Hierarchical Gaussian Processes in Stan

Rob Trangucci

Introduction

Stan's library has been expanded with functions that facilitate adding Gaussian processes (GPs) to Stan models. I will share the best practices for coding GPs in Stan, and demonstrate how GPs can be added as one component of a larger model.

I would like to thank Aki Vehtari, Michael Betancourt, and the reviewers for their helpful comments and suggestions.

Gaussian processes regression

Suppose there are N observations of univariate data, y, each associated with scalar x.

The generative model for a Gaussian process regression is as follows:

$$\begin{aligned}\theta &\sim g(\phi)\\ f(x) &\sim \text{GP}\left(\mu(x), k_\theta(x)\right)\\ y_i &\sim \mathcal{N}\left(f(x_i), \sigma\right) \forall i\end{aligned}$$

A GP is a stochastic process, indexed by $x \in \mathbb{R}$. Any finite sample realized from this stochastic process is jointly multivariate normal (Cramér and Leadbetter 2004). $\mu(x)$ is the mean of $f(x)$, and $k_\theta(x)$, a kernel, defines the covariance between any two evaluations of $f(x)$:

$$\text{cov}(f(x_i), f(x_j)) = k(x_i, x_j|\theta)$$

The kernel k is parameterized by a vector θ, and is required to a be a positive-semidefinite function (Rasmussen and Williams 2005).

The finite-dimensional generative model for the GP is:

$$\begin{aligned}\theta &\sim g(\phi)\\ f &\sim \text{MultiNormal}(0, K_\theta(x))\\ y_i &\sim \text{Normal}(f_i, \sigma) \forall i \in \{1, \ldots, N\}\end{aligned}$$

Above, $K_\theta(x)$ is an $N \times N$ covariance matrix, where each entry $K[i, j] = k(x_i, x_j|\theta)$. The exponentiated quadratic kernel has two components to theta, α, the marginal standard deviation of the stochastic process f and ℓ, the process length-scale.

$$k(x_i, x_j|\theta) = \alpha^2 \exp\left(-\frac{1}{2\ell^2}(x_i - x_j)^2\right)$$

This kernel's defining quality is its smoothness; the function is infinitely differentiable. This can sometimes be unrealistic for applied work (see (Stein 2012)), but it will suffice for the examples that follow, as it happens to be the only covariance function that has been implemented in the Stan library as of January 2017.

Example: GP with normal outcome

Latent variable formulation

The Stan program for the generative model is defined exactly as the finite-dimensional GP probability model above (of course, choosing priors for ℓ and α).

```
functions {
  vector gp_pred_rng(real[] x_pred,
                     vector y_is,
                     real[] x_is,
                     real alpha,
                     real length_scale,
                     real sigma) {
    vector[size(x_pred)] f_pred;
    int N_pred;
    int N;
    N_pred = size(x_pred);
    N = rows(y_is);

    {
      matrix[N, N] L_Sigma;
      vector[N] K_div_y_is;
      matrix[N, N_pred] k_x_is_x_pred;
      matrix[N, N_pred] v_pred;
      vector[N_pred] f_pred_mu;
      matrix[N_pred, N_pred] cov_f_pred;
      matrix[N_pred, N_pred] nug_pred;
      matrix[N, N] Sigma;
      Sigma = cov_exp_quad(x_is, alpha, length_scale);
      for (n in 1:N)
        Sigma[n, n] = Sigma[n,n] + square(sigma);
      L_Sigma = cholesky_decompose(Sigma);
      K_div_y_is = mdivide_left_tri_low(L_Sigma, y_is);
      K_div_y_is = mdivide_right_tri_low(K_div_y_is',L_Sigma)';
      k_x_is_x_pred = cov_exp_quad(x_is, x_pred, alpha, length_scale);
      f_pred_mu = (k_x_is_x_pred' * K_div_y_is);
      v_pred = mdivide_left_tri_low(L_Sigma, k_x_is_x_pred);
      cov_f_pred = cov_exp_quad(x_pred, alpha, length_scale) - v_pred' * v_pred;
      nug_pred = diag_matrix(rep_vector(1e-12,N_pred));

      f_pred = multi_normal_rng(f_pred_mu, cov_f_pred + nug_pred);
    }
    return f_pred;
  }
}
data {
  int<lower=1> N;
  int<lower=1> N_pred;
  vector[N] y;
  real x[N];
  real x_pred[N_pred];
}
parameters {
```

```
  real<lower=0> length_scale;
  real<lower=0> alpha;
  real<lower=0> sigma;
  vector[N] eta;
}
transformed parameters {
  vector[N] f;
  {
    matrix[N, N] L;
    matrix[N, N] K;
    K = cov_exp_quad(x, alpha, length_scale);
    for (n in 1:N)
      K[n, n] = K[n, n] + 1e-12;
    L = cholesky_decompose(K);
    f = L * eta;
  }
}
model {
  length_scale ~ gamma(2, 20);
  alpha ~ normal(0, 1);
  sigma ~ normal(0, 1);
  eta ~ normal(0, 1);
  y ~ normal(f, sigma);
}
generated quantities {
  vector[N_pred] f_pred;
  vector[N_pred] y_pred;

  f_pred = gp_pred_rng(x_pred, y, x, alpha, length_scale, sigma);
  for (n in 1:N_pred)
    y_pred[n] = normal_rng(f_pred[n], sigma);
}
```

Several details of the Stan program are important to note. First is the generation of the covariance matrix with an exponentiated quadratic kernel. The function `cov_exp_quad` will generate an $N \times N$ covariance matrix if given a length-N array of either reals or vectors, the signal standard deviation, α, and the length-scale ℓ.

The second detail is the small positive number added to the diagonal. In this case it is 1e-12. Because we're using a positive-definite function k to build a covariance matrix, the resulting matrix should theoretically be positive definite. However, because we deal in floating-point numbers, covariance matrices generated by `cov_exp_quad` beyond a small dimension will typically not be numerically positive definite. In that case, we need to force the matrix to be positive definite by adding a small bit of noise to the diagonal called jitter. This will conflict with the parameter σ. This is OK for our purposes, as the amount of noise we've added is quite small compared to the scale of σ. In most real-world settings where we have noisy observations from a GP, the scale of the jitter will be small compared to the noise.

The third detail is that we have Cholesky decomposed the covariance matrix K with `cholesky_decompose`. In any applied application, the finite dimensional sample from a GP is ultimately a multivariate normal with a parameterized covariance matrix. As such, we will need to invert the matrix, or decompose it in some way in order to add a multivariate normal density over f to our log-posterior density. It turns out that the Cholesky decomposition is the best way to decompose that matrix in Stan right now.

We could have taken our Cholesky factor L of the covariance matrix K above and used the function `multi_normal_cholesky` to add a multivariate normal density over f to the log-posterior. Instead, the fourth detail to note above is that we've multiplied the Cholesky factor of the covariance matrix by a vector of univariate normals *eta* so f is implicitly distributed as a multivariate normal random variable. This is

called the non-centered parameterization of a multivariate normal. Suppose we have a covariance matrix, Σ. Because it is a proper covariance matrix, there exists a lower-triangular matrix L such that $L \times L^T = \Sigma$. We know that:

$$
\begin{aligned}
L \times L^T &= \Sigma \\
\eta_i &\sim \text{Normal}(0, 1) \, \forall i \in \{1, \ldots, N\} \\
f &= L\eta \\
f &\sim \text{MultiNormal}(0, \Sigma)
\end{aligned}
$$

The reason to express f as a `transformed_parameter` is because it removes the prior dependence of the density of f on α and ℓ. When the data are weakly informative about the f, this can aid in sampling efficiently from the joint posterior. See Betancourt and Girolami's excellent paper (Betancourt and Girolami 2013) for more color on the univariate non-centered parameterization. As with any Stan program, we should generate some fake data from a model with fixed parameters and see whether we can recover our parameters.

```
data {
  int<lower=1> N;
  real<lower=0> length_scale;
  real<lower=0> alpha;
  real<lower=0> sigma;
}
transformed data {
  vector[N] zeros;
  zeros = rep_vector(0, N);
}
model {}
generated quantities {
  real x[N];
  vector[N] y;
  vector[N] f;
  for (n in 1:N)
    x[n] = uniform_rng(-2,2);
  {
    matrix[N, N] cov;
    matrix[N, N] L_cov;
    cov = cov_exp_quad(x, alpha, length_scale);
    for (n in 1:N)
      cov[n, n] = cov[n, n] + 1e-12;
    L_cov = cholesky_decompose(cov);
    f = multi_normal_cholesky_rng(zeros, L_cov);
  }
  for (n in 1:N)
    y[n] = normal_rng(f[n], sigma);
}
```

```
dat_list <- list(N = 2000, alpha = 1, length_scale = 0.15, sigma = sqrt(0.1))
set <- sample(1:dat_list$N,size = 30, replace = F)
draw <- sampling(sim_data_model,iter=1,algorithm='Fixed_param', chains = 1, data = dat_list,
                 seed = 363360090)
```

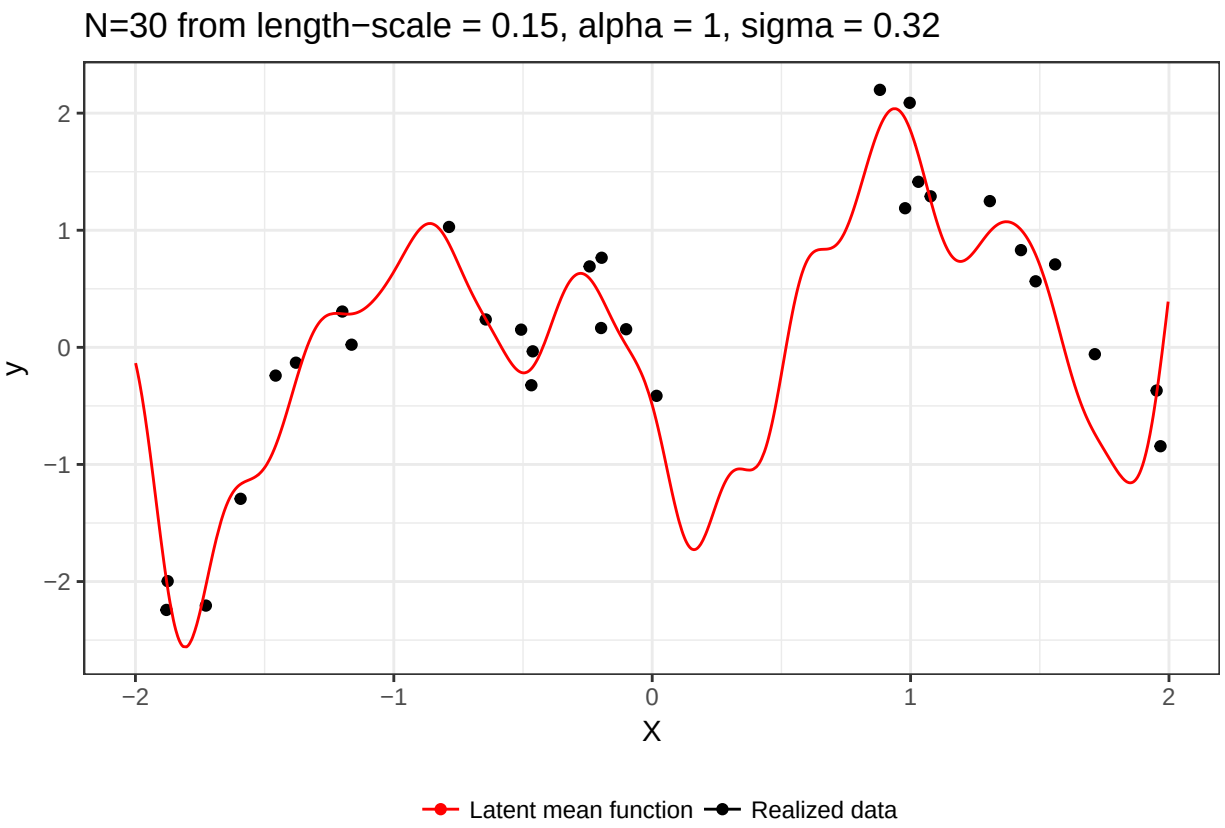
```
##
## SAMPLING FOR MODEL 'sim_gp_latent' NOW (CHAIN 1).
##
## Chain 1, Iteration: 1 / 1 [100%]  (Sampling)
```

```
##  Elapsed Time: 4e-06 seconds (Warm-up)
##                0.304017 seconds (Sampling)
##                0.304021 seconds (Total)
```

```
samps <- rstan::extract(draw)
plt_df = with(samps,data.frame(x = x[1,], y = y[1,], f = f[1,]))
```

Here's the data:

```
ggplot(data = plt_df[set,], aes(x=x, y=y)) +
  geom_point(aes(colour = 'Realized data')) +
  geom_line(data = plt_df, aes(x = x, y = f, colour = 'Latent mean function')) +
   theme_bw() + theme(legend.position="bottom") +
  scale_color_manual(name = '', values = c('Realized data'='black','Latent mean function'='red')) +
  xlab('X') +
  ylab('y') +
  ggtitle(paste0('N=',length(set),' from length-scale = 0.15, alpha = 1, sigma = 0.32'))
```



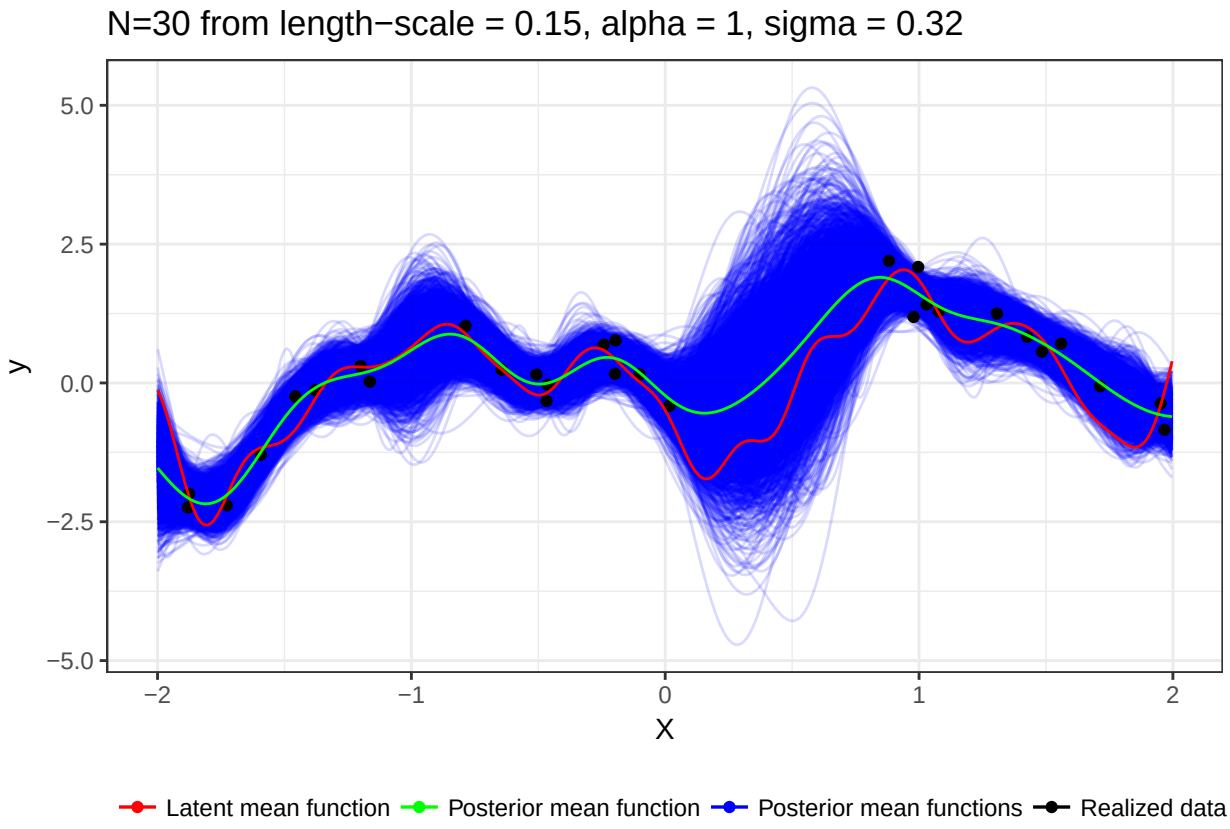
We prep the data for modeling and inference in Stan:

```
stan_data <- list(N = length(set), N_pred = dat_list$N - length(set),
                  x = samps$x[1,set], y = samps$y[1,set],
                  x_pred = samps$x[1,-set], f_pred = samps$f[1,-set])
```

We'll run 4 chains for 2000 iterations each, with `adapt_delta` set to 0.95.

```
gp_mod_lat <- sampling(comp_gp_mod_lat, data = stan_data, cores = 4, chains = 4, iter = 2000, control =
samps_gp_mod_lat <- extract(gp_mod_lat)
post_pred <- data.frame(x = stan_data$x_pred,
                        pred_mu = colMeans(samps_gp_mod_lat$f_pred))
```

```
plt_df_rt = data.frame(x = stan_data$x_pred, f = t(samps_gp_mod_lat$f_pred))
plt_df_rt_melt = melt(plt_df_rt,id.vars = 'x')
p <- ggplot(data = plt_df[set,], aes(x=x, y=y)) +
  geom_line(data = plt_df_rt_melt, aes(x = x, y = value, group = variable, colour = 'Posterior mean fun
  geom_point(aes(colour = 'Realized data')) +
  geom_line(data = plt_df, aes(x = x, y = f, colour = 'Latent mean function')) +
  geom_line(data = post_pred, aes(x = x, y = pred_mu, colour = 'Posterior mean function')) +
   theme_bw() + theme(legend.position="bottom") +
  scale_color_manual(name = '', values = c('Realized data'='black','Latent mean function'='red','Poster
  xlab('X') +
  ylab('y') +
  ggtitle(paste0('N=',length(set),' from length-scale = 0.15, alpha = 1, sigma = 0.32'))
p
```



How does the model do at capturing out-of-sample data? One way to examine the model's use in quantifying the uncertainty in predicting out-of-sample data is to measure how many out-of-sample data points fall into a certain posterior predictive interval. To that end, let's quantify the 50% and the 95% posterior predictive intervals generated by our model in the generated quantities block, `y_pred`.

```
ppc_interval_df <- function(yrep, y) {
  q_95 <- apply(yrep,2,quantile,0.95)
  q_75 <- apply(yrep,2,quantile,0.75)
  q_50 <- apply(yrep,2,median)
  q_25 <- apply(yrep,2,quantile,0.25)
  q_05 <- apply(yrep,2,quantile,0.05)
  mu <- colMeans(yrep)
  df_post_pred <- data.frame(y_obs = y,
                             q_95 = q_95,
```

```
                              q_75 = q_75,
                              q_50 = q_50,
                              q_25 = q_25,
                              q_05 = q_05,
                              mu = mu)
  return(df_post_pred)
}
ppc_interval_norm_df <- function(means, sds, y) {
  q_95 <- qnorm(0.95,mean = means, sd = sds)
  q_75 <- qnorm(0.75,mean = means, sd = sds)
  q_50 <- qnorm(0.5,mean = means, sd = sds)
  q_25 <- qnorm(0.25,mean = means, sd = sds)
  q_05 <- qnorm(0.05,mean = means, sd = sds)
  df_post_pred <- data.frame(y_obs = y,
                              q_95 = q_95,
                              q_75 = q_75,
                              q_50 = q_50,
                              q_25 = q_25,
                              q_05 = q_05,
                              mu = means)
  return(df_post_pred)
}
interval_cover <- function(upper, lower, elements) {
  return(mean(upper >= elements & lower <= elements))
}
```

```
ppc_full_bayes <- ppc_interval_df(samps_gp_mod_lat$y_pred, samps$y[1,-set])
```

85% of out-of-sample data points are in the 90% posterior predictive interval.

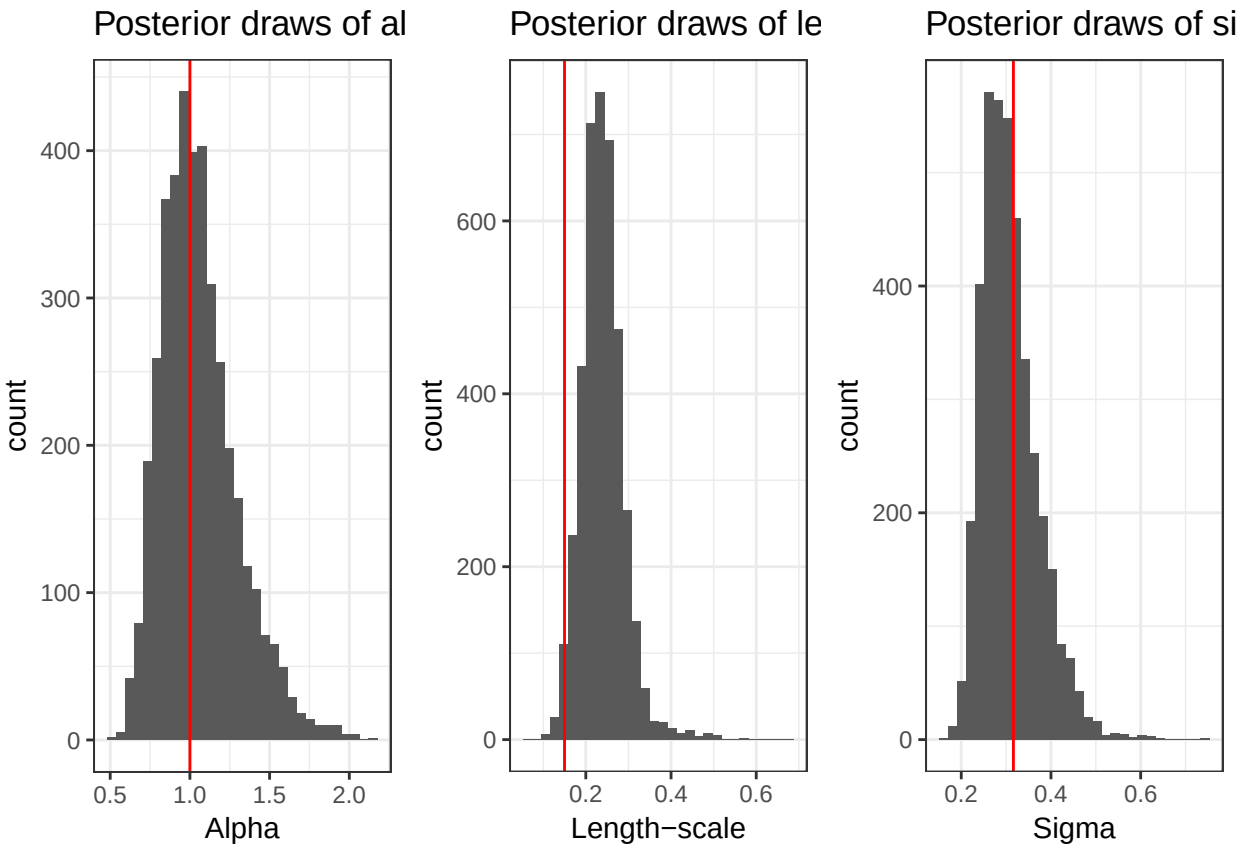
42% of out-of-sample data points are in the central 50% posterior predictive interval.

We'll also check the posterior samples for our unknown hyperparameters, ℓ, σ, and α.

```
df1 <- data.frame(x = samps_gp_mod_lat$alpha)
p1 <- ggplot(data = df1,
             aes(x = x)) + geom_histogram() + theme_bw() +
  labs(title = 'Posterior draws of alpha') +
       xlab('Alpha') + geom_vline(xintercept = dat_list$alpha,colour = 'red')
df2 <- data.frame(x = samps_gp_mod_lat$length_scale)
p2 <- ggplot(data = df2,
             aes(x = x)) + geom_histogram() + theme_bw() +
  labs(title = 'Posterior draws of length-scale') +
       xlab('Length-scale') + geom_vline(xintercept = dat_list$length_scale,colour = 'red')

df3 <- data.frame(x = samps_gp_mod_lat$sigma)
p3 <- ggplot(data = df3,
             aes(x = x)) + geom_histogram() + theme_bw() +
  labs(title = 'Posterior draws of sigma') +
        geom_vline(xintercept = dat_list$sigma,colour = 'red') + xlab('Sigma')

multiplot(p1, p2, p3, cols = 3)
```



All three of the known parameters lie in areas of large posterior mass.

To marginalize or to maximize?

We've taken it for granted that we want to marginalize over the hyperparameters rather than using a frequentist solution like is so often advocated in the literature (often due to computational constraints!). Let's examine this decision more closely. First, I'll take a quick diversion to learn how we can reparameterize the GP with a normal outcome that will make the model more computationally efficient and make the model more amenable to penalized maximum likelihood estimation of the hyperparameters.

Marginal likelihood formulation

We can also express the GP with a normal likelihood like so:

$$p(y|\alpha, \ell, \sigma) = \int p(y|f, \sigma)p(f|\alpha, \ell)df$$

It turns out that $p(y|\alpha, \ell, \sigma)$ is multivariate normal with with a covariance matrix parameterized by a new kernel that integrates the noise σ with the exponentiated quadratic:

$$k(x_i, x_j|\theta) = \alpha^2 \exp\left(-\frac{1}{2\ell^2}(x_i - x_j)^2\right) + \delta_{ij}\sigma^2$$

Above, δ_{ij} is the Dirac delta function, taking the value of 1 when $i = j$ and remaining zero otherwise.

Thus, the generative model is now:

$$\begin{aligned}\mathbf{y} &\sim \text{MultiNormal}(0, K_{\ell,\alpha}(\mathbf{x}, \mathbf{x}) + \sigma^2 I_n) \\ \mathbf{y}, \mathbf{x} &\in \mathbb{R}^N\end{aligned}$$

This is a much lower-dimensional representation of a GP. It takes the model from a $N + 3$ dimensional parameter space to a 3-dimensional parameter space. This will reduce the memory requirements substantially for our inference on the hyperparameters, as well as decreasing the dimension of the gradients.

We code that in Stan as follows, omitting the function block and the generated quantities block:

```
data {
  int<lower=1> N;
  int<lower=1> N_pred;
  vector[N] y;
  real x[N];
  real x_pred[N_pred];
}
transformed data {
  vector[N] zeros;

  zeros = rep_vector(0, N);
}
parameters {
  real<lower=0> length_scale;
  real<lower=0> alpha;
  real<lower=0> sigma;
}
model {
  matrix[N, N] L_cov;
  {
    matrix[N, N] cov;
    cov = cov_exp_quad(x, alpha, length_scale);
    for (n in 1:N)
      cov[n, n] = cov[n, n] + square(sigma);
    L_cov = cholesky_decompose(cov);
  }
  y ~ multi_normal_cholesky(zeros, L_cov);
}
```

Let's find the MLEs for α, δ, and ℓ. We can do this easily in RStan by calling `optimizing` on the compiled model. The call to optimizing runs L-BFGS to arrive at point-estimates for our hyperparameters.

```
marg_model_opt <- optimizing(marg_model, data = stan_data)
```

The estimates for the parameters is shown below.

```
marg_model_opt$par
```

```
## length_scale        alpha        sigma
##    0.6448427    1.3467679    0.3891376
```

We can see that these estimates are quite biased, but this is to be expected because we have a finite sample of data.

```
stan_data_marg <- stan_data
stan_data_marg$length_scale = marg_model_opt$par['length_scale']
stan_data_marg$alpha = marg_model_opt$par['alpha']
stan_data_marg$sigma = marg_model_opt$par['sigma']
```

```
functions {
  matrix gp_pred_rng(real[] x_pred,
                     vector y_is,
                     real[] x_is,
                     real alpha,
                     real length_scale,
                     real sigma) {
    matrix[2,size(x_pred)] f_pred;
    int N_pred;
    int N;
    N_pred = size(x_pred);
    N = rows(y_is);

    {
      matrix[N, N] L_Sigma;
      vector[N] K_div_y_is;
      matrix[N, N_pred] k_x_is_x_pred;
      matrix[N, N_pred] v_pred;
      vector[N_pred] f_pred_mu;
      matrix[N_pred, N_pred] cov_f_pred;
      matrix[N_pred, N_pred] nug_pred;
      matrix[N, N] Sigma;
      Sigma = cov_exp_quad(x_is, alpha, length_scale);
      for (n in 1:N)
        Sigma[n, n] = Sigma[n,n] + square(sigma);
      L_Sigma = cholesky_decompose(Sigma);
      K_div_y_is = mdivide_left_tri_low(L_Sigma, y_is);
      K_div_y_is = mdivide_right_tri_low(K_div_y_is',L_Sigma)';
      k_x_is_x_pred = cov_exp_quad(x_is, x_pred, alpha, length_scale);
      f_pred_mu = (k_x_is_x_pred' * K_div_y_is);
      v_pred = mdivide_left_tri_low(L_Sigma, k_x_is_x_pred);
      cov_f_pred = cov_exp_quad(x_pred, alpha, length_scale) - v_pred' * v_pred;

      f_pred[1,] = f_pred_mu';
      for (n in 1:N_pred)
        f_pred[2,n] = sqrt(cov_f_pred[n,n] + square(sigma));
    }
    return f_pred;
  }
}
data {
  int<lower=1> N;
  int<lower=1> N_pred;
  vector[N] y;
  real x[N];
  real x_pred[N_pred];
  real<lower=0> sigma;
  real<lower=0> length_scale;
  real<lower=0> alpha;
```

```
}
model {
}
generated quantities {
  matrix[2,N_pred] f_pred;

  f_pred = gp_pred_rng(x_pred, y, x, alpha, length_scale, sigma);
}
```

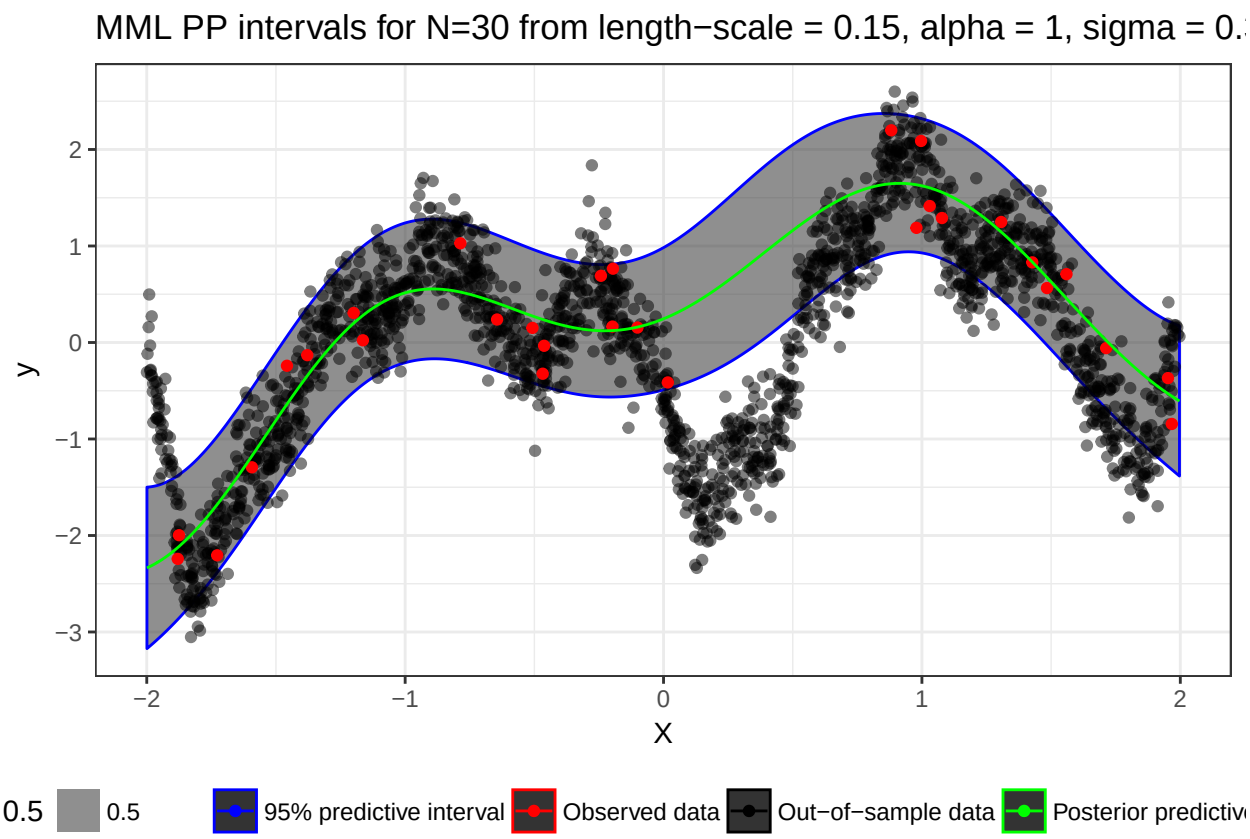
```
samps_marg <- rstan::extract(marg_draws)
ppc_max_marg <- ppc_interval_norm_df(samps_marg$f_pred[1,1,], samps_marg$f_pred[1,2,], samps$y[1,-set])
```

72% of out-of-sample data points are in the 90% posterior predictive interval.

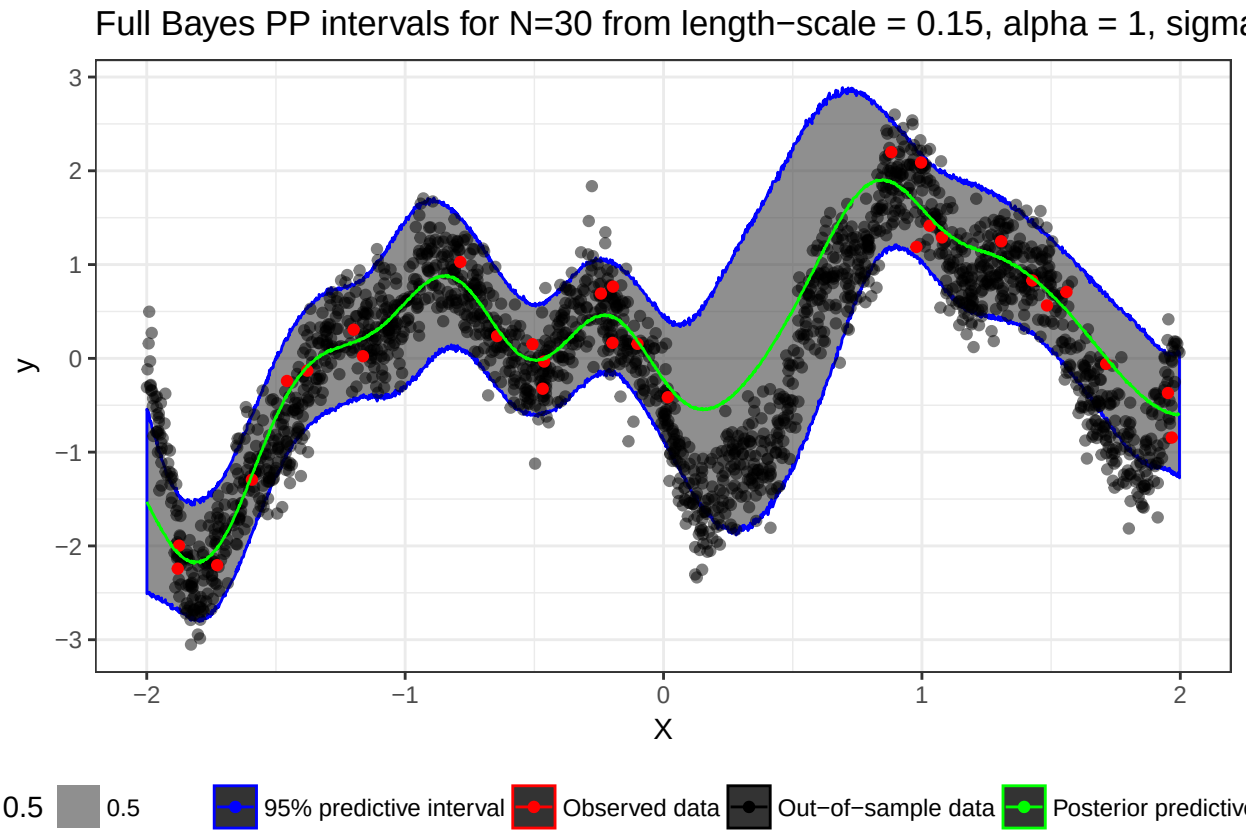
35% of out-of-sample data points are in the 50% posterior predictive interval.

Clearly the intervals do not provide adequate cover compared to their full Bayesian counterparts. The following plots will elucidate why we should put priors on our hyperparameters and integrate over the uncertainty in our hyperparameter estimates.

```
ppc_max_marg$x <- samps$x[1,-set]
ggplot(data = ppc_max_marg, aes(x = x, y = y_obs)) +
  geom_ribbon(aes(ymax = q_95, ymin = q_05,alpha=0.5, colour = '95% predictive interval')) + geom_point
  geom_point(data = plt_df[set,], aes(x = x, y = y, colour='Observed data')) + theme(legend.position="b
  geom_line(data = ppc_max_marg, aes(x = x, y = mu, colour = 'Posterior predictive mean')) +
  scale_color_manual(name = '', values = c('Observed data'='red',
                                           '95% predictive interval'='blue',
                                           'Out-of-sample data'= 'black',
                                           'Posterior predictive mean'='green')) +
  xlab('X') +
  ylab('y') +
  ggtitle(paste0('MML PP intervals for N=',length(set),' from length-scale = 0.15, alpha = 1, sigma = 0
```




```
ppc_full_bayes$x <- samps$x[1,-set]
ggplot(data = ppc_full_bayes, aes(x = x, y = y_obs)) +
  geom_ribbon(aes(ymax = q_95, ymin = q_05,alpha=0.5, colour = '95% predictive interval')) + geom_point
  geom_point(data = plt_df[set,], aes(x = x, y = y, colour='Observed data')) + theme(legend.position="b
  geom_line(data = ppc_full_bayes, aes(x = x, y = mu, colour = 'Posterior predictive mean')) +
  scale_color_manual(name = '', values = c('Observed data'='red',
                                           '95% predictive interval'='blue',
                                           'Out-of-sample data'= 'black',
                                           'Posterior predictive mean'='green')) +
  xlab('X') +
  ylab('y') +
  ggtitle(paste0('Full Bayes PP intervals for N=',length(set),' from length-scale = 0.15, alpha = 1, sig
```



GP with Poisson outcome

Let's say we have count data now, and we want to fit the data using a GP prior for the latent mean.

The generative model for the data will be nearly identical to our normal latent variable model:

$$
\begin{aligned}
\theta &\sim g(\phi) \\
f &\sim \text{MultiNormal}(0, K_\theta(x)) \\
y_i &\sim \text{Poisson}(\exp(f_i)) \, \forall i \in \{1, \ldots, N\}
\end{aligned}
$$

Luckily, we can reuse much of our code from the normal latent variable model while removing the σ parameter and changing our likelihood to a Poisson with a log link.

```
data {
  int<lower=1> N;
  int<lower=1> N_pred;
  int y[N];
  real x[N];
  real x_pred[N_pred];
}
transformed data {
  int<lower=1> N_tot;
  int<lower=1> k;
  real x_tot[N_pred + N];
```

```
  N_tot = N_pred + N;
  k = 1;
  for (n in 1:N) {
    x_tot[k] = x[n];
    k = k + 1;
  }
  for (n in 1:N_pred) {
    x_tot[k] = x_pred[n];
    k = k + 1;
  }
}
parameters {
  real<lower=0> length_scale;
  real<lower=0> alpha;
  vector[N_tot] eta;
}
transformed parameters {
  vector[N_tot] f;
  {
    matrix[N_tot, N_tot] L;
    matrix[N_tot, N_tot] K;
    K = cov_exp_quad(x_tot, alpha, length_scale);
    for (n in 1:N_tot)
      K[n, n] = K[n, n] + 1e-12;
    L = cholesky_decompose(K);
    f = L * eta;
  }
}
model {
  length_scale ~ gamma(2, 20);
  alpha ~ normal(0, 1);
  eta ~ normal(0, 1);
  y ~ poisson_log(f[1:N]);
}
```

We can repurpose the fake data from above, by exponentiating the f and generating Poisson random variables conditioned on $\exp(f)$:

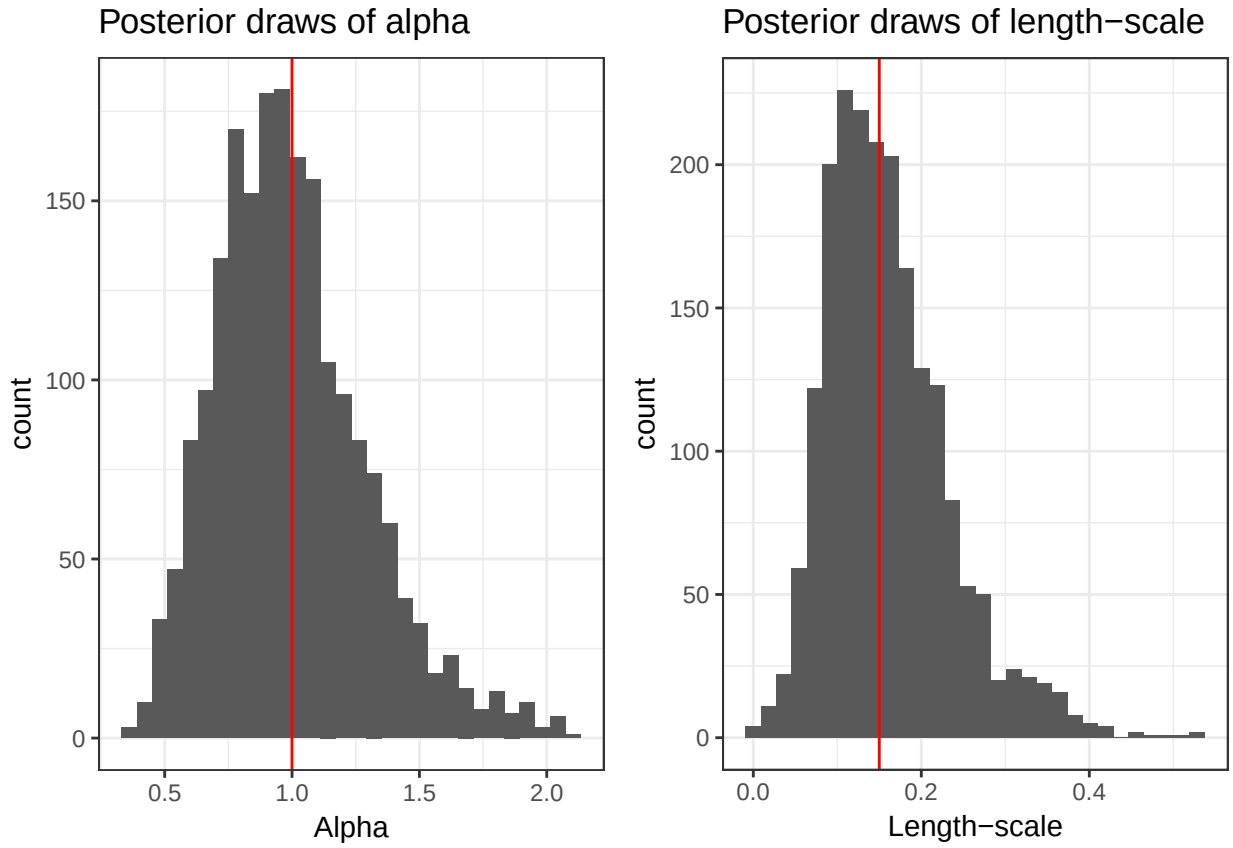
```
pois_oos_set <- sample((1:dat_list$N)[-set],size = 150, replace = F)
pois_N_oos <- length(pois_oos_set)
N_set <- length(set)
pois_full_set <- c(set,pois_oos_set)
pois_N_full_set <- pois_N_oos + N_set
stan_data_pois <- list(N = length(set), N_pred = pois_N_oos,
                       f_all = exp(samps$f[1,]),
                       x = samps$x[1,set], x_all = samps$x[1,],
                       y_all = rpois(n = dat_list$N,
                                     lambda = exp(samps$f[1,])),
                       x_pred = samps$x[1,pois_oos_set])
stan_data_pois$y <- stan_data_pois$y_all[set]
```

```
mod_run_lat_pois <- sampling(gp_mod_lat_pois, data = stan_data_pois, cores = 2, chains = 4, iter = 1000)
```

```
## Warning: There were 2 divergent transitions after warmup. Increasing adapt_delta above 0.8 may help.
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

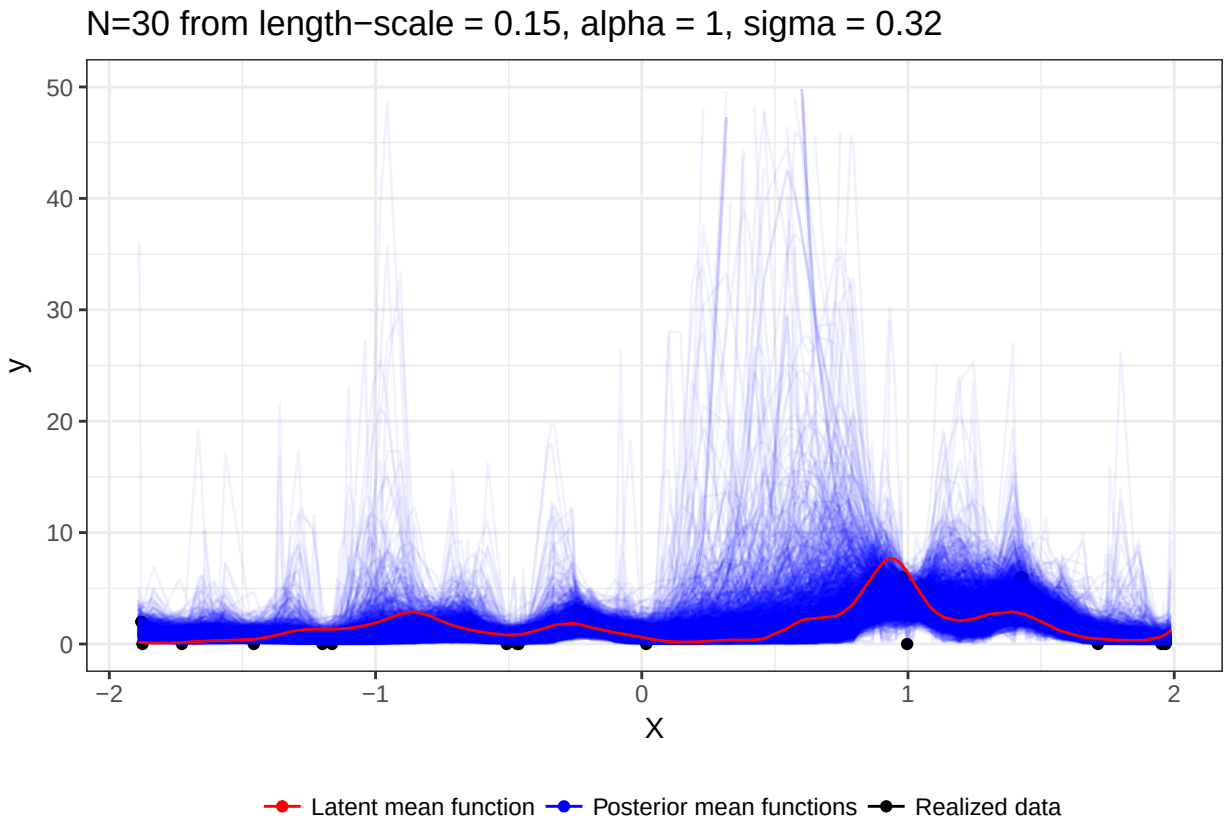
```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
samps_lat_pois <- rstan::extract(mod_run_lat_pois)
df1 <- data.frame(x = samps_lat_pois$alpha)
p1 <- ggplot(data = df1,
             aes(x = x)) + geom_histogram() + theme_bw() +
  labs(title = 'Posterior draws of alpha') +
       xlab('Alpha') + geom_vline(xintercept = dat_list$alpha,colour = 'red')
df2 <- data.frame(x = samps_lat_pois$length_scale)
p2 <- ggplot(data = df2,
             aes(x = x)) + geom_histogram() + theme_bw() +
  labs(title = 'Posterior draws of length-scale') +
       xlab('Length-scale') + geom_vline(xintercept = dat_list$length_scale,colour = 'red')
multiplot(p1, p2, cols = 2)
```




```
plt_df = with(stan_data_pois,data.frame(x = c(x_all[set],x_all[pois_oos_set]),
                                        y = c(y_all[set],y_all[pois_oos_set]),
                                        f = c(f_all[set],f_all[pois_oos_set])))
plt_df_rt = data.frame(x = plt_df$x, f = exp(t(samps_lat_pois$f)))
plt_df_rt_melt = melt(plt_df_rt,id.vars = 'x')
p <- ggplot(data = plt_df[1:length(set),], aes(x=x, y=y)) +
  geom_point(aes(colour = 'Realized data')) +
  geom_line(data = plt_df_rt_melt, aes(x = x, y = value, group = variable, colour = 'Posterior mean fun
  geom_line(data = plt_df, aes(x = x, y = f, colour = 'Latent mean function')) +
  scale_color_manual(name = '', values = c('Realized data'='black','Latent mean function'='red', 'Poste
  xlab('X') +
  ylab('y') +
  ggtitle(paste0('N=',length(set),' from length-scale = 0.15, alpha = 1, sigma = 0.32')) +
```

```
  ylim(c(0,50))
p
```



GP example

Andrew hosted an interesting model on his blog recently; the NYTimes had asked some researchers if they would forecast the 2020, 2024 and 2028 state-wide presidential votes. Let's dive into the problem, and see how we can use GPs in Stan to generate forecasts. Much of the code I'll present below was featured on Andrew's blog, and the model we'll walk through is a slight variation on what he fitted.

Loading the data:

```
past_votes <- readRDS('data_pres_forecast/pres_vote_historical.RDS') %>%
  filter(state != 'DC')
```

We're going to exclude DC because it's so different from the other states. We're going to use exchangeable priors to model time-invariant state effects and time-varying state effects, and DC is sufficiently different from the other states that we should build separate priors for the nation's capital. I won't do that here because I want to showcase integrating GPs into a more complex model, but we certainly could do this in the interest of generating better and more-detailed forecasts.

```
past_votes %>% head()
```

year	incumbent	state	total	dem	rep
1976	R	AK	123574	44058	71555
1976	R	AL	1182850	659170	504070
1976	R	AR	769396	499614	268753

year	incumbent	state	total	dem	rep
1976	R	AZ	742719	295602	418642
1976	R	CA	7867117	3742284	3882244
1976	R	CO	1081135	460353	584367

The observations are counts of the two-party vote in each state for each presidential election.

```
table(past_votes$state)
```

AK	AL	AR	AZ	CA	CO	CT	DE	FL	GA	HI	IA	ID	IL	IN	KS	KY	LA	MA	MD	ME
11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11

We have only 11 observations per state, which isn't very many. But there is extra structure that we can use to partially pool observations together.

The outcome we care about is the Republican share of the two-party vote by year by state.

```
state_groups <- list(c("ME","NH","VT","MA","RI","CT"),
                     c("NY","NJ","PA","MD","DE"),
                     c("OH","MI","IL","WI","MN"),
                     c("WA","OR","CA","HI"),
                     c("AZ","CO","NM","NV"),
                     c("IA","NE","KS","ND","SD"),
                     c("KY","TN","MO","WV","IN"),
                     c("VA","OK","FL","TX","NC"),
                     c("AL","MS","LA","GA","SC","AR"),
                     c("MT","ID","WY","UT","AK"))
region_names <- c("New England", "Mid-Atlantic", "Midwest", "West Coast",
                  "Southwest","Plains", "Border South", "Outer South", "Deep South",
                  "Mountain West")
state_region_map <- mapply(FUN = function(states, region)
  data.frame(state = states,
             region = rep(region,length(states)),
             stringsAsFactors = F),state_groups,region_names,
  SIMPLIFY = F)

state_region_map <- bind_rows(state_region_map) %>%
  arrange(state) %>% mutate(
    region_ind = as.integer(as.factor(region))
  )
```

We have 10 regions, with about 5 states per region. In order to get the data into the right form for Stan, we need a list of integers that map each observation to a state, and a separate vector for regions.

Joining all the data together will allow us to plot everything, which will elucidate the structure of the data.

```
year_map <- data.frame(year = sort(unique(past_votes$year)),
                       year_ind = 1:11)
past_votes <- past_votes %>%
  arrange(state, year) %>%
  left_join(state_region_map, by = 'state') %>%
  left_join(year_map, by = 'year') %>%
  mutate(
    state_ind = as.integer(as.factor(state)),
```

```
    two_party_turnout = dem + rep,
    y = rep / two_party_turnout
  )
```

Here are the state and region indices matched to each observation:

```
head(past_votes[,c('year','state','state_ind','region','region_ind','y')])
```

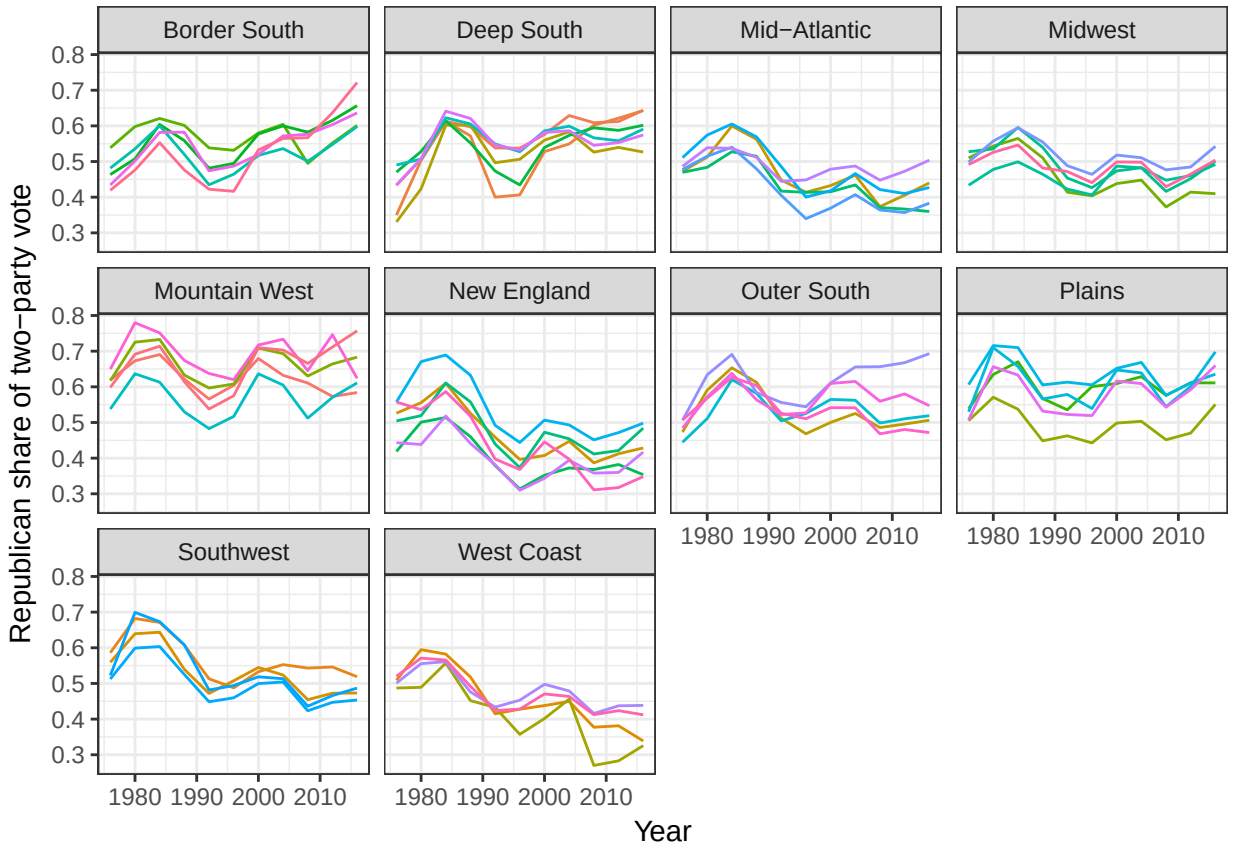
year	state	state_ind	region	region_ind	y
1976	AK	1	Mountain West	5	0.6189183
1980	AK	1	Mountain West	5	0.6729919
1984	AK	1	Mountain West	5	0.6905591
1988	AK	1	Mountain West	5	0.6216332
1992	AK	1	Mountain West	5	0.5657426
1996	AK	1	Mountain West	5	0.6042850

```
tail(past_votes[,c('year','state','state_ind','region','region_ind','y')])
```

	year	state	state_ind	region	region_ind	y
545	1996	WY	50	Mountain West	5	0.5748792
546	2000	WY	50	Mountain West	5	0.7098231
547	2004	WY	50	Mountain West	5	0.7031270
548	2008	WY	50	Mountain West	5	0.6656202
549	2012	WY	50	Mountain West	5	0.7116063
550	2016	WY	50	Mountain West	5	0.7570532

We're going to plot the time series of the Republican share of the two-party vote in each state for the past 11 presidential elections.

```
past_votes %>%
  ggplot(aes(x = year, y = y, colour = state)) +
  geom_line() + facet_wrap(~ region) +
  theme_bw() + theme(legend.position = 'None') +
  ylab('Republican share of two-party vote') + xlab('Year')
```



We can patterns that we might want to include in a model. Most notably, and perhaps not surprisingly, we see that there is a cross-sectional national correlation each year. There look to be time-invariant state-level mean Republican shares. There should likely be a time-invariant regional offset.

Within regions, there is also a clear time trend. Some states don't strictly adhere to the regional trend. There is a longer-term trend, and then short-term deviations away from the trend. This suggests a model structure that accounts for time-varying and time-invariant state and regional factors, as well as national trends. Because our outcome data will be a proportion we'll use a beta density for our likelihood. The beta distribution has support in $[0, 1]$ and is canonically parameterized with two shape parameters, γ and β.

$$p(y|\gamma, \beta) = \frac{1}{\mathrm{B}(\gamma, \beta)} y^{\gamma-1}(1-y)^{\beta-1}$$

We can reparameterize the distribution in terms of its mean $\mathbb{E}[y] = \mu$ and precision, $\mathrm{Var}[y] = \frac{\mu(1-\mu)}{(1+\nu)}$ (Ferrari and Cribari-Neto 2004).

$$p(y|\mu, \nu) = \frac{1}{\mathrm{B}(\mu\nu, (1-\mu)\nu)} y^{\mu\nu-1}(1-y)^{(1-\mu)\nu-1}$$

We'll use the alternative parameterization for our regression. Note that $\nu = \gamma + \beta$. We can think of ν as being the prior sample size, like when using a beta distribution as a conjugate prior for the probability parameter in a binomial likelihood. The interpretation of ν as sample size helps us to formulate an informative prior for ν. We use a Gamma distribution with a mean of 500, $\nu \sim \mathrm{Gamma}(5, \frac{1}{100})$ because we have about 500 observations.

$$
\begin{aligned}
y_{t,j} &\sim \text{Beta}(\text{inv\_logit}\,\mu_{t,j}, \nu) \\
\mu_{t,j} &= \theta_t^{\text{year}} + \theta_j^{\text{state}} + \theta_{k[j]}^{\text{region}} \\
&\quad + \gamma_{t,j} + \delta_{t,k[j]} \\
\boldsymbol{\gamma_j} &\sim \text{MultiNormal}(0, K_{\ell_1^\gamma,\alpha_1^\gamma} + K_{\ell_2^\gamma,\alpha_2^\gamma}) \\
\boldsymbol{\delta_k} &\sim \text{MultiNormal}(0, K_{\ell_1^\delta,\alpha_1^\delta} + K_{\ell_2^\delta,\alpha_2^\delta})
\end{aligned}
$$

In order to properly identify the model, we'll need strong priors over all of the parameters, because we don't have very much data to work with. A quick way to formulate priors that have good shrinkage properties is to put hierarchical shrinkage priors on our θ_j^{state} and θ_k^{region}:

$$
\begin{aligned}
\theta_j^{\text{state}} &\sim \text{Normal}(0, \sigma^{\text{state}}) \\
\theta_k^{\text{region}} &\sim \text{Normal}(0, \sigma^{\text{region}})
\end{aligned}
$$

Note that we can put more structure into the variance parameters for the state-level time-invariant means.

$$
\theta_j^{\text{state}} \sim \text{Normal}(0, \sigma_{k[j]}^{\text{state}})
$$

In order to build forecasts, we'll also need a map of state to region. We have the map from observation to regions from above, but we can use that to build a map that is correctly ordered from state to region:

```
stan_state_region_map <- unique(past_votes[,c('state_ind','region_ind')]) %>%
  arrange(state_ind)
```

```
stan_state_region_map %>% head()
```

state_ind	region_ind
1	5
2	2
3	2
4	9
5	10
6	9

Now we prep the data for RStan:

```
to_stan <- with(past_votes,
                list(
                  N = dim(past_votes)[1],
                  state_region_ind = stan_state_region_map$region_ind,
                  N_states = length(unique(past_votes$state)),
                  N_regions = length(unique(past_votes$region)),
                  N_years_obs = length(unique(past_votes$year)),
                  state_ind = state_ind,
                  region_ind = region_ind,
```

```
                    y = y,
                    year_ind = year_ind,
                    N_years = 14))
```

Stan program for election forecasting

```
data {
  int<lower=1> N;
  int<lower=1> N_states;
  int<lower=1> N_regions;
  int<lower=1> N_years_obs;
  int<lower=1> N_years;
  int<lower=1> state_region_ind[N_states];
  int<lower=1,upper=50> state_ind[N];
  int<lower=1,upper=10> region_ind[N];
  int<lower=1> year_ind[N];
  vector<lower=0,upper=1>[N] y;
}
transformed data {
  real years[N_years];
  vector[16] counts;
  int n_comps;

  for (t in 1:N_years)
    years[t] = t;
  n_comps = rows(counts);
  for (i in 1:n_comps)
    counts[i] = 2;
}
parameters {
  matrix[N_years,N_regions] GP_region_std;
  matrix[N_years,N_states] GP_state_std;
  vector[N_years_obs] year_std;
  vector[N_states] state_std;
  vector[N_regions] region_std;
  real<lower=0> tot_var;
  simplex[n_comps] prop_var;
  real mu;
  real<lower=0> nu;

  real<lower=0> length_GP_region_long;
  real<lower=0> length_GP_state_long;
  real<lower=0> length_GP_region_short;
  real<lower=0> length_GP_state_short;
}
transformed parameters {
  matrix[N_years,N_regions] GP_region;
  matrix[N_years,N_states] GP_state;

  vector[N_years_obs] year_re;
  vector[N_states] state_re;
  vector[N_regions] region_re;
  vector[n_comps] vars;
```

```
  real sigma_year;
  real sigma_region;
  vector[10] sigma_state;

  real sigma_GP_region_long;
  real sigma_GP_state_long;
  real sigma_GP_region_short;
  real sigma_GP_state_short;

  vars = n_comps * prop_var * tot_var;
  sigma_year = sqrt(vars[1]);
  sigma_region = sqrt(vars[2]);
  for (i in 1:10)
    sigma_state[i] = sqrt(vars[i + 2]);

  sigma_GP_region_long = sqrt(vars[13]);
  sigma_GP_state_long = sqrt(vars[14]);
  sigma_GP_region_short = sqrt(vars[15]);
  sigma_GP_state_short = sqrt(vars[n_comps]);

  region_re = sigma_region * region_std;
  year_re = sigma_year * year_std;
  state_re = sigma_state[state_region_ind] .* state_std;

  {
    matrix[N_years, N_years] cov_region;
    matrix[N_years, N_years] cov_state;
    matrix[N_years, N_years] L_cov_region;
    matrix[N_years, N_years] L_cov_state;

    cov_region = cov_exp_quad(years, sigma_GP_region_long,
                              length_GP_region_long)
              + cov_exp_quad(years, sigma_GP_region_short,
                              length_GP_region_short);
    cov_state = cov_exp_quad(years, sigma_GP_state_long,
                              length_GP_state_long)
              + cov_exp_quad(years, sigma_GP_state_short,
                              length_GP_state_short);
    for (year in 1:N_years) {
      cov_region[year, year] = cov_region[year, year] + 1e-12;
      cov_state[year, year] = cov_state[year, year] + 1e-12;
    }

    L_cov_region = cholesky_decompose(cov_region);
    L_cov_state = cholesky_decompose(cov_state);
    GP_region = L_cov_region * GP_region_std;
    GP_state = L_cov_state * GP_state_std;
  }
}
model {
  vector[N] obs_mu;

  for (n in 1:N) {
```

```
    obs_mu[n] = nu * inv_logit(mu + year_re[year_ind[n]]
              + state_re[state_ind[n]]
              + region_re[region_ind[n]]
              + GP_region[year_ind[n],region_ind[n]]
              + GP_state[year_ind[n],state_ind[n]]);
  }
  y ~ beta(obs_mu, (nu - obs_mu));

  to_vector(GP_region_std) ~ normal(0, 1);
  to_vector(GP_state_std) ~ normal(0, 1);
  year_std ~ normal(0, 1);
  state_std ~ normal(0, 1);
  region_std ~ normal(0, 1);
  mu ~ normal(0, .5);
  tot_var ~ gamma(3, 3);
  nu ~ gamma(5, 0.01);
  prop_var ~ dirichlet(counts);
  length_GP_region_long ~ weibull(30,8);
  length_GP_state_long ~ weibull(30,8);
  length_GP_region_short ~ weibull(30,3);
  length_GP_state_short ~ weibull(30,3);
}
generated quantities {
  matrix[N_years,N_states] y_new;
  matrix[N_years,N_states] y_new_pred;

  {
    real level;
    level = normal_rng(0.0, sigma_year);
    for (state in 1:N_states) {
      for (t in 1:N_years) {
        if (t < 12) {
          y_new[t,state] = state_re[state]
                         + region_re[state_region_ind[state]]
                         + GP_state[t,state]
                         + GP_region[t,state_region_ind[state]]
                         + mu + year_re[t];
        } else {
          y_new[t,state] = state_re[state]
                         + region_re[state_region_ind[state]]
                         + GP_state[t,state]
                         + GP_region[t,state_region_ind[state]]
                         + level;
        }
        y_new_pred[t,state] =
          beta_rng(inv_logit(y_new[t,state]) * nu,
                   nu * (1 - inv_logit(y_new[t,state])));
      }
    }
  }
}
```

Of note in the model above is how we parameterize the hierarchical latent GP model. In order to put multivariate normal priors on each $\boldsymbol{\gamma_j}$:

$$\boldsymbol{\gamma_j} \sim \text{MultiNormal}(0, K_{\ell_1^\gamma,\alpha_1^\gamma} + K_{\ell_2^\gamma,\alpha_2^\gamma})$$

we generate an auxiliary set of random variables, $\eta \in \mathbb{R}^{T,J}$, $\eta[t,j] \sim \text{Normal}(0,1)\,\forall\, t \in \{1,\ldots,T\}, j \in \{1,\ldots,J\}$, and decompose the covariance matrix $K_{\ell_1^\gamma,\alpha_1^\gamma} + K_{\ell_2^\gamma,\alpha_2^\gamma}$ into a lower triangular matrix L:

$$L \times L^T = K_{\ell_1^\gamma,\alpha_1^\gamma} + K_{\ell_2^\gamma,\alpha_2^\gamma}$$

Left multiplying L by *eta* yields $\gamma \in \mathbb{R}^{T,J}$, where each column of γ, γ_j, is an independent draw from a multivariate normal density $\text{MultiNormal}(0, K_{\ell_1^\gamma,\alpha_1^\gamma} + K_{\ell_2^\gamma,\alpha_2^\gamma})$.

Now we fit the model.

```
compiled_model <- stan_model("hierarchical_gp.stan")
```

```
fit <- sampling(compiled_model, data = to_stan, iter = 2000,
                chains = 4, cores = 4, seed = 1245860998)
```

Inspecting the results for convergence of parameters:

```
sum_fit <- summary(fit)
tail(sort(sum_fit$summary[,'Rhat']))
```

```
##     GP_region_std[3,2]               vars[15]  sigma_GP_region_short
##               1.004519               1.004573               1.004774
##                   lp__                     nu length_GP_region_short
##               1.005193               1.005646               1.011872
```

```
head(round(100*sort(sum_fit$summary[,'n_eff'])/max(sum_fit$summary[,'n_eff'])))
```

```
## length_GP_region_short  sigma_GP_region_short              vars[15]
##                     11                    14                    15
##           prop_var[15]                   lp__            year_re[10]
##                     20                    21                    24
```

It appears that this model converged nicely and the effective number of samples is in the right ballpark for a well-behaved model. What do our forecasts look like for the next 3 presidential elections? We have the data we need to answer this question in our generated quantities block.

```
samps <- rstan::extract(fit)
state_preds_new <- samps$y_new_pred
dim(state_preds_new)
```

```
## [1] 4000   14   50
```

Here's an example of what our forecasts look like for Wyoming. We'd expect this state to be a pretty firm Republican firewall state.

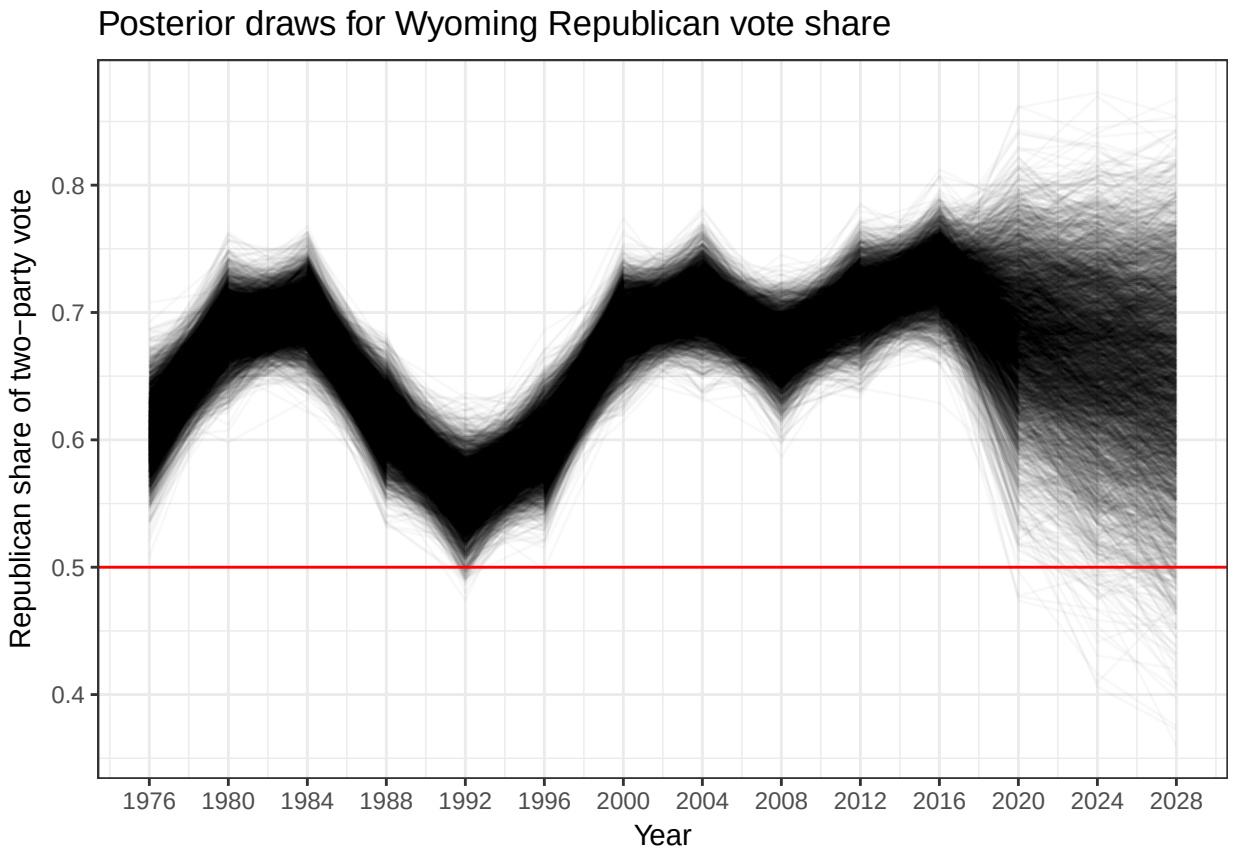
```
st_map <- unique(past_votes[,c('state','state_ind')])
st_map %>% filter(state == 'WY') -> wy_ind
```

```
year_vec <- c(sort(unique(past_votes$year)),2020,2024,2028)
wy_preds <- state_preds_new[,,wy_ind$state_ind]
wy_df <- data.frame(year = year_vec,
                    y = t(wy_preds))
```

```
wy_melt <- melt(wy_df,id.vars = 'year')
head(wy_melt)
```

year	variable	value
1976	y.1	0.6713510
1980	y.1	0.6844241
1984	y.1	0.6731283
1988	y.1	0.6300858
1992	y.1	0.5756072
1996	y.1	0.5615600

```
ggplot(wy_melt, aes(x = year, y = value, group = variable)) +
  geom_line(alpha = 0.03) + geom_hline(yintercept = 0.5, colour = 'red') +
  scale_x_continuous(breaks=year_vec) +
  theme_bw() +
  labs(title = 'Posterior draws for Wyoming Republican vote share') +
       xlab('Year') +
       ylab('Republican share of two-party vote')
```



This comports with our intuition that Wyoming should be firmly Republican in 2020. The model's posterior mass on a Democratic win in Wyoming in 2020 is `round(100*apply(wy_preds < 0.5,2,mean),1)[year_vec == 2016]`:

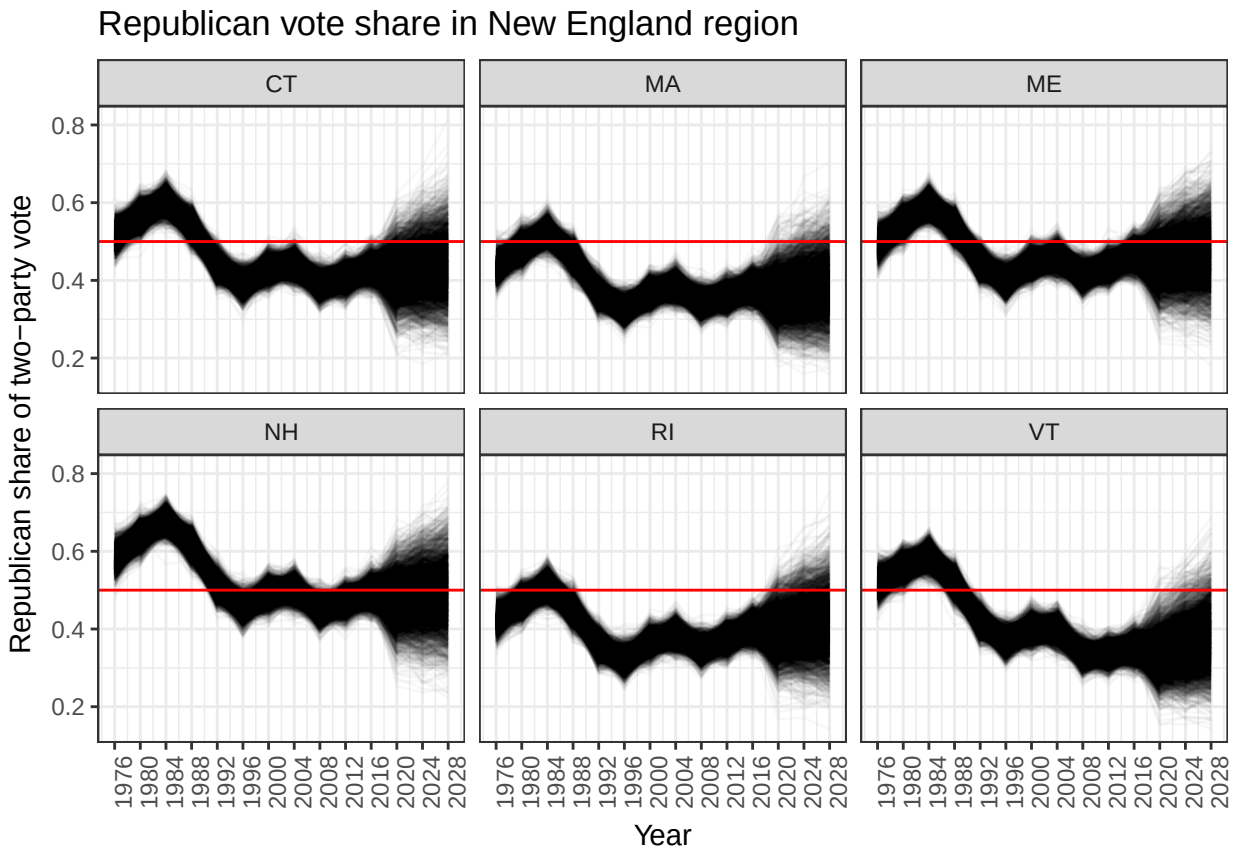
```
dem_win_wy <- data.frame(year = year_vec,
                         pct_prob_dem_win = round(100*apply(wy_preds < 0.5,2,mean),2))
dem_win_wy %>% filter(year > 2016)
```

year	pct_prob_dem_win
2020	0.10
2024	0.90
2028	3.08

```
l_df <- list()
for (i in 1:50) {
  st_nm <- st_map %>% filter(state_ind == i)
  st_df <- state_preds_new[,,i]
  preds <- data.frame(year = year_vec,
                      y = t(st_df))
  preds_melt <- melt(preds, id.vars = 'year')
  preds_melt$state = st_nm$state
  l_df[[i]] <- preds_melt
}
l_df <- bind_rows(l_df)
l_df <- l_df %>% left_join(state_region_map, by = 'state')
plts <- list()
for (nm_i in seq_along(region_names)) {
  region_nm <- region_names[nm_i]
  plts[[region_nm]] <- l_df %>% filter(region == region_nm) %>%
    ggplot(aes(x = year, y = value, group = variable)) +
    geom_line(alpha = 0.03) + geom_hline(yintercept = 0.5, colour = 'red') +
    scale_x_continuous(breaks=year_vec) +
    theme_bw() +
    labs(title = paste0('Republican vote share in ',region_nm,' region')) +
         xlab('Year') +
         ylab('Republican share of two-party vote') +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
   facet_wrap(~ state)
}
```

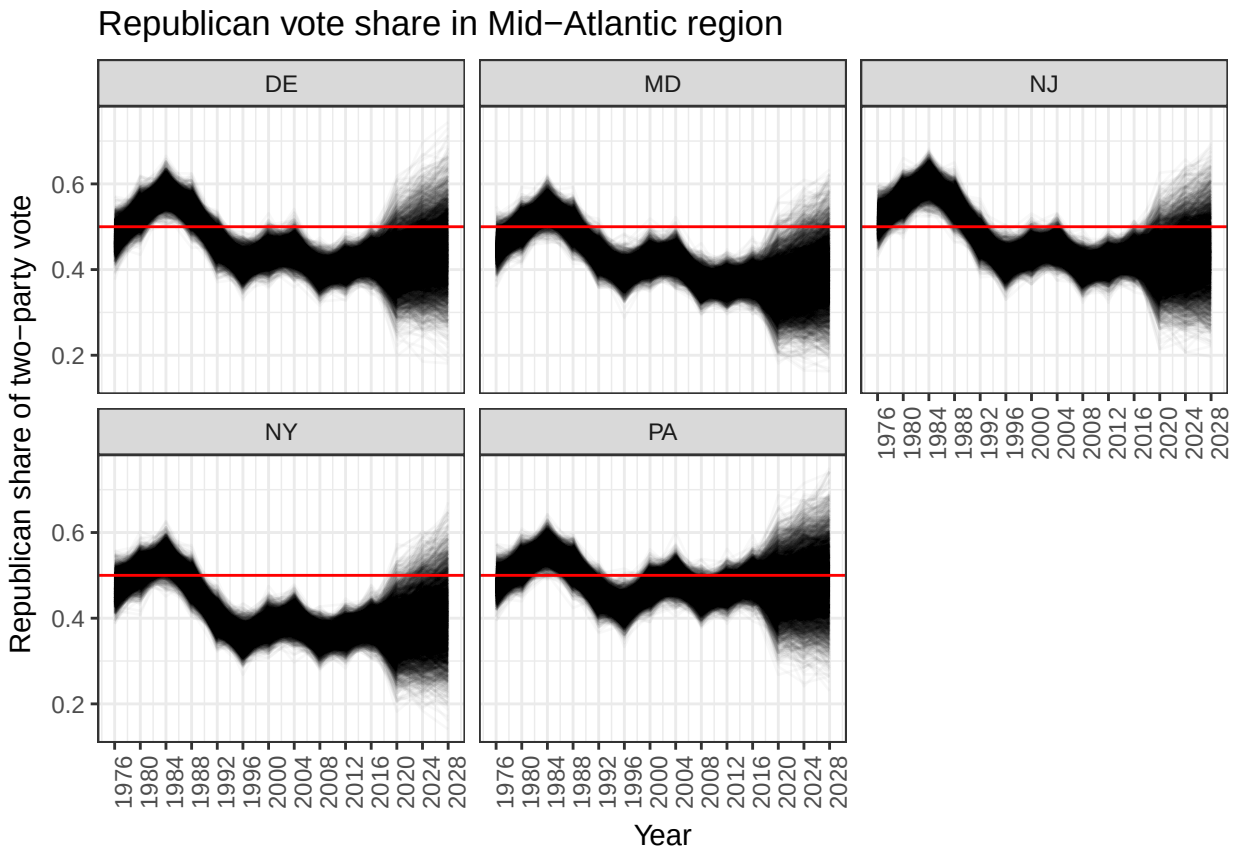
Here're the forecasts for New England's states:

```
plts[['New England']]
```



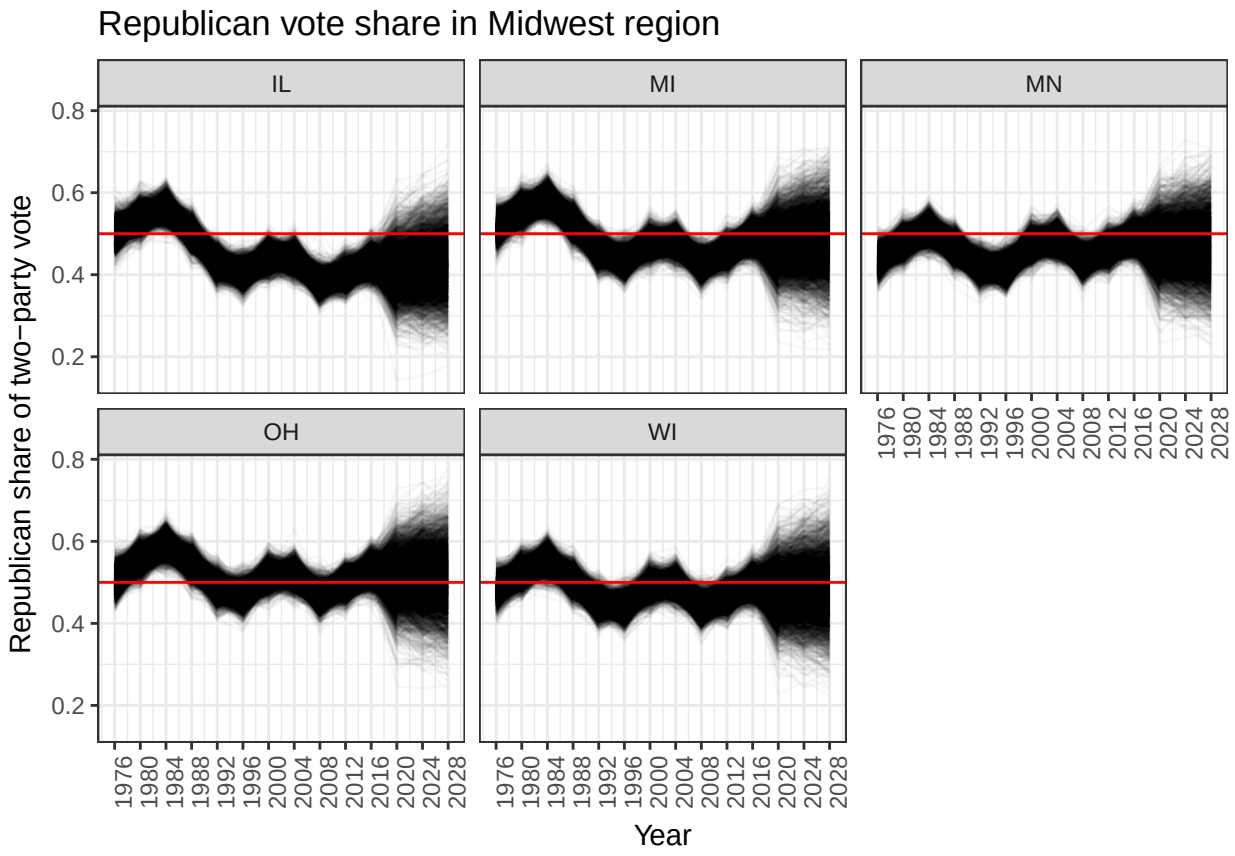
Here're the forecasts for the Mid-Atlantic states:

```
plts[['Mid-Atlantic']]
```



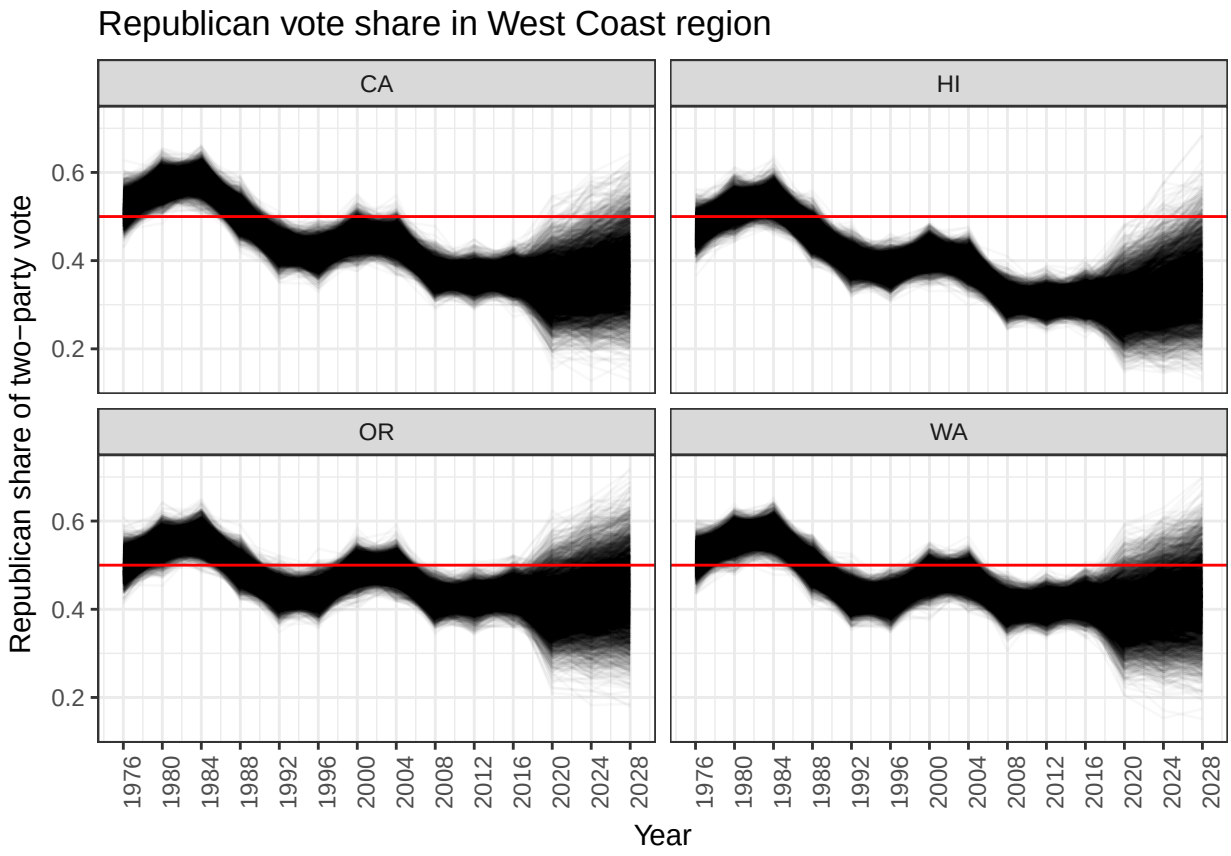
And Midwest states:

```
plts[['Midwest']]
```



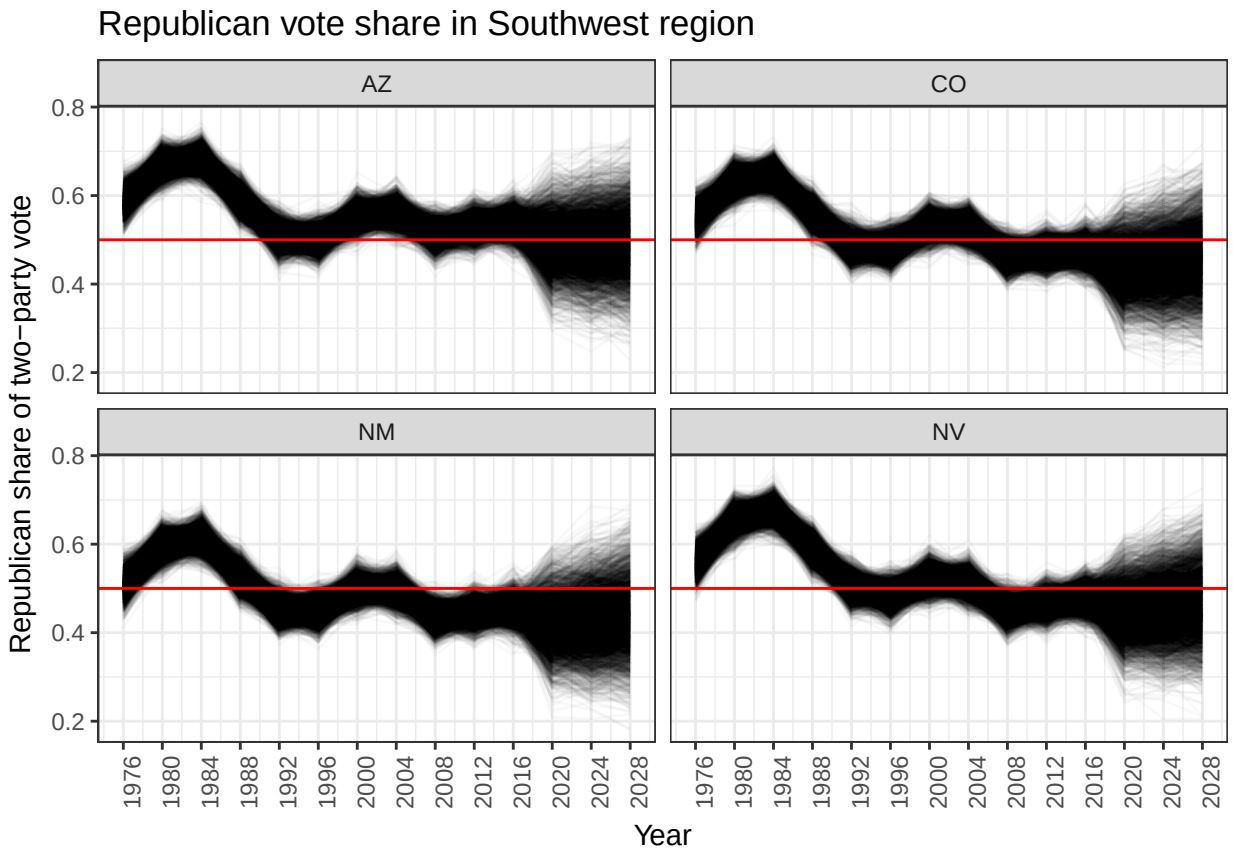
Making our way further west, West Coast states:

```
plts[['West Coast']]
```



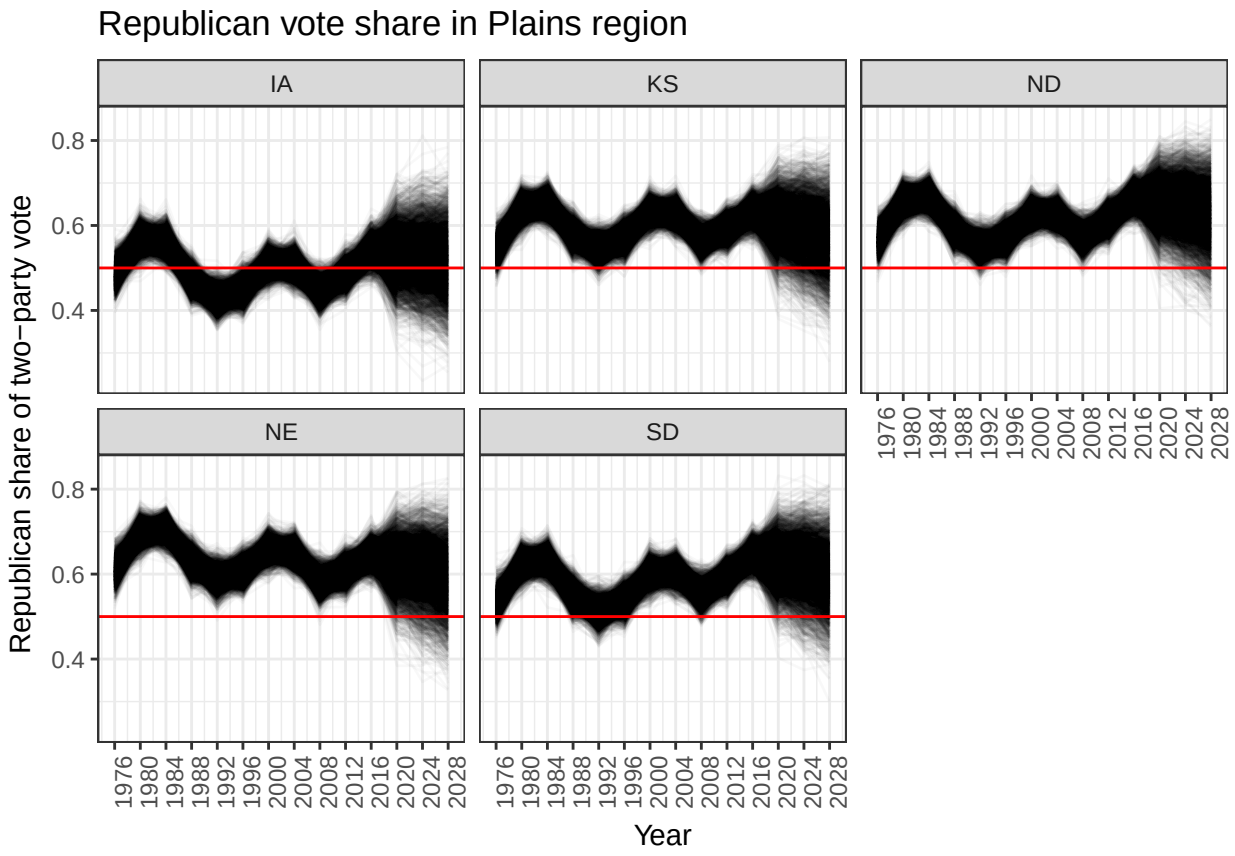
A southerly move:

```
plts[['Southwest']]
```



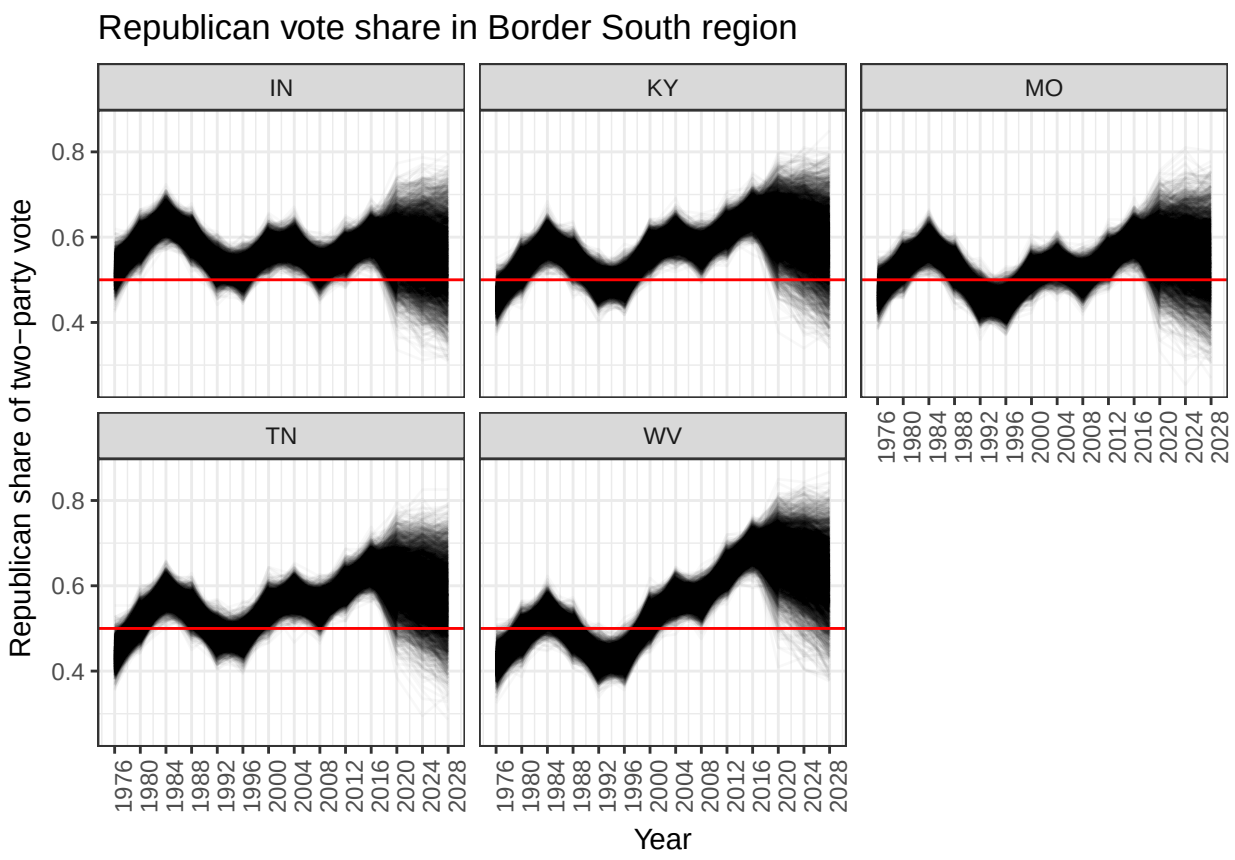
And onward to the Plains region:

```
plts[['Plains']]
```



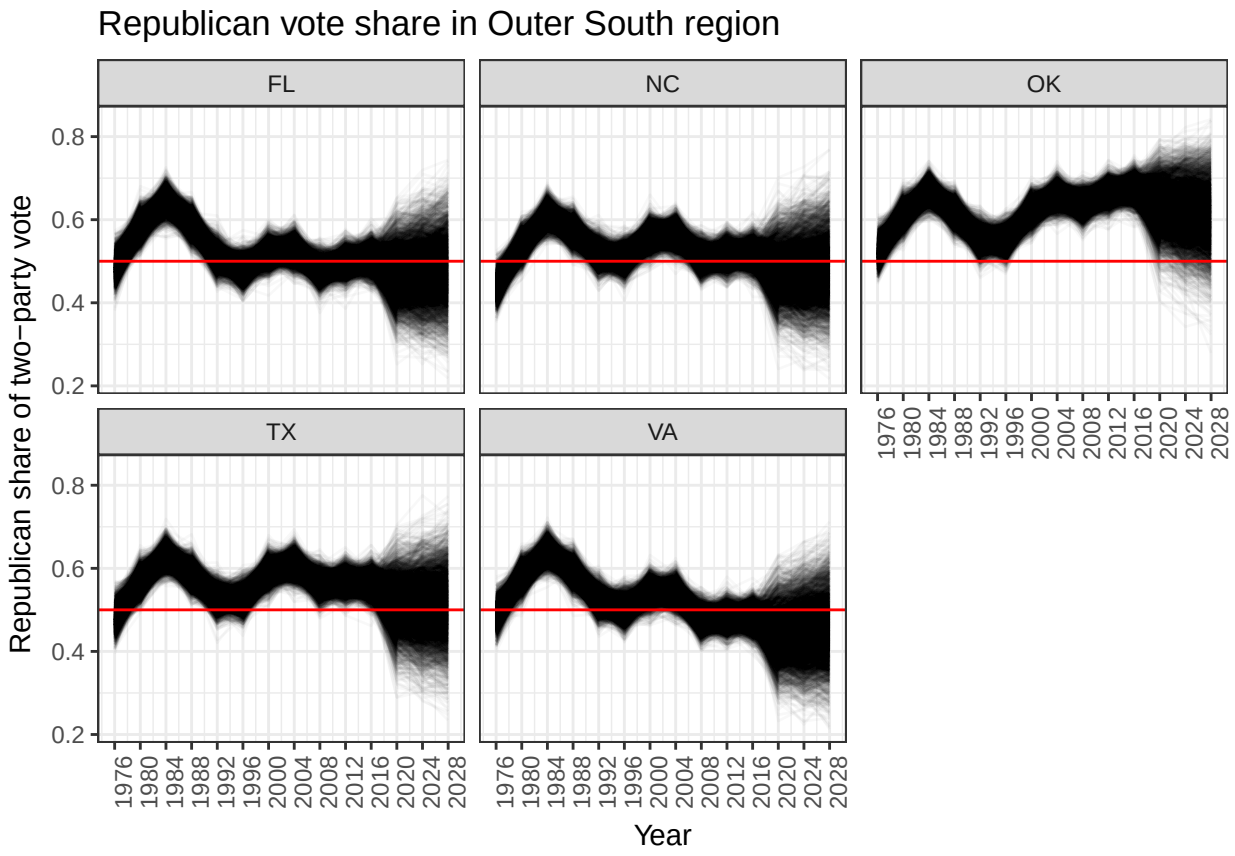
Southern states:

```
plts[['Border South']]
```



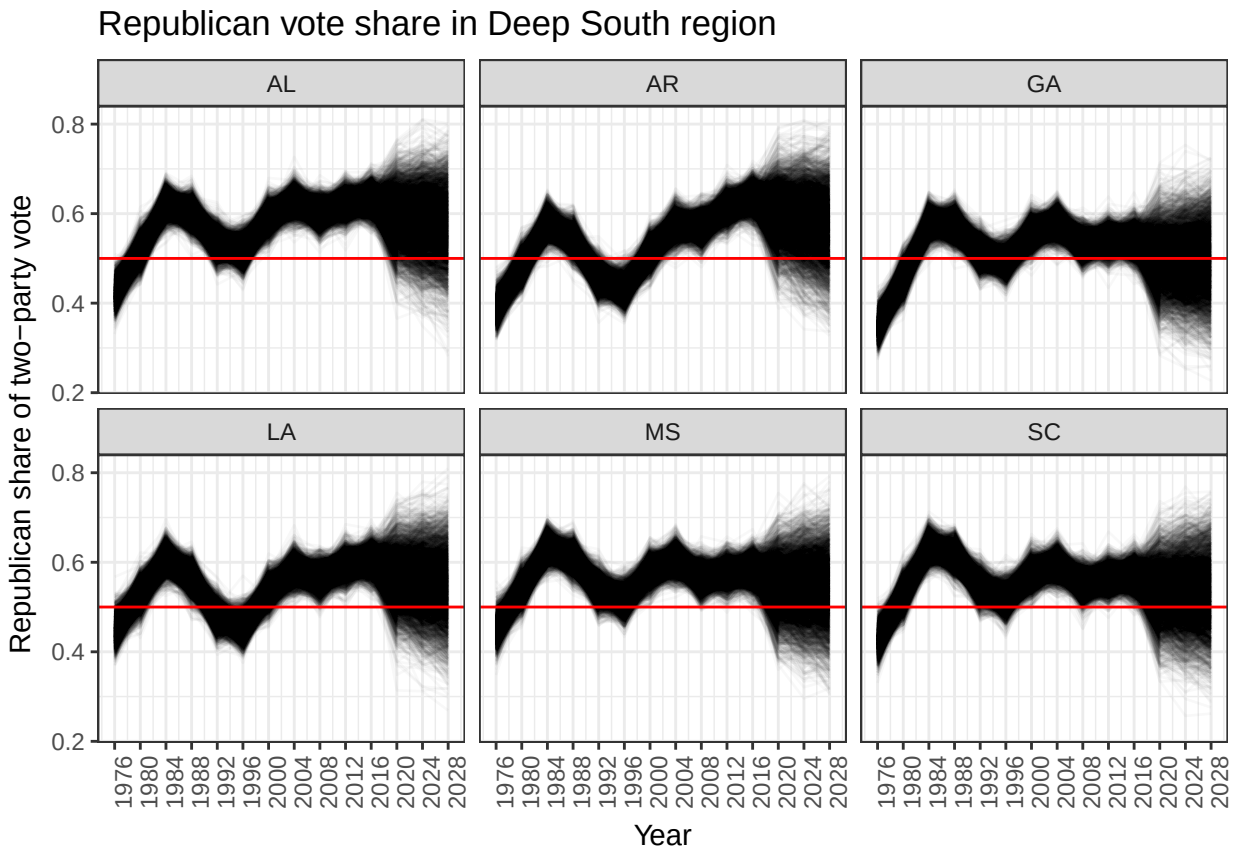
More Southern states:

```
plts[['Outer South']]
```



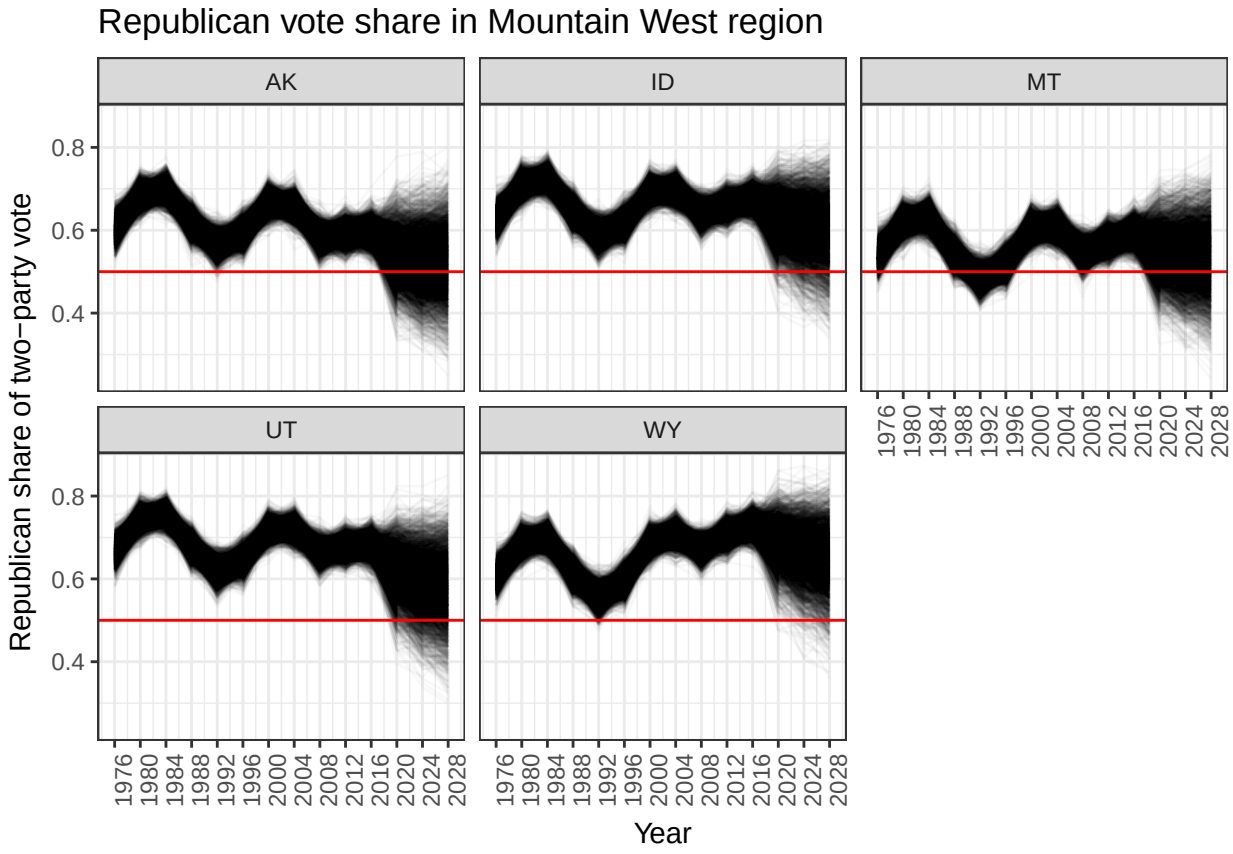
The Deep South:

```
plts[['Deep South']]
```



And, finally, the Mountain West:

```
plts[['Mountain West']]
```



Conclusion

GPs are a flexible class of priors over random variables in probabilistic models. We can integrate Gaussian processes into Stan models easily using `cov_exp_quad` and `cholesky_decompose`. The non-centered latent variable formulation of the GP is particularly useful when defining the GP prior over parameters that are weakly informed by the data.

REFERENCES

Betancourt, M. J., and M. Girolami. 2013. "Hamiltonian Monte Carlo for Hierarchical Models." *ArXiv E-Prints*, December.

Cramér, Harald, and M Ross Leadbetter. 2004. *Stationary and Related Stochastic Processes: Sample Function Properties and Their Applications*. Courier Corporation.

Ferrari, Silvia, and Francisco Cribari-Neto. 2004. "Beta Regression for Modelling Rates and Proportions." *Journal of Applied Statistics* 31 (7). Taylor & Francis: 799–815.

Rasmussen, Carl Edward, and Christopher KI Williams. 2005. *Gaussian Processes for Machine Learning*. The MIT Press.

Stein, Michael L. 2012. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Science & Business Media.