Package ‘spatPomp’

August 4, 2022

R topics documented:

`spatPomp-package` *Inference for SpatPOMPs (Spatiotemporal Partially Observed Markov Processes)*

Description

The **spatPomp** package provides facilities for inference on panel data using spatiotemporal partially-observed Markov process (SPATPOMP) models. To do so, it relies on and extends a number of facilities that the **pomp** package provides for inference on time series data using partially-observed Markov process (POMP) models.

The **spatPomp** package concerns models consisting of a collection of interacting units. The methods in **spatPomp** may be applicable whether or not these units correspond to spatial locations.

Data analysis using spatPomp

The first step in using **spatPomp** is to encode one's model(s) and data in objects of class `spatPomp`. This can be done via a call to the spatPomp constructor function.

Extending the pomp platform for developing inference tools

spatPomp extends to panel data the general interface to the components of POMP models provided by **pomp**. In doing so, it contributes to the goal of the **pomp** project of facilitating the development of new algorithms in an environment where they can be tested and compared on a growing body of models and datasets.

Documentation

spatPomp is described by Asfaw et al. (2020)

License

spatPomp is provided under the MIT License.

Author(s)

Kidus Asfaw, Joonha Park, Allister Ho, Edward Ionides, Aaron King

References

Asfaw, K. et al. (2020) Statistical inference for spatiotemporal partially observed Markov processes via the R package spatPomp. *Manuscript in preparation*.

See Also

pomp package

---

abf *Adapted Bagged Filter (ABF)*

Description

An algorithm for estimating the likelihood of a spatiotemporal partially-observed Markov process model. Running `abf` causes the algorithm to run bootstrap replicate jobs which each yield an imperfect adapted simulation. Simulating from the "adapted filter" distribution runs into a curse of dimensionality (COD) problem, which is mitigated by keeping particles in each replicate close to each other through resampling down to one particle per replicate at each observation time point. The adapted simulations are then weighted in a way that mitigates COD by making a weak coupling assumption to get an approximate filter distribution. As a by-product, we also get an estimate of the likelihood of the data.

Usage

```
## S4 method for signature 'spatPomp'
abf(
  object,
  Nrep,
  Np,
  nbhd,
  tol = 1e-300,
  ...,
  verbose = getOption("verbose", FALSE)
)

## S4 method for signature 'abfd_spatPomp'
abf(
  object,
  Nrep,
  Np,
  nbhd,
  tol = 1e-300,
  ...,
  verbose = getOption("verbose", FALSE)
)
```

Arguments

`object`	A `spatPomp` object.
`Nrep`	The number of bootstrap replicates for the adapted simulations.
`Np`	The number of particles used within each replicate for the adapted simulations.

`nbhd` — A neighborhood function with three arguments: `object`, `time` and `unit`. The function should return a `list` of two-element vectors that represent space-time neighbors of (u, n), which is represented by `c(unit,time)`. See example below for more details.

`tol` — If the resampling weight for a particle is zero due to floating-point precision issues, it is set to the value of `tol` since resampling has to be done.

`...` — If a `params` argument is specified, `abf` will estimate the likelihood at that parameter set instead of at `coef(object)`.

`verbose` — logical; if `TRUE`, messages updating the user on progress will be printed to the console.

Value

Upon successful completion, `abf()` returns an object of class 'abfd_spatPomp' containing the algorithmic parameters used to run `abf()` and the estimated likelihood.

Methods

The following methods are available for such an object:

`logLik` yields an estimate of the log-likelihood of the data under the model.

See Also

Other particle filter methods: `abfir()`, `bpfilter()`, `enkf()`, `girf()`, `ienkf()`, `igirf()`, `iubf()`

Examples

```
# Create a simulation of a Brownian motion
b <- bm(U=3, N=10)

# Create a neighborhood function mapping a point in space-time
# to a list of neighboring points in space-time
bm_nbhd <- function(object, time, unit) {
  nbhd_list = list()
  if(time > 1 && unit > 1){
    nbhd_list = c(nbhd_list, list(c(unit-1, time-1)))
  }
  return(nbhd_list)
}

# Run ABF specified number of Monte Carlo replicates and particles per replicate
abfd_bm <- abf(b, Nrep=2, Np=10, nbhd=bm_nbhd)

# Get the likelihood estimate from ABF
logLik(abfd_bm)
```

abfir *Adapted Bagged Filter with Intermediate Resampling (ABF-IR)*

Description

An algorithm for estimating the filter distribution and likelihood of a spatiotemporal partially-observed Markov process model. Running `abfir` causes the algorithm to run Monte Carlo replicated jobs which each carry out an adapted simulation using intermediate resampling. Adapted simulation is an easier task than filtering, since particles in each replicate remain close to each other. Intermediate resampling further assists against the curse of dimensionality (COD) problem for importance sampling. The adapted simulations are then weighted in a way that mitigates COD by making a weak coupling assumption to get an approximate filter distribution. As a by-product, we also get an approximation to the likelihood of the data.

Usage

```
## S4 method for signature 'spatPomp'
abfir(
  object,
  Np,
  Nrep,
  nbhd,
  Ninter,
  tol = (1e-300),
  ...,
  verbose = getOption("verbose", FALSE)
)

## S4 method for signature 'abfird_spatPomp'
abfir(object, Np, Nrep, nbhd, Ninter, tol, ...)
```

Arguments

`object`	A `spatPomp` object.
`Np`	The number of particles used within each replicate for the adapted simulations.
`Nrep`	The number of bootstrap replicates for the adapted simulations.
`nbhd`	A neighborhood function with three arguments: `object`, `time` and `unit`. The function should return a `list` of two-element vectors that represent space-time neighbors of (u, n), which is represented by `c(unit,time)`. See example below for more details.
`Ninter`	the number of intermediate resampling time points.
`tol`	If the resampling weight for a particle is zero due to floating-point precision issues, it is set to the value of `tol` since resampling has to be done.
`...`	If a `params` argument is specified, `abf` will estimate the likelihood at that parameter set instead of at `coef(object)`.

verbose — logical; if TRUE, messages updating the user on progress will be printed to the console.

Value

Upon successful completion, `abfir()` returns an object of class 'abfird_spatPomp' containing the algorithmic parameters used to run `abfir()` and the estimated likelihood.

Methods

The following methods are available for such an object:

`logLik` yields a biased estimate of the log-likelihood of the data under the model.

See Also

Other particle filter methods: `abf()`, `bpfilter()`, `enkf()`, `girf()`, `ienkf()`, `igirf()`, `iubf()`

Examples

```
# Create a simulation of a Brownian motion
b <- bm(U=3, N=10)

# Create a neighborhood function mapping a point in space-time
# to a list of ``neighboring points" in space-time
bm_nbhd <- function(object, time, unit) {
  nbhd_list = list()
  if(time > 1 && unit > 1){
    nbhd_list = c(nbhd_list, list(c(unit-1, time-1)))
  }
  return(nbhd_list)
}
# Run ABFIR with specified number of Monte Carlo replicates and particles per replicate
abfird_bm <- abfir(b,
                  Nrep = 2,
                  Np=20,
                  nbhd = bm_nbhd,
                  Ninter = length(unit_names(b)))
# Get the likelihood estimate from ABFIR
logLik(abfird_bm)
```

`as.data.frame` *Coerce to data frame*

Description

spatPomp objects can be recast as data frames.

Usage

```
## S3 method for class 'spatPomp'
as.data.frame(x, ...)
```

Arguments

x	a spatPomp object.
...	additional arguments to be passed to or from methods.

Details

When object is a simple 'spatPomp' object, as(object,"data.frame") or as.data.frame(object) results in a data frame with the times, units, observables, states (if known), and interpolated covariates (if any).

Value

A 'data.frame' with columns for time, spatial unit and observations.

---

as_spatPomp *Coerce to spatPomp*

Description

Convert to class spatPomp object

Details

When object is a simple 'pomp' object, construct and return a one-dimensional 'spatPomp' object.

Value

a class 'spatPomp' representation of the object.

bm — *Brownian motion spatPomp simulator*

Description

Generate a class 'spatPomp' object representing a `U`-dimensional Brownian motion with spatial correlation decaying geometrically with distance around a circle. The model is defined in continuous time though in this case an Euler approximation is exact at the evaluation times.

Usage

```
bm(U = 5, N = 100, delta_t = 0.1)
```

Arguments

`U`	A length-one numeric signifying dimension of the process.
`N`	A length-one numeric signifying the number of observation time steps to evolve the process.
`delta_t`	Process simulations are performed every `delta_t` time units whereas observations occur every one time unit

Value

An object of class 'spatPomp' representing a simulation from a `U`-dimensional Brownian motion

Examples

```
b <- bm(U=4, N=20)
# See all the model specifications of the object
spy(b)
```

bpfilter — *Block particle filter (BPF)*

Description

An implementation of the block particle filter algorithm of Rebeschini and van Handel (2015), which is used to estimate the filter distribution of a spatiotemporal partially-observed Markov process. `bpfilter` requires a partition of the spatial units which can be provided by either the `block_size` or the `block_list` argument. The elements of the partition are called blocks. We perform resampling for each block independently based on sample weights within the block. Each resampled block only contains latent states for the spatial components within the block which allows for a "cross-pollination" of particles where the highest weighted segments of each particle are more likely to be resampled and get combined with resampled components of other particles. The method mitigates the curse of dimensionality by resampling locally.

Usage

```
## S4 method for signature 'spatPomp'
bpfilter(
  object,
  Np,
  block_size,
  block_list,
  save_states,
  ...,
  verbose = getOption("verbose", FALSE)
)
```

Arguments

object	A spatPomp object.
Np	The number of particles used within each replicate for the adapted simulations.
block_size	The number of spatial units per block. If this is provided, the method subdivides units approximately evenly into blocks with size block_size.
block_list	List that specifies an exact partition of the spatial units. Each partition element, or block, is an integer vector of neighboring units.
save_states	logical. If True, the state-vector for each particle and block is saved.
...	If a params argument is specified, bpfilter will estimate the likelihood at that parameter set instead of at coef(object).
verbose	logical; if TRUE, messages updating the user on progress will be printed to the console.

Value

Upon successful completion, bpfilter() returns an object of class 'bpfilterd_spatPomp' containing the algorithmic parameters used to run bpfilter() and the estimated likelihood.

Details

Only one of block_size or block_list should be specified. If both or neither is provided, an error is triggered.

Methods

The following methods are available for such an object:

logLik yields an estimate of the log-likelihood of the data under the model.

References

Rebeschini, P., & Van Handel, R. (2015). Can local particle filters beat the curse of dimensionality?. *The Annals of Applied Probability*, **25(5)**, 2809-2866.

See Also

Other particle filter methods: abfir(), abf(), enkf(), girf(), ienkf(), igirf(), iubf()

Examples

```
# Create a simulation of a Brownian motion
b <- bm(U=6, N=10)

# Run BPF with the specified number of units per block
bpfilterd_b1 <- bpfilter(b, Np = 100, block_size = 2)

# Run BPF with the specified partition
bpfilterd_b2 <- bpfilter(b,
                         Np = 20,
                         block_list = list(c(1,2), c(3,4), c(5,6)))

# Get a likelihood estimate
logLik(bpfilterd_b2)
```

---

city_data_UK *City data in the United Kingdom*

Description

Population and birth information about cities in England and Wales during the measles pre-vaccine era.

Details

Data includes births and population at bi-weekly observations from 40 cities and towns.

Value

a 'data.frame' of the 40 largest cities and towns in the UK and Wales, their latitude, longitude and mean population during the measles pre-vaccine period.

References

Dalziel, Benjamin D. et al. (2016) Persistent chaos of measles epidemics in the prevaccination United States caused by a small change in seasonal transmission patterns. *PLoS computational biology*, **12(2)**, e1004655. DOI: 10.5061/dryad.r4q34

See Also

Other datasets: measlesUK

`dunit_measure` *dunit_measure* `dunit_measure` *evaluates the unit measurement density of a unit's observation given the entire state*

Description

dunit_measure `dunit_measure` evaluates the unit measurement density of a unit's observation given the entire state

Usage

```
## S4 method for signature 'spatPomp'
dunit_measure(object, y, x, unit, time, params, log = TRUE, ...)
```

Arguments

`object`	An object of class `spatPomp`
`y`	A U by 1 matrix of observations for all units
`x`	A state vector for all units
`unit`	The unit for which to evaluate the unit measurement density
`time`	The time for which to evaluate the unit measurement density
`params`	parameters at which to evaluate the unit measurement density
`log`	logical; should the density be returned on log scale?
`...`	additional arguments will be ignored

Value

A class 'matrix' with the unit measurement density for spatial unit `unit` corresponding to the corresponding measurement in `y` and states in `x`.

Examples

```
b <- bm(U=3)
s <- states(b)[,1,drop=FALSE]
rownames(s) -> rn
dim(s) <- c(3,1,1)
dimnames(s) <- list(variable=rn, rep=NULL)
p <- coef(b); names(p) -> rnp
dim(p) <- c(length(p),1); dimnames(p) <- list(param=rnp)
o <- obs(b)[,1,drop=FALSE]
dunit_measure(b, y=o, x=s, unit=1, time=1, params=p)
```

enkf *Generalized Ensemble Kalman filter (EnKF)*

Description

A function to perform filtering using the ensemble Kalman filter of Evensen, G. (1994). This function is generalized to allow for an measurement covariance matrix that varies over time. This is useful if the measurement model varies with the state.

Usage

```
## S4 method for signature 'spatPomp'
enkf(data, Np, ..., verbose = getOption("verbose", FALSE))
```

Arguments

`data`	A `spatPomp` object.
`Np`	The number of Monte Carlo particles used to approximate the filter distribution.
`...`	If a `params` argument is specified, `abf` will estimate the likelihood at that parameter set instead of at `coef(object)`.
`verbose`	logical; if `TRUE`, messages updating the user on progress will be printed to the console.

Value

An object of class 'enkfd_spatPomp' that contains the estimate of the log likelihood (via the `loglik` attribute), algorithmic parameters used to run `enkf()`. Also included are estimated filter means, prediction means and forecasts that are generated during an `enkf()` run.

References

G. Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans* **99**, 10143–10162, 1994.

G. Evensen. *Data assimilation: the ensemble Kalman filter*. Springer-Verlag, 2009.

J.L. Anderson. An Ensemble Adjustment Kalman Filter for Data Assimilation. *Monthly Weather Review* **129**, 2884–2903, 2001.

See Also

Other particle filter methods: `abfir()`, `abf()`, `bpfilter()`, `girf()`, `ienkf()`, `igirf()`, `iubf()`

Examples

```
# Create a simulation of a Brownian motion
b <- bm(U=6, N=10)

# Run EnKF
enkfd_bm <- enkf(b, Np = 100)

# Get a likelihood estimate
logLik(enkfd_bm)
```

---

eunit_measure *eunit_measure*

Description

`eunit_measure` evaluates the expectation of a unit's observation given the entire state

Usage

```
## S4 method for signature 'spatPomp'
eunit_measure(object, x, unit, time, params, Np = 1, log = FALSE)
```

Arguments

`object`	An object of class `spatPomp`
`x`	A state vector for all units
`unit`	The unit for which to evaluate the expectation
`time`	The time for which to evaluate the expectation
`params`	parameters at which to evaluate the unit expectation
`Np`	numeric; defaults to 1 and the user need not change this
`log`	logical; should the density be returned on log scale?

Value

A class 'matrix' with the unit expected observation for spatial unit `unit` corresponding to the corresponding states in `x`.

Examples

```
b <- bm(U=3)
s <- states(b)[,1,drop=FALSE]
rownames(s) -> rn
dim(s) <- c(3,1,1)
dimnames(s) <- list(variable=rn, rep=NULL)
p <- coef(b); names(p) -> rnp
dim(p) <- c(length(p),1); dimnames(p) <- list(param=rnp)
o <- obs(b)[,1,drop=FALSE]
eunit_measure(b, x=s, unit=2, time=1, params=p)
```

gbm — *Geometric Brownian motion spatPomp simulator*

Description

Generate a spatPomp object representing a `U`-dimensional geometric Brownian motion with spatial correlation decaying geometrically with distance around a circle. The model is defined in continuous time, but an Euler approximation is used for this numerical implementation.

Usage

```
gbm(U = 5, N = 100, delta_t = 0.1, IVP_values = 1, delta_obs = 1)
```

Arguments

`U`	A length-one numeric signifying dimension of the process.
`N`	A length-one numeric signifying the number of time steps to evolve the process.
`delta_t`	process simulations are performed every `delta_t` time units
`IVP_values`	initial value parameters for the latent states
`delta_obs`	observations occur every `delta_obs` time units

Value

An object of class 'spatPomp' representing a simulation from a `U`-dimensional geometric Brownian motion

Examples

```
g <- gbm(U=4, N=20)
# See all the model specifications of the object
spy(g)
```

girf — *Guided intermediate resampling filter (GIRF)*

Description

An implementation of the algorithm of Park and Ionides (2020), following the pseudocode in Asfaw et al. (2020).

Usage

```
## S4 method for signature 'missing'
girf(object, ...)

## S4 method for signature 'ANY'
girf(object, ...)

## S4 method for signature 'spatPomp'
girf(
  object,
  Np,
  Ninter,
  lookahead = 1,
  Nguide,
  kind = c("bootstrap", "moment"),
  tol,
  ...,
  verbose = getOption("verbose", FALSE)
)

## S4 method for signature 'girfd_spatPomp'
girf(
  object,
  Np,
  Ninter,
  lookahead,
  Nguide,
  kind = c("bootstrap", "moment"),
  tol,
  ...
)
```

Arguments

`object`	A `spatPomp` object.
`...`	If a `params` argument is specified, `abf` will estimate the likelihood at that parameter set instead of at `coef(object)`.
`Np`	The number of particles used within each replicate for the adapted simulations.
`Ninter`	the number of intermediate resampling time points.
`lookahead`	The number of future observations included in the guide function.
`Nguide`	The number of simulations used to estimate state process uncertainty for each particle.
`kind`	One of two types of guide function construction. Defaults to `'bootstrap'`. See Park and Ionides (2020) for more details.
`tol`	If all of the guide function evaluations become too small (beyond floating-point precision limits), we set them to this value.

verbose logical; if TRUE, messages updating the user on progress will be printed to the console.

Value

Upon successful completion, `girf()` returns an object of class 'girfd_spatPomp' which contains the algorithmic parameters that were used to run `girf()` and the resulting log likelihood estimate.

Methods

The following methods are available for such an object:

`logLik` yields an unbiased estimate of the log-likelihood of the data under the model.

References

Park, J. and Ionides, E. L. (2020) Inference on high-dimensional implicit dynamic models using a guided intermediate resampling filter. *Statistics and Computing*, DOI: 10.1007/s11222-020-09957-3

Asfaw, K. et al. (2020) Statistical inference for spatiotemporal partially observed Markov processes via the R package spatPomp. *Manuscript in preparation*.

See Also

Other particle filter methods: `abfir()`, `abf()`, `bpfilter()`, `enkf()`, `ienkf()`, `igirf()`, `iubf()`

Examples

```
# Create a simulation of a Brownian motion
b <- bm(U=3, N=10)

# Run GIRF
girfd_bm <- girf(b,
                 Np = 100,
                 Ninter = length(unit_names(b)),
                 lookahead = 1,
                 Nguide = 50
)
# Get the likelihood estimate from GIRF
logLik(girfd_bm)

# Compare with the likelihood estimate from particle filter
pfd_bm <- pfilter(b, Np = 500)
logLik(pfd_bm)
```

ienkf *Iterated ensemble Kalman filter (IEnKF)*

Description

An implementation of a parameter estimation algorithm that uses the ensemble Kalman filter (Evensen, G. (1994)) to perform the filtering step in the parameter-perturbed iterated filtering scheme of Ionides et al. (2015) following the pseudocode in Asfaw, et al. (2020).

Usage

```
## S4 method for signature 'spatPomp'
ienkf(
  data,
  Nenkf = 1,
  rw.sd,
  cooling.type = c("geometric", "hyperbolic"),
  cooling.fraction.50,
  Np,
  ...,
  verbose = getOption("verbose", FALSE)
)
```

Arguments

`data` an object of class `spatPomp`

`Nenkf` number of iterations of perturbed EnKF.

`rw.sd` specification of the magnitude of the random-walk perturbations that will be applied to some or all model parameters. Parameters that are to be estimated should have positive perturbations specified here. The specification is given using the `rw.sd` function, which creates a list of unevaluated expressions. The latter are evaluated in a context where the model time variable is defined (as `time`). The expression `ivp(s)` can be used in this context as shorthand for

`ifelse(time==time[1],s,0).`

Likewise, `ivp(s,lag)` is equivalent to

`ifelse(time==time[lag],s,0).`

See below for some examples.

The perturbations that are applied are normally distributed with the specified s.d. If parameter transformations have been supplied, then the perturbations are applied on the transformed (estimation) scale.

`cooling.type` specifications for the cooling schedule, i.e., the manner and rate with which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. See below (under “Specifying the perturbations”) for more detail.

`cooling.fraction.50`
: specifications for the cooling schedule, i.e., the manner and rate with which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. See below (under "Specifying the perturbations") for more detail.

`Np`
: The number of particles used within each replicate for the adapted simulations.

`...`
: If a `params` argument is specified, `abf` will estimate the likelihood at that parameter set instead of at `coef(object)`.

`verbose`
: logical; if `TRUE`, messages updating the user on progress will be printed to the console.

Value

Upon successful completion, `ienkf` returns an object of class 'ienkfd_spatPomp'. This object contains the convergence record of the iterative algorithm with respect to the likelihood and the parameters of the model (which can be accessed using the `traces` attribute) as well as a final parameter estimate, which can be accessed using the `coef()`.

Methods

The following methods are available for such an object:

`coef` gives the Monte Carlo estimate of the maximum likelihood.

References

Evensen, G. (1994) Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics Journal of Geophysical Research: Oceans 99:10143–10162

Evensen, G. (2009) Data assimilation: the ensemble Kalman filter Springer-Verlag.

Anderson, J. L. (2001) An Ensemble Adjustment Kalman Filter for Data Assimilation Monthly Weather Review 129:2884–2903

See Also

Other particle filter methods: `abfir()`, `abf()`, `bpfilter()`, `enkf()`, `girf()`, `igirf()`, `iubf()`

Other spatPomp parameter estimation methods: `igirf()`, `iubf()`

`igirf` *Iterated guided intermediate resampling filter (IGIRF)*

Description

An implementation of a parameter estimation algorithm combining the intermediate resampling scheme of the guided intermediate resampling filter of Park and Ionides (2020) and the parameter perturbation scheme of Ionides et al. (2015) following the pseudocode in Asfaw, et al. (2020).

Usage

```
## S4 method for signature 'missing'
igirf(data, ...)

## S4 method for signature 'ANY'
igirf(data, ...)

## S4 method for signature 'spatPomp'
igirf(
  data,
  Ngirf,
  Np,
  rw.sd,
  cooling.type,
  cooling.fraction.50,
  Ninter,
  lookahead = 1,
  Nguide,
  kind = c("bootstrap", "moment"),
  tol = 1e-300,
  ...,
  verbose = getOption("verbose", FALSE)
)

## S4 method for signature 'igirfd_spatPomp'
igirf(
  data,
  Ngirf,
  Np,
  rw.sd,
  cooling.type,
  cooling.fraction.50,
  Ninter,
  lookahead,
  Nguide,
  kind = c("bootstrap", "moment"),
  tol,
  ...,
  verbose = getOption("verbose", FALSE)
)
```

Arguments

`data`	an object of class `spatPomp` or `igirfd_spatPomp`
`...`	If a `params` argument is specified, `abf` will estimate the likelihood at that parameter set instead of at `coef(object)`.
`Ngirf`	the number of iterations of parameter-perturbed GIRF.

`Np`	The number of particles used within each replicate for the adapted simulations.
`rw.sd`	specification of the magnitude of the random-walk perturbations that will be applied to some or all model parameters. Parameters that are to be estimated should have positive perturbations specified here. The specification is given using the `rw.sd` function, which creates a list of unevaluated expressions. The latter are evaluated in a context where the model time variable is defined (as `time`). The expression `ivp(s)` can be used in this context as shorthand for `ifelse(time==time[1],s,0)`. Likewise, `ivp(s,lag)` is equivalent to `ifelse(time==time[lag],s,0)`. See below for some examples. The perturbations that are applied are normally distributed with the specified s.d. If parameter transformations have been supplied, then the perturbations are applied on the transformed (estimation) scale.
`cooling.type`	specifications for the cooling schedule, i.e., the manner and rate with which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. See below (under "Specifying the perturbations") for more detail.
`cooling.fraction.50`	specifications for the cooling schedule, i.e., the manner and rate with which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. See below (under "Specifying the perturbations") for more detail.
`Ninter`	the number of intermediate resampling time points.
`lookahead`	The number of future observations included in the guide function.
`Nguide`	The number of simulations used to estimate state process uncertainty for each particle.
`kind`	One of two types of guide function construction. Defaults to `'bootstrap'`. See Park and Ionides (2020) for more details.
`tol`	If all of the guide function evaluations become too small (beyond floating-point precision limits), we set them to this value.
`verbose`	logical; if `TRUE`, messages updating the user on progress will be printed to the console.

Value

Upon successful completion, `igirf()` returns an object of class 'igirfd_spatPomp'. This object contains the convergence record of the iterative algorithm with respect to the likelihood and the parameters of the model (which can be accessed using the `traces` attribute) as well as a final parameter estimate, which can be accessed using the `coef()`. The algorithmic parameters used to run `igirf()` are also included.

Methods

The following methods are available for such an object:

`coef` gives the Monte Carlo maximum likelihood parameter estimate.

References

Park, J. and Ionides, E. L. (2020) Inference on high-dimensional implicit dynamic models using a guided intermediate resampling filter. *Statistics and Computing*, DOI: 10.1007/s11222-020-09957-3

Asfaw, K. et al. (2020) Statistical inference for spatiotemporal partially observed Markov processes via the R package spatPomp. *Manuscript in preparation*.

See Also

Other particle filter methods: `abfir()`, `abf()`, `bpfilter()`, `enkf()`, `girf()`, `ienkf()`, `iubf()`

Other spatPomp parameter estimation methods: `ienkf()`, `iubf()`

---

iubf *Iterated Unadapted Bagged Filter (IUBF)*

Description

An algorithm for estimating the parameters of a spatiotemporal partially-observed Markov process. Running `iubf` causes the algorithm to perform a specified number of iterations of unadapted simulations with parameter perturbation and parameter resamplings. At each iteration, unadapted simulations are performed on a perturbed version of the model, in which the parameters to be estimated are subjected to random perturbations at each observation. After cycling through the data, each replicate's weight is calculated and is used to rank the bootstrap replictates. The highest ranking replicates are recycled into the next iteration. This extra variability introduced through parameter perturbation effectively smooths the likelihood surface and combats particle depletion by introducing diversity into particle population. As the iterations progress, the magnitude of the perturbations is diminished according to a user-specified cooling schedule.

Usage

```
## S4 method for signature 'spatPomp'
iubf(
  object,
  Nubf = 1,
  Nrep_per_param,
  Nparam,
  nbhd,
  prop,
  rw.sd,
  cooling.type = c("geometric", "hyperbolic"),
```

```
  cooling.fraction.50,
  tol = (1e-18)^17,
  verbose = getOption("verbose"),
  ...
)
```

Arguments

`object` A `spatPomp` object.

`Nubf` The number of iterations to perform

`Nrep_per_param` The number of replicates used to estimate the likelihood at a parameter

`Nparam` The number of parameters that will undergo the iterated perturbation

`nbhd` A neighborhood function with three arguments: `object`, `time` and `unit`. The function should return a `list` of two-element vectors that represent space-time neighbors of (u, n), which is represented by `c(unit,time)`. See example below for more details.

`prop` A numeric between 0 and 1. The top `prop`*100% of the parameters are resampled at each observation

`rw.sd` specification of the magnitude of the random-walk perturbations that will be applied to some or all model parameters. Parameters that are to be estimated should have positive perturbations specified here. The specification is given using the `rw.sd` function, which creates a list of unevaluated expressions. The latter are evaluated in a context where the model time variable is defined (as `time`). The expression `ivp(s)` can be used in this context as shorthand for

`ifelse(time==time[1],s,0).`

Likewise, `ivp(s,lag)` is equivalent to

`ifelse(time==time[lag],s,0).`

See below for some examples.

The perturbations that are applied are normally distributed with the specified s.d. If parameter transformations have been supplied, then the perturbations are applied on the transformed (estimation) scale.

`cooling.type` specifications for the cooling schedule, i.e., the manner and rate with which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. See below (under "Specifying the perturbations") for more detail.

`cooling.fraction.50`
specifications for the cooling schedule, i.e., the manner and rate with which the intensity of the parameter perturbations is reduced with successive filtering iterations. `cooling.type` specifies the nature of the cooling schedule. See below (under "Specifying the perturbations") for more detail.

`tol` If the resampling weight for a particle is zero due to floating-point precision issues, it is set to the value of `tol` since resampling has to be done.

`verbose` logical; if `TRUE`, diagnostic messages will be printed to the console.

... additional arguments supply new or modify existing model characteristics or components. See pomp for a full list of recognized arguments.

When named arguments not recognized by pomp are provided, these are made available to all basic components via the so-called *userdata* facility. This allows the user to pass information to the basic components outside of the usual routes of covariates (covar) and model parameters (params). See userdata for information on how to use this facility.

Value

Upon successful completion, iubf() returns an object of class 'iubfd_spatPomp'. This object contains the convergence record of the iterative algorithm with respect to the likelihood and the parameters of the model (which can be accessed using the traces attribute) as well as a final parameter estimate, which can be accessed using the coef(). The algorithmic parameters used to run iubf() are also included.

Methods

The following methods are available for such an object:

coef extracts the point estimate

See Also

Other particle filter methods: abfir(), abf(), bpfilter(), enkf(), girf(), ienkf(), igirf()

Other spatPomp parameter estimation methods: ienkf(), igirf()

---

logLik *Log likelihood*

Description

Extract the estimated log likelihood.

Usage

```
## S4 method for signature 'girfd_spatPomp'
logLik(object)

## S4 method for signature 'bpfilterd_spatPomp'
logLik(object)

## S4 method for signature 'abfd_spatPomp'
logLik(object)

## S4 method for signature 'iubfd_spatPomp'
logLik(object)
```

```
## S4 method for signature 'abfird_spatPomp'
logLik(object)

## S4 method for signature 'igirfd_spatPomp'
logLik(object)
```

Arguments

`object`	fitted model object

Value

a numeric which is the estimated log likelihood

---

lorenz — *Lorenz '96 spatPomp simulator*

Description

Generate a spatPomp object representing a `U`-dimensional stochastic Lorenz '96 process with `N` measurements made at times $t_n = n * delta_obs$, simulated using an Euler method with time increment `delta_t`.

Usage

```
lorenz(
  U = 5,
  N = 100,
  delta_t = 0.01,
  delta_obs = 0.5,
  regular_params = c(F = 8, sigma = 1, tau = 1)
)
```

Arguments

`U`	A length-one numeric signifying the number of spatial units for the process.
`N`	A length-one numeric signifying the number of observations.
`delta_t`	A length-one numeric giving the Euler time step for the numerical solution.
`delta_obs`	A length-one numeric giving the time between observations.
`regular_params`	A named numeric vector containing the values of the `F`, `sigma` and `tau` parameters. `F=8` is a common value that causes chaotic behavior.

Value

An object of class 'spatPomp' representing a simulation from a `U`-dimensional Lorenz 96 model

References

Lorenz, E. N. (1996) Predictability: A problem partly solved. *Proceedings of the seminar on predictability*

Examples

```
l <- lorenz(U=5, N=100, delta_t=0.01, delta_obs=1)
# See all the model specifications of the object
spy(l)
```

`mcap` *Monte Carlo adjusted profile*

Description

Given a collection of points maximizing the likelihood over a range of fixed values of a focal parameter, this function constructs a profile likelihood confidence interval accommodating both Monte Carlo error in the profile and statistical uncertainty present in the likelihood function.

Usage

```
mcap(lp, parameter, confidence = 0.95, lambda = 1, Ngrid = 1000)
```

Arguments

`lp`	a vector of profile likelihood evaluations.
`parameter`	the corresponding values of the focal parameter.
`confidence`	the required level of the confidence interval.
`lambda`	the loess parameter used to smooth the profile.
`Ngrid`	the number of points to evaluate the smoothed profile.

Value

`mcap()` returns a list including the smoothed profile, a quadratic approximation, and the constructed confidence interval.

Author(s)

Edward L. Ionides

measles *Measles in UK spatPomp generator*

Description

Generate a spatPomp object for measles in the top-U most populous cities in England and Wales. The model is adapted from He et al. (2010) with gravity transport following Park and Ionides (2019). The data in the object is simulated using the process and measurement models of He et al. (2010).

Usage

```
measles(
  U = 6,
  dt = 2/365,
  fixed_ivps = TRUE,
  shared_ivps = TRUE,
  S_0 = 0.032,
  E_0 = 5e-05,
  I_0 = 4e-05
)
```

Arguments

U	A length-one numeric signifying the number of cities to be represented in the spatPomp object.
dt	a numeric (in unit of years) that is used as the Euler time-increment for simulating measles data.
fixed_ivps	a logical. If TRUE initial value parameters will be declared in the globals slot and will not be part of the parameter vector.
shared_ivps	a logical. If TRUE and fixed_ivps=TRUE the values of S_0, E_0 and I_0 in the call to measles will be used as initial value parameters for all spatial units.
S_0	a numeric. If shared_ivps=TRUE and fixed_ivps=TRUE this is the initial proportion of all of the spatial units that are susceptible.
E_0	a numeric. If shared_ivps=TRUE and fixed_ivps=TRUE this is the initial proportion of all of the spatial units that are exposed.
I_0	a numeric. If shared_ivps=TRUE and fixed_ivps=TRUE this is the initial proportion of all of the spatial units that are infected.

Value

An object of class 'spatPomp' representing a U-dimensional spatially coupled measles POMP model.

Note

This function goes through a typical workflow of constructing a typical spatPomp object (1-4 below). This allows the user to have a file that replicates the exercise of model building as well as function that creates a typical nonlinear model in epidemiology in case they want to test a new inference methodology. We purposely do not modularize this function because it is not an operational piece of the package and is instead useful as an example.
1. Getting a measurements data.frame with columns for times, spatial units and measurements.
2. Getting a covariates data.frame with columns for times, spatial units and covariate data.
3. Constructing model components (latent state initializer, latent state transition simulator and measurement model). Depending on the methods used, the user may have to supply a vectorfield to be integrated that represents the deterministic skeleton of the latent process.
4. Bringing all the data and model components together to form a spatPomp object via a call to spatPomp().

References

Robert J. Hijmans (2019). The **geosphere** spherical trigonometry package. https://CRAN.R-project.org/package=geosphere.

Examples

```
m <- measles(U = 5)
# See all the model specifications of the object
spy(m)
```

---

measlesUK *Measles in the United Kingdom*

Description

Measles case data from various cities and towns in England and Wales during the pre-vaccine era.

Details

Data includes bi-weekly case counts as well as births and population from 40 cities and towns.

Value

a 'data.frame' of the 40 largest cities and towns in the UK and Wales, their latitude, longitude and bi-weekly measles case counts, population and birthrates.

References

Dalziel, Benjamin D. et al. (2016) Persistent chaos of measles epidemics in the prevaccination United States caused by a small change in seasonal transmission patterns. *PLoS computational biology*, **12(2)**, e1004655. DOI: 10.5061/dryad.r4q34

See Also

Other datasets: city_data_UK

---

munit_measure *munit_measure*

Description

munit_measure returns a moment-matched parameter set given an empirically calculated measurement variance and latent states. This is used in girf() and igirf() when they are run with kind='moment'.

Usage

```
## S4 method for signature 'spatPomp'
munit_measure(object, x, vc, unit, time, params, Np = 1)
```

Arguments

object	An object of class spatPomp
x	A state vector for all units
vc	The empirically calculated variance used to perform moment-matching
unit	The unit for which to obtain a moment-matched parameter set
time	The time for which to obtain a moment-matched parameter set
params	parameters to use to obtain a moment-matched parameter set
Np	Number of particle replicates for which to get parameter sets

Value

An array with dimensions dim(array.params)[1] by dim(x)[2] by length(unit) bylength(time) representing the moment-matched parameter set(s) corresponding to the variance of the measurements, vc, and the states, x.

Examples

```
b <- bm(U=3)
s <- states(b)[,1,drop=FALSE]
rownames(s) -> rn
dim(s) <- c(3,1,1)
dimnames(s) <- list(variable=rn, rep=NULL)
p <- coef(b); names(p) -> rnp
dim(p) <- c(length(p),1); dimnames(p) <- list(param=rnp)
o <- obs(b)[,1,drop=FALSE]
array.params <- array(p,
                      dim = c(length(p),
                              length(unit_names(b)), 1, 1),
```

```
                    dimnames = list(params = rownames(p)))
vc <- c(4, 9, 16); dim(vc) <- c(length(vc), 1, 1)
munit_measure(b, x=s, vc=vc, Np=1, unit = 1, time=1, params=array.params)
```

---

plot — *Plotting* spatPomp *data*

Description

Visualize spatPomp data

Diagnostic plot for igirf()

Visualize spatPomp data

Usage

```
## S4 method for signature 'igirfd_spatPomp'
plot(x, params = names(coef(x)), ncol = 3)

## S4 method for signature 'spatPomp'
plot(x, type = c("l", "h"), log = F, ...)
```

Arguments

x	a spatPomp object
params	the names of the parameters for which the user would like to see a trace plot
ncol	the number of columns in the grid plot
type	for visualizing an object of class spatPomp, the user can obtain a grid of line plots by default ('l') or a heat map by supplying argument 'h'.
log	should the data be log-transformed before plotting? This helps in contexts where there are spikes that could take away attention from the dynamics illustrated by the rest of the data.
...	for visualizing an object of class spatPomp, the user can add arguments like nrow to specify the number of rows in the grid.

Value

a ggplot facet plot of class ‘gg’ and ‘ggplot’ visualizing the convergence record of running igirf() with respect to the likelihood and the parameters of the model.

a ggplot plot of class ‘gg’ and ‘ggplot’ visualizing the time series data over multiple spatial units via a tile-plot.

print *Print methods*

Description

Prints its argument.

Usage

```
## S4 method for signature 'spatPomp'
print(x)
```

Arguments

`x`	a `spatPomp` object

Value

An object of class ‘spatPomp’ is returned *invisibly*. The user is notified on the console only the class of the object.

Note

Use `spy()` to see model components of `x` instead.

pStop *pStop*

Description

Custom error function

Usage

```
pStop(fn, ...)

pStop_(...)

pWarn(fn, ...)

pWarn_(...)
```

Arguments

`fn`	name of function (will be enclosed in single quotes)
`...`	message

Value

No return value as this is simply a custom error function.

`reexports` *Objects exported from other packages*

Description

These objects are imported from other packages. Follow the links below to see their documentation.

magrittr `%>%`

`runit_measure` *runit_measure*

Description

`runit_measure` simulates a unit's observation given the entire state

Usage

```
## S4 method for signature 'spatPomp'
runit_measure(object, x, unit, time, params, log = FALSE)
```

Arguments

`object`	An object of class `spatPomp`
`x`	A state vector for all units
`unit`	The unit for which to simulate an observation
`time`	The time for which to simulate an observation
`params`	parameters to use to simulate an observation
`log`	logical; should the density be returned on log scale?

Value

A matrix with the simulated observation corresponding to state `x` and unit `unit` with parameter set `params`.

Examples

```
b <- bm(U=3)
s <- states(b)[,1,drop=FALSE]
rownames(s) -> rn
dim(s) <- c(3,1,1)
dimnames(s) <- list(variable=rn, rep=NULL)
p <- coef(b); names(p) -> rnp
dim(p) <- c(length(p),1); dimnames(p) <- list(param=rnp)
o <- obs(b)[,1,drop=FALSE]
runit_measure(b, x=s, unit=2, time=1, params=p)
```

---

simulate — *Simulation of a spatiotemporal partially-observed Markov process*

Description

`simulate` generates simulations of the latent and measurement processes.

Usage

```
## S4 method for signature 'spatPomp'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  format = c("spatPomps", "data.frame"),
  include.data = FALSE,
  ...
)
```

Arguments

`object`	optional; if present, it should be a data frame or a 'pomp' object.
`nsim`	number of simulations.
`seed`	optional; if set, the pseudorandom number generator (RNG) will be initialized with seed. the random seed to use. The RNG will be restored to its original state afterward.
`format`	the format of the simulated results. If the argument is set to `'spatPomps'`, the default behavior, then the output is a `list` of `spatPomp` objects. Options are `'spatPomps'` and `'data.frame'`.
`include.data`	if `TRUE`, the original data and covariates (if any) are included (with `.id = "data"`). This option is ignored unless `format = "data.frame"`.
`...`	additional arguments supply new or modify existing model characteristics or components. See `pomp` for a full list of recognized arguments.

When named arguments not recognized by pomp are provided, these are made available to all basic components via the so-called *userdata* facility. This allows the user to pass information to the basic components outside of the usual routes of covariates (covar) and model parameters (params). See userdata for information on how to use this facility.

Value

if format='spatPomps' and nsim=1 an object of class 'spatPomp' representing a simulation from the model in object is returned. If format='spatPomps' and nsim>1 a list of class 'spatPomp' objects is returned. If format='data.frame' then a class 'data.frame' object is returned.

Examples

```
# Get a spatPomp object
b <- bm(U=5, N=10)
# Get 10 simulations from same model as data.frame
sims <- simulate(b, nsim=10, format='data.frame')
```

---

spatPomp — *Constructor of the spatPomp object*

Description

This function constructs a class 'spatPomp' object, encoding a spatiotemporal partially observed Markov process (SPATPOMP) model together with a uni- or multi-variate time series on a collection of units. Users will typically develop a POMP model for a single unit before embarking on a coupled SpatPOMP analysis. Consequently, we assume some familiarity with **pomp** and its description by King, Nguyen and Ionides (2016). The spatPomp class inherits from pomp with the additional unit structure being a defining feature of the resulting models and inference algorithms.

Usage

```
spatPomp(
  data,
  units,
  times,
  covar,
  t0,
  ...,
  eunit_measure,
  munit_measure,
  vunit_measure,
  dunit_measure,
  runit_measure,
  rprocess,
  rmeasure,
  dprocess,
```

```
  dmeasure,
  skeleton,
  rinit,
  rprior,
  dprior,
  unit_statenames,
  unit_accumvars,
  shared_covarnames,
  globals,
  paramnames,
  params,
  cdir,
  cfile,
  shlib.args,
  PACKAGE,
  partrans,
  compile = TRUE,
  verbose = getOption("verbose", FALSE)
)
```

Arguments

`data` either a dataframe holding the spatiotemporal data, or an object of class 'spatPomp', i.e., the output of another **spatPomp** calculation. If dataframe, the user must provide the name of the times column using the `times` argument and the spatial unit column name using the `units` argument. The dataframe provided should be sorted in increasing order of time and unit name respectively, i.e. observation 1 in unit A should come before observation 1 in unit B, which should come before observation 2 in unit A.

`units` when `data` is a `data.frame` this is the name of the column containing the spatial units.

`times` the sequence of observation times. `times` must indicate the column of observation times by name or index. The time vector must be numeric and non-decreasing.

`covar` An optional dataframe for supplying covariate information. If provided, there must be two columns that provide the observation time and the observation spatial unit with the same names and arrangement as the `data`.

`t0` The zero-time, i.e., the time of the initial state. This must be no later than the time of the first observation, i.e., `t0 <= times[1]`.

`...` If there are arguments that the user would like to pass to **pomp**'s basic constructor function's ... argument, this argument passes them along. Not recommended for this version of **spatPomp**.

`eunit_measure` Evaluator of the expected measurement given the latent states and model parameters. The `unit` variable is pre-defined, which allows the user to specify differing specifications for each unit using `if` conditions. Only C snippets are accepted. The C snippet should assign the scalar approximation to the expected measurement to the pre-defined variable `ey` given the latent state and the parameters. For more information, see the examples section below.

`munit_measure` Evaluator of a moment-matched parameter set (like the standard deviation parameter of a normal distribution or the size parameter of a negative binomial distribution) given an empirical variance estimate, the latent states and all model parameters. Only Csnippets are accepted. The Csnippet should assign the scalar approximation to the measurement variance parameter to the pre-defined variable corresponding to that parameter, which has been predefined with a `M_` prefix. For instance, if the moment-matched parameter is `psi`, then the user should assign `M_psi` to the moment-matched value. For more information, see the examples section below.

`vunit_measure` Evaluator of the theoretical measurement variance given the latent states and model parameters. The `unit` variable is pre-defined, which allows the user to specify differing specifications for each unit using `if` conditions. Only C snippets are accepted. The C snippet should assign the scalar approximation to the measurement variance to the pre-defined variable `vc` given the latent state and the parameters. For more information, see the examples section below.

`dunit_measure` Evaluator of the unit measurement model density given the measurement, the latent states and model parameters. The `unit` variable is pre-defined, which allows the user to specify differing specifications for each unit using `if` conditions. Only Csnippets are accepted. The Csnippet should assign the scalar measurement density to the pre-defined variable `lik`. The user is encouraged to provide a logged density in an `if` condition that checks whether the predefined `give_log` variable is true. For more information, see the examples section below.

`runit_measure` Simulator of the unit measurement model given the latent states and the model parameters. The `unit` variable is pre-defined, which allows the user to specify differing specifications for each unit using `if` conditions. Only Csnippets are accepted. The Csnippet should assign the scalar measurement density to the pre-defined which corresponds to the name of the observation for each unit (e.g. `cases` for the measles spatPomp example). For more information, see the examples section below.

`rprocess` simulator of the latent state process, specified using one of the rprocess plugins. Setting `rprocess=NULL` removes the latent-state simulator. For more information, see rprocess specification for the documentation on these plugins.

`rmeasure` simulator of the measurement model, specified either as a C snippet, an R function, or the name of a pre-compiled native routine available in a dynamically loaded library. Setting `rmeasure=NULL` removes the measurement model simulator. For more information, see rmeasure specification.

`dprocess` optional; specification of the probability density evaluation function of the unobserved state process. Setting `dprocess=NULL` removes the latent-state density evaluator. For more information, see dprocess specification.

`dmeasure` evaluator of the measurement model density, specified either as a C snippet, an R function, or the name of a pre-compiled native routine available in a dynamically loaded library. Setting `dmeasure=NULL` removes the measurement density evaluator. For more information, see dmeasure specification.

`skeleton` optional; the deterministic skeleton of the unobserved state process. Depending on whether the model operates in continuous or discrete time, this is either a vectorfield or a map. Accordingly, this is supplied using either the `vectorfield` or `map` fnctions. For more information, see skeleton specification. Setting `skeleton=NULL` removes the deterministic skeleton.

`rinit` simulator of the initial-state distribution. This can be furnished either as a C snippet, an R function, or the name of a pre-compiled native routine available in a dynamically loaded library. Setting `rinit=NULL` sets the initial-state simulator to its default. For more information, see rinit specification.

`rprior` optional; prior distribution sampler, specified either as a C snippet, an R function, or the name of a pre-compiled native routine available in a dynamically loaded library. For more information, see prior specification. Setting `rprior=NULL` removes the prior distribution sampler.

`dprior` optional; prior distribution density evaluator, specified either as a C snippet, an R function, or the name of a pre-compiled native routine available in a dynamically loaded library. For more information, see prior specification. Setting `dprior=NULL` resets the prior distribution to its default, which is a flat improper prior.

`unit_statenames` The names of the components of the latent state. E.g. if the user is constructing an joint SIR model over many spatial units, `c('S','I','R')` would be passed.

`unit_accumvars` a subset of the `unit_statenames` argument that are accumulator variables. See accumulator variables for more on the concept of **pomp** accumulator variables.

`shared_covarnames` If `covar` is supplied, covariates that are shared must still be specified for each unit, i.e., rows with equal values for the same time over all units must be supplied. However, if such covariates exists, supply the names using this argument.

`globals` optional character; arbitrary C code that will be hard-coded into the shared-object library created when C snippets are provided. If no C snippets are used, `globals` has no effect.

`paramnames` optional character vector; names of model parameters. It is typically only necessary to supply `paramnames` when C snippets are in use.

`params` optional; named numeric vector of parameters. This will be coerced internally to storage mode `double`.

`cdir` optional character variable. `cdir` specifies the name of the directory within which C snippet code will be compiled. By default, this is in a temporary directory specific to the R session. One can also set this directory using the `pomp_cdir` global option.

`cfile` optional character variable. `cfile` gives the name of the file (in directory `cdir`) into which C snippet codes will be written. By default, a random filename is used. If the chosen filename would result in over-writing an existing file, an error is generated.

`shlib.args` optional character variables. Command-line arguments to the `R CMD SHLIB` call that compiles the C snippets.

`PACKAGE` optional character; the name (without extension) of the external, dynamically loaded library in which any native routines are to be found. This is only useful if one or more of the model components has been specified using a precompiled

	dynamically loaded library; it is not used for any component specified using C snippets. `PACKAGE` can name at most one library.
`partrans`	optional parameter transformations, constructed using `parameter_trans`. Many algorithms for parameter estimation search an unconstrained space of parameters. When working with such an algorithm and a model for which the parameters are constrained, it can be useful to transform parameters. One should supply the `partrans` argument via a call to `parameter_trans`. For more information, see parameter_trans. Setting `partrans=NULL` removes the parameter transformations, i.e., sets them to the identity transformation.
`compile`	logical; if `FALSE`, compilation of the C snippets will be postponed until they are needed.
`verbose`	logical; if `TRUE`, diagnostic messages will be printed to the console.

Details

One implements a SPATPOMP model by specifying some or all of its *basic components*, including:

rinit, the simulator from the distribution of the latent state process at the zero-time;

rprocess, the transition simulator of the latent state process;

dunit_measure, the evaluator of the conditional density at a unit's measurement given the unit's latent state;

eunit_measure, the evaluator of the expectation of a unit's measurement given the unit's latent state;

munit_measure, the evaluator of the moment-matched parameter set given a unit's latent state and some empirical measurement variance;

vunit_measure, the evaluator of the variance of a unit's measurement given the unit's latent state;

runit_measure, the simulator of a unit's measurement conditional on the unit's latent state;

dprocess, the evaluator of the density for transitions of the latent state process;

rmeasure, the simulator of the measurements conditional on the latent state;

dmeasure, the evaluator of the conditional density of the measurements given the latent state;

rprior, the simulator from a prior distribution on the parameters;

dprior, the evaluator of the prior density;

skeleton, which computes the deterministic skeleton of the unobserved state process;

partrans, which performs parameter transformations.

The basic structure and its rationale are described in Asfaw et al. (2020).

Each basic component is supplied via an argument of the same name to `spatPomp()`. The five unit-level model components must be provided via C snippets. The remaining components, whose behaviors are inherited from **pomp** may be furnished using C snippets, R functions, or pre-compiled native routine available in user-provided dynamically loaded libraries.

Value

An object of class 'spatPomp' representing observations and model components from the spatiotemporal POMP model.

References

Asfaw, K. et al. (2020) Statistical inference for spatiotemporal partially observed Markov processes via the R package spatPomp. *Manuscript in preparation*.

`spatPomp-class` *An S4 class to represent a spatiotemporal POMP model and data.*

Description

An S4 class to represent a spatiotemporal POMP model and data.

Slots

`unit_names` A vector containing the spatial units of a spatiotemporal POMP.

`unit_statenames` A vector containing the state names such that appending the unit indices to the unit statenames will result in the each unit's corresponding states.

`unit_obsnames` A vector of observation types for a spatial unit.

`eunit_measure` A pomp_fun representing the expected measurement for each spatial unit given its states.

`dunit_measure` A pomp_fun representing the unit measurement density for each spatial unit.

`runit_measure` A pomp_fun representing the unit observation simulator.

`spatPomp_Csnippet` *C snippets*

Description

`spatPomp_Csnippet()` is used to provide snippets of C code that specify model components. It functions similarly to `Csnippet()` from the **pomp** package; in fact, the output of `spatPomp_Csnippet` is an object of class `Csnippet`. It additionally provides some arguments that allow the user to stay focused on model development in the spatiotemporal context where model size grows.

Usage

```
## S4 method for signature 'character'
spatPomp_Csnippet(
  code,
  unit_statenames,
  unit_obsnames,
  unit_covarnames,
  unit_ivpnames,
  unit_paramnames,
  unit_vfnames
)
```

Arguments

`code` encodes a component of a spatiotemporal POMP model using C code

`unit_statenames` a subset of the `unit_statenames` slot of the `spatPomp` object for which we are writing a model. This argument allows the user to get variables that can be indexed conveniently to update states and measurements in a loop. See examples for more details.

`unit_obsnames` a subset of the `unit_obsnames` slot of the `spatPomp` object for which we are writing a model. This argument allows the user to get variables that can be indexed conveniently to update states and measurements in a loop. See examples for more details.

`unit_covarnames` if the model has covariate information for each unit, the names of the covariates for each unit can be supplied to this argument. This allows the user to get variables that can be indexed conveniently to use incorporate the covariate information in a loop. See examples for more details.

`unit_ivpnames` This argument is particularly useful when specifying the `rinit` model component. The `paramnames` argument to the `spatPomp()` constructor often has names for initial value parameters for the latent states (e.g. `S1_0`, `S2_0` for the the quantity of susceptibles at unit 1 and unit 2 at the initial time in an SIR model). By supplying `unit_ivpnames`, we can get variables that can be easily indexed to reference the initial value parameters (in the previous example, `unit_ivpnames=c('S')` we can get a variable named `S_0` that we can index as `S_0[0]` and `S_0[1]` to refer to `S1_0` and `S2_0`). See examples for more details.

`unit_paramnames` This argument is particularly useful when there are non-initial value parameters that are unit-specific.

`unit_vfnames` This argument is particularly useful when specifying the `skeleton` model component. For all components of the latent state, the user can assume a variable defining the time-derivative is pre-defined (e.g. `DS1` and `DS2` for the time-derivative of the quantity of the susceptibles at unit 1 and unit 2 in an SIR model). By supplying `unit_vfnames`, we can get variables that can be easily indexed to reference these variables (in the previous example, setting `unit_vfnames=c('S')` gets us a variable named `DS` that we can index as `DS[0]` and `DS[1]` to refer to `DS1` and `DS2`). See examples for more details.

Value

An object of class 'Csnippet' which represents a model specification in C code.

Examples

```
# Set initial states for Brownian motion
bm_rinit <- spatPomp_Csnippet(
  unit_statenames = c("X"),
  unit_ivpnames = c("X"),
  code = "
    for (int u = 0; u < U; u++) {
```

```
      X[u]=X_0[u];
    }
  "
)
# Skeleton for Brownian motion
bm_skel <- spatPomp_Csnippet(
  unit_statenames = c("X"),
  unit_vfnames = c("X"),
  code = "
      for (int u = 0 ; u < U ; u++) {
        DX[u] = 0;
      }
  "
)
```

---

undefined *Undefined*

Description

Check for undefined methods.

Usage

```
undefined(object, ...)

## S4 method for signature '`NULL`'
undefined(object, ...)

## S4 method for signature 'ANY'
undefined(object, ...)

## S4 method for signature 'missing'
undefined(object, ...)

## S4 method for signature 'pomp_fun'
undefined(object, ...)

## S4 method for signature 'partransPlugin'
undefined(object, ...)

## S4 method for signature 'rprocPlugin'
undefined(object, ...)

## S4 method for signature 'covartable'
undefined(object)

## S4 method for signature 'skelPlugin'
undefined(object, ...)
```

Arguments

`object`	object to test.
`...`	currently ignored.

Value

Returns `TRUE` if the **pomp** workhorse method is undefined, `FALSE` if it is defined, and `NA` if the question is inapplicable.

`unit_names` *Unit names of a spatiotemporal model*

Description

`unit_names` outputs the contents of the `unit_names` slot of a `spatPomp` object. The order in which the units appear in the output vector determines the order in which latent states and observations for the spatial units are stored.

Usage

```
## S4 method for signature 'spatPomp'
unit_names(x)
```

Arguments

`x`	a `spatPomp` object

Value

A character vector with the unit names used to create the 'spatPomp' object.

`vec_dmeasure` *Vector of measurement densities*

Description

Evaluate the unit measurement model density function for each unit. This method is used primarily as part of likelihood evaluation and parameter inference algorithms.

Usage

```
## S4 method for signature 'spatPomp'
vec_dmeasure(object, y, x, units, times, params, log = FALSE, ...)
```

Arguments

`object`	a `spatPomp` object
`y`	numeric; measurements whose densities given the latent states are evaluated
`x`	numeric; state at which conditional measurement densities are evaluated
`units`	numeric; units at which measurement densities are evaluated
`times`	numeric; time at which measurement densities are evaluated
`params`	numeric; parameter set at which measurement densities is evaluated
`log`	logical; should the outputted measurement densities be on log scale?
`...`	additional parameters will be ignored

Value

An array of dimension `length(unit_names(object))` by `dim(x)[2]` by `dim(x)[3]` representing each unit's measurement density assessed for each replicate in `x` for each observation time.

---

vec_rmeasure *Vector of simulated measurements*

Description

Simulate from the unit measurement model density function for each unit

Usage

```
## S4 method for signature 'spatPomp'
vec_rmeasure(object, x, times, params, ...)
```

Arguments

`object`	a `spatPomp` object
`x`	numeric; state at which measurements are simulated
`times`	numeric; time at which measurements are simulated
`params`	numeric; parameter set at which measurements are simulated
`...`	additional parameters will be ignored

Value

An array of dimension `length(unit_names(object))` by `dim(x)[2]` by `dim(x)[3]` representing each unit's simulated measurement assessed for each replicate in `x` for each observation time.

vunit_measure *vunit_measure*

Description

vunit_measure evaluates the variance of a unit's observation given the entire state

Usage

```
## S4 method for signature 'spatPomp'
vunit_measure(object, x, unit, time, params, Np = 1)
```

Arguments

object	An object of class spatPomp
x	A state vector for all units
unit	The unit for which to evaluate the variance
time	The time for which to evaluate the variance
params	parameters at which to evaluate the unit variance
Np	numeric; defaults to 1 and the user need not change this

Value

A matrix with the unit measurement variance implied by the state, x, and the parameter set params for unit unit.

Examples

```
b <- bm(U=3)
s <- states(b)[,1,drop=FALSE]
rownames(s) -> rn
dim(s) <- c(3,1,1)
dimnames(s) <- list(variable=rn, rep=NULL)
p <- coef(b); names(p) -> rnp
dim(p) <- c(length(p),1); dimnames(p) <- list(param=rnp)
o <- obs(b)[,1,drop=FALSE]
vunit_measure(b, x=s, unit=2, time=1, params=p)
```

Index