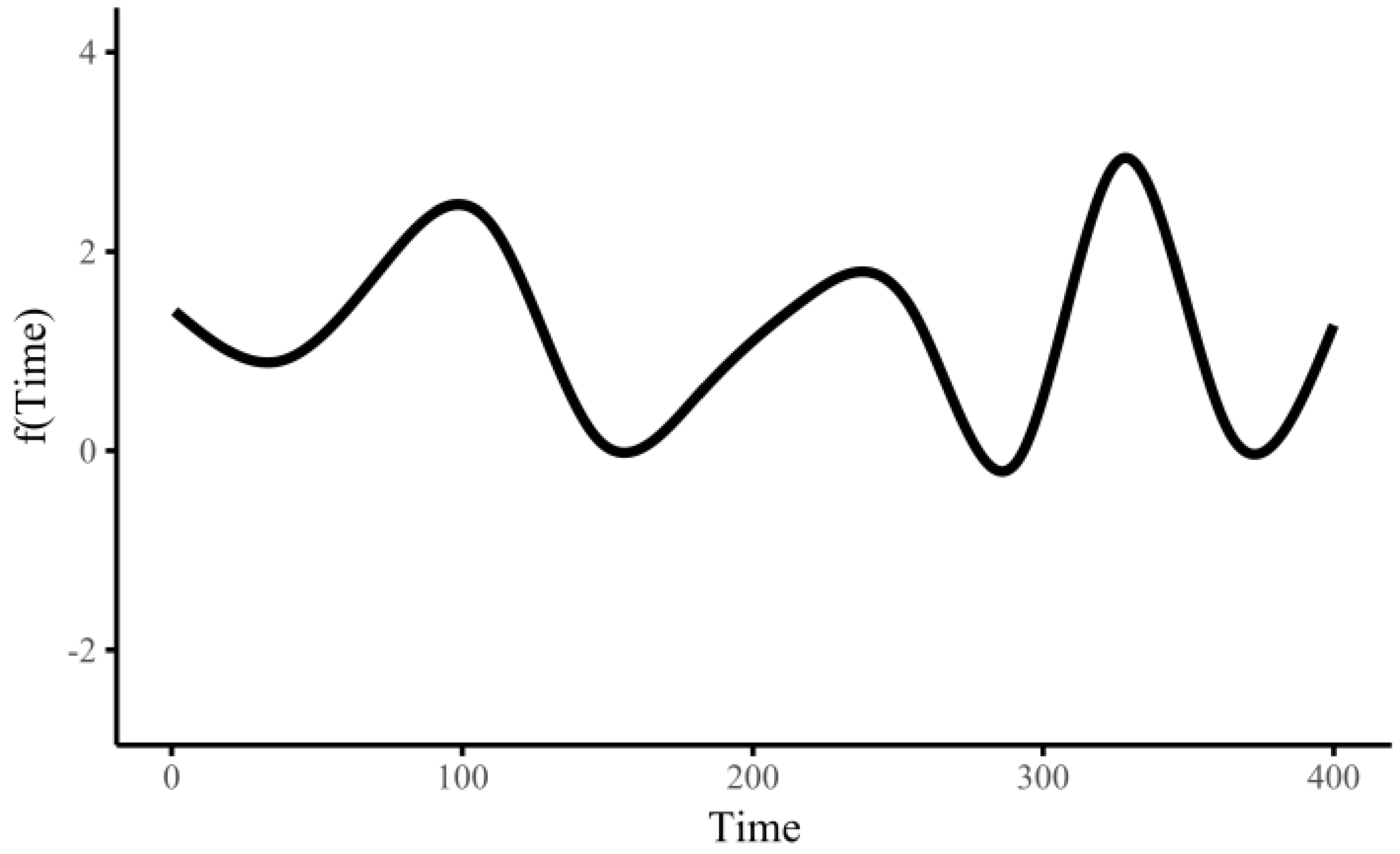
Ecological forecasting in R

Lecture 2: dynamic GLMs and GAMs

Nicholas Clark

School of Veterinary Science, University of Queensland

0900–1200 CET Monday 4th September, 2023



f(Time)
4
2
0
-2
0
100
200
300
400
Time

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:

- An introduction to Bayesian multilevel modeling with `brms`
- Introduction to Generalized Additive Models with and `mgcv`
- Statistical Rethinking 2023 - 04 - Categories & Curves
- Statistical Rethinking 2023 - 12 - Multilevel Models

This lecture's topics

Useful probability distributions for ecologists

Generalized Linear and Additive Models

Temporal random effects

Temporal residual correlation structures

Useful probability distributions

Normal (Gaussian)

$$\boldsymbol{Y} \sim \mathrm{Normal}(\mu, \sigma)$$

Properties

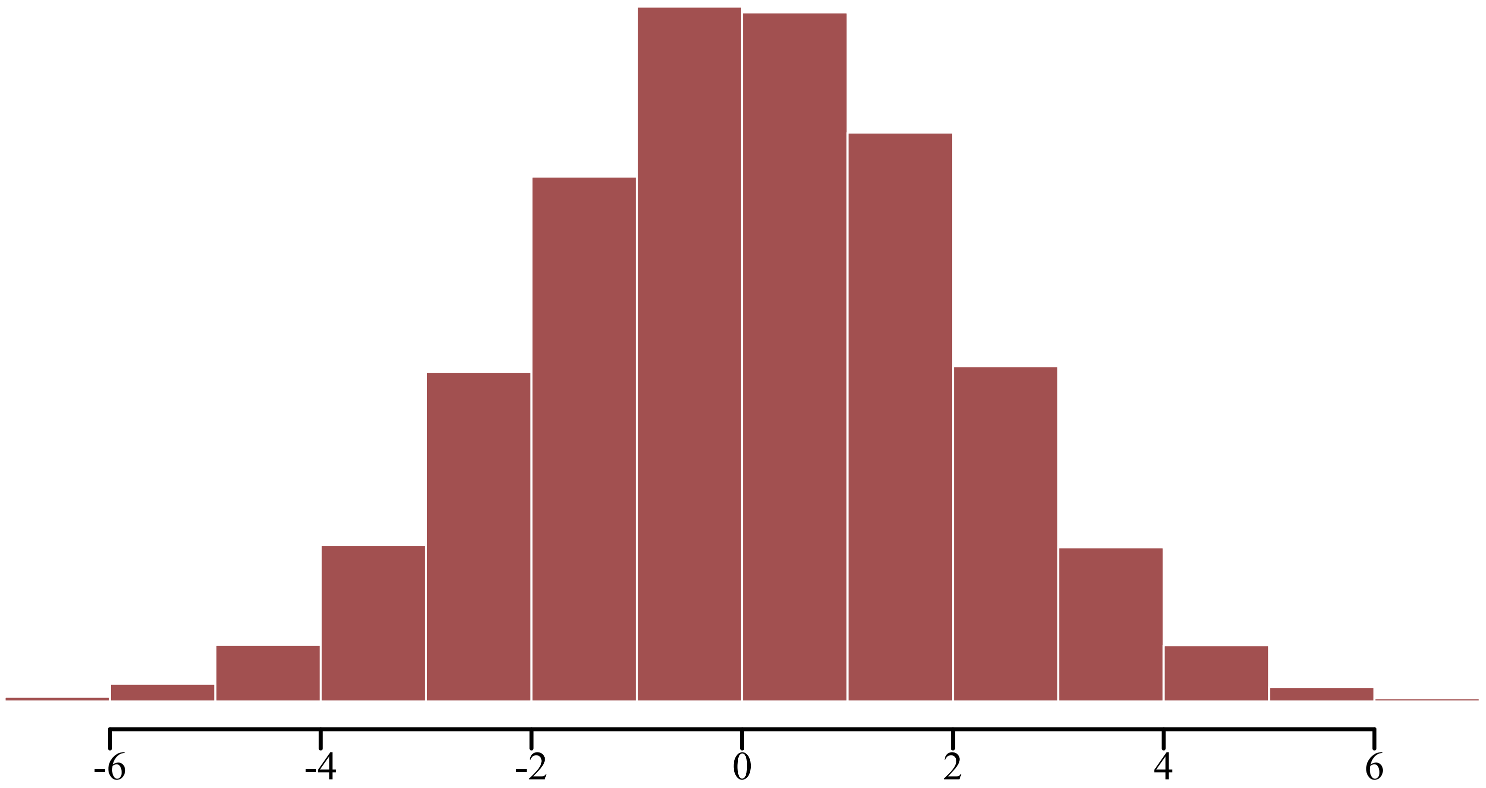
- Real-valued continuous observations (including any decimal)
- Unbounded (supports $-\infty$ to ∞)
- Symmetric spread, controlled by σ, about the mean (μ)

Nearly all common time series models assume this data distribution

- RW, AR, and ARIMA
- ETS and TBATS
- Meta's Prophet 📦

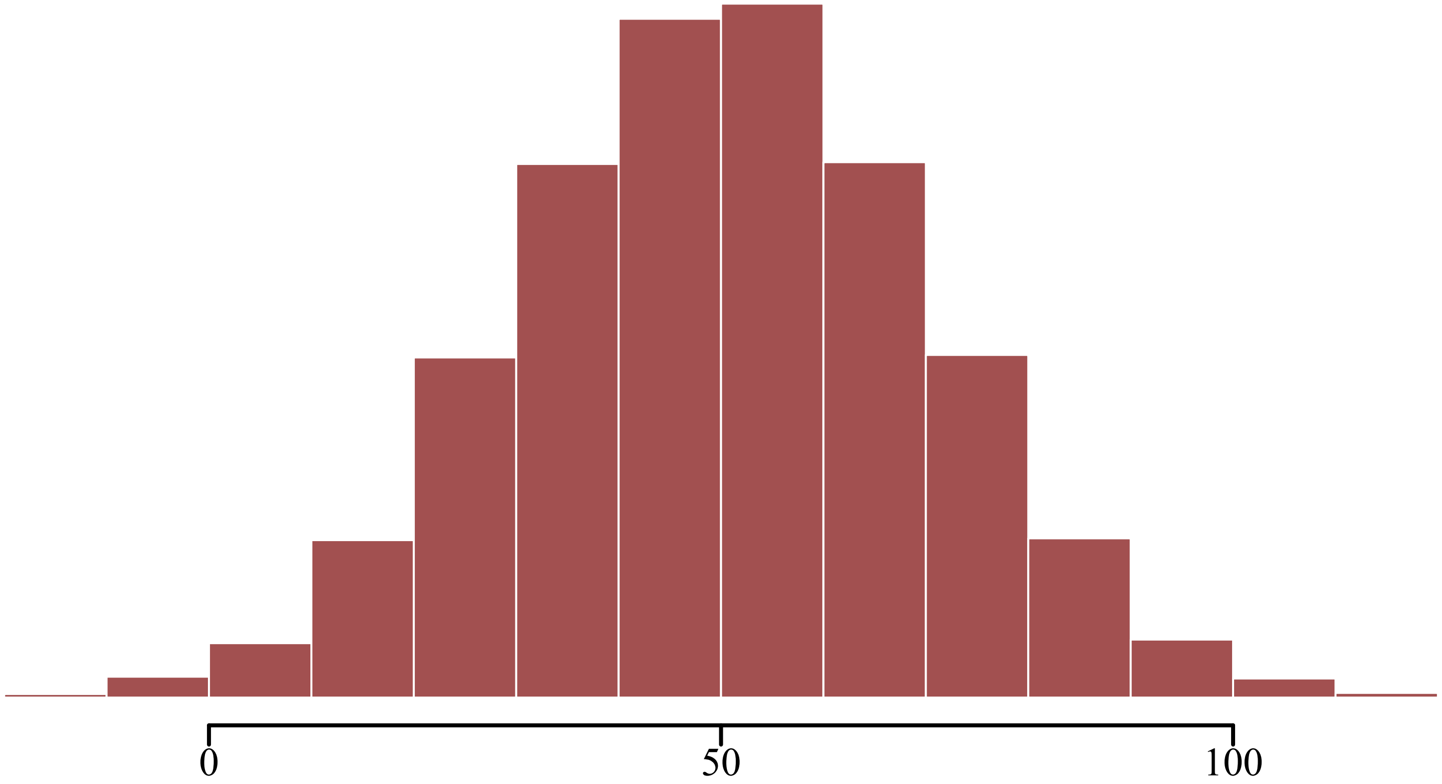
Normal (Gaussian)

$$\boldsymbol{Y} \sim \text{Normal}(0, 2)$$



Normal (Gaussian)

$$\boldsymbol{Y} \sim \mathrm{Normal}(50, 20)$$



Linear regression

It is common to estimate linear predictors of μ with ***regression***

$$\boldsymbol{Y} \sim \mathrm{Normal}(\alpha + \beta * \boldsymbol{X}, \sigma)$$

Where:

- $\boldsymbol{X}$ represents a design matrix of covariates that contribute linearly to variation in μ
- α is an intercept coefficient
- β is a vector of regression coefficients
- σ controls the spread of the errors about μ

ETS(A,A,A)

Exponential smoothing with additive components for trend, seasonality and error assumes a Normal (Gaussian) distribution

$$\boldsymbol{Y}_t \sim \text{Normal}(l_{t-1} + b_{t-1} + s_{t-m}, \sigma)$$

Where:

l gives the value of the level

b gives the value of the trend

s gives the value of the seasonality

m represents the seasonal period

ARMA(p, q)

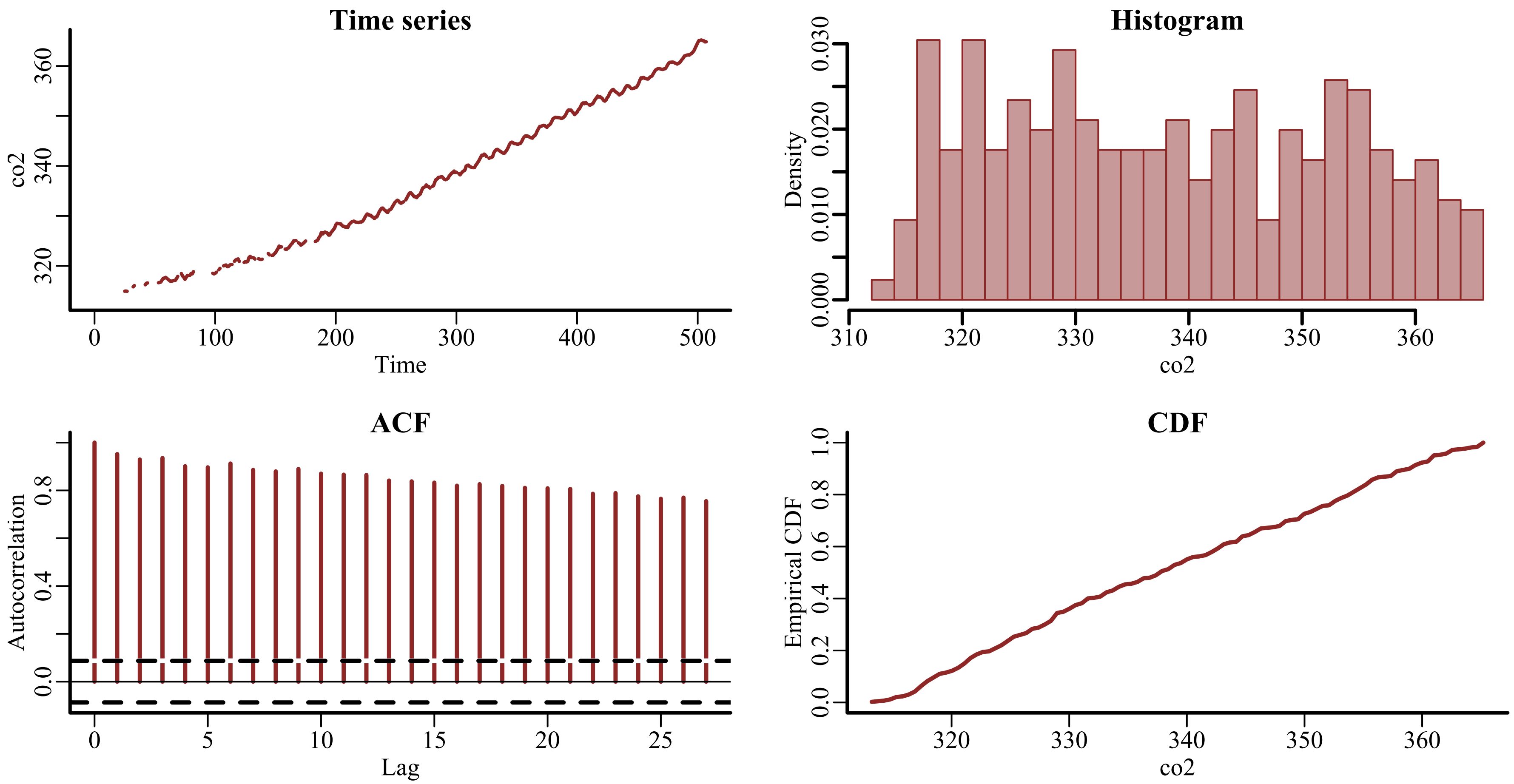
ARMA processes also assume Normality

$$\boldsymbol{Y}_t \sim \mathrm{Normal}(c + \sum_{k=1}^{p} \phi_k(\boldsymbol{Y}_{t-k} - c) + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}, \sigma)$$

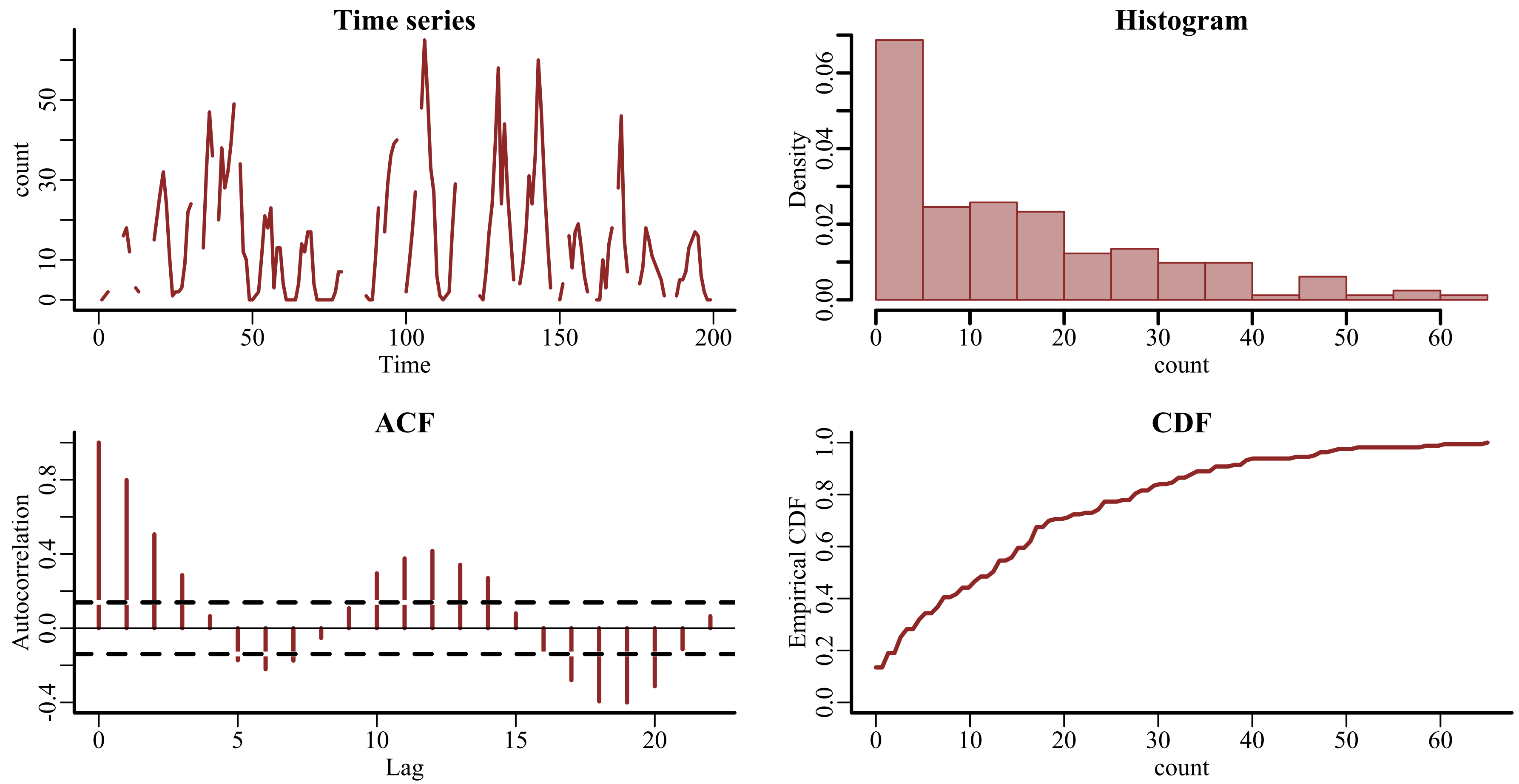
Where:

- c is a constant (drift parameter)
- p and q gives orders of AR and MA processes
- ϕ and θ are AR and MA coefficients
- ϵ are historical errors (which are $\mathrm{Normal}(0, \sigma)$)

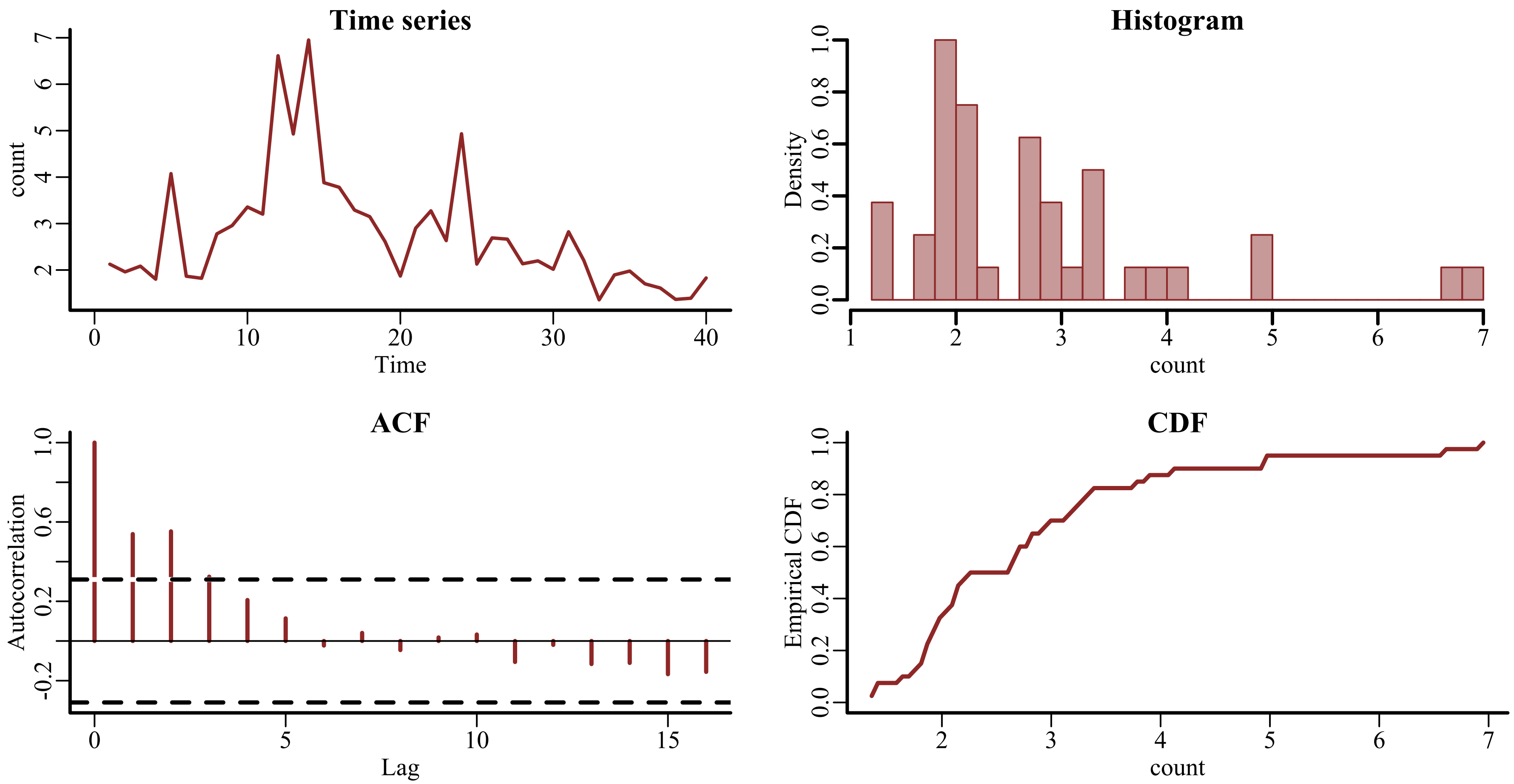
But most real-world ecological observations, including time series, *are not Gaussian*



Properties of monthly CO2 measurement time series at the South Pole



Properties of lunar monthly Desert Pocket Mouse capture time series from a long-term monitoring study in Portal, Arizona, USA



Properties of annual American kestrel abundance time series in British Columbia, Canada

“If our data contains small counts (0,1,2,...), then we need to use forecasting methods that are more appropriate for a sample space of non-negative integers.

Such models are beyond the scope of this book”

Hyndman and Athanasopoulos, Forecasting Principles and Practice

Ok. So now what?

Poisson

$$\boldsymbol{Y} \sim \text{Poisson}(\lambda)$$

Properties

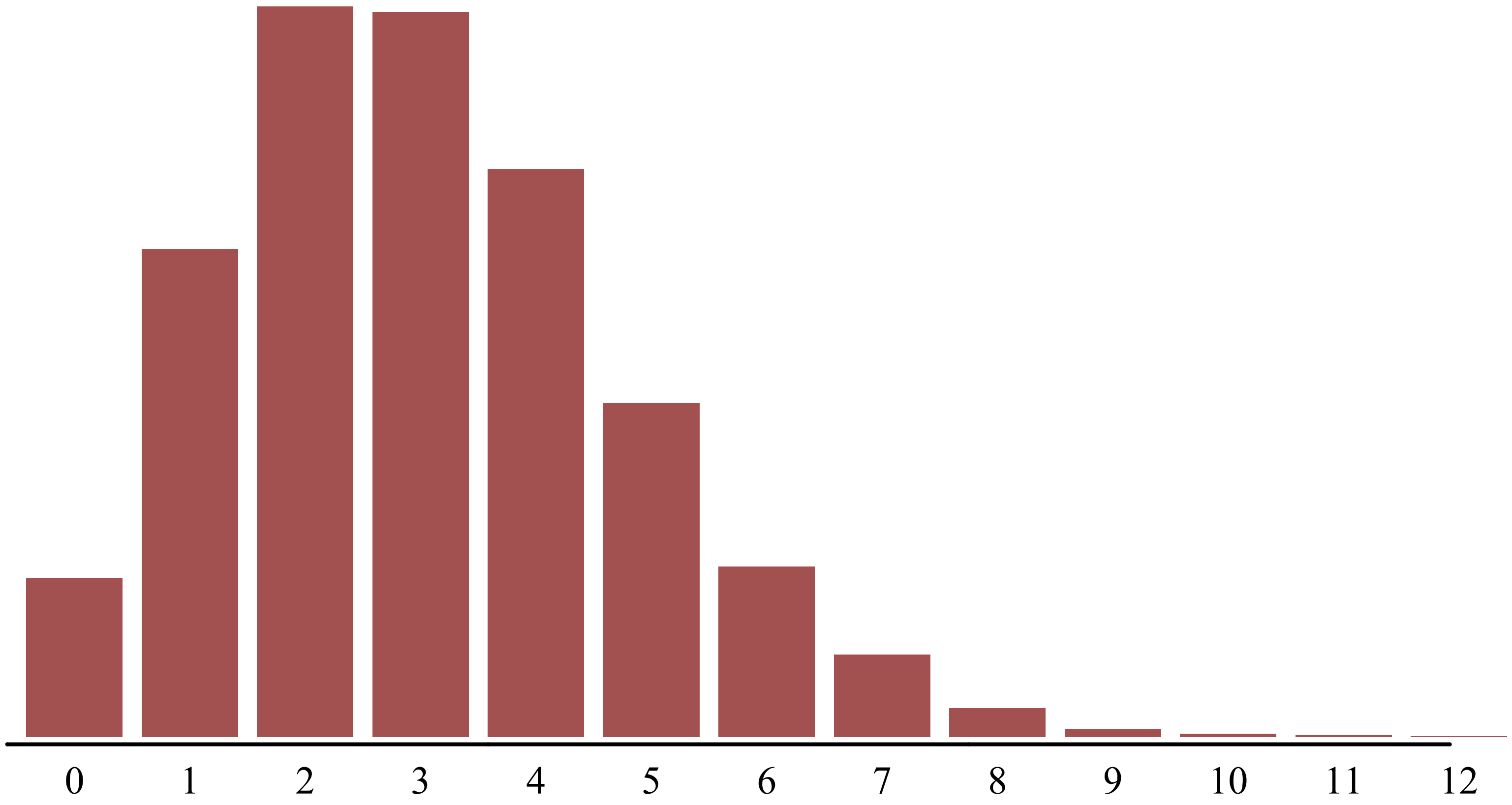
- Discrete, integer-valued observations (including 0)
- Lower bound (supports 0 to ∞)
- mean = variance = λ

Virtually no time series models support this distribution

- Most analysts use `log` or `Box-Cox` transformation
- But see the `tscount` 📦

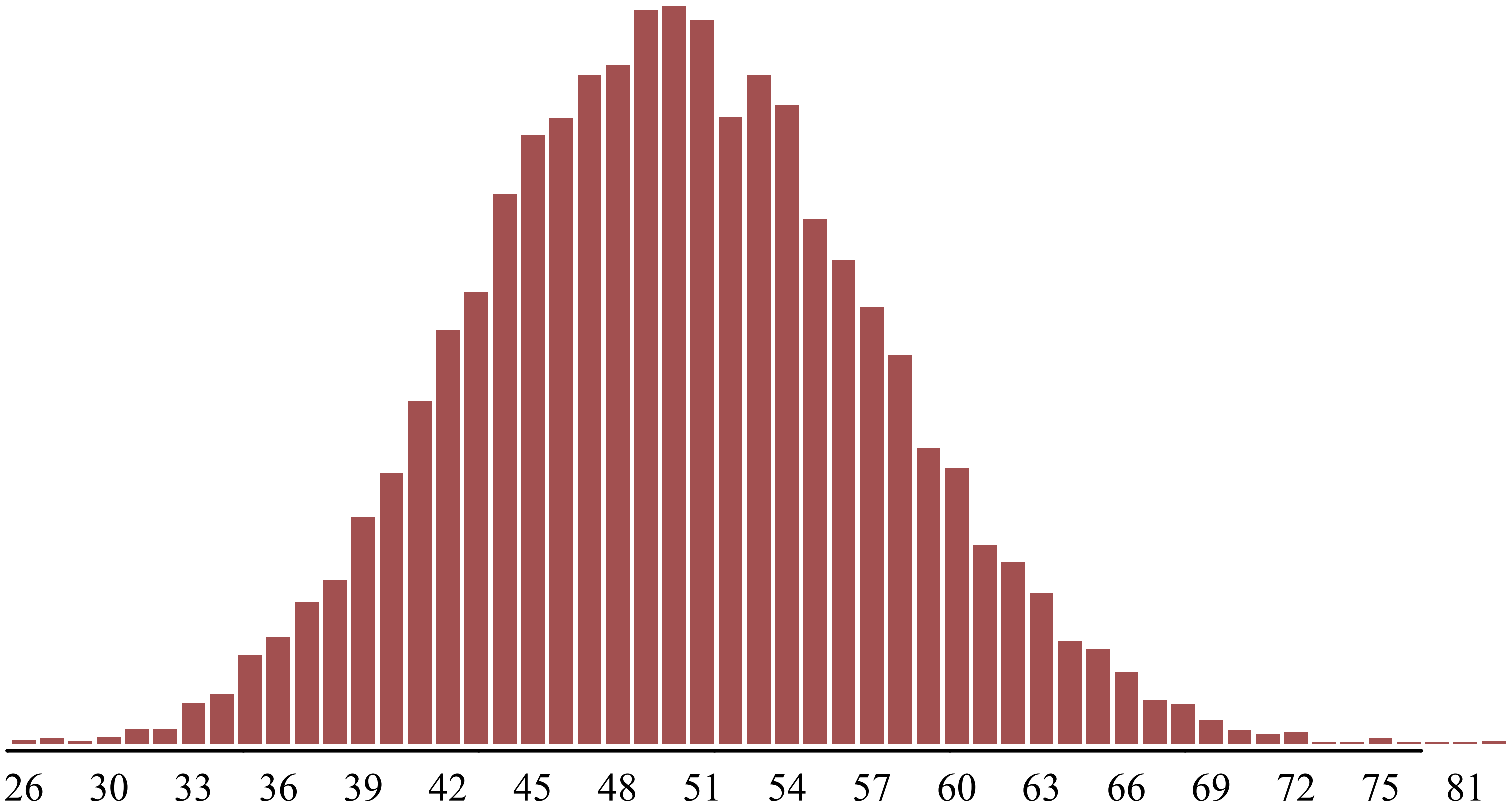
Poisson

$$Y \sim \text{Poisson}(3)$$



Poisson

$$\boldsymbol{Y} \sim \text{Poisson}(50)$$



How can we model non-Normal data using regression?

Generalized linear models

Linear regression can't be trusted to give sensible predictions for non-negative count data (or other types of bounded / discrete / non-Normal data)

We can do better by choosing distributions that obey the constraints on our outcome variables

The idea is to ***generalize*** the linear regression by replacing parameters from other probability distributions with linear models

This requires a ***link function*** that transforms from the unbounded scale of the linear predictor to a scale that is appropriate for the parameters being modeled

Modelling the mean

Most GLMs are used to model the conditional mean (μ)

$$\mathbb{E}(\boldsymbol{Y}|\boldsymbol{X}) = \mu = g^{-1}(\alpha + \boldsymbol{X}\beta)$$

Where:

- $\mathbb{E}$ is the *expected value* of $\boldsymbol{Y}$ conditional on $\boldsymbol{X}$
- g^{-1} is the *inverse* of the link function
- α is an intercept coefficient
- β is a vector of regression coefficients

Poisson GLM

A Poisson GLM models the conditional mean with a *log* link

$$
\begin{aligned}
\boldsymbol{Y}_i &\sim \text{Poisson}(\lambda_i) \\
log(\lambda_i) &= \boldsymbol{X}_i\beta \\
&= \alpha + \beta_1\boldsymbol{x}_{1i} + \beta_2\boldsymbol{x}_{2i} + \cdots + \beta_j\boldsymbol{x}_{ij}
\end{aligned}
$$

Where:

- $\boldsymbol{X}_i$ is the matrix of predictor values at observation i
- α is an intercept coefficient
- β is a vector of regression coefficients
- $\mathbb{E}(\boldsymbol{Y}_i|\boldsymbol{X}_i) = exp(\alpha + \boldsymbol{X}_i\beta)$

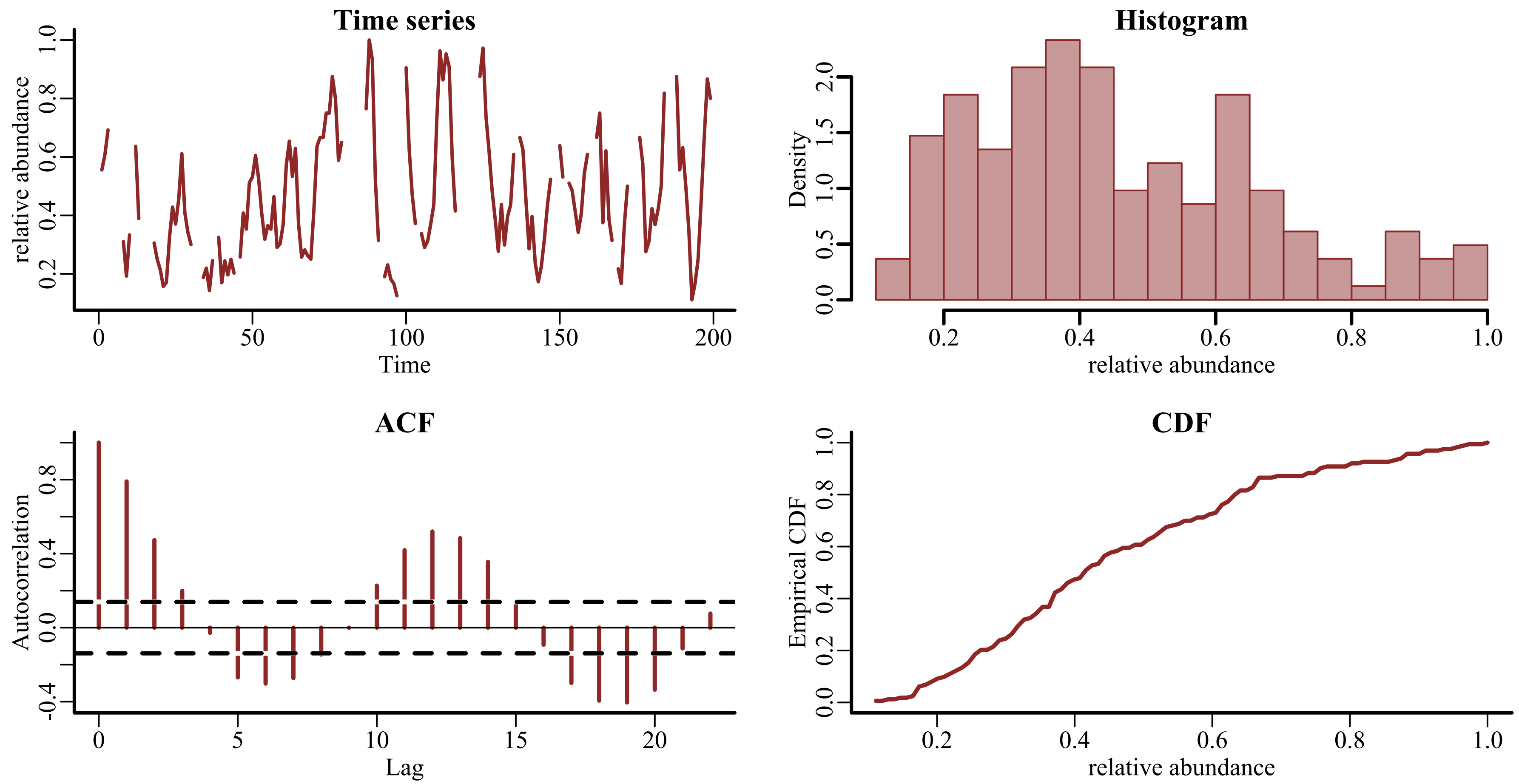
Poisson GLM

A Poisson GLM models the conditional mean with a *log* link

$$\begin{aligned} \boldsymbol{Y}_i &\sim \text{Poisson}(\lambda_i) \\ log(\lambda_i) &= \boldsymbol{X}_i\beta \\ &= \alpha + \beta_1\boldsymbol{x}_{1i} + \beta_2\boldsymbol{x}_{2i} + \cdots + \beta_j\boldsymbol{x}_{ij} \end{aligned}$$

The ***linear predictor component can be hugely flexible***, as we will see in later slides

What if our data are proportional instead?



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

Beta

$$\boldsymbol{Y} \sim \text{Beta}(\mu, \phi)$$

Properties

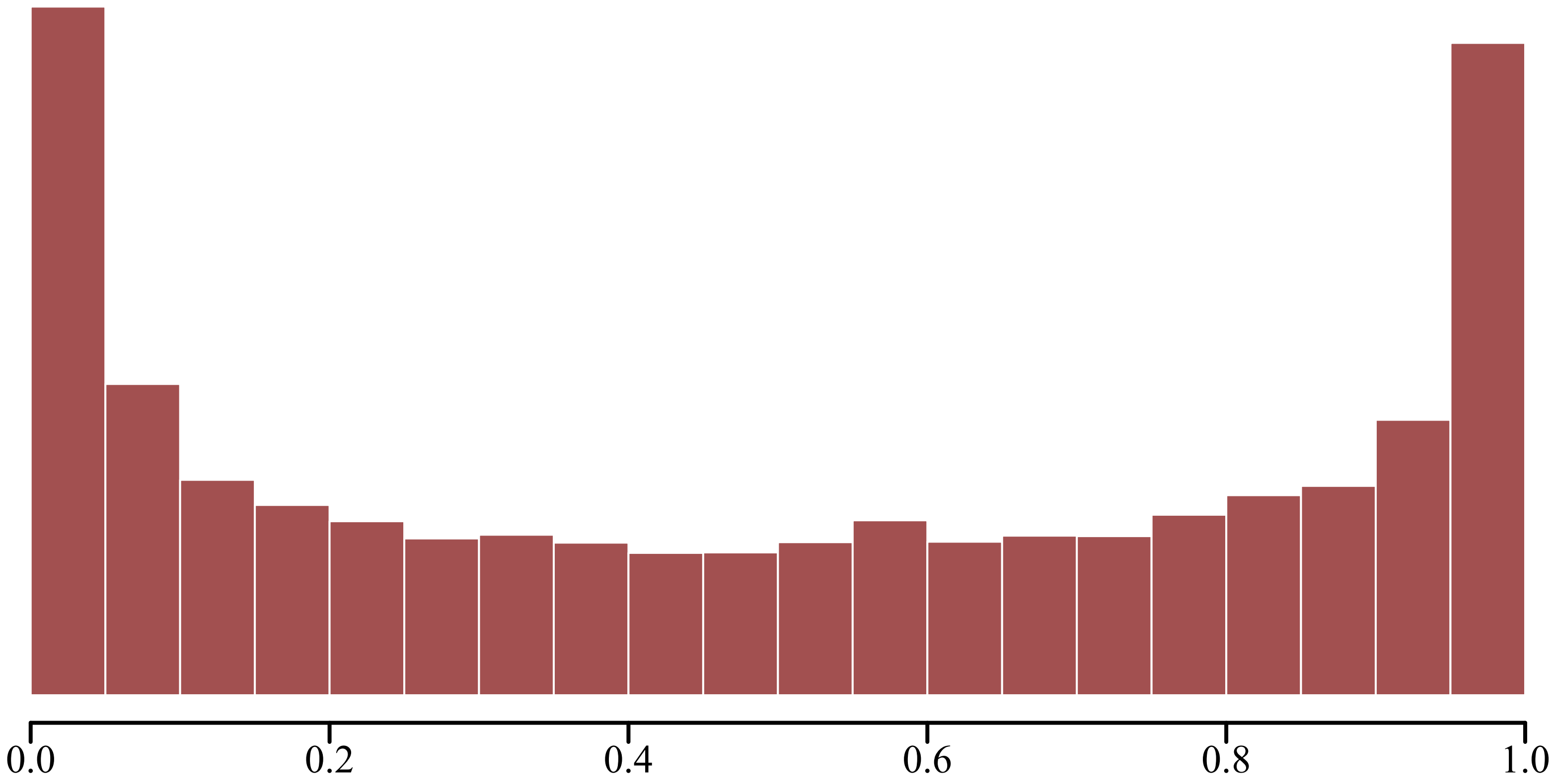
- Real-valued continuous observations (including any decimal)
- Both lower and upper bounds (supports 0 to 1, noninclusive)
- mean = μ
- precision = ϕ (higher ϕ = a tighter spread about μ)

Virtually no time series models support this distribution

- Most analysts would use `logit` transformation

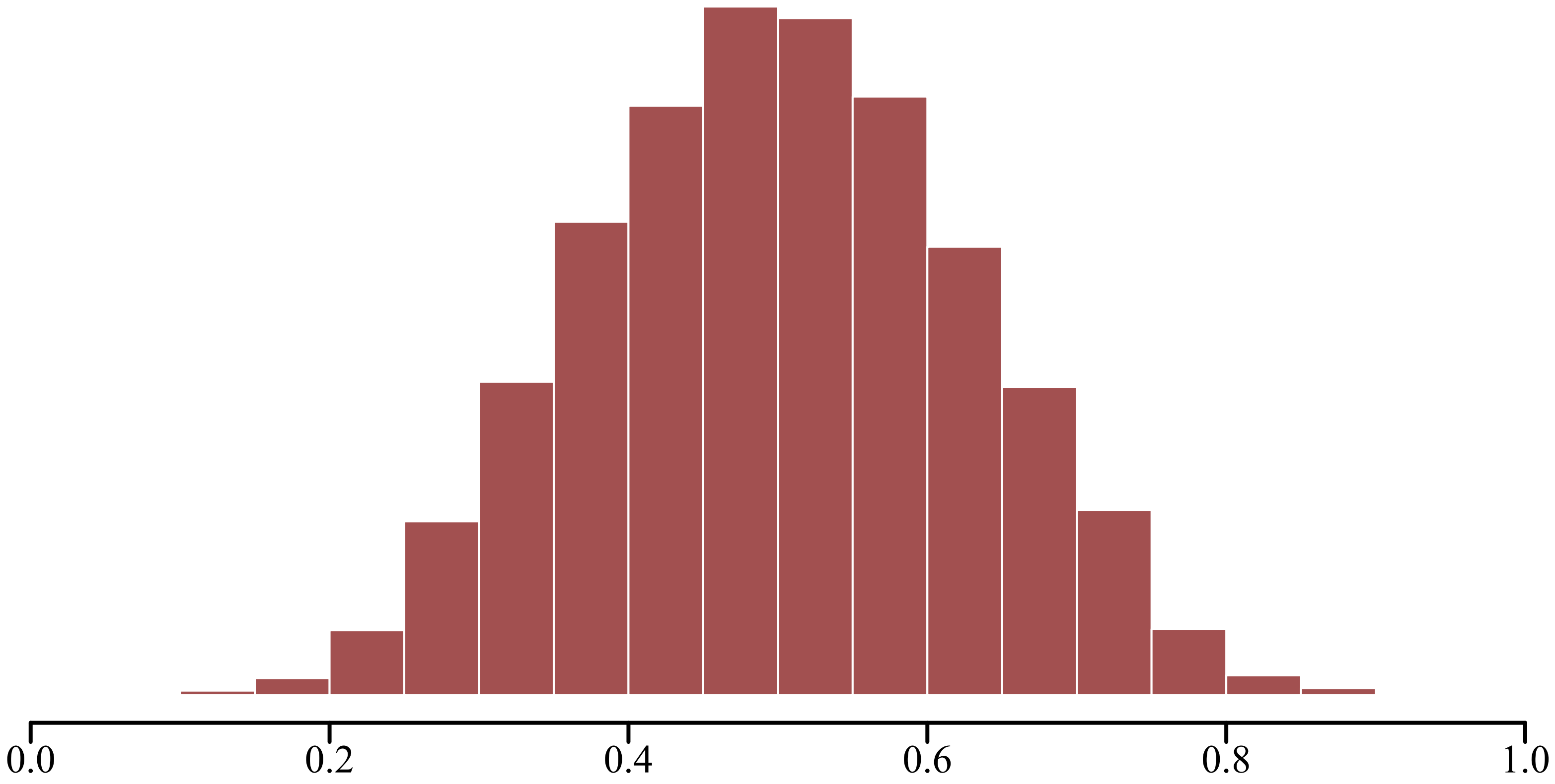
Beta

$$Y \sim \text{Beta}(0.5, 1)$$



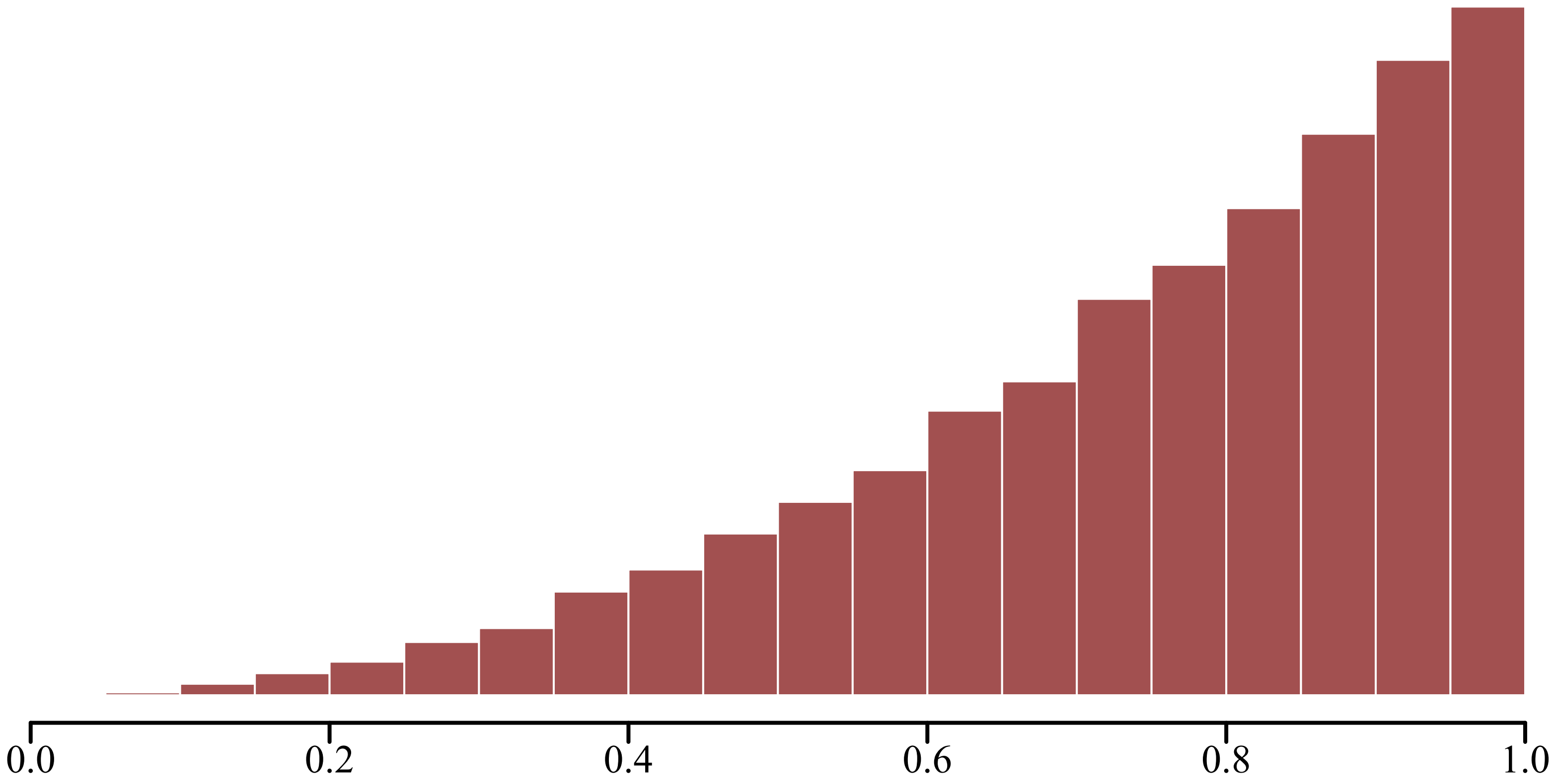
Beta

$$\boldsymbol{Y} \sim \mathrm{Beta}(0.5, 15)$$



Beta

$$Y \sim \text{Beta}(0.75, 4)$$



Beta GLM

A Beta GLM models the conditional mean with a *logit* link

$$
\begin{aligned}
\boldsymbol{Y}_i &\sim \mathrm{Beta}(\mu_i, \phi) \\
logit(\mu_i) &= \boldsymbol{X}_i \beta \\
&= \alpha + \beta_1 \boldsymbol{x}_{1i} + \beta_2 \boldsymbol{x}_{2i} + \dots + \beta_j \boldsymbol{x}_{ij}
\end{aligned}
$$

Where:

$\boldsymbol{X}_i$ is the matrix of predictor values at observation i

α is an intercept coefficient

β is a vector of regression coefficients

$\mathbb{E}(\boldsymbol{Y}_i | \boldsymbol{X}_i) = logit^{-1}(\alpha + \boldsymbol{X}_i \beta)$

Some other relevant distributions

Many other useful GLM probability distributions exist. Some of these include:

- ***Negative Binomial*** – overdispersed integers in $(0, 1, 2, \ldots)$
- ***Bernoulli*** – presence-absence data in $\{0, 1\}$
- ***Student's T*** – heavy-tailed (skewed) real values in $(-\infty, \infty)$
- ***Lognormal*** – heavy-tailed (right skewed) real values in $(0, \infty)$
- ***Gamma*** – lighter-tailed (less skewed) real values in $(0, \infty)$
- ***Multinomial*** – integers representing K unordered categories in $(0, 1, \ldots, K)$
- ***Ordinal*** – integers representing K ordered categories in $(0, 1, \ldots, K)$

GLMs allow us to build models that respect the bounds and distributions of our observed data.

But there are many other properties we'd like to model

Remember these?

Temporal autocorrelation

Lagged effects

Non-Gaussian data and missing observations

Measurement error

Time-varying effects

Nonlinearities

Multi-series clustering

Remember these?

Temporal autocorrelation

Lagged effects

Non-Gaussian data and missing observations

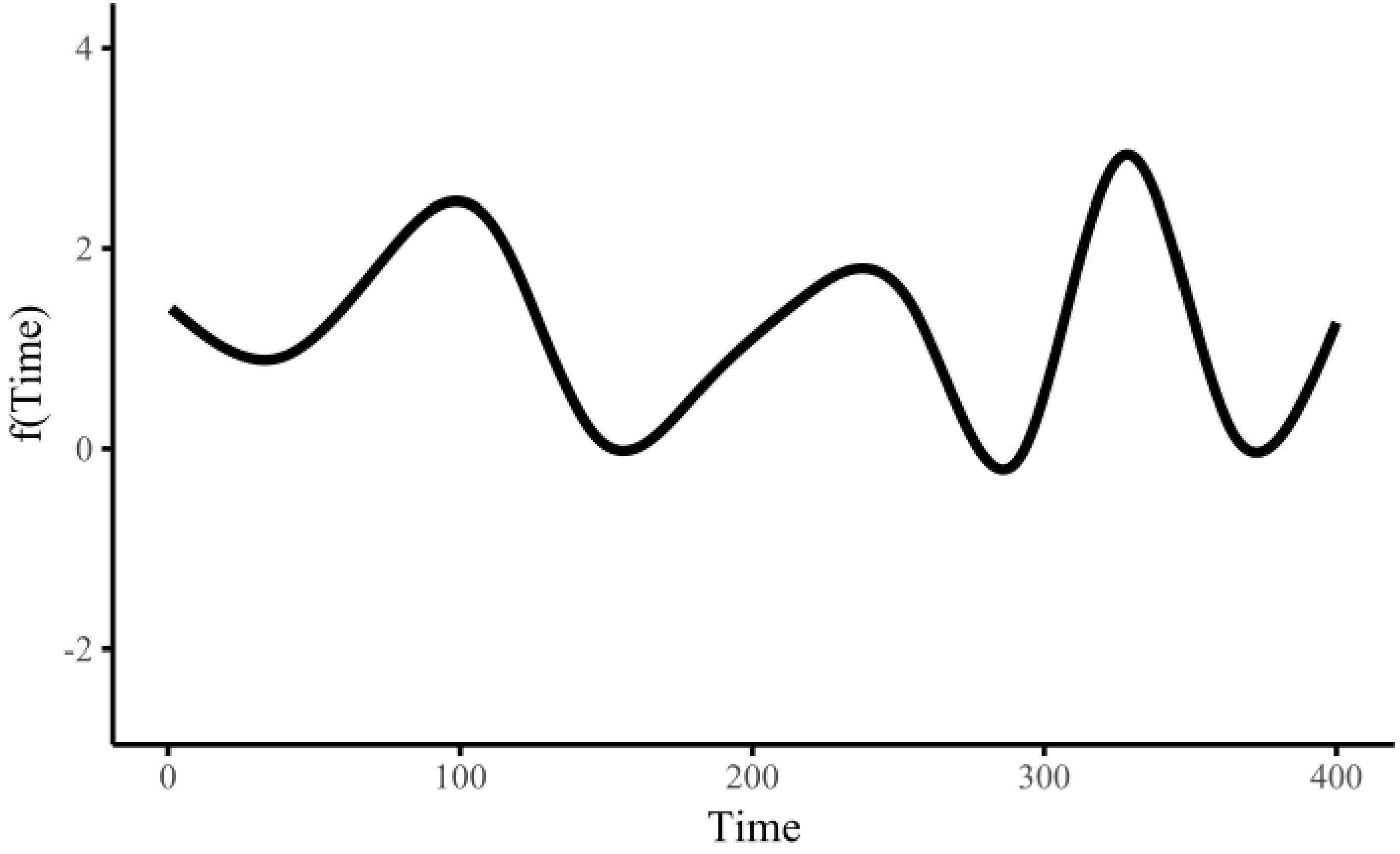
Measurement error

Time-varying effects

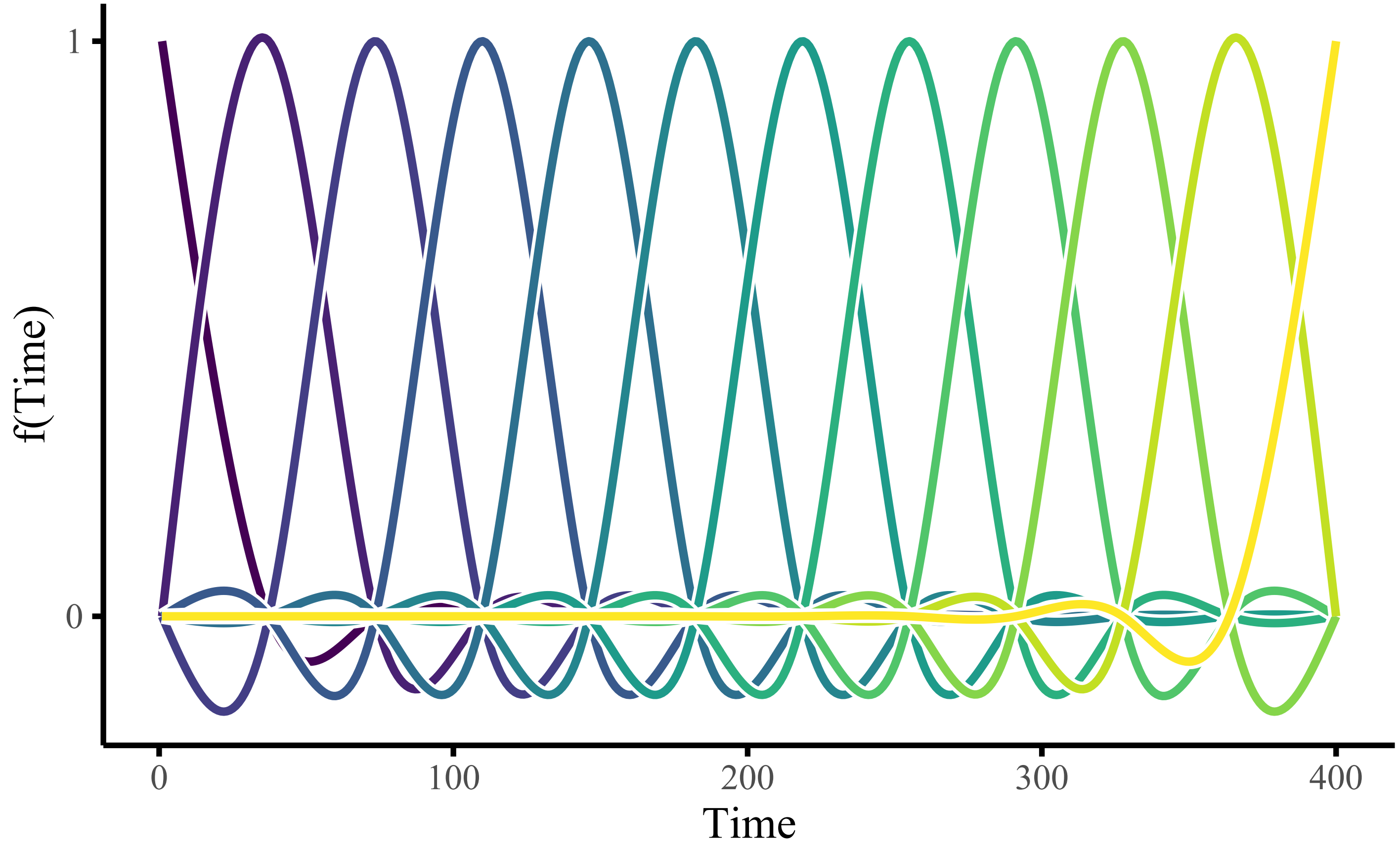
Nonlinearities

Multi-series clustering

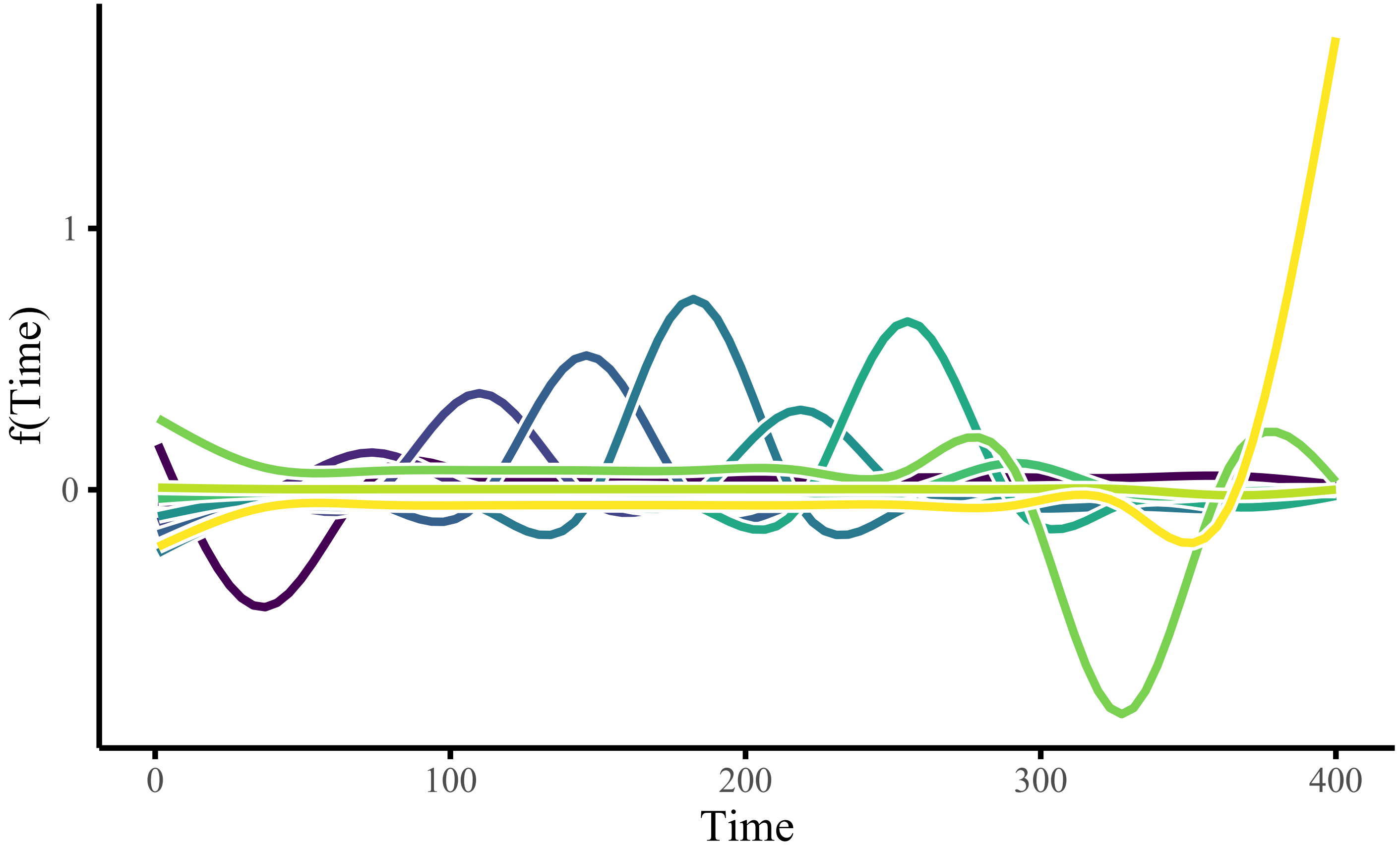
GAMs use splines ...



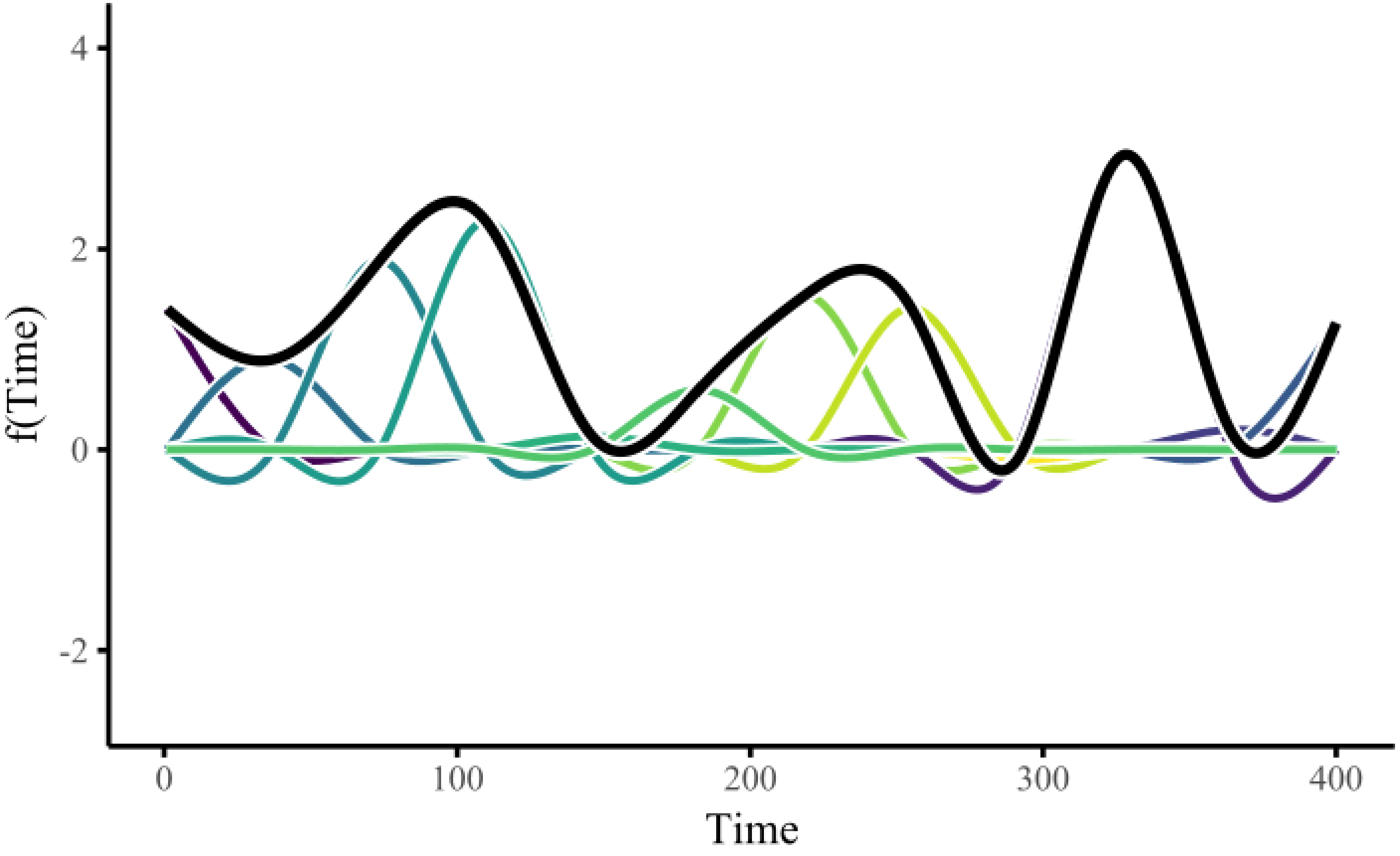
... made of basis functions



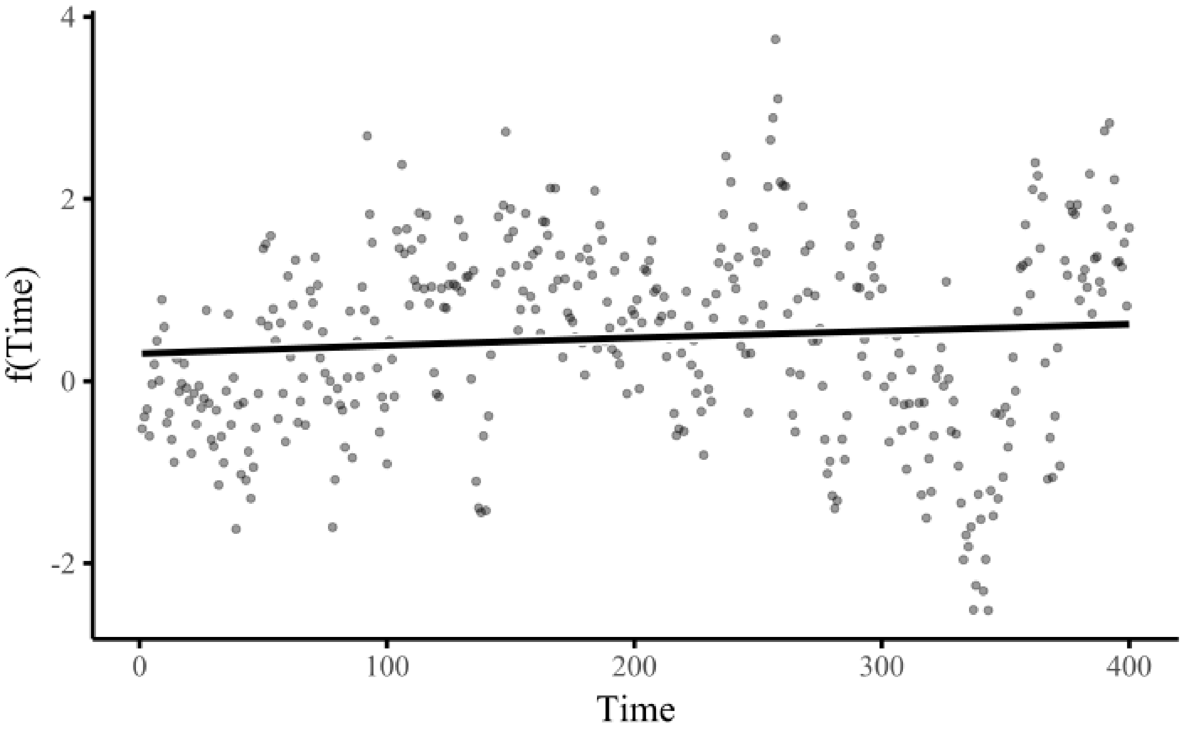
Weighting basis functions ...

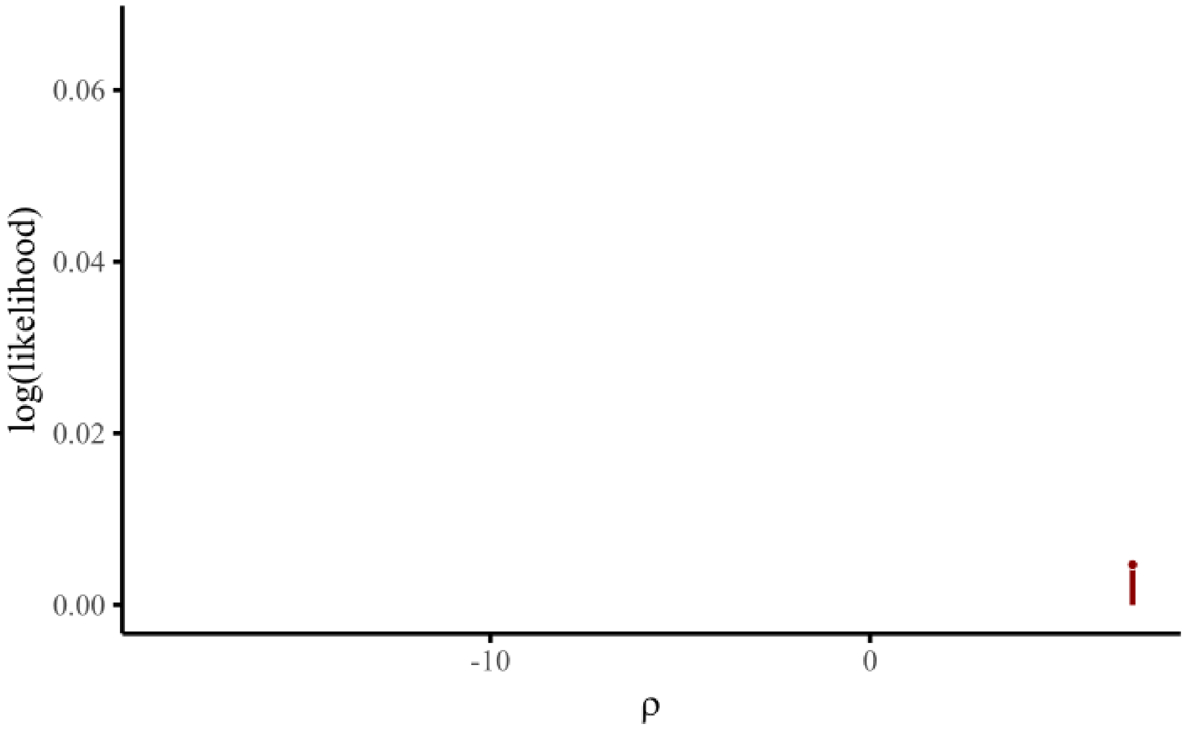


... gives a spline ($f(x)$)

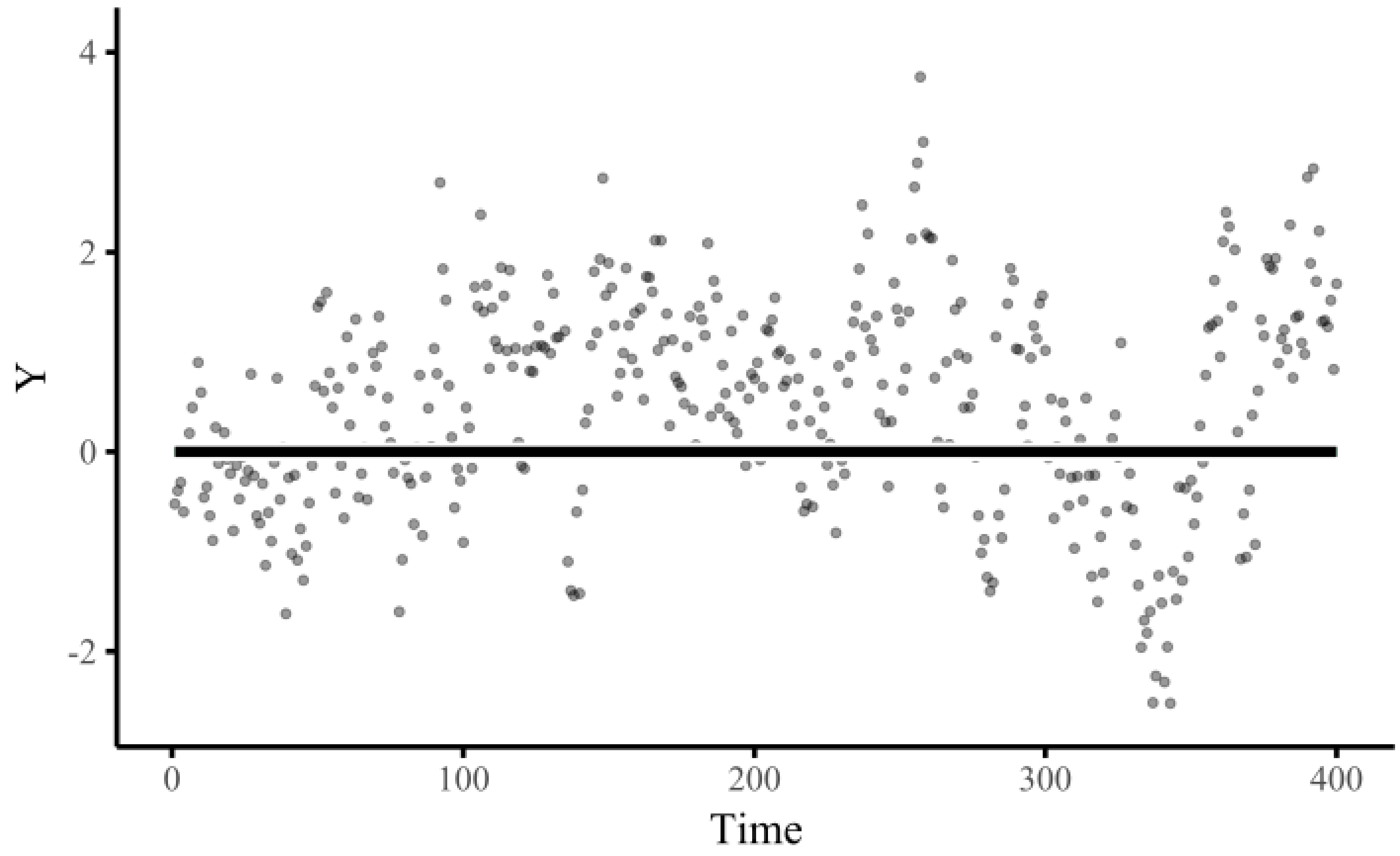


Penalize $f''(x)$ to learn weights



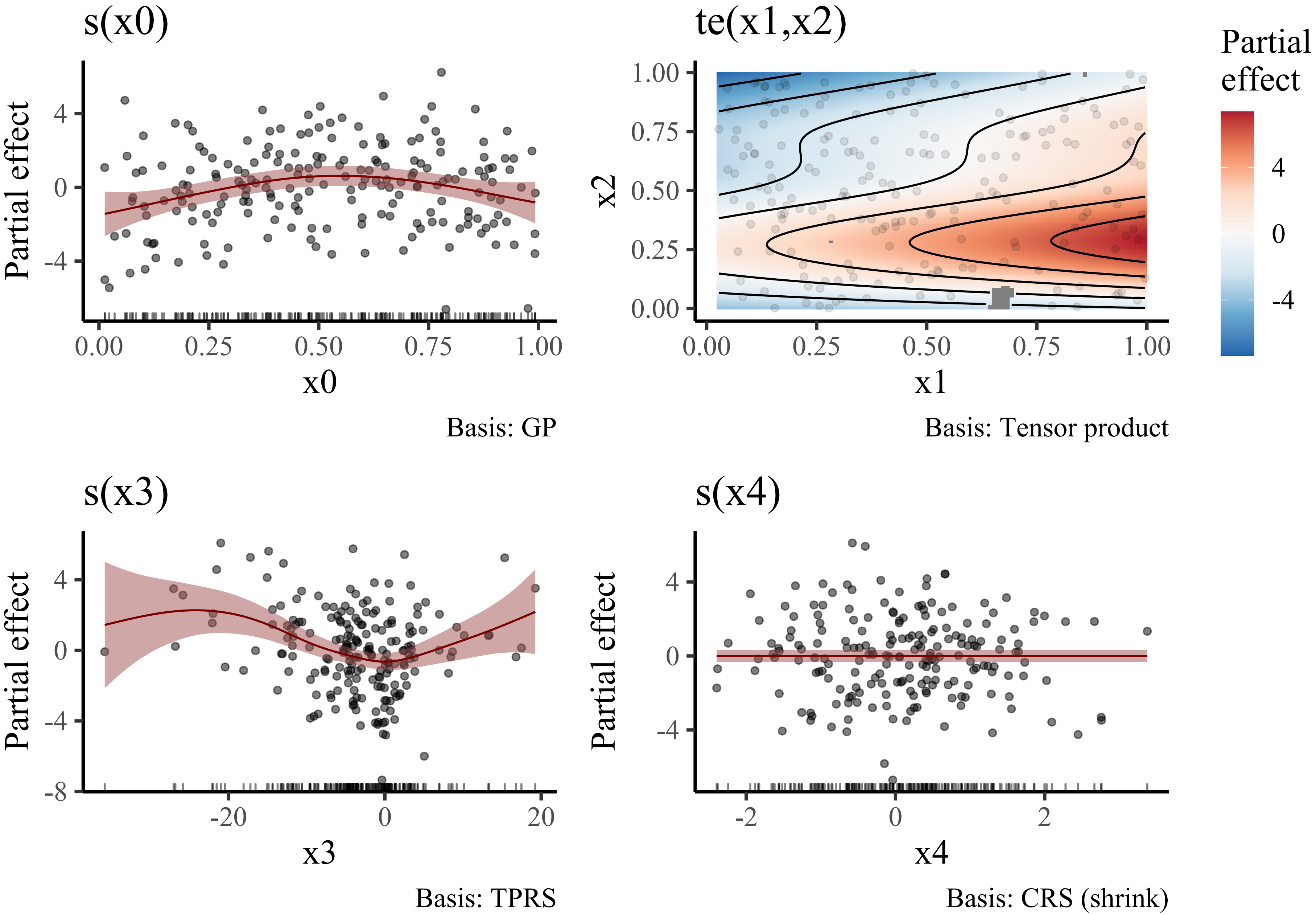


Penalize $f''(x)$ to learn weights



GAMs are just fancy GLMs, where some (or all) of the predictor effects are estimated as smooth functions.

But the complexity they can handle is *enormous*



s(x0)
Partial effect
4
0
-4
0.00
0.25
0.50
0.75
1.00
x0
Basis: GP
te(x1,x2)
1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
Basis: Tensor product
Partial
effect
4
0
-4
s(x3)
Partial effect
4
0
-4
-8
-20
0
20
x3
Basis: TPRS
s(x4)
Partial effect
4
0
-4
-2
0
2
x4
Basis: CRS (shrink)

GAMs easy to fit in R

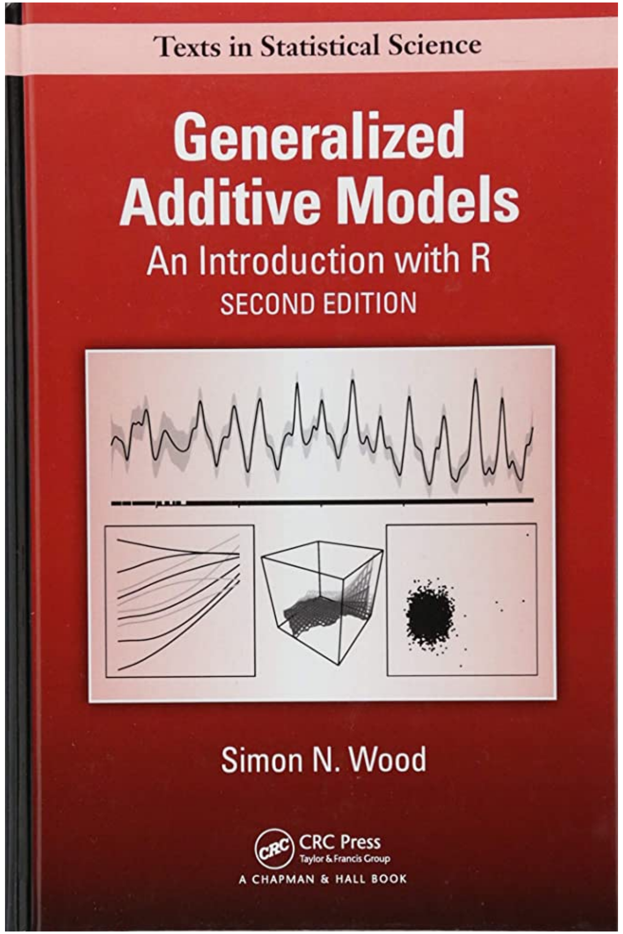
$$\mathbb{E}(\boldsymbol{Y}|\boldsymbol{X}) = g^{-1}(\alpha + \sum_{j=1}^{J} f(x_j))$$

Where:

g^{-1} is the *inverse* of the link function

α is the intercept

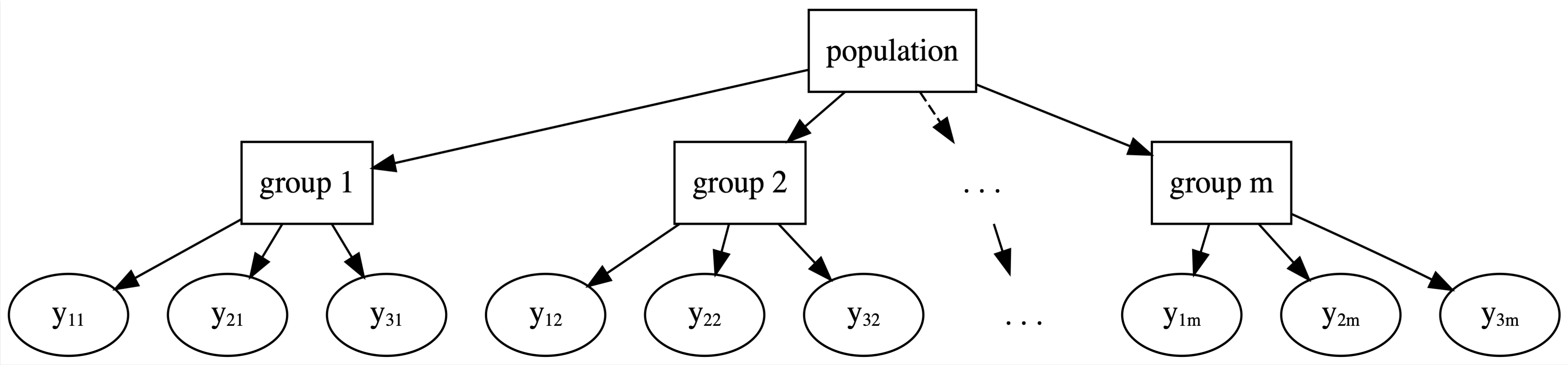
$f(x)$ are potentially nonlinear functions of the J predictors



But how can GAMs and GLMs be useful for modelling ecological time series?

Temporal random effects

Random effects are *hierarchical*



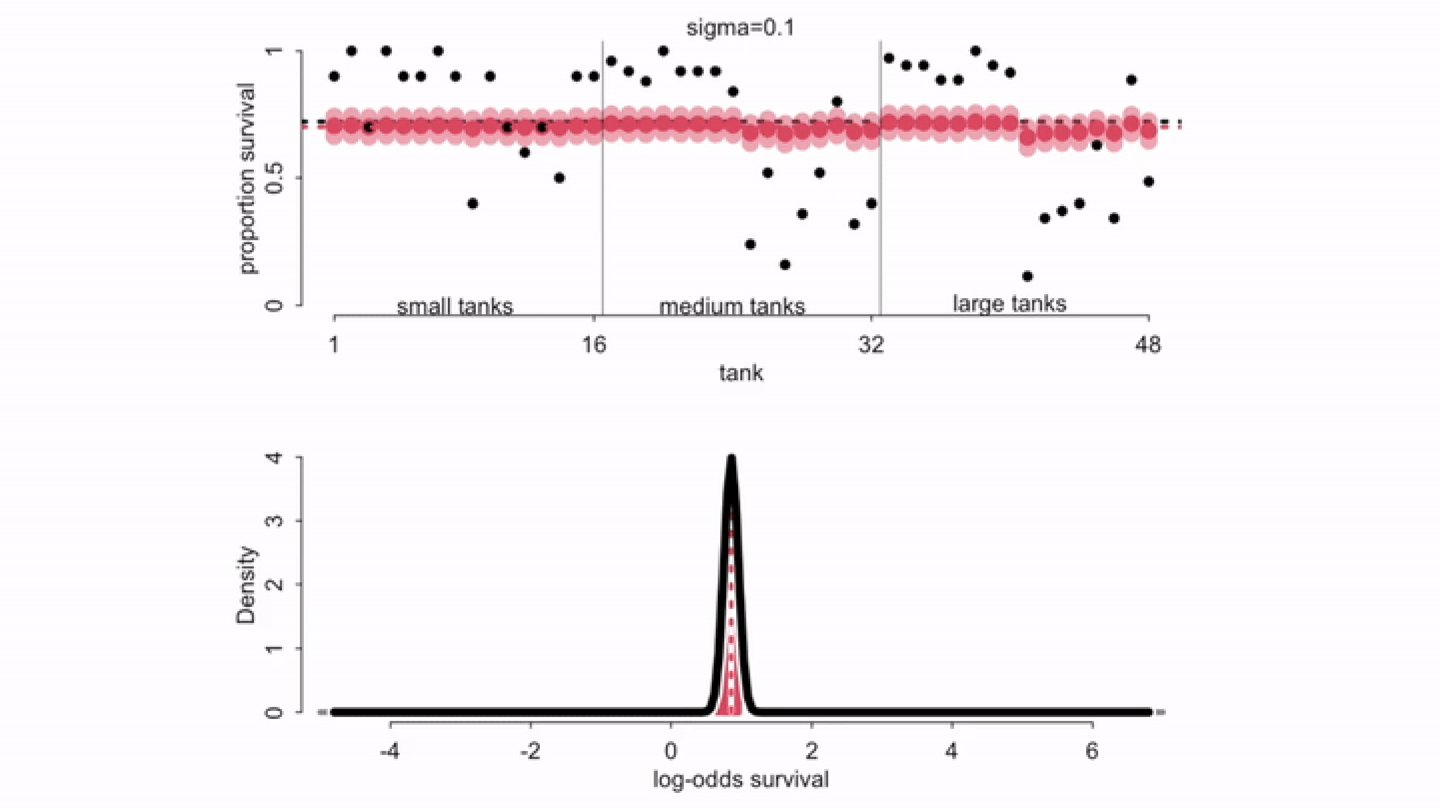
[Johnson *et al* 2021]

Hierarchical models *learn from all groups at once* to inform group-level estimates

This induces *regularization*, where noisy estimates are pulled towards the overall mean

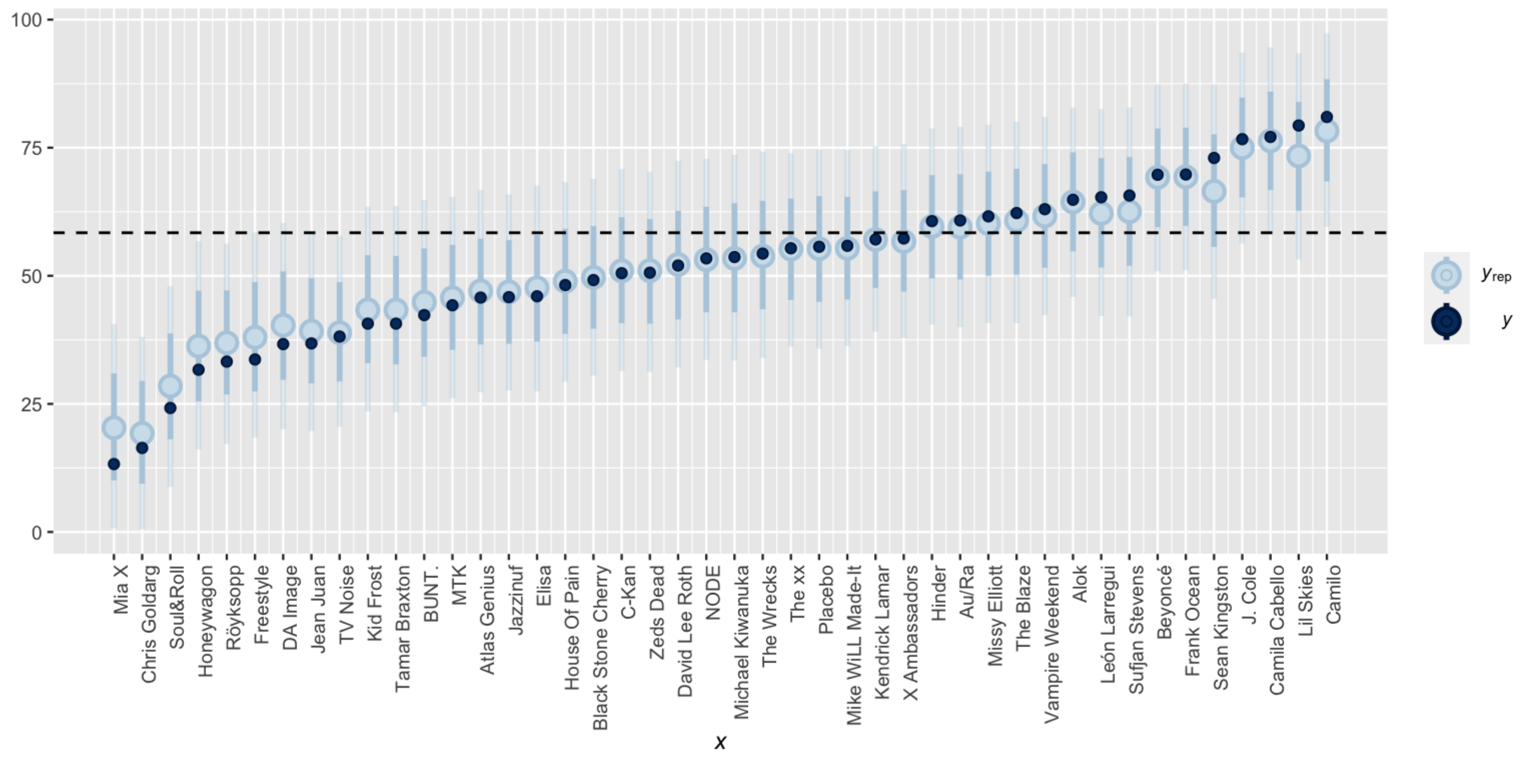
The regularization is known as partial pooling

Partial pooling in action



McElreath 2023

Noisy estimates *pulled* to the mean



Johnson *et al* 2021

How are they modelled?

$$
\begin{aligned}
\boldsymbol{Y_i} &\sim \text{Poisson}(\lambda_i) \\
log(\lambda_i) &= \beta_{year[year_t]} * \boldsymbol{year}_t \\
\beta_{year} &\sim \text{Normal}(\mu_{year}, \sigma_{year}) \\
\mu_{year} &\sim \text{Normal}(0, 1) \\
\sigma_{year} &\sim \text{Exponential}(2)
\end{aligned}
$$

Where

β_{year} are yearly intercepts (*one effect per year*)

μ_{year} estimates *mean effect among all years*

σ_{year} estimates *how much effects vary across years*

Modelling with the mvgam

Bayesian framework to fit Dynamic GLMs and Dynamic GAMs

- Hierarchical intercepts, slopes *and smooths*
- Latent dynamic processes
- State Space models with measurement error

Built off the mgcv to construct penalized smoothing splines

Convenient and familiar R formula interface

Uni- or multivariate series from a range of response distributions

Uses Stan for efficient Hamiltonian Monte Carlo sampling

Extended `mgcv` interface

```r
model <- mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

Formula for the response

```r
model <- mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

A random intercept effect

```r
model <- mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

A random slope effect

```
model <- mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

A linear parametric effect

```
model <- mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

A one-dimensional smooth

```r
model ← mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

A two-dimensional smooth

```
model ← mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

Data and response distribution

```
model ← mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

✷ latent dynamics

```
model <- mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

Sampler parameters

```r
model <- mvgam(
  formula = y ~
    s(series, bs = 're') +
    s(x0, series, bs = 're') +
    x1 +
    s(x2, bs = 'tp', k = 5) +
    te(x3, x4, bs = c('cr', 'tp')),
  data = data,
  family = poisson(),
  trend_model = 'AR1',
  burnin = 500,
  samples = 500,
  chains = 4,
  parallel = TRUE
  )
```

Data structure (long format)

y	series	time
1	series_1	1
1	series_2	1
NA	series_3	1
NA	series_4	1
1	series_1	2
0	series_2	2
3	series_3	2
4	series_4	2

Response (NAs allowed)

y	series	time
1	series_1	1
1	series_2	1
NA	series_3	1
NA	series_4	1
1	series_1	2
0	series_2	2
3	series_3	2
4	series_4	2

Series indicator (as `factor`)

y	series	time
1	series_1	1
1	series_2	1
NA	series_3	1
NA	series_4	1
1	series_1	2
0	series_2	2
3	series_3	2
4	series_4	2

Time indicator

y	series	time
1	series_1	1
1	series_2	1
NA	series_3	1
NA	series_4	1
1	series_1	2
0	series_2	2
3	series_3	2
4	series_4	2

Any other predictors

y	series	time	x0	site
1	series_1	1	-0.3758237	A
1	series_2	1	-0.7086380	A
NA	series_3	1	0.0458092	B
NA	series_4	1	0.7708798	B
1	series_1	2	0.2855102	A
0	series_2	2	0.3425233	A
3	series_3	2	-0.3776768	B
4	series_4	2	1.3207727	B

Examples

The data structure

```
dplyr::glimpse(model_data)
```

```
## Rows: 199
## Columns: 6
## $ series  <fct> PP, PP, PP, PP, PP, PP, PP, PP, PP, PP, PP, PP, PP, PP, PP,
PP…
## $ year    <fct> 2004, 2004, 2004, 2004, 2004, 2004, 2004, 2004, 2004, 2004,
20…
## $ time    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18,…
## $ count   <int> 0, 1, 2, NA, 10, NA, NA, 16, 18, 12, NA, 3, 2, NA, NA, 13,
NA,…
## $ mintemp <dbl> -9.710, -5.924, -0.220, 1.931, 6.568, 11.590, 14.370,
16.520, …
## $ ndvi    <dbl> 1.4658889, 1.5585069, 1.3378172, 1.6589129, 1.8536561,
```

The observations

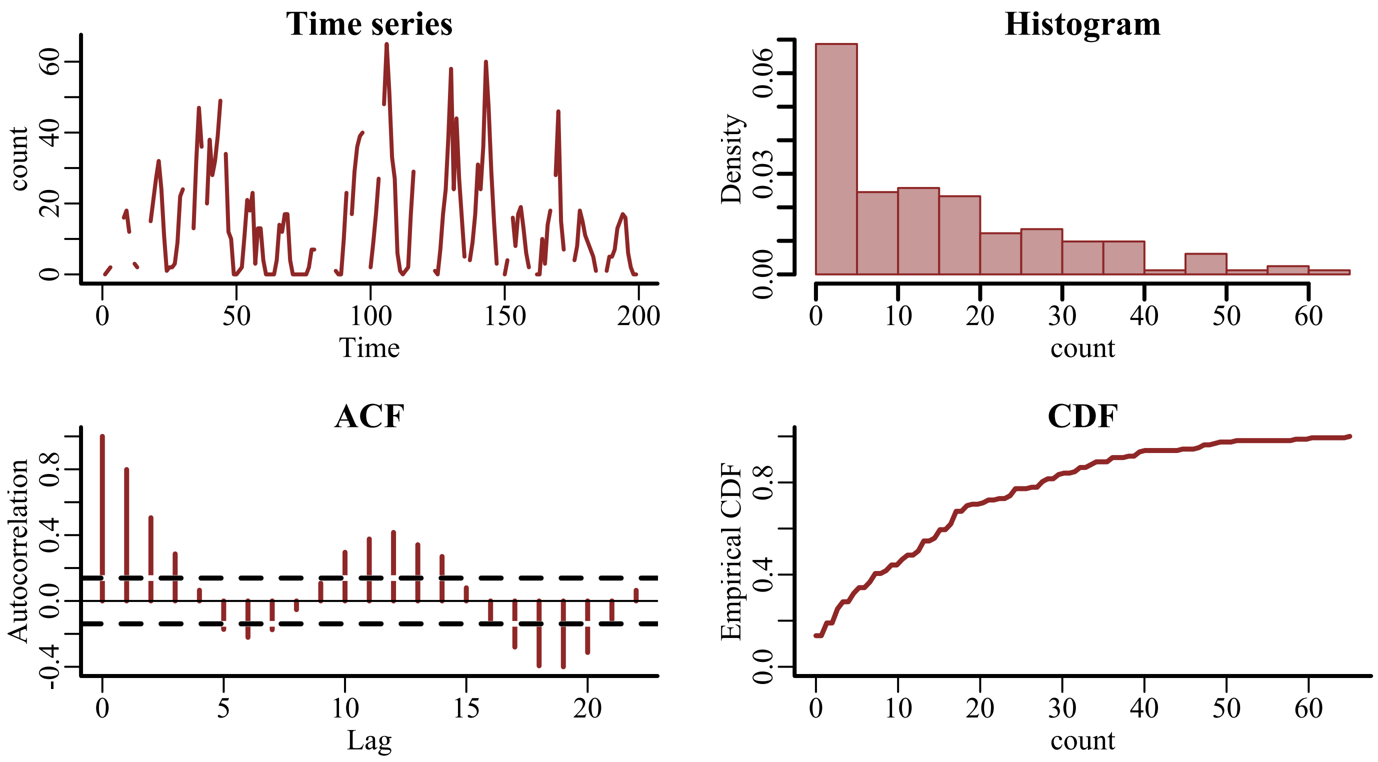
Code Plot

```
# use mvgam's plot utility to view properties of the observations
plot_mvgam_series(data = model_data, y = 'count')
```

The observations

Code

Plot



Yearly random intercepts

```
year_random <- mvgam(count ~
                       s(year, bs = 're') - 1,
                     family = poisson(),
                     data = model_data,
                     burnin = 500,
                     samples = 500,
                     chains = 4)
```

Random effect basis in `mgcv` language

Global intercept suppressed

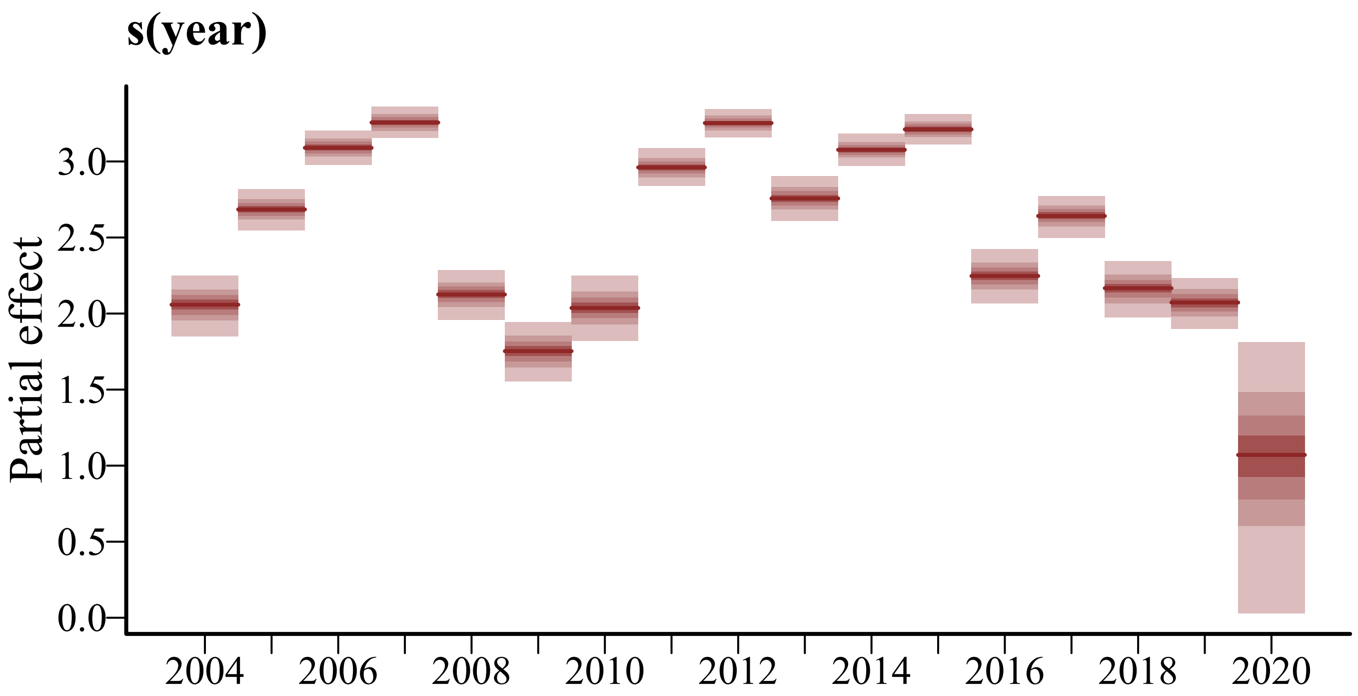
Estimated yearly intercepts

Code Plot

```
# plot the random effect posterior estimates
plot(year_random, type = 're')
```

Estimated yearly intercepts

Code Plot



Population parameters

Code Means SDs

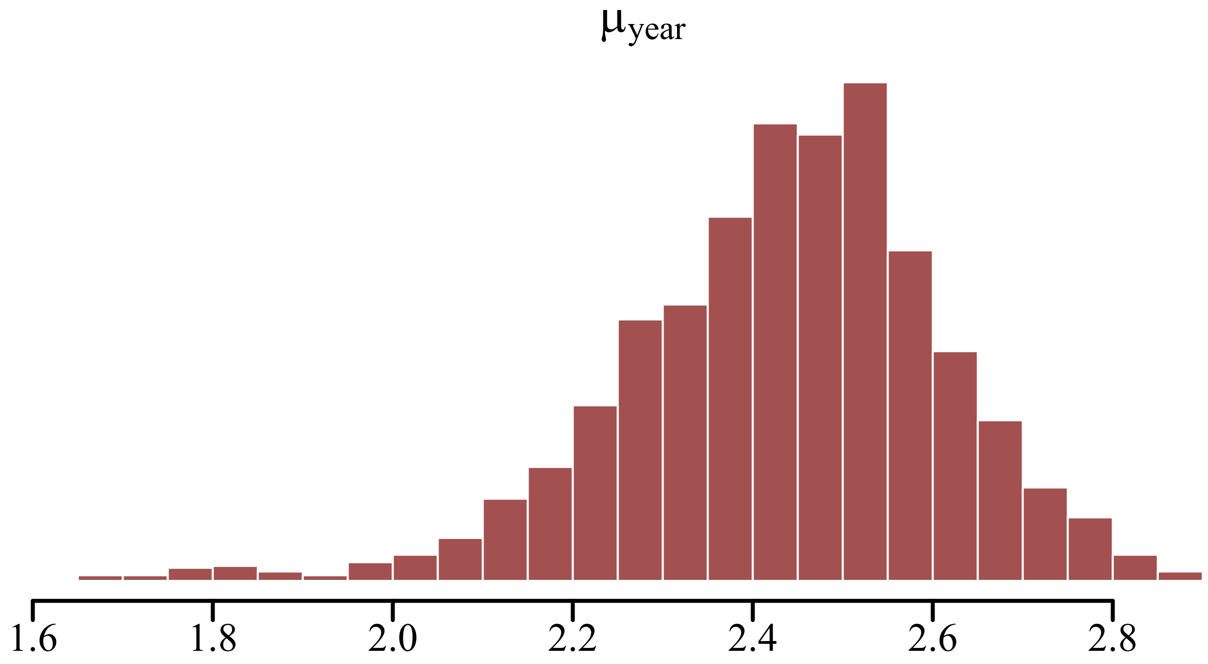
```
# extract population estimates
pop_params ← as.data.frame(year_random,
                           variable = c('mean(year)',
                                        'sd(year)'))

# plot as histograms
hist(pop_params$`mean(year)`, main = expression(mu[year]))

hist(pop_params$`sd(year)`, main = expression(sigma[year]))
```

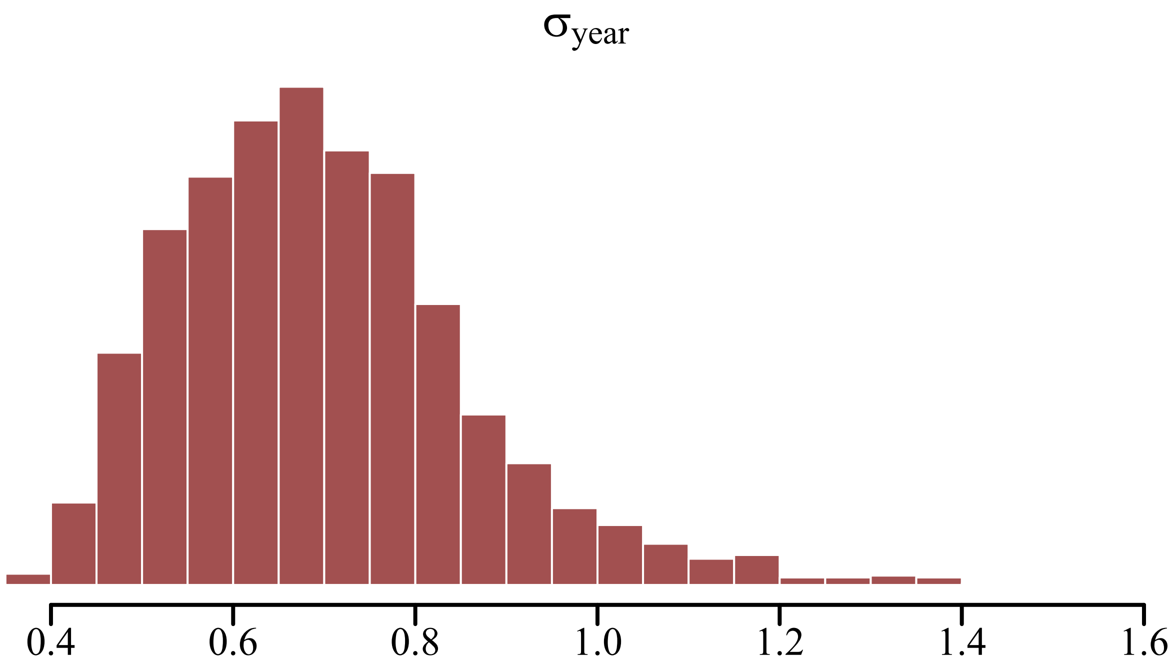
Population parameters

Code | **Means** | SDs



Population parameters

Code Means **SDs**



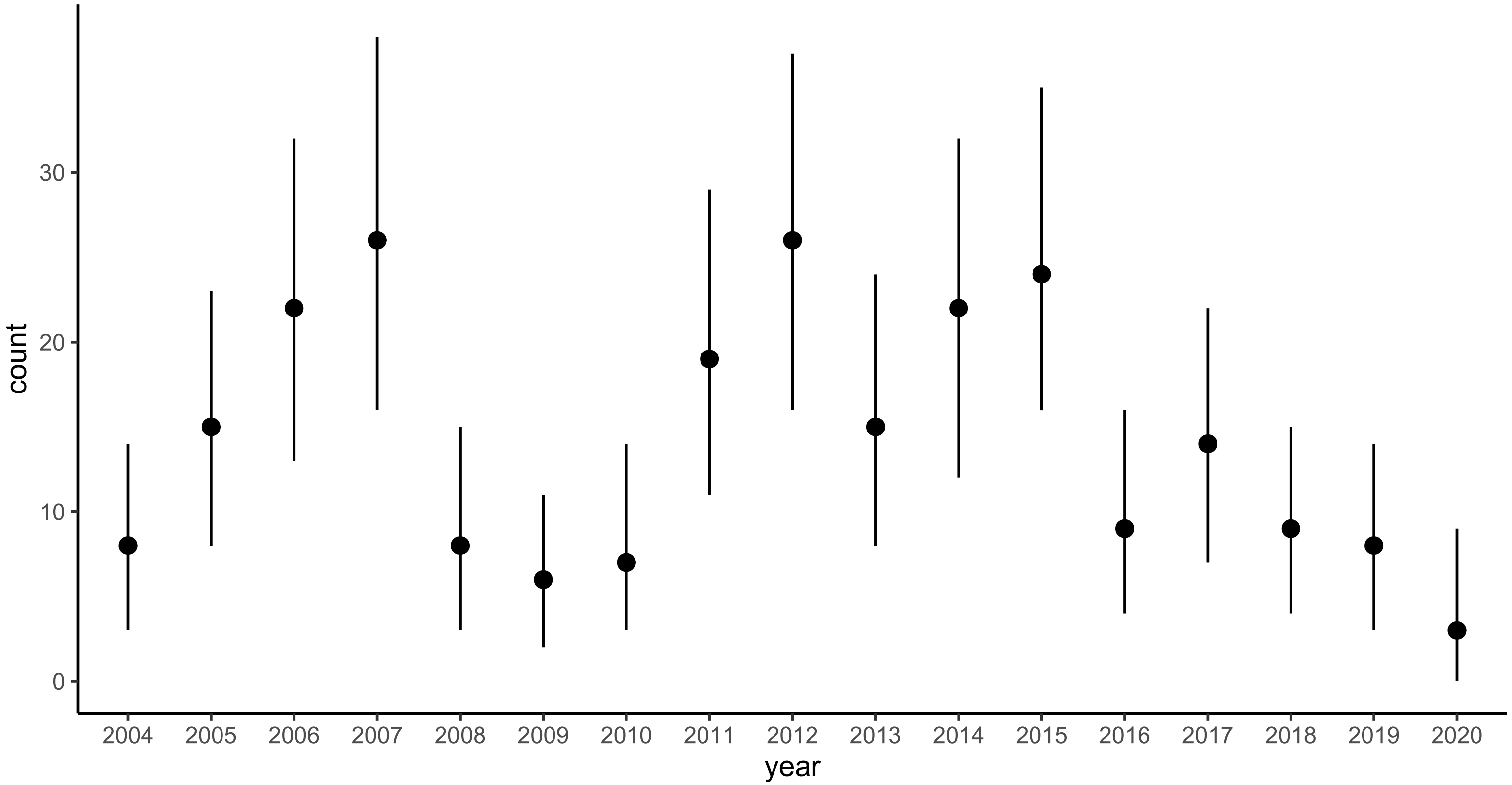
Conditional predictions

Code Plot

```r
# use marginaleffects utilities to plot conditional predictions
library(ggplot2)
plot_predictions(year_random,
                 condition = 'year',
                 n_cores = 2) +
  theme_classic()
```

Conditional predictions

Code Plot



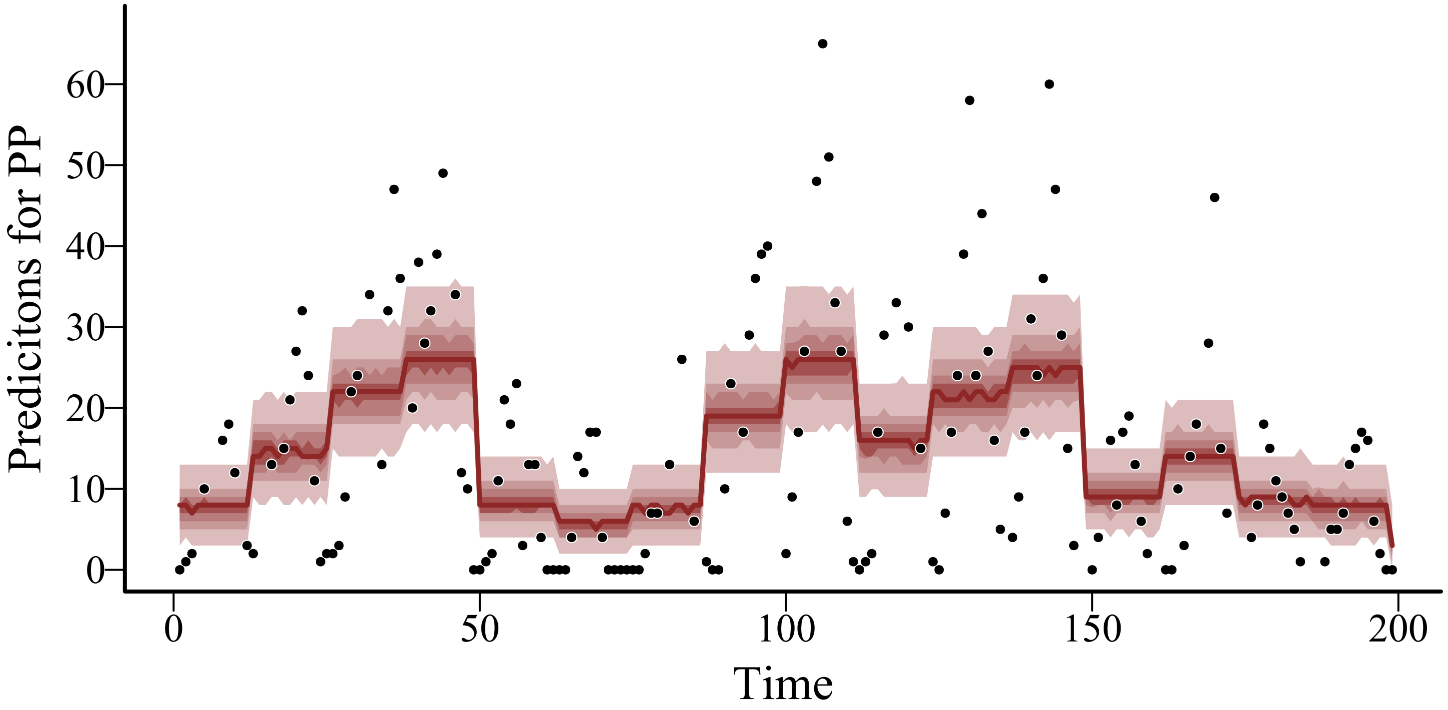
Hindcast predictions

Code Plot

```
# use mvgam's plot to view hindcast predictions
plot(year_random, type = 'forecast')
```

Hindcast predictions

Code Plot



`mvgam` with yearly smooth

```
model_data %>%
  dplyr::mutate(year = as.numeric(as.character(year))) -> model_data

year_smooth <- mvgam(count ~
                       s(year, bs = 'tp', k = 15),
                    family = poisson(),
                    data = model_data,
                    burnin = 500,
                    samples = 500,
                    chains = 4)
```

A thin plate regression spline of the numeric `year` variable

Retain intercept because smooths are zero-centered

Coefficients uninterpretable

```
rownames(coef(year_smooth))
```

```
##  [1] "(Intercept)" "s(year).1"   "s(year).2"   "s(year).3"   "s(year).4"
##  [6] "s(year).5"   "s(year).6"   "s(year).7"   "s(year).8"   "s(year).9"
## [11] "s(year).10"  "s(year).11"  "s(year).12"  "s(year).13"  "s(year).14"
```

We ***must*** use predictions and plots to understand the model

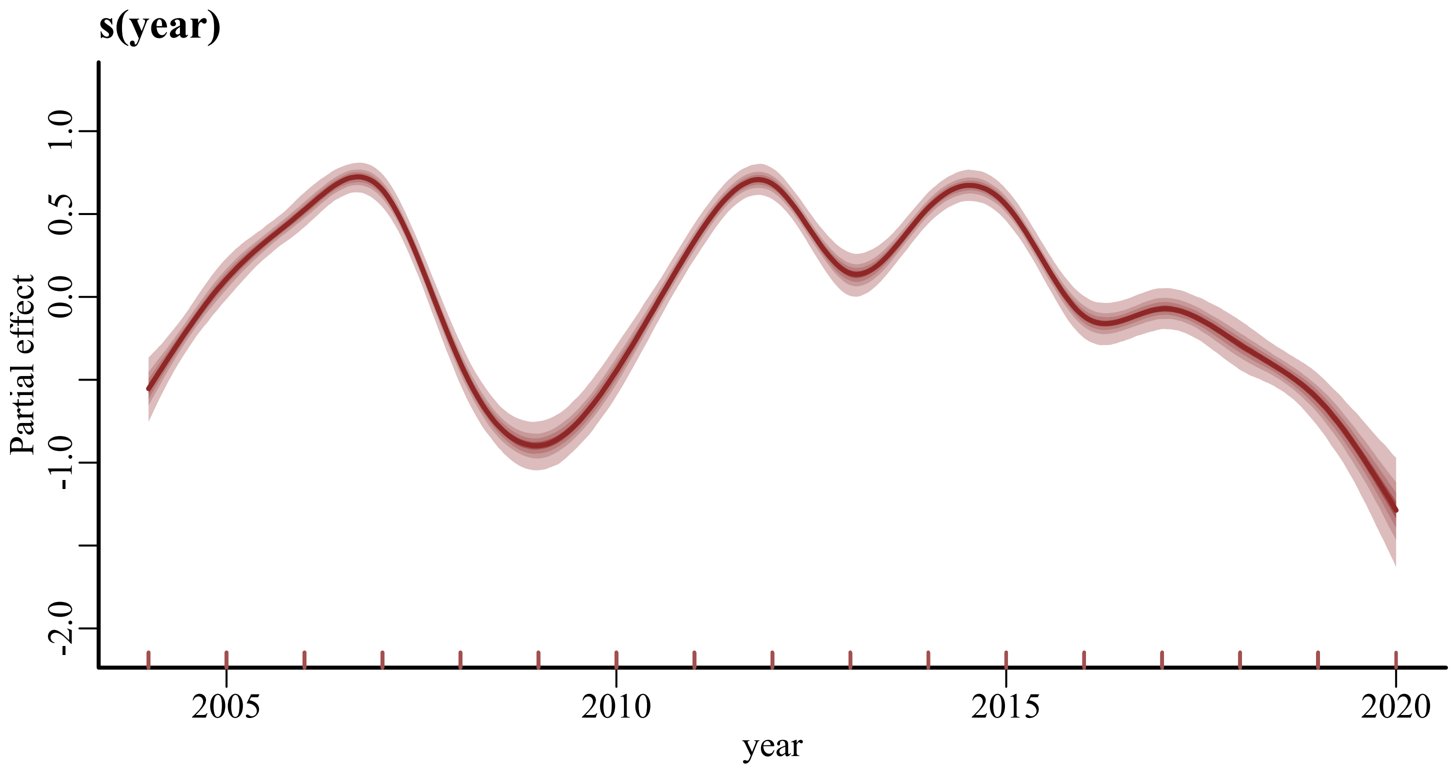
Estimated yearly smooth

Code Plot

```
# plot the smooth effect posterior estimates
plot(year_smooth, type = 'smooth')
```

Estimated yearly smooth

Code Plot



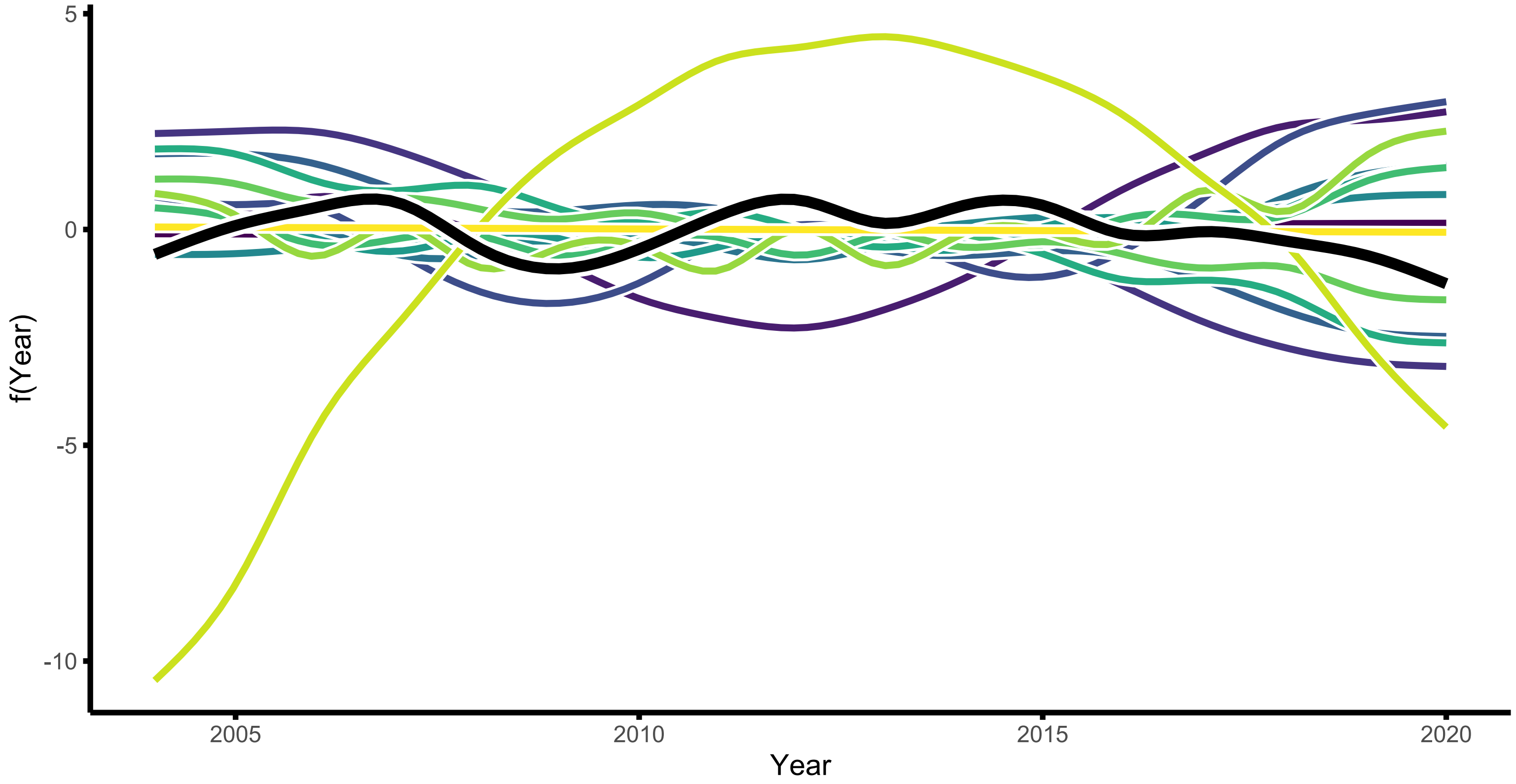
Plotting basis functions

Code Plot

```
# plot the basis functions with gratia
library(ggplot2); library(viridis); library(gratia)
theme_set(theme_classic())
ggplot(basis(year_smooth$mgcv_model),
       aes(x = year, y = value, color = bf)) +
  geom_borderline(linewidth = 1.25, bordercolour = "white") +
  geom_borderline(data = smooth_estimates(year_smooth$mgcv_model),
                  inherit.aes = FALSE,
                  mapping = aes(x = year, y = est), linewidth = 2) +
  scale_color_viridis(discrete = TRUE) +
  theme(legend.position = 'none', axis.line = element_line(size = 1),
        axis.ticks = element_line(colour = "black", size = 1)) +
  ylab('f(Year)') + xlab('Year')
```

Plotting basis functions

Code Plot



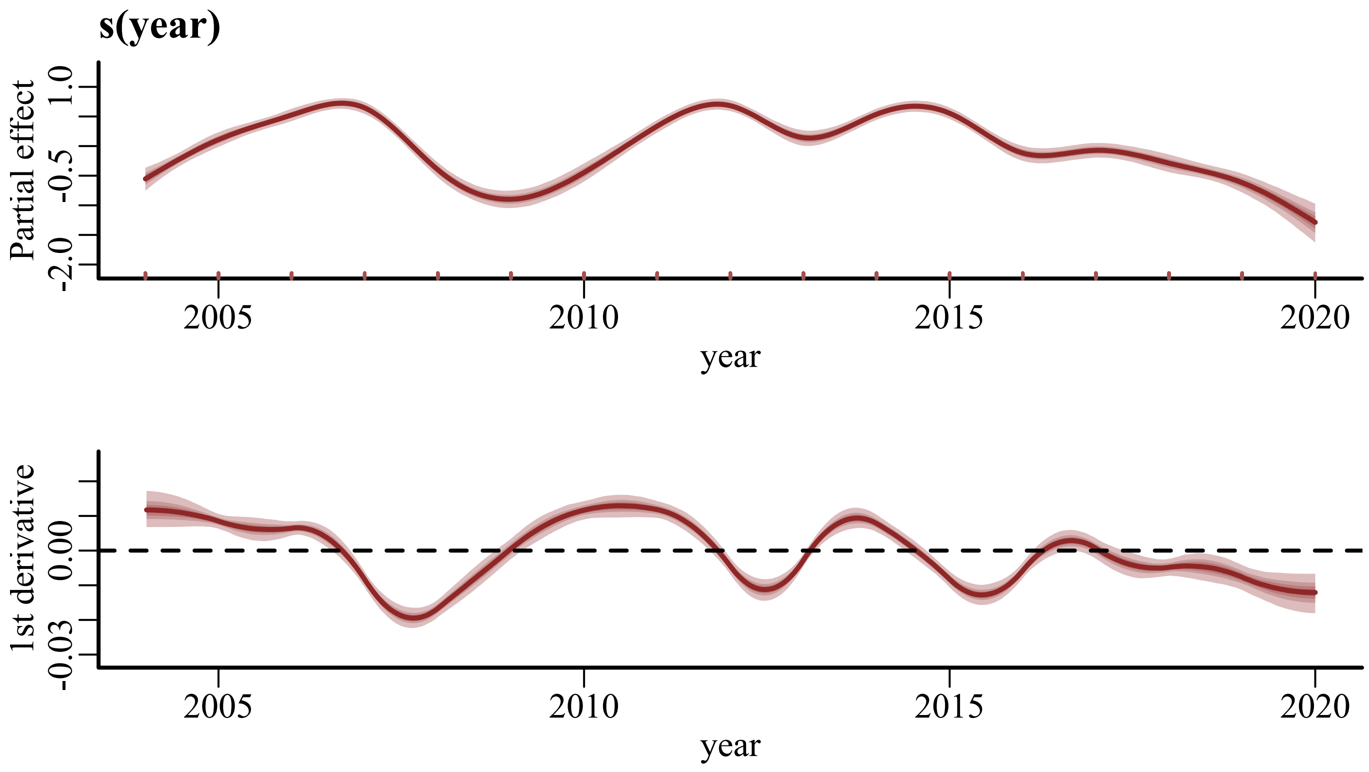
Rates of change

Code Plot

```r
# plot the smooth effect posterior estimates
plot(year_smooth, type = 'smooth', derivatives = TRUE)
```

Rates of change

Code Plot



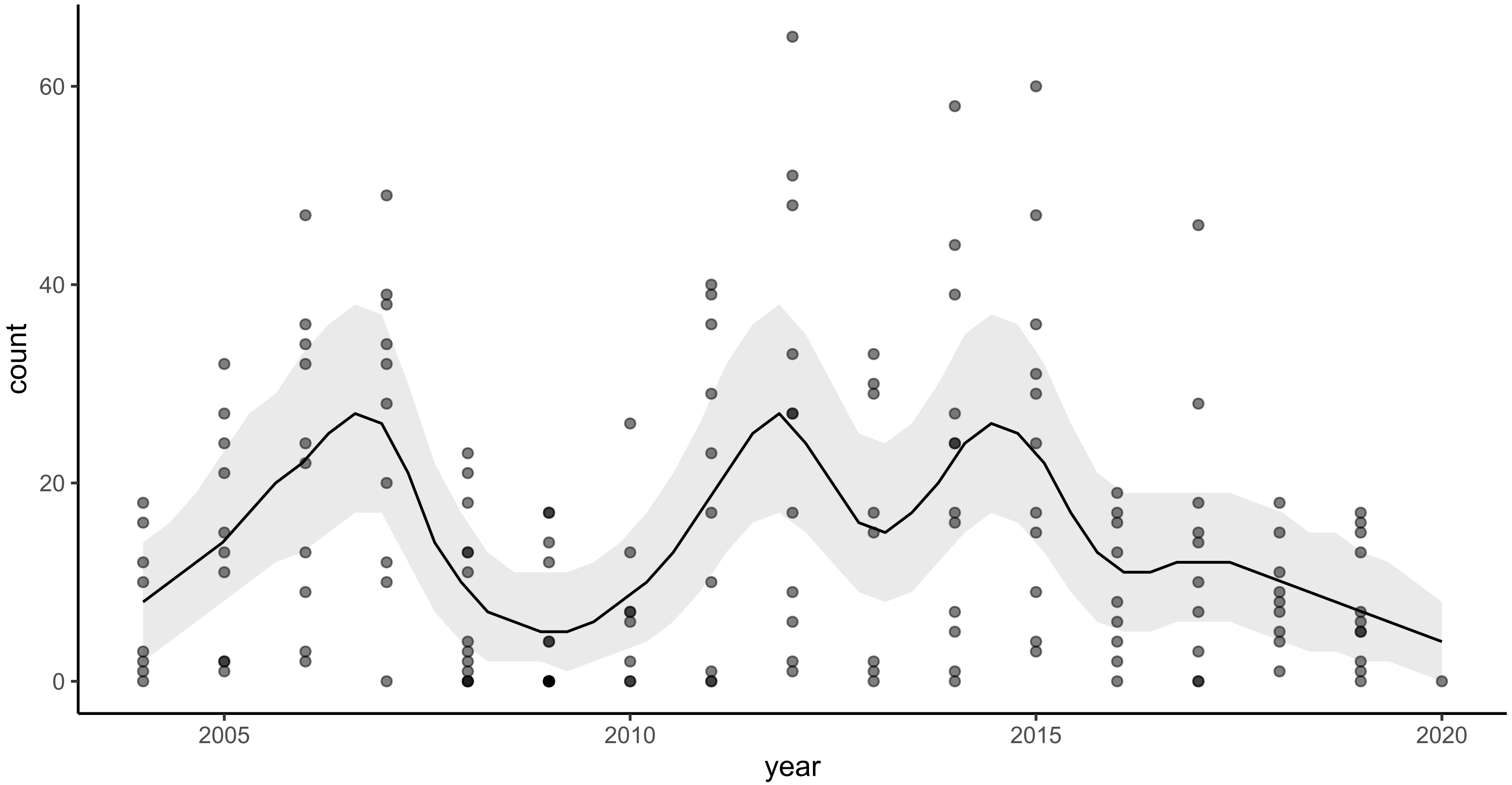
Conditional predictions

Code Plot

```r
# use marginaleffects utilities to plot conditional predictions
library(ggplot2)
plot_predictions(year_smooth,
                 condition = 'year',
                 points = 0.5,
                 n_cores = 2) +
  theme_classic()
```

Conditional predictions

Code Plot



Hindcast predictions

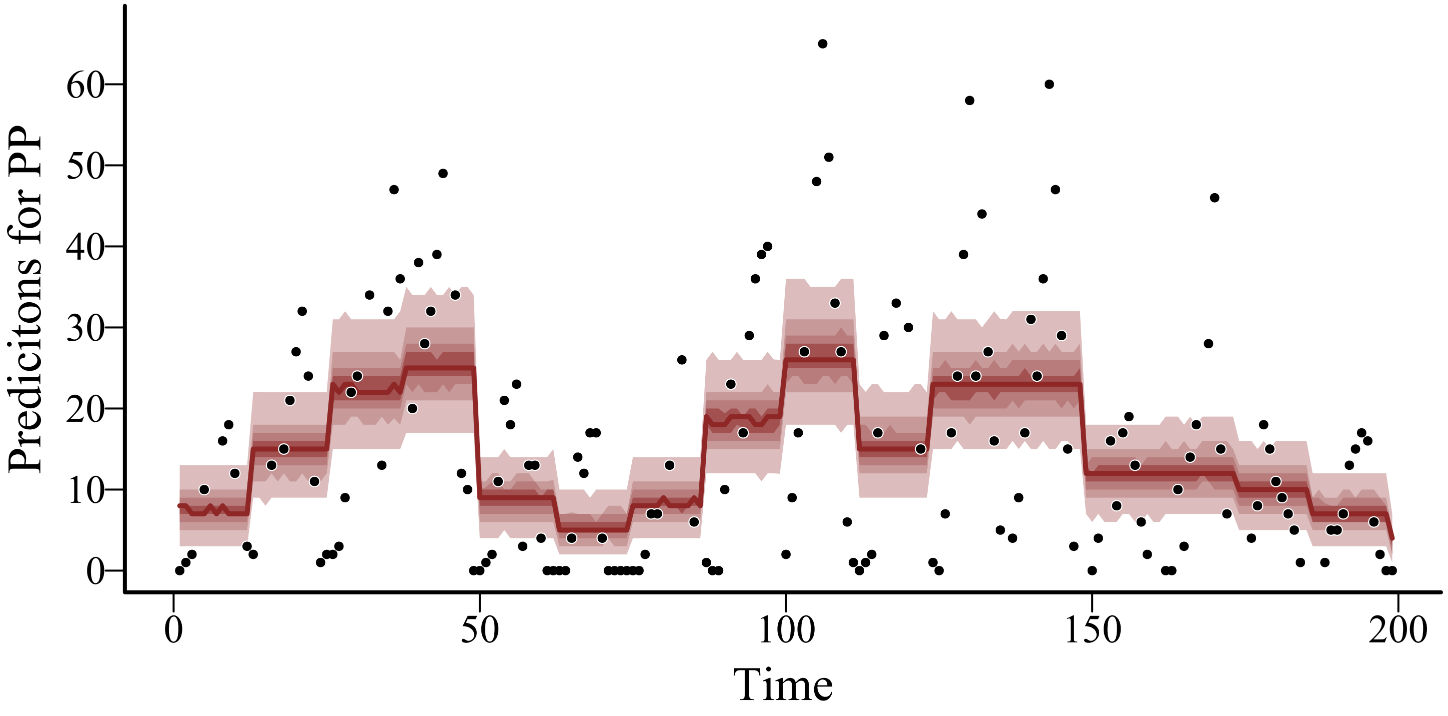
Code Plot

```
# use mvgam's plot to view hindcast predictions
plot(year_smooth, type = 'forecast')
```

Hindcast predictions

Code

Plot

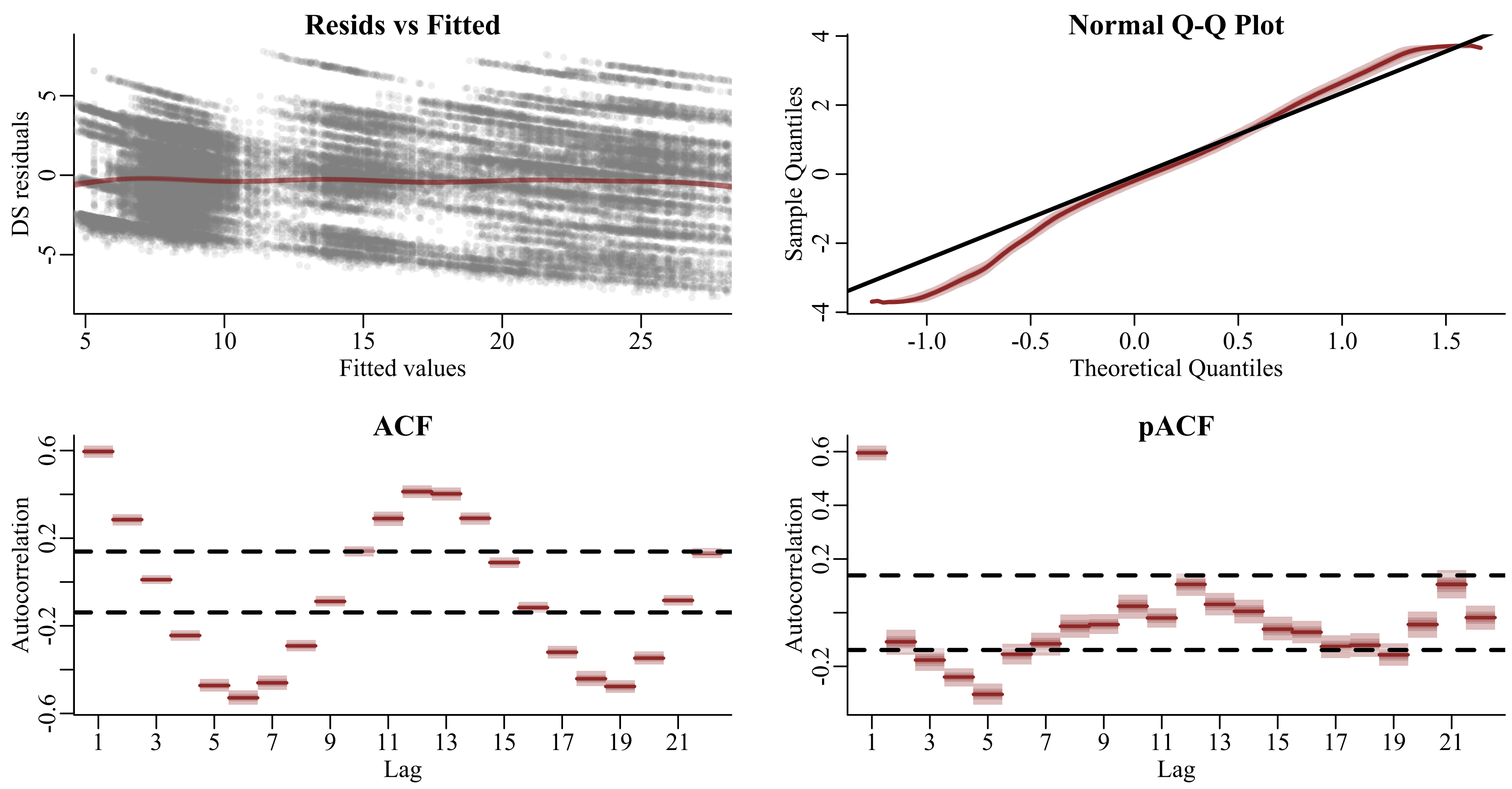


Forecasts will differ. Why?

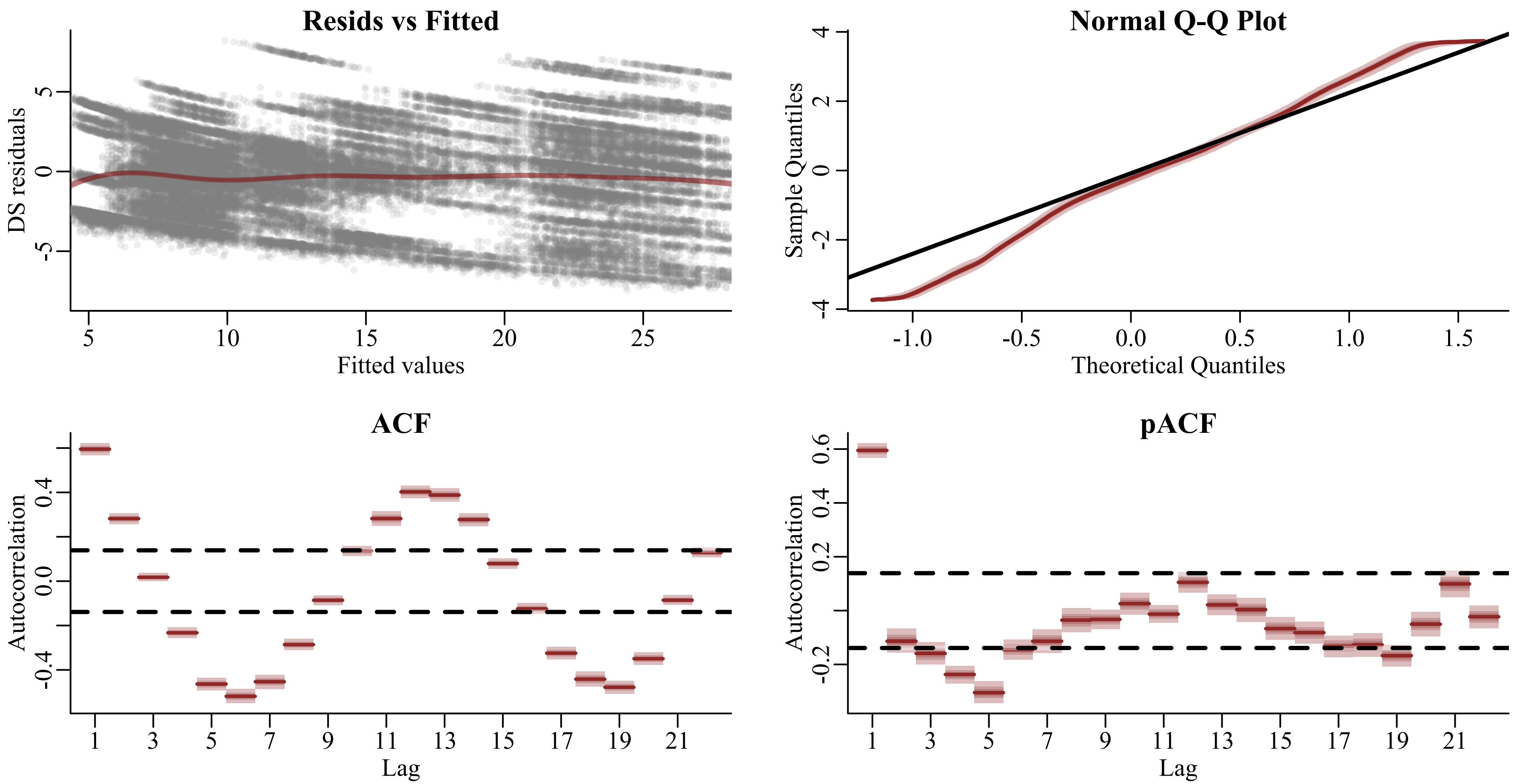
We will explore this further in the tutorial and in the next lecture

But how do model diagnostics look?

Random year diagnostics



Smooth year diagnostics



Randomized quantile residuals show evidence of unmodelled autocorrelation and seasonality

How can we deal with the seasonality?

Adding a smooth of mintemp

```r
year_temp_smooth <- mvgam(count ~
                            s(year, bs = 'tp', k = 15) +
                            s(mintemp, bs = 'tp', k = 8),
                          family = poisson(),
                          data = model_data,
                          burnin = 500,
                          samples = 500,
                          chains = 4)
```

A thin plate regression spline of `mintemp`

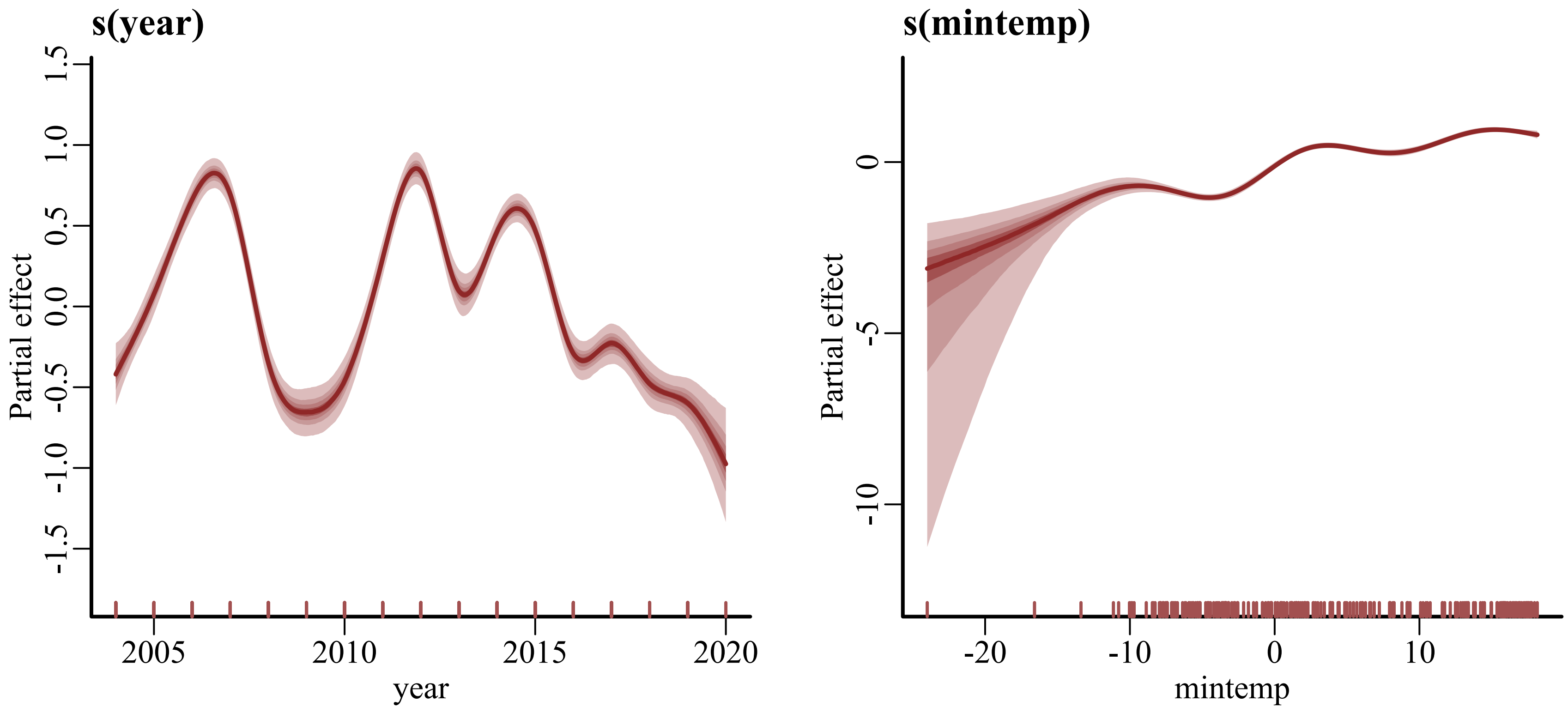
Estimated smooths

Code Plot

```r
# use mvgam's plot to view both smooth functions
plot(year_temp_smooth, type = 'smooth')
```

Estimated smooths

Code Plot



Partial residuals

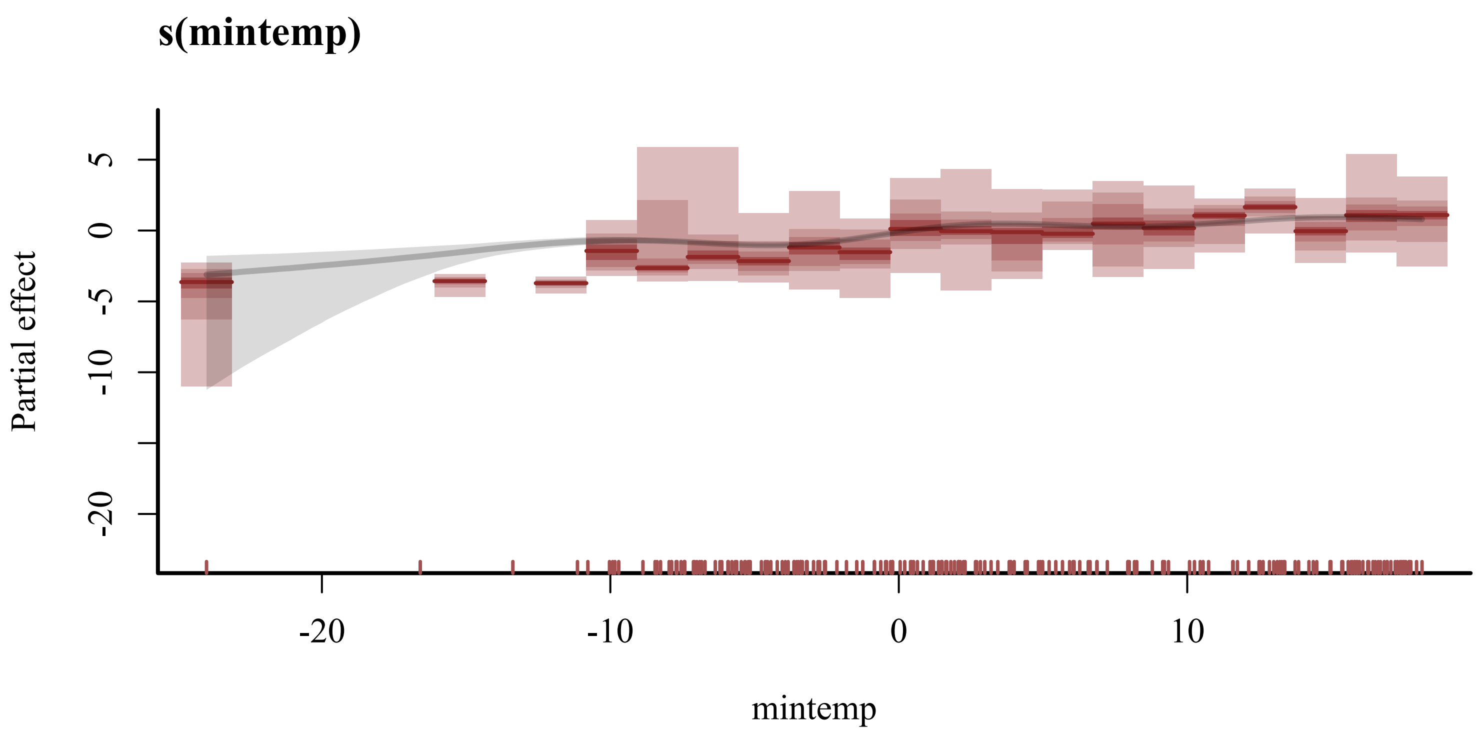
Code

Plot

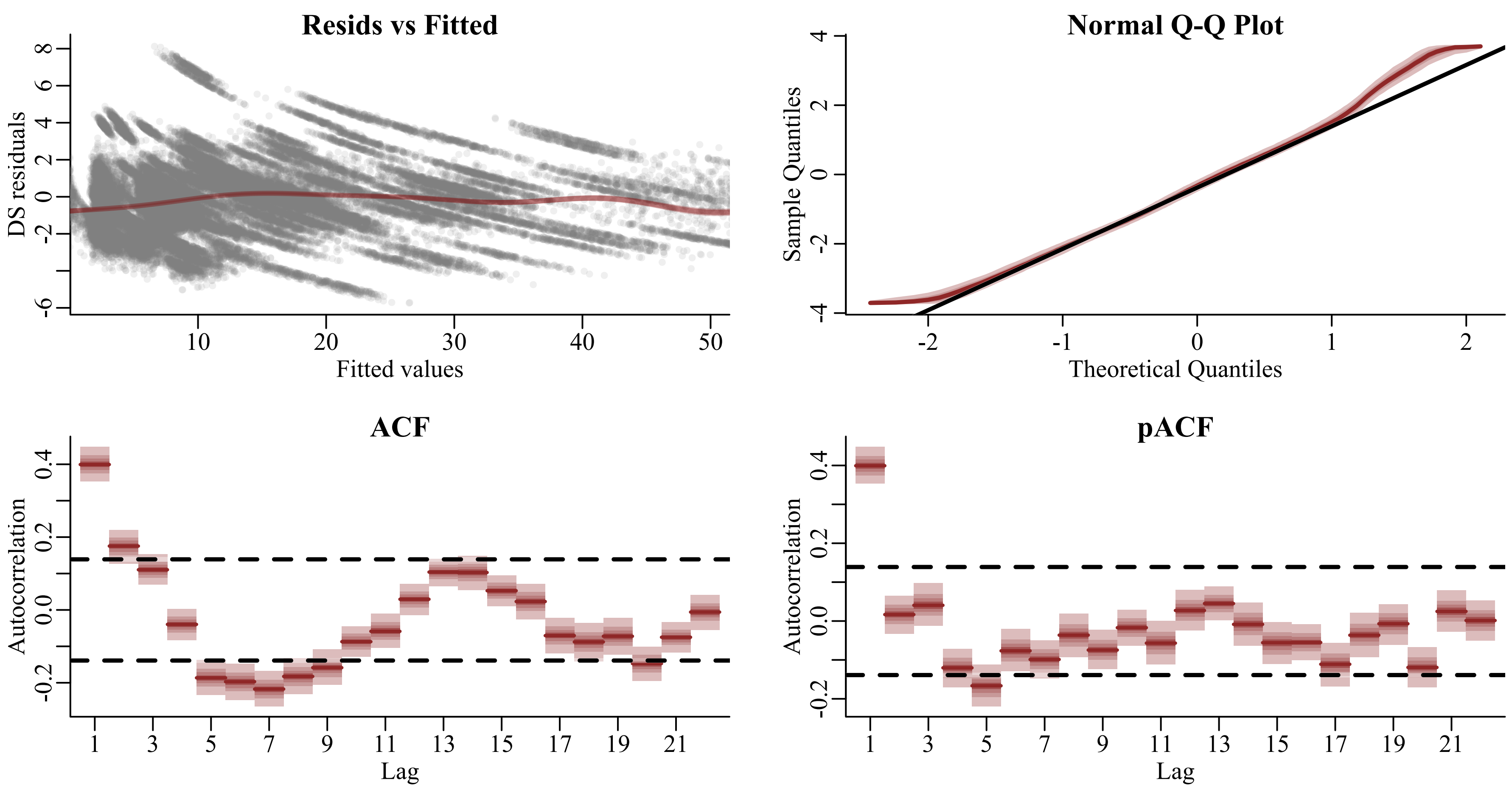
```
# use mvgam's plot_mgvam_smooth to view partial residuals
plot_mvgam_smooth(year_temp_smooth, smooth = 2, residuals = TRUE)
```

Partial residuals

Code Plot



Diagnostics



Randomized quantile residuals still show evidence of unmodelled autocorrelation

How can we deal with this?

Adding a latent AR term

```
year_temp_smooth_ar <- mvgam(count ~
                               s(year, bs = 'tp', k = 15) +
                               s(mintemp, bs = 'tp', k = 8),
                             family = poisson(),
                             trend_model = 'AR1',
                             data = model_data,
                             burnin = 500,
                             samples = 500,
                             chains = 4)
```

A latent (unobserved) dynamic model for residuals

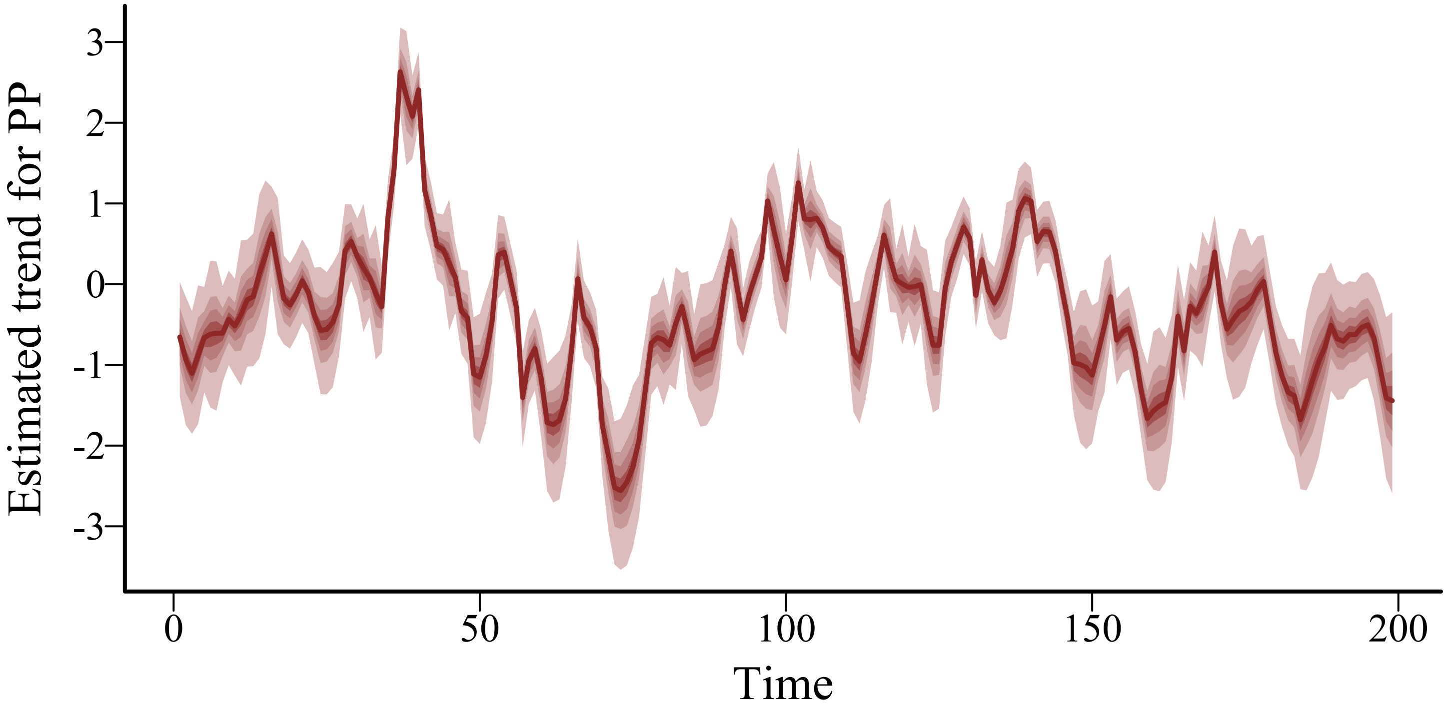
The latent trend

Code Plot

```
# use mvgam's plot to view the trend
plot(year_temp_smooth_ar, type = 'trend')
```

The latent trend

Code Plot



AR parameters

Code ARls SDs

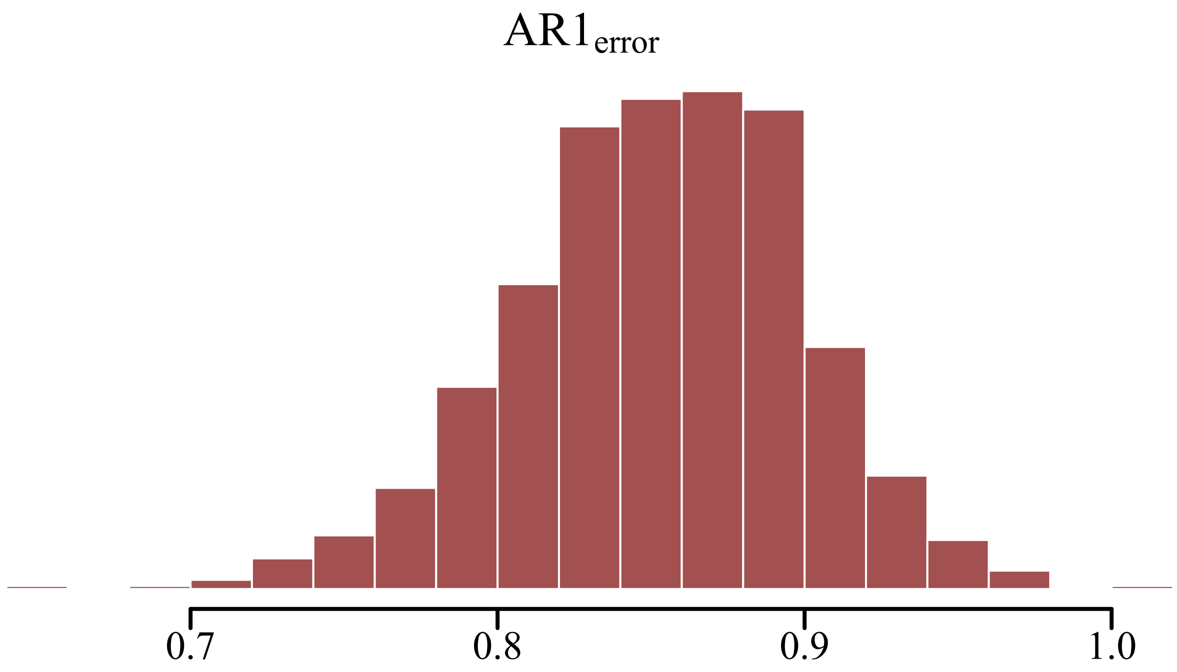
```
# extract AR estimates
ar_params ← as.data.frame(year_temp_smooth_ar,
                            variable = c('ar1[1]',
                                         'sigma[1]'))

# plot as histograms
hist(ar_params$`ar1[1]`, main = expression(AR1[error]))

hist(ar_params$`sigma[1]`, main = expression(sigma[error]))
```

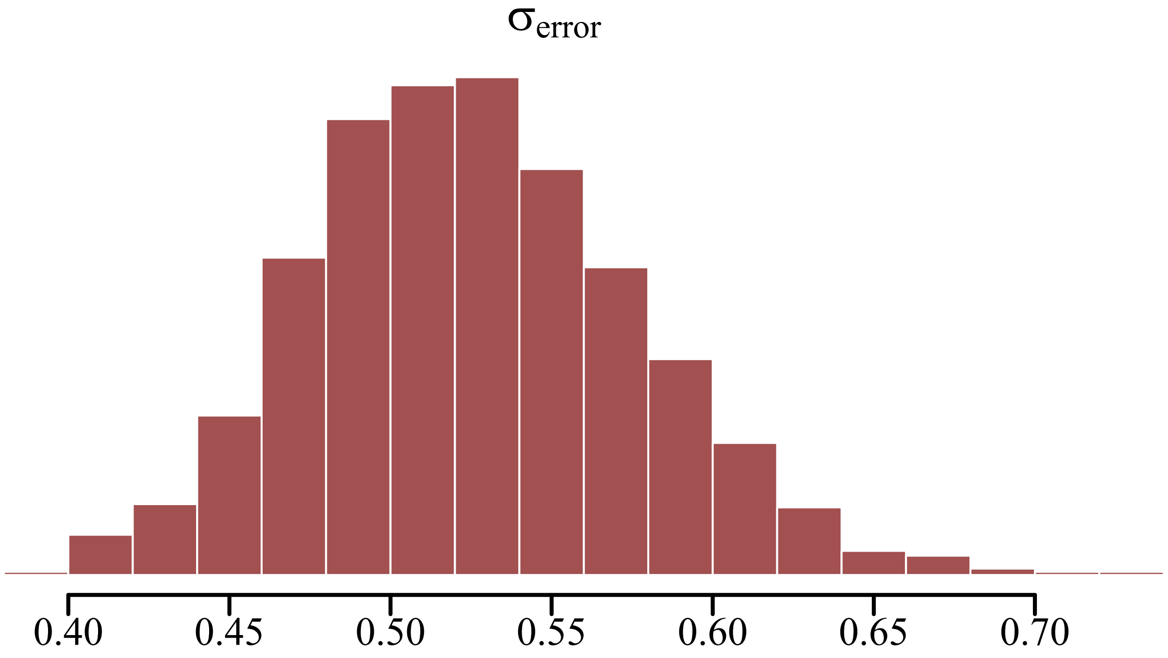
AR parameters

Code AR1s SDs

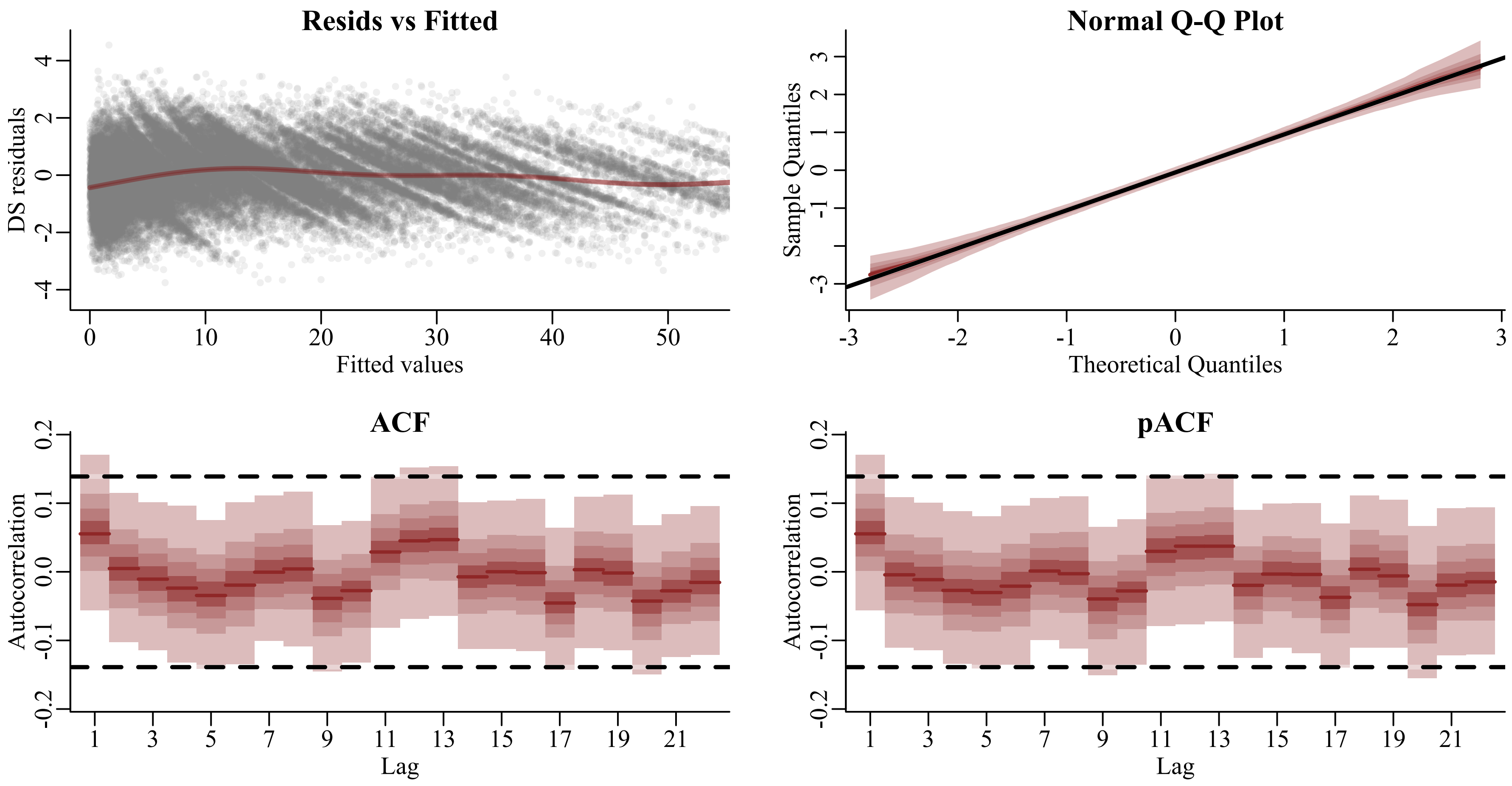


AR parameters

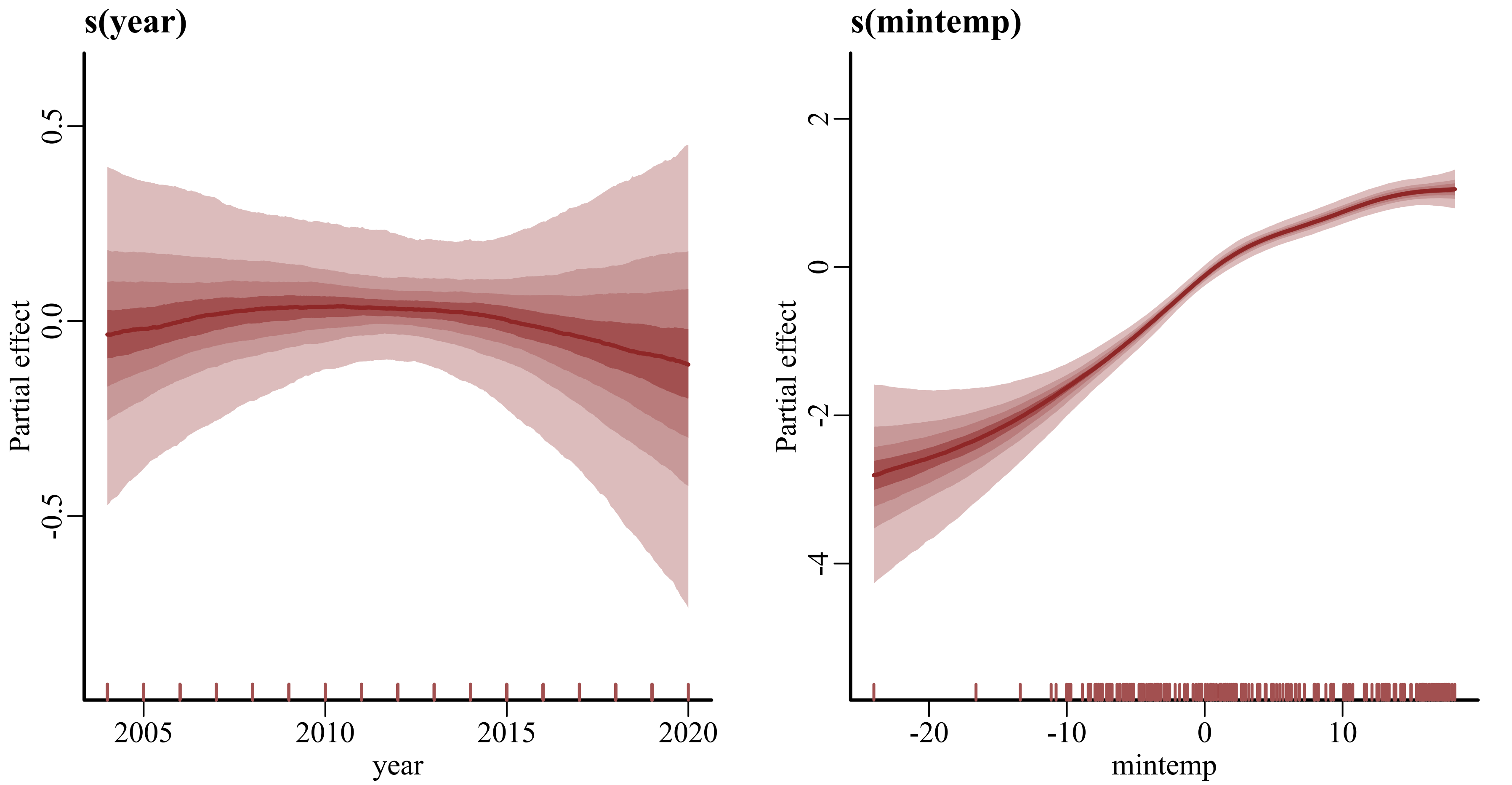
Code AR1s **SDs**



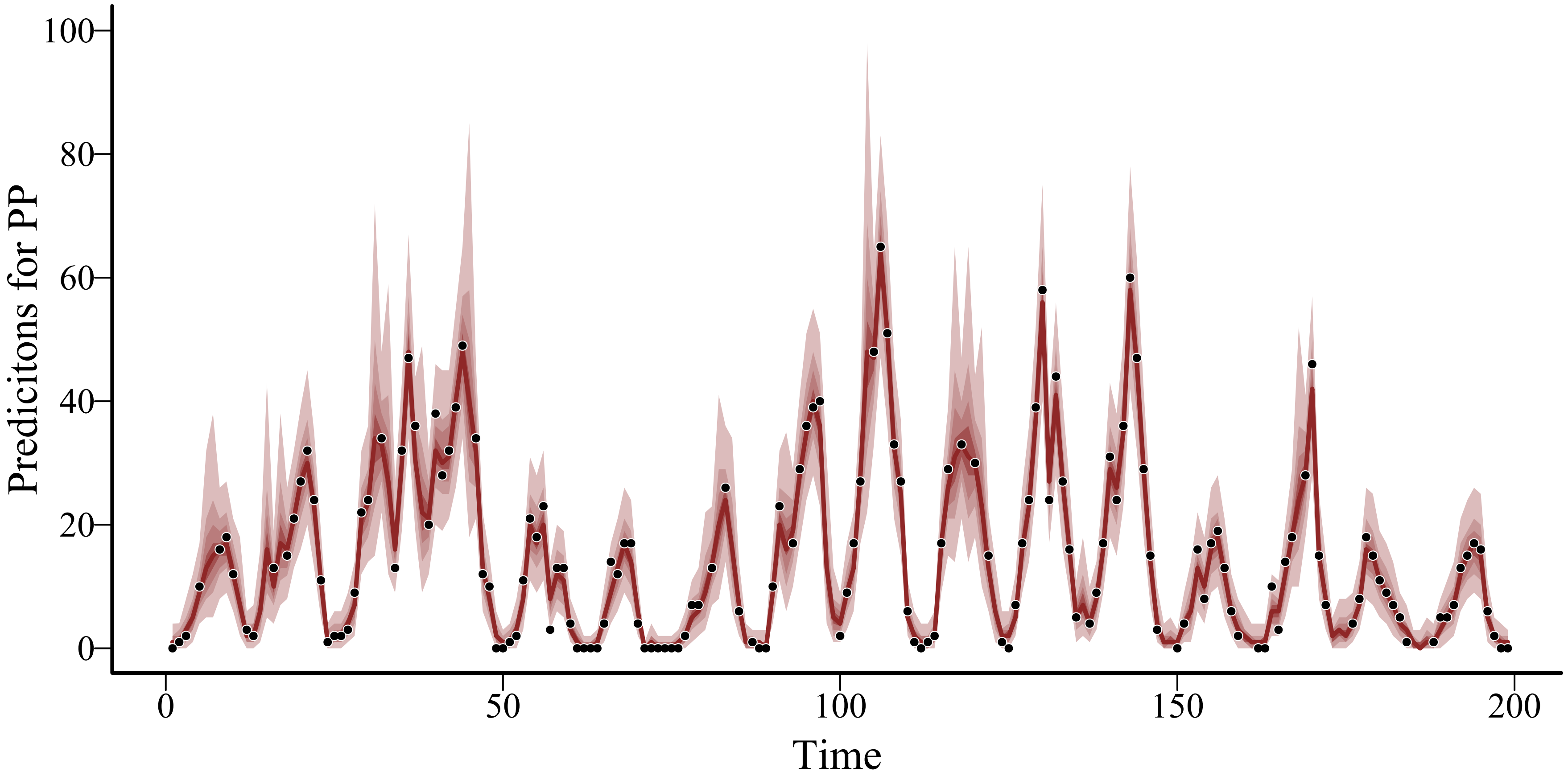
Diagnostics



Updated smooths



Hindcast predictions



In the next lecture, we will cover

Latent autoregressive processes

Latent Gaussian Processes

Dynamic coefficient models