Web Appendix A: Hamiltonian Monte Carlo

We use a fully Bayesian approach for inference. As outlined in Section 2, the joint density of all unknowns is given by

$$p(\boldsymbol{y}, \{\boldsymbol{\alpha}_k\}, \boldsymbol{\delta}, \boldsymbol{\phi}, \sigma^2) = \left[\prod_{i=1}^{I}\prod_{j=1}^{M_i} p(y_{ij}|\alpha_{ij}, \delta_i)p(\delta_i|\sigma^2)\right]\left[\prod_{k=1}^{K} p(\boldsymbol{\alpha}_k|\boldsymbol{\phi}_k)\right]p(\sigma^2)p(\boldsymbol{\phi}). \quad (1)$$

As the full posterior distribution $p(\{\boldsymbol{\alpha}_k\}, \boldsymbol{\delta}, \boldsymbol{\phi}, \sigma^2|\boldsymbol{y})$ is not available analytically, we use the Hamiltonian Monte Carlo (HMC) algorithm to draw samples of the unknown function values $\boldsymbol{\alpha}_k$, customer-specific random effects $\boldsymbol{\delta}$, population parameters σ^2, and the GP hyperparameters $\boldsymbol{\phi}$, from the posterior. For completeness, we include a brief overview of HMC here, and refer the reader to Neal (2011) for further details.

HMC is a variant of the Metropolis-Hastings algorithm that uses a proposal distribution that is based on the Hamiltonian dynamics of a particle moving in a potential field. Suppose our interest is in sampling a set of parameters $\boldsymbol{\theta} \in R^p$ (i.e., particle positions) from a target posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y})$. For our model, $\boldsymbol{\theta}$ can contain the entire set of unknown function values and GP hyperparameters. HMC uses a vector of auxilliary momentum variables $\boldsymbol{\zeta} \in R^p$ drawn from a multivariate normal $N(\boldsymbol{\zeta}|0, M)$ where the covariance matrix M is the mass matrix. Both the positions and the momentum variables are jointly sampled from a joint density $p(\boldsymbol{\theta}, \boldsymbol{\zeta}|\boldsymbol{y}) = p(\boldsymbol{\theta}|\boldsymbol{y})p(\boldsymbol{\zeta})$. The values of $\boldsymbol{\theta}$ are retained, where as the samples of $\boldsymbol{\zeta}$ are ignored. Algorithm 1 outlines a single HMC iteration.

Algorithm 1 HMC Iteration (Given stepsize ϵ, number of leapfrog steps, L mass matrix M, and $\boldsymbol{\theta}_{current}$)

Initialize $\boldsymbol{\theta}_{(0)} \leftarrow \boldsymbol{\theta}_{current}$, $\boldsymbol{\zeta}_{(0)} \sim \mathcal{N}(0, M)$

for $l = 0, \ldots, L-1$ **do** ▷ Perform Leapfrog steps
  $\boldsymbol{\zeta}_{(l+1/2)} \leftarrow \boldsymbol{\zeta}_{(l)} + \frac{1}{2}\epsilon\nabla_\theta \log p(\boldsymbol{\theta}_{(l)}|\boldsymbol{y})$
  $\boldsymbol{\theta}_{(l+1)} \leftarrow \boldsymbol{\theta}_{(l)} + \epsilon M^{-1}\boldsymbol{\zeta}_{(l+1/2)}$
  $\boldsymbol{\zeta}_{(l+1)} \leftarrow \boldsymbol{\zeta}_{(l+1/2)} + \frac{1}{2}\epsilon\nabla_\theta \log p(\boldsymbol{\theta}_{(l+1)}|\boldsymbol{y})$
end for

$r = \min\left[1, \frac{p(\boldsymbol{\theta}_{(L)}|\boldsymbol{y})\, p(\boldsymbol{\zeta}_{(L)})}{p(\boldsymbol{\theta}_{(0)}|\boldsymbol{y})p(\boldsymbol{\zeta}_{(0)})}\right]$ ▷ Compute acceptance probability
$u \sim \text{Uniform}(0, 1)$ ▷ Uniform draw
if u $< r$, **then** return $\boldsymbol{\theta}_{(L)}$ ▷ Accept or reject proposal
else return $\boldsymbol{\theta}_{(0)}$
end if

As can be seen from Algorithm 1, each iteration of the HMC algorithm involves several leapfrog steps in which $\boldsymbol{\theta}$ and $\boldsymbol{\zeta}$ evolve according to a discretization of Hamilton's equations.

The HMC sampler uses the gradient of the log-posterior to direct the exploration of the posterior. This allows it to avoid the random walk behavior of ordinary Metropolis-Hastings procedures and it therefore traverses the posterior in an efficient fashion. HMC methods are ideal for non-conjugate GP settings such as ours, as they can efficiently sample both the latent function values as well as the hyperparameters.

In practice, we need to specify values for the step size ϵ, the number of leapfrog steps L and the mass matrix M, and finding the right set of values for these can be sometimes challenging. We therefore use the No U-Turn Sampling (NUTS) variant of HMC as implemented in the Stan probabilistic programming language (Hoffman and Gelman, 2014; Carpenter et al., 2017). Stan uses an adaptive version of the HMC algorithm wherein ϵ, L and M are updated across the MCMC iterations to ensure rapid mixing, while still maintaining detailed balance. Since each iteration of HMC involves multiple leapfrog steps, an HMC iteration is not directly comparable to that of the ordinary Metropolis-Hastings algorithm, and convergence is achieved in much fewer MCMC iterations. Details of NUTS are given in Hoffman et al. (2014).

References

Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M. A., Li, P., and Riddell, A. (2017). Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, 76(1).

Hoffman, M. and Gelman, A. (2014). The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15:1351–1381.

Web Appendix B: Simulation Studies

In this appendix, we use simulated data to explore various aspects of the GPPM. In the first section, we show how the GPPM can be extended to accommodate different length-scales of variation along different time dimensions, to capture things like "loyalty" promotions, for example, that might occur along the lifetime dimension. In the second section, we explore links between the GPPM and classic buy-till-you-die (BTYD) models for customer base analysis, focusing on the BGNBD model as our example. BTYD models have served as the backbone for many customer base analysis applications, showing a particularly robust ability to forecast future spending and compute customer-centric quantities of interest by modeling just interpurchase times and customer lifetimes. The GPPM extends this framework by also allowing for the consideration of an additional input, calendar time. To explore how the GPPM generalizes these ideas, we simulate data from both models, and show first how the recency and lifetime components of the GPPM are able to capture the equivalent BTYD effects, and second why the inclusion of calendar time effects is important in accurately estimating individual-level spend rates. Since we use simulated data across all of these studies, we can see throughout examples of how the GPPM can capture the shape of events of interest automatically, as we know in these cases exactly the impact a given event.

Extending the GPPM

In Section 2.2 of the paper, we described the modular approach to specifying the GPPM. Recall that each kernel represents a broad type of functions. In the main paper, we used SE kernels to pick up variation along two length-scales, short and long, for the calendar time effects, along with a predictable periodic component. We used a single SE kernel with a monotonic power mean function to isolate variability along the other dimensions. In practice, we may want to extend the model in various ways. One potential deviation from the general model explained in the body of the paper is the need to capture short-run effects of interest that may occur along other dimensions, particularly along the lifetime dimension. These effects could exist, for instance, if the company has loyalty based rewards or promotions, such that the consumer is given a special after a certain number of days after first purchase.

To cope with shocks along the lifetime dimension, we can extend the GPPM quite simply by adding an additional SE component to the lifetime specification. By the additive property of GPs, this specification remains a GP, just with an additive kernel. Hence, we now model:

$$\alpha_{\mathrm{L}}(\ell) = \alpha_{\mathrm{L}}^{\mathrm{Long}}(\ell) + \alpha_{\mathrm{L}}^{\mathrm{Short}}(\ell),$$

where:

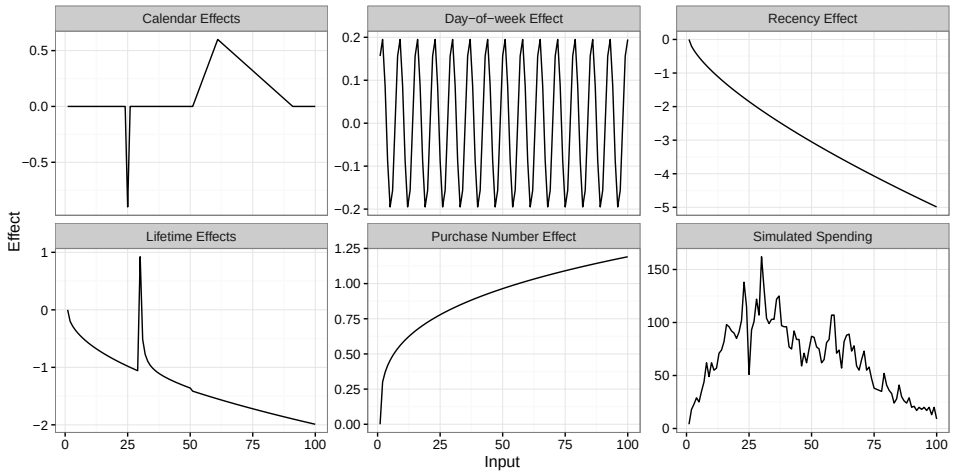

Figure 1: True effects used to generate the simulated data, with the simulated spending time series shown in the bottom right panel.

$$\alpha_{\mathrm{L}}^{\mathrm{Long}}(\ell) \sim \mathcal{GP}(m(\ell), k_{\mathrm{SE}}(\ell, \ell'; \eta_{\mathrm{LL}}, \rho_{\mathrm{LL}})),$$
$$\alpha_{\mathrm{L}}^{\mathrm{Short}}(\ell) \sim \mathcal{GP}(0, k_{\mathrm{SE}}(\ell, \ell'; \eta_{\mathrm{LS}}, \rho_{\mathrm{LS}})),$$
$$\rho_{\mathrm{LS}} < \rho_{\mathrm{LL}}$$

With this setup, we can capture short-run departures from the smooth trend component along the lifetime dimension, just like we captured both trends and short-run shocks in the calendar time component before. In this case, we include the same mean function as before (power mean) along the long-run curve.[1]

Simulation We simulated data within the GPPM framework, similar to the data from our application. We simulated the spending of 2,000 customers, entering over a period of 30 days, using the effects displayed in Figure 1. The sum of these effects results in the spending time series displayed in the bottom right panel of Figure 1. We then estimated the GPPM on this data, using the extension described above. The resulting extended dashboard is shown in Figure 2. We see that the GPPM recovers all of the effects in the data generating process, without specifying any of them as inputs to the model. More importantly, we see the natural extension of the GPPM to capture the shock to the lifetime dimension. The instantaneous effect of the loyalty reward is captured in the Lifetime, Short panel, with the residual effect slight, but noticeable in the Lifetime, Long panel.

[1]By additivity, the results would be equivalent if the mean function were included in the short-run term; however, we find the idea of a trend + shock formulation more intuitive, and hence model it as such.

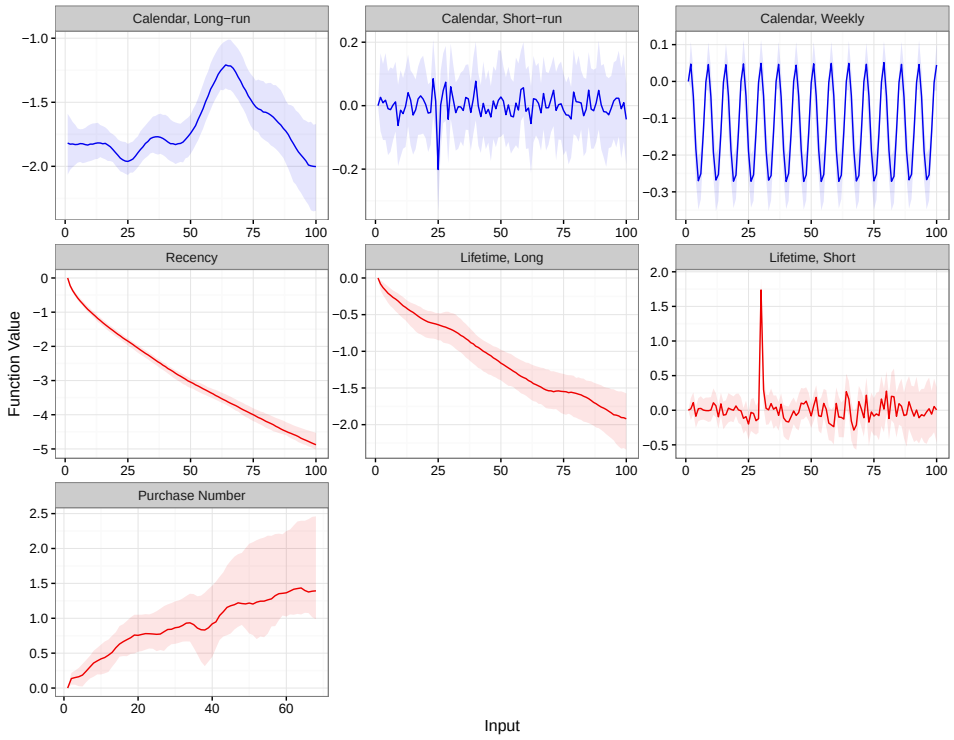

Figure 2: Extended GPPM dashboard for simulated data.

Links between GPPM and BTYD

The GPPM provides a natural generalization of buy-till-you-die customer base analysis models that rely solely on recency and lifetime, such as the BGNBD. While the GPPM does not explicitly account for customer death, it does so asymptotically by allowing the probability of purchase to go to zero via the lifetime and recency effects. To explore this link deeper, we ran a series of simulation studies, testing in which cases the GPPM is able to capture BGNBD data, and vice versa.

We hypothesize that the dynamic spending patterns that are captured by the BGNBD can also be captured by the GPPM; however, the BGNBD will have a difficult time fitting data generated by the GPPM, depending on the strength of calendar time effects present. This is because the BGNBD and other parametric probability models based on individual-level effects have no way of separating out temporary shifts in spend propensity due to calendar time effects from underlying, predictable individual-level effects. To test these two hypotheses, we first see how the GPPM does at fitting data generated by the BGNBD model. Then we do the reverse and estimate the BGNBD on data from GPPM specifications that vary the strength and nature of the calendar time effects.

BGNBD Data, GPPM Fit If the recency and lifetime components of the GPPM do capture the dynamic patterns inherent in the BGNBD, then the GPPM should be able to do

DGP	Model	Overall	Training	Holdout
BGNBD, FHL Parameters	GPPM	0.07	0.05	0.09
BGNBD, Random*	GPPM	0.10	0.06	0.14
GPPM, All*	BGNBD	0.54	0.21	0.87
GPPM, `Nocal` Only*	BGNBD	0.22	0.15	0.29

Table 1: Fit summaries for the simulation studies. The first column contains the data generating process, while the second contains the model used to forecast spending. An asterisk (*) is used to denote the statistics that are the average value across many simulations. The statistics presented are MAPE (mean absolute percentage error). RMSE is not relevant here as each simulation results in spending on a different scale, and hence RMSE is not comparable across simulations.

well on data generated from the BGNBD. To see this, we generate data from 8,000 spenders across 30 first spend dates, similar to our real data. We simulate spending over 100 days according to a BGNBD model, and then fit the GPPM on the first 50 days of simulated data, and forecast the activity on days 51 to 100. As our main example, we use the estimated BGNBD parameters ($r = 0.243$, $\alpha = 4.414$, $a = 0.793$, $b = 2.426$) from the original BGNBD paper (Fader, Hardie, and Lee, 2010, subsequently FHL). We also used many combinations of randomly generated parameters to test robustness, with smaller sample sizes of 2,000 customers. The fit statistics for all of the simulations are summarized in Table 1. The good fit offers substantial evidence to our claim that the GPPM nests these traditional probability models.

GPPM Data, BGNBD Fit We also study the reverse situation and examine the performance of the BGNBD on data generated from the GPPM. We show that BGNBD is not able to fit such data very well, especially in the presence of calendar time dynamics. Specifically, we use three levels of the day of the week effect — none (`Nocyc`), weak (`Weakcyc`), and strong (`Strongcyc`) — and three kinds of non-cyclic calendar time effects: none (`Nocal`), a long-run peak similar to the general holiday season bump seen in our application (`Peakcal`), and a nonlinear decreasing trend across the whole time period (`NonlinDeccal`). The cyclic effect was set as $\alpha_w(t) = \theta \sin(2\pi t/7)$, where $\theta = 0$, for no cyclic effect, $\theta = 0.15$, for the weak effect, and $\theta = 0.4$, for the strong effect. For the calendar time effects, the non-linear decreasing calendar time trend is given by $\alpha_{\mathrm{T}}(t) = -0.2t^{0.3}$; the peak effect is given by the piecewise function: $\alpha_{\mathrm{T}}(t) = 0$, when $t \leq 20$; $\alpha_{\mathrm{T}}(t) = 0.5(t-20)$, when $t \in [21, 40]$; $\alpha_{\mathrm{T}}(t) = 0.1(50-t)$, when $t \in [41, 50]$ and $\alpha_{\mathrm{T}}(t) = 0$, when $t > 50$.

Figure 3 and Table 1 show the results from these simulations. We see that BGNBD fits the mean of the curve in the presence of a cyclic effect. We also see that the BGNBD generally does well in the cases where there is no short or long-run calendar variation, underpredicts in the beginning and then overpredicts in the end when there is a decreasing calendar time effect, and fails significantly at capturing the peak effect. We see in the `Peakcal` case (last row of Figure 3) that the BGNBD attributes the peak to higher rates of spending, and then dramatically overestimates future spending.

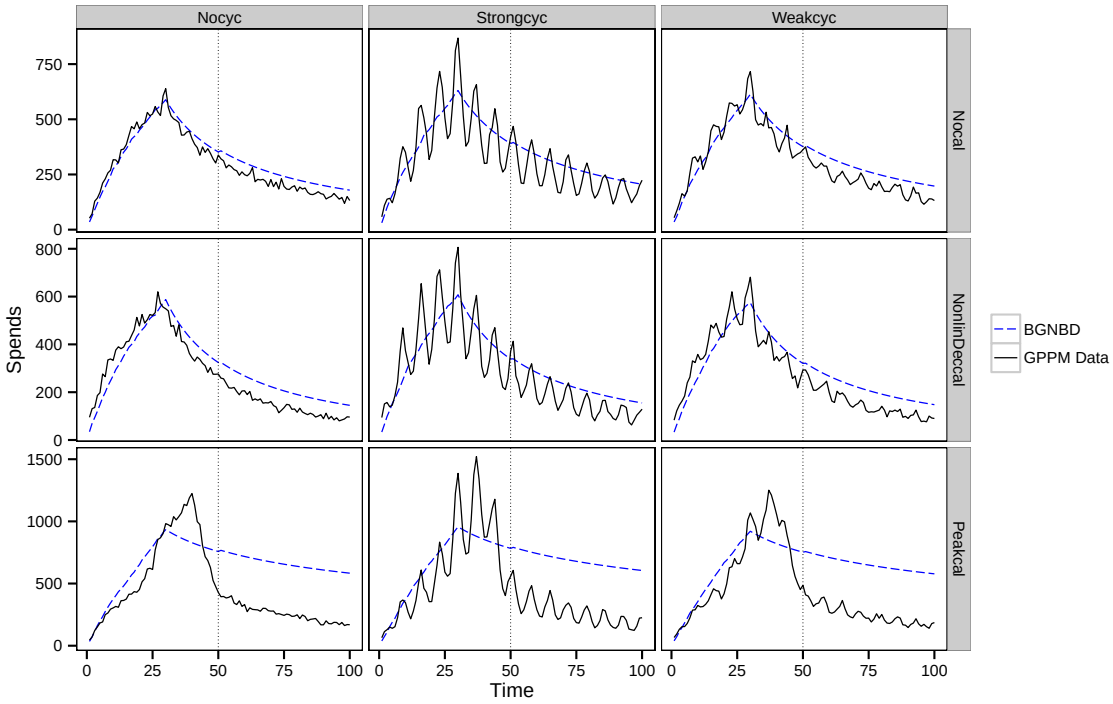

Figure 3: The BGNBD fit on various types of data drawn from the GPPM: `Nocyc`, `Strongcyc`, and `Weakcyc` indicate no, strong, and weak cyclic (day of the week) effects respectively; `Nocal` indicates no calendar time dynamics, `NonlinDeccal` indicates a non-linear decreasing long-run calendar time process, and `Peakcal` indicates a calendar time process that is flat but with a peak during the calibration period.

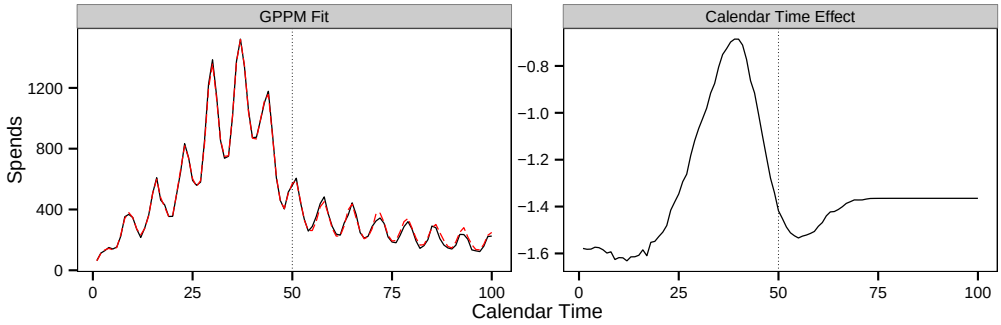

Figure 4: The GPPM fit and forecast on the `Strongcyc/Peakcal` simulated data, together with the estimated calendar time effect. We see that the GPPM captures the pointed piecewise effect, and is therefore able to isolate the predictable, individual-level effects that allow it to accurately forecast future spending.

The GPPM does not fall prey to this same bias because of its ability to separate out calendar time effects. To emphasize this, we see the GPPM fit to the worst case (`Strongcyc/Peakcal`), together with the estimated calendar time effect, in Figure 4. The excellent fit and near perfect forecast is not surprising: the GPPM is capturing data generated from a GPPM. One thing to point out is that this, again, demonstrates the ability of the GPPM to nonparametrically recover the effects of events, as we see the peak in calendar time is equivalent to the piecewise function described above.

Web Appendix C: Stan Code

```
data{
  int<lower=1> P; // number of periods
  int<lower=1> N; // number of customers
  int<lower=1> M; // number of data points
  int<lower=1> K; // max number of purchases
  int<lower=1> t[M];    // calendar time
  int<lower=1> l[M];    // lifetime
  int<lower=1> r[M];    // interpurchase time (recency)
  int<lower=1> pnum[M]; // purchase number
  int<lower=1> id[M]; // customer id of observation m
  int<lower=0,upper=1> y[M]; // outcome of observation m (purchase or not)
}
parameters{
  // non-restricted components of the GP's
  vector[P] alpha_long;
  vector[P-1] free_week;
  vector[P-1] free_short;
  vector[P-1] free_rec;
  vector[P-1] free_life;
  vector[K-1] free_pnum;

  // unobs. heterogeneity, random effects
  vector[N] delta;
  real<lower=0> sigsq;

  // kernel hyperparameters
  real<lower=0> etasq_week;
  real<lower=0> rhosq_week;
  real<lower=0> etasq_short;
  real<lower=0> etasq_long;
  positive_ordered[2] rhosq_cal;
  real<lower=0> etasq_rec;
  real<lower=0> rhosq_rec;
  real<lower=0> etasq_life;
  real<lower=0> rhosq_life;
  real<lower=0> etasq_pnum;
  real<lower=0> rhosq_pnum;

  // mean function hyperparameters
  real mu; // baseline spending level (cal mean function)
  real lambda_rec1;
  real<lower=0> lambda_rec2;
  real lambda_life1;
  real<lower=0> lambda_life2;
  real lambda_pnum1;
  real<lower=0> lambda_pnum2;
}
transformed parameters{
  // full GPs (with the first period restriction)
  vector[P] alpha_week;
  vector[P] alpha_short;
  vector[P] alpha_rec;
  vector[P] alpha_life;
  vector[K] alpha_pnum;
  vector[P] week_mean;
  vector[P] short_mean;
  vector[P] long_mean;
  vector[P] rec_mean;
  vector[P] life_mean;
  vector[K] pnum_mean;
  vector[1] z;

  // add zero first period restriction
  z[1] <- 0;
```

```
  alpha_week <- append_row(z,free_week);
  alpha_short <- append_row(z,free_short);
  alpha_rec <- append_row(z,free_rec);
  alpha_life <- append_row(z,free_life);
  alpha_pnum <- append_row(z,free_pnum);

  // set mean functions
  for(p in 1:P) {
    rec_mean[p] <- lambda_rec1*(p-1)^lambda_rec2;
    life_mean[p] <- lambda_life1*(p-1)^lambda_life2;
    short_mean[p] <- 0;
    long_mean[p] <- mu;
    week_mean[p] <- 0; }

  for(k in 1:K)
    pnum_mean[k] <- lambda_pnum1*(k-1)^lambda_pnum2;
}
model{
  vector[M] theta;
  matrix[P,P] Sigma_short;
  matrix[P,P] L_short;
  matrix[P,P] Sigma_long;
  matrix[P,P] L_long;
  matrix[P,P] Sigma_week;
  matrix[P,P] L_week;
  matrix[P,P] Sigma_rec;
  matrix[P,P] L_rec;
  matrix[P,P] Sigma_life;
  matrix[P,P] L_life;
  matrix[K,K] Sigma_pnum;
  matrix[K,K] L_pnum;

  // calendar time, cyclic component kernel
  for(i in 1:P) {
    for(j in 1:P) {
      Sigma_week[i,j] <- etasq_week * exp(-2*(sin(pi()*(i-j)/7))^2/rhosq_week);
      Sigma_week[j,i] <- Sigma_week[i,j]; }
    Sigma_week[i,i] <- Sigma_week[i,i]+0.0001; }
  L_week <- cholesky_decompose(Sigma_week);

  // calendar time, short-run component kernel
  for(i in 1:P) {
    for(j in 1:P) {
      Sigma_short[i,j] <- etasq_short * exp(-((i-j)^2)/rhosq_cal[1]);
      Sigma_short[j,i] <- Sigma_short[i,j]; }
    Sigma_short[i,i] <- Sigma_short[i,i]+0.0001; }
  L_short <- cholesky_decompose(Sigma_short);

  // calendar time, long-run component kernel
  for(i in 1:P) {
    for(j in 1:P) {
      Sigma_long[i,j] <- etasq_long * exp(-((i-j)^2)/rhosq_cal[2]);
      Sigma_long[j,i] <- Sigma_long[i,j]; }
    Sigma_long[i,i] <- Sigma_long[i,i]+0.0001; }
  L_long <- cholesky_decompose(Sigma_long);

  // recency kernel
  for(i in 1:P) {
    for(j in 1:P) {
      Sigma_rec[i,j] <- etasq_rec * exp(-((i-j)^2)/rhosq_rec);
      Sigma_rec[j,i] <- Sigma_rec[i,j]; }
    Sigma_rec[i,i] <- Sigma_rec[i,i]+0.0001; }
  L_rec <- cholesky_decompose(Sigma_rec);

  // lifetime kernel
  for(i in 1:P) {
    for(j in 1:P) {
      Sigma_life[i,j] <- etasq_life * exp(-((i-j)^2)/rhosq_life);
```

```
      Sigma_life[j,i] <- Sigma_life[i,j]; }
    Sigma_life[i,i] <- Sigma_life[i,i]+0.0001; }
  L_life <- cholesky_decompose(Sigma_life);

  // pnum kernel
  for(i in 1:K) {
    for(j in 1:K) {
      Sigma_pnum[i,j] <- etasq_pnum * exp(-((i-j)^2)/rhosq_pnum);
      Sigma_pnum[j,i] <- Sigma_pnum[i,j]; }
    Sigma_pnum[i,i] <- Sigma_pnum[i,i]+0.0001; }
  L_pnum <- cholesky_decompose(Sigma_pnum);

  // kernel hyperparameter priors
  etasq_week ~ normal(0,5);
  rhosq_week ~ normal(P/2,P);
  etasq_short ~ normal(0,5);
  etasq_long ~ normal(0,5);
  rhosq_cal ~ normal(P/2,P);
  etasq_rec ~ normal(0,5);
  rhosq_rec ~ normal(P/2,P);
  etasq_life ~ normal(0,5);
  rhosq_life ~ normal(P/2,P);
  etasq_pnum ~ normal(0,5);
  rhosq_pnum ~ normal(20,10);

  // kernel mean function priors
  mu ~ normal(0,5);
  lambda_rec1 ~ normal(0,5);
  lambda_pnum1 ~ normal(0,5);
  lambda_life1 ~ normal(0,5);
  lambda_rec2 ~ normal(0,5);
  lambda_pnum2 ~ normal(0,5);
  lambda_life2 ~ normal(0,5);

  // sample GPs
  alpha_week ~ multi_normal_cholesky(week_mean,L_week);
  alpha_short ~ multi_normal_cholesky(short_mean,L_short);
  alpha_long ~ multi_normal_cholesky(long_mean,L_long);
  alpha_rec ~ multi_normal_cholesky(rec_mean,L_rec);
  alpha_life ~ multi_normal_cholesky(life_mean,L_life);
  alpha_pnum ~ multi_normal_cholesky(pnum_mean,L_pnum);

  // sample random effects
  delta ~ normal(0,sigsq);
  sigsq ~ normal(0,2.5);

  // gppm likelihood
  for(m in 1:M)
    theta[m] <- alpha_week[t[m]]+alpha_long[t[m]]+alpha_short[t[m]]+
                alpha_life[l[m]]+alpha_rec[r[m]]+alpha_pnum[pnum[m]]+
                delta[id[m]];

  y ~ bernoulli_logit(theta);
}
```