Ecological forecasting in R

Lecture 3: latent AR and GP models

Nicholas Clark

School of Veterinary Science, University of Queensland

0900–1200 CET Tuesday 28th 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 R script from the html file
- Complete exercises and use Slack to ask questions

Relevant open-source materials include:

- Introduction to Generalized Additive Models with R and `mgcv`
- Temporal autocorrelation in Generalized Additive Models
- Statistical Rethinking 2023 - 16 - Gaussian Processes

This lecture's topics

Extrapolating splines

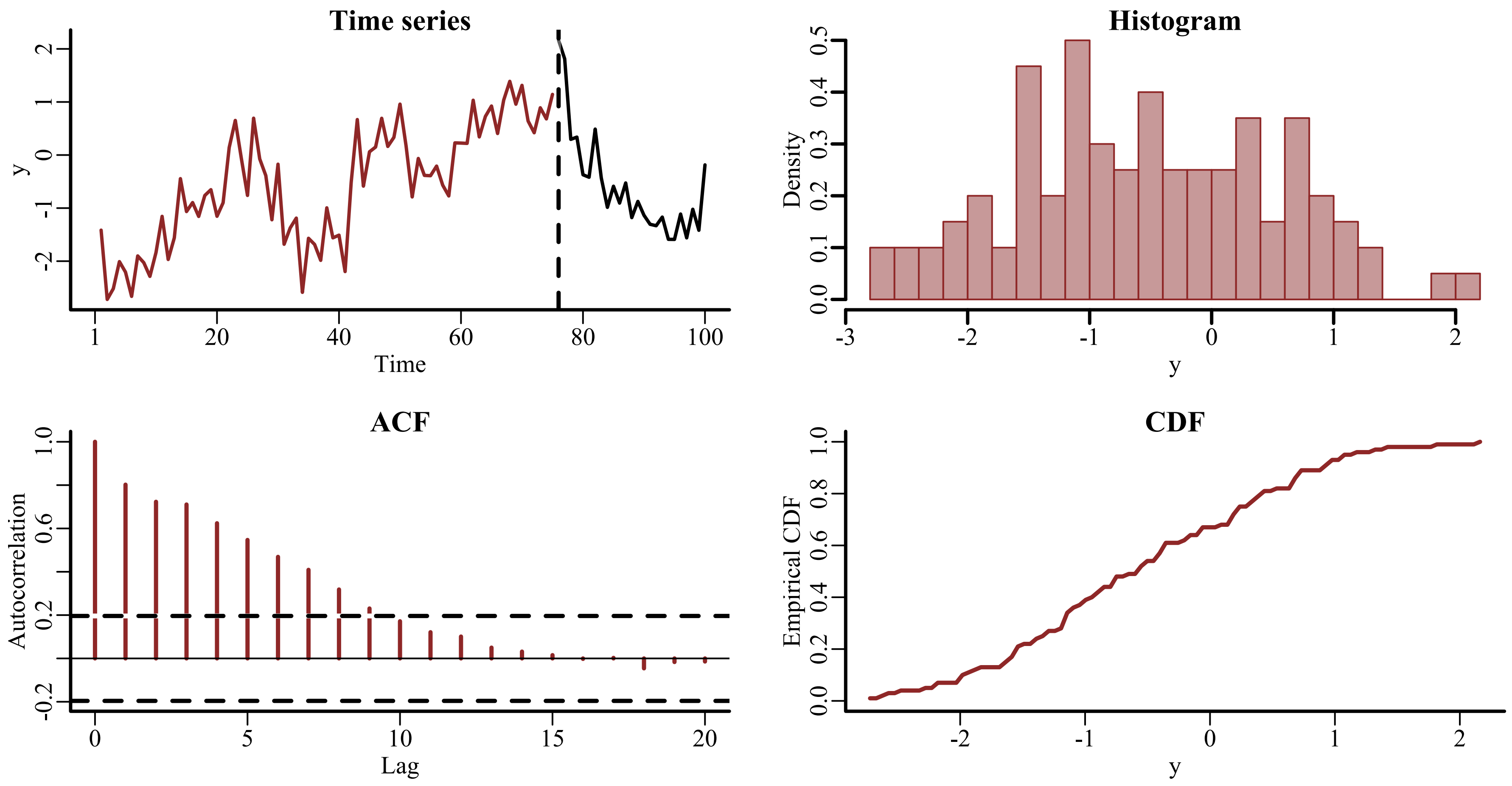
Latent autoregressive processes

Latent Gaussian Processes

Dynamic coefficient models

Extrapolating splines

Simulated data



A spline of `time`

```
library(mvgam)
model <- mvgam(y ~
                 s(time, k = 20, bs = 'bs', m = 2),
               data = data_train,
               newdata = data_test,
               family = gaussian())
```

A B-spline (`bs = 'bs'`) with `m = 2` sets the penalty on the second derivative

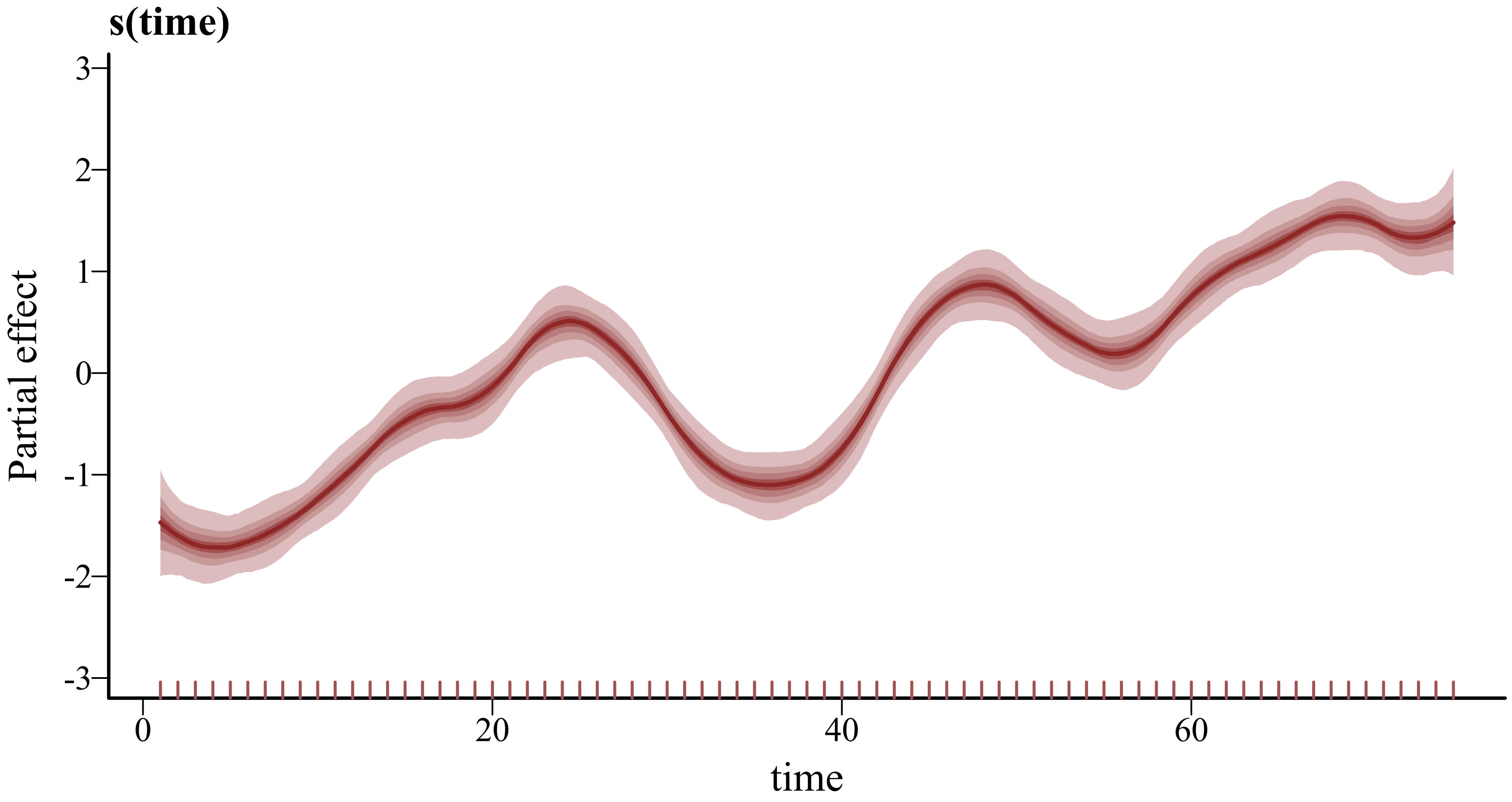
A spline of `time`

```r
library(mvgam)
model <- mvgam(y ~
                s(time, k = 20, bs = 'bs', m = 2),
               data = data_train,
               newdata = data_test,
               family = gaussian())
```

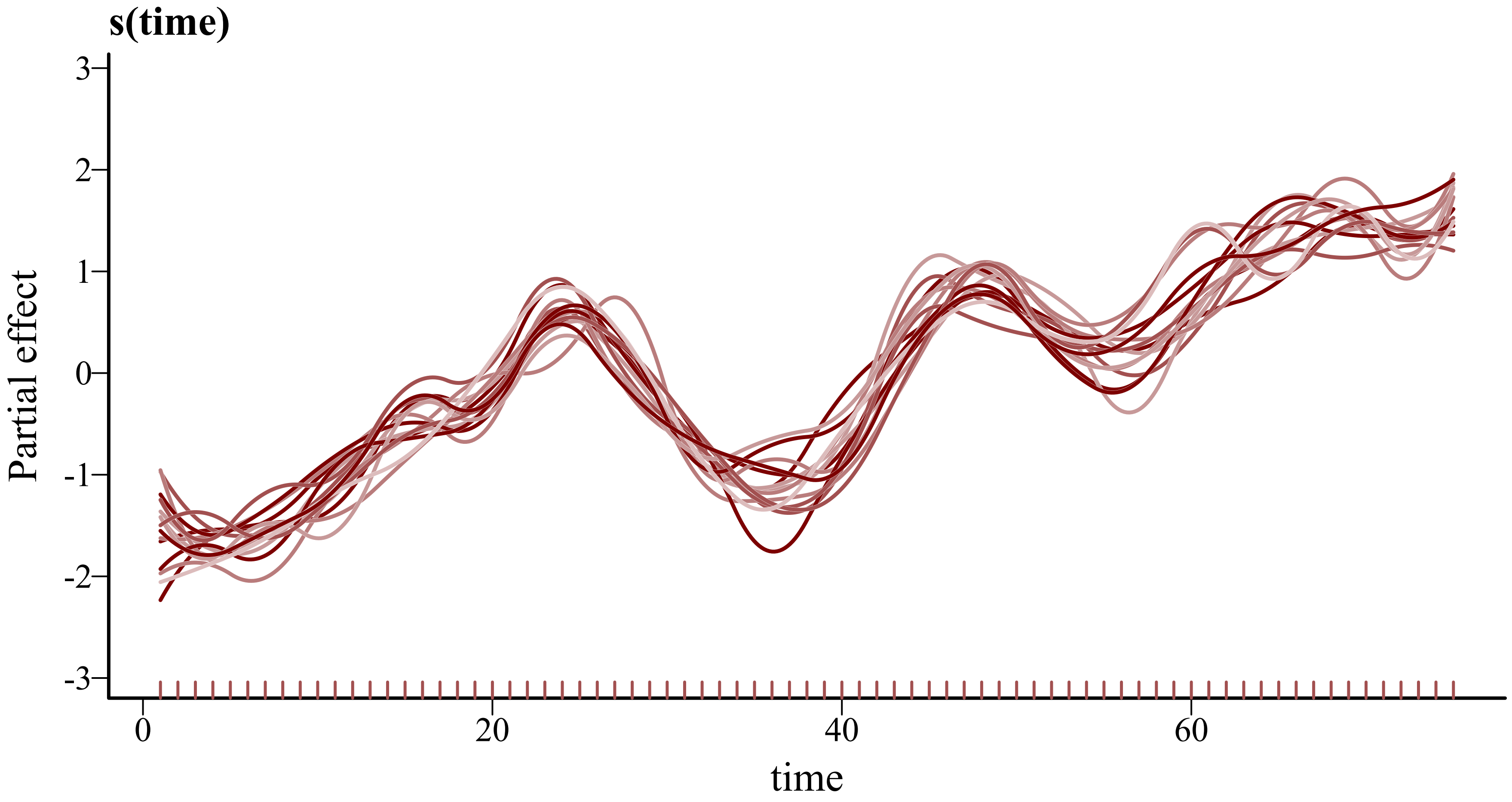
A B-spline (`bs = 'bs'`) with `m = 2` sets the penalty on the second derivative

Use `newdata` argument to generate automatic probabilistic forecasts

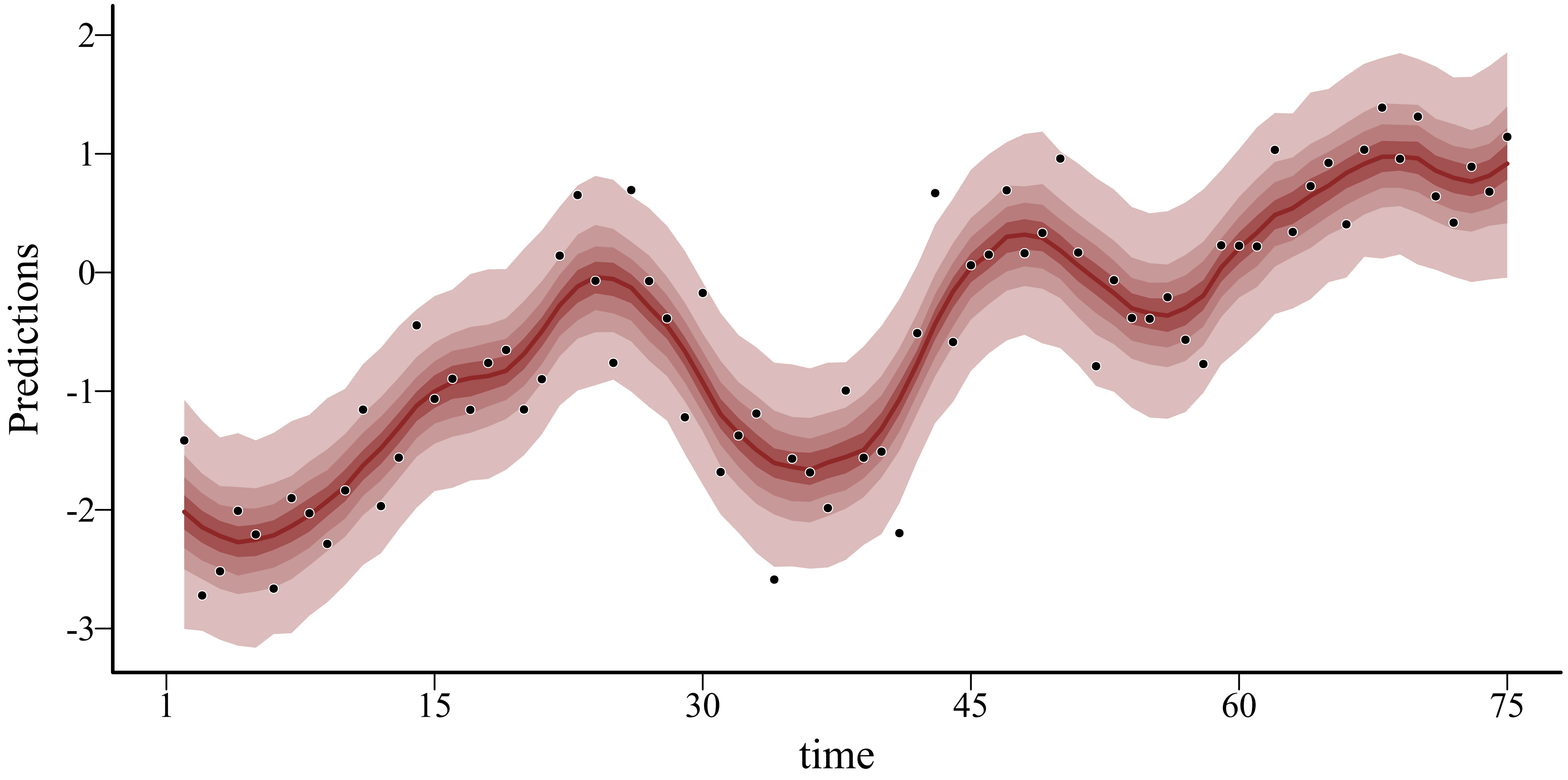
The smooth function



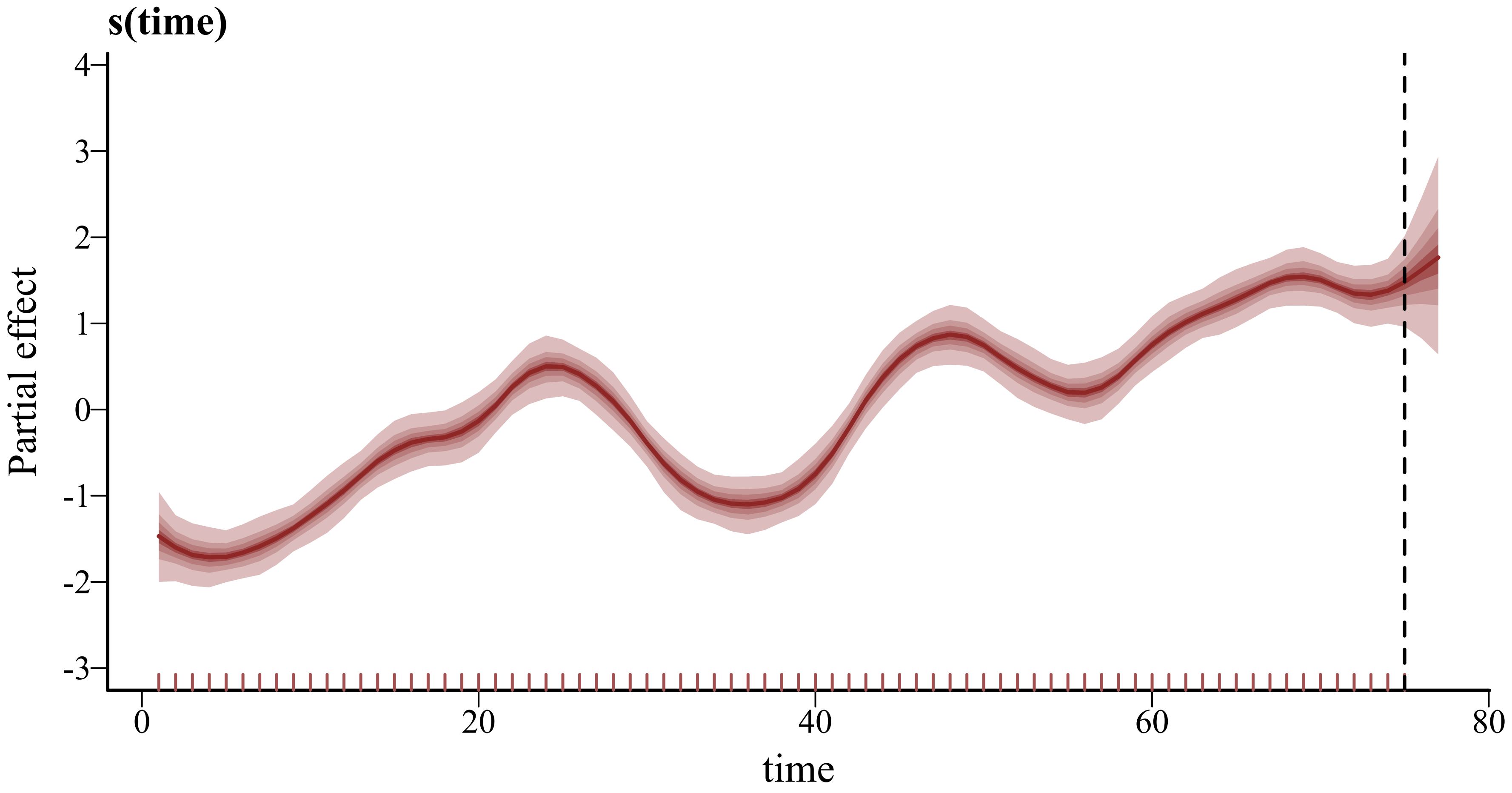
Realizations of the function



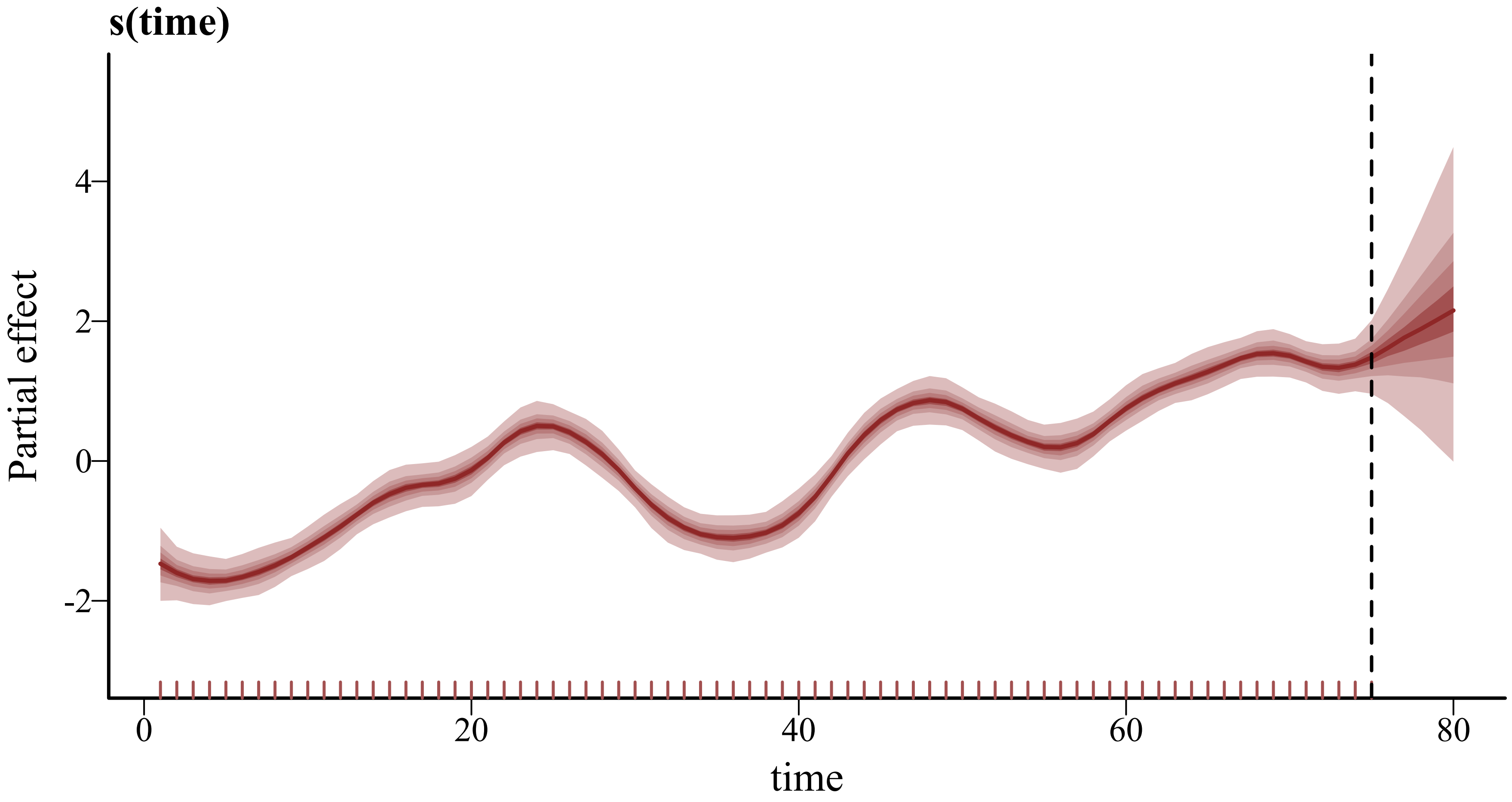
Hindcasts 😊



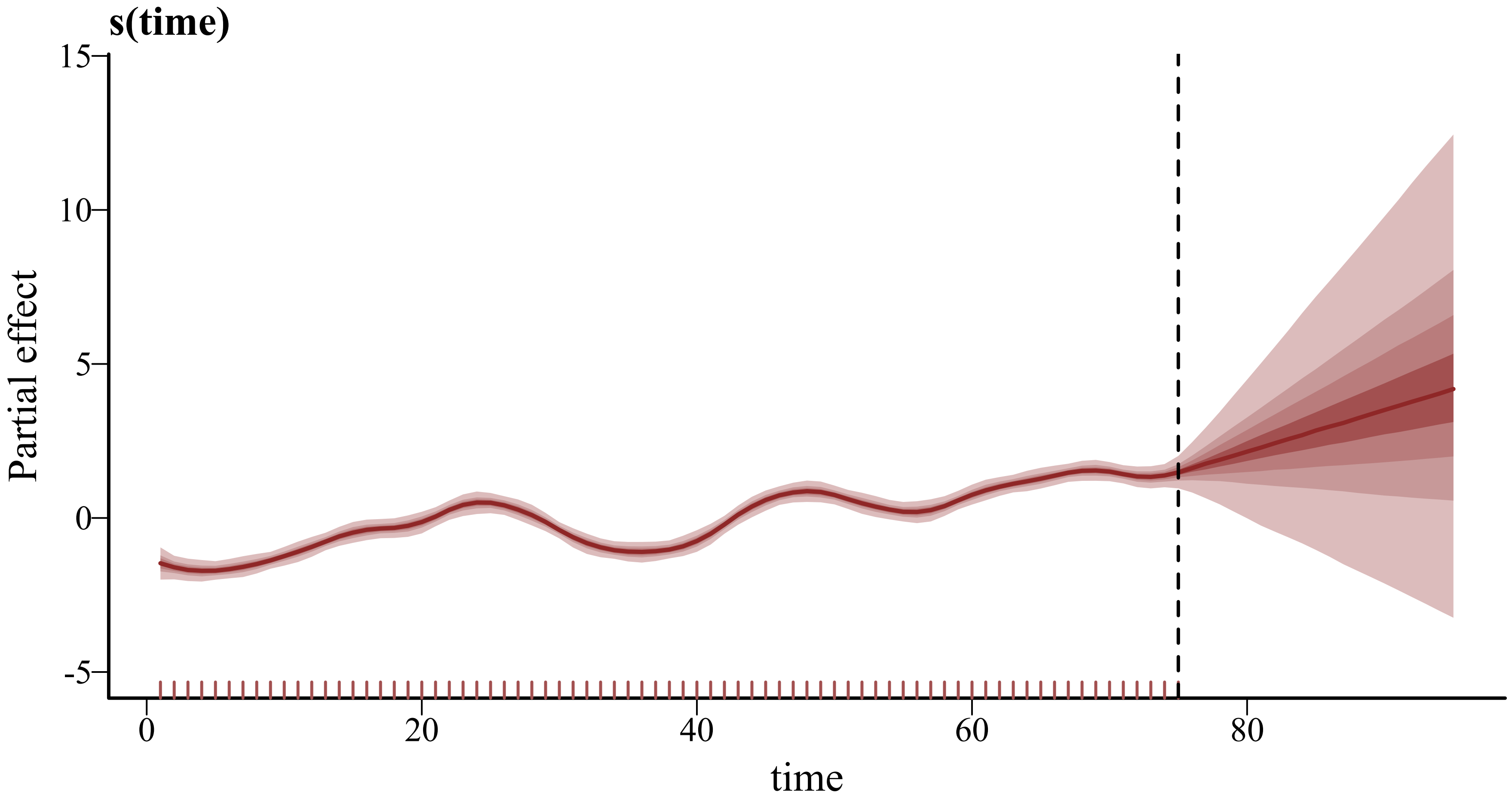
Extrapolate 2-steps ahead ☺



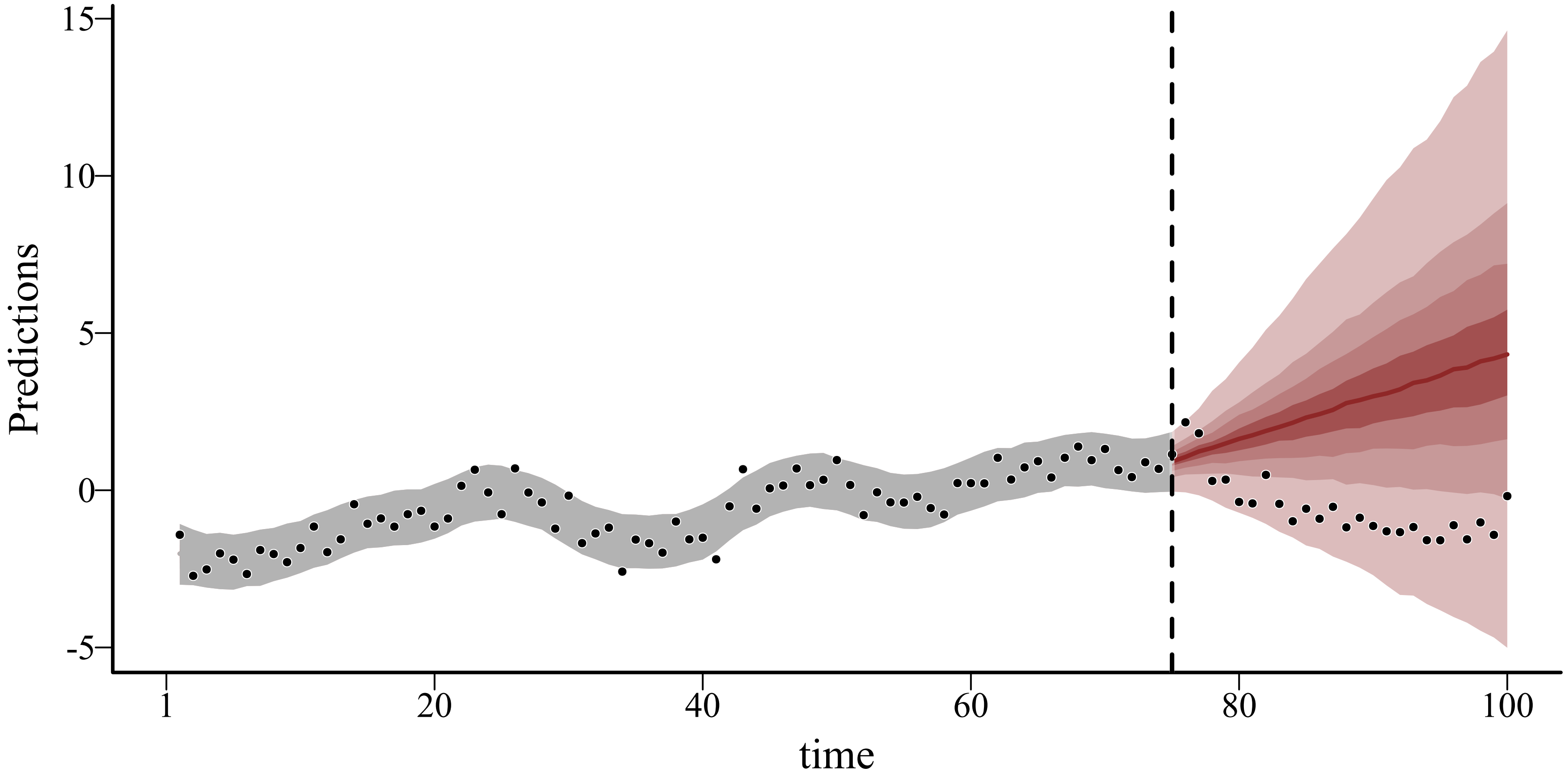
5-steps ahead ☹



20-steps ahead 😥



Forecasts 😨



2nd derivative penalty

Penalizes the overall ***curvature*** of the spline

This is default behaviour in 📦 's `mgcv`, `brms` and `mvgam`

Provides linear extrapolations

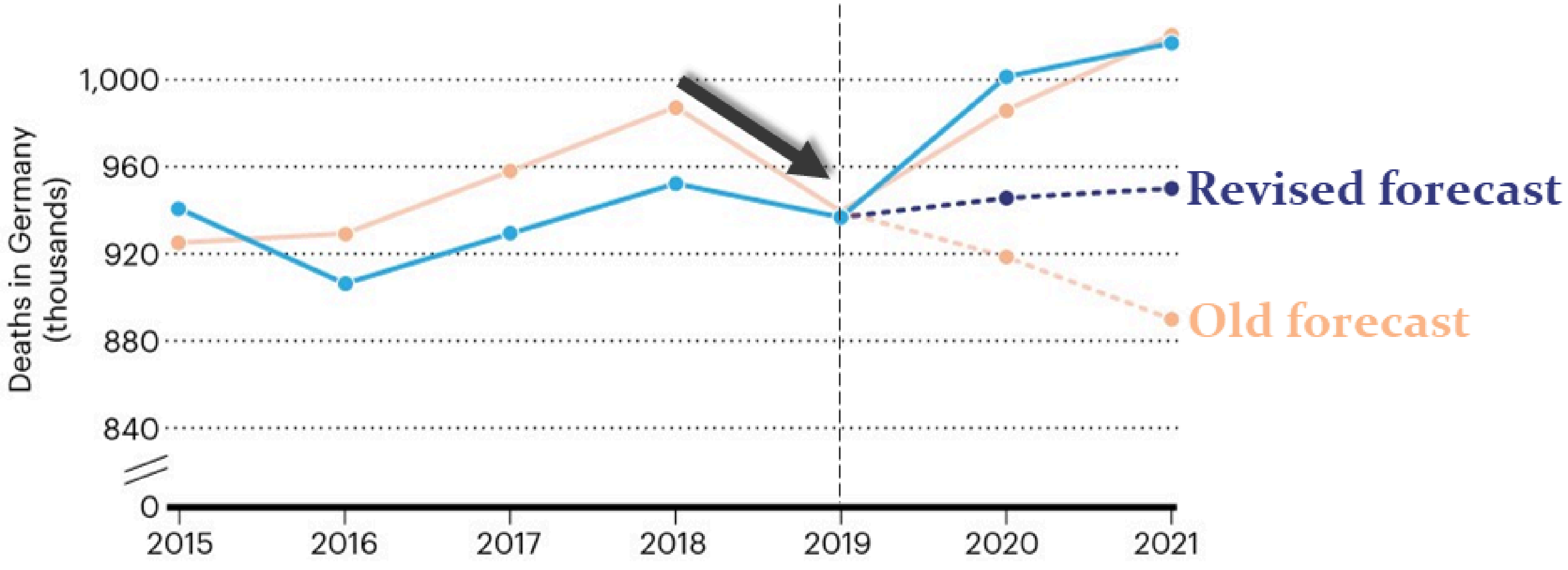
- Slope remains unchanged from the last boundary of training data
- Uncertainty grows but has no probabilistic understanding of time

This behaviour is widely known; ***but spline extrapolation is still commonplace***

NEWS FEATURE | 01 June 2022

COVID death tolls: scientists acknowledge errors in WHO estimates

Researchers with the World Health Organization explain mistakes in high-profile mortality estimates for Germany and Sweden.



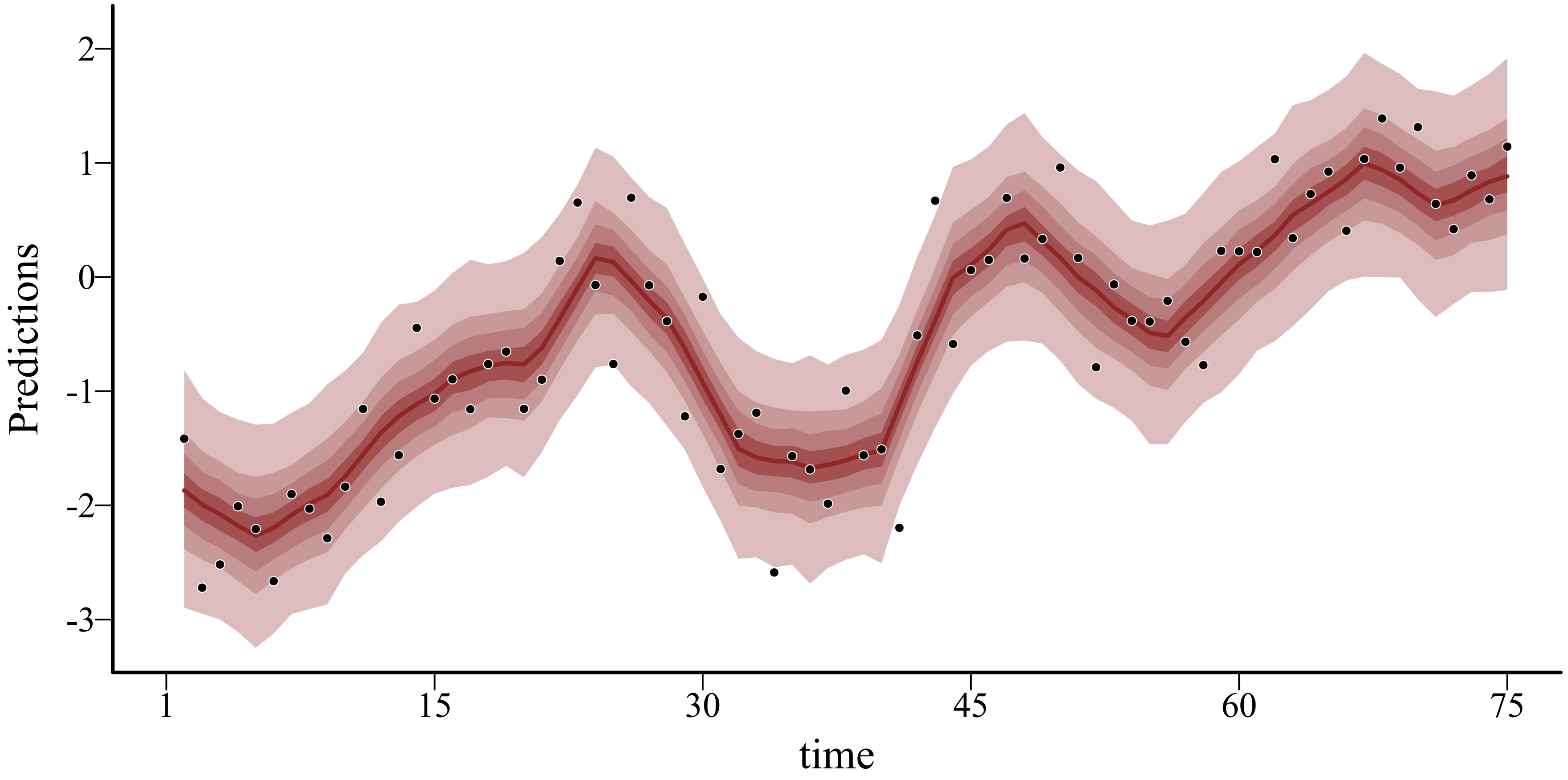
*World Health Organization. **World Mortality Dataset; dots show yearly totals. Range of uncertainty in estimates not shown, for clarity.

1st derivative penalty?

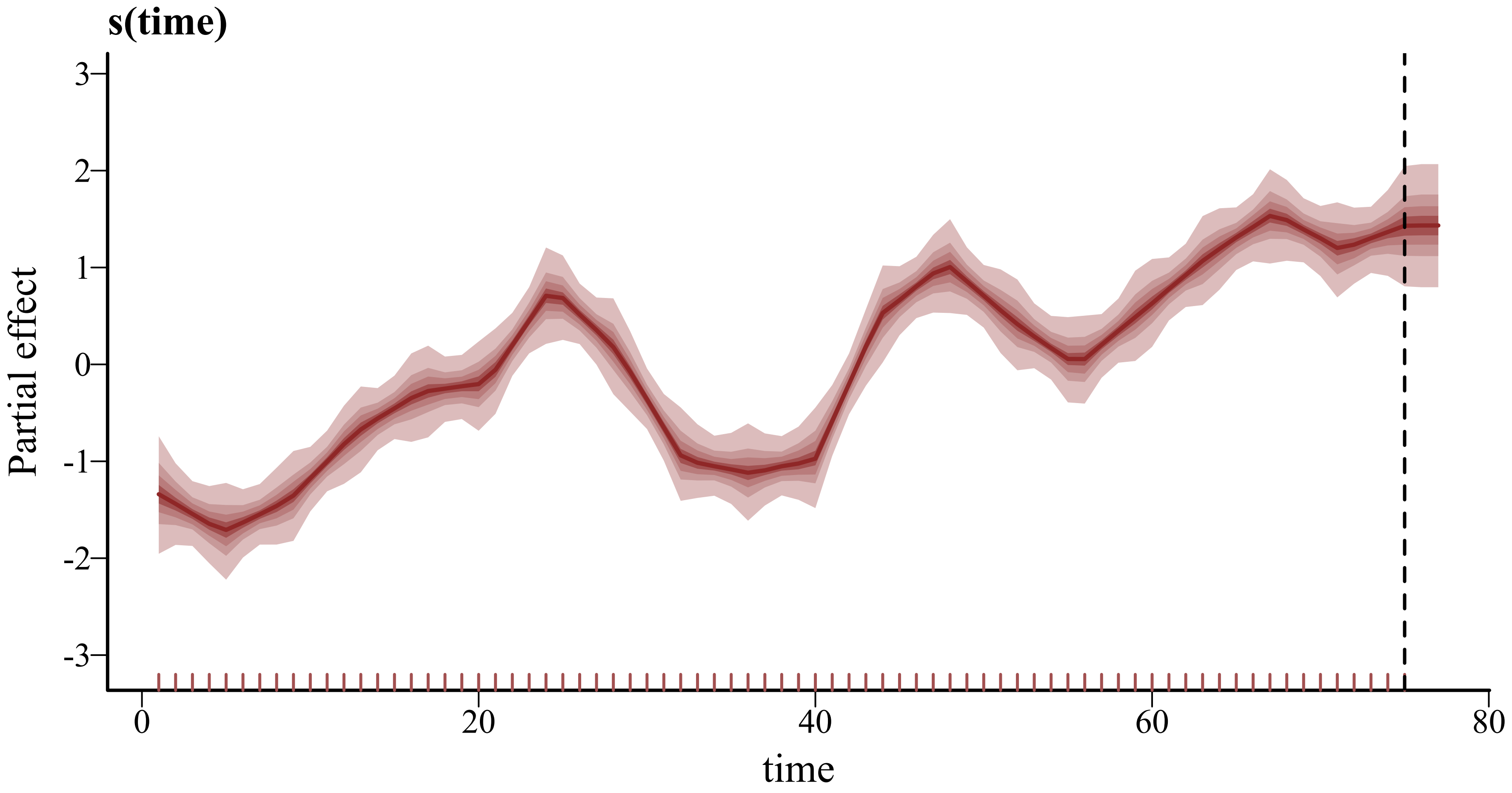
```
model <- mvgam(y ~
                 s(time, k = 20, bs = 'bs', m = 1),
               data = data_train,
               newdata = data_test,
               family = gaussian())
```

Using `m = 1` sets the penalty on the first derivative

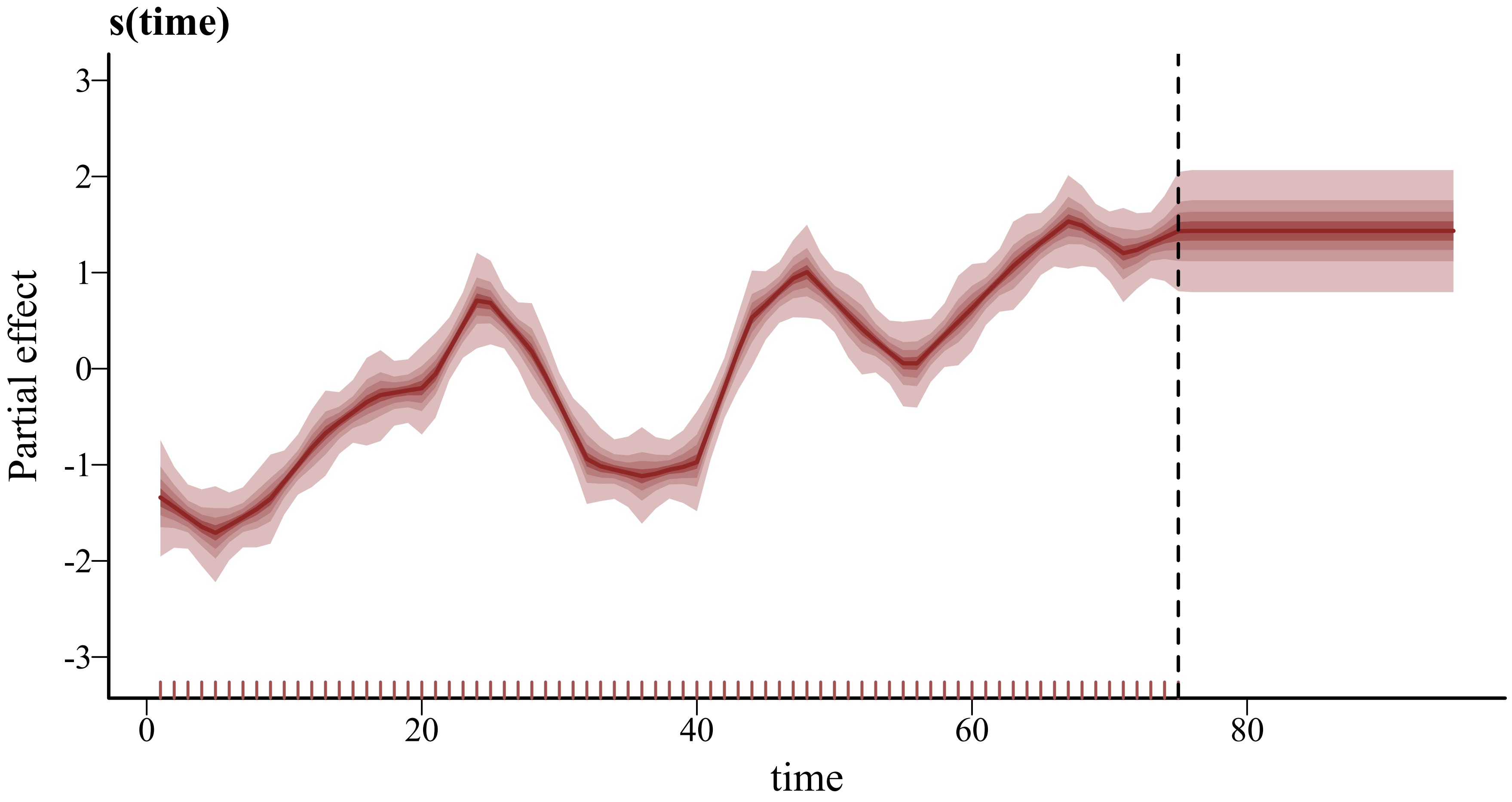
Hindcasts ☺



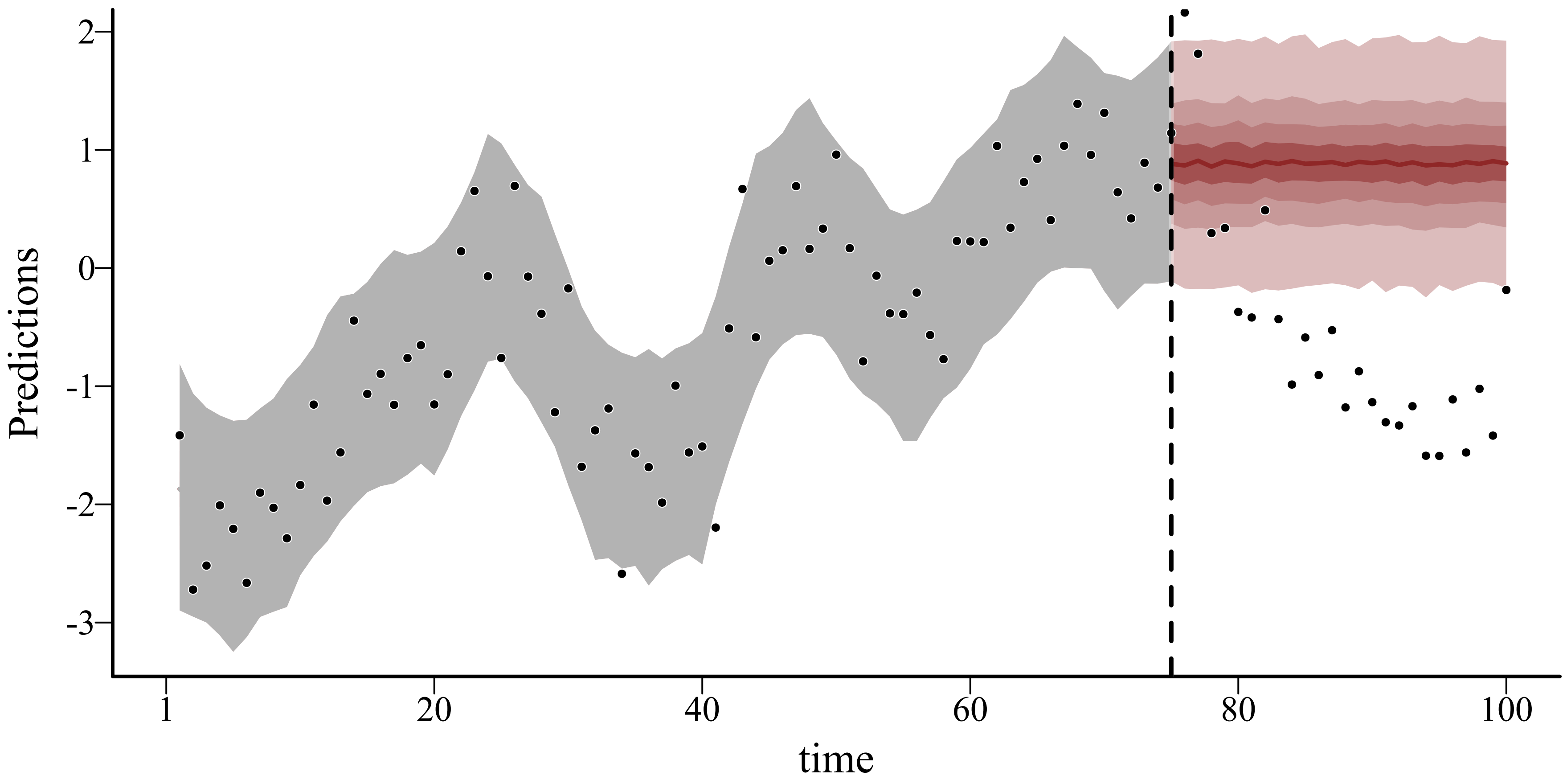
2-step ahead prediction ☺



20-steps ahead 😬



Forecasts 😳



1st derivative penalty

Penalizes deviations from a flat function

Provides flat extrapolations

- Mean remains unchanged from last boundary of the training data
- Uncertainty remains unrealistically narrow

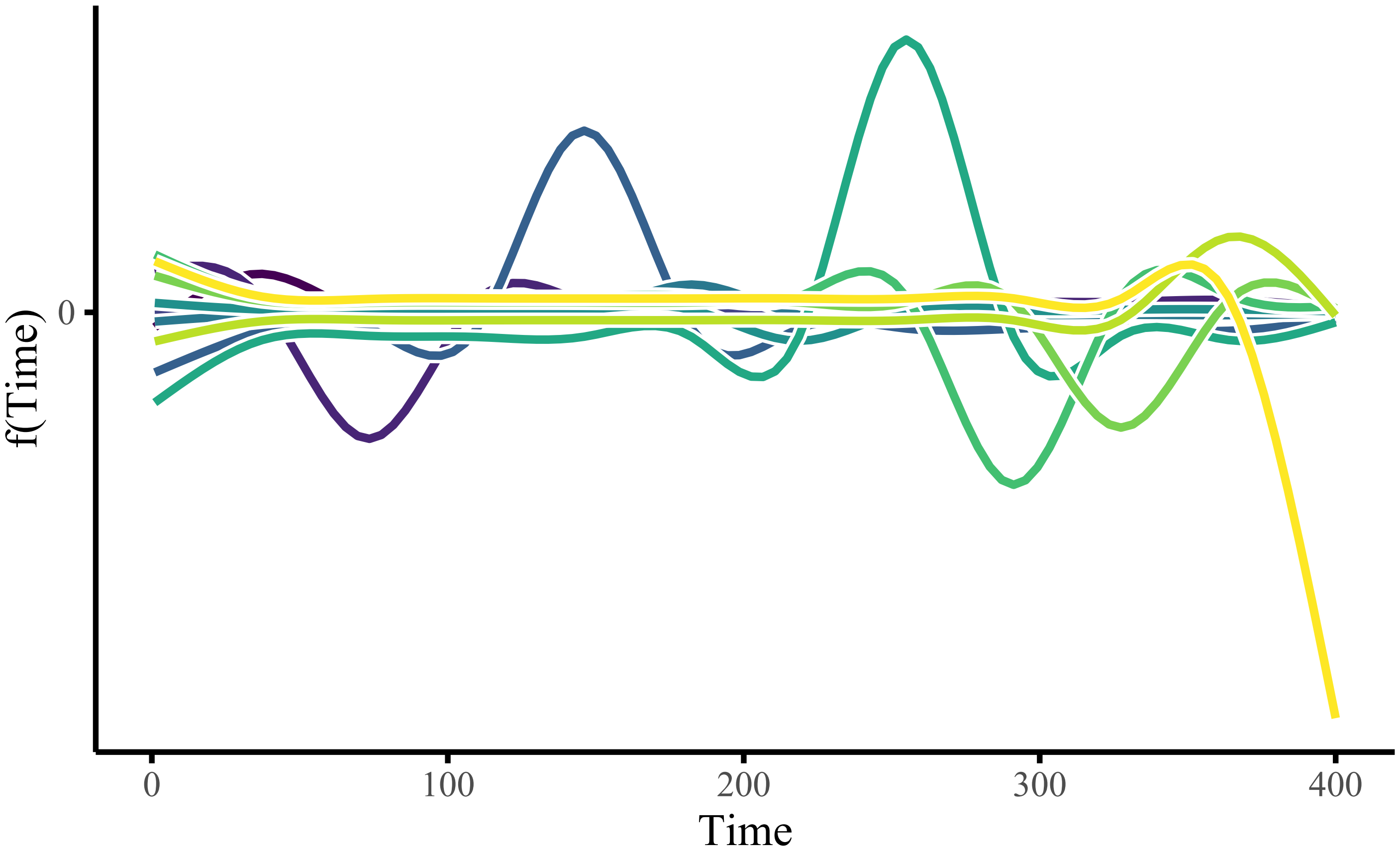
Not commonly used, though there are exceptions

Changing penalties when using splines will impact how they extrapolate

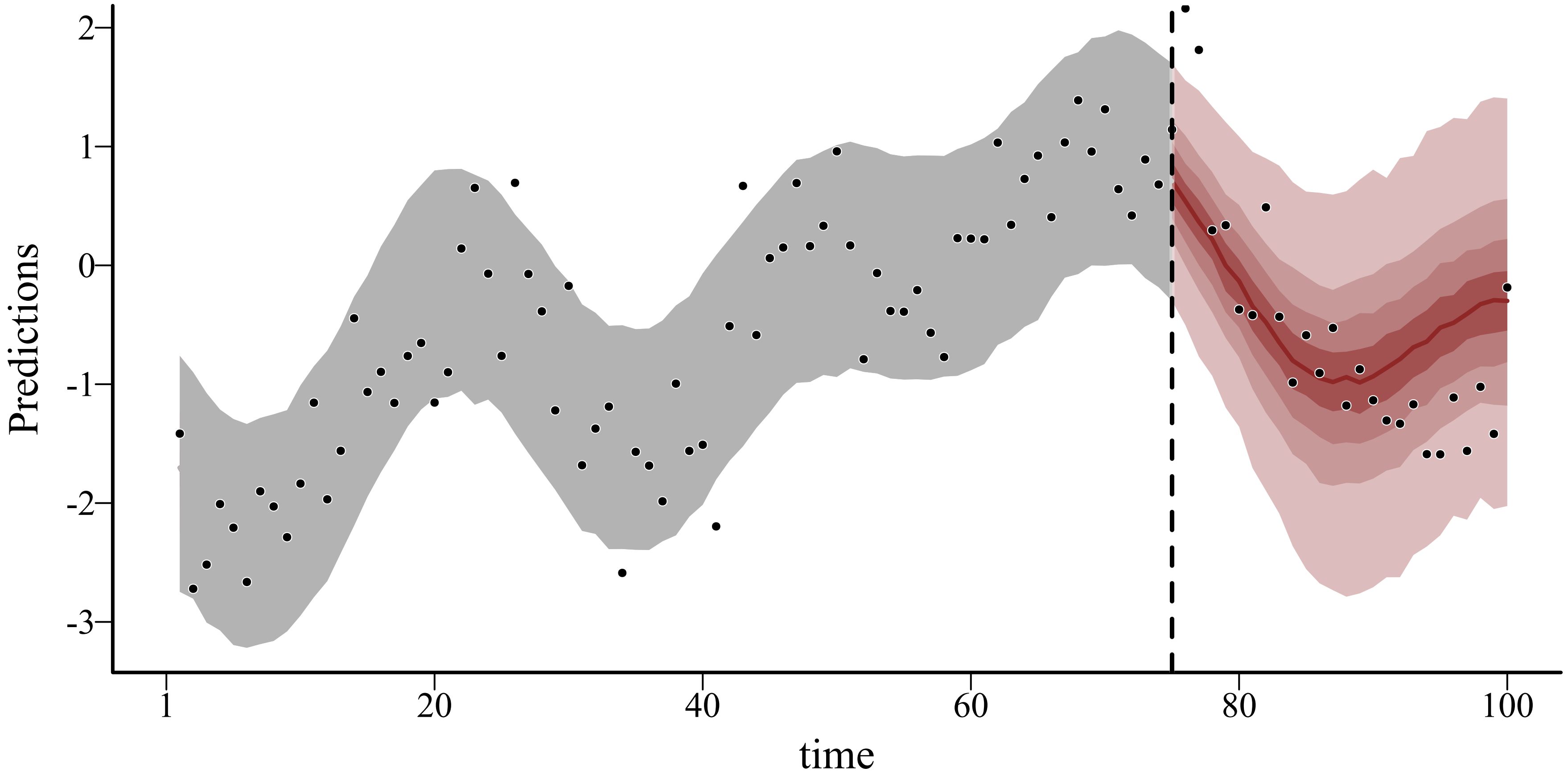
Extrapolation also reacts *strongly* to what the spline is doing at the boundaries

This is because splines only have *local knowledge*

Basis functions ⇨ local knowledge



We need *global knowledge*



First, a few other pitfalls

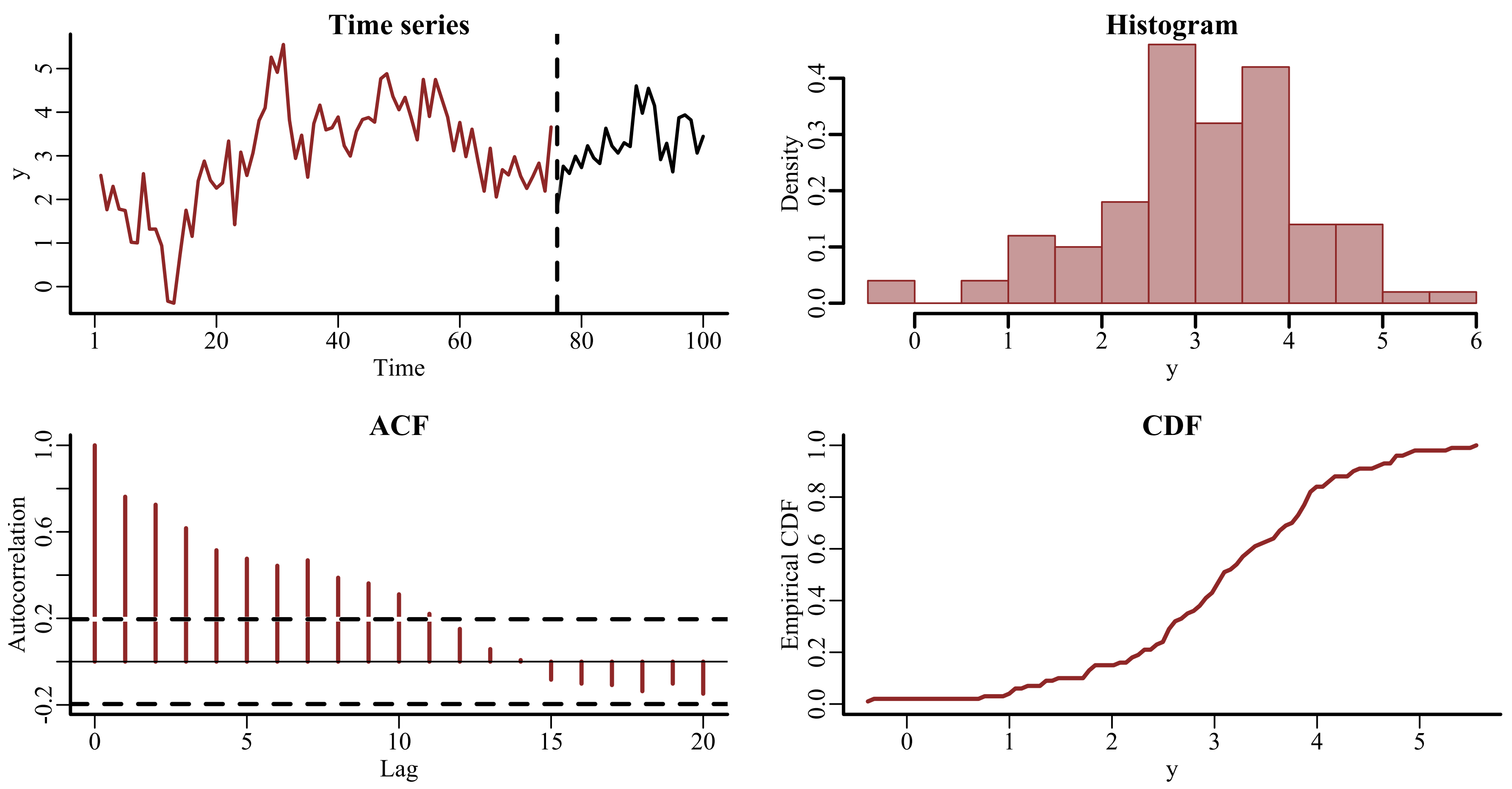
`mgcv` 📦 has a heuristic checking function (`gam.check`) to inform whether a spline is *wiggly enough*

Can be useful to understand if your functions are complex enough to capture patterns in observed data

But can also be misleading when dealing with time series

`mvgam` 📦 includes an underlying object of class `gam` that can be checked with `gam.check`

Simulated data



Restricted smooth of `time`

```
model <- mvgam(y ~ s(time, k = 6),
                 family = gaussian(),
               data = data_train,
               newdata = data_test)
```

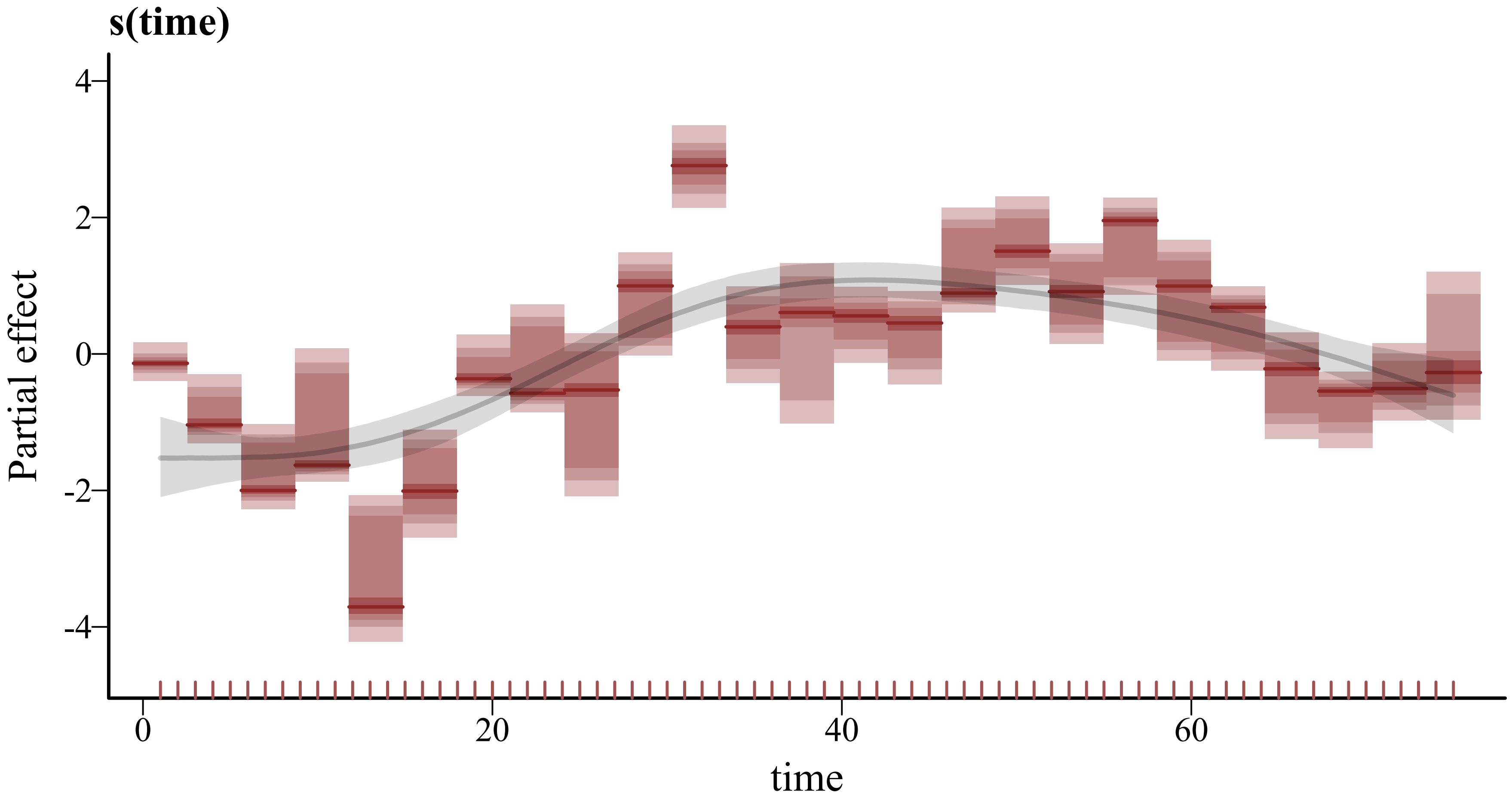
Using a thin plate spline with low maximum complexity (`k = 6`)

Check basis complexity

```
gam.check(model$mgcv_model)
```

```
##
## Method: REML   Optimizer: outer newton
## full convergence after 6 iterations.
## Gradient range [-2.516432e-07,-8.657903e-09]
## (score 94.00124 & scale 0.590227).
## Hessian positive definite, eigenvalue range [1.028413,36.58177].
## Model rank =  6 / 6
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##           k'  edf k-index p-value
## s(time) 5.00 4.41    0.55  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Unmodelled variation

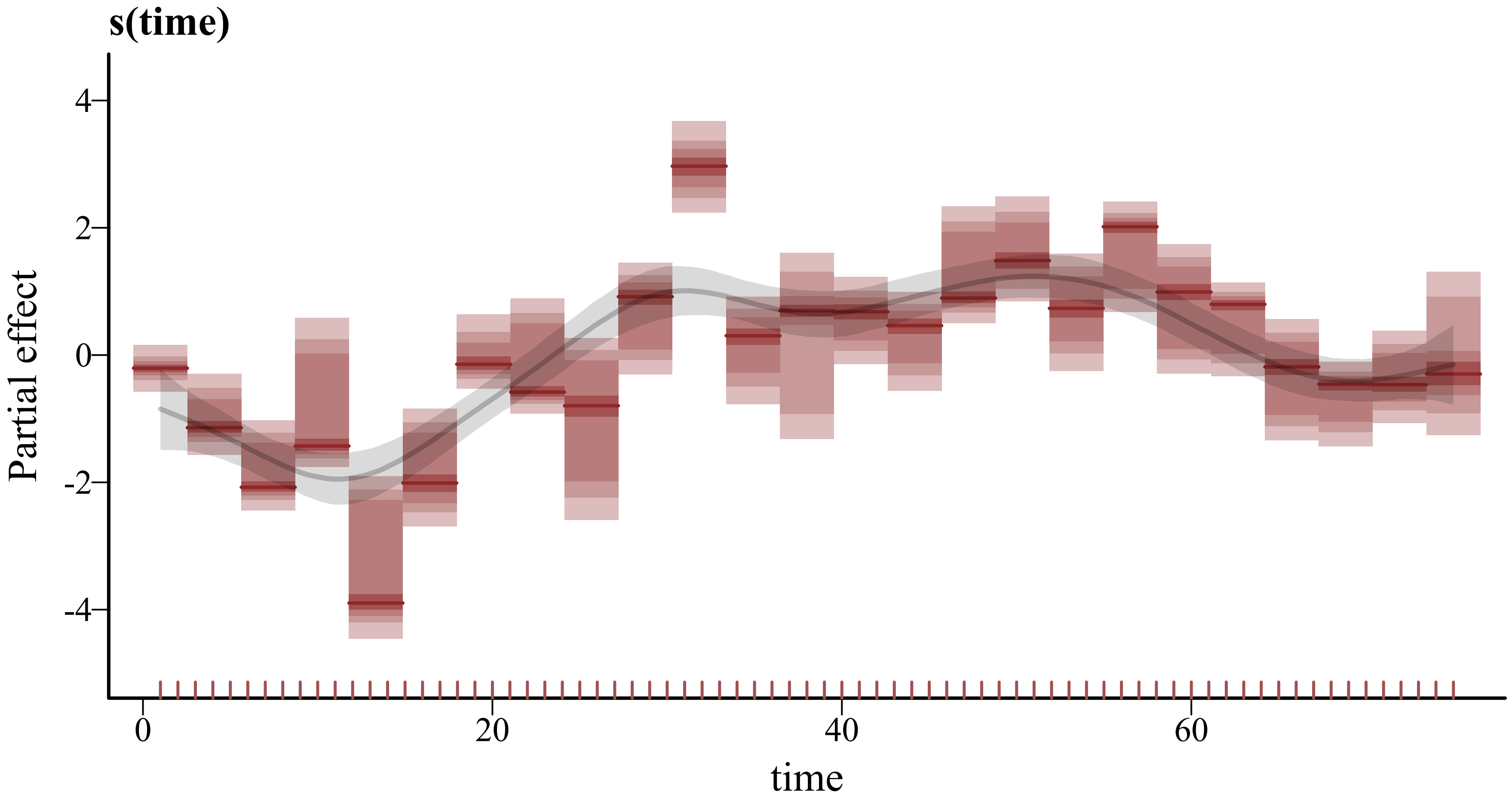


Increase complexity?

```r
model <- mvgam(y ~ s(time, k = 15), family = gaussian(),
               data = data_train, newdata = data_test)

gam.check(model$mgcv_model)
```

```
##
## Method: REML   Optimizer: outer newton
## full convergence after 5 iterations.
## Gradient range [-1.64113e-07,3.260311e-08]
## (score 86.84541 & scale 0.4085855).
## Hessian positive definite, eigenvalue range [1.993815,36.91466].
## Model rank =  15 / 15
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##            k'   edf k-index p-value
## s(time) 14.00  8.57    0.82   0.045 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Not wiggly enough

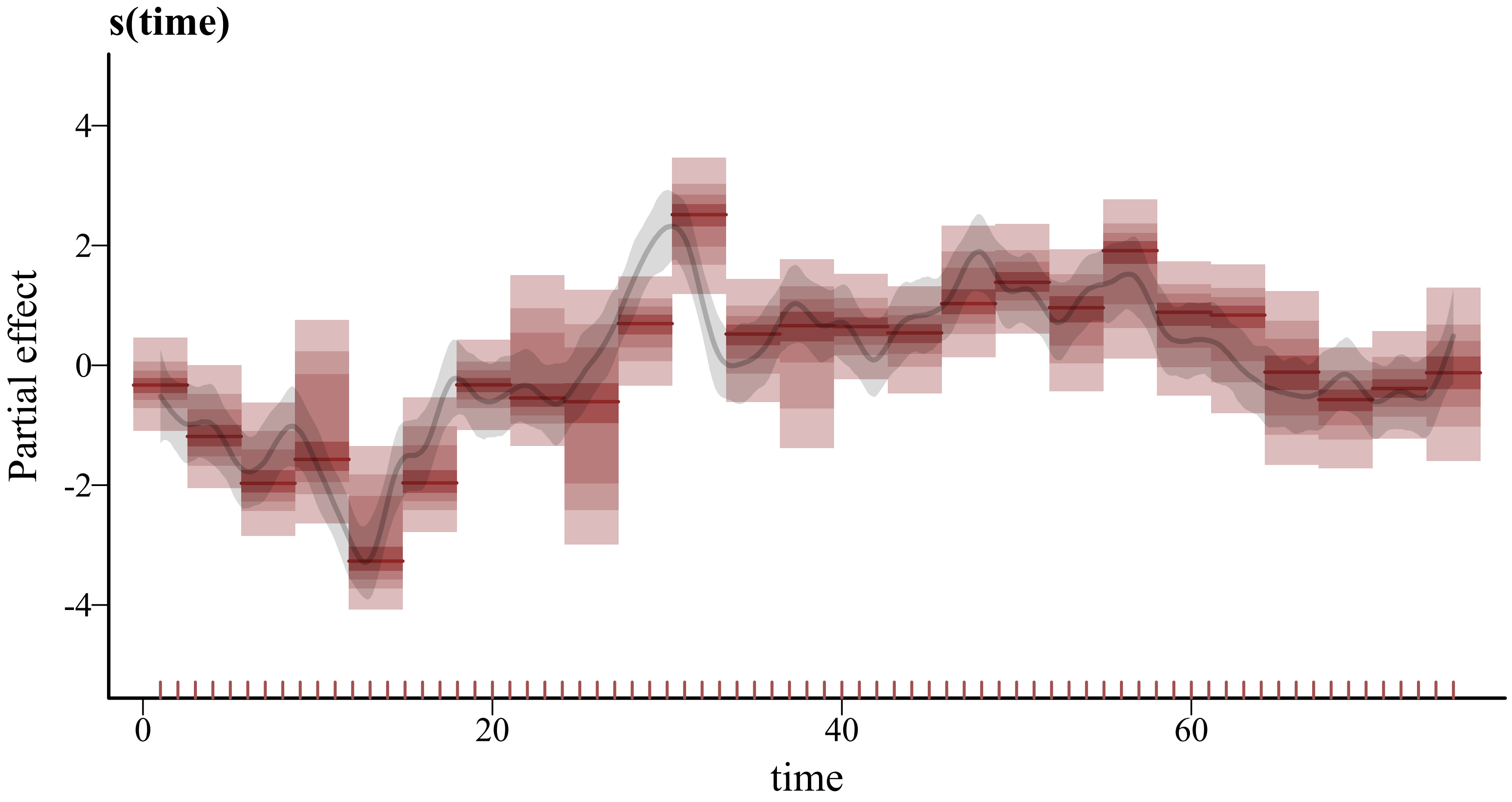


Even more complex?

```r
model <- mvgam(y ~ s(time, k = 50), family = gaussian(),
               data = data_train, newdata = data_test)

gam.check(model$mgcv_model)
```

```
##
## Method: REML   Optimizer: outer newton
## full convergence after 6 iterations.
## Gradient range [1.752317e-07,4.46345e-07]
## (score 84.14271 & scale 0.372791).
## Hessian positive definite, eigenvalue range [0.883367,37.11506].
## Model rank =  50 / 50
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##           k'  edf k-index p-value
## s(time) 49.0 10.4     0.9    0.19
```

Finally wiggly enough

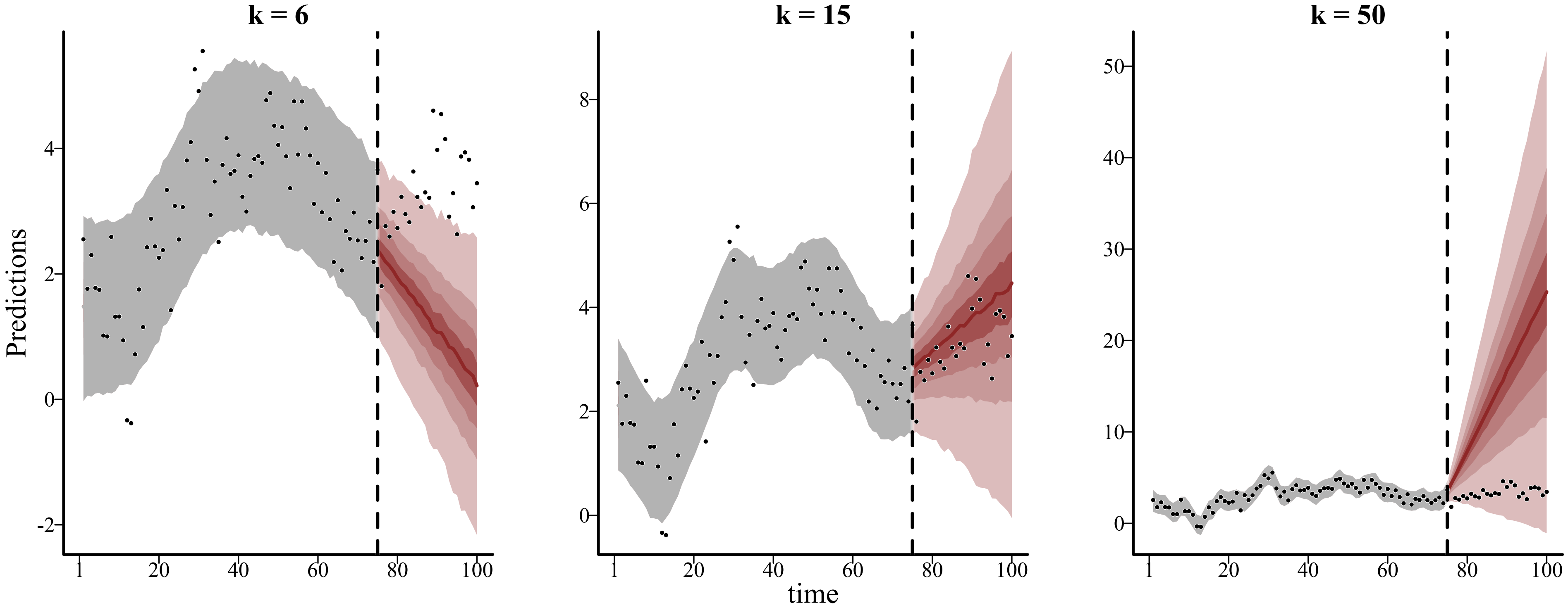


Capturing this autocorrelation is important

Improves inferences on other parts of the model, while also giving more appropriate p-values, confidence intervals etc... in frequentist paradigms

But what effect does this variation in wiggliness have on forecasts?

Forecasts vary *hugely*



TOO MANY WIGGLES

Image (shamelessly) borrowed from: https://fortemag.com.au/

`gam.check` is sensitive to unmodelled autocorrelation

Raising `k` to satisfy warnings may improve inference on historical patterns, but leads to even more unpredictable extrapolation behaviour

If the goal is to produce predictions (i.e. to forecast), we can do better with appropriate *time series models*

Ok. Can we just do this?

A linear model with an autoregressive term

$$
\begin{aligned}
\boldsymbol{Y}_t &\sim \text{Normal}(\mu_t, \sigma) \\
\mu_t &= \alpha + \beta_1 \boldsymbol{Y}_{t-1} + \cdots
\end{aligned}
$$

Where:

α is an intercept coefficient

β_1 is a ***first-order autoregressive coefficient***

Can sometimes work because of identity link; but missingness, measurement error will still cause problems

What about Poisson?

A Poisson GLM with an autoregressive term

$$\boldsymbol{Y}_t \sim \text{Poisson}(\lambda_t)$$
$$log(\lambda_t) = \alpha + \beta_1 \boldsymbol{Y}_{t-1} + \cdots$$

Where:

α is an intercept coefficient

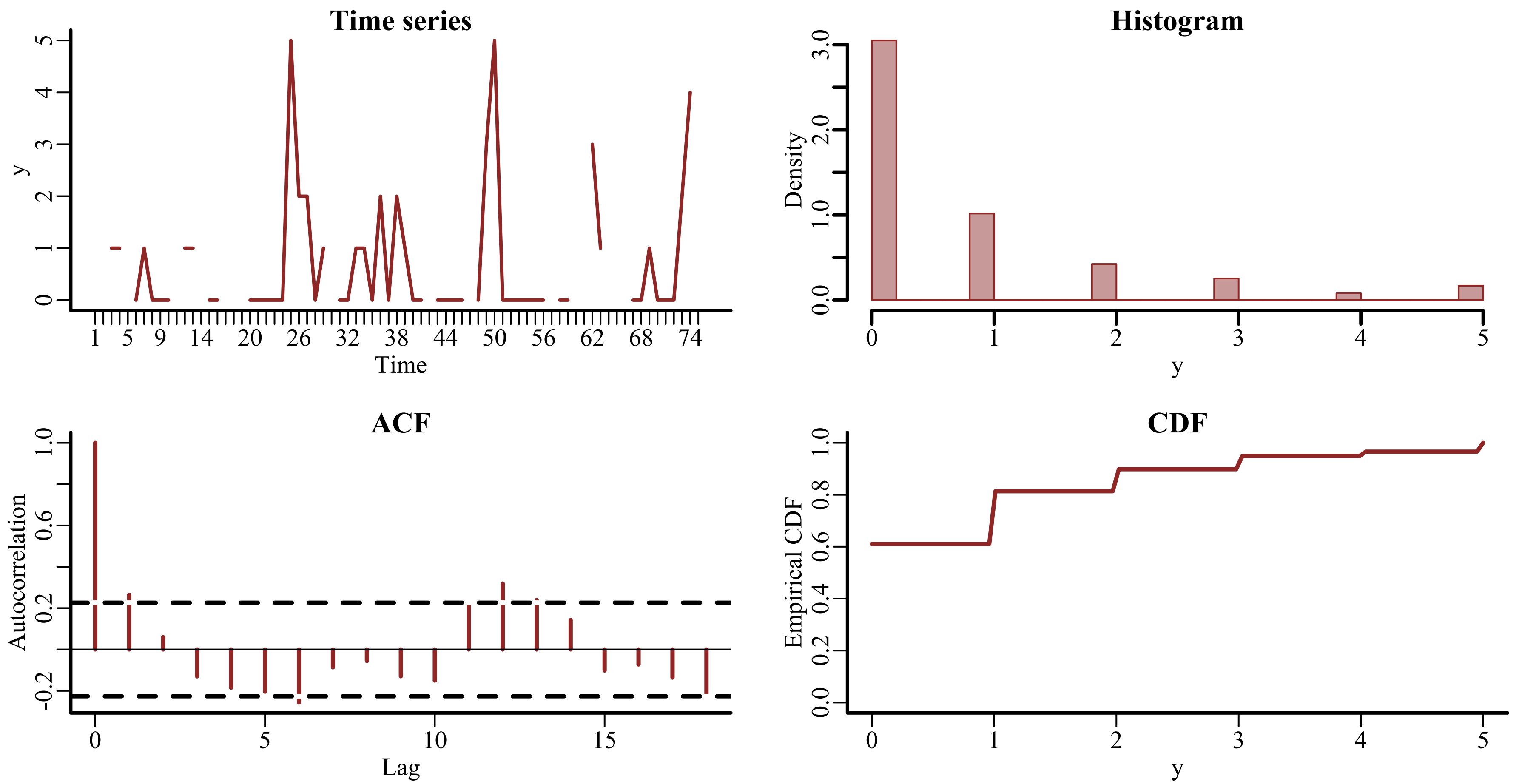
β_1 is a ***first-order autoregressive coefficient***

Motivating example (skip)

```r
# set seed for reproducibility
set.seed(222)

# simulate an integer-valued time series with some missing observations
sim_data <- sim_mvgam(T = 100, n_series = 1,
                      trend_model = 'RW',
                      prop_missing = 0.2)
```

y	season	year	series	time
NA	1	1	series_1	1
1	2	1	series_1	2
1	3	1	series_1	3
NA	4	1	series_1	4

Simulated data (skip)



Use `tscount` 📦? (skip)

```r
# attempt a tscount time series model
# which can fit autoregressive models for count time series
library(tscount)

# use the tsglm function for AR modelling
tsglm(sim_data$data_train$y,

      # model using outcome at lag 1 as the predictor
      model = list(past_obs = 1))
```

```
## Error in tsglm.meanfit(ts = ts, model = model, xreg = xreg, link = link, :
Cannot make estimation with missing values in time series or covariates
```

`NA`s cause big problems in autoregressive models

NAs compound (skip)

time	y	y_lag1	y_lag2	season	year	series
1	NA	NA	NA	1	1	series_1
2	1	NA	NA	2	1	series_1
3	1	1	NA	3	1	series_1
4	NA	1	1	4	1	series_1
5	0	NA	1	5	1	series_1
6	1	0	NA	6	1	series_1
7	0	1	0	7	1	series_1
8	0	0	1	8	1	series_1

2/8 rows complete (skip)

time	y	y_lag1	y_lag2	season	year	series
1	NA	NA	NA	1	1	series_1
2	1	NA	NA	2	1	series_1
3	1	1	NA	3	1	series_1
4	NA	1	1	4	1	series_1
5	0	NA	1	5	1	series_1
6	1	0	NA	6	1	series_1
7	0	1	0	7	1	series_1
8	0	0	1	8	1	series_1

Other problems of AR observations

- Measurement errors also compound
- Difficult / impossible to ensure stability of forecasts
  - Can use $log(Y_{t-lag})$ as predictors, but this doesn't always work
- Challenging to link dynamics across multiple series
- Not extendable to other types of dynamics
  - Smooth temporal evolution
  - Changepoint models
  - Stochastic variance / volatility
  - etc...

Latent autoregressive processes

Dynamic Poisson GLM

A dynamic Poisson GLM can use ***autocorrelated latent residuals***

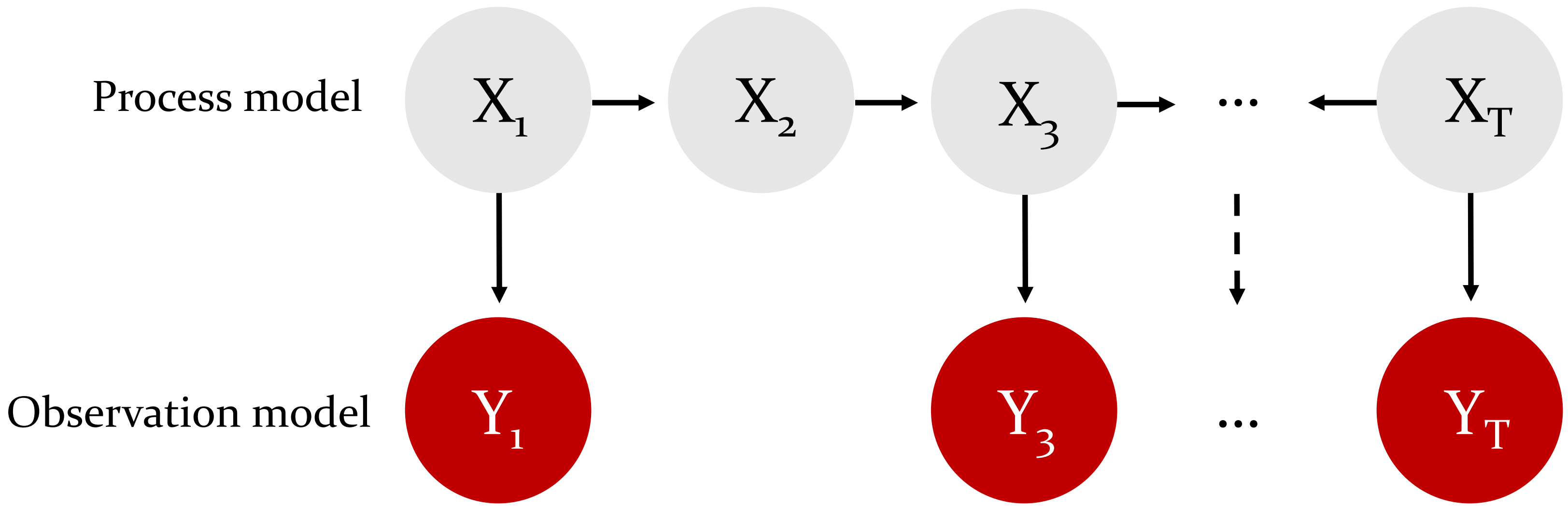
$$
\begin{aligned}
\boldsymbol{Y}_t &\sim \text{Poisson}(\lambda_t) \\
log(\lambda_t) &= \alpha + \cdots + z_t \\
z_t &\sim \text{Normal}(z_{t-1}, \sigma) \\
\sigma &\sim \text{Exponential}(2)
\end{aligned}
$$

Where:

z_t is the value of the latent residual at time t

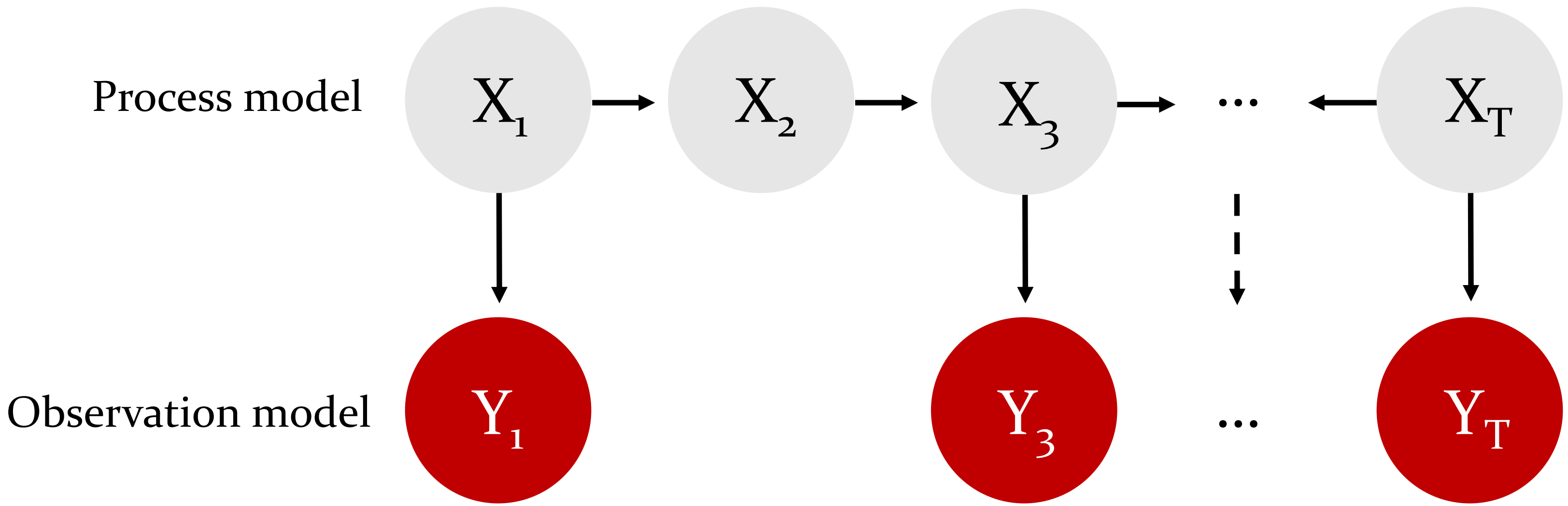
σ captures variation in the latent dynamic process

Evolves *independently*



Missing observations do not impede evolution of the *latent* process

Evolves *independently*



The latent process model can take on a *huge variety* of forms

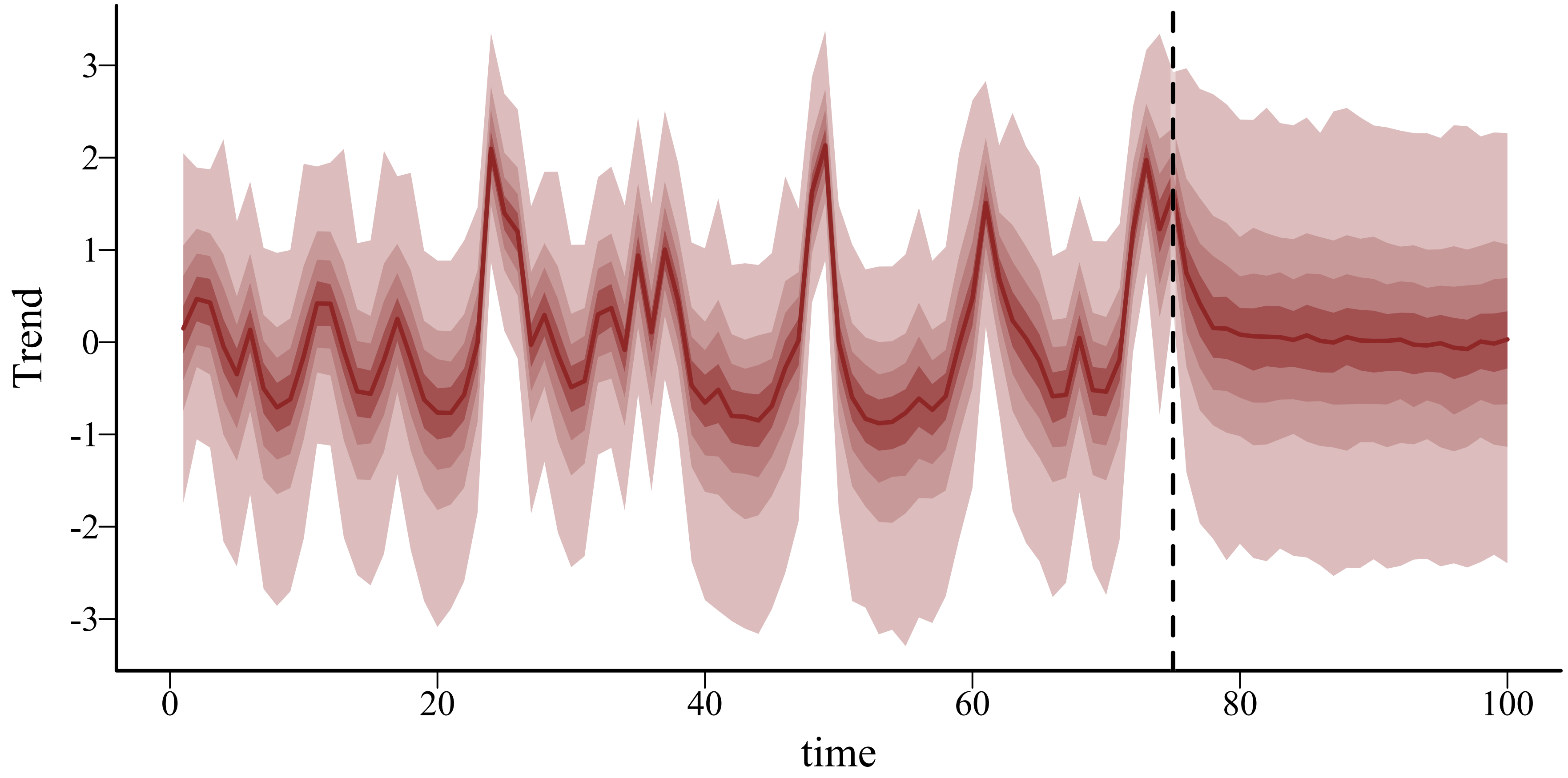
Back to the example

```
mod_example <- mvgam(y ~ 1,
                     trend_model = AR(p = 1),
                     data = sim_data$data_train,
                     newdata = sim_data$data_test,
                     family = poisson())
```

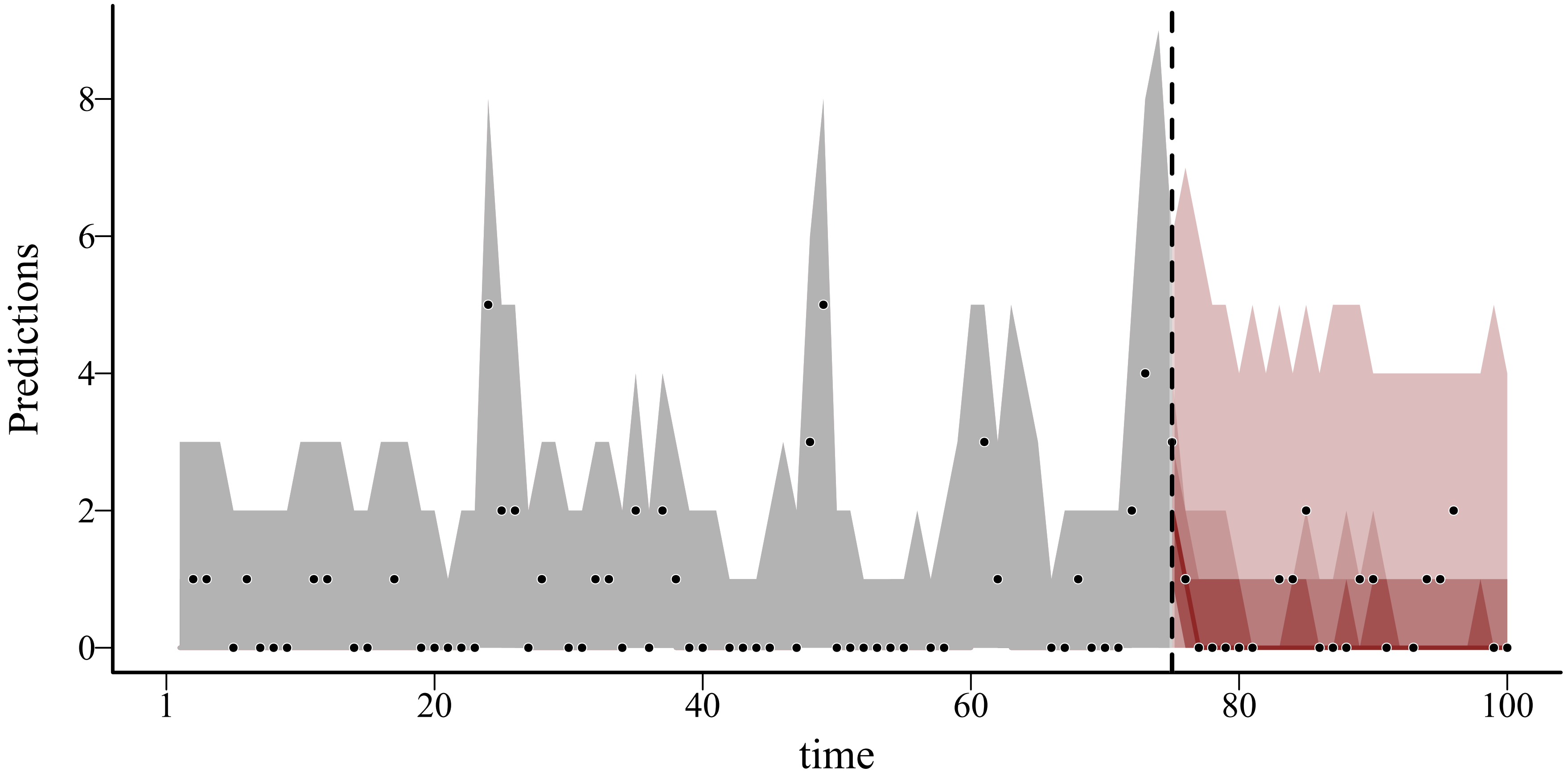
`mvgam` 📦 has no problem with these observations

Fit a model with latent AR1 dynamics and just an intercept in the observation model

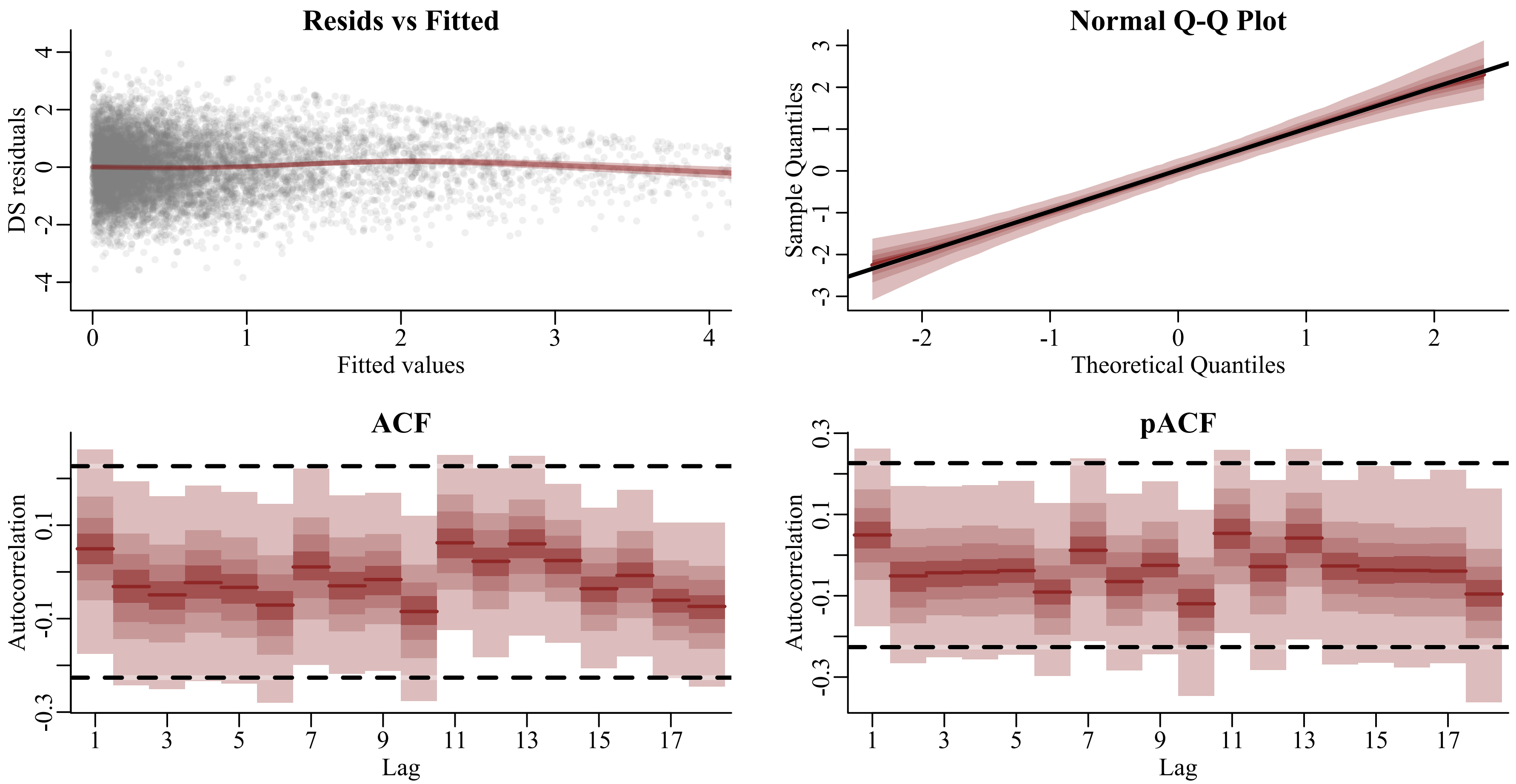
The latent trend



Forecasts



Residuals



Resids vs Fitted
DS residuals
Fitted values
Normal Q-Q Plot
Sample Quantiles
Theoretical Quantiles
ACF
Autocorrelation
Lag
pACF
Autocorrelation
Lag

A tougher example?

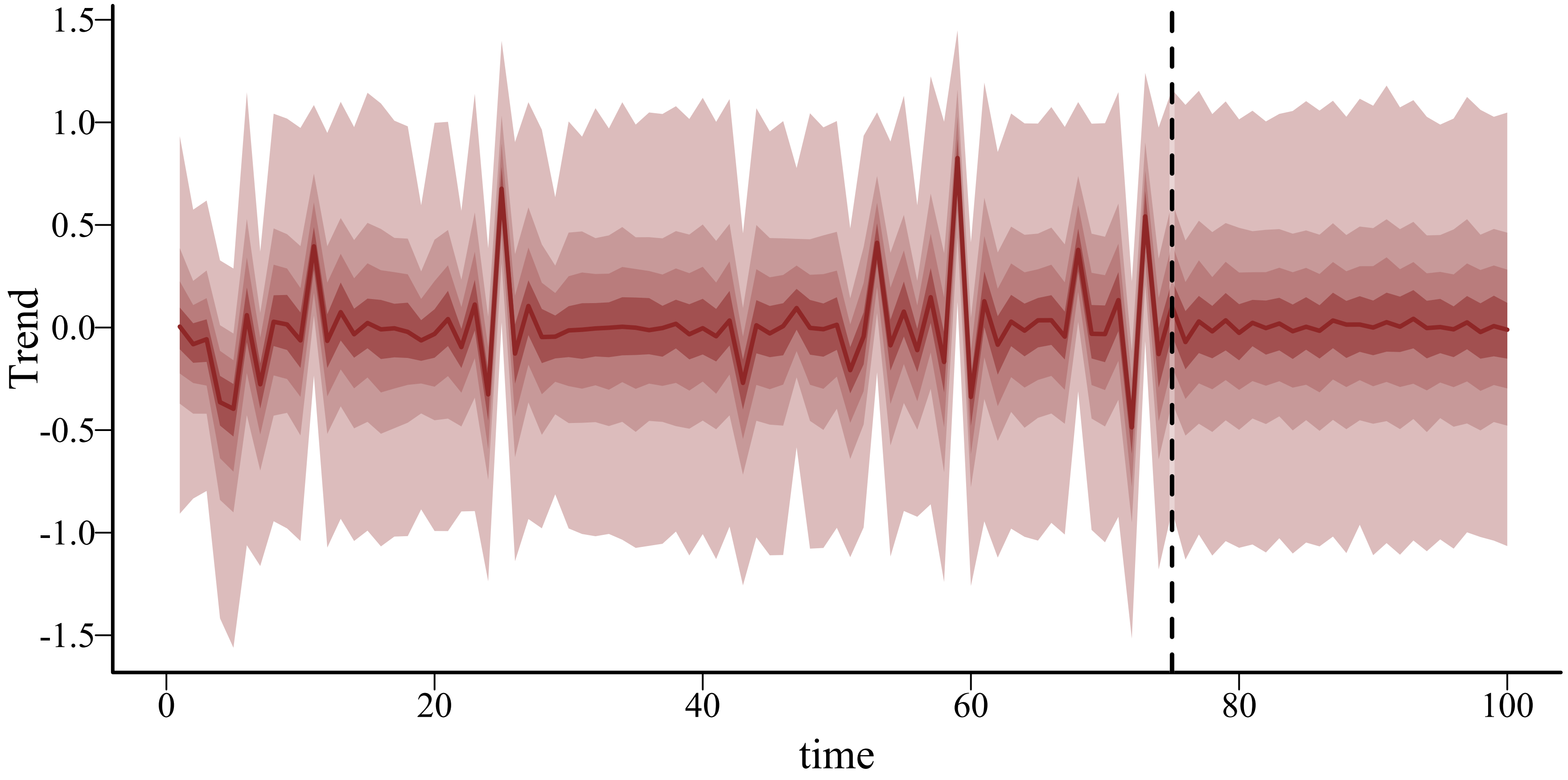
```r
# set seed for reproducibility
set.seed(100)

# simulate an integer-valued time series with some missing observations
sim_data2 <- sim_mvgam(T = 100, n_series = 1,
                        mu = 1,
                       trend_model = 'RW',
                       prop_missing = 0.75)
```

75% of observations missing!

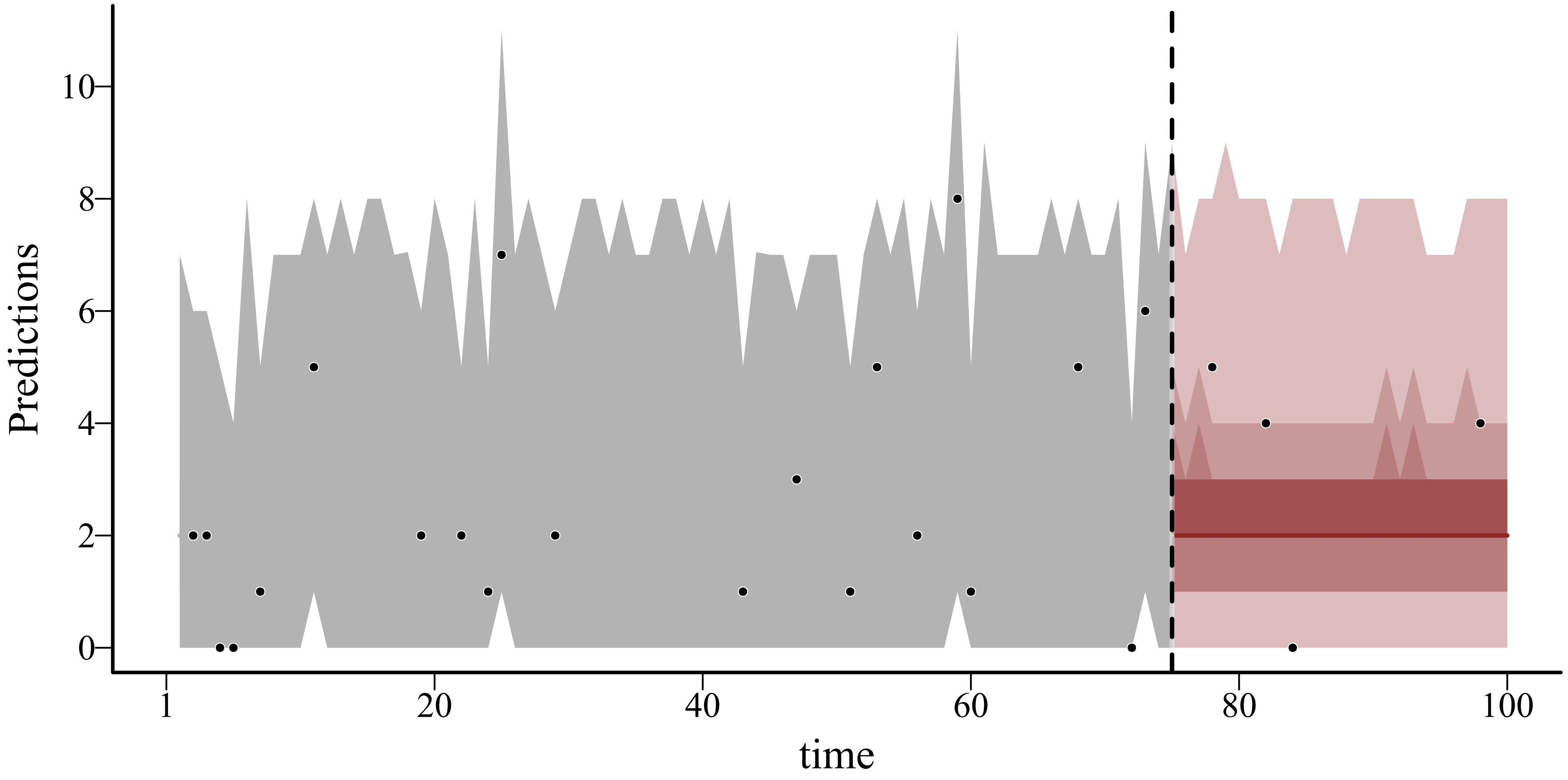
Same model

```
mod_example2 <- mvgam(y ~ 1,
                      trend_model = AR(p = 1),
                      data = sim_data2$data_train,
                      newdata = sim_data2$data_test,
                      family = poisson())
```

The latent trend



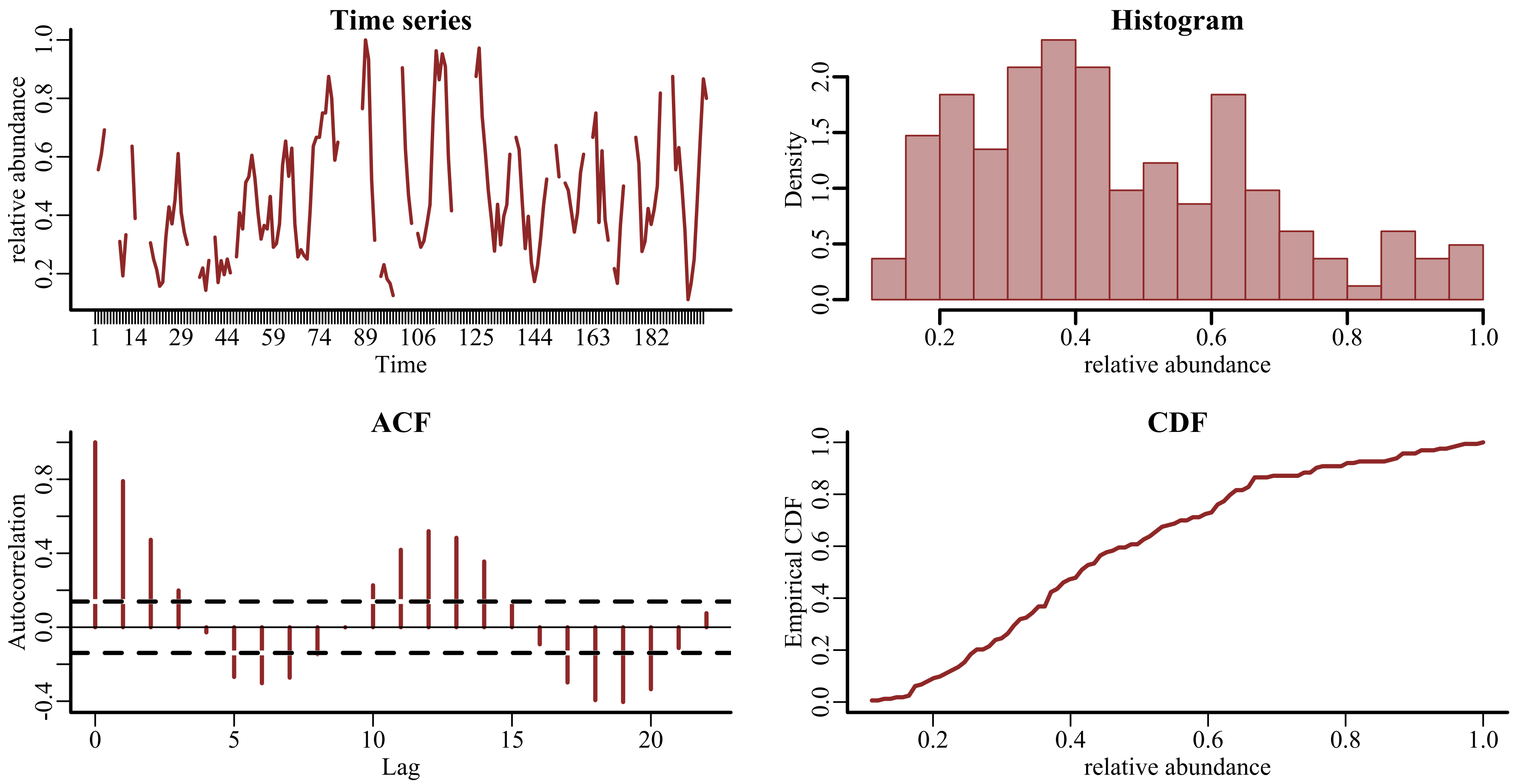
Forecasts



***Some* packages exist to model count-valued time series using autoregressive terms**

But you must not have missing data or measurement error, and you cannot handle multiple series at once

Fine for some situations. But what if your data look like this?



Properties of Merriam's kangaroo rat relative abundance time series from a long-term monitoring study in Portal, Arizona, USA

Live code example

Dynamic Beta GAM

```
mod_beta <- mvgam(relabund ~
                    te(mintemp, ndvi),
                  trend_model = AR(p = 3),
                  family = betar(),
                  data = dm_data)
```

Beta regression using the `mgcv` 📦 's `betar` family

Dynamic Beta GAM

```
mod_beta <- mvgam(relabund ~
                    te(mintemp, ndvi),
                  trend_model = AR(p = 3),
                  family = betar(),
                  data = dm_data)
```

Beta regression using the `mgcv` 📦 's `betar` family

AR3 dynamic trend model

Dynamic Beta GAM

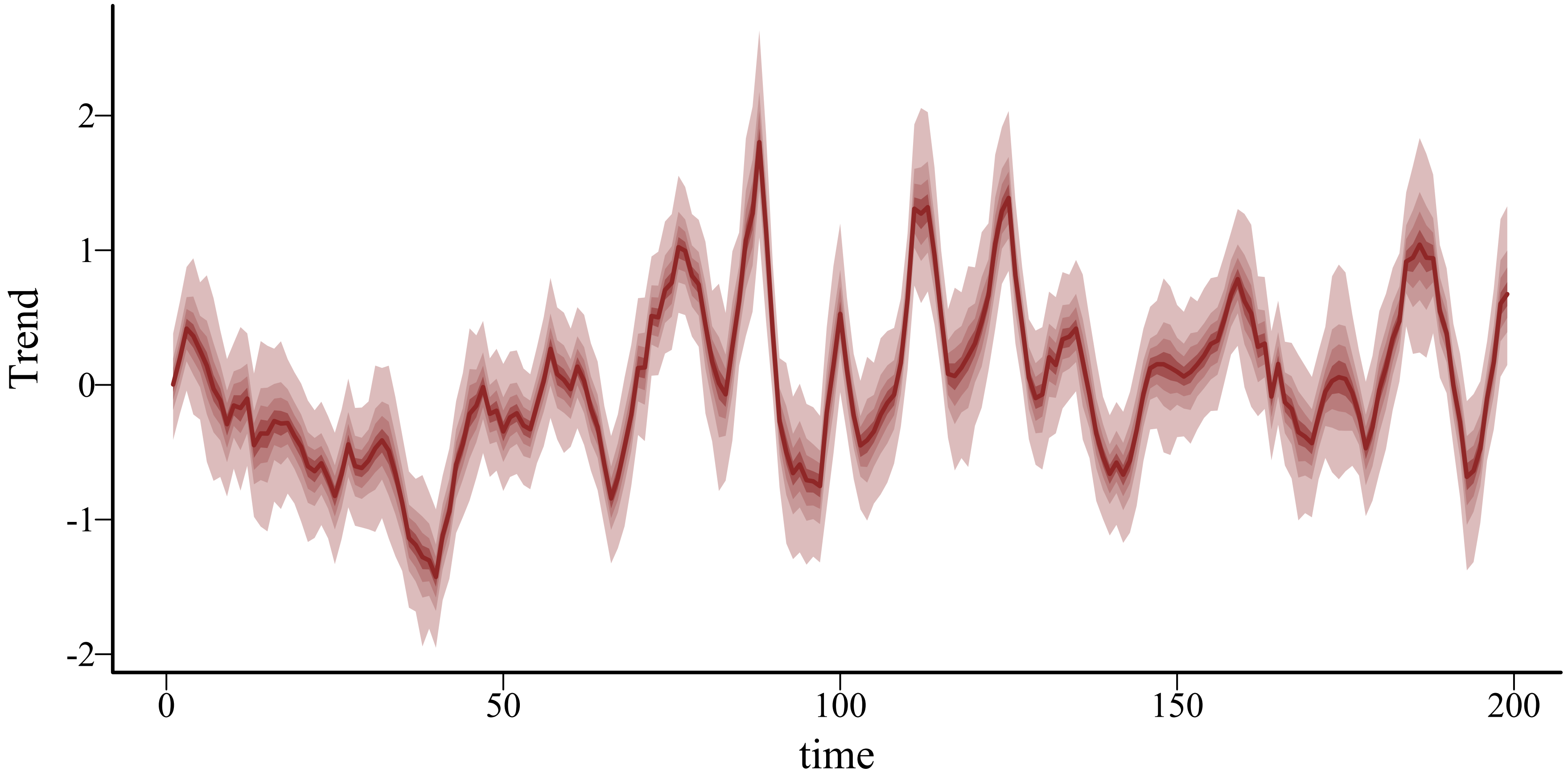
```
mod_beta <- mvgam(relabund ~
                    te(mintemp, ndvi),
                  trend_model = AR(p = 3),
                  family = betar(),
                  data = dm_data)
```

Beta regression using the `mgcv` 📦 's `betar` family

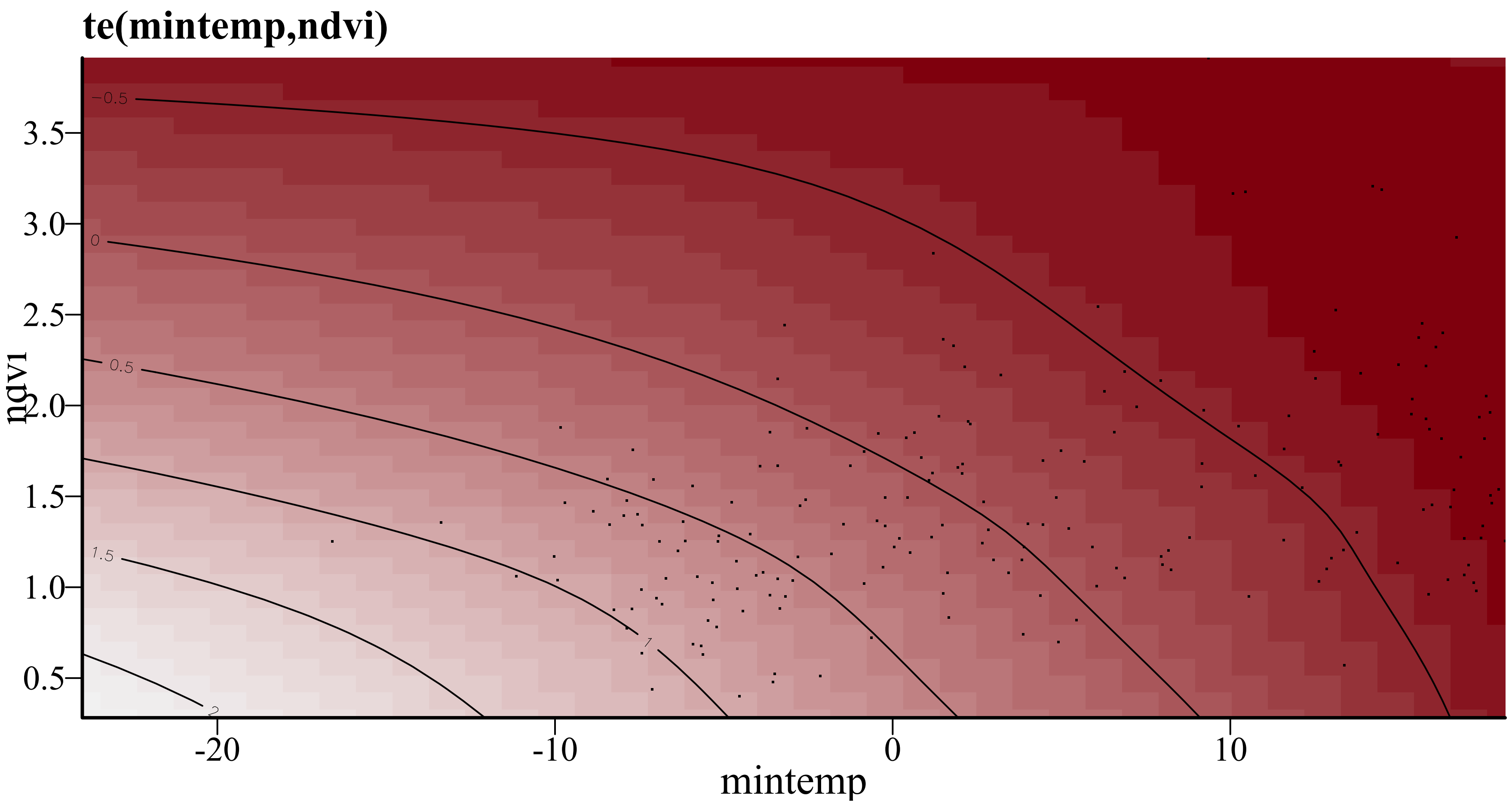
AR3 dynamic trend model

Multidimensional [tensor product smooth function for nonlinear covariate interactions (using `te`)]()

The latent trend



Multidimensionial smooth



Huh?

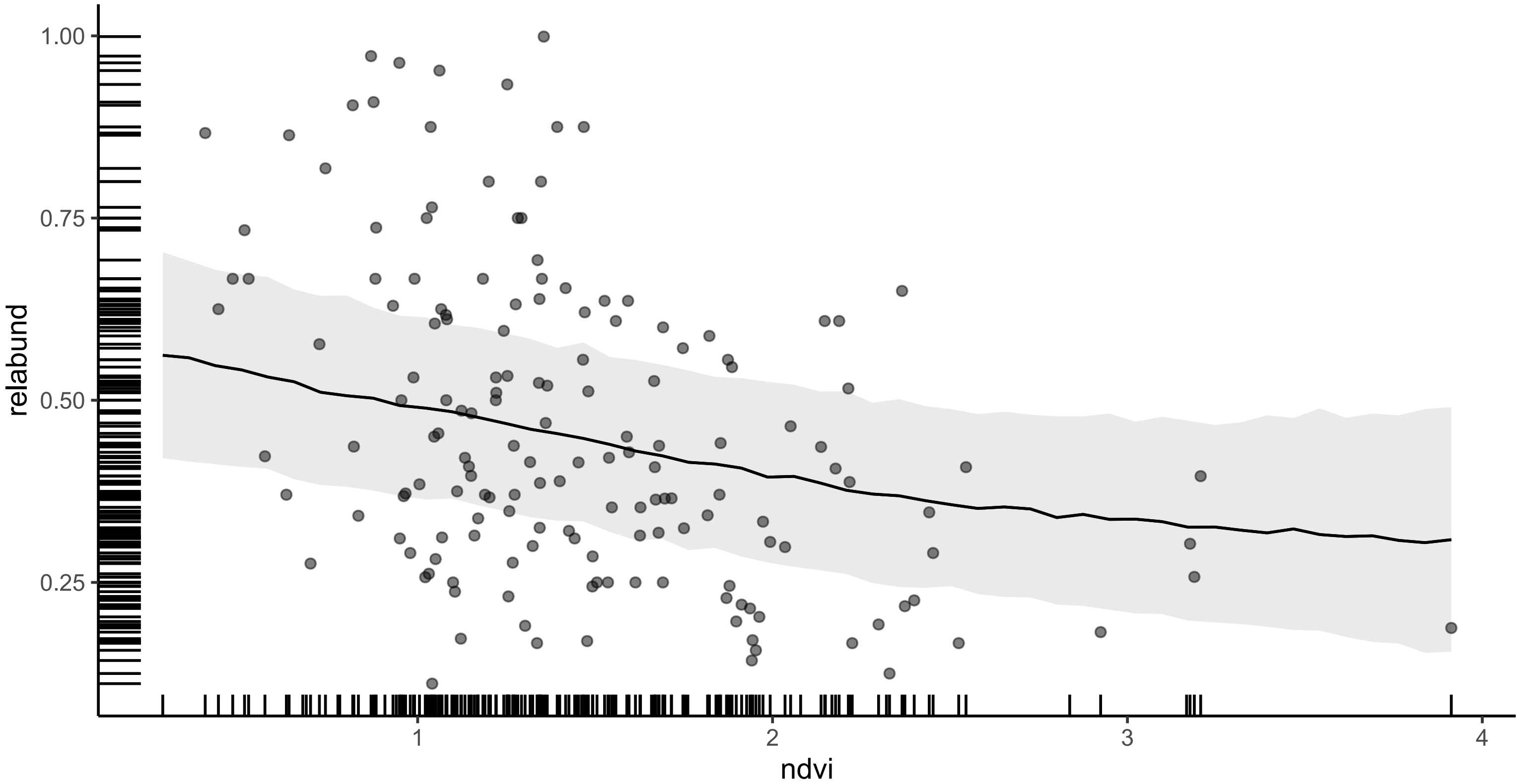
marginaleffects for clarity

Code Plot

```
# plot conditional effect of NDVI on the outcome scale
plot_predictions(mod_beta, condition = 'ndvi',
                 points = 0.5, conf_level = 0.8, rug = TRUE) +
  theme_classic()
```

marginaleffects for clarity

Code Plot



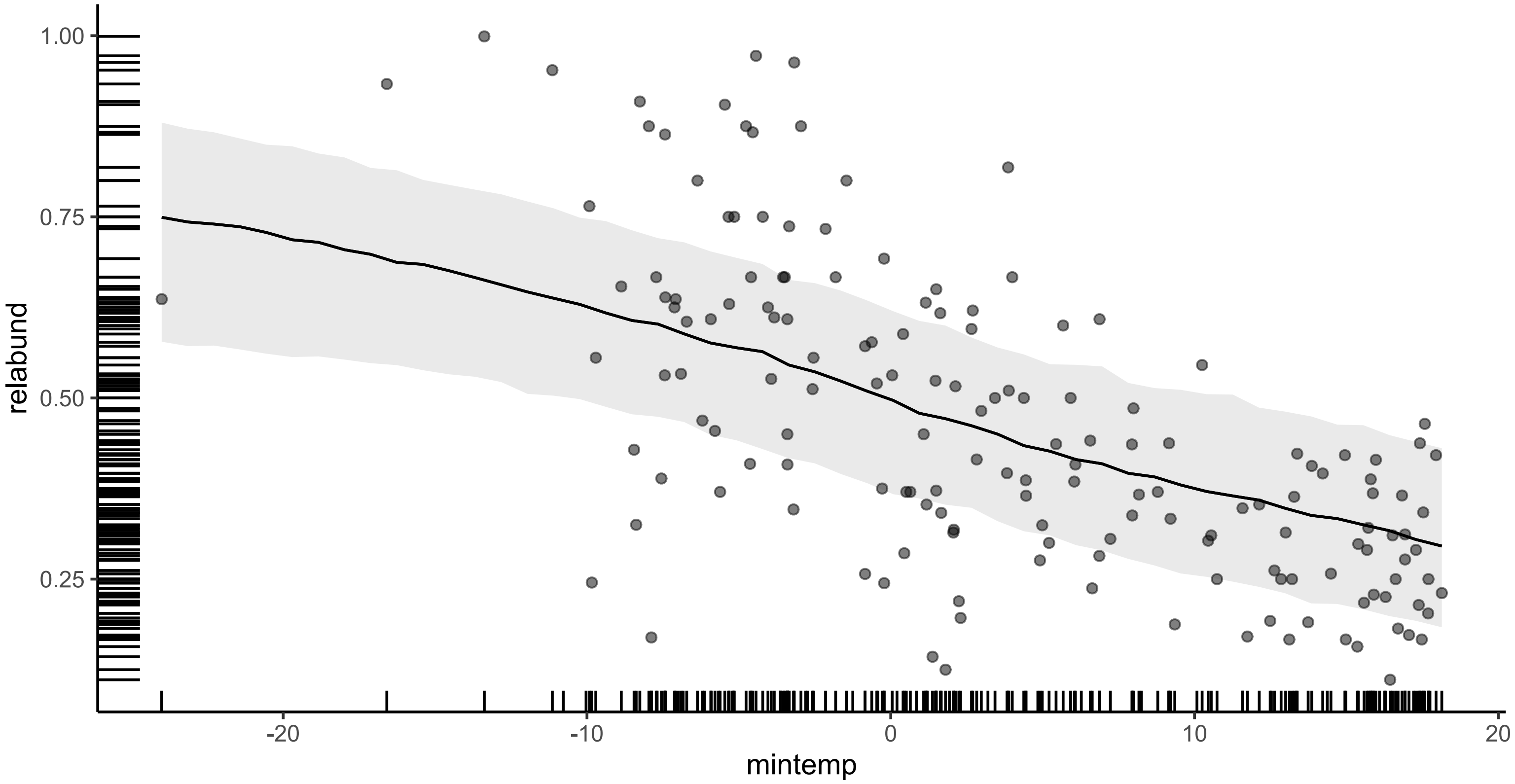
marginaleffects for clarity

Code Plot

```
# plot conditional effect of Min Temp on the outcome scale
plot_predictions(mod_beta, condition = 'mintemp',
                 points = 0.5, conf_level = 0.8, rug = TRUE) +
  theme_classic()
```

marginaleffects for clarity

Code Plot



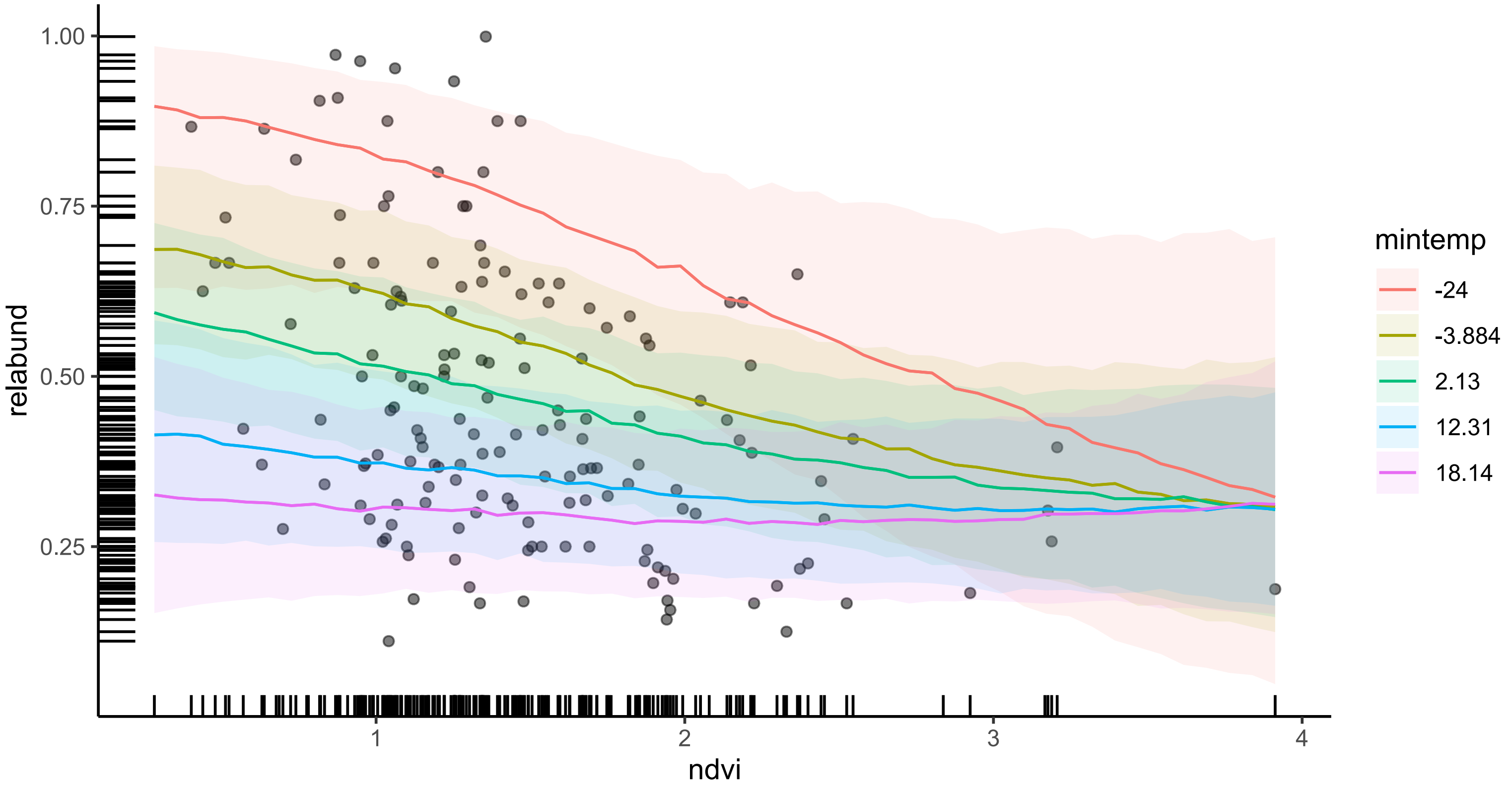
marginaleffects for clarity

Code Plot

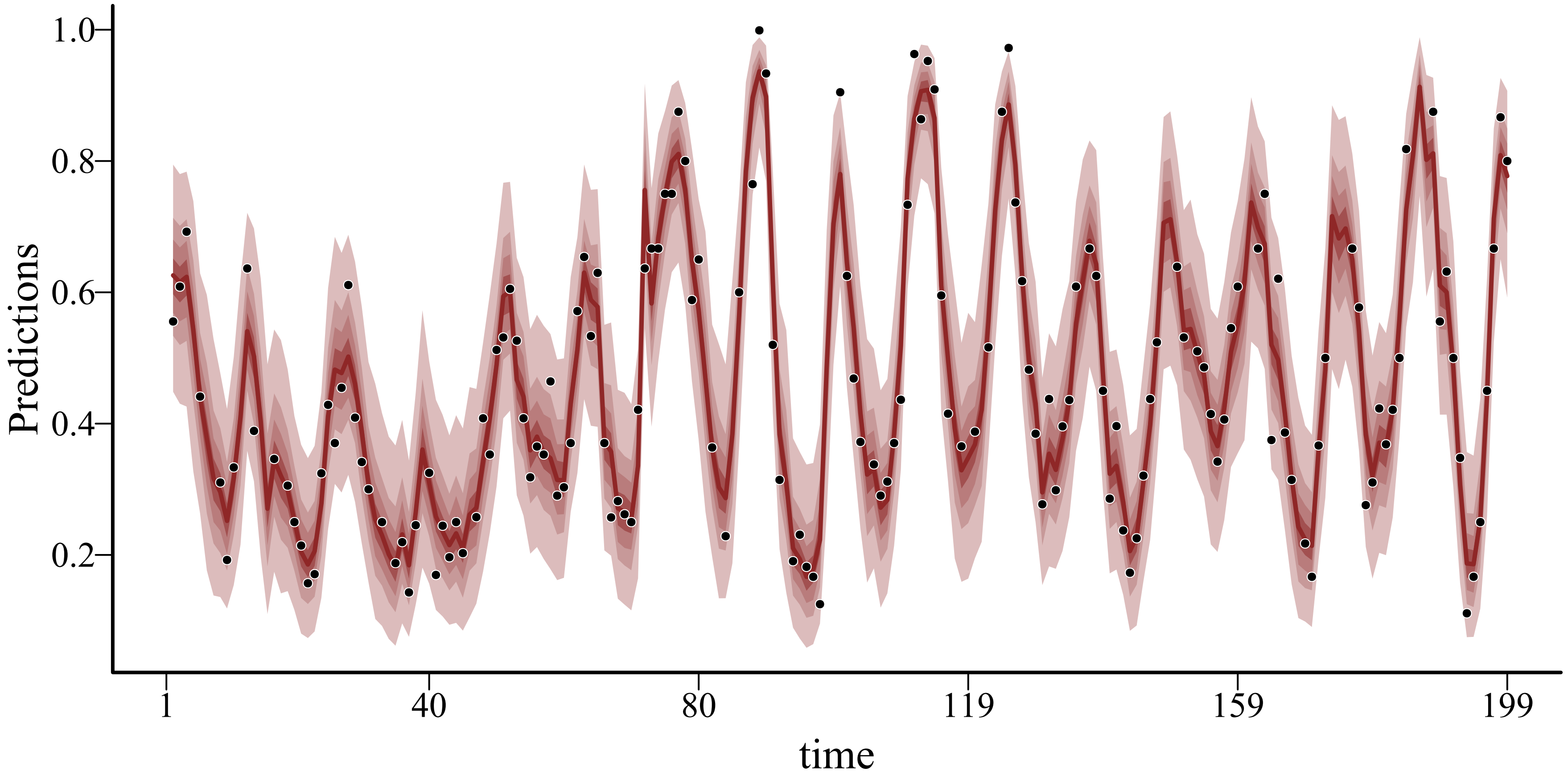
```
# plot conditional effect of BOTH covariates on the outcome scale
plot_predictions(mod_beta, condition = c('ndvi', 'mintemp'),
                 points = 0.5, conf_level = 0.8, rug = TRUE) +
  theme_classic()
```

marginaleffects for clarity

Code Plot



Hindcasts

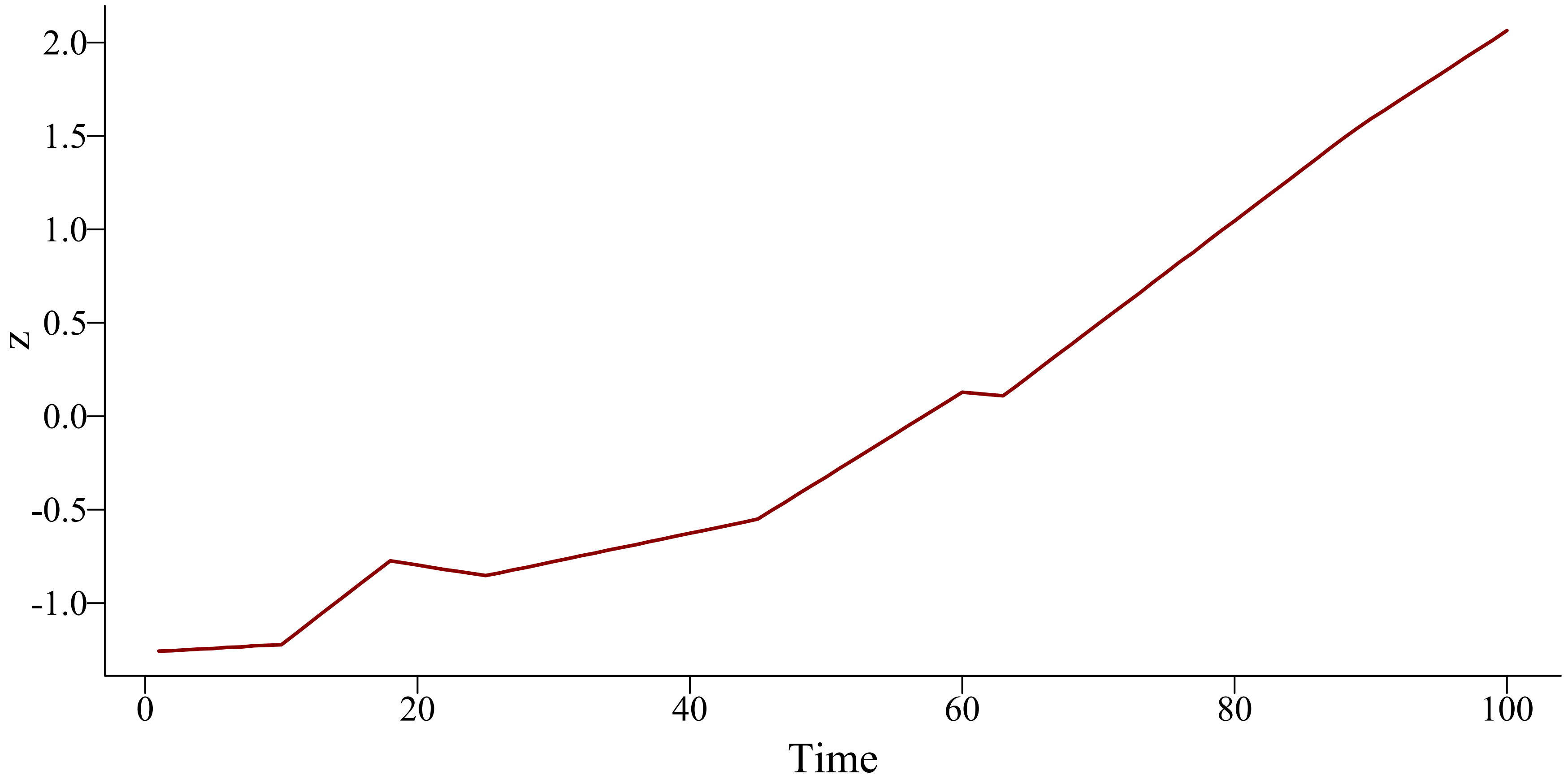


We can estimate latent dynamic residuals for *many* types of GLMs / GAMs, thanks to the link function

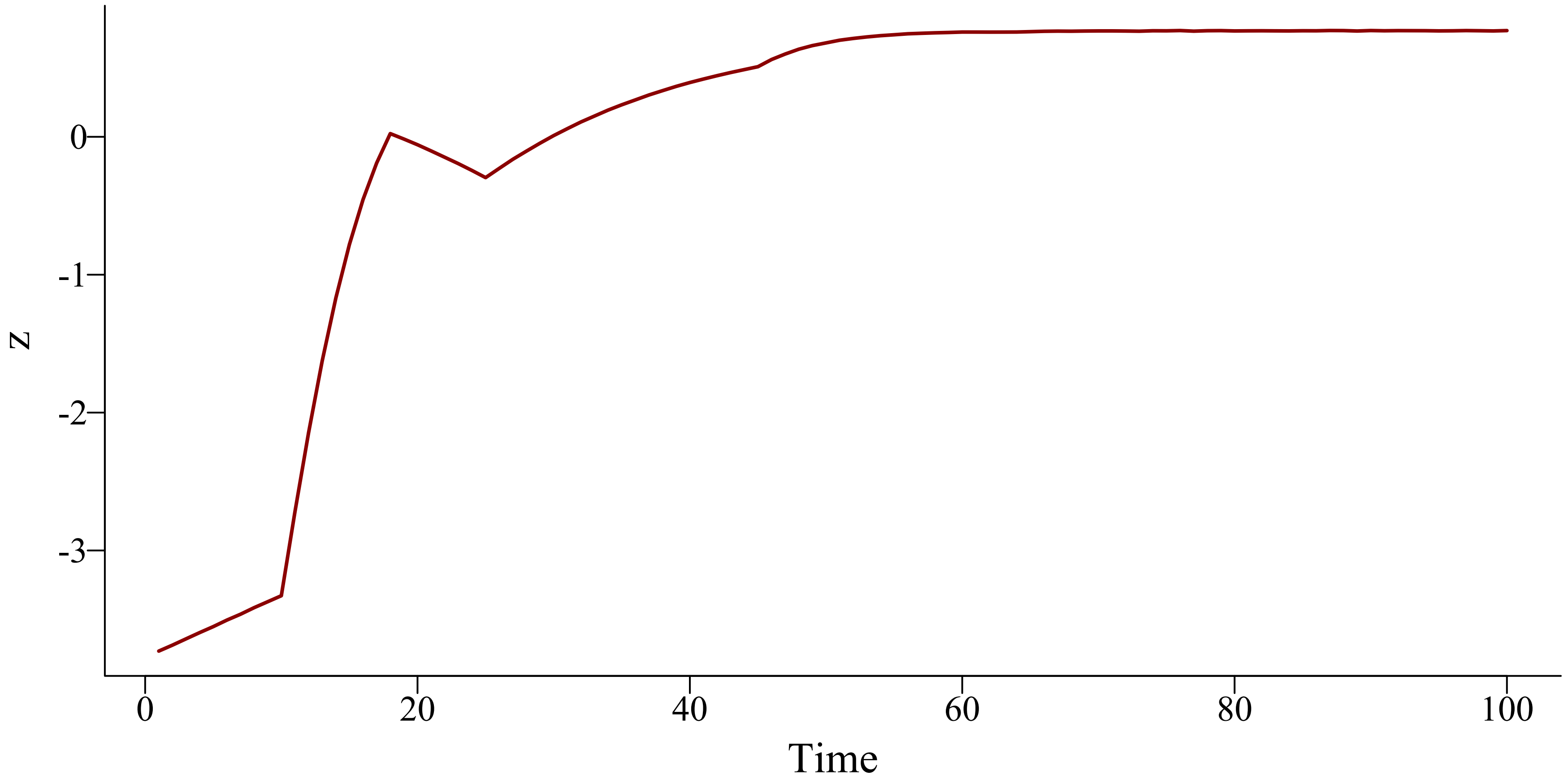
We do not need to regress the outcome on its own past values

Very advantageous for ecological time series. But what kinds of dynamic processes are available in the `mvgam` and `brms` 📦 's?

Piecewise linear...



...or logistic with upper saturation



Random walks

Simple stochastic processes that can fit a wide range of data

$$z_t \sim \text{Normal}(\alpha + z_{t-1}, \sigma)$$

Where:

- σ determines the spread (or flexibility) of the process
- α is an optional intercept or *drift* parameter

Process at time t is centred around it's own value at time $t - 1$, with spread determined by probabilistic error

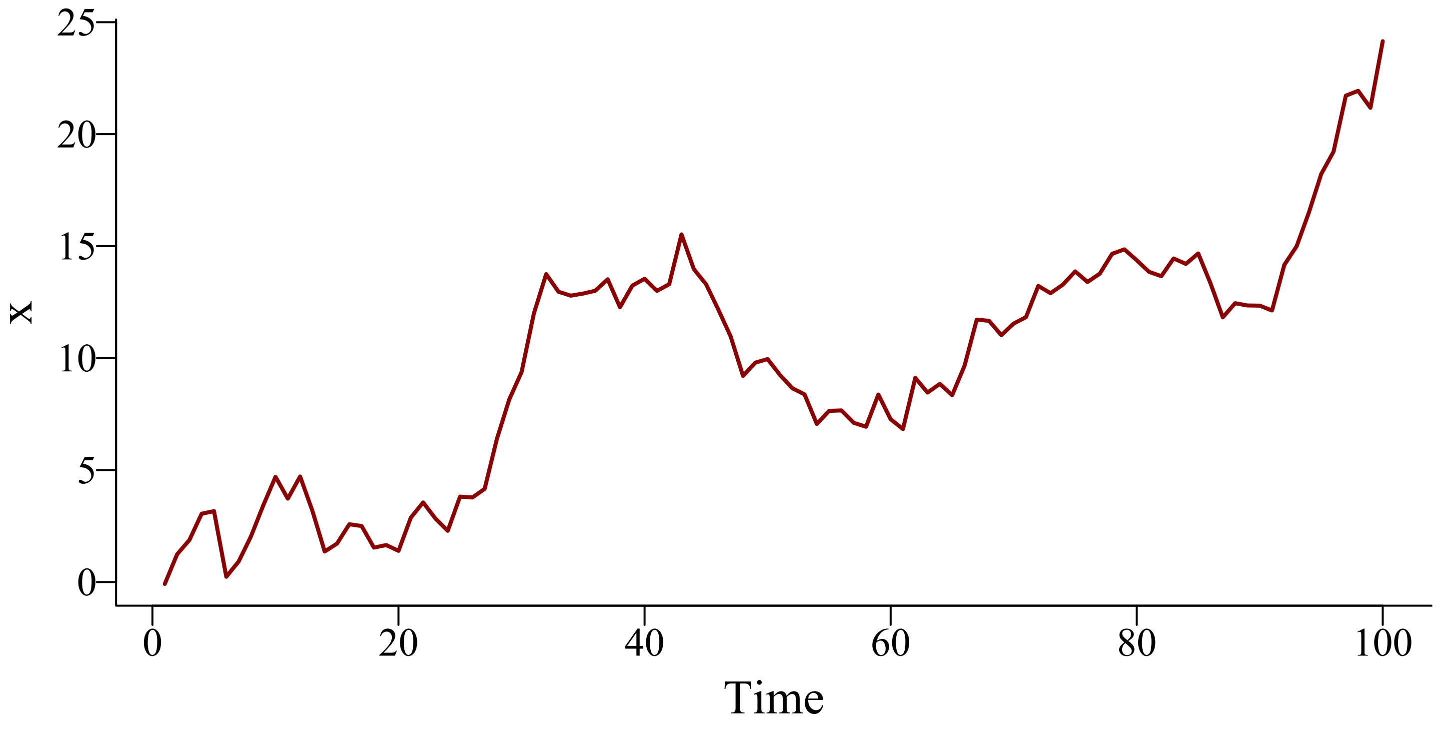
A Random Walk

Code Plot

```r
# set seed and number of timepoints
set.seed(1111); T ← 100

# initialize first value
series ← vector(length = T); series[1] ← rnorm(n = 1, mean = 0, sd = 1)

# compute values 2 through T
for (t in 2:T) {
    series[t] ← rnorm(n = 1, mean = series[t - 1], sd = 1)
}

# plot the time series as a line
plot(series, type = 'l', bty = 'l', lwd = 2,
     col = 'darkred', ylab = 'x', xlab = 'Time')
```

A Random Walk

Code Plot



AR1

Similar to a Random Walk and can fit a wide range of data

$$z_t \sim \mathrm{Normal}(\alpha + \phi * z_{t-1}, \sigma)$$

Where:

σ determines the spread (or flexibility) of the process

α is an optional intercept or *drift* parameter

ϕ is a coefficient estimating correlation between z_t and z_{t-1}

Process at time t is *a function* of it's own value at time $t - 1$

AR2 and AR3

As with AR1, but with additional autoregressive terms

$$z_t \sim \mathrm{Normal}(\alpha + \phi_1 * z_{t-1} + \phi_2 * z_{t-2} + \phi_3 * z_{t-3}, \sigma)$$

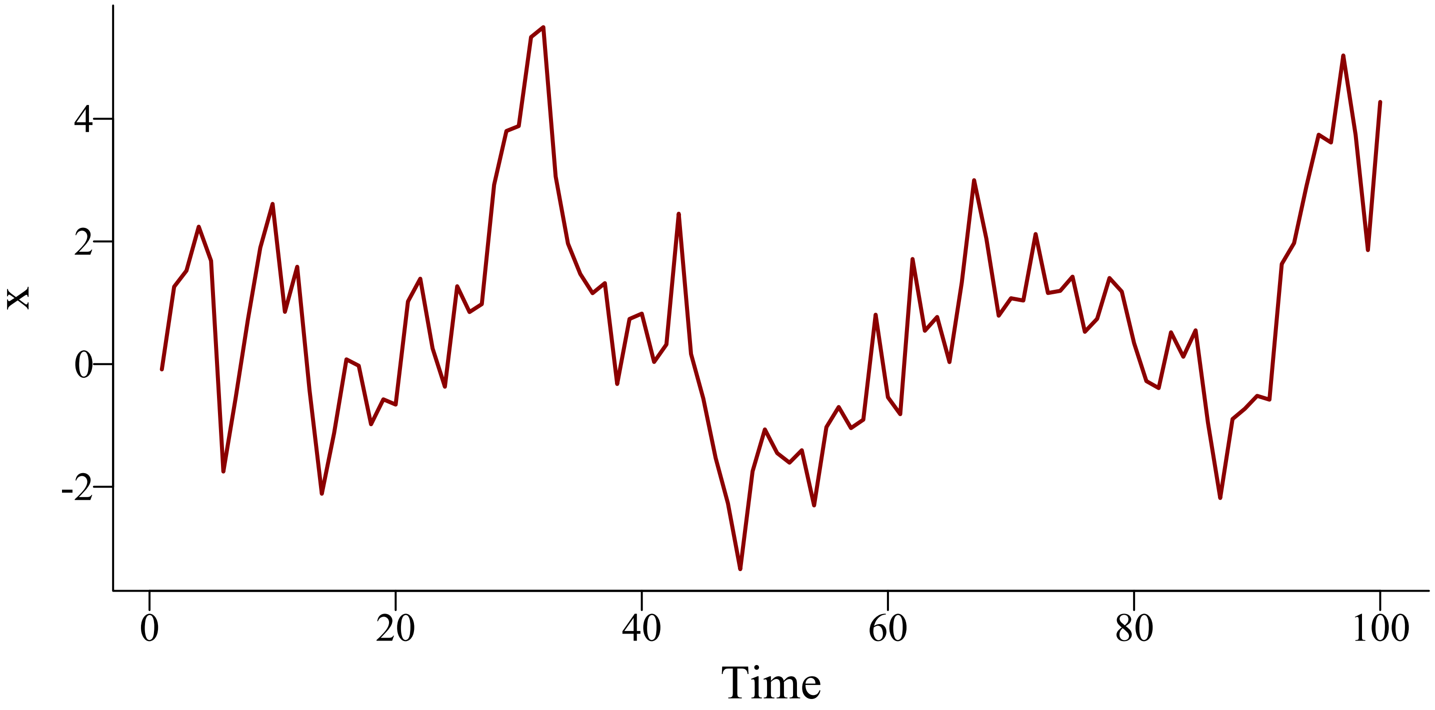
An AR1

Code Plot

```r
# set seed and number of timepoints
set.seed(1111); T <- 100

# initialize first value
series <- vector(length = T); series[1] <- rnorm(n = 1, mean = 0, sd = 1)

# compute values 2 through T, with phi = 0.7
for (t in 2:T) {
    series[t] <- rnorm(n = 1, mean = 0.7 * series[t - 1], sd = 1)
}

# plot the time series as a line
plot(series, type = 'l', bty = 'l', lwd = 2,
     col = 'darkred', ylab = 'x', xlab = 'Time')
```

An AR1

Code Plot



Properties of an AR1

$\phi = 0$ and $\alpha = 0$, process is white noise

$\phi = 1$ and $\alpha = 0$, process is a Random Walk

$\phi = 1$ and $\alpha \neq 0$, process is a Random Walk with drift

$|\phi| < 1$, process oscillates around α and is ***stationary***

Stationarity

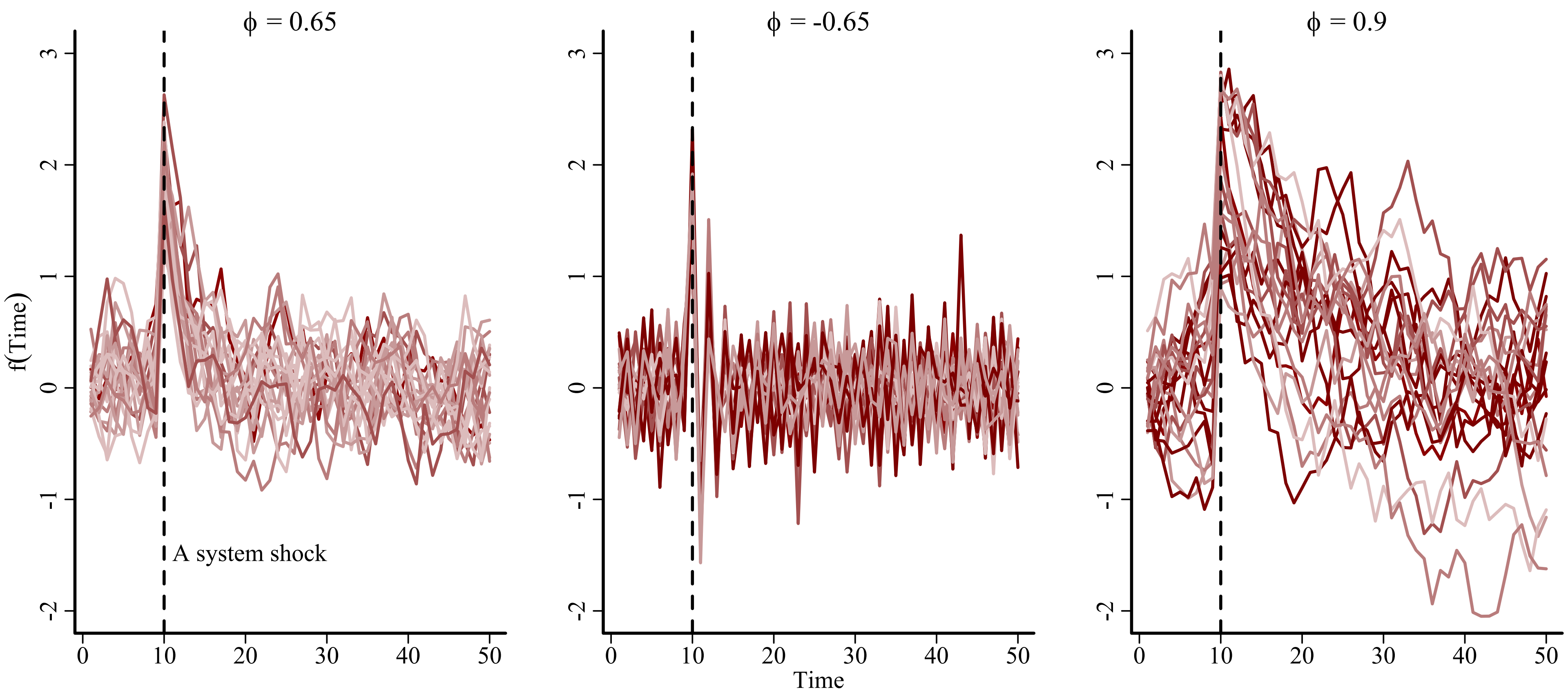
"*A stationary time series is one whose statistical properties do not depend on the time at which the series is observed*" (Hyndman and Athanasopoulos, Forecasting Principles and Practice)

Non-stationary series are more difficult to predict

- Either mean, variance, and/or autocorrelation structure can change over time
- Random Walk is nonstationary because it has no long-term mean

Stationary time series are useful for inferring properties of ***stability***

Stationarity ⇨ stability



It is straightforward to fit latent dynamic models with RW or AR models up to order 3 in `mvgam`. Bayesian regularization helps shrink un-needed AR coefficients toward 0

In `brms`, only AR1 can be fit for non-Gaussian observations (though can also handle ARMA(1,1)) models. However, implementation is different and much slower

But what if we think the latent dynamic process is *smooth*?

Gaussian Processes

Gaussian Processes

"*A Gaussian Process defines a probability distribution over functions; in other words every sample from a Gaussian Process is an entire function from the covariate space X to the real-valued output space.*" (Betancourt; Robust Gaussian Process Modeling)

$$z \sim \mathrm{MVNormal}(0, \Sigma)$$
$$\Sigma_{t_i,t_j} = \alpha^2 * exp(-0.5 * ((|t_i - t_j|/\rho))^2)$$

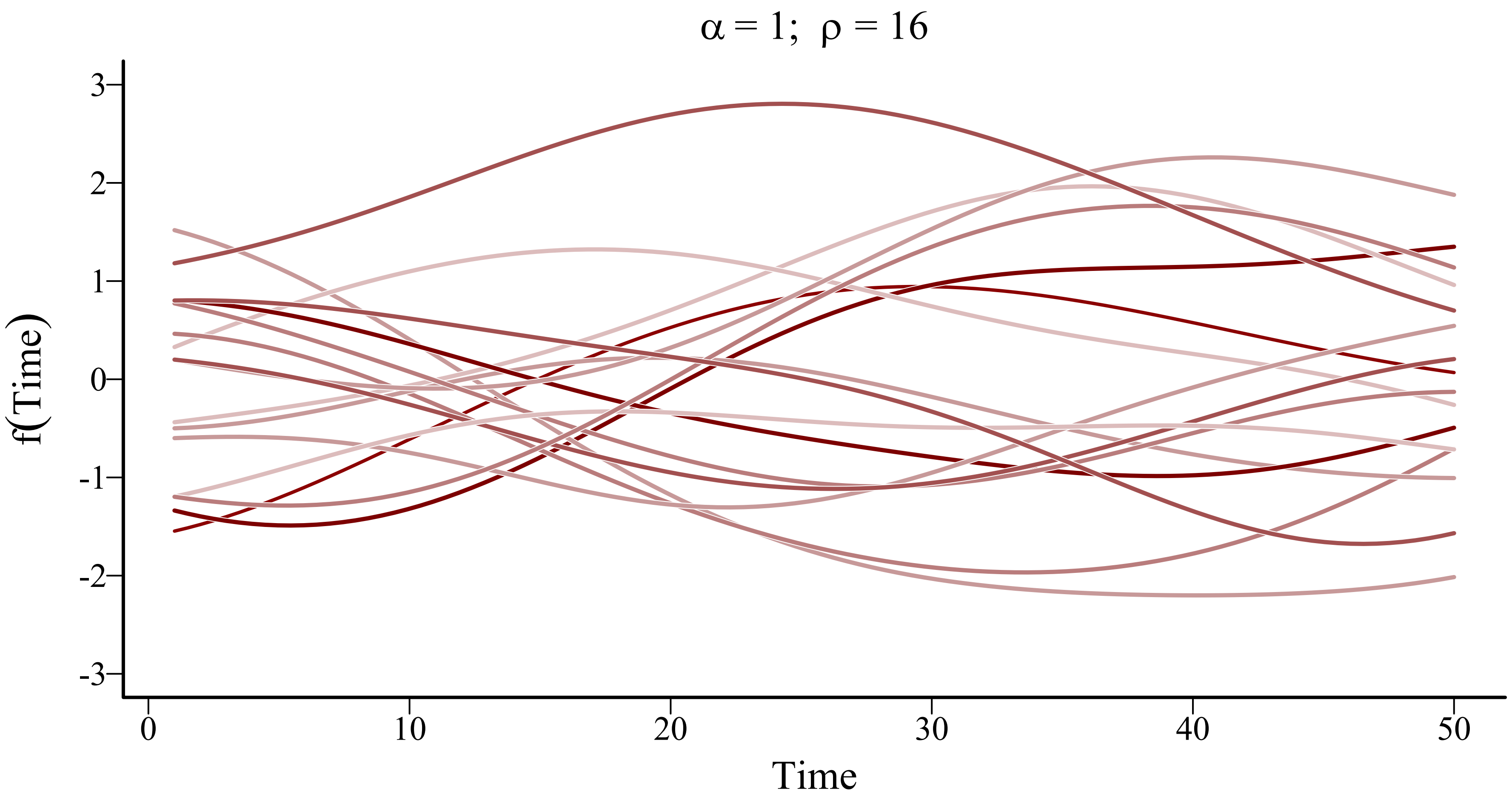
Where:

α controls the marginal variability (magnitude) of the function

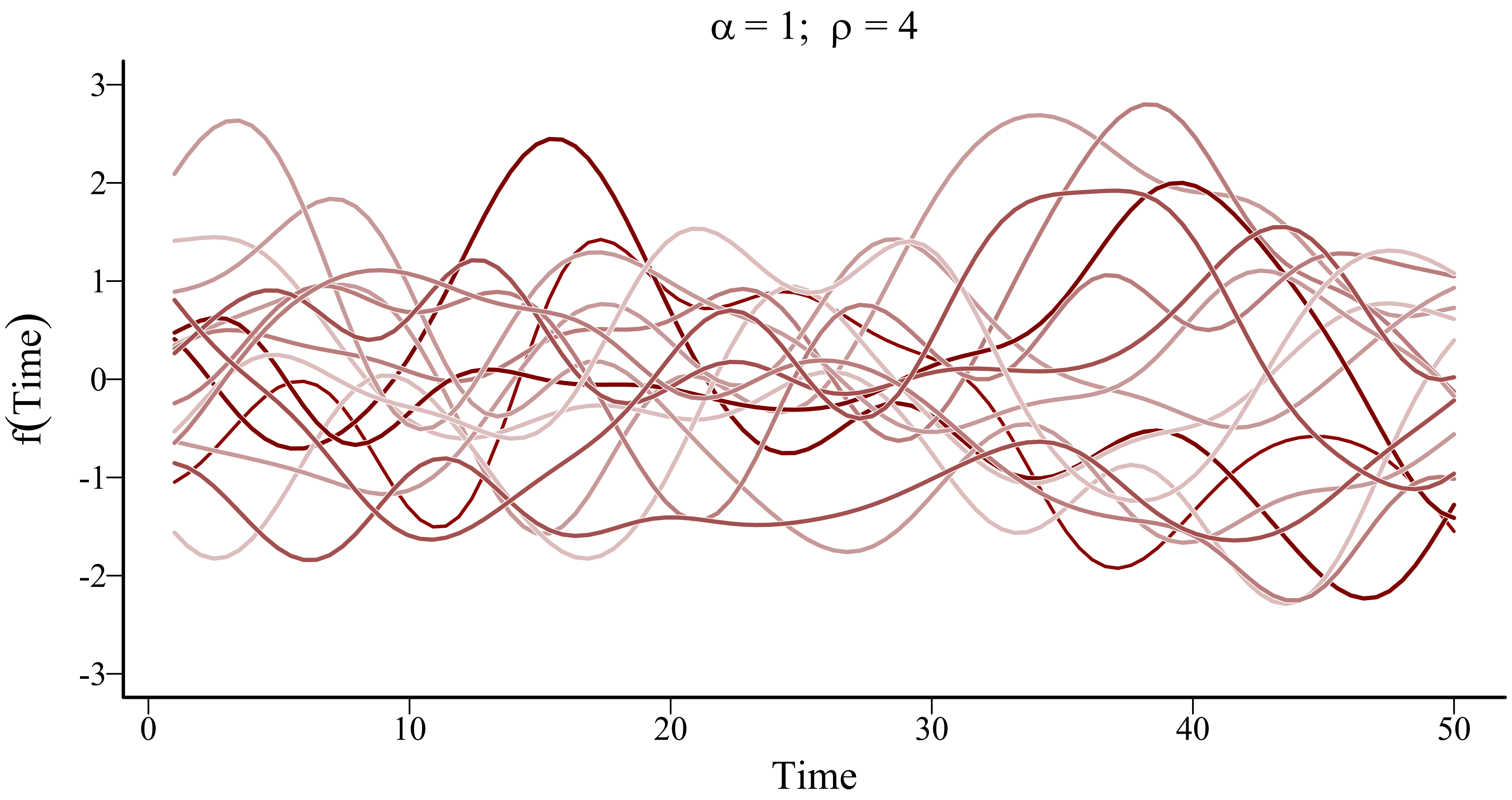
ρ controls how correlations decay as a function of time lag

Σ is the kernel, in this case a squared exponential kernel

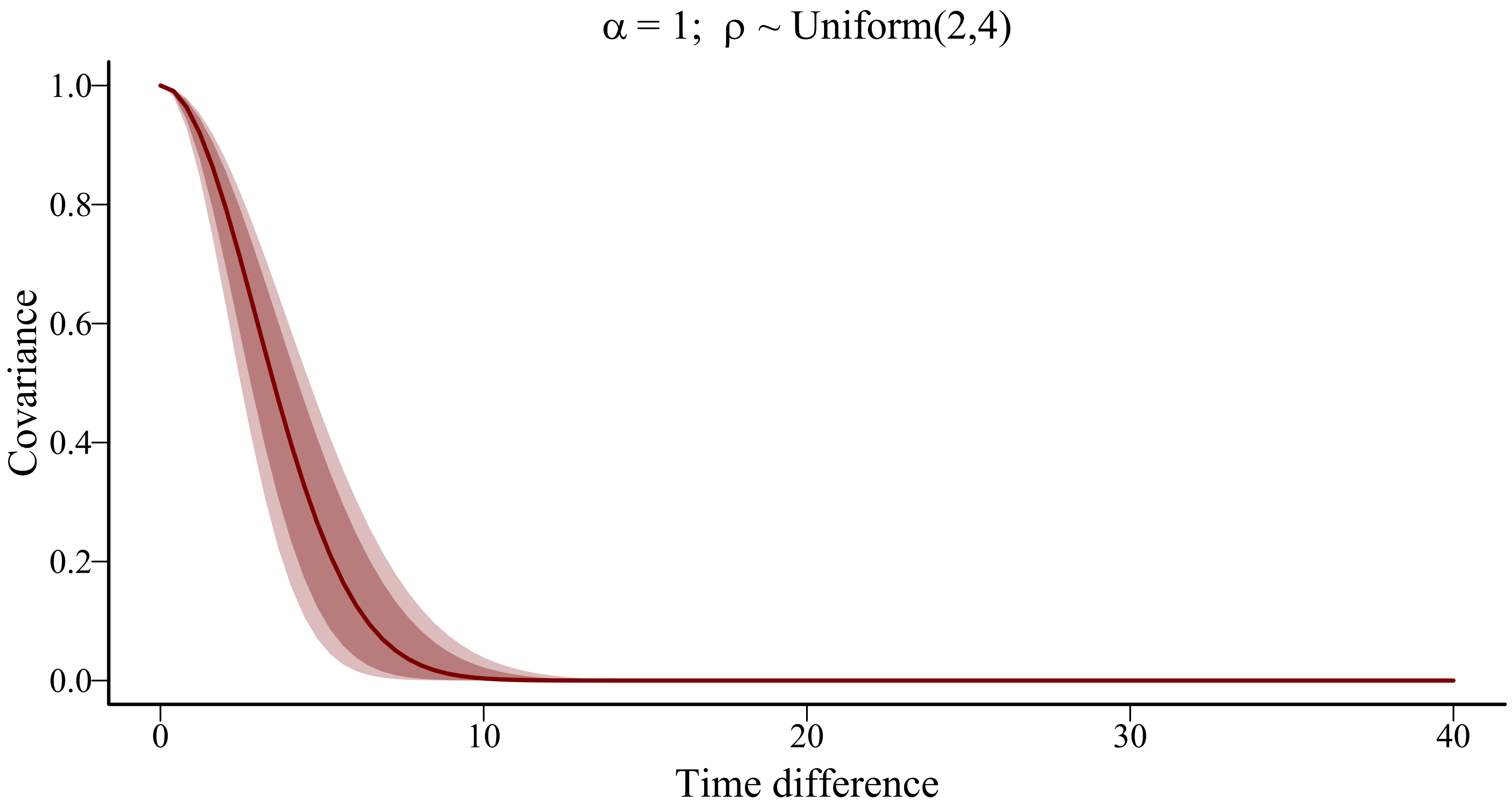
Random *functions*



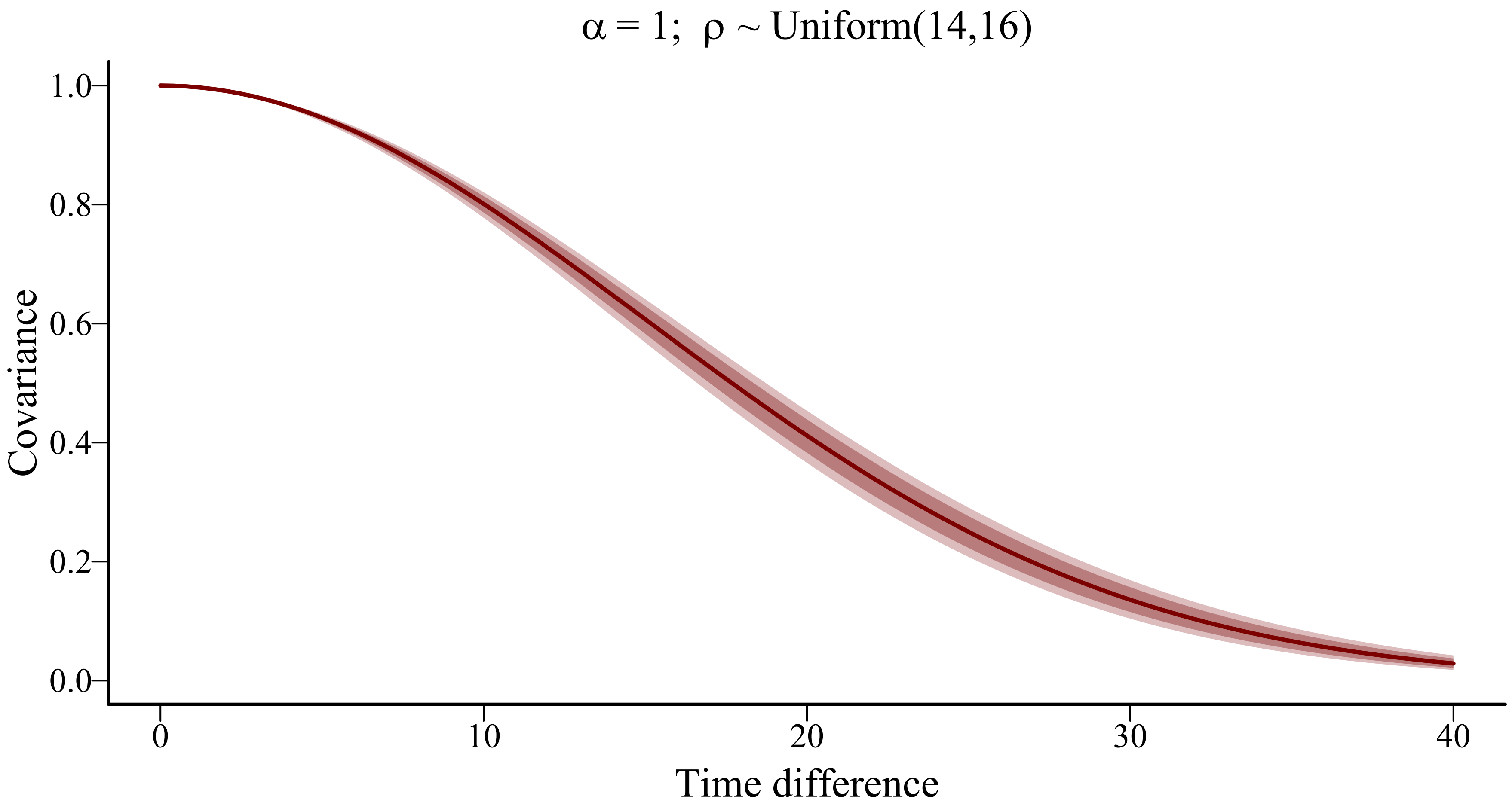
Length scale ⇨ *memory*



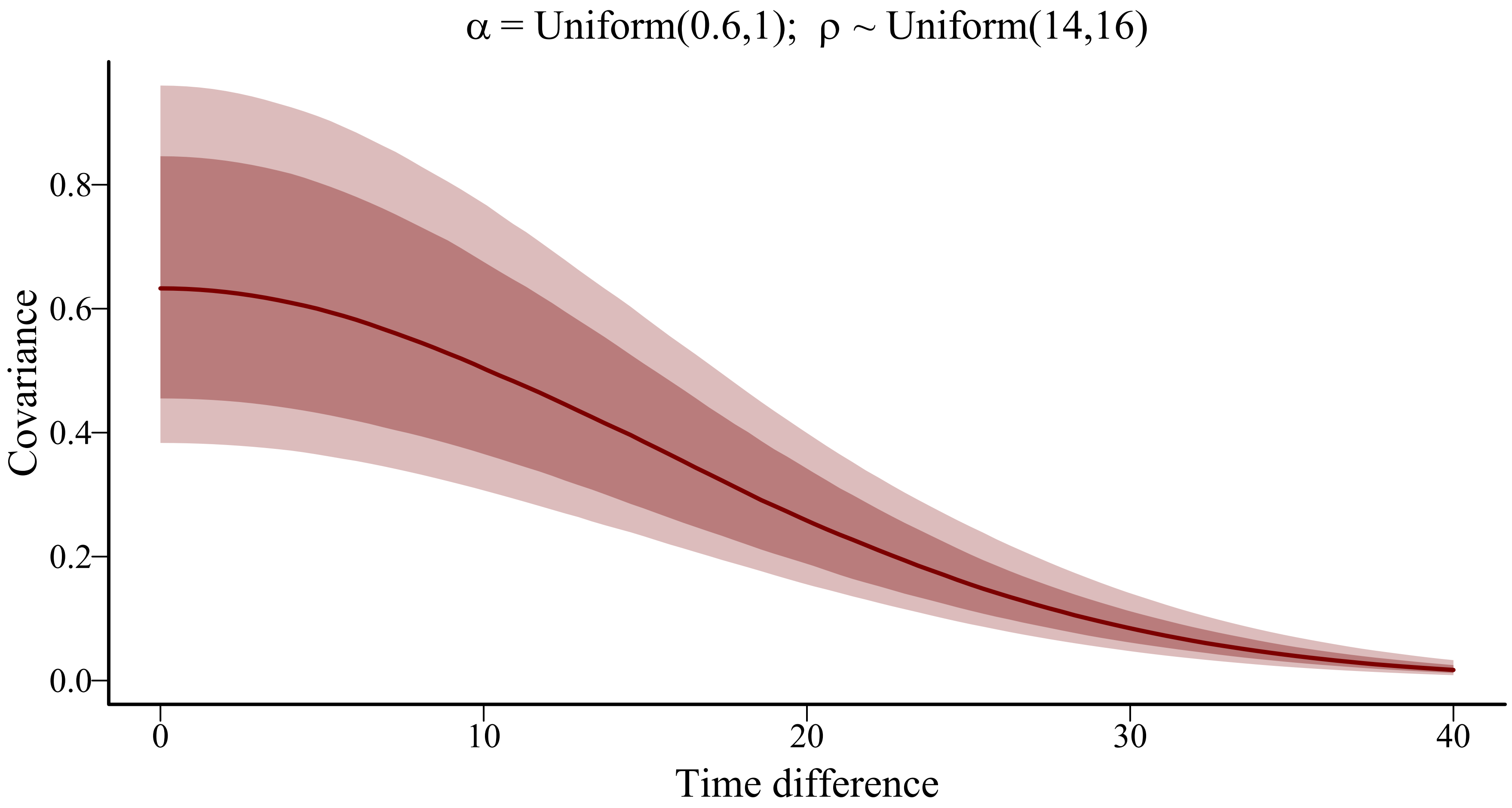
Kernel ⇨ covariance decay



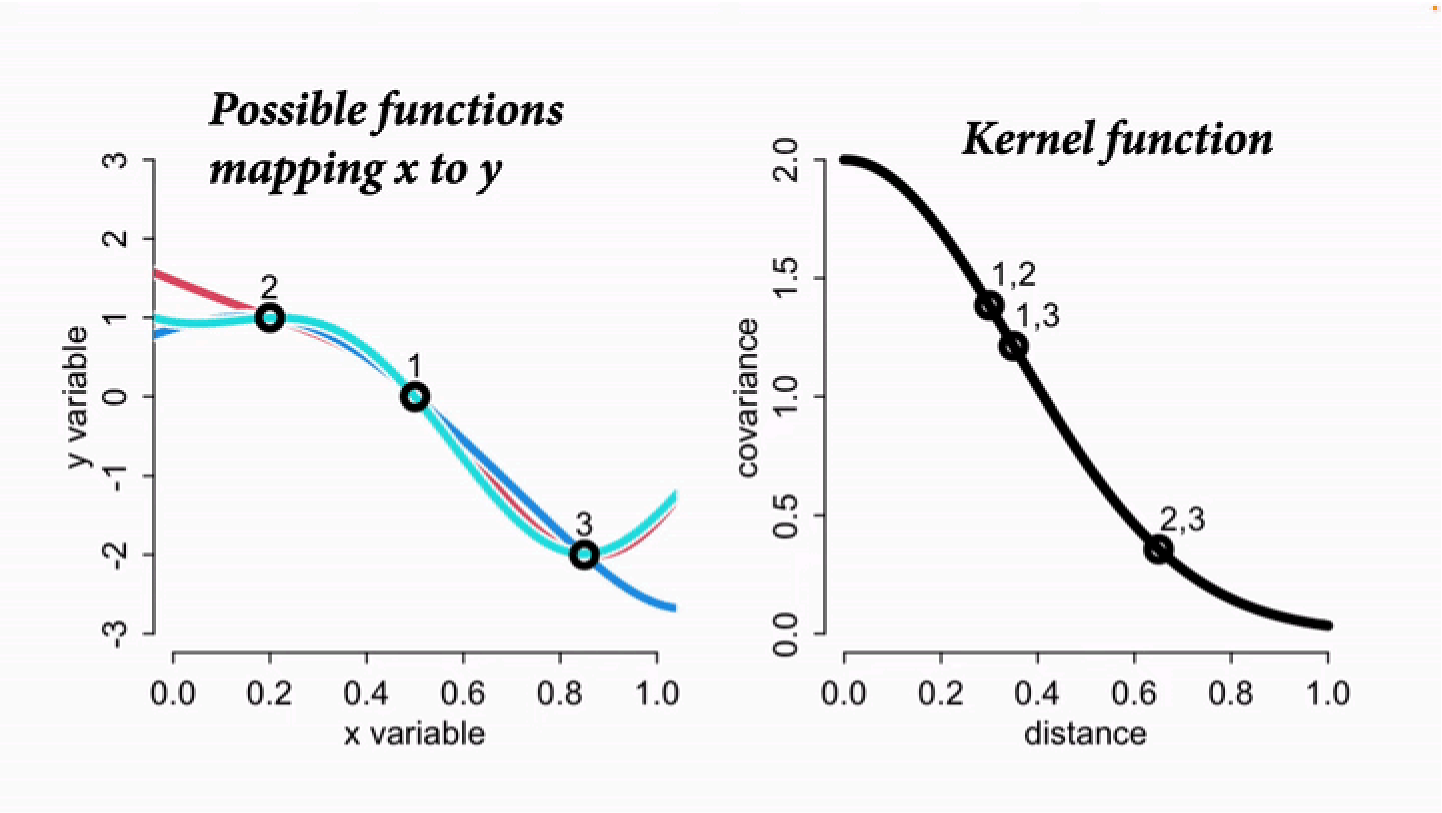
Kernel ⇨ covariance decay



Kernel ⇨ covariance decay



Kernel smoothing in action



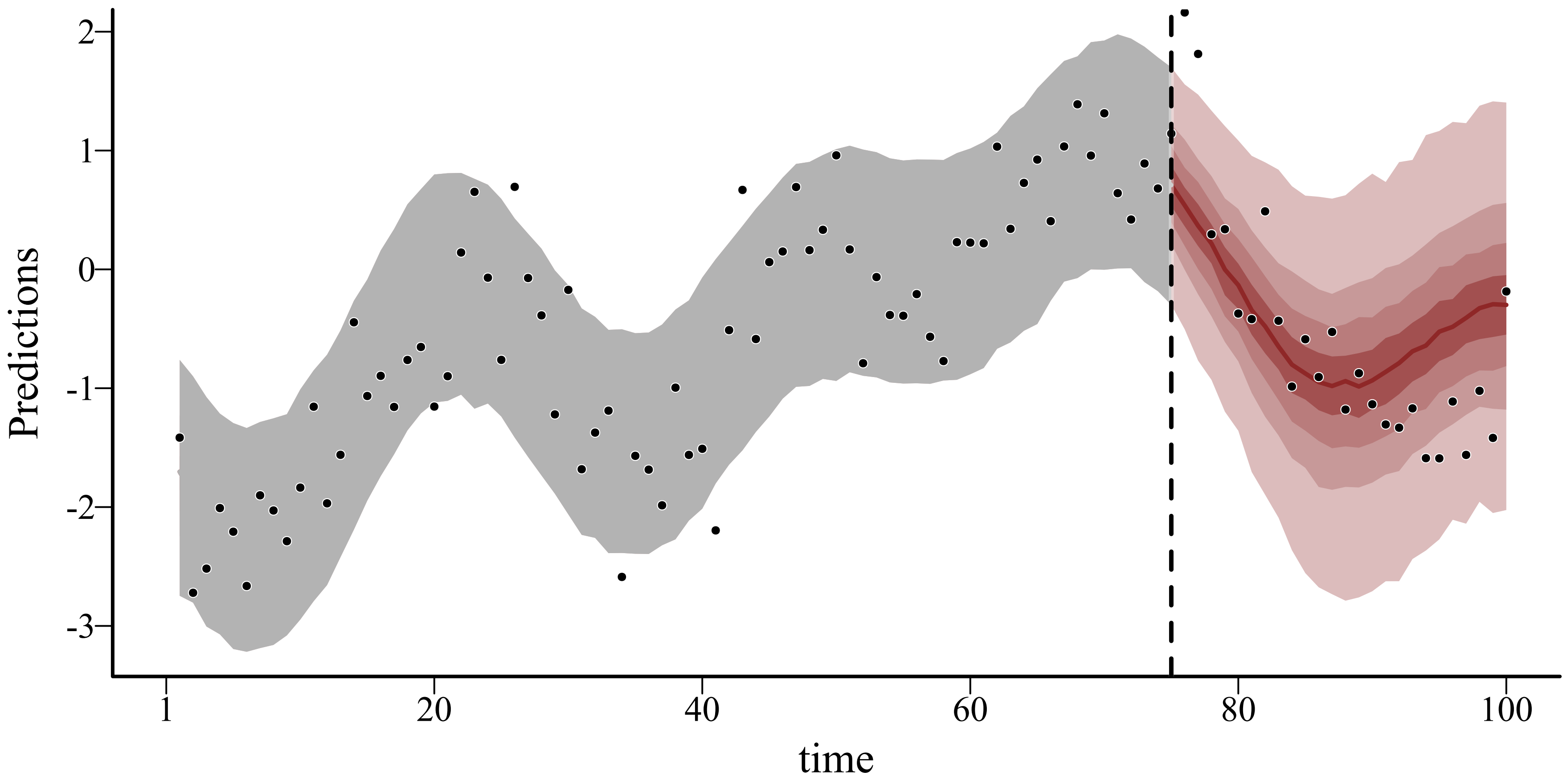
McElreath 2023

A latent GP allows prediction for *any* time point because all we need is the distance to each training time point

The cross-covariance for prediction vs training time points provides the kernel used to extend functions forward in time

Allows GPs to make much better predictions than splines, but at a high computational cost

Global knowledge ✓



Approximating GPs

A quick note that both the `mvgam` and `brms` 📦 's can employ an approximation method to improve computational efficiency for estimating Gaussian Process parameters

Relies on basis expansions to reduce dimensionality of the problem

Details not focus of this lecture, but can be found in this reference

> Riutort-Mayol et al 2023; Practical Hilbert space approximate Bayesian Gaussian processes for probabilistic programming

Both packages use automatic, informative priors for length scales ρ, but these can be changed (more on this in Tutorial 2)

Estimation in brms and mvgam

Use the gp function with `time` as the covariate

```
brm(y ~ x + ... +
      gp(time, c = 5/4, k = 20, scale = FALSE),
    family = poisson(),
    data = data)

mvgam(y ~ x + ... +
        gp(time, c = 5/4, k = 20, scale = FALSE),
      family = poisson(),
      data = data)
```

Requires arguments to determine behaviour of the approximation (`c` and `k`). Good defaults are `5/4` and `20`, but depends on number of timepoints and expected smoothness

No examples here as we will go deeper into GPs in the tutorial

But if you want extra detail, watch this lecture: - Statistical Rethinking 2023 - 16 - Gaussian Processes

Live code example

Dynamic coefficient models

Dynamic coefficients

Major advantage of flexible interfaces such as `brms`, `mgcv` and `mvgam` 📦 's is ability to handle many types of nonlinear effects

These can include smooth functions of covariates, as we have been using so far

But they can also include other types of nonlinearities

- Spatial autocorrelation functions
- Distributed lag functions
- ***Time-varying effects***

Smooth time-varying effects

If a covariate effect changes over time, we'd usually expect this change to be ***smooth***

Splines and Gaussian Processes provide useful tools to estimate these effects

But as we've seen previously, splines will often give poor predictions about how effects will change in the future

In mvgam

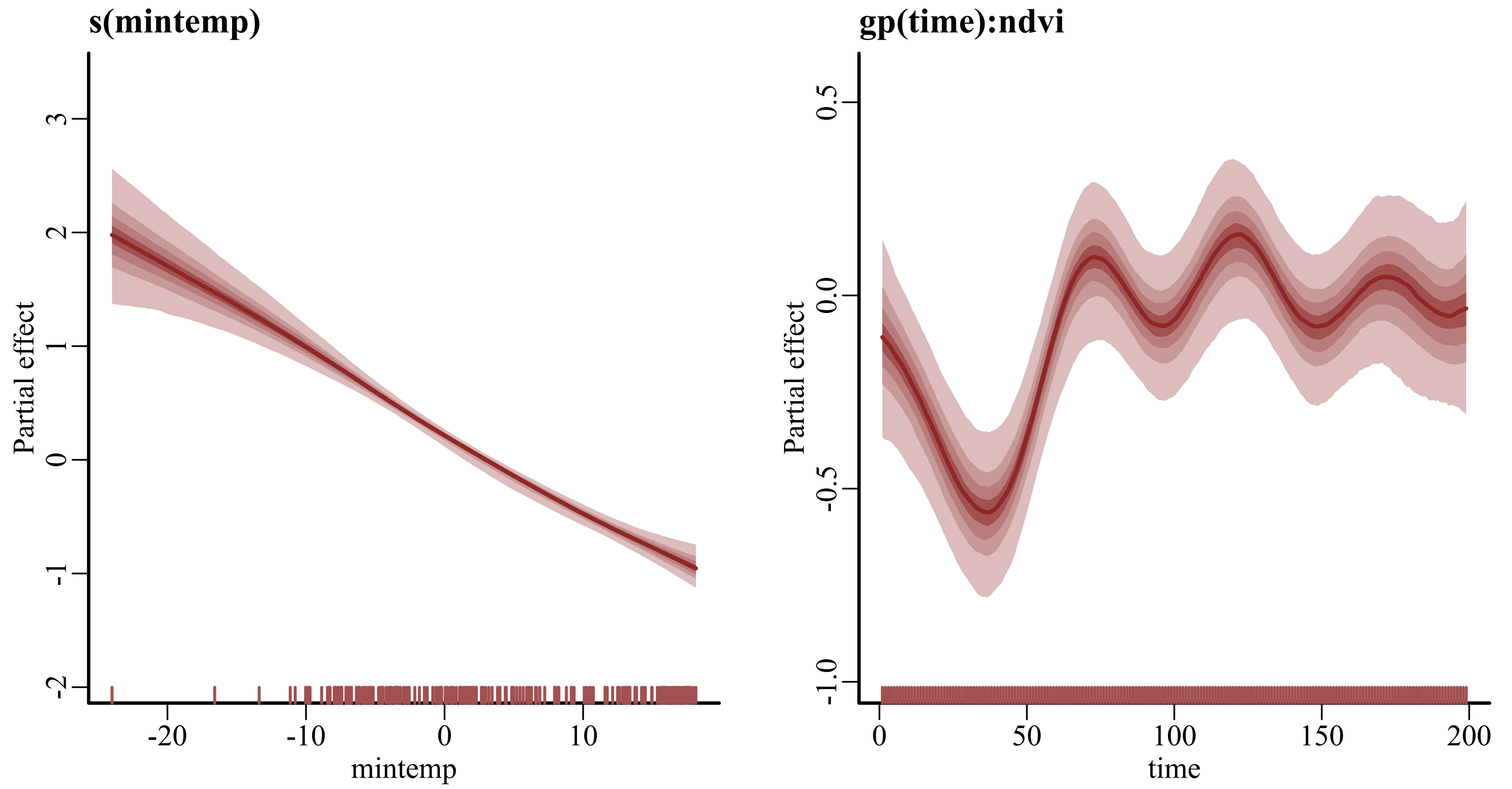
In `mvgam` 📦, use `dynamic` to set up time-varying effects

```r
mod_beta_dyn <- mvgam(relabund ~ s(mintemp, k = 6) +
                                dynamic(ndvi, scale = FALSE, k = 20),
                              family = betar(),
                              data = dm_data)
```

Requires user to set k, as the function is approximated using a low-rank GP smooth from the `brms` 📦

Estimates full uncertainty in GP parameters to yield a squared exponential GP

Estimated smooths



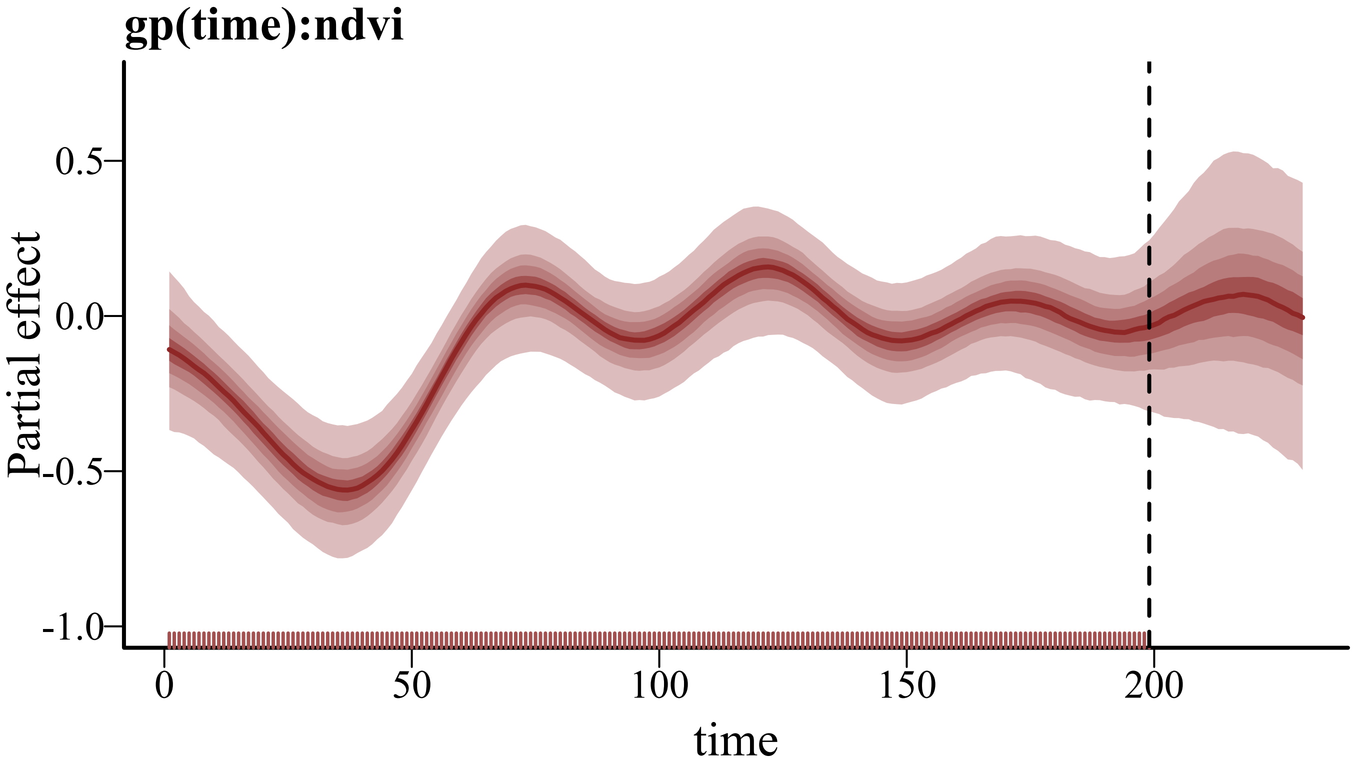
Predicted effects

Code Plot

```
# use mvgam's plot_mvgam_smooth to view predicted effects
plot_mvgam_smooth(mod_beta_dyn, smooth = 2,
                  # datagrid from marginaleffects is useful
                  # to set up prediction scenarios
                  newdata = datagrid(time = 1:230,
                                     model = mod_beta_dyn))
abline(v = max(dm_data$time), lwd = 2, lty = 'dashed')
```

Predicted effects

Code Plot



In brms

In brms 📦, use gp with the by argument

```
brm_beta_dyn <- brm(relabund ~ s(mintemp, k = 6) +
                      gp(time, by = ndvi, c = 5/4, k = 20),
                    family = Beta(),
                    data = dm_data,
                    chains = 4,
                    cores = 4,
                    backend = 'cmdstanr')
```

A GP specifying time-varying effects of ndvi

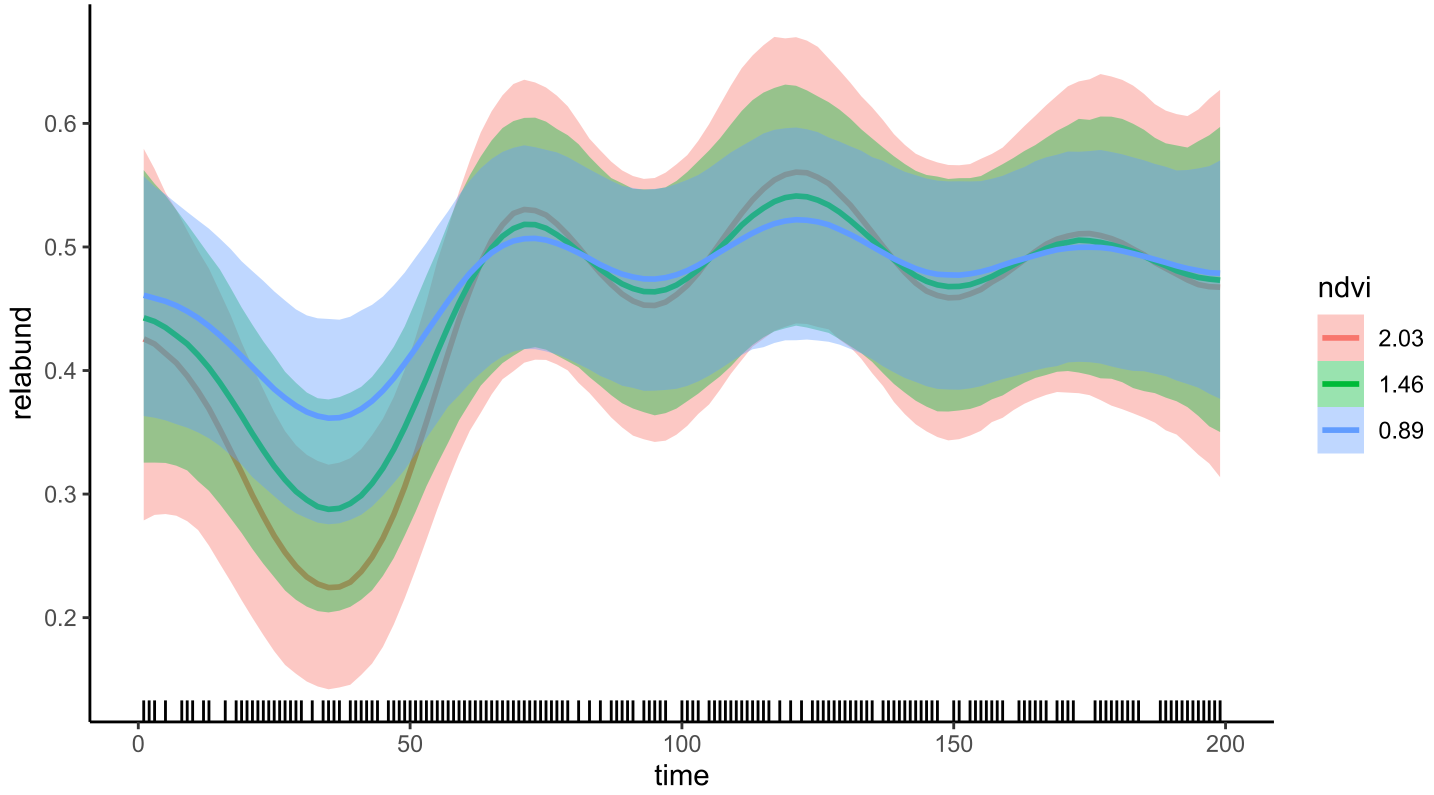
Time-vaying effect

Code Plot

```
# use brms' conditional_effects to view predictions
plot(conditional_effects(brm_beta_dyn, effects = c('time:ndvi')),
                    theme = theme_classic(),
                    mean = FALSE,
                    rug = TRUE)
```

Time-vaying effect

Code Plot



We have seen many ways to handle dynamic components in Bayesian regression models

These flexible processes can capture time-varying effects and give realistic forecasts, while also allowing us to respect the properties of the observations

But how do we evaluate and compare dynamic GAMs / GLMs?

In the next lecture, we will cover

Forecasting from dynamic models

Bayesian posterior predictive checks

Point-based forecast evaluation

Probabilistic forecast evaluation