

# Package ‘RcppSMC’

May 7, 2026

**Type** Package

**Title** Rcpp Bindings for Sequential Monte Carlo

**Version** 0.2.9

**Date** 2025-09-09

**Description** R access to the Sequential Monte Carlo Template Classes by Johansen <[doi:10.18637/jss.v030.i06](https://doi.org/10.18637/jss.v030.i06)> is provided. At present, four additional examples have been added, and the first example from the JSS paper has been extended. Further integration and extensions are planned.

**License** GPL (>= 2)

**LazyLoad** yes

**Imports** Rcpp (>= 0.11.0), methods, FKF

**Suggests** pkgKitten (>= 0.2.2)

**LinkingTo** Rcpp, RcppArmadillo

**LazyData** true

**URL** <https://github.com/rcppsmc/rcppsmc>,  
<https://dirk.eddelbuettel.com/code/rcpp.smc.html>

**BugReports** <https://github.com/rcppsmc/rcppsmc/issues>

**NeedsCompilation** yes

**Author** Dirk Eddelbuettel [aut, cre] (ORCID:  
<<https://orcid.org/0000-0001-6419-907X>>),  
Adam M. Johansen [aut] (ORCID: <<https://orcid.org/0000-0002-3531-7628>>),  
Leah F. South [aut] (ORCID: <<https://orcid.org/0000-0002-5646-2963>>),  
Ilya Zarubin [aut]

**Maintainer** Dirk Eddelbuettel <edd@debian.org>

**Repository** CRAN

**Date/Publication** 2025-09-09 12:50:12 UTC

## Contents

blockpfGaussianOpt . . . . .	2
compareNCestimates . . . . .	3
LinReg . . . . .	6
nonLinPMMH . . . . .	8
pfLinearBS . . . . .	10
pfNonlinBS . . . . .	11
radiata . . . . .	12
RcppSMC.package.skeleton . . . . .	13
simNonlin . . . . .	14
<b>Index</b>	<b>16</b>

---

blockpfGaussianOpt	<i>Block Sampling Particle Filter (Linear Gaussian Model; Optimal Proposal)</i>
--------------------	---

---

### Description

The `blockpfGaussianOpt` function provides a simple example for **RcppSMC**. It is based on a block sampling particle filter for a linear Gaussian model. This is intended only to illustrate the potential of block sampling; one would not ordinarily use a particle filter for a model in which analytic solutions are available. The 'optimal' block sampler in the sense of Doucet, Briers and Senecal (2006) can be implemented in this case.

The `simGaussian` function simulates data from the associated linear Gaussian state space model.

### Usage

```
blockpfGaussianOpt(data, particles=1000, lag=5, plot=FALSE)
simGaussian(len)
```

### Arguments

<code>data</code>	A vector variable containing the sequence of observations.
<code>particles</code>	An integer specifying the number of particles.
<code>lag</code>	An integer specifying the length of block to use.
<code>plot</code>	A boolean variable describing whether plot should illustrate the estimated path along with the uncertainty.
<code>len</code>	The length of the data sequence to simulate.

## Details

The `blockpfGaussianOpt` function provides a simple example for **RcppSMC**. It is based on a simple linear Gaussian state space model in which the state evolution and observation equations are:  $x(n) = x(n-1) + e(n)$  and  $y(n) = x(n) + f(n)$  where  $e(n)$  and  $f(n)$  are mutually-independent standard normal random variables. The 'optimal' block-sampling proposal described by Doucet et al (2006) is employed.

The `simGaussian` function simulates from the same model returning both the state and observation vectors.

## Value

The `blockpfGaussianOpt` function returns a matrix containing the final sample paths and a vector containing their weights. The logarithm of the estimated ratio of normalising constants between the final and initial distributions is also returned.

The `simGaussian` function returns a list containing the state and data sequences.

## Author(s)

Adam M. Johansen and Dirk Eddelbuettel

## References

A. Doucet, M. Briers, and S. Senecal. Efficient Block Sampling Strategies for sequential Monte Carlo methods. *Journal of Computational and Graphical Statistics*, 15(3):693-711, 2006.

## Examples

```
sim <- simGaussian(len=250)
res <- blockpfGaussianOpt(sim$data, lag=5, plot=TRUE)
```

## Description

The `compareNCestimates` function generates a Monte Carlo study to compare log-likelihood (normalizing constant) estimates in the standard linear Gaussian state space (LGSS) model: Kalman filter estimates, as the benchmark, are compared to the standard bootstrap particle filter and the conditional bootstrap particle filter estimates (see `Details`).

The `simGaussianSSM` function simulates data from a LGSS model (can be used manually to simulate data or runs as a default, if no data is provided, with a default parameter setup, see `parameters`).

The `kalmanFFBS` function runs a Kalman (exact) forward filter, computes a log-likelihood estimate and generates a joint smoothing state trajectory via a backward simulation pass.

**Usage**

```

compareNCestimates(dataY,
                   trueStates = NULL,
                   numParticles = 1000L,
                   simNumber = 100L,
                   modelParameters = list(stateInit = 0,
                                         phi = 0.7,
                                         varStateEvol = 1,
                                         varObs = 1),
                   plot = FALSE)
simGaussianSSM(len = 100,
               stateInit = 0,
               phi = 0.7,
               varStateEvol = 1,
               varObs = 1)
kalmanFFBS(dataY,
            stateInit,
            phi,
            varStateEvol,
            varObs,
            simNumber)

```

**Arguments**

<code>dataY</code>	A one-column matrix or dataframe or vector containing measurements (y values) from a standard linear Gaussian SSM. If not provided, defaults to a LGSS model with time series length <code>len=250</code> and parameter setup specified with default values in the <code>parameters</code> argument, see <a href="#">simGaussianSSM</a> or <a href="#">compareNCestimates</a> .
<code>trueStates</code>	defaults to <code>NULL</code> for a real dataset as the true state values are not observed; for simulated data, these can be passed and then will also be plotted if <code>plot=TRUE</code>
<code>numParticles</code>	An integer specifying the number of particles.
<code>simNumber</code>	An integer specifying the number of repeated simulation runs of each of which produces $2 \times 4 = 8$ normalizing constant estimates: BPF and conditional BPF estimates under four conditional resampling schemes, as well as a ground truth Kalman forward filter estimate and a backward filter output required for the reference trajectory of the conditional sampler.
<code>modelParameters</code>	<p>a named list of parameters of the LGSSM model in the following order:</p> <ul style="list-style-type: none"> <li>• <code>phi</code>: autoregressive parameter</li> <li>• <code>stateInit</code>: initial state value (i.e. <math>X_0</math>)</li> <li>• <code>varStateEvol</code>: state process variance</li> <li>• <code>varObs</code>: measurement/observation process variance</li> </ul> <p>These parameters are used to for the Kalman forward filtering and backward simulation pass, and, if no data argument is provided, to simulate data from a linear Gaussian state space model internally via <code>simGaussianSSM</code>.</p>
<code>phi</code>	autoregressive parameter

stateInit	initial state value (i.e. $X_0$ )
varStateEvol	state process variance
varObs	measurement/observation process variance
plot	A boolean variable describing whether plot should illustrate the estimated results along with the data.
len	Length of data series to simulate.

## Details

compareNCestimates runs a simulation study that provides log-likelihood (normalizing constant) estimates; there are simNumber runs of the standard BPF and the conditional BPF under four resampling schemes:

- multinomial
- stratified
- systematic
- residual

The "ground truth" Kalman forward filter estimate of the normalizing constant is compared to the BPF normalizing constant estimates, which are unbiased for all above schemes; specifically the conditional BPF estimate is unbiased if the reference trajectory is simulated from the target distribution which is obtained here as a backward simulation run of the Kalman filter.

Box plots illustrate the unbiasedness of standard BPF and conditional BPF estimates for the normalizing constant estimate in the linear Gaussian SSM, and serve as an small example for to illustrate conditional SMC algorithms (in their most basic BPF version) with different conditional resampling schemes as implemented within **RcppSMC**.

simGaussianSSM simulates from a Linear Gaussian state space model of the following form:

$$x_t = \phi x_{t-1} + u_t$$

$$y_t = x_t + w_t$$

where  $\phi$  is set via the phi argument,  $u_t \sim N(0, \sigma_x^2)$ ,  $w_t \sim N(0, \sigma_y^2)$  for which the innovation ( $\sigma_x^2$ ) and measurement ( $\sigma_y^2$ ) variances are set via arguments varStateEvol and varObs, respectively.

## Value

compareNCestimates returns a named list of two

- outSMC a named list of two:
  - smcOut: a matrix of dimension simNum x 4 which contains single log-likelihood estimates of the standard BPF for each of the 4 resampling schemes and for each simulation run
  - csmcOut: a matrix of dimension simNum x 4 which contains single log-likelihood estimates of the conditional BPF for each of the 4 resampling schemes and for each simulation run
- outKalman the output of kalmanFFBS, see below

kalmanFFBS returns a named list of two:

- `logLikeliEstim`: (exact) estimate of the log-likelihood
- `xBackwardSimul`: a backward simulation (joint smoothing) trajectory of latent states given parameters and measurement

### Author(s)

Adam M. Johansen, Dirk Eddelbuettel, Leah South and Ilya Zarubin

### References

A. M. Johansen. SMCTC: Sequential Monte Carlo in C++. Journal of Statistical Software, 30(6):1-41, April 2009. <https://www.jstatsoft.org/article/view/v030i06>

### See Also

The SMCTC paper and code at <https://www.jstatsoft.org/article/view/v030i06>.

---

LinReg

*Simple Linear Regression*

---

### Description

A simple example based on estimating the parameters of a linear regression model using

\* Data annealing sequential Monte Carlo (`LinReg`).

\* Likelihood annealing sequential Monte Carlo (`LinRegLA`).

\* Likelihood annealing sequential Monte Carlo with the temperature schedule, number of MCMC repeats and random walk covariance matrices adapted online (`LinRegLA_adapt`).

### Usage

```
LinReg(model, particles = 1000, plot = FALSE)
```

```
LinRegLA(model, particles = 1000, temperatures = seq(0, 1, 0.05)^5)
```

```
LinRegLA_adapt(model, particles = 1000, resampTol = 0.5, tempTol = 0.9)
```

### Arguments

<code>model</code>	Choice of regression model (1 for density as the predictor and 2 for adjusted density as the predictor).
<code>particles</code>	An integer specifying the number of particles.
<code>plot</code>	A boolean variable to determine whether to plot the posterior estimates.
<code>temperatures</code>	In likelihood annealing SMC the targets are defined as $P(y \theta)^{\gamma_t} P(\theta)$ where $0 = \gamma_0 \leq \dots \leq \gamma_T = 1$ can be referred to as the temperatures, $P(y \theta)$ is the likelihood and $P(\theta)$ is the prior.

resampTol	The adaptive implementation of likelihood annealing SMC allows for the resampling tolerance to be specified. This parameter can be set to a value in the range [0,1) corresponding to a fraction of the size of the particle set or it may be set to an integer corresponding to an actual effective sample size.
tempTol	A tolerance for adaptive choice of the temperature schedule such that the conditional ESS is maintained at tempTol*particles.

## Details

Williams (1959) considers two competing linear regression models for the maximum compression strength parallel to the grain for radiata pine. Both models are of the form

$$y_i = \alpha + \beta(x_i - \bar{x}) + \epsilon_i,$$

where  $\epsilon_i \sim N(0, \sigma^2)$  and  $i = 1, \dots, 42$ . Here  $y$  is the maximum compression strength in pounds per square inch. The density (in pounds per cubic foot) of the radiata pine is considered a useful predictor, so model 1 uses density for  $x$ . Model 2 instead considers the density adjusted for resin content, which is associated with high density but not with strength.

This example is frequently used as a test problem in model choice (see for example Carlin and Chib (1995) and Friel and Pettitt (2008)). We use the standard uninformative normal and inverse gamma priors for this example along with the transformation  $\phi = \log(\sigma^2)$  so that all parameters are on the real line and  $\theta = [\alpha, \beta, \phi]$ . The evidence can be computed using numerical estimation for both of the competing models. The log evidence is -309.9 for model 1 and -301.4 for model 2.

The `LinReg` function implements a data annealing approach to this example.

The `LinRegLA` function implements a likelihood annealing approach to this example.

The `LinRegLA_adapt` function implements a likelihood annealing approach to this example with adaptation of the temperature schedule, number of MCMC repeats and random walk covariance matrices.

## Value

The `LinReg` function returns a list containing the final particle approximation to the target ( $\theta$  and the corresponding weights) as well as the logarithm of the estimated model evidence.

The `LinRegLA` function returns a list containing the population of particles and their associated log likelihoods, log priors and weights at each iteration. The effective sample size at each of the iterations and several different estimates of the logarithm of the model evidence are also returned.

The `LinRegLA_adapt` function returns a list containing all of the same output as `LinRegLA`, in addition to the adaptively chosen temperature schedule and number of MCMC repeats.

## Author(s)

Adam M. Johansen, Dirk Eddelbuettel and Leah F. South

## References

B. P. Carlin and S. Chib. Bayesian model choice via Markov chain Monte Carlo. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. 57(3):473-484, 1995.

N. Friel and A. N. Pettitt. Marginal likelihood estimation via power posteriors. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. 70(3):589-607, 2008.

Williams, E. (1959). *Regression analysis*. Wiley.

### Examples

```
res <- LinReg(model=1, particles=1000, plot=TRUE)
```

```
res <- LinRegLA(model=1, particles=1000)
```

```
res <- LinRegLA_adapt(model=1, particles=1000)
```

---

nonLinPMMH	<i>Particle marginal Metropolis-Hastings for a non-linear state space model.</i>
------------	--

---

### Description

The nonLinPMMH function implements particle marginal Metropolis Hastings for the non-linear state space model described in Section 3.1 of Andrieu et al. (2010).

### Usage

```
nonLinPMMH(data, particles = 5000, iterations = 10000, burnin = 0,
  verbose = FALSE, msg_freq = 100, plot = FALSE)
```

### Arguments

data	A vector of the observed data.
particles	An integer specifying the number of particles in the particle filtering estimates of the likelihood.
iterations	An integer specifying the number of MCMC iterations.
burnin	The number of iterations to remove from the beginning of the MCMC chain (for plotting purposes only).
verbose	Logical; if TRUE convergence diagnostics are printed to the console (each msg_freq iterations) displaying the running means of parameters, the log-prior, the log-likelihood and the MH acceptance rates up to the current iteration; defaults to FALSE in which case only percentage completion of the procedure is printed.
msg_freq	Specifies the printing frequency of percentage completion or, if verbose = TRUE, percentage completion as well as convergence diagnostics.
plot	A boolean variable to determine whether to plot the posterior estimates and MCMC chain.

## Details

This example uses particle marginal Metropolis Hastings to estimate the standard deviation of the evolution and observation noise in the following non-linear state space model:

$$x(n) = 0.5x(n-1) + 25x(n-1)/(1+x(n-1)^2) + 8\cos(1.2n) + e(n) \text{ and}$$

$$y(n) = x(n)^2/20 + f(n)$$

where  $e(n)$  and  $f(n)$  are mutually-independent normal random variables of variances `var_evol` and `var_obs`, respectively, and  $x(0) \sim N(0, 5)$ .

Following Andrieu, Doucet and Holenstein (2010), the priors are  $var_{evol} \sim IG(0.01, 0.01)$  and  $var_{obs} \sim IG(0.01, 0.01)$  where IG is the inverse gamma distribution.

Data can be simulated from the model using [simNonlin](#).

## Value

A data frame containing the chain of simulated  $\sigma_v$  and  $\sigma_w$  values, as well as the corresponding log likelihood estimates and log prior values.

## Author(s)

Adam M. Johansen, Dirk Eddelbuettel and Leah F. South

## References

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269-342, 2010.

## See Also

[simNonlin](#) for a function to simulate from the model and [pfNonlinBS](#) for a simple bootstrap particle filter applied to a similar non-linear state space model.

## Examples

```
## Not run:
sim <- simNonlin(len=500, var_init=5, var_evol=10, var_obs=1, cosSeqOffset=0)
res <- nonLinPMMH(sim$data, particles=5000, iterations=50000, burnin=10000, plot=TRUE)

## End(Not run)
```

---

pfLineartBS

*Particle Filter Example*

---

## Description

The pfLineartBS function provides a simple example for **RcppSMC**. It is based on the first example in SMCTC and the discussion in Section 5.1 of Johansen (2009). A simple 'vehicle tracking' problem of 100 observations is solved with 1000 particles.

The pfLineartBSOnlinePlot function provides a simple default 'online' plotting function that is invoked during the estimation process.

The simLineart function simulates data from the model.

## Usage

```
pfLineartBS(data, particles=1000, plot=FALSE, onlinePlot)
pfLineartBSOnlinePlot(xm, ym)
simLineart(len)
```

## Arguments

data	A two-column matrix or dataframe containing x and y values. The default data set from Johansen (2009) is used as the default if no data is supplied.
particles	An integer specifying the number of particles.
plot	A boolean variable describing whether plot should illustrate the estimated path along with the data.
onlinePlot	A user-supplied callback function which is called with the x and y position vectors during each iteration of the algorithm; see pfExOnlinePlot for a simple example.
xm	Vector with x position.
ym	Vector with y position.
len	Length of sequence to simulate

## Details

The pfLineartBS function provides a simple example for **RcppSMC**. The model is linear with t-distributed innovations. It is based on the pf example in the SMCTC library, and discussed in the Section 5.1 of his corresponding paper (Johansen, 2009). simLineart simulates from the model.

Using the simple pfExOnlinePlot function illustrates how callbacks into R, for example for plotting, can be made during the operation of SMC algorithm.

**Value**

The `pfLineartBS` function returns a `data.frame` containing as many rows as in the input data, and four columns corresponding to the estimated  $x$  and  $y$  coordinates as well as the estimated velocity in these two directions.

The `simLineart` function returns a list containing the vector of states and the associated vector of observations.

**Author(s)**

Adam M. Johansen and Dirk Eddelbuettel

**References**

A. M. Johansen. SMCTC: Sequential Monte Carlo in C++. *Journal of Statistical Software*, 30(6):1-41, April 2009, doi:10.18637/jss.v030.i06.

**See Also**

The SMCTC paper and code at doi:10.18637/jss.v030.i06.

**Examples**

```
res <- pfLineartBS(plot=TRUE)
if (interactive()) ## if not running R CMD check etc
  res <- pfLineartBS(onlinePlot=pfLineartBSOnlinePlot)
```

---

pfNonlinBS	<i>Nonlinear Bootstrap Particle Filter (Univariate Non-Linear State Space Model)</i>
------------	--

---

**Description**

The `pfNonlinBS` function provides a simple example for **ReppSMC**. It is a simple “bootstrap” particle filter which employs multinomial resampling after each iteration applied to the ubiquitous “nonlinear state space model” following Gordon, Salmond and Smith (1993).

**Usage**

```
pfNonlinBS(data, particles=500, plot=FALSE)
```

**Arguments**

<code>data</code>	A vector variable containing the sequence of observations.
<code>particles</code>	An integer specifying the number of particles.
<code>plot</code>	A boolean variable describing whether a plot should illustrate the (posterior mean) estimated path along with one and two standard deviation intervals.

**Details**

The `pfNonlinBS` function provides a simple example for **RcppSMC**. It is based on a simple nonlinear state space model in which the state evolution and observation equations are:  $x(n) = 0.5 x(n-1) + 25 x(n-1) / (1+x(n-1)^2) + 8 \cos(1.2(n-1)) + e(n)$  and  $y(n) = x(n)^2 / 20 + f(n)$  where  $e(n)$  and  $f(n)$  are mutually-independent normal random variables of variances 10.0 and 1.0, respectively. A bootstrap proposal (i.e. sampling from the state equation) is used, together with multinomial resampling after each iteration.

**Value**

The `pfNonlinBS` function returns two vectors, the first containing the posterior filtering means; the second the posterior filtering standard deviations.

**Author(s)**

Adam M. Johansen, Dirk Eddelbuettel and Leah F. South

**References**

N. J. Gordon, S. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings-F*, 140(2):107-113, April 1993.

**See Also**

[simNonlin](#) for a function to simulate from the model and [nonLinPMMH](#) for an example of particle marginal Metropolis Hastings applied to a non-linear state space model.

**Examples**

```
sim <- simNonlin(len=50)
res <- pfNonlinBS(sim$data,particles=500,plot=TRUE)
```

---

radiata

*Radiata pine dataset (linear regression example)*

---

**Description**

This dataset was originally presented in Table 5.1 of Williams (1959) where two non-nested linear regression models were considered.

**Usage**

radiata

**Format**

A data frame with 42 rows and three variables:

**y** Maximum compression strength (response) in pounds per square inch

**x1** Density (predictor 1) in pounds per cubic foot

**x2** Adjusted density (predictor 2) in pounds per cubic foot

**Source**

E. Williams. Regression analysis. Wiley, 1959.

---

RcppSMC.package.skeleton

*Create a skeleton for a new package that intends to use RcpSMCp*

---

**Description**

RcppSMC.package.skeleton automates the creation of a new source package that intends to use features of RcppSMC.

It is based on the [package.skeleton](#) and [kitten](#) (from [pkgKitten](#)) functions, the latter being a Wrapper around [package.skeleton](#) to make a package pass ‘R CMD check’ without complaints. If [pkgKitten](#) is not installed, [package.skeleton](#) is executed instead.

**Usage**

```
RcppSMC.package.skeleton(name = "anRpackage", list = character(),
environment = .GlobalEnv, path = ".")
```

**Arguments**

name	See <a href="#">package.skeleton</a>
list	See <a href="#">package.skeleton</a>
environment	See <a href="#">package.skeleton</a>
path	See <a href="#">package.skeleton</a>

**Details**

In addition to [package.skeleton](#) :

The ‘DESCRIPTION’ file gains a Depends line requesting that the package depends on Rcpp, RcppArmadillo and RcppSMC and a LinkingTo line so that the package finds the associated header files.

The ‘NAMESPACE’, if any, gains a useDynLib directive.

The ‘src’ directory is created if it does not exist and a ‘Makevars’ file is added setting the environment variable ‘PKG\_LIBS’ to accommodate the necessary flags to link with the Rcpp library.

An example file ‘rcppsmc\_hello\_world.cpp’ is created in the ‘src’. An R file ‘rcppsmc\_hello\_world.R’ is expanded in the ‘R’ directory, the rcppsmc\_hello\_world function defined in this files makes use of the C++ function ‘rcppsmc\_hello\_world’ defined in the C++ file. These files are given as an example and should eventually be removed from the generated package.

### Value

Nothing, used for its side effects

### References

Read the *Writing R Extensions* manual for more details.

Once you have created a *source* package you need to install it: see the *R Installation and Administration* manual, [INSTALL](#) and [install.packages](#).

### See Also

[package.skeleton](#) [kitten](#)

### Examples

```
## Not run:
RcppSMC.package.skeleton( "foobar" )

## End(Not run)
```

---

simNonlin

*Simulates from a simple nonlinear state space model.*

---

### Description

The simNonlin function simulates data from the models used in `link{pfNonlinBS}` and `link{nonLinPMMH}`.

### Usage

```
simNonlin(len = 50, var_init = 10, var_evol = 10, var_obs = 1,
  cosSeqOffset = -1)
```

### Arguments

len	The length of data sequence to simulate.
var_init	The variance of the noise for the initial state.
var_evol	The variance of the noise for the state evolution .
var_obs	The variance of the observation noise.
cosSeqOffset	This is related to the indexing in the cosine function in the evolution equation. A value of -1 can be used to follow the specification of Gordon, Salmond and Smith (1993) and 0 can be used to follow Andrieu, Doucet and Holenstein (2010).

**Details**

The `simNonlin` function simulates from a simple nonlinear state space model with state evolution and observation equations:

$$x(n) = 0.5x(n-1) + 25x(n-1)/(1+x(n-1)^2) + 8\cos(1.2(n + \text{cosSeqOffset})) + e(n) \text{ and} \\ y(n) = x(n)^2/20 + f(n)$$

where  $e(n)$  and  $f(n)$  are mutually-independent normal random variables of variances `var_evol` and `var_obs`, respectively, and  $x(0) \sim N(0, \text{var\_init})$ .

Different variations of this model can be found in Gordon, Salmond and Smith (1993) and Andrieu, Doucet and Holenstein (2010). A `cosSeqOffset` of -1 is consistent with the former and 0 is consistent with the latter.

**Value**

The `simNonlin` function returns a list containing the state and data sequences.

**Author(s)**

Adam M. Johansen, Dirk Eddelbuettel and Leah F. South

**References**

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269-342, 2010.

N. J. Gordon, S. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings-F*, 140(2):107-113, April 1993.

**See Also**

[pfNonlinBS](#) for a simple bootstrap particle filter applied to this model and [nonLinPMMH](#) for particle marginal Metropolis Hastings applied to estimating the standard deviation of the state evolution and observation noise.

# Index

- \* **Bayesian PMMH PF**
  - nonLinPMMH, 8
- \* **datasets**
  - radiata, 12
- \* **programming**
  - blockpfGaussianOpt, 2
  - compareNCestimates, 3
  - LinReg, 6
  - nonLinPMMH, 8
  - pfLineartBS, 10
  - pfNonlinBS, 11
  - RcppSMC.package.skeleton, 13
- \*
  - nonLinPMMH, 8
- blockpfGaussianOpt, 2
- compareNCestimates, 3, 4
- INSTALL, 14
- install.packages, 14
- kalmanFFBS (compareNCestimates), 3
- kitten, 13, 14
- LinReg, 6
- LinRegLA (LinReg), 6
- LinRegLA\_adapt (LinReg), 6
- nonLinPMMH, 8, 12, 15
- package.skeleton, 13, 14
- pfLineartBS, 10
- pfLineartBSOnlinePlot (pfLineartBS), 10
- pfNonlinBS, 9, 11, 15
- pkgKitten, 13
- radiata, 12
- RcppSMC.package.skeleton, 13
- simGaussian (blockpfGaussianOpt), 2
- simGaussianSSM, 4
- simGaussianSSM (compareNCestimates), 3
- simLineart (pfLineartBS), 10
- simNonlin, 9, 12, 14