Ecological forecasting in R

Lecture 4: evaluating dynamic models

Nicholas Clark

School of Veterinary Science, University of Queensland

0900–1200 CET Wednesday 29th May, 2024

Workflow

Press the "o" key on your keyboard to navigate among slides

Access the tutorial html here

- Download the data objects and exercise script from the html file
- Complete exercises and use Slack to ask questions

Relevant open-source materials include:

- Evaluating distributional forecasts
- Approximate leave-future-out cross-validation for Bayesian time series models
- The Marginal Effects Zoo (0.14.0)

This lecture's topics

Forecasting from dynamic models

Bayesian posterior predictive checks

Point-based forecast evaluation

Probabilistic forecast evaluation

Forecasting from dynamic models

Forecasting in `mvgam`

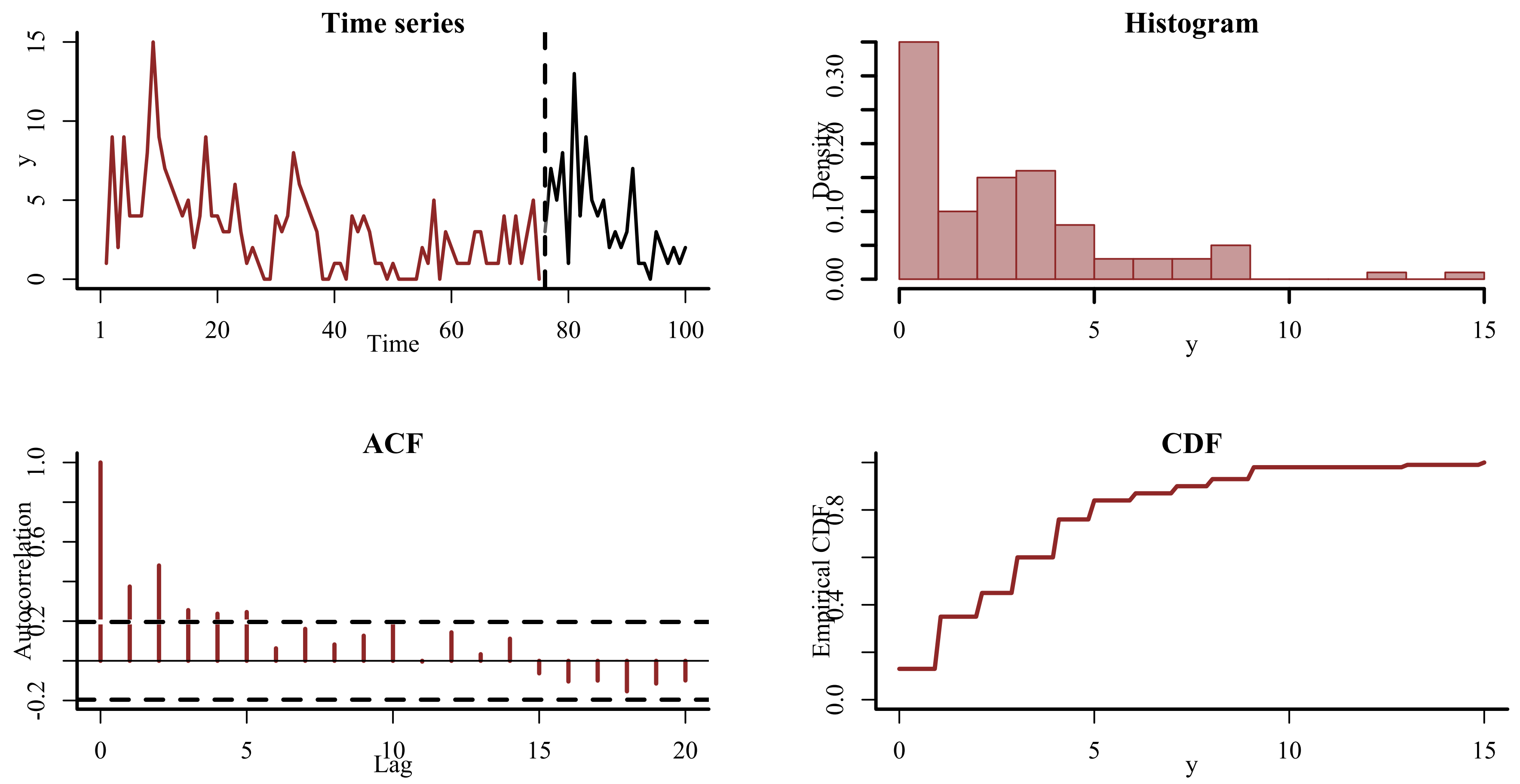
Two options

- Feed `newdata` into the `mvgam()` function for automatic probabilistic forecasts through `Stan`
- Produce forecasts outside of `Stan` by feeding `newdata` and the fitted model into the `forecast()` function

Both require any out-of-sample covariates to be supplied

Both should give equivalent results

Simulated data



The model

```r
library(mvgam)
model <- mvgam(y ~
                 s(season, bs = 'cc', k = 8),
                data = data_train,
                newdata = data_test,
                trend_model = GP(),
                family = poisson())
```

A cyclic smooth of `season` to capture repeated periodic variation

The model

```r
library(mvgam)
model ← mvgam(y ~
                 s(season, bs = 'cc', k = 8),
                data = data_train,
                newdata = data_test,
                trend_model = GP(),
                family = poisson())
```

A Gaussian Process trend (approximated with Hilbert basis functions)

The model

```r
library(mvgam)
model <- mvgam(y ~
                 s(season, bs = 'cc', k = 8),
                data = data_train,
                newdata = data_test,
                trend_model = GP(),
                family = poisson())
```

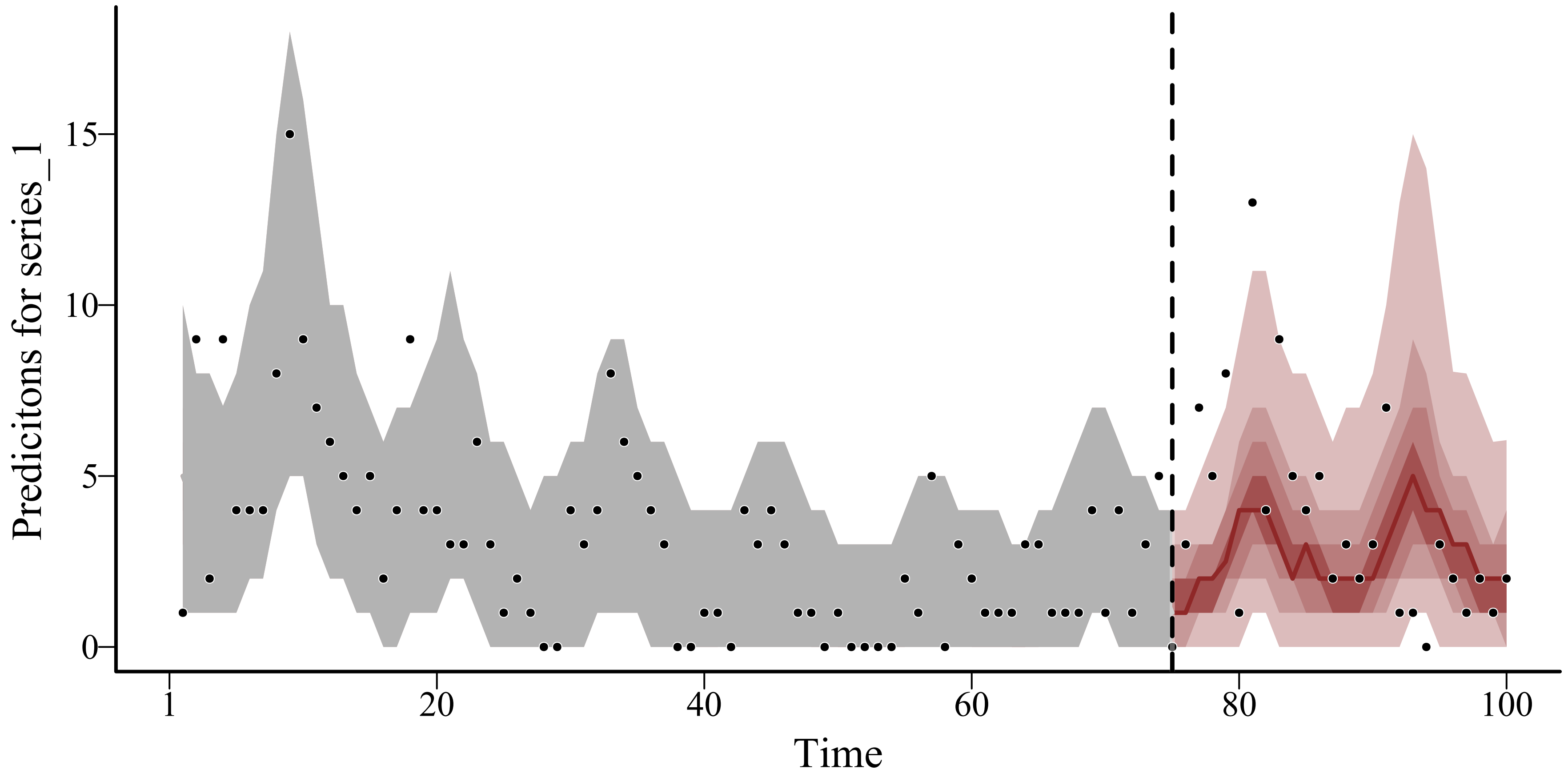
Forecasts will be computed automatically using the `generated quantities` block in `Stan`

Dropping `newdata`

```r
model2 <- mvgam(y ~
                 s(season, bs = 'cc', k = 8),
                data = data_train,
                trend_model = GP(),
                family = poisson())
```

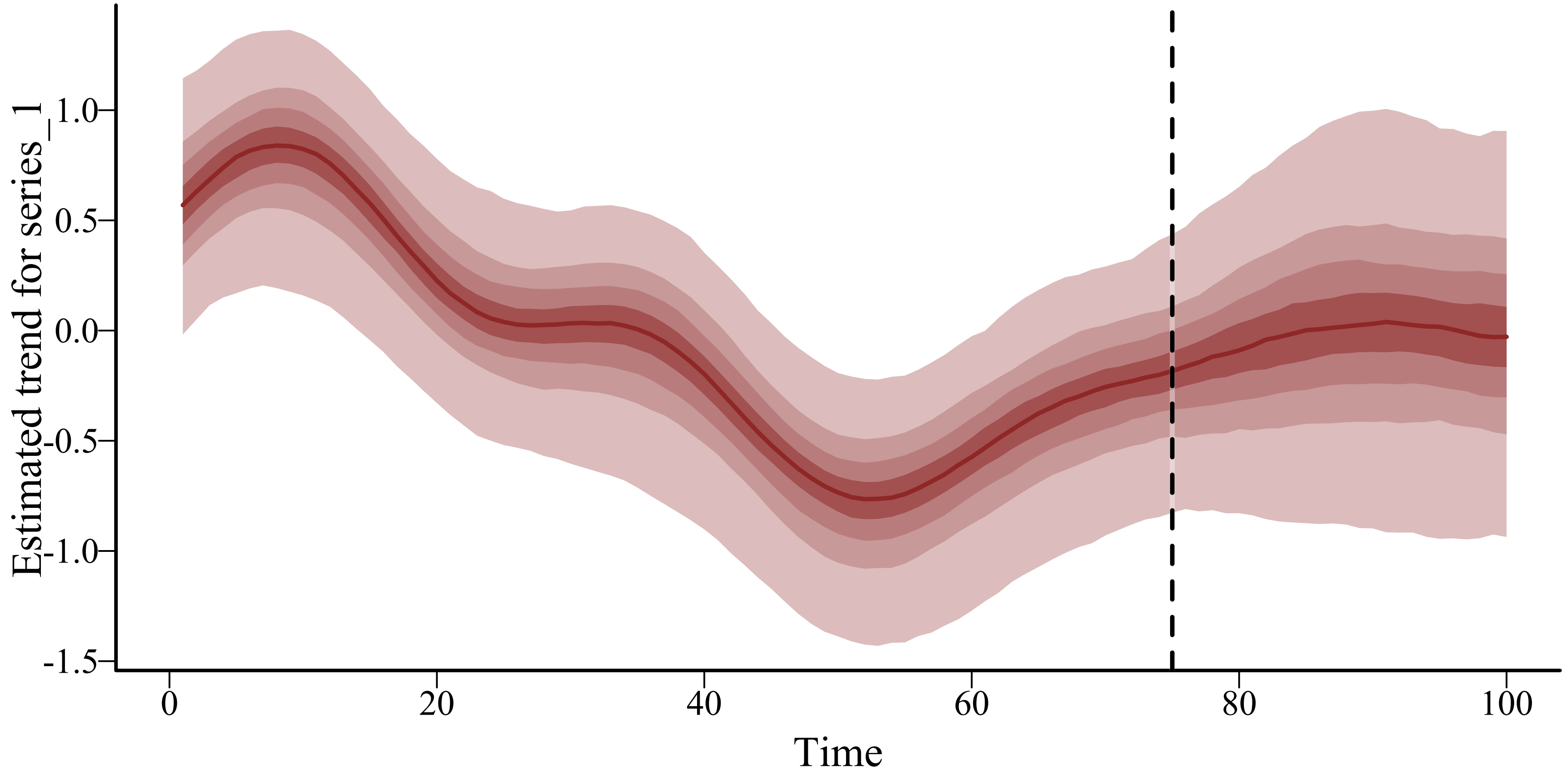
Predictions will only be calculated for the training data if no testing data (i.e. `newdata`) are supplied

```
plot(model, type = 'forecast')
```



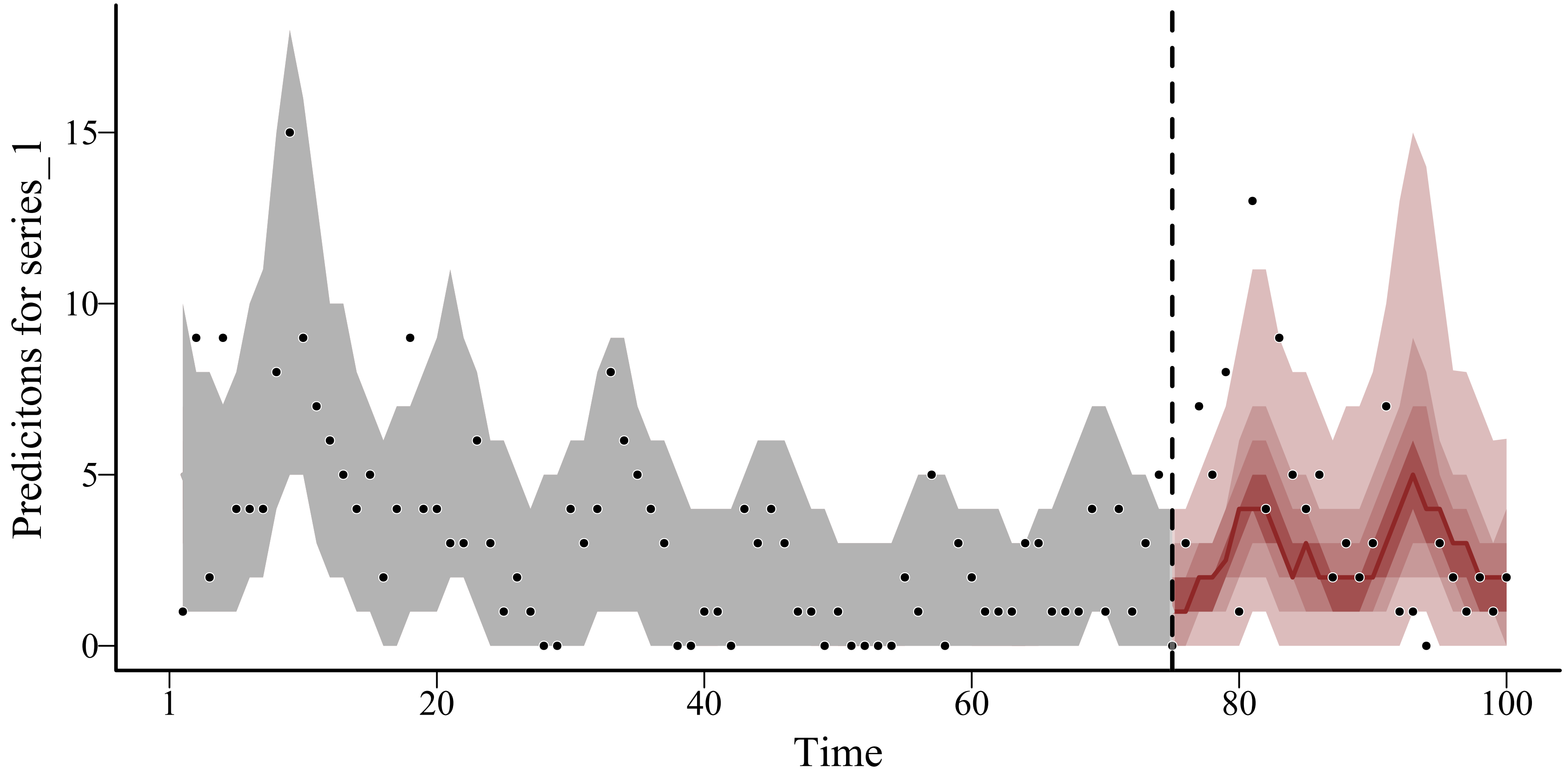
Automatic forecasts because `newdata` were supplied

```
plot(model, type = 'trend', newdata = data_test)
```



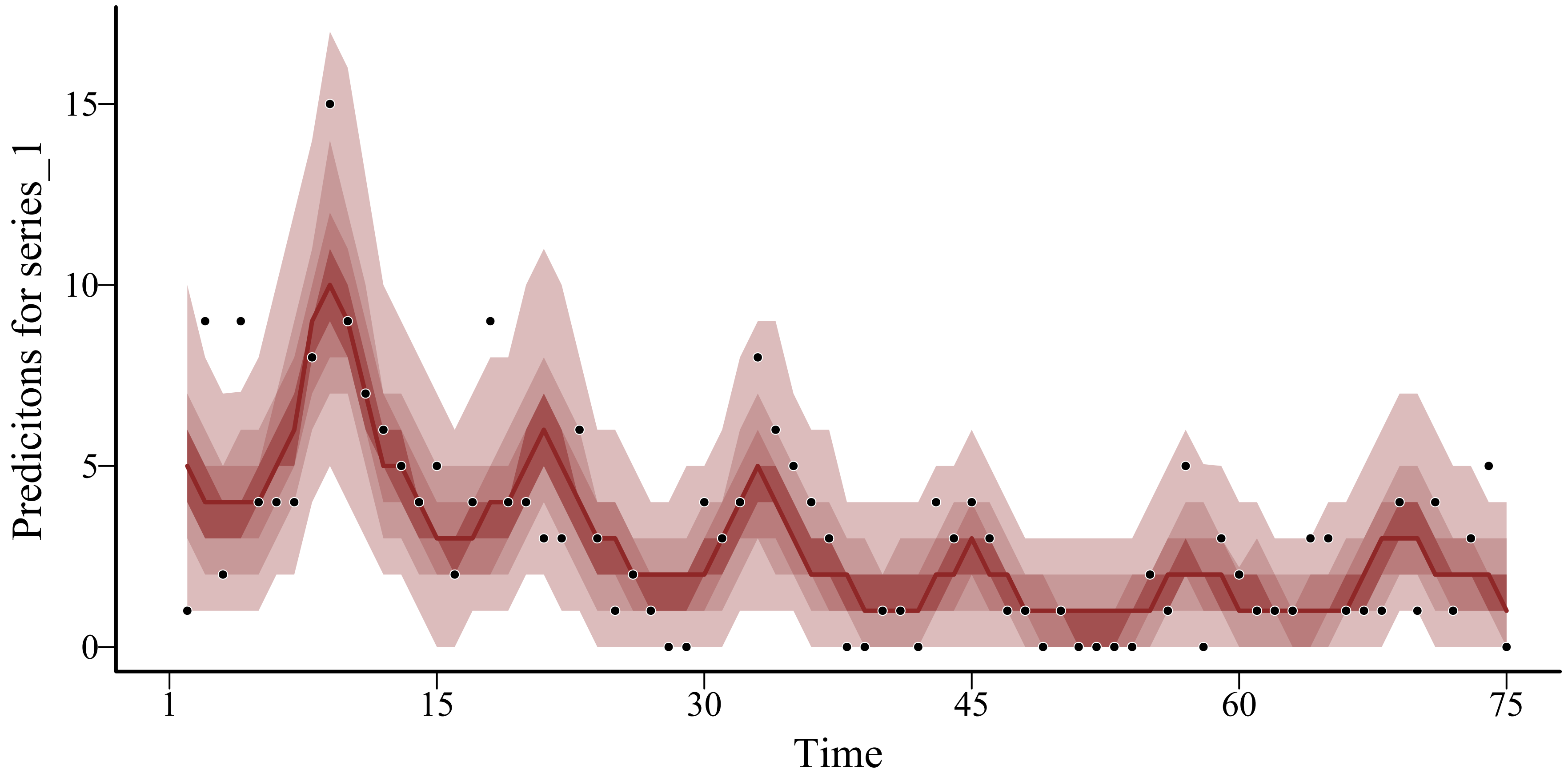
Trend extends into the future

```r
plot(model, type = 'forecast', newdata = data_test)
```



Forecasts can be compared to truths quickly

```r
plot(model2, type = 'forecast')
```



No forecasts in this case. Now what?

Posterior draws

dynamic `mvgam` models contain draws for many quantities

- β coefficients for linear predictor terms (called `b`)
- Any family-specific shape / scale parameters (i.e. ϕ for Negative Binomial; σ_{obs} for Normal / LogNormal etc…)
- Any trend-specific parameters (i.e. α and ρ for GP trends; σ and $ar1$ for AR trends etc…)
- In-sample posterior predictions (called `ypred`)
- In-sample posterior trend estimates (called `trend`)

All stored as MCMC draws in an object of class `stanfit` in the `model_output` slot

The `stanfit` object

```r
summary(model2$model_output)
```

```
##   Length    Class     Mode 
##        1  stanfit       S4
```

```r
model2$model_output@model_pars
```

```
##  [1] "rho"      "b"        "ypred"    "mus"      "lambda"   "trend"   
##  [7] "alpha_gp" "rho_gp"   "b_gp"     "lp__"
```

```r
model2$model_output@sim$chains
```

```
## [1] 4
```

```r
model2$model_output@sim$iter
```

```
## [1] 1000
```

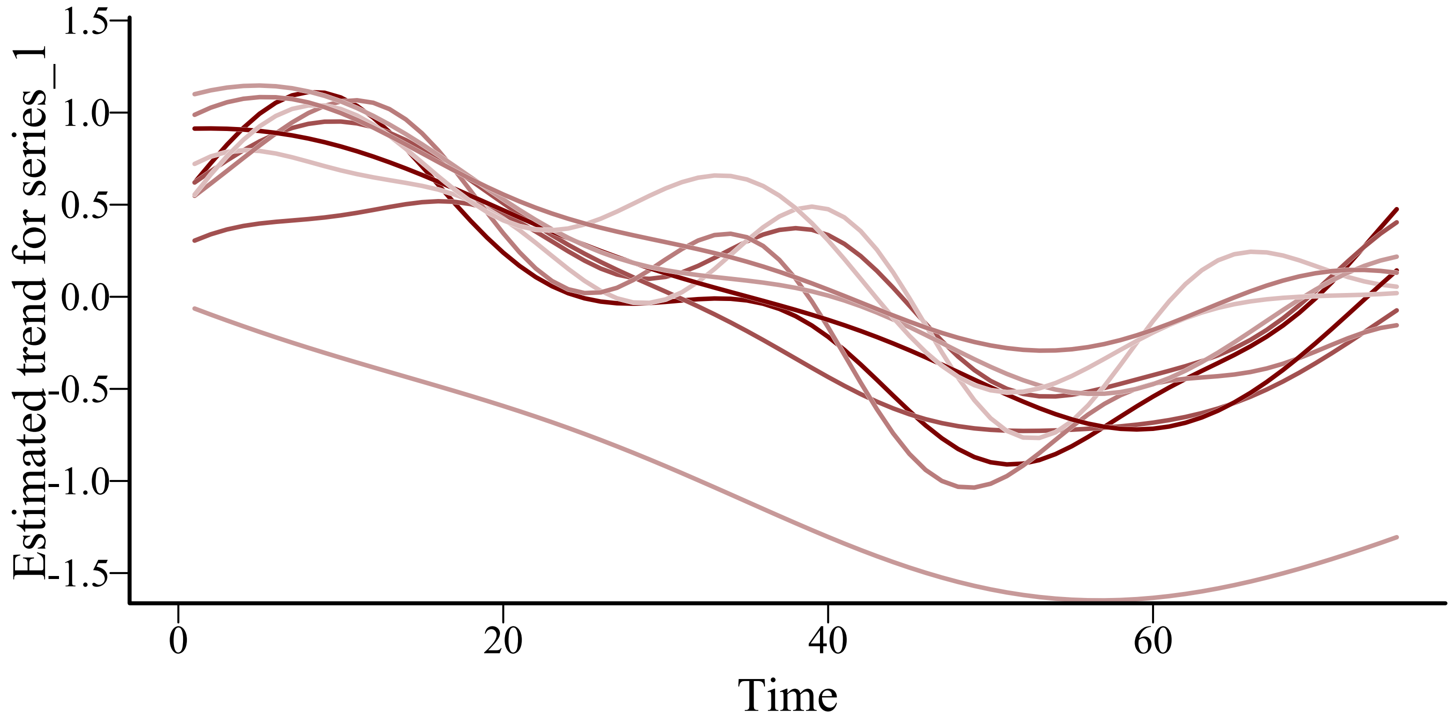
Draws of `trend`

Code Plot

```r
# view posterior draws of the trend
plot(model2, type = 'trend', realisations = TRUE,
     n_realisations = 10)
```

Draws of `trend`

Code Plot



But how can we extrapolate these to the future?

Ready for some multivariate statistical wizardry?

Ready

```r
sim_gp = function(trend_draw, h, rho, alpha){
  # extract training and testing times
  t <- 1:length(trend_draw); t_new <- 1:(length(trend_draw) + h)
  # calculate training covariance
  Sigma <- alpha^2 * exp(-0.5 * ((outer(t, t, "-") / rho) ^ 2)) +
    diag(1e-9, length(t))
  # calculate training vs testing cross-covariance
  Sigma_new <- alpha^2 * exp(-0.5 * ((outer(t, t_new, "-") / rho) ^ 2))
  # calculate testing covariance
  Sigma_star <- alpha^2 * exp(-0.5 * ((outer(t_new, t_new, "-") / rho) ^ 2))
+
    diag(1e-9, length(t_new))
  # draw one function realization of the stochastic Gaussian Process
  t(Sigma_new) %*% solve(Sigma, trend_draw) +
    MASS::mvrnorm(1, mu = rep(0, length(t_new)),
                  Sigma = Sigma_star - t(Sigma_new) %*% solve(Sigma,
Sigma_new))
}
```

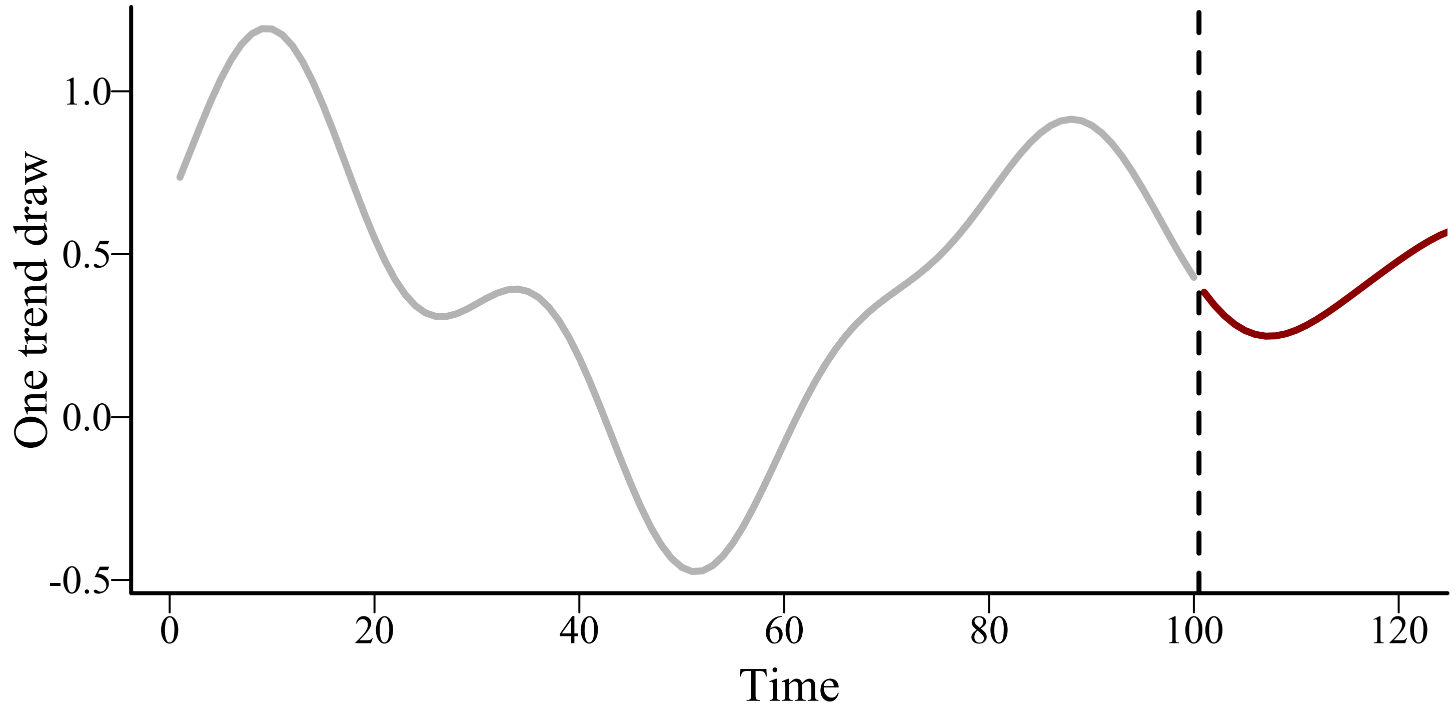
Wizardize one trend draw

Wizardry   Plot

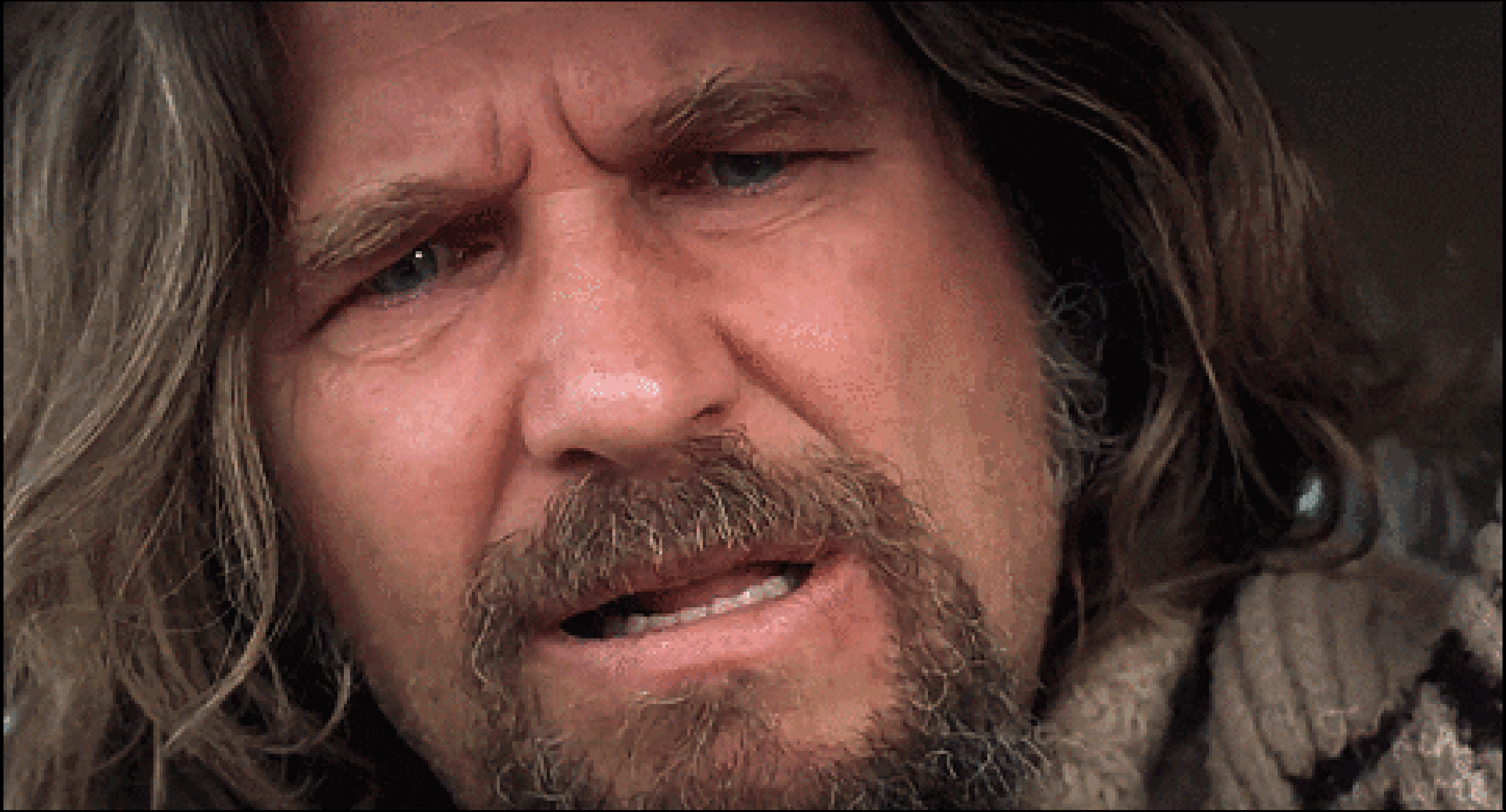
```
# extract trend parameter draws and plot one draw
trend_draws ← as.matrix(model2, variable = 'trend', regex = TRUE)
alpha_draws ← as.matrix(model2, variable = 'alpha_gp', regex = TRUE)
rho_draws ← as.matrix(model2, variable = 'rho_gp', regex = TRUE)
plot(1, type = 'n', bty = 'l',
     xlim = c(1, 130), ylim = range(trend_draws[1,]),
     ylab = 'One trend draw', xlab = 'Time')
lines(trend_draws[1,], col = 'gray70', lwd = 3.5)
# wizardize to extend draw forward 30 timesteps and plot
forecast_draw = sim_gp(trend_draw = trend_draws[1,], h = 30
                       alpha = alpha_draws[1,], rho = rho_draws[1,])
lines(x = 101:130, y = forecast_draw[101:130], lwd = 3.5, col = 'darkred')
abline(v = 100.5, lty = 'dashed', lwd = 2.5)
```

Wizardize one trend draw

Wizardry Plot



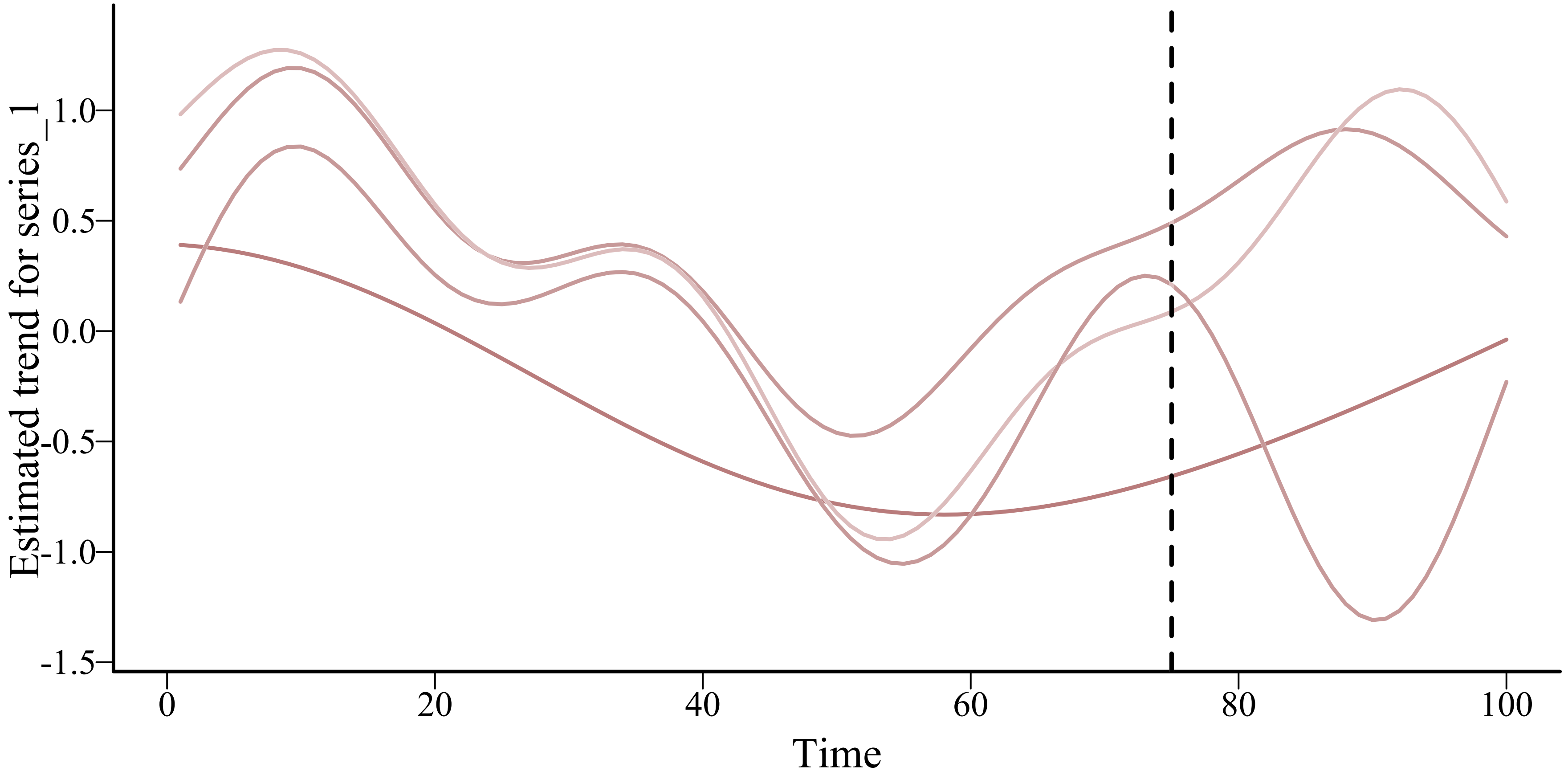
Piece of cake?



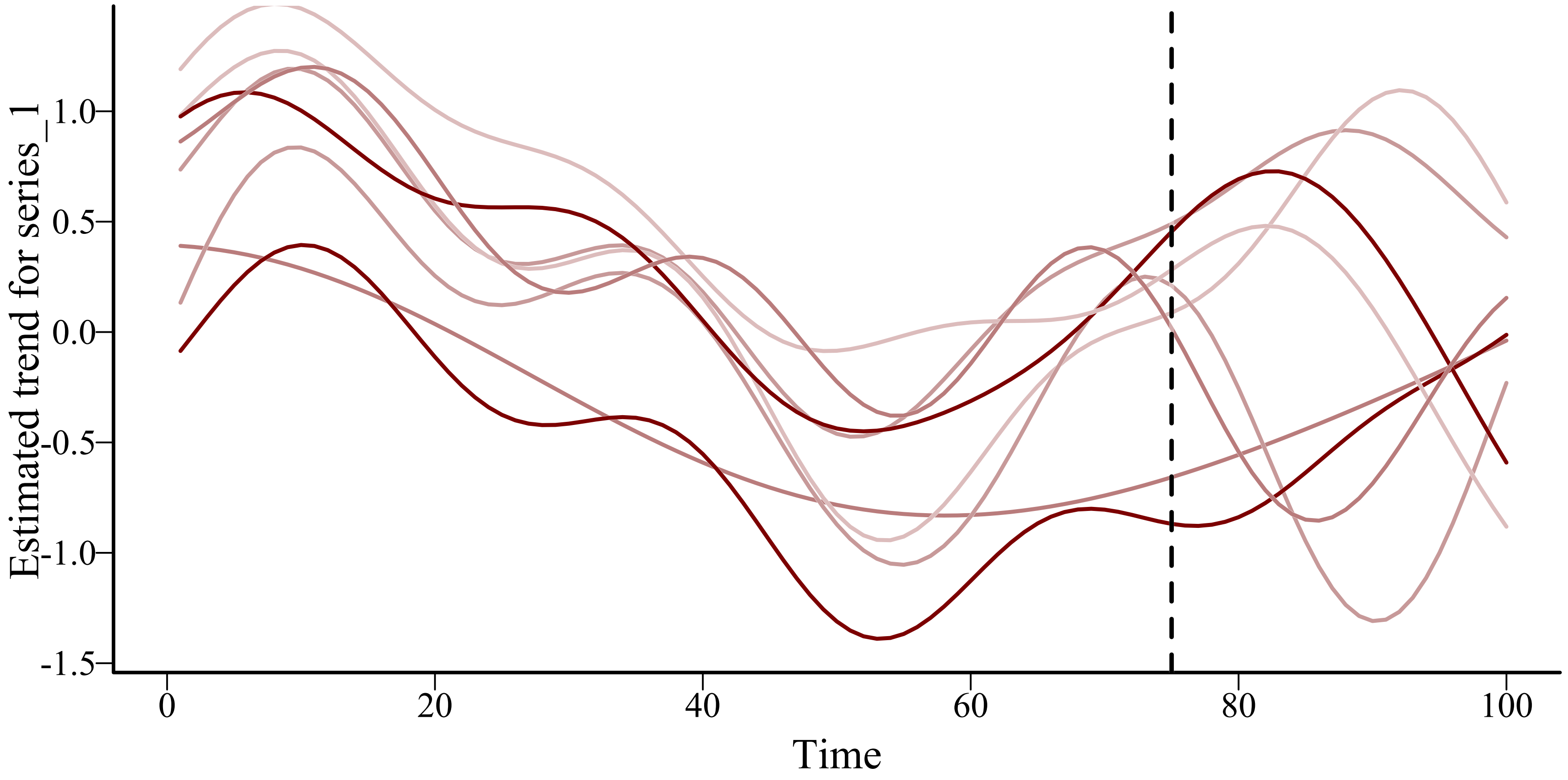
There is no wizardry 😥. Rather, each kind of trend (AR, GP etc...) has an underlying stochastic equation that can be used to extrapolate draws to the future

But doing this manually is slow and error-prone. `mvgam` does this *automatically* using `newdata`

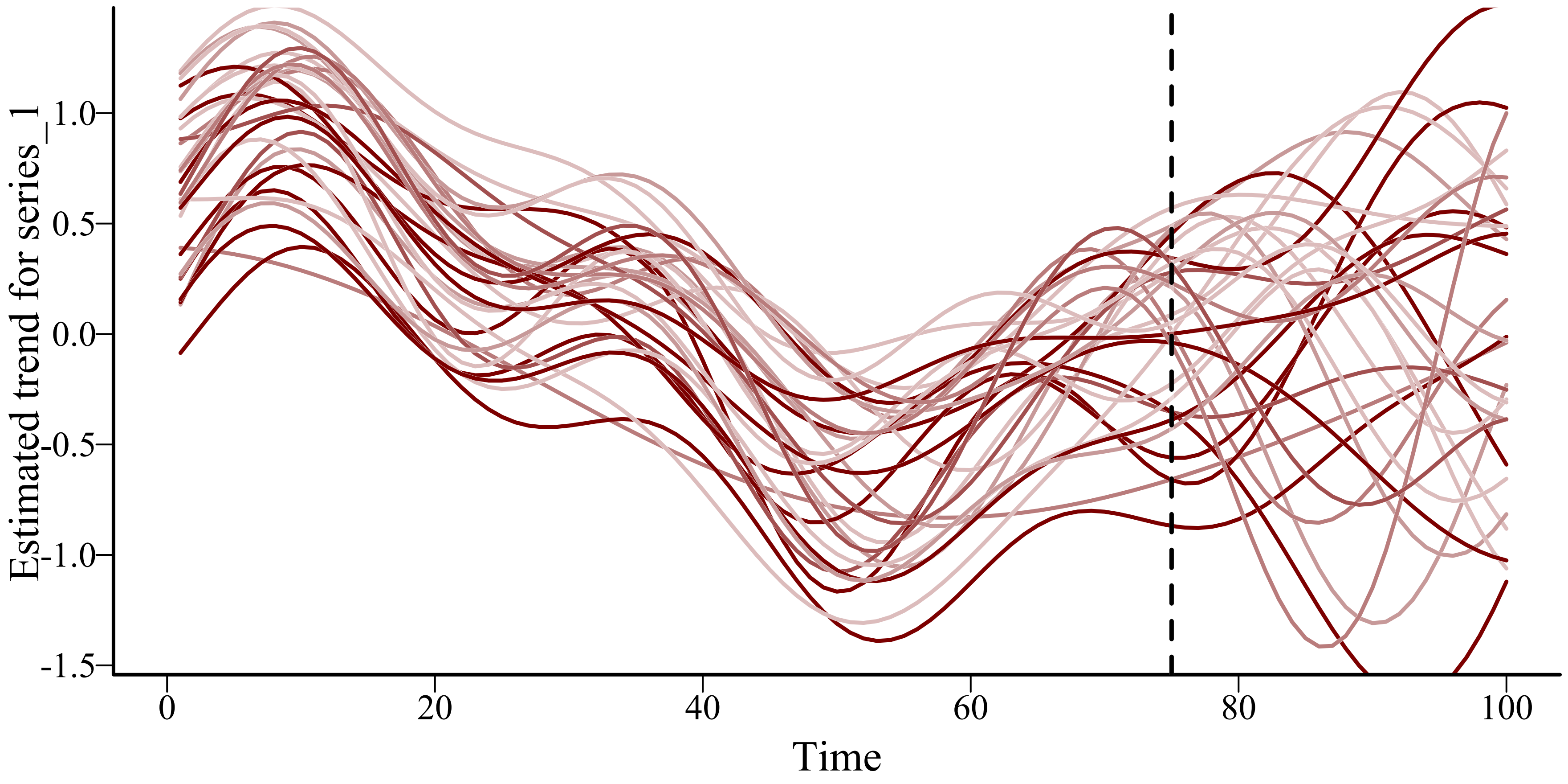
```r
plot(model2, type = 'trend', newdata = data_test,
realisations = TRUE, n_realisations = 4)
```



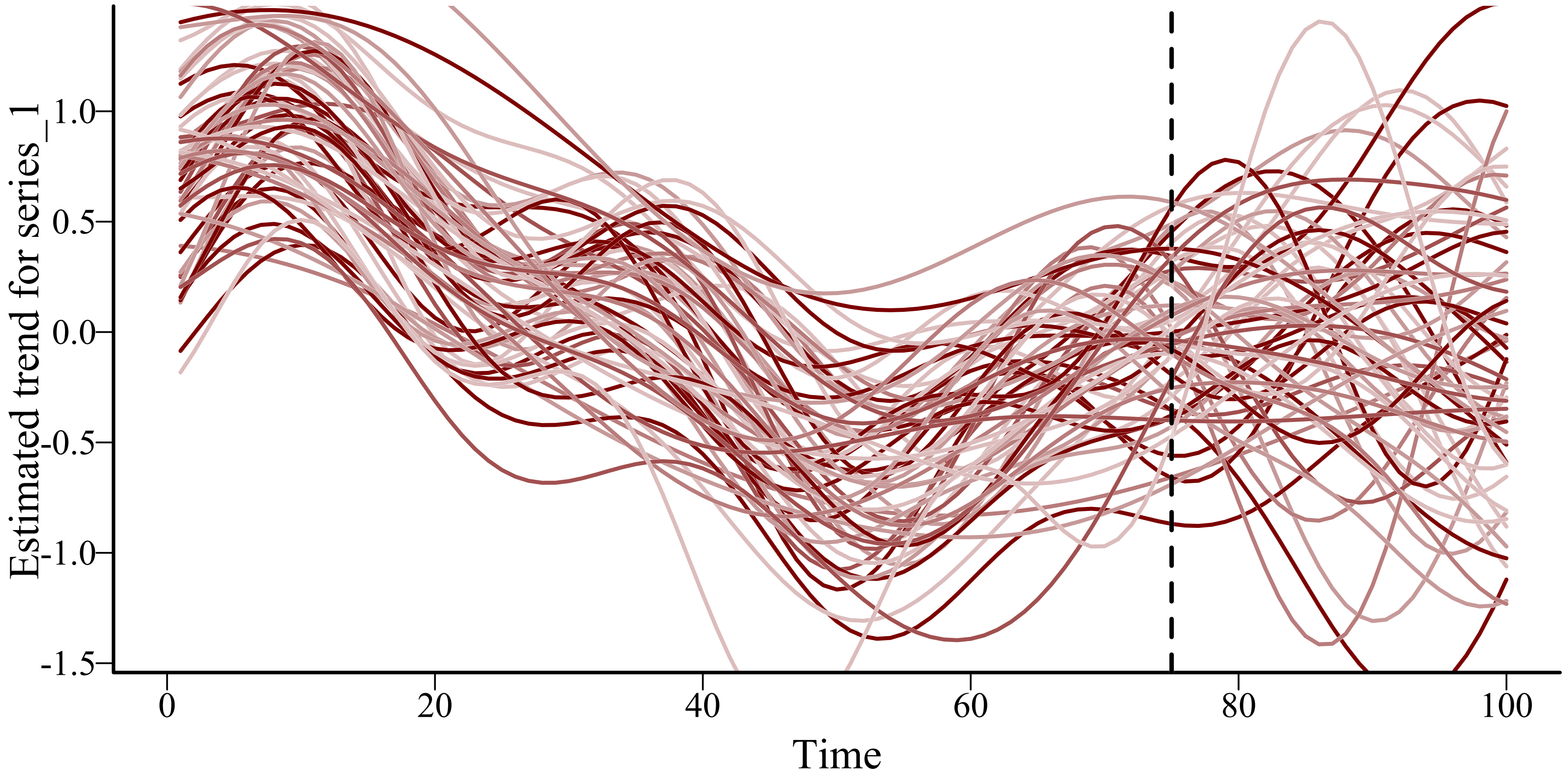
```
plot(model2, type = 'trend', newdata = data_test,
realisations = TRUE, n_realisations = 8)
```



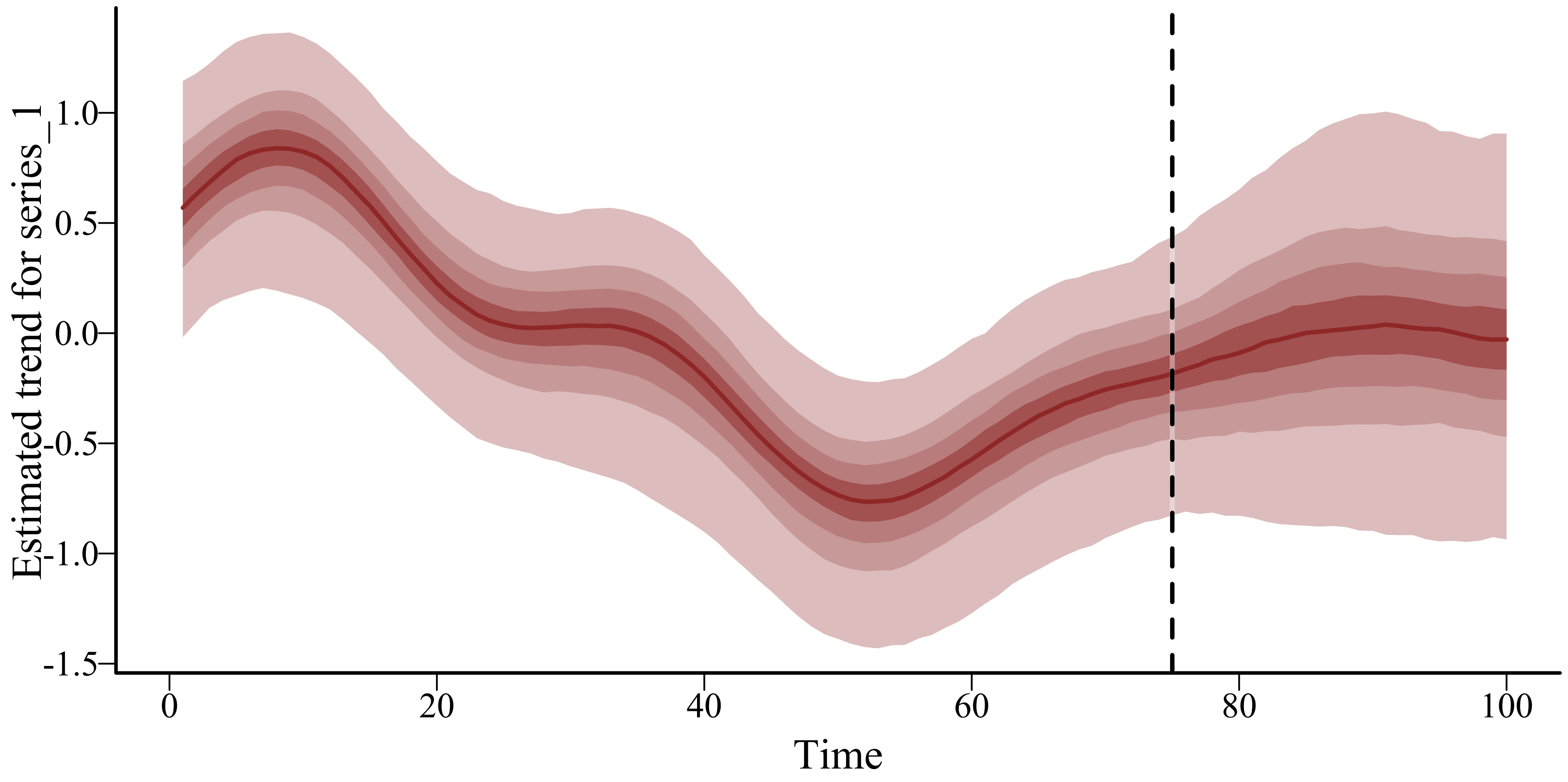
```r
plot(model2, type = 'trend', newdata = data_test,
realisations = TRUE, n_realisations = 30)
```



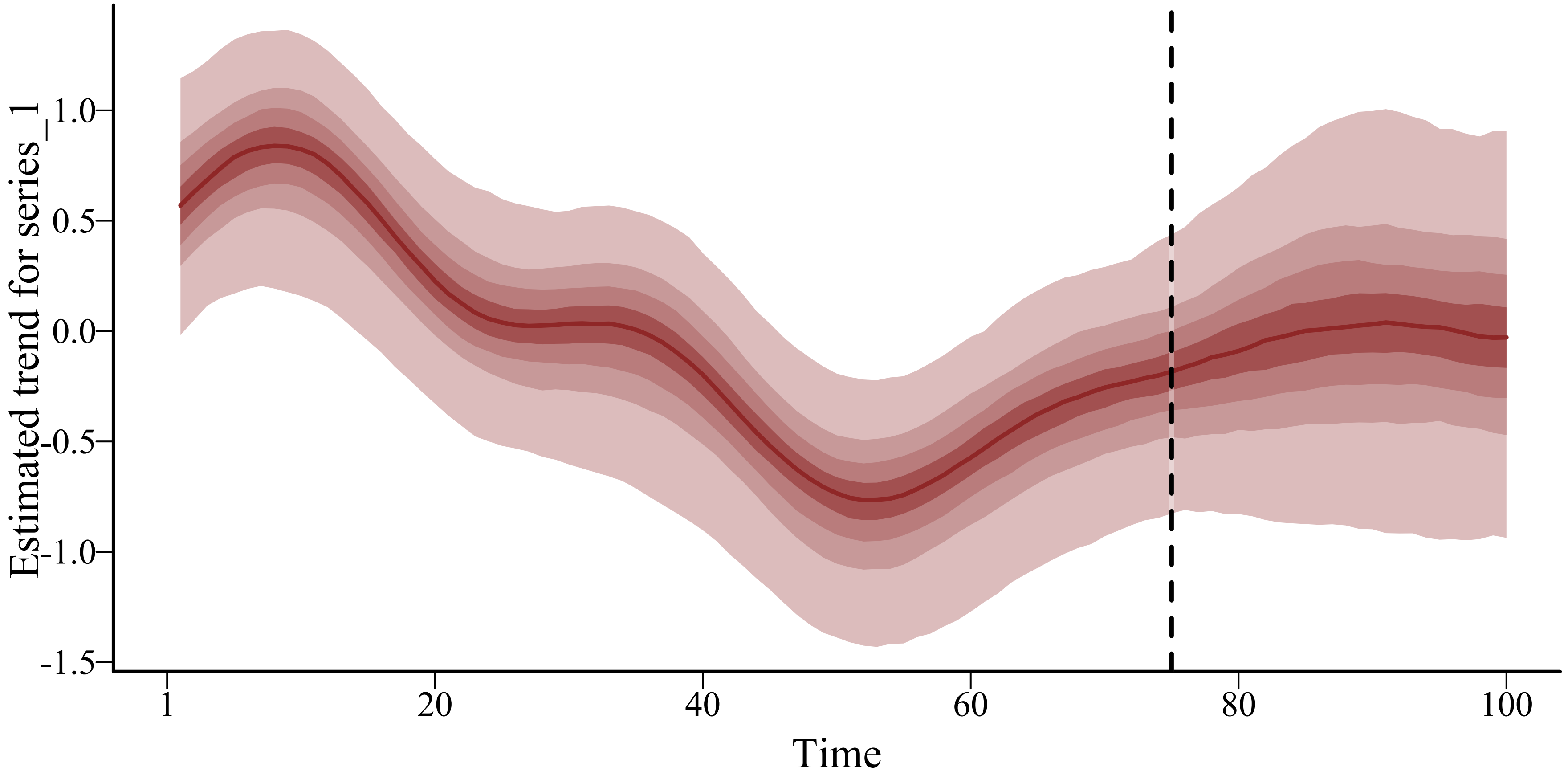
```
plot(model2, type = 'trend', newdata = data_test,
realisations = TRUE, n_realisations = 60)
```



```
plot(model2, type = 'trend', newdata = data_test,
realisations = FALSE)
```



```
Or: plot(forecast(model2, type = 'trend', newdata =
data_test), realisations = FALSE)
```



Once dynamic trend is extrapolated, computing forecasts is easy

We only need to supply any remaining "future" predictor values from covariates

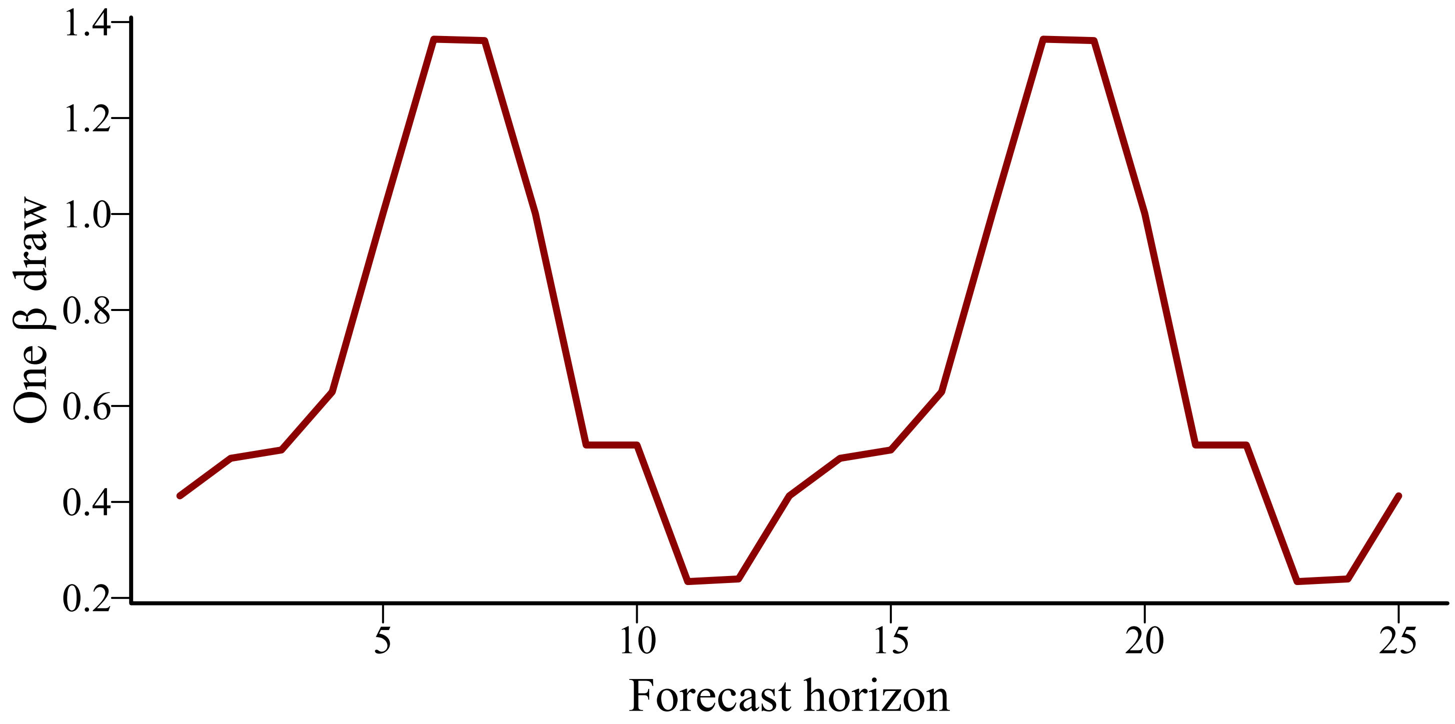
GAM covariate predictions

Code Plot

```r
# extract beta regression coefficient draws
beta_draws <- as.matrix(model2, variable = 'betas')
# calculate the linear predictor matrix for the GAM component
lpmatrix <- mvgam:::obs_Xp_matrix(newdata = poisdat$data_test,
                                  mgcv_model = model$mgcv_model)
# calculate linear predictor (link-scale) predictions for one draw
linkpreds <- lpmatrix %*% beta_draws[1,] + attr(lpmatrix, 'model.offset')
# plot the linear predictor values
plot(1, type = 'n', bty = 'l',
     xlim = c(1, length(linkpreds)),
     ylim = range(linkpreds),
     ylab = expression(One~beta~draw), xlab = 'Forecast horizon')
lines(linkpreds, col = 'darkred', lwd = 3.5)
```

GAM covariate predictions

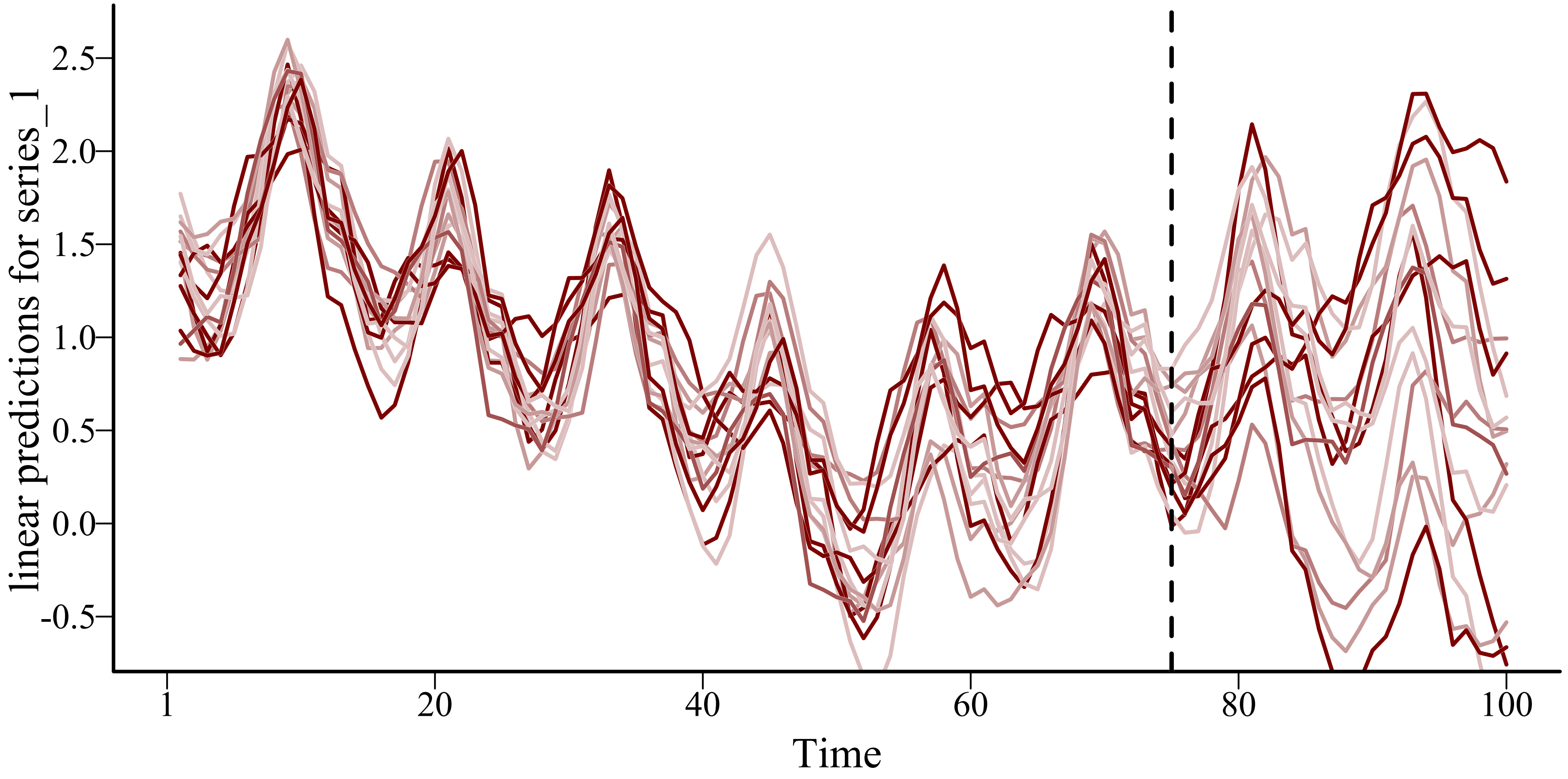
Code Plot



Covariate predictions are added to the trend predictions to give the full predictions *on the link scale*

`mvgam` **does this *automatically* using the** `forecast` **function**

```
plot(forecast(model2, type = 'link', newdata = data_test),
realisations = TRUE)
```



Live code example

Forecasting is easier if `newdata` are fed to `mvgam()`, but this results in a larger model object and requires test data be available now

When testing data not available, you can generate forecasts for new data later using `forecast.mvgam` (note, `time` values in `newdata` must follow immediately from `time` values in original training data)

But there are multiple *types* of predictions available. What are they?

Types of mvgam predictions

$$\mathbb{E}(Y_t)$$

$$Y_t \sim \text{Poisson}(\lambda_t)$$

$$\log(\lambda_t) = \alpha + f(season)_t + z_t$$

type = 'expected'
gives draws from the expected value of the posterior predictive distribution (i.e. the average of type = 'response')

In Poisson regression, this is the inverse of λ_t (i.e. exp(λ_t))

type = 'response'
gives draws from a random Poisson distribution using λ_t

type = 'link'
gives posterior draws from the linear predictor on the log scale

modified from Heiss 2022

`predict(object, type = 'link')`

Gives the real-valued, unconstrained linear predictor

- Takes into account uncertainty in GAM regression coefficients
- Can include uncertainty in any dynamic trend components
- Can be extracted from the fitted model as parameter `mus`

```
range(predict(model, type = 'link', process_error = FALSE))
```

```
## [1] -0.02145857  2.22647337
```

```
range(predict(model, type = 'link', process_error = TRUE))
```

```
## [1] -3.801796  5.165899
```

```
range(as.matrix(model, variable = 'mus', regex = TRUE))
```

```
## [1] -1.90681  4.60981
```

Hang on. Why do these differ?

```r
range(predict(model, type = 'link', process_error = TRUE))
```

```
## [1] -3.801796  5.165899
```

```r
range(as.matrix(model, variable = 'mus', regex = TRUE))
```

```
## [1] -1.90681  4.60981
```

`predict()` assumes the dynamic process has reached stationarity to tell us what we might expect if we see these same covariate values ***sometime in the future***

`mus` includes estimates for where the trend was ***at each point in the training data*** (hindcasts), so it is has less uncertainty

link predictions

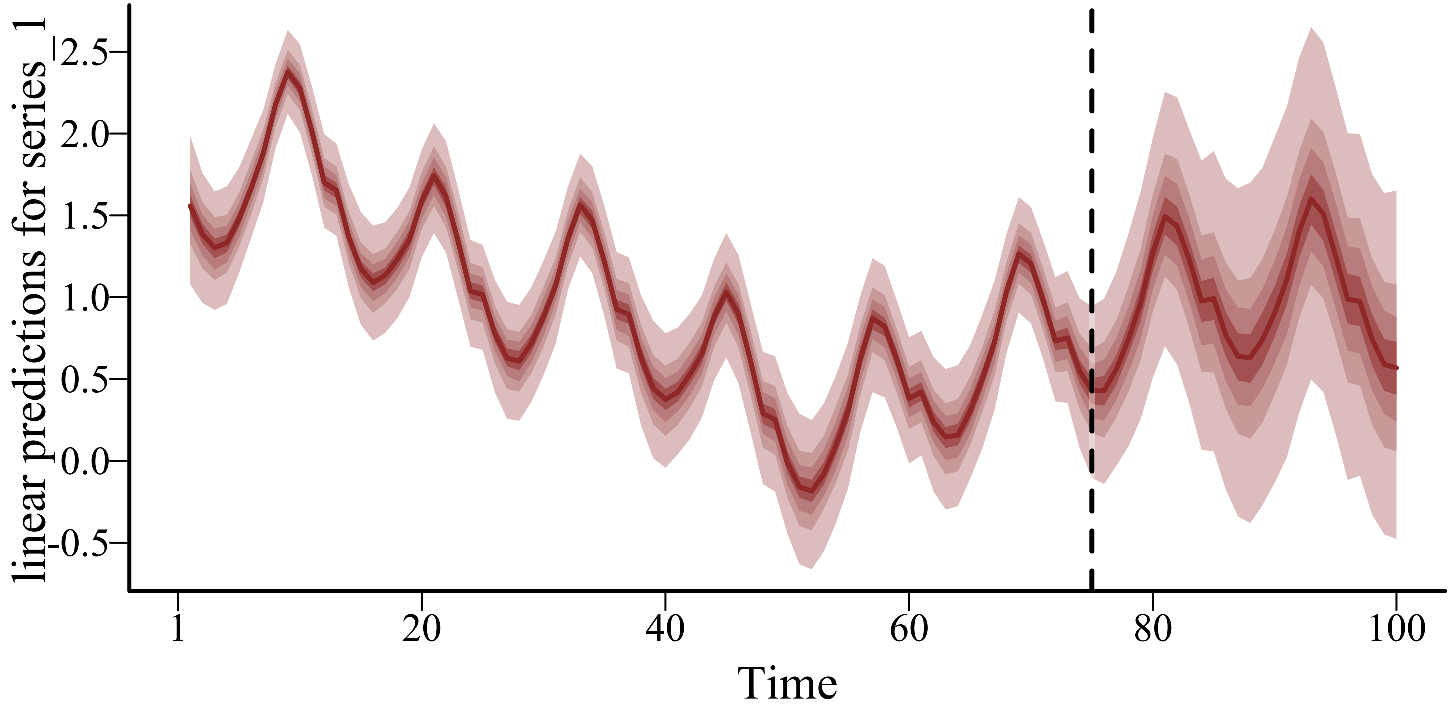
Code Plot

```r
# extract link-scale forecasts from the model
fc <- forecast(model, type = 'link')

# plot using the available S3 plotting function
plot(fc)
```

`link` predictions

Code Plot



`predict(object, type = 'expected')`

Gives the ***average*** prediction on the observation (response) scale

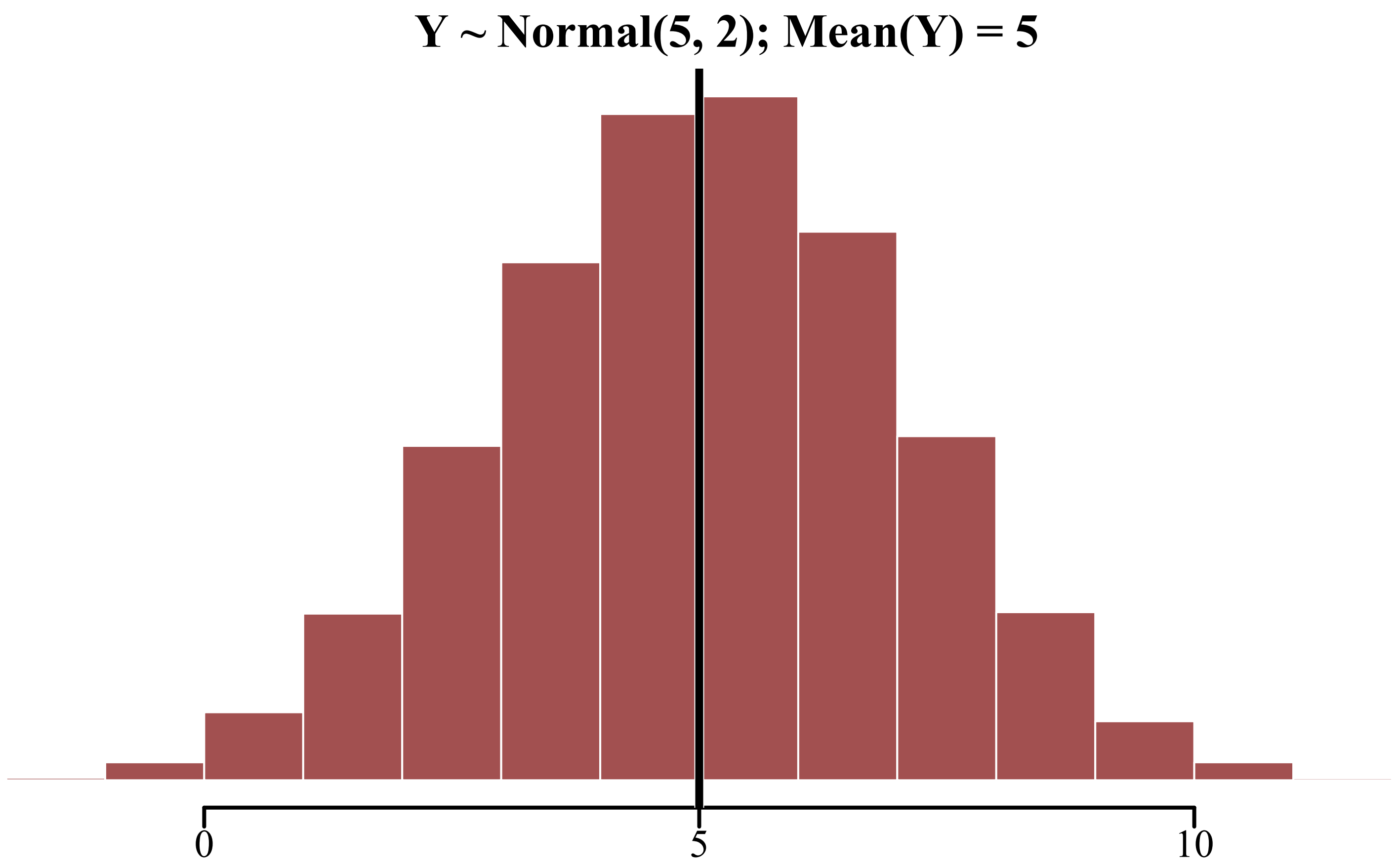
- Useful as we often want to get a sense of long-term averages for guiding scenario analyses
- ***Usually*** it is just the inverse link function applied to a prediction from `type = link`
- But not always!

This is probably the most confusing type of prediction

Normal distribution (skip)

$$\begin{aligned}
\boldsymbol{Y}_t &\sim \mathrm{Normal}(\mu_t, \sigma) \\
\mu_t &= \alpha + \boldsymbol{X}_t\beta + z_t && \leftarrow \text{type} = \text{'link'} \\
\mathbb{E}(\boldsymbol{Y}_t|\mu_t, \sigma) &= \mu_t && \leftarrow \text{type} = \text{'expected'}
\end{aligned}$$

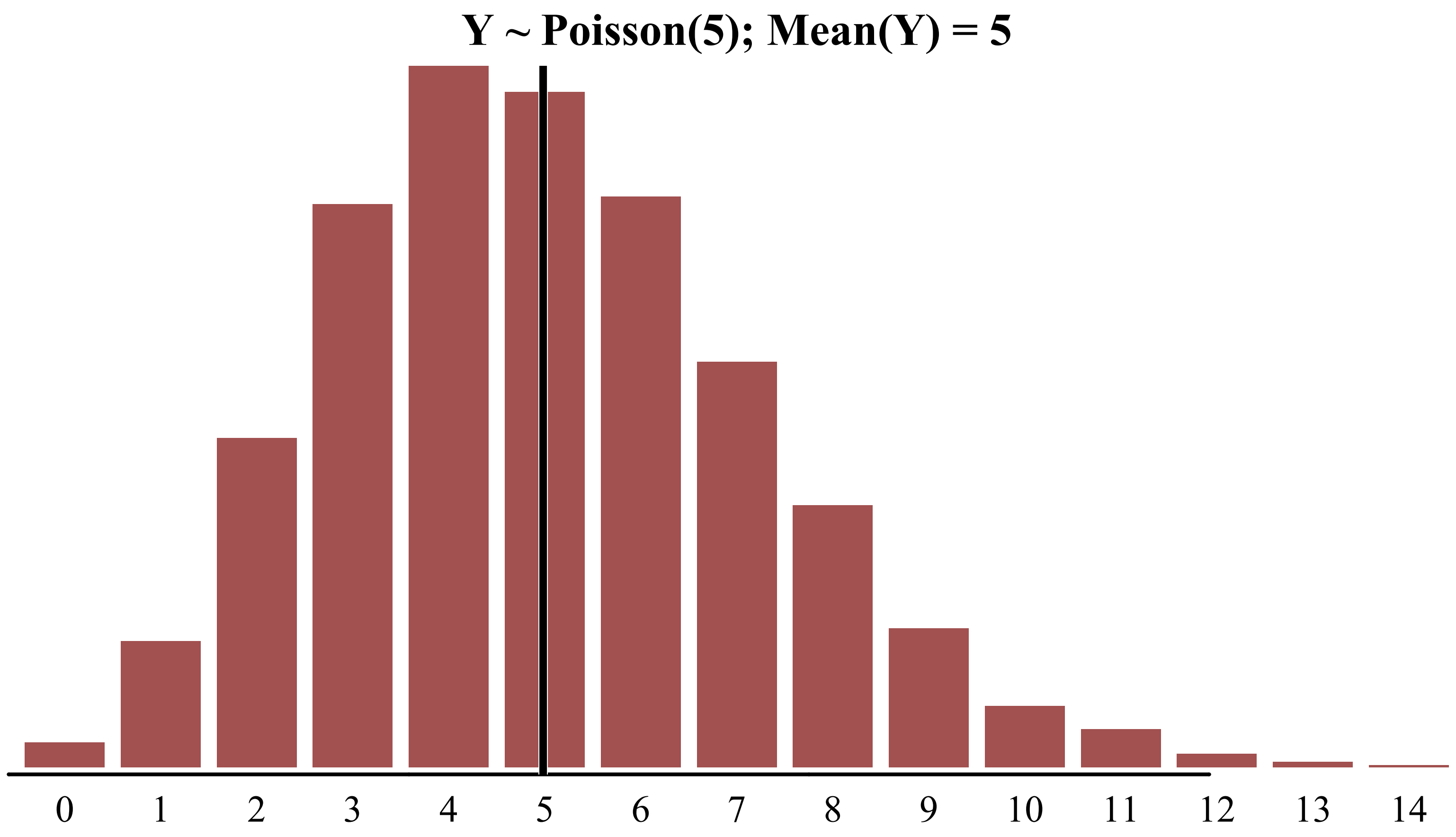
Normal distribution (skip)



Poisson distribution (skip)

$$\begin{aligned} \boldsymbol{Y}_t &\sim \text{Poisson}(\lambda_t) \\ log(\lambda_t) &= \alpha + \boldsymbol{X}_t\beta + z_t && \leftarrow \text{type} = \text{'link'} \\ \mathbb{E}(\boldsymbol{Y}_t|\lambda_t) &= \lambda_t && \leftarrow \text{type} = \text{'expected'} \end{aligned}$$

Poisson distribution (skip)



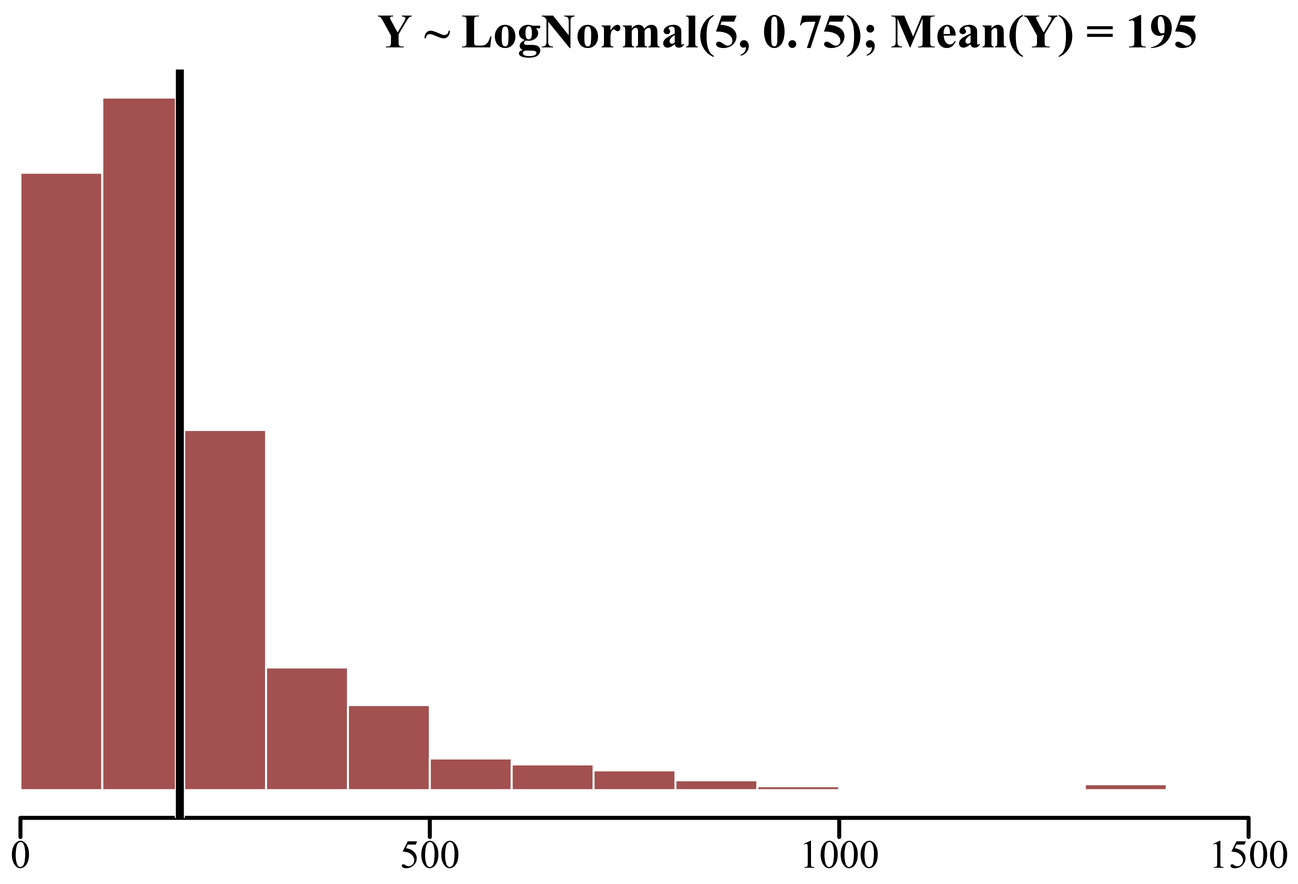
LogNormal distribution (skip)

$$\boldsymbol{Y}_t \sim \text{LogNormal}(\mu_t, \sigma)$$

$$\mu_t = \alpha + \boldsymbol{X}_t \beta + z_t \qquad \leftarrow \text{type} = \text{'link'}$$

$$\mathbb{E}(\boldsymbol{Y}_t | \mu_t, \sigma) = exp(\mu_t + \frac{\sigma^2}{2}) \qquad \leftarrow \text{type} = \text{'expected'}$$

LogNormal distribution (skip)



expected predictions

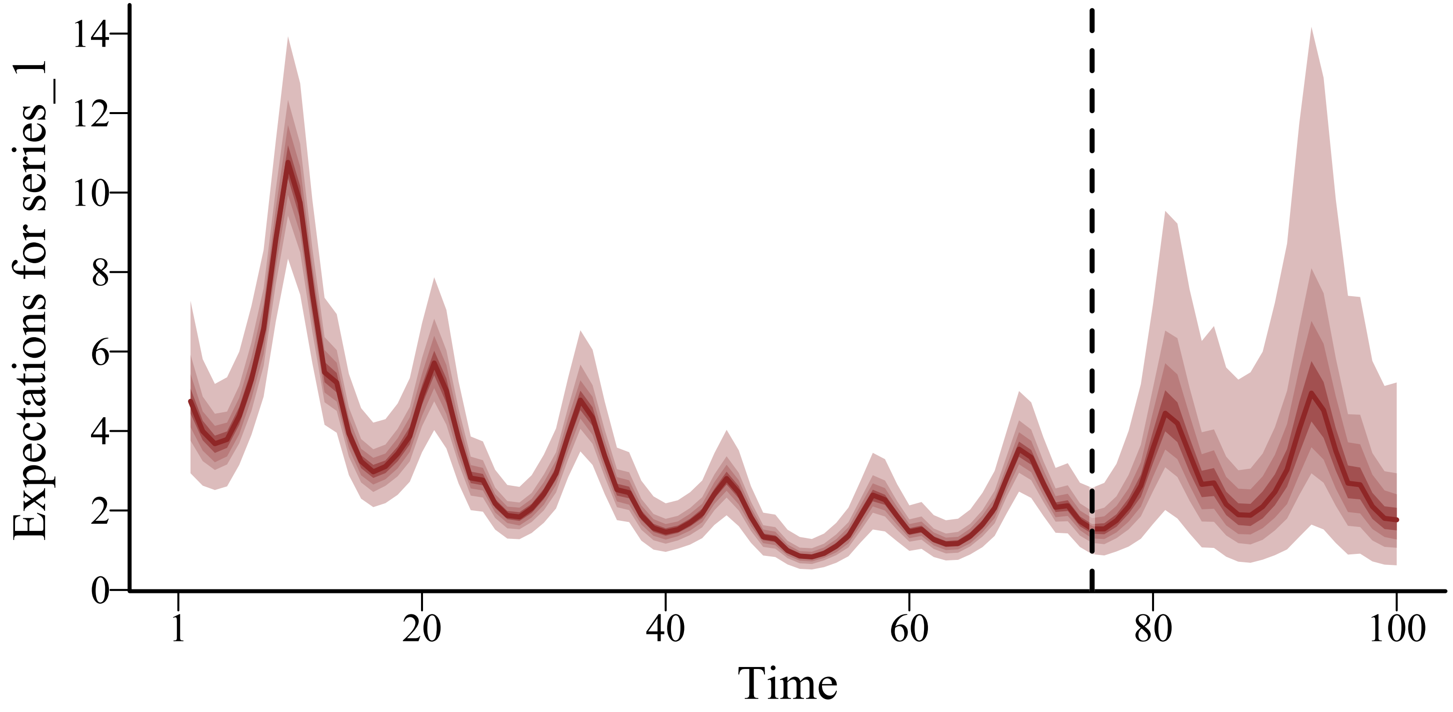
Code Plot

```r
# extract expectation-scale forecasts from the model
fc <- forecast(model, type = 'expected')

# plot using the available S3 plotting function
plot(fc)
```

expected predictions

Code Plot



```
predict(object, type = 'response')
```

Gives the predictions on the observation (response) scale

- Includes uncertainty in the linear predictor ***and*** any uncertainty arising from the observation process
- Some distributions only depend on the inverse link of the linear predictor (i.e. $Poisson(\lambda)$ or $Bernoulli(\pi)$))
- Others depend on additional shape / scale parameters (i.e. $Normal(\mu, \sigma)$ or $StudentT(\nu, \mu, \sigma)$)

These are the most often used type of predictions for evaluating forecasts

response predictions

Code

Plot

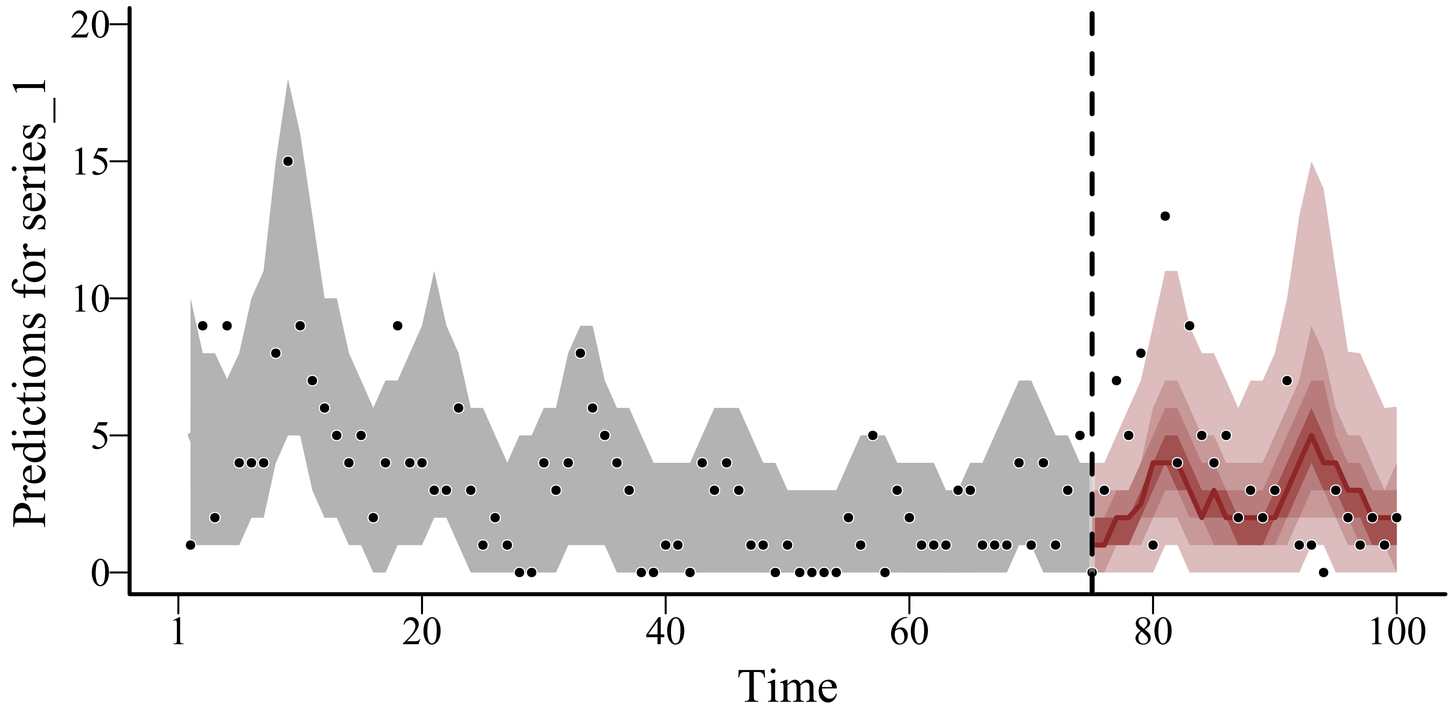
```r
# extract response-scale forecasts from the model
fc <- forecast(model, type = 'response')

# plot using the available S3 plotting function
plot(fc)
```

response predictions

Code

Plot



mvgam and brms 📦's

Type	mvgam	brms
link	`predict(type = 'link')`	`posterior_linpred()`
expected	`predict(type = 'expected')`	`posterior_epred()`
response	`predict(type = 'response')`	`posterior_predict()`

For all `mvgam` predictions, whether to include error in the dynamic process can be controlled using `process_error = TRUE` or `process_error = FALSE`

Posterior predictive checks

Fitted models yield coefficients

```r
coef(model)
```

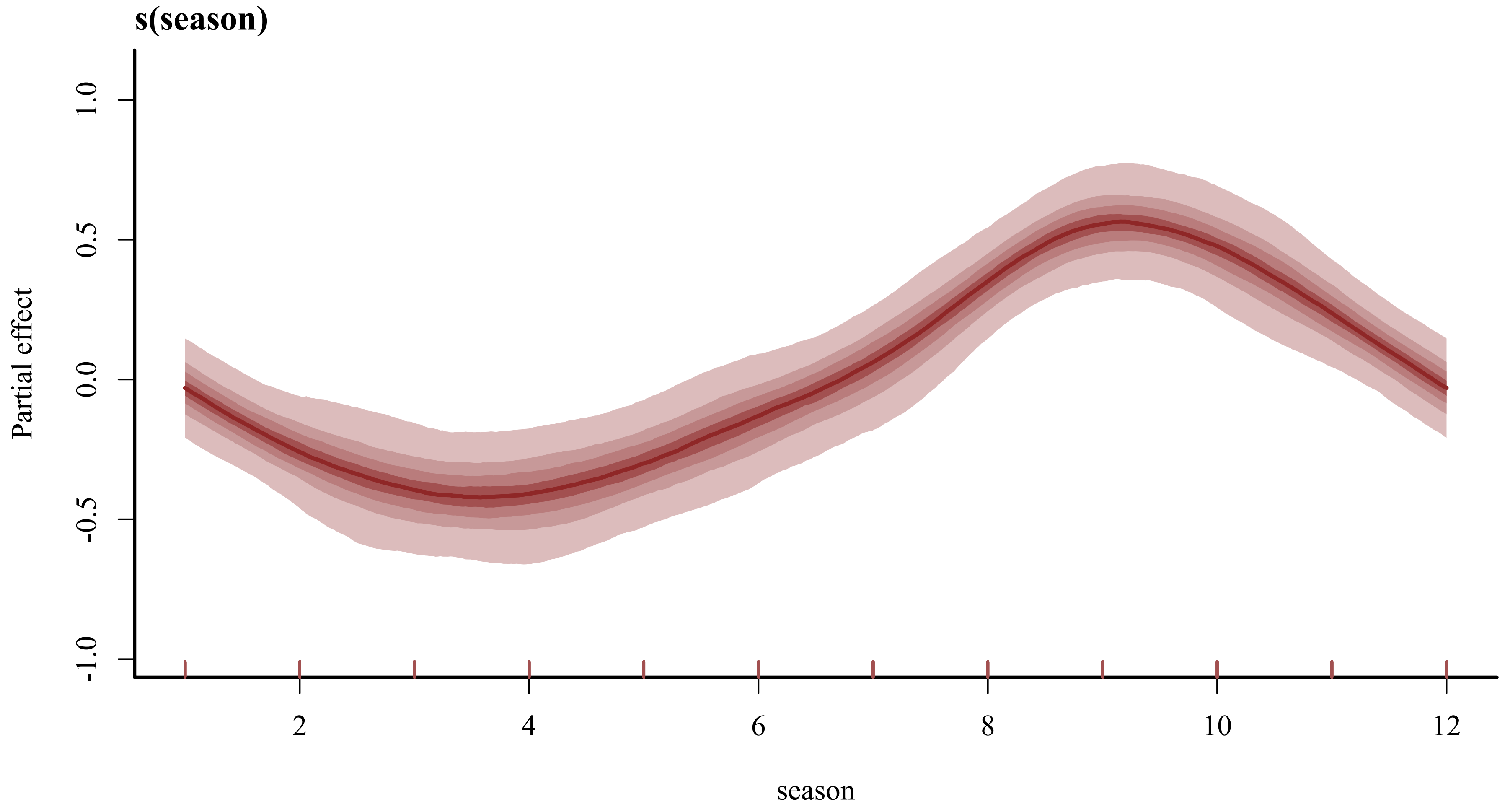
```
##                    2.5%        50%       97.5% Rhat n_eff
## (Intercept)  0.3774783  0.9876980  1.76805950    1   510
## s(season).1 -0.6334312 -0.3413885 -0.04340306    1  1586
## s(season).2 -0.7103705 -0.3934130 -0.09796405    1  2081
## s(season).3 -0.4893098 -0.1710475  0.12470842    1  2116
## s(season).4 -0.1712199  0.1409405  0.41121075    1  2022
## s(season).5  0.2836828  0.5544740  0.81570900    1  2011
## s(season).6  0.1086623  0.3892950  0.66113575    1  1973
```

Interpret coefficients?

- These coefficients are acting on the ***link scale***
  - Often result in nonlinear relationships on response scale
  - Very often, the coefficients are ***correlated somehow***
  - This is especially the case in GAMs!
  - Don't worry about p-values or intervals, use ***posterior predictions*** instead

- Start with ***partial effects*** on link scale
  - These are conditional on all other effects being zero
  - negative values ⇨ covariate reduces the response
  - positive values ⇨ covariate increases the response

```
plot(model, type = 'smooths')
```



Look at partial residuals

Partial effect residuals can be thought of as ***residuals that would be obtained*** by dropping a specific term from the model

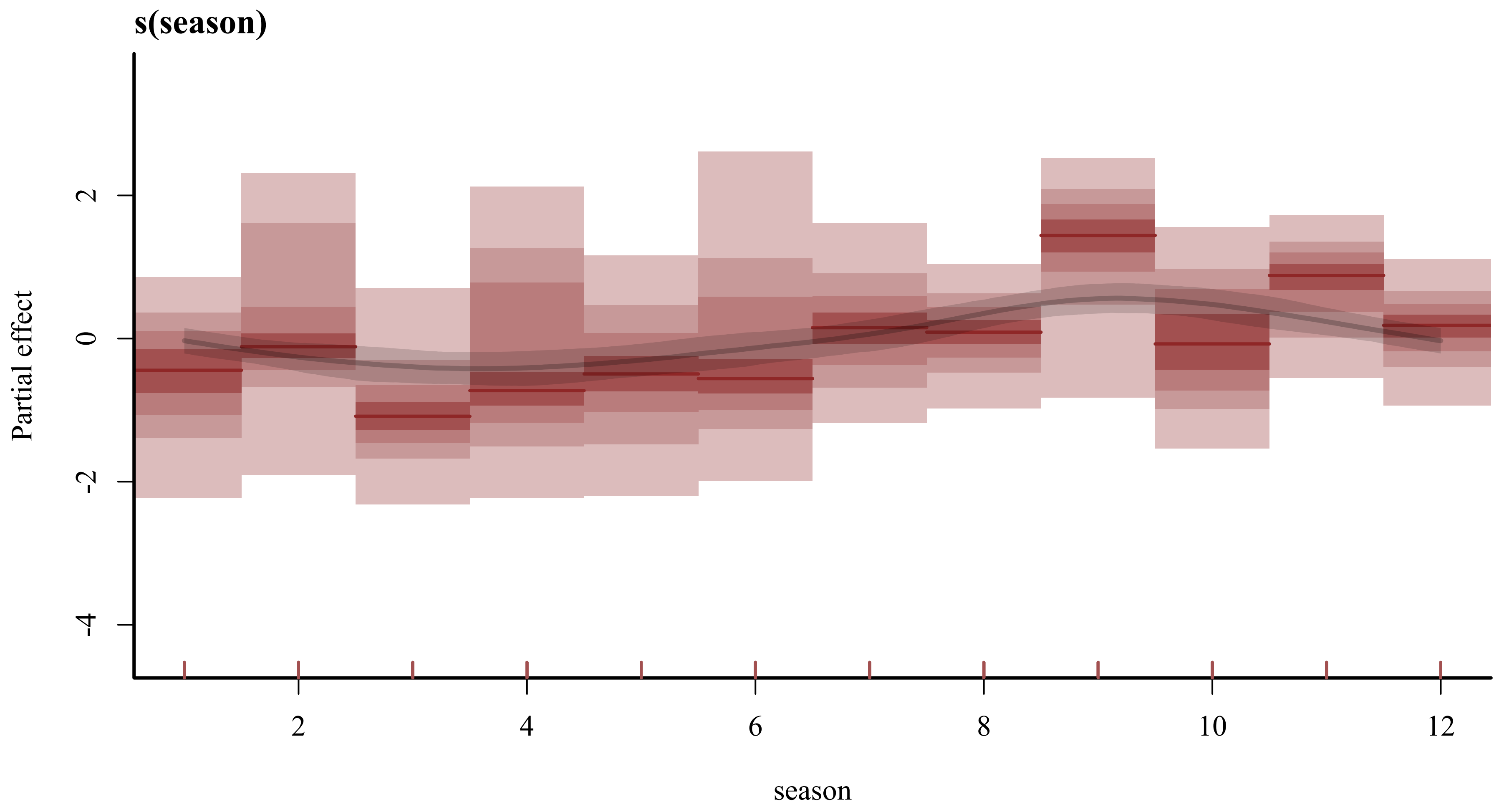
$$\hat{\epsilon}^{partial} = \hat{f}\left(x\right) + \hat{\epsilon}^{DS}$$

Where:

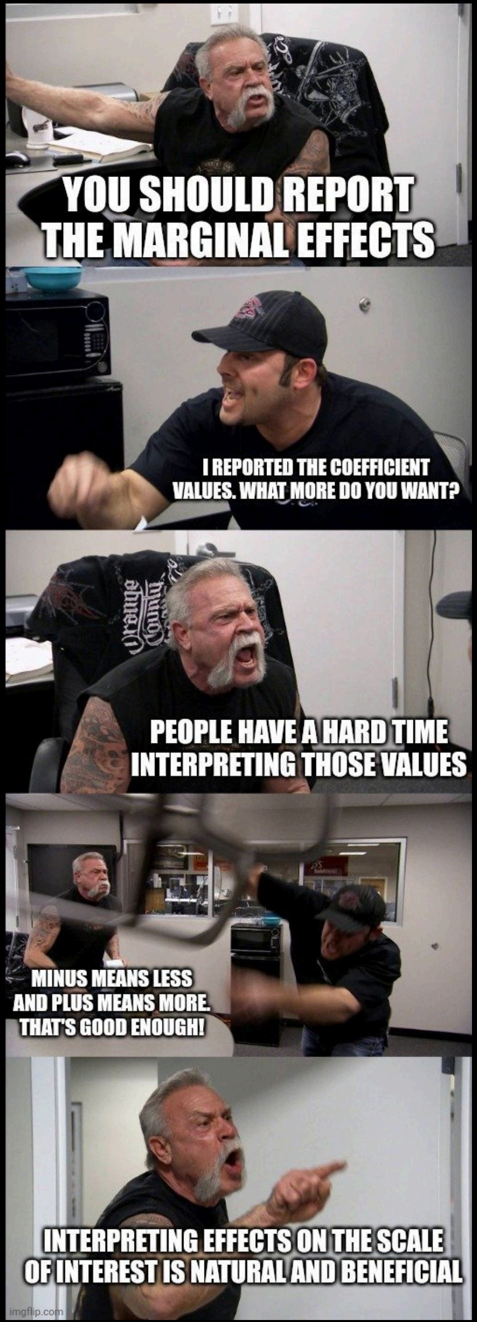
- $\hat{f}\left(x\right)$ is estimated smooth function for the effect of covariate x
- $\hat{\epsilon}^{DS}$ is a draw of randomized quantile (Dunn-Smyth) residuals

We would expect these to be scattered evenly around the smooth for a well fitting model

```
plot(model, type = 'smooths', residuals = TRUE)
```



Ok. but what do these things actually, really *mean*?


YOU SHOULD REPORT THE MARGINAL EFFECTS
I REPORTED THE COEFFICIENT VALUES. WHAT MORE DO YOU WANT?
PEOPLE HAVE A HARD TIME INTERPRETING THOSE VALUES
MINUS MEANS LESS AND PLUS MEANS MORE. THAT'S GOOD ENOUGH!
INTERPRETING EFFECTS ON THE SCALE OF INTEREST IS NATURAL AND BENEFICIAL
imgflip.com

Credit @stephenjwild

Interpreting on the *response* scale

Some key questions you should ask of a fitted model

- Can the model simulate realistic data?
- Does the model capture salient features of the data that you'd like to predict?
- What criteria would you use to determine whether one model is more suitable than another?

Very often, these questions can only be answered by looking at what kinds of predictions a model makes ***on the response scale***

Posterior predictive checks

Statistical models can be used to generate (i.e. simulate) new outcome data

- Can either use the same covariates used to train the model
- Or can use `newdata` for scenario modelling (including forecasting)

To generate new outcome data we can simulate from the model's posterior predictive distribution

" *The idea is simple: if a model is a good fit then we should be able to use it to generate data that looks a lot like the data we observed*"
Gabry & Mahr

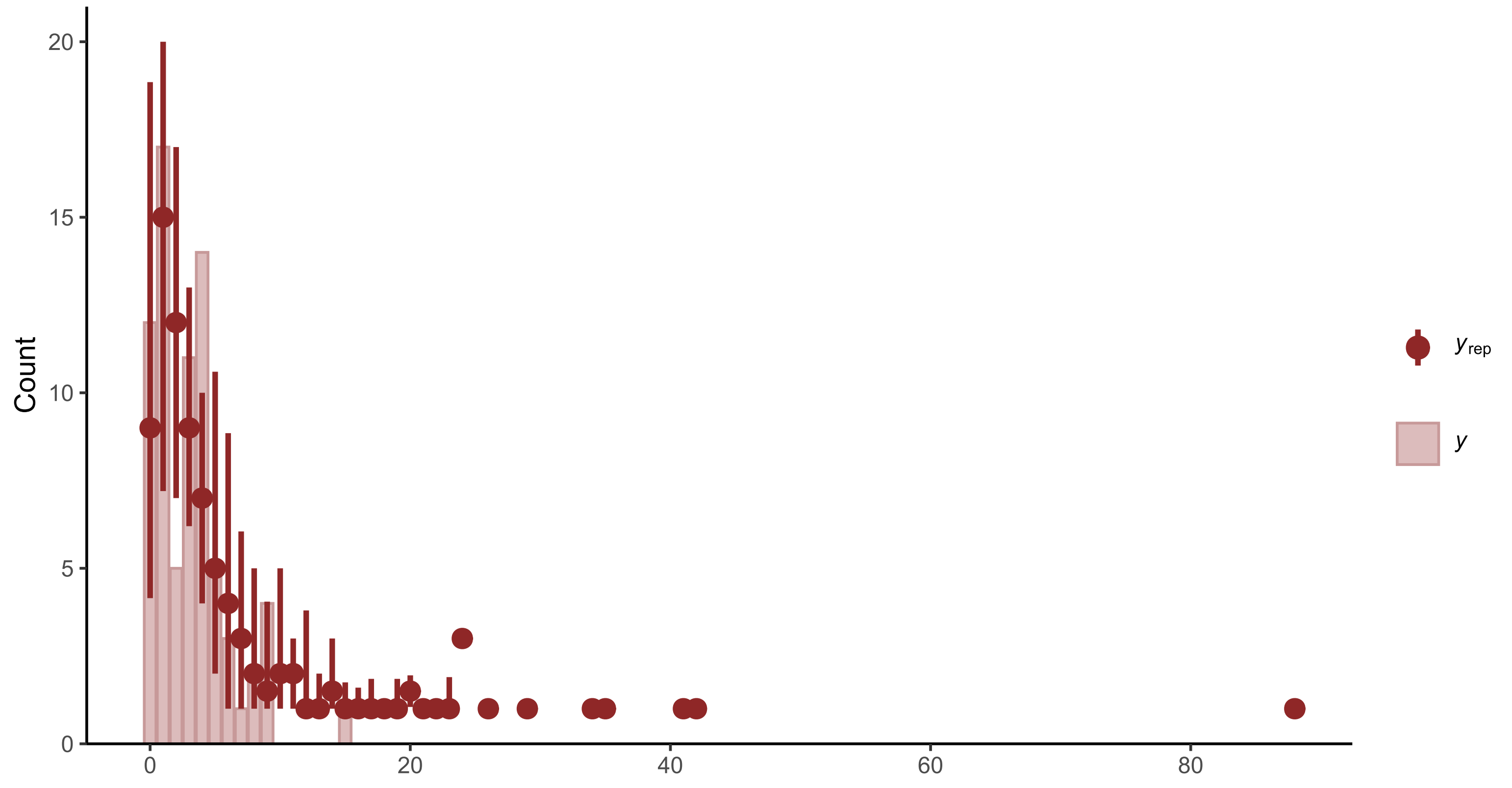
A PPC barplot

Code Plot

```
# view barplots of true data vs simulated predictions
pp_check(model, type = 'bars', ndraws = 25)
```

A PPC barplot

Code Plot



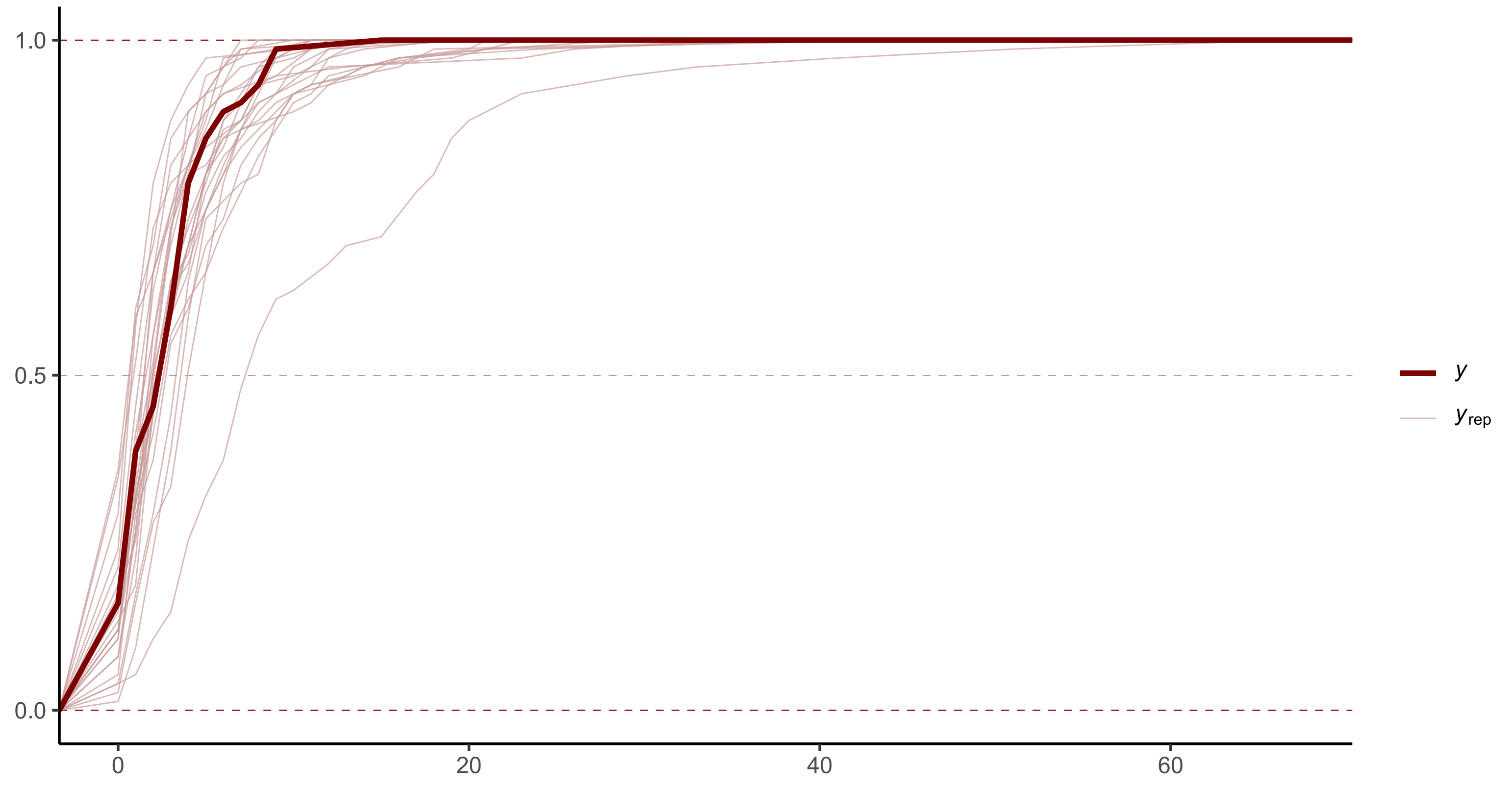
A PPC cumulative distribution

Code Plot

```
# view the simulated vs true cumulative distribution functions
pp_check(model, type = 'ecdf_overlay', ndraws = 25)
```

A PPC cumulative distribution

Code Plot



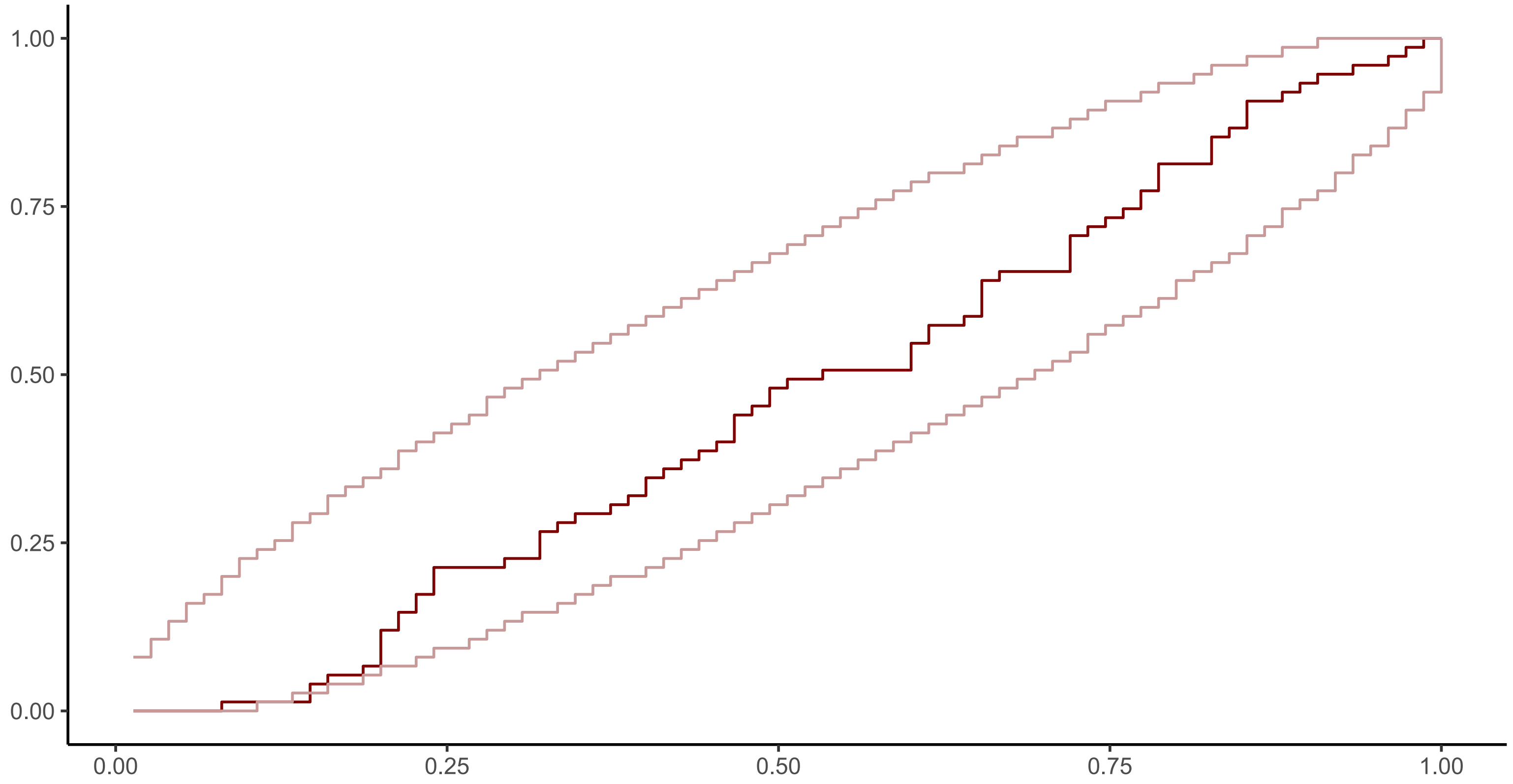
A PIT CDF

Code Plot

```
# view the simulated vs true count frequencies
pp_check(model, type = 'pit_ecdf')
```

A PIT CDF

Code Plot



Comparing fits with `loo()`

```r
model_good <- mvgam(y ~ s(season, bs = 'cc', k = 8) +
                      gp(time, k = 20, c = 5/4, scale = FALSE),
                    data = poisdat$data_train,
                    trend_model = 'None',
                    family = poisson())
```

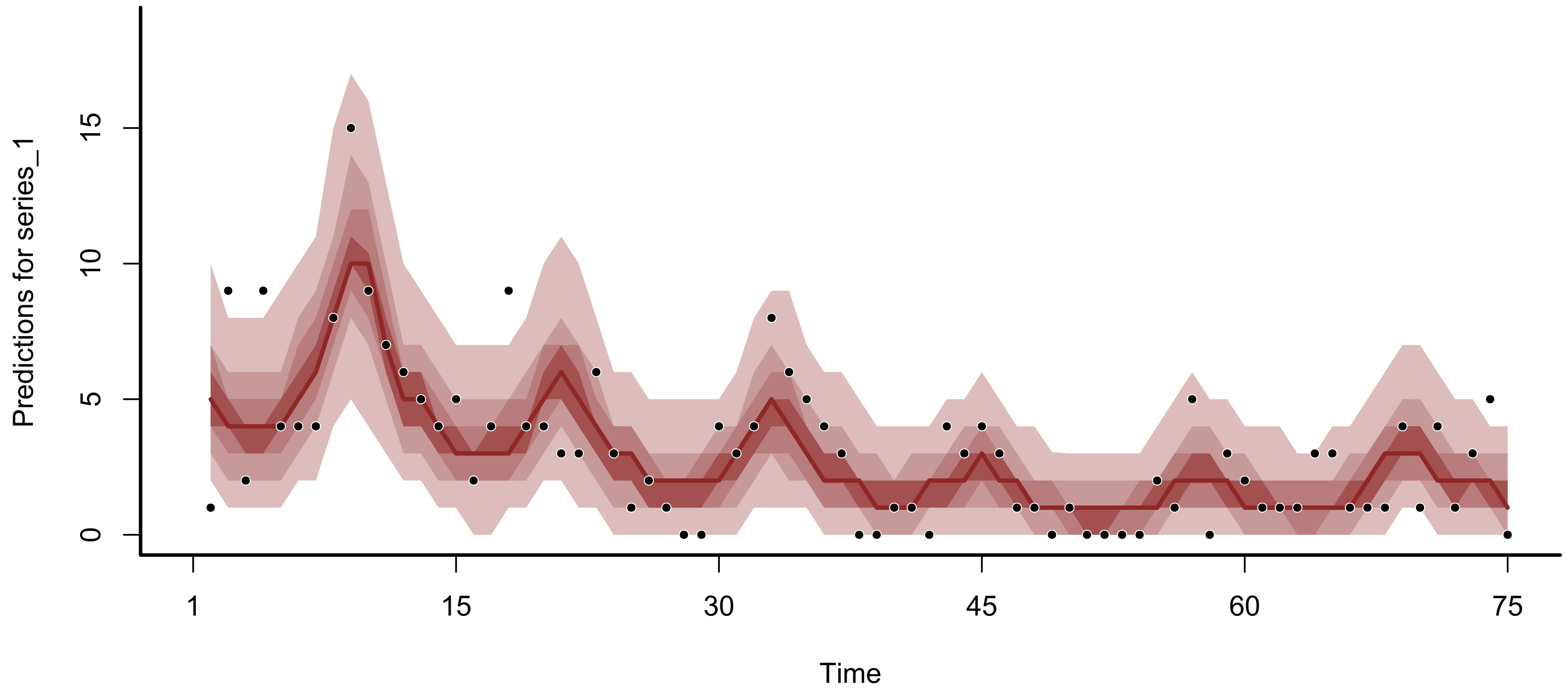
A GP of `time`, together with the cyclic seasonality, is a good model here

Comparing fits with `loo()`

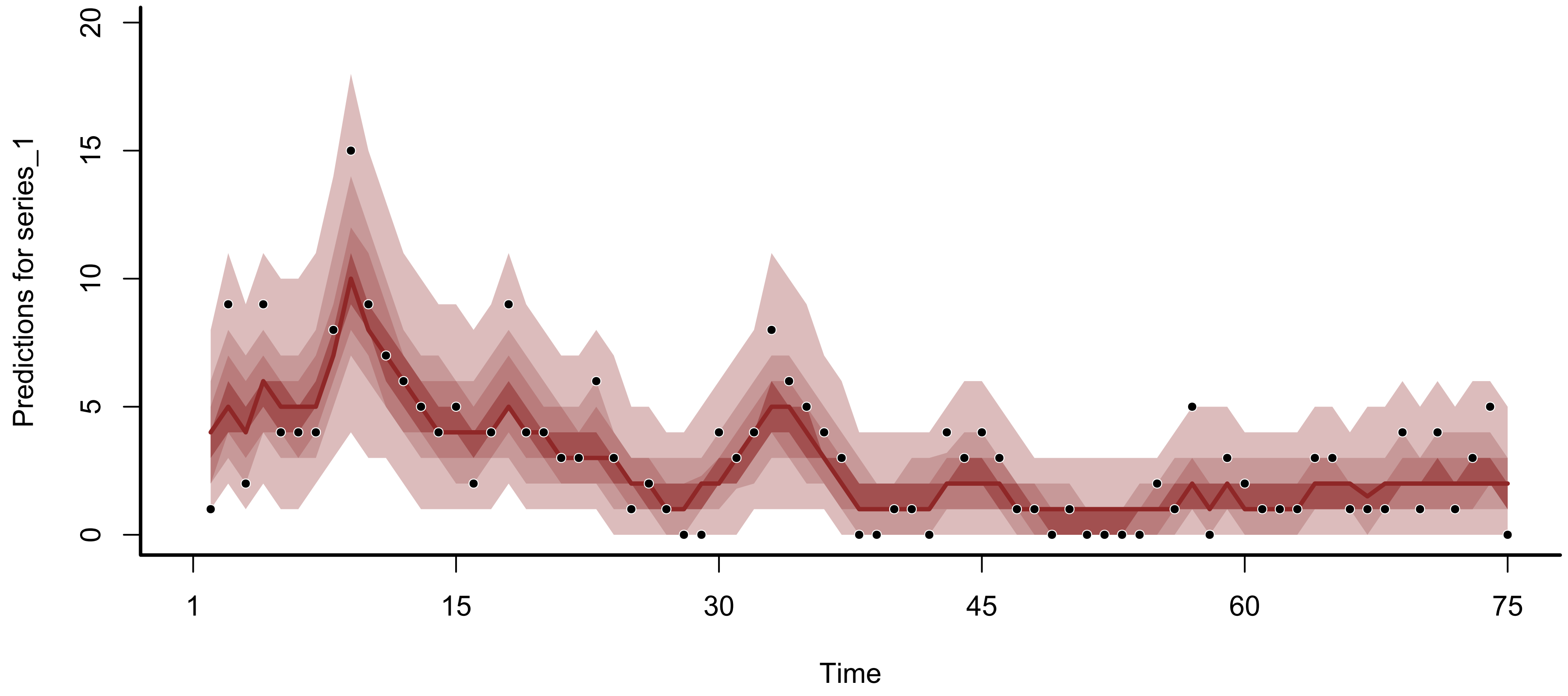
```r
model_bad <- mvgam(y ~ 1,
                   data = poisdat$data_train,
                   trend_model = RW(),
                   family = poisson())
```

A RW with no seasonality will fit the data *very well*, but gives *bad* predictions

```
plot(hindcast(model_good))
```



```
plot(hindcast(model_bad))
```



```
loo_compare(model_good, model_bad)
```

```
##             elpd_diff se_diff
## model_good     0.0       0.0
## model_bad   -753.7      43.3
```

PPCs and `loo()` using training covariates are a great first step to check model validity and begin comparing models

But they only assess how well the model predicts against the training data

How else can we verify models? Using `newdata` for response predictions ⇨ counterfactual *scenarios*

Marginal & conditional predictions

"*Applied researchers are keen to report simple quantities that carry clear scientific meaning*" (Arel-Bundock 2023)

This is often challenging because:

- Intuitive estimands and uncertainties are tedious to compute
- Nonlinear terms, nonlinear link functions, interaction effects and observation parameters all make these effects nearly impossible to gain from looking at coefficients alone
- Most software emphasizes coefficients and *p*-values over meaningful interpretations

`predict.mvgam()`

Feed `newdata` consisting of particular covariate values that represent scenarios you'd like to explore

- Can be simple: predict a smooth function along a fine-spaced grid to explore the smooth's shape and / or derivatives
- Or can be complex: integrate over a high-dimensional grid of predictors to understand the average impact of a predictor on the response

Users can implement the wonderful `datagrid()` function from `marginaleffects` 📦 to effortlessly generate a `data.frame` of covariate values for scenario predictions

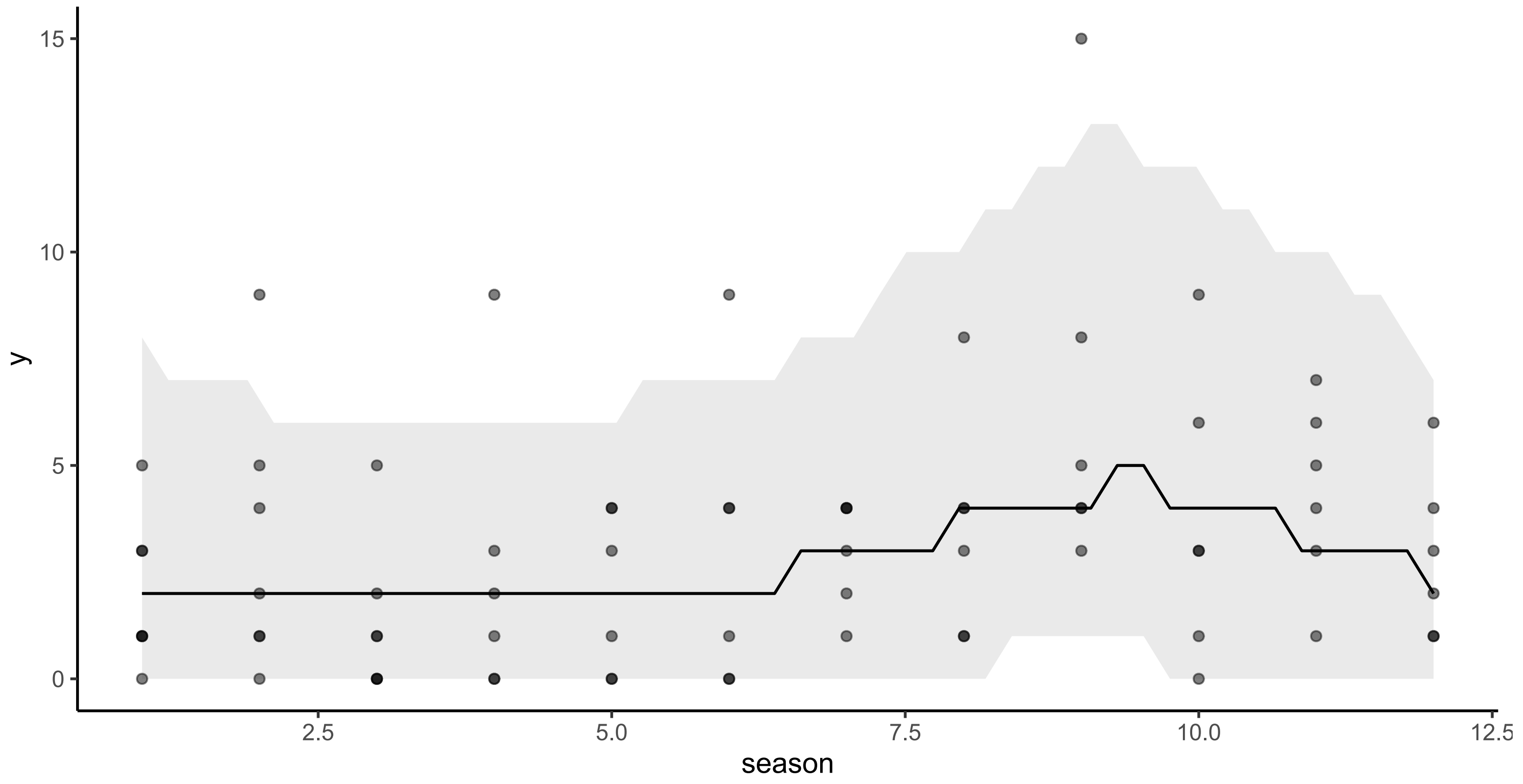
Conditional smooths

Code Plot

```r
# use plot_predictions to visualise conditional effects
# on the scale of the response
library(ggplot2)
plot_predictions(model, condition = 'season',
                 points = 0.5, process_error = FALSE) +
  theme_classic()
```

Conditional smooths

Code Plot



Posterior contrasts

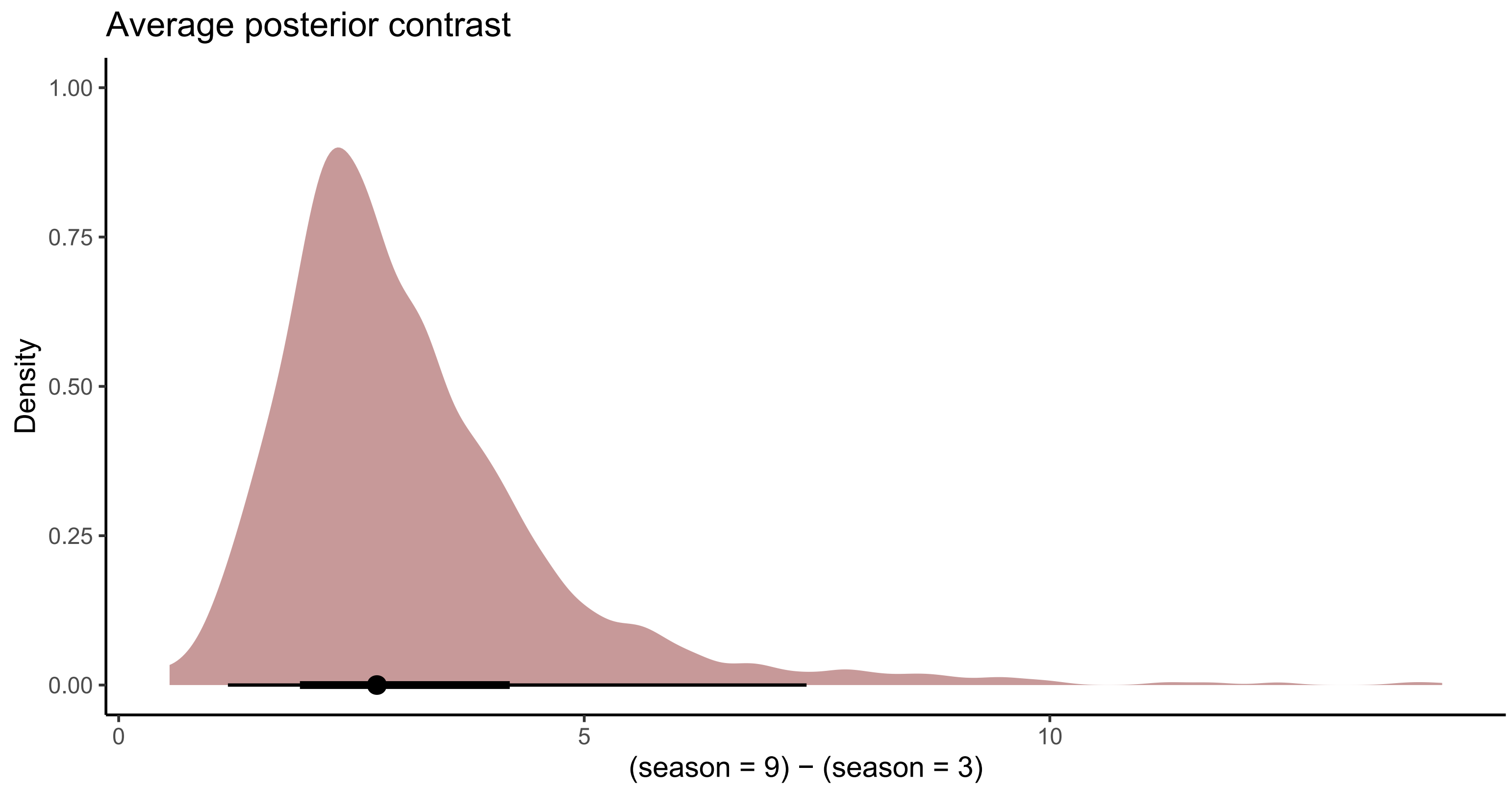
Code Plot

```r
# take draws of average comparison between season = 9 vs season = 3
post_contrasts ← avg_comparisons(model,
                                  variables = list(season = c(9, 3)),
                                  proces_error = FALSE) %>%
  posteriordraws()

# use the resulting posterior draw object to plot a density of the
# posterior contrasts
library(tidybayes)
post_contrasts %>% ggplot(aes(x = draw)) +
  # use the stat_halfeye function from tidybayes for a nice visual
  stat_halfeye(fill = "#C79999") +
  labs(x = "(season = 9) - (season = 3)", y = "Density",
       title = "Average posterior contrast") + theme_classic()
```

Posterior contrasts

Code Plot



The ability to readily interpret models from `mvgam` and `brms` 📦 's is a *huge advantage* over traditional time series models. See [my blogpost on interpeting GAMs for more examples]

But this is a forecasting course. So how can we evaluate forecast distributions?

The forecasting workflow

" *The accuracy of forecasts can only be determined by considering how well a model performs on new data that were not used when fitting the model.*" Hyndman and Athanasopoulos

We must evaluate on data that was not used to train the model (i.e. ***leave-future-out cross-validation***) because:

- Models that fit training data well do not always provide good forecasts
- We can easily engineer a model that perfectly fits the training data, leading to overfitting
- See the `mvgam` forecasting vignette for more guidance

Leave-future-out CV

Important to train the model on some portion of data and use a hold-out portion (test data) to evaluate forecasts:

$$p\left(y_{T+H} \middle| y_{1:T}\right)$$

Some points to consider:

- The test set should ideally be at least as large as the maximum forecast horizon required for decision-making
- Ideally, this process would be repeated many times to incorporate variation in forecast performance
- Usually good to compare models against simpler ***benchmark*** models to ensure added complexity improves forecasts

We must obtain leave-future-out forecasts (ideally for many different training / testing splits) to compare ecological forecasting models

But how do we *evaluate* forecasts?

The most common evaluation practice in forecasting tasks is to evaluate point predictions

Point-based forecast evaluation

Forecast errors

A forecast error (or forecast residual) is the difference between the true value in an out-of-sample set and the predicted response value:

$$\epsilon_{T+H} = \boldsymbol{y}_{T+H} - \hat{y}_{T+H}$$

Where:

T is the total length of the training set

H is the forecast horizon

$\hat{y}_{T+H}$ is the prediction at time $T + H$

Point-based measures use these errors in different ways

Common point-based measures

Scale-dependent measures

Mean Absolute Error: $mean(|\epsilon_t|)$

Root Mean Squared Error: $\sqrt{mean(\epsilon_t^2)}$

Scale-independent measures

Mean Absolute Percentage Error: $mean(|p_t|)$, where $p_t = 100\epsilon_t/y_t$

Mean Absolute Scaled Error: $mean(|q_t|)$, where q_t is the error scaled against errors from an appropriate ***benchmark*** forecast

Lower values are better for all these measures

We won't dwell much on point-based measures because ecological predictions and their associated management decisions are inherently *uncertain* (but see this video for more details)

Point-based measures ignore far too much information in the forecast distribution

It is better to evaluate the *entire forecast distribution*

Live code example

Probabilistic forecast evaluation

Scaled Interval Score

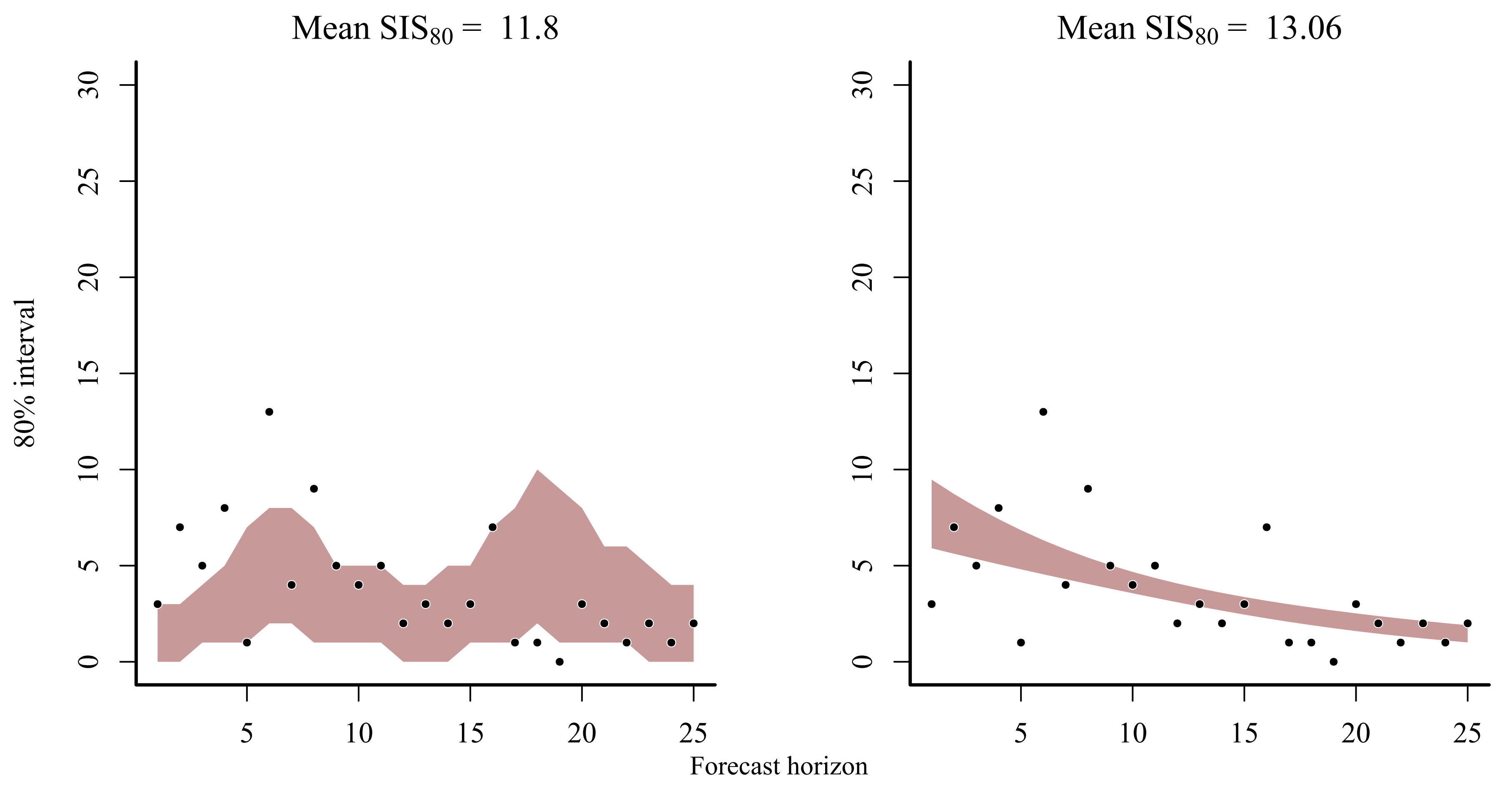
A common step to evaluate a forecast distribution is to compute how well it's prediction intervals perform:

$$SIS = (U_t - L_t) + \frac{2}{\alpha}(L_t - y_t)\mathbb{1}(y_t < L_t) + \frac{2}{\alpha}(y_t - U_t)\mathbb{1}(y_t > U_t)$$

Where:

- y_t is the true observed value at horizon H
- α is $1 -$ interval width
- The $100(1-\alpha)\%$ interval for horizon H is $[L_t, U_t]$
- $\mathbb{1}$ is a binary indicator function

Penalize *overly precise* forecasts



Evaluating the full distribution

Interval scores are very useful when we want to target a particular interval or if we don't have the full distribution

- Allows different teams to submit a few intervals rather than thousands of posterior samples
- Can compare forecasts from many different algorithms / models

But if we do have a full distribution, we have other options

"*Scoring rules provide summary measures for the evaluation of probabilistic forecasts, by assigning a numerical score based on the predictive distribution and on the event or value that materializes*" (Gneiting and Raftery 2007)

What is a good forecast?

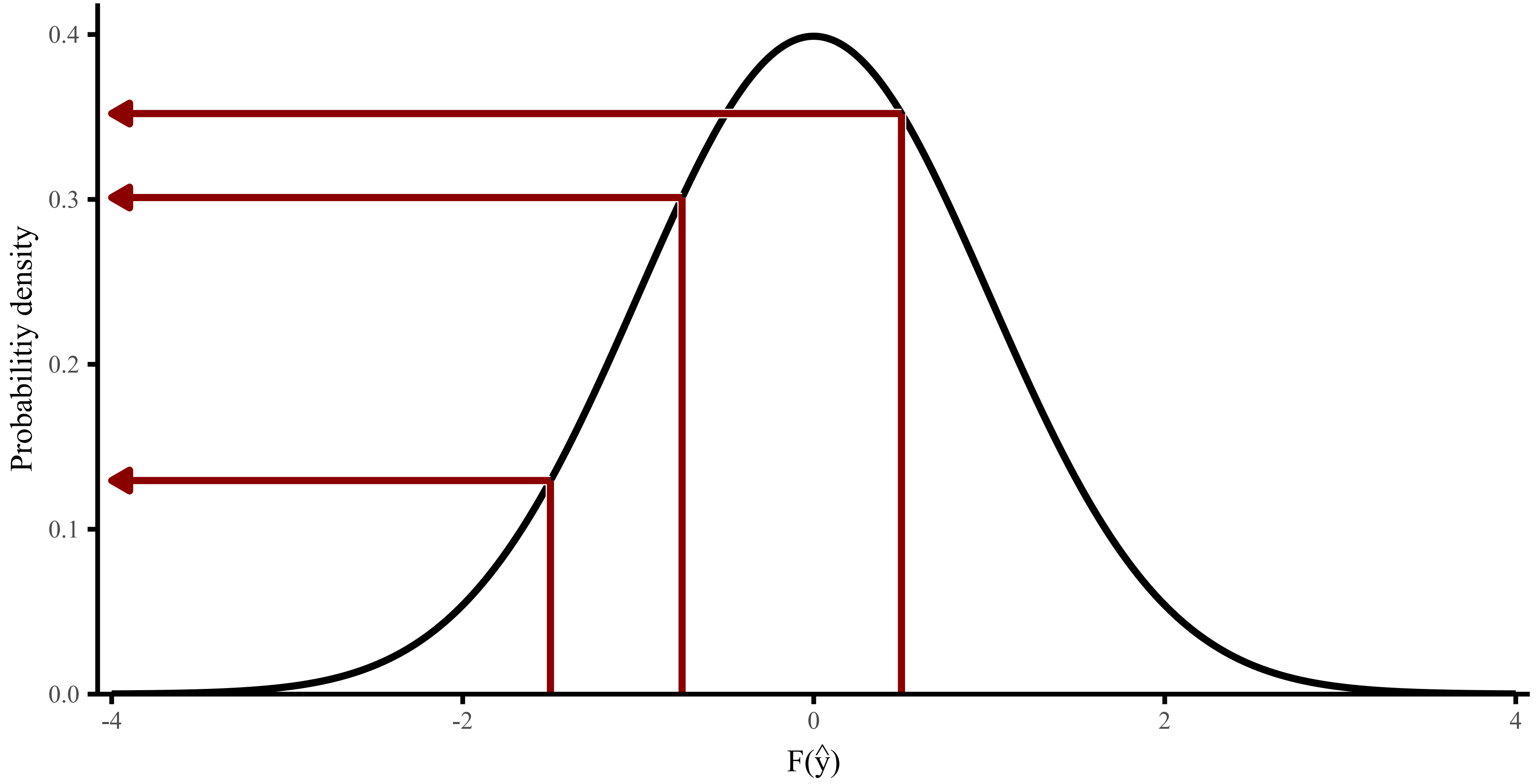
Reliable: good probabilistic calibration

Sharp: informative, with tight enough intervals to guide decisions

Skilled: performs better overall than simpler benchmark forecasts

Proper scoring rules attempt to address each of these goals using the full forecast distribution

Predictive density



Log predictive density

Compute *log(probability)* of a given truth given distributional assumptions:

$$log\, p(y_{T+H} | y_{t:T}, \theta)$$

Use density functions in R, such as `dnorm` or `dnbinom`; higher values are better

θ captures all unknown parameters:

- Regression coefficients β
- Dynamic parameters; α or ρ for GP; σ_{error} for RW
- Observation parameters; ν for StudentT or σ_{obs} for Normal

logging is stabile and makes joint calculations easier

But the log score can severly penalize over-confidence and is sensitive to outliers

Other proper scoring rules can provide more robust comparisons, without needing to rely on distributional assumptions

CRPS

Continuous Ranked Probability Score compares true Cumulative Distribution Function (CDF) to forecast CDF

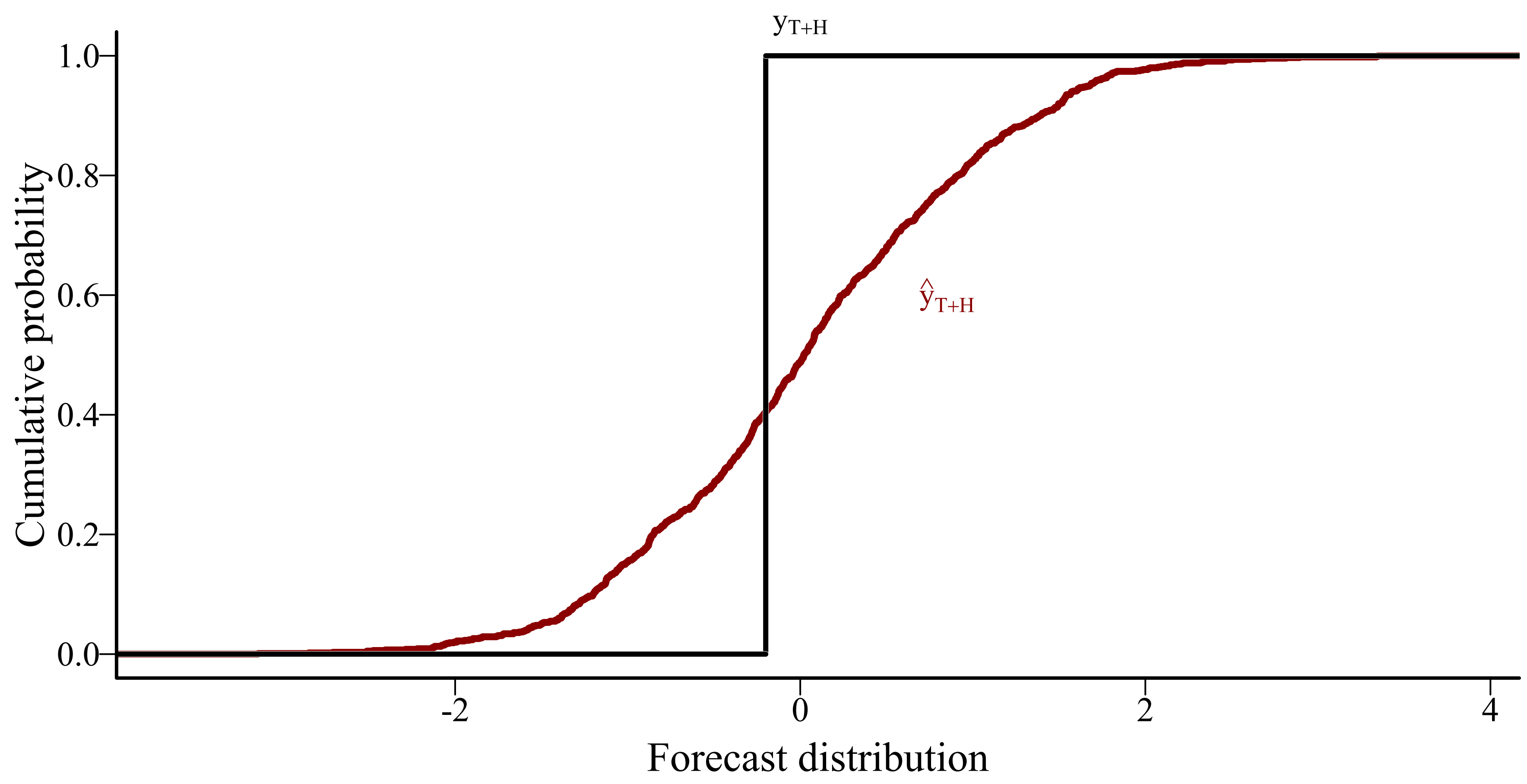
$$CRPS(F, y) = \int_{-\infty}^{\infty} (F(\hat{y}) - \mathbb{1}(\hat{y} \geq y))^2 dy$$

Where:

- $F(\hat{y})$ is the forecast CDF evaluated at many points
- $\mathbb{1}(\hat{y} \geq y)$ gives the true observed CDF

SIS converges to CRPS when evaluating an increasing number of equally spaced intervals

CRPS



y_{T+H}
$\hat{y}_{T+H}$
1.0
0.8
0.6
0.4
0.2
0.0
Cumulative probability
-2
0
2
4
Forecast distribution

CRPS useful for both parametric and non-parametric predictions because we just need to calculate the CDF of the forecast distribution

Penalises over- and under-confidence similarly, and gives more stable handling of outliers

Score is on the scale of the outcome variable being forecasted, so is somewhat intuitive (a lower score is better)

DRPS

Similar to CRPS, the discrete version (DRPS) can be used to evaluate a forecast that is composed only of integers

Uses an approximation of the forecast and true CDFs at a range of possible count values

Interpretation is similar

`score.mvgam_forecast()`

Once forecasts are computed and stored in an object of class `mvgam_forecast`, scores can be directly applied

User chooses among the Scaled Interval Score (`sis`), log score (`elpd`), CRPS (`crps`), DRPS (`drps`) and two multivariate scores (`energy` or `variogram`; more on this in the next lecture)

User also specifies an interval for calculating coverage and/or which interval to use for the Scaled Interval Score

`return` is a `list()` with scores for each series in the data and an overall score (usually just the sum of series-level scores)

```r
sc <- score(forecast(model),
            score = 'crps',
            interval = 0.90)
sc$series_1[1:10,]
```

```
##        score in_interval interval_width eval_horizon score_type
## 1  0.9483460           1            0.9            1       crps
## 2  4.2652035           0            0.9            2       crps
## 3  2.0320750           1            0.9            3       crps
## 4  4.1233913           0            0.9            4       crps
## 5  1.6279930           1            0.9            5       crps
## 6  6.4760912           0            0.9            6       crps
## 7  0.6714025           1            0.9            7       crps
## 8  3.8893157           1            0.9            8       crps
## 9  1.4326503           1            0.9            9       crps
## 10 0.8742835           1            0.9           10       crps
```

Calculating the CRPS using the previously generated forecasts

```
sc ← score(forecast(model),
           score = 'sis',
           interval = 0.90)
sc$series_1[1:10,]
```

```
##    score in_interval interval_width eval_horizon score_type
## 1      4           1            0.9            1        sis
## 2     45           0            0.9            2        sis
## 3      6           1            0.9            3        sis
## 4     27           0            0.9            4        sis
## 5      9           1            0.9            5        sis
## 6     50           0            0.9            6        sis
## 7     10           1            0.9            7        sis
## 8      9           1            0.9            8        sis
## 9      8           1            0.9            9        sis
## 10     8           1            0.9           10        sis
```

Calculating the SIS using the previously generated forecasts; values outside interval are more heavily penalized

We have seen how to produce out-of-sample forecasts from `mvgam` models and evaluate them against new observations

We have also investigated other ways that models can be critiqued, particularly making use of conditional predictions using `newdata`

But so far we have only considered univariate investigations. What happens if we want to forecast *multiple time series*?

In the next lecture, we will cover

Multivariate ecological time series

Vector autoregressive processes

Dynamic factor models

Multivariate forecast evaluation