.724
Chips	FO	.665	.599	.873	.671	.671	.890	.689	.817	.975	.631	.434	.706
Coffee	GPDH	.782	.754	.940	.780	.780	.945	.804	.844	.985	.756	.675	.879
Coffee	FO	.729	.696	.926	.726	.726	.931	.763	.805	.983	.673	.608	.854
Detergent	GPDH	.836	.813	.967	.843	.843	.969	.868	.918	.992	.796	.732	.927
Detergent	FO	.801	.774	.960	.811	.811	.962	.844	.905	.991	.717	.665	.913
Peanut Butter	GPDH	.832	.819	.956	.830	.830	.958	.874	.872	.984	.810	.749	.929
Peanut Butter	FO	.789	.776	.946	.792	.792	.948	.858	.843	.982	.717	.632	.911
Tissues	GPDH	.762	.751	.916	.761	.761	.920	.776	.788	.970	.741	.703	.866
Tissues	FO	.717	.704	.901	.718	.718	.906	.735	.752	.963	.701	.630	.840
Toilet Paper	GPDH	.791	.781	.957	.792	.792	.958	.818	.846	.984	.742	.710	.924
Toilet Paper	FO	.746	.736	.948	.750	.750	.950	.789	.818	.981	.709	.665	.911

Forecast

		Macro			Micro			Max			Min		
Category	Heterogeneity	Prec	Sens	Spec	Prec	Sens	Spec	Prec	Sens	Spec	Prec	Sens	Spec
Chips	GPDH	.646	.528	.865	.647	.647	.882	.777	.769	.991	.524	.182	.704
Chips	FO	.618	.527	.863	.643	.643	.881	.778	.758	.987	.434	.200	.708
Coffee	GPDH	.631	.629	.904	.652	.652	.913	.756	.718	.967	.522	.521	.812
Coffee	FO	.615	.607	.900	.634	.634	.909	.753	.699	.969	.492	.521	.810
Detergent	GPDH	.717	.618	.947	.764	.764	.953	.878	.931	.996	.453	.177	.858
Detergent	FO	.720	.611	.943	.751	.751	.950	.877	.910	.996	.502	.218	.833
Peanut Butter	GPDH	.519	.545	.886	.547	.547	.887	.752	.650	.950	.331	.332	.822
Peanut Butter	FO	.492	.520	.879	.526	.526	.882	.673	.643	.931	.312	.279	.820
Tissues	GPDH	.558	.559	.846	.545	.545	.848	.712	.735	.924	.459	.470	.768
Tissues	FO	.541	.539	.840	.529	.529	.843	.699	.706	.923	.445	.458	.759
Toilet Paper	GPDH	.588	.574	.920	.613	.613	.923	.747	.782	.978	.369	.239	.859
Toilet Paper	FO	.562	.550	.917	.598	.598	.920	.722	.813	.980	.304	.169	.852

Table 2: Fit statistics average across mean model. The statistics are described above.

E: Average Elasticity Over Time

Similar to previous analyses of the Great Recession, our GPDH results can be used to nonparametrically study how price elasticity, on average, changed during the Great Recession, by simply averaging over individuals. Below, we present the full set of price elasticity plots over time. In the detergent, chips, and toilet paper categories, we find many brands experienced significant increases in average price elasticity. This is intuitive as the Great Recession negatively affected many people's earnings, which should lead to higher price sensitivity. Peanut butter and coffee, on the other hand, do not appear to have been significantly impacted. Finally, tissues appears to have behaved almost countercyclically during the recession: for all brands in tissues, the average recession-era price elasticity was smaller than before and after. There are several caveats to this population-level analysis, which may limit its interpretability or generalizability, and which also limits its comparability to previous studies, e.g., Gordon et al. (2013). Importantly, in this work, we only modeled choice conditional on the purchase decision, and do not capture effects like stockpiling. We also use a relatively lenient rule for retaining consumers in the panel, such that consumers that purchased at least five times were included. This means our estimates of average price elasticities may be subject to panelist attrition.

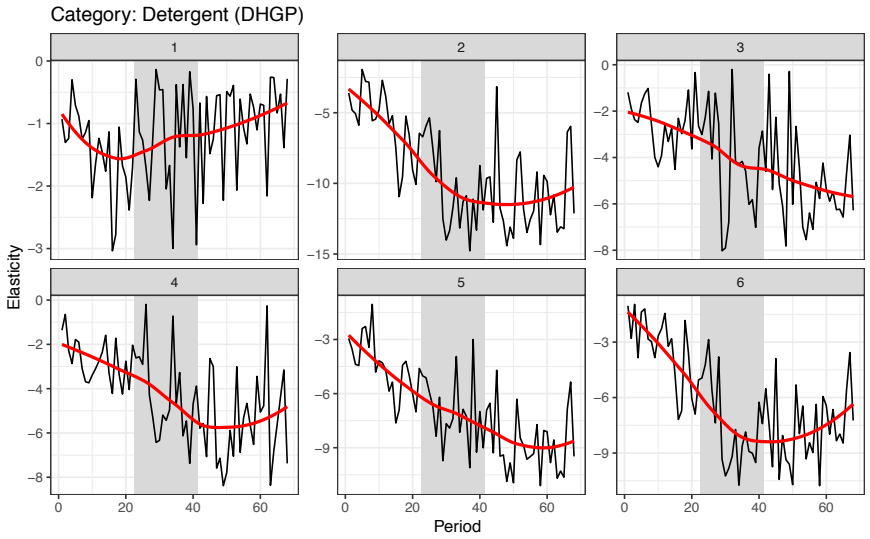


Figure 7: The average price elasticity of demand across detergent brands over time, as estimated by the GPDH logit model. The recession era, as defined by NBER, is marked by the grey rectangle. Overlaid on the estimated average price elasticities is a local linear smoothing (LOESS).

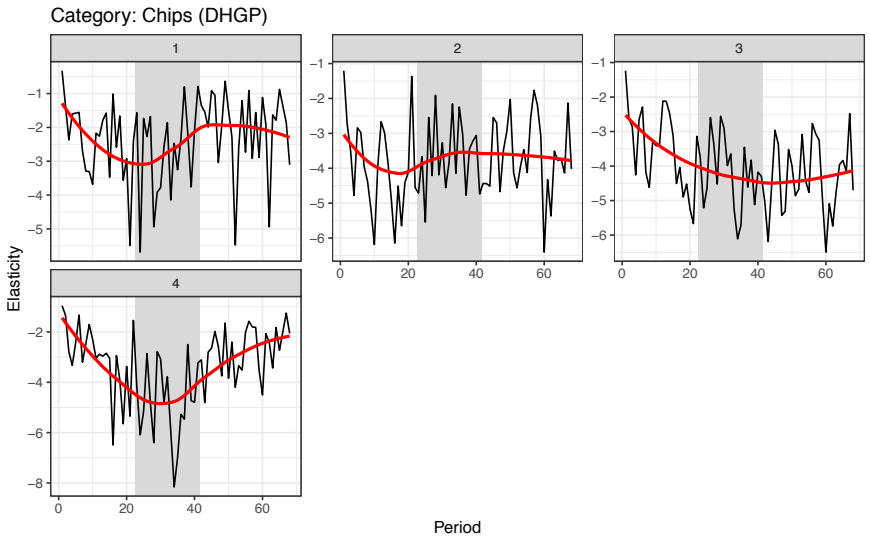


Figure 8: The average price elasticity of demand across chips brands over time, as estimated by the GPDH logit model. The recession era, as defined by NBER, is marked by the grey rectangle. Overlaid on the estimated average price elasticities is a local linear smoothing (LOESS).

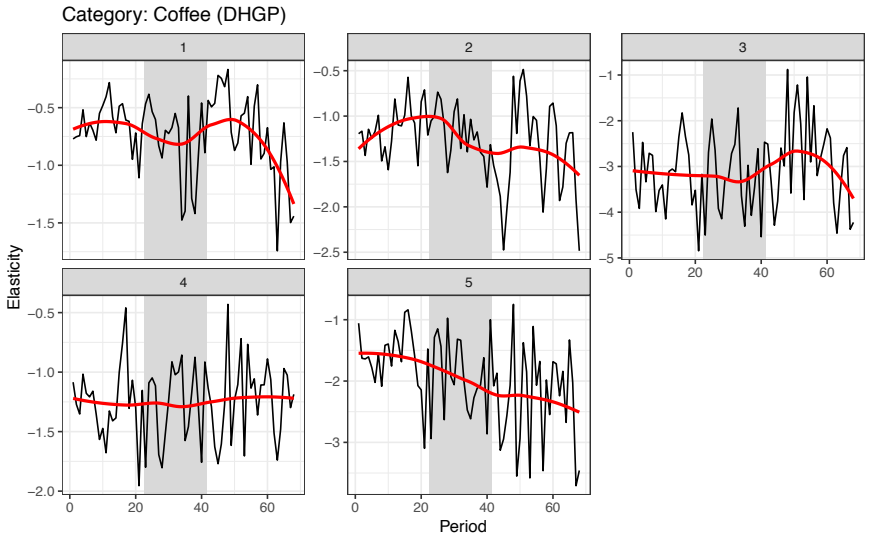


Figure 9: The average price elasticity of demand across coffee brands over time, as estimated by the GPDH logit model. The recession era, as defined by NBER, is marked by the grey rectangle. Overlaid on the estimated average price elasticities is a local linear smoothing (LOESS).

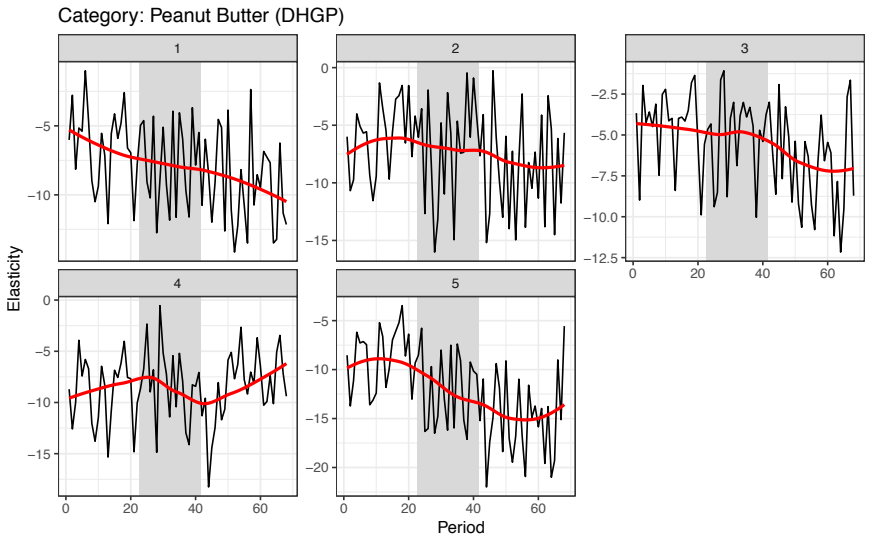


Figure 10: The average price elasticity of demand across peanut butter brands over time, as estimated by the GPDH logit model. The recession era, as defined by NBER, is marked by the grey rectangle. Overlaid on the estimated average price elasticities is a local linear smoothing (LOESS).

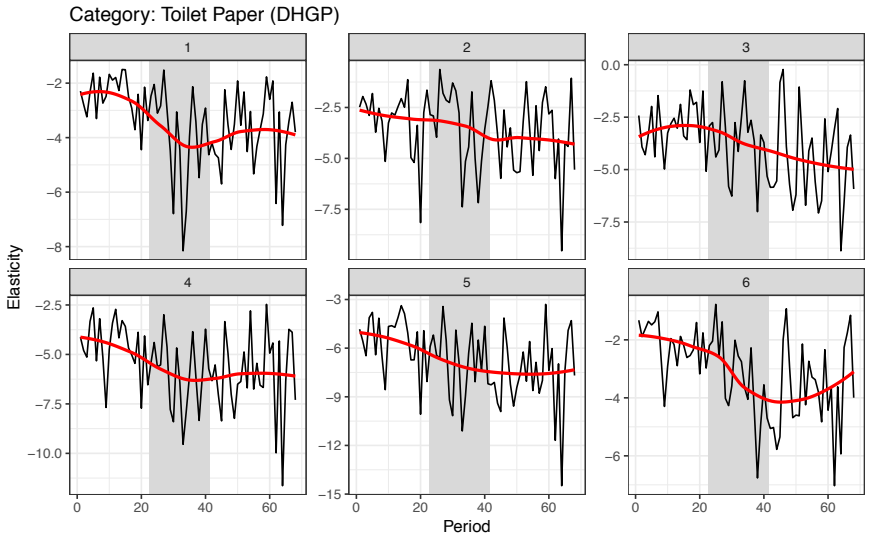


Figure 11: The average price elasticity of demand across toilet paper brands over time, as estimated by the GPDH logit model. The recession era, as defined by NBER, is marked by the grey rectangle. Overlaid on the estimated average price elasticities is a local linear smoothing (LOESS).

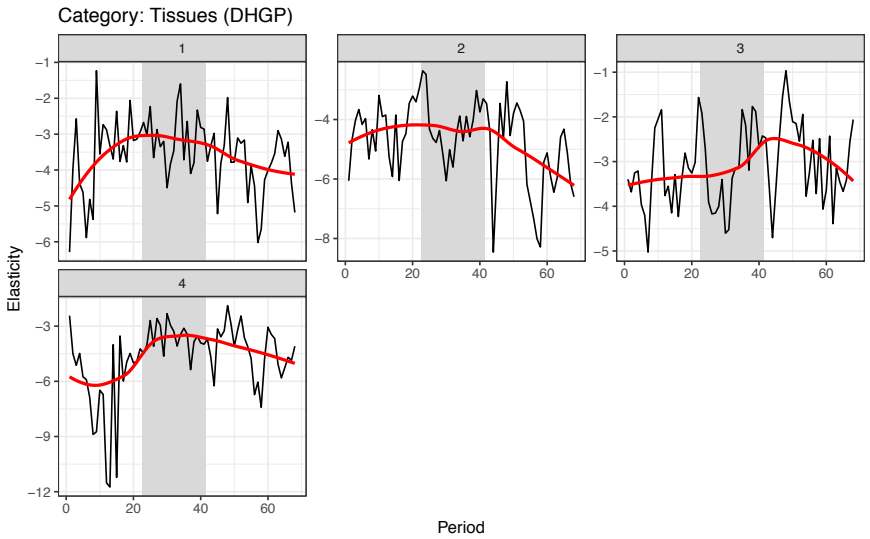


Figure 12: The average price elasticity of demand across tissue brands over time, as estimated by the GPDH logit model. The recession era, as defined by NBER, is marked by the grey rectangle. Overlaid on the estimated average price elasticities is a local linear smoothing (LOESS).

F: Curve Timing Plots

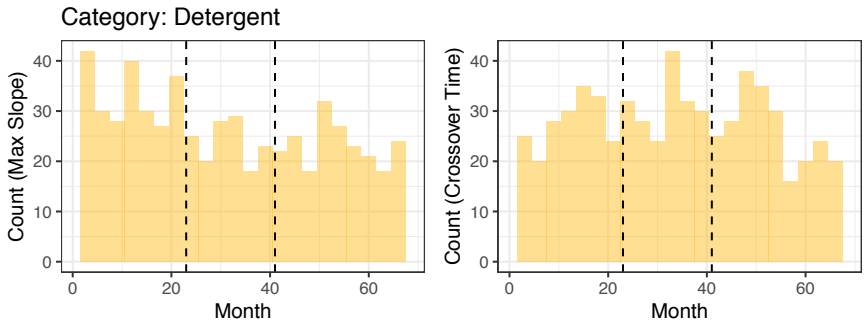


Figure 13: At left, the distribution of the timings of maximal slopes for individual-level curves in the detergent category, with the recession bounded by the dashed lines. At right, the distribution of the timings of crossovers in the chips category, again with the recession bounded by dashed lines.

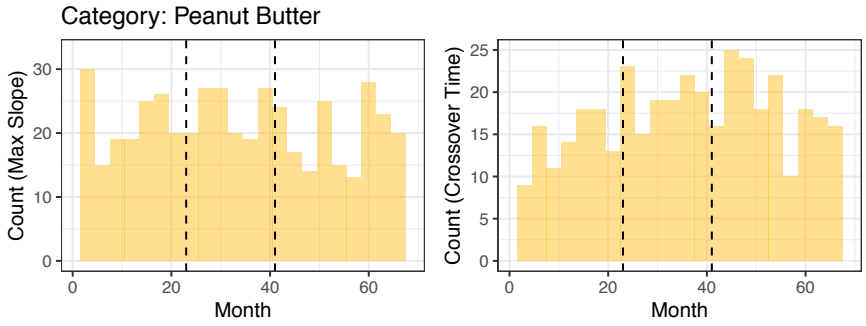


Figure 14: At left, the distribution of the timings of maximal slopes for individual-level curves in the peanut butter category, with the recession bounded by the dashed lines. At right, the distribution of the timings of crossovers in the coffee category, again with the recession bounded by dashed lines.

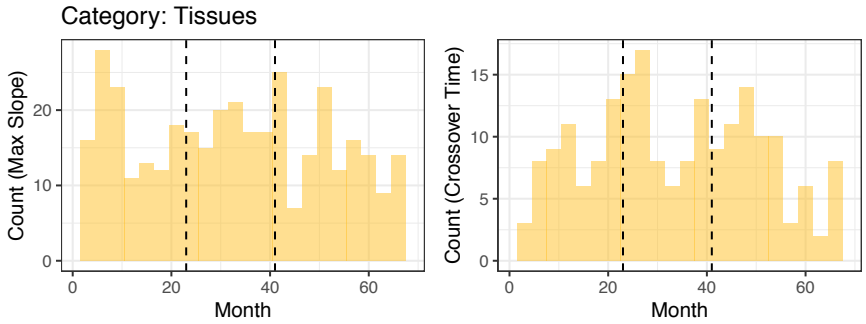


Figure 15: At left, the distribution of the timings of maximal slopes for individual-level curves in the tissues category, with the recession bounded by the dashed lines. At right, the distribution of the timings of crossovers in the chips category, again with the recession bounded by dashed lines.

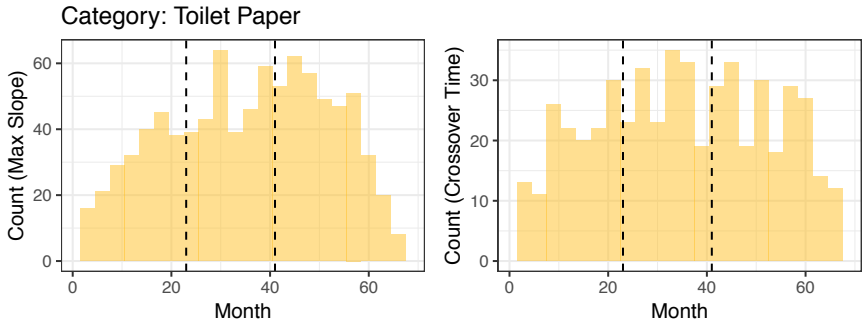


Figure 16: At left, the distribution of the timings of maximal slopes for individual-level curves in the toilet paper category, with the recession bounded by the dashed lines. At right, the distribution of the timings of crossovers in the coffee category, again with the recession bounded by dashed lines.

G: Computation Times

We report in Table 3 the computation times for each of the product categories from Application 1, for each of the mean model specifications. There is a positive correlation between the number of people, the number of purchases, and the computation time, although with only six categories, it is difficult to make further claims about scalability. The average time across all of the categories was 26 hours.

Category	# People	# Purchases	ARMA	GP	Param	RW	Average
Chips	1552	36152	33.6	13.8	25.4	16.9	22.41
Coffee	912	14298	17.6	22.4	15.5	12.2	16.92
Detergent	1117	16784	37.6	40.7	46.6	41.3	41.56
Peanut Butter	1085	16212	18.9	10.7	19.8	16.2	16.40
Tissues	979	15005	18.2	22.1	25.6	21.6	21.89
Toilet Paper	1512	26958	34.6	69.1	27.3	19.2	37.52

Table 3: Computation time, in hours, across the categories and mean model specifications.

H: Hyperparameter Estimates

Parameter		Chips	Coffee	Detergent	Peanut Butter	Tissues	Toilet Paper
Brand 2	η	1.18	2.29	3.18	1.86	2.01	2.05
	κ	.02	.02	.02	.03	.03	.02
Brand 3	η	1.71	2.76	1.47	1.82	1.51	1.95
	κ	.04	.02	.04	.03	.03	.02
Brand 4	η	1.15	2.53	1.24	1.40	1.43	2.96
	κ	.04	.03	.01	.00	.07	.03
Brand 5	η		2.38	1.83	3.08		2.92
	κ		.07	.03	.06		.02
Brand 6	η			2.06			1.24
	κ			.08			.09
Ft/Dsp	η	.09	.27	.37	.28	.30	.41
	κ	.04	.00	.01	.00	.01	.03
Price	η	.65	1.34	2.35	1.22	6.94	.86
	κ	.03	.02	.02	.02	.02	.02

Table 4: Hyperparameter estimates for application 1, across all categories and coefficients.

I: Stan Code

Here, we include the Stan code for the GPDH-ARMA choice model.

```
functions{
 real maternk(real x1, real x2, real eta, real kappa, int type){
    // NOTE ON THE TYPE INPUT:
    // type 0: matern-1/2
    // type 1: matern-3/2
    // type 2: matern-5/2
    // type 3: squared exponential (technically, should be type infinity)

    real r = fabs(x1-x2);
    real out;
    if (type == 0) {out = eta^2 * exp(-kappa*r); }
    if (type == 1) { out = eta^2 * (1+kappa*r) * exp(-kappa*r); }
    if (type == 2) { out = eta^2 * (1 + kappa*r + pow(kappa*r, 2)/3.0) * exp(-kappa*r); }
    if (type > 2) { out = eta^2 * exp(-pow(kappa*r, 2));}
    return out;
 }

 matrix Kmat(int P, real eta, real kappa, int type, real jitter){
    matrix[P, P] cov;

    for(i in 1:P){
        for(j in 1:P){
            cov[i,j] = maternk(i, j, eta, kappa, type);
         }
        cov[i,i] = cov[i,i] + jitter;
    }
    return cov;
 }

  // Penalize complexity prior for the hyperparameters of a matern kernel,
  // from Simpson et al., 2017
 real pc_prior_lpdf(vector hypers, real ktype, real eta_upper, real alpha_eta, real rho_lower,
   real alpha_rho){
    real degree = ktype + 0.5;
    real eta = hypers[1];
    real kappa = hypers[2];
    real lambda1 = -log(alpha_rho) * sqrt(rho_lower / sqrt(8.0*degree));
    real lambda2 = -log(alpha_eta) / eta_upper;

    return (log(0.5) + log(lambda1) - 0.5*log(kappa) - lambda1*sqrt(kappa) +log(lambda2) - lambda2*eta);
  }
}

data{
 int<lower=2> B;     // no. goods
 int<lower=1> N;     // no. customers
 int<lower=1> P;     // no. periods per customer
 int<lower=1> M;     // no. total choices

 int y[M];              // choice of customer on each choice occassion
 matrix[M,B] price;     // prices for each good at each choice occassion
 matrix[M,B] ftdsp;     // display/feature for each good at ...
 int<lower=1,upper=N> id[M];   // person id
 int<lower=1,upper=P> pd[M];   // period id
 int ktype;
}

parameters{
 // mean functions: ------------------------------------------------
 vector[P] mu_icept[B-1];
 vector[P] mu_price;
```

```
 vector[P] mu_ftdsp;

 // mf var parameters:
 real<lower=0> tau_icept[B-1];
 real<lower=0> tau_price;
 real<lower=0> tau_ftdsp;

 // mf arma parameters:
 real m_icept[B-1];
 real<lower = -1, upper = 1> phi_icept[B-1];
 real<lower = -1, upper = 1> theta_icept[B-1];

 real m_price;
 real<lower = -1, upper = 1> phi_price;
 real<lower = -1, upper = 1> theta_price;

 real m_ftdsp;
 real<lower = -1, upper = 1> phi_ftdsp;
 real<lower = -1, upper = 1> theta_ftdsp;

 // individual-specific GPs: ---------------------------------------
 vector[P] z_icept[N,B-1];
 vector[P] z_price[N];
 vector[P] z_ftdsp[N];

 // lower-level GP hyperparameters (shared across people):
 vector<lower=0>[2] hypers_icept[B-1];
 vector<lower=0>[2] hypers_price;
 vector<lower=0>[2] hypers_ftdsp;
}

transformed parameters{
 // individual-specific GPs: ---------------------------------------
 vector[P] beta_icept[N,B-1];
 vector[P] beta_price[N];
 vector[P] beta_ftdsp[N];

 // module to contain the covariance matrices:
 {
 // individual-level kernel matrices: ------------------------------
 matrix[P,P] K[B-1];
 matrix[P,P] L[B-1];
 matrix[P,P] K_price;
 matrix[P,P] L_price;
 matrix[P,P] K_ftdsp;
 matrix[P,P] L_ftdsp;

 // IN THIS SECTION: use the user defined functions to create covariance matrices,
 // then use the reparametrization of the normal distribution with the cholesky
 // decomposition of the kernel to form the mean function and function values

 // intercept kernels and function values:
 for(b in 1:(B-1)){
   K[b] = Kmat(P, hypers_icept[b,1], hypers_icept[b,2], ktype, 1e-8);
   L[b] = cholesky_decompose(K[b]);
   for(n in 1:N){
    beta_icept[n,b] = mu_icept[b] + L[b] * z_icept[n,b];
  }
 }
 K_price = Kmat(P, hypers_price[1], hypers_price[2], ktype, 1e-8);
 L_price = cholesky_decompose(K_price);
 for(n in 1:N){
  beta_price[n] = mu_price + L_price * z_price[n];
 }
 K_ftdsp = Kmat(P, hypers_ftdsp[1], hypers_ftdsp[2], ktype, 1e-8);
 L_ftdsp = cholesky_decompose(K_ftdsp);
 for(n in 1:N){
  beta_ftdsp[n] = mu_ftdsp + L_ftdsp * z_ftdsp[n];
```

```
  }
 }
}
model{
 // mean function:

 vector[P] err_icept[B-1];
 vector[P] err_price;
 vector[P] err_ftdsp;

 vector[P] nu_icept[B-1];
 vector[P] nu_price;
 vector[P] nu_ftdsp;

 for(b in 1:(B-1)){
  m_icept[b] ~ normal(0,10);
  phi_icept[b] ~ normal(0,2);
  theta_icept[b] ~ normal(0,2);
  tau_icept[b] ~ normal(0,1);

  nu_icept[b][1]=m_icept[b]+phi_icept[b]*m_icept[b];
  mu_icept[b][1] ~ normal(nu_icept[b][1], tau_icept[b]);
  err_icept[b][1]=mu_icept[b][1]-nu_icept[b][1];

  for(t in 2:P){
    nu_icept[b][t]= m_icept[b]+phi_icept[b]*mu_icept[b][t-1]+theta_icept[b]*err_icept[b][t-1];
    mu_icept[b][t] ~ normal(nu_icept[b][t], tau_icept[b]);
    err_icept[b][t] = mu_icept[b][t] - nu_icept[b][t];
  }
 }

 m_price ~ normal(0,10);
 phi_price ~ normal(0,2);
 theta_price ~ normal(0,2);
 tau_price ~ normal(0,1);
 nu_price[1]=m_price+phi_price*m_price;
 mu_price[1] ~ normal(nu_price[1], tau_price);
 err_price[1] = mu_price[1]-nu_price[1];

 for(t in 2:P){
  nu_price[t]=m_price+phi_price*mu_price[t-1]+theta_price*err_price[t-1];
  mu_price[t] ~ normal (nu_price[t], tau_price);
  err_price[t]=mu_price[t] - nu_price[t];
 }

 m_ftdsp ~ normal(0,10);
 phi_ftdsp ~ normal(0, 2);
 theta_ftdsp ~ normal(0,2);
 tau_ftdsp ~ normal(0,1);

 nu_ftdsp[1] = m_ftdsp+phi_ftdsp*m_ftdsp;
 mu_ftdsp[1] ~ normal(nu_ftdsp[1], tau_ftdsp);
 err_ftdsp[1] = mu_ftdsp[1]-nu_ftdsp[1];

 for(t in 2:P){
  nu_ftdsp[t]=m_ftdsp+phi_ftdsp*mu_ftdsp[t-1]+theta_ftdsp*err_ftdsp[t-1];
  mu_ftdsp[t] ~ normal (nu_ftdsp[t], tau_ftdsp);
  err_ftdsp[t]=mu_ftdsp[t] - nu_ftdsp[t];
 }

 // lower-level hyperparameters (function values)
 for(b in 1:(B-1)){
  hypers_icept[b] ~ pc_prior(ktype, 5.0, 0.01, 1.0, 0.001);
 }
 hypers_price ~ pc_prior(ktype, 5.0, 0.01, 1.0, 0.001);
 hypers_ftdsp ~ pc_prior(ktype, 5.0, 0.01, 1.0, 0.001);

 // individual-specific functions (reparametrization form, don't save this)
```

```
 for(i in 1:N){
  for(b in 1:(B-1)){
    z_icept[i,b] ~ normal(0,1);
  }
  z_price[i] ~ normal(0,1);
  z_ftdsp[i] ~ normal(0,1);
 }

 // likelihood for each choice occassion:
 for(m in 1:M){
  vector[B] util;

  // compute utility for each good; first good has util = 0
  util[1] = beta_price[id[m]][pd[m]]*price[m,1] + beta_ftdsp[id[m]][pd[m]]*ftdsp[m,1];
  for(b in 2:B){
    util[b] = beta_icept[id[m],b-1][pd[m]] + beta_price[id[m]][pd[m]]*price[m,b] +
    beta_ftdsp[id[m]][pd[m]]*ftdsp[m,b];
  }
  y[m] ~ categorical_logit(util);
 }
}
generated quantities{
 vector[M] log_lik;

 // compute observation-level log-likelihood for computing WAIC/LOO:
 for(m in 1:M){
    vector[B] util;

    util[1] = beta_price[id[m]][pd[m]]*price[m,1]+beta_ftdsp[id[m]][pd[m]]*ftdsp[m,1];
    for(b in 2:B){
      util[b] = beta_icept[id[m],b-1][pd[m]]+beta_price[id[m]][pd[m]]*price[m,b]+
      beta_ftdsp[id[m]][pd[m]]*ftdsp[m,b];
    }
    log_lik[m] = categorical_logit_lpmf(y[m] | util);
 }
}
```

REFERENCES

Gordon, B., Goldfarb, A., and Li, Y. (2013). Does Price Elasticity Vary with Economic Growth? A Cross-category Analysis. *Journal of Marketing Research*, 50(February):4–23.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Web Appendix J: Incorporating Drivers of Shifts

In the main body of the paper, we show how a GPDH specification can be used to capture individual-level parameter dynamics. One critique is that GPDH may capture these trajectories, but it does not capture drivers of these changes, which may shed light on why preferences are changing, or be used to predict when preferences are likely to change again. In this note, we show how the GPDH mean function can be leveraged to capture these drivers. In this case, GPDH then captures heterogeneity around these predicted shifts in preferences. In particular, we will consider the case where a recession causes a decrease in price sensitivity. This is a simple case, but conveys how the model could be modified to include drivers of shifts in preferences.

Required Packages:

```
library(mvtnorm)
library(rstan)
```

```
## Loading required package: ggplot2
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Loading required package: StanHeaders
```

```
## rstan (Version 2.18.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

In our paper, we assess the impact of the recession nonparametrically. However, let's assume we instead wanted to model the recession as a driver of changes in price sensitivity. In the GPDH framework, this can be accomplished by encoding the recession parametrically in the mean function:

$$\mu^{Price}(t) = \gamma_0 + \gamma_1 \text{Recession}_t,$$

where $\text{Recession}_t = 1$ during the period of the recession and zero otherwise. Then, the rest of the GPDH framework can be specified as normal. Below, we simulate data from this process: we assume there are 20 periods observed, and a recession occurs in periods 8-13.

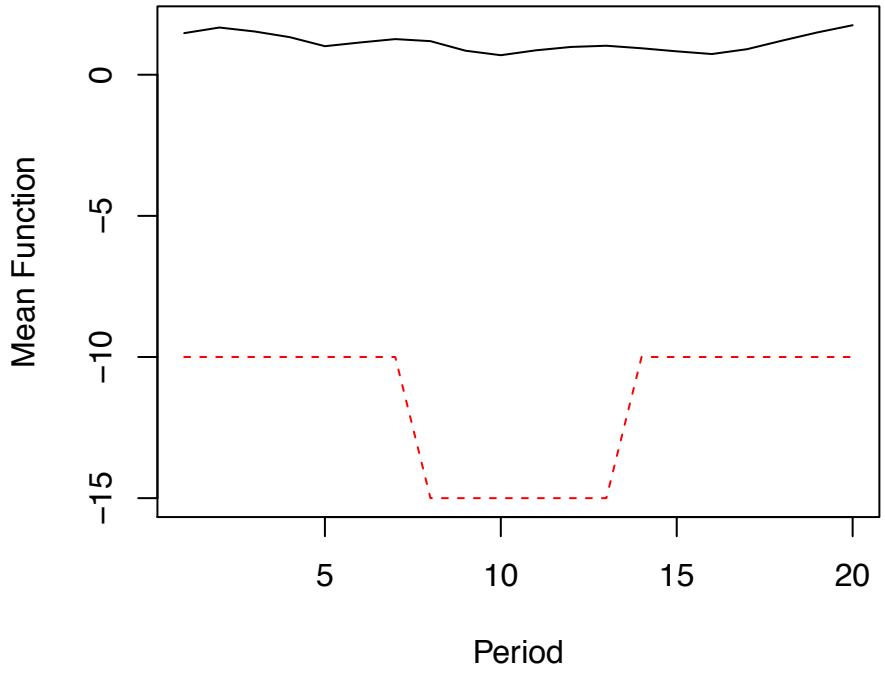
```
x_recession = c(rep(0, 7), rep(1, 6), rep(0, 7))
```

For the rest of the data generating process, we assume a similar setup to our simulation studies, except with just two brands for simplicity:

```
N = 200
min_m = 5
P = 20
K = 3
m_shape = 1
m_scale = 8
price_shape = 2
price_scale = 0.3
eta = c(2,2,5)
```

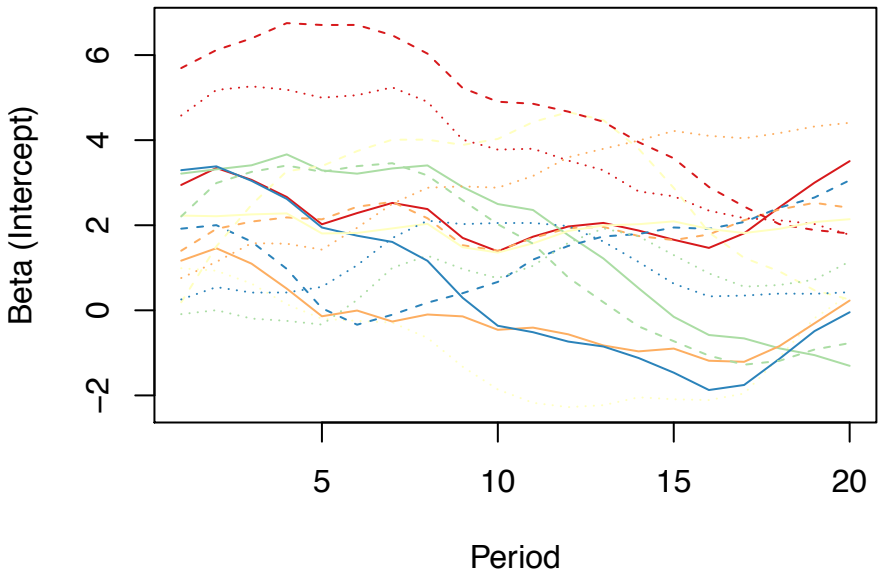
Here, we generate a GP mean model for the intercept, and the parametric mean model specified above for the recession:

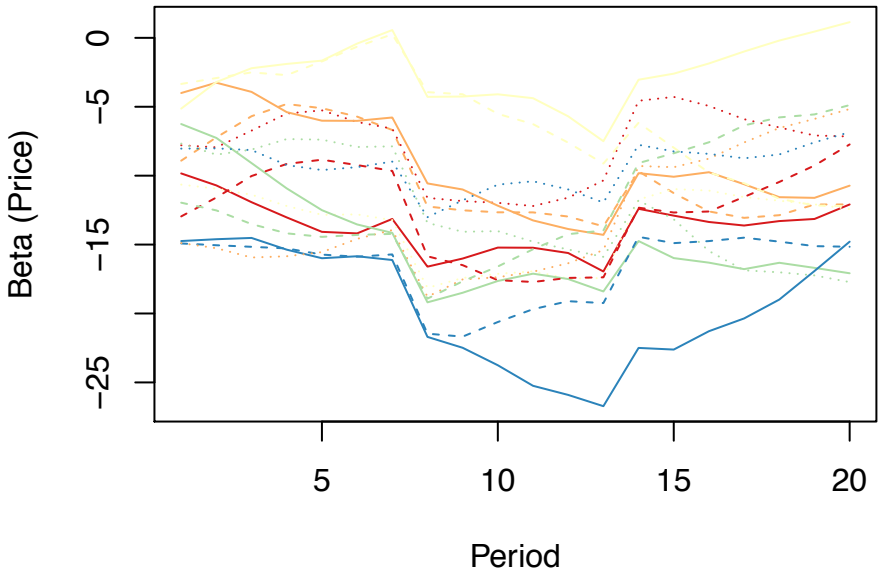
```r
set.seed(7926)
source("gp.R")

icept_mf_cov = make.cov(1:P, matern.kern, c(2, P/2, 3/2))
icept_mf = c(rmvnorm(1, sigma = icept_mf_cov))

price_mf = -10 - 5*x_recession

matplot(cbind(icept_mf, price_mf), type = "l", xlab="Period", ylab="Mean Function")
```



Next, we generate the individual-level parameters, assuming GPDH:

```r
set.seed(7926)
beta_icept = rmvnorm(N, mean = icept_mf, sigma = make.cov(1:P, matern.kern, c(2, P/2, 3/2)))
beta_price = rmvnorm(N, mean = price_mf, sigma = make.cov(1:P, matern.kern, c(5, P/2, 3/2)))

par(mfrow=c(2,1))
colpal = RColorBrewer::brewer.pal(5, "Spectral")
matplot(t(beta_icept[1:15,]), type = "l", xlab="Period", ylab="Beta (Intercept)", col=colpal, lty=c(1,2
matplot(t(beta_price[1:15,]), type = "l", xlab="Period", ylab="Beta (Price)", col=colpal, lty=c(1,2,3))
```





In the price sensitivity plot, we can see a distinct recession effect for most individuals, although we also notice that the individuals nonparametrically deviate from the piecewise flat mean function.

Now, we simulate choice data using these individual-level parameters:

```
set.seed(7926)
m = round(rgamma(N, shape=m_shape, scale=m_scale)) + min_m

pd = list()
price = list()
y = list()
for(i in 1:N){
```

```
  pd[[i]] = sort(sample(1:P, m[i], replace=T), decreasing=F)
  price[[i]] = matrix(1+rgamma(m[i]*K, shape = price_shape, scale = price_scale), nrow=m[i], ncol=K)
  y[[i]] = rep(0, length(pd[[i]]))
  for(p in seq_along(pd[[i]])){
    month = pd[[i]][p]
    util = c(0,0)
    util[1] = beta_price[i,month]*price[[i]][p,1]
    util[2] = beta_icept[i,month]+beta_price[i,month]*price[[i]][p,2]
    choice_probs = exp(util)/sum(exp(util))
    y[[i]][p] = sample(1:2, size = 1, prob = choice_probs)
  }
}
```

To estimate this in Stan, we can adapt the Stan code from the full model:

```
recession_model = "
  functions{
    real maternk(real x1, real x2, real eta, real kappa, int type){
        // NOTE ON THE TYPE INPUT:
        // type 0: matern-1/2
        // type 1: matern-3/2
        // type 2: matern-5/2
        // type 3: squared exponential (technically, should be type infinity)

        real r = fabs(x1-x2);
        real out;

        if (type == 0) {
            out = eta^2 * exp(-kappa*r);
        }

        if (type == 1) {
            out = eta^2 * (1+kappa*r) * exp(-kappa*r);
        }

        if (type == 2) {
            out = eta^2 * (1 + kappa*r + pow(kappa*r, 2)/3.0) * exp(-kappa*r);
        }

        if (type > 2) {
            out = eta^2 * exp(-pow(kappa*r, 2));
        }

        return out;
    }

    matrix Kmat(int P, real eta, real kappa, int type, real jitter){
      matrix[P, P] cov;

      for(i in 1:P){
        for(j in 1:P){
          cov[i,j] = maternk(i, j, eta, kappa, type);
        }
        cov[i,i] = cov[i,i] + jitter;
```

```
    }

    return cov;
  }

  // Penalize complexity prior for the hyperparameters of a matern kernel,
  // from Simpson et al., 2017
  real pc_prior_lpdf(vector hypers, real ktype, real eta_upper, real alpha_eta, real rho_lower, real a
    real degree = ktype + 0.5;
    real eta = hypers[1];
    real kappa = hypers[2];
    real lambda1 = -log(alpha_rho) * sqrt(rho_lower / sqrt(8.0*degree));
    real lambda2 = -log(alpha_eta) / eta_upper;

    return log(0.5) + log(lambda1) - 0.5*log(kappa) - lambda1*sqrt(kappa) + log(lambda2) - lambda2*eta
  }
}

data{
  int<lower=2> B;     // no. goods
  int<lower=1> N;     // no. customers
  int<lower=1> P;     // no. periods per customer
  int<lower=1> M;     // no. total choices

  int y[M];              // choice of customer on each choice occassion
  matrix[M,B] price;     // prices for each good at each choice occassion

  vector[P] x_recession; // indicator variable for the periods of the recession

  int<lower=1,upper=N> id[M];   // person id
  int<lower=1,upper=P> pd[M];   // period id

  int ktype;
}

parameters{
  // constant hypermeans: ---------------------------------------------
  real lambda_icept[B-1];
  real gamma0;
  real gamma1;

  // mean functions: --------------------------------------------------
  vector[P] mu_icept[B-1];

  // mf GP hyperparameters:
  vector<lower=0>[2] hypers0_icept[B-1];

  // individual-specific GPs: -----------------------------------------
  vector[P] z_icept[N,B-1];
  vector[P] z_price[N];

  // lower-level GP hyperparameters (shared across people):
  vector<lower=0>[2] hypers_icept[B-1];
```

```
  vector<lower=0>[2] hypers_price;
}

transformed parameters{
  vector[P] mu_price;

  // individual-specific GPs: ----------------------------------------
  vector[P] beta_icept[N,B-1];
  vector[P] beta_price[N];


  mu_price = gamma0 + gamma1 * x_recession;

  // module to contain the covariance matrices:
  {
    // individual-level kernel matrices: ------------------------------
    matrix[P,P] K_icept[B-1];
    matrix[P,P] L_icept[B-1];
    matrix[P,P] K_price;
    matrix[P,P] L_price;

    // IN THIS SECTION: use the user defined functions to create covariance matrices,
    // then use the reparametrization of the normal distribution with the cholesky
    // decomposition of the kernel to form the mean function and function values

    // intercept kernels and function values:
    for(b in 1:(B-1)){
      K_icept[b] = Kmat(P, hypers_icept[b,1], hypers_icept[b,2], ktype, 1e-8);
      L_icept[b] = cholesky_decompose(K_icept[b]);

      for(n in 1:N){
        beta_icept[n,b] = mu_icept[b] + L_icept[b] * z_icept[n,b];
      }
    }

    // price kernels and function values:
    K_price = Kmat(P, hypers_price[1], hypers_price[2], ktype, 1e-8);
    L_price = cholesky_decompose(K_price);

    for(n in 1:N){
      beta_price[n] = mu_price + L_price * z_price[n];
    }

  }
}
model{
  // mean functions kernel matrices: --------------------------------
  matrix[P,P] K0_icept[B-1];
  matrix[P,P] L0_icept[B-1];

  // mean function hyperparameters, constant mean, and repar z values:
  for(b in 1:(B-1)){
    lambda_icept[b] ~ normal(0,5);
```

```
      hypers0_icept[b] ~ pc_prior(ktype, 5.0, 0.01, 0.2*P, 0.001);

      K0_icept[b] = Kmat(P, hypers0_icept[b,1], hypers0_icept[b,2], ktype, 1e-8);
      L0_icept[b] = cholesky_decompose(K0_icept[b]);

      mu_icept[b] ~ multi_normal_cholesky(rep_vector(lambda_icept[b], P), L0_icept[b]) ;
    }

    gamma0 ~ normal(0,10);
    gamma1 ~ normal(0,10);

    // lower-level hyperparameters (function values)
    for(b in 1:(B-1)){
      hypers_icept[b] ~ pc_prior(ktype, 5.0, 0.01, 1.0, 0.001);
    }

    hypers_price ~ pc_prior(ktype, 5.0, 0.01, 1.0, 0.001);

    // individual-specific functions (reparametrization form, don't save this)
    for(i in 1:N){
      for(b in 1:(B-1)){
        z_icept[i,b] ~ normal(0,1);
      }
      z_price[i] ~ normal(0,1);
    }


    // likelihood for each choice occassion:
    for(m in 1:M){
      vector[B] util;

      // compute utility for each good; first good has intercept = 0
      util[1] = beta_price[id[m]][pd[m]]*price[m,1];
      for(b in 2:B)
        util[b] = beta_icept[id[m],b-1][pd[m]] + beta_price[id[m]][pd[m]]*price[m,b];

      y[m] ~ categorical_logit(util);
    }
  }
"
```

```
set.seed(7926)
dl = list(
  B = K,
  N = N,
  P = P,
  M = sum(m),
  y = do.call(c, y),
  price = do.call(rbind, price),
  id = rep(1:N, times = m),
  pd = do.call(c, pd),
  ktype = 1,
  x_recession = x_recession
)
```

```
stanout = stan(model_code = recession_model, data = dl, iter = 400, chains = 1,
               control = list(adapt_delta=0.95, max_treedepth=20), seed=7926)
```

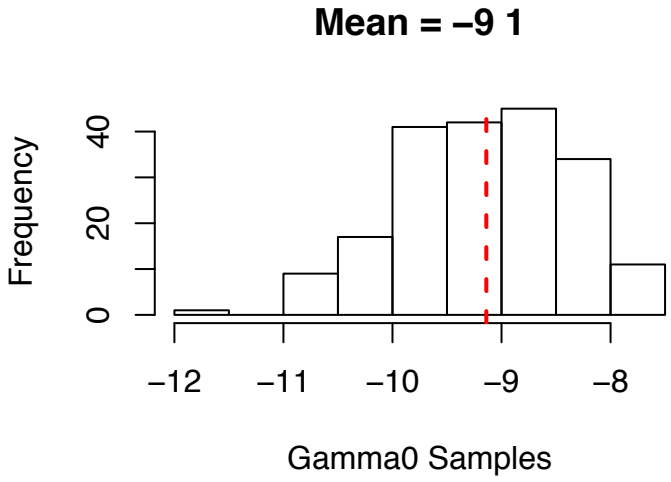
```
##
## SAMPLING FOR MODEL '8b7ca14e22f00287680585c04e01741d' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.006262 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 62.62 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 400 [  0%]  (Warmup)
## Chain 1: Iteration:  40 / 400 [ 10%]  (Warmup)
## Chain 1: Iteration:  80 / 400 [ 20%]  (Warmup)
## Chain 1: Iteration: 120 / 400 [ 30%]  (Warmup)
## Chain 1: Iteration: 160 / 400 [ 40%]  (Warmup)
## Chain 1: Iteration: 200 / 400 [ 50%]  (Warmup)
## Chain 1: Iteration: 201 / 400 [ 50%]  (Sampling)
## Chain 1: Iteration: 240 / 400 [ 60%]  (Sampling)
## Chain 1: Iteration: 280 / 400 [ 70%]  (Sampling)
## Chain 1: Iteration: 320 / 400 [ 80%]  (Sampling)
## Chain 1: Iteration: 360 / 400 [ 90%]  (Sampling)
## Chain 1: Iteration: 400 / 400 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 727.859 seconds (Warm-up)
## Chain 1:                641.125 seconds (Sampling)
## Chain 1:                1368.98 seconds (Total)
## Chain 1:
```

```
## Warning: There were 13 divergent transitions after warmup. Increasing adapt_delta above 0.95 may hel
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

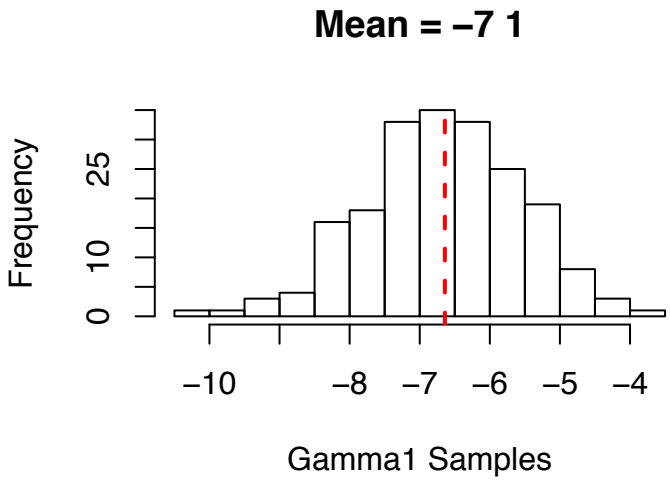
```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
fl = extract(stanout)
```

```
hist(fl$gamma0, xlab="Gamma0 Samples", main=paste("Mean =", round(mean(fl$gamma0)),1))
abline(v=mean(fl$gamma0), lwd=2, col=2, lty=2)
```



```r
hist(fl$gamma1, xlab="Gamma1 Samples", main=paste("Mean =", round(mean(fl$gamma1)),1))
abline(v=mean(fl$gamma1), lwd=2, col=2, lty=2)
```



We see that the correct coefficients are recovered.