Introduction to mvgam

Here we introduce dynamic Generalised Additive Models and some of the key utility functions provided in `mvgam`. Briefly, assume $\tilde{\boldsymbol{y}}_t$ is the conditional expectation of a discrete response variable $\boldsymbol{y}$ at time $\boldsymbol{t}$. Asssuming $\boldsymbol{y}$ is drawn from an exponential distribution (such as Poisson or Negative Binomial) with a log link function, the linear predictor for a dynamic GAM is written as:

$$log(\tilde{\boldsymbol{y}}_t) = \boldsymbol{B}_0 + \sum_{i=1}^{I} \boldsymbol{s}_{i,t}\boldsymbol{x}_{i,t} + \boldsymbol{z}_t \,,$$

Here $\boldsymbol{B}_0$ is the unknown intercept, the $\boldsymbol{s}$'s are unknown smooth functions of the covariates ($\boldsymbol{x}$'s) and $\boldsymbol{z}$ is a dynamic latent trend component. Each smooth function $\boldsymbol{s}_i$ is composed of spline like basis expansions whose coefficients, which must be estimated, control the shape of the functional relationship between $\boldsymbol{x}_i$ and $log(\tilde{\boldsymbol{y}})$. The size of the basis expansion limits the smooth's potential complexity, with a larger set of basis functions allowing greater flexibility. Several advantages of GAMs are that they can model a diversity of response families, including discrete distributions (i.e. Poisson or Negative Binomial) that accommodate ecological features such as zero-inflation, and that they can be formulated to include hierarchical smoothing for multivariate responses. For the dynamic component, in its most basic form we assume a random walk with drift:

$$\boldsymbol{z}_t = \phi + \boldsymbol{z}_{t-1} + \boldsymbol{e}_t \,,$$

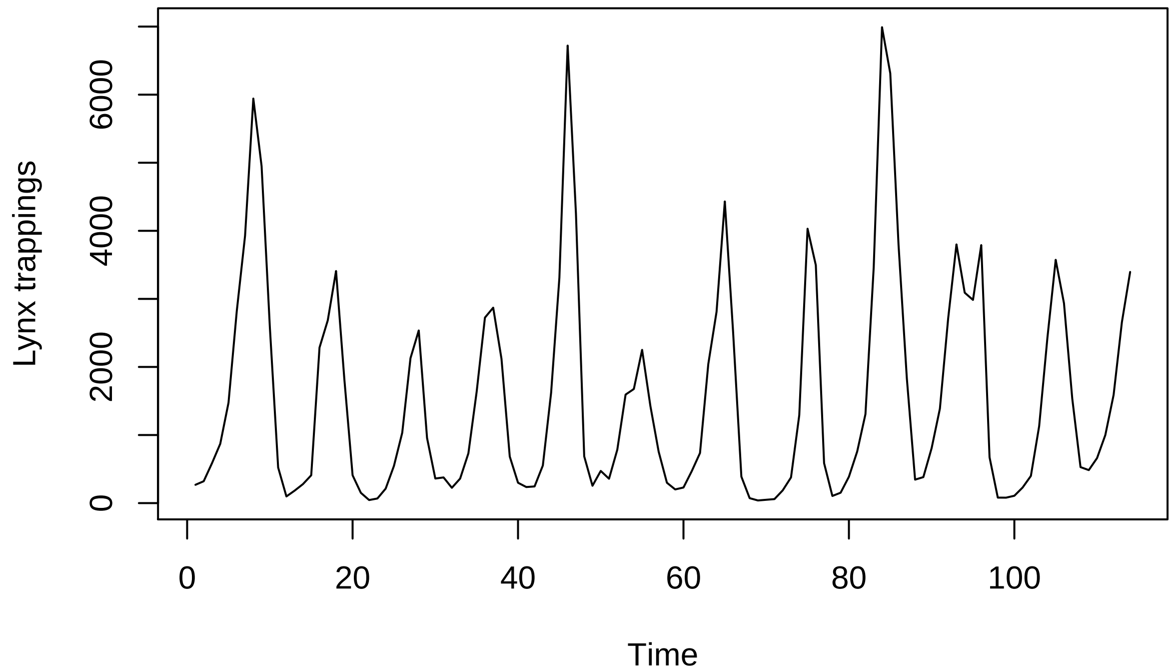
where ϕ is the estimated drift parameter and $\boldsymbol{e}$ is drawn from a zero-centred Gaussian distribution. This model is easily modified to include autoregressive terms, which `mvgam` accomodates up to `order = 3`.

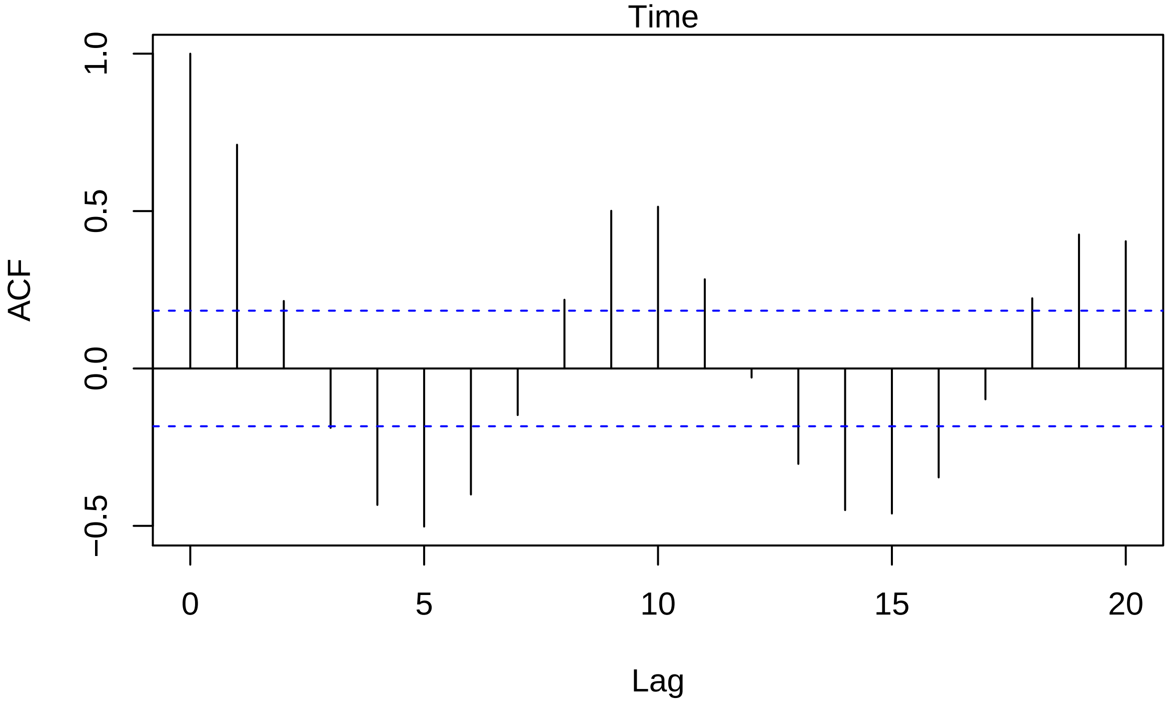
In this vignette we demonstrate the model and some of the key functions used to interrogate `mvgam` objects. First a univariate example to show how challenging it can be to forecast ahead with conventional GAMs and how `mvgam` overcomes these challenges. We begin by replicating the lynx analysis from 2018 Ecological Society of America workshop on GAMs that was hosted by Eric Pedersen, David L. Miller, Gavin Simpson, and Noam Ross, with some minor adjustments. First, load the data and plot the series as well as its estimated autocorrelation function

```
library(mvgam)
```

```
## Warning: package 'mgcv' was built under R version 3.6.2
```

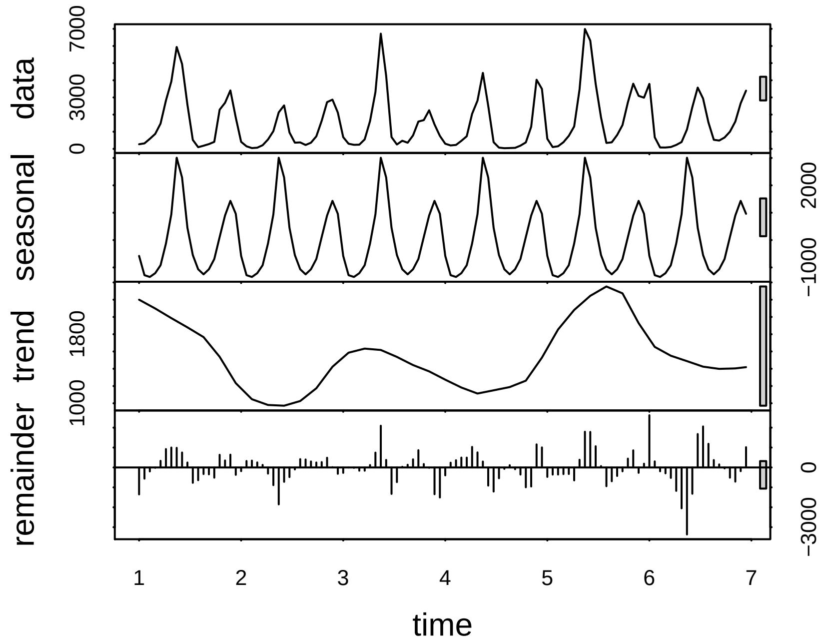
```
data(lynx)
lynx_full = data.frame(year = 1821:1934,
    population = as.numeric(lynx))
plot(lynx_full$population, type = "l", ylab = "Lynx trappings",
    xlab = "Time")
acf(lynx_full$population, main = "")
```





There is a clear ~19-year cyclic pattern to the data, so I create a `season` term that can be used to model this effect and give a better representation of the data generating process. I also create a new `year` term that represents which long-term cycle each observation is in

```
plot(stl(ts(lynx_full$population, frequency = 19),
    s.window = "periodic"))
```




```
lynx_full$season <- (lynx_full$year%%19) +
    1
cycle_ends <- c(which(lynx_full$season ==
    19), NROW(lynx_full))
cycle_starts <- c(1, cycle_ends[1:length(which(lynx_full$season ==
    19))] + 1)
cycle <- vector(length = NROW(lynx_full))
for (i in 1:length(cycle_starts)) {
    cycle[cycle_starts[i]:cycle_ends[i]] <- i
}
lynx_full$year <- cycle
```

Add lag indicators needed to fit the nonlinear lag models that gave the best one step ahead point forecasts in the ESA workshop example. As in the example, we specify the `default` argument in the `lag` function as the mean log population.

```
mean_pop_l = mean(log(lynx_full$population))
lynx_full = dplyr::mutate(lynx_full, popl = log(population),
    lag1 = dplyr::lag(popl, 1, default = mean_pop_l),
    lag2 = dplyr::lag(popl, 2, default = mean_pop_l),
    lag3 = dplyr::lag(popl, 3, default = mean_pop_l),
    lag4 = dplyr::lag(popl, 4, default = mean_pop_l),
    lag5 = dplyr::lag(popl, 5, default = mean_pop_l),
    lag6 = dplyr::lag(popl, 6, default = mean_pop_l))
```

```
## Warning: replacing previous import 'vctrs::data_frame' by 'tibble::data_frame'
## when loading 'dplyr'
```

For `mvgam` models, the response needs to be labelled `y` and we also need an indicator of the series name as a `factor` variable

```
lynx_full$y <- lynx_full$population
lynx_full$series <- factor("series1")
```

Split the data into training (first 40 years) and testing (next 10 years of data) to evaluate multi-step ahead forecasts

```
lynx_train = lynx_full[1:40, ]
lynx_test = lynx_full[41:50, ]
```

The best-forecasting model in the course was with nonlinear smooths of lags 1 and 2; we use those here is that we also include a cyclic smooth for the 19-year cycles as this seems like an important feature, as well as a yearly smooth for the long-term trend. Following advice from Gavin Simpson's blog post about the unpredictable and tricky behaviour's of smooths when extrapolating, we fit a cubic B spline for the trend with a mix of penalties and we extend the penalty to cover the years that we wish to predict, both of which are done to try and reign in any wacky extrapolation behaviours. This will hopefully give us better uncertainty estimates for the forecast

```
lynx_mgcv = gam(population ~ s(season, bs = "cc",
    k = 19) + s(year, bs = "bs", m = c(3,
    2, 1, 0)) + s(lag1, k = 5) + s(lag2,
    k = 5), knots = list(season = c(0.5,
    19.5), year = c(min(lynx_train$year) -
    1, min(lynx_train$year), max(lynx_train$year),
    max(lynx_test$year))), data = lynx_train,
    family = "poisson", method = "REML")
```

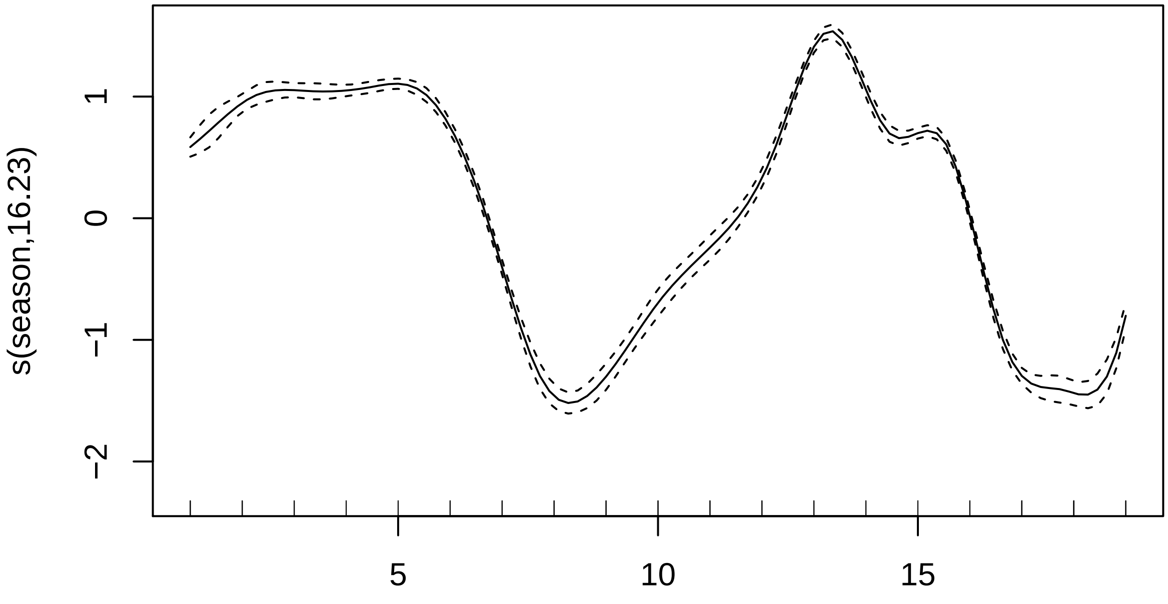
```
## Warning in smooth.construct.bs.smooth.spec(object, dk$data, dk$knots): there is
## *no* information about some basis coefficients

## Warning in smooth.construct.bs.smooth.spec(object, dk$data, dk$knots): basis
## dimension is larger than number of unique covariates
```

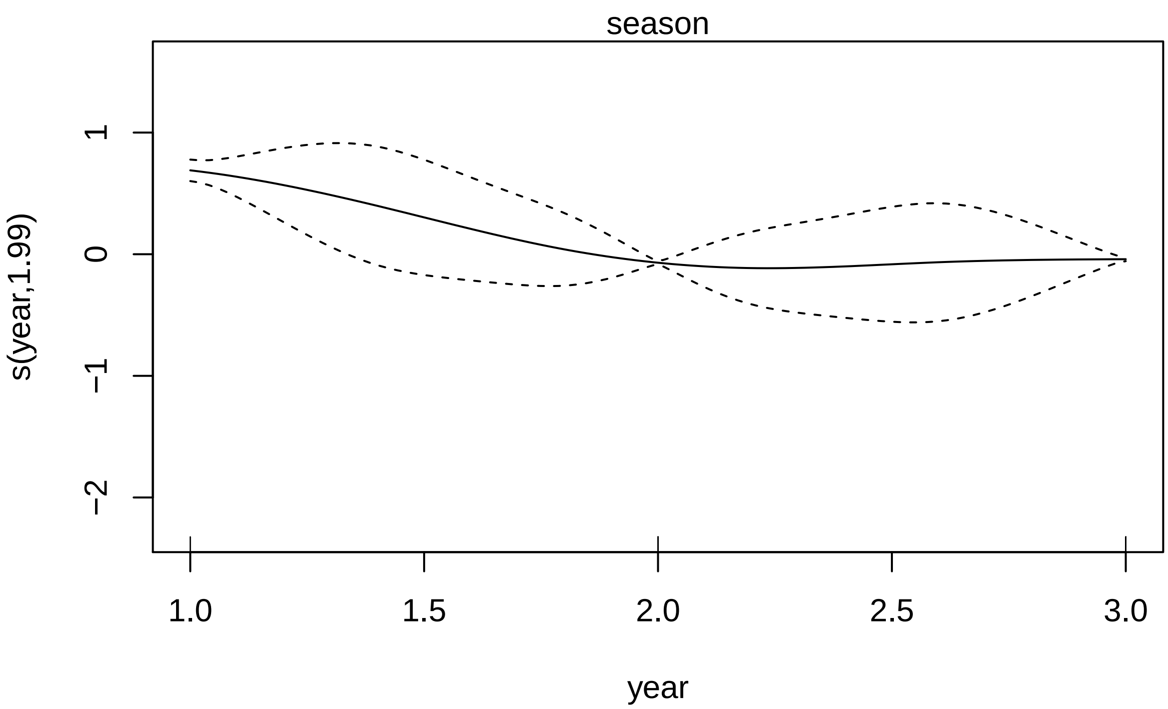
Inspect the model's summary and estimated smooth functions for the season, year and lag terms

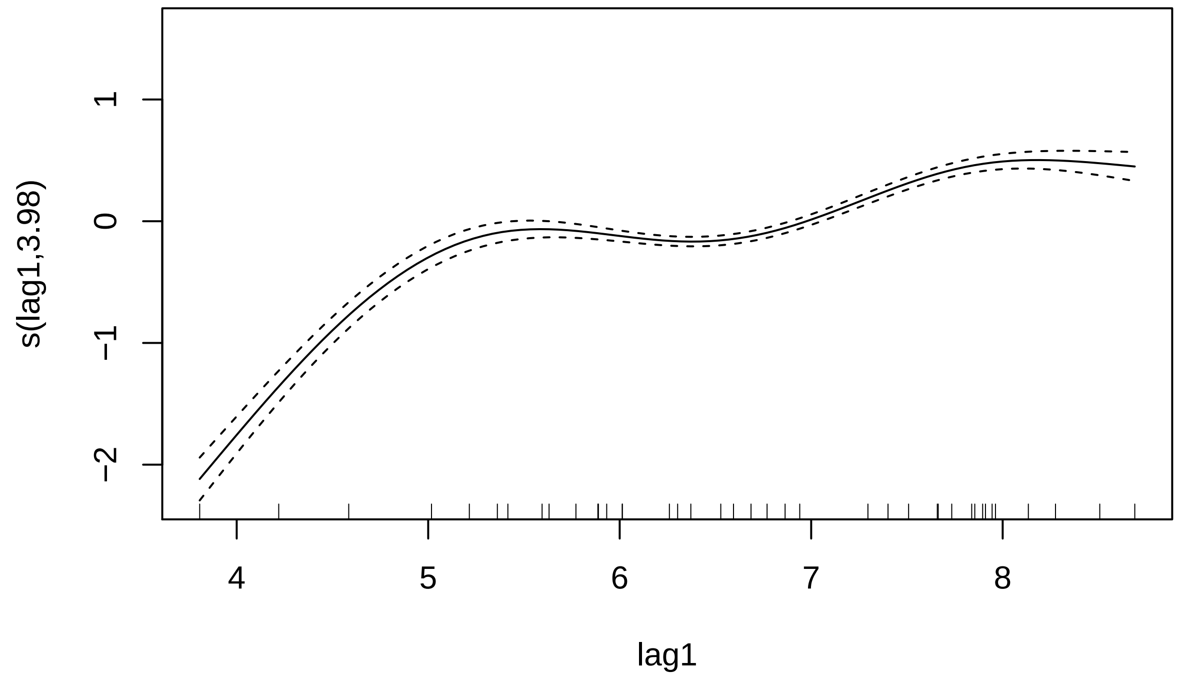
```
summary(lynx_mgcv)
```

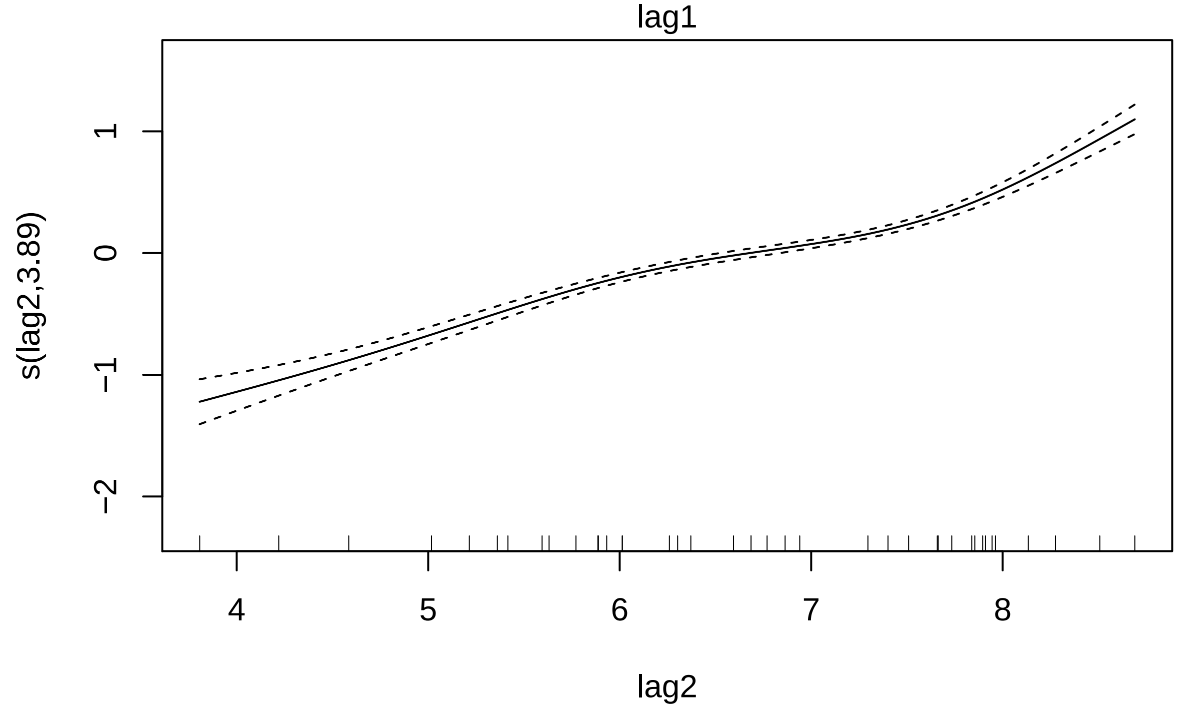
```
##
## Family: poisson
## Link function: log
##
## Formula:
## population ~ s(season, bs = "cc", k = 19) + s(year, bs = "bs",
##     m = c(3, 2, 1, 0)) + s(lag1, k = 5) + s(lag2, k = 5)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 6.666629   0.007511   887.6   <2e-16 ***
## ---
## 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) 16.229 17.000 6769.8  <2e-16 ***
## s(year)    1.991  2.000  244.1  <2e-16 ***
## s(lag1)    3.984  3.999  712.2  <2e-16 ***
## s(lag2)    3.892  3.993  488.2  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.967   Deviance explained = 97.7%
## -REML = 880.31  Scale est. = 1         n = 40
```

```
plot(lynx_mgcv, select = 1)
plot(lynx_mgcv, select = 2)
plot(lynx_mgcv, select = 3)
plot(lynx_mgcv, select = 4)
```



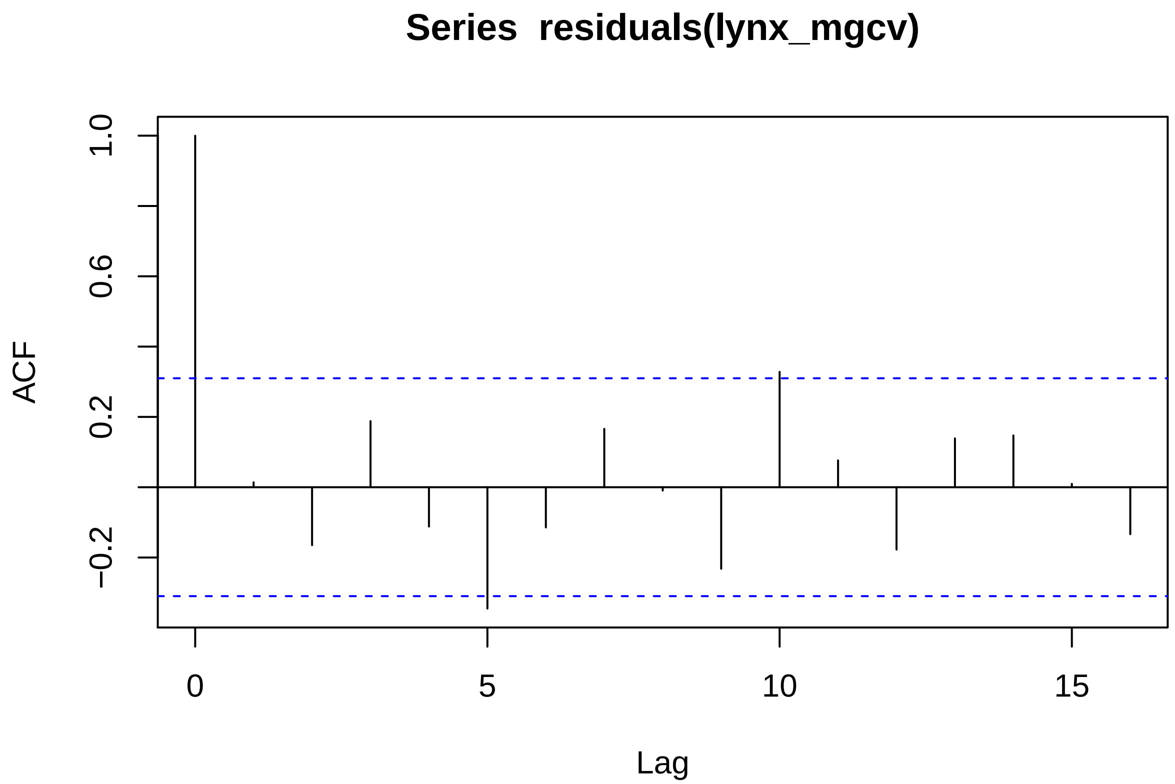






This model captures most of the deviance in the series and the functions are all confidently estimated to be non-zero and non-flat. So far, so good, but what about the residuals? Have we captured autocorrelation, suggesting we can make appropriate inferences?

```
acf(residuals(lynx_mgcv))
```



No autocorrelation left, which is encouraging. Now for some forecasts for the out of sample period. First we must take posterior draws of smooth beta coefficients to incorporate the uncertainties around smooth functions when simulating forecast paths

```
coef_sim <- gam.mh(lynx_mgcv)$bs
```

Next we define a function to perform forecast simulations from the nonlinear lag model in a recursive fashion. Using starting values for the last two lags, the function will iteratively project the path ahead with a random sample from the model's coefficient posterior

```
recurse_nonlin = function(model, lagged_vals,
    h) {
    # Initiate state vector
    states <- rep(NA, length = h + 2)
    # Last two values of the conditional
    # expectations begin the state vector
    states[1] <- as.numeric(exp(lagged_vals[2]))
    states[2] <- as.numeric(exp(lagged_vals[1]))
    # Get a random sample of the smooth
    # coefficient uncertainty matrix to use
    # for the entire forecast horizon of this
    # particular path
    gam_coef_index <- sample(seq(1, NROW(coef_sim)),
        1, T)
    # For each following timestep,
    # recursively predict based on the
    # predictions at each previous lag
    for (t in 3:(h + 2)) {
        # Build the GAM linear predictor matrix
        # using the two previous lags of the
```

```
        # (log) density
        newdata <- data.frame(lag1 = log(states[t -
            1] + 0.01), lag2 = log(states[t -
            2] + 0.01), season = lynx_test$season[t -
            2], year = lynx_test$year[t -
            2])
        colnames(newdata) <- c("lag1", "lag2",
            "season", "year")
        Xp <- predict(model, newdata = newdata,
            type = "lpmatrix")
        # Calculate the posterior prediction for
        # this timepoint
        mu <- rpois(1, lambda = exp(Xp %*%
            coef_sim[gam_coef_index, ]))
        # Fill in the state vector and iterate to
        # the next timepoint
        states[t] <- mu
    }
    # Return the forecast path
    states[-c(1:2)]
}
```

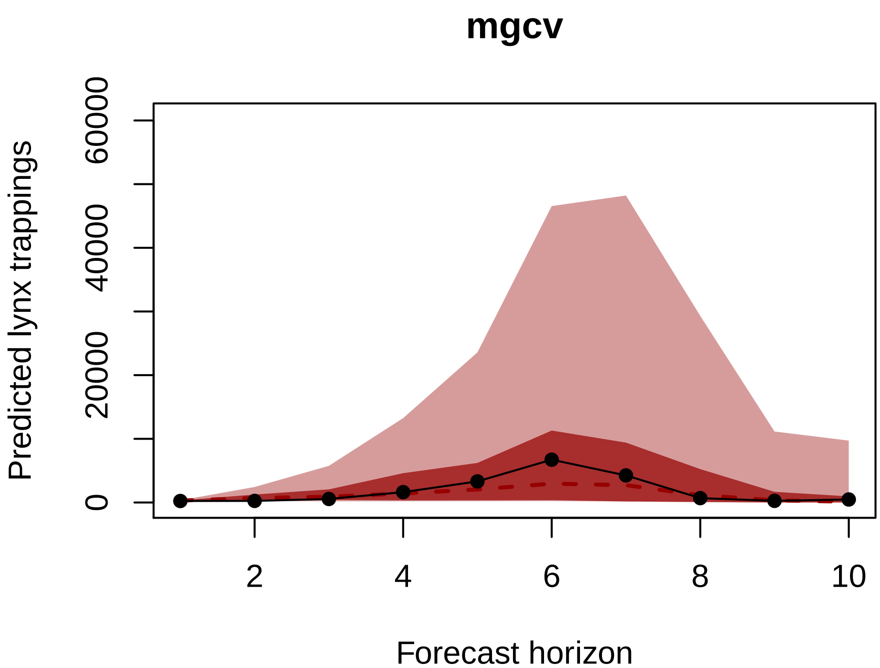
Create the GAM's forecast distribution by generating `1000` simulated forecast paths. Each path is fed the true observed values for the last two lags of the first out of sample timepoint, but they can deviate when simulating ahead depending on their particular draw of possible coefficients. Note, this is a bit slow and could easily be parallelised to speed up computations

```
gam_sims <- matrix(NA, nrow = 1000, ncol = 10)
for (i in 1:1000) {
    gam_sims[i, ] <- recurse_nonlin(lynx_mgcv,
        lagged_vals = c(lynx_test$lag1[1],
            lynx_test$lag2[1]), h = 10)
}
```

Plot the mgcv model's out of sample forecast for the next 10 years ahead

```
cred_ints <- apply(gam_sims, 2, function(x) hpd(x,
    0.95))
yupper <- max(cred_ints) * 1.25
plot(cred_ints[3, ] ~ seq(1:NCOL(cred_ints)),
    type = "l", col = rgb(1, 0, 0, alpha = 0),
    ylim = c(0, yupper), ylab = "Predicted lynx trappings",
    xlab = "Forecast horizon", main = "mgcv")
polygon(c(seq(1:(NCOL(cred_ints))), rev(seq(1:NCOL(cred_ints)))),
    c(cred_ints[1, ], rev(cred_ints[3, ])),
    col = rgb(150, 0, 0, max = 255, alpha = 100),
    border = NA)
cred_ints <- apply(gam_sims, 2, function(x) hpd(x,
    0.68))
polygon(c(seq(1:(NCOL(cred_ints))), rev(seq(1:NCOL(cred_ints)))),
    c(cred_ints[1, ], rev(cred_ints[3, ])),
    col = rgb(150, 0, 0, max = 255, alpha = 180),
    border = NA)
lines(cred_ints[2, ], col = rgb(150, 0, 0,
    max = 255), lwd = 2, lty = "dashed")
```

```
points(lynx_test$population[1:10], pch = 16)
lines(lynx_test$population[1:10])
```

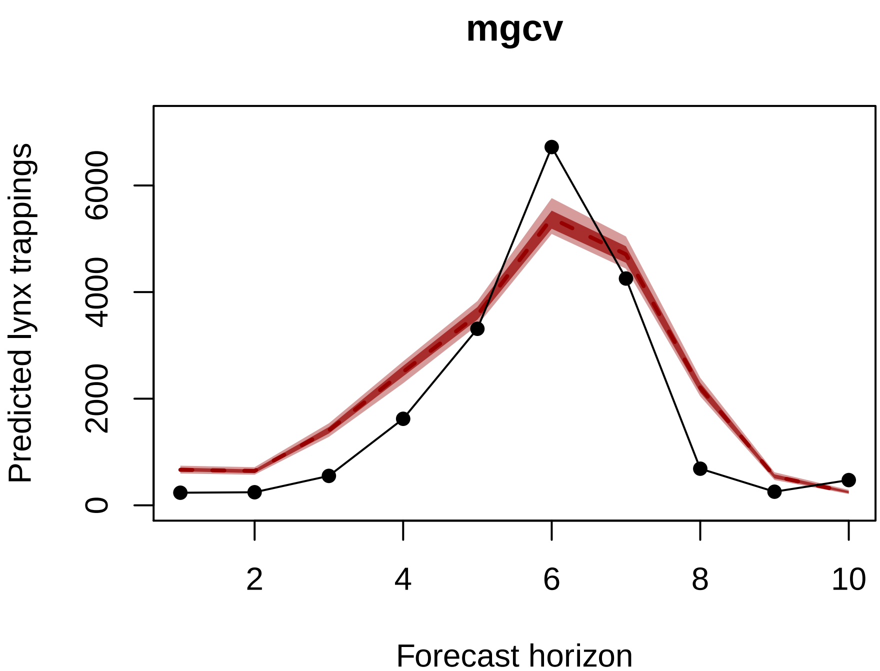


A decent forecast? The shape is certainly correct, but the 95% uncertainty intervals are far too wide (i.e. our upper interval extends to up to 8 times the maximum number of trappings that have ever been recorded up to this point). This is almost entirely due to the extrapolation behaviour of the B spline, as the lag smooth functions are not encountering values very far outside the ranges they've already been trained on so they are resorting mostly to interpolation. On that topic, interpolation is also a problem for the B spline in some situations. For example, if some of the observations in the testing data are in a year that was also included in training, these observations will be interpolated from the yearly B spline smooth (often with overconfidence), while the uncertainty will quickly grow once we hit years that are outside the training intervals.

Ok so that is not a terrible start, but what if we remove the yearly trend and let the lag smooths capture more of the temporal dependencies? Will that improve the forecast distribution? Run a second model and plot the forecast

```
lynx_mgcv2 = gam(population ~ s(season, bs = "cc",
    k = 19) + s(lag1, k = 5) + s(lag2, k = 5),
    data = lynx_train, family = "poisson",
    method = "REML")
coef_sim <- gam.mh(lynx_mgcv2)$bs
gam_sims <- matrix(NA, nrow = 1000, ncol = 10)
for (i in 1:1000) {
    gam_sims[i, ] <- recurse_nonlin(lynx_mgcv2,
        lagged_vals = c(lynx_test$lag1[1],
            lynx_test$lag2[1]), h = 10)
}
cred_ints <- apply(gam_sims, 2, function(x) hpd(x,
    0.95))
yupper <- max(cred_ints) * 1.25
plot(cred_ints[3, ] ~ seq(1:NCOL(cred_ints)),
    type = "l", col = rgb(1, 0, 0, alpha = 0),
    ylim = c(0, yupper), ylab = "Predicted lynx trappings",
```

```
    xlab = "Forecast horizon", main = "mgcv")
polygon(c(seq(1:(NCOL(cred_ints))), rev(seq(1:NCOL(cred_ints)))),
    c(cred_ints[1, ], rev(cred_ints[3, ])),
    col = rgb(150, 0, 0, max = 255, alpha = 100),
    border = NA)
cred_ints <- apply(gam_sims, 2, function(x) hpd(x,
    0.68))
polygon(c(seq(1:(NCOL(cred_ints))), rev(seq(1:NCOL(cred_ints)))),
    c(cred_ints[1, ], rev(cred_ints[3, ])),
    col = rgb(150, 0, 0, max = 255, alpha = 180),
    border = NA)
lines(cred_ints[2, ], col = rgb(150, 0, 0,
    max = 255), lwd = 2, lty = "dashed")
points(lynx_test$population[1:10], pch = 16)
lines(lynx_test$population[1:10])
```



This forecast is highly overconfident, with very unrealistic uncertainty intervals due to the interpolation behaviours of the lag smooths. You can certainly keep trying different formulations (our experience is that the B spline variant above produces the best probabilstic forecasts from any tested `mgcv` model, but we did not test an exhaustive set), but hopefully it is clear that forecasting using splines is tricky business and it is likely that each time you do it you'll end up honing in on different combinations of penalties, knot selections etc.... Now we will fit an `mvgam` model for comparison. This model fits a similar model to the `mgcv` model directly above but with a full time series model for the errors (in this case an `AR1` process), rather than smoothing splines that do not incorporate a concept of the future. We do not use a `year` term to reduce any possible extrapolation and because the latent dynamic component should capture this temporal variation. We estimate the model in `JAGS` using MCMC sampling (Note that `JAGS` 4.3.0 is required; installation links are found here)

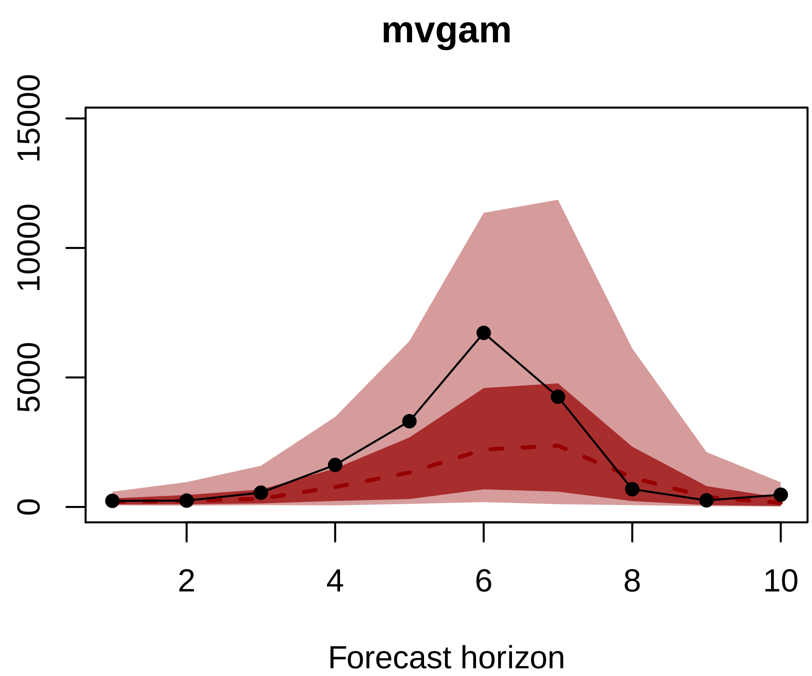
```
lynx_mvgam <- mvjagam(data_train = lynx_train,
    data_test = lynx_test, formula = y ~
        s(season, bs = "cc", k = 19), knots = list(season = c(0.5,
        19.5)), family = "poisson", trend_model = "AR1",
    n.burnin = 25000, n.iter = 5000, thin = 5,
```

```
    auto_update = F)
```

```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 40
##    Unobserved stochastic nodes: 116
##    Total graph size: 1927
##
## Initializing model
```

Calculate the out of sample forecast from the fitted `mvgam` model and plot

```
fits <- MCMCvis::MCMCchains(lynx_mvgam$jags_output,
    "ypred")
fits <- fits[, (NROW(lynx_mvgam$obs_data) +
    1):(NROW(lynx_mvgam$obs_data) + 10)]
cred_ints <- apply(fits, 2, function(x) hpd(x,
    0.95))
yupper <- max(cred_ints) * 1.25
plot(cred_ints[3, ] ~ seq(1:NCOL(cred_ints)),
    type = "l", col = rgb(1, 0, 0, alpha = 0),
    ylim = c(0, yupper), ylab = "", xlab = "Forecast horizon",
    main = "mvgam")
polygon(c(seq(1:(NCOL(cred_ints))), rev(seq(1:NCOL(cred_ints)))),
    c(cred_ints[1, ], rev(cred_ints[3, ])),
    col = rgb(150, 0, 0, max = 255, alpha = 100),
    border = NA)
cred_ints <- apply(fits, 2, function(x) hpd(x,
    0.68))
polygon(c(seq(1:(NCOL(cred_ints))), rev(seq(1:NCOL(cred_ints)))),
    c(cred_ints[1, ], rev(cred_ints[3, ])),
    col = rgb(150, 0, 0, max = 255, alpha = 180),
    border = NA)
lines(cred_ints[2, ], col = rgb(150, 0, 0,
    max = 255), lwd = 2, lty = "dashed")
points(lynx_test$population[1:10], pch = 16)
lines(lynx_test$population[1:10])
```



The `mvgam` has much more realistic uncertainty than the `mgcv` versions above, with all out of sample observations falling within the model's 95% credible intervals. Of course this is just one out of sample comparison, and to really determine which model is most appropriate for forecasting we would want to run many of these tests using a rolling window approach. Have a look at this model's summary to see what is being estimated (note that longer MCMC runs would probably be needed to increase effective sample sizes)

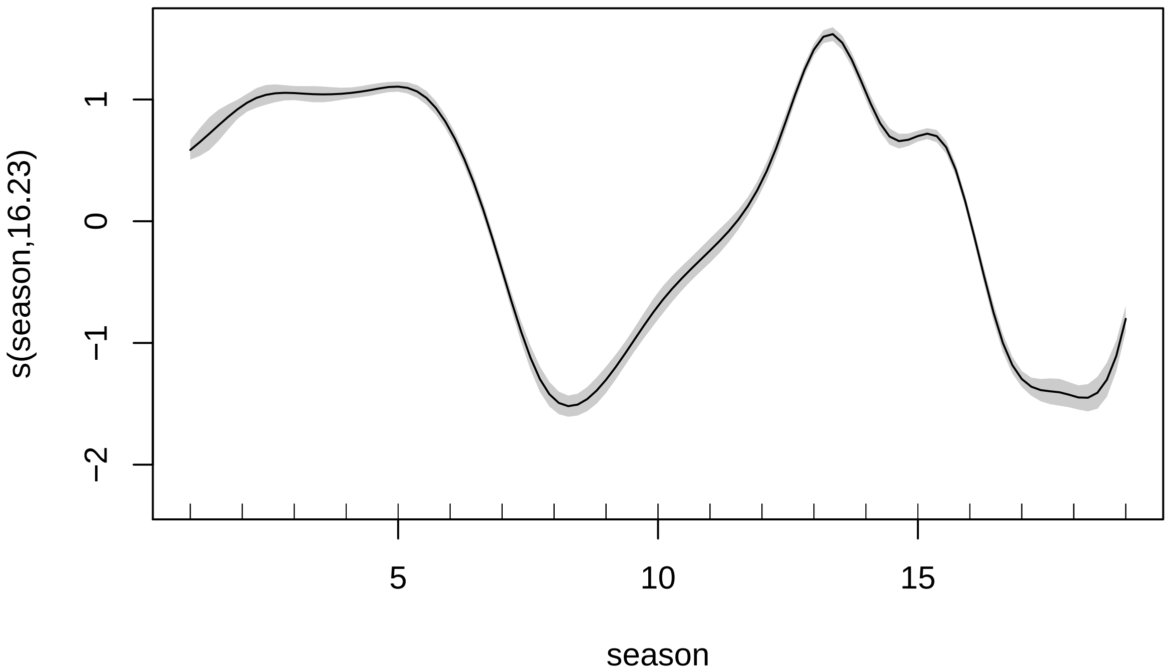
```
summary_mvgam(lynx_mvgam)
```

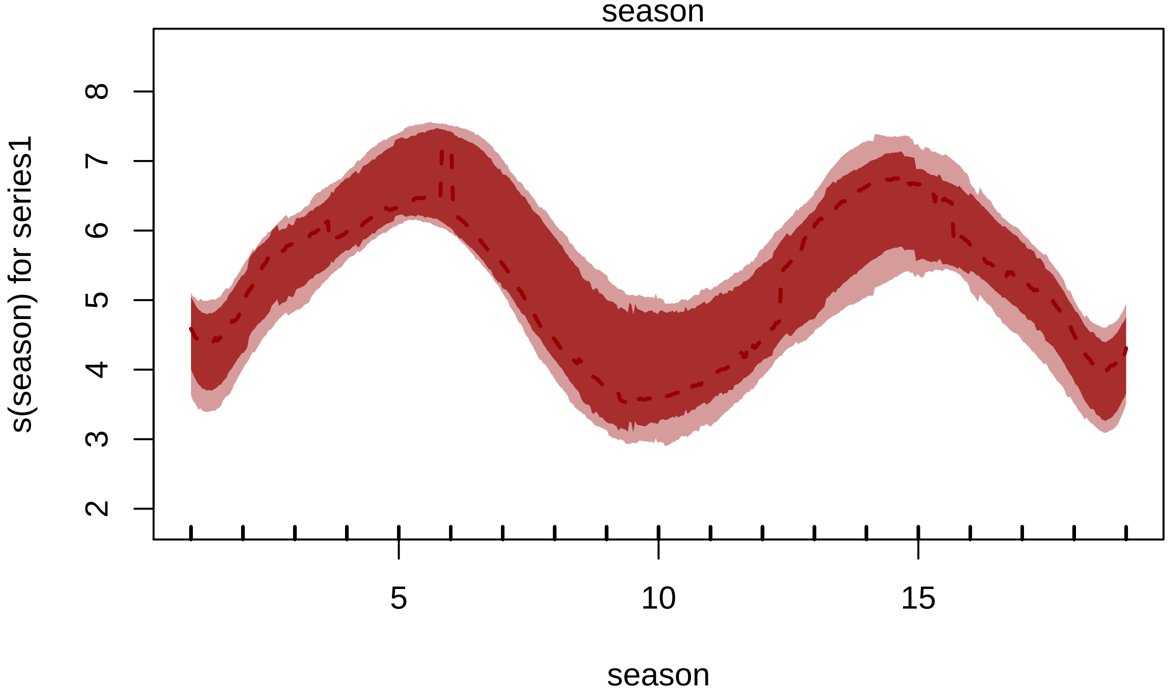
```
## GAM formula:

## y ~ s(season, bs = "cc", k = 19)

##

## Family:

## Poisson

##

## N series:

## 1

##

## N observations per series:

## 40

##

## GAM smooth term approximate significances:

##              edf Ref.df Chi.sq p-value
## s(season) 14.27  17.00  28508  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

##
```

```
## GAM coefficient (beta) estimates:
##                      2.5%         50%       97.5% Rhat n.eff
## (Intercept)   4.618697087  5.18568428  5.87246065 6.93    10
## s(season).1  -1.209923115 -0.70538392 -0.09453820 1.68    34
## s(season).2  -0.317124420  0.03656451  0.62520270 1.43    61
## s(season).3   0.355983654  0.79830859  1.27925814 1.35    27
## s(season).4   0.973851310  1.46083072  1.91637517 1.07    20
## s(season).5   1.306182418  1.79646368  2.34347954 1.37    12
## s(season).6   0.656280508  1.29676965  2.07247832 2.68    18
## s(season).7  -0.477628507  0.14423542  0.91741374 2.83    45
## s(season).8  -1.412337660 -0.79592700 -0.04076629 2.07    45
## s(season).9  -1.637126251 -1.02696185 -0.41602870 1.56    48
## s(season).10 -1.408790640 -0.71640464 -0.10000813 1.15    45
## s(season).11 -0.538147469  0.01476545  0.59598222 1.26    37
## s(season).12  0.028184773  1.05316677  1.69508886 1.39    16
## s(season).13  0.436748919  1.43940648  1.99543371 1.11     9
## s(season).14  0.671528164  1.23102432  1.79711485 1.01    17
## s(season).15  0.003161317  0.46107305  1.00996669 1.21    37
## s(season).16 -0.806352893 -0.39411181  0.22056654 1.36    97
## s(season).17 -1.444921882 -0.98137081 -0.40976065 1.49    54
##
## GAM smoothing parameter (rho) estimates:
##              2.5%      50%    97.5% Rhat n.eff
## s(season) 3.80493 4.766151 5.665331 1.04   197
##
## Latent trend drift and AR parameter estimates:
##           2.5%       50%     97.5% Rhat n.eff
## phi 0.08542556 0.3542258 0.7327097 1.57   377
## ar1 0.52187009 0.7225680 0.9098906 1.12  1092
## ar2 0.00000000 0.0000000 0.0000000  NaN     0
## ar3 0.00000000 0.0000000 0.0000000  NaN     0
##
```

Now inspect each model's estimated smooth for the 19-year cyclic pattern. Note that the `mvgam` smooth plot is on a different scale compared to the `mgcv` plot, but interpretation is similar. The `mgcv` smooth is much wigglier, likely because it is compensating for any remaining autocorrelation not captured by the lag smooths. We could probably remedy this by reducing `k` in the seasonal smooth for the `mgcv` model (in practice this works well, but leaving `k` larger for the `mvgam`'s seasonal smooth is recommended as our experience is that this tends to lead to better performance and convergence)

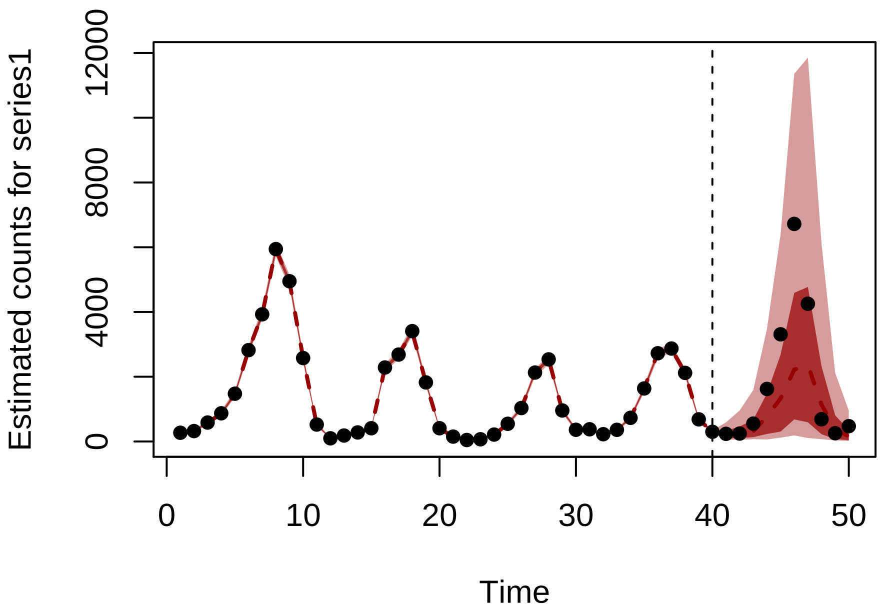
```
plot(lynx_mgcv, select = 1, shade = T)
plot_mvgam_smooth(lynx_mvgam, 1, "season")
```





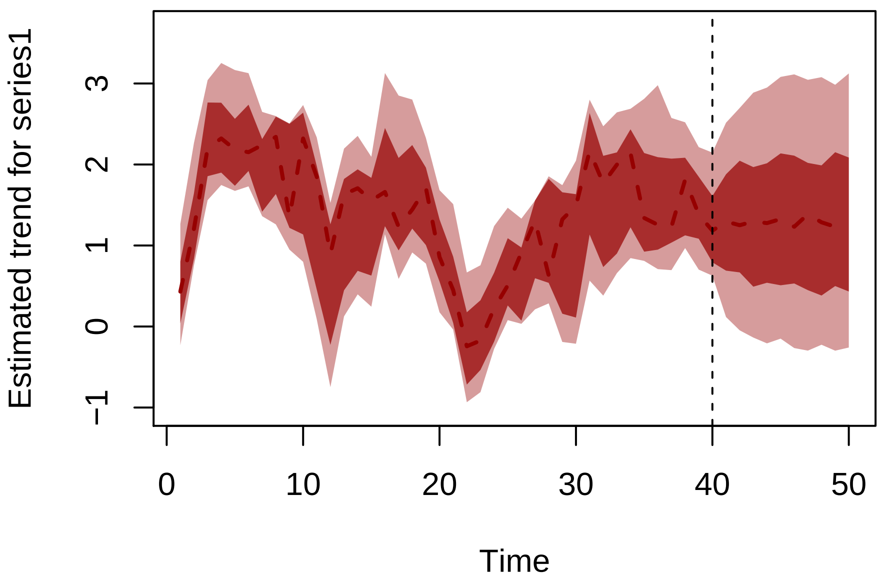
We can also view the mvgam's posterior predictions for the entire series (testing and training)

```
plot_mvgam_fc(lynx_mvgam, data_test = lynx_test)
```



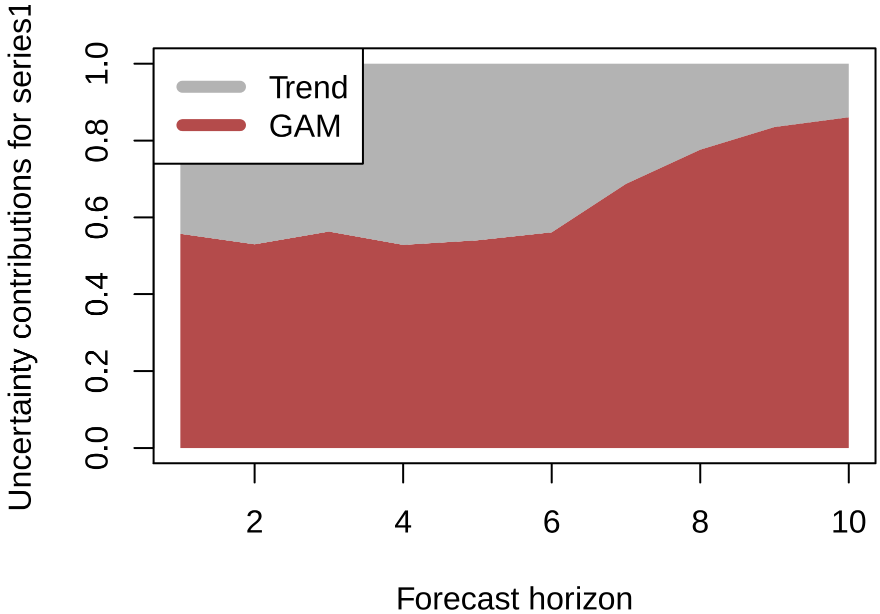
And the estimated trend

```
plot_mvgam_trend(lynx_mvgam, data_test = lynx_test)
```



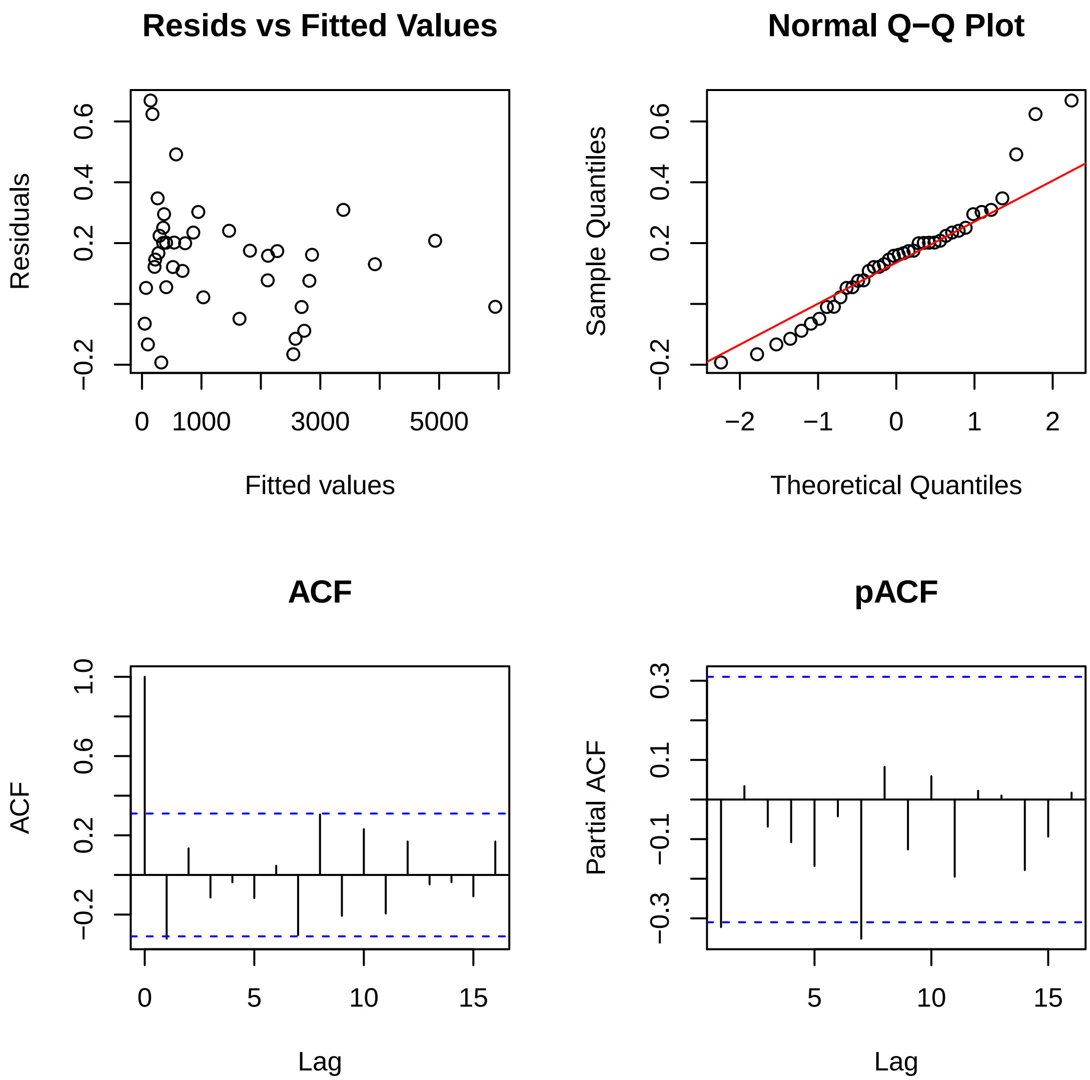
A key aspect of ecological forecasting is to understand how different components of a model contribute to forecast uncertainty. We can estimate contributions to forecast uncertainty for the GAM smooth functions and the latent trend using `mvgam`

```
plot_mvgam_uncertainty(lynx_mvgam, data_test = lynx_test)
```



Both components contribute to forecast uncertainty, suggesting we would still need some more work to learn about factors driving the dynamics of the system. But we will leave the model as-is for this example. Diagnostics of the model can also be performed using `mvgam`. Have a look at the model's residuals, which are posterior medians of Dunn-Smyth randomised quantile residuals so should follow approximate normality. We are primarily looking for a lack of autocorrelation, which would suggest our AR1 model is appropriate for the latent trend

```
plot_mvgam_resids(lynx_mvgam)
```



Another useful utility of **mvgam** is the ability to use rolling window forecasts to evaluate competing models that may represent different hypotheses about the series dynamics. Here we will fit a poorly specified model to showcase how this evaluation works

```
lynx_mvgam_poor <- mvjagam(data_train = lynx_train,
    data_test = lynx_test, formula = y ~
        s(season) + s(year, bs = "bs", m = c(3,
            2, 1, 0)), family = "poisson",
    trend_model = "None", n.burnin = 1000,
    n.iter = 1000, thin = 1, auto_update = F)
```

```
## Warning in smooth.construct.bs.smooth.spec(object, dk$data, dk$knots): basis
## dimension is larger than number of unique covariates

## Warning in smooth.construct.bs.smooth.spec(object, dk$data, dk$knots): basis
## dimension is larger than number of unique covariates

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

```
##    Observed stochastic nodes: 40
##    Unobserved stochastic nodes: 119
##    Total graph size: 2063
##
## Initializing model
```

We choose a set of timepoints within the training data to forecast from, allowing us to simulate a situation where the model's parameters had already been estimated but we have only observed data up to the evaluation timepoint and would like to generate forecasts from the latent trends. Here we use year 10 as our last observation and forecast ahead for the next 10 years.

```
mod1_eval <- eval_mvgam(lynx_mvgam, eval_timepoint = 10,
    fc_horizon = 10)
mod2_eval <- eval_mvgam(lynx_mvgam_poor,
    eval_timepoint = 10, fc_horizon = 10)
```

Summary statistics of the two models' out of sample Discrete Rank Probability Score (DRPS) indicate that the well-specified model performs markedly better (far lower DRPS)

```
summary(mod1_eval$series1$drps)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.256  19.256  56.428  77.648 135.648 197.436
```

```
summary(mod2_eval$series1$drps)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   111.6   302.3   570.8   542.6   784.3   925.1
```

Nominal coverages for both models' 90% prediction intervals

```
mean(mod1_eval$series1$in_interval)
```

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

```
mean(mod2_eval$series1$in_interval)
```

```
## [1] 0.6
```

The `compare_mvgams` function automates this process by rolling along a set of timepoints for each model, ensuring a more in-depth evaluation of each competing model at the same set of timepoints