Ecological forecasting in R

Lecture 4: multivariate ecological timeseries

Nicholas Clark

School of Veterinary Science, University of Queensland

0900–1200 CET Thursday 30th 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:

GAMs for time series

Smoothed dynamic factor analysis for identifying trends in multivariate time series

Multivariate State-Space models

This lecture's topics

Multivariate ecological time series

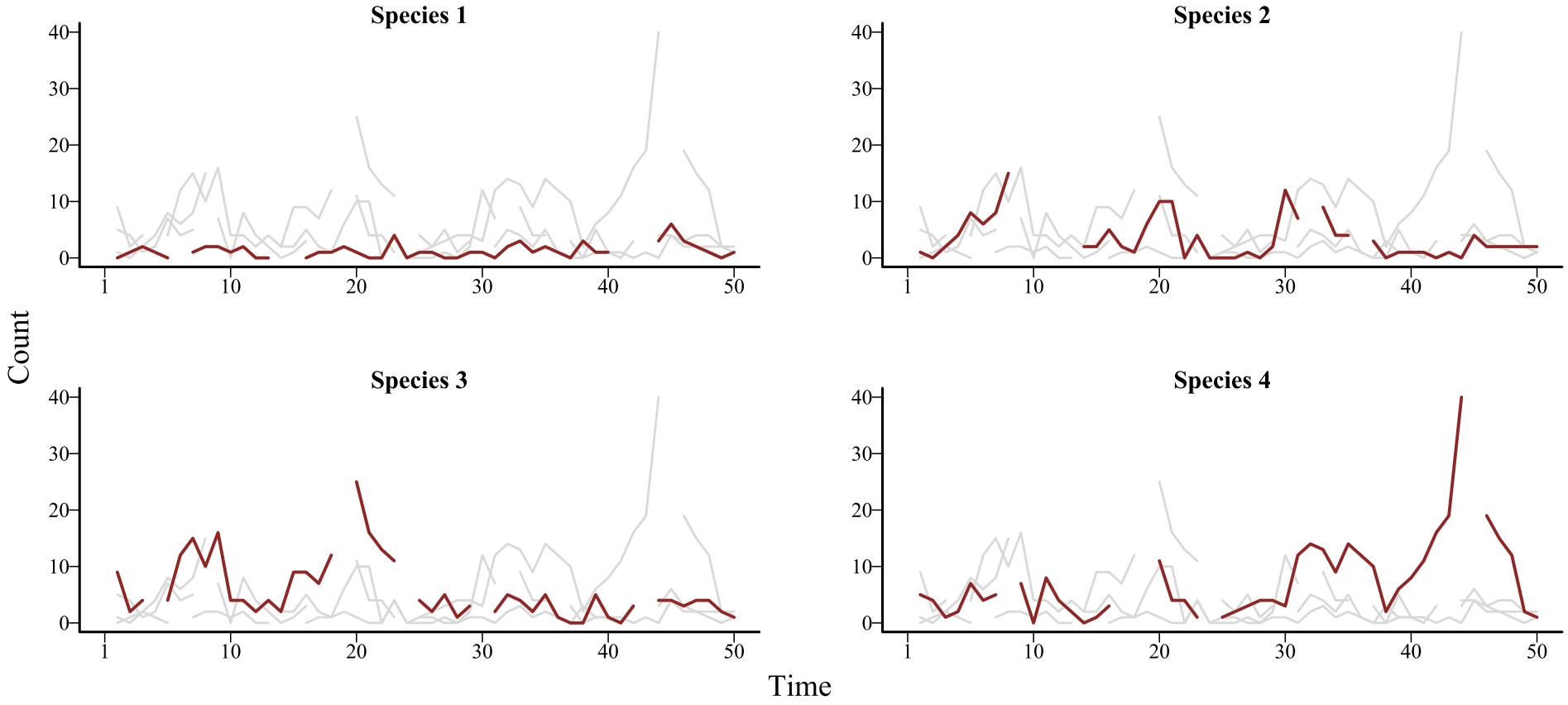
Vector autoregressive processes

Dynamic factor models

Multivariate forecast evaluation

Multivariate ecological time series

We often measure *multiple* series



Applicable for many situations

Multivariate time series arise when we have:

- Multiple species in one site
- Same species in multiple sites
- Multiple subjects in an experiment
- Multiple plots within a site
- etc...

Often the structure of the data is grouped in some way (i.e. ***hierarchical***)

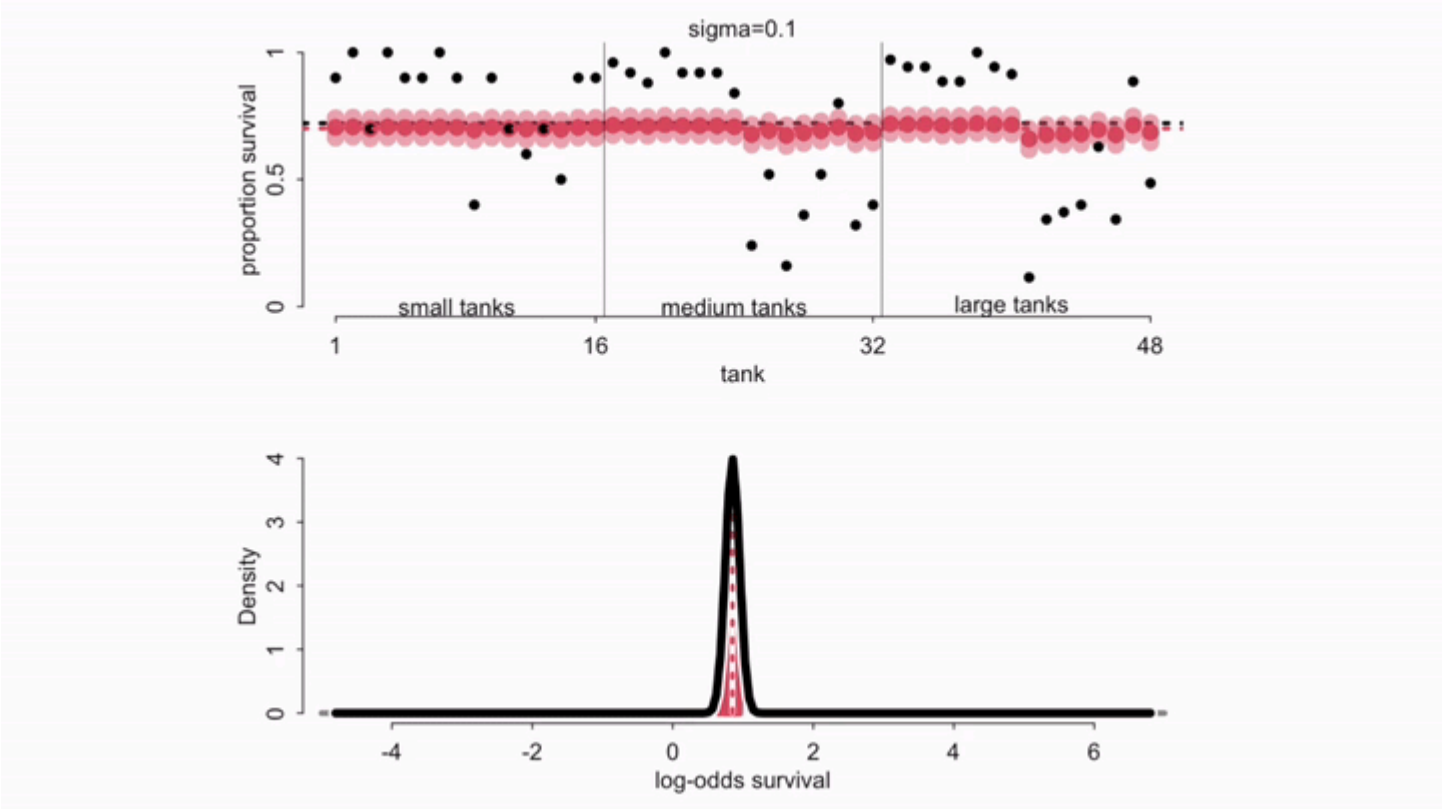
Both `mvgam` and `brms` 📦 's were designed to handle this kind of data

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

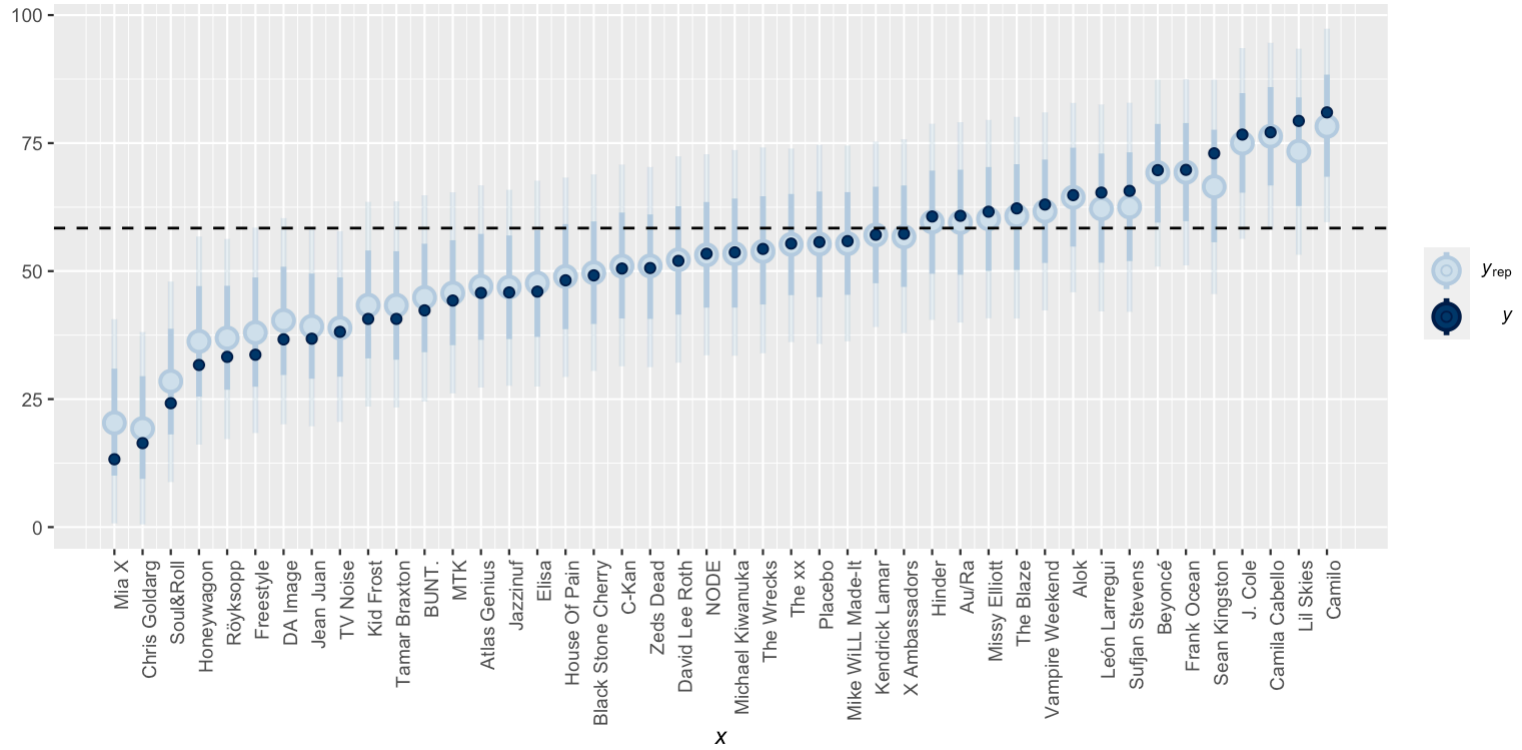
The regularization is known as partial pooling

Partial pooling in action



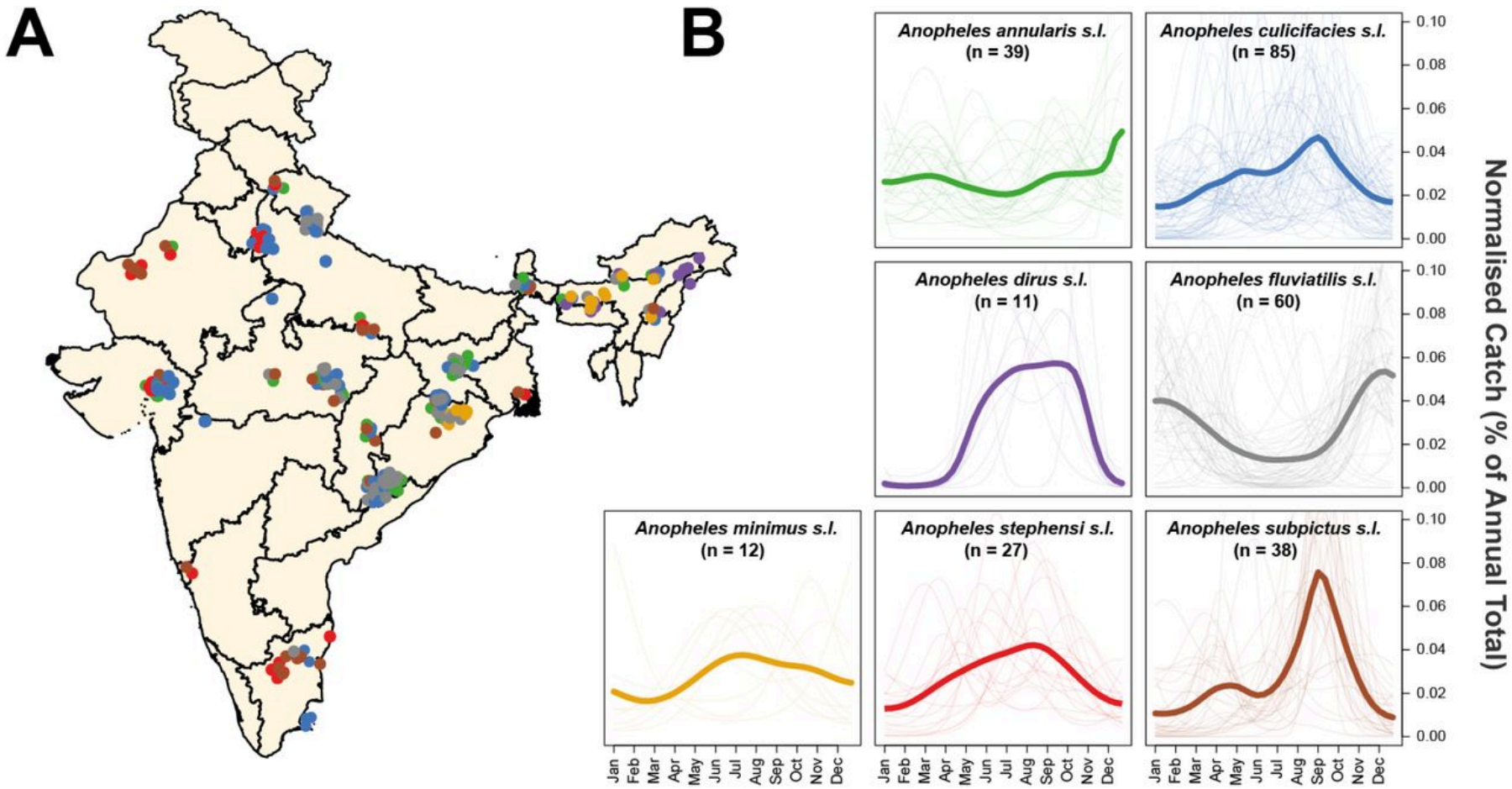
McElreath 2023

Noisy estimates *pulled* to the mean



Johnson *et al* 2021

What about *nonlinear* effects?



Whittaker et al 2022

What about *nonlinear* effects?

We very often expect to encounter nonlinear effects in ecology

But if we measure multiple species / plots / individuals etc.. through time, we can also encounter hierarchical nonlinear effects

- Same species may respond similarly to environmental change over different sites
- Different species may respond similarly in the same site

Our data may not be rich enough to estimate all effects individually; so what can we do?

Simulate some hierarchical data

Code Data

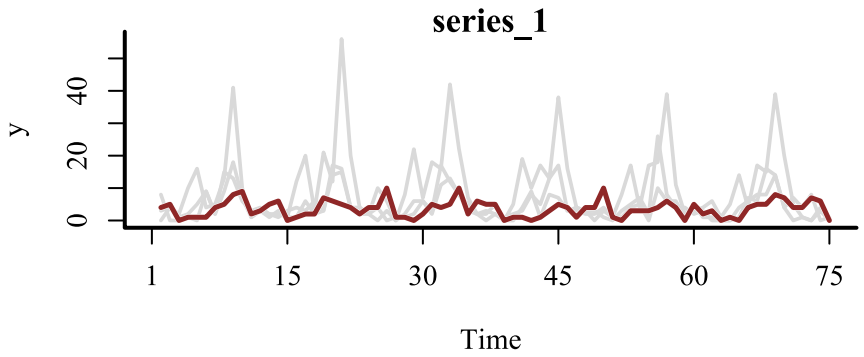
```
library(mvgam)

# set a seed for reproducibility
set.seed(650)

# simulate four time series with very little
# trend and with hierarchical seasonality
simdat ← sim_mvgam(seasonality = 'hierarchical',
                   trend_rel = 0.05,
                   n_series = 4,
                   mu = c(1, 1, 2, 1.2))
```

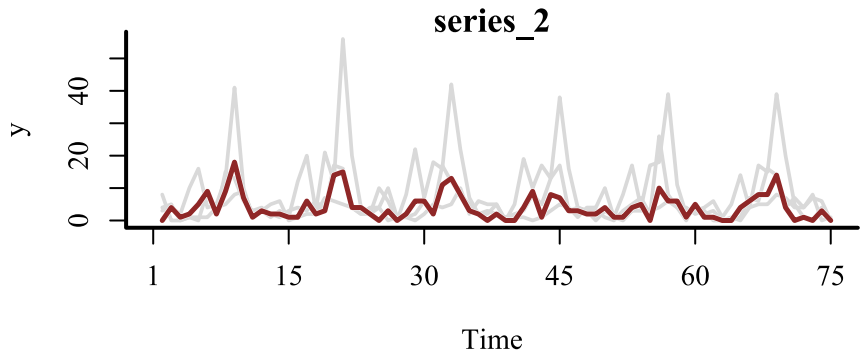
Simulate some hierarchical data

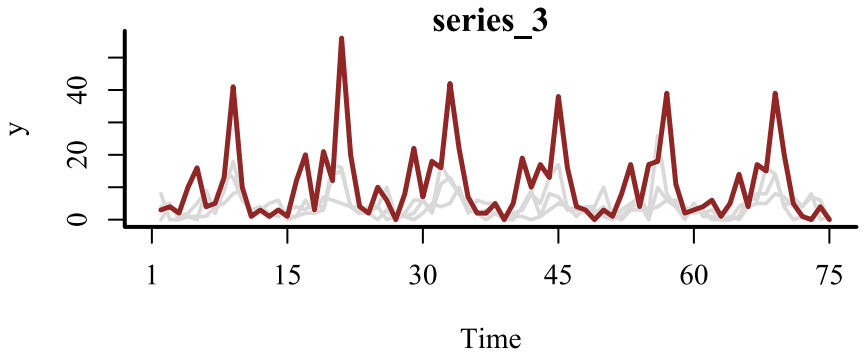
Code Data

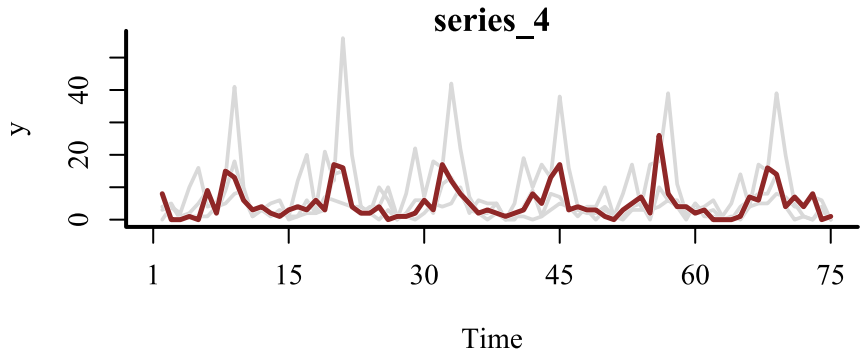
y	season	year	series	time
4	1	1	series_1	1
0	1	1	series_2	1
3	1	1	series_3	1
8	1	1	series_4	1
5	2	1	series_1	2
4	2	1	series_2	2

The series

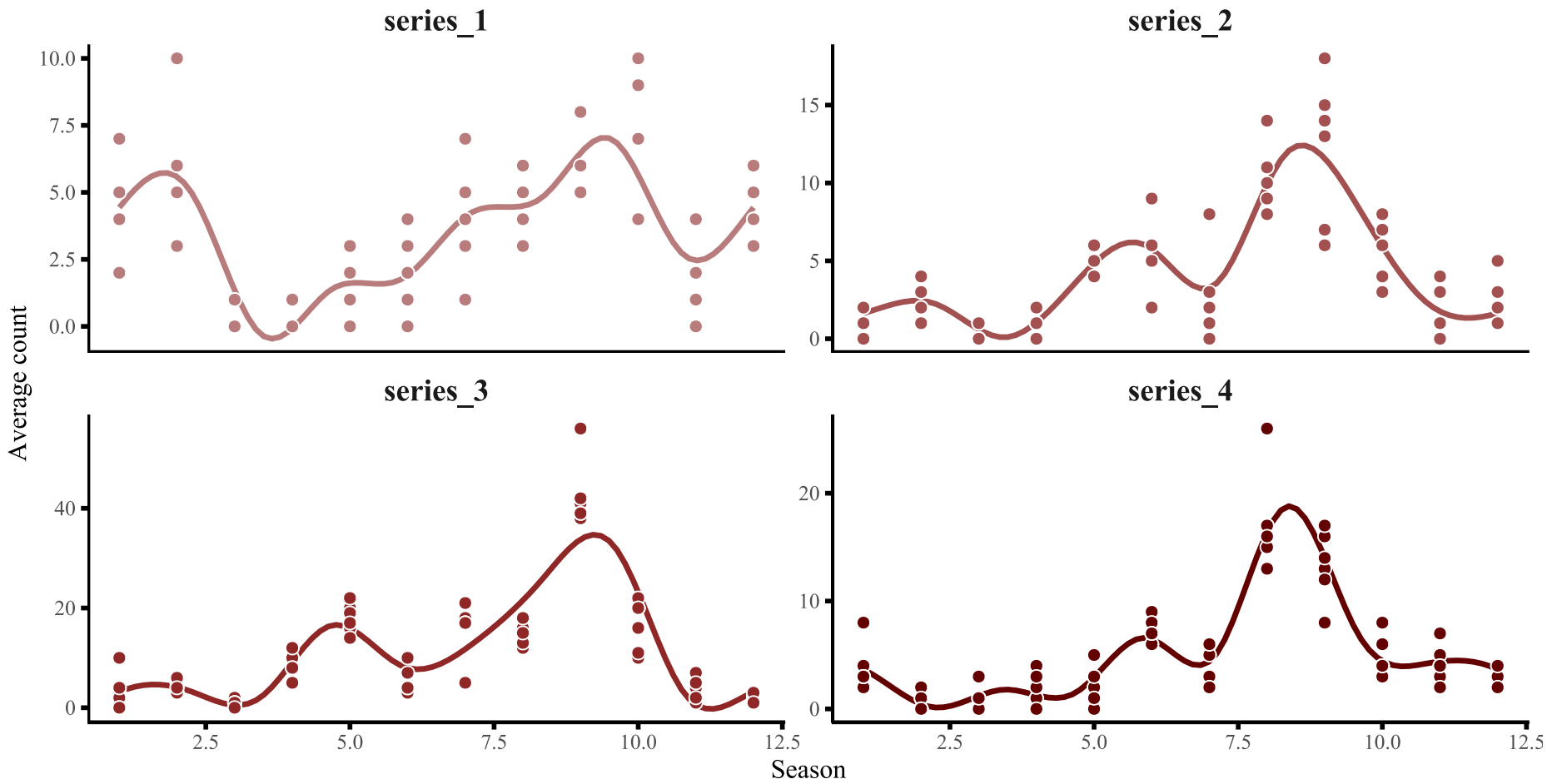

series_1
y
0
20
40
1
15
30
45
60
75
Time


series_2
y
0
20
40
1
15
30
45
60
75
Time


series_3
y
0
20
40
1
15
30
45
60
75
Time


series_4
y
0
20
40
1
15
30
45
60
75
Time

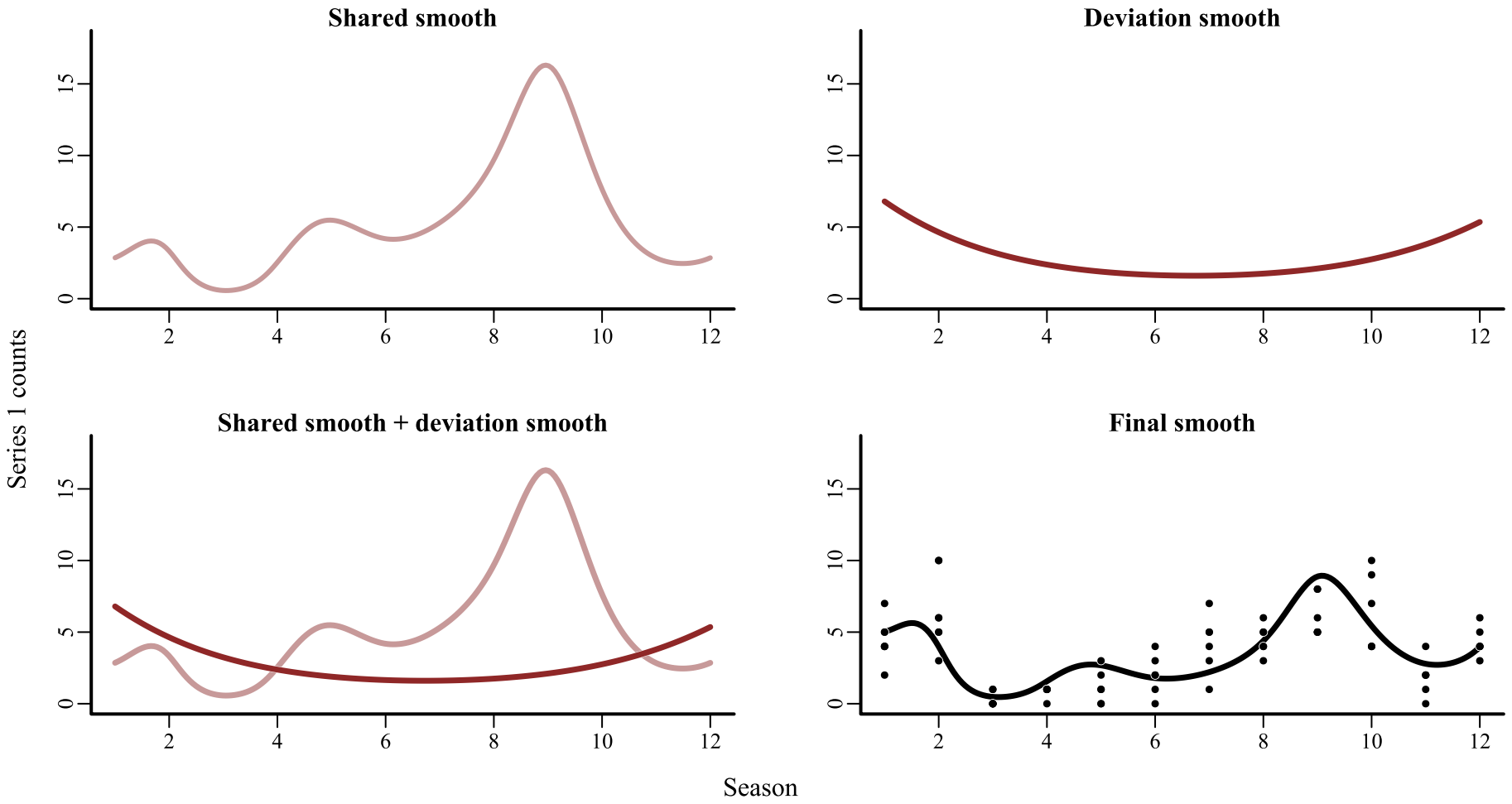
Similar seasonal patterns



Can we somehow estimate the average population smooth *and* a smooth to determine how each series deviates from the population?

Yes! We can use hierarchical GAMs

Decomposing seasonality



How did we model this?

```
mod ← mvgam(y ~
              s(season, bs = 'cc', k = 12) +
              s(season, series, k = 6, bs = 'fs'),
            data = data,
            family = poisson())
```

How did we model this?

```
mod ← mvgam(y ~
              s(season, bs = 'cc', k = 12) +
              s(season, series, k = 6, bs = 'fs'),
            data = data,
            family = poisson())
```

A ***shared*** smooth of seasonality

This is a group-level smooth, similar to what we might expect the average seasonal function to be in this set of series

How did we model this?

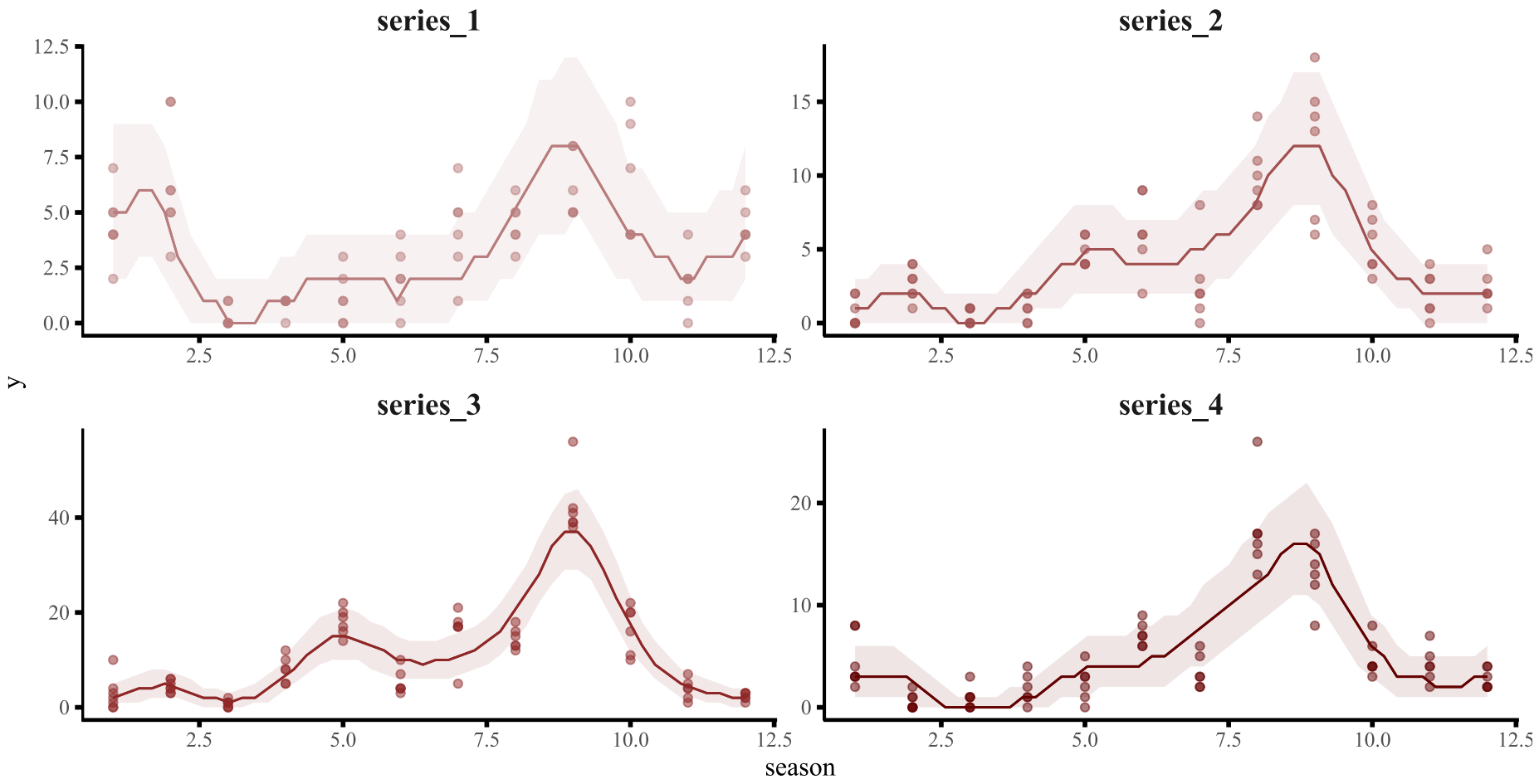
```
mod ← mvgam(y ~
              s(season, bs = 'cc', k = 12) +
              s(season, series, k = 6, bs = 'fs'),
            data = data,
            family = poisson())
```

Series-level ***deviation*** smooths of seasonality, which all share a common smoothing penalty

These are individual-level smooths that capture how each series' seasonal pattern differs from the shared smooth

- There are a number of ways to do this using splines
- See Pedersen et al 2019 for useful guidance

Conditional predictions



Hierarchical GAMs (HGAMs)

By decomposing the linear predictor into a set of additive smooths, HGAMs offer a number of very useful ways to model multivariate data

Each of these strategies allows us to:

- Learn smooth functions for each series using data from ***all series in the data***
- Regularize functions when using noisy and/or sparse data
- Make predictions for ***new series that haven't been measured yet***

See `?mgcv::factor.smooth.interaction` for more details

But what if we have smooths of two or more covariates? Can we still learn these hierarchically?

Hierarchical smooth interactions

```
mvgam(y ~
        te(temp, month, k = c(4, 4)) +
        te(temp, month, k = c(4, 4), by = series),
      family = gaussian(),
      data = plankton_train,
      newdata = plankton_test,
      trend_model = 'None')
```

Here each series is given a tensor product smooth of two covariates

But we learn the series-specific smooth as a deviation from the "shared" smooth

More on this example in Tutorial 4

Hierarchical distributed lags

```
mvgam(y ~  s(ndvi_ma12, trend, bs = 're') +
        te(mintemp, lag, k = c(3, 4), bs = c('tp', 'cr')) +
        te(mintemp, lag, by = weights_dm, k = c(3, 4), bs = c('tp', 'cr')) +
        te(mintemp, lag, by = weights_do, k = c(3, 4), bs = c('tp', 'cr')) +
        te(mintemp, lag, by = weights_ot, k = c(3, 4), bs = c('tp', 'cr')) +
        te(mintemp, lag, by = weights_pp, k = c(3, 4), bs = c('tp', 'cr')),
      ... )
```

Here the effect of minimum temperature changes smoothly over lags for each series, using the `by` argument with a set of series-specific weights for deviations

More on this type of example in this recent blogpost

HGAMs offer a solution to estimate the hierarchical, nonlinear effects that we think are common in ecology

This is a huge advantage over traditional time series models

But how can we handle multivariate *dynamic components*?

Live code example

Vector autoregressive processes

VAR1

Similar to an AR1, but the response is now a ***vector***

$$x_t \sim \text{Normal}(A * x_{t-1}, \Sigma)$$

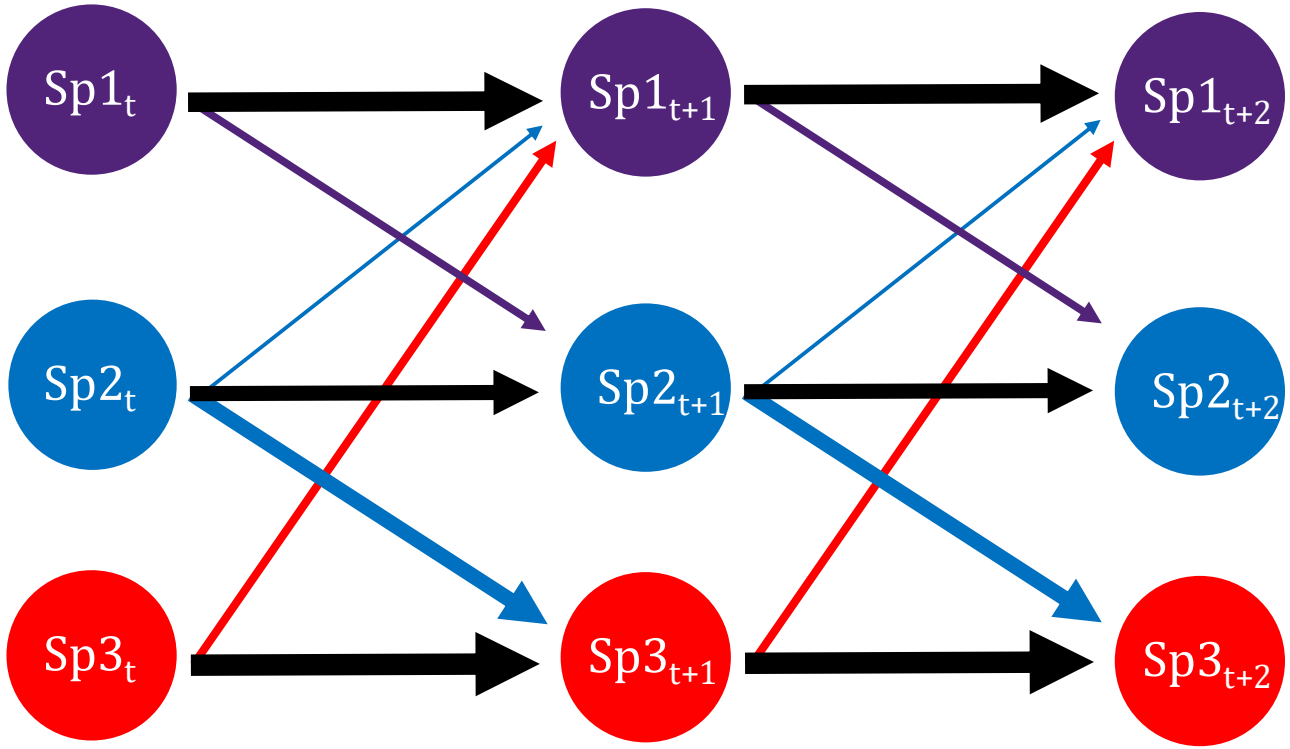
Where:

x_t is a vector of time series observations at time t

Σ determines the spread (or flexibility) of the process and any correlations among time series errors

A is a matrix of coefficients estimating lagged dependence and cross-dependence between the elements of x

VAR1



Properties of a VAR1

If off-diagonals in both A and Σ = 0, process is an AR1

If off-diagonals in A, but not in Σ, = 0, process is an AR1 with correlated errors

If A has no zero entries, the process can quickly become ***nonstationary***, leading to explosive forecasts

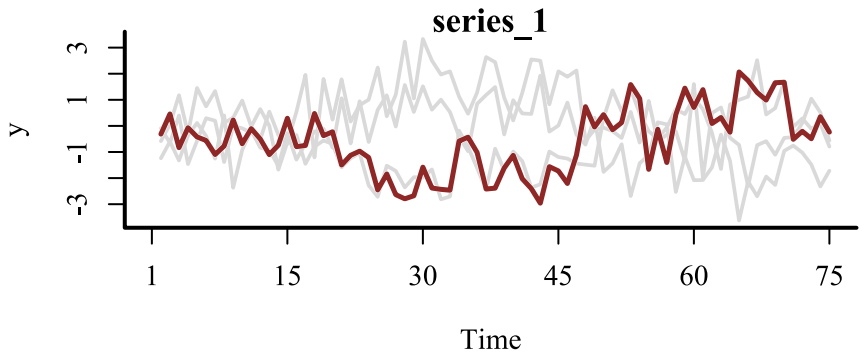
Advantages of VARs

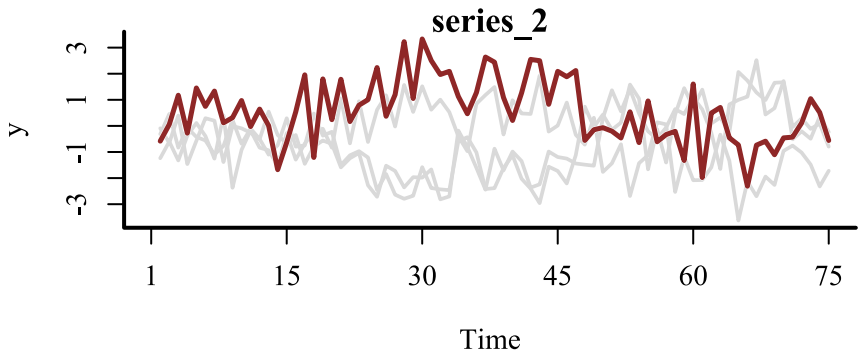
They allow us to explore whether one response variable can be useful in predicting another, giving us a means to tackle ***interactions***

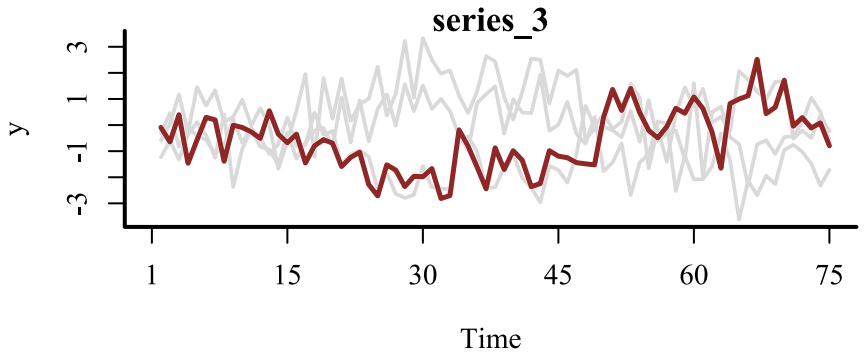
They allow for impulse response analyses, where the response of one variable to a sudden but temporary change in another variable can be predicted

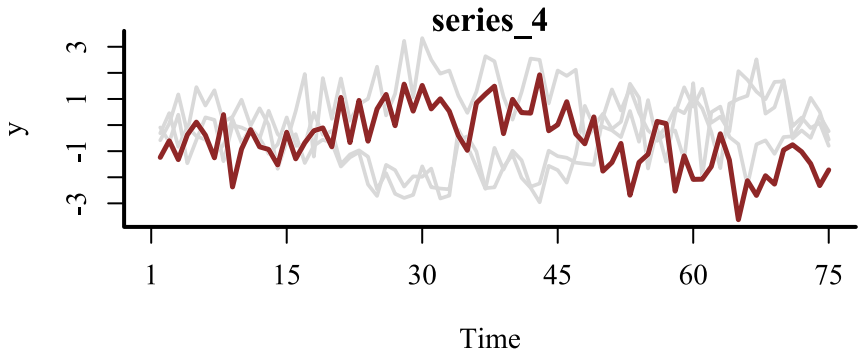
They allow us to identify which other responses are most influential in driving our uncertainty in a focal response

Some simulated data









Latent VAR1s in mvgam 📦

```
varmod ← mvgam(y ~ 1,
               trend_model = VAR(),
               data = data_train,
               newdata = data_test,
               family = gaussian())
```

If multiple series are included in the data, we can use a VAR1 to estimate latent dynamics

- `trend_model = VAR()`: a VAR1 with uncorrelated process errors (off-diagonals in Σ set to zero)
- `trend_model = VAR(cor = TRUE)`: a VAR1 with possibly correlated process errors

Enforcing stationarity

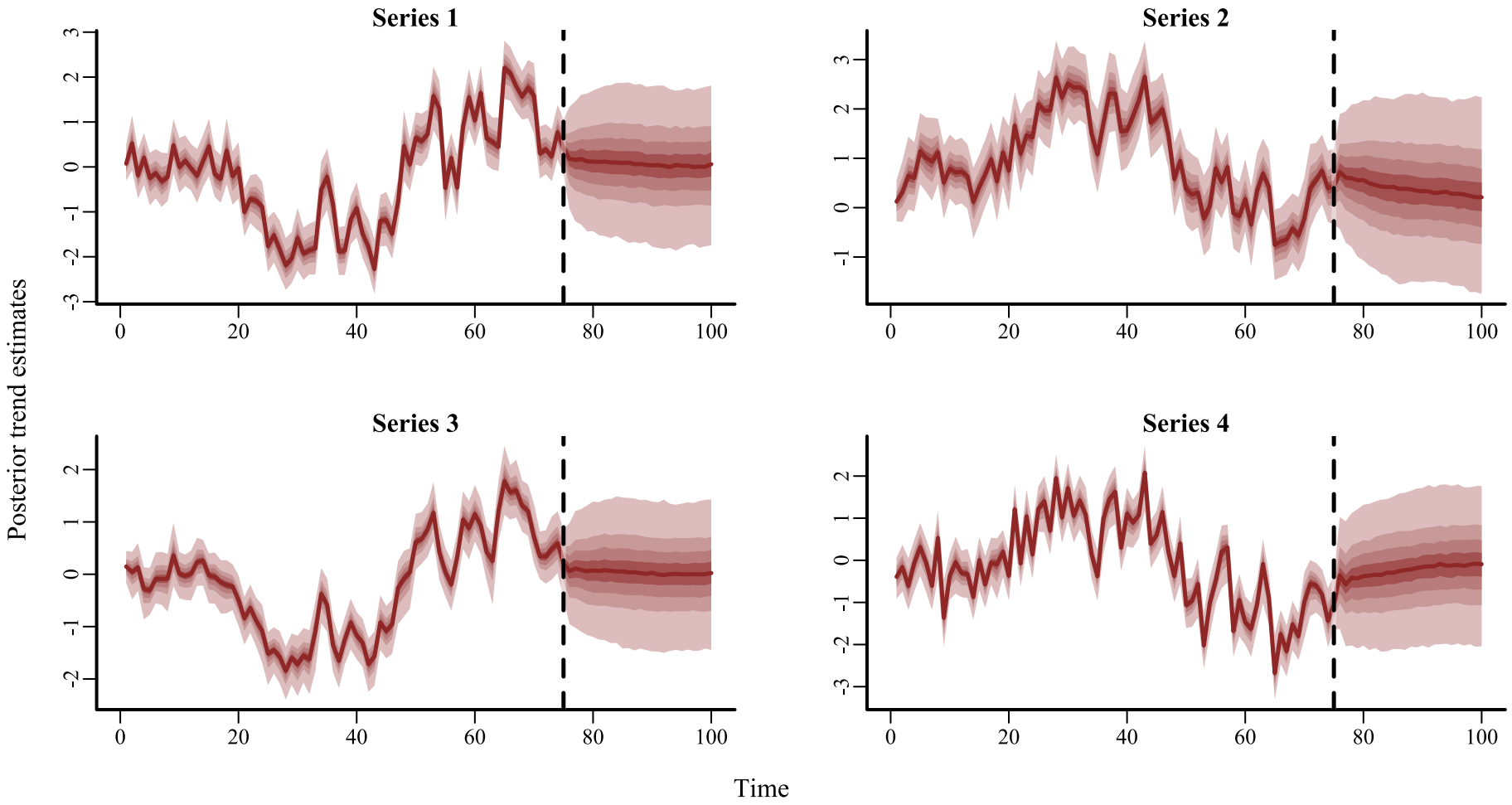
Forecasts from VAR models, especially those with many series, can very quickly become nonstationary

Not only will forecast variance increase indefinitely, but dynamics can become unstable and forecasts can ***explode***

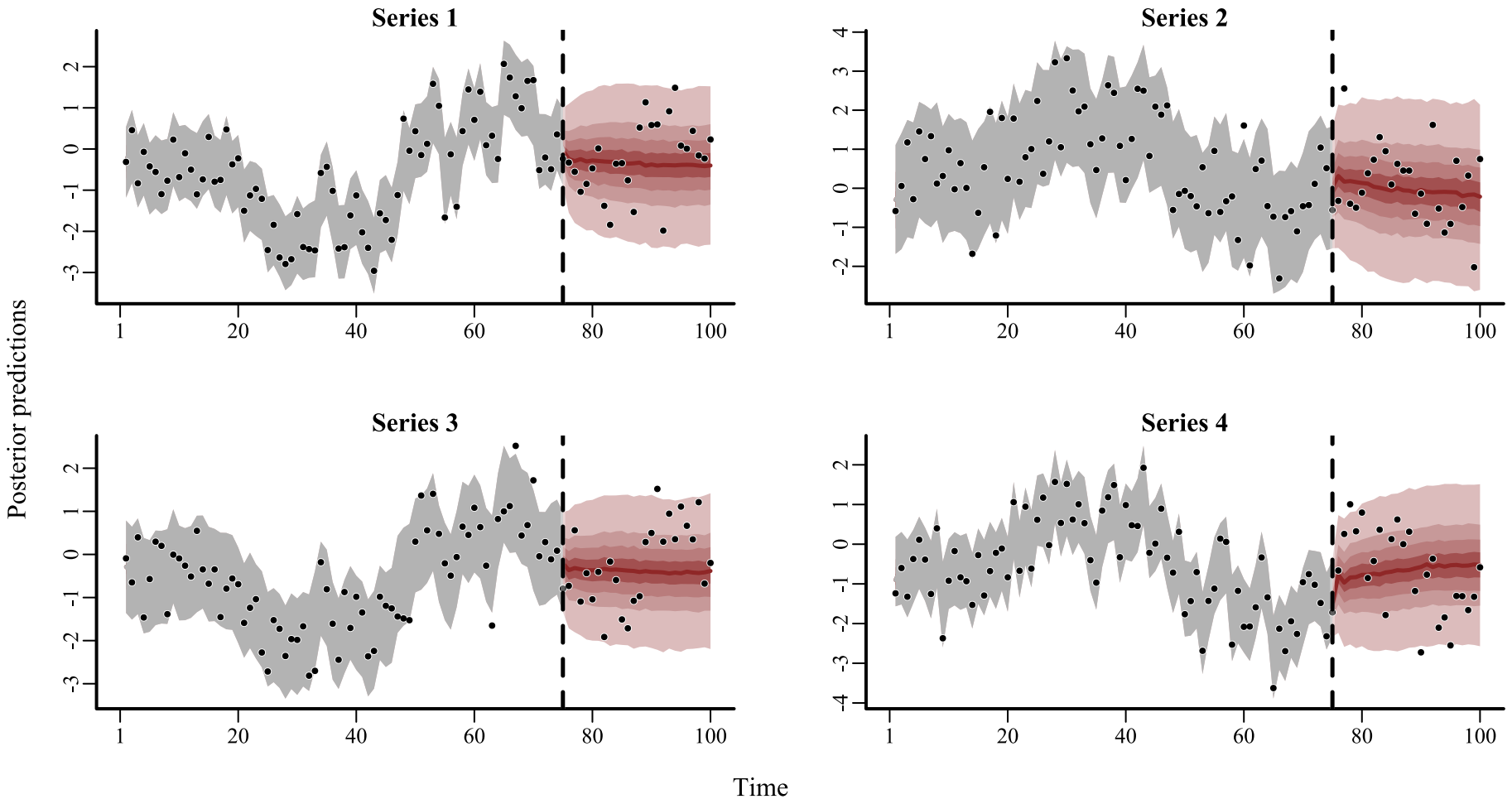
`mvgam` 📦 enforces stationarity in all VAR1 models using careful prior choices

- Maps the process to unconstrained partial autocorrelations, which can be constrained to preserve stationarity
- Derived and explained in detail by Sarah Heaps here and here

Trend estimates



Hindasts / forecasts



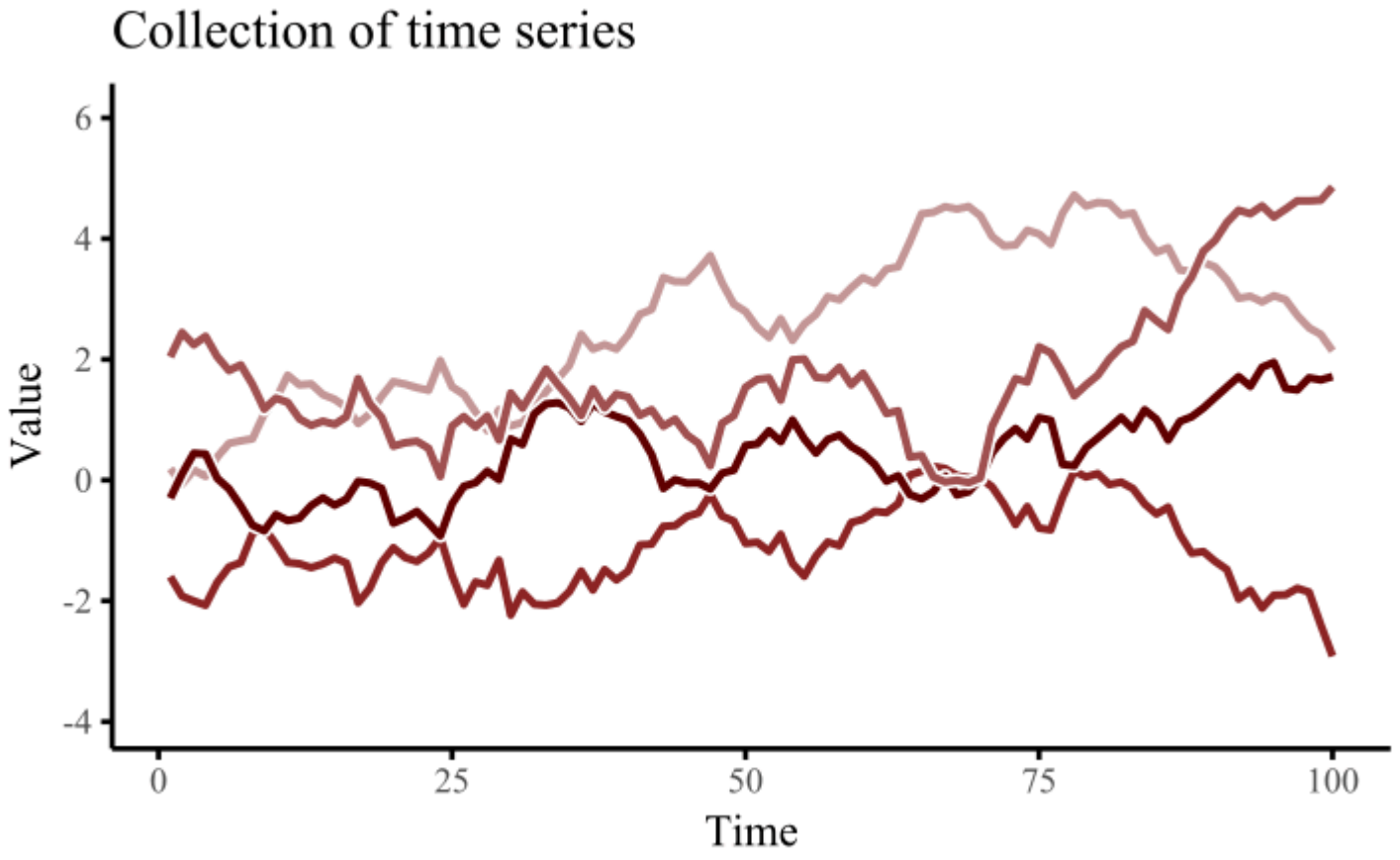
VAR models give us a tool to capture complex multi-series dynamics while providing sensible forecasts

But often we have enough series that estimating VAR parameters becomes intractible

How can we achieve dimension reduction when forecasting multiple series?

Dynamic factor models

Inducing multiseries correlations



Dynamic factors

Propagate a set of latent factors and allow each observed series to ***depend*** on these factors with a set of weights

$$x_t = \theta * z_t$$

Where:

x_t is a vector of time series observations at time t

z_t is a vector of dynamic factor estimates at time t

θ is a matrix of loading coefficients that control how each series in x depends on the factors in z

Dynamic factor models are hugely flexible

The factors can take on a number of possible time series dynamics (RW, AR, GP)

Similar loadings ⇨ correlated trends

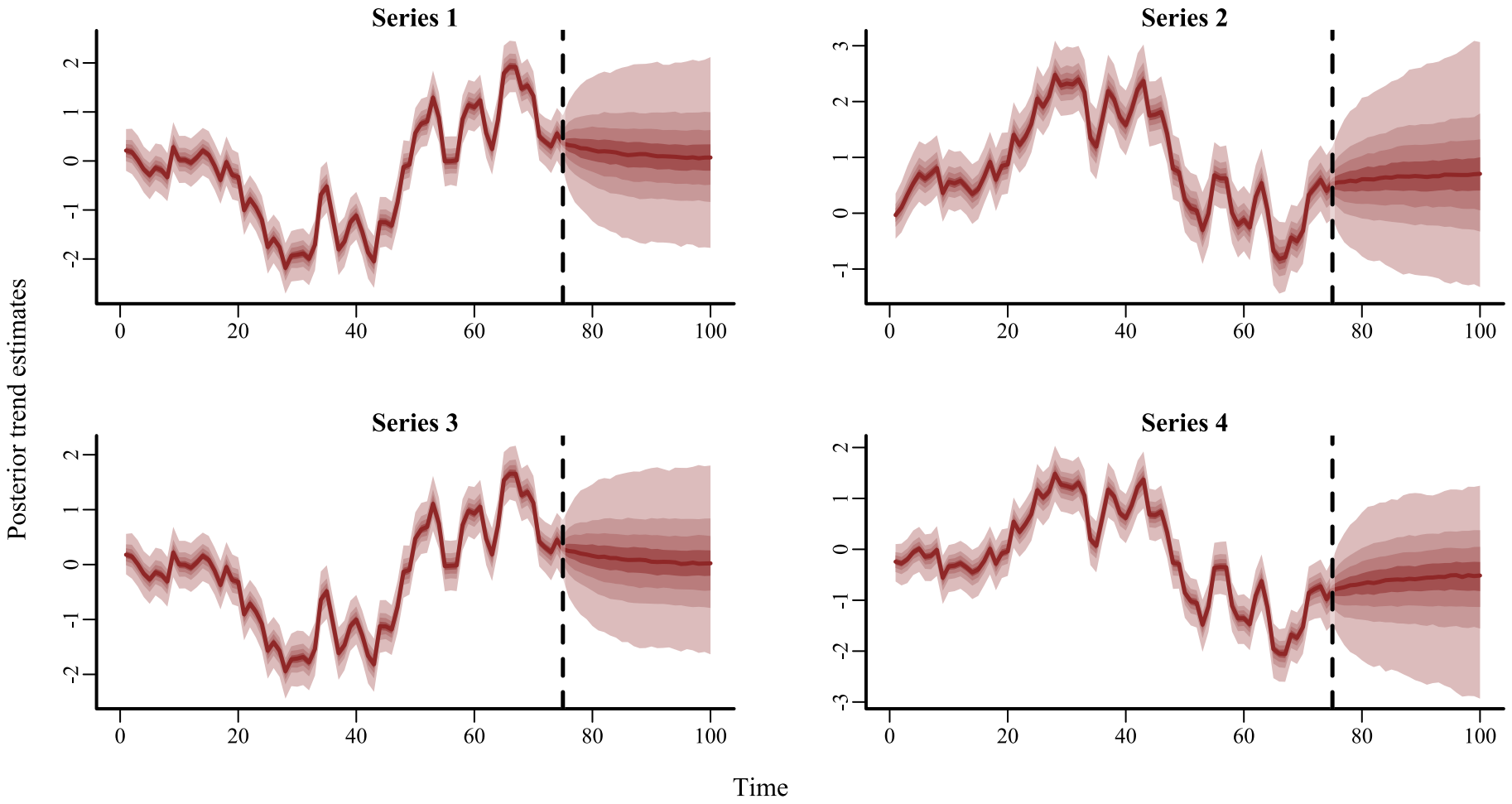
Dynamic factors in `mvgam` 📦

```
dfmod ← mvgam(y ~ 1,
              use_lv = TRUE,
              n_lv = 2,
              trend_model = AR(),
              data = data_train,
              newdata = data_test,
              family = gaussian())
```

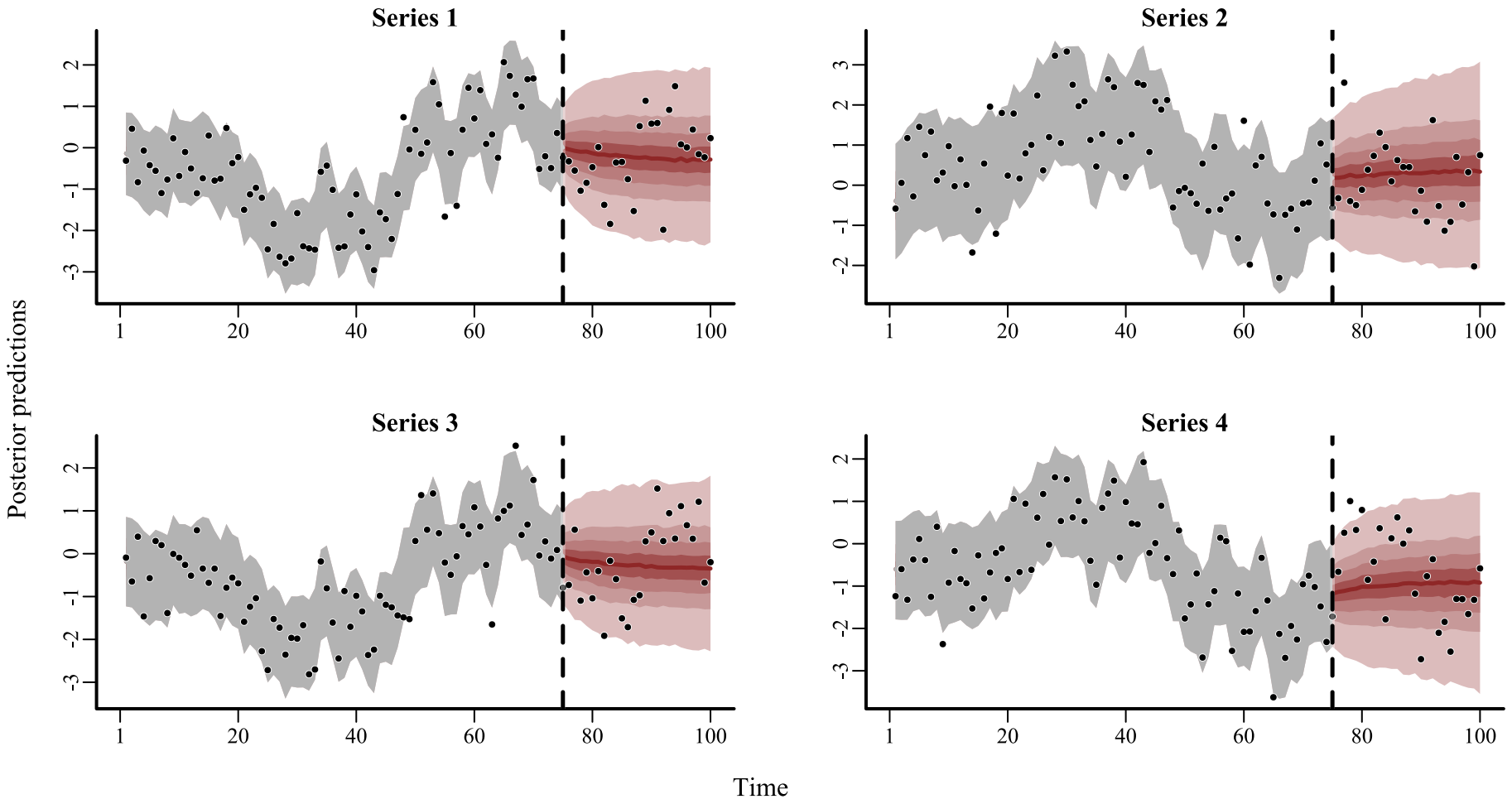
If multiple series are included in the data, we can use a dynamic factor model to estimate latent dynamics

- `n_lv`: the number of latent factors to estimate
- `trend_model`: can be `RW()`, `AR(p = 1, 2, or 3)` or `GP()`
- variance parameters for all factors fixed to ensure identifiability

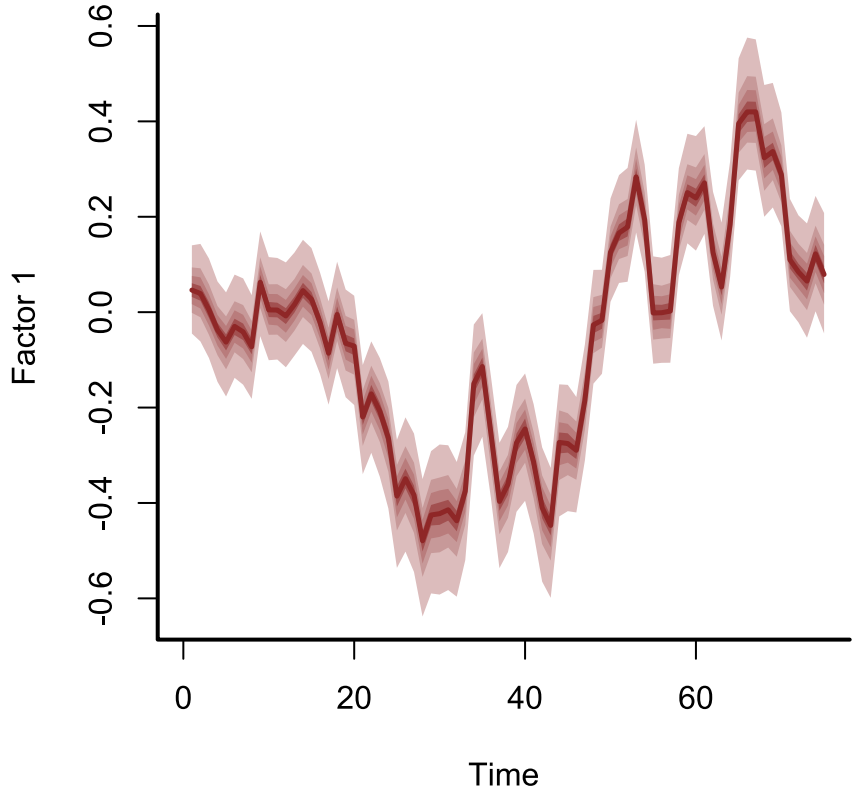
Trend estimates

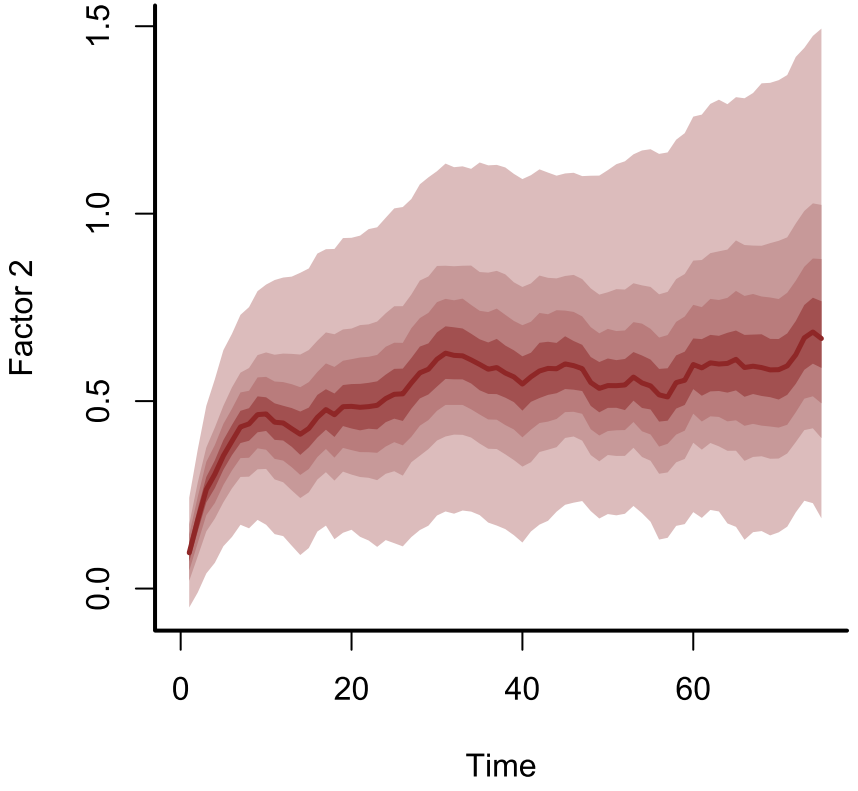


Hindasts / forecasts

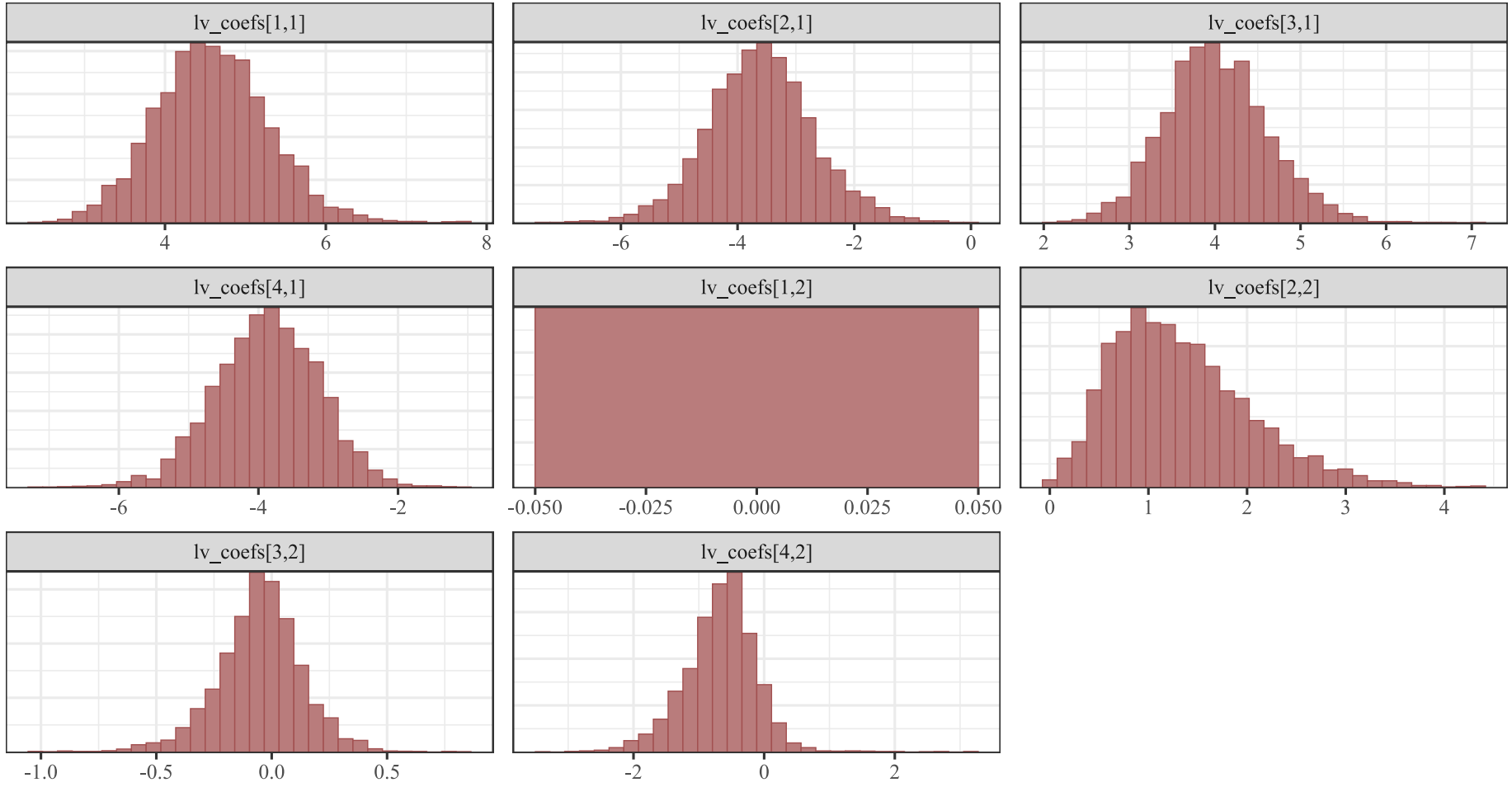


```
plot_mvgam_factors(dfmod)
```





```
mcmc_hist(dfmod$model_output, regex_pars = 'lv_coefs')
```

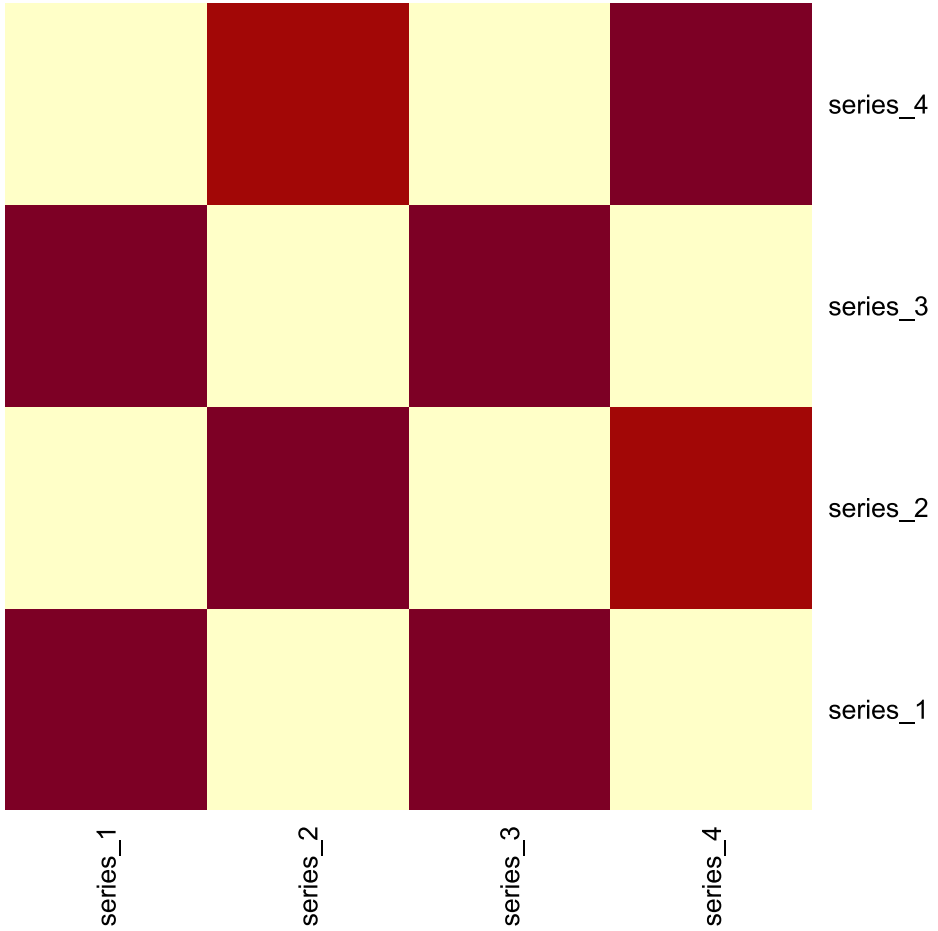


Induced correlations

```
mean_corrs ← lv_correlations(object = dfmod)$mean_correlations
mean_corrs
```

	series_1	series_2	series_3	series_4
series_1	1.00	-0.92	1.00	-0.97
series_2	-0.92	1.00	-0.92	0.83
series_3	1.00	-0.92	1.00	-0.96
series_4	-0.97	0.83	-0.96	1.00

```
heatmap(mean_corrs, distfun = function(c) as.dist(1 - c))
```



We have many ways to estimate multivariate dynamic components in the `mvgam` framework

VARs can approximate Granger causality and allow for targeted hypotheses about interactions to be evaluated; Dynamic factors can capture time series correlations with fewer parameters

But how do we go about *evaluating* forecasts to choose among models?

Live code example

Multivariate forecast evaluation

Energy score

Generalizes the CRPS for multivariate forecasts

$$ES(F, y) = \frac{1}{m}\sum_{i=1}^{m}||F_i - y|| - \frac{1}{2m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}||F_i - F_j||$$

Where:

$F(\hat{y})$ is a set of m samples from the forecast distribution

$||\cdot||$ is the Euclidean norm

Essentially a weighted distance between the forecast distribution and distribution of observations

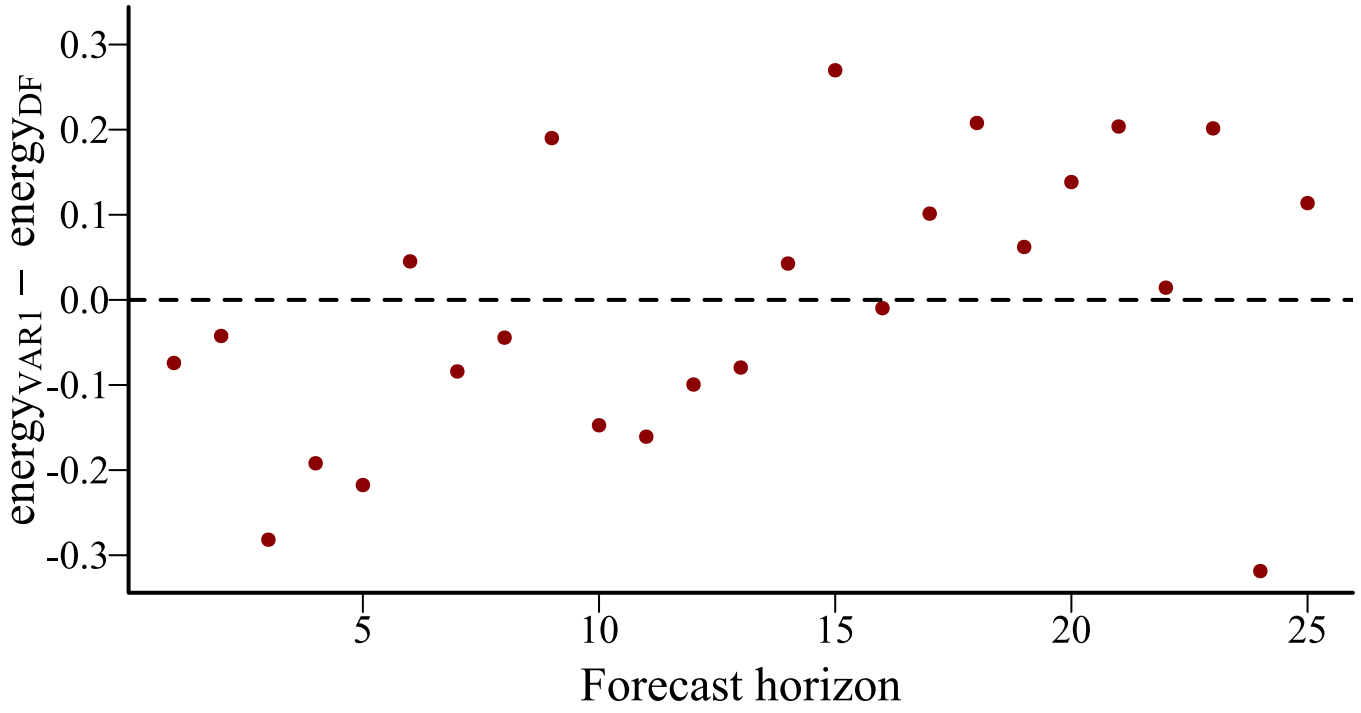
Compare energy scores

Code Plot

```
# score forecasts from each model and
# compute differences (VAR scores - DF scores)
fc_var ← forecast(varmod); fc_df ← forecast(dfmod)
diff_scores ← score(fc_var, score = 'energy')$all_series$score -
  score(fc_df, score = 'energy')$all_series$score

# plot the differences (negative means VAR1 was better;
# positive means DF was better)
plot(diff_scores, pch = 16, col = 'darkred',
     ylim = c(-1*max(abs(diff_scores), na.rm = TRUE),
              max(abs(diff_scores), na.rm = TRUE)),
     bty = 'l', xlab = 'Forecast horizon',
     ylab = expression(energy[VAR1]~-~energy[DF]))
abline(h = 0, lty = 'dashed', lwd = 2)
```

Compare energy scores

Code Plot



Energy score gernealizes to the univariate CRPS; is readily applicable to a variety of forecasts (ensemble, samples)

May not be able to tell us whether a forecast misses correlation structures that are evident in out of sample observations

We may also need a score that penalizes forecasts which do not replicate the observed correlation structure

Variogram score

Asks if forecast distribution captures ***correlation structure*** evident in observations

$$Variogram(F, y) = \sum_{i,j=1}^{d} w_{ij}(\sqrt{|y_i - y_j|} - \sqrt{|F_i - F_j|})^2$$

Where:

$F(\hat{y})$ is the mean (or median) forecast for the d series

w is a set of non-negative weights

Penalizes forecasts whose variograms don't match the observed variogram

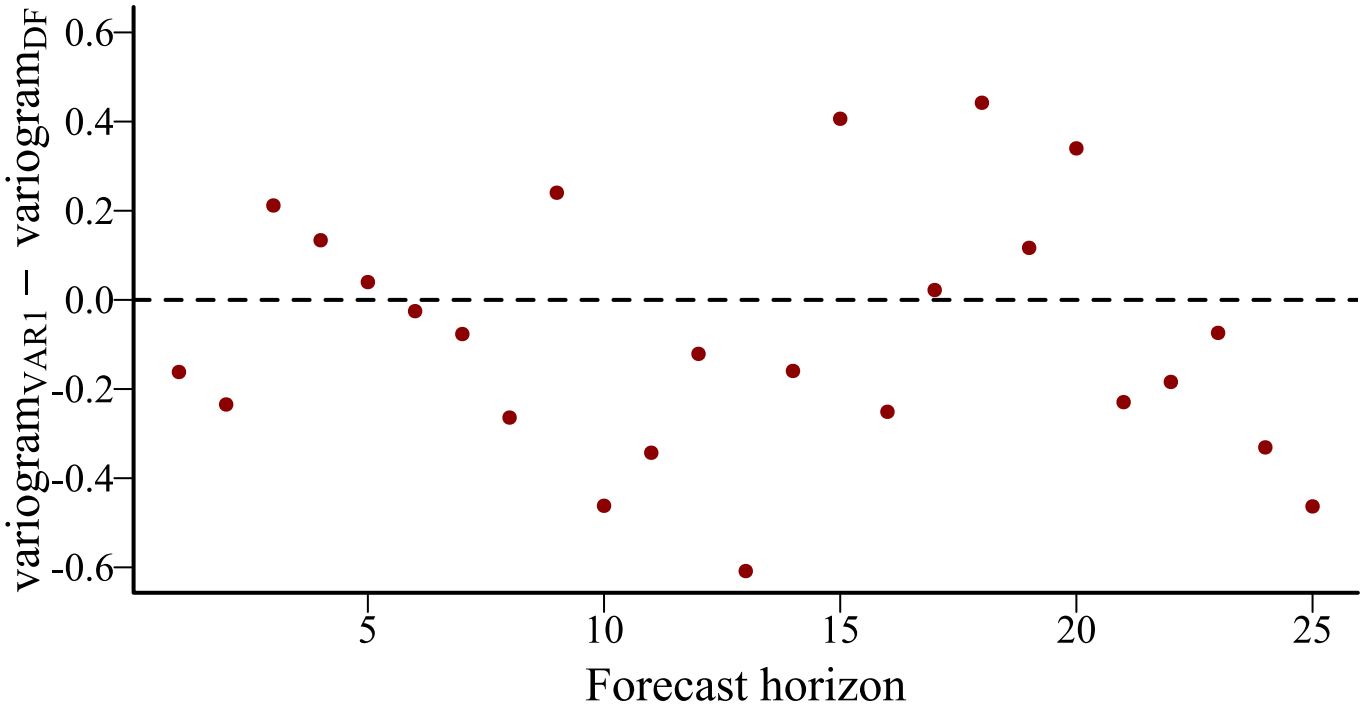
Compare variogram scores

Code Plot

```
# score forecasts from each model and
# compute differences (VAR scores - DF scores)
fc_var ← forecast(varmod); fc_df ← forecast(dfmod)
diff_scores ← score(fc_var, score = 'variogram')$all_series$score -
  score(fc_df, score = 'variogram')$all_series$score

# plot the differences (negative means VAR1 was better;
# positive means DF was better)
plot(diff_scores, pch = 16, col = 'darkred',
     ylim = c(-1*max(abs(diff_scores), na.rm = TRUE),
              max(abs(diff_scores), na.rm = TRUE)),
     bty = 'l', xlab = 'Forecast horizon',
     ylab = expression(variogram[VAR1]~-~variogram[DF]))
abline(h = 0, lty = 'dashed', lwd = 2)
```

Compare variogram scores

Code Plot



The variogram score is more useful for directly asking if a forecast distribution resembles the dependency structure in the observations

But it only uses a summary of the forecast (i.e. the mean or median forecast for each series), so does not address the sharpness or calibration of the forecast

Weighted combinations of variogram and energy scores may be necessary to fully characterize the performance of a forecast

In the next lecture, we will cover

Extended practical examples using `mvgam`

Review and open discussion