Package ‘is2’

November 4, 2018

Type Package

Title Iterated smoothing for partially observed Markov processes

Version 0.11-1

Date 2018-11-03

URL http://is2.r-forge.r-project.org

Description Inference methods using iterated smoothing for partially-observed Markov processes

Depends R(>= 3.0.0), pomp

Imports stats, graphics, mvtnorm, deSolve, coda

License GPL(>= 2)

LazyData true

Collate generics.R pfilter1.R pfilter1_methods.R pfilter2.R
pfilter2_methods.R pfilter3.R pfilter3_methods.R pfilter4.R
pfilter4_methods.R aif.R aif_methods.R mif1.R mif1_methods.R
avif.R avif_methods.R mifMomentum.R mifMomentum_methods.R
pmif.R pmif_methods.R psmooth.R psmooth_methods.R is2.R
is2_methods.R is3.R is3_methods.R mif3.R mif3_methods.R

Author Dao Nguyen [aut, cre], Xin Dang [aut]

Maintainer Dao Nguyen <dxnguyenxd@olemiss.edu>

Repository R-Forge

Repository/R-Forge/Project is2

Repository/R-Forge/Revision 1.0

Repository/R-Forge/DateTimeStamp 2018-11-03 11:11:11

Date/Publication 2018-11-03 11:11:11

R topics documented:

`aif` *Iterated Smoothing*

Description

Iterated smoothing algorithms for estimating the parameters of a partially-observed Markov process.

Usage

```
## S4 method for signature 'pomp'
aif(object, Nis = 1, start, pars, ivps = character(0),
    particles, rw.sd, Np, ic.lag, var.factor, lag,
    cooling.type = c("geometric","hyperbolic"),
    cooling.fraction, cooling.factor,
    method = c("aif","ris1","is1"),
    tol = 1e-17, max.fail = Inf,
    verbose = getOption("verbose"), transform = FALSE, ...)
## S4 method for signature 'pfilterd2.pomp'
aif(object, Nis = 1, Np, tol, ...)
## S4 method for signature 'aif'
aif(object, Nis, start, pars, ivps,
    particles, rw.sd, Np, ic.lag, lag, var.factor,
    cooling.type, cooling.fraction,
    method, tol, transform, ...)
## S4 method for signature 'aif'
continue(object, Nis = 1, ...)
```

Arguments

`object`	An object of class pomp.
`Nis`	The number of filtering iterations to perform.
`start`	named numerical vector; the starting guess of the parameters.

`pars`	optional character vector naming the ordinary parameters to be estimated. Every parameter named in `pars` must have a positive random-walk standard deviation specified in `rw.sd`. Leaving `pars` unspecified is equivalent to setting it equal to the names of all parameters with a positive value of `rw.sd` that are not `ivps`.
`ivps`	optional character vector naming the initial-value parameters (IVPs) to be estimated. Every parameter named in `ivps` must have a positive random-walk standard deviation specified in `rw.sd`. If `pars` is empty, i.e., only IVPs are to be estimated, see below "Using aif to estimate initial-value parameters only".
`particles`	Function of prototype `particles(Np,center,sd,...)` which sets up the starting particle matrix by drawing a sample of size `Np` from the starting particle distribution centered at `center` and of width `sd`. If `particles` is not supplied by the user, the default behavior is to draw the particles from a multivariate normal distribution with mean `center` and standard deviation `sd`.
`rw.sd`	numeric vector with names; the intensity of the random walk to be applied to parameters. The random walk is only applied to parameters named in `pars` (i.e., not to those named in `ivps`). The algorithm requires that the random walk be nontrivial, so each element in `rw.sd[pars]` must be positive. `rw.sd` is also used to scale the initial-value parameters (via the `particles` function). Therefore, each element of `rw.sd[ivps]` must be positive. The following must be satisfied: `names(rw.sd)` must be a subset of `names(start)`, `rw.sd` must be non-negative (zeros are simply ignored), the name of every positive element of `rw.sd` must be in either `pars` or `ivps`.
`Np`	the number of particles to use in filtering. This may be specified as a single positive integer, in which case the same number of particles will be used at each timestep. Alternatively, if one wishes the number of particles to vary across timestep, one may specify `Np` either as a vector of positive integers (of length `length(time(object,t0=TRUE))`) or as a function taking a positive integer argument. In the latter case, `Np(k)` must be a single positive integer, representing the number of particles to be used at the `k`-th timestep: `Np(0)` is the number of particles to use going from `timezero(object)` to `time(object)[1]`, `Np(1)`, from `timezero(object)` to `time(object)[1]`, and so on, while when `T=length(time(object,t0=TRUE))`, `Np(T)` is the number of particles to sample at the end of the time-series.
`ic.lag`	a positive integer; the timepoint for fixed-lag smoothing of initial-value parameters. The `aif` update for initial-value parameters consists of replacing them by their filtering mean at time `times[ic.lag]`, where `times=time(object)`. It makes no sense to set `ic.lag>length(times)`; if it is so set, `ic.lag` is set to `length(times)` with a warning.
`var.factor`	a positive number; the scaling coefficient relating the width of the starting particle distribution to `rw.sd`. In particular, the width of the distribution of particles at the start of the first aif iteration will be `random.walk.sd*var.factor`.
`lag`	a positive integer; the timepoint for fixed-lag smoothing parameters estimation.
`cooling.type, cooling.fraction, cooling.factor`	specifications for the cooling schedule, i.e., the manner in which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. When `cooling.type="geometric"`, on the n-th aif iteration, the relative perturbation intensity is

	`cooling.fraction-^(n/50)`. When `cooling.type="hyperbolic"`, on the n-th aif iteration, the relative perturbation intensity is `(s+1)/(s+n)`, where `(s+1)/-(s+50)=cooling.fraction`. `cooling.fraction` is the relative magnitude of the parameter perturbations after 50 aif iterations. `cooling.factor` is now deprecated: to achieve the old behavior, use `cooling.type="geometric"` and `cooling.fraction- =(cooling.factor)^50`.
`method`	`method` sets the update rule used in the algorithm. `method="aif"` uses the iterated smoothing update rule (Ionides 2015 submitted); `method="ris1"` uses the reduced iterated smoothing update rule (Doucet 2013, Ionides 2015 submitted); `method="is1"` uses the reduced iterated smoothing update rule (Doucet 2013);
`tol`	See the description under `pfilter2`.
`max.fail`	See the description under `pfilter2`.
`verbose`	logical; if TRUE, print progress reports.
`transform`	logical; if `TRUE`, optimization is performed on the transformed scale.
`...`	additional arguments that override the defaults.

Re-running aif Iterations

To re-run a sequence of aif iterations, one can use the `aif` method on a `aif` object. By default, the same parameters used for the original aif run are re-used (except for `weighted`, `tol`, `max.fail`, and `verbose`, the defaults of which are shown above). If one does specify additional arguments, these will override the defaults.

Continuing aif Iterations

One can resume a series of aif iterations from where one left off using the `continue` method. A call to `aif` to perform `Nis=m` iterations followed by a call to `continue` to perform `Nis=n` iterations will produce precisely the same effect as a single call to `aif` to perform `Nis=m+n` iterations. By default, all the algorithmic parameters are the same as used in the original call to `aif`. Additional arguments will override the defaults.

Using aif to estimate initial-value parameters only

One can use aif fixed-lag smoothing to estimate only initial value parameters (IVPs). In this case, `pars` is left empty and the IVPs to be estimated are named in `ivps`. If `theta` is the current parameter vector, then at each aif iteration, `Np` particles are drawn from a distribution centered at `theta` and with width proportional to `var.factor*rw.sd`, a particle filtering operation is performed, and `theta` is replaced by the filtering mean at `time(object)[ic.lag]`. Note the implication that, when `aif` is used in this way on a time series any longer than `ic.lag`, unnecessary work is done. If the time series in `object` is longer than `ic.lag`, consider replacing `object` with `window(object,end=ic.lag)`.

Details

If `particles` is not specified, the default behavior is to draw the particles from a multivariate normal distribution. It is the user's responsibility to ensure that, if the optional `particles` argument is given, that the `particles` function satisfies the following conditions:

`particles` has at least the following arguments: `Np`, `center`, `sd`, and `...`. `Np` may be assumed to be a positive integer; `center` and `sd` will be named vectors of the same length. Additional arguments may be specified; these will be filled with the elements of the `userdata` slot of the underlying `pomp` object (see `pomp`).

`particles` returns a `length(center)` x `Np` matrix with rownames matching the names of `center` and `sd`. Each column represents a distinct particle.

The center of the particle distribution returned by `particles` should be `center`. The width of the particle distribution should vary monotonically with `sd`. In particular, when `sd=0`, the `particles` should return matrices with `Np` identical columns, each given by the parameters specified in `center`.

Author(s)

Dao Nguyen `<dxnguyen at olemiss dot edu>`, Xin Dang `<xdang at olemiss dot edu>`, Duc Anh Doan `<ddoan at olemiss dot edu>`

References

D. Nguyen, E. L. Ionides, A second-order iterated smoothing, Computational Statistics, 2017.

A., Doucet, P. E. Jacob, and S. Rubenthaler, Derivative-free estimation of the score vector and observed information matrix with application to state-space models, Preprint arXiv:1304.5768.

A. A. King, D. Nguyen, and E. L. Ionides, Statistical inference for partially observed Markov processes via the R package pomp, Journal of Statistical Software, 2016.

See Also

`aif-methods`, `pfilter2`. See the “intro_to_aif” vignette for examples.

`aif-methods` *Methods of the "aif" class*

Description

Methods of the `aif` class.

Usage

```
## S4 method for signature 'aif'
logLik(object, ...)
## S4 method for signature 'aif'
conv.rec(object, pars, transform = FALSE, ...)
## S4 method for signature 'aifList'
conv.rec(object, ...)
## S4 method for signature 'aif'
plot(x, y, ...)
## S4 method for signature 'aifList'
plot(x, y, ...)
```

```
## S4 method for signature 'aif'
c(x, ..., recursive = FALSE)
## S4 method for signature 'aifList'
c(x, ..., recursive = FALSE)
compare.aif(z)
```

Arguments

`object`	The `aif` object.
`pars`	Names of parameters.
`x`	The `aif` object.
`y, recursive`	Ignored.
`z`	A `aif` object or list of `aif` objects.
`transform`	optional logical; should the parameter transformations be applied? See `coef` for details.
`...`	Further arguments (either ignored or passed to underlying functions).

Methods

conv.rec `conv.rec(object, pars = NULL)` returns the columns of the convergence-record matrix corresponding to the names in `pars`. By default, all rows are returned.

logLik Returns the value in the `loglik` slot.

c Concatenates `aif` objects into an `aifList`.

plot Plots a series of diagnostic plots.

compare.aif Deprecated: use `plot` instead.

Author(s)

Dao Nguyen <`dxnguyen at olemiss dot edu`>, Xin Dang <`xdang at olemiss dot edu`>, Duc Anh Doan <`ddoan at olemiss dot edu`>

See Also

`aif`, `pfilter2`

`avif` *Iterated Smoothing*

Description

Iterated smoothing algorithms for estimating the parameters of a partially-observed Markov process.

Usage

```
## S4 method for signature 'pomp'
avif(object, Nis = 1, start, pars, ivps = character(0),
    particles, rw.sd, Np, ic.lag, var.factor, lag,
    cooling.type = c("geometric","hyperbolic"),
    cooling.fraction, cooling.factor,
    method = c("avif","ris1","is1"),
    tol = 1e-17, max.fail = Inf,
    verbose = getOption("verbose"), transform = FALSE, ...)
## S4 method for signature 'pfilterd2.pomp'
avif(object, Nis = 1, Np, tol, ...)
## S4 method for signature 'avif'
avif(object, Nis, start, pars, ivps,
    particles, rw.sd, Np, ic.lag, lag, var.factor,
    cooling.type, cooling.fraction,
    method, tol, transform, ...)
## S4 method for signature 'avif'
continue(object, Nis = 1, ...)
```

Arguments

`object`	An object of class pomp.
`Nis`	The number of filtering iterations to perform.
`start`	named numerical vector; the starting guess of the parameters.
`pars`	optional character vector naming the ordinary parameters to be estimated. Every parameter named in `pars` must have a positive random-walk standard deviation specified in `rw.sd`. Leaving `pars` unspecified is equivalent to setting it equal to the names of all parameters with a positive value of `rw.sd` that are not `ivps`.
`ivps`	optional character vector naming the initial-value parameters (IVPs) to be estimated. Every parameter named in `ivps` must have a positive random-walk standard deviation specified in `rw.sd`. If `pars` is empty, i.e., only IVPs are to be estimated, see below "Using avif to estimate initial-value parameters only".
`particles`	Function of prototype `particles(Np,center,sd,...)` which sets up the starting particle matrix by drawing a sample of size `Np` from the starting particle distribution centered at `center` and of width `sd`. If `particles` is not supplied by the user, the default behavior is to draw the particles from a multivariate normal distribution with mean `center` and standard deviation `sd`.
`rw.sd`	numeric vector with names; the intensity of the random walk to be applied to parameters. The random walk is only applied to parameters named in `pars` (i.e., not to those named in `ivps`). The algorithm requires that the random walk be nontrivial, so each element in `rw.sd[pars]` must be positive. `rw.sd` is also used to scale the initial-value parameters (via the `particles` function). Therefore, each element of `rw.sd[ivps]` must be positive. The following must be satisfied: `names(rw.sd)` must be a subset of `names(start)`, `rw.sd` must be non-negative (zeros are simply ignored), the name of every positive element of `rw.sd` must be in either `pars` or `ivps`.

`Np`	the number of particles to use in filtering. This may be specified as a single positive integer, in which case the same number of particles will be used at each timestep. Alternatively, if one wishes the number of particles to vary across timestep, one may specify `Np` either as a vector of positive integers (of length `length(time(object,t0=TRUE))`) or as a function taking a positive integer argument. In the latter case, `Np(k)` must be a single positive integer, representing the number of particles to be used at the `k`-th timestep: `Np(0)` is the number of particles to use going from `timezero(object)` to `time(object)[1]`, `Np(1)`, from `timezero(object)` to `time(object)[1]`, and so on, while when `T=length(time(object,t0=TRUE))`, `Np(T)` is the number of particles to sample at the end of the time-series.
`ic.lag`	a positive integer; the timepoint for fixed-lag smoothing of initial-value parameters. The `avif` update for initial-value parameters consists of replacing them by their filtering mean at time `times[ic.lag]`, where `times=time(object)`. It makes no sense to set `ic.lag>length(times)`; if it is so set, `ic.lag` is set to `length(times)` with a warning.
`var.factor`	a positive number; the scaling coefficient relating the width of the starting particle distribution to `rw.sd`. In particular, the width of the distribution of particles at the start of the first avif iteration will be `random.walk.sd*var.factor`.
`lag`	a positive integer; the timepoint for fixed-lag smoothing parameters estimation.
`cooling.type, cooling.fraction, cooling.factor`	specifications for the cooling schedule, i.e., the manner in which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. When `cooling.type="geometric"`, on the n-th avif iteration, the relative perturbation intensity is `cooling.fraction^(n/50)`. When `cooling.type="hyperbolic"`, on the n-th avif iteration, the relative perturbation intensity is `(s+1)/(s+n)`, where `(s+1)/(s+50)=cooling.fraction`. `cooling.fraction` is the relative magnitude of the parameter perturbations after 50 avif iterations. `cooling.factor` is now deprecated: to achieve the old behavior, use `cooling.type="geometric"` and `cooling.fraction=(cooling.factor)^50`.
`method`	`method` sets the update rule used in the algorithm. `method="avif"` uses the iterated smoothing update rule (Ionides 2015 submitted); `method="ris1"` uses the reduced iterated smoothing update rule (Doucet 2013, Ionides 2015 submitted); `method="is1"` uses the reduced iterated smoothing update rule (Doucet 2013);
`tol`	See the description under `pfilter2`.
`max.fail`	See the description under `pfilter2`.
`verbose`	logical; if TRUE, print progress reports.
`transform`	logical; if `TRUE`, optimization is performed on the transformed scale.
`...`	additional arguments that override the defaults.

Re-running avif Iterations

To re-run a sequence of avif iterations, one can use the `avif` method on a `avif` object. By default, the same parameters used for the original avif run are re-used (except for `weighted`, `tol`, `max.fail`, and `verbose`, the defaults of which are shown above). If one does specify additional arguments, these will override the defaults.

Continuing avif Iterations

One can resume a series of avif iterations from where one left off using the `continue` method. A call to `avif` to perform `Nis=m` iterations followed by a call to `continue` to perform `Nis=n` iterations will produce precisely the same effect as a single call to `avif` to perform `Nis=m+n` iterations. By default, all the algorithmic parameters are the same as used in the original call to `avif`. Additional arguments will override the defaults.

Using avif to estimate initial-value parameters only

One can use avif fixed-lag smoothing to estimate only initial value parameters (IVPs). In this case, `pars` is left empty and the IVPs to be estimated are named in `ivps`. If `theta` is the current parameter vector, then at each avif iteration, `Np` particles are drawn from a distribution centered at `theta` and with width proportional to `var.factor*rw.sd`, a particle filtering operation is performed, and `theta` is replaced by the filtering mean at `time(object)[ic.lag]`. Note the implication that, when `avif` is used in this way on a time series any longer than `ic.lag`, unnecessary work is done. If the time series in `object` is longer than `ic.lag`, consider replacing `object` with `window(object,end=ic.lag)`.

Details

If `particles` is not specified, the default behavior is to draw the particles from a multivariate normal distribution. It is the user's responsibility to ensure that, if the optional `particles` argument is given, that the `particles` function satisfies the following conditions:

`particles` has at least the following arguments: `Np`, `center`, `sd`, and `...`. `Np` may be assumed to be a positive integer; `center` and `sd` will be named vectors of the same length. Additional arguments may be specified; these will be filled with the elements of the `userdata` slot of the underlying `pomp` object (see pomp).

`particles` returns a `length(center)` x `Np` matrix with rownames matching the names of `center` and `sd`. Each column represents a distinct particle.

The center of the particle distribution returned by `particles` should be `center`. The width of the particle distribution should vary monotonically with `sd`. In particular, when `sd=0`, the `particles` should return matrices with `Np` identical columns, each given by the parameters specified in `center`.

Author(s)

Dao Nguyen `<dxnguyen at olemiss dot edu>`, Xin Dang `<xdang at olemiss dot edu>`, Duc Anh Doan `<ddoan at olemiss dot edu>`

References

D. Nguyen, E. L. Ionides, A second-order iterated smoothing, Computational Statistics, 2017.

A., Doucet, P. E. Jacob, and S. Rubenthaler, Derivative-free estimation of the score vector and observed information matrix with application to state-space models, Preprint arXiv:1304.5768.

A. A. King, D. Nguyen, and E. L. Ionides, Statistical inference for partially observed Markov processes via the R package pomp, Journal of Statistical Software, 2016.

See Also

`avif-methods`, `pfilter2`. See the “intro_to_avif” vignette for examples.

`avif-methods` *Methods of the "avif" class*

Description

Methods of the `avif` class.

Usage

```
## S4 method for signature 'avif'
logLik(object, ...)
## S4 method for signature 'avif'
conv.rec(object, pars, transform = FALSE, ...)
## S4 method for signature 'avifList'
conv.rec(object, ...)
## S4 method for signature 'avif'
plot(x, y, ...)
## S4 method for signature 'avifList'
plot(x, y, ...)
## S4 method for signature 'avif'
c(x, ..., recursive = FALSE)
## S4 method for signature 'avifList'
c(x, ..., recursive = FALSE)
compare.avif(z)
```

Arguments

`object`	The `avif` object.
`pars`	Names of parameters.
`x`	The `avif` object.
`y, recursive`	Ignored.
`z`	A `avif` object or list of `avif` objects.
`transform`	optional logical; should the parameter transformations be applied? See `coef` for details.
`...`	Further arguments (either ignored or passed to underlying functions).

Methods

conv.rec `conv.rec(object, pars = NULL)` returns the columns of the convergence-record matrix corresponding to the names in `pars`. By default, all rows are returned.

logLik Returns the value in the `loglik` slot.

c Concatenates `avif` objects into an `avifList`.

plot Plots a series of diagnostic plots.

compare.avif Deprecated: use `plot` instead.

Author(s)

Dao Nguyen <dxnguyen at olemiss dot edu>, Xin Dang <xdang at olemiss dot edu>, Duc Anh Doan <ddoan at olemiss dot edu>

See Also

avif, pfilter2

---

is2 *Iterated Smoothing*

Description

Iterated smoothing algorithms for estimating the parameters of a partially-observed Markov process.

Usage

```
## S4 method for signature 'pomp'
is2(object, Nis = 1, start, pars, ivps = character(0),
    particles, rw.sd, Np, ic.lag, var.factor, lag,
    cooling.type = c("geometric","hyperbolic"),
    cooling.fraction, cooling.factor,
    method = c("is2","ris1","is1"),
    tol = 1e-17, max.fail = Inf,
    verbose = getOption("verbose"), transform = FALSE, ...)
## S4 method for signature 'pfilterd2.pomp'
is2(object, Nis = 1, Np, tol, ...)
## S4 method for signature 'is2'
is2(object, Nis, start, pars, ivps,
    particles, rw.sd, Np, ic.lag, lag, var.factor,
    cooling.type, cooling.fraction,
    method, tol, transform, ...)
## S4 method for signature 'is2'
continue(object, Nis = 1, ...)
```

Arguments

`object`	An object of class pomp.
`Nis`	The number of filtering iterations to perform.
`start`	named numerical vector; the starting guess of the parameters.
`pars`	optional character vector naming the ordinary parameters to be estimated. Every parameter named in `pars` must have a positive random-walk standard deviation specified in `rw.sd`. Leaving `pars` unspecified is equivalent to setting it equal to the names of all parameters with a positive value of `rw.sd` that are not `ivps`.

`ivps`	optional character vector naming the initial-value parameters (IVPs) to be estimated. Every parameter named in `ivps` must have a positive random-walk standard deviation specified in `rw.sd`. If `pars` is empty, i.e., only IVPs are to be estimated, see below “Using is2 to estimate initial-value parameters only”.
`particles`	Function of prototype `particles(Np,center,sd,...)` which sets up the starting particle matrix by drawing a sample of size `Np` from the starting particle distribution centered at `center` and of width `sd`. If `particles` is not supplied by the user, the default behavior is to draw the particles from a multivariate normal distribution with mean `center` and standard deviation `sd`.
`rw.sd`	numeric vector with names; the intensity of the random walk to be applied to parameters. The random walk is only applied to parameters named in `pars` (i.e., not to those named in `ivps`). The algorithm requires that the random walk be nontrivial, so each element in `rw.sd[pars]` must be positive. `rw.sd` is also used to scale the initial-value parameters (via the `particles` function). Therefore, each element of `rw.sd[ivps]` must be positive. The following must be satisfied: `names(rw.sd)` must be a subset of `names(start)`, `rw.sd` must be non-negative (zeros are simply ignored), the name of every positive element of `rw.sd` must be in either `pars` or `ivps`.
`Np`	the number of particles to use in filtering. This may be specified as a single positive integer, in which case the same number of particles will be used at each timestep. Alternatively, if one wishes the number of particles to vary across timestep, one may specify `Np` either as a vector of positive integers (of length `length(time(object,t0=TRUE))`) or as a function taking a positive integer argument. In the latter case, `Np(k)` must be a single positive integer, representing the number of particles to be used at the `k`-th timestep: `Np(0)` is the number of particles to use going from `timezero(object)` to `time(object)[1]`, `Np(1)`, from `timezero(object)` to `time(object)[1]`, and so on, while when `T=length(time(object,t0=TRUE))`, `Np(T)` is the number of particles to sample at the end of the time-series.
`ic.lag`	a positive integer; the timepoint for fixed-lag smoothing of initial-value parameters. The `is2` update for initial-value parameters consists of replacing them by their filtering mean at time `times[ic.lag]`, where `times=time(object)`. It makes no sense to set `ic.lag>length(times)`; if it is so set, `ic.lag` is set to `length(times)` with a warning.
`var.factor`	a positive number; the scaling coefficient relating the width of the starting particle distribution to `rw.sd`. In particular, the width of the distribution of particles at the start of the first is2 iteration will be `random.walk.sd*var.factor`.
`lag`	a positive integer; the timepoint for fixed-lag smoothing parameters estimation.
`cooling.type, cooling.fraction, cooling.factor`	specifications for the cooling schedule, i.e., the manner in which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. When `cooling.type="geometric"`, on the n-th is2 iteration, the relative perturbation intensity is `cooling.fraction^(n/50)`. When `cooling.type="hyperbolic"`, on the n-th is2 iteration, the relative perturbation intensity is `(s+1)/(s+n)`, where `(s+1)/(s+50)=cooling.fraction`. `cooling.fraction` is the relative magnitude of the parameter perturbations after 50 is2 iterations. `cooling.factor` is now

	deprecated: to achieve the old behavior, use `cooling.type="geometric"` and `cooling.fraction- =(cooling.factor)^50`.
`method`	`method` sets the update rule used in the algorithm. `method="is2"` uses the iterated smoothing update rule (Ionides 2015 submitted); `method="ris1"` uses the reduced iterated smoothing update rule (Doucet 2013, Ionides 2015 submitted); `method="is1"` uses the reduced iterated smoothing update rule (Doucet 2013);
`tol`	See the description under `pfilter2`.
`max.fail`	See the description under `pfilter2`.
`verbose`	logical; if TRUE, print progress reports.
`transform`	logical; if `TRUE`, optimization is performed on the transformed scale.
`...`	additional arguments that override the defaults.

Re-running is2 Iterations

To re-run a sequence of is2 iterations, one can use the `is2` method on a `is2` object. By default, the same parameters used for the original is2 run are re-used (except for `weighted`, `tol`, `max.fail`, and `verbose`, the defaults of which are shown above). If one does specify additional arguments, these will override the defaults.

Continuing is2 Iterations

One can resume a series of is2 iterations from where one left off using the `continue` method. A call to `is2` to perform `Nis=m` iterations followed by a call to `continue` to perform `Nis=n` iterations will produce precisely the same effect as a single call to `is2` to perform `Nis=m+n` iterations. By default, all the algorithmic parameters are the same as used in the original call to `is2`. Additional arguments will override the defaults.

Using is2 to estimate initial-value parameters only

One can use is2 fixed-lag smoothing to estimate only initial value parameters (IVPs). In this case, `pars` is left empty and the IVPs to be estimated are named in `ivps`. If `theta` is the current parameter vector, then at each is2 iteration, `Np` particles are drawn from a distribution centered at `theta` and with width proportional to `var.factor*rw.sd`, a particle filtering operation is performed, and `theta` is replaced by the filtering mean at `time(object)[ic.lag]`. Note the implication that, when `is2` is used in this way on a time series any longer than `ic.lag`, unnecessary work is done. If the time series in `object` is longer than `ic.lag`, consider replacing `object` with `window(object,end=ic.lag)`.

Details

If `particles` is not specified, the default behavior is to draw the particles from a multivariate normal distribution. It is the user's responsibility to ensure that, if the optional `particles` argument is given, that the `particles` function satisfies the following conditions:

`particles` has at least the following arguments: `Np`, `center`, `sd`, and `...`. `Np` may be assumed to be a positive integer; `center` and `sd` will be named vectors of the same length. Additional arguments may be specified; these will be filled with the elements of the `userdata` slot of the underlying `pomp` object (see `pomp`).

particles returns a length(center) x Np matrix with rownames matching the names of center and sd. Each column represents a distinct particle.

The center of the particle distribution returned by particles should be center. The width of the particle distribution should vary monotonically with sd. In particular, when sd=0, the particles should return matrices with Np identical columns, each given by the parameters specified in center.

Author(s)

Dao Nguyen <dxnguyen at olemiss dot edu>, Edward L. Ionides <ionides at umich dot edu>

References

D. Nguyen, E. L. Ionides, A second-order iterated smoothing, Computational Statistics, 2017.

A., Doucet, P. E. Jacob, and S. Rubenthaler, Derivative-free estimation of the score vector and observed information matrix with application to state-space models, Preprint arXiv:1304.5768.

A. A. King, D. Nguyen, and E. L. Ionides, Statistical inference for partially observed Markov processes via the R package pomp, Journal of Statistical Software, 2016.

See Also

is2-methods, pfilter2. See the “intro_to_is2” vignette for examples.

---

is2-methods *Methods of the "is2" class*

Description

Methods of the is2 class.

Usage

```
## S4 method for signature 'is2'
logLik(object, ...)
## S4 method for signature 'is2'
conv.rec(object, pars, transform = FALSE, ...)
## S4 method for signature 'is2List'
conv.rec(object, ...)
## S4 method for signature 'is2'
plot(x, y, ...)
## S4 method for signature 'is2List'
plot(x, y, ...)
## S4 method for signature 'is2'
c(x, ..., recursive = FALSE)
## S4 method for signature 'is2List'
c(x, ..., recursive = FALSE)
compare.is2(z)
```

Arguments

`object`	The `is2` object.
`pars`	Names of parameters.
`x`	The `is2` object.
`y, recursive`	Ignored.
`z`	A `is2` object or list of `is2` objects.
`transform`	optional logical; should the parameter transformations be applied? See `coef` for details.
`...`	Further arguments (either ignored or passed to underlying functions).

Methods

conv.rec `conv.rec(object, pars = NULL)` returns the columns of the convergence-record matrix corresponding to the names in `pars`. By default, all rows are returned.

logLik Returns the value in the `loglik` slot.

c Concatenates `is2` objects into an `is2List`.

plot Plots a series of diagnostic plots.

compare.is2 Deprecated: use `plot` instead.

Author(s)

Dao Nguyen `<dxnguyen at oelmiss dot edu>`, Edward L. Ionides `<ionides at umich dot edu>`

See Also

`is2`, `pfilter2`

is3 *Iterated Smoothing*

Description

Iterated smoothing algorithms for estimating the parameters of a partially-observed Markov process.

Usage

```
## S4 method for signature 'pomp'
is3(object, Nis = 1, start, pars, ivps = character(0),
    particles, rw.sd, Np, ic.lag, var.factor, lag,
    cooling.type = c("geometric","hyperbolic"),
    cooling.fraction, cooling.factor,
    method = c("is3","ris1","is1"),
    tol = 1e-17, max.fail = Inf,
    verbose = getOption("verbose"), transform = FALSE, ...)
## S4 method for signature 'pfilterd2.pomp'
is3(object, Nis = 1, Np, tol, ...)
## S4 method for signature 'is3'
is3(object, Nis, start, pars, ivps,
    particles, rw.sd, Np, ic.lag, lag, var.factor,
    cooling.type, cooling.fraction,
    method, tol, transform, ...)
## S4 method for signature 'is3'
continue(object, Nis = 1, ...)
```

Arguments

`object`	An object of class pomp.
`Nis`	The number of filtering iterations to perform.
`start`	named numerical vector; the starting guess of the parameters.
`pars`	optional character vector naming the ordinary parameters to be estimated. Every parameter named in `pars` must have a positive random-walk standard deviation specified in `rw.sd`. Leaving `pars` unspecified is equivalent to setting it equal to the names of all parameters with a positive value of `rw.sd` that are not `ivps`.
`ivps`	optional character vector naming the initial-value parameters (IVPs) to be estimated. Every parameter named in `ivps` must have a positive random-walk standard deviation specified in `rw.sd`. If `pars` is empty, i.e., only IVPs are to be estimated, see below “Using is3 to estimate initial-value parameters only”.
`particles`	Function of prototype `particles(Np,center,sd,...)` which sets up the starting particle matrix by drawing a sample of size `Np` from the starting particle distribution centered at `center` and of width `sd`. If `particles` is not supplied by the user, the default behavior is to draw the particles from a multivariate normal distribution with mean `center` and standard deviation `sd`.
`rw.sd`	numeric vector with names; the intensity of the random walk to be applied to parameters. The random walk is only applied to parameters named in `pars` (i.e., not to those named in `ivps`). The algorithm requires that the random walk be nontrivial, so each element in `rw.sd[pars]` must be positive. `rw.sd` is also used to scale the initial-value parameters (via the `particles` function). Therefore, each element of `rw.sd[ivps]` must be positive. The following must be satisfied: `names(rw.sd)` must be a subset of `names(start)`, `rw.sd` must be non-negative (zeros are simply ignored), the name of every positive element of `rw.sd` must be in either `pars` or `ivps`.
`Np`	the number of particles to use in filtering. This may be specified as a single positive integer, in which case the same number of particles will be used at each timestep. Alternatively, if one wishes the number of particles to vary across timestep, one may specify `Np` either as a vector of positive integers (of length `length(time(object,t0=TRUE))`) or as a function taking a positive integer argument. In the latter case, `Np(k)` must be a single positive integer, representing the number of particles to be used at the `k`-th timestep: `Np(0)` is the number of particles to use going from `timezero(object)` to `time(object)[1]`, `Np(1)`, from `timezero(object)` to `time(object)[1]`, and so on, while when

	`T=length(time(object,t0=TRUE))`, `Np(T)` is the number of particles to sample at the end of the time-series.
`ic.lag`	a positive integer; the timepoint for fixed-lag smoothing of initial-value parameters. The `is3` update for initial-value parameters consists of replacing them by their filtering mean at time `times[ic.lag]`, where `times=time(object)`. It makes no sense to set `ic.lag>length(times)`; if it is so set, `ic.lag` is set to `length(times)` with a warning.
`var.factor`	a positive number; the scaling coefficient relating the width of the starting particle distribution to `rw.sd`. In particular, the width of the distribution of particles at the start of the first is3 iteration will be `random.walk.sd*var.factor`.
`lag`	a positive integer; the timepoint for fixed-lag smoothing parameters estimation.
`cooling.type, cooling.fraction, cooling.factor`	specifications for the cooling schedule, i.e., the manner in which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. When `cooling.type-="geometric"`, on the n-th is3 iteration, the relative perturbation intensity is `cooling.fraction-^(n/50)`. When `cooling.type="hyperbolic"`, on the n-th is3 iteration, the relative perturbation intensity is `(s+1)/(s+n)`, where `(s+1)/-(s+50)=cooling.fraction`. `cooling.fraction` is the relative magnitude of the parameter perturbations after 50 is3 iterations. `cooling.factor` is now deprecated: to achieve the old behavior, use `cooling.type="geometric"` and `cooling.fraction- =(cooling.factor)^50`.
`method`	`method` sets the update rule used in the algorithm. `method="is3"` uses the iterated smoothing update rule (Ionides 2015 submitted); `method="ris1"` uses the reduced iterated smoothing update rule (Doucet 2013, Ionides 2015 submitted); `method="is1"` uses the reduced iterated smoothing update rule (Doucet 2013);
`tol`	See the description under `pfilter2`.
`max.fail`	See the description under `pfilter2`.
`verbose`	logical; if TRUE, print progress reports.
`transform`	logical; if `TRUE`, optimization is performed on the transformed scale.
`...`	additional arguments that override the defaults.

Re-running is3 Iterations

To re-run a sequence of is3 iterations, one can use the `is3` method on a `is3` object. By default, the same parameters used for the original is3 run are re-used (except for `weighted`, `tol`, `max.fail`, and `verbose`, the defaults of which are shown above). If one does specify additional arguments, these will override the defaults.

Continuing is3 Iterations

One can resume a series of is3 iterations from where one left off using the `continue` method. A call to `is3` to perform `Nis=m` iterations followed by a call to `continue` to perform `Nis=n` iterations will produce precisely the same effect as a single call to `is3` to perform `Nis=m+n` iterations. By default, all the algorithmic parameters are the same as used in the original call to `is3`. Additional arguments will override the defaults.

Using is3 to estimate initial-value parameters only

One can use is3 fixed-lag smoothing to estimate only initial value parameters (IVPs). In this case, `pars` is left empty and the IVPs to be estimated are named in `ivps`. If `theta` is the current parameter vector, then at each is3 iteration, `Np` particles are drawn from a distribution centered at `theta` and with width proportional to `var.factor*rw.sd`, a particle filtering operation is performed, and `theta` is replaced by the filtering mean at `time(object)[ic.lag]`. Note the implication that, when `is3` is used in this way on a time series any longer than `ic.lag`, unnecessary work is done. If the time series in `object` is longer than `ic.lag`, consider replacing `object` with `window(object,end=ic.lag)`.

Details

If `particles` is not specified, the default behavior is to draw the particles from a multivariate normal distribution. It is the user's responsibility to ensure that, if the optional `particles` argument is given, that the `particles` function satisfies the following conditions:

`particles` has at least the following arguments: `Np`, `center`, `sd`, and `...`. `Np` may be assumed to be a positive integer; `center` and `sd` will be named vectors of the same length. Additional arguments may be specified; these will be filled with the elements of the `userdata` slot of the underlying pomp object (see `pomp`).

`particles` returns a `length(center)` x `Np` matrix with rownames matching the names of `center` and `sd`. Each column represents a distinct particle.

The center of the particle distribution returned by `particles` should be `center`. The width of the particle distribution should vary monotonically with `sd`. In particular, when `sd=0`, the `particles` should return matrices with `Np` identical columns, each given by the parameters specified in `center`.

Author(s)

Dao Nguyen `<dxnguyen at olemiss dot edu>`, Xin Dang `<xdang at olemiss dot edu>`, Duc Anh Doan `<ddoan at olemiss dot edu>`

References

D. Nguyen, E. L. Ionides, A second-order iterated smoothing, Computational Statistics, 2017.

A., Doucet, P. E. Jacob, and S. Rubenthaler, Derivative-free estimation of the score vector and observed information matrix with application to state-space models, Preprint arXiv:1304.5768.

A. A. King, D. Nguyen, and E. L. Ionides, Statistical inference for partially observed Markov processes via the R package pomp, Journal of Statistical Software, 2016.

See Also

`is3-methods`, `pfilter2`. See the “intro_to_is3” vignette for examples.

is3-methods — *Methods of the "is3" class*

Description

Methods of the `is3` class.

Usage

```
## S4 method for signature 'is3'
logLik(object, ...)
## S4 method for signature 'is3'
conv.rec(object, pars, transform = FALSE, ...)
## S4 method for signature 'is3List'
conv.rec(object, ...)
## S4 method for signature 'is3'
plot(x, y, ...)
## S4 method for signature 'is3List'
plot(x, y, ...)
## S4 method for signature 'is3'
c(x, ..., recursive = FALSE)
## S4 method for signature 'is3List'
c(x, ..., recursive = FALSE)
compare.is3(z)
```

Arguments

`object`	The `is3` object.
`pars`	Names of parameters.
`x`	The `is3` object.
`y, recursive`	Ignored.
`z`	A `is3` object or list of `is3` objects.
`transform`	optional logical; should the parameter transformations be applied? See `coef` for details.
`...`	Further arguments (either ignored or passed to underlying functions).

Methods

conv.rec `conv.rec(object, pars = NULL)` returns the columns of the convergence-record matrix corresponding to the names in `pars`. By default, all rows are returned.

logLik Returns the value in the `loglik` slot.

c Concatenates `is3` objects into an `is3List`.

plot Plots a series of diagnostic plots.

compare.is3 Deprecated: use `plot` instead.

Author(s)

Dao Nguyen <dxnguyen at olemiss dot edu>, Xin Dang <xdang at olemiss dot edu>, Duc Anh Doan <ddoan at olemiss dot edu>

See Also

is3, pfilter2

---

mifMomentum *Iterated Smoothing*

Description

Iterated smoothing algorithms for estimating the parameters of a partially-observed Markov process.

Usage

```
## S4 method for signature 'pomp'
mifMomentum(object, Nis = 1, start, pars, ivps = character(0),
    particles, rw.sd, Np, ic.lag, var.factor, lag,
    cooling.type = c("geometric","hyperbolic"),
    cooling.fraction, cooling.factor,
    method = c("mifMomentum","ris1","is1"),
    tol = 1e-17, max.fail = Inf,
    verbose = getOption("verbose"), transform = FALSE, ...)
## S4 method for signature 'pfilterd2.pomp'
mifMomentum(object, Nis = 1, Np, tol, ...)
## S4 method for signature 'mifMomentum'
mifMomentum(object, Nis, start, pars, ivps,
    particles, rw.sd, Np, ic.lag, lag, var.factor,
    cooling.type, cooling.fraction,
    method, tol, transform, ...)
## S4 method for signature 'mifMomentum'
continue(object, Nis = 1, ...)
```

Arguments

`object`	An object of class pomp.
`Nis`	The number of filtering iterations to perform.
`start`	named numerical vector; the starting guess of the parameters.
`pars`	optional character vector naming the ordinary parameters to be estimated. Every parameter named in `pars` must have a positive random-walk standard deviation specified in `rw.sd`. Leaving `pars` unspecified is equivalent to setting it equal to the names of all parameters with a positive value of `rw.sd` that are not `ivps`.

`ivps`	optional character vector naming the initial-value parameters (IVPs) to be estimated. Every parameter named in `ivps` must have a positive random-walk standard deviation specified in `rw.sd`. If `pars` is empty, i.e., only IVPs are to be estimated, see below "Using mifMomentum to estimate initial-value parameters only".
`particles`	Function of prototype `particles(Np,center,sd,...)` which sets up the starting particle matrix by drawing a sample of size `Np` from the starting particle distribution centered at `center` and of width `sd`. If `particles` is not supplied by the user, the default behavior is to draw the particles from a multivariate normal distribution with mean `center` and standard deviation `sd`.
`rw.sd`	numeric vector with names; the intensity of the random walk to be applied to parameters. The random walk is only applied to parameters named in `pars` (i.e., not to those named in `ivps`). The algorithm requires that the random walk be nontrivial, so each element in `rw.sd[pars]` must be positive. `rw.sd` is also used to scale the initial-value parameters (via the `particles` function). Therefore, each element of `rw.sd[ivps]` must be positive. The following must be satisfied: `names(rw.sd)` must be a subset of `names(start)`, `rw.sd` must be non-negative (zeros are simply ignored), the name of every positive element of `rw.sd` must be in either `pars` or `ivps`.
`Np`	the number of particles to use in filtering. This may be specified as a single positive integer, in which case the same number of particles will be used at each timestep. Alternatively, if one wishes the number of particles to vary across timestep, one may specify `Np` either as a vector of positive integers (of length `length(time(object,t0=TRUE))`) or as a function taking a positive integer argument. In the latter case, `Np(k)` must be a single positive integer, representing the number of particles to be used at the `k`-th timestep: `Np(0)` is the number of particles to use going from `timezero(object)` to `time(object)[1]`, `Np(1)`, from `timezero(object)` to `time(object)[1]`, and so on, while when `T=length(time(object,t0=TRUE))`, `Np(T)` is the number of particles to sample at the end of the time-series.
`ic.lag`	a positive integer; the timepoint for fixed-lag smoothing of initial-value parameters. The `mifMomentum` update for initial-value parameters consists of replacing them by their filtering mean at time `times[ic.lag]`, where `times=time(object)`. It makes no sense to set `ic.lag>length(times)`; if it is so set, `ic.lag` is set to `length(times)` with a warning.
`var.factor`	a positive number; the scaling coefficient relating the width of the starting particle distribution to `rw.sd`. In particular, the width of the distribution of particles at the start of the first mifMomentum iteration will be `random.walk.sd*var.factor`.
`lag`	a positive integer; the timepoint for fixed-lag smoothing parameters estimation.
`cooling.type, cooling.fraction, cooling.factor`	specifications for the cooling schedule, i.e., the manner in which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. When `cooling.type="geometric"`, on the n-th mifMomentum iteration, the relative perturbation intensity is `cooling.fraction^(n/50)`. When `cooling.type="hyperbolic"`, on the n-th mifMomentum iteration, the relative perturbation intensity is `(s+1)/(s+n)`,

	where `(s+1)/-(s+50)=cooling.fraction`. `cooling.fraction` is the relative magnitude of the parameter perturbations after 50 mifMomentum iterations. `cooling.factor` is now deprecated: to achieve the old behavior, use `cooling.type="geometric"` and `cooling.fraction-=(cooling.factor)^50`.
`method`	`method` sets the update rule used in the algorithm. `method="mifMomentum"` uses the iterated smoothing update rule (Ionides 2015 submitted); `method="ris1"` uses the reduced iterated smoothing update rule (Doucet 2013, Ionides 2015 submitted); `method="is1"` uses the reduced iterated smoothing update rule (Doucet 2013);
`tol`	See the description under `pfilter2`.
`max.fail`	See the description under `pfilter2`.
`verbose`	logical; if TRUE, print progress reports.
`transform`	logical; if `TRUE`, optimization is performed on the transformed scale.
`...`	additional arguments that override the defaults.

Re-running mifMomentum Iterations

To re-run a sequence of mifMomentum iterations, one can use the `mifMomentum` method on a `mifMomentum` object. By default, the same parameters used for the original mifMomentum run are re-used (except for `weighted`, `tol`, `max.fail`, and `verbose`, the defaults of which are shown above). If one does specify additional arguments, these will override the defaults.

Continuing mifMomentum Iterations

One can resume a series of mifMomentum iterations from where one left off using the `continue` method. A call to `mifMomentum` to perform `Nis=m` iterations followed by a call to `continue` to perform `Nis=n` iterations will produce precisely the same effect as a single call to `mifMomentum` to perform `Nis=m+n` iterations. By default, all the algorithmic parameters are the same as used in the original call to `mifMomentum`. Additional arguments will override the defaults.

Using mifMomentum to estimate initial-value parameters only

One can use mifMomentum fixed-lag smoothing to estimate only initial value parameters (IVPs). In this case, `pars` is left empty and the IVPs to be estimated are named in `ivps`. If `theta` is the current parameter vector, then at each mifMomentum iteration, `Np` particles are drawn from a distribution centered at `theta` and with width proportional to `var.factor*rw.sd`, a particle filtering operation is performed, and `theta` is replaced by the filtering mean at `time(object)[ic.lag]`. Note the implication that, when `mifMomentum` is used in this way on a time series any longer than `ic.lag`, unnecessary work is done. If the time series in `object` is longer than `ic.lag`, consider replacing `object` with `window(object,end=ic.lag)`.

Details

If `particles` is not specified, the default behavior is to draw the particles from a multivariate normal distribution. It is the user's responsibility to ensure that, if the optional `particles` argument is given, that the `particles` function satisfies the following conditions:

`particles` has at least the following arguments: `Np`, `center`, `sd`, and `...`. `Np` may be assumed to be a positive integer; `center` and `sd` will be named vectors of the same length. Additional arguments may be specified; these will be filled with the elements of the `userdata` slot of the underlying `pomp` object (see `pomp`).

`particles` returns a `length(center)` x `Np` matrix with rownames matching the names of `center` and `sd`. Each column represents a distinct particle.

The center of the particle distribution returned by `particles` should be `center`. The width of the particle distribution should vary monotonically with `sd`. In particular, when `sd=0`, the `particles` should return matrices with `Np` identical columns, each given by the parameters specified in `center`.

Author(s)

Dao Nguyen <`dxnguyen at olemiss dot edu`>, Xin Dang <`xdang at olemiss dot edu`>, Duc Anh Doan <`ddoan at olemiss dot edu`>

References

D. Nguyen, E. L. Ionides, A second-order iterated smoothing, Computational Statistics, 2017.

A., Doucet, P. E. Jacob, and S. Rubenthaler, Derivative-free estimation of the score vector and observed information matrix with application to state-space models, Preprint arXiv:1304.5768.

A. A. King, D. Nguyen, and E. L. Ionides, Statistical inference for partially observed Markov processes via the R package pomp, Journal of Statistical Software, 2016.

See Also

`mifMomentum-methods`, `pfilter2`. See the “intro_to_mifMomentum” vignette for examples.

`mifMomentum-methods` *Methods of the "mifMomentum" class*

Description

Methods of the `mifMomentum` class.

Usage

```
## S4 method for signature 'mifMomentum'
logLik(object, ...)
## S4 method for signature 'mifMomentum'
conv.rec(object, pars, transform = FALSE, ...)
## S4 method for signature 'mifMomentumList'
conv.rec(object, ...)
## S4 method for signature 'mifMomentum'
plot(x, y, ...)
## S4 method for signature 'mifMomentumList'
plot(x, y, ...)
## S4 method for signature 'mifMomentum'
c(x, ..., recursive = FALSE)
```

```
## S4 method for signature 'mifMomentumList'
c(x, ..., recursive = FALSE)
compare.mifMomentum(z)
```

Arguments

`object`	The `mifMomentum` object.
`pars`	Names of parameters.
`x`	The `mifMomentum` object.
`y, recursive`	Ignored.
`z`	A `mifMomentum` object or list of `mifMomentum` objects.
`transform`	optional logical; should the parameter transformations be applied? See `coef` for details.
`...`	Further arguments (either ignored or passed to underlying functions).

Methods

conv.rec `conv.rec(object, pars = NULL)` returns the columns of the convergence-record matrix corresponding to the names in `pars`. By default, all rows are returned.

logLik Returns the value in the `loglik` slot.

c Concatenates `mifMomentum` objects into an `mifMomentumList`.

plot Plots a series of diagnostic plots.

compare.mifMomentum Deprecated: use `plot` instead.

Author(s)

Dao Nguyen <`dxnguyen at olemiss dot edu`>, Xin Dang <`xdang at olemiss dot edu`>, Duc Anh Doan <`ddoan at olemiss dot edu`>

See Also

`mifMomentum`, `pfilter2`

`Particle Iterated Filtering`

The particle Iterated Filtering algorithm

Description

The Particle MCMC algorithm for estimating the parameters of a partially-observed Markov process. Running `pmif` causes a particle random-walk Metropolis-Hastings Markov chain algorithm to run for the specified number of proposals.

Usage

```
## S4 method for signature 'pomp'
pmif(object, Nmif = 1, start, proposal, Np,
    tol = 1e-17, max.fail = Inf, verbose = getOption("verbose"), ...)
## S4 method for signature 'pfilterd.pomp'
pmif(object, Nmif = 1, Np, tol, ...)
## S4 method for signature 'pmif'
pmif(object, Nmif, start, proposal, Np, tol,
    max.fail = Inf, verbose = getOption("verbose"), ...)
## S4 method for signature 'pmif'
continue(object, Nmif = 1, ...)
```

Arguments

`object`	An object of class pomp.
`Nmif`	The number of pmif iterations to perform.
`start`	named numeric vector; the starting guess of the parameters.
`proposal`	optional function that draws from the proposal distribution. Currently, the proposal distribution must be symmetric for proper inference: it is the user's responsibility to ensure that it is. Several functions that construct appropriate proposal function are provided: see MCMC proposal functions for more information.
`Np`	a positive integer; the number of particles to use in each filtering operation.
`tol`	numeric scalar; particles with log likelihood below `tol` are considered to be "lost". A filtering failure occurs when, at some time point, all particles are lost.
`max.fail`	integer; maximum number of filtering failures permitted. If the number of failures exceeds this number, execution will terminate with an error.
`verbose`	logical; if TRUE, print progress reports.
`...`	additional arguments that override the defaults.

Value

An object of class `pmif`.

Re-running pmif Iterations

To re-run a sequence of pmif iterations, one can use the `pmif` method on a `pmif` object. By default, the same parameters used for the original pmif run are re-used (except for `tol`, `max.fail`, and `verbose`, the defaults of which are shown above). If one does specify additional arguments, these will override the defaults.

Continuing pmif Iterations

One can continue a series of pmif iterations from where one left off using the `continue` method. A call to `pmif` to perform `Nmif=m` iterations followed by a call to `continue` to perform `Nmif=n` iterations will produce precisely the same effect as a single call to `pmif` to perform `Nmif=m+n` iterations. By default, all the algorithmic parameters are the same as used in the original call to `pmif`. Additional arguments will override the defaults.

Details

`pmif` implements an MCMC algorithm in which the true likelihood of the data is replaced by an unbiased estimate computed by a particle filter. This gives an asymptotically correct Bayesian procedure for parameter estimation (Andrieu and Roberts, 2009).

Note that `pmif` does not make use of any parameter transformations supplied by the user.

Methods

c Concatenates `pmif` objects into a `pmifList`.

`conv.rec(object, pars)` returns the columns of the convergence-record matrix corresponding to the names in `pars` as an object of class `mcmc` or `mcmc.list`.

`filter.traj(object, vars)` returns filter trajectories from a `pmif` or `pmifList` object.

plot Diagnostic plots.

logLik Returns the value in the `loglik` slot.

coef Returns the last state of the MCMC chain. As such, it's not very useful for inference.

`covmat(object, start, thin, expand)` computes the empirical covariance matrix of the MCMC samples beginning with iteration `start` and thinning by factor `thin`. It expands this by a factor `expand^2/n`, where `n` is the number of parameters estimated. By default, `expand=2.38`. The intention is that the resulting matrix is a suitable input to the proposal function `mvn.rw`.

Author(s)

Dao Nguyen <`dxnguyen at olemiss dot edu`>, Xin Dang <`xdang at olemiss dot edu`>, Duc Anh Doan <`ddoan at olemiss dot edu`>

References

C. Andrieu, A. Doucet and R. Holenstein, Particle Markov chain Monte Carlo methods, J. R. Stat. Soc. B, to appear, 2010.

C. Andrieu and G.O. Roberts, The pseudo-marginal approach for efficient computation, Ann. Stat. 37:697-725, 2009.

See Also

`pomp`, `pfilter`, MCMC proposal distributions, and the tutorials on the package website.

Examples

```
## Not run:
library(pomp)

pompExample(ou2)

pmif(
      pomp(ou2,dprior=Csnippet("
   lik = dnorm(alpha_2,-0.5,1,1) + dnorm(alpha_3,0.3,1,1);
   lik = (give_log) ? lik : exp(lik);"),
           paramnames=c("alpha.2","alpha.3")),
      Nmif=2000,Np=500,verbose=TRUE,
      proposal=mvn.rw.adaptive(rw.sd=c(alpha.2=0.01,alpha.3=0.01),
        scale.start=200,shape.start=100)) -> chain
continue(chain,Nmif=2000,proposal=mvn.rw(covmat(chain))) -> chain
plot(chain)
chain <- pmif(chain)
plot(chain)

library(coda)
trace <- window(conv.rec(chain,c("alpha.2","alpha.3")),start=2000)
rejectionRate(trace)
effectiveSize(trace)
autocorr.diag(trace)

summary(trace)
plot(trace)

heidel.diag(trace)
geweke.diag(trace)

## End(Not run)
```

pfilter2 *Particle filter*

Description

Run a plain vanilla particle filter. Resampling is performed at each observation.

Usage

```
## S4 method for signature 'pomp'
pfilter2(object, params, Np, tol = 1e-17,
    max.fail = Inf, pred.mean = FALSE, pred.var = FALSE,
    filter.mean = FALSE,
    save.states = FALSE,
    save.params = FALSE, lag=0, seed = NULL,
    verbose = getOption("verbose"), ...)
## S4 method for signature 'pfilterd2.pomp'
pfilter2(object, params, Np, tol, ...)
```

Arguments

`object`	An object of class pomp or inheriting class pomp.
`params`	A `npars` x `Np` numeric matrix containing the parameters corresponding to the initial state values in `xstart`. This must have a 'rownames' attribute. If it desired that all particles should share the same parameter values, one one may supply `params` as a named numeric vector.

`Np`	the number of particles to use. This may be specified as a single positive integer, in which case the same number of particles will be used at each timestep. Alternatively, if one wishes the number of particles to vary across timesteps, one may specify `Np` either as a vector of positive integers (`length(time(object,t0=TRUE))`) or as a function taking a positive integer argument. In the latter case, `Np(k)` must be a single positive integer, representing the number of particles to be used at the `k`-th timestep: `Np(0)` is the number of particles to use going from `timezero(object)` to `time(object)[1]`, `Np(1)`, from `timezero(object)` to `time(object)[1]`, and so on, while when `T=length(time(object,t0=TRUE))`, `Np(T)` is the number of particles to sample at the end of the time-series. When `object` is of class `is2`, this is by default the same number of particles used in the `is2` iterations.
`tol`	positive numeric scalar; particles with likelihood less than `tol` are considered to be "lost". A filtering failure occurs when, at some time point, all particles are lost. When all particles are lost, the conditional likelihood at that time point is set to `tol`.
`max.fail`	integer; the maximum number of filtering failures allowed. If the number of filtering failures exceeds this number, execution will terminate with an error. By default, `max.fail` is set to infinity, so no error can be triggered.
`pred.mean`	logical; if `TRUE`, the prediction means are calculated for the state variables and parameters.
`pred.var`	logical; if `TRUE`, the prediction variances are calculated for the state variables and parameters.
`filter.mean`	logical; if `TRUE`, the filtering means are calculated for the state variables and parameters.
`save.states, save.params`	logical. If `save.states=TRUE`, the state-vector for each particle at each time is saved in the `saved.states` slot of the returned `pfilterd2.pomp` object. If `save.params=TRUE`, the parameter-vector for each particle at each time is saved in the `saved.params` slot of the returned `pfilterd2.pomp` object.
`lag`	positive numeric scalar; use for fixed lag smoothing.
`seed`	optional; an object specifying if and how the random number generator should be initialized ('seeded'). If `seed` is an integer, it is passed to `set.seed` prior to any simulation and is returned as the "seed" element of the return list. By default, the state of the random number generator is not changed and the value of `.Random.seed` on the call is stored in the "seed" element of the return list.
`verbose`	logical; if `TRUE`, progress information is reported as `pfilter2` works.
`...`	By default, when `pfilter2` pfilter is run on a `pfilterd2.pomp` object, the settings in the original call are re-used. This default behavior can be overridden by changing the settings (see Examples below).

Value

An object of class `pfilterd2.pomp`. This class inherits from class `pomp` and contains the following additional slots:

pred.mean, pred.var, filter.mean matrices of prediction means, variances, and filter means, respectively. In each of these, the rows correspond to states and parameters (if appropriate), in that order, the columns to successive observations in the time series contained in `object`.

eff.sample.size numeric vector containing the effective number of particles at each time point.

cond.loglik numeric vector containing the conditional log likelihoods at each time point.

saved.states If `pfilter2` was called with `save.states=TRUE`, this is the list of state-vectors at each time point, for each particle. It is a length-`ntimes` list of `nvars`-by-`Np` arrays. In particular, `saved.states[[t]][,i]` can be considered a sample from $f[X_t|y_{1:t}]$.

saved.params If `pfilter2` was called with `save.params=TRUE`, this is the list of parameter-vectors at each time point, for each particle. It is a length-`ntimes` list of `npars`-by-`Np` arrays. In particular, `saved.params[[t]][,i]` is the parameter portion of the i-th particle at time t.

seed the state of the random number generator at the time `pfilter2` was called. If the argument `seed` was specified, this is a copy; if not, this is the internal state of the random number generator at the time of call.

Np, tol, nfail the number of particles used, failure tolerance, and number of filtering failures, respectively.

loglik the estimated log-likelihood.

These can be accessed using the `$` operator as if the returned object were a list. In addition, `logLik` returns the log likelihood. Note that if the argument `params` is a named vector, then these parameters are included in the `params` slot of the returned `pfilterd2.pomp` object. That is `coef(pfilter2(obj,params=theta))==theta` if `theta` is a named vector of parameters.

Author(s)

Dao Nguyen `<dxnguyen at olemiss dot edu>`, Edward L. Ionides `<ionides at umich dot edu>`

References

M. S. Arulampalam, S. Maskell, N. Gordon, & T. Clapp. A Tutorial on Particle Filters for Online Nonlinear, Non-Gaussian Bayesian Tracking. IEEE Trans. Sig. Proc. 50:174–188, 2002.

See Also

`is2`

`pfilter2-methods` *Methods of the "pfilterd2.pomp" class*

Description

Methods of the "pfilterd2.pomp" class.

Usage

```
## S4 method for signature 'pfilterd2.pomp'
logLik(object, ...)
## S4 method for signature 'pfilterd2.pomp'
pred.mean(object, pars, ...)
## S4 method for signature 'pfilterd2.pomp'
pred.var(object, pars, ...)
## S4 method for signature 'pfilterd2.pomp'
filter.mean(object, pars, ...)
## S4 method for signature 'pfilterd2.pomp'
eff.sample.size(object, ...)
## S4 method for signature 'pfilterd2.pomp'
cond.logLik(object, ...)
## S4 method for signature 'pfilterd2.pomp'
as(object, class)
## S4 method for signature 'pfilterd2.pomp,data.frame'
coerce(from, to = "data.frame", strict = TRUE)
## S3 method for class 'pfilterd2.pomp'
as.data.frame(x, row.names, optional, ...)
```

Arguments

`object, x`	An object of class `pfilterd2.pomp` or inheriting class `pfilterd2.pomp`.
`pars`	Names of parameters.
`class`	character; name of the class to which `object` should be coerced.
`from, to`	the classes between which coercion should be performed.
`strict`	ignored.
`row.names, optional`	ignored.
`...`	Additional arguments unused at present.

Author(s)

Dao Nguyen <`dxnguyen at olemiss dot edu`>, Edward L. Ionides <`ionides at umich dot edu`>

See Also

`pfilter2`

`pfilter3` *Particle filter*

Description

Run a plain vanilla particle filter. Resampling is performed at each observation.

Usage

```
## S4 method for signature 'pomp'
pfilter3(object, params, Np, tol = 1e-17,
    max.fail = Inf, pred.mean = FALSE, pred.var = FALSE,
    filter.mean = FALSE,
    save.states = FALSE,
    save.params = FALSE, lag=0, seed = NULL,
    verbose = getOption("verbose"), ...)
## S4 method for signature 'pfilter3d.pomp'
pfilter3(object, params, Np, tol, ...)
```

Arguments

`object`	An object of class pomp or inheriting class pomp.
`params`	A `npars` x `Np` numeric matrix containing the parameters corresponding to the initial state values in `xstart`. This must have a 'rownames' attribute. If it desired that all particles should share the same parameter values, one one may supply `params` as a named numeric vector.
`Np`	the number of particles to use. This may be specified as a single positive integer, in which case the same number of particles will be used at each timestep. Alternatively, if one wishes the number of particles to vary across timesteps, one may specify `Np` either as a vector of positive integers (`length(time(object,t0=TRUE))`) or as a function taking a positive integer argument. In the latter case, `Np(k)` must be a single positive integer, representing the number of particles to be used at the `k`-th timestep: `Np(0)` is the number of particles to use going from `timezero(object)` to `time(object)[1]`, `Np(1)`, from `timezero(object)` to `time(object)[1]`, and so on, while when `T=length(time(object,t0=TRUE))`, `Np(T)` is the number of particles to sample at the end of the time-series. When `object` is of class `is2`, this is by default the same number of particles used in the `is2` iterations.
`tol`	positive numeric scalar; particles with likelihood less than `tol` are considered to be "lost". A filtering failure occurs when, at some time point, all particles are lost. When all particles are lost, the conditional likelihood at that time point is set to `tol`.
`max.fail`	integer; the maximum number of filtering failures allowed. If the number of filtering failures exceeds this number, execution will terminate with an error. By default, `max.fail` is set to infinity, so no error can be triggered.

`pred.mean`	logical; if `TRUE`, the prediction means are calculated for the state variables and parameters.
`pred.var`	logical; if `TRUE`, the prediction variances are calculated for the state variables and parameters.
`filter.mean`	logical; if `TRUE`, the filtering means are calculated for the state variables and parameters.
`save.states, save.params`	logical. If `save.states=TRUE`, the state-vector for each particle at each time is saved in the `saved.states` slot of the returned `pfilter3d.pomp` object. If `save.params=TRUE`, the parameter-vector for each particle at each time is saved in the `saved.params` slot of the returned `pfilter3d.pomp` object.
`lag`	positive numeric scalar; use for fixed lag smoothing.
`seed`	optional; an object specifying if and how the random number generator should be initialized (‘seeded’). If `seed` is an integer, it is passed to `set.seed` prior to any simulation and is returned as the “seed” element of the return list. By default, the state of the random number generator is not changed and the value of `.Random.seed` on the call is stored in the “seed” element of the return list.
`verbose`	logical; if `TRUE`, progress information is reported as `pfilter3` works.
`...`	By default, when `pfilter3` pfilter is run on a `pfilter3d.pomp` object, the settings in the original call are re-used. This default behavior can be overridden by changing the settings (see Examples below).

Value

An object of class `pfilter3d.pomp`. This class inherits from class `pomp` and contains the following additional slots:

pred.mean, pred.var, filter.mean matrices of prediction means, variances, and filter means, respectively. In each of these, the rows correspond to states and parameters (if appropriate), in that order, the columns to successive observations in the time series contained in `object`.

eff.sample.size numeric vector containing the effective number of particles at each time point.

cond.loglik numeric vector containing the conditional log likelihoods at each time point.

saved.states If `pfilter3` was called with `save.states=TRUE`, this is the list of state-vectors at each time point, for each particle. It is a length-`ntimes` list of `nvars`-by-`Np` arrays. In particular, `saved.states[[t]][,i]` can be considered a sample from $f[X_t|y_{1:t}]$.

saved.params If `pfilter3` was called with `save.params=TRUE`, this is the list of parameter-vectors at each time point, for each particle. It is a length-`ntimes` list of `npars`-by-`Np` arrays. In particular, `saved.params[[t]][,i]` is the parameter portion of the i-th particle at time t.

seed the state of the random number generator at the time `pfilter3` was called. If the argument `seed` was specified, this is a copy; if not, this is the internal state of the random number generator at the time of call.

Np, tol, nfail the number of particles used, failure tolerance, and number of filtering failures, respectively.

loglik the estimated log-likelihood.

These can be accessed using the `$` operator as if the returned object were a list. In addition, `logLik` returns the log likelihood. Note that if the argument `params` is a named vector, then these parameters are included in the `params` slot of the returned `pfilter3d.pomp` object. That is `coef(pfilter3(obj,params=theta))==theta` if `theta` is a named vector of parameters.

Author(s)

Dao Nguyen <dxnguyen at olemiss dot edu>, Xin Dang <xdang at olemiss dot edu>, Duc Anh Doan <ddoan at olemiss dot edu>

References

M. S. Arulampalam, S. Maskell, N. Gordon, & T. Clapp. A Tutorial on Particle Filters for Online Nonlinear, Non-Gaussian Bayesian Tracking. IEEE Trans. Sig. Proc. 50:174–188, 2002.

See Also

`is2`

`pfilter3-methods` *Methods of the "pfilter3d.pomp" class*

Description

Methods of the "pfilter3d.pomp" class.

Usage

```
## S4 method for signature 'pfilter3d.pomp'
logLik(object, ...)
## S4 method for signature 'pfilter3d.pomp'
pred.mean(object, pars, ...)
## S4 method for signature 'pfilter3d.pomp'
pred.var(object, pars, ...)
## S4 method for signature 'pfilter3d.pomp'
filter.mean(object, pars, ...)
## S4 method for signature 'pfilter3d.pomp'
eff.sample.size(object, ...)
## S4 method for signature 'pfilter3d.pomp'
cond.logLik(object, ...)
## S4 method for signature 'pfilter3d.pomp'
as(object, class)
## S4 method for signature 'pfilter3d.pomp,data.frame'
coerce(from, to = "data.frame", strict = TRUE)
## S3 method for class 'pfilter3d.pomp'
as.data.frame(x, row.names, optional, ...)
```

Arguments

`object, x`	An object of class `pfilter3d.pomp` or inheriting class `pfilter3d.pomp`.
`pars`	Names of parameters.
`class`	character; name of the class to which `object` should be coerced.
`from, to`	the classes between which coercion should be performed.
`strict`	ignored.
`row.names, optional`	ignored.
`...`	Additional arguments unused at present.

Author(s)

Dao Nguyen <`dxnguyen at olemiss dot edu`>, Xin Dang <`xdang at olemiss dot edu`>, Duc Anh Doan <`ddoan at olemiss dot edu`>

See Also

`is2`

Index