Package 'walker'

May 8, 2026

Type Package

Title Bayesian Generalized Linear Models with Time-Varying Coefficients

Version 1.0.10

Description Efficient Bayesian generalized linear models with time-varying coefficients as in Helske (2022, <doi:10.1016/j.softx.2022.101016>). Gaussian, Poisson, and binomial observations are supported. The Markov chain Monte Carlo (MCMC) computations are done using
Hamiltonian Monte Carlo provided by Stan, using a state space representation of the model in order to marginalise over the coefficients for efficient sampling.
For non-Gaussian models, the package uses the importance sampling type estimators based on approximate marginal MCMC as in Vihola, Helske, Franks (2020, <doi:10.1111/sjos.12492>).

License GPL (>= 3)

Suggests diagis, gridExtra, knitr (>= 1.11), rmarkdown (>= 0.8.1), testthat

Depends bayesplot, R (>= 3.4.0), rstan (>= 2.26.0)

Imports coda, dplyr, Hmisc, ggplot2, KFAS, loo, methods, Rcpp (>= 0.12.9), RcppParallel, rlang, rstantools (>= 2.0.0)

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.9), RcppArmadillo, RcppEigen (>= 0.3.3.3.0), RcppParallel, rstan (>= 2.26.0), StanHeaders (>= 2.26.0)

SystemRequirements GNU make

Biarch true

VignetteBuilder knitr

RoxygenNote 7.3.1

ByteCompile true

URL https://github.com/helske/walker

BugReports https://github.com/helske/walker/issues

Encoding UTF-8

NeedsCompilation yes

Author Jouni Helske [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7130-793X>)

Maintainer Jouni Helske <jouni.helske@iki.fi>

Repository CRAN

Date/Publication 2024-08-30 06:40:02 UTC

Contents

`as.data.frame.walker_fit` *Coerce Posterior Samples of walker Fit to a Data Frame*

Description

Creates a data.frame object from the output of walker fit.

Usage

```
## S3 method for class 'walker_fit'
as.data.frame(x, row.names = NULL, optional = FALSE, type, ...)
```

Arguments

x	An output from walker() or walker_glm().
row.names	NULL (default) or a character vector giving the row names for the data frame.
optional	Ignored (part of generic as.data.frame signature).
type	Either tiv (time-invariant parameters) or tv (time-varying coefficients).
...	Ignored.

Examples

```
## Not run:
 as.data.frame(fit, "tiv") %>%
 group_by(variable) %>%
 summarise(mean = mean(value),
           lwr = quantile(value, 0.05),
           upr = quantile(value, 0.95))

## End(Not run)
```

---

coef.walker_fit *Extract Coeffients of Walker Fit*

Description

Returns the time-varying regression coefficients from output of walker or walker_glm.

Usage

```
## S3 method for class 'walker_fit'
coef(object, summary = TRUE, transform = identity, ...)
```

Arguments

object	Output of walker or walker_glm.
summary	If TRUE (default), return summary statistics. Otherwise returns samples.
transform	Optional vectorized function for transforming the coefficients (for example exp).
...	Ignored.

Value

Time series containing coefficient values.

fitted.walker_fit *Extract Fitted Values of Walker Fit*

Description

Returns fitted values (posterior means) from output of `walker` or `walker_glm`.

Usage

```
## S3 method for class 'walker_fit'
fitted(object, summary = TRUE, ...)
```

Arguments

`object`	Output of `walker` or `walker_glm`.
`summary`	If `TRUE` (default), return summary statistics. Otherwise returns samples.
`...`	Ignored.

Value

If `summary=TRUE`, matrix containing summary statistics of fitted values. Otherwise a matrix of samples.

lfo *Leave-Future-Out Cross-Validation*

Description

Estimates the leave-future-out (LFO) information criterion for `walker` and `walker_glm` models.

Usage

```
lfo(object, L, exact = FALSE, verbose = TRUE, k_thres = 0.7)
```

Arguments

`object`	Output of `walker` or `walker_glm`.
`L`	Positive integer defining how many observations should be used for the initial fit.
`exact`	If `TRUE`, computes exact 1-step predictions by re-estimating the model repeatedly. If `FALSE` (default), uses approximate method based on Bürkner, Gabry and Vehtari (2020).
`verbose`	If `TRUE` (default), print the progress of the LFO computations to the console.
`k_thres`	Threshold for the pareto k estimate triggering refit. Default is 0.7.

Details

The LFO for non-Gaussian models is (currently) based on the corresponding Gaussian approximation and not the importance sampling corrected true posterior.

Value

List with components `ELPD` (Expected log predictive density), `ELPDs` (observation-specific ELPDs), `ks` (Pareto k values in case of approximation was used), and `refits` (time points where model was re-estimated)

References

Paul-Christian Bürkner, Jonah Gabry & Aki Vehtari (2020). Approximate leave-future-out cross-validation for Bayesian time series models, Journal of Statistical Computation and Simulation, 90:14, 2499-2523, DOI: 10.1080/00949655.2020.1783262.

Examples

```
## Not run:
fit <- walker(Nile ~ -1 +
  rw1(~ 1,
    beta = c(1000, 100),
    sigma = c(2, 0.001)),
  sigma_y_prior = c(2, 0.005),
  iter = 2000, chains = 1)

fit_lfo <- lfo(fit, L = 20, exact = FALSE)
fit_lfo$ELPD

## End(Not run)
```

---

plot_coefs *Posterior predictive check for walker object*

Description

Plots sample quantiles from posterior predictive sample. See `bayesplot::ppc_ribbon()` for details.

Usage

```
plot_coefs(
  object,
  level = 0.05,
  alpha = 0.33,
  transform = identity,
  scales = "fixed",
  add_zero = TRUE
)
```

Arguments

object	An output from walker().
level	Level for intervals. Default is 0.05, leading to 90% intervals.
alpha	Transparency level for ggplot2::geom_ribbon().
transform	Optional vectorized function for transforming the coefficients (for example exp).
scales	Should y-axis of the panels be "fixed" (default) or "free"?
add_zero	Logical, should a dashed line indicating a zero be included?

---

plot_fit *Plot the fitted values and sample quantiles for a walker object*

Description

Plot the fitted values and sample quantiles for a walker object

Usage

```
plot_fit(object, level = 0.05, alpha = 0.33, ...)
```

Arguments

object	An output from walker() or walker_glm().
level	Level for intervals. Default is 0.05, leading to 90% intervals.
alpha	Transparency level for ggplot2::geom_ribbon().
...	Further arguments to bayesplot::ppc_ribbon().

---

plot_predict *Prediction intervals for walker object*

Description

Plots sample quantiles and posterior means of the predictions of the predict.walker_fit output.

Usage

```
plot_predict(object, draw_obs = NULL, level = 0.05, alpha = 0.33)
```

Arguments

object	An output from predict.walker_fit().
draw_obs	Either "response", "mean", or "none", where "mean" is response variable divided by number of trials or exposures in case of binomial/poisson models.
level	Level for intervals. Default is 0.05, leading to 90% intervals.
alpha	Transparency level for ggplot2::geom_ribbon().

Examples

```
set.seed(1)
n <- 60
slope <- 0.0001 + cumsum(rnorm(n, 0, sd = 0.01))
beta <- numeric(n)
beta[1] <- 1
for(i in 2:n) beta[i] <- beta[i-1] + slope[i-1]
ts.plot(beta)
x <- rnorm(n, 1, 0.5)
alpha <- 2
ts.plot(beta * x)

signal <- alpha + beta * x
y <- rnorm(n, signal, 0.25)
ts.plot(cbind(signal, y), col = 1:2)
data_old <- data.frame(y = y[1:(n-10)], x = x[1:(n-10)])

# note very small number of iterations for the CRAN checks!
rw2_fit <- walker(y ~ 1 +
                    rw2(~ -1 + x,
                        beta = c(0, 10),
                        nu = c(0, 10)),
                  beta = c(0, 10), data = data_old,
                  iter = 300, chains = 1, init = 0, refresh = 0)

pred <- predict(rw2_fit, newdata = data.frame(x=x[(n-9):n]))
data_new <- data.frame(t = (n-9):n, y = y[(n-9):n])
plot_predict(pred) +
  ggplot2::geom_line(data = data_new, ggplot2:: aes(t, y),
  linetype = "dashed", colour = "red", inherit.aes = FALSE)
```

`pp_check.walker_fit` *Posterior predictive check for walker object*

Description

Plots sample quantiles from posterior predictive sample.

Usage

```
## S3 method for class 'walker_fit'
pp_check(object, ...)
```

Arguments

`object`	An output from `walker()`.
`...`	Further parameters to `bayesplot::ppc_ribbon()`.

Details

For other types of posterior predictive checks for example with `bayesplot`, you can extract the variable `yrep` from the output, see examples.

Examples

```
## Not run:
# Extracting the yrep variable for general use:
# extract yrep
y_rep <- extract(object$stanfit, pars = "y_rep", permuted = TRUE)$y_rep

# For non-gaussian model:
weights <- extract(object$stanfit,
pars = "weights", permuted = TRUE)$weights
y_rep <- y_rep[sample(1:nrow(y_rep),
  size = nrow(y_rep), replace = TRUE, prob = weights), , drop = FALSE]

## End(Not run)
```

`predict.walker_fit` *Predictions for walker object*

Description

Given the new covariate data and output from `walker`, obtain samples from posterior predictive distribution for future time points.

Usage

```
## S3 method for class 'walker_fit'
predict(
  object,
  newdata,
  u,
  type = ifelse(object$distribution == "gaussian", "response", "mean"),
  ...
)
```

Arguments

`object`	An output from `walker()` or `walker_glm()`.
`newdata`	A `data.frame` containing covariates used for prediction.
`u`	For Poisson model, a vector of future exposures i.e. E(y) = u*exp(x*beta). For binomial, a vector containing the number of trials for future time points. Defaults 1.

type	If "response" (default for Gaussian model), predictions are on the response level (e.g., number of successes for Binomial case, and for Gaussian case the observational level noise is added to the mean predictions). If "mean" (default for non-Gaussian case), predict means (e.g., success probabilities in Binomial case). If "link", predictions for non-Gaussian models are returned before applying the inverse of the link-function.
...	Ignored.

Value

A list containing samples from posterior predictive distribution.

See Also

plot_predict() for example.

`predict_counterfactual` *Predictions for walker object*

Description

Given the new covariate data and output from walker, obtain samples from posterior predictive distribution for counterfactual case, i.e. for past time points with different covariate values.

Usage

```
predict_counterfactual(
  object,
  newdata,
  u,
  summary = TRUE,
  type = ifelse(object$distribution == "gaussian", "response", "mean")
)
```

Arguments

object	An output from walker() or walker_glm().
newdata	A data.frame containing covariates used for prediction. Should have equal number of rows as the original data
u	For Poisson model, a vector of exposures i.e. E(y) = u*exp(x*beta). For binomial, a vector containing the number of trials. Defaults 1.
summary	If TRUE (default), return summary statistics. Otherwise returns samples.

type — If "response" (default for Gaussian model), predictions are on the response level (e.g., number of successes for Binomial case, and for Gaussian case the observational level noise is added to the mean predictions). If "mean" (default for non-Gaussian case), predict means (e.g., success probabilities in Binomial case). If "link", predictions for non-Gaussian models are returned before applying the inverse of the link-function.

Value

If summary=TRUE, time series containing summary statistics of predicted values. Otherwise a matrix of samples from predictive distribution.

Examples

```
## Not run:
set.seed(1)
n <- 50
x1 <- rnorm(n, 0, 1)
x2 <- rnorm(n, 1, 0.5)
x3 <- rnorm(n)
beta1 <- cumsum(c(1, rnorm(n - 1, sd = 0.1)))
beta2 <- cumsum(c(0, rnorm(n - 1, sd = 0.1)))
beta3 <- -1
u <- sample(1:10, size = n, replace = TRUE)
y <- rbinom(n, u, plogis(beta3 * x3 + beta1 * x1 + beta2 * x2))

d <- data.frame(y, x1, x2, x3)
out <- walker_glm(y ~ x3 + rw1(~ -1 + x1 + x2, beta = c(0, 2),
  sigma = c(2, 10)), distribution = "binomial", beta = c(0, 2),
  u = u, data = d,
  iter = 2000, chains = 1, refresh = 0)

# what if our covariates were constant?
newdata <- data.frame(x1 = rep(0.4, n), x2 = 1, x3 = -0.1)

fitted <- fitted(out)
pred <- predict_counterfactual(out, newdata, type = "mean")

ts.plot(cbind(fitted[, c(1, 3, 5)], pred[, c(1, 3, 5)]),
  col = rep(1:2, each = 3), lty = c(1, 2, 2))

## End(Not run)
```

---

print.walker_fit *Print Summary of walker_fit Object*

Description

Prints the summary information of time-invariant model parameters. In case of non-Gaussian models, results based on approximate model are returned with a warning.

Usage

```
## S3 method for class 'walker_fit'
print(x, ...)
```

Arguments

x	An output from walker() or walker_glm().
...	Additional arguments to rstan::print.stanfit().

---

rw1 *Construct a first-order random walk component*

Description

Auxiliary function used inside of the formula of walker.

Usage

```
rw1(formula, data, beta, sigma = c(2, 1e-04), gamma = NULL)
```

Arguments

formula	Formula for RW1 part of the model. Only right-hand-side is used.
data	Optional data.frame.
beta	A length vector of length two which defines the prior mean and standard deviation of the Gaussian prior for coefficients at time 1.
sigma	A vector of length two, defining the Gamma prior for the coefficient level standard deviation. First element corresponds to the shape parameter and second to the rate parameter. Default is Gamma(2, 0.0001).
gamma	An optional k times n matrix defining a known non-negative weights of the random walk noises, where k is the number of coefficients and n is the number of time points. Then, the standard deviation of the random walk noise for each coefficient is of form gamma_t * sigma (instead of just sigma).

rw2 *Construct a second-order random walk component*

Description

Auxiliary function used inside of the formula of `walker`.

Usage

```
rw2(formula, data, beta, sigma = c(2, 1e-04), nu, gamma = NULL)
```

Arguments

`formula`	Formula for RW2 part of the model. Only right-hand-side is used.
`data`	Optional data.frame.
`beta`	A vector of length two which defines the prior mean and standard deviation of the Gaussian prior for coefficients at time 1.
`sigma`	A vector of length two, defining the Gamma prior for the slope level standard deviation. First element corresponds to the shape parameter and second to the rate parameter. Default is Gamma(2, 0.0001).
`nu`	A vector of length two which defines the prior mean and standard deviation of the Gaussian prior for the slopes nu at time 1.
`gamma`	An optional k times n matrix defining a known non-negative weights of the slope noises, where k is the number of coefficients and n is the number of time points. Then, the standard deviation of the noise term for each coefficient's slope is of form gamma_t * sigma (instead of just sigma).

summary.walker_fit *Summary of walker_fit Object*

Description

Return summary information of time-invariant model parameters.

Usage

```
## S3 method for class 'walker_fit'
summary(object, type = "tiv", ...)
```

Arguments

`object`	An output from `walker()` or `walker_glm()`.
`type`	Either `tiv` (time-invariant parameters, the default) or `tv` (time-varying coefficients).
`...`	Ignored.

walker *Bayesian regression with random walk coefficients*

Description

Function walker performs Bayesian inference of a linear regression model with time-varying, random walk regression coefficients, i.e. ordinary regression model where instead of constant coefficients the coefficients follow first or second order random walks. All Markov chain Monte Carlo computations are done using Hamiltonian Monte Carlo provided by Stan, using a state space representation of the model in order to marginalise over the coefficients for efficient sampling.

Usage

```
walker(
  formula,
  data,
  sigma_y_prior = c(2, 0.01),
  beta,
  init,
  chains,
  return_x_reg = FALSE,
  gamma_y = NULL,
  return_data = TRUE,
  ...
)
```

Arguments

formula	An object of class {formula} with additional terms rw1 and/or rw2 e.g. y ~ x1 + rw1(~ -1 + x2). See details.
data	An optional data.frame or object coercible to such, as in {lm}.
sigma_y_prior	A vector of length two, defining the a Gamma prior for the observation level standard deviation with first element corresponding to the shape parameter and second to rate parameter. Default is Gamma(2, 0.0001). Not used in walker_glm.
beta	A length vector of length two which defines the prior mean and standard deviation of the Gaussian prior for time-invariant coefficients
init	Initial value specification, see rstan::sampling(). Note that compared to default in rstan, here the default is a to sample from the priors.
chains	Number of Markov chains. Default is 4.
return_x_reg	If TRUE, does not perform sampling, but instead returns the matrix of predictors after processing the formula.
gamma_y	An optional vector defining known non-negative weights for the standard deviation of the observational level noise at each time point. More specifically, the observational level standard deviation sigma_t is defined as $\sigma_t = gamma_t * \sigma_y$ (in default case $\sigma_t = sigma_y$)

return_data	if TRUE, returns data input to rstan::sampling(). This is needed for lfo.
...	Further arguments to rstan::sampling().

Details

The rw1 and rw2 functions used in the formula define new formulas for the first and second order random walks. In addition, these functions need to be supplied with priors for initial coefficients and the standard deviations. For second order random walk model, these sigma priors correspond to the standard deviation of slope disturbances. For rw2, also a prior for the initial slope nu needs to be defined. See examples.

Value

A list containing the stanfit object, observations y, and covariates xreg and xreg_new.

Note

Beware of overfitting and identifiability issues. In particular, be careful in not defining multiple intercept terms (only one should be present). By default rw1 and rw2 calls add their own time-varying intercepts, so you should use 0 or -1 to remove some of them (or the time-invariant intercept in the fixed-part of the formula).

See Also

walker_glm() for non-Gaussian models.

Examples

```
## Not run:
set.seed(1)
x <- rnorm(10)
y <- x + rnorm(10)

# different intercept definitions:

# both fixed intercept and time-varying level,
# can be unidentifiable without strong priors:
fit1 <- walker(y ~ rw1(~ x, beta = c(0, 1)),
  beta = c(0, 1), chains = 1, iter = 1000, init = 0)

# only time-varying level, using 0 or -1 removes intercept:
fit2 <- walker(y ~ 0 + rw1(~ x, beta = c(0, 1)), chains = 1, iter = 1000,
  init = 0)

# time-varying level, no covariates:
fit3 <- walker(y ~ 0 + rw1(~ 1, beta = c(0, 1)), chains = 1, iter = 1000)

# fixed intercept no time-varying level:
fit4 <- walker(y ~ rw1(~ 0 + x, beta = c(0, 1)),
  beta = c(0, 1), chains = 1, iter = 1000)
```

```
# only time-varying effect of x:
fit5 <- walker(y ~ 0 + rw1(~ 0 + x, beta = c(0, 1)), chains = 1, iter = 1000)

## End(Not run)

## Not run:

rw1_fit <- walker(Nile ~ -1 +
  rw1(~ 1,
    beta = c(1000, 100),
    sigma = c(2, 0.001)),
  sigma_y_prior = c(2, 0.005),
  iter = 2000, chains = 1)

rw2_fit <- walker(Nile ~ -1 +
  rw2(~ 1,
    beta = c(1000, 100),
    sigma = c(2, 0.001),
    nu = c(0, 100)),
  sigma_y_prior = c(2, 0.005),
  iter = 2000, chains = 1)

g_y <- geom_point(data = data.frame(y = Nile, x = time(Nile)),
  aes(x, y, alpha = 0.5), inherit.aes = FALSE)
g_rw1 <- plot_coefs(rw1_fit) + g_y
g_rw2 <- plot_coefs(rw2_fit) + g_y
if(require("gridExtra")) {
  grid.arrange(g_rw1, g_rw2, ncol=2, top = "RW1 (left) versus RW2 (right)")
} else {
  g_rw1
  g_rw2
}

y <- window(log10(UKgas), end = time(UKgas)[100])
n <- 100
cos_t <- cos(2 * pi * 1:n / 4)
sin_t <- sin(2 * pi * 1:n / 4)
dat <- data.frame(y, cos_t, sin_t)
fit <- walker(y ~ -1 +
  rw1(~ cos_t + sin_t, beta = c(0, 10), sigma = c(2, 1)),
  sigma_y_prior = c(2, 10), data = dat, chains = 1, iter = 2000)
print(fit$stanfit, pars = c("sigma_y", "sigma_rw1"))

plot_coefs(fit)
# posterior predictive check:
pp_check(fit)

newdata <- data.frame(
  cos_t = cos(2 * pi * 101:108 / 4),
  sin_t = sin(2 * pi * 101:108 / 4))
pred <- predict(fit, newdata)
plot_predict(pred)
```

```
# example on scalability
set.seed(1)
n <- 2^12
beta1 <- cumsum(c(0.5, rnorm(n - 1, 0, sd = 0.05)))
beta2 <- cumsum(c(-1, rnorm(n - 1, 0, sd = 0.15)))
x1 <- rnorm(n, mean = 2)
x2 <- cos(1:n)
rw <- cumsum(rnorm(n, 0, 0.5))
signal <- rw + beta1 * x1 + beta2 * x2
y <- rnorm(n, signal, 0.5)

d <- data.frame(y, x1, x2)

n <- 2^(6:12)
times <- numeric(length(n))
for(i in seq_along(n)) {
  times[i] <- sum(get_elapsed_time(
    walker(y ~ 0 + rw1(~ x1 + x2,
      beta = c(0, 10)),
      data = d[1:n[i],],
      chains = 1, seed = 1, refresh = 0)$stanfit))
}
plot(log2(n), log2(times))

## End(Not run)
```

---

walker_glm *Bayesian generalized linear model with time-varying coefficients*

Description

Function walker_glm is a generalization of walker for non-Gaussian models. Compared to walker, the returned samples are based on Gaussian approximation, which can then be used for exact-approximate analysis by weighting the sample properly. These weights are also returned as a part of the stanfit (they are generated in the generated quantities block of Stan model). Note that plotting functions pp_check, plot_coefs, and plot_predict resample the posterior based on weights before plotting, leading to "exact" analysis.

Usage

```
walker_glm(
  formula,
  data,
  beta,
  init,
  chains,
  return_x_reg = FALSE,
  distribution,
  initial_mode = "kfas",
```

```
  u,
  mc_sim = 50,
  return_data = TRUE,
  ...
)
```

Arguments

formula	An object of class {formula} with additional terms rw1 and/or rw2 e.g. y ~ x1 + rw1(~ -1 + x2). See details.
data	An optional data.frame or object coercible to such, as in {lm}.
beta	A length vector of length two which defines the prior mean and standard deviation of the Gaussian prior for time-invariant coefficients
init	Initial value specification, see rstan::sampling(). Note that compared to default in rstan, here the default is a to sample from the priors.
chains	Number of Markov chains. Default is 4.
return_x_reg	If TRUE, does not perform sampling, but instead returns the matrix of predictors after processing the formula.
distribution	Either "poisson" or "binomial".
initial_mode	The initial guess of the fitted values on log-scale. Defines the Gaussian approximation used in the MCMC. Either "obs" (corresponds to log(y+0.1) in Poisson case), "glm" (mode is obtained from time-invariant GLM), "mle" (default; mode is obtained from maximum likelihood estimate of the model), or numeric vector (custom guess).
u	For Poisson model, a vector of exposures i.e. $E(y) = u * exp(x * beta)$. For binomial, a vector containing the number of trials. Defaults 1.
mc_sim	Number of samples used in importance sampling. Default is 50.
return_data	if TRUE, returns data input to rstan::sampling(). This is needed for lfo.
...	Further arguments to rstan::sampling().

Details

The underlying idea of walker_glm is based on Vihola, Helske, Franks (2020).

walker_glm uses the global approximation (i.e. start of the MCMC) instead of more accurate but slower local approximation (where model is approximated at each iteration). However for these restricted models global approximation should be sufficient, assuming the the initial estimate of the conditional mode of p(xbeta | y, sigma) not too far away from the true posterior. Therefore by default walker_glm first finds the maximum likelihood estimates of the standard deviation parameters (using KFAS::KFAS()) package, and constructs the approximation at that point, before running the Bayesian analysis.

Value

A list containing the stanfit object, observations y, covariates xreg_fixed, and xreg_rw.

References

Vihola, M, Helske, J, Franks, J. (2020). Importance sampling type estimators based on approximate marginal Markov chain Monte Carlo. Scandinavian Journal of Statistics. 47: 1339–1376. doi:10.1111/sjos.12492

See Also

Package `diagis` in CRAN, which provides functions for computing weighted summary statistics.

Examples

```
set.seed(1)
n <- 25
x <- rnorm(n, 1, 1)
beta <- cumsum(c(1, rnorm(n - 1, sd = 0.1)))

level <- -1
u <- sample(1:10, size = n, replace = TRUE)
y <- rpois(n, u * exp(level + beta * x))
ts.plot(y)

# note very small number of iterations for the CRAN checks!
out <- walker_glm(y ~ -1 + rw1(~ x, beta = c(0, 10),
  sigma = c(2, 10)), distribution = "poisson",
  iter = 200, chains = 1, refresh = 0)
print(out$stanfit, pars = "sigma_rw1") ## approximate results
if (require("diagis")) {
  weighted_mean(extract(out$stanfit, pars = "sigma_rw1")$sigma_rw1,
    extract(out$stanfit, pars = "weights")$weights)
}
plot_coefs(out)
pp_check(out)

## Not run:
data("discoveries", package = "datasets")
out <- walker_glm(discoveries ~ -1 +
  rw2(~ 1, beta = c(0, 10), sigma = c(2, 10), nu = c(0, 2)),
  distribution = "poisson", iter = 2000, chains = 1, refresh = 0)

plot_fit(out)

# Dummy covariate example

fit <- walker_glm(VanKilled ~ -1 +
  rw1(~ law, beta = c(0, 1), sigma = c(2, 10)), dist = "poisson",
   data = as.data.frame(Seatbelts), chains = 1, refresh = 0)

# compute effect * law
d <- coef(fit, transform = function(x) {
  x[, 2, 1:170] <- 0
  x
})
```

```
require("ggplot2")
d %>% ggplot(aes(time, mean)) +
  geom_ribbon(aes(ymin = `2.5%`, ymax = `97.5%`), fill = "grey90") +
  geom_line() + facet_wrap(~ beta, scales = "free") + theme_bw()

## End(Not run)
```

`walker_rw1` *Comparison of naive and state space implementation of RW1 regression model*

Description

This function is the first iteration of the function `walker`, which supports only time-varying model where all coefficients ~ rw1. This is kept as part of the package in order to compare "naive" and state space versions of the model in the vignette, but there is little reason to use it for other purposes.

Usage

```
walker_rw1(
  formula,
  data,
  beta,
  sigma,
  init,
  chains,
  naive = FALSE,
  return_x_reg = FALSE,
  ...
)
```

Arguments

`formula`	An object of class `stats::formula()`. See `lm()` for details.
`data`	An optional data.frame or object coercible to such, as in `lm()`.
`beta`	A matrix with k rows and 2 columns, where first columns defines the prior means of the Gaussian priors of the corresponding k regression coefficients, and the second column defines the the standard deviations of those prior distributions.
`sigma`	A matrix with $k + 1$ rows and two colums with similar structure as `beta`, with first row corresponding to the prior of the standard deviation of the observation level noise, and rest of the rows define the priors for the standard deviations of random walk noise terms. The prior distributions for all sigmas are Gaussians truncated to positive real axis. For non-Gaussian models, this should contain only k rows. For second order random walk model, these priors correspond to the slope level standard deviations.

init	Initial value specification, see rstan::sampling().
chains	Number of Markov chains. Default is 4.
naive	Only used for walker function. If TRUE, use "standard" approach which samples the joint posterior $p(beta, sigma|y)$. If FALSE (the default), use marginalisation approach where we sample the marginal posterior $p(sigma|y)$ and generate the samples of $p(beta|sigma, y)$ using state space modelling techniques (namely simulation smoother by Durbin and Koopman (2002)). Both methods give asymptotically identical results, but the latter approach is computationally much more efficient.
return_x_reg	If TRUE, does not perform sampling, but instead returns the matrix of predictors after processing the formula.
...	Additional arguments to rstan::sampling().

Examples

```
## Not run:
## Comparing the approaches, note that with such a small data
## the differences aren't huge, but try same with n = 500 and/or more terms...
set.seed(123)
n <- 100
beta1 <- cumsum(c(0.5, rnorm(n - 1, 0, sd = 0.05)))
beta2 <- cumsum(c(-1, rnorm(n - 1, 0, sd = 0.15)))
x1 <- rnorm(n, 1)
x2 <- 0.25 * cos(1:n)
ts.plot(cbind(beta1 * x1, beta2 *x2), col = 1:2)
u <- cumsum(rnorm(n))
y <- rnorm(n, u + beta1 * x1 + beta2 * x2)
ts.plot(y)
lines(u + beta1 * x1 + beta2 * x2, col = 2)
kalman_walker <- walker_rw1(y ~ -1 +
  rw1(~ x1 + x2, beta = c(0, 2), sigma = c(0, 2)),
  sigma_y = c(0, 2), iter = 2000, chains = 1)
print(kalman_walker$stanfit, pars = c("sigma_y", "sigma_rw1"))
betas <- extract(kalman_walker$stanfit, "beta")[[1]]
ts.plot(cbind(u, beta1, beta2, apply(betas, 2, colMeans)),
  col = 1:3, lty = rep(2:1, each = 3))
sum(get_elapsed_time(kalman_walker$stanfit))
naive_walker <- walker_rw1(y ~ x1 + x2, iter = 2000, chains = 1,
  beta = cbind(0, rep(2, 3)), sigma = cbind(0, rep(2, 4)),
  naive = TRUE)
print(naive_walker$stanfit, pars = c("sigma_y", "sigma_b"))
sum(get_elapsed_time(naive_walker$stanfit))

## Larger problem, this takes some time with naive approach

set.seed(123)
n <- 500
beta1 <- cumsum(c(1.5, rnorm(n - 1, 0, sd = 0.05)))
beta2 <- cumsum(c(-1, rnorm(n - 1, 0, sd = 0.5)))
beta3 <- cumsum(c(-1.5, rnorm(n - 1, 0, sd = 0.15)))
beta4 <- 2
```

```
x1 <- rnorm(n, 1)
x2 <- 0.25 * cos(1:n)
x3 <- runif(n, 1, 3)
ts.plot(cbind(beta1 * x1, beta2 * x2, beta3 * x3), col = 1:3)
a <- cumsum(rnorm(n))
signal <- a + beta1 * x1 + beta2 * x2 + beta3 * x3
y <- rnorm(n, signal)
ts.plot(y)
lines(signal, col = 2)
kalman_walker <- walker_rw1(y ~ x1 + x2 + x3, iter = 2000, chains = 1,
  beta = cbind(0, rep(2, 4)), sigma = cbind(0, rep(2, 5)))
print(kalman_walker$stanfit, pars = c("sigma_y", "sigma_b"))
betas <- extract(kalman_walker$stanfit, "beta")[[1]]
ts.plot(cbind(u, beta1, beta2, beta3, apply(betas, 2, colMeans)),
  col = 1:4, lty = rep(2:1, each = 4))
sum(get_elapsed_time(kalman_walker$stanfit))
# need to increase adapt_delta in order to get rid of divergences
# and max_treedepth to get rid of related warnings
# and still we end up with low BFMI warning after hours of computation
naive_walker <- walker_rw1(y ~ x1 + x2 + x3, iter = 2000, chains = 1,
  beta = cbind(0, rep(2, 4)), sigma = cbind(0, rep(2, 5)),
  naive = TRUE, control = list(adapt_delta = 0.9, max_treedepth = 15))
print(naive_walker$stanfit, pars = c("sigma_y", "sigma_b"))
sum(get_elapsed_time(naive_walker$stanfit))

## End(Not run)
```

Index