mvgam case study 2: multivariate models

Nicholas Clark (n.clark@uq.edu.au)

In this example we will examine multivariate forecasting models using `mvgam`, which fits dynamic GAMs (DGAMs; Clark & Wells, 2022) using MCMC sampling (note that either `JAGS` or `Stan` is required; installation links are found here and here). First a simulation experiment to determine whether `mvgam`'s inclusion of complexity penalisation works by reducing the number of un-needed dynamic factors. In any factor model, choosing the appropriate number of factors `M` can be difficult. The approach used by `mvgam` when sampling with `JAGS` is to estimate a penalty for each factor that squeezes the factor's variance toward zero, effectively forcing the factor to evolve as a flat white noise process. By allowing each factor's penalty to be estimated in an exponentially increasing manner (following Welty et al 2009), we hope that we can guard against specifying too large a `M`. Note that when sampling with `Stan`, we capitalise on the superior sampling and exploration of Hamiltonian Monte Carlo to choose the number of factors. Begin by simulating `6` series that evolve with a shared seasonal pattern and that depend on `2` latent random walk factors. Each series is `100` time steps long, with a seasonal frequency of `12`. We give the trend moderate importance by setting `trend_rel = 0.6` and we allow each series' observation process to be drawn from slightly different Poisson distributions

```
set.seed(1111)
library(mvgam)
```

```
## Loading required package: mgcv
```

```
## Warning: package 'mgcv' was built under R version 4.2.2
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-41. For overview type 'help("mgcv-package")'.
```

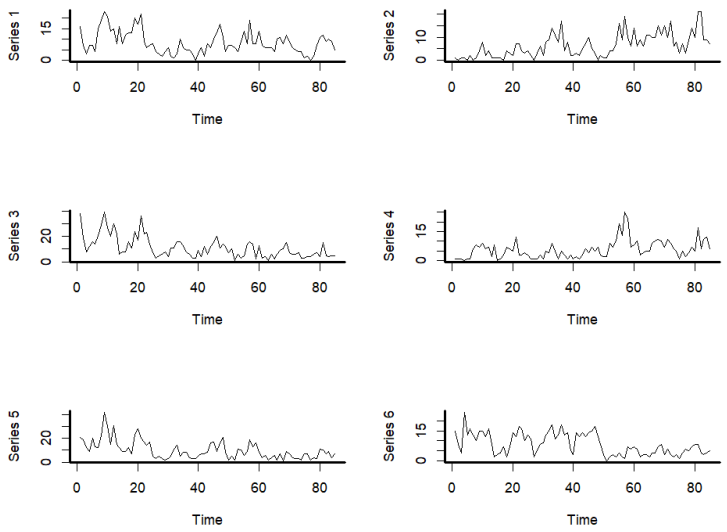
```
## Loading required package: parallel
```

```
## Welcome to mvgam. Please cite as: Clark, NJ, and Wells, K. 2022. Dynamic Generalized Additive Models
```

```
dat <- sim_mvgam(T = 100, n_series = 6, n_lv = 2,
                 family = 'poisson',
                 mu_obs = runif(6, 4, 8),
                 trend_rel = 0.6, train_prop = 0.85)
```

Have a look at the series

```
par(mfrow = c(3,2))
for(i in 1:6){
  plot(dat$data_train$y[which(as.numeric(dat$data_train$series) == i)], type = 'l',
       ylab = paste('Series', i), xlab = 'Time', bty = 'L')
  box(bty = 'L', lwd = 2)
}
```




```
par(mfrow = c(1,1))
```

Clearly there are some correlations in the trends for these series. But how does a dynamic factor process allow us to potentially capture these dependencies? The below example demonstrates how. Essentially, a dynamic factor is an *unobserved* (latent) random process that induces correlations between time series via a set of factor loadings (β) while exercising dimension reduction. The loadings represent constant associations between the observed time series and the dynamic factor, but each series can still deviate from the factor through its error process and its associations with other factors (if we estimate `>1` latent factor in our model).

A challenge with any factor model is the need to determine the number of factors `M`. Setting `M` too small prevents temporal dependencies from being adequately modelled, leading to poor convergence and difficulty estimating smooth parameters. By contrast, setting `M` too large leads to unnecessary computation. `mvgam` approaches this problem by either formulating a prior distribution that enforces exponentially increasing penalties on the factor variances to allow any un-needed factors to evolve as flat lines (if using `JAGS`) or by regularising the factor loadings (if using `Stan`). Now let's fit a well-specified model for our simulated series in which we estimate random intercepts, a shared seasonal cyclic smooth and `2` latent dynamic factors. Note that when using `Stan`, which is heavily recommended for complex models such as this, the dynamic factor variances are fixed to allow the loading coefficients to control the magnitudes of the induced trends. This is necessary to ensure identifiability

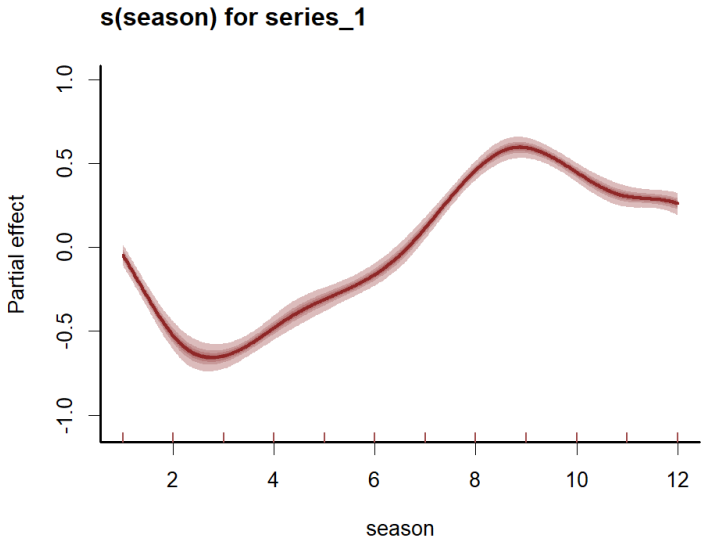
```
mod1 <- mvgam(data = dat$data_train,
              newdata = dat$data_test,
              formula = y ~ s(series, bs = 're') +
                s(season, bs = c('cc'), k = 7) - 1,
              knots = list(season = c(0.5, 12.5)),
              use_lv = TRUE,
              n_lv = 2,
              family = 'poisson',
              trend_model = 'RW',
              use_stan = TRUE,
              chains = 4,
              burnin = 300,
              samples = 400)
```

```
## Running MCMC with 4 parallel chains...
##
```

```
## Chain 1 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 2 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 3 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 4 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 2 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 3 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 4 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 2 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 1 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 3 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 4 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 1 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 1 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 2 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 2 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 3 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 3 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 4 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 4 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 1 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 3 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 2 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 4 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 3 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 1 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 2 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 3 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 1 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 4 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 3 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 3 finished in 43.1 seconds.
## Chain 1 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 1 finished in 43.9 seconds.
## Chain 2 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 4 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 2 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 2 finished in 51.1 seconds.
## Chain 4 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 4 finished in 55.1 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 48.3 seconds.
## Total execution time: 55.2 seconds.
```

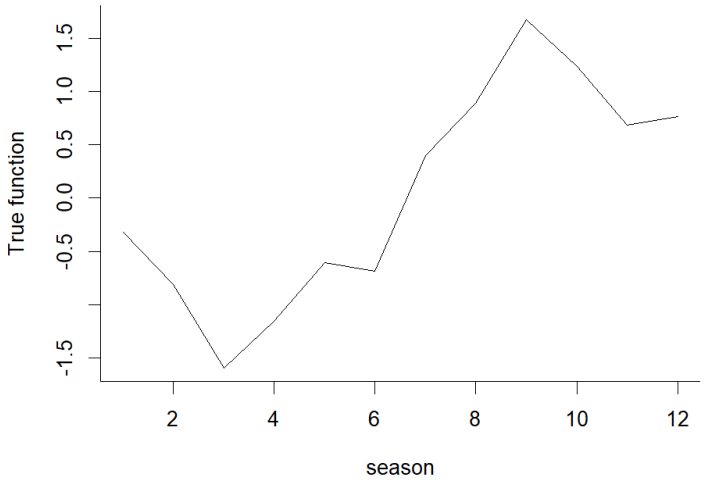
Look at a few plots. The estimated smooth function

```
plot_mvgam_smooth(object = mod1, series = 1, smooth = 'season')
```



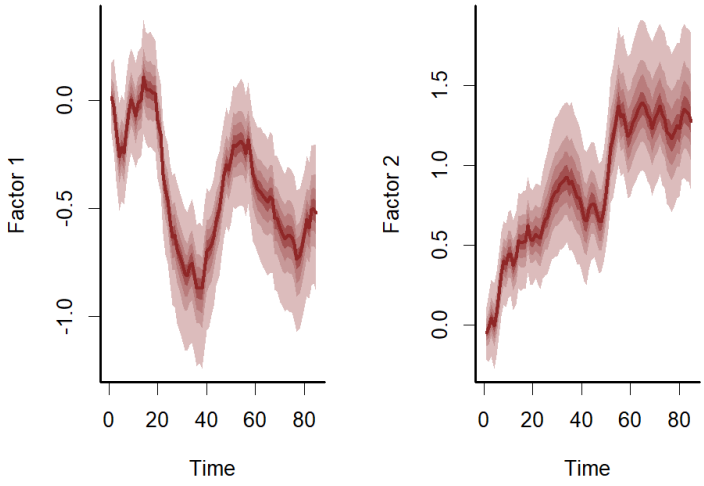
And the true seasonal function in the simulation

```
plot(dat$global_seasonality[1:12], type = 'l', bty = 'L', ylab = 'True function', xlab = 'season')
```



Check whether each factor was retained using the `plot_mvgam_factors` function. Here, each factor is tested against a null hypothesis of white noise by calculating the sum of the factor's 1st derivatives. A factor that has a larger contribution to the series' latent trends will have a larger sum, both because that factor's absolute magnitudes will be larger (due to the weaker penalty on the factor's precision) and because the factor will move around more. By normalising these estimated first derivative sums, it should be apparent whether some factors have been dropped from the model. Here we see that each factor is contributing to the series' latent trends, and the plots show that neither has been forced to evolve as white noise

```
plot_mvgam_factors(mod1)
```




```
##         Contribution
## Factor1    0.4917258
## Factor2    0.5082742
```

Now we fit the same model but assume that we no nothing about how many factors to use, so we specify the maximum allowed (the total number of series; 6). Note that this model is computationally more expensive so it will take longer to fit

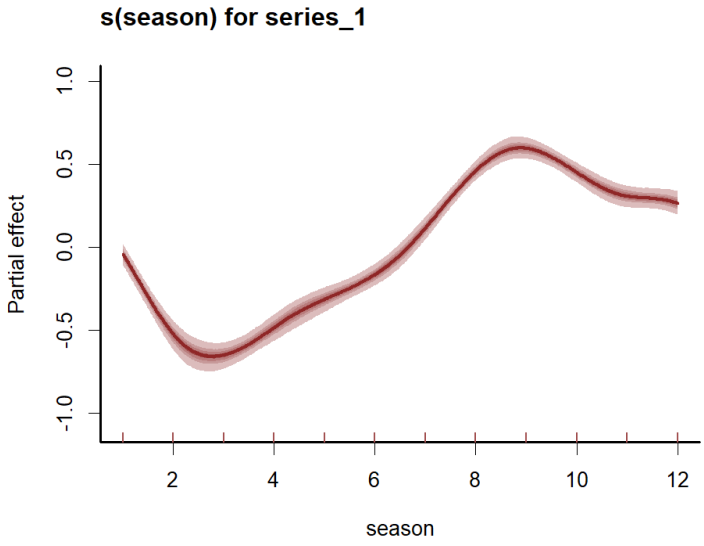
```
mod2 <- mvgam(data = dat$data_train,
              newdata = dat$data_test,
              formula = y ~ s(series, bs = 're') +
                s(season, bs = c('cc'), k = 7) - 1,
              knots = list(season = c(0.5, 12.5)),
              use_lv = TRUE,
              n_lv = 6,
              family = 'poisson',
              use_stan = TRUE,
              trend_model = 'RW',
              chains = 4,
              burnin = 300,
              samples = 400)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 2 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 3 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 4 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 4 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 1 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 3 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 2 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 4 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 1 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 2 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 3 Iteration: 200 / 700 [ 28%]  (Warmup)
```

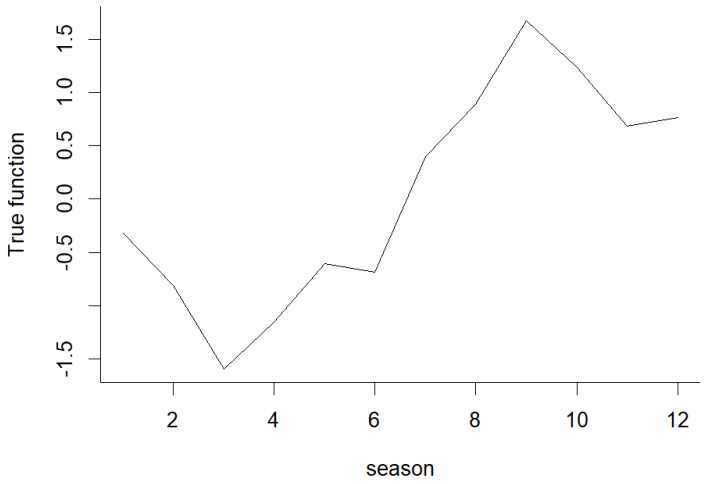
```
## Chain 4 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 4 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 1 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 1 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 3 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 3 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 2 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 2 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 4 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 1 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 3 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 2 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 1 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 4 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 3 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 2 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 1 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 4 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 2 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 3 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 1 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 1 finished in 158.1 seconds.
## Chain 4 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 4 finished in 164.2 seconds.
## Chain 2 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 2 finished in 170.8 seconds.
## Chain 3 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 3 finished in 177.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 167.5 seconds.
## Total execution time: 177.1 seconds.
```

Use the same plots as model 1 to see if this model has also fit the data well

```
plot_mvgam_smooth(object = mod2, series = 1, smooth = 'season')
```

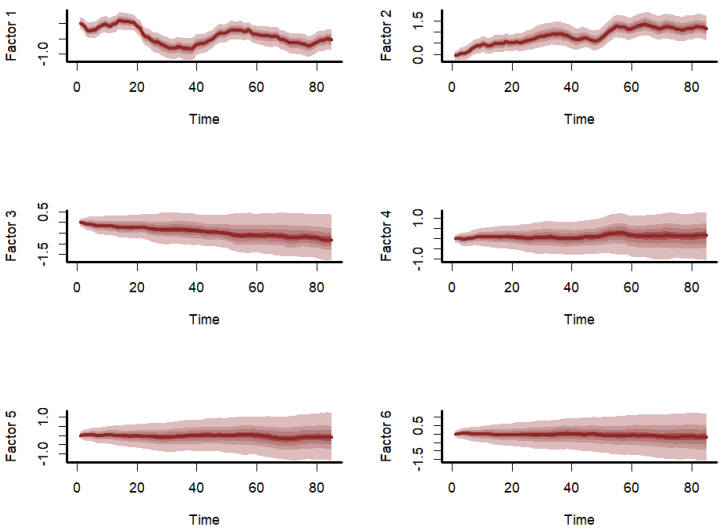



```
plot(dat$global_seasonality[1:12], type = 'l', bty = 'L', ylab = 'True function', xlab = 'season')
```



Examining the factor contributions gives us some insight into whether we set `n_lv` larger than we perhaps needed to (with some of the factors clearly evolving as unconstrained, zero-centred random walks). These contributions can be interpreted similarly to ordination axes when deciding how many latent variables to specify

```
plot_mvgam_factors(mod2)
```




```
##         Contribution
## Factor1   0.31052538
## Factor2   0.32856545
## Factor3   0.10701094
## Factor4   0.09875010
## Factor5   0.08003450
## Factor6   0.07511363
```

The very weak contributions by some of the factors are a result of the loading regularisation, which will become more important as the dimensionality of the data grows. Now onto an empirical example. Here we will access monthly search volume data from `Google Trends`, focusing on relative importances of search terms related to tick paralysis in Queensland, Australia

```
library(tidyr)
if(!require(gtrendsR)){
  install.packages('gtrendsR')
}

terms = c("tick bite",
          "tick paralysis",
          "dog tick",
          "paralysis tick dog")
trends <- gtrendsR::gtrends(terms, geo = "AU-QLD",
                            time = "all", onlyInterest = T)
```

`Google Trends` modified their algorithm for extracting search volume data in 2012, so we filter the series to only include observations after this point in time

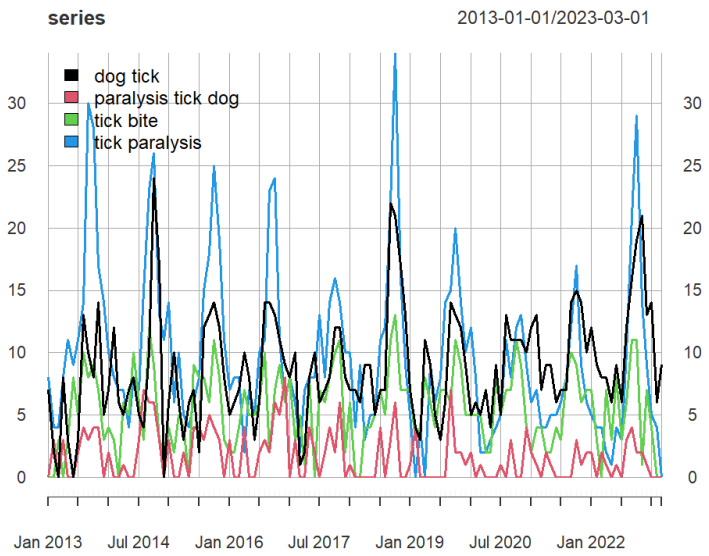
```
trends$interest_over_time %>%
  tidyr::spread(keyword, hits) %>%
  dplyr::select(-geo, -time, -gprop, -category) %>%
  dplyr::mutate(date = lubridate::ymd(date)) %>%
  dplyr::mutate(year = lubridate::year(date)) %>%
  dplyr::filter(year > 2012) %>%
  dplyr::select(-year) -> gtest
```

Convert to an `xts` object and then to the required `mvgam` format, holding out the final 10% of observations as the test data

```
series <- xts::xts(x = gtest[,-1], order.by = gtest$date)
trends_data <- series_to_mvgam(series, freq = 12, train_prop = 0.9)
```

Plot the series to see how similar their seasonal shapes are over time

```
plot(series, legend.loc = 'topleft')
```



Now we will fit an `mvgam` model with shared seasonality and random intercepts per series. Our first attempt will ignore any temporal component in the residuals so that we can identidy which GAM predictor combination gives us the best fit, prior to investigating how to deal with any remaining autocorrelation. We assume a Poisson observation model for the response. Also note that any smooths using the random effects basis (`s(series, bs = "re")` below) are automatically re-parameterised to use the non-centred parameterisation that is necessary to help avoid common posterior degeneracies in hierarchical models. This parameterisation tends to work better for most ecological problems where the data for each group / context are not highly informative, but it is still probably worth investigating whether a centred or even a mix of centred / non-centred will give better computational performance. We suppress the global intercept as it is not needed and will lead to identifiability issues when estimating the series-specific random intercepts

```
trends_mod1 <- mvgam(data = trends_data$data_train,
                     newdata = trends_data$data_test,
                     formula = y ~ s(series, bs = 're') +
                       s(season, k = 7, m = 2, bs = 'cc') - 1,
                     knots = list(season = c(0.5, 12.5)),
                     trend_model = 'None',
                     family = 'poisson',
                     use_stan = TRUE,
                     chains = 4,
                     burnin = 300,
                     samples = 400)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 2 Iteration:   1 / 700 [  0%]  (Warmup)
```

```
## Chain 3 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 4 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 1 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 2 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 3 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 4 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 2 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 3 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 4 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 1 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 1 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 2 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 3 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 4 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 2 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 3 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 4 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 1 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 2 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 3 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 4 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 3 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 2 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 4 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 1 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 3 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 2 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 3 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 3 finished in 1.5 seconds.
## Chain 1 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 4 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 2 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 2 finished in 1.8 seconds.
## Chain 1 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 4 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 1 finished in 2.0 seconds.
## Chain 4 finished in 1.9 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 1.8 seconds.
## Total execution time: 2.1 seconds.
```

Given that these series could potentially be following a hierarchical seasonality, we will also trial a slghtly more complex model with an extra smoothing term per series that allows its seasonal curve to deviate from the global seasonal smooth. Ignore the warning about repeated smooths, as this is not an issue for estimation.

```
trends_mod2 <- mvgam(data = trends_data$data_train,
                     newdata = trends_data$data_test,
                     formula = y ~ s(season, k = 7, m = 2, bs = 'cc') +
                       s(season, series, k = 4, bs = 'fs', m = 1),
                     knots = list(season = c(0.5, 12.5)),
                     trend_model = 'None',
                     family = 'poisson',
                     use_stan = TRUE,
```

```
                      chains = 4,
                      burnin = 300,
                      samples = 400)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 2 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 3 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 4 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 2 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 2 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 3 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 3 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 4 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 4 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 1 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 1 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 1 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 2 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 3 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 4 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 2 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 3 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 4 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 4 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 1 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 2 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 3 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 2 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 4 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 1 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 3 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 1 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 2 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 4 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 3 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 4 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 4 finished in 1.7 seconds.
## Chain 1 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 2 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 1 finished in 1.7 seconds.
## Chain 2 finished in 1.7 seconds.
## Chain 3 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 3 finished in 2.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 1.8 seconds.
## Total execution time: 2.1 seconds.
```

How can we compare these models to ensure we choose one that performs well and provides useful inferences? Beyond posterior retrodictive and predictive checks, we can take advantage of the fact that `mvgam` fits an `mgcv` model to provide all the necessary penalty matrices, as well as to identify good initial values for smoothing

parameters. Because we did not modify this model by adding a trend component (the only modification is that we estimated series-specific overdispersion parameters), we can still employ the usual `mgcv` model comparison routines

```
anova(trends_mod1$mgcv_model,
      trends_mod2$mgcv_model, test = 'LRT')
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ s(series, bs = "re") + s(season, k = 7, m = 2, bs = "cc") -
##     1
## Model 2: y ~ s(season, k = 7, m = 2, bs = "cc") + s(season, series, k = 4,
##     bs = "fs", m = 1)
##   Resid. Df Resid. Dev     Df Deviance  Pr(>Chi)
## 1    431.01     714.32
## 2    421.27     663.31 9.7478   51.014 1.385e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
AIC(trends_mod1$mgcv_model,
      trends_mod2$mgcv_model)
```

```
##                               df      AIC
## trends_mod1$mgcv_model  8.601681 2118.780
## trends_mod2$mgcv_model 16.828262 2084.219
```

```
summary(trends_mod1$mgcv_model)
```

```
##
## Family: poisson
## Link function: log
##
## Formula:
## y ~ s(series, bs = "re") + s(season, k = 7, m = 2, bs = "cc") -
##     1
##
## Approximate significance of smooth terms:
##             edf Ref.df Chi.sq p-value
## s(series) 3.997      4   9696  <2e-16 ***
## s(season) 4.503      5 199957  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.686   Deviance explained = 64.3%
## -REML = 1078.3  Scale est. = 1         n = 440
```

```
summary(trends_mod2$mgcv_model)
```

```
##
## Family: poisson
## Link function: log
##
## Formula:
## y ~ s(season, k = 7, m = 2, bs = "cc") + s(season, series, k = 4,
##     bs = "fs", m = 1)
##
## Parametric coefficients:
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.5877     0.4084   3.888 0.000101 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                       edf Ref.df Chi.sq p-value
## s(season)           4.538      5  136.2  <2e-16 ***
## s(season,series) 10.538     15  617.8  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.711   Deviance explained = 66.8%
## -REML = 1062.5  Scale est. = 1         n = 440
```

Model 2 seems to fit better so far, suggesting that hierarchical seasonality gives better performance for these series. But a problem with both of the above models is that their forecast uncertainties will not increase into the future, which is not how time series forecasts should behave. Here we fit Model 2 again but specifying a time series model for the latent trends. We assume the dynamic trends can be represented using latent factors that each follow a squared exponential Gaussian Process (GP), and we will set `n_lv = 2`. Note that `mvgam` uses a Hilbert space approximate Gaussian Process for computational efficiency, with the number of basis functions m set to `min(25, n_timepoints)`

```
trends_mod3 <- mvgam(data = trends_data$data_train,
                     newdata = trends_data$data_test,
                     formula = y ~ s(season, k = 7, m = 2, bs = 'cc') +
                       s(season, series, k = 4, bs = 'fs', m = 1),
                     knots = list(season = c(0.5, 12.5)),
                     trend_model = 'GP',
                     use_lv = TRUE,
                     n_lv = 2,
                     family = 'poisson',
                     use_stan = TRUE,
                     chains = 4,
                     burnin = 300,
                     samples = 400)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 2 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 3 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 4 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 3 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 2 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 1 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 4 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 3 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 1 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 2 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 4 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 3 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 3 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 1 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 2 Iteration: 300 / 700 [ 42%]  (Warmup)
```

```
## Chain 1 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 2 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 3 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 4 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 4 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 2 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 1 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 3 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 4 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 2 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 3 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 1 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 4 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 2 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 3 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 3 finished in 21.1 seconds.
## Chain 4 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 1 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 2 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 2 finished in 22.9 seconds.
## Chain 4 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 4 finished in 23.8 seconds.
## Chain 1 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 1 finished in 24.3 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 23.0 seconds.
## Total execution time: 24.5 seconds.
```

Have a look at the returned `Stan` model file to see how the GP factors are incorporated

```
cat(c("```stan", trends_mod3$model_file, "```"), sep = "\n")
```

```
// Stan model code generated by package mvgam
functions {
real lambda_gp(real L, int m) {
real lam;
lam = ((m*pi())/(2*L))^2;
return lam;
}

vector phi_SE(real L, int m, vector x) {
vector[rows(x)] fi;
fi = 1/sqrt(L) * sin(m*pi()/(2*L) * (x+L));
return fi;
}

real spd_SE(real alpha, real rho, real w) {
real S;
S = (alpha^2) * sqrt(2*pi()) * rho * exp(-0.5*(rho^2)*(w^2));
return S;
}
}

data {
```

```
int idx1[12]; // discontiguous index values
int idx2[4]; // discontiguous index values
int<lower=0> total_obs; // total number of observations
int<lower=0> n; // number of timepoints per series
int<lower=0> n_lv; // number of dynamic factors
int<lower=0> n_sp; // number of smoothing parameters
int<lower=0> n_series; // number of series
int<lower=0> num_basis; // total number of basis coefficients
real p_taus[1]; // prior precisions for parametric coefficients
real p_coefs[1]; // prior locations for parametric coefficients
vector[num_basis] zero; // prior locations for basis coefficients
matrix[total_obs, num_basis] X; // mgcv GAM design matrix
int<lower=0> ytimes[n, n_series]; // time-ordered matrix (which col in X belongs to each [time, series]
matrix[5,5] S1; // mgcv smooth penalty matrix S1
int<lower=0> n_nonmissing; // number of nonmissing observations
int<lower=0> flat_ys[n_nonmissing]; // flattened nonmissing observations
matrix[n_nonmissing, num_basis] flat_xs; // X values for nonmissing observations
int<lower=0> obs_ind[n_nonmissing]; // indices of nonmissing observations
}
transformed data {
vector<lower=1>[n] times;
vector[n] times_cent;
real mean_times;
real<lower=0> boundary;
int<lower=1> num_gp_basis;
num_gp_basis = min(20, n);
matrix[n, num_gp_basis] gp_phi;

for (t in 1:n){
times[t] = t;
}

mean_times = mean(times);
times_cent = times - mean_times;
boundary = (5.0/4) * (max(times_cent) - min(times_cent));
for (m in 1:num_gp_basis){
gp_phi[,m] = phi_SE(boundary, m, times_cent);
}

// Number of non-zero lower triangular factor loadings
// Ensures identifiability of the model - no rotation of factors
int<lower=1> M;
M = n_lv * (n_series - n_lv) + n_lv * (n_lv - 1) / 2 + n_lv;
}

parameters {
// raw basis coefficients
vector[num_basis] b_raw;

// dynamic factor lower triangle loading coefficients
vector[M] L;

// gp parameters
```

```
vector<lower=0>[n_lv] rho_gp;

// gp coefficient weights
matrix[num_gp_basis, n_lv] b_gp;
// smoothing parameters

vector<lower=0>[n_sp] lambda;
}

transformed parameters {
// gp spectral densities
matrix[n, n_lv] LV_raw;
matrix[num_gp_basis, n_lv] diag_SPD;
matrix[num_gp_basis, n_lv] SPD_beta;

// trends and dynamic factor loading matrix
matrix[n, n_series] trend;
matrix[n_series, n_lv] lv_coefs_raw;

// basis coefficients
vector[num_basis] b;

b[1:num_basis] = b_raw[1:num_basis];
// constraints allow identifiability of loadings
for (i in 1:(n_lv - 1)) {
for (j in (i + 1):(n_lv)){
lv_coefs_raw[i, j] = 0;
}
}
{
int index;
index = 0;
for (j in 1:n_lv) {
for (i in j:n_series) {
index = index + 1;
lv_coefs_raw[i, j] = L[index];
}
}
}


// gp LV estimates
for (m in 1:num_gp_basis){
for (s in 1:n_lv){
diag_SPD[m, s] = sqrt(spd_SE(0.25, rho_gp[s], sqrt(lambda_gp(boundary, m))));
}
}
SPD_beta = diag_SPD .* b_gp;
LV_raw = gp_phi * SPD_beta;
// derived latent trends
for (i in 1:n){;
for (s in 1:n_series){
trend[i, s] = dot_product(lv_coefs_raw[s,], LV_raw[i,1:n_lv]);
```

```
}
}

}

model {
// parametric effect priors (regularised for identifiability)
for (i in 1:1) {
b_raw[i] ~ normal(p_coefs[i], 1 / p_taus[i]);
}

// prior for s(season)...
b_raw[2:6] ~ multi_normal_prec(zero[2:6],S1[1:5,1:5] * lambda[1]);

// prior for s(season,series)...
for (i in idx1) { b[i] ~ normal(0, lambda[2]); }

for (i in idx2) { b[i] ~ normal(0, lambda[3]); }

// priors for smoothing parameters
lambda ~ normal(10, 25);

// priors for gp parameters
for (s in 1:n_lv){
b_gp[1:num_gp_basis, s] ~ std_normal();
}
rho_gp ~ inv_gamma(1.499007, 5.670433);

// priors for dynamic factor loading coefficients
L ~ student_t(5, 0, 1);

{
// likelihood functions
vector[n_nonmissing] flat_trends;
flat_trends = (to_vector(trend))[obs_ind];
flat_ys ~ poisson_log_glm(append_col(flat_xs, flat_trends),
0.0,append_row(b, 1.0));
}
}


generated quantities {
vector[total_obs] eta;
matrix[n, n_series] mus;
matrix[n, n_lv] LV;
matrix[n_series, n_lv] lv_coefs;
vector[n_sp] rho;
vector[n_lv] alpha_gp;
array[n, n_series] int ypred;
rho = log(lambda);
alpha_gp = rep_vector(0.25, n_lv);

// Sign correct factor loadings and factors
```

```
for(j in 1:n_lv){
if(lv_coefs_raw[j, j] < 0){
lv_coefs[,j] = -1 * lv_coefs_raw[,j];
LV[,j] = -1 * LV_raw[,j];
} else {
lv_coefs[,j] = lv_coefs_raw[,j];
LV[,j] = LV_raw[,j];
}
}

// posterior predictions
eta = X * b;
for(s in 1:n_series){
mus[1:n, s] = eta[ytimes[1:n, s]] + trend[1:n, s];
ypred[1:n, s] = poisson_log_rng(mus[1:n, s]);
}
}
```

There is a clear problem with this model, as we `Stan`'s dynamic HMC algorightm has encountered a large number of divergent transitions

```
summary(trends_mod3)
```

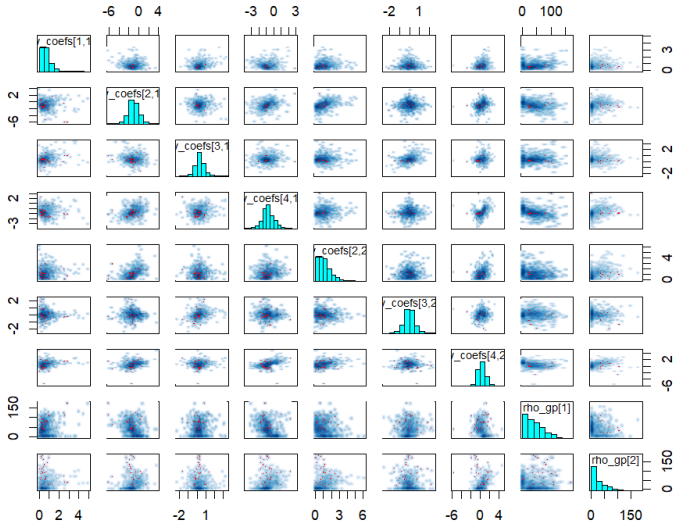
```
## GAM formula:

## y ~ s(season, k = 7, m = 2, bs = "cc") + s(season, series, k = 4,
##     bs = "fs", m = 1)

##

## Family:

## Poisson

##

## Link function:

## log

##

## Trend model:

## GP

##

## N latent factors:

## 2

##

## N series:

## 4

##

## N observations:

## 440
```

```
##
## Status:
## Fitted using Stan
##
## GAM coefficient (beta) estimates:
##                             2.5%          50%       97.5% Rhat n.eff
## (Intercept)          0.77575952  1.598690000  2.37826650 1.00   563
## s(season).1         -0.73506105 -0.543086000 -0.37131948 1.01   423
## s(season).2         -0.64750398 -0.398707500 -0.17886503 1.01   431
## s(season).3         -0.38389583 -0.178628500  0.04250393 1.01   458
## s(season).4          0.31288342  0.519544000  0.76098957 1.02   326
## s(season).5          0.56144083  0.713164500  0.87355277 1.01   435
## s(season,series).1  -0.04914862  0.110531000  0.29050792 1.00   546
## s(season,series).2  -0.23634515 -0.066703300  0.08381504 1.01   418
## s(season,series).3  -0.23897670 -0.155493500 -0.08266110 1.01   343
## s(season,series).4  -1.29744700 -0.456713000  0.37194512 1.00   612
## s(season,series).5  -0.35996410 -0.113476500  0.07861872 1.00  1288
## s(season,series).6  -0.23407987 -0.043924850  0.12464887 1.01   663
## s(season,series).7  -0.12200510 -0.009260345  0.10250092 1.00   811
## s(season,series).8   0.21437178  1.141305000  2.08539275 1.00   621
## s(season,series).9  -0.11205485  0.058614400  0.26675505 1.00   654
## s(season,series).10 -0.02699548  0.138903500  0.31304960 1.01   462
## s(season,series).11 -0.19605990 -0.106206500 -0.02516352 1.01   426
## s(season,series).12 -0.85225643 -0.014961550  0.78845732 1.00   578
## s(season,series).13 -0.26951467 -0.108571000  0.05015742 1.00   651
## s(season,series).14 -0.19757710 -0.029167400  0.12503177 1.01   439
## s(season,series).15 -0.12453627 -0.042363100  0.03173487 1.01   379
## s(season,series).16 -1.45833750 -0.610199000  0.35884055 1.00   567
##
## GAM smoothing parameter (rho) estimates:
##                         2.5%        50%     97.5% Rhat n.eff
## s(season)           2.047758  3.2459300  4.021358    1  1494
## s(season,series)   -2.676836 -2.0542000 -1.395715    1   576
## s(season,series)2 -0.824108  0.0600546  1.288150    1  1262
##
## Latent trend length scale (rho) estimates:
##                2.5%      50%    97.5% Rhat n.eff
## rho_gp[1] 1.417861 36.33735 110.3578 1.02   167
## rho_gp[2] 1.210425 15.78545 101.0992 1.00   181
##
## Stan MCMC diagnostics
## n_eff / iter looks reasonable for all parameters
## Rhat looks reasonable for all parameters
## 45 of 1600 iterations ended with a divergence (2.8125%)
## *Try running with larger adapt_delta to remove the divergences
## 0 of 1600 iterations saturated the maximum tree depth of 12 (0%)
## E-FMI indicated no pathological behavior
```

##

Our experience is that there are fundamental degeneracies in latent dynamic factor models, particularly occurring due to the inherent weak identifiability of GP length scale parameters `rho` (which is magnified when trying to estimate latent GPs in a factor model). A pairs plot of the loading coefficients against the estimated GP length scale parameters helps to reveal the problem:

```
pairs(trends_mod3$model_output, pars = c('lv_coefs','rho_gp'))
```



There are some nasty funnel shapes in this plot, primarily occurring when some of the loadings go to `0` and the length scales approach small values, suggesting that the latent length scales are completely unidentifiable in this space. The default prior for length scales in `mvgam` is an informative inverse Gamma, following principled prior recommendations by Michael Betancourt. But defaults should always be critiqued, and in our experience it is far more efficient to decide on an appropriate level of smoothness that we wish to induce in the latent GPs for a given problem and to then use a stronger prior. For these series, we may expect that any changes in online search behaviour could reflect environmental / biological processes that extend over a number of months, indicating that length scales of `4-8` would be more in line with domain expertise. We can use a suitable prior for the length scales to help pull values away from very small or very large length scales that go against our prior expectations

```
priors <- get_mvgam_priors(data = trends_data$data_train,
                           formula = y ~ s(season, k = 7, m = 2, bs = 'cc') +
                             s(season, series, k = 4, bs = 'fs', m = 1),
                           trend_model = 'GP',
                           use_lv = TRUE,
                           n_lv = 2,
                           family = 'poisson',
                           use_stan = TRUE)
priors$prior[2] <- 'rho_gp ~ normal(6, 1);'
trends_mod3 <- mvgam(data = trends_data$data_train,
                     newdata = trends_data$data_test,
                     formula = y ~ s(season, k = 7, m = 2, bs = 'cc') +
                       s(season, series, k = 4, bs = 'fs', m = 1),
                     knots = list(season = c(0.5, 12.5)),
                     trend_model = 'GP',
                     use_lv = TRUE,
```

```
                    n_lv = 2,
                    family = 'poisson',
                    use_stan = TRUE,
                    chains = 4,
                    burnin = 300,
                    samples = 400,
                    priors = priors)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 2 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 3 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 4 Iteration:   1 / 700 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 2 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 3 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 4 Iteration: 100 / 700 [ 14%]  (Warmup)
## Chain 1 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 4 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 2 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 3 Iteration: 200 / 700 [ 28%]  (Warmup)
## Chain 4 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 1 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 4 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 1 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 2 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 2 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 3 Iteration: 300 / 700 [ 42%]  (Warmup)
## Chain 3 Iteration: 301 / 700 [ 43%]  (Sampling)
## Chain 1 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 4 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 2 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 3 Iteration: 400 / 700 [ 57%]  (Sampling)
## Chain 1 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 4 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 2 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 3 Iteration: 500 / 700 [ 71%]  (Sampling)
## Chain 1 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 2 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 4 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 3 Iteration: 600 / 700 [ 85%]  (Sampling)
## Chain 1 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 1 finished in 22.3 seconds.
## Chain 2 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 2 finished in 23.1 seconds.
## Chain 4 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 4 finished in 23.2 seconds.
## Chain 3 Iteration: 700 / 700 [100%]  (Sampling)
## Chain 3 finished in 24.1 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 23.2 seconds.
## Total execution time: 24.3 seconds.
```

This model fits efficiently and should provide far superior forecasts than relying only on the estimated smooths. Inspect the model summary

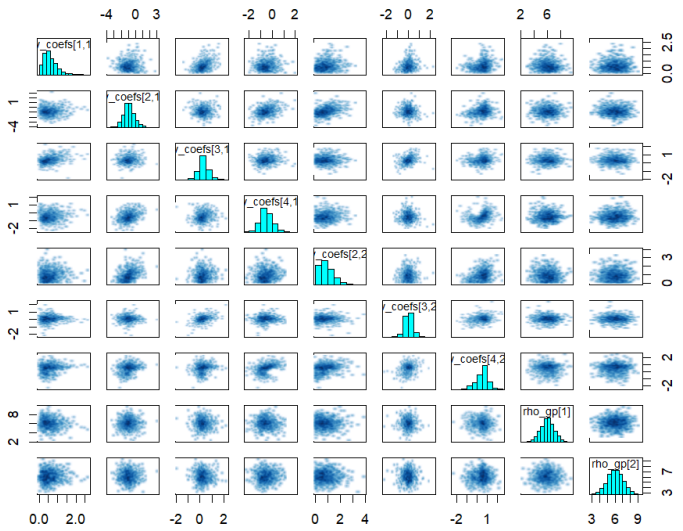
```
summary(trends_mod3)
```

```
## GAM formula:
## y ~ s(season, k = 7, m = 2, bs = "cc") + s(season, series, k = 4,
##     bs = "fs", m = 1)
##
## Family:
## Poisson
##
## Link function:
## log
##
## Trend model:
## GP
##
## N latent factors:
## 2
##
## N series:
## 4
##
## N observations:
## 440
##
## Status:
## Fitted using Stan
##
## GAM coefficient (beta) estimates:
##                             2.5%          50%       97.5% Rhat n.eff
## (Intercept)          0.68784517  1.596290000  2.40865600 1.01   492
## s(season).1         -0.73078657 -0.534022500 -0.36915103 1.00   529
## s(season).2         -0.68495030 -0.406439000 -0.18941575 1.00   433
## s(season).3         -0.39635055 -0.181314500  0.02847552 1.00   652
## s(season).4          0.30840067  0.518353000  0.74930052 1.00   382
## s(season).5          0.56374097  0.714126500  0.91222420 1.00   447
## s(season,series).1  -0.03746629  0.110858500  0.32090840 1.00   429
## s(season,series).2  -0.23177005 -0.066575150  0.10079745 1.00   625
## s(season,series).3  -0.25198223 -0.156308500 -0.07878920 1.01   337
## s(season,series).4  -1.38759225 -0.477626500  0.33960513 1.01   479
```

```
## s(season,series).5  -0.36487025 -0.110728500  0.09139520 1.00  1253
## s(season,series).6  -0.23698520 -0.039365450  0.15543465 1.00  1100
## s(season,series).7  -0.13509795 -0.006859185  0.10004640 1.00   648
## s(season,series).8   0.29382730  1.195695000  2.08519350 1.01   576
## s(season,series).9  -0.10101973  0.063731350  0.28060102 1.00   420
## s(season,series).10 -0.01427130  0.144339000  0.34456905 1.00   669
## s(season,series).11 -0.20501297 -0.106972000 -0.02280779 1.01   367
## s(season,series).12 -0.90983825 -0.026213150  0.78931995 1.01   483
## s(season,series).13 -0.27022590 -0.105056500  0.07125049 1.00   428
## s(season,series).14 -0.19159687 -0.026221800  0.13125650 1.00   593
## s(season,series).15 -0.13431343 -0.043108950  0.03000398 1.01   351
## s(season,series).16 -1.49183000 -0.568550000  0.25883737 1.01   510

##

## GAM smoothing parameter (rho) estimates:

##                          2.5%        50%     97.5% Rhat n.eff
## s(season)           1.938653  3.2577250  4.049817    1  1980
## s(season,series)  -2.669204 -2.0626650 -1.350627    1   509
## s(season,series)2 -0.735051  0.0580852  1.407613    1  1428

##

## Latent trend length scale (rho) estimates:

##                2.5%      50%    97.5% Rhat n.eff
## rho_gp[1] 3.922424 6.044235 8.050827    1  2342
## rho_gp[2] 4.064420 6.038705 8.096381    1  2452

##

## Stan MCMC diagnostics

## n_eff / iter looks reasonable for all parameters
## Rhat looks reasonable for all parameters
## 0 of 1600 iterations ended with a divergence (0%)
## 0 of 1600 iterations saturated the maximum tree depth of 12 (0%)
## E-FMI indicated no pathological behavior

##
```

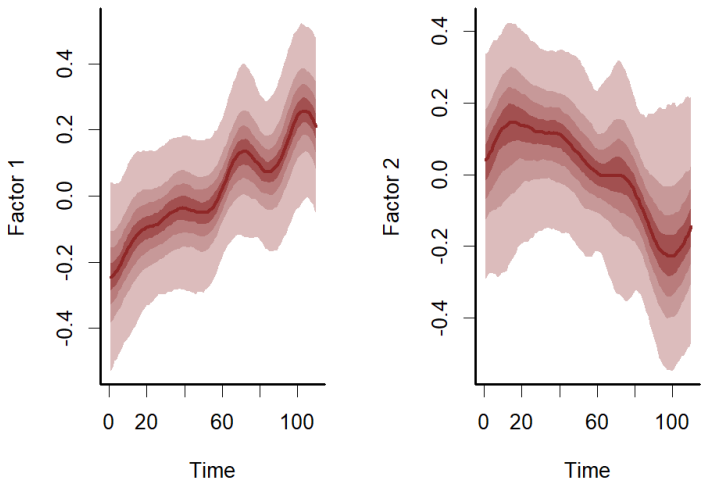
The pairs plot now looks much better

```
pairs(trends_mod3$model_output, pars = c('lv_coefs','rho_gp'))
```



Inspection of the dynamic factors and their relative contributions indicates that both factors are contributing the the series' latent trends

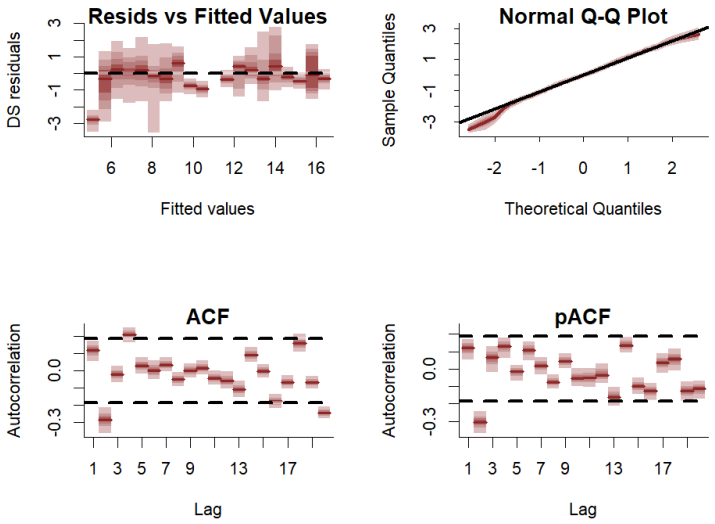
```
plot_mvgam_factors(trends_mod3)
```



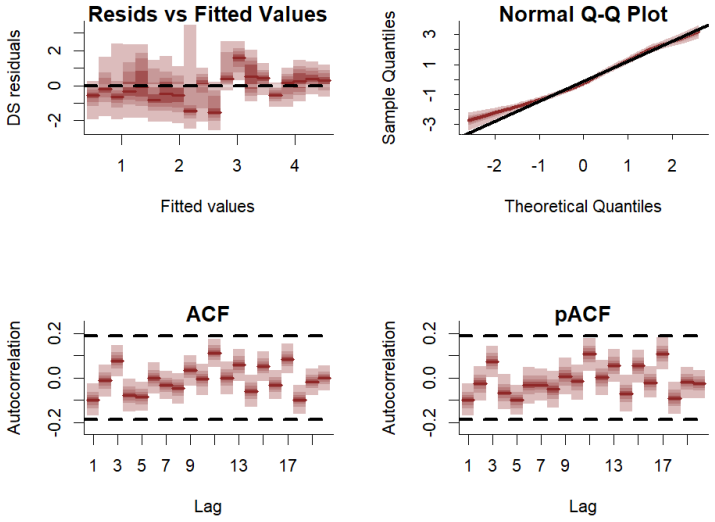

```
##         Contribution
## Factor1    0.5024311
## Factor2    0.4975689
```

Look at Dunn-Smyth residuals for some series from this preferred model to ensure that our dynamic factor process has captured most of the temporal dependencies in the observations

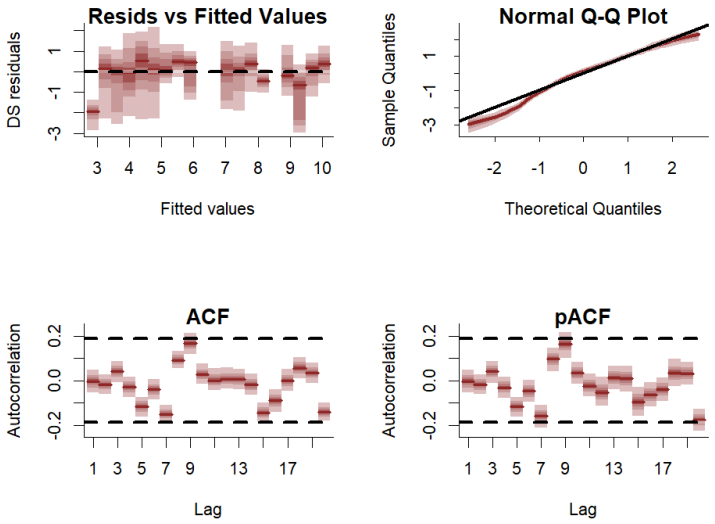
```
plot_mvgam_resids(trends_mod3, series = 1)
```



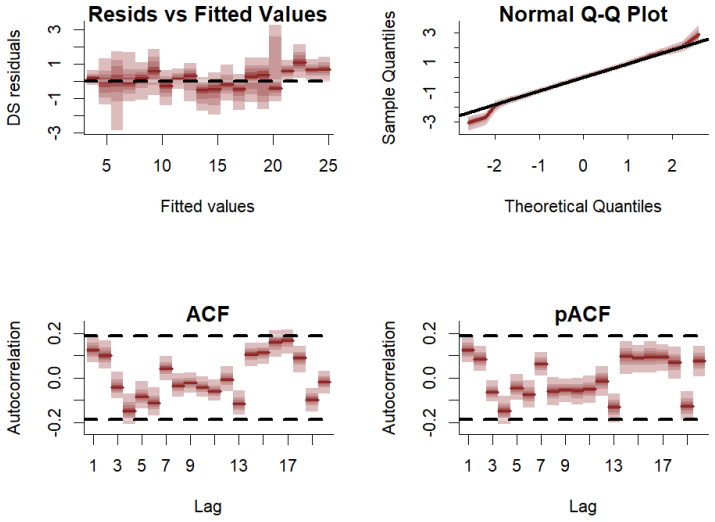

```
plot_mvgam_resids(trends_mod3, series = 2)
```




```
plot_mvgam_resids(trends_mod3, series = 3)
```

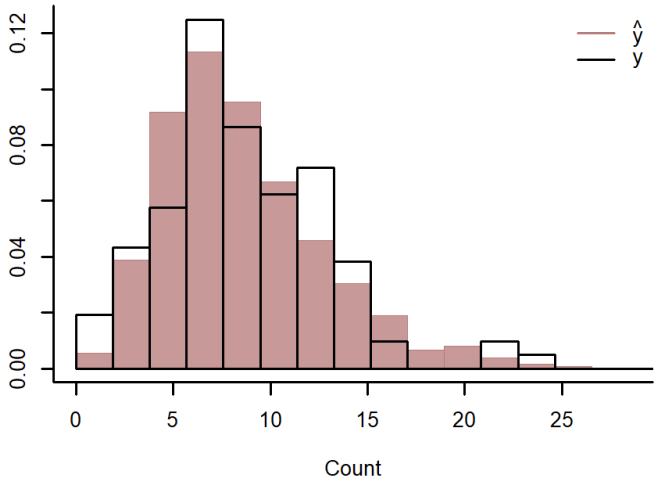



```
plot_mvgam_resids(trends_mod3, series = 4)
```

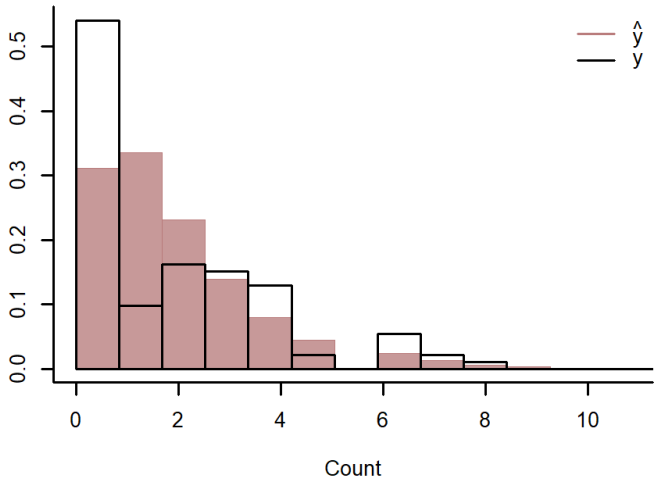


Perform posterior predictive checks to see if the model is able to simulate data that looks realistic and unbiased by examining simulated kernel densities for posterior predictions (`yhat`) compared to the density of the observations (`y`). This will be particularly useful for examining whether the Poisson observation model is able to produce realistic looking simulations for each individual series.

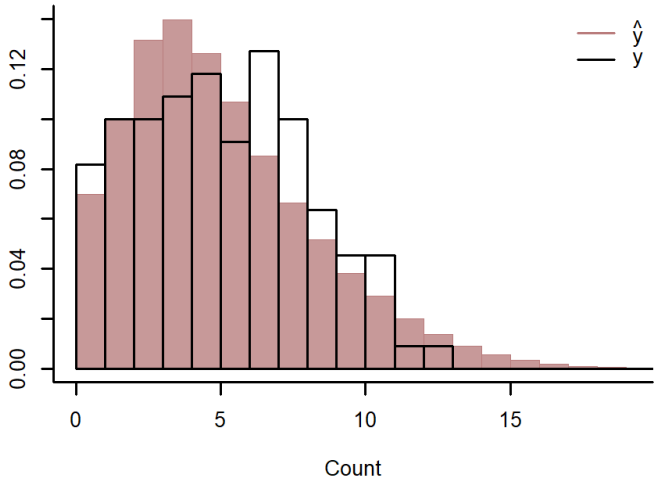
```
ppc(trends_mod3, series = 1, type = 'hist')
```



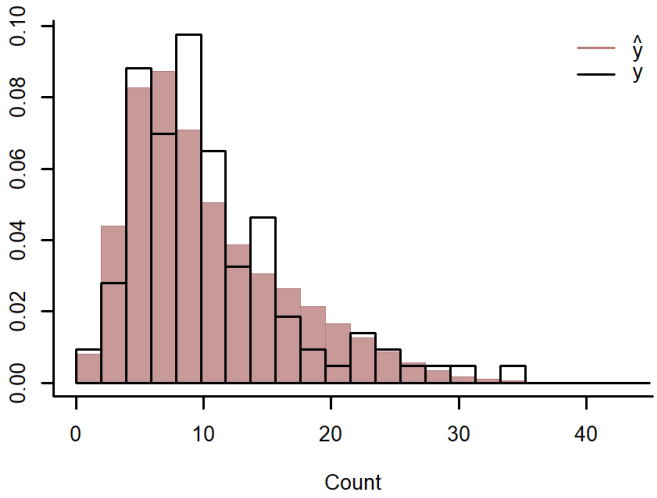

```
ppc(trends_mod3, series = 2, type = 'hist')
```




```
ppc(trends_mod3, series = 3, type = 'hist')
```

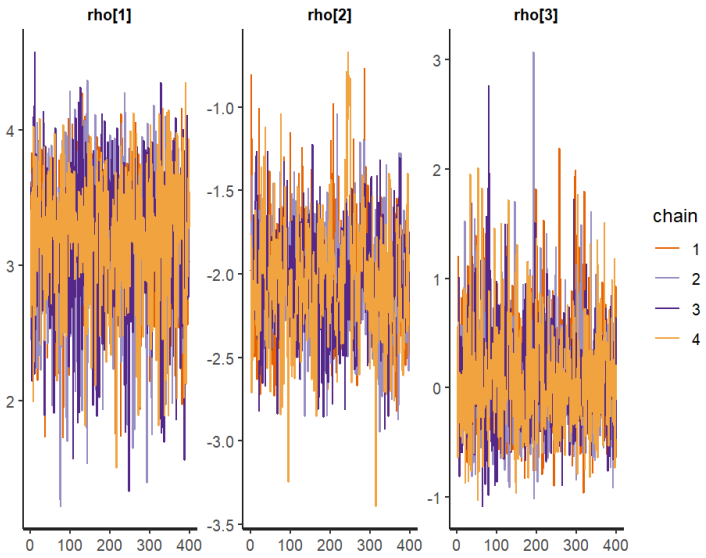



```
ppc(trends_mod3, series = 4, type = 'hist')
```



Look at traceplots for the smoothing parameters (`rho`)

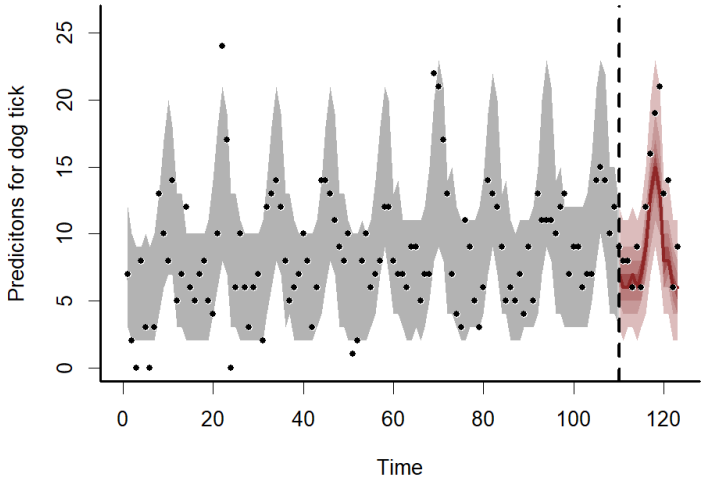
```
rstan::stan_trace(trends_mod3$model_output, pars = 'rho')
```



Plot posterior predictive distributions for the training and testing periods for each series

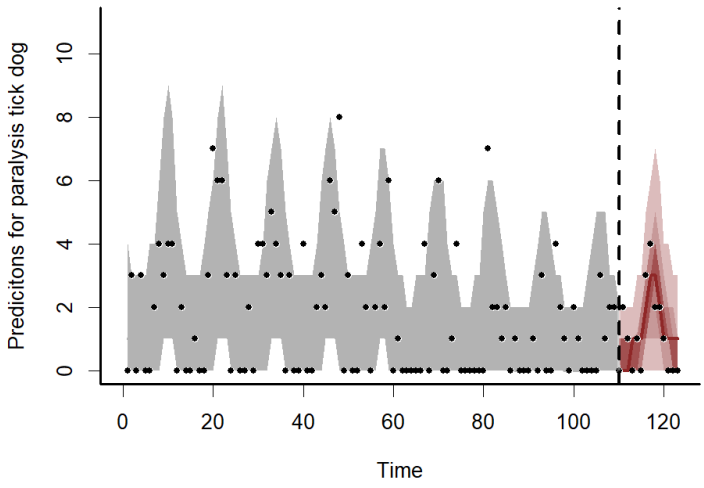
```
plot_mvgam_fc(object = trends_mod3, series = 1, newdata = trends_data$data_test)
```

```
## Out of sample DRPS:
## [1] 26.10525
##
```



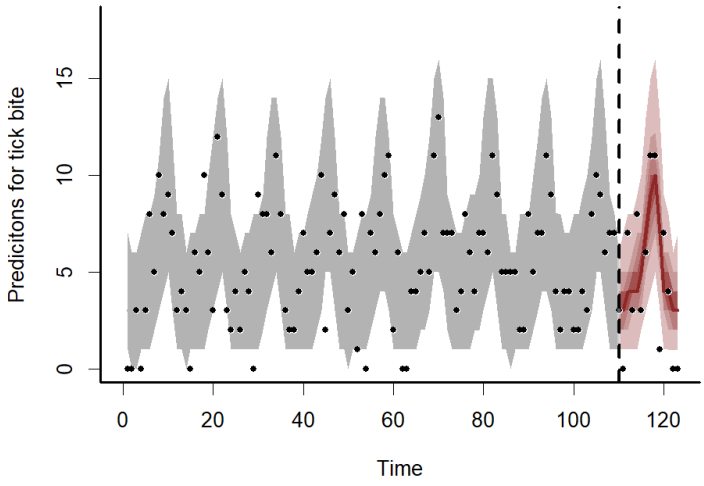

```
plot_mvgam_fc(object = trends_mod3, series = 2, newdata = trends_data$data_test)
```

```
## Out of sample DRPS:
## [1] 6.710453
##
```



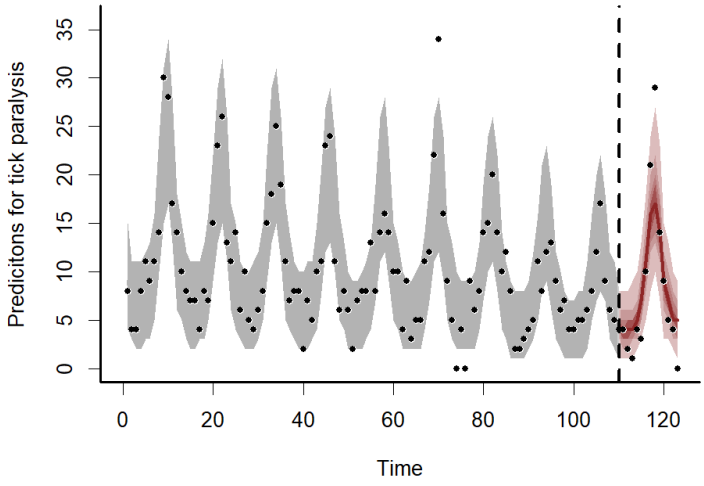

```
plot_mvgam_fc(object = trends_mod3, series = 3, newdata = trends_data$data_test)
```

```
## Out of sample DRPS:
## [1] 23.47054
##
```



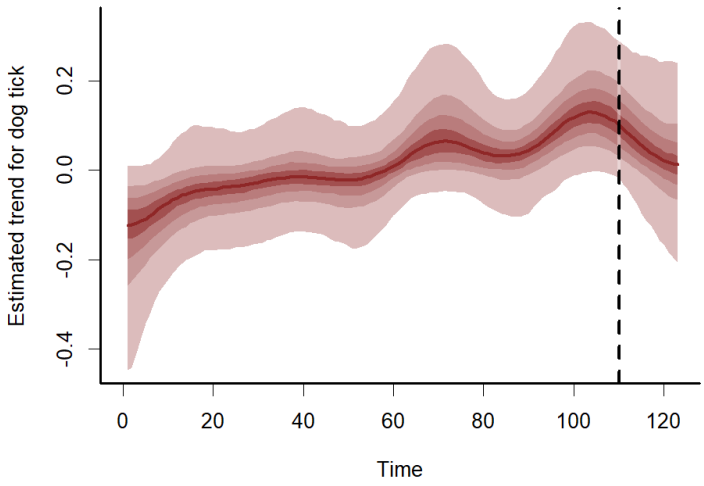

```
plot_mvgam_fc(object = trends_mod3, series = 4, newdata = trends_data$data_test)
```

```
## Out of sample DRPS:
## [1] 27.49546
##
```

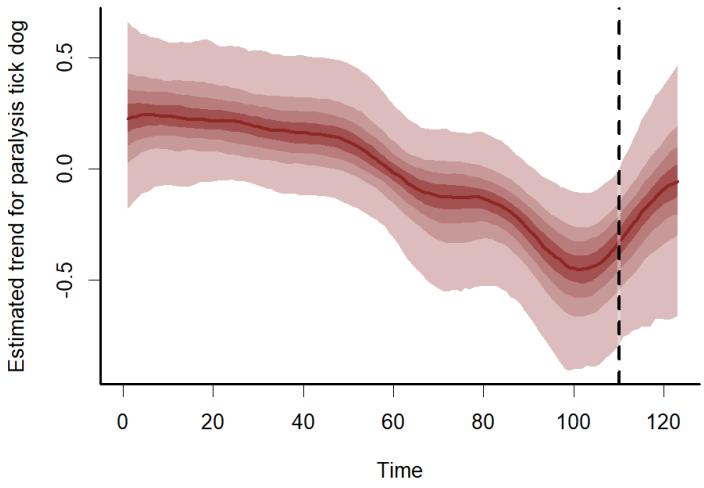


Plot posterior distributions for the latent trend estimates, again for the training and testing periods

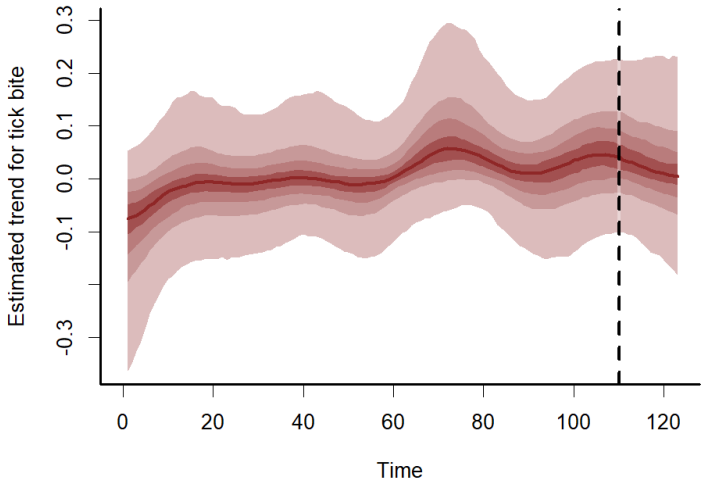
```
plot_mvgam_trend(object = trends_mod3, series = 1, newdata = trends_data$data_test)
```



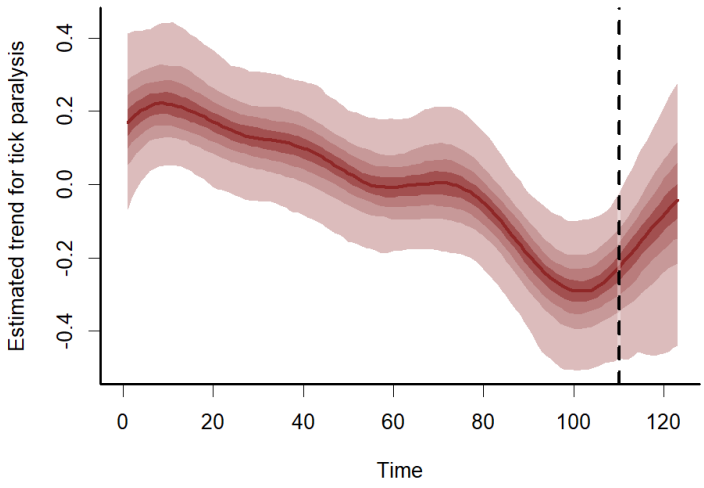

```
plot_mvgam_trend(object = trends_mod3, series = 2, newdata = trends_data$data_test)
```




```
plot_mvgam_trend(object = trends_mod3, series = 3, newdata = trends_data$data_test)
```

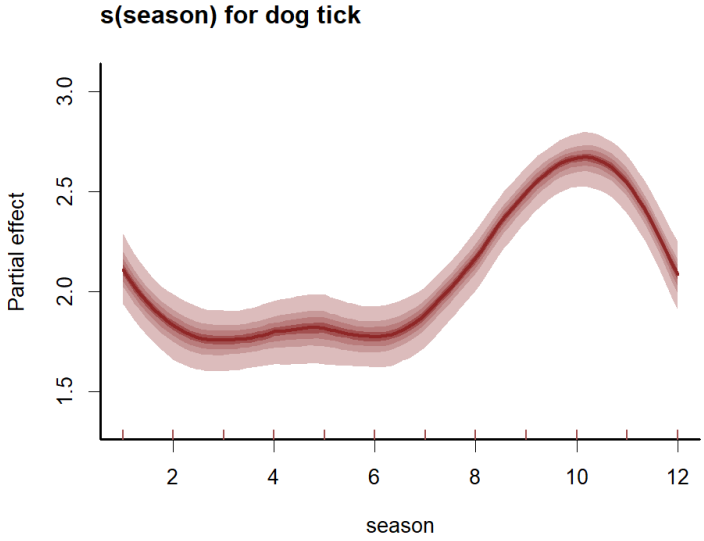



```
plot_mvgam_trend(object = trends_mod3, series = 4, newdata = trends_data$data_test)
```



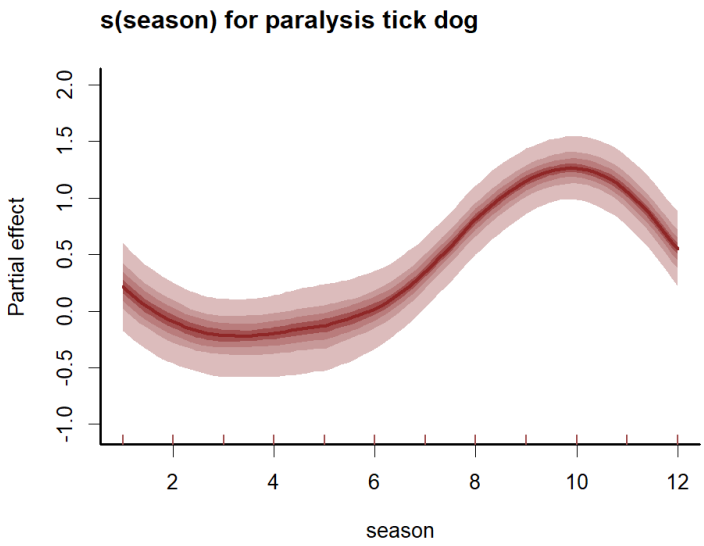
Given that we fit a model with hierarchical seasonality, the seasonal smooths are able to deviate from one another (though they share the same wiggliness and all deviate from a common 'global' seasonal function). Here we use the `newdata` argument to generate predictions for each of the hierarchical smooth functions (note that the intercept is still included in these plots so they do not center on zero)

```
newdat <- data.frame(season = seq(1, 12, length.out = 100),
                     series = levels(trends_data$data_train$series)[1])

plot_mvgam_smooth(object = trends_mod3, series = 1,
                  smooth = 'season',
                  newdata = newdat)
```

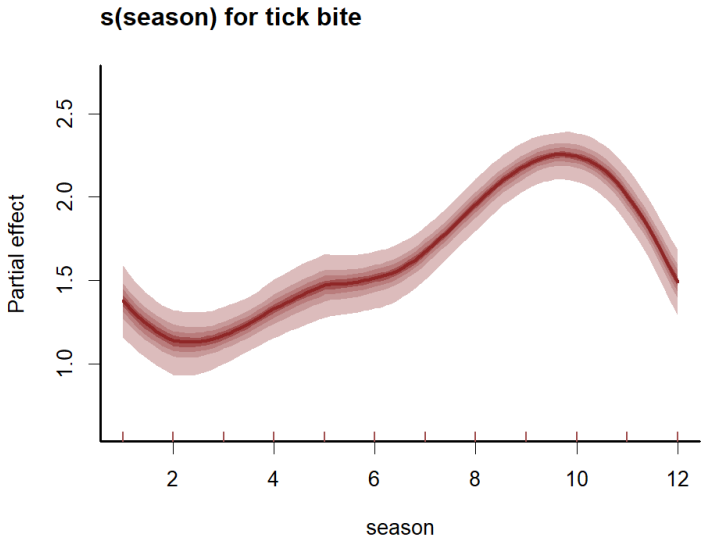



```
newdat <- data.frame(season = seq(1, 12, length.out = 100),
                     series = levels(trends_data$data_train$series)[2])

plot_mvgam_smooth(object = trends_mod3, series = 2,
```

```
                  smooth = 'season',
                  newdata = newdat)
```

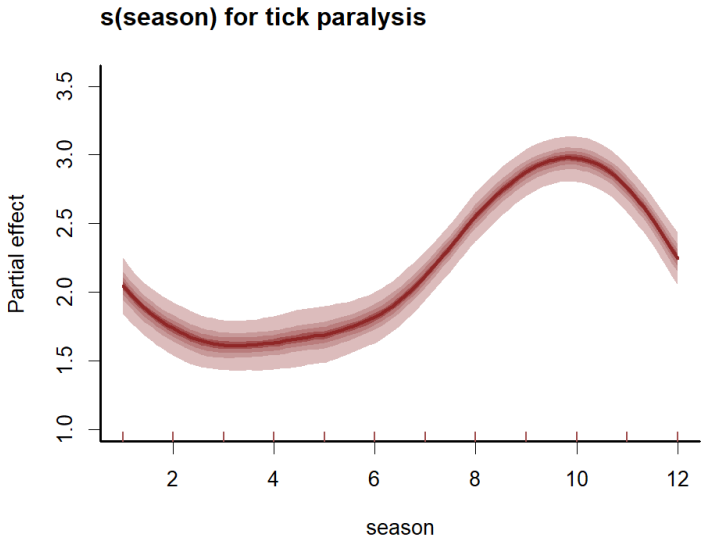



```
newdat <- data.frame(season = seq(1, 12, length.out = 100),
                     series = levels(trends_data$data_train$series)[3])

plot_mvgam_smooth(object = trends_mod3, series = 3,
                  smooth = 'season',
                  newdata = newdat)
```




```
newdat <- data.frame(season = seq(1, 12, length.out = 100),
                     series = levels(trends_data$data_train$series)[4])

plot_mvgam_smooth(object = trends_mod3, series = 4,
                  smooth = 'season',
```

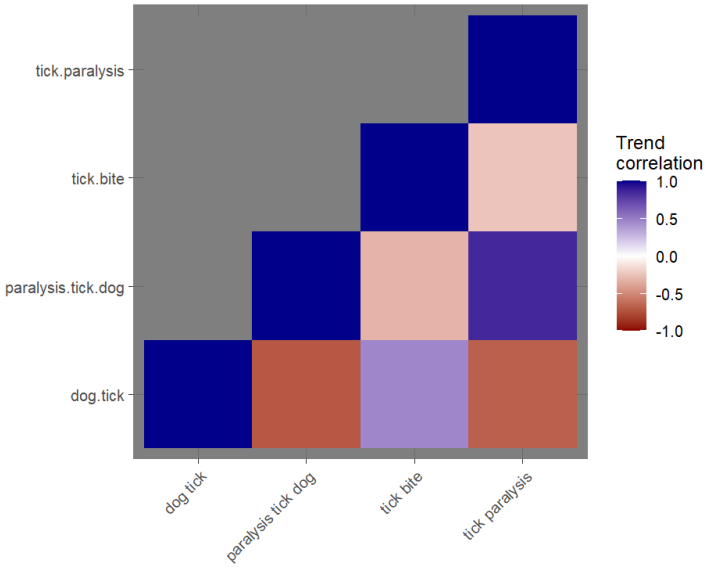
```
                  newdata = newdat)
```



Plot posterior mean estimates of latent trend correlations. These correlations are more useful than looking at latent factor loadings, for example to inspect ordinations. This is because the orders of the loadings (although constrained for identifiability purposes) can vary from chain to chain

```
correlations <- lv_correlations(object = trends_mod3)
```

```
library(ggplot2)
mean_correlations <- correlations$mean_correlations
mean_correlations[upper.tri(mean_correlations)] <- NA
mean_correlations <- data.frame(mean_correlations)
ggplot(mean_correlations %>%
         tibble::rownames_to_column("series1") %>%
         tidyr::pivot_longer(-c(series1), names_to = "series2", values_to = "Correlation"),
       aes(x = series1, y = series2)) + geom_tile(aes(fill = Correlation)) +
  scale_fill_gradient2(low="darkred", mid="white", high="darkblue",
                       midpoint = 0,
                       breaks = seq(-1,1,length.out = 5),
                       limits = c(-1, 1),
                       name = 'Trend\ncorrelation') + labs(x = '', y = '') + theme_dark() +
  theme(axis.text.x = element_text(angle = 45, hjust=1))
```

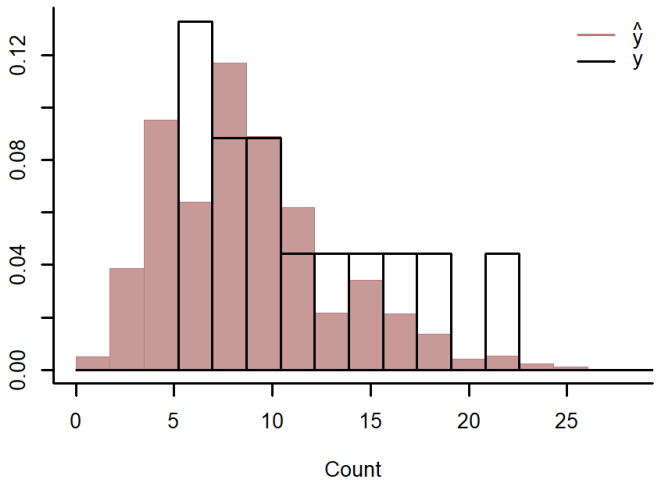


There is certainly some evidence of positive trend correlations for a few of these search terms, which is not surprising given how similar some of them are and how closely linked they should be to interest about tick paralysis in Queensland. Plot some STL decompositions of these series to see if these trends are noticeable in the data

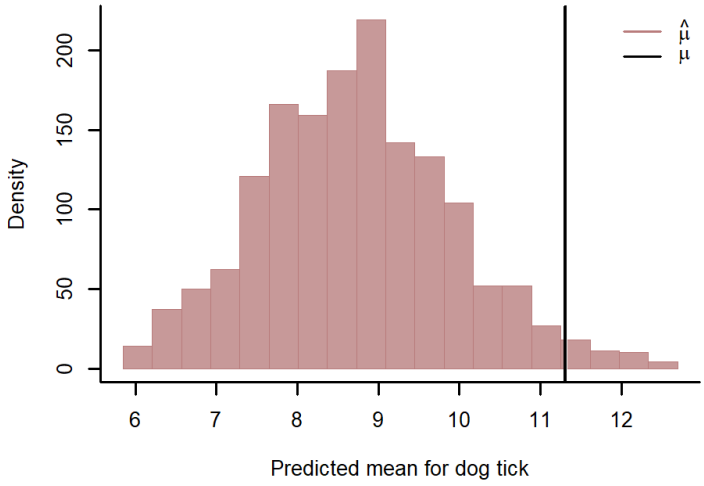
```
plot(stl(ts(as.vector(series$`tick paralysis`), frequency = 12), 'periodic'))
plot(stl(ts(as.vector(series$`paralysis tick dog`), frequency = 12), 'periodic'))
plot(stl(ts(as.vector(series$`dog tick`), frequency = 12), 'periodic'))
plot(stl(ts(as.vector(series$`tick bite`), frequency = 12), 'periodic'))
```

Forecast period posterior predictive checks suggest that the model still has room for improvement:

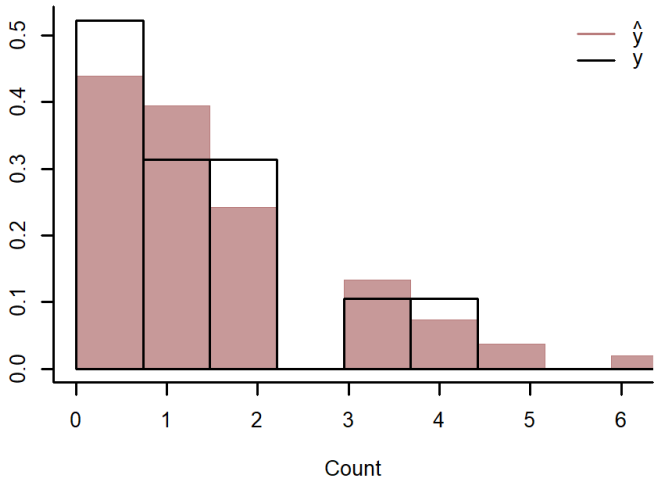
```
ppc(trends_mod3, series = 1, type = 'hist', newdata = trends_data$data_test)
```



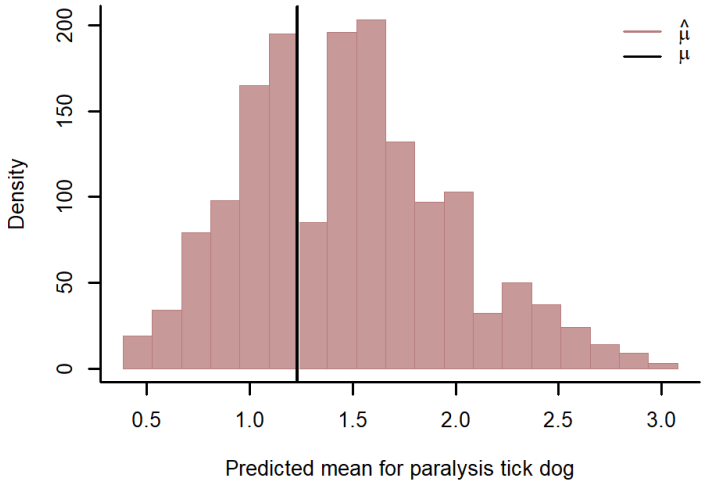

```
ppc(trends_mod3, series = 1, type = 'mean', newdata = trends_data$data_test)
```



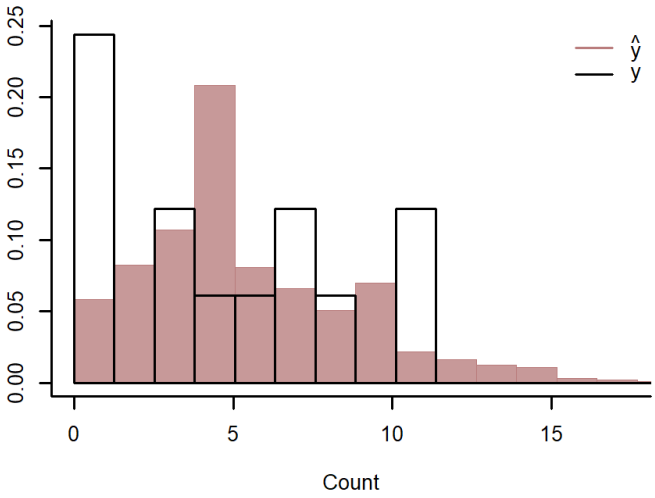

```
ppc(trends_mod3, series = 2, type = 'hist', newdata = trends_data$data_test)
```



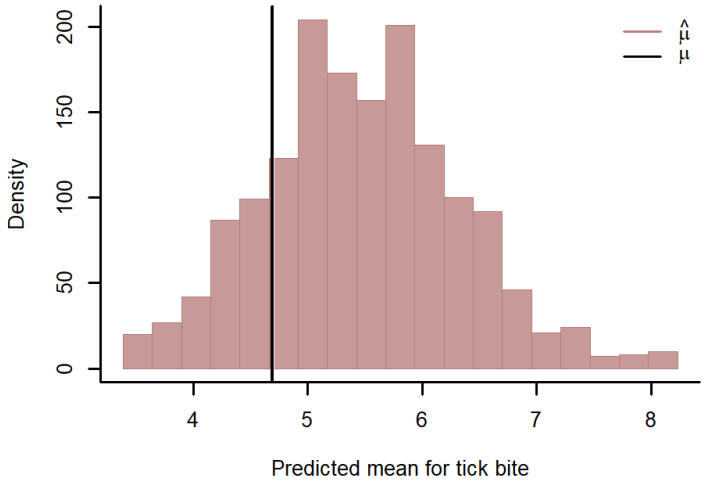

```
ppc(trends_mod3, series = 2, type = 'mean', newdata = trends_data$data_test)
```



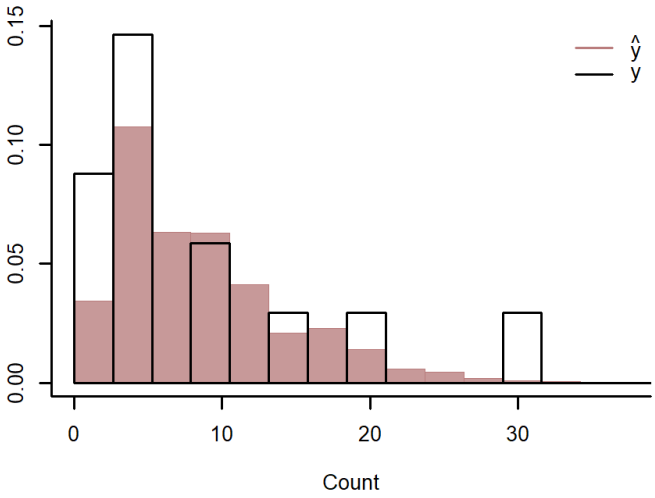

```
ppc(trends_mod3, series = 3, type = 'hist', newdata = trends_data$data_test)
```



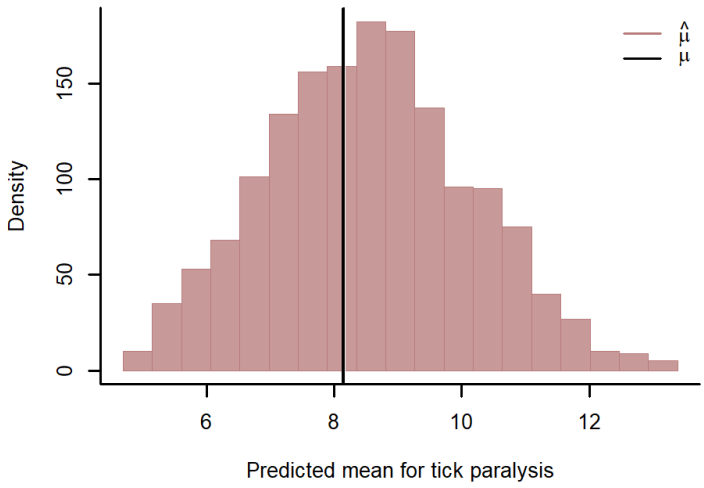

```
ppc(trends_mod3, series = 3, type = 'mean', newdata = trends_data$data_test)
```




```
ppc(trends_mod3, series = 4, type = 'hist', newdata = trends_data$data_test)
```




```
ppc(trends_mod3, series = 4, type = 'mean', newdata = trends_data$data_test)
```



Other next steps could involve devising a more goal-specific set of posterior predictive checks (see this paper by Gelman et al and relevant works by Betancourt for examples) and compare out of sample Discrete Rank Probability Scores for this model and other versions for the observations (Negative Binomial) and latent trends (i.e. AR1, Random Walk)

References

Clark, N.J. and Wells, K. (2022). Dynamic Generalized Additive Models (DGAMs) for forecasting discrete ecological time series. *Methods in Ecology and Evolution*. DOI: https://doi.org/10.1111/2041-210X.13974

Welty, L.J., et al. (2009). Bayesian distributed lag models: estimating effects of particulate matter air pollution on daily mortality *Biometrics* 65.1: 282-291