mvgam case study 3: distributed lag models

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

Here we will use the `mvgam` package (DGAMs; Clark & Wells, 2022), which fits dynamic GAMs using MCMC sampling via either the `JAGS` software (installation links are found here) or via the `Stan` software (installation links are found here), to estimate paramaters of a Bayesian distributed lag model. These models are used to describe simultaneously non-linear and delayed functional relationships between a covariate and a response, and are sometimes referred to as exposure-lag-response models. If we assume $\tilde{\boldsymbol{y}}_t$ is the conditional expectation of a discrete response variable $\boldsymbol{y}$ at time $\boldsymbol{t}$, the linear predictor for a dynamic distributed lag GAM with one lagged covariate is written as:

$$log(\tilde{\boldsymbol{y}}_t) = \boldsymbol{B}_0 + \sum_{k=1}^{K} f(\boldsymbol{b}_{k,t}\boldsymbol{x}_{k,t}) + \boldsymbol{z}_t \,,$$

where $\boldsymbol{B}_0$ is the unknown intercept, the $\boldsymbol{b}$'s are unknown spline coefficients estimating how the functional effect of covariate $(\boldsymbol{x})$ on $log(\tilde{\boldsymbol{y}}_t)$ changes over increasing lags (up to a maximum lag of $(\boldsymbol{K})$) and $\boldsymbol{z}$ is a dynamic latent trend component.

To demonstrate how these models are estimated in `mvgam`, first we load the Portal rodents capture data, which are available from the `portalr` package

```
#devtools::install_github("nicholasjclark/mvgam")
library(mvgam)
library(dplyr)
portal_dat <- read.csv('https://raw.githubusercontent.com/nicholasjclark/mvgam/master/manuscript_analys
```

```
## 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

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:nlme':
##
##     collapse

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

We'll keep data from the year 2004 onwards to make the model quicker to estimate for this simple example

```
portal_dat %>%
  dplyr::filter(year >= 2004) %>%
  dplyr::group_by(year, month) %>%
  dplyr::slice_head(n = 1) -> portal_dat_all
```

Below is an exact reproduction of Simon Wood's lag matrix function (which he uses in his distributed lag example from his book Generalized Additive Models - An Introduction with R 2nd edition). Here we supply a vector and specify the maximum lag that we want, and it will return a matrix of dimension `length(x) * lag`. Note that `NA`s are used for the missing lag values at the beginning of the matrix. In essence, the matrix objects represent exposure histories, where each row represents the lagged values of the predictor that correspond to each observation in `y`

```
lagard <- function(x, n.lag = 6) {
  n <- length(x); X <- matrix(NA, n, n.lag)
  for (i in 1:n.lag) X[i:n, i] <- x[i:n - i + 1]
  X
}
```

Organise all data needed for modelling into a list. We will focus only on the species *Chaetodipus penicillatus* (labelled as `PP`), which shows reasonable seasonality in its captures over time

```
data_all <- list(lag=matrix(0:5,nrow(portal_dat_all),6,byrow=TRUE),
            y = portal_dat_all$PP,
            season = portal_dat_all$month,
            year = portal_dat_all$year,
            series = rep(as.factor('series1'), NROW(portal_dat_all)),
            time = 1:NROW(portal_dat_all))
data_all$precip <- lagard(portal_dat_all$precipitation)
data_all$mintemp <- lagard(portal_dat_all$mintemp)
```

The exposure history matrix elements of the data list look as follows:

```
head(data_all$lag, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    1    2    3    4    5
## [2,]    0    1    2    3    4    5
## [3,]    0    1    2    3    4    5
## [4,]    0    1    2    3    4    5
## [5,]    0    1    2    3    4    5
```

```
head(data_all$precip, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 37.8   NA   NA   NA   NA   NA
## [2,]  8.7 37.8   NA   NA   NA   NA
## [3,] 43.5  8.7 37.8   NA   NA   NA
## [4,] 23.9 43.5  8.7 37.8   NA   NA
## [5,]  0.9 23.9 43.5  8.7 37.8   NA
```

```
head(data_all$mintemp, 5)
```

```
##        [,1]   [,2]   [,3]   [,4]  [,5] [,6]
## [1,] -9.710     NA     NA     NA    NA   NA
## [2,] -5.924 -9.710     NA     NA    NA   NA
## [3,] -0.220 -5.924 -9.710     NA    NA   NA
```

```
## [4,]  1.931 -0.220 -5.924 -9.710    NA    NA
## [5,]  6.568  1.931 -0.220 -5.924 -9.71   NA
```

All other elements of the data list are in the usual vector format

```r
head(data_all$y, 5)
```

```
## [1]  0  1  2 NA 10
```

```r
head(data_all$series, 5)
```

```
## [1] series1 series1 series1 series1 series1
## Levels: series1
```

```r
head(data_all$year, 5)
```

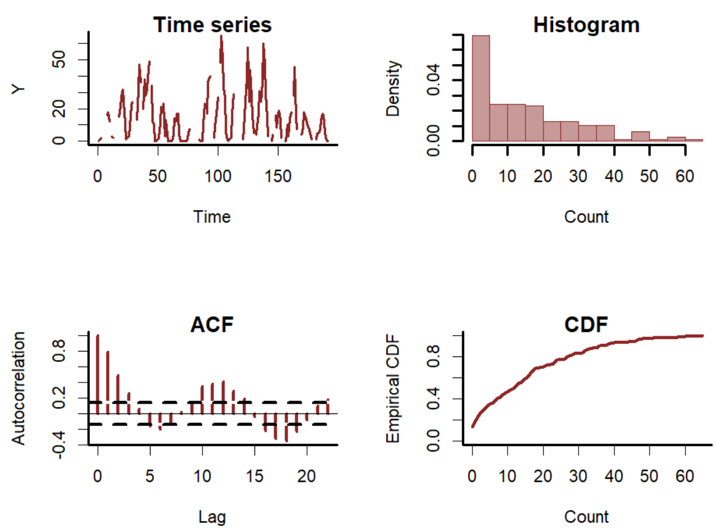
```
## [1] 2004 2004 2004 2004 2004
```

```r
head(data_all$time, 5)
```

```
## [1] 1 2 3 4 5
```

View the raw series. There is a clear seasonal pattern to the data, and there are missing values scattered throughout

```r
plot_mvgam_series(data = data_all)
```



Create training and testing sets; start at observation 7 so that the `NA` values at the beginning of the covariate lag matrices are not included. Currently there is no option for on-the-fly imputation of missing covariate values in `mvgam` models, though this can easily be done in `JAGS` by specifying prior distributions over these missing entries

```r
data_train <- list(lag = data_all$lag[7:174,],
                   y = data_all$y[7:174],
                   series = data_all$series[7:174],
                   season = data_all$season[7:174],
                   year = data_all$year[7:174],
                   time = 7:174,
                   precip = data_all$precip[7:174,],
                   mintemp = data_all$mintemp[7:174,])
```

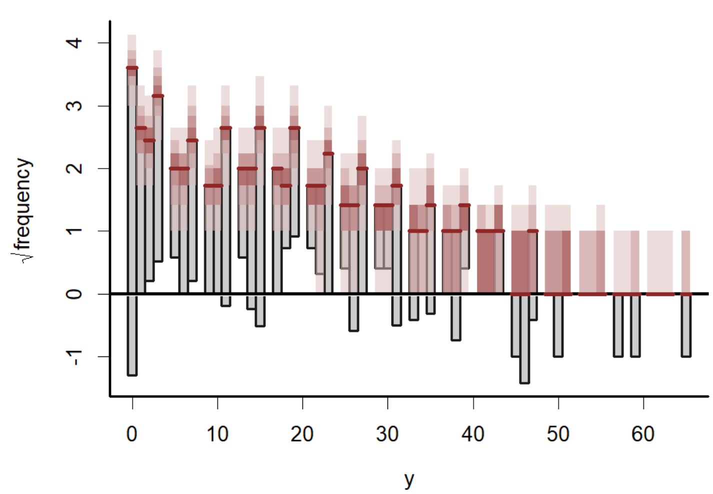
```
data_test <- list(lag = data_all$lag[175:length(data_all$y),],
                  y = data_all$y[175:length(data_all$y)],
                  series = data_all$series[175:length(data_all$y)],
                  season = data_all$season[175:length(data_all$y)],
                  year = data_all$year[175:length(data_all$y)],
                  time = 175:length(data_all$y),
                  precip = data_all$precip[175:length(data_all$y),],
                 mintemp = data_all$mintemp[175:length(data_all$y),])
```

Now we can fit a Bayesian GAM with distributed lag terms for precipitation and minimum temperature. The distributed lags are set up as tensor product smooth functions (see `help(te)` for an explanation of tensor product smooth constructions in the `mgcv` package) between `lag` and each covariate. We will start simply by assuming our data follow a `Poisson` observation process

```
mod1 <- mvgam(formula =  y ~ te(mintemp, lag, k = c(6, 6)) +
                te(precip, lag, k = c(6, 6)),
               data = data_train,
               newdata = data_test,
               family = 'poisson',
               chains = 4,
               burnin = 15000,
               trend_model = 'None')
```

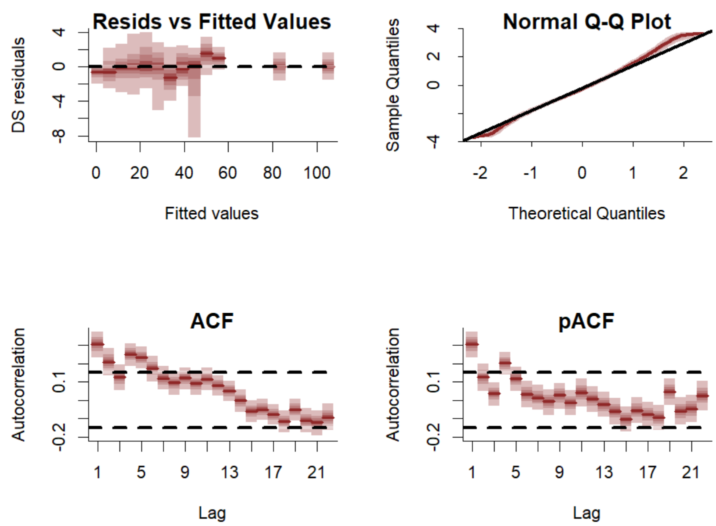
Posterior predictive rootograms are a useful way to explore whether a discrete model is able to capture relevant dispersion in the observed data. This plot compares the frequencies of observed vs predicted values for each bin, which can help to identify aspects of poor model fit. For example, if the gray bars (representing observed frequencies) tend to stretch below zero, this suggests the model's simulations predict the values in that particular bin less frequently than they are observed in the data. A well-fitting model that can generate realistic simulated data will provide a rootogram in which the lower boundaries of the grey bars are generally near zero

```
ppc(mod1, type = 'rootogram')
```



The `Poisson` model is not doing a great job of capturing dispersion, underpredicting the zeros in the data and overpredicting some of the medium-range values (counts of `~5-30`). The residual `Q-Q` plot confirms that the `Poisson` is not an appropriate distribution for these data

```r
plot(mod1, type = 'residuals')
```

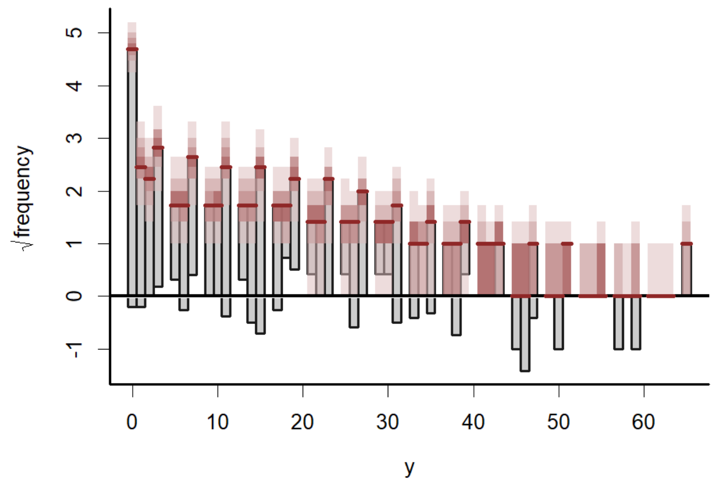


Given the overdispersion present in the data, we will now assume a `Geometric-Poisson` observation model, which can be more flexible than the `Negative binomial` for modelling overdispersed count data. In `mvgam` the `Geometric-Poisson` is estimated as a `Tweedie-Poisson` model with the power parameter `p` fixed at `1.5`

```r
mod2 <- mvgam(formula =  y ~ te(mintemp, lag, k = c(6, 6)) +
                 te(precip, lag, k = c(6, 6)),
                data = data_train,
                newdata = data_test,
                family = 'tw',
                chains = 4,
                burnin = 15000,
                trend_model = 'None')
```

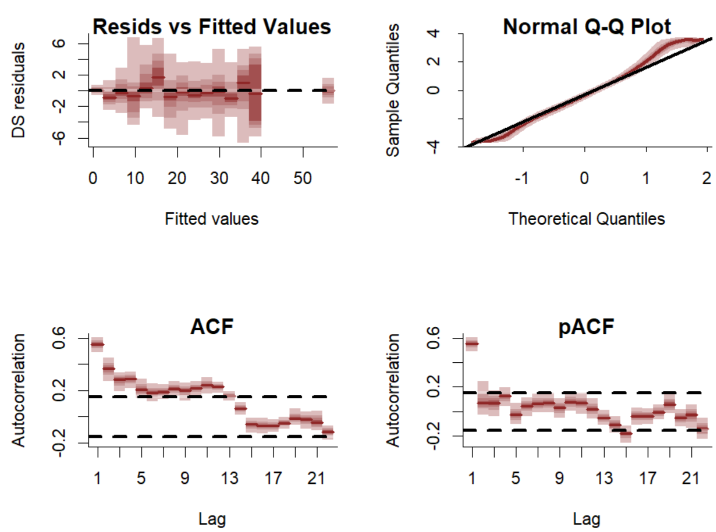
The rootogram for this model looks better, though of course there is still some overprediction of medium-range values

```r
ppc(mod2, type = 'rootogram')
```



However, the residual plot looks much better for this model

```r
plot(mod2, type = 'residuals')
```



The summary of the model provides useful information on convergence for unobserved parameters. Notice how strongly positive the overdispersion parameter is estimated to be, providing further evidence that this overdispersion is important to capture for these data. However, the Gibbs samplers used by `JAGS` have not come close to converging and we really shouldn't put much faith in our estimates of parameter uncertainty

```r
summary(mod2)
```

```
## GAM formula:

## y ~ te(mintemp, lag, k = c(6, 6)) + te(precip, lag, k = c(6,
##     6))

##
```

```
## Family:
## Tweedie
##
## Link function:
## log
##
## Trend model:
## None
##
## N series:
## 1
##
## N observations:
## 168
##
## Status:
## Fitted using JAGS
##
## Dispersion parameter estimates:
##           2.5%      50%    97.5% Rhat n.eff
## twdis 1.448251 1.897852 2.548567 1.05    33
##
## GAM coefficient (beta) estimates:
##                              2.5%           50%        97.5% Rhat n.eff
## (Intercept)          2.2731484059  2.4413785145  2.584366680 1.08    59
## te(mintemp,lag).1  -0.3604488091  0.0529614556  0.430121232 2.36    13
## te(mintemp,lag).2  -0.7929549097 -0.1292851799  0.347877702 3.05    32
## te(mintemp,lag).3  -0.8460771978 -0.2277312802  0.236074076 3.43    16
## te(mintemp,lag).4  -0.8683007068 -0.3641203394  0.058736420 2.05    11
## te(mintemp,lag).5  -1.0646225415 -0.4573841749  0.014208064 2.24    10
## te(mintemp,lag).6  -1.5344385647 -1.1004407300 -0.189743031 6.60    25
## te(mintemp,lag).7  -0.6931986984 -0.5535144278 -0.341549083 1.81    20
## te(mintemp,lag).8  -0.3233507651 -0.0134466027  0.290489418 4.31    23
## te(mintemp,lag).9   0.0266759088  0.2525224885  0.759075168 3.64    15
## te(mintemp,lag).10  0.0243533704  0.3542570204  0.645535348 1.77     9
## te(mintemp,lag).11 -0.1459796530  0.2875222328  0.888740591 3.35    16
## te(mintemp,lag).12 -0.9477064997 -0.5965323414  0.165049954 6.23    27
## te(mintemp,lag).13 -0.4758890359 -0.2754686811 -0.157602209 1.39    32
## te(mintemp,lag).14 -0.5424751106 -0.2137627721  0.171015921 5.00    12
## te(mintemp,lag).15 -0.3275437483 -0.0874481228  0.282580752 5.39    24
## te(mintemp,lag).16  0.0208140919  0.1426392642  0.309001665 1.62    15
## te(mintemp,lag).17 -0.0124544435  0.2371409060  0.653327111 2.68    17
## te(mintemp,lag).18 -0.4427820478 -0.2623611140  0.382379990 2.76    21
```

```
## te(mintemp,lag).19 -0.2203176912 -0.0575121388  0.128427716 1.91    21
## te(mintemp,lag).20 -0.3199531387 -0.1228486726  0.295865551 3.87    14
## te(mintemp,lag).21 -0.3352604782 -0.1465185816  0.245162648 4.12    17
## te(mintemp,lag).22 -0.2850961028  0.0184437174  0.278680754 2.68    13
## te(mintemp,lag).23 -0.0004168315  0.1579149678  0.351945924 1.23    16
## te(mintemp,lag).24 -0.2137611287  0.1282806175  0.460935613 1.44    13
## te(mintemp,lag).25 -0.1124248029  0.1190433446  0.399121396 2.23    14
## te(mintemp,lag).26 -0.1987911376  0.0414237424  0.271986631 2.27    21
## te(mintemp,lag).27 -0.3213291262 -0.0641567535  0.176391670 2.92    20
## te(mintemp,lag).28 -0.4224972973 -0.0899725779  0.082753369 3.90    24
## te(mintemp,lag).29 -0.4175936099 -0.1204835326  0.152290729 1.39    15
## te(mintemp,lag).30 -0.0659720757  0.5111200790  0.934600996 2.79    19
## te(mintemp,lag).31  0.0468764534  0.2773058799  0.458671555 2.25    20
## te(mintemp,lag).32 -0.1766718805  0.0979073411  0.229956140 2.01    25
## te(mintemp,lag).33 -0.2216620755 -0.0723127763  0.117741765 1.28    21
## te(mintemp,lag).34 -0.4123941925 -0.2539720926 -0.080615804 1.98    30
## te(mintemp,lag).35 -1.0180297699 -0.6124845598 -0.137637901 2.59    18
## te(precip,lag).1   -0.1620325055  0.0085991938  0.125132306 4.76    33
## te(precip,lag).2   -0.1536521022 -0.0113744881  0.079087580 2.30    29
## te(precip,lag).3   -0.1659387082 -0.0002769955  0.113387207 2.09    14
## te(precip,lag).4   -0.2260725872 -0.0152844970  0.107461966 3.38    25
## te(precip,lag).5   -0.3153282557 -0.0886925019  0.100619406 2.73    16
## te(precip,lag).6   -0.0193519045  0.0712119521  0.129200975 1.10    23
## te(precip,lag).7   -0.1225476939  0.0100653395  0.100237189 2.60    17
## te(precip,lag).8   -0.1155566775 -0.0131179981  0.059649853 1.96    26
## te(precip,lag).9   -0.1323016728 -0.0320119270  0.108441174 2.72    32
## te(precip,lag).10  -0.1845031823 -0.0376553442  0.031760244 2.93    26
## te(precip,lag).11  -0.3167778012 -0.1293117102  0.043412274 3.09    23
## te(precip,lag).12  -0.2098989539 -0.0011098801  0.106316904 1.32    14
## te(precip,lag).13  -0.1170669946 -0.0498200378  0.016362477 1.13    28
## te(precip,lag).14  -0.2150554466 -0.0761556146  0.029287876 1.76    17
## te(precip,lag).15  -0.2283829590 -0.1183178463 -0.003860745 2.09    15
## te(precip,lag).16  -0.2515442418 -0.1468871724 -0.008933124 1.59    15
## te(precip,lag).17  -0.3601679295 -0.1921159085 -0.014328538 1.76    11
## te(precip,lag).18  -0.1746976548  0.0001561644  0.183366905 1.49    12
## te(precip,lag).19  -0.1362025869 -0.0301385549  0.033498952 1.99    20
## te(precip,lag).20  -0.1657601975 -0.0796147182  0.024664389 1.75    19
## te(precip,lag).21  -0.2761990830 -0.1593822626 -0.089822083 1.71    24
## te(precip,lag).22  -0.2834436697 -0.1909639386 -0.126457198 1.17    14
## te(precip,lag).23  -0.2972757531 -0.1741699076 -0.041675310 1.41    16
## te(precip,lag).24  -0.2875034931 -0.1194384521  0.095785917 2.15    21
## te(precip,lag).25  -0.1177330281 -0.0164836415  0.125221455 2.84    19
## te(precip,lag).26  -0.0279477623  0.0291678370  0.077249084 1.14    30
## te(precip,lag).27  -0.0564017720  0.1162873951  0.271466991 1.71    21
## te(precip,lag).28  -0.0957368611 -0.0027673881  0.114429819 2.20    33
## te(precip,lag).29  -0.0628872462  0.0033466389  0.177678987 2.76    18
## te(precip,lag).30  -0.0552937158  0.0032661290  0.207455722 2.56    25

##

## GAM smoothing parameter (rho) estimates:

##                          2.5%      50%    97.5% Rhat n.eff
## te(mintemp,lag)    1.52390717 2.686716 3.620421 1.39   292
## te(mintemp,lag)2   2.80948955 3.698925 4.531046 1.18    90
```

```
## te(mintemp,lag)3  0.23528463 2.264601 4.092929 1.42    88
## te(precip,lag)    1.71026224 3.278495 4.269096 1.24   122
## te(precip,lag)2   1.82842693 3.199328 4.485885 2.36   221
## te(precip,lag)3  -0.02486839 2.727858 4.343052 1.16   914
```

```
##
```

```
## JAGS MCMC diagnostics
```

```
## Rhats above 1.05 found for 269 parameters
## *Diagnose further to investigate why the chains have not mixed
```

```
##
```

As this is a timeseries and the residual plot hints at some autocorrelation remaining in the short-term lags, lets check if an `AR` latent trend process improves forecasts compared to the no-trend model. An important note here is the choice of prior for the overdispersion parameter `twdis`. This parameter and the latent trend variance can interact strongly, particularly when overdispersion is in the data high. This is because at high values of the dispersion parameter, there is less need for a latent trend to be able to capture any outliers and so the latent trend precision can go up toward infinity, approaching a space of very diffuse likelihood that forces the Gibbs samplers to take on frustratingly small step sizes. Likewise when there is not much need for overdispersion, the dispersion parameter can approach zero and move around in an equally uninformative parameter space. The latent trend operates on the log scale, so really we should not expect autocorrelated jumps in trappings of more than `6-8` from timepoint to timepoint (any larger and the trend will compete strongly with the overdispersion parameter, making it difficult for us to model the inherent overdispersion process and instead assuming it is all autocorrelation). A containment prior on the latent trend `sigma` will help achieve this, but first we need to see the structure of the `priors` argument for making alterations to defaults

```
priors <- get_mvgam_priors(formula =  y ~ te(mintemp, lag, k = c(6, 6)) +
                 te(precip, lag, k = c(6, 6)),
                data = data_train,
                family = 'tw',
                trend_model = 'AR3')
priors
```

```
##       param_name param_length                                     param_info
## 1            ar1            1      trend AR1 coefficient (for each series s)
## 2            ar2            1      trend AR2 coefficient (for each series s)
## 3            ar3            1      trend AR3 coefficient (for each series s)
## 4 sigma<lower=0>            1                   trend sd (for each series s)
## 5      twdis_raw            1 log of Tweedie dispsersion (for each series s)
##                                    prior                 example_change
## 1                  ar1[s] ~ dnorm(0, 10)       ar1[s] ~ dnorm(-0.35, 0.28)
## 2                  ar2[s] ~ dnorm(0, 10)        ar2[s] ~ dnorm(-0.1, 0.82)
## 3                  ar3[s] ~ dnorm(0, 10)       ar3[s] ~ dnorm(-0.67, 0.63)
## 4          sigma[s] ~ dexp(2)T(0.075, 5)             sigma[s] ~ dexp(0.48)
## 5 twdis_raw[s] ~ dnorm(0, 2)T(-3.5, 3.5) twdis_raw[s] ~ dnorm(-0.49, 4.95)
```

The 5th row of this `data.frame` contains information on the prior for the latent trend variance component. It is the entry for the `prior` column of this row that needs to be modified before supplying the argument for `priors` to the `mvgam` function:

```
priors$prior[4] <- "dexp(2.5)T(0.15, 2)"
mod3 <- mvgam(formula =  y ~ te(mintemp, lag, k = c(6, 6)) +
                 te(precip, lag, k = c(6, 6)),
                data = data_train,
                newdata = data_test,
```

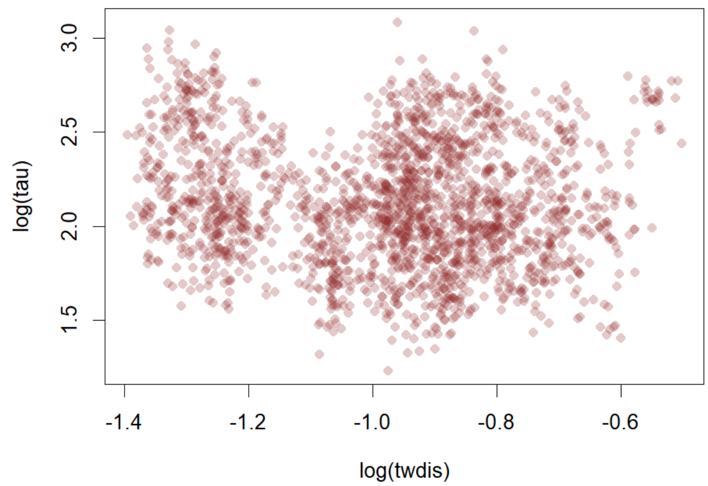
```r
               family = 'tw',
               priors = priors,
               chains = 4,
               burnin = 15000,
               trend_model = 'AR3')
summary(mod3)
```

```
## y ~ te(mintemp, lag, k = c(6, 6)) + te(precip, lag, k = c(6,
##     6))
## Tweedie
## log
## AR3
## 1
## 168
## Fitted using JAGS
##            2.5%      50%     97.5% Rhat n.eff
## twdis 0.2599945 0.390535 0.5381924 2.63    23
##                          2.5%          50%        97.5% Rhat n.eff
## (Intercept)          2.01252071  2.320126903  2.855187129 3.36    26
## te(mintemp,lag).1  -0.87097891  0.164084355  0.735975176 8.52    17
## te(mintemp,lag).2  -0.23405324 -0.002035650  0.697649886 6.57    22
## te(mintemp,lag).3  -0.20555527  0.050669337  0.455197784 4.12    22
## te(mintemp,lag).4  -0.59869404 -0.256313990  0.252806638 6.24    21
## te(mintemp,lag).5  -1.05563221 -0.544474581  0.322898386 5.05    17
## te(mintemp,lag).6  -1.29169356 -0.835204699 -0.131035563 6.47    19
## te(mintemp,lag).7  -0.77273933 -0.585210244 -0.079048360 5.33    19
## te(mintemp,lag).8  -0.50129894 -0.029170923  0.283537937 6.21    21
## te(mintemp,lag).9  -0.11842528  0.387926722  0.806431225 7.74    21
## te(mintemp,lag).10  0.09149259  0.406329814  0.826566021 5.01    10
## te(mintemp,lag).11 -0.05114064  0.498343134  1.231630615 8.38    25
## te(mintemp,lag).12 -0.73059958 -0.311156976  0.003043597 3.36    13
## te(mintemp,lag).13 -0.50089810 -0.267567906 -0.078088811 3.01    15
## te(mintemp,lag).14 -0.34864469 -0.113822194  0.256514238 5.80    25
## te(mintemp,lag).15 -0.14938238  0.086215896  0.580965961 8.13    15
## te(mintemp,lag).16  0.10982158  0.434245185  0.638605242 5.70    23
## te(mintemp,lag).17 -0.18056757  0.457367260  0.887594444 6.07    17
## te(mintemp,lag).18 -0.36565155 -0.096708526  0.184605217 3.44    16
## te(mintemp,lag).19 -0.32986644 -0.014711624  0.236098530 2.87    11
## te(mintemp,lag).20 -0.28817650 -0.023966407  0.296115077 4.96    16
## te(mintemp,lag).21 -0.24243382  0.031862320  0.516428000 6.38    12
## te(mintemp,lag).22 -0.06737608  0.255209786  0.520714821 5.87    19
## te(mintemp,lag).23 -0.24768125  0.342167430  0.677651300 3.89    12
## te(mintemp,lag).24 -0.36213337  0.098369555  0.447635169 3.52    11
## te(mintemp,lag).25 -0.21185494  0.085581962  0.343135619 4.13    39
## te(mintemp,lag).26 -0.19732007  0.082328257  0.417868255 3.21    25
## te(mintemp,lag).27 -0.27585918  0.018305106  0.348415767 5.50    11
## te(mintemp,lag).28 -0.41211050 -0.130442150  0.344498283 7.25    19
## te(mintemp,lag).29 -0.50043558 -0.204517294  0.138618503 5.11    30
## te(mintemp,lag).30 -0.07460849  0.438732812  0.666628459 4.12    17
## te(mintemp,lag).31 -0.07666520  0.157493379  0.318042474 2.09    20
## te(mintemp,lag).32 -0.19949213  0.046871776  0.312414964 3.30    25
## te(mintemp,lag).33 -0.52357991 -0.135400976  0.219829973 3.69    12
## te(mintemp,lag).34 -0.50174239 -0.286706003  0.156415426 5.03    23
## te(mintemp,lag).35 -1.07051863 -0.468931730 -0.089243082 3.74    18
```

```
## te(precip,lag).1   -0.09297213  0.029364090  0.083930181 1.29    12
## te(precip,lag).2   -0.17458355  0.005393439  0.125938480 4.43    20
## te(precip,lag).3   -0.22456621 -0.045136000  0.095017737 5.70    29
## te(precip,lag).4   -0.20281094 -0.070319436  0.099114734 4.24    18
## te(precip,lag).5   -0.24048625 -0.085100516  0.170127035 3.30    14
## te(precip,lag).6   -0.06956940  0.047319160  0.128891728 2.53    16
## te(precip,lag).7   -0.06108266  0.007047921  0.053901278 2.02    27
## te(precip,lag).8   -0.11856369 -0.007391333  0.116507144 5.13    21
## te(precip,lag).9   -0.12938353 -0.042900846  0.079600781 4.61    37
## te(precip,lag).10  -0.11263754 -0.044949955  0.076829423 3.22    36
## te(precip,lag).11  -0.24366544 -0.043730280  0.168865987 5.68    19
## te(precip,lag).12  -0.23671407 -0.044824830  0.086427572 1.80    12
## te(precip,lag).13  -0.13049094 -0.048632472  0.073675825 2.56    27
## te(precip,lag).14  -0.12501919 -0.026529225  0.126750398 3.16    26
## te(precip,lag).15  -0.11745298 -0.024405937  0.075223878 2.22    17
## te(precip,lag).16  -0.12922225 -0.037125975  0.058530463 1.77    12
## te(precip,lag).17  -0.22821528 -0.016336255  0.132690731 3.23    10
## te(precip,lag).18  -0.22628626 -0.050773669  0.110371512 2.53    17
## te(precip,lag).19  -0.08528298 -0.029732586  0.036909686 1.55    21
## te(precip,lag).20  -0.10904592  0.003606645  0.076829235 3.48    22
## te(precip,lag).21  -0.14106257  0.026245485  0.111829420 4.04    20
## te(precip,lag).22  -0.17253319  0.002371445  0.106502481 2.84    15
## te(precip,lag).23  -0.13426439  0.018906416  0.177757709 3.45    17
## te(precip,lag).24  -0.41753950 -0.223533085 -0.005985728 2.07    16
## te(precip,lag).25  -0.10815082 -0.035598148  0.015254324 1.17    19
## te(precip,lag).26  -0.06168471 -0.005722032  0.062577092 1.46    20
## te(precip,lag).27  -0.10000512  0.043537865  0.199831629 1.95    16
## te(precip,lag).28  -0.19332550  0.008147668  0.227090787 3.67    18
## te(precip,lag).29  -0.08915872 -0.003144400  0.139554122 3.25    23
## te(precip,lag).30  -0.08635054  0.005455415  0.129704530 3.61    26
##                         2.5%      50%    97.5% Rhat n.eff
## te(mintemp,lag)   2.4024266 3.376868 4.376163 1.38    85
## te(mintemp,lag)2  2.0120425 3.378736 4.577591 1.98   376
## te(mintemp,lag)3 -0.9132402 2.815344 4.325631 2.96   666
## te(precip,lag)    1.9644878 3.608315 4.656943 1.47    93
## te(precip,lag)2   2.1115783 3.565391 4.795401 2.31   242
## te(precip,lag)3   0.4052581 2.930770 4.524352 1.04   837
##              2.5%        50%     97.5% Rhat n.eff
## ar1    0.5578250  0.8595408 1.1682943 1.07    45
## ar2   -0.3578623  0.1292964 0.5937600 1.16    29
## ar3   -0.4123643 -0.1094742 0.1834124 1.08    45
## sigma  0.2513970  0.3512191 0.4640151 1.33    82
## Rhats above 1.05 found for 407 parameters
## *Diagnose further to investigate why the chains have not mixed
```

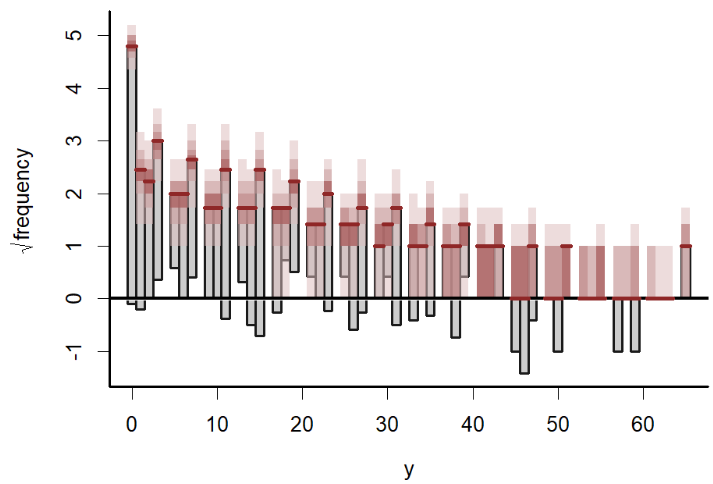
A pairs plot of the logged versions of the latent trend precision and the overdispersion parameter can give insight into whether there is still any strange behaviour or problematic parameter spaces that we must think more deeply about

```
plot(log(MCMCvis::MCMCchains(mod3$model_output, 'twdis')),
     log(MCMCvis::MCMCchains(mod3$model_output, 'tau')),
     ylab = 'log(tau)', xlab = 'log(twdis)', pch = 16, col = "#8F272740")
```



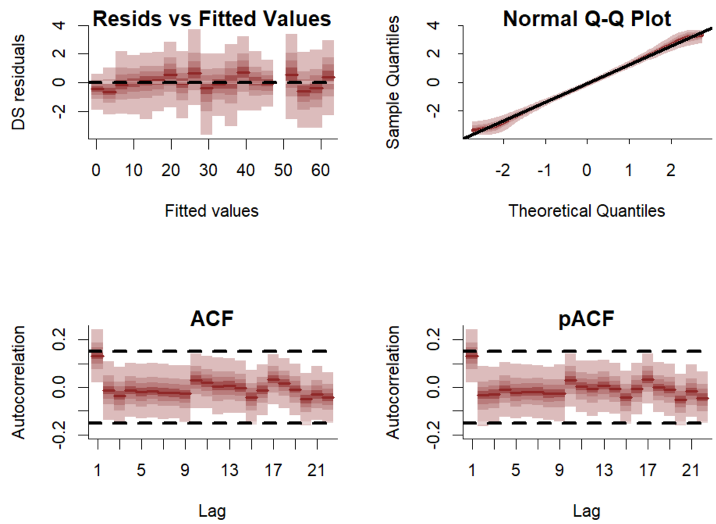
Our rootogram has not improved much with the addition of the latent trend

```
ppc(mod3, type = 'rootogram')
```



But the residual distributions look better

```
plot(mod3, type = 'residuals')
```



We can also demonstrate another feature of `mvgam`, which is the ability to use Hamiltonian Monte Carlo for parameter estimation via the software `Stan` (using the `rstan` or `cmdstanr` interfaces). Also note that currently there is no support for fitting `Tweedie` responses in `Stan`. We will therefore stick with a `Negative Binomial` observation process for the `Stan` version. However there are great advantages when using `Stan`, which includes the option to estimate smooth latent trends via Hilbert space approximate Gaussian Processes. This often makes sense for ecological series, which we expect to change smoothly over time. As expected when compared to the inferior Gibbs samplers in `JAGS`, the `Stan` version converges very nicely

```
mod4 <- mvgam(formula =  y ~ te(mintemp, lag, k = c(6, 6)) +
                 te(precip, lag, k = c(6, 6)),
                data = data_train,
                newdata = data_test,
                family = 'nb',
                chains = 4,
                trend_model = 'GP',
              use_stan = TRUE)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2 Iteration: 300 / 1000 [ 30%]  (Warmup)
```

```
## Chain 1 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1 finished in 26.5 seconds.
## Chain 2 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4 finished in 26.6 seconds.
## Chain 3 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2 finished in 28.6 seconds.
## Chain 3 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3 finished in 29.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 27.7 seconds.
## Total execution time: 29.3 seconds.
```

```r
summary(mod4)
```

```
## y ~ te(mintemp, lag, k = c(6, 6)) + te(precip, lag, k = c(6,
##     6))
## Negative Binomial
## log
## GP
## 1
## 168
## Fitted using Stan
##          2.5%      50%    97.5% Rhat n.eff
## r[1] 2.970416 4.356255 6.404514    1  1974
##                           2.5%          50%         97.5% Rhat n.eff
## (Intercept)         2.17709225  2.378635000  2.5798585000 1.00  2932
```

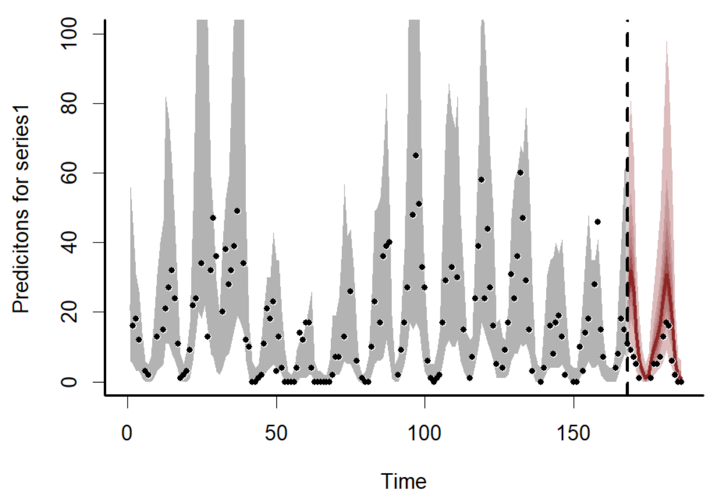
```
## te(mintemp,lag).1  -1.75897400 -0.591842000  0.3467717750 1.00   361
## te(mintemp,lag).2  -1.16355800 -0.297803000  0.4966987000 1.00   770
## te(mintemp,lag).3  -0.94254450 -0.227236000  0.5028276000 1.00  1233
## te(mintemp,lag).4  -1.13709300 -0.411032000  0.4075036000 1.01   513
## te(mintemp,lag).5  -1.76699075 -0.699035500  0.5642556250 1.01   490
## te(mintemp,lag).6  -1.08317950 -0.310882500  0.7519208000 1.01   317
## te(mintemp,lag).7  -0.65963015 -0.102975500  0.6157478000 1.02   294
## te(mintemp,lag).8  -0.40354732  0.189950000  0.8512601750 1.01   371
## te(mintemp,lag).9  -0.08797773  0.479991500  1.2585837500 1.00   343
## te(mintemp,lag).10  0.04498978  0.609563000  1.5567427500 1.01   257
## te(mintemp,lag).11 -0.21117485  0.743301500  1.9505895000 1.02   222
## te(mintemp,lag).12 -0.70052625 -0.059545050  0.8940880750 1.01   259
## te(mintemp,lag).13 -0.37750770 -0.015631350  0.5541217500 1.02   260
## te(mintemp,lag).14 -0.43865388  0.009277140  0.4233179000 1.01   425
## te(mintemp,lag).15 -0.32827285  0.115066500  0.5749798750 1.00   435
## te(mintemp,lag).16 -0.17206740  0.367995500  0.9914956500 1.02   226
## te(mintemp,lag).17 -0.32708990  0.562433500  1.4985940000 1.02   189
## te(mintemp,lag).18 -0.39303617  0.233741000  1.2354155000 1.01   275
## te(mintemp,lag).19 -0.18838400  0.220528500  0.8423709750 1.02   269
## te(mintemp,lag).20 -0.28408025  0.150833500  0.6237651000 1.01   427
## te(mintemp,lag).21 -0.36734553  0.112103000  0.6547499500 1.00   421
## te(mintemp,lag).22 -0.29774083  0.241148500  0.9465001750 1.02   258
## te(mintemp,lag).23 -0.42058440  0.455622000  1.4806845000 1.02   187
## te(mintemp,lag).24 -0.16396938  0.521780500  1.5416302500 1.01   290
## te(mintemp,lag).25 -0.06138123  0.333035000  0.9416676000 1.02   286
## te(mintemp,lag).26 -0.22019745  0.215250500  0.7101146750 1.01   428
## te(mintemp,lag).27 -0.43724927  0.034005100  0.6137922000 1.00   378
## te(mintemp,lag).28 -0.65693743 -0.122752500  0.6168214000 1.02   240
## te(mintemp,lag).29 -0.97922525 -0.099958350  0.9747536750 1.02   195
## te(mintemp,lag).30 -0.09982370  0.659532000  1.7756460000 1.01   303
## te(mintemp,lag).31 -0.10418115  0.339601500  0.9799016500 1.02   300
## te(mintemp,lag).32 -0.44717802  0.061006450  0.5440591750 1.01   541
## te(mintemp,lag).33 -0.69495515 -0.220066000  0.3034353000 1.00   514
## te(mintemp,lag).34 -1.02428625 -0.451858500  0.1844592500 1.01   287
## te(mintemp,lag).35 -1.70982325 -0.721514000  0.3091133500 1.02   239
## te(precip,lag).1   -0.16797343 -0.006423245  0.1405342750 1.00   998
## te(precip,lag).2   -0.24347042 -0.070998150  0.0928665450 1.00  1188
## te(precip,lag).3   -0.22970667 -0.068856100  0.0924677775 1.00  1341
## te(precip,lag).4   -0.21604287 -0.054726250  0.1093430750 1.00  1234
## te(precip,lag).5   -0.30620473 -0.047686000  0.1893136000 1.00  1201
## te(precip,lag).6   -0.08521782  0.057373150  0.2149875500 1.00  1046
## te(precip,lag).7   -0.13924068 -0.026830100  0.0905797325 1.00   884
## te(precip,lag).8   -0.19966258 -0.066997950  0.0547781775 1.00  1110
## te(precip,lag).9   -0.21362125 -0.077283950  0.0491141175 1.00  1172
## te(precip,lag).10  -0.20825428 -0.071111500  0.0592810450 1.00  1003
## te(precip,lag).11  -0.28716628 -0.067583350  0.1210358750 1.00   861
## te(precip,lag).12  -0.27132980 -0.075326500  0.0969955175 1.00  1361
## te(precip,lag).13  -0.20467015 -0.074957950  0.0500495875 1.00  1473
## te(precip,lag).14  -0.23420850 -0.089988700  0.0419540525 1.00  1502
## te(precip,lag).15  -0.25788950 -0.106843500  0.0282790475 1.00  1307
## te(precip,lag).16  -0.26424815 -0.106290000  0.0447183800 1.00   914
## te(precip,lag).17  -0.33729233 -0.094944400  0.1027900250 1.00   837
## te(precip,lag).18  -0.17473812  0.006447150  0.1935603500 1.00  1115
## te(precip,lag).19  -0.15138833 -0.039784450  0.0774041500 1.00  1041
```

```
## te(precip,lag).20  -0.17708745 -0.073350850  0.0368370575 1.00  1485
## te(precip,lag).21  -0.21430360 -0.104336000 -0.0008811142 1.00  1968
## te(precip,lag).22  -0.23798005 -0.115015500 -0.0057779943 1.00  1494
## te(precip,lag).23  -0.28291065 -0.082036250  0.1061037500 1.00  1105
## te(precip,lag).24  -0.36472762 -0.125658000  0.1108020000 1.00  1553
## te(precip,lag).25  -0.14093530 -0.024285450  0.0913640000 1.00  1826
## te(precip,lag).26  -0.08696450  0.040684700  0.1650882000 1.00  1266
## te(precip,lag).27  -0.13978840  0.115922000  0.3641700000 1.00  1277
## te(precip,lag).28  -0.31068465 -0.012538950  0.2486941750 1.00  2610
## te(precip,lag).29  -0.14036878  0.001755855  0.1447976250 1.00  2478
## te(precip,lag).30  -0.12672487  0.004267450  0.1354091750 1.00  2345
##                         2.5%       50%     97.5% Rhat n.eff
## te(mintemp,lag)   2.1752797 3.3798900 4.201017 1.00  1341
## te(mintemp,lag)2  1.7309555 3.3151550 4.243644 1.01   368
## te(mintemp,lag)3 -2.2150650 0.3249225 3.536089 1.00   295
## te(precip,lag)    1.7433150 3.3081400 4.202463 1.00  1186
## te(precip,lag)2   1.9602245 3.4377100 4.268304 1.00   502
## te(precip,lag)3   0.7809191 3.1214400 4.148958 1.00   935
##                   2.5%      50%     97.5% Rhat n.eff
## alpha_gp[1] 0.5015211 0.809107  1.285081    1  1604
## rho_gp[1]   1.7062995 4.844170 12.392967    1  2706
## n_eff / iter looks reasonable for all parameters
## Rhat looks reasonable for all parameters
## 0 of 2000 iterations ended with a divergence (0%)
## 0 of 2000 iterations saturated the maximum tree depth of 12 (0%)
## E-FMI indicated no pathological behavior
```

As with all other `mvgam` objects, we can create plots of the estimated forecast distribution

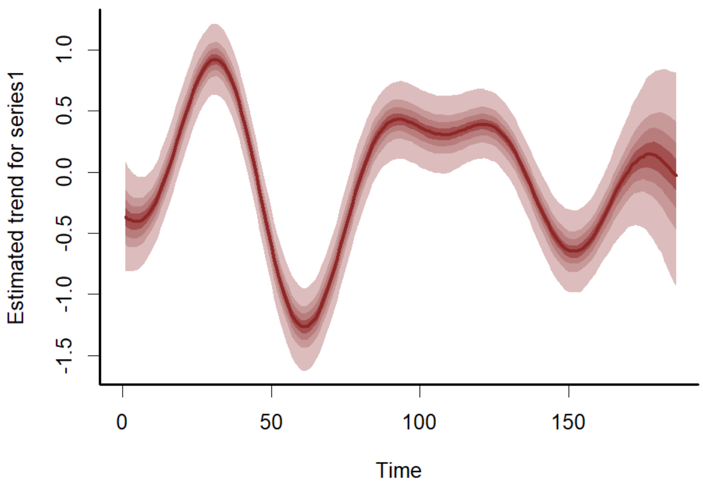
```r
plot_mvgam_fc(mod4, series = 1, newdata = data_test, ylim = c(0, 100))
```

```
## Out of sample DRPS:

## [1] 94.71623

##
```



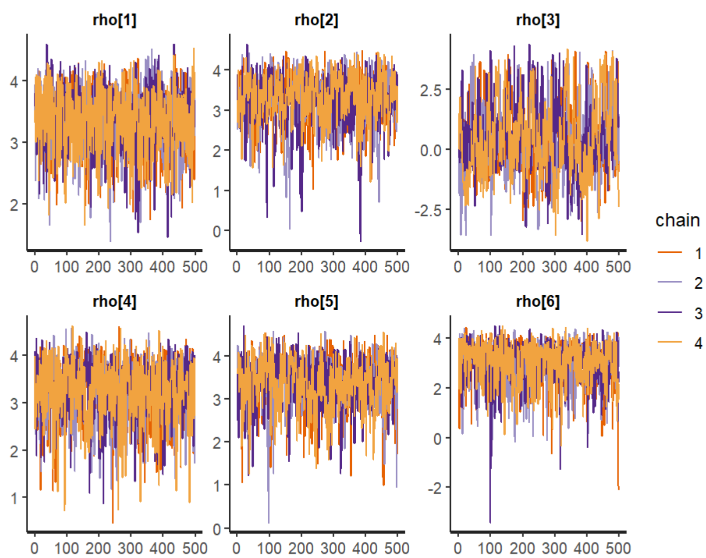
The trend now evolves smoothly via an infinite dimensional Gaussian Process

```
plot_mvgam_trend(mod4, series = 1)
```



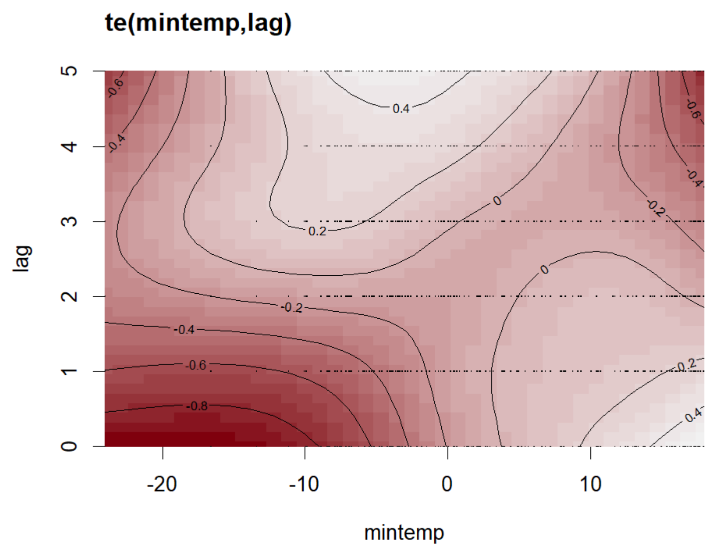
Traceplots of smooth penalties indicate good mixing and convergence of the four MCMC chains

```
rstan::stan_trace(mod4$model_output, pars = 'rho')
```

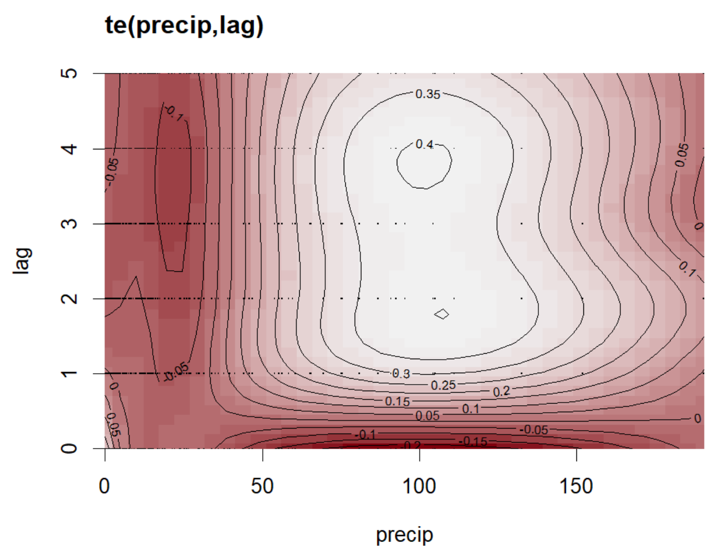


We can also create quick plots of the estimated smooth tensor product interactions for the distributed lag terms, which basically follow `mgcv`'s two-dimensional plotting utility but uses the `mvgam`'s estimated coefficients

```
plot_mvgam_smooth(mod4, series = 1, smooth = 1)
```




```
plot_mvgam_smooth(mod4, series = 1, smooth = 2)
```



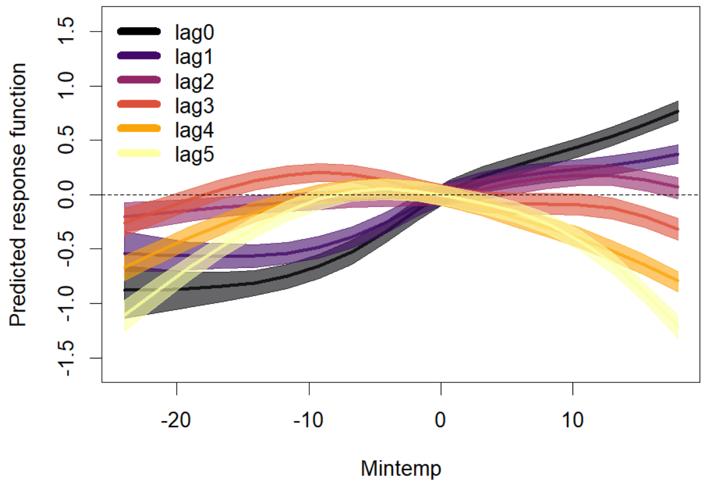
If you are like me then you'll find these plots rather difficult to interpret! The more intense yellow/white colours indicate higher predicted values, with the deeper red colours representing lower predicted values, but actually making sense of how the functional response is expected to change over different lags is not easy from these plots. HOwever, we can use the `predict_mvgam` function to generate much more interpretable plots. First we will focus on the effect of `mintemp` and generate a series of predictions to visualise how the estimated function changes over different lags. Set up prediction data by zeroing out all covariates apart from the covariate of interest

```
newdata <- data_test
newdata$year <- rep(0, length(newdata$year))
newdata$season <- rep(0, length(newdata$season))
newdata$precip <- matrix(0, ncol = ncol(newdata$precip),
                         nrow = nrow(newdata$precip))
```

Set up `viridis` plot colours and initiate the plot window to be centred around zero. We will then keep all `mintemp` values at zero apart from the particular lag being predicted so that we can visualise how the predicted function changes over lags of `mintemp`. Predictions are generated on the link scale in this case, though you could also use the response scale. Note that we need to first generate predictions with all covariates (including the `mintemp` covariate) zeroed out to find the 'baseline' prediction so that we can shift by this baseline for generating a zero-centred plot. That way our resulting plot will roughly follow the traditional `mgcv` partial effect plots

```
cols <- viridis::inferno(6)
plot(1, type = "n",
     xlab = 'Mintemp',
     ylab = 'Predicted response function',
     xlim = c(min(data_train$mintemp), max(data_train$mintemp)),
     ylim = c(-1.6, 1.6))

# Calculate predictions for when mintemp is all zeros to find the baseline
# value for centring the plot
newdata$mintemp <- matrix(0, ncol = ncol(newdata$mintemp),
                          nrow = nrow(newdata$mintemp))
preds <- predict(mod4, newdata = newdata, type = 'link')
offset <- mean(preds)

for(i in 1:6){
  # Set up prediction matrix for mintemp with lag i as the prediction sequence;
  # use a sequence of mintemp values across the full range of observed values in the training data
  newdata$mintemp <- matrix(0, ncol = ncol(newdata$precip),
                            nrow = nrow(newdata$precip))
  newdata$mintemp[,i] <- seq(min(data_train$mintemp),
                             max(data_train$mintemp),
                             length.out = length(newdata$year))

  # Predict on the link scale and shift by the offset so that values are roughly centred at zero
  preds <- predict(mod4, newdata = newdata, type = 'link') - offset

  # Calculate empirical prediction quantiles
  probs = c(0.05, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.95)
  cred <- sapply(1:NCOL(preds),
                 function(n) quantile(preds[,n],
                                      probs = probs))

  # Plot expected function posterior intervals (40-60%) and medians in varying colours per lag
  pred_upper <- cred[4,]
  pred_lower <- cred[6,]
  pred_vals <- seq(min(data_train$mintemp),
                   max(data_train$mintemp),
                   length.out = length(newdata$year))
  polygon(c(pred_vals, rev(pred_vals)), c(pred_upper, rev(pred_lower)),
          col = scales::alpha(cols[i], 0.6), border = scales::alpha(cols[i], 0.7))
  lines(pred_vals, cred[5,],
        col = scales::alpha(cols[i], 0.8), lwd = 2.5)
}
abline(h = 0, lty = 'dashed')
legend('topleft', legend = paste0('lag', seq(0, 5)),
       bg = 'white', bty = 'n',
```

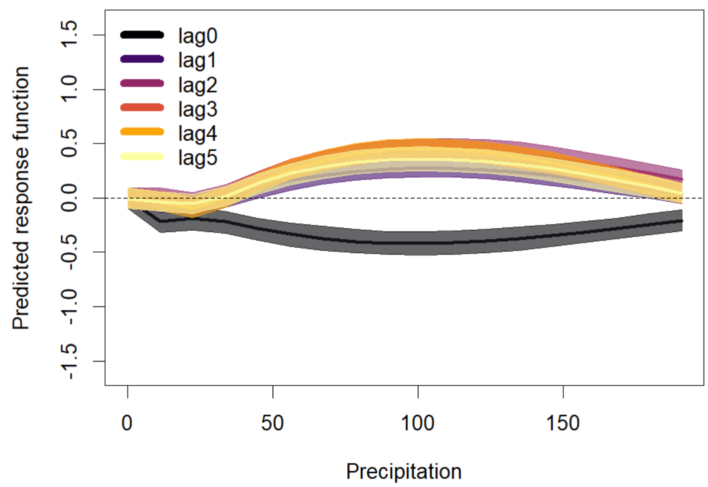
```
       col = cols, lty = 1, lwd = 6)
```



This plot demonstrates how the effect of `mintemp` is expected to change over different exposure lags, with the 3 - 5 month lags showing more of a cyclic seasonal pattern (catches expected to increase in the summer and autumn, roughly 3 - 5 months following cold minimum winter temperatures) while the recent lags (lags 0 and 1) demonstrate a more linear response function (catches broadly increasing as minimum temperature increases). This is hopefully a useful example for developing a better understanding of how a distributed lag model is attempting to recreate the data generating process. And here is the same plot for precipitation, which demonstrates how a u-shaped functional relationship diminishes toward a flat function at lags 2 - 5 (though this effect is clearly less important in the model than the mintemp * lag effect above)

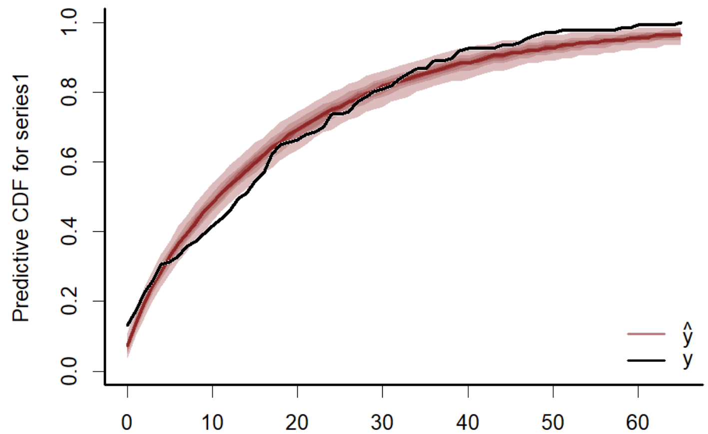
```
newdata <- data_test
newdata$year <- rep(0, length(newdata$year))
newdata$season <- rep(0, length(newdata$season))
newdata$mintemp <- matrix(0, ncol = ncol(newdata$mintemp),
                          nrow = nrow(newdata$mintemp))
newdata$precip <- matrix(0, ncol = ncol(newdata$precip),
                          nrow = nrow(newdata$precip))
preds <- predict(mod4, newdata = newdata, type = 'link')
offset <- mean(preds)
plot(1, type = "n",
     xlab = 'Precipitation',
     ylab = 'Predicted response function',
     xlim = c(min(data_train$precip), max(data_train$precip)),
     ylim = c(-1.6, 1.6))

for(i in 1:6){
  newdata$precip <- matrix(0, ncol = ncol(newdata$precip),
                           nrow = nrow(newdata$precip))
  newdata$precip[,i] <- seq(min(data_train$precip),
                            max(data_train$precip),
                            length.out = length(newdata$year))
  preds <- predict(mod4, newdata = newdata, type = 'link') - offset
  probs = c(0.05, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.95)
  cred <- sapply(1:NCOL(preds),
```

```
                 function(n) quantile(preds[,n],
                                     probs = probs))
  pred_upper <- cred[4,]
  pred_lower <- cred[6,]
  pred_vals <- seq(min(data_train$precip),
                   max(data_train$precip),
                   length.out = length(newdata$year))
  polygon(c(pred_vals, rev(pred_vals)), c(pred_upper, rev(pred_lower)),
          col = scales::alpha(cols[i], 0.6), border = scales::alpha(cols[i], 0.7))
  lines(pred_vals, cred[5,],
        col = scales::alpha(cols[i], 0.8), lwd = 2.5)
}
abline(h = 0, lty = 'dashed')
legend('topleft', legend = paste0('lag', seq(0, 5)),
       bg = 'white', bty = 'n',
       col = cols, lty = 1, lwd = 6)
```



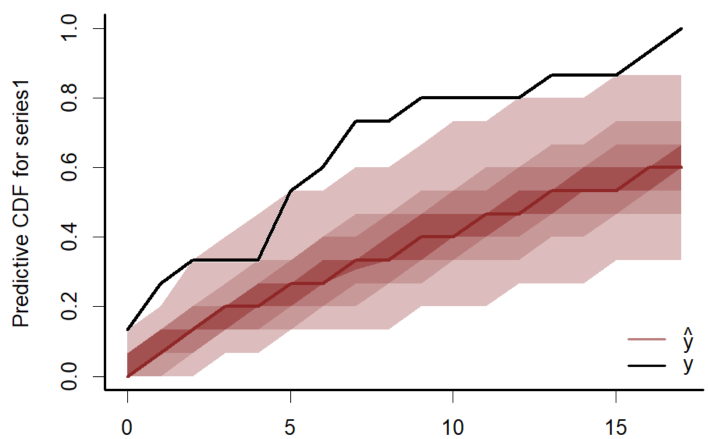
All of the usual functions in `mvgam` can also be used for list data objects and for models fitted with `Stan`, so long as they contain the necessary fields `series`, `season` and `year`. For example, posterior retrodictive checks for the in-sample training period:

```
ppc(mod4, series = 1, type = 'cdf')
```



and predictive checks for the out of sample forecast period (which demonstrates how the model tends to overpredict for the forecast period in this particular example):

```
ppc(mod4, newdata = data_test, series = 1, type = 'cdf')
```



Logical next steps for interrogating this model would be to trial different trend types (i.e. random walk), replace the distributed lag function for `precip` with a standard smooth function (that does not include lag interactions, as clearly the model above indicates that these are not supported) and inspect whether different covariates (such as `ndvi` or `maxtemp`) might play a role in modulating catches of `PP`. Finally, once we are satisfied that we have a well-performing model that we can understand and interrogate, we could expand up to a multivariate model by including other species as response variables. This would allow us to capture any possible unobserved dependencies in the catches of multiple co-occurring species in a single unified modelling framework

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