mvgam :: CHEATSHEET

The mvgam package provides tools for fitting and interrogating univariate or multivariate State-Space time series models that can include nonlinear smooth functions of covariates, dynamic temporal processes and random effects. A wide variety of latent dynamic processes can be specified. The package also provides tools for interpreting effects, computing and scoring forecasts, as well as generating model code and data objects for further customisation. Models are fitted using Stan for full Bayesian inference.

Modelling with mvgam()

Usage: `mvgam(formula, trend_formula, data, family, ...)`

`formula`: observation model regression formula, in brms syntax with mgcv smooths. See `?mvgam` for more guidance

`trend_formula`: optional process model formula. It names the latent dynamic process with a trend constructor, such as `~ RW()`. It can also hold covariates of the process

`data`: a `data.frame` or `list` containing the response variable(s) and optional predictor variables. See the data formatting vignette for guidance on data preparation

Trend constructors for `trend_formula`. Options include (among others):

- No `trend_formula`: default, no dynamic process
- `RW(ma = FALSE, cor = FALSE)`: random walk
- `AR(p = 1, ma = FALSE, cor = FALSE)`: autoregressive
- `VAR(ma = FALSE)`: vector autoregressive
- `PW(growth = 'linear')`: piecewise linear, or `'logistic'` with saturation
- `CAR()`: continuous-time autoregressive, for irregular sampling times

For autoregressive processes (`RW(), AR() or VAR()`), moving average and correlated process errors can also be specified with the `ma` and `cor` arguments. `n_lv` sets dynamic factors. `gr` and `subgr` pool hierarchical trends

`family`: observation distribution. Options include (among others):

- Continuous: `gaussian()`, `student()`, `lognormal()`, `Gamma(link = 'log')`, `Beta()`
- Counts: `poisson()`, `negbinomial()`, `beta_nb()`, `tweedie()`
- Trials and binary: `binomial()`, `bernoulli()`, `com_binomial()`
- Imperfect detection: `nmix()`, `occ()` (N-mixture article)
- Multi-response: `mvn()`, `mvt()`, `diri()`, `multi()`, `categ()`

Multivariate models give each response its own family, as in `bf(y1 ~ x, family = poisson()) + bf(y2 ~ x, family = Beta())`. The responses can share one latent process (multivariate and integrated data articles). `jsdgam()` fits joint species models

See the introductory vignette for more guidance on supported families and dynamic processes

`...`: other arguments such as user-specified `priors`, `newdata` for generating probabilistic forecasts and options to control `Stan` MCMC parameters

Prior to modelling, it is useful to:

- Check the data and plot each series with `mvgam_data()`
- Ensure there are no `NA`'s in predictors (though `NA`'s are allowed in response variables)
- Inspect default priors with `get_prior()`
- Make any necessary changes to default priors with `prior()`

`sim_mvgam()` is useful to generate simple example datasets

```
simdat <- sim_mvgam(family = poisson(), trend_model = RW())
model <- mvgam(formula = y ~ s(x, k = 5), trend_formula = ~ RW(),
               data = simdat$data_train, family = poisson())
```

Use `stancode(model)` to see the auto-generated Stan code

Diagnostics and Inference

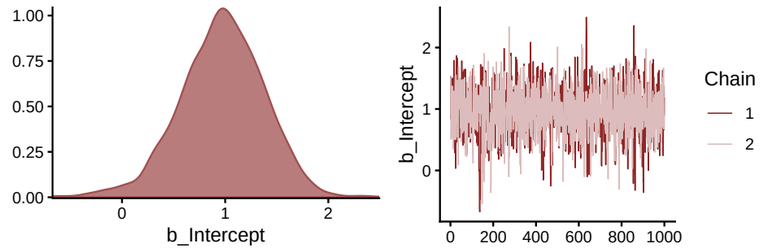
What effects has the model estimated?

`summary(model)` and `fixef(model)`: posterior summaries and diagnostics

`fitted(model)`, `logLik(model)` and `residuals(model)`: posterior expectations, pointwise Log-Likelihoods and randomized quantile residuals

`loo(model)` and `loo_compare(model1, model2, ...)`: calculate approximate leave-one-out information criteria for model comparisons

`mcmc_plot(model)`: visualize posterior summaries, pairs plots and a wide range of MCMC diagnostics using functionality from the bayesplot package

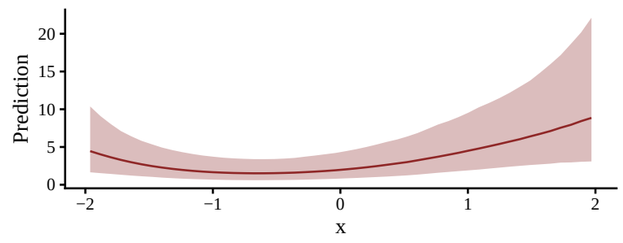


Use `as.data.frame(model)`, `as.matrix(model)`, or `as.array(model)` to extract posterior parameter estimates. Use `variables(model)` to determine what parameters are available for extraction

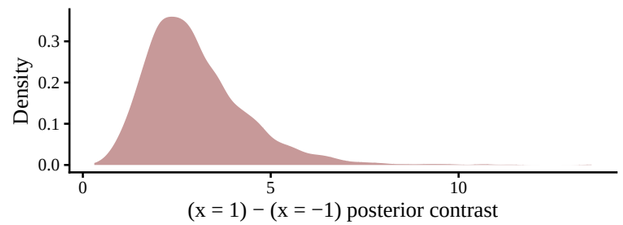
The S3 `plot()` function applied to models can visualise smooth functions (`type = 'smooths'`), trend estimates (`type = 'trend'`), the observed series (`type = 'series'`), latent factors (`type = 'factors'`) or randomized quantile residual diagnostics (`type = 'residuals'`)

`residual_cor()`, `irf()`, `fevd()`, `ordinate()`: latent dynamics

`conditional_effects(model)` gives useful conditional effect plots on either the response or the link scale



For most mvgam models, functions from the `marginaleffects` package can be used for more targeted prediction-based inference. See The Marginal Effects Zoo and How to interpret effects from GAMs for guidance on inspecting predictions, slopes and comparisons



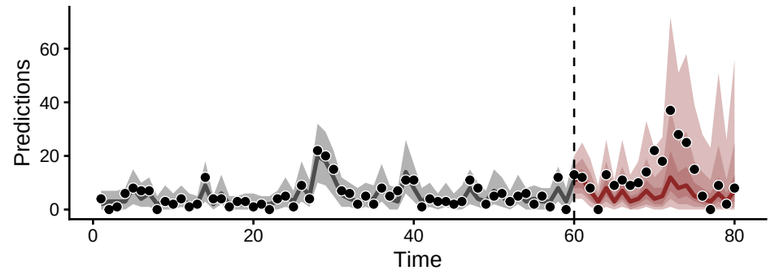
Prediction and forecasting

How good are model predictions?

Use `predict(model)` with `newdata` to make predictions for inference purposes. Change the `type` argument for different types of predictions (link scale, expectation or response scale). Or use the brms package equivalents `posterior_predict(model)`, `posterior_linpred(model)` or `posterior_epred(model)`, which omit the fitted trend unless `incl_autocor = TRUE`. If generating forecasts for future timepoints, use the `forecast()` function (see below)

Use `pp_check(model)` to compute posterior predictive checks and compare model predictions against the true observations

Extract in-sample posterior predictions with `hindcast(model)`. If validation data exist, generate forecast predictions with `forecast(model, newdata = newdata)`. As above, change the `type` argument for predictions on different scales. Both functions generate an object of class `mvgam_forecast`, that can be plotted with an S3 `plot()` function. See the forecasting vignette for more details about how to produce forecasts.



Compute probabilistic forecast scores using proper scoring rules with the `score()` function:

```
score(fc, score = 'drps')
```

```
## DRPS of 1 forecast series, lower is better
##
##  series     mean_score coverage times
##  series_1   5.1        0.95     20
##  all_series 5.1                 20
##
## 'all_series' sums the series at each time.
## 'coverage' is the share inside the 90% interval.
```

Rules include `'crps'`, `'drps'`, `'logs'` and `'sis'` for each series. `'energy'` and `'variogram'` score all series jointly

`compare_scores()` scores several models with one rule and ranks them:

```
compare_scores(GAM = fc_gam, RW = fc, score = 'drps')
```

```
## DRPS of 2 models, lower is better
##
##  model mean_score difference
##  RW    5.1        0.0
##  GAM   6.7        1.6
##
## 'difference' is the gap to the best. plot() shows each horizon.
```

`lfo_cv(model)` compares models by approximate leave-future-out cross-validation (Bürkner et al. 2020)

CC BY-SA 4.0 ○ Nicholas J. Clark ○ Learn more at https://nicholasjclark.github.io/mvgam/ ○ package version 2.0.0 ○ updated: 2026-11-26