

# Package ‘walker’

January 23, 2020

**Type** Package

**Title** Bayesian Regression with Time-Varying Coefficients

**Version** 0.3.1-1

**Date** 2020-01-23

**Description** Bayesian dynamic regression models where the regression coefficients can vary over time as random walks. Gaussian, Poisson, and binomial observations are supported. The 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. For non-Gaussian models, walker uses the importance sampling type estimators based on approximate marginal MCMC as in Vihola, Helske, Franks (2017, <arXiv:1609.02541>).

**License** GPL (>= 3)

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

**Depends** R (>= 3.4.0), Rcpp (>= 0.12.9), bayesplot, rstan (>= 2.18.1)

**Imports** dplyr, ggplot2, KFAS, methods

**LinkingTo** StanHeaders (>= 2.18.0), rstan (>= 2.18.1), BH (>= 1.66.0), Rcpp (>= 0.12.9), RcppArmadillo, RcppEigen (>= 0.3.3.3.0)

**SystemRequirements** C++14, GNU make

**VignetteBuilder** knitr

**RoxygenNote** 6.1.1

**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] (<<https://orcid.org/0000-0001-7130-793X>>)

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

**Repository** CRAN

**Date/Publication** 2020-01-23 15:50:02 UTC

## R topics documented:

coef.walker_fit . . . . .	2
fitted.walker_fit . . . . .	3
plot_coefs . . . . .	3
plot_fit . . . . .	4
plot_predict . . . . .	4
pp_check.walker_fit . . . . .	5
predict.walker_fit . . . . .	5
print.walker_fit . . . . .	6
rw1 . . . . .	7
rw2 . . . . .	7
walker . . . . .	8
walker_glm . . . . .	10
walker_rw1 . . . . .	12

<b>Index</b>	<b>15</b>
--------------	-----------

---

coef.walker_fit	<i>Extract Coeffients of Walker Fit</i>
-----------------	---

---

### Description

Returns the regression coeffients 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 coeffients values.

---

fitted.walker_fit	<i>Extract Fitted Values of Walker Fit</i>
-------------------	--

---

**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**

Time series containing fitted values.

---

plot_coefs	<i>Posterior predictive check for walker object</i>
------------	---

---

**Description**

Plots sample quantiles from posterior predictive sample. See [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 <a href="#">walker</a> .
level	Level for intervals. Default is 0.05, leading to 90% intervals.
alpha	Transparency level for 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	<i>Plot the fitted values and sample quantiles for a walker object</i>
----------	--

---

**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 <a href="#">walker</a> or <a href="#">walker_glm</a> .
level	Level for intervals. Default is 0.05, leading to 90% intervals.
alpha	Transparency level for <a href="#">geom_ribbon</a> .
...	Further arguments to <a href="#">ppc_ribbon</a> .

---

plot_predict	<i>Prediction intervals for walker object</i>
--------------	---

---

**Description**

Plots sample quantiles from pre See [ppc\\_ribbon](#) for details.

**Usage**

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

**Arguments**

object	An output from <a href="#">predict.walker_fit</a> .
level	Level for intervals. Default is 0.05, leading to 90% intervals.
alpha	Transparency level for <a href="#">geom_ribbon</a> .

**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)])
rw2_fit <- walker(y ~ 1 +
  rw2(~ -1 + x,
    beta_prior = c(0, 10),
    sigma_prior = c(0, 10),
    slope_prior = c(0, 10)),
  sigma_y_prior = c(0, 10),
  beta_prior = c(0, 10),
  iter = 400, chains = 1, data = data_old)

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) +
  geom_line(data=data_new, 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. See [ppc\\_ribbon](#) for details.

## Usage

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

## Arguments

object	An output from <a href="#">walker</a> .
...	Further parameters to <a href="#">ppc_ribbon</a> .

---

predict.walker\_fit    *Predictions for walker object*

---

## Description

Given the new covariate data and output from walker, obtain samples from posterior predictive distribution.

**Usage**

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

**Arguments**

object	An output from <a href="#">walker</a> or <a href="#">walker_glm</a> .
newdata	A data.frame containing covariates used for prediction.
u	For Poisson model, a vector of future exposures i.e. $E(y) = u \cdot \exp(x \cdot \beta)$ . 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., 0/1 for Bernoulli 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).
...	Ignored.

**Value**

A list containing samples from posterior predictive distribution.

**See Also**

[plot\\_predict](#) for example.

---

print.walker\_fit

*Print Summary of walker\_fit Object*

---

**Description**

Prints the summary information of time-invariant model parameters.

**Usage**

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

**Arguments**

x	An output from <a href="#">walker</a> or <a href="#">walker_glm</a> .
...	Additional arguments to <a href="#">print.stanfit</a> .

---

rw1	<i>Construct a first-order random walk component</i>
-----	--

---

**Description**

Auxiliary function used inside of the formula of walker.

**Usage**

```
rw1(formula, data, beta_prior, sigma_prior, gamma = NULL)
```

**Arguments**

formula	Formula for RW1 part of the model. Only right-hand-side is used.
data	Optional data.frame.
beta_prior	A length vector of length two which defines the prior mean and standard deviation of the Gaussian prior for coefficients at time 1.
sigma_prior	A vector of length two, defining the truncated Gaussian prior for the coefficient level standard deviation.
gamma	An optional vector defining a damping of the random walk noises. More specifically, the variance of the conditional distribution of state <sub>t+1</sub> given state is of form $\gamma_t * \sigma$ .

---

rw2	<i>Construct a second-order random walk component</i>
-----	---

---

**Description**

Auxiliary function used inside of the formula of walker.

**Usage**

```
rw2(formula, data, beta_prior, sigma_prior, slope_prior, gamma = NULL)
```

**Arguments**

formula	Formula for RW2 part of the model. Only right-hand-side is used.
data	Optional data.frame.
beta_prior	A vector of length two which defines the prior mean and standard deviation of the Gaussian prior for coefficients at time 1.
sigma_prior	A vector of length two, defining the truncated Gaussian prior for the slope level standard deviation.

slope_prior	A vector of length two which defines the prior mean and standard deviation of the Gaussian prior for the slopes at time 1.
gamma	An optional vector defining a damping of the slope level noises. More specifically, the variance of the conditional distribution of state <sub>t+1</sub> given state is of form $\gamma_t * \sigma$ .

---

 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, beta_prior, init, chains,
        return_x_reg = FALSE, gamma_y = NULL, ...)
```

### Arguments

formula	An object of class <code>{formula}</code> with additional terms <code>rw1</code> and/or <code>rw2</code> e.g. $y \sim x1 + rw1(\sim -1 + x2)$ . See details.
data	An optional <code>data.frame</code> or object coercible to such, as in <code>{lm}</code> .
sigma_y_prior	A vector of length two, defining the truncated Gaussian prior for the observation level standard deviation. Not used in <code>walker_glm</code> .
beta_prior	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 <a href="#">sampling</a> . Note that compared to default in <code>rstan</code> , 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 a damping of the observational level noise. More specifically, $\sigma_t = \gamma_t * \sigma_y$ .
...	Further arguments to <a href="#">sampling</a> .

### 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 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).

**See Also**

[walker\\_glm](#) for non-Gaussian models.

**Examples**

```
## note very low number of iterations for the CRAN checks

rw1_fit <- walker(Nile ~ -1 +
  rw1(~ 1,
    beta_prior = c(1000, 100),
    sigma_prior = c(0, 100)),
  sigma_y_prior = c(0, 100),
  iter = 200, chains = 1)

rw2_fit <- walker(Nile ~ -1 +
  rw2(~ 1,
    beta_prior = c(1000, 100),
    sigma_prior = c(0, 100),
    slope_prior = c(0, 100)),
  sigma_y_prior = c(0, 100),
  iter = 200, 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
}

## Not run:
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_prior = c(0, 10), sigma_prior = c(0, 2)),
  sigma_y_prior = c(0, 10), data = dat, chains = 1, iter = 500)
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)

## End(Not run)

```

---

walker\_glm

*Bayesian generalized linear regression 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_prior, init, chains, return_x_reg = FALSE,
  distribution, initial_mode = "kfas", u, mc_sim = 50, ...)

```

## Arguments

<code>formula</code>	An object of class <code>{formula}</code> with additional terms <code>rw1</code> and/or <code>rw2</code> e.g. <code>y ~ x1 + rw1 (~ -1 + x2)</code> . See details.
<code>data</code>	An optional <code>data.frame</code> or object coercible to such, as in <code>{lm}</code> .
<code>beta_prior</code>	A length vector of length two which defines the prior mean and standard deviation of the Gaussian prior for time-invariant coefficients
<code>init</code>	Initial value specification, see <a href="#">sampling</a> . Note that compared to default in <code>rstan</code> , here the default is a to sample from the priors.
<code>chains</code>	Number of Markov chains. Default is 4.
<code>return_x_reg</code>	If TRUE, does not perform sampling, but instead returns the matrix of predictors after processing the formula.
<code>distribution</code>	Either "poisson" or "binomial".

<code>initial_mode</code>	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).
<code>u</code>	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.
<code>mc_sim</code>	Number of samples used in importance sampling. Default is 50.
<code>...</code>	Further arguments to <a href="#">sampling</a> .

### Details

The underlying idea of `walker_glm` is based on Vihola M, Helske J and Franks J (2016), "Importance sampling type correction of Markov chain Monte Carlo and exact approximations", which is available at ArXiv.

This function is not fully tested yet, so please file an issue and/or pull request on Github if you encounter problems. The reason there might be problems in some cases is the use of 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  $p(x|\beta, y, \sigma)$  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](#)) 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`.

### See Also

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

### Examples

```
## Not run:

data("discoveries", package = "datasets")
out <- walker_glm(discoveries ~ -1 +
  rw2(~ 1, beta_prior = c(0, 10), sigma_prior = c(0, 2), slope_prior = c(0, 2)),
  distribution = "poisson", iter = 1000, chains = 1, refresh = 0)

plot_fit(out)

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)

out <- walker_glm(y ~ -1 + rw1(~ x, beta_prior = c(0, 10),
  sigma_prior = c(0, 10)), distribution = "poisson",
  iter = 1000, 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)

## 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  $\sim$  `rw1`. This is kept as part of the package in order to compare "naive" and state space versions of the model in the vignette.

## Usage

```
walker_rw1(formula, data, beta_prior, sigma_prior, init, chains,
  naive = FALSE, return_x_reg = FALSE, ...)
```

## Arguments

<code>formula</code>	An object of class <code>formula</code> . See <code>lm</code> for details.
<code>data</code>	An optional <code>data.frame</code> or object coercible to such, as in <code>lm</code> .
<code>beta_prior</code>	A matrix with $k$ rows and 2 columns, where first column defines the prior means of the Gaussian priors of the corresponding $k$ regression coefficients, and the second column defines the standard deviations of those prior distributions.
<code>sigma_prior</code>	A matrix with $k + 1$ rows and two columns with similar structure as <code>beta_prior</code> , 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 <a href="#">sampling</a> .
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 <a href="#">sampling</a> .

## 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_prior = c(0, 2), sigma_prior = c(0, 2)),
  sigma_y_prior = 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$stanfit))
naive_walker <- walker_rw1(y ~ x1 + x2, iter = 2000, chains = 1,
  beta_prior = cbind(0, rep(2, 3)), sigma_prior = cbind(0, rep(2, 4)),
  naive = TRUE)
print(naive_walker$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_prior = cbind(0, rep(2, 4)), sigma_prior = 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$stanfit))
# need to increase adapt_delta in order to get rid of divergences
# and max_treedepth to get rid of related warnings
# and still we end up with low BFMI warning after hours of computation
naive_walker <- walker_rw1(y ~ x1 + x2 + x3, iter = 2000, chains = 1,
  beta_prior = cbind(0, rep(2, 4)), sigma_prior = cbind(0, rep(2, 5)),
  naive = TRUE, control = list(adapt_delta = 0.9, max_treedepth = 15))
print(naive_walker$stanfit, pars = c("sigma_y", "sigma_b"))
sum(get_elapsed_time(naive_walker$stanfit))

## End(Not run)

```

# Index

`coef.walker_fit`, 2  
`fitted.walker_fit`, 3  
`formula`, 12  
`geom_ribbon`, 4  
KFAS, 11  
`lm`, 12  
`plot_coefs`, 3  
`plot_fit`, 4  
`plot_predict`, 4, 6  
`pp_check.walker_fit`, 5  
`ppc_ribbon`, 3–5  
`predict.walker_fit`, 4, 5  
`print.stanfit`, 6  
`print.walker_fit`, 6  
`rw1`, 7  
`rw2`, 7  
sampling, 8, 10, 11, 13  
walker, 3–6, 8  
`walker_glm`, 4, 6, 9, 10  
`walker_rw1`, 12